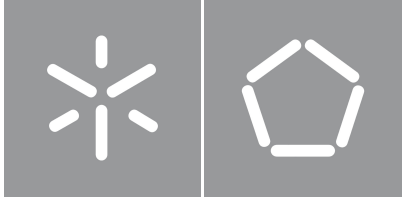


University of Minho
School of Engineering

Miguel Caçador Peixoto

Anomaly Detection as a Quality Control Tool in an Industrial Context

December 2024



University of Minho
School of Engineering

Miguel Caçador Peixoto

Anomaly Detection as a Quality Control Tool in an Industrial Context

Master's Dissertation
Master's in Physics Engineering

Information Physics
Work carried out under the supervision of
Nuno Filipe da Silva Fernandes de Castro

December 2024

Copyright and Terms of Use for Third Party Work

This dissertation reports on academic work that can be used by third parties as long as the internationally accepted standards and good practices are respected concerning copyright and related rights.
This work can thereafter be used under the terms established in the license below.
Readers needing authorization conditions not provided for in the indicated licensing should contact the author through the RepositóriUM of the University of Minho.

License granted to users of this work:



CC BY-NC-SA

https://creativecommons.org/licenses/by-nc-sa/4.0/

Acknowledgements

This thesis represents the culmination of significant dedication, encompassing innumerable hours of coding, research, debugging, and writing. It reflects the challenge of navigating multiple priorities, balancing the objectives of our industry-focused partners with those of the academic community, while also fulfilling my role as a machine learning engineer and personal aspirations. The completion of this work would not have been feasible without the support and guidance of many, to whom I am profoundly grateful.

First and foremost, I am deeply grateful to my advisor, Dr. Nuno Castro, for his continuous support and guidance throughout my academic journey. I am especially thankful for his patience in answering all my questions, even during weekends and late hours. His insights and suggestions have significantly contributed to the development of this thesis. His mentorship has been truly invaluable, and I deeply appreciate all that he has done for me.

I also want to express my heartfelt thanks to Dr. Miguel Romão, who, alongside Dr. Castro, welcomed me into the research community at LIP as a first-year student. It was there that I learned the fundamentals of machine learning and laid the groundwork that have made this thesis possible. I must also acknowledge the entire team at LIP for their support, camaraderie, and the collaborative spirit that made my time there both productive and enjoyable.

I am thoroughly indebted for the insightful discussions with João Paulo and Diogo Costa from *Bosch Car Multimedia Portugal, S.A.*. Their insights into the industry's operations, including its needs and limitations in quality control, were indispensable to the completion of this work. I would also like to express my sincere thanks to Dr. Stéphane Clain for his academic guidance. His boundless curiosity not only enriched my perspective on the field but also enhanced the project itself. Furthermore, I wish to acknowledge the contributions of others to this project, including my predecessor André Amorim, as well as Professors Rui Pereira, João Ferreira, and José Meijome.

Acknowledgement is also due to my manager, Pedro Bastardo, whose flexibility allowed me

to balance my professional and academic responsibilities effectively during my thesis work.
I am deeply grateful to my family for their unwavering support and trust. You have always been my greatest cheerleaders, encouraging me to strive for excellence in my studies and creating an environment where I could pursue my dreams and become the person I aspired to be. Your love and guidance have been the bedrock of my success, and I am forever thankful for all that you have done for me.
A special thanks goes to Inês, whose rock solid support and encouragement have been pivotal to my journey. In moments of doubt and overwhelming pressure, when it felt like everything was falling apart, her presence provided me with strength and stability, lighting the way forward. Her unwavering love, patience, and understanding have been a true source of comfort, and for that, I am sincerely grateful.
I would also like to express my heartfelt appreciation to my university peers—listed alphabetically to keep any egos in check — Bruna Salgado, Fábio Carneiro, Gabriela Jordão, Gonçalo Dores, Margarida Moura, Rui Loureiro, and Vitória Sousa. Each of you has been instrumental in this journey, not only making it possible but also filling it with unforgettable memories that I will cherish forever.
Lastly, this work couldn't have been achieved without the support by the LIP (UIDP/50007/2020, LA/P/0016/2020), the Center of Competence in Simulation and Big Data and national funds, though the Operational Competitiveness and Internationalization Programme (COMPETE 2020) (Project No. 179826, Funding Reference: SIFN-01-9999-FN-179826).

Statement of Integrity

I hereby declare having conducted this academic work with integrity.
I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.
I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

University of Minho, Braga, December 2024

Assinado por: **Miguel Caçador Peixoto**
Num. de Identificação: 15733983
Data: 2024.12.14 18:38:19+01'00'

Miguel Caçador Peixoto

Abstract

This thesis reports on the development of an automated anomaly detection (AD) system for industrial quality control (QC), addressing inefficiencies of manual inspections and current practices. Leveraging machine learning and computer vision techniques, this work optimizes QC using AD algorithms. The research includes a review of AD techniques, algorithm development, hyperparameter optimization, performance evaluation, real-time implementation, and API backend creation. Central to this work is a framework built on an autoencoder (AE) architecture, optimized with a custom objective function that leverages data augmentation techniques to be trained using only defect-free images. Designed as a plug-and-play solution, it can be deployed across various production lines via API calls. This study assessed four methodologies, each building upon the one before, except for the variational approach, which showed no improvements. The best strategy involved a refined hyperparameter optimization coupled with advanced data augmentation techniques. The overall results were primarily attributed to the developed post-processing techniques and the use of the BCE metric for evaluation. In academic evaluations on the Metal Nut subset of the MVTec AD dataset, this work surpassed SOTA in the MSE AE regime, achieving an AUC of 0.87 and an IoU of 0.37, outperforming handcrafted methods in the literature by $\approx 1.2\%$ and $\approx 42.3\%$, respectively. Although only slightly outperformed in AUC by the top SOTA method, which achieved 0.89, the methodology still delivered highly competitive results. In collaboration with industrial partners, the system successfully detected anomalous parts using a real-world dataset.

The results validate the novel methodology and showcases the potential of data-augmented NOKs as an alternative to real-world defects. Aligned with Industry 4.0 objectives, this work contributes to the advancement of smarter, data-driven manufacturing processes, facilitating the development of more automated quality control systems, reducing industry waste, and improving market competitiveness.

Keywords Anomaly Detection, Quality Control, Computer Vision, Autoencoders, Waste Reduction, Machine Learning, Real-Time Implementation, Hyperparameter Optimization, Semi-Supervised Learning, Defective Products, Machine Learning Backend.

Resumo

Esta tese relata o desenvolvimento de um sistema automatizado de deteção de anomalias (DA) para controlo de qualidade (CQ) industrial, abordando ineficiências das inspeções manuais e práticas atuais. Utilizando técnicas de aprendizagem automática e visão computacional, este trabalho otimiza o CQ com algoritmos de DA.

A investigação cobre uma revisão de técnicas de DA, desenvolvimento e otimização de algoritmos, avaliação de desempenho, implementação em tempo real e criação de uma API. A framework central é baseada num autoencoder (AE) otimizado com uma função objetivo personalizada e aumento de dados, treinado apenas com imagens sem defeitos, e concebido como uma solução *plug-and-play* implementável em várias linhas de produção via API.

Este estudo analisou quatro metodologias, cada uma incremental em performance, exceto uma abordagem variacional, sem melhorias. A melhor estratégia combinou otimização de hiperparâmetros e aumento de dados. Os resultados devem-se principalmente às técnicas de pós-processamento e à métrica BCE para avaliação.

Em avaliações no subset Metal Nut do MVTec AD, este trabalho superou o SOTA no regime MSE AE, atingindo AUC de 0.87 e IoU de 0.37, superando métodos tradicionais da literatura em $\approx 1.2\%$ e $\approx 42.3\%$, respetivamente. Ligeiramente inferior ao melhor AUC do SOTA (0.89), a metodologia ainda assim apresentou resultados altamente competitivos. Em colaboração com parceiros industriais, o sistema detetou com sucesso peças anómalas num conjunto de dados real.

Os resultados validam a metodologia inovadora e demonstram o potencial de NOKs aumentados como alternativa a defeitos reais. Alinhado com a Indústria 4.0, este trabalho contribui para manufatura mais inteligente, reduzindo o desperdício e aumentando a competitividade.

Palavras-chave Deteção de Anomalias, Controlo de Qualidade, Visão Computacional, Autoencoders, Redução de Desperdício, Aprendizagem Automática, Implementação em Tempo Real, Otimização de Hiperparâmetros, Aprendizagem Semi-Supervisionada, Produtos Defeituosos, Backend de Aprendizagem Automática.

Contents

1 Introduction **1**

1.1 State of the Art in Manufacturing . . . 3

1.2 Problem Statement . . . 4

1.3 MVTec Dataset . . . 5

2 Methodology **9**

2.1 Autoencoders . . . 9

2.1.1 Convolutional Autoencoder . . . 10

2.1.2 Layers of a Convolutional Autoencoder . . . 11

2.1.3 Training details . . . 15

2.1.4 Hyperparameter Optimization Pipeline . . . 18

2.1.5 Model Creation Using Optuna . . . 23

2.1.6 Performance Enhancing Post-Processing Techniques . . . 27

2.2 Variational Autoencoders . . . 29

2.2.1 Key Concepts . . . 31

3 Results **35**

3.1 Traditional Approach . . . 35

3.1.1 Test Dataset and Performance Analysis . . . 36

3.2 Convolutional Autoencoder . . . 38

3.2.1 Optimization Process . . . 38

3.2.2 Test Results . . . 42

3.3 Improved CAE . . . 52

3.3.1 Optimization Process . . . 53

3.3.2 Test Results . . . 55

3.4 Variational AutoEncoder . . . 64

3.4.1 Optimization Process . . . 65
3.4.2 Test Results . . . 68

4 Industrial System **79**
4.1 API Architecture . . . 80
4.1.1 Endpoint: `/api/train` . . . 81
4.1.2 Endpoint: `/api/inference` . . . 81
4.1.3 Example messages . . . 82
4.1.4 API Configuration . . . 83
4.2 Repository Overview . . . 85
4.3 Real Time Implementation . . . 85

5 Conclusion **87**
5.1 Prospect for future work . . . 89

A Appendix **101**
A.1 Data Augmentation Details . . . 101
A.1.1 Traditional Data Augmentation Techniques . . . 101
A.1.2 Generating Realistic NOKs . . . 103
A.2 Precomputations . . . 107
A.3 Image Filtering Techniques . . . 109
A.4 Supplementary Results - Traditional Approach . . . 116
A.5 Supplementary Results - Convolutional AutoEncoder . . . 119
A.5.1 Validation Results . . . 119
A.5.2 Additional Validation Results . . . 127
A.5.3 Additional Test Results . . . 131
A.6 Supplementary Results - Improved CAE . . . 133
A.6.1 Validation Results . . . 134
A.6.2 Additional Validation Results . . . 137
A.6.3 Additional Test Results . . . 139
A.7 Supplementary Results - VAE . . . 141
A.7.1 Validation Results . . . 142
A.7.2 Additional Validation Results . . . 146
A.7.3 Additional Test Results . . . 147

A.7.4 Other VAE-specific approaches 148

List of Figures

1 The typical cycle of a **QC** process in the industry. . . . 3

2 Demonstration of hardcoded rules in a modern **QC** industry. Rules A to D are used to check part tolerances in its dimensions and rule E checks for black texture in the red region. . . . 4

3 The proposed cycle of a **QC** process in the industry. Note the threshold for a part to be considered NOK is automatically determined by the model by a quick evaluation performance procedure done after training. . . . 5

4 Excerpt of images in the MVTec Dataset. . . . 6

5 NOK images samples of the metal nut dataset. . . . 6

6 Illustration of a convolutional autoencoder architecture. . . . 10

7 Illustration of a Convolutional Layer Operation. . . . 12

8 Illustration of a 2×2 Max Pooling Operation. . . . 13

9 Dropout Neural Net Model. . . . 17

10 Illustration of OK/NOK distributions. . . . 19

11 EIllustration of normalized OK/NOK distributions. . . . 20

12 Objective function candidate metrics. . . . 20

13 Computed objective function and components. . . . 22

14 Clustering pipeline. . . . 29

15 VAE architecture. . . . 31

16 Reparameterization trick. . . . 32

17 Reconstruction of the test dataset using the traditional approach. . . . 37

18 HP optimization history for the CAE architecture. . . . 39

19 HP parameter importance for the CAE architecture. . . . 39

20 Empirical distribution for the CAE architecture. . . . 41

21 Slice plot of non-layer specific HP for the CAE architecture. 42
22 Slice plot of layer specific HP for the CAE architecture. 43
23 Test dataset reconstruction using CAE architecture. 44
24 Pairplot of Latent Space Metrics. 47
25 PCA projection of the latent space 48
26 Comparison of AUC and IoU for filtering techniques. 49
27 Comparison of AUC and IoU with Ground Truth for clustering techniques. . . 50
28 HP optimization history for the improved CAE architecture. 54
29 HP parameter importance for the improved CAE architecture. 56
30 Empirical distribution for the improved CAE architecture. 56
31 Test dataset reconstruction using CAE architecture. 57
32 Pairplot of Latent Space Metrics. 60
33 PCA projection of the latent space 61
34 Comparison of AUC and IoU for filtering techniques. 62
35 Comparison of AUC and IoU with Ground Truth for clustering techniques. . . 63
36 HP optimization history for the VAE architecture. 66
37 HP parameter importance for the VAE architecture. 66
38 Empirical distribution for the VAE architecture. 68
39 Test dataset reconstruction using CAE architecture. 69
40 Pairplot of Latent Space Metrics. 72
41 PCA projection of the latent space 73
42 Comparison of AUC and IoU for filtering techniques. 73
43 Comparison of AUC and IoU with Ground Truth for clustering techniques. . . 74
44 Histograms of the latent space parameters for the test dataset. 76
45 2D PCA projection of the latent space of the test dataset. 77

46 Birds-eye view of the backend. 79
47 VIGIL logo. 85

48 Traditional data augmentation techniques. 101
49 Exemplary process of how Stable Diffusion operates. 104
50 Detailed process of data augmentation using "half mask" on the metal nut dataset, displaying the original, the mask, and the augmented NOK image. . 104

51 Different masks tested for generating artificial NOK conditions. 105
52 Examples of NOKs produced using the "half mask" technique. 105
53 Examples of NOKs produced using the "quarter mask" technique. 106
54 Example of image of an "J". 110
55 "J" image after erosion filter. 111
56 "J" image after dilation filter. 112
57 Canny filter exemplification. 112
58 Median filter exemplification. 114
59 Gaussian blur exemplification. 115
60 Example of usage of NLM techniques. 115
61 Optimization history for the traditional approach 117
62 Parameter importance for the traditional approach 118
63 **EDF** for the **HP** optimization of the traditional approach. 119
64 Slice plot of the non-layer specific HP for the traditional approach. 119
65 Slice plot of the layer specific HP for the traditional approach. 120
66 Reconstruction of the validation dataset using the CAE architecture. 121
67 Histogram of validation dataset MSE reconstruction loss for CAE architecture. 122
68 Threshold evaluation with baseline approach for the **CAE** architecture. 123
69 Dimensionality Reduction Using PCA. 124
70 Thresholds for filtering techniques using the 90th percentile of OK-error values. 125
71 Comparative analysis of AUC and IoU metrics. 126
72 Clustering performance evaluated using AUC and IoU metrics. 127
73 Pairplot of metrics of the validation dataset using the CAE architecture. . . . 128
74 Pairplot visualization of the **MSE** reconstruction loss and the latent space statistical summaries. 129
75 Pairplot of the **PCA**-transformed latent space 130
76 First validation set of filtering techniques for the CAE architecture. 149
77 Second validation set of filtering techniques for the CAE architecture. 150
78 First validation set of clustering techniques for the CAE architecture. 151
79 Second validation set of clustering techniques for the CAE architecture. . . . 152
80 Histogram of the MSE reconstruction loss for the test dataset using the **CAE** architecture. 153

81 Pairplot of the validation dataset using the **CAE** architecture. 153
82 First test set of filtering techniques for the CAE architecture. 154
83 Second test set of filtering techniques for the CAE architecture. 155
84 First test set of clustering techniques for the CAE architecture. 156
85 Second test set of clustering techniques for the CAE architecture. 157
86 **BCE** performance on the test dataset using different filtering techniques. . . 158
87 **BCE** performance on the test dataset using diferent clustering techniques. . . 158
88 Slice plot of non-layer specific HP for the improved CAE architecture. 159
89 Slice plot of layer specific HP for the improved CAE architecture. 160
90 Reconstruction of the validation dataset using the CAE architecture. 161
91 Histogram of validation dataset MSE reconstruction loss for CAE architecture. 161
92 Threshold evaluation with baseline approach for the improved CAE architecture. 162
93 Dimensionality Reduction Using PCA. 162
94 Thresholds for filtering techniques using the 90th percentile of OK-error values. 163
95 Comparative analysis of AUC and IoU metrics. 163
96 Clustering performance evaluated using AUC and IoU metrics. 164
97 Pairplot of metrics of the validation dataset using the CAE architecture. . . . 165
98 Pairplot visualization of the **MSE** reconstruction loss and the latent space statistical summaries. 166
99 Pairplot of the **PCA**-transformed latent space 167
100 First validation set of filtering techniques for the improved CAE architecture. 168
101 Second validation set of filtering techniques for the improved CAE architecture. 169
102 First validation set of clustering techniques for the improved CAE architecture. 170
103 Second validation set of clustering techniques for the improved CAE architecture. 171
104 Histogram of the **MSE** reconstruction loss for the test dataset using the improved **CAE** architecture. 172
105 Pairplot of the validation dataset using the improved **CAE** architecture. . . . 172
106 First test set of filtering techniques for the improved CAE architecture. 173
107 Second test set of filtering techniques for the improved CAE architecture. . . 174
108 First test set of clustering techniques for the improved CAE architecture. . . . 175

109 Second test set of clustering techniques for the improved CAE architecture. . 176
110 **BCE** performance on the test dataset using different filtering techniques. . . 177
111 **BCE** performance on the test dataset using diferent clustering techniques. . . 177
112 Slice plot of non-layer specific HP for the improved CAE architecture. 178
113 Slice plot of layer specific HP for the improved CAE architecture. 179
114 Reconstruction of the validation dataset using the CAE architecture. 180
115 Histogram of validation dataset MSE reconstruction loss for CAE architecture. 180
116 Threshold evaluation with baseline approach for the VAE architecture. 181
117 Dimensionality Reduction Using PCA. 181
118 Thresholds for filtering techniques using the 90th percentile of OK-error values. 182
119 Comparative analysis of AUC and IoU metrics. 182
120 Clustering performance evaluated using AUC and IoU metrics. 183
121 Pairplot of metrics of the validation dataset using the CAE architecture. . . . 184
122 Pairplot visualization of the **MSE** reconstruction loss and the latent space statistical summaries. 185
123 Pairplot of the **PCA**-transformed latent space 186
124 First validation set of filtering techniques for the improved CAE architecture. 187
125 Second validation set of filtering techniques for the improved CAE architecture. 188
126 First validation set of clustering techniques for the improved CAE architecture. 189
127 Second validation set of clustering techniques for the improved CAE architecture. 190
128 Histogram of the **MSE** reconstruction loss for the test dataset using the **VAE** architecture. 191
129 Pairplot of the validation dataset using the **VAE** architecture. 191
130 First test set of filtering techniques for the improved CAE architecture. 192
131 Second test set of filtering techniques for the improved CAE architecture. . . 193
132 First test set of clustering techniques for the improved CAE architecture. . . . 194
133 Second test set of clustering techniques for the improved CAE architecture. . 195
134 **BCE** performance on the test dataset using different filtering techniques. . . 196
135 **BCE** performance on the test dataset using different clustering techniques. . 196
136 Histograms of the latent space parameters for the validation dataset. 197
137 2D PCA projection of the latent space for the validation dataset 198

List of Tables

1 Hyperparameter selection range. 35
2 Comparison of metrics on classification power. 36
3 Table of **IoU** values for different metrics 37
4 Best set of HP for the CAE architecture. 40
5 Evaluation metrics of the test dataset. 44
6 Table of Metric Values 45
7 Performance metrics using different thresholds for the test dataset. 45
8 Confusion Matrices for different thresholds. 45
9 Confusion Matrix of the **SVM** model 46
10 Comparison of AUC and IoU for MSE and BCE. 50
11 Comparison of AUC and IoU values for filtering techniques. 51
12 Comparison of AUC and IoU values for clustering techniques. 51
13 Hyperparameter selection range for the improved CAE architecture. 53
14 Best set of HP for the improved CAE architecture. 55
15 Evaluation metrics of the test dataset. 58
16 Table of Metric Values 58
17 Performance metrics using different thresholds for the test dataset. 58
18 Confusion Matrices for different thresholds. 59
19 Confusion Matrix of the **SVM** model 60
20 Comparison of AUC and IoU for MSE and BCE. 63
21 Comparison of AUC and IoU values for filtering techniques. 64
22 Comparison of AUC and IoU values for clustering techniques. 64
23 Latent space components averages for the VAE architecture. 65
24 Best set of HP for the VAE architecture. 67
25 Evaluation metrics of the test dataset. 69

26 Table of Metric Values 70
27 Performance metrics using different thresholds for the test dataset. 70
28 Confusion Matrices for different thresholds. 71
29 Confusion Matrix of the **SVM** model 71
30 Comparison of AUC and IoU for MSE and BCE. 77
31 Comparison of AUC and IoU values for filtering techniques. 78
32 Comparison of AUC and IoU values for clustering techniques. 78
33 Example messages for the API endpoints, illustrating the input payload and output response for each endpoint. 83
34 Optimized **HP** set for the best trial. 117
35 Evaluation metrics of the validation dataset. 121
36 Table of **IoU** values for different metrics 122
37 Optimal threshold values for metrics. 123
38 Evaluation metrics of the validation dataset. 134
39 Table of **IoU** values for different metrics 135
40 Optimal threshold values for metrics. 135
41 Evaluation metrics of the validation dataset. 142
42 Table of **IoU** values for different metrics 143
43 Optimal threshold values for metrics. 144

Acronyms

AD Anomaly Detection.

AE Autoencoder.

AUC Area Under the Curve.

BCE Binary Cross-Entropy.

CAE Convolutional Autoencoder.

EDF Empirical Distribution Function.

GPU Graphics Processing Unit.

HP Hyperparameter.

HPO Hyperparameter Optimization.

IoU Intersection over Union.

KL Kullback-Leibler.

L1 L1 Loss.

L2 L2 Loss.

LR Learning Rate.

MAE Mean Absolute Error.

ML Machine Learning.

MSE Mean Squared Error.

NLM Non-local Means Denoising.

NRMSE Normalized Root Mean Squared Error.

PCA Principal Component Analysis.

PSNR Peak Signal-to-Noise Ratio.

QC Quality Control.

SGD Stochastic Gradient Descent.

SOTA State of the Art.

SSIM Structural Similarity Index.

SVM Support Vector Machine.

SWA Stochastic Weight Averaging.

VAE Variational Autoencoder.

Chapter 1

Introduction

Over the past decades, advancements in computing technology and the rise of the internet, coupled with various algorithmic innovations, have heralded a new era of digital transformation Usmani et al. (2022); Cui et al. (2022). This revolution has profoundly impacted nearly all facets of human life and various sectors. Particularly notable is the emergence of what is often referred to as the "Industrial Revolution 4.0" Hozdić (2015). This phase is marked by trends in smart automation, extensive data sharing not only among humans but also between machines (as seen in the Internet of Things), and the growing reliance on cloud computing.

A critical component in the manufacturing industry is **Quality Control** (**QC**). This process involves manufacturers checking whether their products meet all set quality standards. Traditionally, this task was carried out by human professionals through manual visual inspections. However, this method is prone to human error, as fatigue and distractions can lead to mistakes leading to a waste of material and production time Judi et al. (2011). **QC** is not just a crucial operational process; it has significant financial implications where the costs associated with selling defective products are substantial, leading to warranty claims, customer dissatisfaction, and a decline in sales Ahmed et al. (2019). Employing skilled professionals to manually perform **QC** on every product is not only resource-intensive but can also be inefficient.

Automating **QC** is thus a priority to any manufacturer's business, it allows the task to be performed more efficiently and with fewer costs. Thus minimizing expenses by reducing or eliminating the need for inspection staff, and often, the performance of automated models matches or surpasses human capabilities. Automated visual inspection models are advantageous as they are not prone to human limitations such as fatigue and can operate continuously without breaks. Typically, these models evaluate product images, identifying whether they are acceptable ("OK") or defective ("NOK").

The challenge thus lies in distinguishing anomalous images from the majority of OK ones, a task known as **Anomaly Detection**, which is a crucial process that involves determining the presence of abnormal patterns within a given image Chalapathy and Chawla (2019). This task relies on a dataset comprising both normal and abnormal images to effectively identify deviations from the norm. This challenge is pervasive across diverse domains, including not just industrial applications for detecting defective objects Bergmann et al. (2019, 2020); Dinh et al. (2016) but also medical imaging Schlegl et al. (2017); Seeböck et al. (2016), video surveillance systems Adam et al. (2008); Liu et al. (2018); Sultani et al. (2018), fraud detection Pourhabibi et al. (2020), insurance Lu et al. (2020), cyber security Vanerio and Casas (2017), particle physics Belis et al. (2024); Schuhmacher et al. (2023); Crispim Romão et al. (2021); Peixoto et al. (2023), among many others.

Detecting anomalies presents a unique and intricate challenge due to their inherently low probability density Chandola et al. (2009). As such, addressing this challenge often requires the application of semi-supervised learning methodologies. In these approaches, models exclusively rely on defect-free samples during the learning phase, aiming to discern and accurately localize anomalies during inference. This approach proves particularly relevant in situations where enumerating all possible anomalies is a formidable task. Furthermore, the utilization of semi-supervised learning not only enhances the model's ability to handle low-density anomalies but also contributes to its adaptability across a wide range of applications and domains.

To address the intricate task of **AD** in an industrial environment, this master's thesis leverages the power of an **Autoencoder** (**AE**) enhanced with convolutional layers Zhou and Paffenroth (2017); Ribeiro et al. (2018); Saeedi and Giusti (2023); Zouar (2021) for vision-based **AD**. **AE**s, a type of neural network architecture, are well-suited for capturing complex patterns in high-dimensional data, making them an ideal choice for image-based **AD** tasks. The incorporation of convolutional layers enhances the **AE**'s ability to extract spatial features from images, enabling it to discriminate effectively between OK and NOK images. This thesis will delve into the design, implementation, and evaluation of such **AE**-based **AD** models, aiming to provide practical solutions for automating **QC** processes in industrial settings culminating in a real world pipeline for **QC** in the manufacturing industry.

1.1 State of the Art in Manufacturing

In modern industries **QC** is not exclusively carried out by human operator. Instead, there are also algorithms in play, usually tools provided by the manufacturer of the machine that uses manually configured rules to detect defects. When a new part enters production, its typical **QC** lifecycle is the one shown at Figure 1. The process starts by a qualified **QC** technician that, with conjunction with production workers, defines a set of standards or rules in specialized software that the part must fulfill to be considered OK. Then, production parts and the maintenance the **QC** begins, where multiple refinements are made and there is an continuous improvement being made to the rules, which usually lasts until the end of the production cycle.

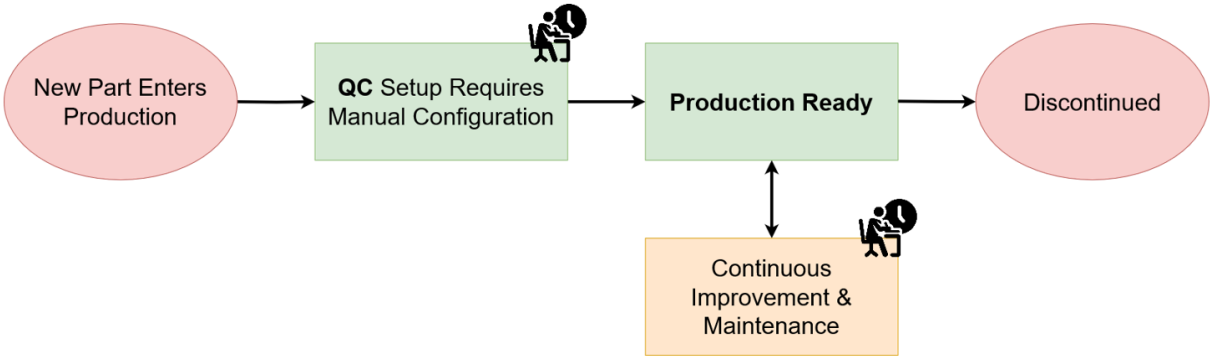


Figure 1: The typical cycle of a **QC** process in the industry.

An example of the standards can be seen on Figure 2. Note that these are often based on the experience of the technician, which in turn explain the interactive process. In industrial parts there are usually hundreds of these conditions that test if dimensions are inside the tolerances specified, if the texture correspond, if the color is the right one, etc. This process is not only time consuming but also prone to human error, as the operator can miss a possible restriction or the rules can be too strict or too loose. It also provides low interpretability, when a part is found to have a defect only an error code is given (e.g. "Failed condition E"), which is then used to search the rule that was violated.

The current **SOTA** in **QC** lacks flexibility for fast-paced industries. The iterative nature of the process often leads to significant waste, as production may commence with defective parts before **QC** refinements are implemented—resulting in wasted materials and money. Furthermore, production halts whenever configuration errors arise, causing time-consuming delays. Additionally, the system is not easily adaptable to variations in raw materials, such as

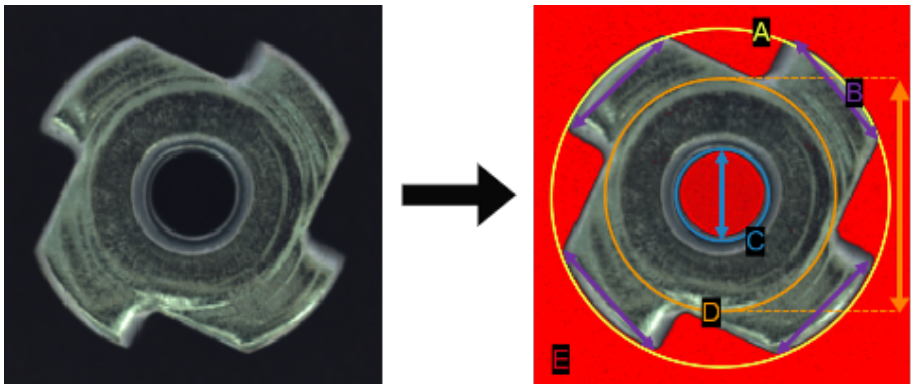


Figure 2: Demonstration of hardcoded rules in a modern **QC** industry. Rules A to D are used to check part tolerances in its dimensions and rule E checks for black texture in the red region.

changes in color or texture from different suppliers. In summary, this approach represents poor resource management, with skilled technicians diverted from improving manufacturing processes to continuously refining **QC** software.

1.2 Problem Statement

The primary challenge of this thesis is to develop an efficient and robust **ML** model that automates **AD** in an industrial setting. The proposed new cycle for **QC** is shown in Figure 3. When a new part enters production, the pipeline is trained on a set of defect-free images which combined with data augmentation can be adaptable to any variations required (e.g. part rotation, lighting conditions, color). The model is then deployed in the production line, where it continuously inspects the parts. When a defect is detected, the model provides instant interpretable feedback to the operator with an error map, highlighting the anomaly's location. In contrast to the current **SOTA**, this approach offers greater flexibility and adaptability. Once production begins (given sufficient computational resources), the system is immediately operational without the need for ongoing rule adjustments or fine-tuning. It requires no prior knowledge of the part and delivers real-time feedback to operators. This not only makes it more environmentally friendly by reducing waste but also more cost-effective, as the continuous iterative improvements are eliminated, ensuring production runs uninterrupted.
An aspect of this research is its alignment with the broader goals of industry 4.0 technologies. According to recent studies Laskurain-Iturbe et al. (2021); Weltwirtschaftsforum (2020), 44 companies worldwide achieved an average waste reduction of more than 40% after imple-

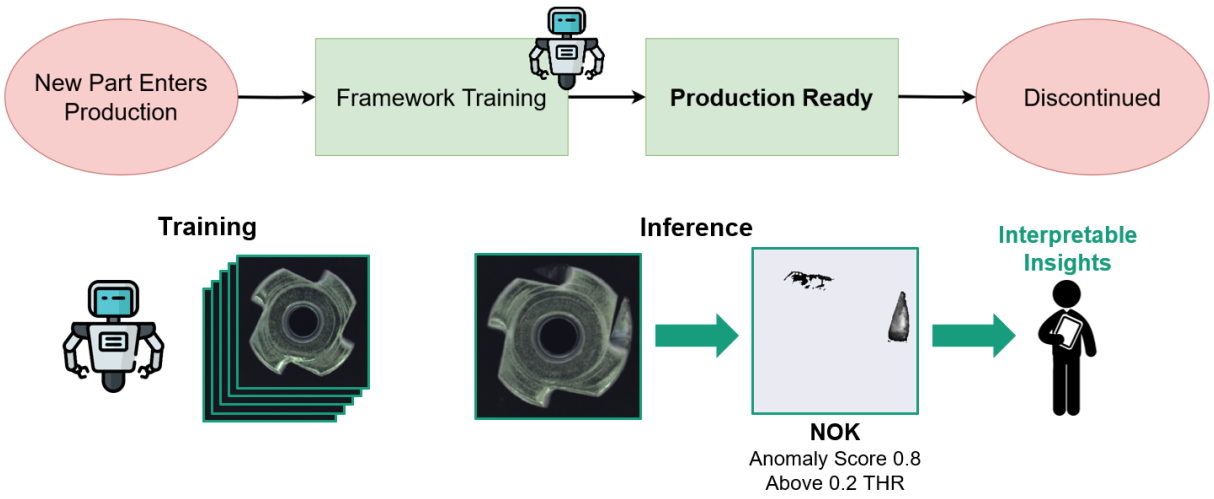


Figure 3: The proposed cycle of a **QC** process in the industry. Note the threshold for a part to be considered NOK is automatically determined by the model by a quick evaluation performance procedure done after training.

menting industry 4.0 technologies. Notably, industry reports from Beijing indicate that **QC** and management not only improved product quality and customer satisfaction by 40% but also resulted in 80% fewer incidents per thousand products and 30% less time required to identify quality issues. Similarly, in Indonesia, there was a 40% reduction in costs associated with raw materials due to decreased waste. All of this was due to increased digitalization and application of data-driven processes in the manufacturing industry, especially **ML** applied to **QC** processes Nagorny et al. (2017).

Given the complexity and variability of industrial products, this thesis explores and develops **ML** techniques with a particular focus on **CAE**s with convolutional layers. The proposed model will be trained and evaluated on the MVTec AD Dataset Bergmann et al. (2019). This dataset provides a realistic representation of the challenges faced in industrial **QC**. The effectiveness of the model will be benchmarked against this dataset, ensuring its relevance and applicability to real world scenarios.

1.3 MVTec Dataset

The MVTec AD Dataset Bergmann et al. (2019) is an well know resource in the field of industrial **AD**. It's a comprehensive collection of high-resolution images, showcasing a variety of industrial products and materials. This dataset is a compilation of images of defect-free products and parts bearing real-world imperfections like scratches and dents (Figure 4).

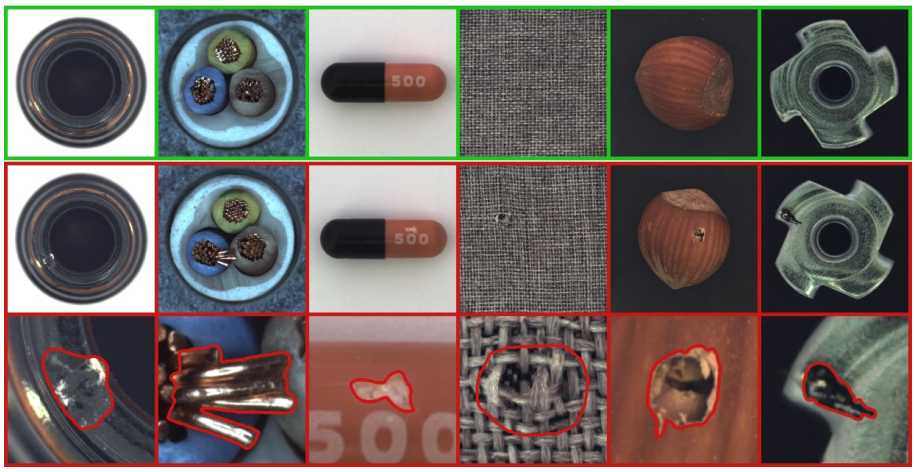


Figure 4: Excerpt of images contained in the MVTec Dataset Bergmann et al. (2019)

This will serve as a testbed for training and evaluating **ML** models. The dataset is divided into 15 categories, each containing a set of images of a specific product or material. Each category is divided into two subsets: training and test. The training set contains only OK images, whereas the test set contains both OK and NOK images (where there are multiple classes of NOK images - different categories of anomalies). Apart from this there is also a ground truth folder containing the ground truth masks of where the anomaly is located for the NOK images. The ground truth masks are binary images that will be to evaluate the model's performance in terms of **IoU**.

For the course of this thesis only one dataset will be considered, the metal nut dataset. Its comprised of 700 by 700 pixel images of metal nuts, with 220 OK images for training and 22 OK images for testing. The NOK images are divided into four categories: bent, color, flip, and scratch (Figure 5), each containing 22 images.

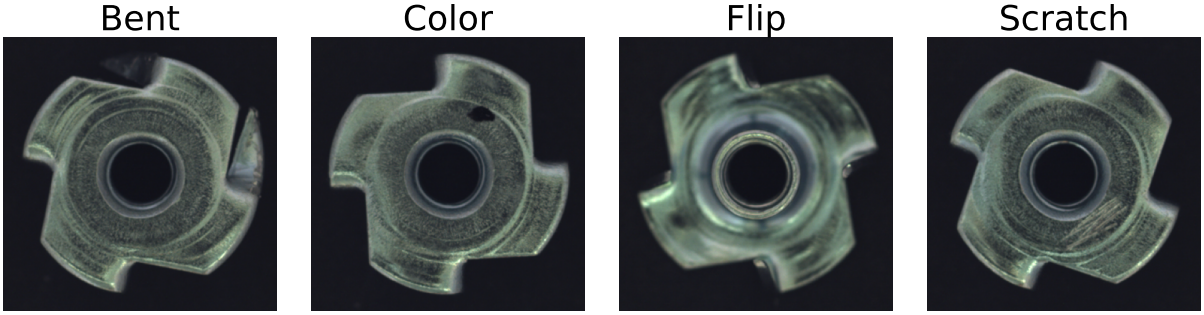


Figure 5: NOK images samples of the metal nut dataset.

The data was separated into three datasets: training, validation, and testing. For this purpose, 22 images were randomly selected from the training folder to form the validation dataset.

Consequently, an 80/10/10 distribution of all OK images across the training, validation, and testing datasets was chosen. The testing dataset remained unchanged to ensure comparability with other studies.

The reader will see along this thesis that data augmentation was done in order to artificially create NOK images, due to the necessity of NOK images for the methodologies developed. To maintain consistency with other research papers, the original NOK images in the testing dataset were left untouched, while new, augmented NOK images were added to the validation dataset.

Our industrial partner, *Bosch Car Multimédia Portugal*, highlighted that typical defect inspection setups use lower resolution cameras compared to the MVTec dataset. While MVTec captures finer details, making the task more challenging, it also demands more computational resources. To address real-world conditions, the model will first be trained on the MVTec dataset. Afterward, it will be tested and evaluated using a dataset provided by *Bosch Car Multimédia Portugal*, reflecting actual industrial **QC** conditions to ensure its practical applicability.

Chapter 2

Methodology

2.1 Autoencoders

Autoencoder (**AE**) are a valuable tool in **Anomaly Detection** (**AD**), offering a clear approach through their specific design and training methods. An **AE** is an artificial neural network used for semi-supervised learning, focusing on capturing the key features of input data. Its main goal is to learn how to encode and decode data efficiently, allowing it to accurately reconstruct the input.

The architecture consists of an encoder and a decoder, each playing a role in the model's ability to identify normal patterns and detect anomalies. The encoder compresses the input data into a lower-dimensional representation, known as a latent space or bottleneck. The decoder then reconstructs the original input from this lower-dimensional representation. The reconstruction is compared to the input data, and the model is trained to minimize the reconstruction error, such as the mean square error between the input and output.

In **AD**, the **AE**'s training primarily involves exposing it to a dataset of normal instances. The model thus learns to reconstruct these instances accurately, emphasizing the extraction of essential features for faithful reproduction. Since the **AE** has been trained on normal instances, and anomalies are often dissimilar, they typically yield reconstruction errors that differ significantly from those associated with normal data.

This lays the foundation of why **AE** are useful for **AD**. Instances deviating from the learned normal patterns exhibit higher reconstruction errors, serving as a reliable indicator of anomalous behavior. This mechanism aligns with the semi-supervised nature of **AD**, where the model relies solely on normal samples during training, yet exhibits the capability to identify anomalies without any explicit labels. This offers a significant advantage over traditional supervised learning methods, as it requires considerably less training data and eases the bur-

den on the factory operator, who otherwise would need to classify every possible defect scenario for each new piece they produce.

2.1.1 Convolutional Autoencoder

In the specific context of visual **AD**, a **Convolutional Autoencoder** (**CAE**, shown in Figure 6) will be utilized to enhance the model's ability with images. **CAE**s extend the basic **AE** architecture by incorporating convolutional layers, which effectively process visual data. Convolutional layers enable the **CAE** to learn hierarchical features by capturing local patterns, making them well-suited for tasks involving image data. This architectural enhancement improves the model's ability to distinguish subtle visual characteristics of industrial parts, thus increasing **AD** accuracy.

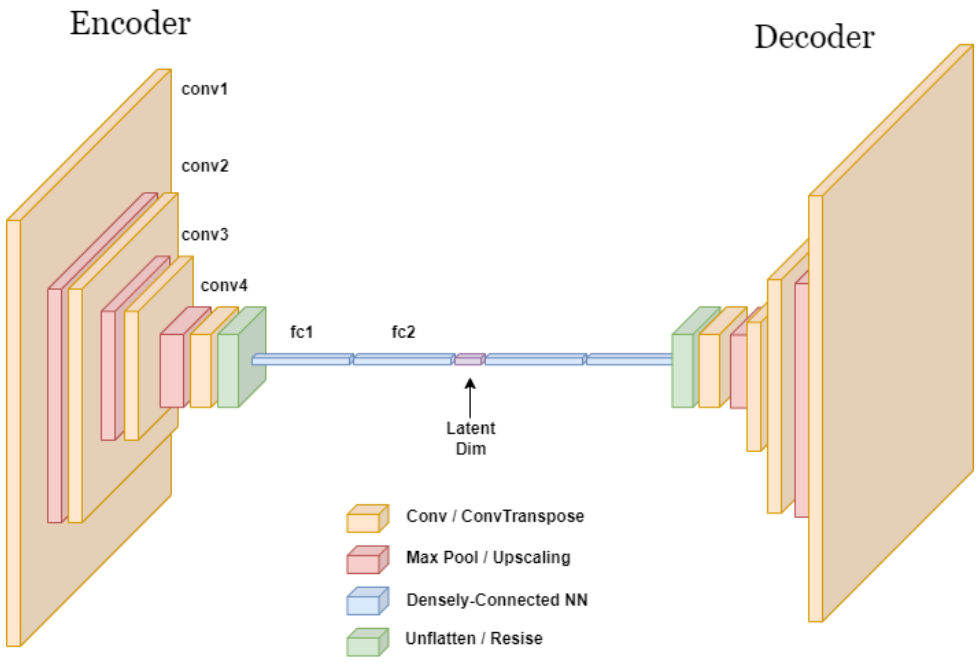


Figure 6: Illustration of a **CAE** architecture, featuring a symmetrical design where the decoder mirrors the encoder. The encoder, detailed on the left side of the legend, compresses the input, while the decoder, corresponding to the right side of the legend, reconstructs the output from the encoded data. Here conv1 to conv4 represent convolutional layers, and f1 and f2 represent fully connected layers.

2.1.2 Layers of a Convolutional Autoencoder

The architecture of a **CAE** consists of several key layers. Understanding the intricacies of these layers provides insight into how this architecture captures hierarchical features in images. Special attention is warranted for convolutional layers. Given a set of kernel size, stride, and padding, their output shapes may differ, so a low-level understanding of how they work is key for future **Hyperparameter** (**HP**) optimization.

The tensor shape that different layers deal with has a format of $(N, C_{in}, H_{in}, W_{in})$, where N represents the batch size, C_{in} the number of input channels, and H_{in} and W_{in} to the height and width of the input respectively. However, in the context of the dataset utilized in this thesis, only square images will be employed, where the width and height are both L_{in}. Consequently, the input tensor shape will be $(N, C_{in}, L_{in}, L_{in})$, and the output tensor shape will be denoted as $(N, C_{out}, L_{out}, L_{out})$. This adjustment simplifies the equations and formulas, while the underlying concepts remain the same.

2.1.2.1 Encoder Layers

1. *Convolutional Layers:*

   Convolutional layers, represented as Conv2D$(C_{in}, C_{out}, K, S, P, D)$, are adept at processing images by capturing local patterns like edges, textures, and shapes. They use learnable filters to create distinct feature maps, with key hyperparameters including kernel size (K) that determines filter dimensions and the input processing area, stride (S) which affects the filter's movement step size and output dimensions, padding (P) that maintains the desired output dimensions by adding zeros around the input, and dilation (D) which influences the receptive field by spacing filter elements.

   The output dimensions are calculated using the input dimensions and convolution parameters as follows (Dumoulin and Visin (2016); Paszke et al. (2019)).

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times P_0 - D_0 \times (K_0 - 1) - 1}{S_0} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times P_1 - D_1 \times (K_1 - 1) - 1}{S_1} + 1 \right\rfloor$$

For the course of this thesis, only square images will be used, thus simplifying both formules to:

$$L_{out} = \frac{L_{in} + 2 \times P - D \times (K - 1) - 1}{S} + 1$$

Furthermore, the dilation will be fixed to 1, further simplifying the formula to:

$$L_{out} = floor\left(\frac{L_{in} + 2 \times P - K}{S} + 1\right) \tag{2.1}$$

Note that $floor$ is there to ensure that the output is an integer.

An example of a Conv2D layer operation is illustrated in Figure 7. In this example, a 6×6 RGB image is processed using a Conv2D layer with two output channels, a 3×3 kernel size, a stride of 1, and no padding resulting in two 4×4 feature maps.

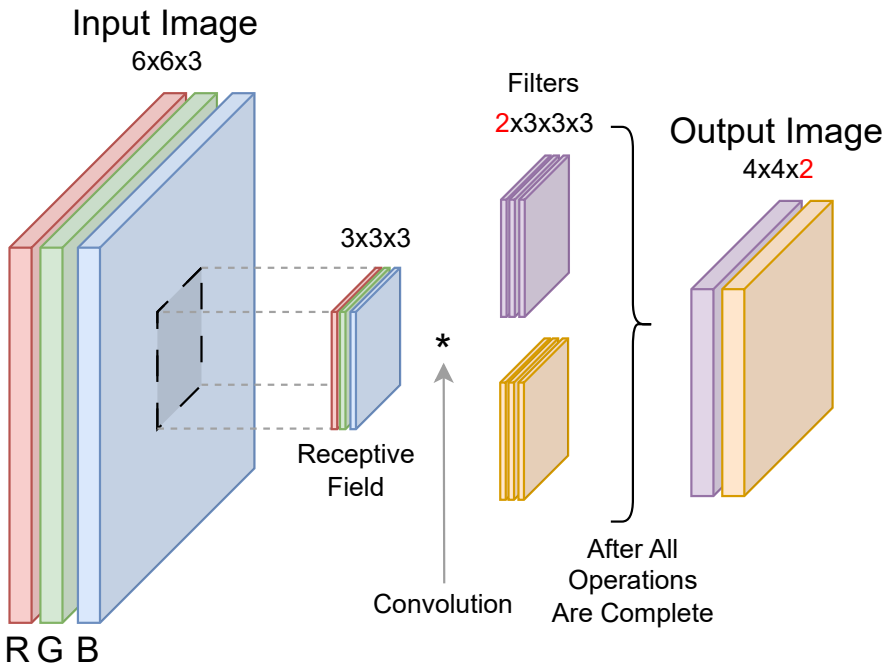


Figure 7: Illustration of a Conv2D(3, 2, 3, 1, 0, 1) layer operation applied to a 6×6 RGB image.

2. *Pooling Layers:*

   Pooling layers facilitate the downsampling of the spatial dimensions in the data, reducing computational complexity while (trying to) preserving critical features. Common pooling operations include max pooling, average pooling, and sum pooling but

preliminary experiments have shown that max pooling is the most effective technique thus will be used for the course of this work.

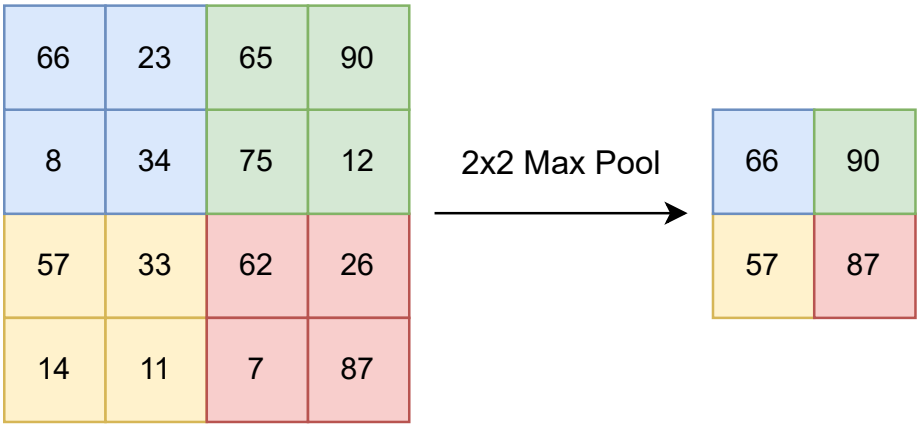


Figure 8: Illustration of a 2×2 Max Pooling Operation with a stride of 2.

Max pool works by selecting the maximum value within a local region, thereby reducing the spatial dimensions of the data (see Figure 8). The layer's configuration includes filter and stride sizes, which determine the size of the local region and the step size of the filter. The output dimensions are calculated using the input dimensions and pooling parameters as before:

$$L_{out} = \frac{L_{in} + 2 \times P - D \times (K - 1) - 1}{S} + 1$$

Now simplifying with the same assumptions as before:

$$L_{out} = floor\left(\frac{L_{in} + 2 \times P - K}{S} + 1\right) \tag{2.2}$$

Where the $floor$ function is used to ensure that the output is an integer.

Note that in addition to the output tensor, MaxPool2D (the max pool implementation in Paszke et al. (2019)) also provides the indices of the maximal values within the pooling window. These indices are used in the subsequent decoder unpooling stage, where the MaxUnpool2D layer, the inverse of MaxPool2D, is employed.

3. *Flatten Layer:*

Restructures the output from previous layers into a one-dimensional vector. This step is needed when transitioning from the encoder to the bottleneck layer. The input to this layer is a tensor of shape $(N, C_{in}, L_{in}, L_{in})$ and the output is a tensor of shape $(N, C \times L_{in} \times L_{in})$.

4. *Bottleneck/Latent Space Layer:*

   The bottleneck or latent space layer represents the compressed latent space, encapsulating the most salient features of the input data in a one dimensional vector. Its size is determined by an hyperparameter that can be tuned to optimize the model's performance.

2.1.2.2 Decoder Layers

1. *Unflatten Layer:*

   The unflatten layer transforms the one-dimensional vector back into a format compatible with the decoder layers, serving as the counterpart to the flattened layer in the encoder. The input to this layer is a tensor of shape $(N, C_{in} \times L_{in} \times L_{in})$ and the output is a tensor of shape $(N, C_{in}, L_{in}, L_{in})$.

2. *Unpooling Layers:*

   Unpooling layers are designed to reconstruct the spatial dimensions that are reduced during the pooling process. This layer operates by accepting information from MaxPool2D, which includes the maximal value indices and pooling parameters (such as kernel size, stride, and padding) and then generating a tensor that mirrors the shape of the original input to MaxPool2D.

   It is important to note that the MaxPool2D operation is not entirely reversible, as it discards non-maximal values during its process. Consequently, the MaxUnpool2D layer can only achieve a partial inversion, where the non-maximal values are assigned a value of zero.

   The input to MaxUnpool2D is a tensor with dimensions $(N, C_{in}, L_{in}, L_{in})$, and it produces a 4-dimensional tensor as output, $(N, C_{out}, L_{out}, L_{out})$. The dimensions of the output are determined based on the input dimensions and the pooling parameters (kernel size, stride, padding), according to the following formula (Dumoulin and Visin (2016); Paszke et al. (2019)):

$$L_{out} = (L_{in} - 1) \times S - 2 \times P + K \tag{2.3}$$

3. *Transposed Convolutional Layers:*

   Transposed convolutional layers, also known as fractionally-strided convolutions or deconvolutions (though not true deconvolutions as they do not compute an actual inverse of convolution), perform the inverse operation of convolutional layers. They upscale feature maps and reconstruct spatial dimensions of the input image. These layers apply a 2D transposed convolution operator over an input image composed of several input planes, essentially acting as the gradient of Conv2D with respect to its input.

   The key parameters of these layers are the same as the corresponding convolutional layers in the encoder. However, output padding and padding may require additional fine-tuning to achieve the desired output dimensions.

   The input to these layers is denoted as $(N, C_{in}, L_{in}, L_{in})$ and the output as $(N, C_{out}, L_{out}, L_{out})$. The output dimensions are calculated using the input dimensions and transposed convolution parameters as follows (Dumoulin and Visin (2016); Paszke et al. (2019)):

   $$L_{out} = (L_{in} - 1) \times S - 2 \times P + D \times (K - 1) + P_o + 1$$

   Now simplifying with the same assumptions as before:

   $$L_{out} = (L_{in} - 1) \times S - 2 \times P + K + P_o \tag{2.4}$$

   Where P_o is the output padding.

4. *Output Layer:*

   The final layer in the decoder adjusts the channel and image dimensions to match the desired output to produce the final reconstructed image.

2.1.3 Training details

For improve training speed, stability and results, several techniques were implemented. These include:

2.1.3.1 Effective Training Techniques

- *Gradient Clipping:*

  Limits the magnitude of the gradient during training. This helps to prevent the gradient from exploding or vanishing[1], thereby improving training.

  The global norm of the gradients will be clipped to a maximum value of 0.5 using the norm algorithm, as per implemented in Falcon and The PyTorch Lightning team (2019).

- ***Stochastic Weight Averaging*** *(**SWA**):*

  Improves model generalization by averaging neural network weights across multiple iterations during training. Instead of finding a single optimal set of weights w^*, **SWA** calculates the averaged weights $\bar{w} = \frac{1}{T}\sum_{t=1}^{T} w_t$ over time, starting after a warm-up period. This averaging leads to a flatter loss landscape, reducing the risk of sharp local minima, which results in improved performance on unseen data and reduces overfitting. The process requires no additional computational cost during inference and works well with common optimization methods like **SGD** and adam.

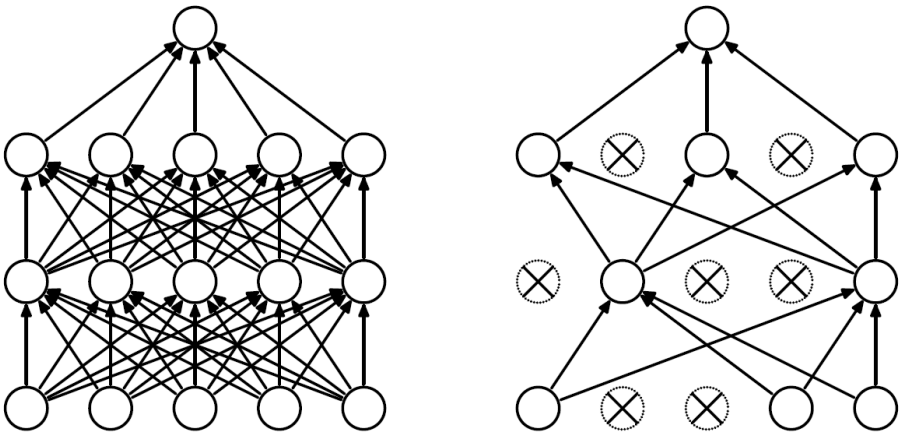
  For this study the **SWA** technique will be implemented using Paszke et al. (2019). The **SWA** learning rate to use will be set to 0.01 and the remaining parameters will be set to their default.

- *Dropout:*

  Dropout (Srivastava et al. (2014)) is a regularization technique that randomly drops a fraction of the units in a neural layer during training (Figure 9). This prevents the network from relying too heavily on any particular unit, forcing it to learn more robust features. Intuitively, this is similar to training multiple models in parallel and averaging their predictions. When applied to a neural network, dropout can be viewed as training a subnetwork on a random subset of the original network's units. This helps to improve generalization to unseen data and reduce overfitting, thereby enhancing performance.

- *Learning Rate Scheduling:*

[1] These are common problems in deep learning, where the gradient becomes too large or too small, respectively, making it difficult for the model to converge to a local minimum. Pascanu et al. (2013).



(a) Standard Neural Net (b) After applying dropout.

Figure 9: Dropout neural net model. Left: A standard neural network with 2 hidden layers. Right: A thinned network created by applying dropout to the left network. Crossed-out units have been dropped. Adapted from Srivastava et al. (2014)

Is a technique that adjusts the **Learning Rate** (**LR**) during training, helpings to the model to converge to a local minimum more efficiently.

This thesis employs the "Reduce Learning Rate on Plateau" scheduler, which dynamically adjusts the **LR** during training based on the validation metric. If the validation metric ceases to improve, the **LR** is reduced by multiplying it with a factor every few epochs. Specifically, this study uses a reduction factor of 0.1 and waits for 25 epochs of no improvement before lowering the **LR**. Additionally, a minimum allowable **LR** of 1×10^{-6} is enforced to prevent it from decreasing too drastically.

2.1.3.2 Callbacks

Callbacks are functions that can be applied at various stages of the training process. In this thesis, the following callbacks were be used:

- *Early Stopping:*

  Early stopping halts training when a monitored metric, typically validation loss, stops improving. This prevents overfitting and enhances model generalization. In this study, training will stop if validation loss doesn't decrease for 50 epochs, and the model will revert to the best-performing weights on the inference phase.

- *Model Checkpoint:*

Model checkpoint is a technique that saves the model's weights at a specified interval. This helps to prevent data loss in the event of a crash, but also helps to save the weight configuration that achieved the best performance until that moment. For our study this callback will continuously save the best performant weights on the training loss, validation loss and Optuna's (Akiba et al. (2019)) objective function (more on this in subsubsection 2.1.4.1).

2.1.4 Hyperparameter Optimization Pipeline

Hyperparameter Optimization (**HPO**) will be used to maximize the performance of a model. This task is particularly challenging due to the exponential increase in potential configurations correlating with the number of parameters involved.

For this process, the open-source library Optuna (Akiba et al. (2019)) has been selected. Optuna is a flexible hyperparameter optimization framework that uses Bayesian optimization techniques, capable of optimizing architectures such as **CAE**s.

Optuna operates by iteratively trying out different **HP** configurations, evaluating their performance, and then selecting the most promising configurations to explore further. This process continues until the optimal configuration is found, or a specified number of iterations is reached. By the end, the best approximation of the optimal set of **HP**, given the number of attempt and random seeds, is returned.

2.1.4.1 Objective Function Design

The primary goal in parameterizing a **CAE** is to achieve a significant discrepancy (ideally full separation) between the reconstruction errors distributions of OK and NOK images. In other words, by maximizing distance between the distributions, effective **AD** can be ensured. An illustrative histogram of OK and NOK ideal error distributions is depicted in Figure 10.

Given the constraints outlined in section 1.2, access to a NOK dataset for optimization may not always be feasible. To circumvent this limitation, data augmentation techniques (section A.1) are employed to synthetically generate NOK images from existing OK images. This approach facilitates the utilization of a single OK image dataset for both training and **HPO**.

An open question is how to quantify the disparity between the distributions. Potential metrics include the Wasserstein distance (Ramdas et al. (2017)) and the **Kullback-Leibler** divergence (Boyd and Vandenberghe (2004)). The former quantifies the distance between two

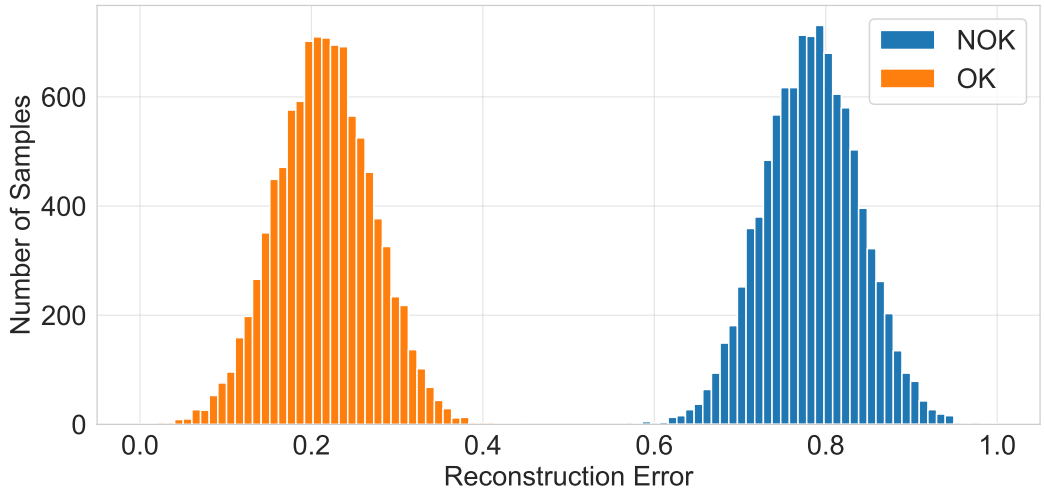


Figure 10: Illustration of ideal OK and NOK distributions with an arbitrary reconstruction error.

probability distributions, whereas the latter measures the divergence of a probability distribution from a reference distribution. Other considerations might include metrics such as the F1 Score or assessing the difference between the maximum loss of the OK dataset and the minimum loss of the NOK dataset.

To validate the feasibility of these metrics, experimental error distributions were generated (Figure 11). Assuming a normalized reconstruction error range of $[0, 1]$, the OK distribution is defined by $\mathcal{N}(0, 0.1)$ and the NOK is characterized by $\mathcal{N}(x, 0.1)$, with x incrementally varied from 0 to 1. The results of the aforementioned metrics are shown in Figure 12.

- *Wasserstein Distance (1D):*

  The Wasserstein distance, also known as the earth mover's distance, measures the discrepancy between two probability distributions. Conceptually, it represents the least amount of "work" required to transform distribution u into distribution v. Here, "work" is defined as the product of the amount of distribution weight moved and the distance it is moved. Mathematically, it is expressed as:

$$WD(u, v) = \inf_{\pi \in \Gamma(u,v)} \int_{\mathbb{R}^2} \|x - y\| d\pi(x, y) \tag{2.5}$$

  In this equation, $\Gamma(u, v)$ denotes the set of all (probability) distributions on $\mathbb{R}^2$ with u and v as their respective marginals.

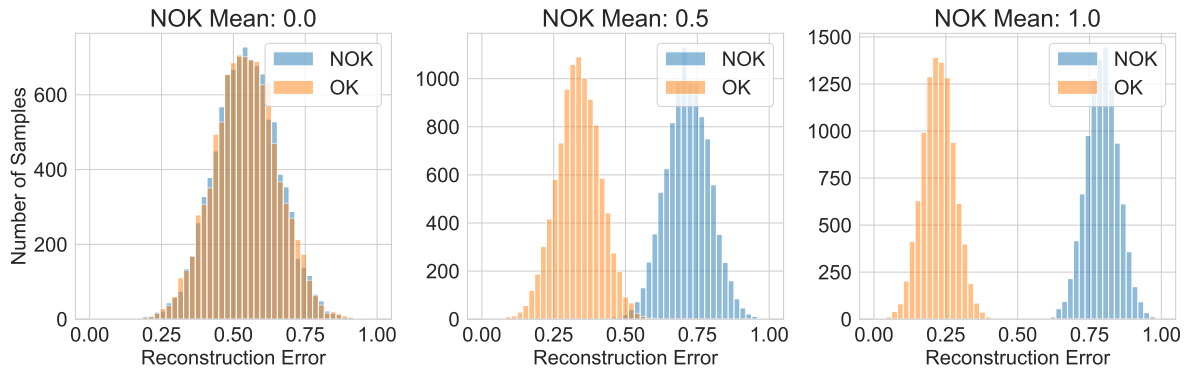


Figure 11: Histogram illustration of OK and NOK normalized distributions. Please note the OK distributions are fixed at $\mathcal{N}(0, 0.1)$ and the NOK distributions vary by $\mathcal{N}(x, 0.1)$, with x incrementally varying from 0 to 1. The final resulting distributions are then normalized to the range $[0, 1]$. $10k$ samples were generated for each distribution.

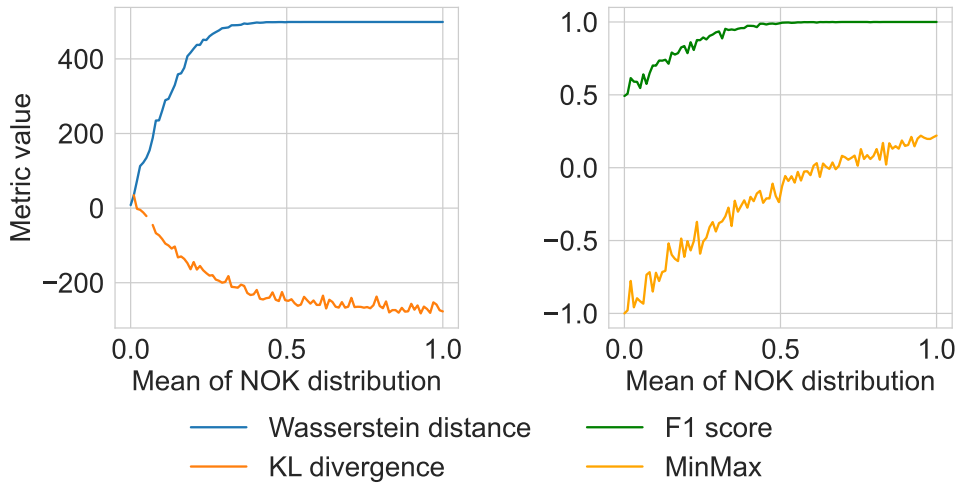


Figure 12: Computed metrics given by the fixing the OK distribution at $\mathcal{N}(0, 0.1)$ and the NOK distribution at $\mathcal{N}(x, 0.1)$, with x incrementally varying from 0 to 1.

It is important to note that this metric requires both distributions to have an equal number of samples. In our specific case, this condition can be met by generating an equal number of NOKs as there are OKs.

As demonstrated in Figure 12, the Wasserstein distance begins at 0 when the distributions are identical and increases as they diverge. However, once the distributions are completely separated, the metric plateaus. In an ideal situation, our chosen metric would continue to increase, increasingly solidifying its **AD** capabilities.

- *Kullback-Leibler Divergence:*

The Kullback-Leibler (**KL**) Divergence is used to quantify the difference between two probability distributions. Specifically, it measures how one probability distribution diverges from a second. Its defined as follows:

$$kl_div(x,y) = \begin{cases} x \log\left(\frac{x}{y}\right) - x + y & \text{if } x > 0, y > 0 \\ y & \text{if } x = 0, y \geq 0 \\ \infty & \text{otherwise} \end{cases} \tag{2.6}$$

Due to its logarithmic nature, it lacks of full continuity across its domain. This can be problematic for Optuna when optimizing the metric. Additionally, while it shares some similarities with the (negative of) Wasserstein distance, such as decreasing in value as distributions diverge and plateauing upon complete separation, its behavior is notably less smooth. This reduced smoothness in this metric could potentially complicate the optimization process, especially when compared to the more gradual changes observed in the Wasserstein distance.

- *F1 Score:*

  The F1 score, a widely utilized metric in classification tasks, serves as the harmonic mean of precision and recall. Its mathematical formulation is given by:

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{2.7}$$

  By balancing precision and recall, at first glance it appears to be a good compromise for our task. By analyzing its behavior, its a function that increases as the distributions diverge and it plateaus when they are completely separated. Furthermore it does not appears to be very smooth even though the distributions have good statistical significance. This could be problematic for Optuna when optimizing the metric.

- *Maximum OK Loss - Minimum NOK Loss:*

  This metric, calculated as the difference between the maximum loss in the OK dataset and the minimum loss in the NOK dataset, offers a straightforward measure. It increases as the distribution gap widens and continues to rise when the distributions are

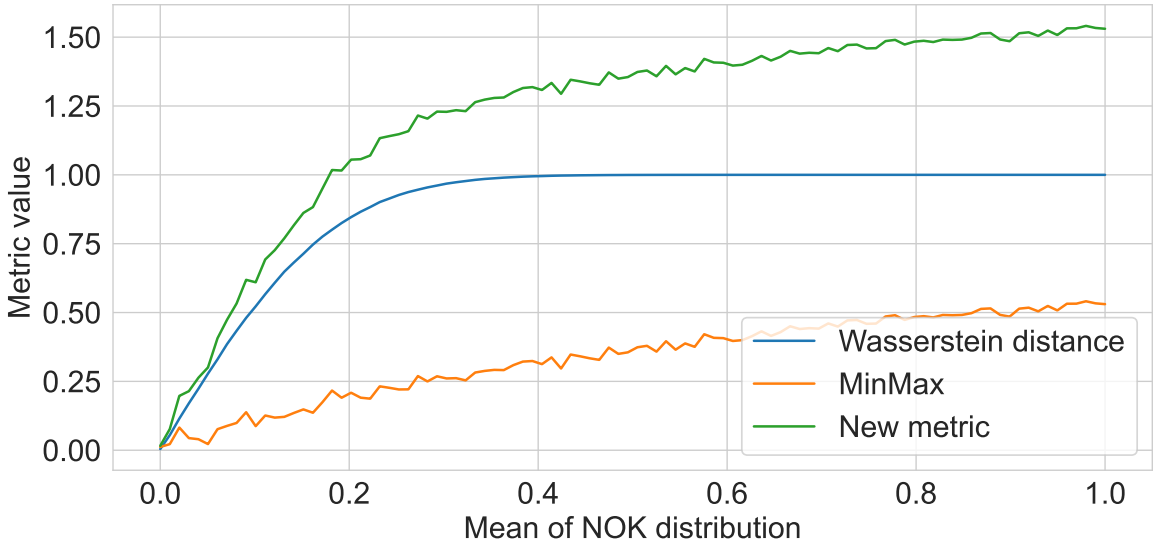


Figure 13: Computed objective function and its components.

entirely separate. However, its non-smooth nature may pose challenges during optimization with Optuna.

The best two candidates appear to be the Wasserstein distance and the Maximum OK Loss - Minimum NOK Loss. The first by its smoothness and desirable behavior when the distributions are diverging and the second by its behavior when the distributions are completely separated (since it continues to increase).

As such, a objective function was defined by the sum of the Wasserstein distance and the Maximum OK Loss - Minimum NOK Loss. In order to normalize the objective function, the Wasserstein distance was divided by the $N_{samples}/2$ and the Maximum OK Loss - Minimum NOK Loss was transformed to $((min(NOK) - max(OK)) + 1)/2$. This way, both metrics present an output range of values of $[0, 1]$. The resulting metric is given by:

$$\text{Metric} = \frac{WD(OK, NOK)}{N_{samples}/2} + \frac{(min(NOK) - max(OK)) + 1}{2} \tag{2.8}$$

where $N_{samples}$ is the number of samples in the dataset. This is illustrated in Figure 13 and the objective function has an output range of $[0, 2]$. The objective function is 0 when the distributions are identical and it increases as the distributions diverge.

2.1.5 Model Creation Using Optuna

Creating an pipeline to generate a **CAE** based on specified **HP** requires understanding the interdependencies between its layers (subsection 2.1.2). Unlike traditional networks, **CAE** layers have specific constraints; for example, each convolutional layer in the encoder must correspond to an inverse layer in the decoder. This correspondence isn't always straightforward for any set of **HP** due to the unique nature of convolutional layers.
To address this, a pipeline was developed to automate **CAE** generation, training, and performance evaluation given **HP**s. The following sections describe the pipeline's structure and functionality, detailing how it ensures compatibility between layers.

2.1.5.1 Encoder

Upon completing the precomputations (section A.2), the pipeline is ready for the encoder stage. The encoder comprises convolutional layers, batch normalization, activation functions, pooling, and dropout mechanisms. The architecture is build by a for loop iterating over the number of convolutional layers (n_conv) specified in the **HP** as follows:

1. **Dropout Layer**: If dropout is enabled (USE_DROPOUT) and the current layer is not the initial layer, a dropout layer is appended to the encoder layers. The dropout probability is denoted by DROPOUT_PROB in the code.

```
if USE_DROPOUT and n_conv != 0:
    self.encoder_layers.append(nn.Dropout(p=DROPOUT_PROB))
```

2. **Convolutional Layer**: Is appended to the encoder layers using the specified **HP**. Padding is set to 0.

```
enc_layer = nn.Conv2d(
    in_channels=PREVIOUS_N_CHANNELS,
    out_channels=N_CHANNELS,
    kernel_size=KERNEL_SIZE,
    stride=STRIDE,
    padding=0,
```

```
)
self.encoder_layers.append(enc_layer)
```

3. **Batch Normalization Layer**: If batch normalization is enabled and the output layer size L_{Out} is greater than 1, a batch normalization layer for N_CHANNELS is appended.

```
if USE_BATCH_NORM and LOut > 1:
    self.encoder_layers.append(nn.BatchNorm2d(N_CHANNELS))
```

4. **Activation Function**: The activation function specified in the **HP** is appended.

5. **Max Pooling Layer**: If max pooling is enabled, a 2D max pooling layer with a kernel size and stride of 2 is included. This layer also returns indices, enabling unpooling in later stages.

```
if USE_MAX_POOL:
    self.encoder_layers.append(
        nn.MaxPool2d(kernel_size=2, stride=2, return_indices=True)
        )
```

2.1.5.2 Decoder

Subsequent to the encoder stage, the pipeline transitions to the decoder segment. The decoder is meticulously designed to reverse the operations of the encoder, thereby reconstructing the input data from its encoded form. The decoder is composed of several layers, each reversing the function of a corresponding layer in the encoder:

1. **Dropout Layer**: If dropout is utilized and the layer is not the initial layer in the decoder, a dropout layer with probability DROPOUT_PROB is added.

```
if USE_DROPOUT and n_conv != 0:
    self.decoder_layers.append(nn.Dropout(p=DROPOUT_PROB))
```

2. **Activation Function**: For all layers except the final one, an activation function defined by the **HP** (ACTIVATION_FUNCTION) is appended.

```
if n_conv != 0:
    self.decoder_layers.append(getattr(nn, ACTIVATION_FUNCTION)())
```

3. **Batch Normalization Layer**: If batch normalization is enabled and the layer is not the first in the decoder, and if the output tensor size LOutT is greater than 1, a batch normalization layer for PREVIOUS_N_CHANNELS is appended.

```
if USE_BATCH_NORM and n_conv != 0 and LOutT > 1:
    self.decoder_layers.append(nn.BatchNorm2d(PREVIOUS_N_CHANNELS))
```

4. **Transposed Convolutional Layer**: A transposed 2D convolutional layer is added to the decoder layers.

```
dec_layer = nn.ConvTranspose2d(
    in_channels=N_CHANNELS,
    out_channels=PREVIOUS_N_CHANNELS,
    kernel_size=KERNEL_SIZE,
    stride=STRIDE,
    padding=P,
    output_padding=P_0,
)
self.decoder_layers.append(dec_layer)
```

5. **Max Unpooling Layer**: If max pooling was used in the encoder, the corresponding unpooling layer with a kernel size and stride of 2 is included in the decoder.

```
if USE_MAX_POOL:
    unpool = nn.MaxUnpool2d(kernel_size=2, stride=2)
    self.decoder_layers.append(unpool)
```

Additionally please note the last layers of the encoder won't have batch normalization or pooling.

2.1.5.3 Variable Update Process

After the establishment of the encoder and decoder layers, the pipeline proceeds to update several key variables. The update mechanism is detailed as follows:

1. **Latent Dimension Check**: The pipeline initially checks if the current dimentionality (given by the convolutional layers dictated by Optuna) is below the desired latent space. If so, the latest added convolutional layers will be ignored and the pipeline will proceed to the subsubsection 2.1.5.4 step.

2. **Variables Update**: The variable IMAGE_SIZE is updated based on whether max pooling is utilized. If max pooling is enabled, IMAGE_SIZE is set to LOut_Pool; otherwise, it is updated to LOut. Furthermore, the variable PREVIOUS_N_CHANNELS is updated to N_CHANNELS.

3.
```
if USE_MAX_POOL:
    IMAGE_SIZE = LOut_Pool
else:
    IMAGE_SIZE = LOut

PREVIOUS_N_CHANNELS = N_CHANNELS
```

4. **Minimum Image Size Check**: The process checks whether the image size has been reduced to the minimum threshold MIN_IMG_SIZE. If the reduced image size meets this criterion, either with or without max pooling, the process is halted to prevent the dimensions from becoming too small.

```
MIN_IMG_SIZE = 4
if USE_MAX_POOL and (LOut // 2) <= MIN_IMG_SIZE:
    break
elif not USE_MAX_POOL and LOut <= MIN_IMG_SIZE:
    break
```

```
        else:
            pass
```

2.1.5.4 Linear Layers

This section details the implementation of linear layers positioned at the end of the encoder and the beginning of the decoder.
Initially, flattening and unflattening layers are respectively integrated into the encoder and decoder. Subsequently, linear layers are appended. In the encoder, the first linear layer is sized to the nearest power of 8 smaller than the flattened size, with each subsequent layer sized to the closest power of 8 smaller than the previous layer, ensuring each layer remains larger than the latent dimension. Conversely, in the decoder, the linear layers mirror those of the encoder in reverse order. Let the reader note the base 8 was chosen arbitrarily, and other bases could be used, but preliminary tests showed its effectiveness.
Finally, the encoder and decoder models are completed by the layers computed in the previous steps:

```
self.encoder_model = nn.Sequential(*self.encoder_layers)
self.decoder_model = nn.Sequential(*self.decoder_layers[::-1])
```

2.1.6 Performance Enhancing Post-Processing Techniques

After training the model, various post-processing techniques can be applied to enhance its performance, minimize false positives, and enhance interpretability. The techniques discussed in the following section are adaptable to any reconstruction-based approaches, making them a valuable toolbox for various applications.
In addition to the standard reconstruction loss metric, latent space analysis, clustering techniques, and **SVM**s[2] and **PCA** will be performed. Furthermore, parameter tuning using the validation dataset will be conducted, followed by testing on the test dataset to verify any improvements.

[2] All SVMs developed in this thesis utilize the radial basis function kernel. The training process for these SVMs employs a balanced sample weights regime to ensure equitable consideration of all classes in the dataset.

2.1.6.1 Baseline Approach

The baseline approach adopts a pragmatic calculation reflective of a production environment. Once a model is trained, a specific threshold for its classification metric is set to classify an image as OK or NOK. In the absence of a dedicated NOK dataset, the validation dataset is employed to establish this threshold, which is then applied to the test dataset.
Setting the ideal threshold is a nuanced task, balancing between false positives and false negatives. To do so, metrics such as the F1-score, recall, and precision are calculated across the entire range of the classification metric. This strategy provides quality operators flexibility in selecting the most suitable threshold for future specific applications, acknowledging that a complete separation of distributions is not always feasible.
Prioritizing recall focuses on detecting NOK instances but may increase false positives, while emphasizing precision can lead to false negatives. A balanced approach could be the F1 score, which optimizes both precision and recall for better class discrimination. Alternatively, the intersection of recall and precision can define a threshold for convergence, which is then validated on the test dataset.

2.1.6.2 Latent Dimension Analysis

The latent dimension of the **AE**-based model provides yet another viable avenue for classification. One approach is to compute metrics like the sum, square, and mean of the latent dimensions for each image and visualize them on a pair plot to distinguish between OK and NOK images. To preserve essential information during dimensionality reduction, **PCA** can be applied, addressing the potential information loss from reductive methods. Even if complete class separation isn't achievable, **SVM** can be utilized to classify images.

2.1.6.3 Error Clustering

Clustering techniques applied to reconstruction errors can differentiate between general noise and actual anomalies, significantly reducing noise to highlight only the defects. This method aims to create a clearer image for easier error identification and improved classification accuracy by separating noise from genuine errors. Ultimately, this enhances interpretability, aiding operators in error detection, with OK images appearing blank and NOK images clearly displaying error clusters corresponding to true anomalies.
Among numerous algorithms designed to clustering, connected-component labeling is par-

ticularly effective. It scans images to identify connected regions based on predefined connectivity criteria. This method differentiates between non-zero elements, identified as features, and zero elements, considered as background, using a structuring element. For this purpose, a centrosymmetric pattern (Weaver (1985)) is used. Specifically, the *scipy.ndimage.label* function (Virtanen and at (2020)) is employed to apply this algorithm. The function scans the image and assigns a unique label to each connected region (or cluster), thereby clustering the errors.

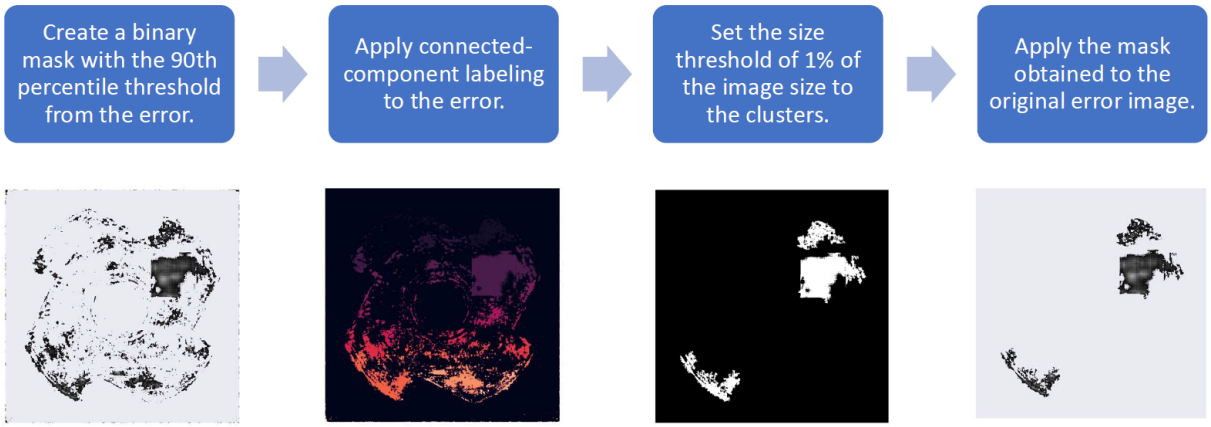


Figure 14: Clustering algorithm for identifying error regions, demonstrating the process from raw error visualization to distinct error clustering.

The complete clustering algorithm begins by calculating the 90th percentile of the error in OK images, establishing a threshold to filter out noise from the input. Next, the connected-component labeling algorithm is applied to the resulting image to cluster the errors. Clusters are then filtered by size, retaining only those with an area greater than 1% of the image size. The resulting image is then used as a binary mask of the original error image. The full process is illustrated in Figure 14.

Additionally, this clustering technique can be used in conjunction with image filtering techniques (section A.3) to further enhance the final output. This strategy may be further optimized by tuning the **HP** of the connected-component labeling algorithm and adjusting the size threshold, though these parameters were not optimized within the scope of this research.

2.2 Variational Autoencoders

This section applies the previously explained techniques to a **Variational Autoencoder** (**VAE**). A **VAE** is a generative model that enhances traditional **Autoencoder** through variational in-

ference. Unlike traditional **AE**s, which learn a deterministic mapping from input data to latent space and back, **VAE**s impose a probabilistic structure on the latent space. Introduced by Kingma and Welling (2013), **VAE**s differ from the previously discussed **AE** in the following ways:

1. They are probabilistic, meaning that their outputs are partly determined by chance, even after training.[3]
2. They are generative models, meaning they can generate new instances that look like they were sampled from the training set.

Let's take a look at how they work. In Figure 15 its possible to recognize the basic structure of all **AE**, with an encoder followed by a decoder, but there is a twist: instead of directly producing a coding for a given input, the encoder produces a mean coding, μ, and a standard deviation coding, σ. The actual coding is then sampled randomly from a Gaussian distribution with mean μ and standard deviation σ, creating the actual latent space vector Z. After that, the decoder decodes the sampled coding normally.
The **VAE** is trained by optimizing two objectives simultaneously: the reconstruction loss and the **Kullback-Leibler** (**KL**) divergence. These objectives ensure that the **VAE** learns to encode input data into a well-structured latent space.

- **Reconstruction Loss**: Measures how well the **VAE** can reconstruct the input data, measured by assessing the discrepancy between the original input and the reconstructed data (e.g. using the **MSE** loss).
- **KL Divergence**: Ensures the latent space distribution approximates the desired prior distribution. The **KL** divergence measures the difference between the learned latent distribution and the prior distribution, encouraging the latent space to follow the desired structure. The prior distribution used though this work is an standard normal distribution. As such, final values close to 0 for the mean (μ) and 1 for the standard deviation (σ) indicate that the model training proceeded smoothly.

In the context of **AD**, **VAE**s can provide several advantages due to their inherent ability to model data distributions. This capability should make the model more generalizable, improving its performance. Additionally, further insights could be extracted by analyzing their latent space components.

[3] For the inference step, the output is obtained by sampling 10 images and averaging them.

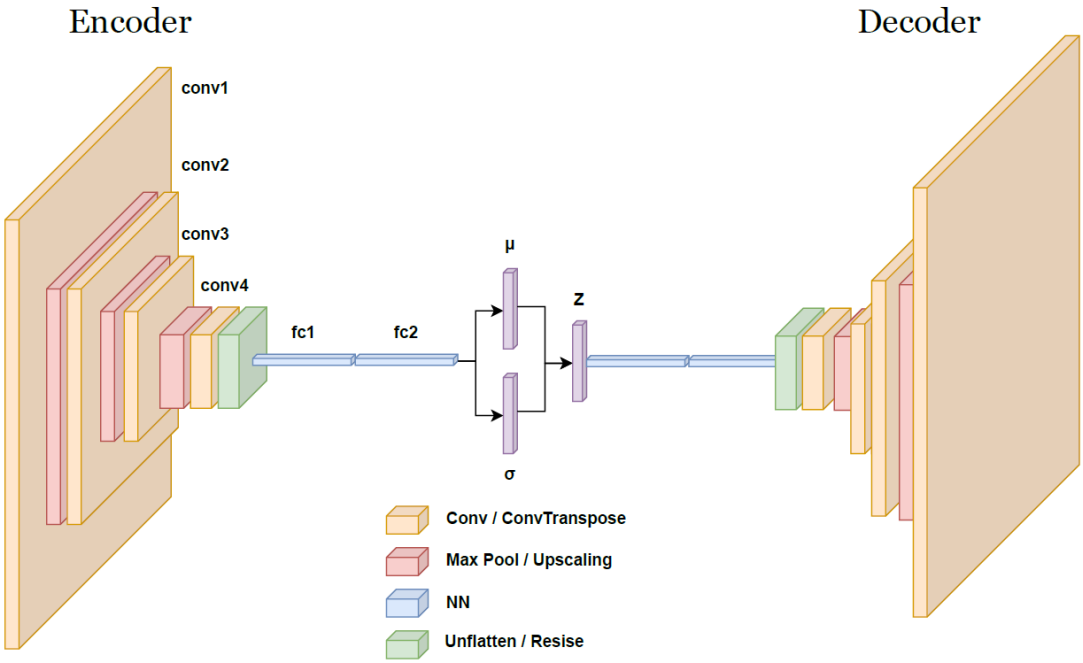


Figure 15: Example of a **VAE** architecture, showing the encoder, latent space, and decoder components. Here conv1 to conv4 represent convolutional layers, and fc1 and fc2 represent fully connected layers. μ and σ are the mean and standard deviation of the latent space respectively and z is the latent space vector.

2.2.1 Key Concepts

2.2.1.1 Reparameterization Trick

Training **VAE**s involves backpropagation through a sampling process, which is inherently non-differentiable. To overcome this, the reparameterization trick is employed, which allows the gradients to propagate through the sampling operation. The reparameterization trick expresses the sampled latent variable z as:

$$z = \mu + \sigma \odot \epsilon \tag{2.9}$$

where $\epsilon \sim \mathcal{N}(0, 1)$. Since μ and γ are learnable parameters, and ϵ is a fixed constant during backpropagation, this formulation allows the gradient to flow through the network. This process is illustrated in Figure 16, where its possible to see the gradient path through the deterministic path of the network, leaving the stochastic part untouched.

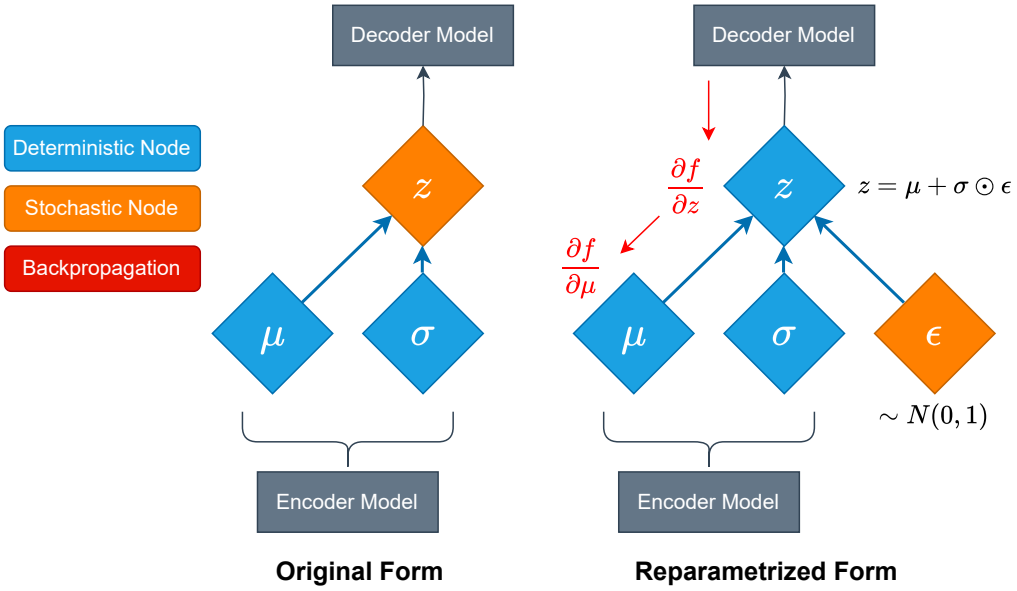


Figure 16: Reparameterization trick visualized.

2.2.1.2 KL Divergence in VAEs

The KL divergence measures the difference between the learned latent distribution and the desired prior distribution. By minimizing this divergence, the VAE aligns the latent space with a well-defined structure. The KL divergence term in the VAE loss function is computed as follows:

$$KL(q(z|x)||p(z)) = \int q(z|x) \log \frac{q(z|x)}{p(z)} dz \tag{2.10}$$

2.2.1.3 Loss Details

The cost function combines two primary components: the reconstruction loss and the latent loss as described above. This section elaborates on the mathematical foundations underlying these loss components.

The reconstruction loss measures how well the decoded output matches the input. Typically, this is quantified using metrics such as MSE or BCE, depending on the nature of the input data. On the other hand, the latent loss is defined using the KL divergence, which measures the difference between the learned latent distribution and the desired prior distribution, and for a Gaussian latent space (Kingma and Welling (2013)) is given by:

$$KL_{loss} = -\frac{1}{2}\sum_{i=1}^{N}\left(1 + \log(\sigma_i^2) - \sigma_i^2 - \mu_i^2\right) \tag{2.11}$$

where μ_i and σ_i are the mean and standard deviation of the i^{th} component of the latent variables output by the encoder.

To improve numerical stability, instead of directly parameterize the standard deviation σ, it is common to parameterize its logarithm (Géron (2022)). Let $\gamma = \log(\sigma^2)$, allowing us to rewrite the **KL** divergence as:

$$KL_{loss} = -\frac{1}{2}\sum_{i=1}^{N}\left(\gamma_i - \exp(\gamma_i) - \mu_i^2\right) \tag{2.12}$$

Thus making the two learnable parameters by the encoder γ and μ.

The total loss function for training the **VAE** is then a weighted sum of the reconstruction loss and the **KL** divergence:

$$L = \alpha \cdot Recon_{loss} + KL_{loss} \tag{2.13}$$

Here, α is a new hyperparameter that balances the two components of the loss function, controlling the trade-off between reconstruction accuracy and adherence to the latent space distribution.

Chapter 3

Results

3.1 Traditional Approach

This section will explore a typical approach. By using a **CAE** architecture, this section objective is to model the architecture such that the validation loss is minimized. For this, a **HP** tuning process (subsection 2.1.5) will be used.
The training and validation dataset consists exclusively of OK images. The validation results are solely used for **HP** optimization and not for fine-tuning the model's performance.

Hyperparameter Name	Range	Log
Learning Rate (**LR**)	[1e-5, 1e-1]	Yes
Latent Dimension	$[1, \frac{\text{IMAGE_DIM}}{10}]$	Yes
Activation Function	ReLU, LeakyReLU, Sigmoid, Tanh, ELU	NA
Use Batch Normalization	True, False	NA
Use Dropout	True, False	NA
Dropout Probability	[0.1, 0.9]	No
Maximum Convolutional Layers	[1, 10]	No
Stride X	$[1, \min(5, \frac{\text{IMAGE_SIZE}}{2})]$	No
Kernel Size X	$[1, \min(10, \text{IMAGE_SIZE})]$	No
# Channels X	[1, 256 (first layer) or as per previous layer #Channels]	No
Use Max Pooling X	True, False	NA

Table 1: **HP** selection range. Note that 'log' refers to whether the **HP** is optimized in the logarithmic space. 'X' refers to the layer number.

The **HP** were selected based on their potential impact on the model's performance and the

need for a comprehensive exploration of the search space as explained on subsection 2.1.5. The selected ranges can be seen in Table 1.

The model was trained for a total of 500 epochs using the Adam optimizer (Kingma and Ba (2014)) with a batch size of 16. The learning rate and the architecture were determined by the Optuna framework (Akiba et al. (2019)), and the **MSE** was used as the loss function. Model checkpointing, early stopping, and reducing the learning rate on a plateau were implemented as callbacks and were monitoring the model's validation loss. Validation step was being computed at each epoch. The training was performed using the PyTorch framework (Paszke et al. (2019)) on a single NVIDIA V100 16GB **GPU** with 16-bit precision. For more detailed information about the training process and the specifications of the environment, please refer to subsection 2.1.3.

3.1.1 Test Dataset and Performance Analysis

After training the model, a qualitative visual analysis can be performed by observing Figure 17. By observing the reconstruction error, it is evident that the model struggles to accurately reconstruct even in OK images, presenting a significant amount of background noise, indicated by the similar error values in the regions of actual anomalies in the NOK images. Furthermore the anomalies are not clearly distinguishable, especially in regions where they were expected to be present. This suggests that this approach not be feasible to accurately localize the anomalies in the images and to differentiate between OK and NOK images.

Metric	AUC	Accuracy	Recall	Precision	F1
MSE	0.4619	0.3565	0.2473	0.8519	0.3833
MAE	0.4399	0.3739	0.2688	0.8621	0.4098
BCE	0.7091	0.5848	0.5929	0.8048	0.6808
SSIM	0.1544	0.6000	0.7419	0.7582	0.6910
PSNR	0.4619	0.3565	0.2796	0.7879	0.4127
NRMSE	0.4399	0.3739	0.2688	0.8621	0.4098

Table 2: Comparison of metrics on classification power.

The results obtained from the test analysis are presented in Table 2 and Table 3. Among the different loss metrics, the **BCE** loss achieved the highest **AUC** value of approximately ≈ 0.71, while the **MAE** loss achieved the highest **IoU** score of approximately ≈ 0.20.

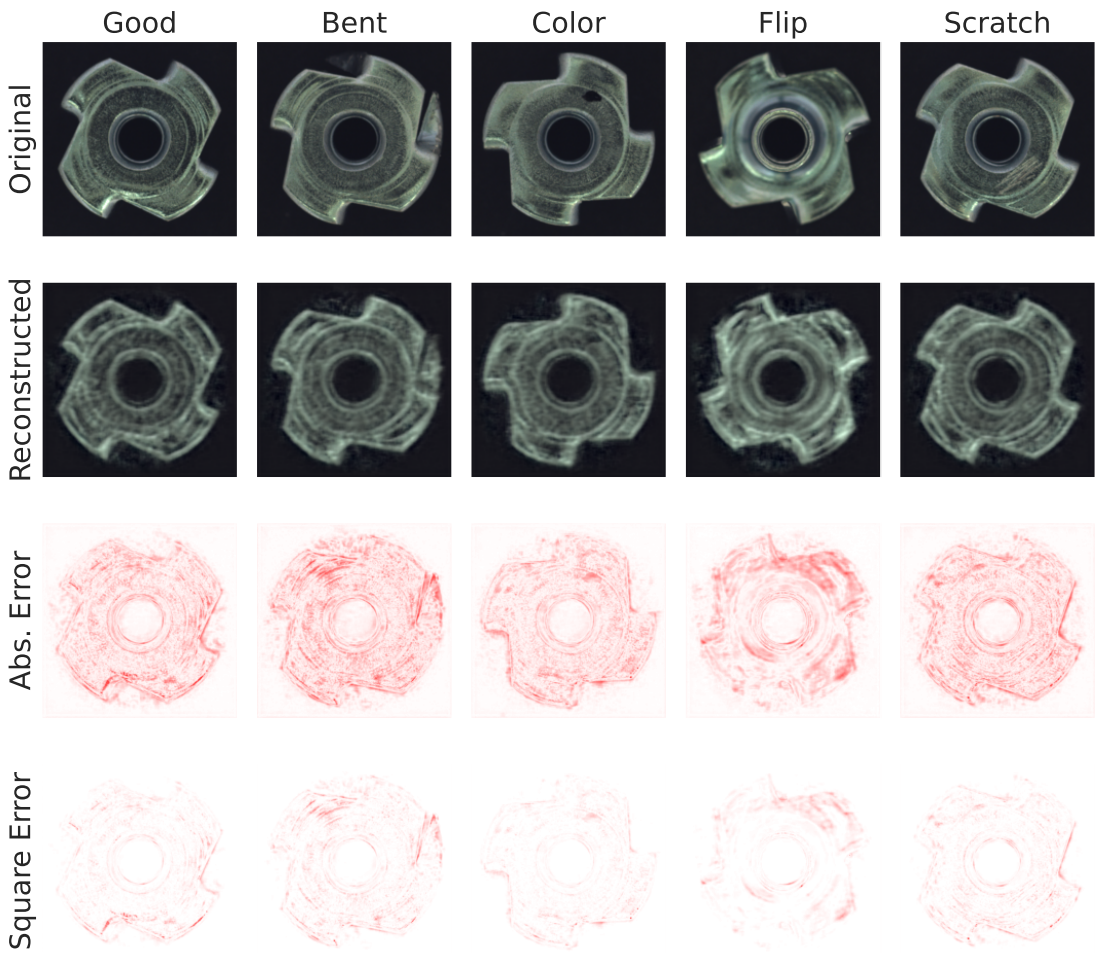


Figure 17: Reconstruction of the test dataset using the traditional approach. On each column it can be seen the different labels of the test dataset. The first row shows the original images, the second row the reconstructed images, the third row the absolute error map and the last row the squared error map.

Metric	IoU
MSE	0.1964
MAE	0.1970
BCE	0.1620
NRMSE	0.1968

Table 3: Table of **IoU** values for different metrics

Additional optimization visualizations and discussion can be found in section A.4.

3.2 Convolutional Autoencoder

Building on the foundational concepts outlined in section 2.1, this section presents the outcomes of a focused **HPO** effort aimed at developing a **CAE** architecture tailored for enhanced **AD** performance in a industrial environment.

A summary of the **HP** used and their ranges is shown in Table 1, the same as the previous approach and discussed in section 3.1. As before, the optimization process will leverage the Optuna framework to systematically explore the **HP** space and refine the objective function with the goal of maximizing it. The objective used in this section is now Equation 2.8 as established in subsubsection 2.1.4.1.

To meet the requirements of the objective metric, the validation dataset has been expanded to include both OK images and a diverse array of synthetically generated NOK images. These NOK images are created using data augmentation techniques on OK images (subsection A.1.1). The benchmarked model model is configured with the optimal **HP**s found and uses the best weights on the OK-only validation loss.

3.2.1 Optimization Process

Following the **HPO**, several figures have been generated to illustrate the optimization insights. Figure 18 shows that the optimization process has been successful, with the objective function value increasing over time. It's found that with a maximum number of trials of 500, Optuna finds the best set of **HP** in the trial 105 with a score of ≈ 1.20 for the objective function. Note that when a trial failed, it returned a value of 0 for the objective function. This happened because Optuna chose a set of **HP** whose which yielded a model architecture with more parameters than our **GPU** memory was capable of.

The best trial parameters obtained from the optimization process are detailed in Table 4. These parameters were used to configure the **CAE** architecture for the final model training and have shown to provide the best performance in terms of the objective function considered during the optimization phase.

As Figure 19 indicates, the latent dimension emerges as the paramount **HP**, which was expected since it represents the bottleneck of the network for data compression and reconstruction. It is followed by other expected parameters such as the maximum number of convolutional layers, which limits the model's capability to capture complex features and subse-

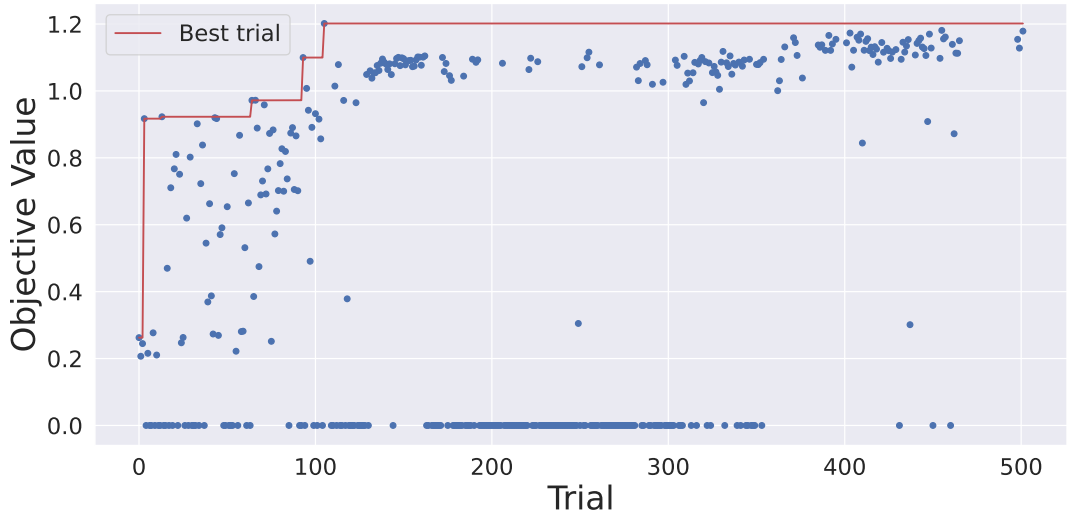


Figure 18: Optimization history of the **HP** for the **AE** architecture. Note that, if a trial failed is plotted as 0.

quently parameters pertaining to the model's initial convolutional layer, including the number of channels and the application of max pooling. Notably, the learning rate does not rank among the top 10 **HP**s.

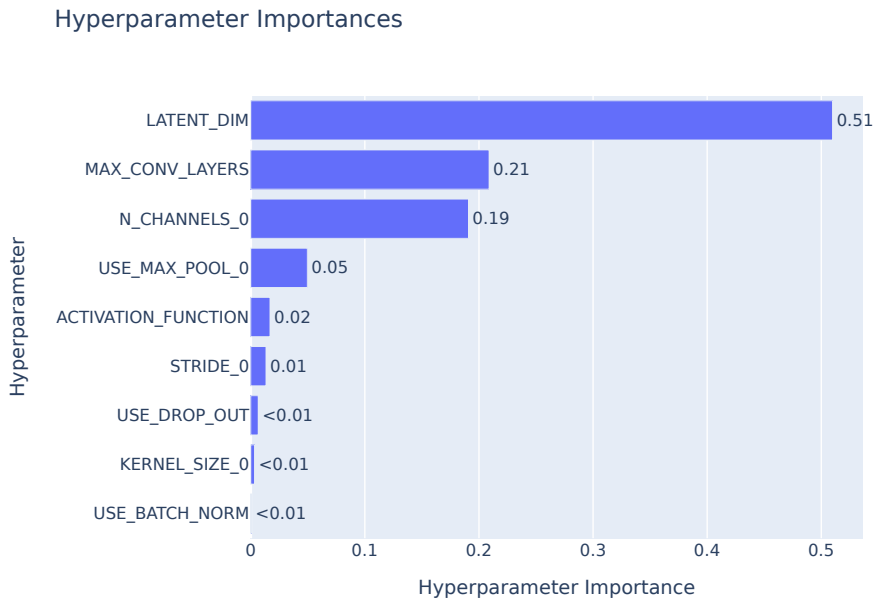


Figure 19: Parameter importance as determined by the Optuna framework for the **CAE** architecture.

On Figure 20 the **EDF** shows the cumulative probability that Optuna has successfully identi-

Parameter	Value
Latent Dimension	25260
Activation Function	LeakyReLU
Use Batch Normalization	True
Use Dropout	False
Maximum Convolutional Layers	3
Stride #1	2
Kernel Size #1	10
Number of Channels #1	77
Use Max Pooling #1	True
Stride #2	2
Kernel Size #2	9
Number of Channels #2	73
Use Max Pooling #2	False
Stride #3	2
Kernel Size #3	4
Number of Channels #3	53
Use Max Pooling #3	True
Starting **LR**	≈ 0.00038

Table 4: Best set of **HP**, achieved during the best Optuna's trial.

fied the optimal set of **HP**. This is important because it indicates the likelihood of achieving similar results in future experiments. It can be seen that the probability reaches its maximum when the objective function value from the best trial is 1.20, suggesting that the optimal **HP** configuration has been successfully identified.

By analyzing the slice plot of non-layer specific **HP** (Figure 21) it is possible to make several observations. Firstly, the optimization process did not benefit from using dropout, as a larger number of trials, which achieved higher scores, did not employ dropout. This was somewhat unexpected since, theoretically, dropout is supposed to aid in model generalization (Srivastava et al. (2014)). Secondly, the maximum number of convolutional layers exhibited an approximately uniform distribution for configurations exceeding two layers. This outcome likely stems not only from the **HP** settings but also from the memory constraints of the **GPU**.

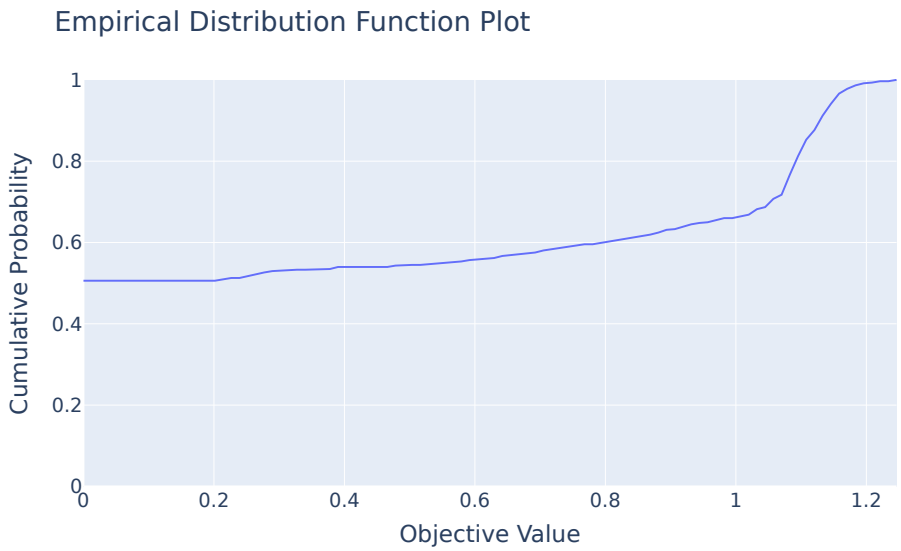


Figure 20: Empirical distribution function for the **HP** of the autoencoder architecture.

Specifically, models with up to two layers had suboptimal complexity, leading to lower performance scores, whereas models with more than two layers maintained a roughly constant complexity level due to them hitting the memory limit. Regarding batch normalization, the optimization process showed a preference for incorporating it, aligning with expectations given its known benefits for faster model convergence. Furthermore, the **LR** was optimized towards the lower end of the spectrum, consistent with the understanding that a higher **LR** could cause model divergence and training instability. Finally, the latent dimension was also optimized towards the lower end, reflecting the anticipation that a larger latent dimension could lead to model overfitting. Even though the latent dimension appears to be quite high ($\approx 10^4$), when compared to the original dimensions of the image ($700 \times 700 \times 3 = \approx 10^6$), it is still a significant reduction in the number of parameters by two orders of magnitude.
The slice plot of layer-specific **HP** is illustrated in Figure 22. It is evident that the kernel sizes in various layers of the **CAE** tend to yield higher performance as they approach their upper limit of 10. This suggests that potential improvements could be achieved by expanding the upper limits in the **HP** optimization pipeline. Regarding the number of channels, although optimization seems typical, reaching upper limits was constrained by **GPU** memory limitations. Despite this, saturation in performance metrics occurred at approximately 100 channels for the first layer and around 50 channels for subsequent layers. The optimization of stride parameters progressed without reaching any predefined **HP** constraints. The benefits of max pooling remain inconclusive, as the optimization process did not exhibit a clear pref-

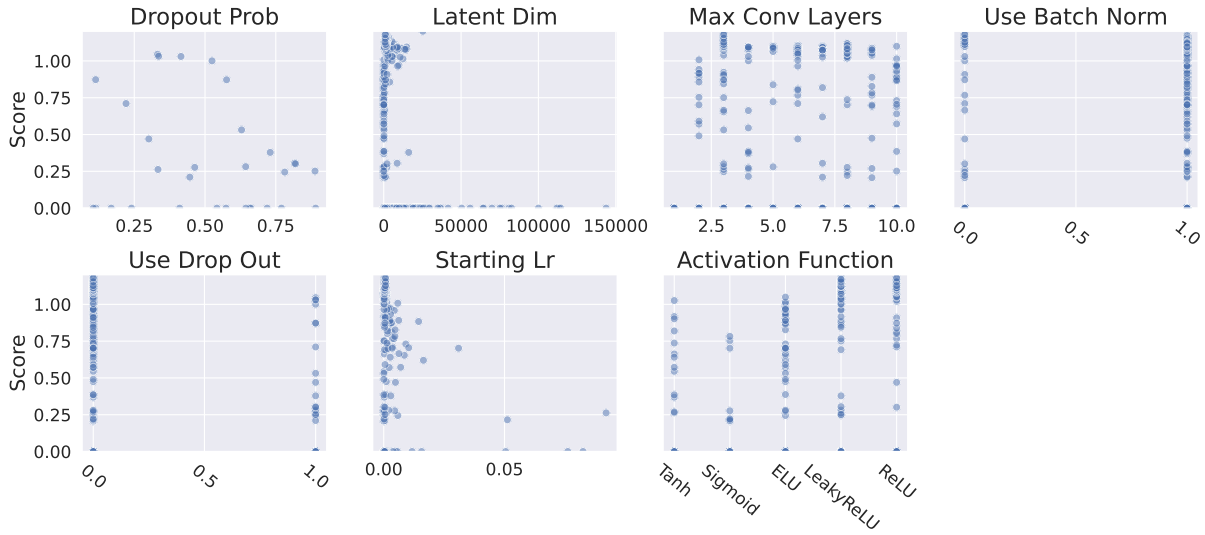


Figure 21: Slice plot of the non-layer specific **HP** for the **CAE** architecture. Note that some **HP** have a boolean value of 1 (True) and 0 (False). The y-axis is the objective function score and the x-axis is the **HP** value.

erence for its use. It should be noted that hardware constraints limited the exploration of the full spectrum of **HP**s, and the results presented here reflect the best possible configurations within the available resources.
A detailed examination of the trained model is discussed in the subsequent sections, focusing on evaluating its performance, with the use of post-processing techniques (subsection 2.1.6).

3.2.2 Test Results

By visually inspecting Figure 23, it is evident that the model performs well in reconstructing OK images, as the reconstructed images closely resemble the original ones. However, the model struggles in reconstructing anomalies, particularly in the 'bent' and 'color' classes. The regions corresponding to these anomalies show significant reconstruction errors. In the case of the 'scratch' class, although the anomaly is noticeable, it is somewhat similar to the background noise, making it less distinct. Additionally, for the 'flip' class, the model tends to produce larger errors near the edges of the metal nut. These observations highlight the various challenges faced by the model in accurately reconstructing anomalous images.
In evaluating the model's performance on the test dataset, Table 5 details the results using various metrics applied to the reconstruction error. Notably, the model achieves an **AUC** of approximately 0.79 with the **BCE** metric, which is competitive when compared to a reported

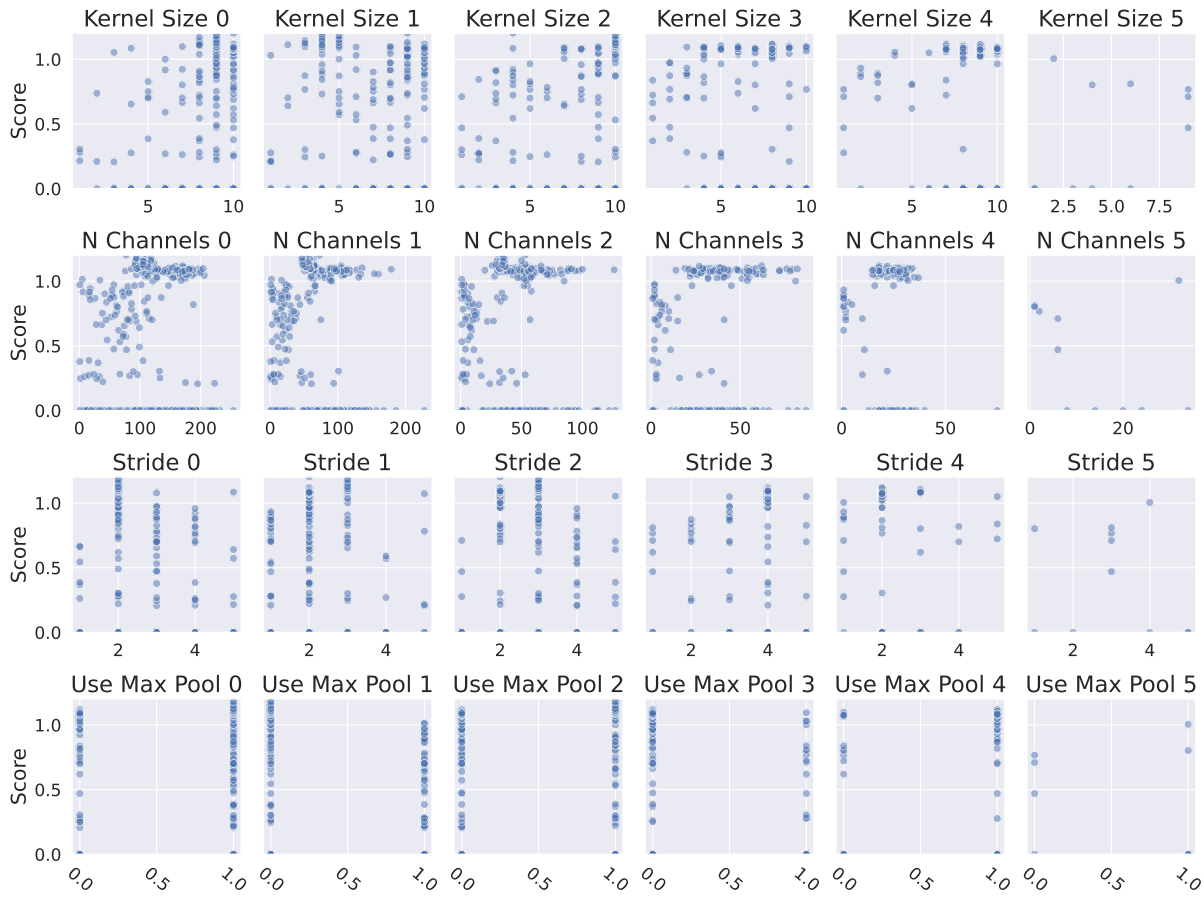


Figure 22: Slice plot of the layer specific **HP** for the **CAE** architecture. The number on the **HP** name refers to the layer number. The y-axis is the objective function score and the x-axis is the **HP** value.

AUC of 0.82 in the literature Bergmann et al. (2019). Furthermore, an **IoU** of approximately 0.25 is obtained using the model's **MSE** loss, which closely approaches the literature's benchmark of 0.26. With this in mind, it's evident this bespoke method of a general pipeline to tackle **AD** in industrial environments is competitive with **SOTA**, rivaling handcrafted architectures to tackle similar challenges.

In subsequent sections, post-processing techniques (subsection 2.1.6) will be explored, some of which leverage parameters previously optimized with the validation dataset. Additionally, the histogram of **MSE** reconstruction loss for the test dataset is presented in subsection A.5.3.

3.2.2.1 Evaluation Metrics

- Baseline Approach

  In Table 7 the performance metrics computed for the different thresholds calculated

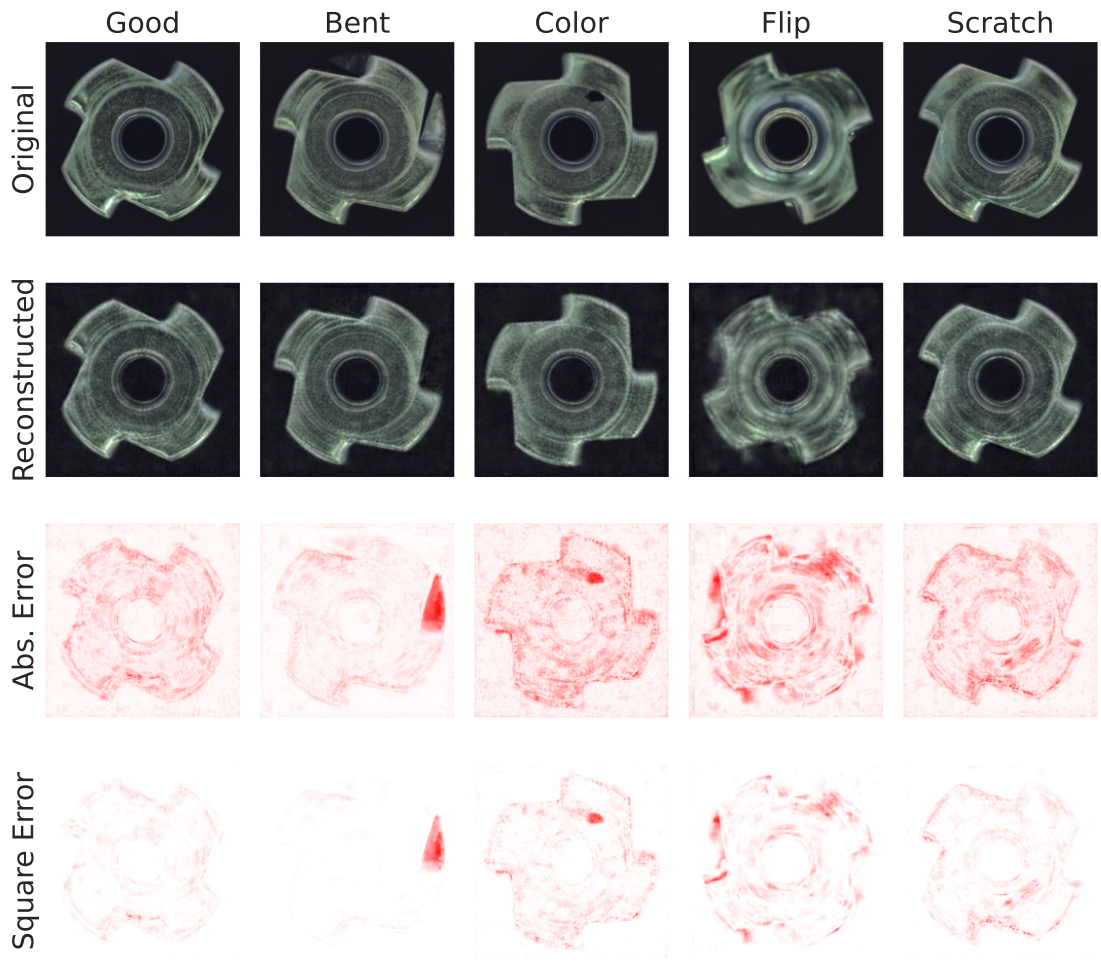


Figure 23: Reconstruction of the test dataset using the **CAE** architecture. The first row shows the original images, the second row the reconstructed images, the third row the absolute error map and the last row the square error.

Metric	AUC	Accuracy	Recall	Precision	F1
MSE	0.6227	0.4000	0.2581	1.0000	0.4103
MAE	0.5860	0.4000	0.2581	1.0000	0.4103
BCE	0.7923	0.6957	0.6882	0.9143	0.7853
SSIM	0.5611	0.3913	0.2473	1.0000	0.3966
PSNR	0.6227	0.4174	0.2903	0.9643	0.4463
NRMSE	0.5860	0.4000	0.2581	1.0000	0.4103

Table 5: Evaluation metrics of the test dataset.

previously in Table 37 are shown. At first glance, the Recall threshold has the best performance in terms of F1 score and Recall, achieving ≈ 0.89 and ≈ 0.98 respectively.

Metric	IoU
MSE	0.2533
MAE	0.2491
BCE	0.1808
NRMSE	0.2491

Table 6: Table of Metric Values

And in terms of precision, the precision threshold has the best performance with a precision of ≈ 0.97.

Threshold Type	F1 Score	Recall	Precision
F1	0.8788	0.9355	0.8286
Recall	0.8932	0.9892	0.8142
Precision	0.4839	0.3226	0.9677
Intersection Recall Precision	0.7152	0.6344	0.8194

Table 7: Performance metrics using different thresholds for the test dataset.

The following confusion matrices in Table 8 represent the outcomes for different thresholds. By getting a deeper dive into the results, it's apparent that there is a tradeoff if the goal is to habilitate this model for production purposes. This necessitates a balanced approach when choosing a threshold, considering the specific needs and tolerances of the intended application that should be negotiated with production line leaders and QC operators.

F1 Thr.		Recall Thr.		Precision Thr.		R-P Inter. Thr.	
8	14	0	22	19	3	9	13
11	82	1	92	64	29	24	69

Table 8: Confusion Matrices for different thresholds. R-P Inter. stands for Recall-Precision Intersection. Thr. stands for Threshold.

- Latent Dimension

The pairplot visualization of the latent space is shown in Figure 24. The 2D-**PCA** projection of the latent space is shown in Figure 25.

The pairplot visualization reveals a moderate overlap between the OK images and the anomalies in the test dataset. Visual inspection suggests that the combination of the latent square with either the latent sum or mean exhibits a greater distinction between the two classes, compared to other pairings which show significant overlap. Notably, the "flip" anomaly is most readily distinguishable from the others, which aligns with its distinct visual transformation. Beyond this, there are no significant differences observable among the other anomalies.

When **PCA** is applied using the transformation parameters learned from the validation dataset, the results differ. Although there is no clear clustering or linear separation between the classes, they seem to be distinguishable using an **SVM** model.

A **SVM** was trained on the validation data presented on Figure 69. Due to the immense overlap between OK and NOK data presented in the validation, the final model could not be used in the test dataset due to pour performance (Table 9), even though Figure 25 does appear to show that is possible a separation between the OK and NOK classes.

10	12
20	70

Table 9: Confusion Matrix of the **SVM** model

Despite these challenges, Figure 33 indicates a potential for distinguishing between the OK and NOK classes. One promising research direction could involve considering real world NOK data on the training dataset ir order to train a more suitable **SVM** model, instead of basing it of data-augmented NOKs. This could potentially improve this approach classification power. However, the practical implementation of this approach in industrial settings remains to be explored.

- Filtering

  Just as in the validation dataset, the test dataset was passed through the filtering techniques (section A.3). This works by firstly applying the filtering technique and then filter

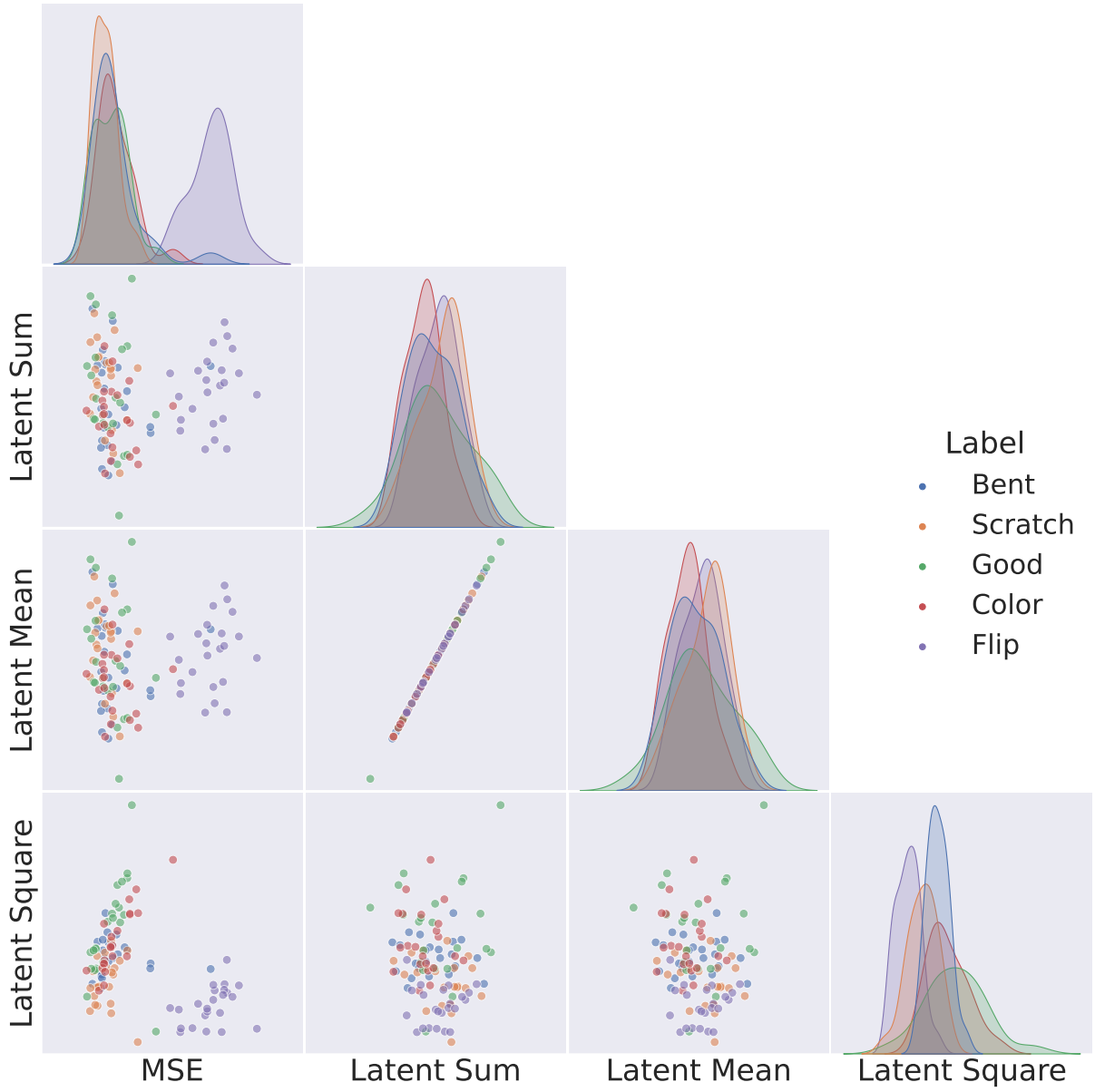


Figure 24: Pairplot visualization of different metrics computed from the latent space from the test dataset.

the error using the threshold values calculated using the validation dataset, presented on Figure 70.

The performance of the filtering techniques in the test dataset is shown in Figure 26. It shows that in comparison to without using filtering techniques (Table 5, Table 6) **MSE** remains unchanged with an **AUC** and **IoU** of ≈ 0.62 (no change) and ≈ 0.26 $(+0.01)$ respectively. But when looking at when filtering techniques are applied, the erosion techniques is highlighted by its performance of ≈ 0.72 **AUC**, and the **NLM** Gray for its ≈ 0.32 **IoU**, corresponding to a bump in performance for the **AUC** and **IoU** of $+0.10$ and

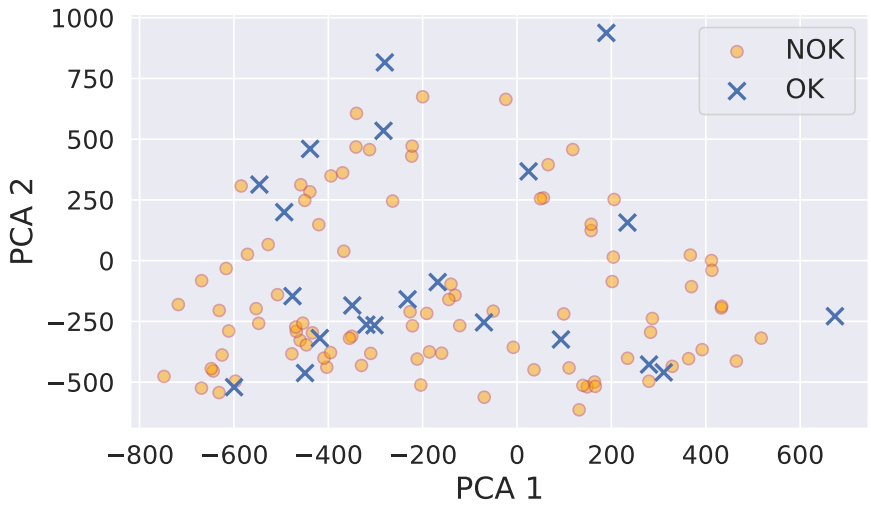


Figure 25: PCA projection of the latent space

$+0.06$ respectively.

Other techniques such as **NLM** Color and **NLM** Gray also had significant improvements to the **AUC** and others such as Gauss Blur, Median Filter and Erosion also had bums in performance in terms of **IoU** when compared to the traditional approach of using the out of the box **MSE** from the model. This indicates that the filtering techniques are improving the model's performance in terms of **AD**.

Furthermore the Edge technique is a special case, since it outputs where the edges are it has ≈ 0 **IoU** and ≈ 0.5 **AUC**, thus its not adequate to use it for classification problems but can be still used as an visualization layer that the factory operator may find useful.

Additionally no metrics registered a worst performance when compared to the **MSE** reference without any post-processing techniques since in the limit, they had comparable performance. This indicates that the filtering techniques are indeed improving the model's performance and can be useful for the purpose of this thesis.

On subsubsection A.5.3.1 the Figure 82 and Figure 83 can be visualized. These figures show the original error image after a filtering technique is performed, the mask generated by the validation inferred threshold and the ground truth mask of the anomaly. This visualization is important to understand the impact of the filtering techniques on the model's performance and by visually inspecting them there are clear examples where the model could identify perfectly the anomaly.

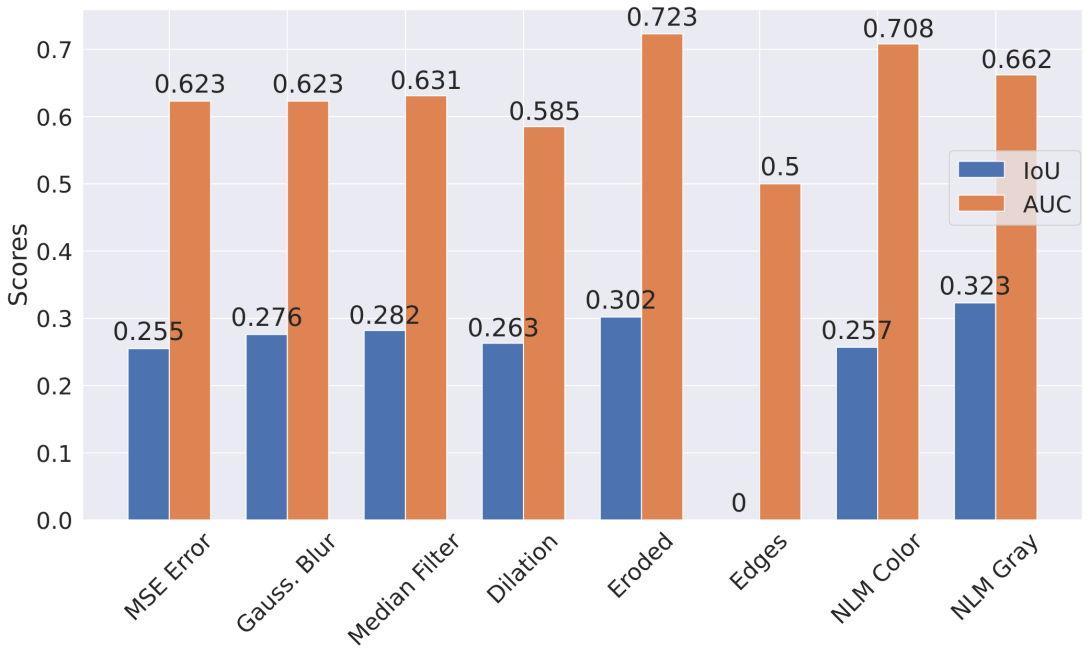


Figure 26: **AUC** and **IoU** when compared with ground truth (both label and true error mask) of the various filtering techniques.

With this results in mind, it's clear that the methodology highlights the anomaly on its output, although with residual noise still present which affects both classification power and interpretability.

- Clustering the error

  The performance of these techniques is depicted in Figure 27. Upon analyzing the results and comparing them with the performance of standalone filtering techniques, it is clear that clustering the errors resulted in weaker classification power, as evidenced by a decrease in the **AUC**, across all filtering techniques subjected to clustering. This suggests that clustering may not be appropriate for classifying images, which is plausible given that clustering primarily reduces the standalone error by stripping away possible significant information, leaving only remnants of the original error image (large enough error clusters).

  In terms of interpretability and **IoU**, the same thing happens. The **IoU** achieved is either slightly worse or comparable to that obtained without the use of clustering techniques, further indicating the unsuitability of clustering for this problem.

  Continuing with the theme of interpretability, subsubsection A.5.3.2 presents Figure 84 and Figure 85. Subjectively, despite a slight reduction in **IoU**, the clustering technique

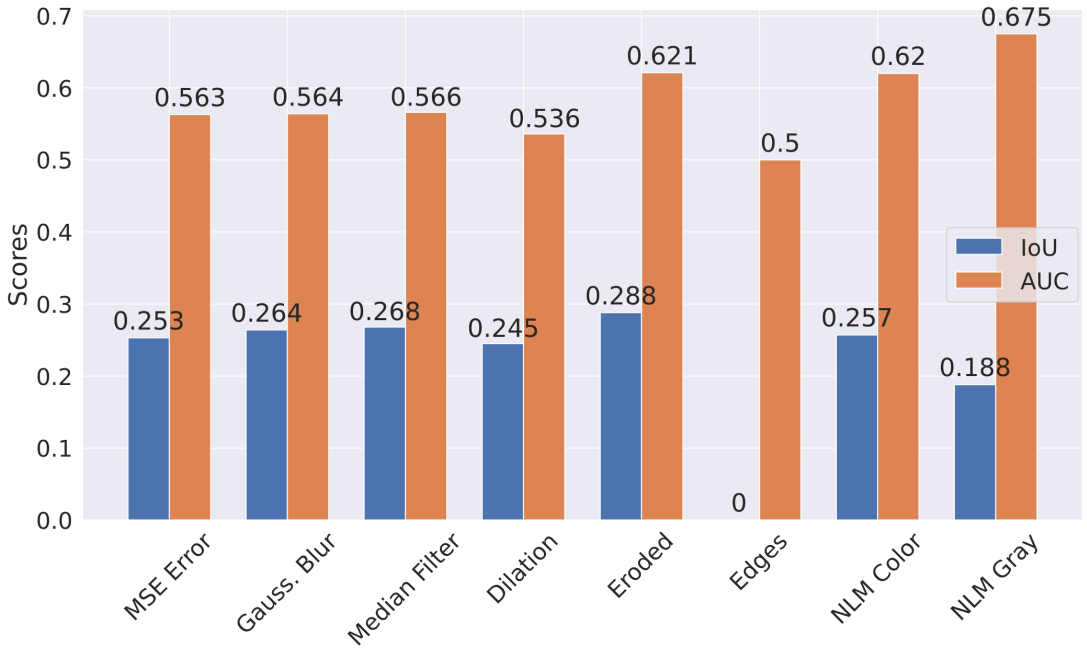


Figure 27: **AUC** and **IoU** compared with the ground truth (both label and true error mask) for various clustering techniques.

appears to have a beneficial effect. The anomalies are more prominently highlighted as noise is reduced, which potentially makes it easier for operators to identify the anomalies.

3.2.2.2 Best Performance Achieved

In order to obtain a final assessment of the performance achieved, one must not only focus on the reconstruction loss of the model. Since **BCE** has a better performance in terms of **AUC**, it was subjected to the same pack of post-processing techniques to explore potential performance enhancements. With this purpose, additional benchmarks are provided in subsubsection A.5.3.3. The initial performance, without any post-processing techniques, is summarized in Table 10.

Metric	AUC	IoU
MSE	0.62	**0.25**
BCE	**0.79**	0.18

Table 10: Comparison of **AUC** and **IoU** for **MSE** and **BCE**. Best performance is highlighted.

It can be seen that the best **AUC** was achieved using the **BCE** metric, with an **AUC** of 0.79 and

the best **IoU** using the **MSE** metric, with an **IoU** of 0.25. In terms of **AUC**, the **BCE** outperformed the **MSE** by 0.17, while in terms of **IoU**, the **MSE** outperformed the **BCE** by 0.07.
The impact of various filtering techniques on performance metrics is shown in Table 11. Here, it can be observed that the highest **AUC** was achieved with the **BCE** using the "Eroded" technique, reaching an **AUC** of approximately 0.82. The highest **IoU** was achieved with the **MSE** using the "**NLM** Gray" technique, with an **IoU** of approximately 0.32. These results indicate that post-processing techniques can significantly enhance performance, depending on the specific metric and technique applied.

Metric	Value	Technique Used
MSE	0.72 **AUC**	Eroded
	0.32 IoU	**NLM** Gray
BCE	**0.82 AUC**	Eroded
	0.22 **IoU**	Dilation

Table 11: Comparison of the best **AUC** and **IoU**, discriminated by metric, and which filtering technique was used to obtain it.

The impact of different clustering techniques on performance metrics is shown in Table 12. Here, the highest **AUC** was achieved with the **BCE** metric using the "Eroded" technique, with an **AUC** of 0.72. The highest **IoU** was achieved with the **MSE** metric using the "Eroded" technique, with an **IoU** of 0.29. These results further emphasize the variability in performance based on the applied technique, suggesting that both filtering and clustering methods can be effectively utilized to improve specific performance metrics.

Metric	Value	Technique Used
MSE	0.68 **AUC**	**NLM** Gray
	0.29 IoU	Eroded
BCE	**0.72 AUC**	Eroded
	0.21 **IoU**	Eroded

Table 12: Comparison of the best **AUC** and **IoU**, discriminated by metric, and which clustering technique was used to obtain it.

In summary, the best performance was achieved using the filtering techniques, resulting in

an **AUC** of 0.82 and an **IoU** of 0.32. Compared to the **SOTA** , which has an **AUC** of 0.86 and an **IoU** of 0.26 (L2 **AE**), these results are quite competitive in terms of classification power, and they even surpass the **SOTA** in terms of interpretability.

3.3 Improved CAE

Competitive results have been achieved, yet there remains rooms for improvement. This section delineates the outcomes of a refined **HPO**, where both search parameters and data utilized in the objective function have been enhanced. The ensuing subsections will elucidate the principal findings, paralleling prior sections.

A predominant critique of the study done until now relates to the inadequacy of data augmentation techniques in mirroring real-world anomalies. To rectify this, a new approach has been formulated to generate more realistic anomalies (subsection A.1.2). Henceforth, methodologies that simulate non-realistic anomalies or pose oversimplified challenges for the model, such as those employing Gaussian noise or random symbols, will be discarded. To be verbose, the data augmentation techniques employed in this section will be "Patch", "In Picture", "Radial Distortion" (subsection A.1.1) and the new data augmented NOKs - "Realistic NOKs Half", "Realistic NOKs Quarter" (subsection A.1.2).

For the **HPO**, adjustments in the degrees of freedom are also being made based on the analysis of Optuna's slice plots from the previous study (Figure 21, Figure 22). Furthermore additional degrees of freedom were also added. These modifications include:

- Exclusively retaining activation functions based on ReLU.
- Increasing the upper limit of the kernel size, particularly in the first layer.
- Increasing the upper limit for the stride.
- Eliminating the use of dropout.
- Allowing Optuna to determine the architecture of the linear layer in the middle layer of the autoencoder.

The scope of potential loss functions will also be expanded. Besides **MSE**, the range will now encompass the **MAE**, **BCE** and **L1**. Furthermore, the upper limits for the maximum number

of convolutional layers and the number of channels (*n_channels*) will be increased to give optuna more freedom. All of these changes are summarized in Table 13.

Hyperparameter Name	Range	Log
LR	[1e-5, 1e-1]	Yes
Model Loss	[MSE, BCE, L1, MAE]	N/A
Latent Dimension	$[1, \frac{\text{IMAGE_DIM}}{10}]$	Yes
Activation Function	ReLU, LeakyReLU	NA
Use Batch Normalization	True, False	NA
Maximum Convolutional Layers	[1, 20]	No
Stride X	$[1, \min(20, \frac{\text{IMAGE_SIZE}}{2})]$	No
Kernel Size X	$[1, \min(20, \text{IMAGE_SIZE})]$	No
#Channels X	[1, 512 (first layer) or as per previous layer #Channels]	No
Use Max Pooling X	True, False	NA

Table 13: **HP** selection range for the improved **AE** architecture. Note that 'log' refers to whether the **HP** is optimized in the logarithmic space.

As similarly to the previous study, the **HP** are optimized using the Optuna framework to maximize the objective function (subsubsection 2.1.4.1).

3.3.1 Optimization Process

Following the **HP** tuning process, several figures were generated. As depicted in Figure 28, the optimization trajectory demonstrates a successful increase in the objective function value over time. It was determined that, after 1000 trials, Optuna identified the optimal set of **HP** during trial 757, achieving an objective function score of approximately 1.06. For a balanced comparison, the earlier study continued optimization until reaching 1000 trials, the revised maximum, yet no further improvements were noted beyond this extension. It is important to acknowledge that comparisons between the current and previous studies trial final values are not feasible, as the validation dataset has been altered, as previously described.
In contrast to the previous study, this optimization process experienced numerous trials where the architectures exceeded the available video memory, leading to unsuccessful outcomes. The suspected reason for this phenomenon is the utilization of a more refined **HP** space in

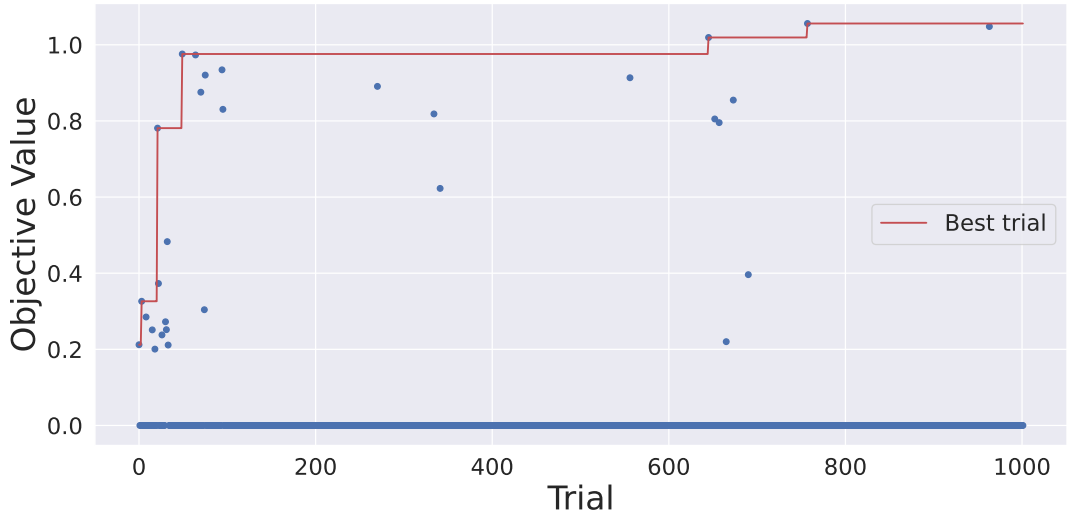


Figure 28: Optimization history of the **HP** for the **AE** architecture. Note that, if a trial failed is plotted as 0.

the current study, which effectively reduced "noise"—variations in **HP**s, such as activation functions, that do not necessarily increase **GPU** utilization. Consequently, Optuna rapidly optimized the **HP** to their fullest potential, accelerating the optimization process but simultaneously resulting in fewer successful trials. The best trial parameters obtained from the optimization process are detailed in Table 14.

Following the evaluation of **HP**s, the Optuna framework offers a detailed visualization to facilitate a deeper understanding of the optimization process. By analyzing Figure 29, it is evident that the most influential parameters include the latent dimension and the number of channels in the first layer, which aligns with the findings from the previous study. Furthermore the new parameter, the maximum number of linear layers, is ranked 2nd in importance giving attention to the linear layers in the middle of the **AE**. This is expected since those layers are are responsible for the crux of the compression and decompression process.

On the **EDF** plot (Figure 30) it is evident that Optuna attained maximum confidence at the objective value corresponding to the best trial. In comparison with the previous study, the function's growth rate is markedly accelerated, indicating a more rapid optimization process. This observation aligns with the optimization history, succinctly indicating that Optuna required significantly fewer successful trials to achieve peak performance compared to the previous study. This observation is substantiated by the data showing that the cumulative probability of approximately 1 was achieved with only 38 successful trials in this study, compared to 431

Parameter	Value
Latent Dimension	26154
Activation Function	ReLU
Use Batch Normalization	False
Maximum Convolutional Layers	3
Stride #1	2
Kernel Size #1	19
Number of Channels #1	152
Use Max Pooling #1	True
Stride #2	5
Kernel Size #2	14
Number of Channels #2	84
Use Max Pooling #2	True
Stride #3	4
Kernel Size #3	11
Number of Channels #3	44
Maximum Linear Layers	14
Linear Dimension #1	5793
Starting **LR**	≈ 0.00011
Chosen Loss	L1

Table 14: Best set of **HP**, achieved during the best Optuna's trial.

in the previous study, representing a speed increase by a factor of approximately 11. This acceleration is attributed to the refined **HP** space, enabling the optimizer to converge more rapidly to the optimal solution.

To provide a comprehensive understanding of the selected **HP** and their impact on the objective function, two slice plots were generated and can be viewed in section A.6, which provide additional discussion for the curious reader.

3.3.2 Test Results

After applying the validation dataset methodology from the previous section, the model's performance is now assessed on the test dataset. Parameters tuned during validation are

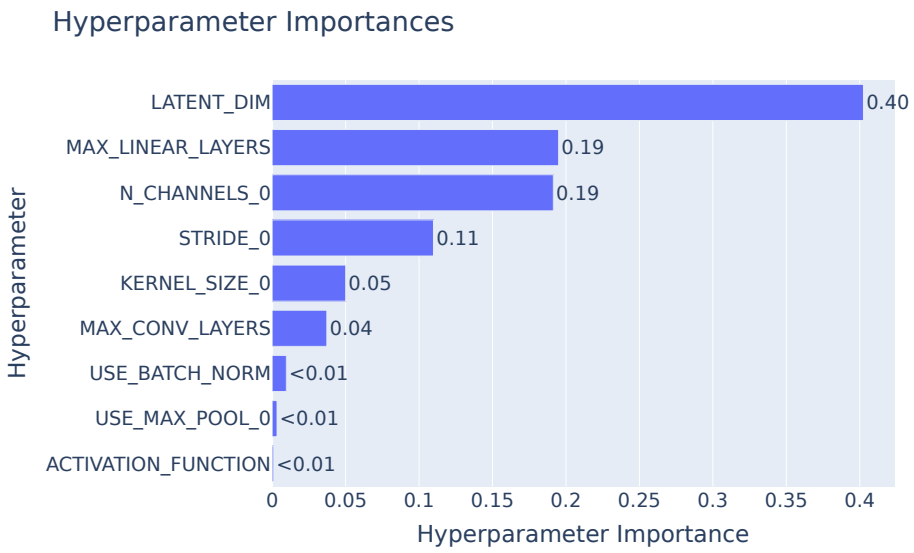


Figure 29: Parameter importance as determined by the Optuna framework for the **AE** architecture.

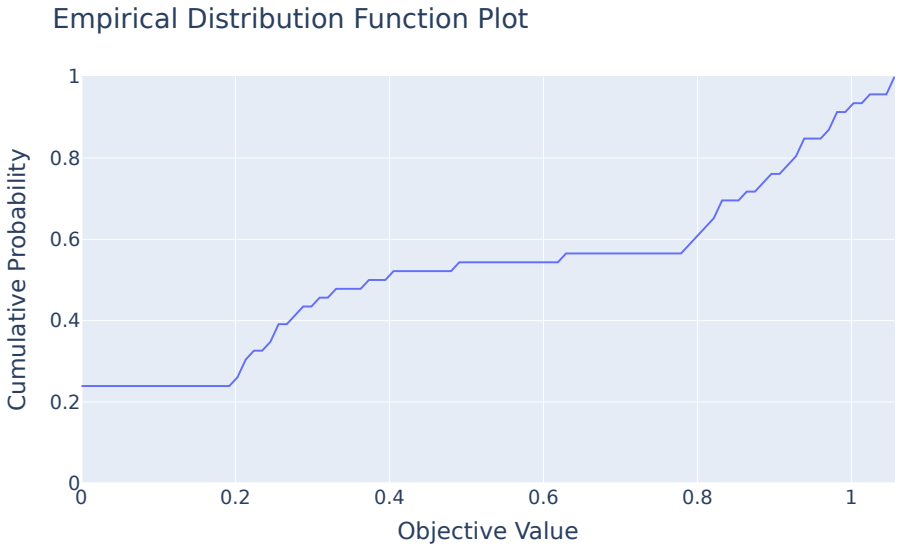


Figure 30: Empirical distribution function for the **HP** of the autoencoder architecture.

carried over for test dataset evaluation for metrics computation.

Visual inspection of Figure 31 reveals the model's proficiency in reconstructing OK images, closely resembling originals. However, anomalies, particularly in 'bent', have noticeable reconstruction errors with help identify where the anomaly is located. 'Scratch' and 'color' anomalies, though discernible, often blend with background noise, reducing distinctiveness. Furthermore, errors near the edges of 'flip' anomalies indicate model difficulty in this class.

In evaluating the performance of the current model on the test dataset, results are detailed

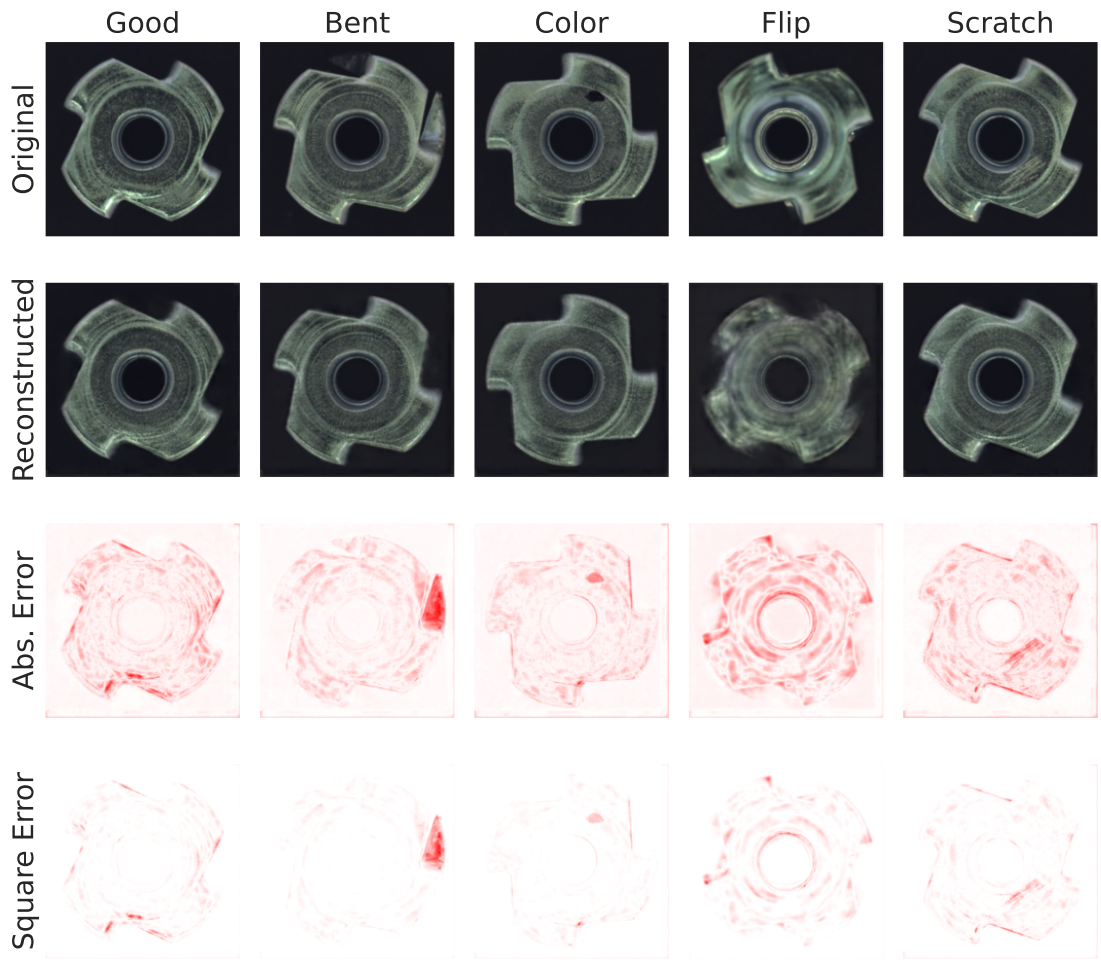


Figure 31: Reconstruction of the test dataset using the improved **CAE** architecture. The first row shows the original images, the second row the reconstructed images, the third row the absolute error map and the last row the square error.

in Table 15 using various reconstruction error metrics. Notably, the model achieves an **AUC** of approximately 0.86 ($+0.07$ than the previous approach) with the **BCE** metric, surpassing to the **SOTA** reported **AUC** of 0.82 Bergmann et al. (2019). Additionally, an **IoU** of approximately 0.30 ($+0.05$ compared to the previous approach) is obtained using the model's **MSE** loss, again surpassing the literature's benchmark of 0.26. These results signify that the new approach not only enhances the model's performance compared to the previous one but also outperforms hand-crafted state-of-the-art methods.

As done before, subsequent sections will explore post-processing techniques (referenced in subsection 2.1.6), utilizing parameters optimized earlier with the validation dataset. These enhancements are intended to further boost model performance, potentially exceeding current benchmarks.

Metric	AUC	Accuracy	Recall	Precision	F1
MSE	0.6163	0.3826	0.2473	0.9583	0.3932
MAE	0.5665	0.3826	0.2473	0.9583	0.3932
BCE	0.8622	0.6609	0.6022	0.9655	0.7417
SSIM	0.5635	0.3739	0.2366	0.9565	0.3793
PSNR	0.6163	0.4174	0.3011	0.9333	0.4553
NRMSE	0.5665	0.3826	0.2473	0.9583	0.3932

Table 15: Evaluation metrics of the test dataset.

Metric	IoU
MSE	0.3012
MAE	0.2982
BCE	0.1848
NRMSE	0.2982

Table 16: Table of Metric Values

3.3.2.1 Evaluation Metrics

- Baseline Approach

  As noted in Table 17, the performance metrics for the thresholds outlined in Table 40 are presented. Similar to the previous section, the Recall threshold exhibits superior performance in terms of F1 score and Recall, approximately 0.90 and 0.98 respectively. Regarding precision, the Precision threshold leads, achieving approximately 0.97 on the test dataset.

Threshold Type	F1 Score	Recall	Precision
F1 Threshold	0.8202	0.7849	0.8588
Recall Threshold	0.8976	0.9892	0.8214
Precision Threshold	0.4590	0.3011	0.9655
Intersection Recall Precision	0.7273	0.6452	0.8333

Table 17: Performance metrics using different thresholds for the test dataset.

The confusion matrices for these thresholds are shown in Table 18, identical to the previous sections. Yet again it should be noted that different metrics enhance different facets of performance, and no single threshold emerges as universally optimal.

F1 Thr.		**Recall Thr.**		**Precision Thr.**		**R-P Inter. Thr.**	
10	12	2	20	21	1	10	12
20	73	1	92	65	28	33	60

Table 18: Confusion matrices for different thresholds. R-P Inter. stands for Recall-Precision Intersection. Thr. stands for Threshold.

- Latent Dimension

  As in the previous section with the validation dataset, the latent space in the test dataset was analized. The pairplot visualization of the latent space is displayed in Figure 32, while the 2D **PCA** projection is shown in Figure 33.

  The pairplot reveals a heavy overlap between OK images and anomalies in the test dataset. Notably, the "flip" anomaly stands out the most, likely due to its unique visual transformation in combinations of the latent space metrics with the **MSE** loss. However, no significant differences are observed among the other anomalies.

  When applying **PCA** with transformation parameters learned from the validation dataset, the results differ from the previous section. Although no clear clustering or linear separation between classes is observed, they appear distinguishable using an **SVM** model, with minimal overlap between OK and NOK classes.

  The **SVM** model was trained using validation data as the training dataset. This approach led to significant issues, as there was considerable overlap between the OK and NOK data within the validation set, resulting in poor model performance on the test dataset (Table 19). The model predominantly misclassified all images as NOK, correctly identifying none of the OK images and incorrectly classifying three images as OK which were actually NOK.

  Again, this line of though could have some fruition if the training step was presented with real world NOK data. This could be a possible line of research to further explore

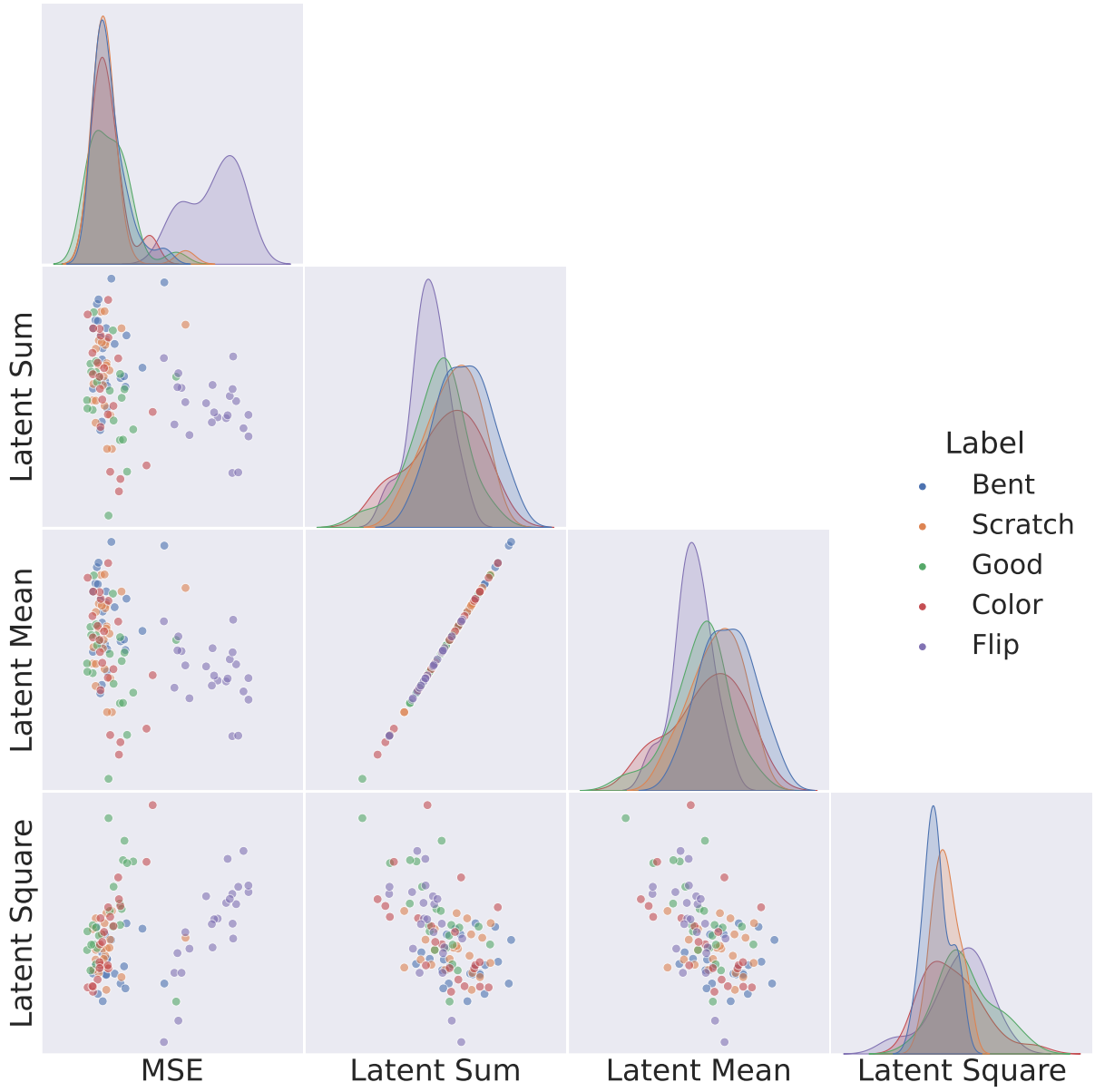


Figure 32: Pairplot visualization of different metrics computed from the latent space from the test dataset.

0	22
3	90

Table 19: Confusion Matrix of the **SVM** model

the latent space since this would allow the **SVM** to learn the features of the NOK data and then be able to separate the OK and NOK data.

- Filtering

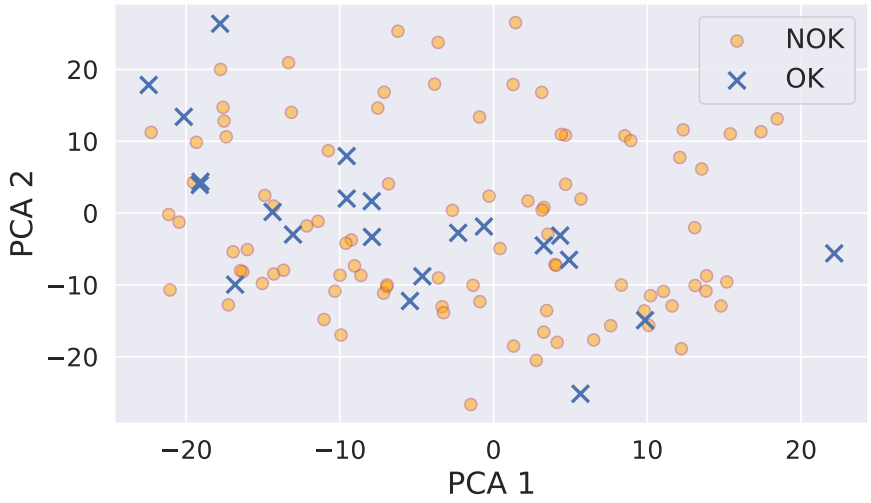


Figure 33: PCA projection of the latent space

The test dataset underwent filtering techniques which involve applying the filter, followed by error thresholding using values calculated from the validation dataset, as depicted in Figure 94.

The Figure 34 illustrates the impact of filtering techniques on the test dataset. Without filtering, **MSE** remains stable, while **AUC** and **IoU** hold steady at approximately 0.62 and 0.31, respectively (Table 15, Table 16). Although most filtering methods do not significantly enhance performance, notable improvements include Erosion, which raises **AUC** to 0.68. Additionally, both color and gray **NLM** filters improve, unlike the outcomes with dilation and edge techniques for **AUC**. For **IoU**, color and gray **NLM** filters show gains, reaching about 0.33 and 0.37, respectively. Overall, filtering boosts **AUC** and **IoU** performance by roughly 10% and 19%.

Furthermore, the Edge technique presents a unique case. As it primarily delineates error outlines, its **IoU** is approximately zero and **AUC** around 0.5, rendering it unsuitable for classification tasks. Nevertheless, it can still serve as a valuable visualization tool for factory operators.

Visualization of various filtering techniques is presented in subsubsection A.6.3.1, specifically in Figure 106 and Figure 107. These figures illustrate the original error image following a filtering technique, alongside the mask generated by the inferred validation threshold and the ground truth anomaly mask. Additional discussion and examples of

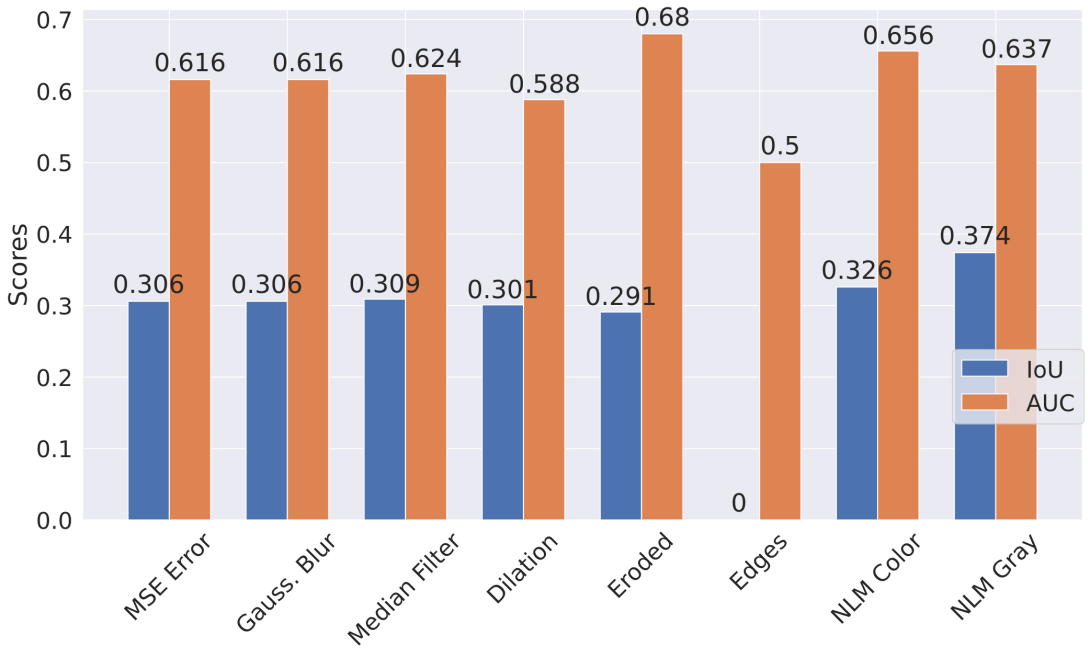


Figure 34: **AUC** and **IoU** when compared with ground truth (both label and true error mask) of the various filtering techniques.

accurate anomaly detection by the model are also included.

- Clustering the error

  The clustering techniques were applied to the test dataset. The performance of these techniques is depicted in Figure 35. Let the reader note the baseline for comparison is the **MSE** loss, which achieved an **AUC** of 0.62 and an **IoU** of 0.31 (Table 15, Table 16).

  Analysis of the results, compared to standalone filtering techniques, shows that clustering errors generally reduces classification power. This is indicated by the **AUC**, which is either unchanged or decreased for all filtered techniques subjected to clustering, except for Erosion, which increased the **AUC** by 0.01. Similarly, in terms of interpretability and **IoU**, only **NLM** color showed improvement, achieving an increase of 0.033.

  Additional visualizations of clustering techniques are available in subsubsection A.6.3.2, featuring Figure 108 and Figure 109.

3.3.2.2 Best Performance Achieved

To achieve a thorough evaluation of performance, it is important to consider more than just the model's reconstruction loss. Given that **BCE** exhibits superior performance in terms of

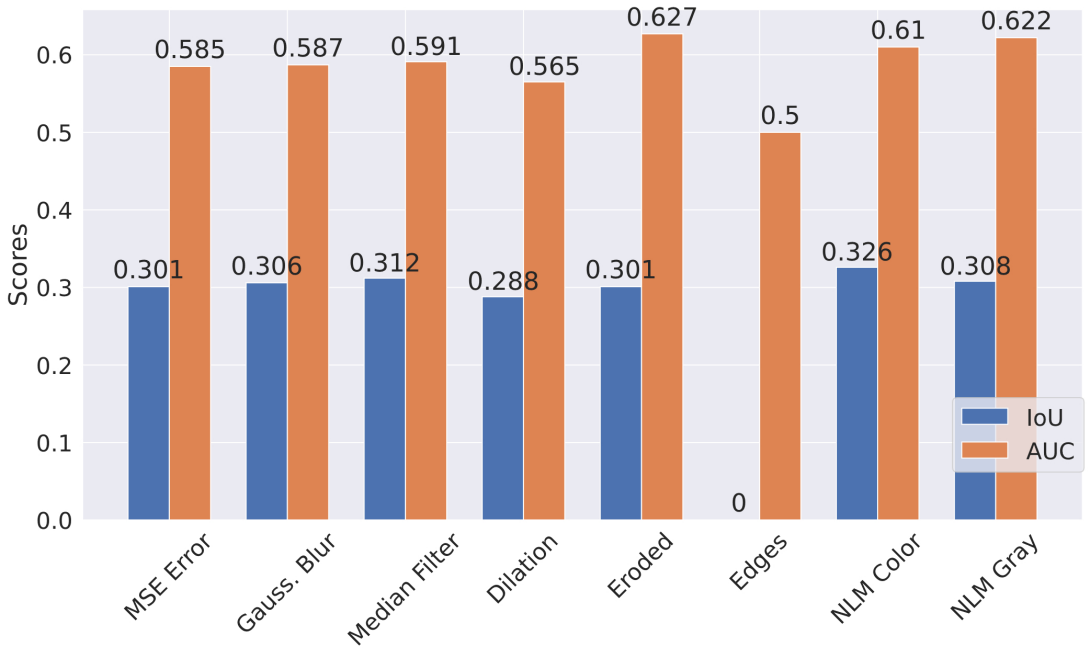


Figure 35: **AUC** and **IoU** compared with the ground truth (both label and true error mask) for various clustering techniques.

AUC, it was subjected to the same set of post-processing techniques to explore potential enhancements in performance. Additional benchmarks are provided in subsubsection A.5.3.3. The initial performance, prior to applying any post-processing techniques, is summarized in Table 10.

Metric	AUC	IoU
MSE	0.62	**0.30**
BCE	**0.86**	0.19

Table 20: Comparison of **AUC** and **IoU** for **MSE** and **BCE**

The results indicate that the highest **AUC** of approximately 0.86 was achieved using the **BCE** metric, while the best **IoU** of approximately 0.30 was obtained with the **MSE** metric. The **BCE** metric outperformed the **MSE** in **AUC** by 0.24, whereas **MSE** outperformed **BCE** in **IoU** by 0.11. Table 21 shows the effect of different filtering techniques on performance metrics. The **NLM** Gray technique yielded the highest **AUC** of approximately 0.87 with the **BCE** metric, and the highest **IoU** of approximately 0.37 with the **MSE** metric.

The impact of different clustering techniques on performance metrics is detailed in Table 12. The highest **AUC**, 0.75, was achieved using the **BCE** metric with the "Eroded" technique.

Metric	Value	Technique Used
MSE	0.68 **AUC**	Eroded
	0.37 IoU	**NLM** Gray
BCE	**0.87 AUC**	**NLM** Gray
	0.19 **IoU**	Dilation

Table 21: Comparison of the best **AUC** and **IoU**, discriminated by metric, and which filtering technique was used to obtain it.

Meanwhile, the highest **IoU**, 0.33, was achieved using the **MSE** metric with the "**NLM** Color" technique.

Metric	Value	Technique Used
MSE	0.63 **AUC**	Eroded
	0.33 IoU	**NLM** Color
BCE	**0.75 AUC**	Eroded
	0.18 **IoU**	Eroded

Table 22: Comparison of the best **AUC** and **IoU**, discriminated by metric, and which clustering technique was used to obtain it.

In summary, the best performance was achieved using the filtering techniques, resulting in an **AUC** of 0.87 ($+0.05$ than the previous model) and an **IoU** of 0.37 ($+0.05$ than the previous model). Compared to the **SOTA**, which has an **AUC** of 0.86 and an **IoU** of 0.26 (L2 **AE**), they surpass the literature's benchmark, by $\approx 1\%$ on **AUC** and $\approx 42\%$ on **IoU**, showcasing the effectiveness of the proposed approach in industrial **AD** tasks.

3.4 Variational AutoEncoder

In this section the results of the **VAE** model will be explored. This model was trained in the same training scenario as section 3.3 since it yielded the best results.
Firstly, the model **HP** space closely resembles that of the **VAE**, but now it includes a new **HP**, denoted as α. This **HP** is used to balance the **KL** divergence loss with the reconstruction loss. The range for α has been set between $[1e-6, 1e+6]$ on a logarithmic scale, while the other **HP**s remain unchanged.

Before proceeding with the usual analysis, it is a common practice to evaluate whether the latent space components of a **VAE** model, specifically z_μ and z_γ (where $\gamma = \log \sigma^2$), converge to the predetermined normal distribution, as almost as a sanity check. Using test data corresponding to OK images, the latent space components were computed, yielding the results shown in Table 23. Since the results closely resemble a normal distribution (Where $\mu = 0$ and $\sigma = 1$), they indicate that the latent space components converged to the desired normal distribution, in part validating the successful training.

Metric	Value
μ	0.051
σ	1.026

Table 23: Latent space components averages for the **VAE** model using OK images from the test dataset. Note that $\sigma = \exp\left(\frac{1}{2}\gamma\right)$.

The following sections will procede with the same methodology as applied in the previous models except for an aditional analysis where approaches only aplicable to **VAE** models will be explored.

3.4.1 Optimization Process

Upon completion of the **HP** tuning process, a series of figures were produced to enhance the comprehension of the optimization process. As shown in Figure 36, the optimization trajectory effectively exhibits a progressive increase in the value of the objective function over time. After performing 1000 trials, Optuna determined the optimal set of **HP** during trial 429, reaching an objective function score approximately equal to 1.08.
Just like the previous study, this optimization process encountered multiple trials where the architectures surpassed the available video memory, resulting in unsuccessful attempts, due to the same causes as before. The optimal trial parameters derived from the optimization process are outlined in Table 24.
Following the evaluation of **HP**s, the Optuna framework offers a detailed visualization to promote a thorough understanding of the optimization process. Interestingly, upon examining Figure 37, the stride used on the first layer of the model emerges as the most influential parameter. This finding is curious as it diverges from the results of the previous study, where the

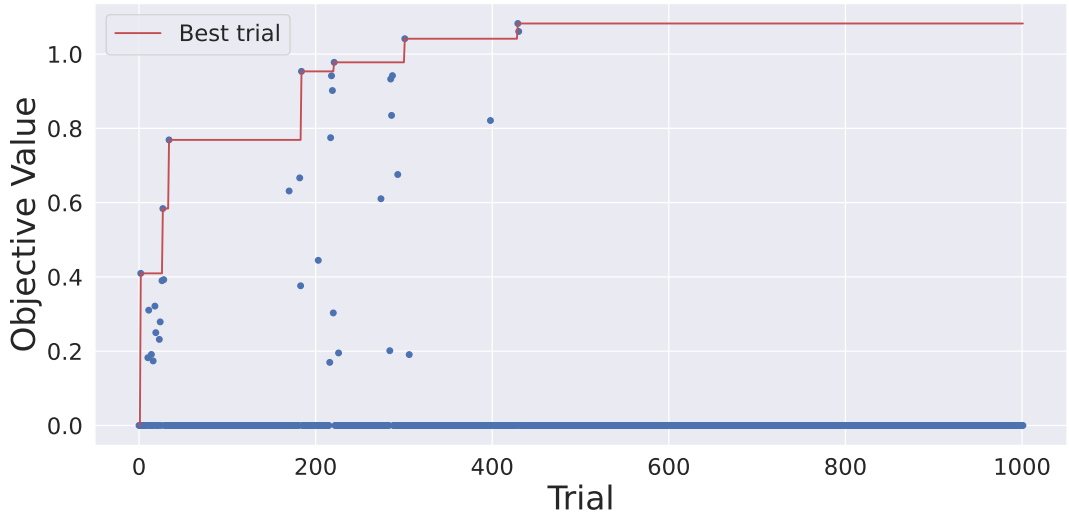


Figure 36: Optimization history of the **HP** for the **VAE** architecture. Note that, if a trial failed is plotted as 0.

latent dimension was the most impactful parameter. The latent dimension comes in as the second most influential parameter, succeeded by the maximum number of linear layers. This shift in parameter importance compared to the previous study suggests that these **HP** may hold more significance when learning the distribution of OK images, as opposed to merely learning to reconstruct them, as was the case with **CAE**.

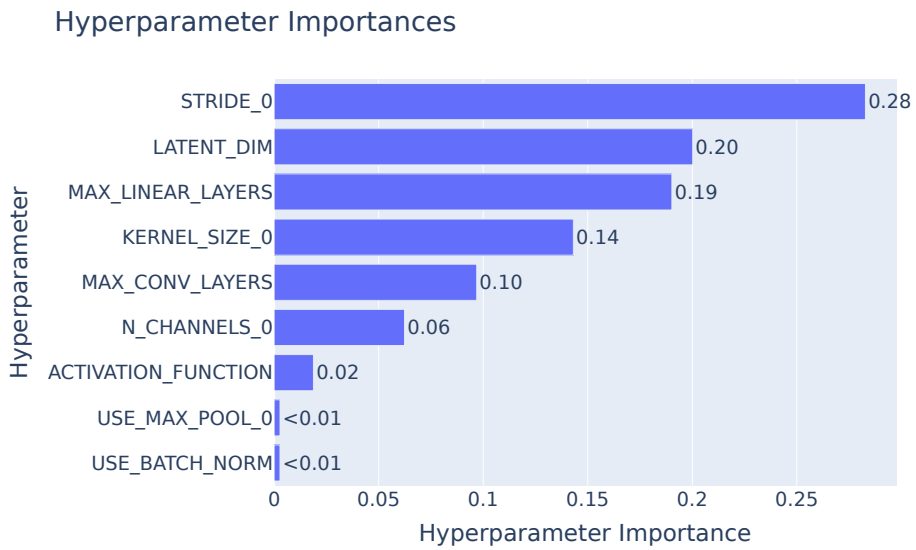


Figure 37: Parameter importance as determined by the Optuna framework for the **VAE** architecture.

As depicted in the **EDF** plot (Figure 38), it is clear that Optuna reached peak confidence at the

Parameter	Value
Latent Dimension	9199
Activation Function	ReLU
Use Batch Normalization	True
Maximum Convolutional Layers	12
Stride #1	3
Kernel Size #1	19
Number of Channels #1	7
Use Max Pooling #1	True
Stride #2	16
Kernel Size #2	17
Number of Channels #2	269
Use Max Pooling #2	False
Stride #3	1
Kernel Size #3	2
Number of Channels #3	106
Use Max Pooling #3	False
Maximum Linear Layers	2
Linear Dimension #1	7878
Starting **LR**	≈ 0.00008
Chosen Loss	**MSE**
Alpha	392061

Table 24: Best set of **HP**, achieved during the best Optuna's trial.

objective value corresponding to the best trial. This also exhibits a growth rate very similar to the previous study. As observed earlier, this refined set of **HP**s, compared to the initial study's parameter batch, enables Optuna to converge more rapidly to the optimal solution. This observation is supported by the data, which shows that a cumulative probability of approximately 1 was achieved with only 40 successful trials in this study, a number very close to the 38 in the previous study, compared to 431 in the initial **CAE** study (section 3.2).
For a thorough understanding of the chosen **HP** and their influence on the objective function, two slice plots have been created and are available in section A.7.

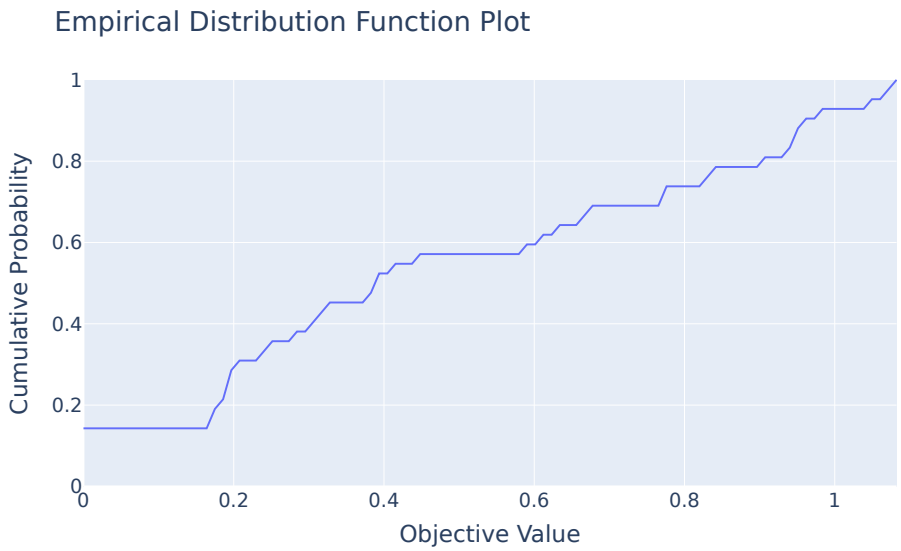


Figure 38: Empirical distribution function for the **HP** of the **VAE** architecture.

The ensuing sections provide an in-depth analysis of the trained model, with a focus on performance evaluation. In addition to traditional benchmarking methods, a range of advanced post-processing techniques (subsection 2.1.6) will be utilized as done before.

3.4.2 Test Results

In this section, the model's performance is evaluated on the test dataset. Parameters optimized during validation are used to assess the test dataset.

A visual examination of Figure 39 reveals the model's effectiveness in reconstructing OK images, which closely match the originals, as indicated by minimal reconstruction errors. The model highlights 'bent' anomalies with localized errors that help pinpoint the location of the anomaly. For other anomalies, while the reconstruction error does suggest where the anomaly is, its intensity might be easily mistaken for noise.

The performance of the current model on the test dataset is summarized in Table 25, using various reconstruction error metrics. Unfortunately, the model achieves an **AUC** of only 0.71, which falls below the scores set by earlier models (section 3.2, section 3.3). Additionally, an **IoU** of approximately 0.11 is recorded, using the model's **MSE** loss, also underperforming compared to previous efforts. These results indicate that the **VAE** model may not be as effective as the previous approaches for **AD**.

Even though the pour performance without any post-processing being done, the model's potential can be further explored through post-processing techniques (referenced in sub-

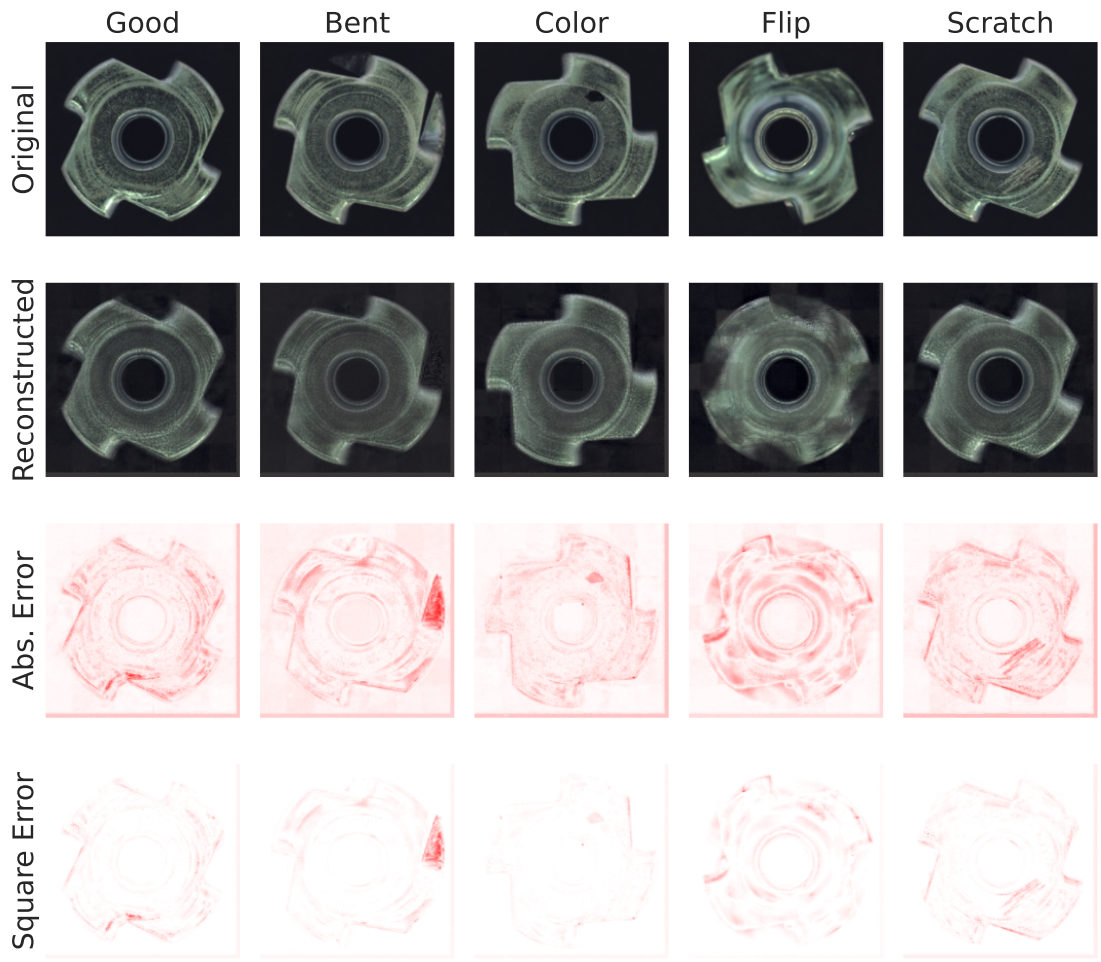


Figure 39: Reconstruction of the test dataset using the **VAE** architecture. The first row shows the original images, the second row the reconstructed images, the third row the absolute error map and the last row the square error.

Metric	AUC	Accuracy	Recall	Precision	F1
MSE	0.2502	0.3043	0.3011	0.6512	0.4118
MAE	0.2669	0.3739	0.3871	0.7059	0.5000
BCE	0.7082	0.6435	0.6344	0.8939	0.7421
SSIM	0.3240	0.4696	0.5269	0.7424	0.6164
PSNR	0.2502	0.4609	0.5161	0.7385	0.6076
NRMSE	0.2669	0.3739	0.3871	0.7059	0.5000

Table 25: Evaluation metrics of the test dataset.

section 2.1.6) as done before. Note that additional results and discussion can be found in subsection A.7.3 regarding the test dataset.

Metric	IoU
MSE Recon Loss	0.1129
MAE	0.1128
BCE	0.0219
NRMSE	0.1128

Table 26: Table of Metric Values

3.4.2.1 Evaluation Metrics

- Baseline Approach

  As observed in Table 27, the performance metrics for the thresholds specified in Table 43 are presented. The thresholds for Recall and F1 score show strong results, achieving approximately 0.89 for F1 score and 0.81 for Precision. The recall threshold records the highest value for recall achieving 1.

Threshold Type	F1 Score	Recall	Precision
F1 Threshold	0.8889	0.9892	0.8070
Recall Threshold	0.8942	1.0000	0.8087
Precision Threshold	0.0000	0.0000	0.0000
Intersection Recall Precision	0.8308	0.8710	0.7941

Table 27: Performance metrics using different thresholds for the test dataset.

The confusion matrices for various thresholds are displayed in Table 28, consistent with earlier sections. It's important to note that different metrics highlight different aspects of performance, and no single threshold proves to be universally optimal. The model's practical utility depends on these detailed evaluations, which should be coordinated with production line leaders and **QC** operators. In terms of performance and comparatively to the previous model (subsection 3.3.2), the confusion matrices generally exhibit pour performance either classifying all images as NOK or failing to classify any images as OK.

- Latent Dimension

  Similar to the validation dataset analysis, the latent space of the test dataset was ex-

F1 Threshold		Recall Threshold		Precision Threshold		Intersection of Recall and Precision	
0	22	0	22	22	0	1	21
1	92	0	93	93	0	12	81

Table 28: Confusion Matrices for different thresholds. The abbreviations used are: F1 Threshold (F1 Thr.), Recall Threshold (Recall Thr.), Precision Threshold (Precision Thr.), and Intersection of Recall and Precision (R-P Inter. Thr.).

amined, with a pairplot illustrating this space in Figure 40. The pairplot analysis reveals substantial overlap between OK and all types of NOK images in the test dataset, complicating their differentiation.

After applying **PCA** with parameters from the validation dataset, the transformation effects on the test dataset are displayed in Figure 41. The plot reveals a substantial overlap among all classes, complicating the use of an **SVM** model for differentiation.

The hypothesis was tested by training an **SVM** model using the validation data. This approach encountered significant challenges due to considerable overlap between the OK and NOK data in the validation set, leading to poor results on the test dataset (see Table 29), similar to earlier findings. The model incorrectly classified all images as NOK, failing to correctly identify any OK images.

0	22
0	93

Table 29: Confusion Matrix of the **SVM** model

Exploring this line of thought could prove fruitful if the training phase incorporated real-world NOK data. This approach would offer a new avenue for research by allowing the **SVM** to learn the characteristics of real NOK data, potentially enhancing its ability to differentiate between OK and NOK data.

- Filtering

  The test dataset was subjected to various filtering techniques. These methods encompass the application of filters, followed by error thresholding using values derived from

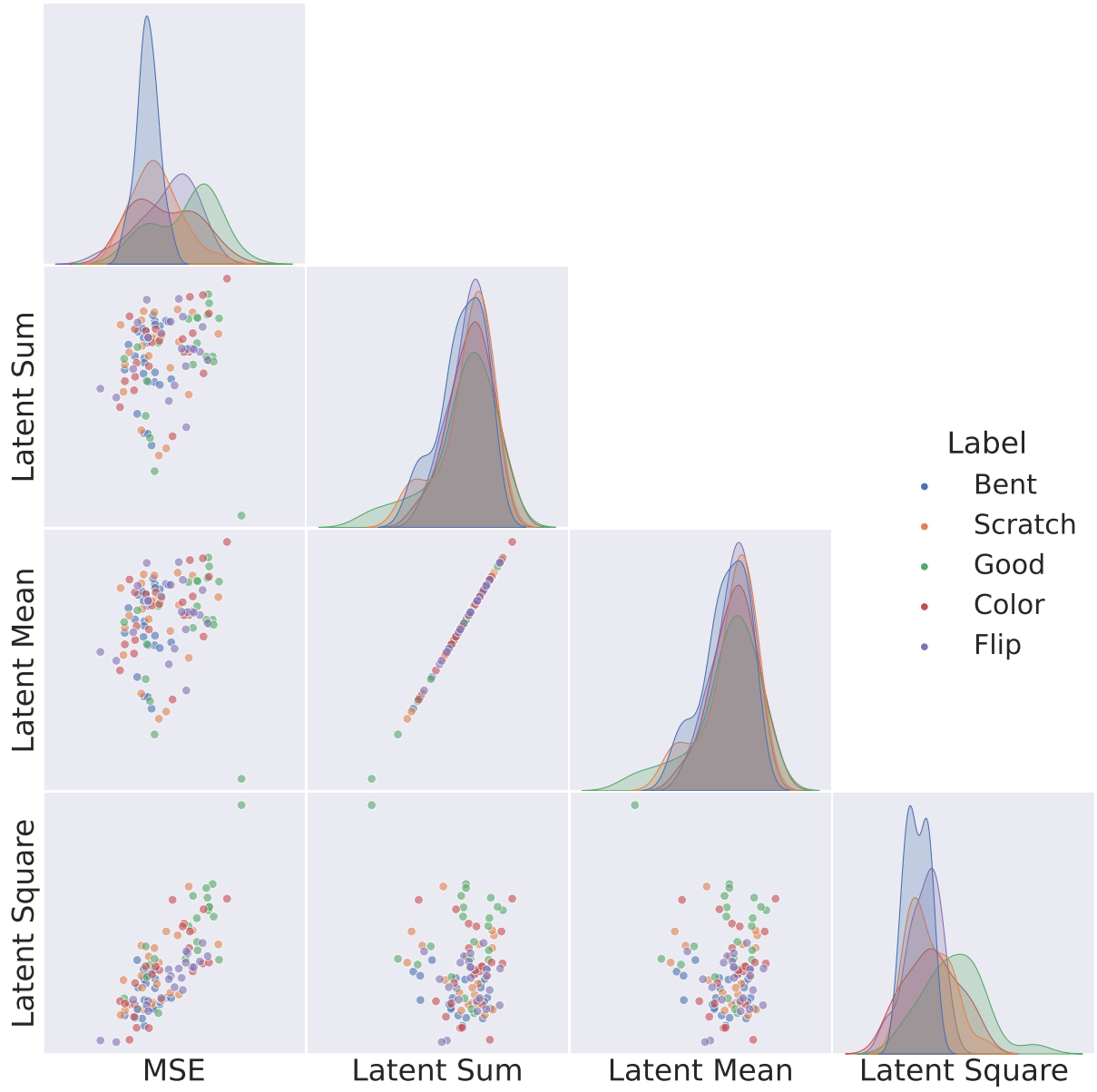


Figure 40: Pairplot visualization of different metrics computed from the latent space from the test dataset.

the validation dataset (Figure 118).

The effectiveness of various techniques on the test dataset is illustrated in Figure 42. Most filtering techniques did not enhance the **AUC** value, except for the Eroded and Edges filtering techniques, which increased it to 0.3 and 0.51 (random choice), respectively. Additionally, all the filtering techniques improved the **IoU** score (except Edges, and **NLM** color and gray), achieving an **IoU** score of 0.21. Unfortunately, the performance of the model is quite poor compared to section 3.3, showing a decrease in performance of -22% and -48% in **AUC** and **IoU**, respectively.

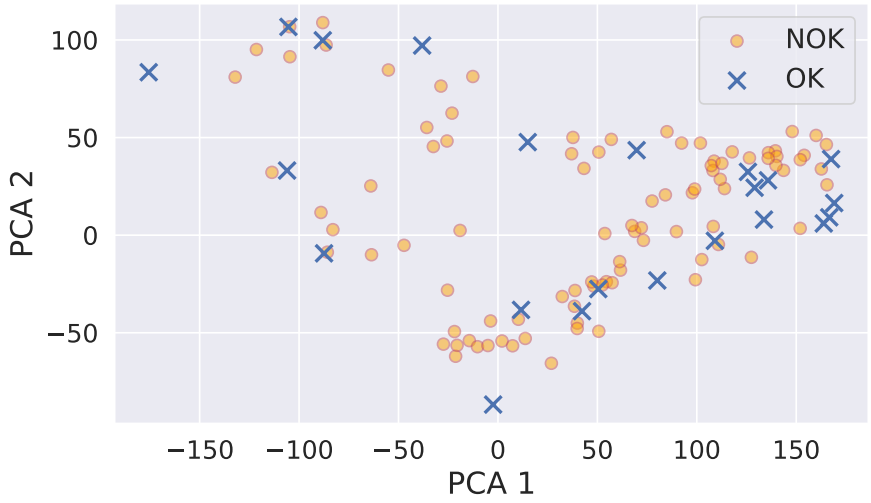


Figure 41: PCA projection of the latent space

It should be noted that the results presented here are less robust than those obtained from previous models, as indicated by significantly lower **AUC** and **IoU** values, with the highest **AUC** being only marginally better than random chance.

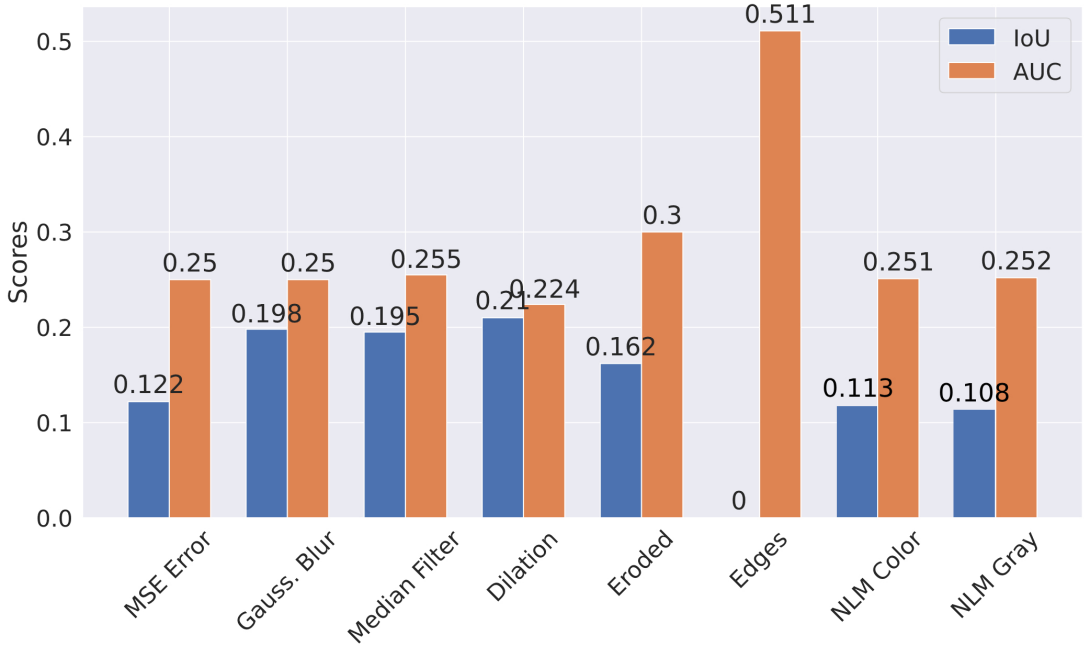


Figure 42: **AUC** and **IoU** when compared with ground truth (both label and true error mask) of the various filtering techniques.

Visualization of the different filtering techniques can be found on subsubsection A.7.3.1 (Figure 130 and Figure 131) can be visualized. These figures display the original error

image post-filtering, the mask generated by the validation inferred threshold, and the ground truth mask of the anomaly.

- Clustering the error

  Clustering techniques were implemented on the test dataset, with their performance illustrated in Figure 43. It should be noted that the baseline for comparison is the **MSE** loss, which attained an **AUC** of 0.25 and an **IoU** of 0.12 (Table 25, Table 26).

  Upon examination of the results and juxtaposing them with the performance of independent filtering techniques, its clear that they did not have an impactful effect on the **AUC** performance obtaining very comparable results to only using the filtering techniques (Figure 42) and in terms of **IoU** clustering surprisingly yielded an negative effect across all metrics. In summary the best results were an **AUC** of 0.51 and an **IoU** of 0.12 which didn't show improvements when compared to just using filtering techniques.

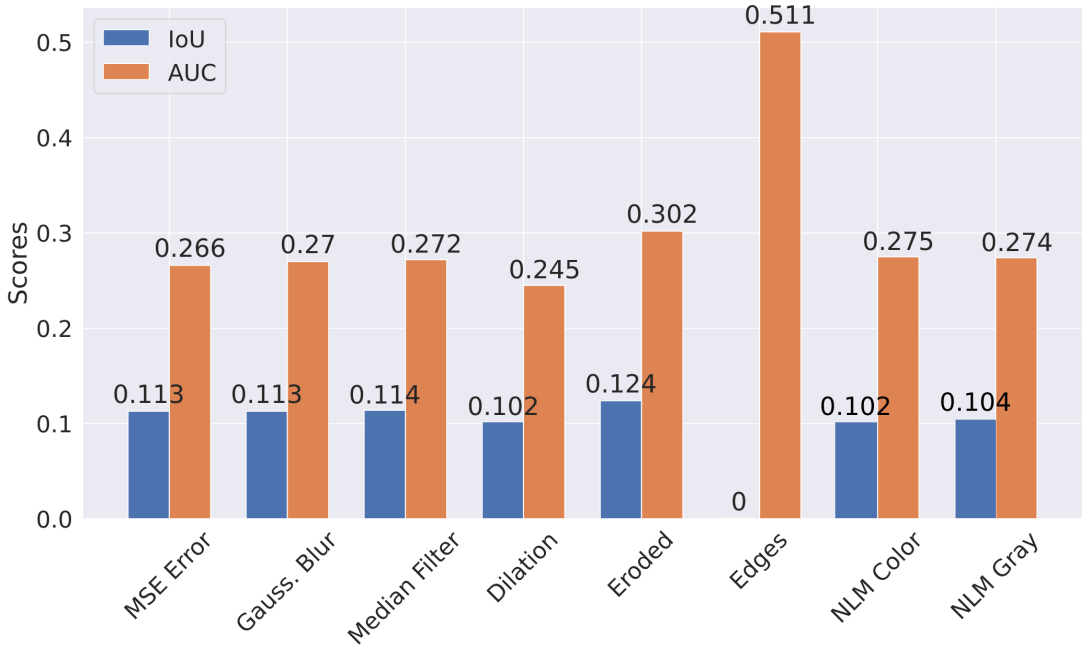


Figure 43: **AUC** and **IoU** compared with the ground truth (both label and true error mask) for various clustering techniques.

Additional illustrations of clustering techniques are available in subsubsection A.7.3.2, featuring Figure 132 and Figure 133.

3.4.2.2 Other VAE-specific Approaches

This section explores **VAE**-specific approaches that can be leveraged to enhance the model's performance. The focus will be particularly on the latent space components, such as z_γ, z_μ and the latent vector itself, z.

After training, the test dataset was passed through the model to obtain the latent space parameters. The histograms of these parameters are shown in Figure 44 where each image is represented by a latent parameter vector, which is then averaged and normalized to obtain a single point on the histogram. The plots reveal a significant overlap between all classes of data, making differentiation challenging. To quantify this, an **AUC** of 0.62, 0.27, and 0.52 was computed for z_μ, z_γ, and z, respectively. Note that since these metrics weren't build for classification, thus not taught if an OK/NOK image must be classified as 0 or 1, it can be inferred that a true **AUC** of $1 - 0.27 = 0.73$ can be achieved for z_γ, which is still not in par with the results obtained from the previous section but competitive.

Even though this approach may seem flimsy, it can be useful for production purposes if the size of the data used for training, validation, and testing meets the confidence requirements to certify the model at the industrial scale.

A 2D **PCA** projection, from the transformation learned from the validation dataset, of the latent parameters is shown in Figure 45. The results show that there are no clear separation between the OK and NOKs classes. To quantify how much classification power can be extracted, a **SVM** was fitted on the validation dataset and tested on this data. Unfortunately it didn't generalized well, achieving an **AUC** of approximately random chance (0.5) on all latent space parameters.

Let the reader know that additional validation specific results for this section are made available in subsection A.7.4.

3.4.2.3 Best Performance Achieved

In the previous section, it was noted that the post-processing pipeline developed for this master thesis did not yield the best results compared to the non-variational models studied. Specifically, the best scores achieved were an **AUC** of 0.71 using the **BCE** metric (Table 30) and an **IoU** of 0.21 using the **MSE** loss with the application of the dilation filter. Additionally, if considering the findings on subsubsection 3.4.2.2, an best new score of 0.73 **AUC** was obtained but its generalization capabilities are yet to be apurated.

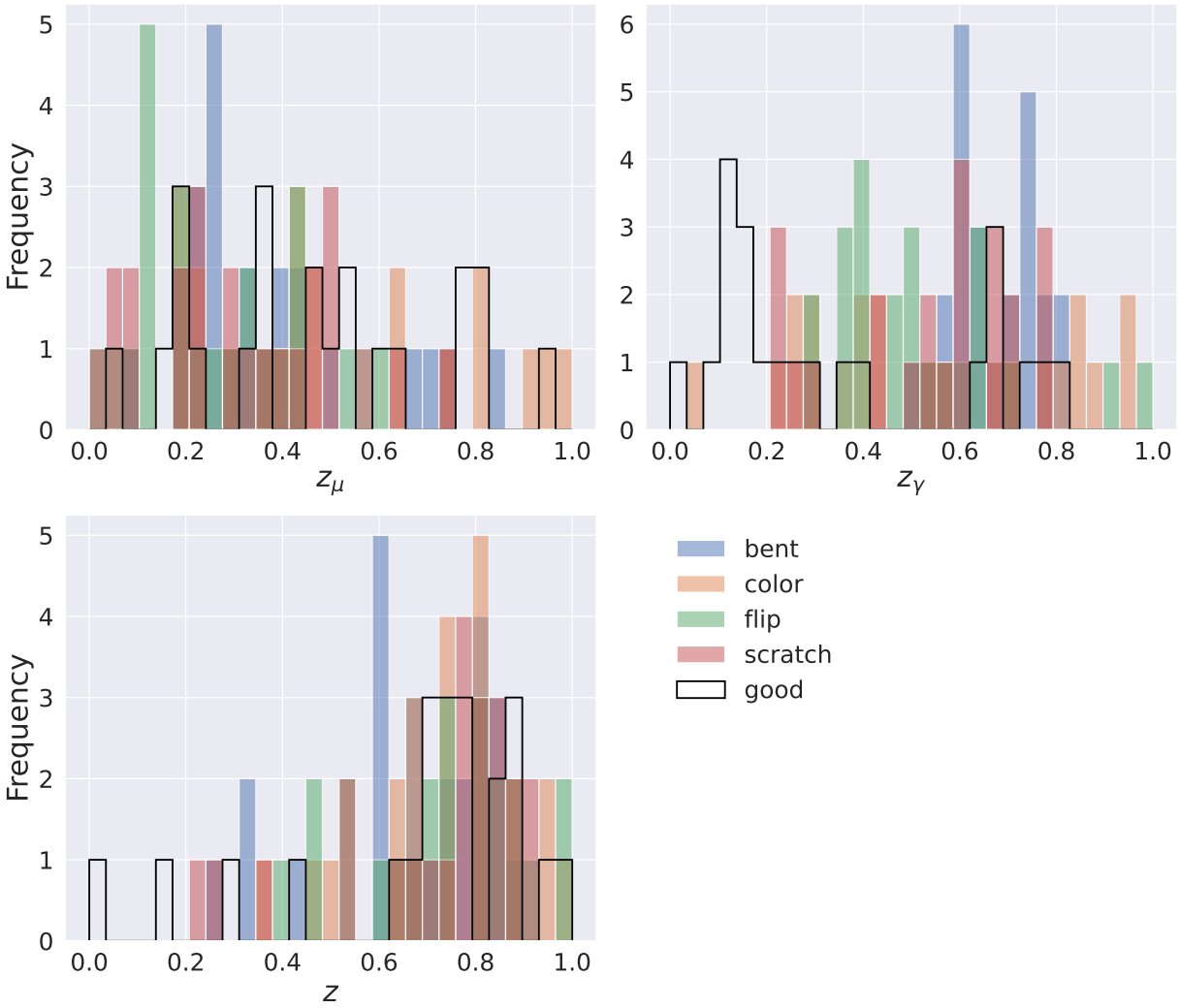


Figure 44: Histograms of the latent space parameters for the test dataset.

Moreover, given that the **BCE** metric showed greater classification power, the post-processing pipeline was also applied to it to determine if any improvement in performance could be achieved.

The Table 31 illustrates the impact of various filtering techniques on performance metrics. The highest **AUC** was achieved using the **BCE** metric with the "Eroded" technique, reaching approximately 0.76. The highest **IoU** was obtained using the **MSE** metric with the "Dilation" technique, with an **IoU** of approximately 0.21. This improves the previous approach classification power by 0.05 **AUC** and 0.1 **IoU**.

The impact of different clustering techniques on performance metrics is detailed in Table 32. The highest **AUC**, 0.53, was achieved using the **BCE** metric with the "Eroded" technique. Meanwhile, the highest **IoU**, 0.12, was achieved using the **MSE** metric with the "Eroded" technique. These results indicate that in this case using clustering may not be beneficial to this

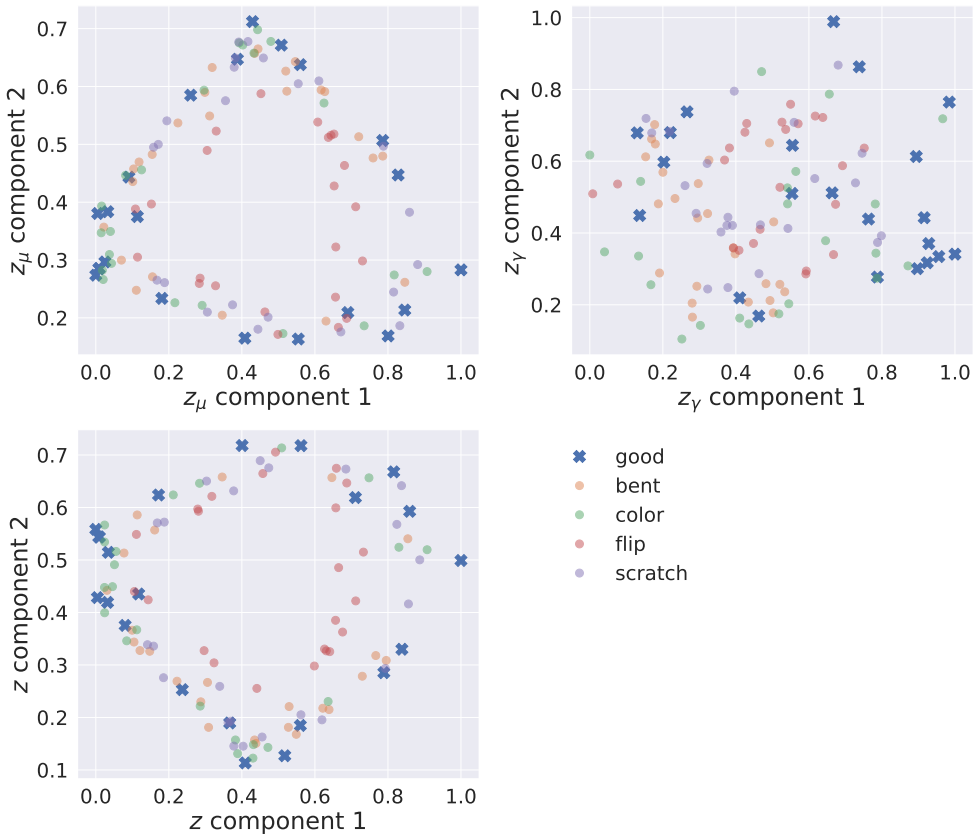


Figure 45: 2D **PCA** projection of the latent space for the test dataset, fitted on the validation dataset.

Metric	AUC	IoU
MSE	0.25	**0.11**
BCE	**0.71**	0.02

Table 30: Comparison of **AUC** and **IoU** for **MSE** and BCE

architecture due to the pour results achieved, in comparison to the previous approach.

In summary, the best performance was achieved using the filtering techniques, resulting in an **AUC** of 0.76 (-0.11 than the previous model) and an **IoU** of 0.21 (-0.16 than the previous model).

Metric	Value	Technique Used
MSE	0.51 **AUC**	Edges
	0.21 **IoU**	Dilation
BCE	**0.76** **AUC**	Eroded
	0.09 **IoU**	Dilation

Table 31: Comparison of the best **AUC** and **IoU**, discriminated by metric, and which filtering technique was used to obtain it.

Metric	Value	Technique Used
MSE	0.51 **AUC**	Edges
	0.12 **IoU**	Eroded
BCE	**0.53** **AUC**	Eroded
	0.05 **IoU**	Dilation

Table 32: Comparison of the best **AUC** and **IoU**, discriminated by metric, and which clustering technique was used to obtain it.

Chapter 4

Industrial System

This section discusses the deployment of an proof-of-concept industrial application that utilizes an API designed for real-time anomaly detection in manufacturing processes. The application centrally integrates the **CAE** architecture (section 3.3) chosen due to its demonstrated superior performance.

This centralized system enables multiple manufacturing lines, producing different components, to easily generate customized models by simply providing samples labeled as OK (and optionally NOK for test statistics after training). The system manages all subsequent processes asynchronously, from training to deployment, and offers real-time monitoring and alerting capabilities.

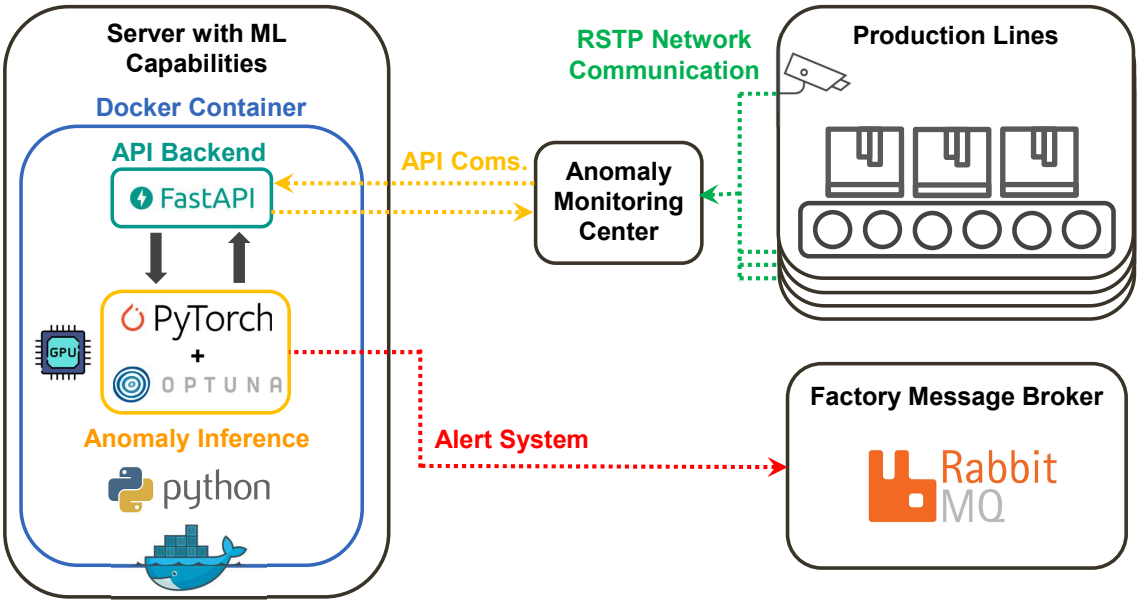


Figure 46: Birds-eye view of the system architecture for the **ML** backend.

A birds eye view of the system architecture is shown in Figure 46, which illustrates an deployment scenario where live camera feeds from multiple manufacturing lines are processed by

a centralized anomaly detection center, serving as the quality control hub for the entire factory. For each new part produced, the center communicates with a server equipped with **ML** capabilities through the developed API. This server runs the training pipeline and **HP** optimization process. After training, for every part emerging from the manufacturing line, the center sends the image to the server and receives a real-time prediction on whether the part is OK or NOK. Additional functionalities, such as the alert system depicted, can be integrated to automatically notify through the factory's message broker (e.g. RabbitMQ) if produced parts are being repeatedly detected as NOK on a single line, signaling that the manufacturing line may require maintenance.

This system offers a significant advantage over the industry's **SOTA**. Typically, software provided by equipment manufacturers demands extensive manual configuration and iterative fine-tuning, which can take weeks to finalize—tasks like defining areas of interest, setting thresholds, and adjusting for variations. In contrast, this system is designed to be intrinsically adaptable, provided there are sufficient OK samples and diversity. This capability makes it essentially self-configurable, requiring minimal human intervention, no maintenance required, and drastically simplifying the deployment process.

4.1 API Architecture

The architecture of the system comprises a **ML** backend implemented using the FastAPI framework Ramírez (2018). The backend operates the deployment and inference of anomaly detection models through two primary endpoints .

The first endpoint, `/api/train`, begins by checking if a model is already trained. If not, it launches the training pipeline which starts with fetching a trial from from Optuna. It then trains the specified architecture on the OK data samples, saving the weights of the model that performs best on the validation dataset. This process is repeated until the maximum number of trials, as specified in the API configurations, is reached.

The second endpoint, `/api/inference`, is tailored for real-time inference. It initiates by loading the model weights from the corresponding training session and computes the loss normalization and classification threshold if these are not already established. For each image received, the endpoint calculates an anomaly score and classifies the image as OK or NOK based on the threshold. This endpoint also features an alert mechanism that triggers if

the number of consecutive NOK predictions surpasses a predefined threshold.
Both endpoints require an input parameter "Dataset", which specifies the dataset to be used for training and inference. The dataset is identified by a string representing any of the datasets available in the system, stored on the backend filesystem. Users have the capability to add or remove datasets as needed via an docker bind mount. Additionally, after training, the backend provides the ROC curve, the AUC score, and histogram of the reconstruction losses for OK and NOK samples for the best model on the test dataset.

4.1.1 Endpoint: `/api/train`

The `/api/train` endpoint is dedicated to initiating the training process for **ML** models. It is designed to handle the training pipeline, including **HPO**, model training, and evaluation. The endpoint also provides updates on the training progress and status.

- **Input:**
  - `directory: str` - This parameter requires the user to specify the directory name of the dataset. This requires that the dataset is inside the datasets folder as specified in the API configurations (see subsection 4.1.4).
- **Output:**
  - `status: str` - Provides real-time updates on the status of the training process. The status can be one of the following: "started", "ongoing", "completed", or "failed", each indicating the current phase or outcome of the training operation.
  - `progression: int` - Delivers a numerical value between 0 and 100, representing the percentage completion of the training process, allowing users to monitor the progress.
  - `message: str` (optional) - Offers additional information about the training session, such as details of any encountered issues or anomalies (e.g. Model already trained, no dataset found, etc.).

4.1.2 Endpoint: `/api/inference`

The `/api/inference` endpoint facilitates the inference of new samples by applying the trained model to detect anomalies. It is designed to handle requests quickly and efficiently, provid-

ing feedback on the classification of new data samples.

- **Input:**
  - `directory: str` - This parameter requires the user to specify the directory name of the dataset. This requires the dataset to be inside the datasets folder as specified in the API configurations (subsection 4.1.4).
  - `image_base64: str` - A base64 encoded string representing the image that will be inferred.
- **Output:**
  - `status: str` - Indicates the result of the inference operation. "ok" for successful inference or "error" in case of a problem.
  - `prediction: str` (optional) - Contains the outcome of the analysis in terms of the model's prediction ("OK" or "NOK").
  - `prediction_value: float` (optional) - Shows the confidence score of the prediction, helping to gauge the reliability of the results. If separation is obtained its a number between 0 and 1 (the loss is normalized), otherwise it is the raw loss.
  - `thr: float` (optional) - A threshold value that is used to decide the cut-off point for classifying predictions. If separation between the OK and NOK classes is achieved using the test dataset, the threshold is defined as the largest OK loss value, otherwise its defined as the lowest obtained NOK loss value.
  - `sep: bool` (optional) - Indicates whether the model has achieved separation between the OK and NOK classes on the test dataset, providing a clear indication of the model's performance.
  - `reconstructed_image: list` (optional) - If configured as enabled (See subsection 4.1.4), this field will contain the base64 encoded string of the reconstructed image.

4.1.3 Example messages

The functionality of the described API endpoints is exemplified in the following table, which aligns inputs and outputs with the respective API calls, providing a reference for expected message formats and contents.

Endpoint	Payload	Output
/api/train	{"directory": "metal_nut"}	{"status": "ongoing", "progression": 70}
/api/inference	{"directory": "metal_nut", "image_base64": "encoded_string"}	{"status": "ok", "prediction": "NOK", "prediction_value": 0.8, "thr": 0.6, "sep": true}

Table 33: Example messages for the API endpoints, illustrating the input payload and output response for each endpoint.

4.1.4 API Configuration

The API configuration is specified in a JSON file, referred to as `config.json`, which sets various operational parameters for the system. Below is a detailed explanation of each parameter included in the configuration:

- `"data_directory"`: The path to the directory that contains the data used by the API, e.g. `"/path/to/data/folder"`. This directory should store the datasets necessary for model training and inference processes.

  Each dataset in this directory is organized into folders representing different data splits and classes. Specifically, there should be three primary folders named "train", "val", and "test". Each of these folders must contain a subfolder named "good", which includes all the OK samples. For NOK samples, the "test" folder should have separate subfolders for each type of anomaly, named after the anomaly type. Below is a schematic of the expected directory structure:

```
my_dataset
    +-- test
    |   +-- good
    |   +-- NOK_Type_1
    |   +-- NOK_Type_2
```

```
+-- val
|   +-- good
+-- train
    +-- good
```

Note that during the training pipeline, data-augmented NOKs will automatically appear under the validation folder. Furthermore inside each of these folders there should be only JPEG images with the same resolution.

- `"models_directory"`: Specifies the relative path where the trained models are stored, e.g. `"api/models"`. This setting allows the API to correctly locate and utilize the **ML** models.
- `"max_optuna_trials"`: Determines the maximum number of trials for optimizing model parameters, capped at 1000 by default.
- `"model_type"`: Defines the type of model to be used, which in this case is set to `"improved"` by default. Other options include "traditional", "CAE", and "VAE" which are the models studied in chapter 3.
- `"gpu_id"`: Identifies the GPU to be used by the API, indexed at `0` by default.
- `"half_res"`: A boolean flag, set to `false` by default, indicating whether the resolution of the processed images should be reduced by half. This is usefull when high resolution is not needed to identify anomalies or computational resources are limited.
- `"output_reconstructed_image"`: Another boolean parameter, also set to `false` by default, which controls whether the API should output images reconstructed during the inference process.
- RabbitMQ Configuration: The configuration file includes several keys for integrating a RabbitMQ message broker with the API: `use_rabbit`, `rabbit_server_ip`, `rabbit_port`, `rabbit_virtual_host`, and `rabbit_queue_name`. The `use_rabbit` key, a boolean, enables RabbitMQ alerting when set to `true`. The other keys specify the server's IP address, port, virtual host, and queue name for connection.

4.2 Repository Overview

The code repository for this project is named **VIGIL**, standing for *Vision-based Intelligent Gist for Industrial Learning* (logo on Figure 47). *Vigil* encapsulates all the necessary development and configuration to deploy a bespoke-made API for real-time anomaly detection in manufacturing processes. By remaining **VIGIL**(ant) – constantly vigilant to the quality of the products – the system guarantees that only the highest quality products are delivered to customers.



Figure 47: The logo of the VIGIL repository, emblematic of a commitment to advanced and intelligent manufacturing monitoring.

The repository contains critical proprietary information and code for the deployment and operation of the anomaly detection system in live manufacturing settings. Due to this, and honoring confidentiality agreements with industrial partners, the **VIGIL** repository remains inaccessible to the public.

4.3 Real Time Implementation

The feasibility of real-time implementation was evaluated by measuring the inference time of the model on a GeForce RTX 3080 12GB. The model processed images at $10ms$ each, achieving a rate of nearly 100 images per second. It's important to highlight that this measurement did not account for any potential network communication delays and the current API is configured to handle one image per call. Future improvements could involve implementing buffers to support batched API calls, depending on the specific needs of the solution and the existing infrastructure.

This performance assessment was based on the Improved **CAE** architecture. While processing 100 images per second might seem limited for some industries, the scalability can be easily achieved. Deploying additional instances on separate GPUs within the same computer can expand capacity, representing a scalable solution with fixed costs.

Chapter 5

Conclusion

This thesis elaborates on efforts to develop a comprehensive solution for reliable industrial anomaly detection using **AE**s. The research involved various approaches, each improving and iterating on the previous one. Initially, a traditional **AE** was deployed, optimized specifically to minimize validation loss, which as expected managed suboptimal results (an **AUC** of 0.71 and an **IoU** of 0.20). The following study progressed to refine the architecture based on a custom objective metric, which aimed to maximize the separation on the validation dataset between OK and data augmented NOK distributions, and with post-processing techniques. This approach significantly improved the **AUC** to 0.82 and the **IoU** to 0.32, producing very competitive results to handcrafted methods in the literature. Further refinements involved more stringent **HP** search limits and the introduction of sophisticated synthetic NOKs using diffusion methods, which led to an **AUC** of 0.87 and an **IoU** of 0.37, outperforming the literature in the same regime. Finally, the thesis explored incorporating a **VAE**. Tested under conditions similar to the prior study, an **AUC** of 0.76 and an **IoU** off 0.21 was achieved. Despite the **VAE**'s theoretical advantage of learning a richer representation of the data distribution to enhance generalization, its performance unexpectedly lagged behind the previous model.
Compared to the **SOTA**, the results in this thesis surpass the existing benchmarks in the **L2 AE** regime, which reported an **AUC** of 0.86 (-1.2%) and an **IoU** of 0.26 (-42%). This general approach not only outperforms handcrafted methods in the literature but also provides competitive results in classification power compared to the best **SOTA** regime, which achieved an **AUC** of 0.89 ($+2\%$), and surpasses the best regime in **IoU**. These results validate the novel methodology employed in this work to address the **AD** problem, centering on an **AE**-based **HP** optimization that leverages synthetic NOKs and custom objective metrics to refine the architecture.
The enhancements demonstrated above are primarily due to the post-processing techniques

and the application of the **BCE** metric during evaluation. While filtering and clustering methods generally improved results compared to those achieved with the **MSE** metric, the **BCE** metric did not always lead to improvements, as seen in the **IoU** scores for both the **CAE** and improved **CAE**, and the **AUC** for the **VAE**. Filtering techniques significantly outperformed others, excelling in both classification and interpretability metrics. Notably, Erosion, Dilation, and **NLM** Gray filters were especially effective. While clustering techniques did improve performance, they did not surpass the filtering methods quantitatively. However, despite not excelling in quantitative metrics, clustering provides valuable interpretative insights for **QC** personnel and factory line workers.

A notable observation from the study was the superior classification performance achieved using the **BCE** metric post-training. Although the **BCE** metric was included in the **HP** range for optimization in the last two models, it was notably absent from the best **HP** set selected. Despite its potential as a loss function in the Improved **CAE** and **VAE** models, the optimizer consistently bypassed it. The underlying reasons for this preference are not immediately clear and warrant further exploration, but the suspicion is that on the beginning of the training the model could output values that when inserted in the **BCE** loss function could fall outside the range of the function, leading PyTorch to clamp the output and suboptimal training.

The final phase focused on developing a plug-and-play solution, easily deployable in factory settings with our industrial partner. This system, which integrates the researched methodology through an API, is designed for minimal configuration, allowing for smooth integration into the existing **QC** monitoring systems of the factory. Internally benchmarked by Bosch Car Multimedia S.A., the system demonstrated competitive performance against other commercial solutions. Specific details remain confidential due to a non-disclosure agreement.

Finally, the main innovation of this thesis lies in the generalizability of the proposed solution. Approaches in the literature often involve specific, hand-crafted models tailored to particular applications. Contrastingly, this work employs **AE**s, a proven approach in various domains for **AD**. This study thus avoids the specialized architectural decisions and **HP** settings typical to the literature. Instead, it focuses on creating a general solution with a highly adaptable methodology, theoretically applicable to any image, industrial part or visual defect, given the computational power required.

5.1 Prospect for future work

Due to the high amount of possible approaches to this problem, one approach for possible avenues could be simply expand the search. Expand the variety and ranges of parameters to discover new possibly better configurations, variety of filtering and clustering techniques to improve classification power and interpretability, data augmentation approaches for an more reliable objective metric and so on. Unfortunately all of these make the already big **HP** search even larger, exploding in computational cost and time. With this in mind, some more educated approaches can be considered.

Throughout this work, the **BCE** metric demonstrated superior classification power compared to other metrics. However, it was surprising to find that when the optimizer had the option to use **BCE** as a loss function, it underperformed, and none of the optimal **HP** sets selected included it. This discrepancy suggests either a flawed implementation or an inherent limitation in the methodology. Future research should further investigate the **BCE** metric, particularly its behavior as a loss function. A useful approach could be to investigate its performance with a sanitized model output to determine if this enhances the loss function's effectiveness, due to **BCE** domain restrictions. Additionally, exploring other loss functions such as **SSIM**, which represents the best regime on **SOTA** for classification power, could be beneficial.

In the industry, certification of **QC** systems requires a dataset containing both OK and NOK data, along with an algorithm capable of accurately distinguishing between the two within tight margins. Typically, operators manually destroy OK parts to generate NOK data for certification, a process that is time-consuming, costly, and wasteful. Additionally, the manually created NOK parts often do not accurately represent real defects produced during machining. Therefore, exploring more sophisticated and reliable data augmentation methods to generate NOK data presents a promising avenue for future research. This approach has the potential to enhance the certification process, elevate industry standards, and contribute to environmental sustainability.

Data augmentation techniques can also help our methodology to become more generalizable. For the course of this work some practical problems have been ignored. For example, different components from the same produced part might change suppliers and that can affect characteristics like color and texture, even if the function remains unchanged. Another aspect was assuming lighting conditions were ideal and constant, which holds true in some

sectors but may not always be the case across all industries. So in order to improve robustness, data augmentation can be used to teach the architecture by adding these variations in the training data.

In some instances, the realistic NOK data generated through data augmentation techniques was so convincingly portrayed that qualitively it could not be distinghished from OK parts. This realism is likely due to the masks covering only half or one-fourth of the image and the high symmetry of this use case, allowing the diffusion model to infer the masked areas with accuracy. Future improvements might include experimenting with various mask types and new strategies to address this issue, aiming to generate only images with anomalies.

During the analysis, it was observed that using **PCA** on the latent dimension data of some models indicated a clear separation between the classes. However, this separation vanished when the same transformation was applied to the test dataset, resulting in validation-trained **SVM** models that lacked classification power. This discrepancy may stem from the use of synthetically generated NOK data during validation, which might not fully capture the complexities of real-world defects. Thus a possible future avenue could be incorporating actual NOK data into the training and validation phases to enable the **SVM** model, combined with the validation-learned **PCA**, to better understand NOK features and more effectively distinguish between OK and NOKs. However, integrating this approach into the industry could poses practical challenges due to the lack of such datasets.

Given the fast pace of advancements in deep learning, future research might also explore benchmarking multimodal **SOTA** models like GPT-4o and Claude 3.5 Sonnet. Their multimodal capabilities could offer zero-shot, zero-configuration solutions for competitive **QC** with the appropriate prompts. Creating an even faster deployment solution, if found reliable enough.

The full pipeline can be trained and operational on a modern GPU within a few days. However, this timeline may not be practical for fast-paced industries with diverse product lines and frequent changes in production. To address this, further research could optimize training and parameter tuning to shorten the model training time. This might involve analyzing the optimal number of trials and epochs needed to achieve peak performance consistently, across a variety of datasets.

On an technical level, the system could be improved to support multi-GPU training and multi-node training. This would allow for faster training times (and thus allow faster changes in manufactoring) and the ability to train on larger datasets and images (e.g. high resolution

images). Additionally, the system could support inference on multiple GPUs and batch inference on each individual GPU, allowing faster inference times, thus increasing the throughput of requests. Furthermore a small bug needs to be addressed during pipeline training. In some cases, the program crashes due to out-of-memory kill signals from the operating system, requiring a manual restart. A temporary workaround involves continuously monitoring the training progress via the API from the user side, but this is not an ideal solution.
In certain cases, production line workers may identify specific anomalies that are not flagged by the current system. Future efforts could focus on developing a procedure to improve the model to detect such anomalies. In other words, the goal would be to make the model reconstruct these specific images poorly, as there is currently no method in place to achieve this.

Bibliography

Amit Adam, Ehud Rivlin, Ilan Shimshoni, and Daviv Reinitz. Robust real-time unusual event detection using multiple fixed-location monitors. *IEEE transactions on pattern analysis and machine intelligence*, 30(3):555–560, 2008.

Altaf Ahmed, Jia Guozhu, Abdus Salam, and Muhammad Kashif Nawaz. Reliability and quality control approach for collaborative assembly process. In *2019 16th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*, pages 210–217. IEEE, 2019.

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

Yogesh Balaji, Seungjun Nah, Xun Huang, Arash Vahdat, Jiaming Song, Qinsheng Zhang, Karsten Kreis, Miika Aittala, Timo Aila, Samuli Laine, et al. ediff-i: Text-to-image diffusion models with an ensemble of expert denoisers. *arXiv preprint arXiv:2211.01324*, 2022.

Vasilis Belis, Patrick Odagiu, and Thea Klaeboe Aarrestad. Machine learning for anomaly detection in particle physics. *Reviews in Physics*, page 100091, 2024.

Paul Bergmann, Michael Fauser, David Sattlegger, and Carsten Steger. Mvtec ad–a comprehensive real-world dataset for unsupervised anomaly detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9592–9600, 2019.

Paul Bergmann, Michael Fauser, David Sattlegger, and Carsten Steger. Uninformed students: Student-teacher anomaly detection with discriminative latent embeddings. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4183–4192, 2020.

Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

Gary Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

Antoni Buades, Bartomeu Coll, and Jean-Michel Morel. Non-Local Means Denoising. *Image Processing On Line*, 1:208–212, 2011. https://doi.org/10.5201/ipol.2011.bcm_nlm.

John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (6):679–698, 1986.

Raghavendra Chalapathy and Sanjay Chawla. Deep learning for anomaly detection: A survey. *arXiv preprint arXiv:1901.03407*, 2019.

Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58, 2009.

Miguel Crispim Romão, Nuno Filipe Castro, and Rute Pedro. Finding new physics without learning about it: anomaly detection as a tool for searches at colliders. *The European Physical Journal C*, 81(1):27, 2021.

Yajie Cui, Zhaoxiang Liu, and Shiguo Lian. A survey on unsupervised visual industrial anomaly detection algorithms. *arXiv e-prints*, pages arXiv–2204, 2022.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016.

William Falcon and The PyTorch Lightning team. PyTorch Lightning, March 2019. URL https://github.com/Lightning-AI/lightning.

Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*. " O'Reilly Media, Inc.", 2022.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

Jan Hosang, Rodrigo Benenson, and Bernt Schiele. Learning non-maximum suppression. *arXiv preprint arXiv:1705.02950*, 2017.

Elvis Hozdić. Smart factory for industry 4.0: A review. *International Journal of Modern Manufacturing Technologies*, 7(1):28–35, 2015.

Hairulliza Mohamad Judi, Devendran Genasan, and Ruzzakiah Jenal. *Quality control implementation in manufacturing companies: motivating factors and challenges*. INTECH Open Access Publisher, 2011.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Iker Laskurain-Iturbe, Germán Arana-Landín, Beñat Landeta-Manzano, and Naiara Uriarte-Gallastegi. Exploring the influence of industry 4.0 technologies on the circular economy. *Journal of Cleaner Production*, 321:128944, 2021.

Yusha Liu, Chun-Liang Li, and Barnabás Póczos. Classifier two sample test for video anomaly detections. In *BMVC*, page 71, 2018.

Jiaqi Lu, Benjamin CM Fung, and William K Cheung. Embedding for anomaly detection on health insurance claims. In *2020 IEEE 7th international conference on data science and advanced analytics (DSAA)*, pages 459–468. IEEE, 2020.

Chenlin Meng, Yutong He, Yang Song, Jiaming Song, Jiajun Wu, Jun-Yan Zhu, and Stefano Ermon. Sdedit: Guided image synthesis and editing with stochastic differential equations. *arXiv preprint arXiv:2108.01073*, 2021.

Kevin Nagorny, Pedro Lima-Monteiro, Jose Barata, and Armando Walter Colombo. Big data analysis in smart manufacturing: A review. *International Journal of Communications, Network and System Sciences*, 10(3):31–58, 2017.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318. Pmlr, 2013.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative

style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Miguel Caçador Peixoto, Nuno Filipe Castro, Miguel Crispim Romão, Maria Gabriela Jordão Oliveira, and Inês Ochoa. Fitting a collider in a quantum computer: tackling the challenges of quantum machine learning for big datasets. *Frontiers in Artificial Intelligence*, 6:1268852, 2023.

Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. Sdxl: Improving latent diffusion models for high-resolution image synthesis. *arXiv preprint arXiv:2307.01952*, 2023.

Tahereh Pourhabibi, Kok-Leong Ong, Booi H Kam, and Yee Ling Boo. Fraud detection: A systematic literature review of graph-based anomaly detection approaches. *Decision Support Systems*, 133:113303, 2020.

Aaditya Ramdas, Nicolás García Trillos, and Marco Cuturi. On wasserstein two-sample testing and related families of nonparametric tests. *Entropy*, 19(2):47, 2017.

Sebastián Ramírez. Fastapi: Fast, modern, web framework for building apis with python 3.7+ based on standard python type hints, 2018. URL https://fastapi.tiangolo.com/. Accessed: [insert the date when you last accessed the website].

Manassés Ribeiro, André Eugênio Lazzaretti, and Heitor Silvério Lopes. A study of deep convolutional auto-encoders for anomaly detection in videos. *Pattern Recognition Letters*, 105: 13–22, 2018.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.

Jamal Saeedi and Alessandro Giusti. Semi-supervised visual anomaly detection based on convolutional autoencoder and transfer learning. *Machine Learning with Applications*, 11: 100451, 2023.

Thomas Schlegl, Philipp Seeböck, Sebastian M Waldstein, Ursula Schmidt-Erfurth, and Georg Langs. Unsupervised anomaly detection with generative adversarial networks to guide

marker discovery. In *International conference on information processing in medical imaging*, pages 146–157. Springer, 2017.

Julian Schuhmacher, Laura Boggia, Vasilis Belis, Ema Puljak, Michele Grossi, Maurizio Pierini, Sofia Vallecorsa, Francesco Tacchino, Panagiotis Barkoutsos, and Ivano Tavernelli. Unravelling physics beyond the standard model with classical and quantum anomaly detection. *Machine Learning: Science and Technology*, 4(4):045031, 2023.

Philipp Seeböck, Sebastian Waldstein, Sophie Klimscha, Bianca S Gerendas, René Donner, Thomas Schlegl, Ursula Schmidt-Erfurth, and Georg Langs. Identifying and categorizing anomalies in retinal imaging data. *arXiv preprint arXiv:1612.00686*, 2016.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

StabilityAI. Stable diffusion xl inpainting 0.1. https://huggingface.co/diffusers/stable-diffusion-xl-1.0-inpainting-0.1, 2023. Model card.

Waqas Sultani, Chen Chen, and Mubarak Shah. Real-world anomaly detection in surveillance videos. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6479–6488, 2018.

Usman Ahmad Usmani, Ari Happonen, and Junzo Watada. A review of unsupervised machine learning frameworks for anomaly detection in industrial applications. In *Science and Information Conference*, pages 158–189. Springer, 2022.

Juan Vanerio and Pedro Casas. Ensemble-learning approaches for network security and anomaly detection. In *Proceedings of the Workshop on Big Data Analytics and Machine Learning for Data Communication Networks*, pages 1–6, 2017.

Pauli Virtanen and el at. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

James R Weaver. Centrosymmetric (cross-symmetric) matrices, their basic properties, eigenvalues, and eigenvectors. *The American Mathematical Monthly*, 92(10):711–717, 1985.

McKinsey & Company Weltwirtschaftsforum. Global lighthouse network: Insights from the forefront of the fourth industrial revolution. In *World Economic Forum Retrieved on Febrauary*, volume 1, page 2021, 2020.

Chong Zhou and Randy C Paffenroth. Anomaly detection with robust deep autoencoders. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 665–674, 2017.

Nourelislam Zouar. *Anomaly Detection for Manufacturing Applications Using Convolutional Autoencoders*. PhD thesis, The University of North Carolina at Charlotte, 2021.

Appendix A
Appendix

A.1 Data Augmentation Details

A.1.1 Traditional Data Augmentation Techniques

Data augmentation is a method employed to artificially expand the size of a training dataset by applying transformations to the input data. The core objective is to emulate real-world imperfections in any given arbitrary manufactured component. However, simulating real-world defects accurately is challenging, particularly in a manner that is generalizable to unknown parts to be inspected. To address this, a suite of data augmentation techniques has been employed.

In the given context, traditional augmentation methods such as flipping, rotation, and cropping are less relevant because images are pre-aligned and industrial components are uniformly oriented. Consequently, the focus has shifted to new augmentation techniques that enhance model generalization to new data. These techniques aim to mimic real-world defects like subtle variances or contaminations, as detailed in Figure 48.

Furthermore, discussions with our industrial partner have revealed the lighting conditions in the production environment are consistently uniform. As such, data augmentation for making the model robust to lighting conditions is not necessary.

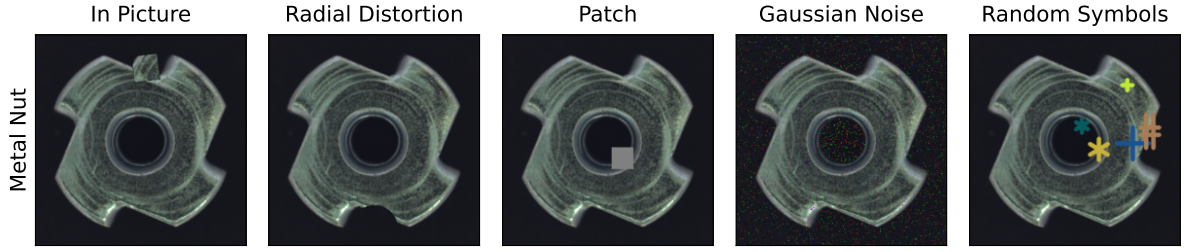


Figure 48: Traditional data augmentation techniques. Displayed are the augmentations such as picture distortion, radial distortion, patch application, gaussian noise, and the inclusion of random symbols, specifically for the metal nut dataset in the MVTeC dataset.

A variety of data augmentation techniques were explored, including:

1. **Patch Augmentation:** This method involves adding a randomly sized and positioned gray square patch to an image. The augmentation is done by first determining a valid location for the patch that ensures it is fully contained within the image and inside the region entailed by the metal nut. The size of the patch is randomly chosen within specified limits, and a gray patch is then placed at the chosen location. This augmentation is designed to simulate the presence of foreign objects on the part's surface.

2. **In-Picture Augmentation:** This technique augments an image by superimposing a rotated segment from the same image. The process begins by selecting a segment of the image, the size of which is determined randomly within given bounds proportional to the size of the image. The segment is then rotated by a random angle and reinserted into the image at a different, valid location that ensures the modified segment does not overlap with its original position. This augmentation is designed to simulate the presence of foreign objects on the part's surface or small defects on arbitrary parts of the image.

3. **Radial Distortion Augmentation:** This augmentation distorts the image in a circular pattern, centered on a randomly chosen point within the image. The degree of distortion can be adjusted, allowing for the simulation of different levels of distortion. This produces defects similar to when a part is bend of deformed.

4. **Gaussian Noise Augmentation:** This augmentation introduces Gaussian noise to an image. The intensity of the noise can be adjusted, allowing for different levels of augmentation. This process involves adding noise to each pixel of the image, where the noise is generated according to a Gaussian distribution with specified mean and standard deviation.

5. **Adding Random Symbols:** This augmentation involves adding random symbols (e.g., '*', '+', '#') to the image. The symbols are placed in various colors and positions within the image. The process includes generating a random number of symbols, randomly choosing each symbol's position, color, and size, and then drawing these symbols onto the image. This augmentation is designed to simulate the presence of foreign objects on the part's surface.

In assessing the realism and industrial applicability of the discussed data augmentation techniques, several considerations can be made. Patch Augmentation, by introducing a synthetic imperfection such as a gray square, may be applicable for mimicking anomalies in uniform surfaces but may lack realism if the simulated defect does not accurately reflect actual imperfections encountered in production environments. On the other hand, In-Picture Augmentation, which reuses parts of the image to simulate defects, provides a more realistic scenario since the textures or patterns present in the patch itself, were already part of the original image.

Radial distortion augmentation may also offer a realistic approach to simulate deformations that, in our case, might occur in metal parts due to bending or other physical stresses. However, its effectiveness is contingent on the precise calibration to the types of distortions typically observed. Lastly, the Adding Random Symbols technique, while useful for increasing robustness against unexpected visual noise, may be less realistic unless such symbols or similar disruptions are common in the specific industrial context, which could be a rare occurrence.

In short, the realism of data augmentation techniques depends on the specific industrial context and the types of defects or anomalies typically encountered. The next section will discuss generative approaches for creating more realistic NOKs.

A.1.2 Generating Realistic NOKs

In this thesis, realistic NOKs enable the development of an NOK-agnostic **AD** pipeline, enhanced by a more accurate metric than what could be provided with handcrafted NOKs. For the industry, realistic NOKs are essential when validating machine learning models for **AD**. Many industries require a diverse dataset for certification of such systems, including both OK and NOK samples. However, acquiring a sufficient number of NOK samples often involves destructive testing or defect introduction, which is costly and time-consuming. Generating realistic NOKs using advanced generative techniques thus offers a solution by creating a more representative dataset of real-world conditions without damaging actual parts.

Stable diffusion (Rombach et al. (2022)) will be used to generate realistic-looking NOKs. This image generation model employs diffusion techniques (Ho et al. (2020)) to iteratively refine pixels, ensuring detailed textures and context integrity. For this project, the comprehensive package StabilityAI (2023), which integrates technologies from Podell et al. (2023); Balaji et al.

(2022); Meng et al. (2021); Rombach et al. (2022), will be utilized. Figure 49 illustrates the process.

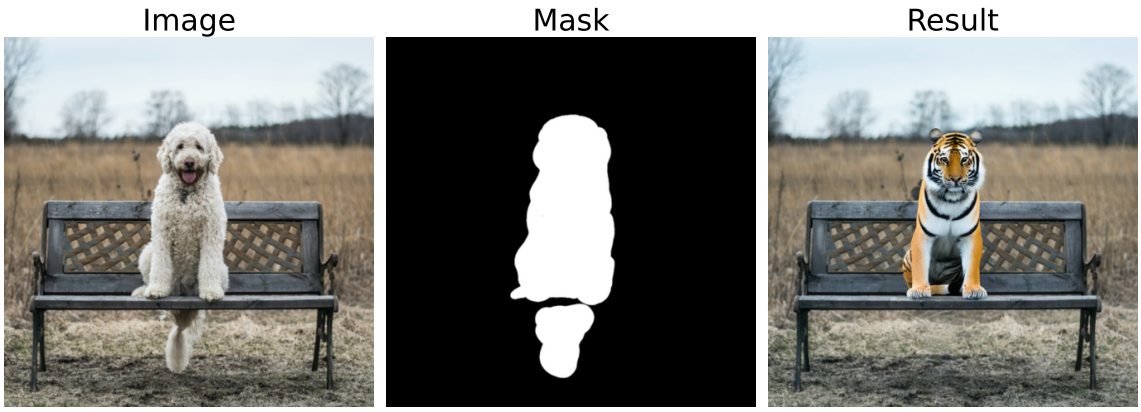


Figure 49: Exemplary process of how Stable Diffusion operates. The sequence shows the original image, the mask of the modification area, and the resulting image post-application with the prompt "a tiger sitting on a park bench".

This technique is applied to OK images, where masks are generated to delineate areas for modification, simulating potential defects. An example of this data augmentation method is demonstrated in Figure 50 with the "half mask" technique on the metal nut dataset, illustrating the transformation. This process occurs without a specific prompt,leaving the model open to synthesize plausible content from the surrounding pixels.



Figure 50: Detailed process of data augmentation using "half mask" on the metal nut dataset, displaying the original, the mask, and the augmented NOK image.

Various masks employed in our experiments are depicted in Figure 51. The random mask, while experimented with, was found to produce results that deviated significantly from realistic scenarios and was therefore not utilized in the final dataset. Furthermore the "half mask" and "quarter mask" produce masks that cover half of the image and one quarter respectively. Furthermore the side (or corner) is determined randomly.



Figure 51: Different masks tested for generating artificial NOK conditions.

To augment the dataset, "half mask" and "quarter mask" augmentation techniques were applied to the OK images, resulting in two augmented sets with the same number of images as the original. The NOKs produced are shown in Figure 52 and Figure 53, highlighting their potential for realism and variability.

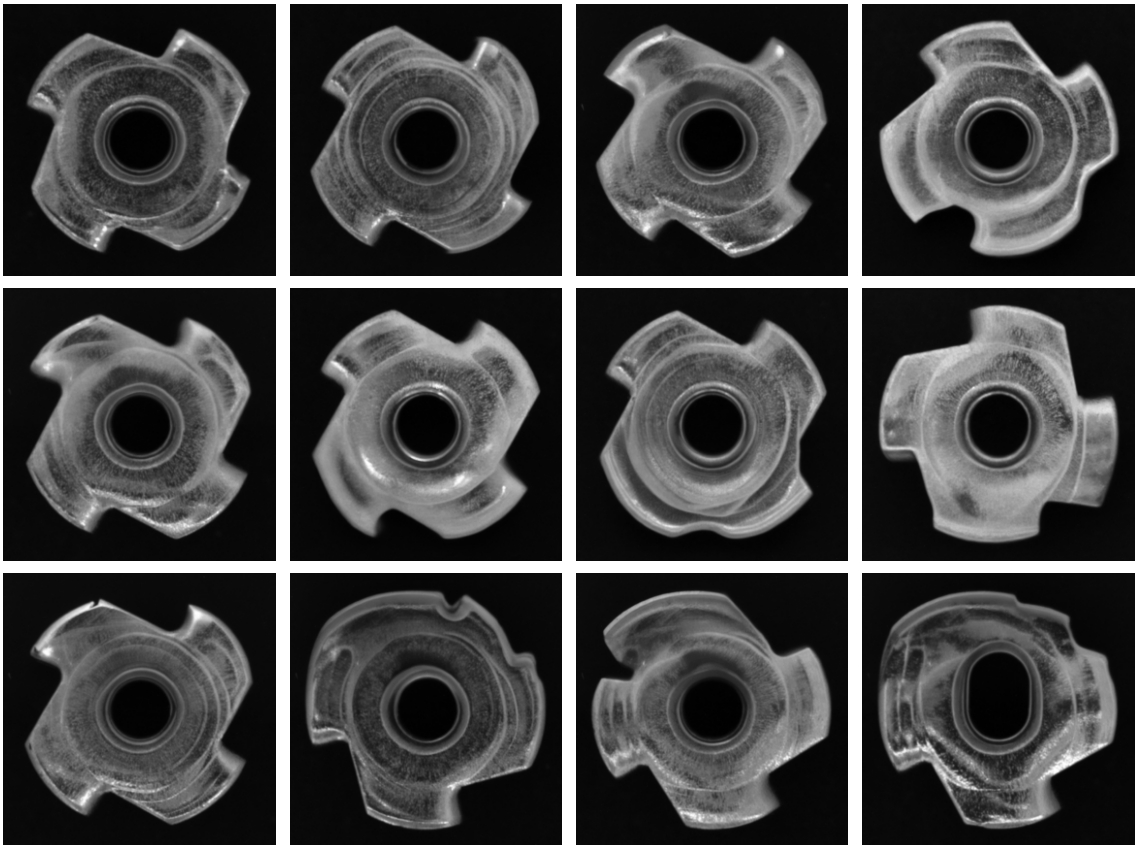


Figure 52: Examples of NOKs produced using the "half mask" technique on the metal nut dataset.

In Figure 52, some images closely resemble the original and might be mistaken for an OK image, while others display more obvious NOK characteristics, such as anomalies that re-

semble a "warp" or a "bends" in the metal nut. There are also more extreme cases where half of the metal nut is completely "warped" or "smashed". Qualitatively, the images produced generally appear realistic and could easily be mistaken for actual NOKs. However, due to the inherent symmetry of the part, some images might also convincingly resemble OKs.

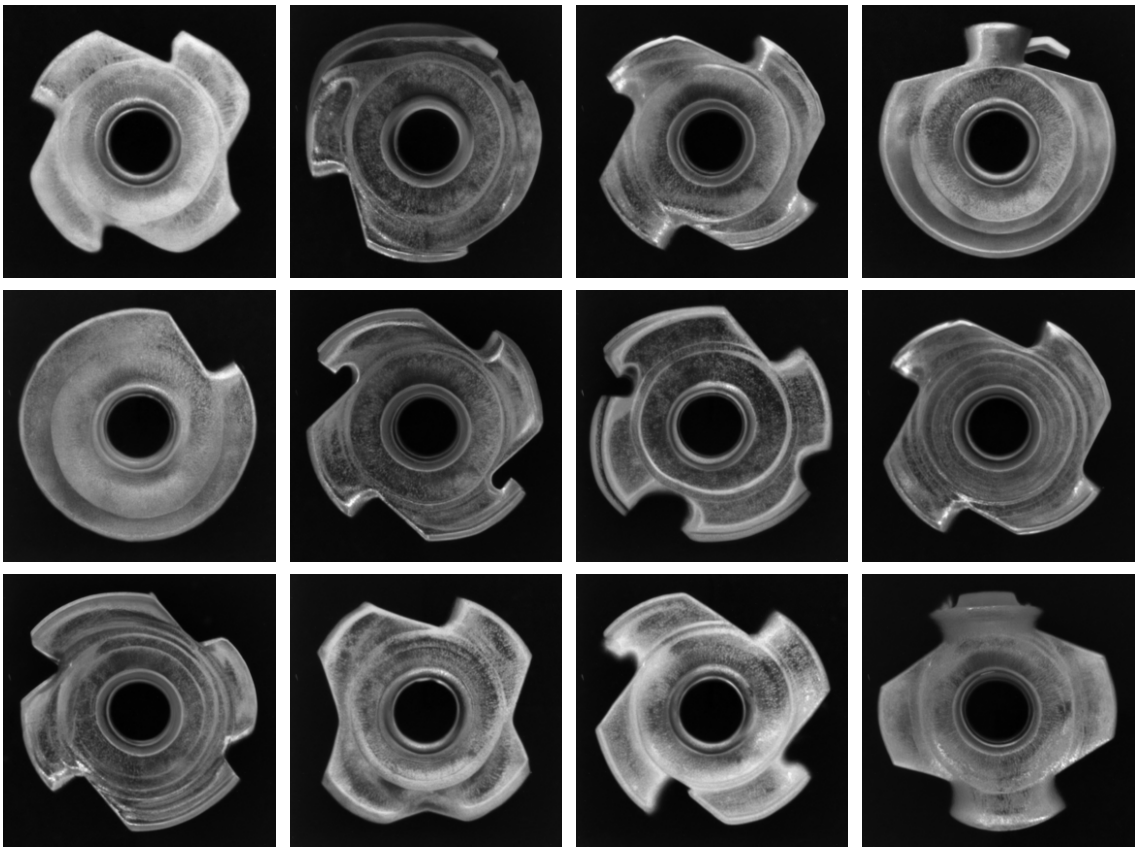


Figure 53: Examples of NOKs produced using the "quarter mask" technique on the metal nut dataset.

Conversely, in Figure 53, it is visually apparent that all images are NOKs. Given that this model works with half the information compared to the previous example, it exhibits more freedom to generate pronounced anomalies. In these images, the texture and color are mostly preserved, yet they feature larger and more distinct deformations.

While not yet perfect, this section has outlined a straightforward method for generating NOKs and provides a lot of potential to other processes affecting the industrial manufacturing process. The creation process can still be further refined and tailored to each specific industrial part. For instance, the prompt can be customized to generate commonly occurring anomalies and the mask adjusted to more prominently affect the regions typically impacted. Although there is room for further improvement, the current method will be used as is since extensive development of this technology is beyond the scope of this thesis.

A.2 Precomputations

This section examines the precomputations in the pipeline using the **AE** approach (chapter 2). It initializes with key precomputations that establish the foundation for configuring subsequent layers. These include setting global model properties like latent dimension, activation function, and the implementation of batch normalization and dropout, specifying dropout probability. A schematic of this process is presented below:

```
BEGIN Precomputations
    # Note: Suggest_x Optuna methods have the arguments:
    # name, low, high, step=None, log=False

    LATENT_DIM ← Optuna.suggest_int("LATENT_DIM", 1,
                    (IMAGE_SIZE*IMAGE_SIZE*PREVIOUS_N_CHANNELS)/10, log=True)

    ACTIVATION_FUNCTION ← Optuna.suggest_categorical("ACTIVATION_FUNCTION",
                                ["ReLU", "LeakyReLU", "Sigmoid", "Tanh", "ELU"])

    USE_BATCH_NORM ← Optuna.suggest_categorical("USE_BATCH_NORM", [True, False])

    USE_DROPOUT ← Optuna.suggest_categorical("USE_DROP_OUT", [True, False])
    IF USE_DROPOUT THEN
        DROPOUT_PROB ← Optuna.suggest_float("DROPOUT_PROB", 0.1, 0.9)
    END IF

    MAX_CONV_LAYERS ← Optuna.suggest_int("MAX_CONV_LAYERS", 1, 10)

END Precomputations
```

Note that in the implementation, `LATENT_DIM` is dynamically determined using Optuna's logarithmic scale (`log=True`), this is done to ensure a wide range of values are considered. Furthermore, the formula

```
(IMAGE_SIZE*IMAGE_SIZE*PREVIOUS_N_CHANNELS)/10
```

sets a sensible upper limit for the latent dimension, based on the input image's size and channel count. Finally, Optuna's categorical suggestion methods select the `ACTIVATION_FUNCTION`, `USE_BATCH_NORM`, and `USE_DROPOUT` and Optuna's integer suggestion selects the `MAX_CONV_LAYERS` parameter.
Furthermore, each convolutional layer, as specified by the `MAX_CONV_LAYERS` **HP**, undergoes additional precomputations. These are for ensuring the correct dimensionality in both the

convolutional and transposed convolutional layers, relative to the current IMAGE_SIZE (L_{in} of the current convolutional iteration). The PREVIOUS_N_CHANNELS parameter refers to the number of channels in the preceding convolutional layer.

The commencement of the pipeline is marked by the specification of parameters for the encoder layers, including stride, kernel size, and the number of channels. For the stride, the selection interval is established in the range $[0, 5]$. However, in scenarios where the image size is less than 5, the interval is adjusted to $[0, \frac{\text{IMAGE_SIZE}}{2}]$. Similarly, the kernel size interval is initially set to $[0, 10]$. Yet, if the image size falls below 10, this interval is modified to $[0, \text{IMAGE_SIZE}]$. These adaptations, even though arbitrary and limited by hardware, help maintain the pipeline flexibility across different image sizes.

Regarding channel numbers, the interval for the first layer is defined from 0 to 256, whereas for subsequent layers, it is determined by the range $[0, \text{PREVIOUS_N_CHANNELS}]$. This approach is designed to systematically reduce the channel count at each layer, acting as a funnel to decrease the overall dimensionality of the information step by step.

The algorithm below succinctly represents the outlined procedures:

```
BEGIN Layer-Specific Precomputations - Part 1
    # Define parameters for each convolutional layer
    STRIDE ← Optuna.suggest_int("STRIDE_" + n_conv, 0,
                min(5, IMAGE_SIZE // 2))
    KERNEL_SIZE ← Optuna.suggest_int("KERNEL_SIZE_" + n_conv, 0,
                    min(10, IMAGE_SIZE))
    N_CHANNELS ← Optuna.suggest_int("N_CHANNELS_" + n_conv, 0,
                    256 if n_conv == 0 else PREVIOUS_N_CHANNELS)
END Layer-Specific Precomputations - Part 1
```

Due to the nature of convolutional layers, the output dimensions of the current layer are calculated using the input dimensions and convolution parameters as auxiliary variables. These calculations are performed using the LOut_compute and LOutT_compute functions which correspond to those in Equation 2.1 and Equation 2.4, respectively. It's important to note that the equation in Equation 2.2 is a variation of Equation 2.1 with fixed parameters: $S = 2$, $K = 2$, and $P = 0$.

```
BEGIN Layer-Specific Precomputations - Part 2
    # Precomputations for Padding
    LOut ← LOut_compute(L_in=IMAGE_SIZE, P=0, S=STRIDE, K=KERNEL_SIZE)
    LOutT ← LOutT_compute(L_in=LOut, P=0, S=STRIDE, K=KERNEL_SIZE)

END Layer-Specific Precomputations - Part 2
```

Now given the output dimensions of the current layer, the pipeline determines the padding and output padding values for the transposed convolutional layers that will be used in the decoder. This is achieved by iteratively increasing the padding values until the output dimensions are greater than or equal to the input dimensions. The algorithm below illustrates this process:

```
BEGIN Layer-Specific Precomputations - Part 3
    P ← 0
    P_0 ← 0
    WHILE LOutT > IMAGE_SIZE DO
        P ← P + 1
        LOutT ← LOutT_compute(L_in=LOut, P=P, S=STRIDE,
                              K=KERNEL_SIZE, P_0=P_0)
    END WHILE

    WHILE LOutT < IMAGE_SIZE DO
        P_0 ← P_0 + 1
        LOutT ← LOutT_compute(L_in=LOut, P=P, S=STRIDE,
                              K=KERNEL_SIZE, P_0=P_0)
    END WHILE
END Layer-Specific Precomputations - Part 3
```

Finally, the pipeline determines whether max pooling should be used for the current layer. This decision is based on the output dimensions of the current layer. If the output dimensions are greater than or equal to 2, max pooling is enabled. Otherwise, it is disabled since, the pooling layer would reduce the output dimensions to 0. The algorithm below illustrates this process:

```
BEGIN Layer-Specific Precomputations - Part 3
    IF LOut >= 2 THEN
        USE_MAX_POOL ← Optuna.suggest_categorical("USE_MAX_POOL_" + n_conv,
            [True, False])
        LOut_Pool ←  LOut_compute(L_in=LOut, P=0, S=2, K=2)
    END IF
END Layer-Specific Precomputations - Part 3
```

A.3 Image Filtering Techniques

Image filtering techniques can be used to enhance the framework ability to detect anomalies by highlighting relevant features indicative of anomalies while reducing noise interference. This enhancement not only have the potential to improve the classification accuracy but also the interpretability of the detection results. Below, it is described various filtering techniques

applied in this study with the help of *OpenCV* library (Bradski (2000)), ordered from basic operations designed to refine geometric structure, through edge detection aimed at identifying significant transitions, to more sophisticated advanced noise reduction methods.

- **Morphological Operations:**

  Morphological operations are a set of image processing techniques that involve applying a structuring element to an input image and generating an output image based on the shapes within the image. The two most basic morphological operations are erosion and dilation. These operations have a wide range of applications, including noise removal, isolation of individual elements, joining disparate elements in an image, and detecting intensity bumps or holes.

  To showcase the morphological operations, Figure 54 will be used as an example.



Figure 54: Example of image to be used in the morphological operations Bradski (2000).

  - *Erosion* – This operation removes small-scale features from images by applying a specific structuring element to the image and retaining the minimum value within the region covered by the structuring element. Erosion computes a local minimum over the area of the given kernel. As the kernel K is scanned over the image, the minimal pixel value overlapped by K is computed, and the image pixel under the anchor point is replaced with that minimal value.

    In Figure 55 it can be seen what applying erosion to Figure 54 does. Mathematically, erosion can be described as:

$$\text{dst}(x, y) = \min_{(x', y'):\text{K}(x', y') \neq 0} \text{src}(x + x', y + y') \tag{A.1}$$



Figure 55: Erosion operation applied to Figure 54, Bradski (2000)

Where src is the original image, dst is the resulting image, and K is the structuring element used for the operation. x and y are the coordinates of the pixel being processed.

This process highlights larger, potentially anomalous structures and is effective in reducing noise by stripping away thin and isolated anomalies. As the kernel K is applied, bright areas of the image become thinner, and dark zones expand, altering the image's visual properties in a way that could be beneficial for analyzing specific features.

- *Dilation* – It enhances prominent features within images by similarly using a structuring element but retains the maximum value encountered within the region it covers. Dilation consists of convolving an image with a kernel K, which can be of any shape or size, typically a square or circle. The kernel K has a defined anchor point, usually at the center of the kernel. As the kernel K is scanned over the image, the maximal pixel value overlapped by K is computed, and the image pixel at the anchor point is replaced with this maximal value. This maximizing operation causes bright regions within an image to expand, enhancing their visibility—an effect particularly useful in environments where detecting subtle anomalies is crucial.

  The dilation of an image by a structuring element K is defined mathematically as:

$$\text{dst}(x, y) = \max_{(x',y'):\text{K}(x',y')\neq 0} \text{src}(x + x', y + y') \tag{A.2}$$

Where src is the original image, dst is the resulting image, and K is the structuring element.

As depicted in Figure 56, the bright area of a letter dilates around the black regions of the background, making the letter more prominent against its surroundings.



Figure 56: Dilation operation applied to the image Figure 54, Bradski (2000)

This can be particularly effective in enhancing the visibility of anomalies, making them more distinguishable from the background. This operation can be especially beneficial in scenarios where anomalies are subtle or embedded within complex structures, as it can help to isolate and highlight these features.

- **Canny Edge Detector** – Identifying significant intensity changes to delineate boundaries, edge detection is particularly effective in uncovering anomalies characterized by abrupt transitions in image intensity. Among the various edge detection algorithms, the canny edge detector is notably robust, offering a multi-stage process to achieve high accuracy in detecting edges. As illustrated in Figure 57, the Canny edge detector effectively traces the edges of objects within an image, providing a clear outline of the object's structure.

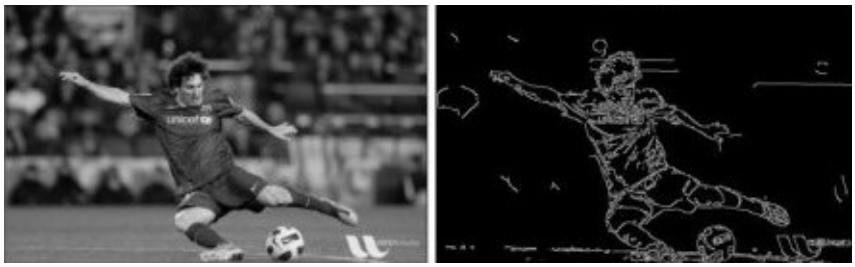


Figure 57: Canny edge detection applied to an image Bradski (2000)

The Canny edge detection algorithm (Canny (1986)), involves several key stages:

1. Noise Reduction: To prevent false edge detection, the image is first smoothed using a Gaussian filter. This step reduces the high-frequency noise components in the image.

2. Finding Intensity Gradient: The smoothed image is then processed using a Sobel kernel in both horizontal and vertical directions to find the first derivative in both orientations, G_x and G_y. From these, the edge gradient and direction can be calculated for each pixel:

$$Edge_Gradient(G) = \sqrt{G_x^2 + G_y^2} \tag{A.3}$$

3. Non-maximum Suppression: This step aims to remove spurious response to edge detection by retaining only the points that are local maxima in the direction of the gradient. Additional details can be found in Hosang et al. (2017).

4. Double Thresholding: It involves using two thresholds to detect strong and weak edges and includes the weak edges in the output only if they are connected to strong edges, reducing the likelihood of false edges while preserving genuine edge structures.

5. Edge Tracking by Hysteresis: Finally, the algorithm categorizes the edges as 'true' or 'false' based on the thresholding, ensuring edge continuity by suppressing all the other edges that are weak and not connected to strong edges.

This comprehensive approach allows the canny edge detector to accurately trace edges with low error rates, which is essential in applications where precise edge delineation is required. For our use case, the canny edge detector can be useful for highlighting the boundaries of anomalies, making them more discernible and interpretable to operators.

- **Noise Reduction Filters:**

  - *Median Filter* – Reduces noise while preserving edges by replacing the value of each pixel with the median of the intensities in its neighborhood. Mathematically, the median filter operation for a pixel located at position (x, y) can be represented as:

$$I_{\text{out}}(x, y) = \text{median}\{I_{\text{in}}(x + i, y + j) \mid (i, j) \in N\} \tag{A.4}$$

    where I_{in} is the input image, I_{out} is the output image, and N represents the neighborhood around the pixel defined by a structuring element (e.g., a square window

of size 3×3). This operation ensures that pixel values that are significantly different from their surroundings (often indicative of noise) are replaced, thus enhancing the overall image quality without affecting the sharpness of abrupt edges. An example of the median filter application is shown in Figure 58, where the original image (left) and the median filtered image (right) are displayed.

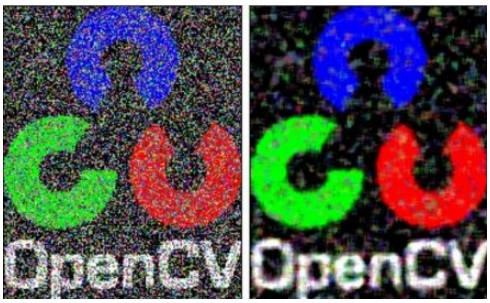


Figure 58: Median filter applied to an image Bradski (2000). Original image (left) and median filtered image (right).

- *Gaussian Blur* – Gaussian Blur applies a Gaussian function to smooth images, effectively reducing noise and detail. It is defined by the convolution of the input image I_{in} with a Gaussian kernel G. The Gaussian kernel is defined as:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{A.5}$$

where σ is the standard deviation of the distribution, which determines the extent of the blurring effect. The output image I_{out} at each pixel (x, y) is computed through the convolution operation:

$$I_{\text{out}}(x, y) = (G * I_{\text{in}})(x, y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} G(i, j) \cdot I_{\text{in}}(x - i, y - j) \tag{A.6}$$

where k is the kernel radius determined by σ. This blurring process smooths variations in image intensity and is particularly effective in removing high-frequency noise, which could be common in the reconstruction error of our application.

- ***Non-local Means Denoising (NLM)*** – **NLM** (Buades et al. (2011)) employs a sophisticated noise reduction technique that averages the color of a pixel with similar pixels across the entire image, not just within its immediate neighborhood. This



Figure 59: Gaussian blur applied to an image. Original image (left) and Gaussian blurred image (right) as illustrated in Figure 59.

global approach is particularly adept at preserving fine textures and intricate details while removing noise, thereby maintaining the high quality of the image in regions with subtle features.

The concept hinges on the repetition of similar patterns within the image. For instance, consider a small window (e.g., a 5×5 window) around a pixel, as illustrated in Figure 60. It is common to find identical or very similar patches elsewhere in the image. By averaging these similar patches, **NLM** reduces noise specifically within that window, leveraging redundancy across the image to improve denoising.

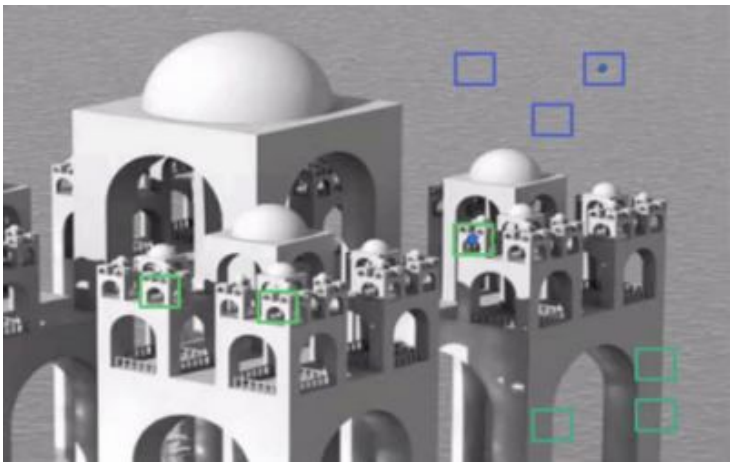


Figure 60: Illustration of **NLM**, showing how similar patches within the image are averaged to denoise a specific window. Distinct patches are highlighted to emphasize the non-local averaging approach Bradski (2000).

In practical terms, a pixel is denoised by averaging it with the pixels from similar windows found throughout the image, thus replacing the original pixel value with this average. The mathematical expression for **NLM** defines the denoised pixel value $I_{\text{NL}}(i)$ as a weighted average:

$$I_{\mathrm{NL}}(i) = \frac{1}{Z(i)} \sum_{j \in I} w(i,j) I(j) \tag{A.7}$$

where $I(j)$ denotes the intensity of pixel j, $w(i,j)$ is the weight function gauging the similarity between the pixels i and j, and $Z(i)$ serves as a normalization factor:

$$Z(i) = \sum_{j \in I} w(i,j) \tag{A.8}$$

The weighting function $w(i,j)$ is generally based on the Gaussian of the Euclidean distance between the intensities in the neighborhoods N_i and N_j of pixels i and j, respectively, and is defined by:

$$w(i,j) = \exp\left(-\frac{\|I(N_i) - I(N_j)\|^2}{h^2}\right) \tag{A.9}$$

where h is a smoothing parameter that controls the exponential decay. This non-local method significantly outperforms traditional local techniques (Buades et al. (2011)), providing superior denoising while preserving essential details.

It is important to acknowledge the extensive array of image processing techniques available for application. However, the few techniques selected for this study were chosen due to their potential in enhancing **AD** capabilities. Moreover, these selected techniques feature parameters that offer further opportunities for performance optimization. It should be noted, though, that the parameters were not exhaustively optimized within the scope of this research.

A.4 Supplementary Results - Traditional Approach

The best achieved **HP** set for the traditional approach is shown in Table 34, which achieved the best validation loss of -0.1271. Surprisingly, given the decent test results discussed on section 3.1, it has a considerably small latent dimension of just 11, which could explain the pour resolution of the reconstructions.

The Optuna library (Akiba et al. (2019)) offers a suite of visualization tools for analyzing optimization processes. From the optimization history of the **HP** (Figure 61), it is evident that

Parameter	Value
Latent Dimension	11
Activation Function	LeakyReLU
Use Batch Normalization	False
Use Dropout	False
Maximum Convolutional Layers	5
Stride #1	1
Kernel Size #1	6
Number of Channels #1	13
Use Max Pooling #1	False
Stride #2	1
Kernel Size #2	6
Number of Channels #2	7
Use Max Pooling #2	True
Stride #3	1
Kernel Size #3	6
Number of Channels #3	7
Use Max Pooling #3	True
Starting Learning Rate	0.001261066333907153

Table 34: Optimized **HP** set for the best trial.

most trials failed, all due to **GPU** memory constraints. This indicates that Optuna's optimizer quickly determined that increasing the model complexity was necessary to minimize validation loss, which consequently led to higher memory usage. As a result, all trials after the 10th successful trial failed due to memory errors.

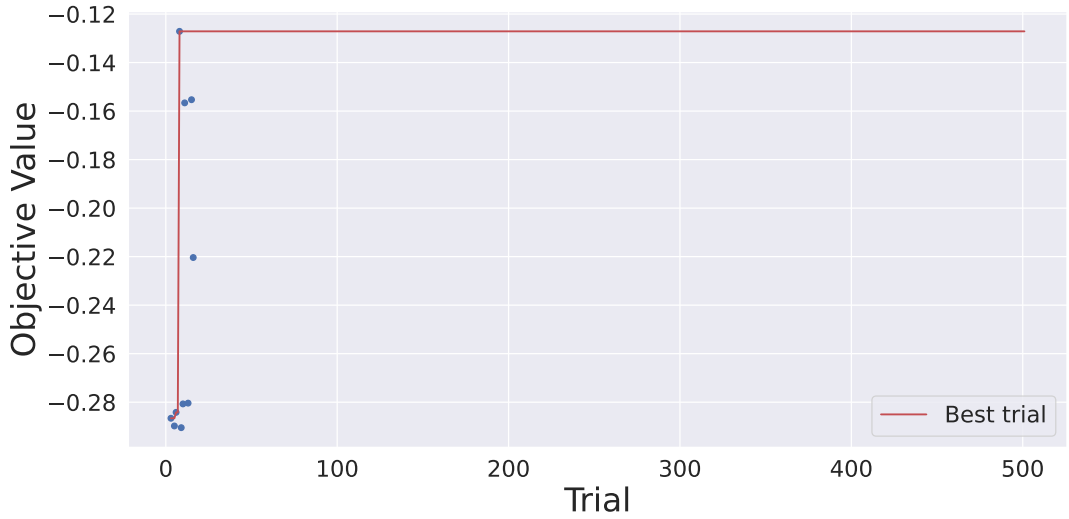


Figure 61: Optimization history of the **HP** for the traditional approach. Failed trials are not plotted.

The parameter importance plot in Figure 62 shows the relative importance of each **HP** in the

optimization process. Intriguingly, Optuna identified the kernel size of the first convolutional layer as the most critical parameter, surpassing others by a significant margin. This suggests that the initial feature extraction phase greatly influences the overall model performance. Following the kernel size, the activation function and the latent dimension rank second and third in importance, respectively.

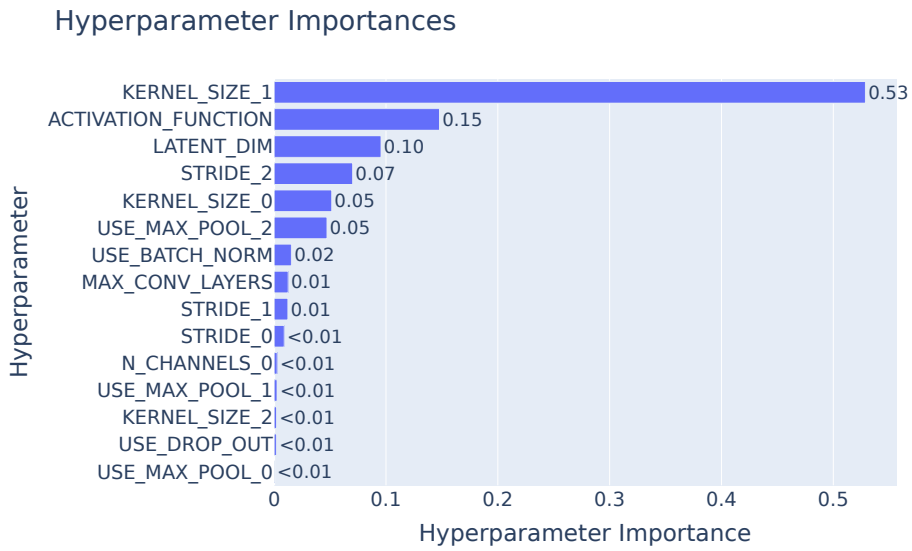


Figure 62: Parameter importance as determined by the Optuna framework for the traditional approach.

The **Empirical Distribution Function** (**EDF**) in Figure 63, shows how the cumulative probability of the trials objective values evolves over the optimization process. The best trial, that achieved ≈ -0.13 corresponds to a cumulative probability of ≈ 0.8. This indicates that the optimization process still had room for improvement, but the memory constraints of the **GPU** limited the exploration of the **HP** space.

The Figure 64 shows a slice plot of the non-layer specific **HP** and Figure 65 depicts the slice plot for layer-specific **HP** for the traditional approach. Unfortunately due to the low number of successful trials it is difficult to extract any meaningful conclusions from these figures due to low statistics.

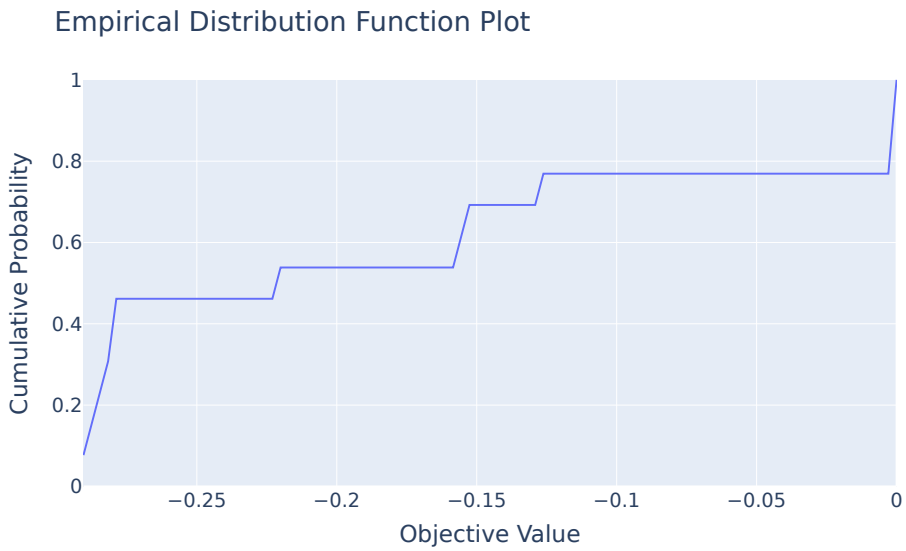


Figure 63: **EDF** for the **HP** optimization of the traditional approach.

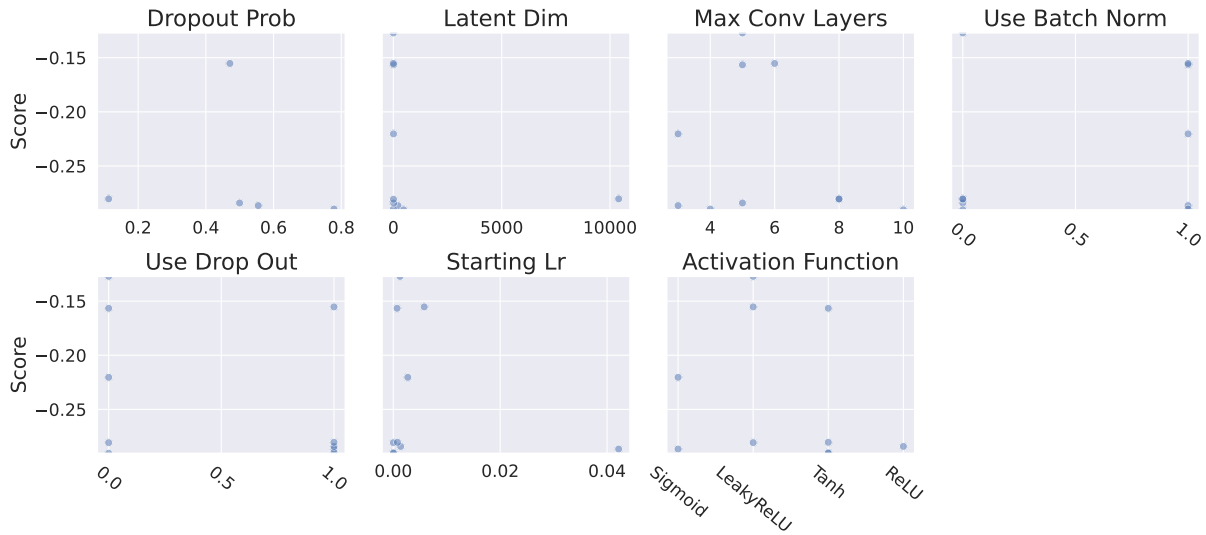


Figure 64: Slice plot of the non-layer specific **HP** for the traditional approach. Note that some **HP** have a boolean value of 1 (True) and 0 (False).

A.5 Supplementary Results - Convolutional AutoEncoder

A.5.1 Validation Results

The validation dataset is an important step for performance evaluation of the **CAE** architecture and its generalization capabilities. In this specific case, it also serves to tune additional parameters used across the post-processing techniques (subsection 2.1.6).

Following the optimization process, the best weights were loaded based on the lowest model loss observed with the validation dataset's OK images. The reconstructions of both OK and

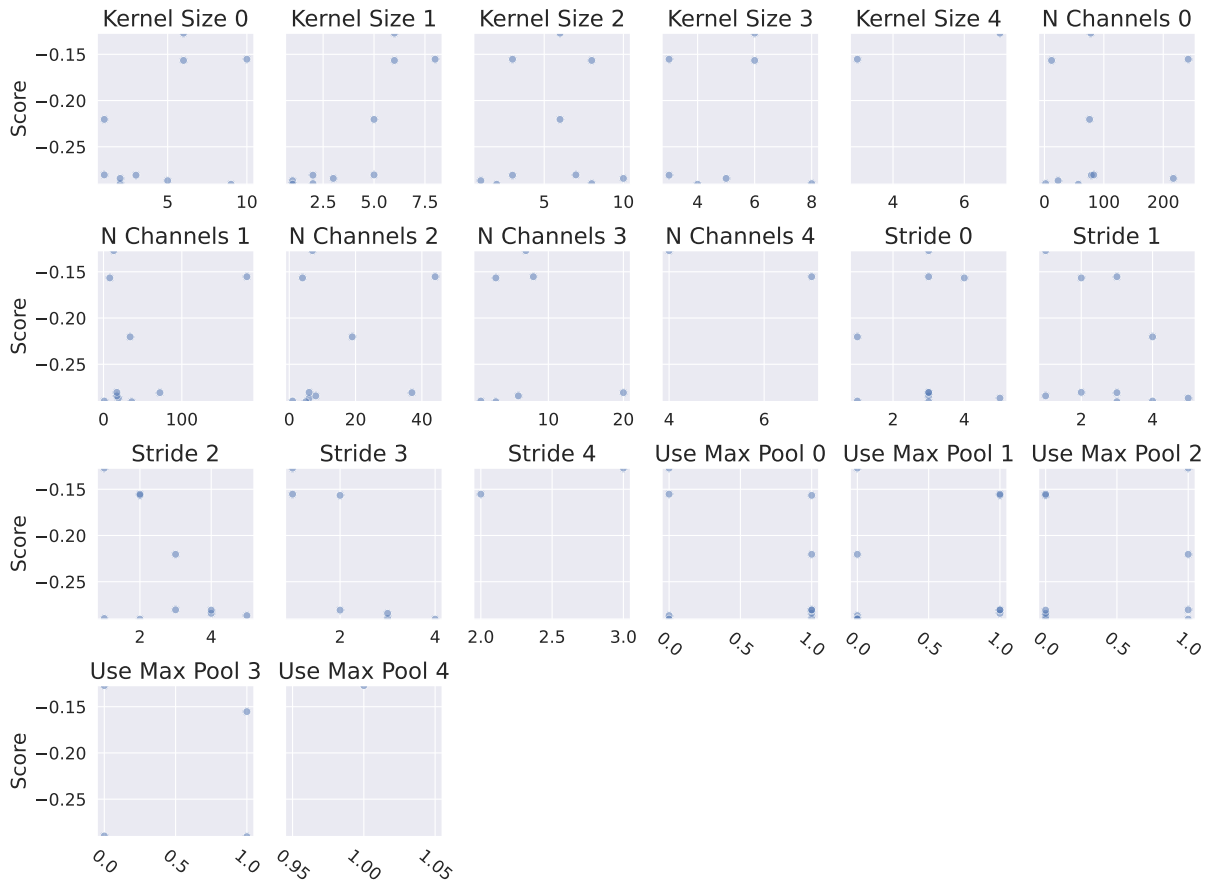


Figure 65: Slice plot of the layer specific **HP** for the traditional approach. The number on the **HP** name refers to the layer number.

synthetically generated NOK images from the validation dataset are presented in Figure 66, which juxtaposes a subset of the reconstructed images with their original versions for each label. In the case of synthetically generated NOK images, the error map produced by the square error loss highlights regions where the model faces difficulties in image reconstruction, as evidenced by "clusters" of high error values depicted in red. Conversely, the error distribution within the OK image is more uniform, featuring lower error magnitudes. This suggests that the model reconstructs these images more accurately, albeit with a residual level of noise. This differential error mapping serves as a diagnostic tool, hinting out the model's reconstruction capabilities.

Following the visual assessment, the performance of the model is shown in Figure 67 using the model's loss. While the separation is not perfect, it is evident that certain classes of NOKs, such as "gaussian noise" and "random symbols," can be completely distinguished from the OK images. However, the "in picture" and "patch" methodologies exhibit significant overlap with the OK images, indicating that the model struggles to differentiate these classes from

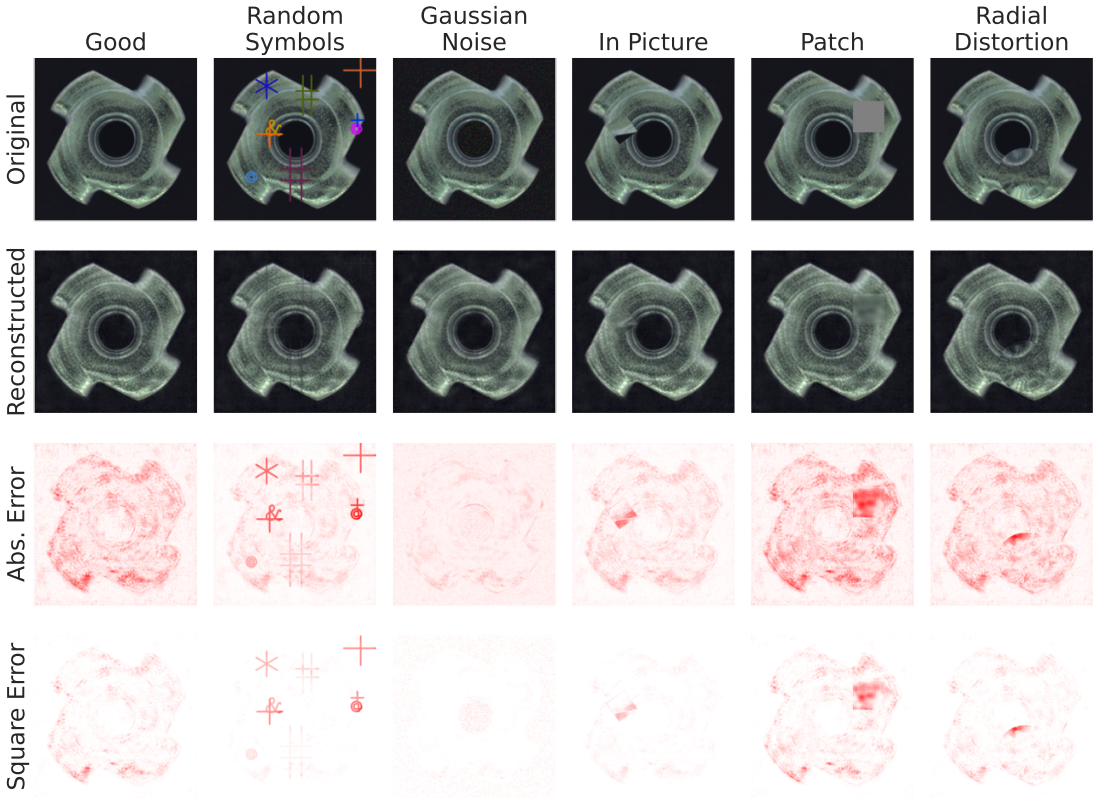


Figure 66: Reconstruction of the validation dataset using the **CAE** architecture. On each column it can be seen the different labels of the validation dataset. The first row shows the original images, the second row the reconstructed images, the third row the absolute error map and the last row the square error.

the OK images. Furthermore, when quantitatively measuring the performance, an **AUC** of ≈ 0.90 is obtained (Table 35).

Subsequent to the training phase, the model's performance was also evaluated using a common suite of metrics, absent of any further post-training enhancements shown in Table 35.

Metric	AUC	Accuracy	Recall	Precision	F1
MSE	0.8983	0.2576	0.1091	1.0000	0.1967
MAE	0.8500	0.4318	0.3182	1.0000	0.4828
BCE	0.4434	0.6818	0.7818	0.8269	0.8037
SSIM	0.8442	0.3333	0.2000	1.0000	0.3333
PSNR	0.8983	0.4848	0.3818	1.0000	0.5526
NRMSE	0.8500	0.4318	0.3182	1.0000	0.4828

Table 35: Evaluation metrics of the validation dataset.

Furthermore, to understand the relationships between different evaluation metrics and to explore the distribution of the validation data, a pairplot is provided on Figure 73. The pairplot offers a visual summary of the relationships between the different metrics. Unfortunately there is no clear separation between the OK and the various augmented-NOK classes,

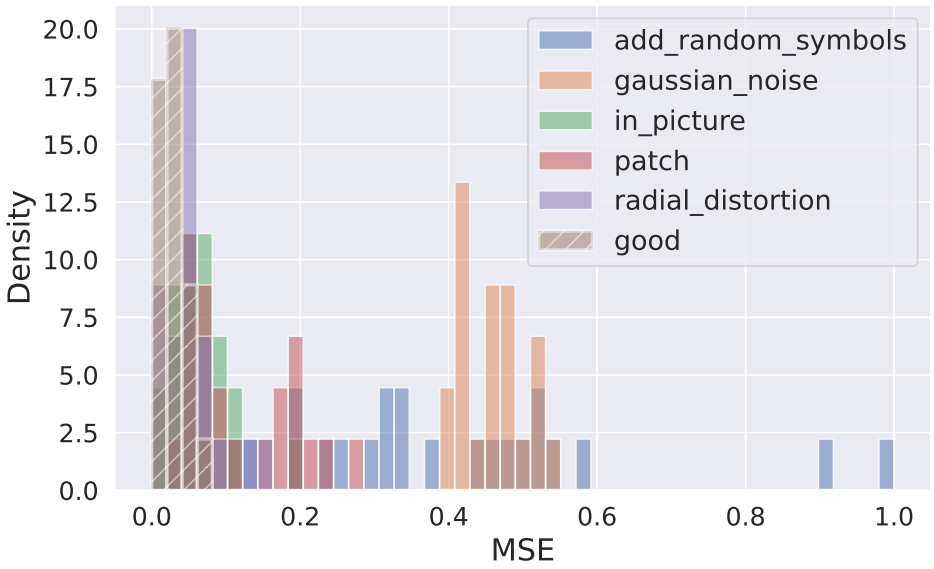


Figure 67: Histogram of the **MSE** reconstruction loss for the validation dataset using the **CAE** architecture.

Metric	IoU
MSE	0.2052
MAE	0.1780
BCE	0.0869
NRMSE	0.1780

Table 36: Table of **IoU** values for different metrics

hinting at the impossibility to to create a methodology for gaining additional classification power using alternative metrics for the reconstruction error.

A.5.1.1 Evaluation Metrics

To enhance model performance, post-processing techniques will be applied (see subsection 2.1.6). These aim to better detect anomalies by emphasizing key features and reducing noise. Such enhancements will likely increase both classification accuracy and result interpretability. Parameter tuning of specific techniques will require the use of the validation dataset, with the detailed outcomes discussed in subsequent sections. For a complete overview of these methods on the validation dataset, refer to subsection A.5.2.

- Baseline Approach

  This approach employs a threshold for the reconstruction error as a criterion to discriminate between OK and NOK images in an attempt to mimic the procedures in a production environment. Identifying the ideal threshold is a significant challenge due

to variability in error magnitude across different samples. To address this, a series of thresholds were systematically tested, with performance metrics calculated for each.

Refer to subsubsection 2.1.6.1 for an extended discussion on the methodological framework employed in determining these thresholds. The evaluation of various thresholds and their impact on classification metrics is visually represented in Figure 68.

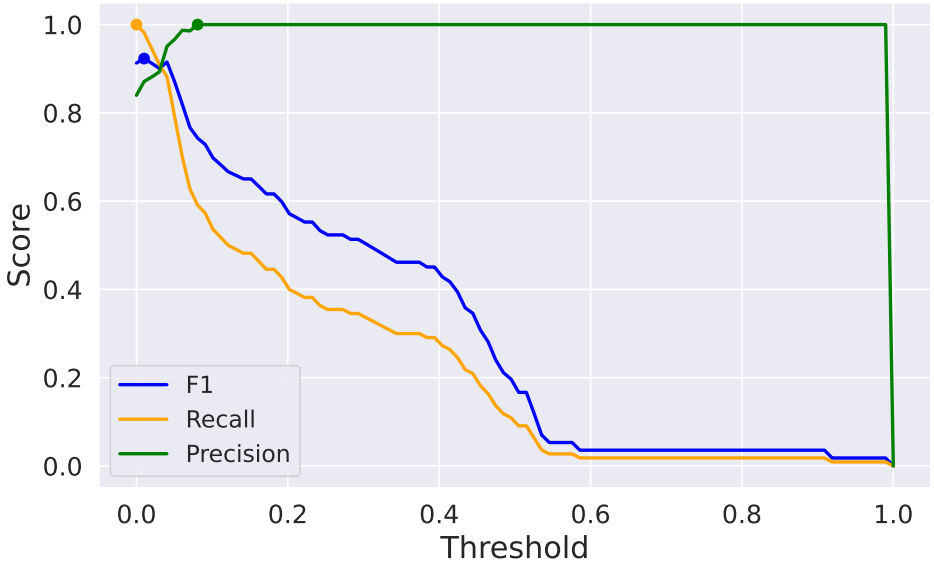


Figure 68: Evaluation of various thresholds using the baseline approach, showing how different metrics vary with each threshold setting on the validation dataset.

The optimal threshold values for each evaluation metric are summarized in Table 37 which will then be used on the test dataset for benchmarking.

Metric	Optimal Threshold
Best Threshold (F1)	0.01
Best Threshold (Recall)	0.00
Best Threshold (Precision)	0.08
Intersection of Recall and Precision	0.03

Table 37: Optimal threshold values for different metrics determined through validation testing of the baseline approach.

The main goal of setting these validation-fitted thresholds is to transition from a controlled validation phase to a rigorous assessment using the test dataset. Applying these thresholds to the test dataset aims to confirm their effectiveness in accurately classifying OK and NOK images in real-world conditions.

- Latent Dimension

The latent dimension is the crux for the **CAE** capacity to compress and reconstruct input data, significantly influencing its generalization ability on unseen data. With this in mind, exploring the latent space could provide insights into the model's performance and areas for enhancement as detailed in subsubsection 2.1.6.2.

To quantify the latent dimension, statistical summaries such as the sum, mean, and square of the latent values were computed. These summaries can be visualized on Figure 74.

Since the operations are reductionist, meaning that the full extent of dimensions in the latent vector is reduced to a single data point for each image, **PCA** was employed. **PCA** was used to compress the latent space into fewer dimensions that retain the most significant features. The **PCA** projection to two dimensions is illustrated in Figure 69, providing a simplified yet informative view of the latent space dynamics.

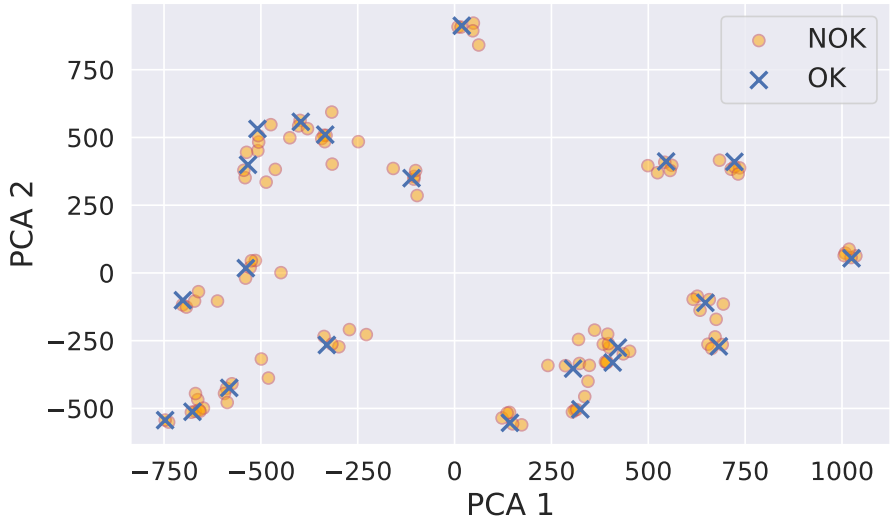


Figure 69: An **PCA** transformation of the latent space, demonstrating dimensionality reduction to two principal components.

Unfortunately, these efforts did not provide a clear separation between the OK and NOK classes, indicating that the latent space is incapable of distinguishing between the two classes in the validation dataset. This suggests that employing more advanced methods, such as **SVM**, for classification on the test dataset might be unfeasible.

- Filtering

This section explores the effectiveness of various image filtering techniques, as elab-

orated in section A.3, focusing on their application in **AD** within images processed by the model. The primary goal of these techniques is to refine the detection of anomalies by enhancing relevant features and minimizing noise interference, thereby improving both the classification power of the methodology and the interpretability of the results.

To quantitatively assess the impact of these filtering techniques, **AUC** and **IoU** will be used. Thresholds were defined to meet **IoU** requirements, converting the error values from the **CAE** into binary masks, with 1s indicating error regions and 0s indicating non-error regions. These thresholds were determined by applying each filtering technique to the error values from OK samples in the validation dataset, using the 90th percentile of these values as the cut-off. The resulting thresholds are shown in Figure 70.

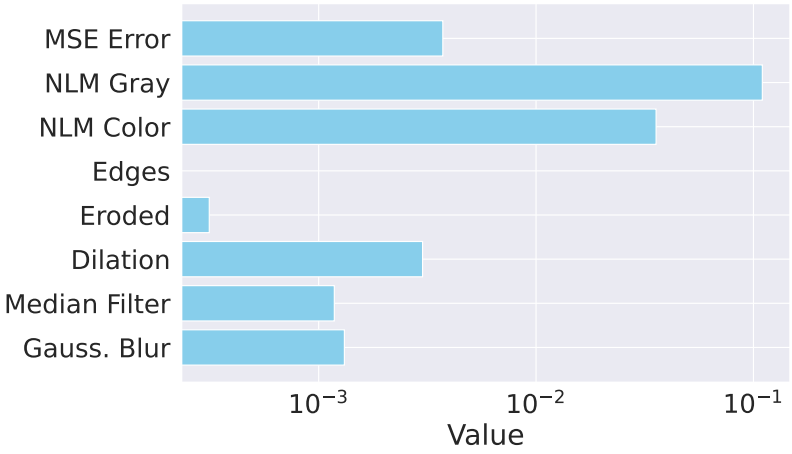


Figure 70: Thresholds for the filtering techniques using the 90th percentile of the OK-error values. Note that "Edges" (Edge Detection) has a *NaN* value since it already outputs a binary image.

The performance of each filtering technique was subsequently evaluated against the ground truth using these metrics. The performance outcomes are depicted in Figure 71.

By analyzing Figure 71 it can be seen that in general the filtering techniques have a positive impact on the model's performance in comparison to just the **MSE** reconstruction loss. **IoU** is better than the **MSE** reconstruction loss when using gaussian blur, median filter, dilation and **NLM** Gray. In regards to **AUC**, gaussian blur and median filter all have comparable performance to the **MSE** reconstruction loss. The **NLM** Color filter has the best performance in terms of **AUC**, with an **AUC** of ≈ 0.93 followed by the **NLM** Gray. This may hint that in terms of performance it may be better to process the error from the model using filtering techniques such as **NLM** color and in terms of interpretability

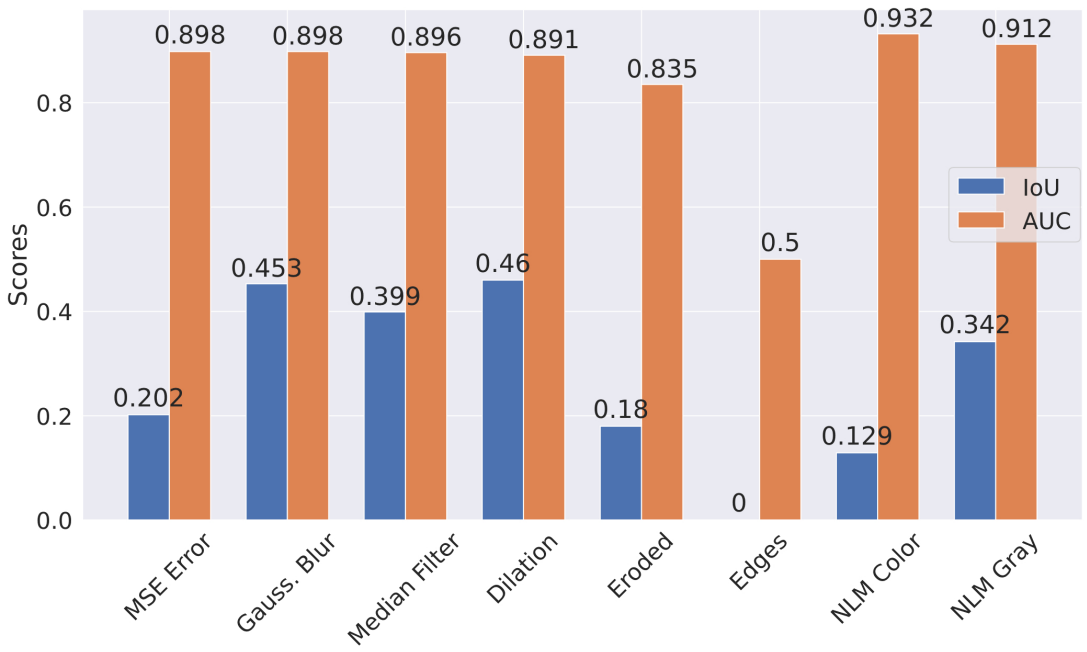


Figure 71: Comparative analysis of **AUC** and **IoU** metrics for various filtering techniques against ground truth labels and true error masks.

(**IoU**) it may be better to use the Dilation with an **IoU** of ≈ 0.46 comparatively to the out of the box **IoU** of our model of ≈ 0.2. These hypotheses will be further tested in the test dataset.

Given these findings, it is anticipated that during testing, the **NLM** Color and Dilation techniques will be the most effective for optimizing **AUC** and **IoU** metrics, respectively, when filtering the error.

Furthermore, visualizations of all applied filtering techniques are presented in subsubsection A.5.2.2 for the validation dataset, showcasing the original error image juxtaposed with masks generated by applying the established 90th percentile threshold for each filter.

- Clustering the error

The efficacy of clustering techniques (subsubsection 2.1.6.3) is illustrated in Figure 72. It is evident that the baseline **MSE** experienced a reduction to approximately 0.78 **AUC**, while maintaining its **IoU** value. When clustering techniques are combined with filtering methods, they generally preserve the performance metrics of the out-of-the-box **MSE**, except at the Median Filter, Eroded and Edges. Additionally, techniques such as Gaussian Blur, and **NLM** color slightly surpass the **MSE**, achieving a peak **AUC** of 0.83

with the latter. However, this performance still falls below the model's out-of-the-box efficacy.

In terms of **IoU**, clustering generally enhances its performance, with the Dilation technique achieving the highest **IoU** of ≈ 0.41, slightly improving upon the baseline performance. Moreover, the **NLM** color, edges, eroded and median filter are the only methods where **IoU** performance decreases.

With this in mind its expected that during testing, the best techniques for **AUC** and **IoU** when clustering the error will be the **NLM** gray and Dilation techniques, respectively.

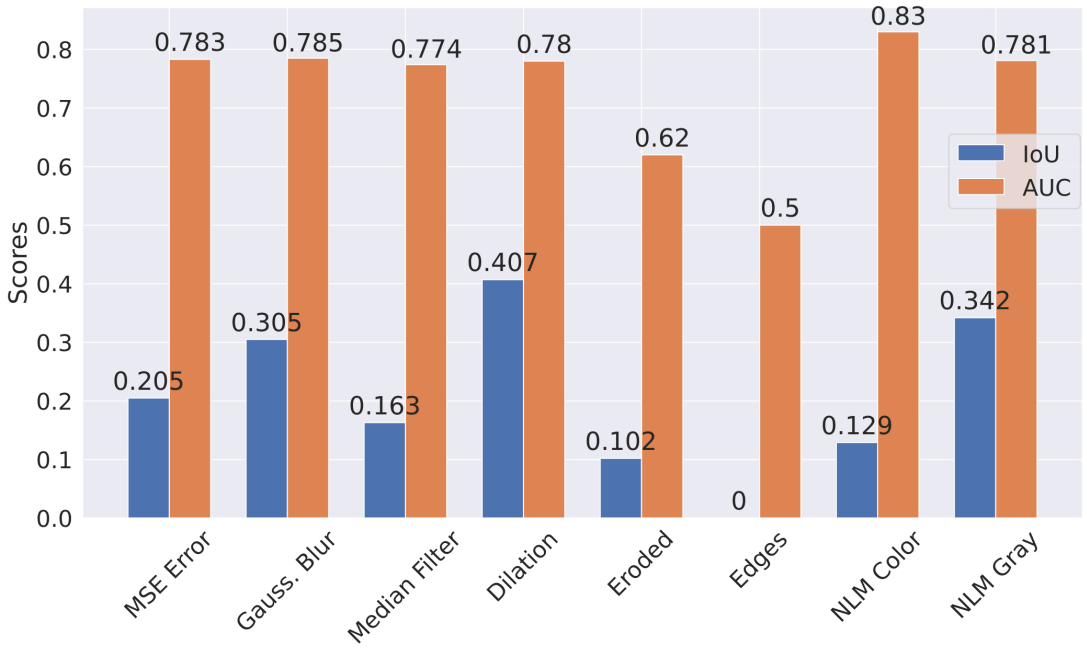


Figure 72: Evaluation of clustering performance using **AUC** and **IoU** metrics compared with ground truth labels and true error masks, illustrating the effectiveness of various clustering techniques.

Additional visualizations of the clustering techniques in the validation dataset are located in subsubsection A.5.2.3 on Figure 78 and Figure 79. In these figures, the original error image is shown alongside the clustered error regions. The ground truth is also included for comparison.

A.5.2 Additional Validation Results

A pairplot, as shown in Figure 73, reveals relationships among various evaluation metrics. This visualization offers insights into the data structure and model behavior on the validation set. Notably, clear separations are observed, particularly when using **SSIM**, between Gaus-

sian noise and other classes. Partial separations are also noticeable with the 'Add Random Symbols' label. However, challenges persist in using alternative metrics to improve classification based on reconstruction errors, indicating limited support for integrating an **SVM** to enhance classification.

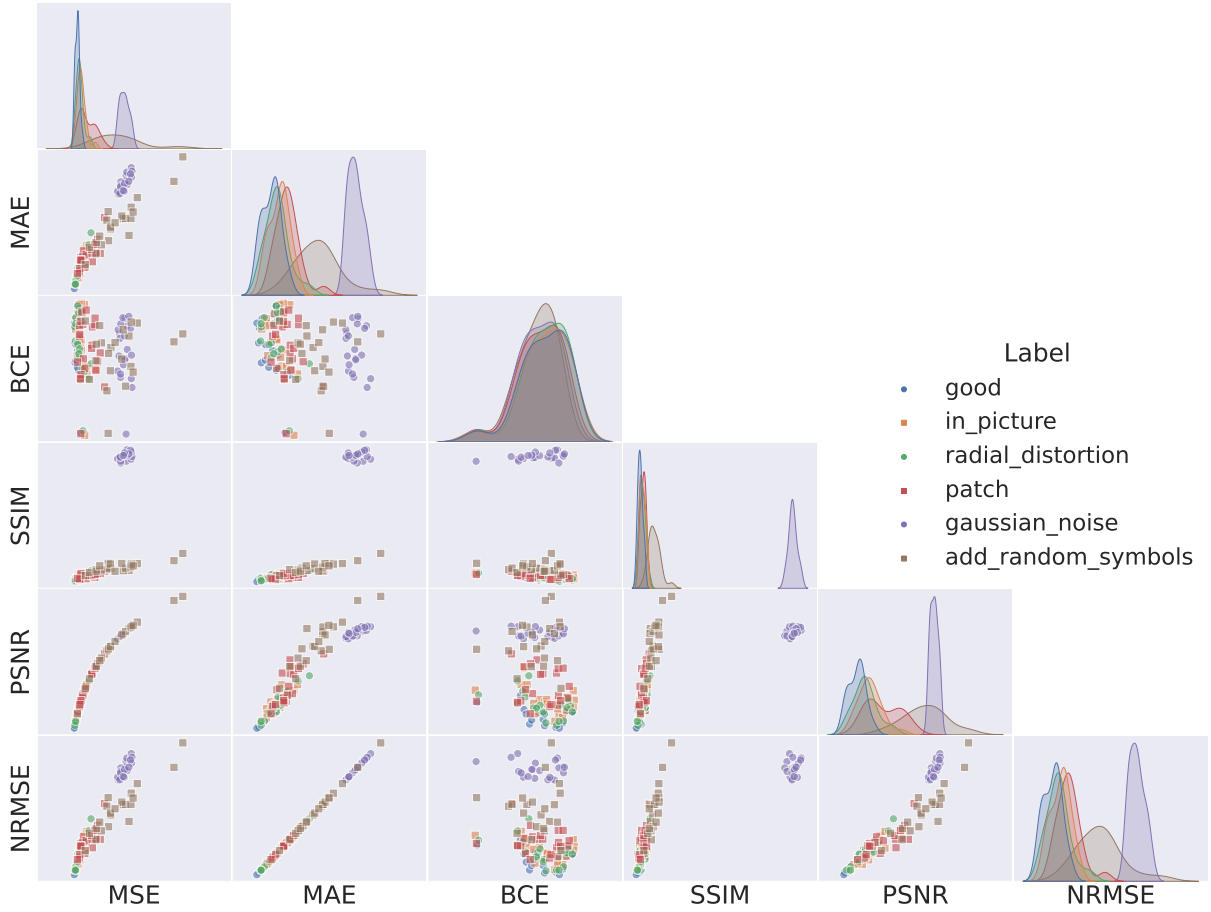


Figure 73: Pairplot of different metrics used to assess the reconstruction error of the validation dataset using the **CAE** architecture. The metrics include **MAE**, **BCE**, **SSIM**, **PSNR**, and **NRMSE**.

A.5.2.1 Latent Space Visualization

To quantify the latent dimension, statistical summaries such as the sum, mean, and square of the latent values were computed. These summaries were then visualized using a pair plot to illustrate the relationships and distributions within the latent space, as shown in Figure 74. Here, a significant overlap between all the labels from the validation set across all metrics is observed, except for the "gaussian noise" label, which is clearly separated from the rest when the **MSE** is used compared to the other metrics.

In Figure 75, the four components of the **PCA**-transformed latent space are displayed. This

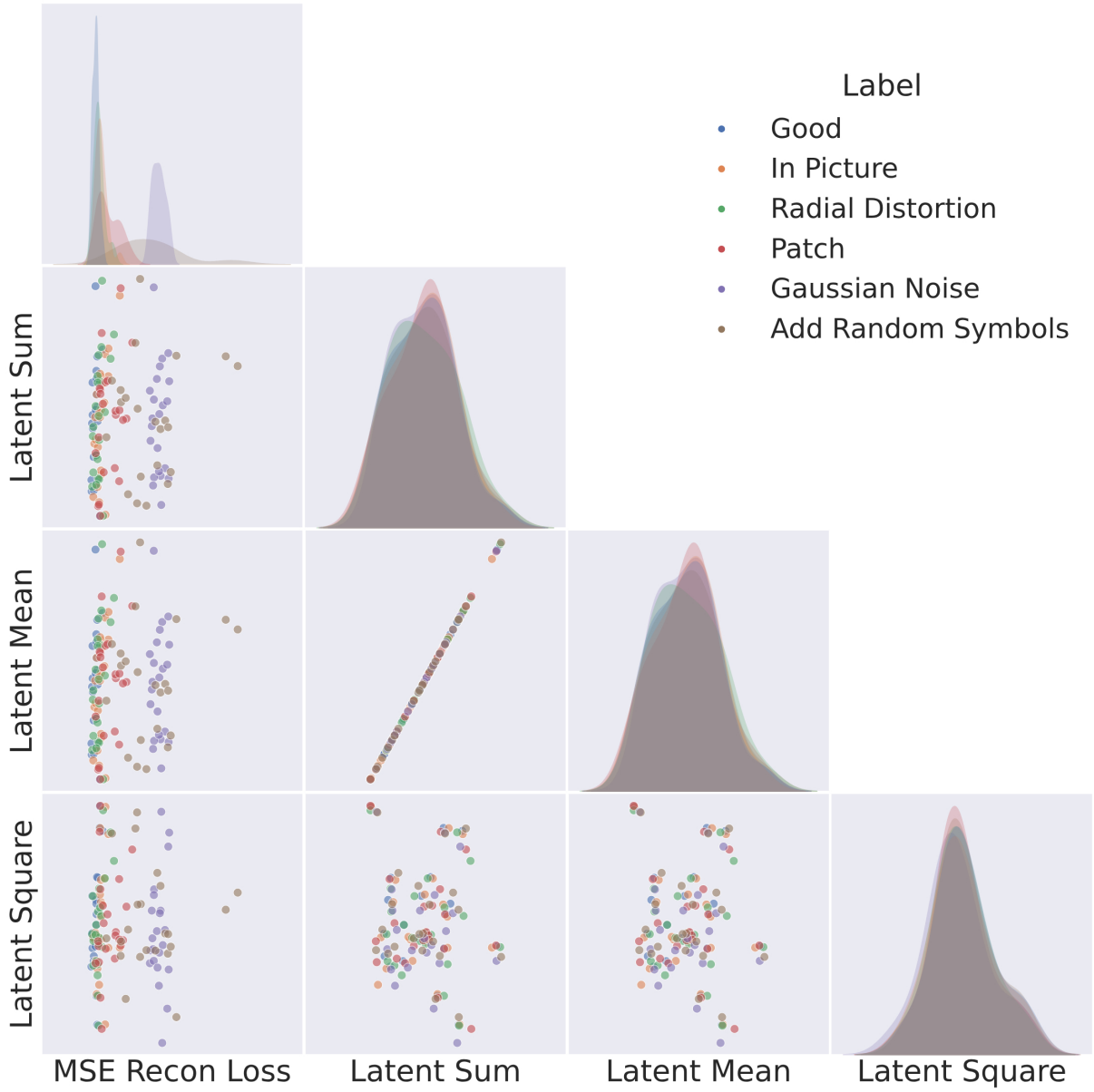


Figure 74: Pairplot visualization of the **MSE** reconstruction loss and the latent space statistical summaries.

approach was prompted by the unsatisfactory results of the 2D **PCA**, however, this too yielded unsatisfactory outcomes. The rationale behind this is that if the latent space is too complex to be meaningfully reduced to 2D, it might still be informative in 4D. The figure shows a significant overlap between all the labels, indicating that the latent space is unable to separate the different classes of the validation dataset.

A.5.2.2 Filtering Techniques

Detailed visualizations of all filtering techniques applied to the validation dataset are provided in Figure 76 and Figure 77. These figures display the original error image alongside

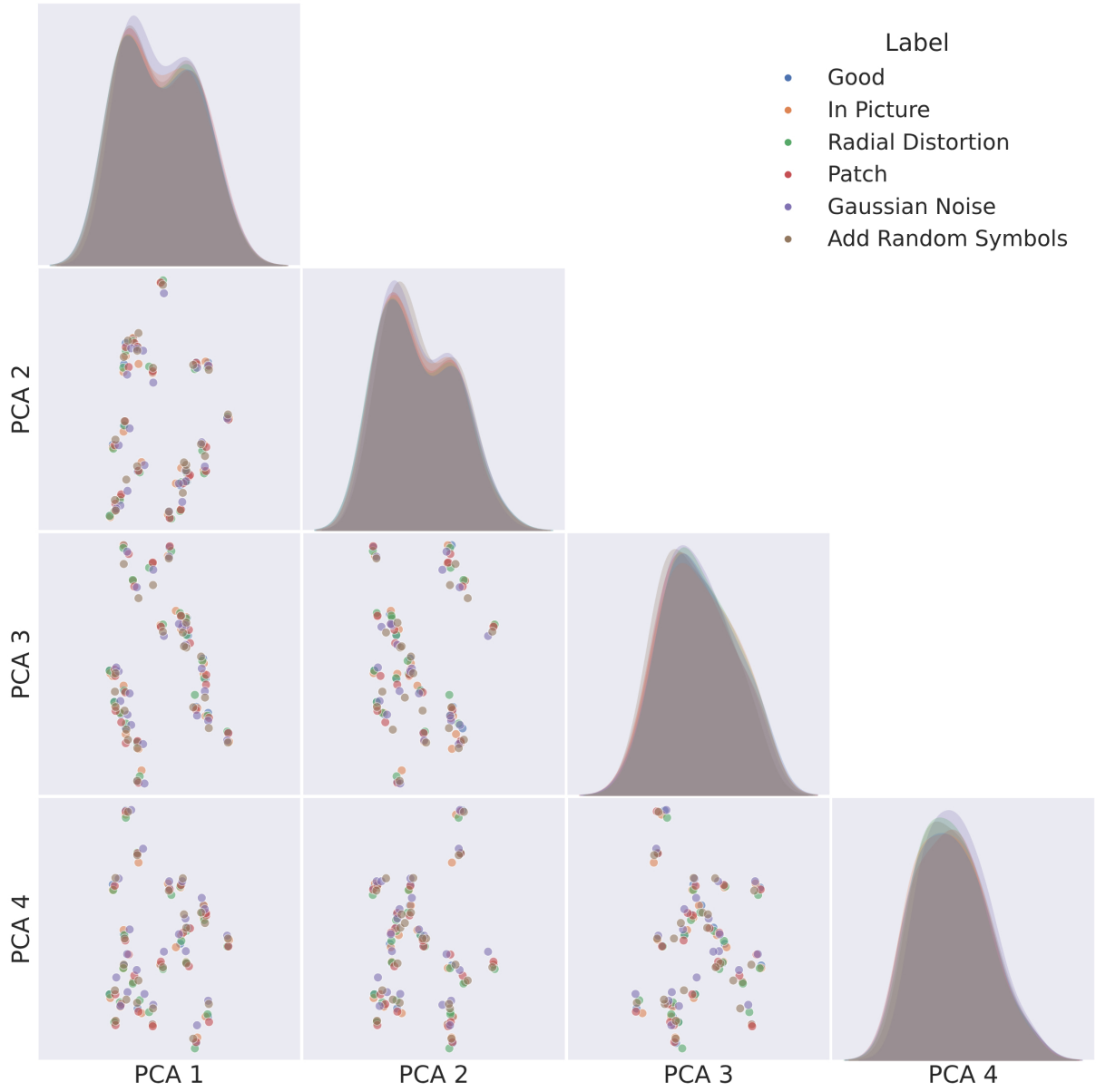


Figure 75: Pairplot of the **PCA**-transformed latent space

masks generated using a 90th percentile threshold for each filtering method. The visual comparisons, complemented by the ground truth shown in the last column of each figure, highlight the impact of filtering techniques in **AD**.

The analysis reveals varied performances across different anomalies. For the random symbols anomaly, the median filter outperforms others by clearly distinguishing anomalies with minimal noise. In the case of the patch anomaly, although all methods introduce some noise, the original **MSE** visually excels. For in-picture and radial distortion anomalies, the best methods are **MSE** and **NLM** gray, despite producing very noisy and less interpretable images. Lastly, for Gaussian noise, all methodologies perform comparably.

A.5.2.3 Clustering Techniques

For the clustering techniques used on the validation dataset, the results are shown in Figure 78 and Figure 79. These figures display the original error image alongside masks generated using a 90th percentile threshold for each clustering method. The visual comparisons, complemented by the ground truth shown in the last column of each figure, highlighting the impact of clustering techniques in improving **AD** capabilities.
Analysis shows that for both the patch and in-picture anomalies, **NLM** color achieves nearly perfect anomaly identification with zero residual error. For the random symbols anomaly, **MSE** exhibits superior performance, though it misses some features. However, for the ground truth and radial distortion, no technique emerges as particularly effective.

A.5.3 Additional Test Results

The following figures and tables present the supplementary test results obtained through the **CAE** approach.
In Figure 80, it is possible to observe the separation of different labels via the **MSE** loss histogram on the test dataset. The flip label is distinguishable from the OK images, as expected due to its abrupt transformation. However, the other NOK labels overlap with the OK labels, making classification more challenging. This resulted in an **AUC** of approximately 0.62.
The pairplot in Figure 81 illustrates the relationships between various metrics used to evaluate the model's performance on the test dataset, including **MAE**, **BCE**, **SSIM**, **PSNR**, and **NRMSE**. Similar to before, it shows a separation between the flip label and the other. However, for the remaining labels, the separation is less evident as they completely overlap.

A.5.3.1 Filtering Techniques

The plots Figure 82 and Figure 83 display the original error image post application of a filtering technique, the mask derived from the inferred validation threshold, and the ground truth anomaly mask.
By examining the filtering mask alongside the ground truth mask reveals notable cases such as the 'bent' anomaly. Despite some residual noise, techniques like erosion or **NLM** effectively delineate anomaly locations in the model's outputs. Similarly, the 'color' anomaly is distinctly identified with minimal noise interference using the erosion method. Though less definitive for the 'scratch' anomaly, it remains detectable in the final output. While many

techniques successfully pinpoint anomalies, **NLM** Color introduces significant noise while identifying anomalies.
These results indicate that while the model can detect anomalies in its reconstruction error, noise persists, thereby impacting both classification accuracy and interpretability. Furthermore, these figures offer qualitative insights which could be utilized by factory production line workers to pinpoint errors.

A.5.3.2 Clustering Techniques

The figures Figure 84 and Figure 85 display the original error image post application of a clustering technique, the mask derived from the inferred validation threshold, and the ground truth anomaly mask. Here its possible to see the diminished noise interference that clustering provides. By clustering the error and filtering only the reasonably larger error clusters, a final crisp image of where the anomaly is located can be obtained. This is evident in the 'bent' anomaly, where the 'Eroded' technique is able to effectively identify the anomaly with zero noise interference. Similarly, the 'color' anomaly is distinctly identified with minimal noise interference using the 'Eroded' method. For the 'Scratch' anomaly the **NLM** color appears to be best with also zero noise interference.

A.5.3.3 Best Performance Achieved

This section details the inclusion of the **BCE** metric on the post-processing pipeline.
The impact of different filtering techniques on the test dataset is captured in Figure 86. 'Eroded' technique achives the best performance, an **AUC** score of 0.82. Regarding the **IoU** metric, the highest score of approximately 0.22 is recorded with the dilation techniques.
Similarly, Figure 87 showcases how various clustering strategies influence the test dataset's performance. The 'Eroded' technique again proves effective, attaining an **AUC** score of approximately 0.72. In terms of **IoU**, the best performance, approximately 0.21, is achieved with the eroded techniques.
In conclusion, when employing the **BCE** metric, the best outcomes are observed with the Eroded and Dilation techniques, achieving **AUC** scores of up to 0.82 and **IoU** scores of 0.22, respectively.

A.6 Supplementary Results - Improved CAE

The slice plot of the non-layer-specific **HP** for the improved **CAE** architecture is shown in Figure 88. The plot shows that the distribution of the maximum number of convolutional layers appears approximately normal, peaking around 5 layers. This result arises not solely from the **HP** settings but also due to the memory constraints imposed by the GPU. Models featuring up to 5 layers demonstrated suboptimal complexity, resulting in lower performance scores. In contrast, models exceeding 5 layers achieved a roughly stable level of complexity, as they reached the GPU's memory capacity. Regarding batch normalization, the optimization process showed a preference for not incorporating it, which was unexpected since its known benefits for faster model convergence. Furthermore, the **LR** was optimized towards the lower end of the spectrum, consistent with the understanding that a higher **LR** could cause model divergence. The optimization of the latent dimension also favored the lower end, aligning with the expectation that a larger latent dimension might induce model overfitting. Interestingly, the best trial model adopted ReLU and **L1** for its activation and loss functions, respectively. However, the optimization process showed a marked preference for Leaky ReLU and **MAE**, as evidenced by the significantly greater number of trials employing these **HP** at higher values. This pattern suggests that further optimization might be achieved by increasing the number of trials. Furthermore since the **BCE** loss achieved a very good classification power on the previous section, it was expected that the **BCE** loss but unfortunately it achieved a lower performance on the objective metric.

The slice plot of layer-specific **HP** is depicted in Figure 89. Unfortunately, it is not densely populated due to the low number of successful trials. From the analysis of kernel size, it is evident that the added degrees of freedom were beneficial, although they perhaps lacked sufficient flexibility, as the trials tended to maximize performance near the upper limit of kernel size. Concerning the number of channels, the optimal number for the first layer appears to be around 200, a finding consistent with prior results. For subsequent layers, there is a trend where Optuna selects progressively fewer channels. The optimization of stride parameters progressed without encountering any predefined **HP** constraints, as in previous instances. The utility of max pooling seems beneficial only in the second layer and not in the first. It should be noted that hardware limitations restricted the exploration of the full spectrum of **HP**s, and the results reflect the best achievable configurations within the available resources.

A.6.1 Validation Results

The validation dataset will be utilized throughout the optimization process, as discussed in section 2.1. The optimization pipeline seeks to optimize the objective function, which necessitates NOK data, as referenced in subsection 2.1.4. In this section, both traditional (subsection A.1.1) and realistic-looking (subsection A.1.2) augmentation techniques are employed to generate new NOK datasets from the OK validation dataset.

Subsequent to the optimization process, the best weights were loaded using a similar approach as before. Exemplary reconstructions of both OK and synthetically generated NOK images from the validation dataset are presented in Figure 90, were reconstructed images and their original image for each label are compared. By analyzing it its possible to see that synthetically generated NOK images include error maps highlighting regions of reconstruction difficulty which mostly concur to where the anomaly is, while OK images show a more uniform error distribution, indicating some residual noise.

Following visual assessment, the model's performance is illustrated in Figure 91 using its loss function. The histogram effectively demonstrates the separation between OK and augmented-OK images. A significant overlap between the OK images and the augmented NOKs is observed, indicating challenges in differentiation. Despite these challenges, the model achieves an **AUC** of approximately 0.90, as shown in Table 38.

Following the training phase, the model's performance evaluation was conducted using the standard suite of metrics without any additional post-training enhancements or processing adjustments, as indicated in Table 38.

Metric	AUC	Accuracy	Recall	Precision	F1
MSE	0.9066	0.2045	0.0455	1.0000	0.0870
MAE	0.8682	0.2803	0.1364	1.0000	0.2400
BCE	0.4603	0.6970	0.8182	0.8182	0.8182
SSIM	0.7996	0.2576	0.1091	1.0000	0.1967
PSNR	0.9066	0.4470	0.3364	1.0000	0.5034
NRMSE	0.8682	0.2803	0.1364	1.0000	0.2400

Table 38: Evaluation metrics of the validation dataset.

Additional visualizations, including a pairplot in Figure 97, show relationships between various evaluation metrics. This pairplot reveals the model's behavior on the validation set but lacks clear separation between OK and augmented-NOK classes. This suggests limitations in using reconstruction error to enhance classification power, thus reducing the feasibility of

Metric	IoU
MSE	0.2044
MAE	0.2088
BCE	0.0847
NRMSE	0.2088

Table 39: Table of **IoU** values for different metrics

employing an **SVM** for this purpose.

A.6.1.1 Evaluation Metrics

As discussed in subsection 2.1.6, post-processing techniques will be applied to improve **AD**. These enhancements will involve parameter tuning using the validation dataset. Additional discussion can be found in subsection A.6.2.

- Baseline Approach

  This method employs a threshold on the model's reconstruction error to differentiate between OK and NOK images, akin to standard production practices. Finding an optimal threshold is difficult due to sample error variability. Multiple thresholds were evaluated systematically, with performance metrics calculated for each.

  For a detailed discussion on the methodological framework and threshold determination, see subsubsection 2.1.6.1. The impact of different thresholds on classification metrics is illustrated in Figure 92.

  The optimal threshold values for each evaluation metric are summarized in Table 40. This includes the thresholds where metrics such as F1 score, recall, and precision individually achieve their best performance, as well as the threshold where precision and recall intersect, indicating a balanced trade-off between these metrics.

Metric	Optimal Threshold
Best Threshold (F1)	≈ 0.03
Best Threshold (Recall)	0.00
Best Threshold (Precision)	≈ 0.14
Intersection of Recall and Precision	≈ 0.04

Table 40: Optimal threshold values for different metrics determined through validation testing of the baseline approach.

  As done before, establishing validation-fitted thresholds facilitates the transition from controlled validation to rigorous testing with the test dataset. This step evaluates the

robustness and generalizability of the determined thresholds.

- Latent Dimension

  Mirroring the approach in the previous section, statistical summaries were computed and visualized via a pair plot (Figure 74) but unfortunately didn't yield satisfactory results. Generally, there was an considerable overlap between the classes that used the comptuted components. The only exception was simply using the **MSE** loss where some separation could be observed between the majority of classes and gaussian noise or random symbols, were the latter was less aparent.

  With this in mind, **PCA** was utilized to compress the latent space into fewer dimensions, aiming to facilitate visualization and enhance data separation. Unfortunately, the resulting two-dimensional projection (Figure 93), exhibits considerable overlap between the classes which complicates the applicability of an **SVM**.

- Filtering

  This section evaluates image filtering techniques (section A.3) used in **AD** for model-processed images, assessing them with **AUC** and **IoU** metrics. The **IoU** requirements define thresholds for converting **CAE** error values into binary masks. These thresholds, determined by applying each technique to OK sample errors in the validation dataset and using the 90th percentile as the cut-off, are detailed in Figure 94.

  The performance of filtering techniques was assessed using **AUC** and **IoU** metrics against ground truth. Figure 95 depicts the results, demonstrating a generally comparable impact to **MSE** reconstruction loss. Only a few cases slightly outperform **MSE** in terms of **AUC**, such as Eroded, Median Filter, **NLM** color, and **NLM** gray, with a maximum **AUC** of approximately 0.93 of the Erosion filter. Regarding **IoU**, nearly all methods outperform **MSE**, with Dilation achieving the highest **IoU** of approximately 0.29, except for **NLM** color.

  Given these findings, it is anticipated that during testing, the Erosion and Dilation techniques will be the most effective for optimizing **AUC** and **IoU** metrics, respectively, when filtering the error.

  Additionally, visualizations of all applied filtering techniques are provided in subsubsection A.6.2.2 (Figure 100 and Figure 101).

- Clustering the error

  This section describes the error clustering methodology (subsubsection 2.1.6.3). Its primary goal is to delineate anomalies clearly for straightforward fault identification, enhancing distinction between OK and NOK images by filtering out noise. Ideally, OK images yield blank outputs, while NOK images display prominent error clusters.

  The efficacy of these clustering techniques was assessed using **AUC** and **IoU** metrics, as illustrated in Figure 96. It is evident that the baseline **MSE** experienced a reduction to approximately 0.84 **AUC**, while maintaining its **IoU** value. When clustering techniques are combined with filtering methods, they generally preserve the performance metrics of the out-of-the-box **MSE**, except at the edges where they register 0.5 **AUC** and 0 **IoU**. Additionally, techniques such as eroded, **NLM** color, and **NLM** gray slightly surpass the **MSE**, achieving a peak **AUC** of 0.86 with **NLM** gray. However, this performance still falls below the model's out-of-the-box efficacy.

  In terms of **IoU**, clustering generally enhances its performance, with the Eroded technique achieving the highest **IoU** of 0.23, slightly improving upon the baseline performance. Moreover, the **NLM** color and **NLM** gray techniques are the only methods where **IoU** performance decreases.

  With this in mind its expected that during testing, the best techniques for **AUC** and **IoU** when clustering the error will be the **NLM** gray and Eroded techniques, respectively.

  The visualization of clustering techniques in the validation dataset can be found in subsubsection A.6.2.3 on Figure 102 and Figure 103. These figures display the original error image alongside the clustered error regions, with ground truth included for comparison, as in the previous section.

A.6.2 Additional Validation Results

A pairplot depicted in Figure 97 elucidates the relationships among various evaluation metrics, providing a deeper understanding of the data structure and the model's behavior on the validation set. Noteworthy is the distinct separation observed between "Realistic NOKs Quarter" and also "Realistic NOKs Half", although less noticeable in the latter. This separation is consistent with the expectation that the model would struggle to reconstruct anomalies that more closely resemble real world parts. These findings suggest viable the training of a **SVM**

to boost classification performance but unfortunately that is left for future research.

A.6.2.1 Latent Space Visualization

To assess the latent dimension, statistical metrics including the sum, mean, and squared values of the latent dimensions were calculated. These metrics were visualized through a pair plot to elucidate the relationships and distributions within the latent space, as illustrated in Figure 98. The visualization reveals overall overlap among the labels from the validation set across all metrics, although after close inspection some distinction could be made at least between the OK and the two classes of realistic NOKs when **MSE** is used.
Figure 99 shows the four components of the **PCA**-transformed latent space. The hypothesis was that a more complex latent space, unsuitable for reduction to 2D, might provide deeper insights in 4D. However, the plot demonstrates significant overlap among all labels, indicating that the latent space does not effectively distinguish between the classes in the validation dataset.

A.6.2.2 Filtering Techniques

Detailed visualizations of all filtering techniques applied to the validation dataset are illustrated in Figure 100 and Figure 101. These figures feature the original error image alongside masks created using a 90th percentile threshold for each filtering method. The side-by-side visual comparisons, enhanced by the inclusion of ground truth in the final column of each figure, show the effectiveness of the filtering techniques in enhancing **AD** capabilities within the **CAE** framework.
Analysis of the images shows that for the in-picture and patch anomalies, the median filter, despite some residual noise, most effectively identifies the anomalies. For radial distortion, **NLM** gray performs best, albeit with significant noise. However, for the two classes of data-augmented anomalies, there is no clear winner.

A.6.2.3 Clustering Techniques

The figures Figure 108 and Figure 109 show the original error image after applying a clustering technique, alongside the mask derived from the inferred validation threshold and the ground truth anomaly mask.
Clustering aims to reduce noise interference, yet initial results reveal noisy outcomes in some images. For the flip anomaly, Gaussian noise technique closely approximates its ground

truth. For scratch and bent anomalies, the median filter proves most effective, although it introduces some noise in bent images. For color anomalies, dilation is superior, despite significant noise.

A.6.3 Additional Test Results

The supplementary test results using the improved **CAE** approach are detailed in the following figures and tables.

The **MSE** loss histogram for the test dataset is displayed in Figure 80and it shows how different labels are separated. The flip label stands out clearly against the OK images, which aligns with expectations due to its distinct transformation. However, the overlap of other NOK labels with OK labels complicates the classification process, resulting in an **AUC** of about 0.62, the same as the previous methodology.

The pairplot in Figure 105 illustrates the relationships between different metrics that assess model performance on the test dataset. There is a notable separation between the flip label and other labels across all metrics. However, these other labels significantly overlap, resulting in less distinct separation. Closer analysis reveals that when combining the **BCE** metric either with others or itself, the separation between OK and NOK images becomes more pronounced.

A.6.3.1 Filtering Techniques

The plots Figure 106 and Figure 107 showcase the original error image after applying a filtering technique, the derived mask from the validation threshold, and the ground truth anomaly mask.

Analyzing these masks, the Bent anomaly is most effectively detected by **NLM** color, though it misses a feature (cluster of error). For color anomalies, **MSE** proves to be the best despite some noise. The scratch anomaly is best identified by dilation, albeit with significant noise. Finally, for the flip anomaly, Gaussian blur remains the superior method as previously noted.

A.6.3.2 Clustering Techniques

The figures Figure 108 and Figure 109 demonstrate how clustering effectively reduces noise. This method groups errors and filters out larger clusters, enhancing image clarity and accurately pinpointing anomalies. This is especially evident with the scratch anomaly, where the median filter excels by identifying it with minimal noise. The bent anomaly is detectable us-

ing **MSE**, although it misses a feature. No clear best method emerges for color anomalies. For the flip anomaly, Gaussian blur is the standout method.

A.6.3.3 Best Performance Achieved

This section details the application of the **BCE** metric across an array of post-processing techniques. The ensuing figures delineate the test dataset's performance subjected to diverse filtering and clustering strategies.

The impact of different filtering techniques on the test dataset is captured in Figure 110. Among these, the **NLM** gray technique stands out, achieving an **AUC** score of 0.87. Regarding the **IoU** metric, the highest score of approximately 0.19 is recorded with the dilation techniques.

Similarly, Figure 111 showcases how various clustering strategies influence the test dataset's performance. The 'Eroded' technique again proves effective, attaining an **AUC** score of 0.75. In terms of **IoU**, the best performance, approximately 0.19, is achieved with the median filter technique.

In conclusion, when employing the **BCE** metric, the best outcomes are observed with the **NLM** gray and dilation techniques from the filtering approach, achieving **AUC** scores of up to 0.87 and **IoU** scores of 0.19, respectively.

A.7 Supplementary Results - VAE

The slice plot for non-layer-specific **HP** in the **VAE** architecture, shown in Figure 112, indicates a roughly normal distribution of the maximum number of convolutional layers, peaking around the 15th layer. This peak suggests that models with up to 15 layers had suboptimal complexity, leading to lower performance scores, while models with more than 15 layers reached a stable complexity level as they hit the GPU's memory limit. Notably, the "ideal" complexity for **VAE** is substantially higher than for **CAE**, reflecting **VAE**'s complexity due to its task of learning a data distribution. Performance for the maximum number of linear layers peaked at 2, with a significant drop in trials exceeding 7 layers. The use of batch normalization proved beneficial, with optimizers preferring it and achieving higher performance, aligning with the literature and contrasting previous studies. The **LR** was optimized towards lower values to prevent model divergence, and a smaller latent dimension was favored to mitigate overfitting. Regarding the loss function, the optimizer clearly favored **MSE**, which yielded higher scores and was selected in more trials.

The slice plot of layer-specific **HP** in Figure 113 reveals distinct preferences for the parameters across different layers. In the first layer, the kernel size is uniformly distributed with a peak around 12. The second layer's kernel size reaches the upper limit set by the parameters, suggesting that increasing this limit might lead to even better performance. Conversely, the third layer shows a preference for a minimal kernel size of 1, indicating that smaller kernels are more effective at this level and suggesting the model benefits from capturing finer details in deeper layers.

Channel numbers also vary significantly across layers, following a uniform distribution with the peaks at approximately 250 channels in the first layer, 100 in the second, and 50 in the third. This pattern aligns with the idea that **AE** benefit from reducing the complexity as the network deepens (i.e. information bottleneck).

Stride settings show that the model favors a stride of 5 in the first layer and 15 in the second, but drops to 1 in the third, possibly to preserve more information in deeper layers. For the linear layers, there's a clear preference for around $10,000$ units. The model also performs better without using max pooling across all layers, which could indicate that max pooling might be detrimental to capturing finer details in this context

A.7.1 Validation Results

The validation dataset is used throughout the optimization process, as detailed in section 2.1. This pipeline aims to tune the objective function using NOK data, discussed in subsection 2.1.4. Both traditional (subsection A.1.1) and more realistic augmentation methods (subsection A.1.2) were used to generate new NOK datasets from the OK validation dataset, similar to the approach in section 3.3.

Following the optimization process, the optimal weights were reloaded using the same method as before. In Figure 114, exemplary reconstructions of both OK and synthetically generated NOK images from the validation dataset are shown. The figure compares the reconstructed images with their original counterparts. Analysis shows a nearly uniform level of residual noise across all labels. Additionally, anomalies in traditional data augmentation techniques, though detectable, are quite faint and could easily be mistaken for noise. Realistic data augmentation techniques, on the other hand, tend to exhibit a high level of noise along their edges. This could indicate poor performance due to unclear reconstruction errors.

After visually assessing the model, Figure 115 shows its performance using the loss function histogram. Unfortunately, there's an overlap between the distributions of OK and NOK images. Quantitatively, the model only manages an **AUC** of about 0.57, which is quite disappointing (Table 41).

Subsequent to the training phase, the model's performance was also evaluated using a common suite of metrics, absent of any further post-training enhancements or processing adjustments shown in Table 41.

Metric	AUC	Accuracy	Recall	Precision	F1
MSE	0.5740	0.2727	0.1364	0.9375	0.2381
MAE	0.5595	0.4167	0.3545	0.8667	0.5032
BCE	0.3826	0.5758	0.6636	0.7935	0.7228
SSIM	0.3979	0.5303	0.5818	0.8000	0.6737
PSNR	0.5740	0.4924	0.4727	0.8525	0.6082
NRMSE	0.5595	0.4167	0.3545	0.8667	0.5032

Table 41: Evaluation metrics of the validation dataset.

A pairplot in Figure 121 shows the relationships between different evaluation metrics and offers insights into the data structure and model performance on the validation set. However, the lack of clear separation between OK and various augmented-NOK classes suggests that alternative metrics may not improve classification based on reconstruction error, potentially

Metric	IoU
MSE	0.1228
MAE	0.1237
BCE	0.0466
NRMSE	0.1237

Table 42: Table of IoU values for different metrics

limiting the effectiveness of training an SVM for enhanced classification.

A.7.1.1 Evaluation Metrics

Building on the previous section, this section will apply the post-processing techniques detailed in subsection 2.1.6. These techniques focus on emphasizing important features and reducing noise, which could improve both classification accuracy and interpretability. Parameter adjustments will be made using the validation dataset. Details on the parameters from these adjustments are outlined in the following subsections. For a comprehensive analysis of how these techniques perform on the validation dataset, refer to subsection A.7.2.

- Baseline Approach

  The method uses a threshold on the model's reconstruction error to distinguish OK from NOK images, aligning with standard practices in production environments. Since setting an optimal threshold is challenging due to error variability among samples, a systematic evaluation of multiple thresholds was performed, and performance metrics were calculated for each.

  For more details on the methodology and threshold determination, see subsubsection 2.1.6.1. The impact of different thresholds on classification metrics is shown in Figure 116.

  The best threshold values for each evaluation metric are detailed in Table 43. This table includes thresholds where metrics such as F1 score, recall, and precision peak individually. It also features the threshold where precision and recall intersect, indicating a balanced trade-off between these metrics.

  As before, setting validation-fitted thresholds helps move from controlled validation to rigorous testing with the test dataset. This step tests the robustness and generalizability of these thresholds. Using these thresholds on the test dataset in the next

Metric	Optimal Threshold
Best Threshold (F1)	≈ 0.03
Best Threshold (Recall)	0.00
Best Threshold (Precision)	≈ 0.77
Intersection of Recall and Precision	≈ 0.12

Table 43: Optimal threshold values for different metrics determined through validation testing of the baseline approach.

section verifies their effectiveness in accurately classifying OK and NOK images under real-world conditions.

- Latent Dimension

  Exploring how the latent dimension affects compression and reconstruction is key to understanding the **CAE**'s generalization capabilities, as discussed in subsubsection 2.1.6.2. Statistical summaries including sums, means, and squares of latent values were computed and visualized in a pair plot (see subsubsection A.7.2.1). However, these measures did not yield satisfactory results.

  Subsequently, **PCA** was applied to reduce the latent space to fewer dimensions, resulting in a two-dimensional **PCA** projection displayed in Figure 117. Unfortunately, this projection shows significant overlap between the classes, which complicates the use of a **SVM** for classification.

  Further analysis includes reducing the latent space to four dimensions, to try to improve visualization and interpretability. This is shown in Figure 123 (refer to subsubsection A.7.2.1) but unfortunately, this reduction did not result in a distinct separation between OK and NOK classes in the validation dataset. This outcome indicates potential limitations of advanced methods like **SVM** for classification on the test dataset, which will be explored in more detail later.

- Filtering

  This section examines the performance of various image filtering techniques (section A.3). These techniques will be evaluated quantitatively using **AUC** and **IoU** metrics. The **IoU** requirements define thresholds for converting **CAE** error values into binary masks. These thresholds, determined by applying each technique to OK sample errors in the validation dataset and using the 90th percentile as the cut-off, are detailed in Figure 118.

The efficacy of filtering techniques was evaluated using **AUC** and **IoU** metrics in comparison to the ground truth. The results are illustrated in Figure 119, showing a generally similar impact to **MSE** reconstruction loss. Only a handful of cases marginally surpass **MSE** in terms of **AUC**, such as Eroded, Median Filter, **NLM** color, and **NLM** gray, with the Erosion filter reaching a peak **AUC** of roughly 0.60 $(+0.03)$. In terms of **IoU**, almost all methods surpass **MSE**, with Dilation achieving the highest **IoU** of roughly 0.25 $(+0.13)$, with the exception of **NLM** color.

Given these findings, it is anticipated that during testing, the Erosion and Dilation techniques will be the most effective for optimizing **AUC** and **IoU** metrics, respectively, when filtering the error.

Additionally, visualizations of all applied filtering techniques are provided in subsubsection A.7.2.2 (see Figure 124 and Figure 125 for additional details).

- Clustering the error

This section outlines the error clustering methodology described in subsubsection 2.1.6.3, aimed at clearly delineating anomalies for easier fault detection. This method enhances the distinction between OK and NOK images by filtering noise, where ideally, OK images produce blank outputs and NOK images show prominent error clusters.

The effectiveness of these clustering techniques was evaluated using **AUC** and **IoU** metrics, as shown in Figure 120. The baseline **MSE** showed a increase in **AUC** to about 0.59 $(+0.02)$, while its **IoU** remained stable. When combined with filtering methods, these techniques generally match the **MSE**'s initial performance, except at the edges where they record 0.5 **AUC** and 0 **IoU**. Notably, approaches like eroded slightly outperform the baseline **MSE**, achiving a peak performance of 0.6 **AUC**. In regards to **IoU** a similar analysis can be done, with the eroded technique (the only one capable of outperforming the baseline) achieving the highest **IoU** of 0.13.

The visualization of clustering techniques in the validation dataset can be found in subsubsection A.7.2.3 (Figure 126 and Figure 127). These figures display the original error image alongside the clustered error regions, with ground truth included for comparison.

A.7.2 Additional Validation Results

The pairplot in Figure 121 illustrates the relationships among different evaluation metrics. Notably, using **SSIM** reveals some separation between the realistic NOKs and other labels, though no clear distinction is seen between OK and other NOK labels. Other metric combinations show significant overlap across all labels, complicating any separation. These observations suggest that training an **SVM** may not enhance classification performance.

A.7.2.1 Latent Space Visualization

To assess the latent dimension, various statistical metrics were calculated and visualized in a pair plot, as shown in Figure 122. This visualization helps elucidate the relationships and distributions within the latent space. It reveals a notable overlap among the labels from the validation set across all metrics.

The four components of the **PCA**-transformed latent space are displayed in Figure 123. Initially, it was hypothesized that a complex latent space, not easily reducible to 2D, might offer more profound insights in 4D. Yet, the plot reveals significant overlap among all labels, suggesting that the latent space transformation fails to effectively distinguish between the classes in the validation dataset.

A.7.2.2 Filtering Techniques

Detailed visualizations of all filtering techniques applied to the validation dataset are illustrated in Figure 124 and Figure 125. These figures feature the original error image alongside masks created using a 90th percentile threshold for each filtering method.

Analysis of the images shows that all filtering techniques show very noisy results making it difficult to clearly identify where the anomaly is. The only exceptions are perhaps radial distortion where the anomaly can be seen using the eroded method although in an extremely noisy environment. Furthermore there is also the patch method were the anomaly can be seen very faintly on the **MSE** loss. In short, **VAE** architecture presents not perceptible, hard to interpret results.

A.7.2.3 Clustering Techniques

The figures Figure 132 and Figure 133 show the original error image after applying a clustering technique, alongside the mask derived from the inferred validation threshold and the ground truth anomaly mask.

Clustering is intended to reduce noise interference, but the results indicate the opposite. All error maps generated using various filtering techniques (except for Edges, which is empty) exhibit very noisy results. The only exception is the **NLM** color method, which shows a very faint anomaly on the **MSE** loss. Moreover, no clear anomalies can be identified in the original **MSE** loss due to the high noise level.

A.7.3 Additional Test Results

The **MSE** loss histogram for the test dataset is shown in Figure 80. Consistent with previous studies, the flip label stands out clearly against the OK images due to its distinct transformation. However, the overlap of other NOK labels with OK labels complicates classification, resulting in an **AUC** of about 0.62, similar to the previous methodology.
The pairplot in Figure 129 illustrates the relationships between different metrics that assess model performance on the test dataset. Unfortunately there are no clear separations in any combination between OK and NOK labels.

A.7.3.1 Filtering Techniques

The plots Figure 130 and Figure 131 showcase the original error image after applying a filtering technique, the derived mask from the validation threshold, and the ground truth anomaly mask.
Analyzing these masks, the same conclusion as the one done in the validation dataset can be done. No method either it using filtering techniques or not can clearly identify the anomalies due mostly to the noise present in the images.

A.7.3.2 Clustering Techniques

The Figure 132 and Figure 133 showcase the use of clustering techniques. Similar to the filtering techniques, the performance is very poor due to high levels of noise.

A.7.3.3 Best Performance Achieved

This section details the application of the **BCE** metric across various post-processing techniques. The following figures illustrate the test dataset's performance with different filtering and clustering strategies.
The impact of filtering techniques on the test dataset is shown in Figure 134. The Eroded technique achieves the highest **AUC** score of approximately 0.76. For the **IoU** metric, the

highest score, approximately 0.09, is recorded with the dilation techniques.

Similarly, Figure 135 showcases how various clustering strategies influence the test dataset's performance. Unfortunately, the classification power is very similar to the flip of a coin, achieving a maximum performance of 0.53 **AUC** on the Eroded technique and an **IoU** of 0.05.

In conclusion, when employing the **BCE** metric, the best outcomes are observed with the **NLM** gray and dilation techniques from the filtering approach, achieving **AUC** scores of up to 0.76 and **IoU** scores of 0.09, respectively.

A.7.4 Other VAE-specific approaches

Figure 136 displays histograms of the latent parameters, where each image is represented by an averaged and normalized latent parameter vector. The histograms reveal significant overlap among all data classes, complicating differentiation. The **AUC** values for z_μ, z_γ, and z were calculated as 0.55, 0.49, and 0.21, respectively, indicating performance close to random chance.

A 2D **PCA** projection of the latent parameters is shown in Figure 137. Unfortunately, the results indicate no clear separation between the OK and NOK classes, making the successful application of an **SVM** for classification unlikely.

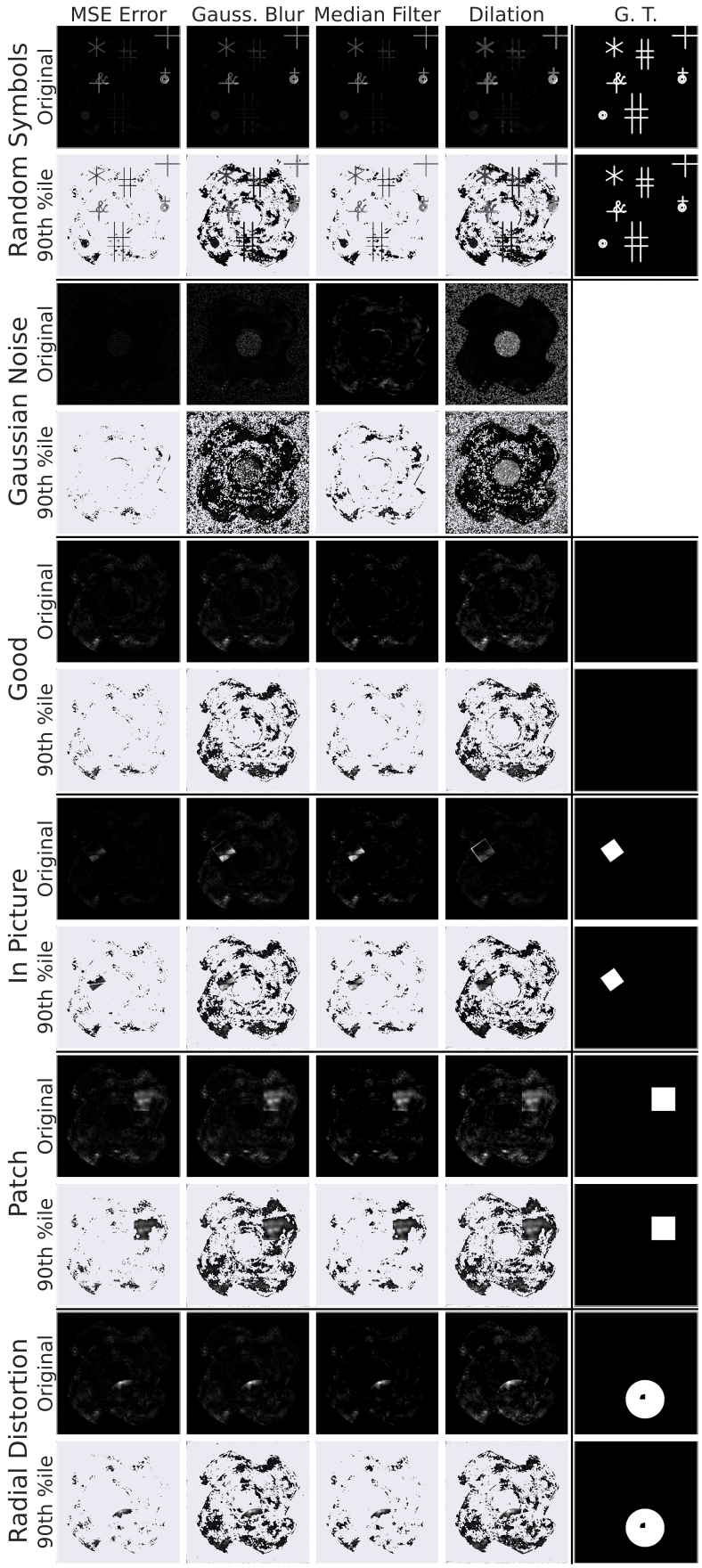


Figure 76: First set of filtering techniques. This includes the original error image, Gaussian blur, median filter, and dilation. Additionally, the 90th percentile mask and the ground truth mask are presented. The application of these filters demonstrates varying degrees of noise reduction and feature preservation.

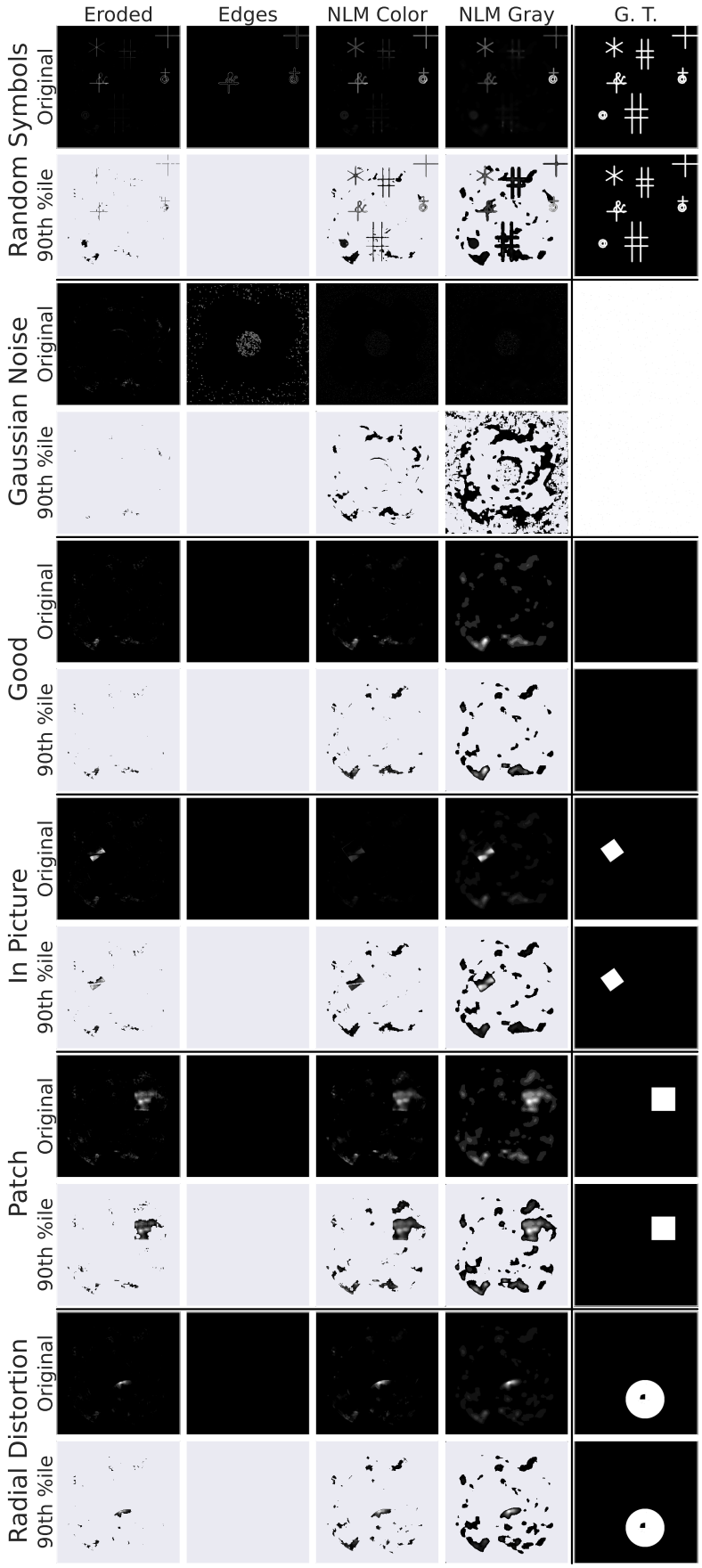


Figure 77: Second set of filtering techniques. This includes erosion, edge detection, and Non-Local Means (NLM) techniques. The figure also presents the 90th percentile mask and the ground truth mask. These techniques provide different approaches to noise reduction and edge preservation, further enhancing the analysis.

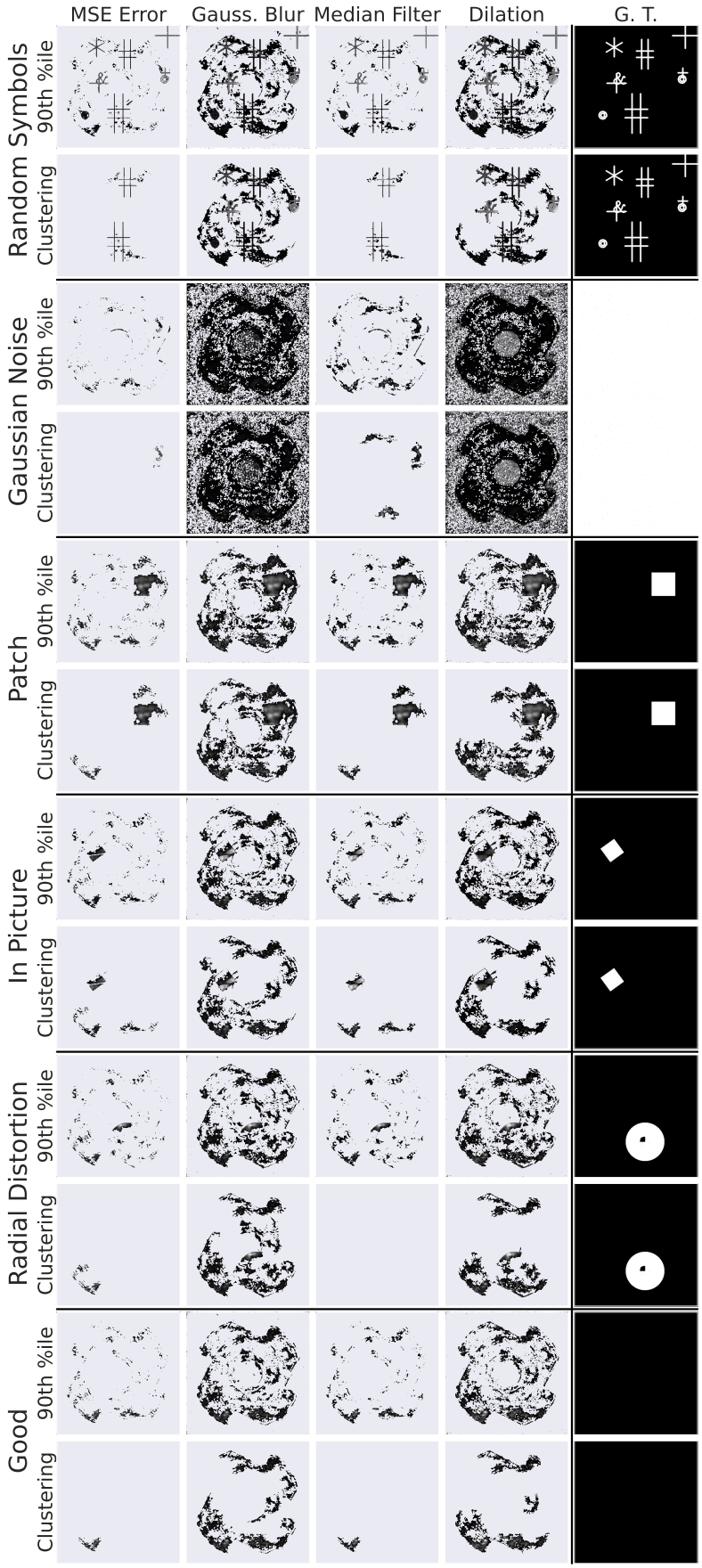


Figure 78: First batch of clustering filtering techniques. It includes the original error image, gaussian blur, median filter, and dilation. The 90th percentile mask and the ground truth mask are also shown.

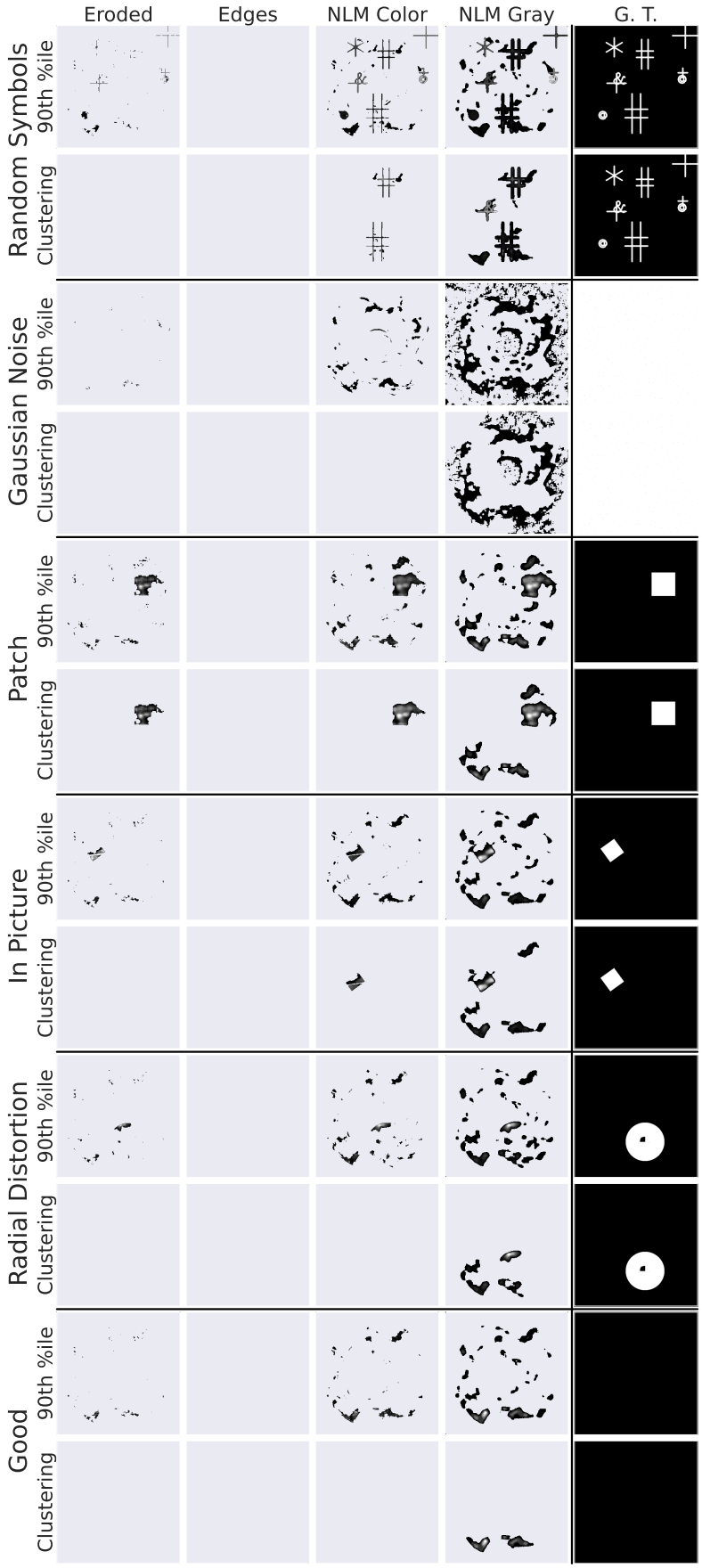


Figure 79: Second batch of clustering filtering techniques. It includes the erosion, edge detection and NLM techniques. The 90th percentile mask and the ground truth mask are also shown.

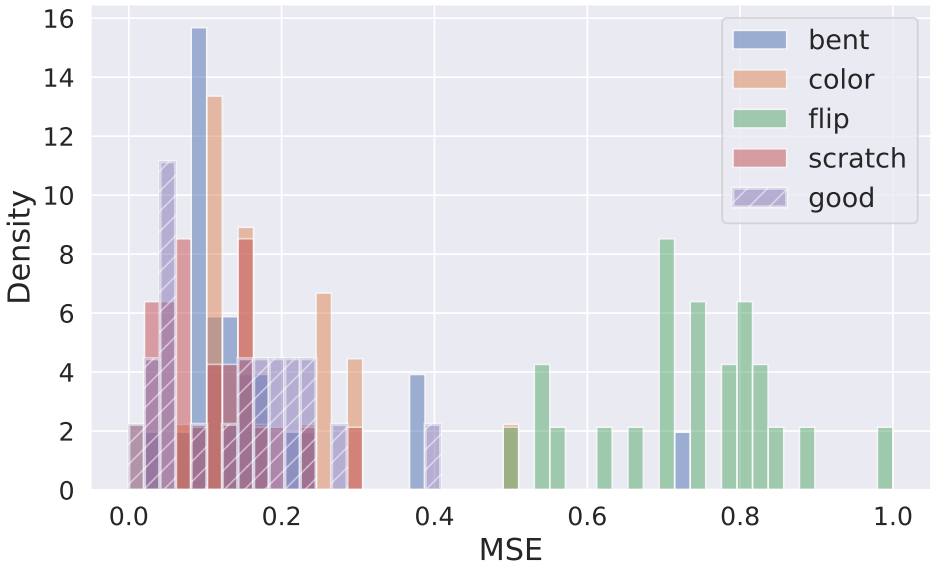


Figure 80: Histogram of the MSE reconstruction loss for the test dataset using the **CAE** architecture.

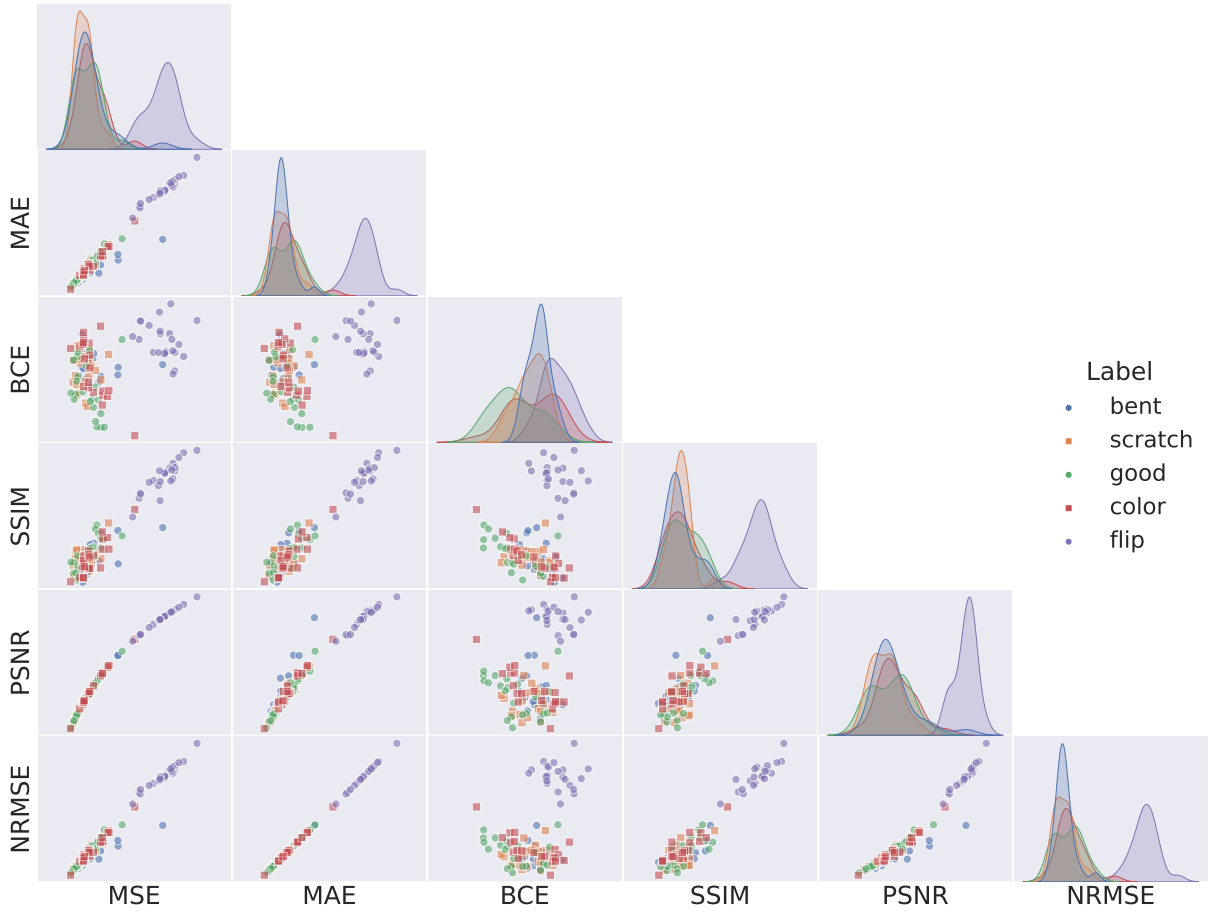


Figure 81: Pairplot of the validation dataset using the **CAE** architecture.

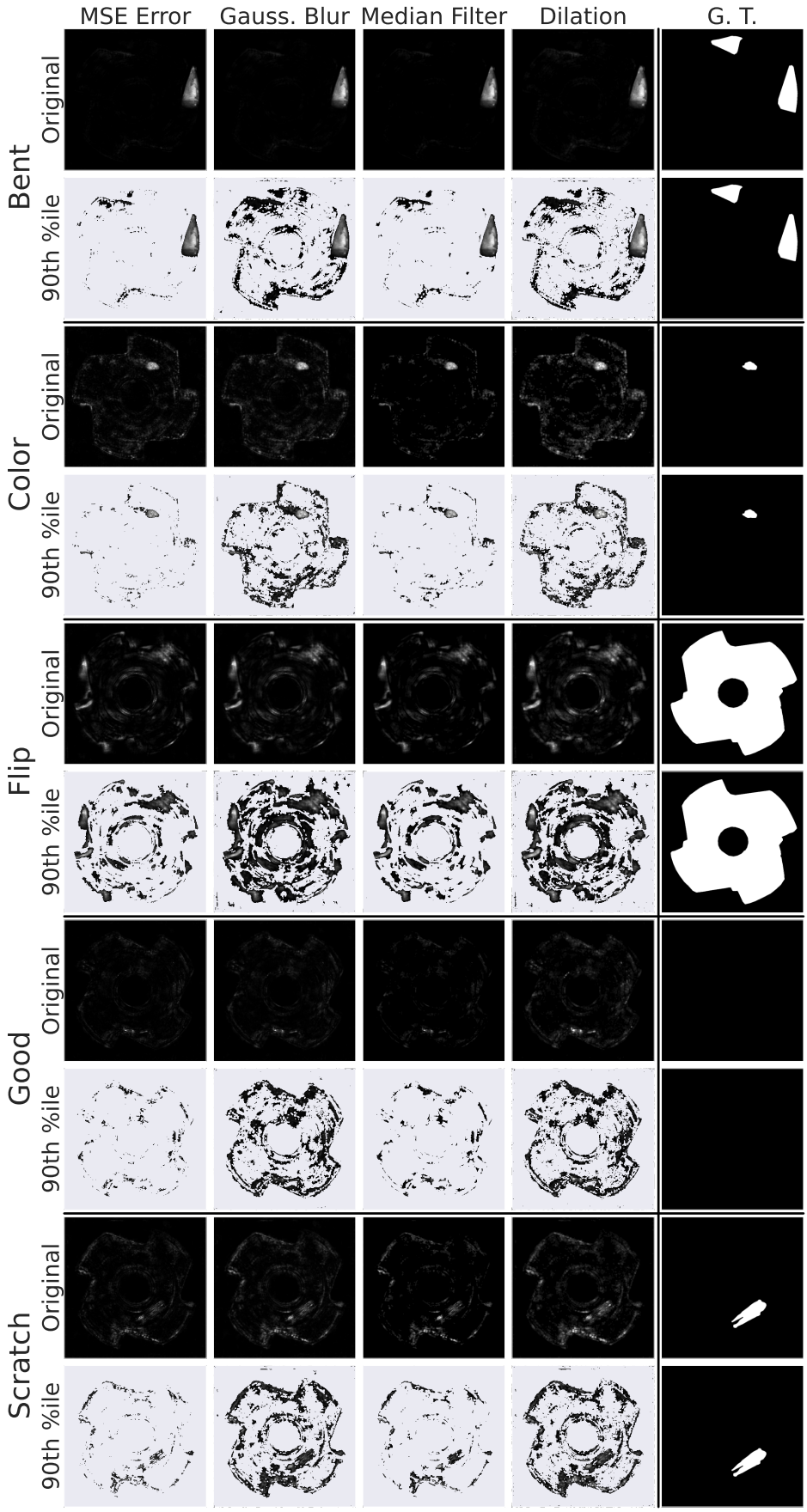


Figure 82: First batch of filtering techniques. It includes the original error image, gaussian blur, median filter, and dilation. The 90th percentile mask and the ground truth mask are also shown.

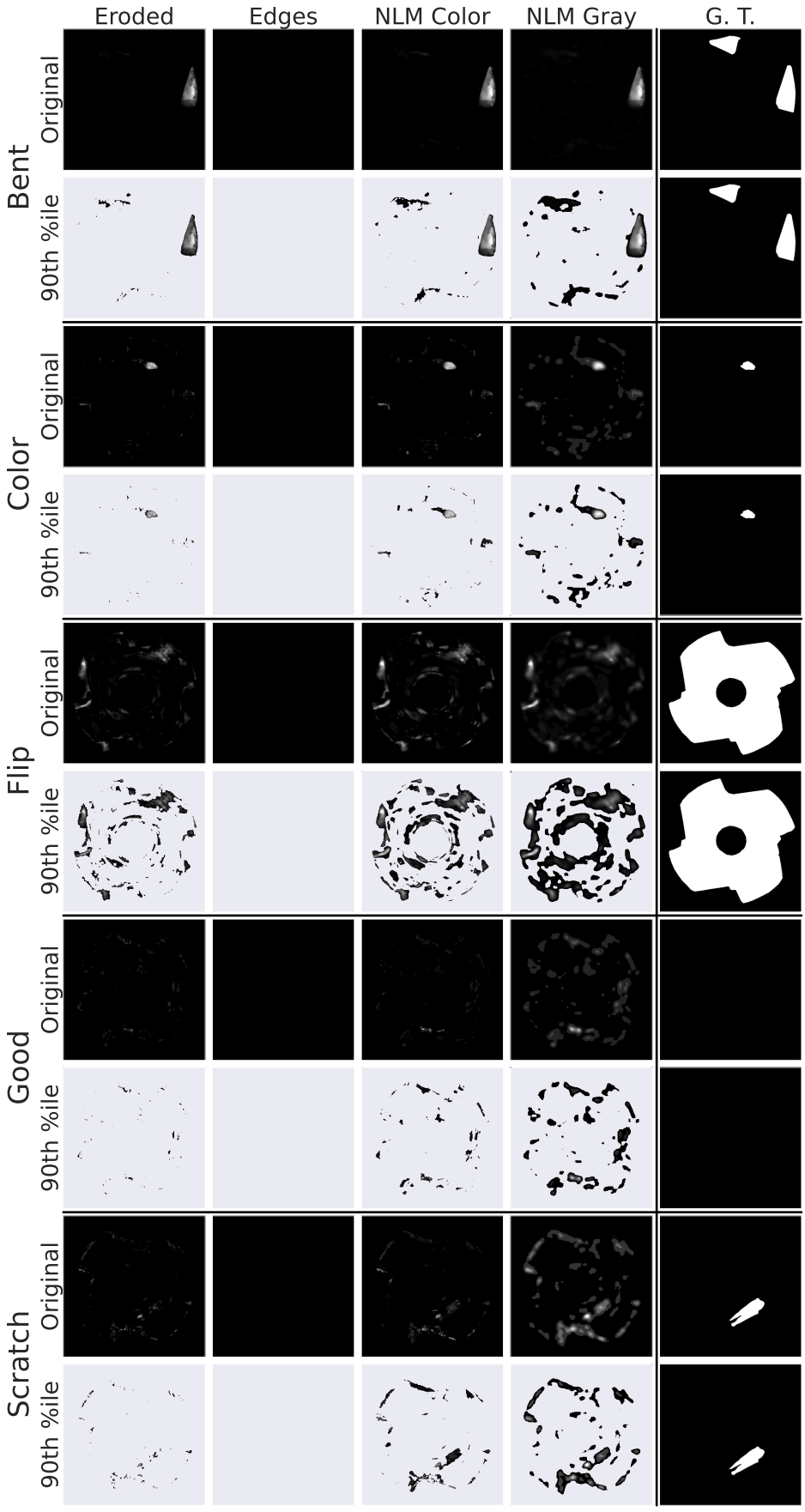


Figure 83: Second batch of filtering techniques. It includes the erosion, edge detection and NLM techniques. The 90th percentile mask and the ground truth mask are also shown.

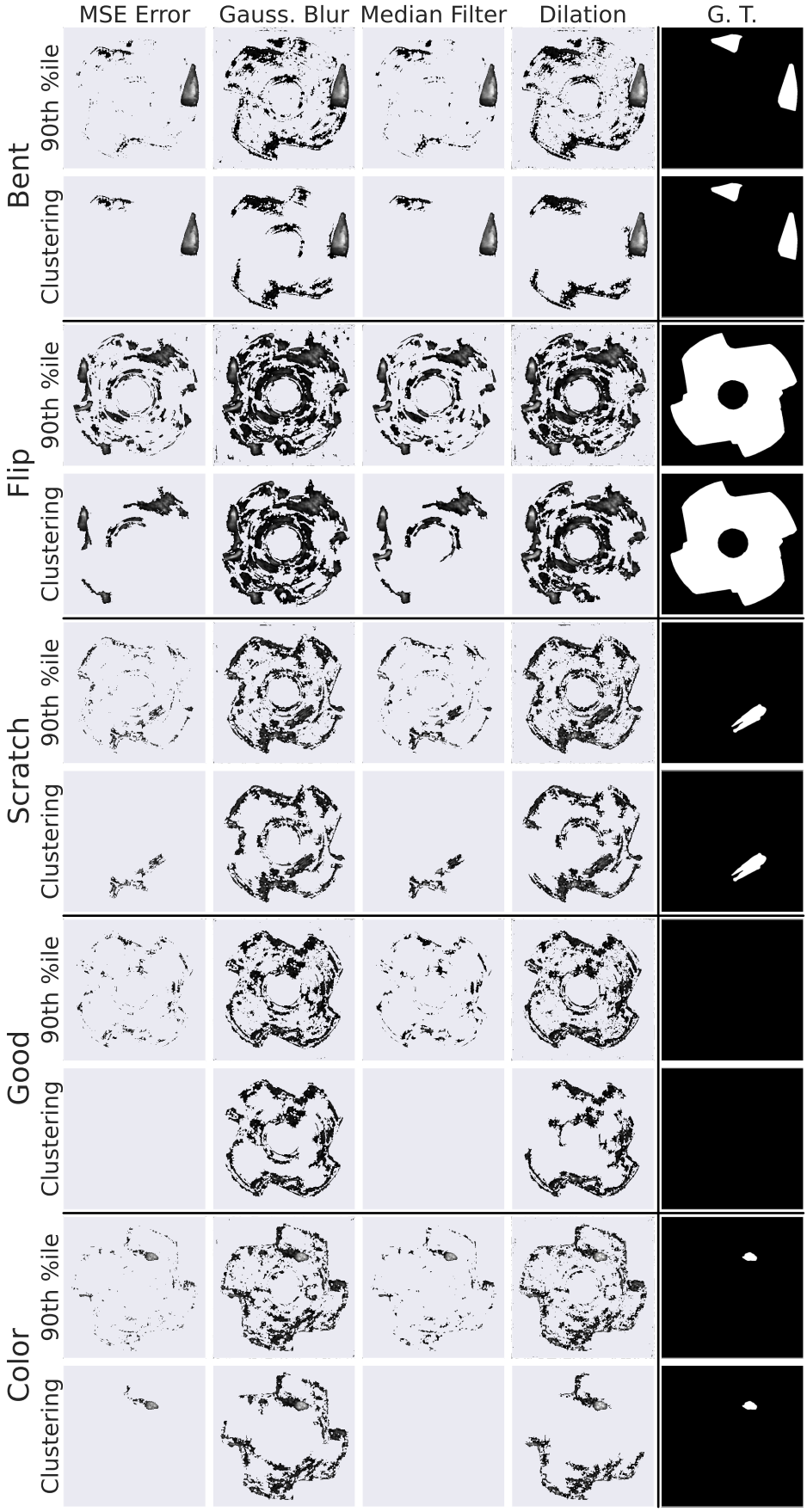


Figure 84: First batch of clustering filtering techniques. It includes the original error image, gaussian blur, median filter, and dilation. The 90th percentile mask and the ground truth mask are also shown.

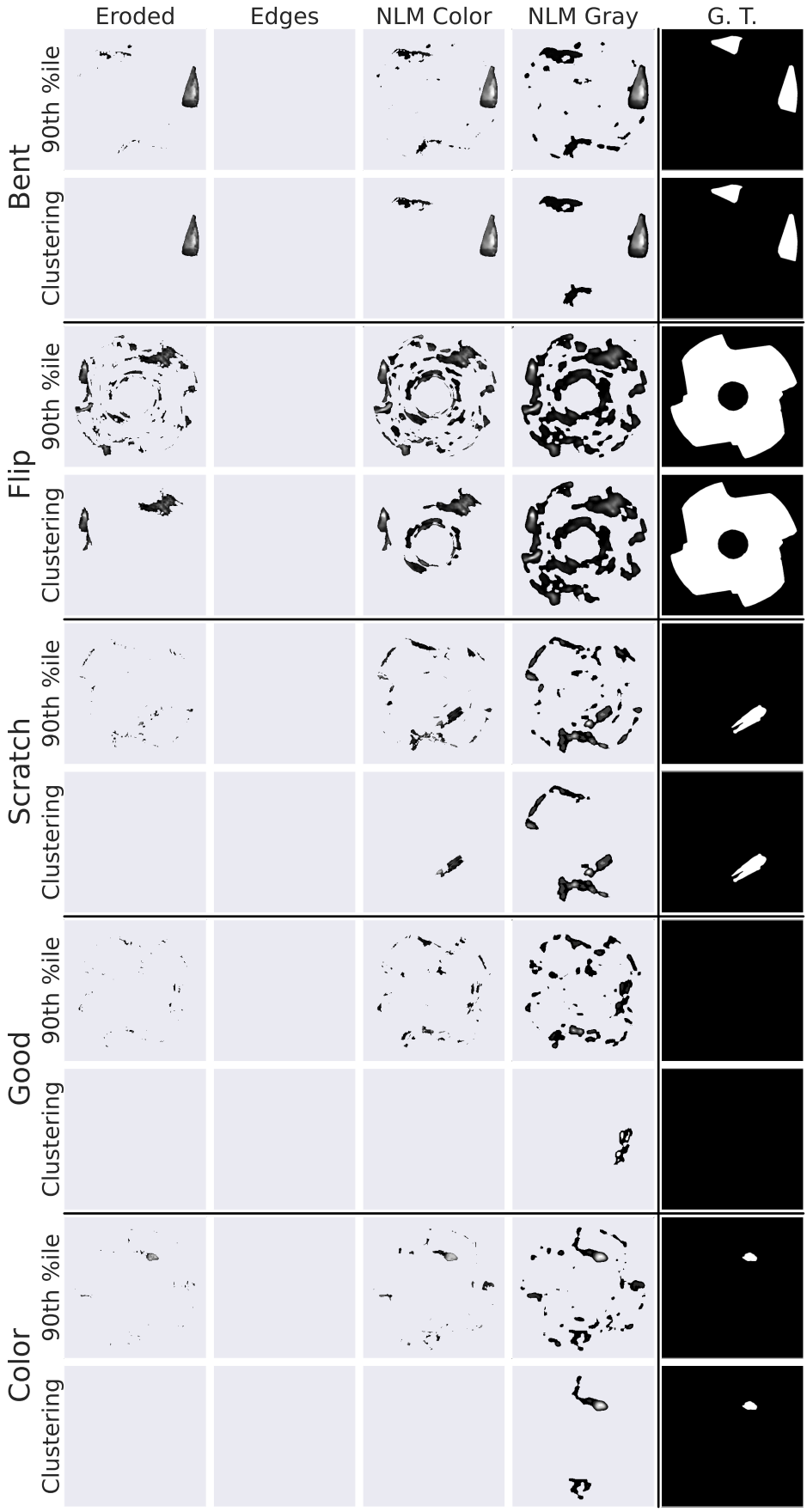


Figure 85: Second batch of clustering filtering techniques. It includes the erosion, edge detection and NLM techniques. The 90th percentile mask and the ground truth mask are also shown.

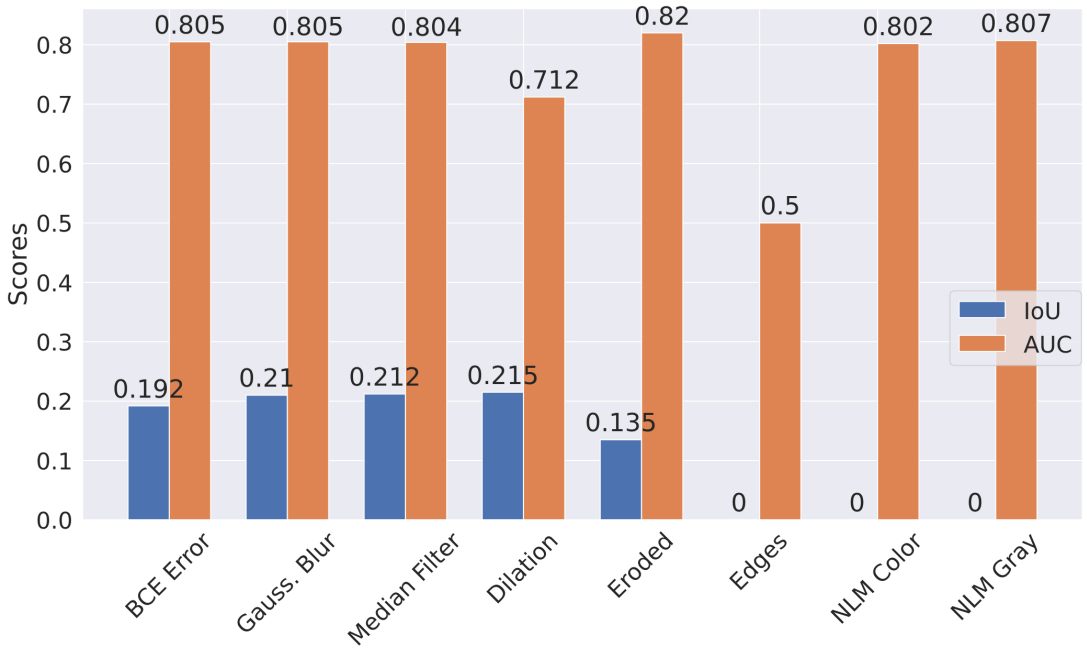


Figure 86: **BCE** performance on the test dataset using different filtering techniques.

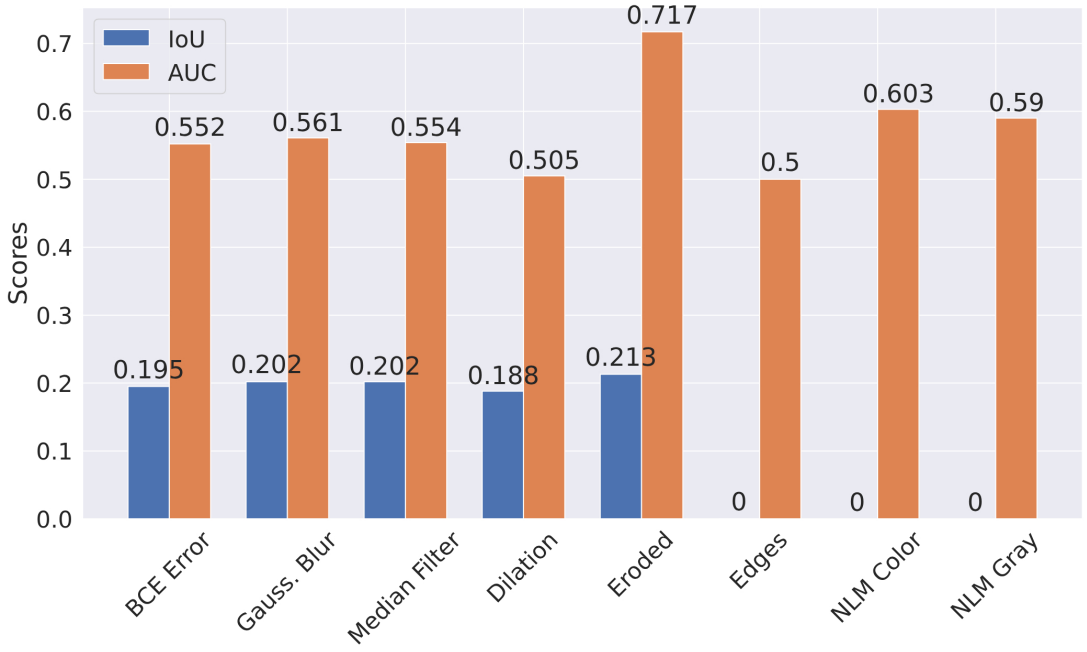


Figure 87: **BCE** performance on the test dataset using diferent clustering techniques.

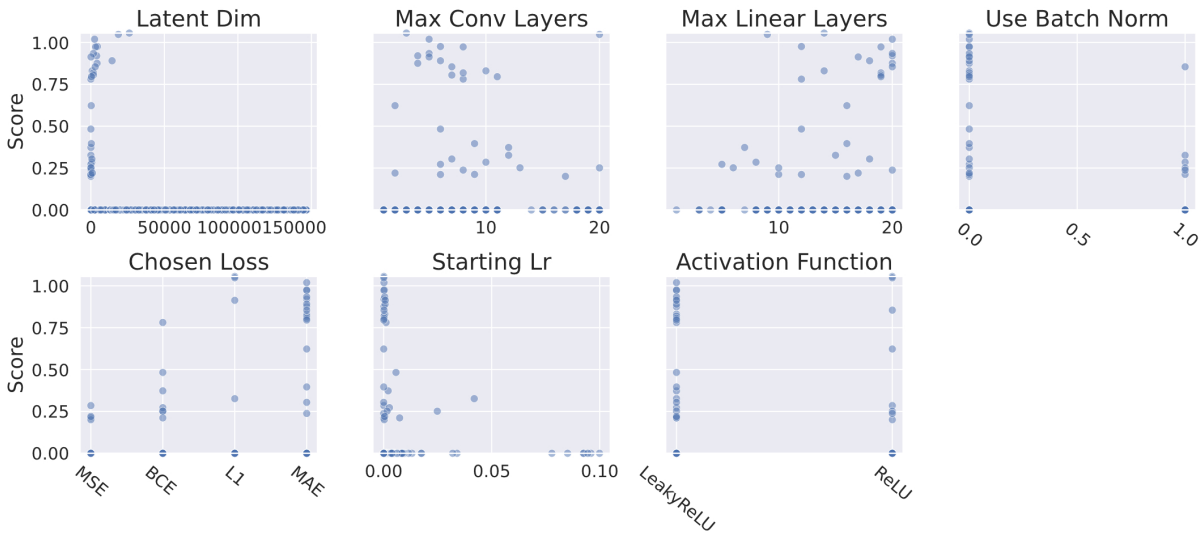


Figure 88: Slice plot of the non-layer specific **HP** for the improved **CAE** architecture. Note that some **HP** have a bolean value of 1 (True) and 0 (False).

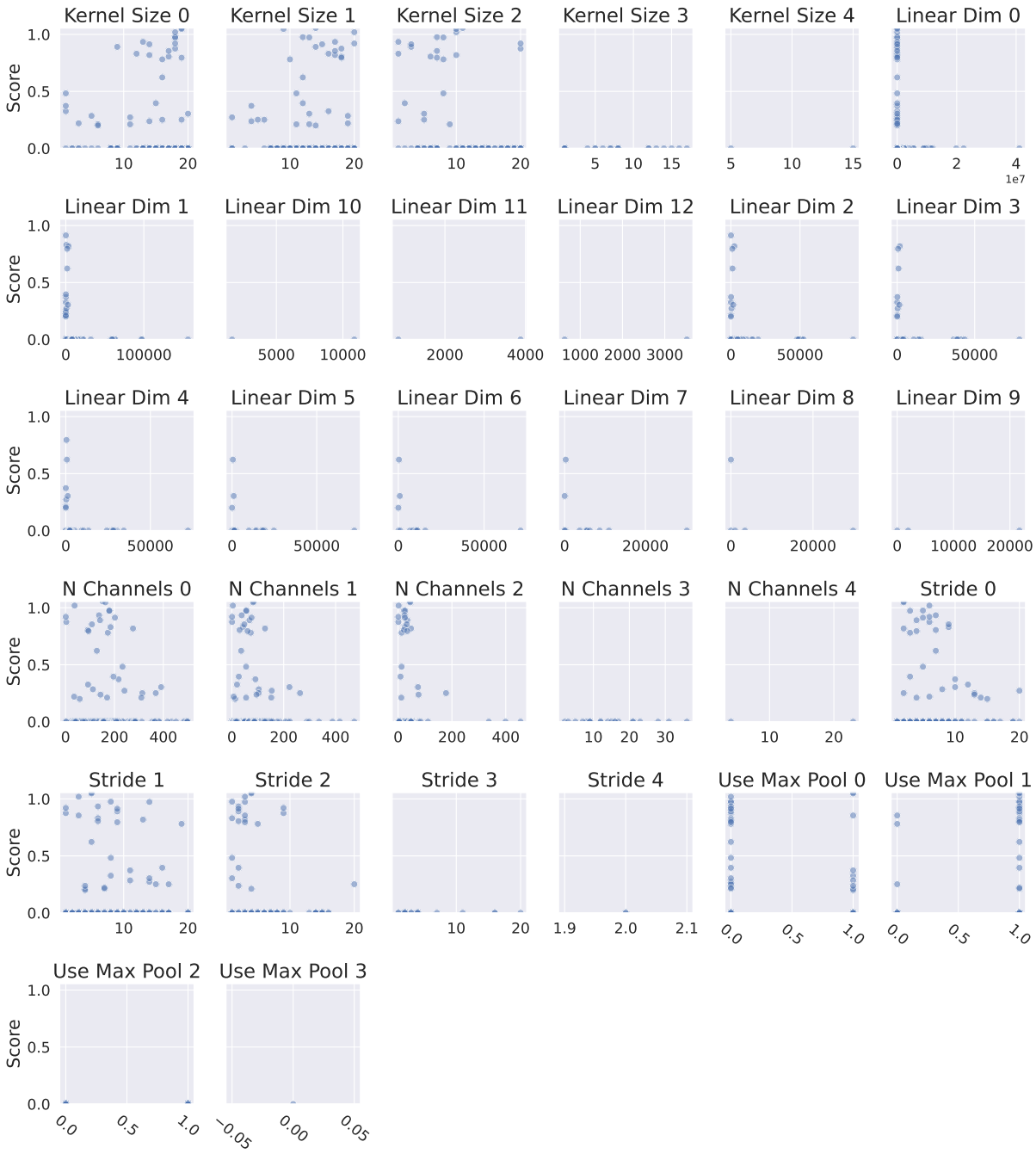


Figure 89: Slice plot of the layer specific **HP** for the improved **CAE** architecture. The number on the **HP** name refers to the layer number.

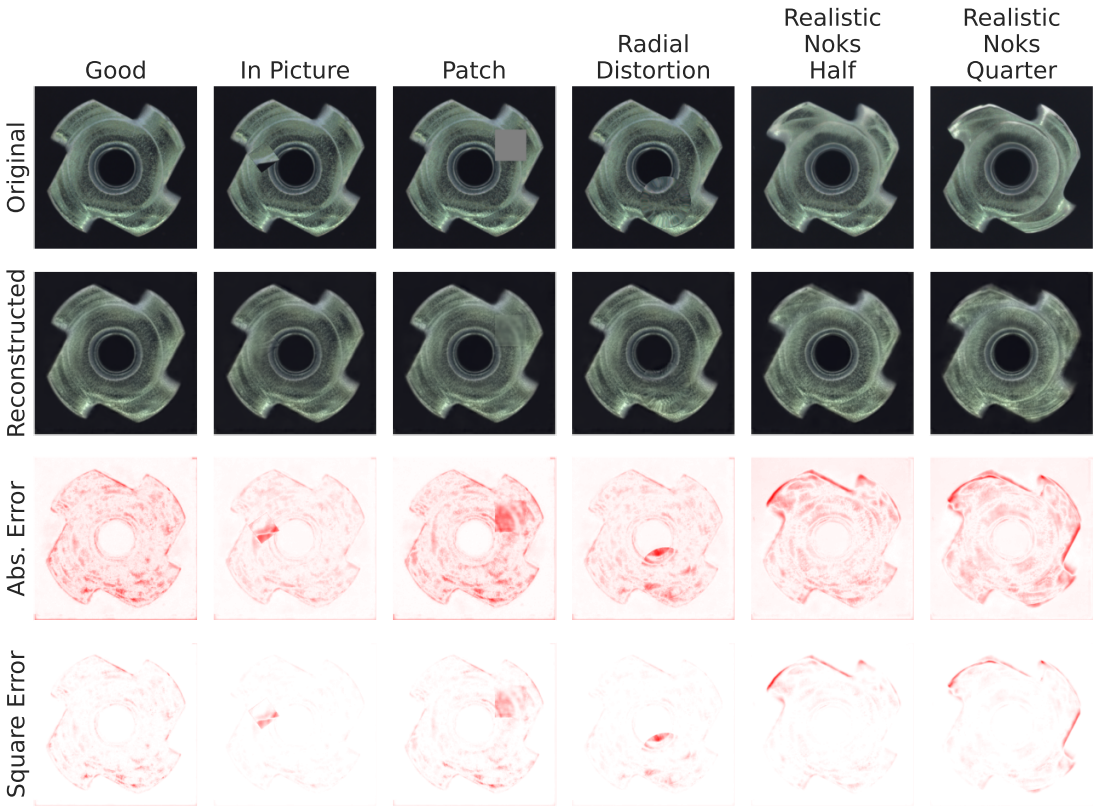


Figure 90: Reconstruction of the validation dataset using the improved **CAE** architecture. On each column it can be seen the different labels of the validation dataset. The first row shows the original images, the second row the reconstructed images, the third row the absolute error map and the last row the square error.

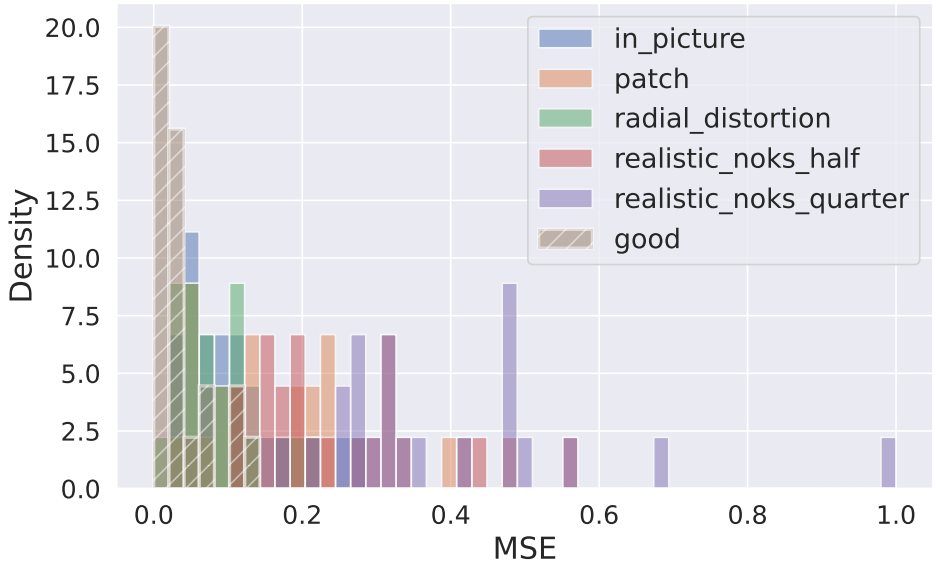


Figure 91: Histogram of the **MSE** reconstruction loss for the validation dataset using the optimized **CAE** architecture.

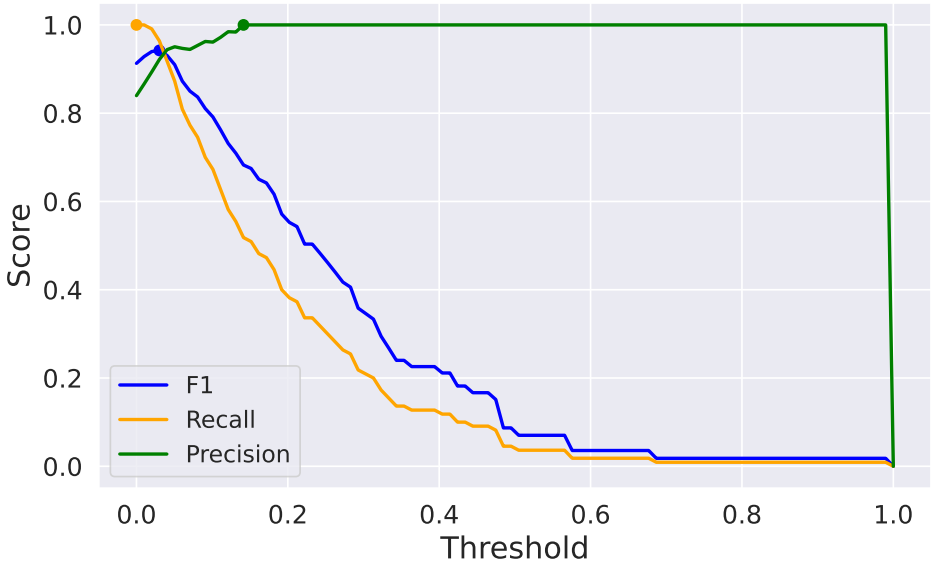


Figure 92: Evaluation of various thresholds using the baseline approach, showing how different metrics vary with each threshold setting on the validation dataset.



Figure 93: **PCA** projection of the latent space, demonstrating dimensionality reduction to two principal components.

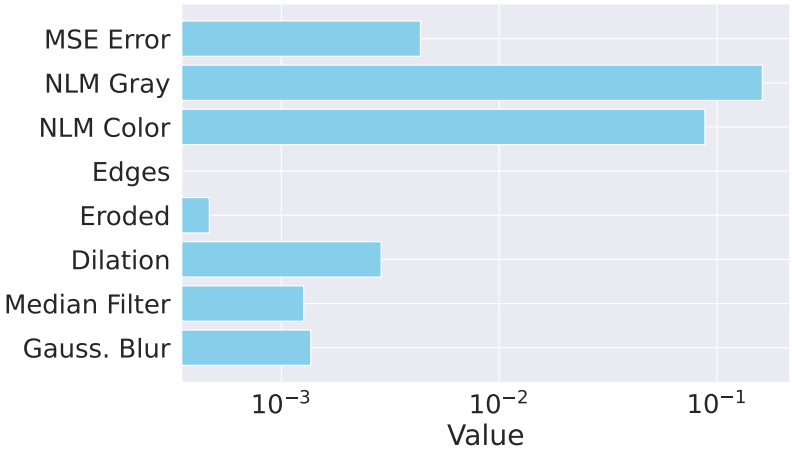


Figure 94: Thresholds for the filtering techniques using the 90th percentile of the OK-error values. Note that "Edges" (Edge Detection) has a *NaN* value since it already outputs a binary image.

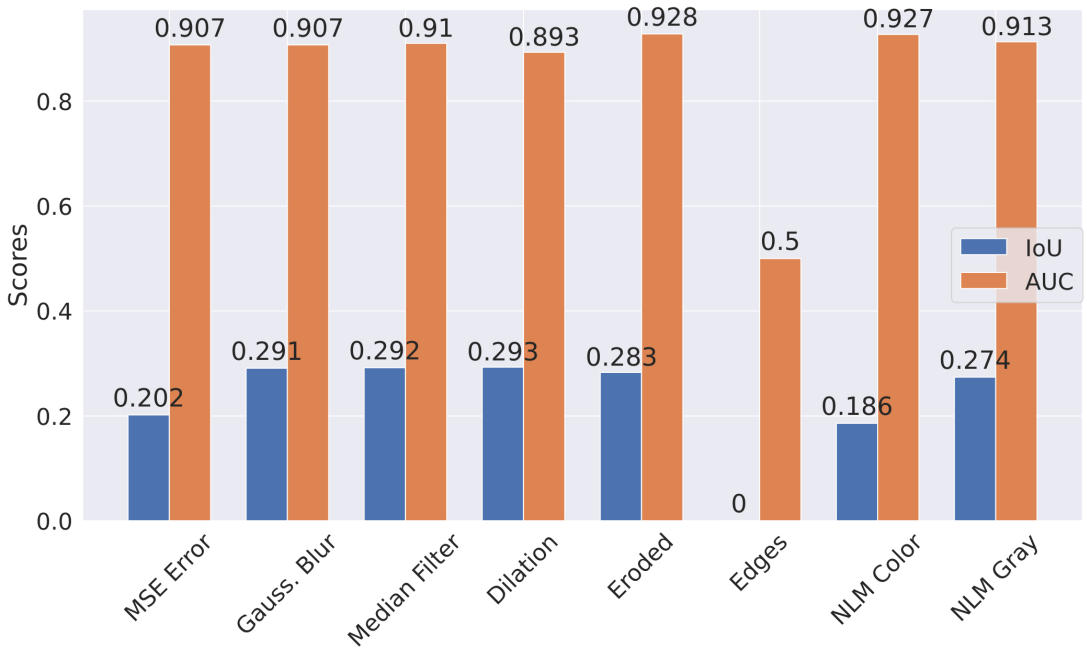


Figure 95: Comparative analysis of **AUC** and **IoU** metrics for various filtering techniques against ground truth labels and true error masks.

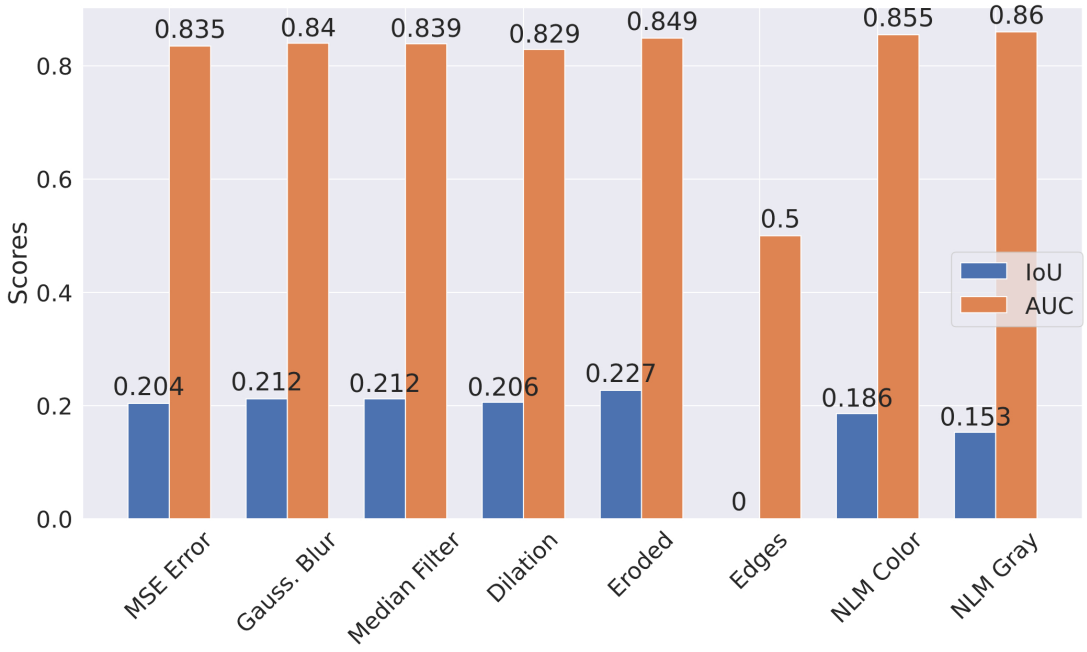


Figure 96: Evaluation of clustering performance using **AUC** and **IoU** metrics compared with ground truth labels and true error masks, illustrating the effectiveness of various clustering techniques.

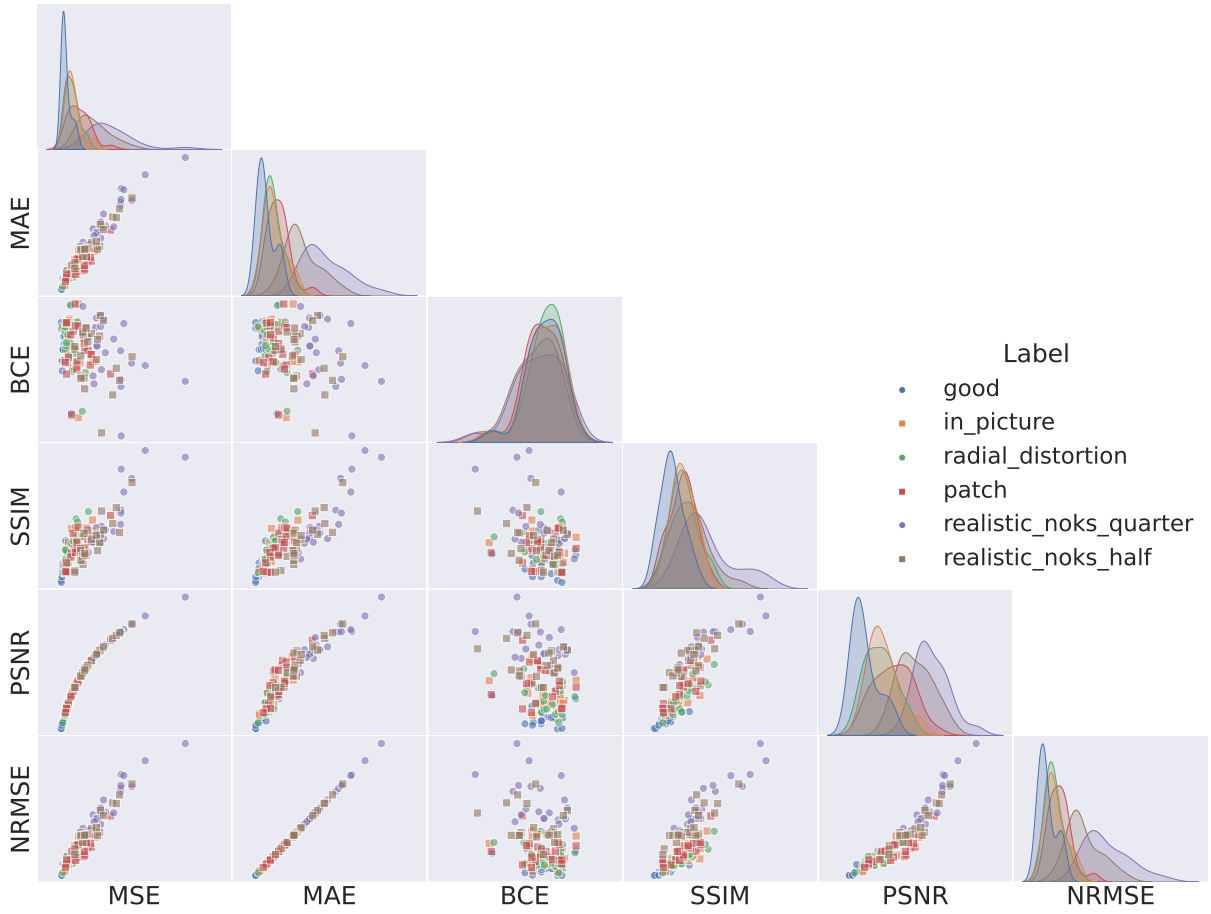


Figure 97: Pairplot of different metrics used to assess the reconstruction error of the validation dataset using the improved **CAE** architecture. The metrics include **MAE**, **BCE**, **SSIM**, **PSNR**, and **NRMSE**.

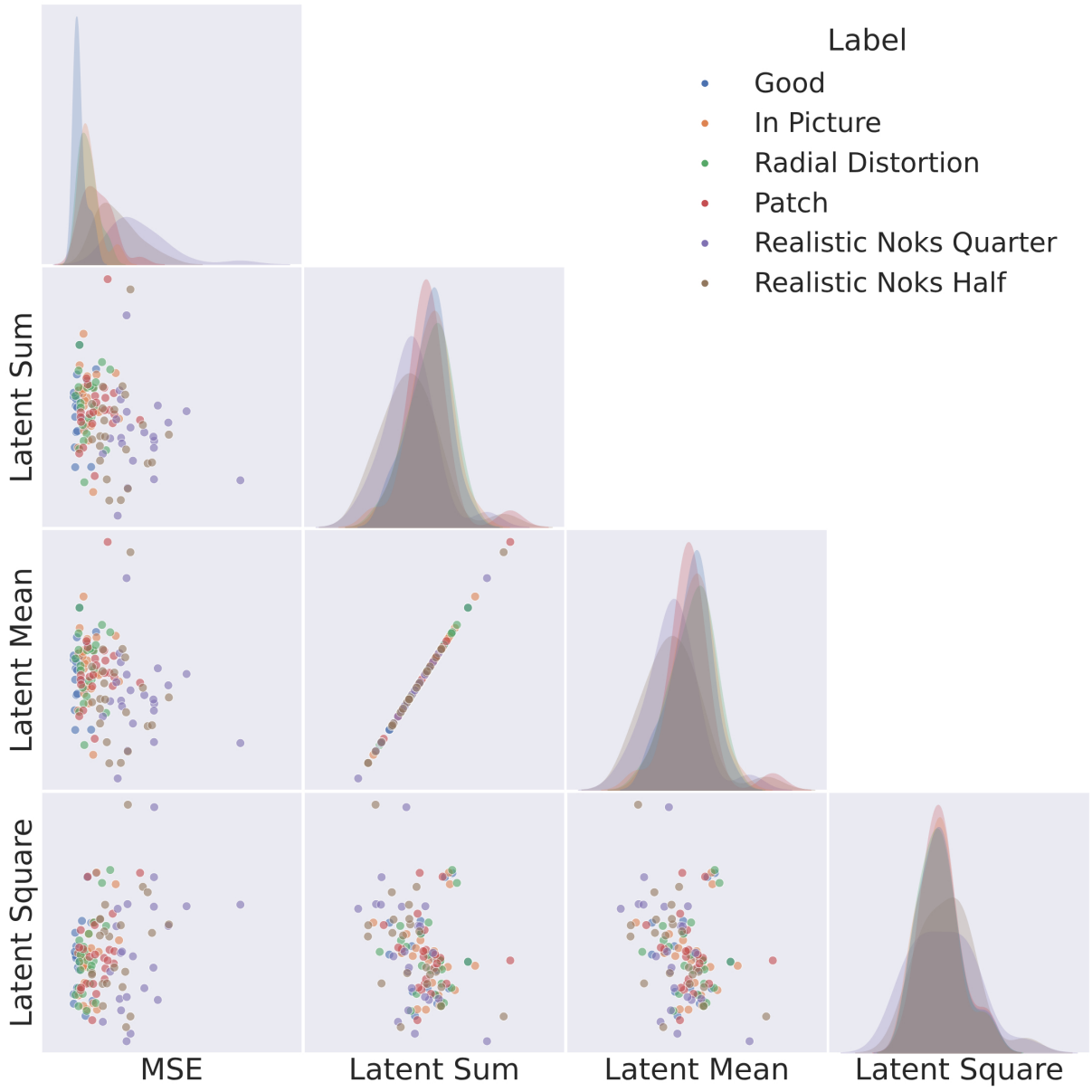


Figure 98: Pairplot visualization of the **MSE** reconstruction loss and the latent space statistical summaries.

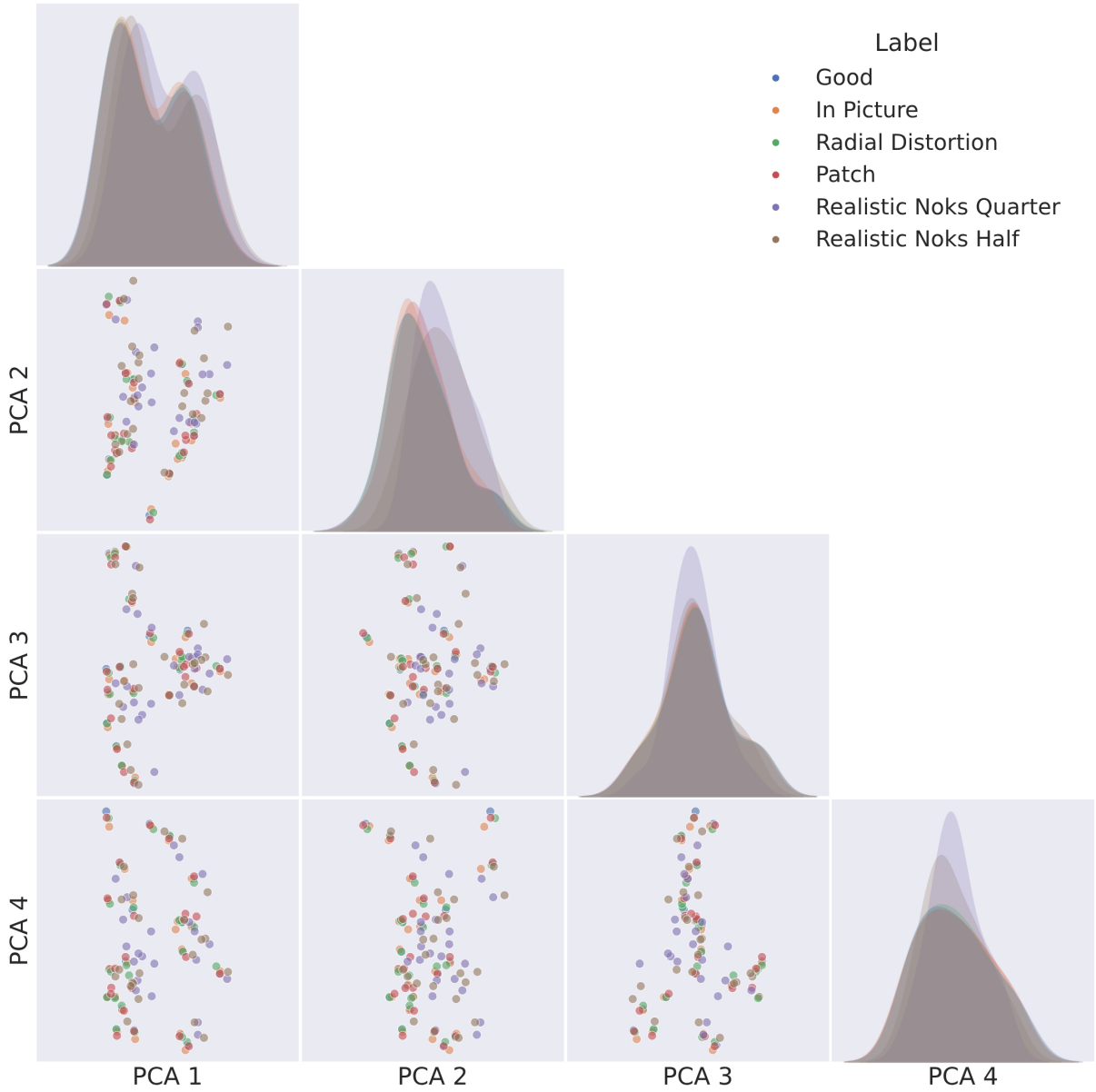


Figure 99: Pairplot of the **PCA**-transformed latent space

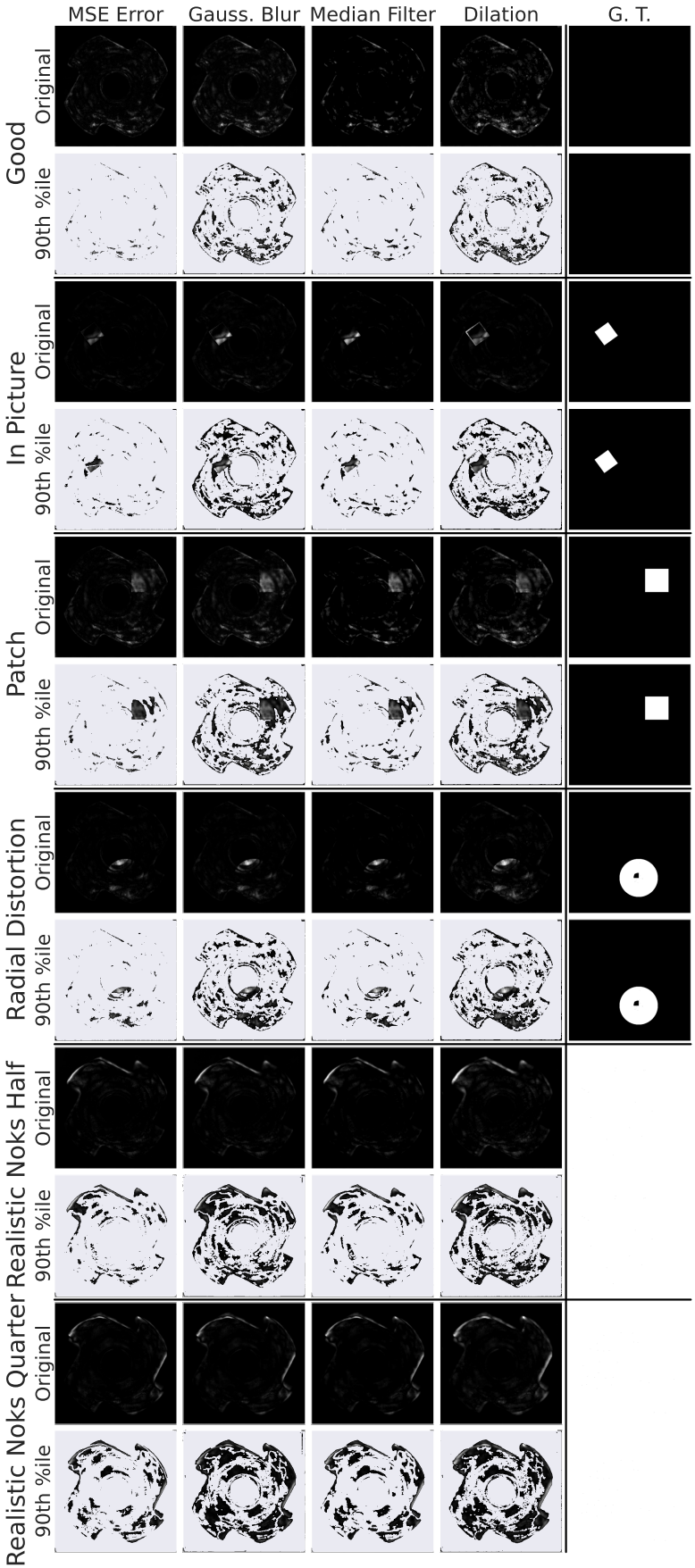


Figure 100: First batch of filtering techniques. It includes the original error image, gaussian blur, median filter, and dilation. The 90th percentile mask and the ground truth mask are also shown.

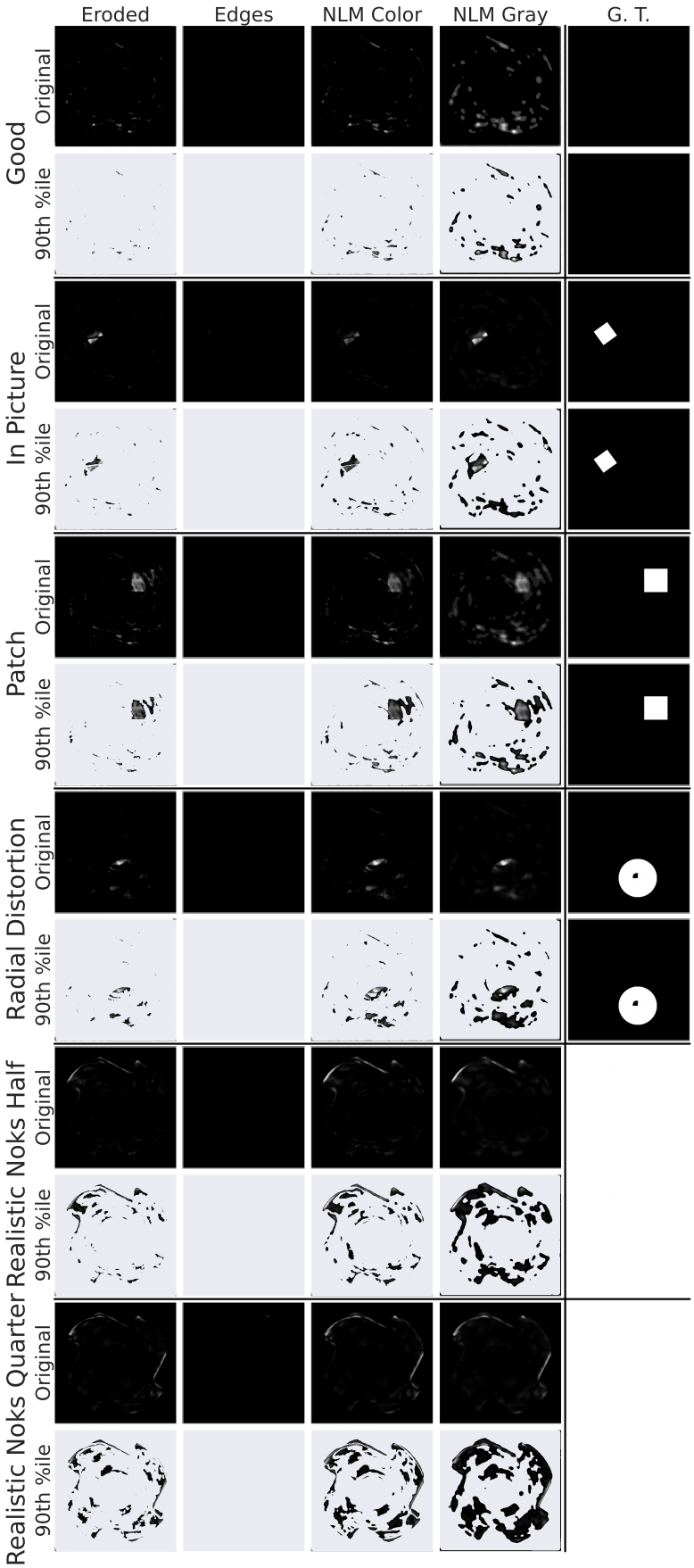


Figure 101: Second batch of filtering techniques. It includes the erosion, edge detection and **NLM** techniques. The 90th percentile mask and the ground truth mask are also shown.

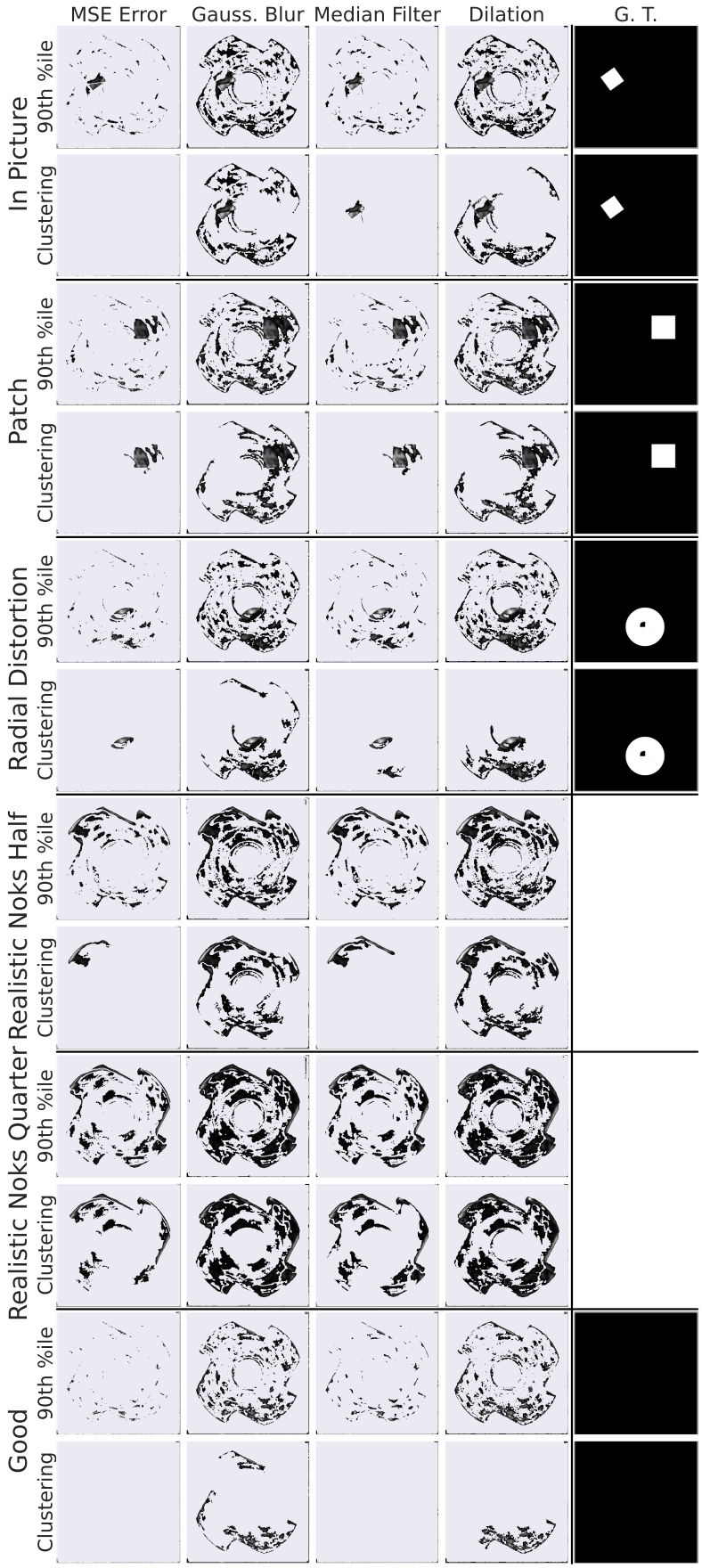


Figure 102: First batch of clustering filtering techniques. It includes the original error image, gaussian blur, median filter, and dilation. The 90th percentile mask and the ground truth mask are also shown.

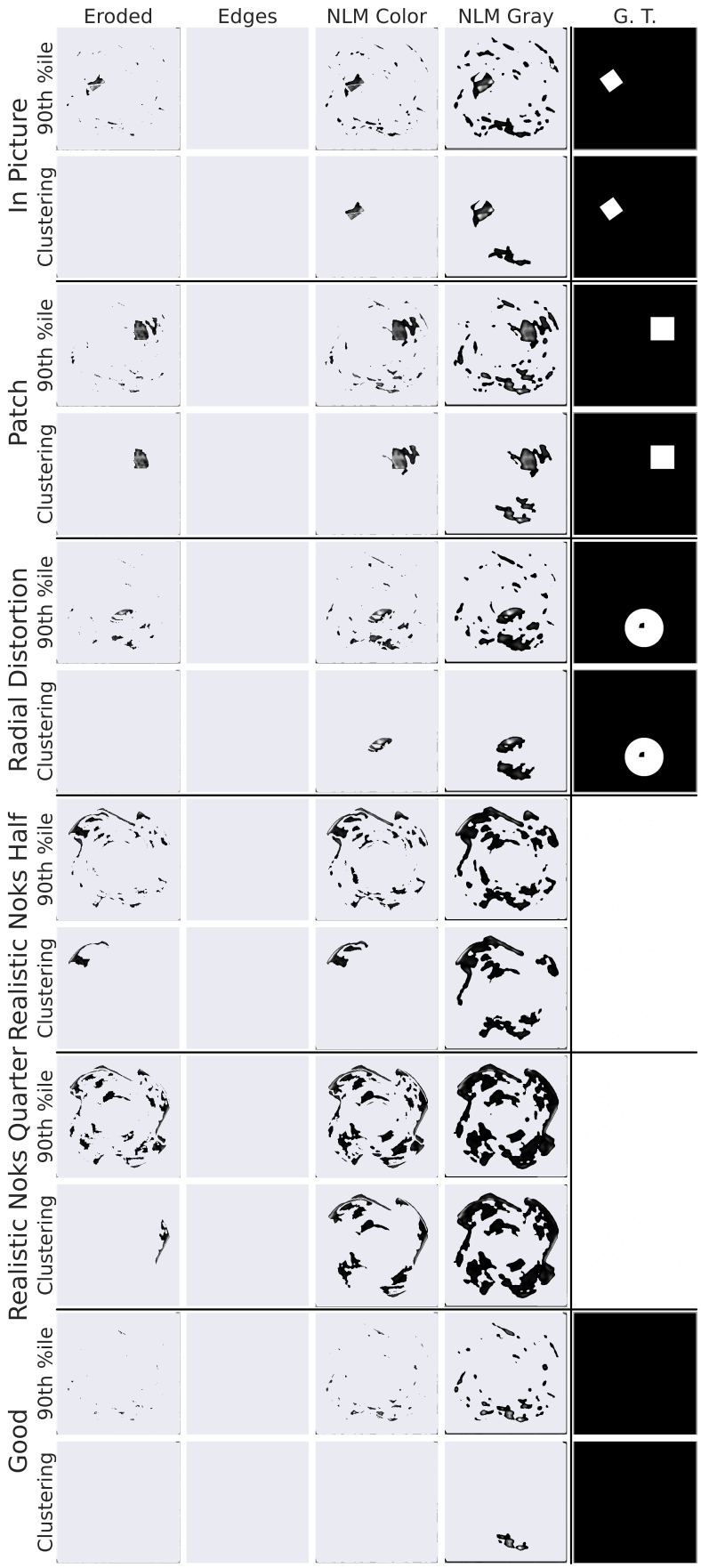


Figure 103: Second batch of clustering filtering techniques. It includes the erosion, edge detection and **NLM** techniques. The 90th percentile mask and the ground truth mask are also shown.

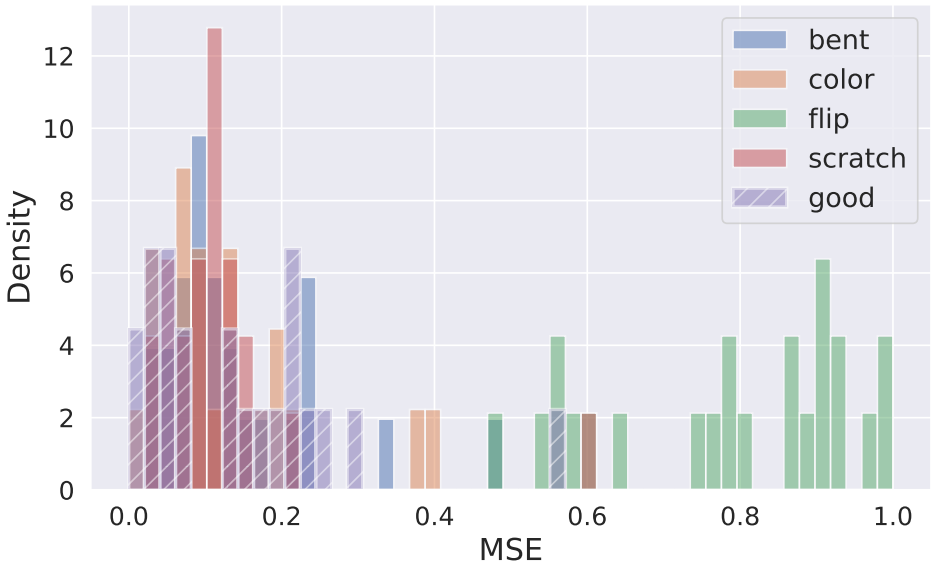


Figure 104: Histogram of the **MSE** reconstruction loss for the test dataset using the improved **CAE** architecture.

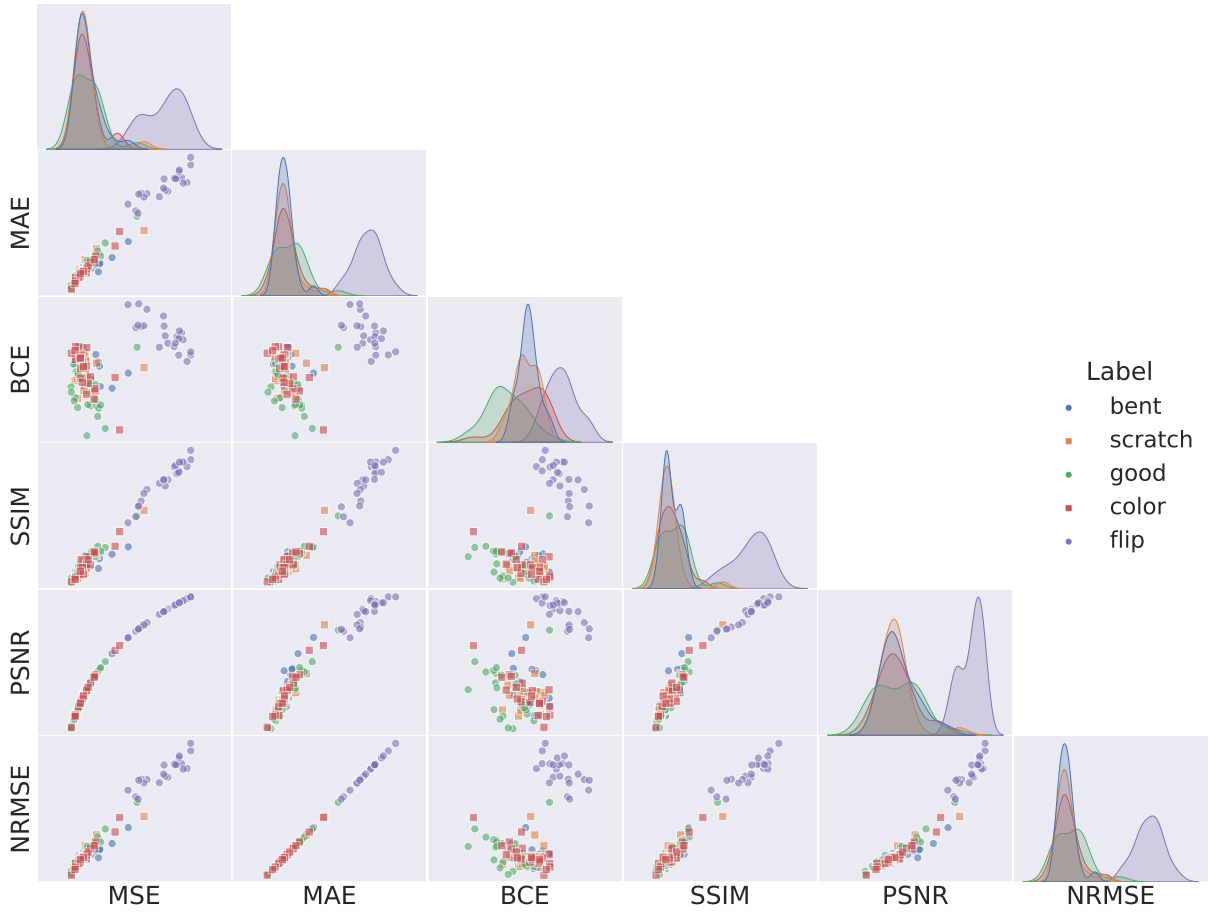


Figure 105: Pairplot of the validation dataset using the improved **CAE** architecture.

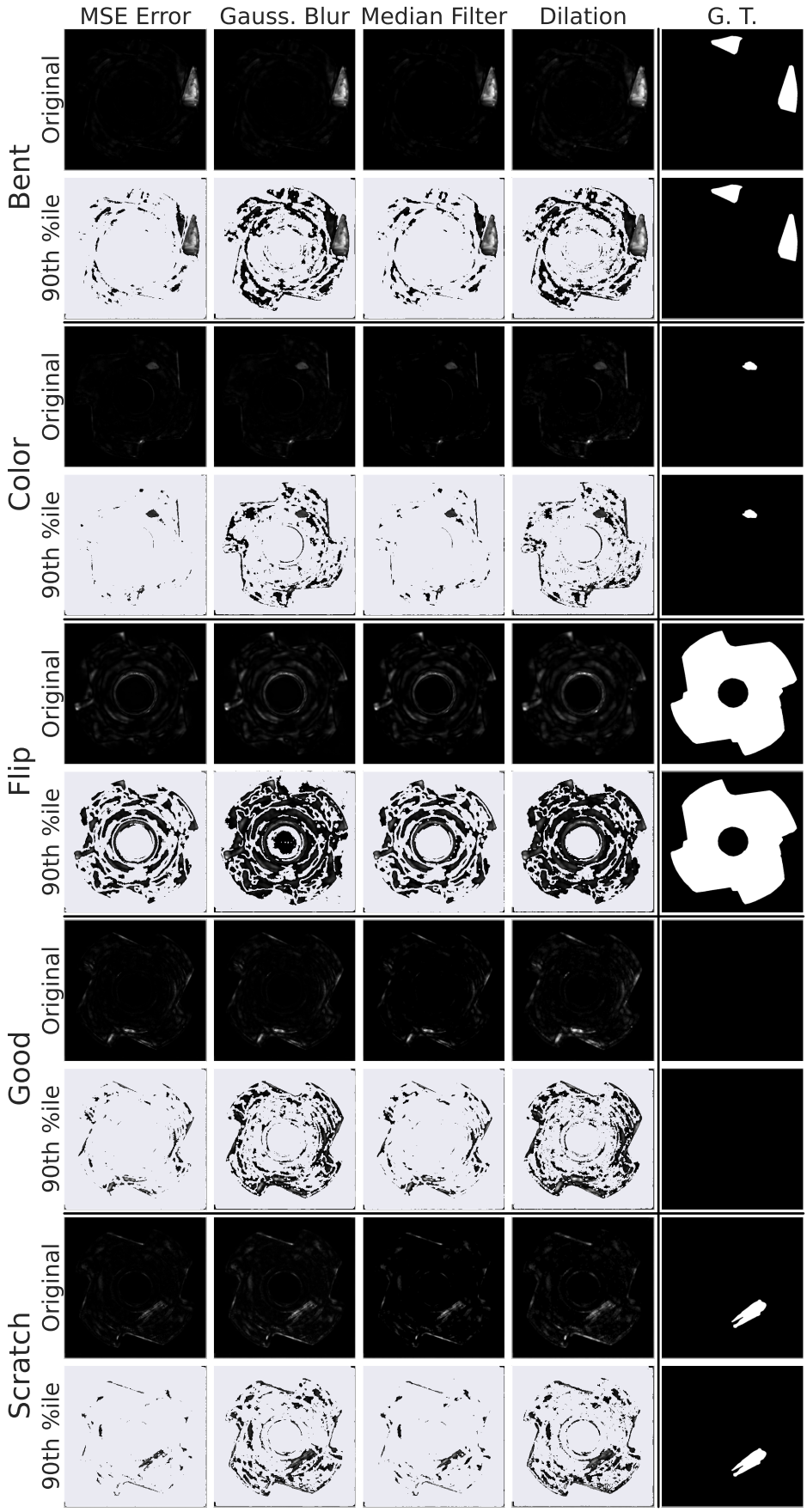


Figure 106: First batch of filtering techniques. It includes the original error image, gaussian blur, median filter, and dilation. The 90th percentile mask and the ground truth mask are also shown.

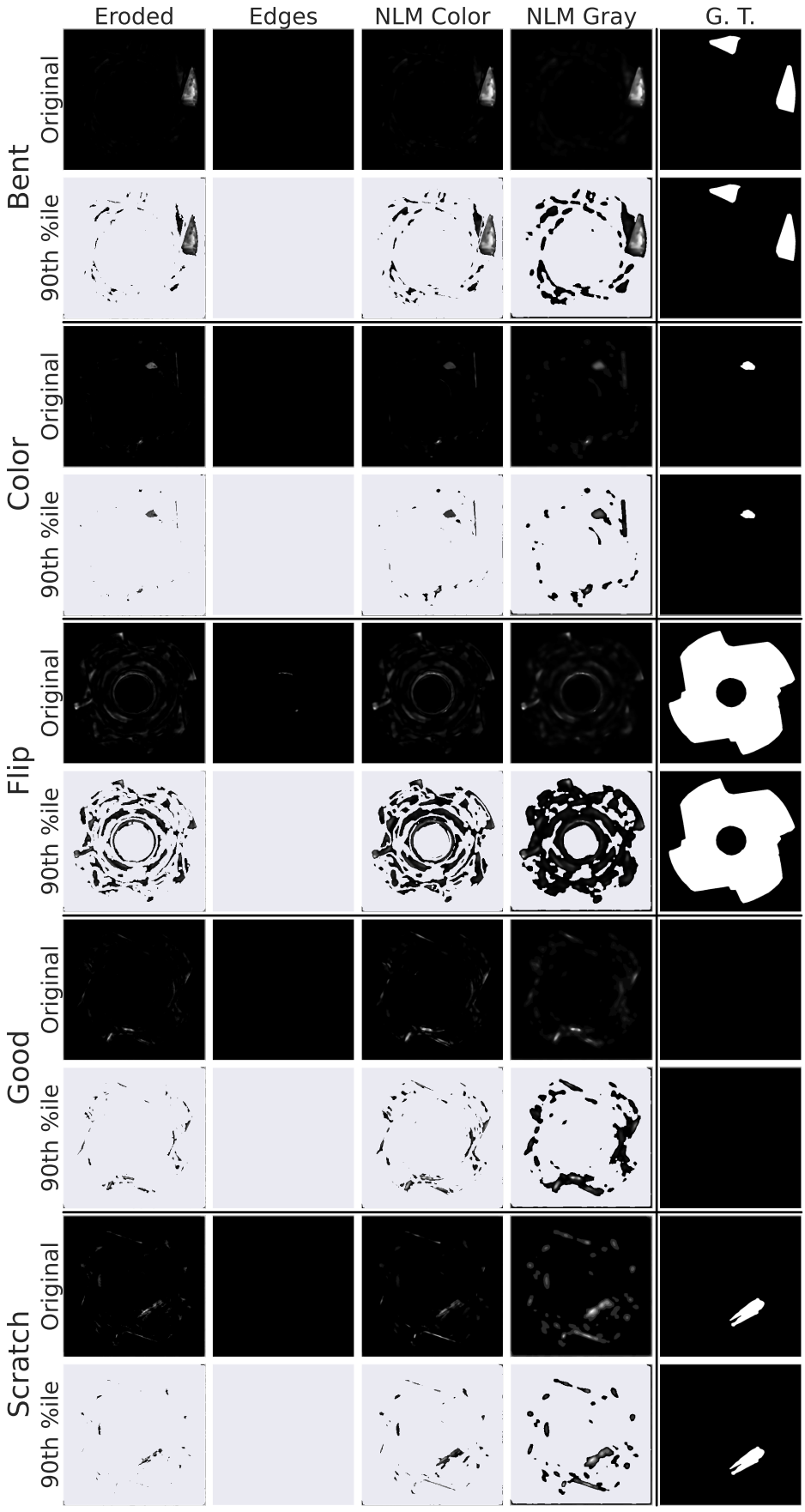


Figure 107: Second batch of filtering techniques. It includes the erosion, edge detection and **NLM** techniques. The 90th percentile mask and the ground truth mask are also shown.

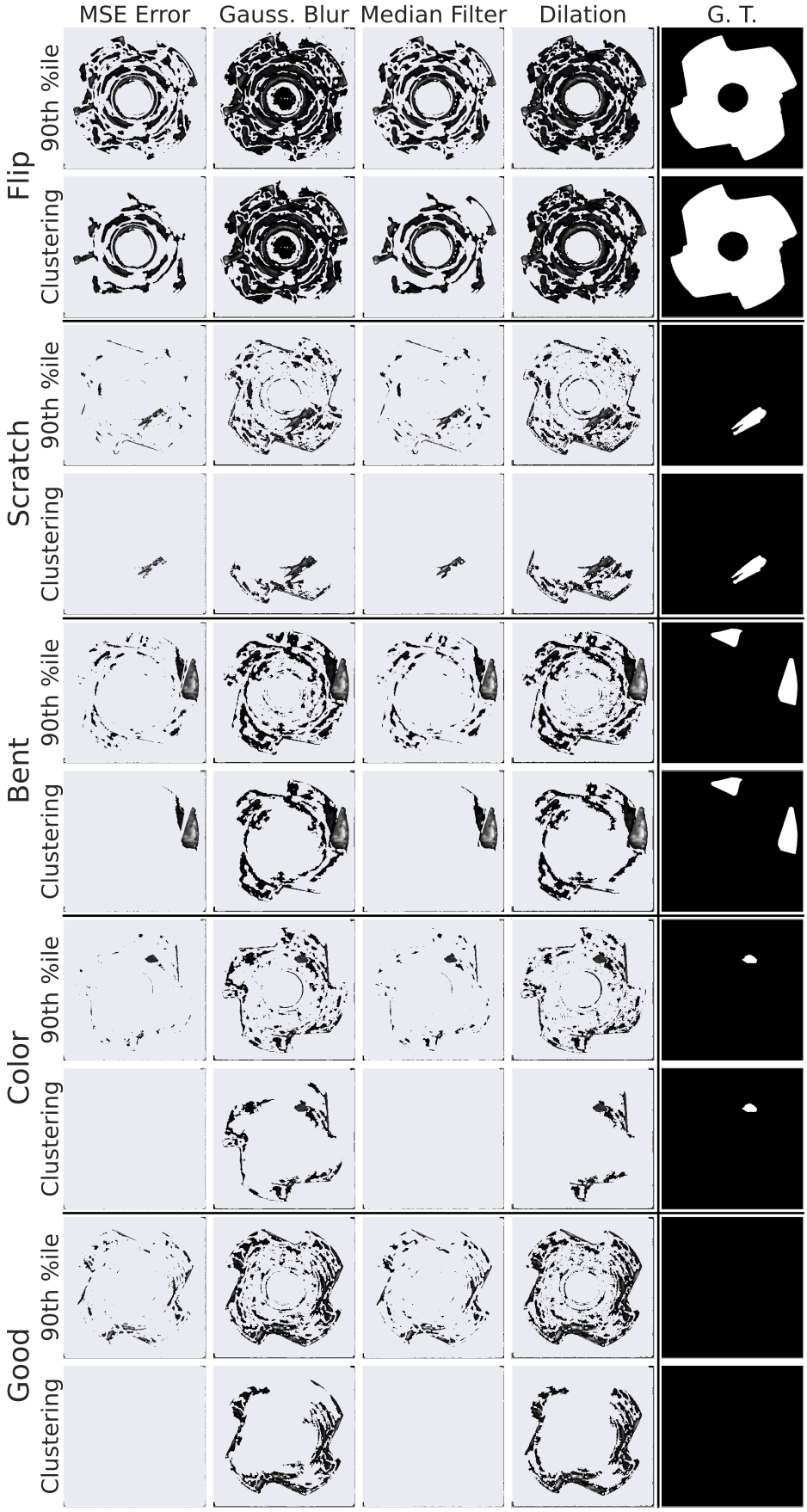


Figure 108: First batch of clustering filtering techniques. It includes the original error image, gaussian blur, median filter, and dilation. The 90th percentile mask and the ground truth mask are also shown.

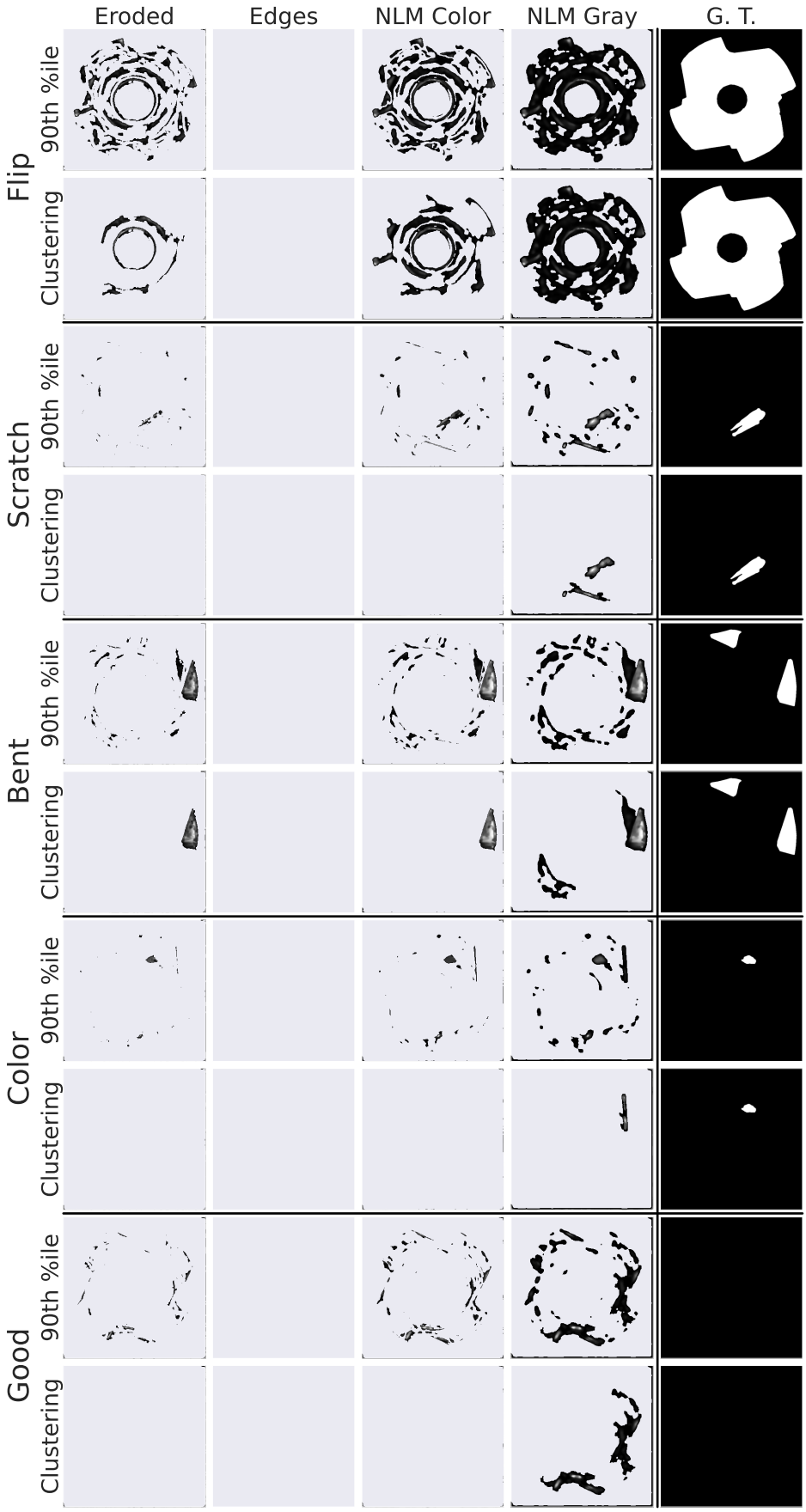


Figure 109: Second batch of clustering filtering techniques. It includes the erosion, edge detection and **NLM** techniques. The 90th percentile mask and the ground truth mask are also shown.

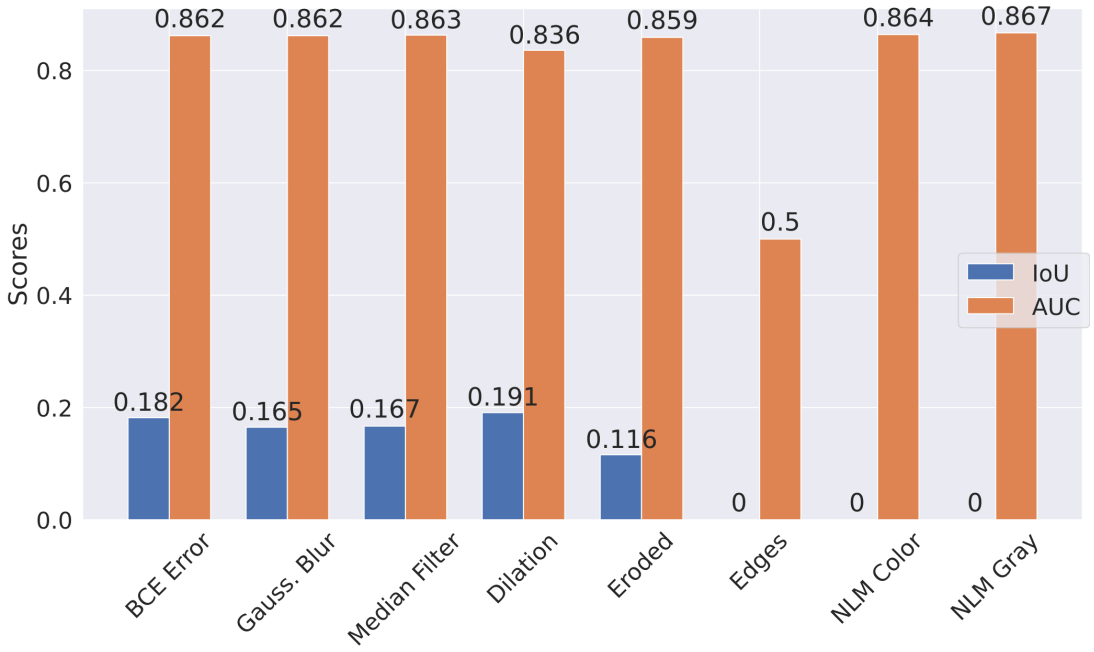


Figure 110: **BCE** performance on the test dataset using different filtering techniques.

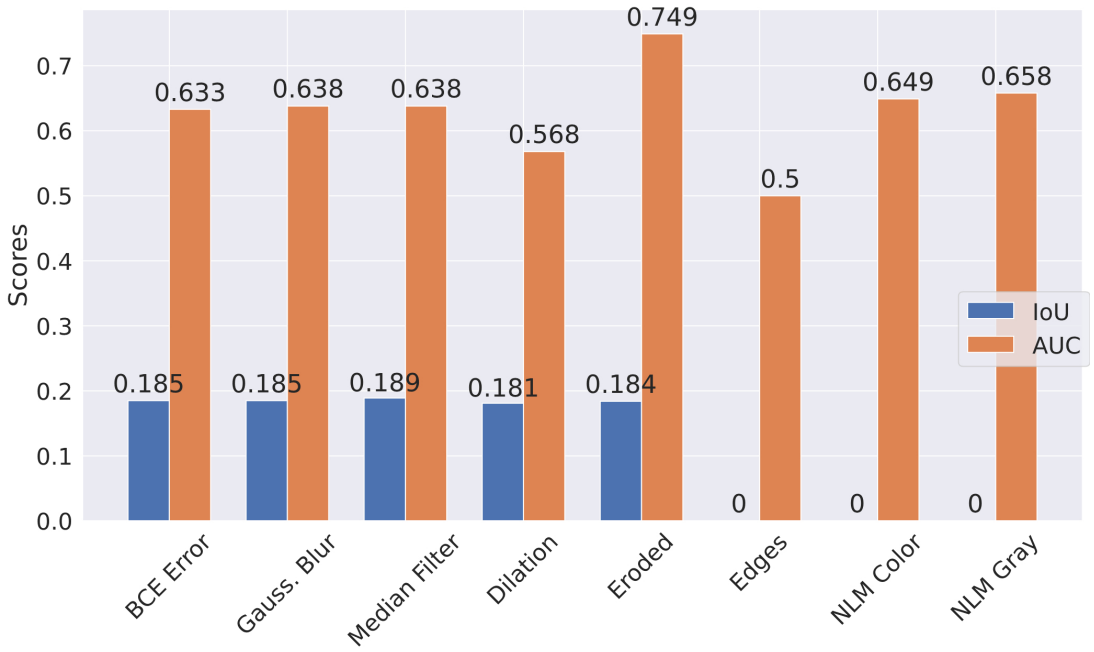


Figure 111: **BCE** performance on the test dataset using diferent clustering techniques.

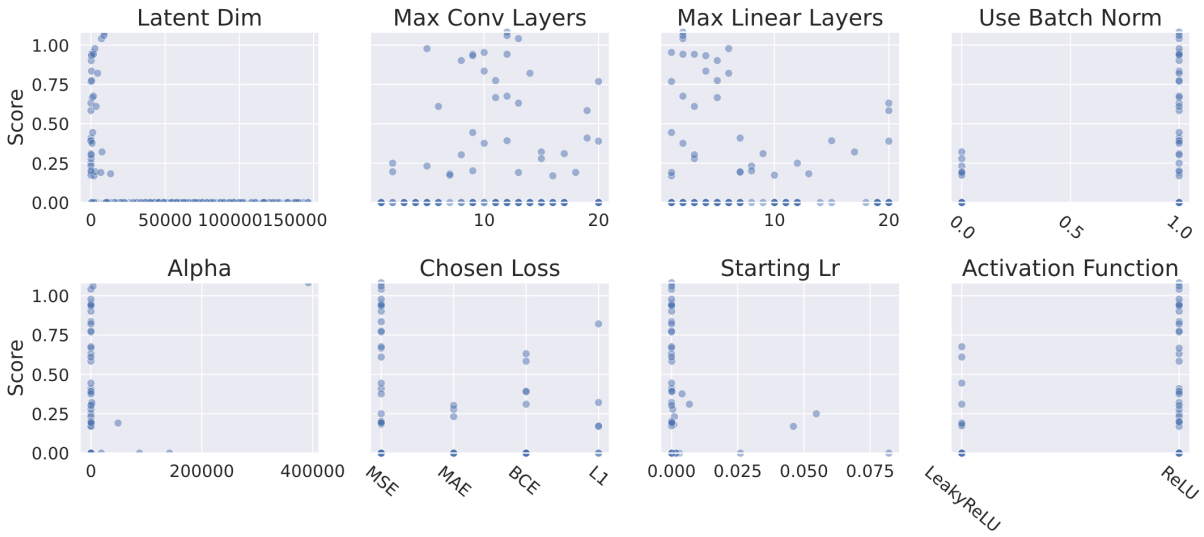


Figure 112: Slice plot of the non-layer specific **HP** for the **VAE** architecture. Note that some **HP** have a boolean value of 1 (True) and 0 (False).



Figure 113: Slice plot of the layer specific **HP** for the **VAE** architecture. The number on the **HP** name refers to the layer number.

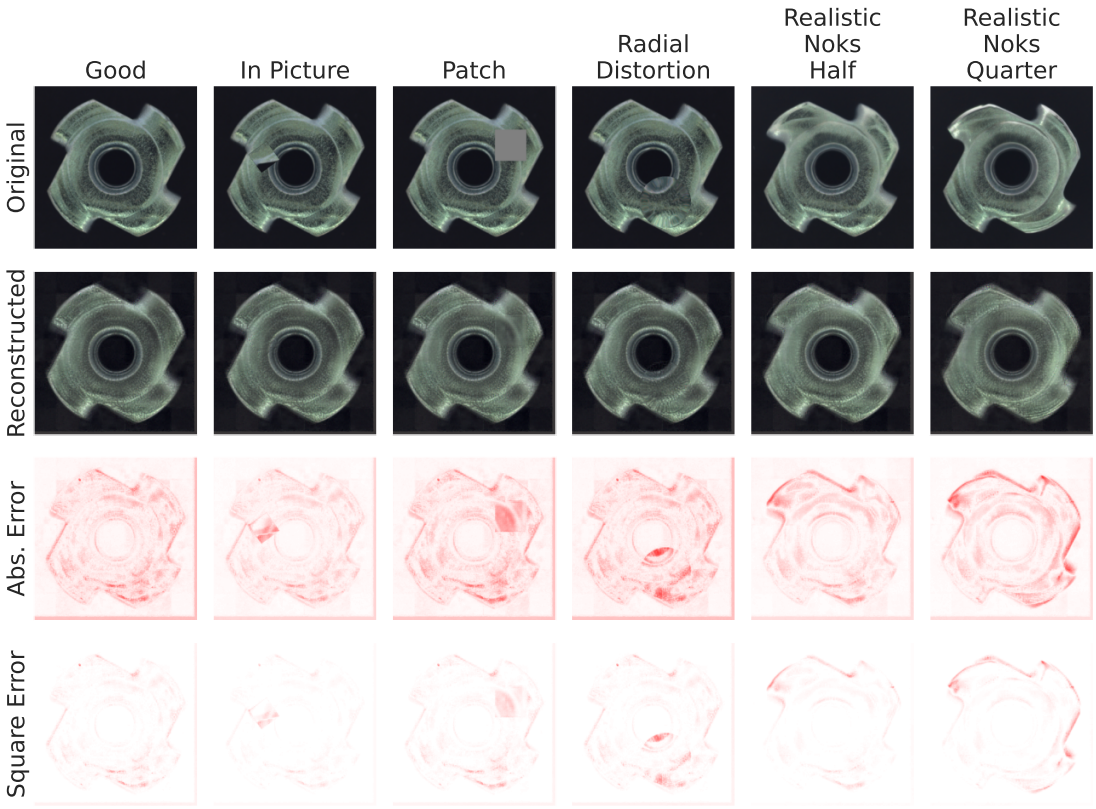


Figure 114: Reconstruction of the validation dataset using the **VAE** architecture. On each column it can be seen the different labels of the validation dataset. The first row shows the original images, the second row the reconstructed images, the third row the absolute error map and the last row the square error.

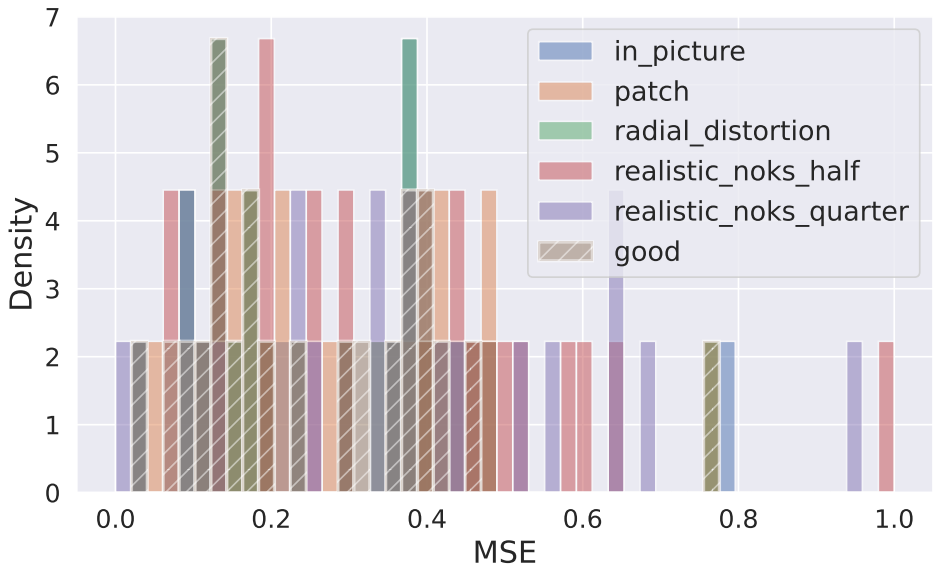


Figure 115: Histogram of the **MSE** reconstruction loss for the validation dataset using the **VAE** architecture.

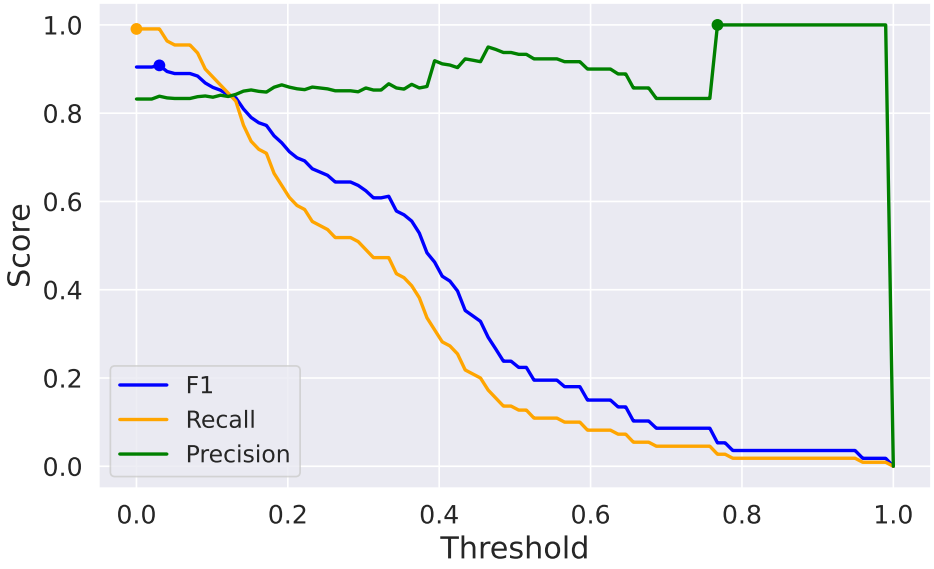


Figure 116: Evaluation of various thresholds using the baseline approach, showing how different metrics vary with each threshold setting on the validation dataset.

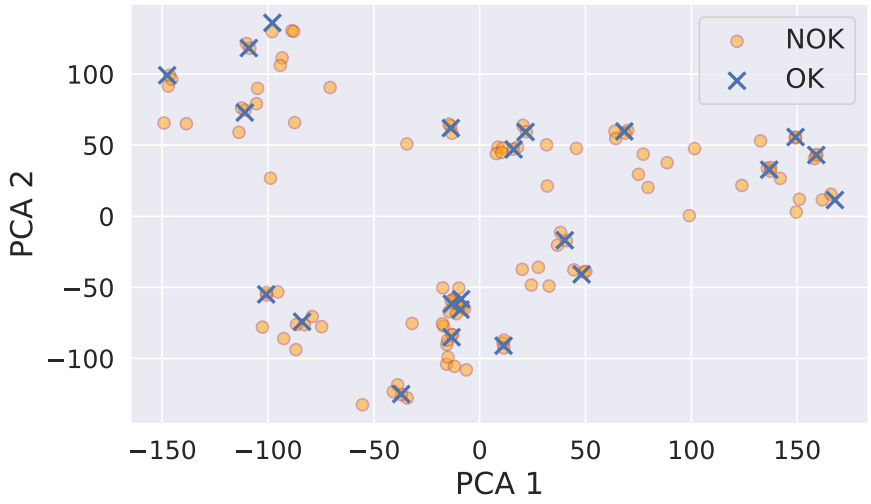


Figure 117: PCA projection of the latent space, demonstrating dimensionality reduction to two principal components.

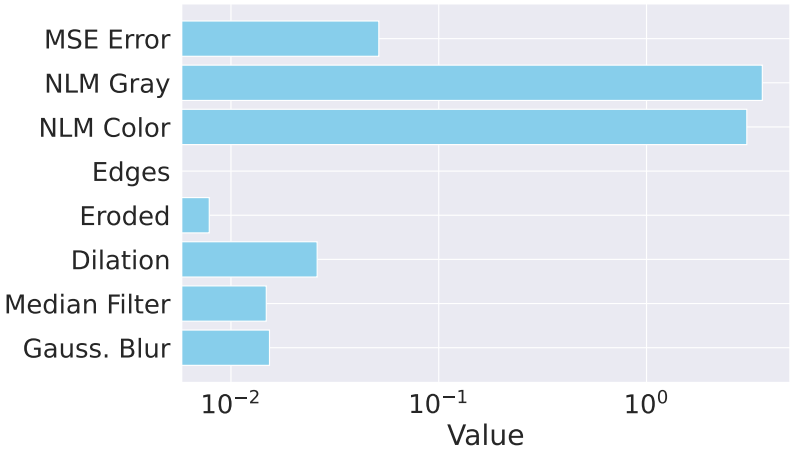


Figure 118: Thresholds for the filtering techniques using the 90th percentile of the OK-error values. Note that "Edges" (Edge Detection) has a *NaN* value since it already outputs a binary image.

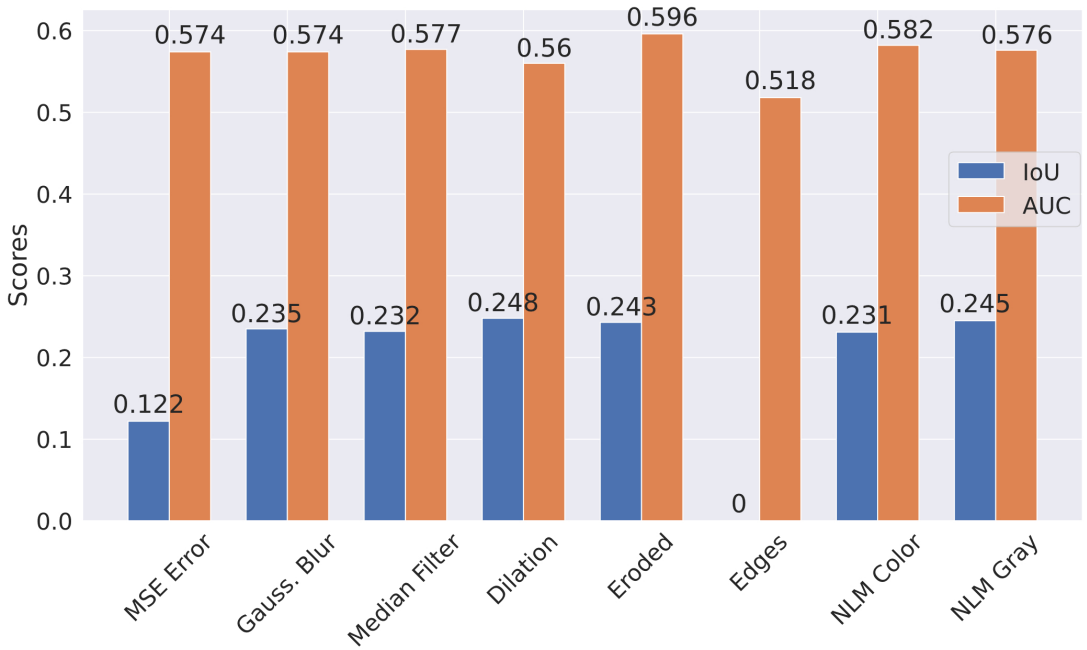


Figure 119: Comparative analysis of **AUC** and **IoU** metrics for various filtering techniques against ground truth labels and true error masks.

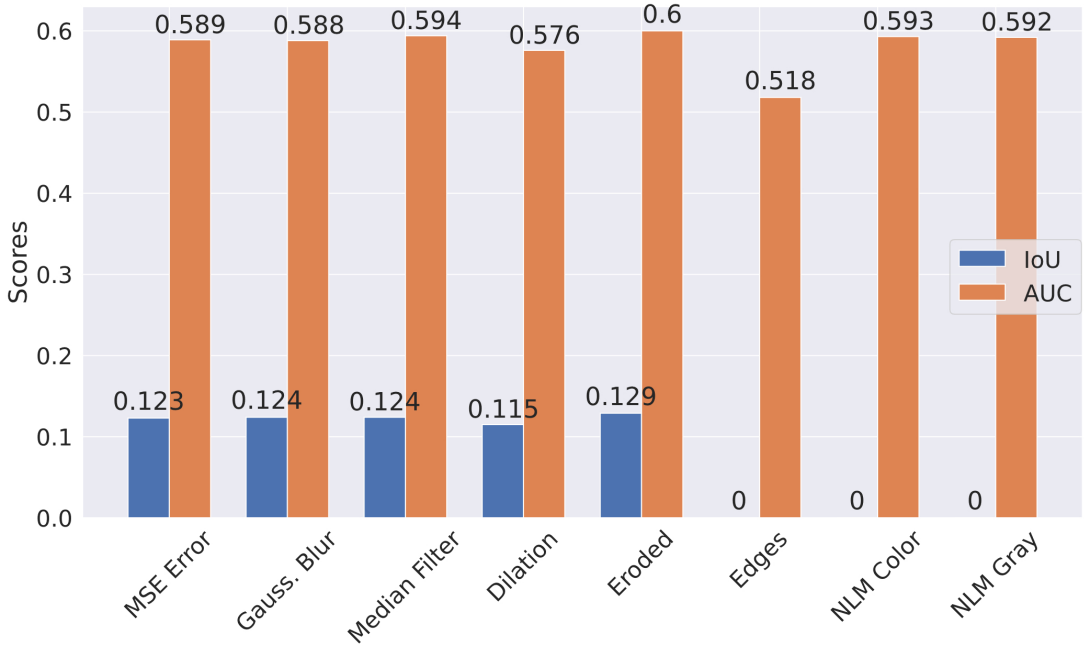


Figure 120: Evaluation of clustering performance using **AUC** and **IoU** metrics compared with ground truth labels and true error masks, illustrating the effectiveness of various clustering techniques.

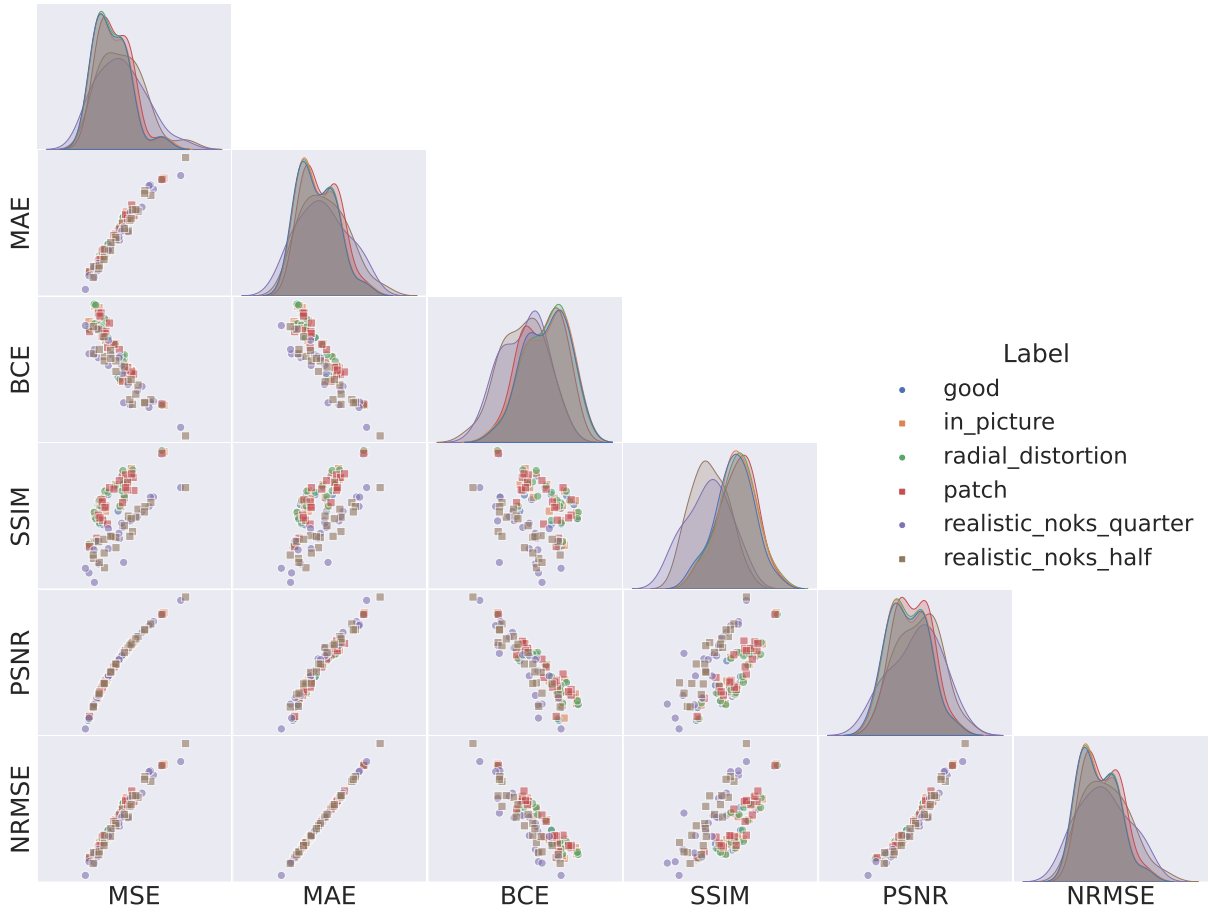


Figure 121: Pairplot of different metrics used to assess the reconstruction error of the validation dataset using the **VAE** architecture. The metrics include **MAE**, **BCE**, **SSIM**, **PSNR**, and **NRMSE**.

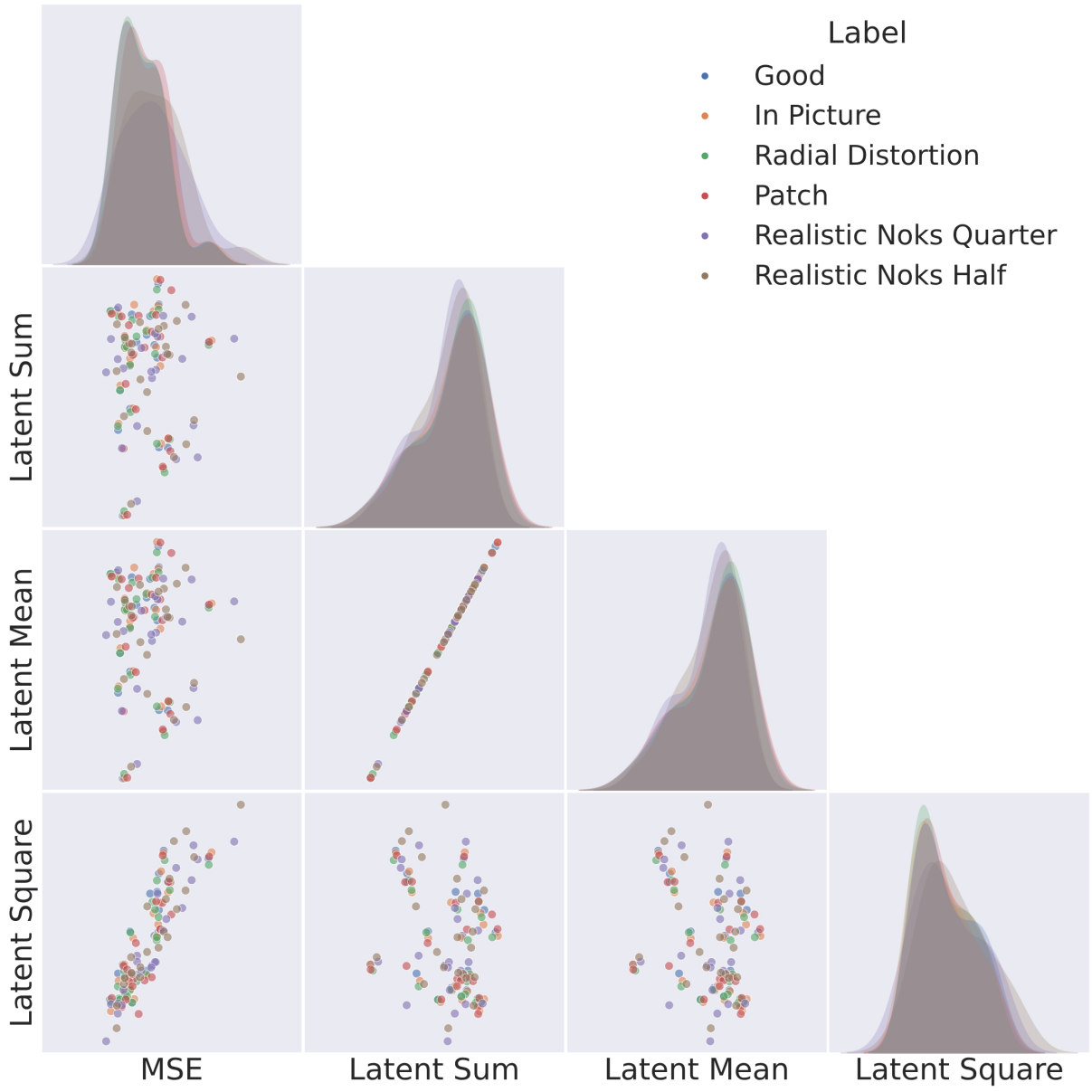


Figure 122: Pairplot visualization of the **MSE** reconstruction loss and the latent space statistical summaries.

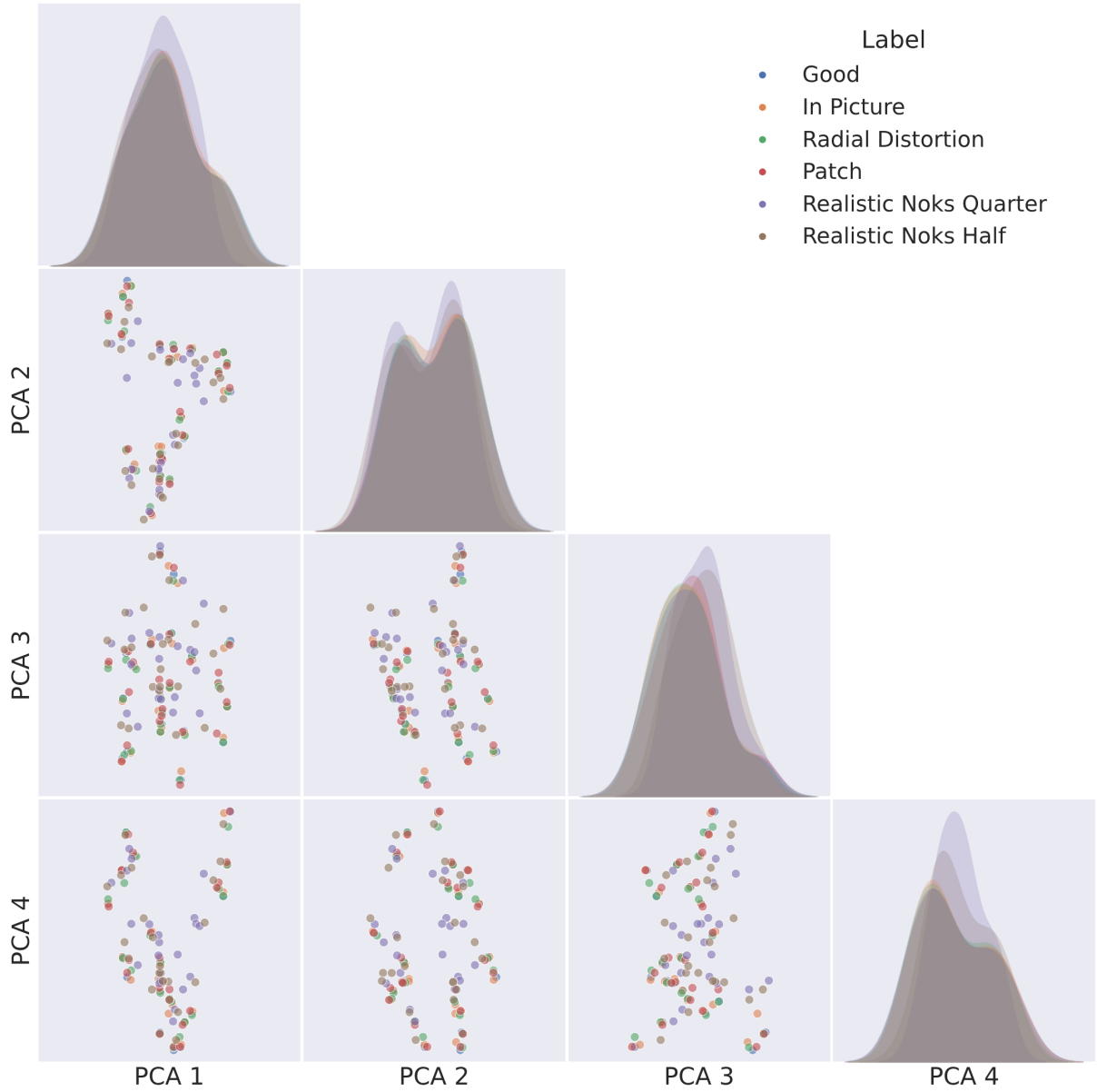


Figure 123: Pairplot of the **PCA**-transformed latent space

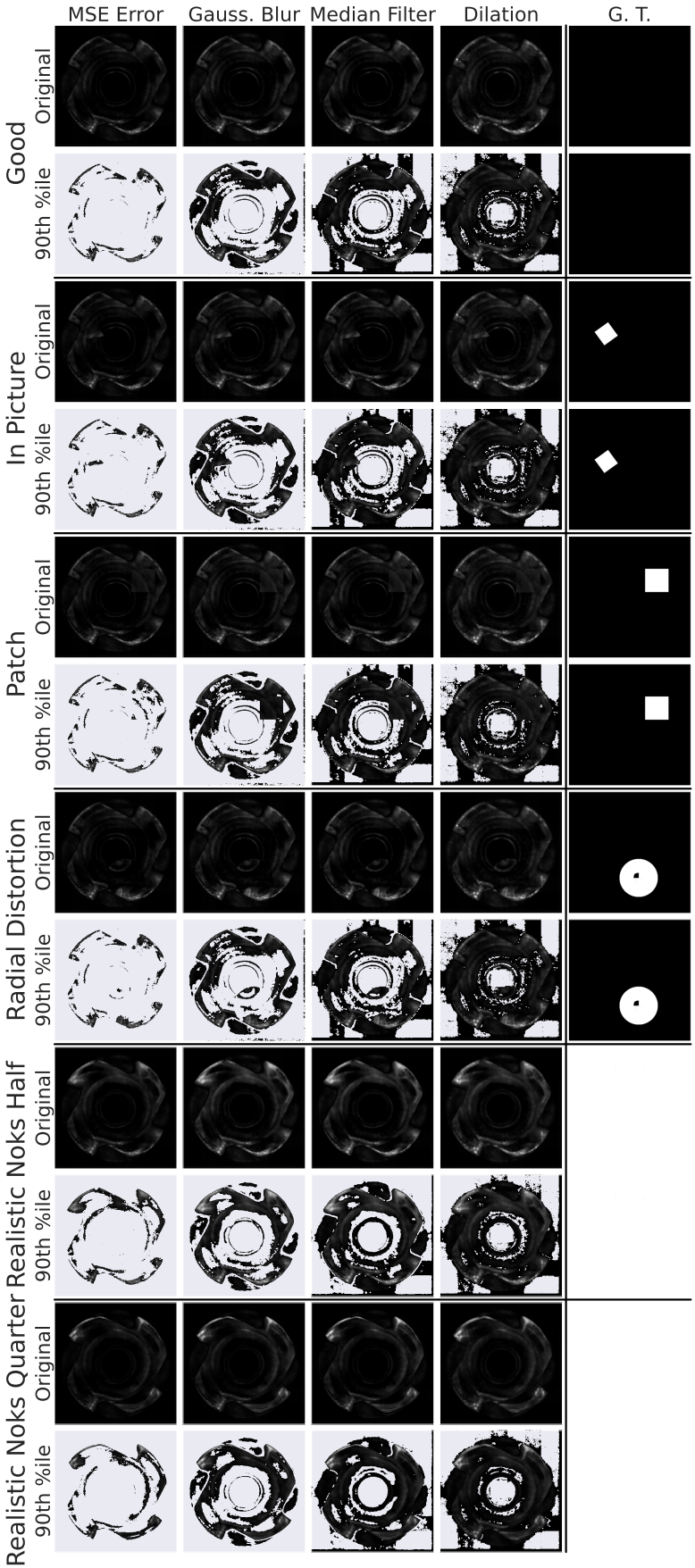


Figure 124: First batch of filtering techniques. It includes the original error image, gaussian blur, median filter, and dilation. The 90th percentile mask and the ground truth mask are also shown.

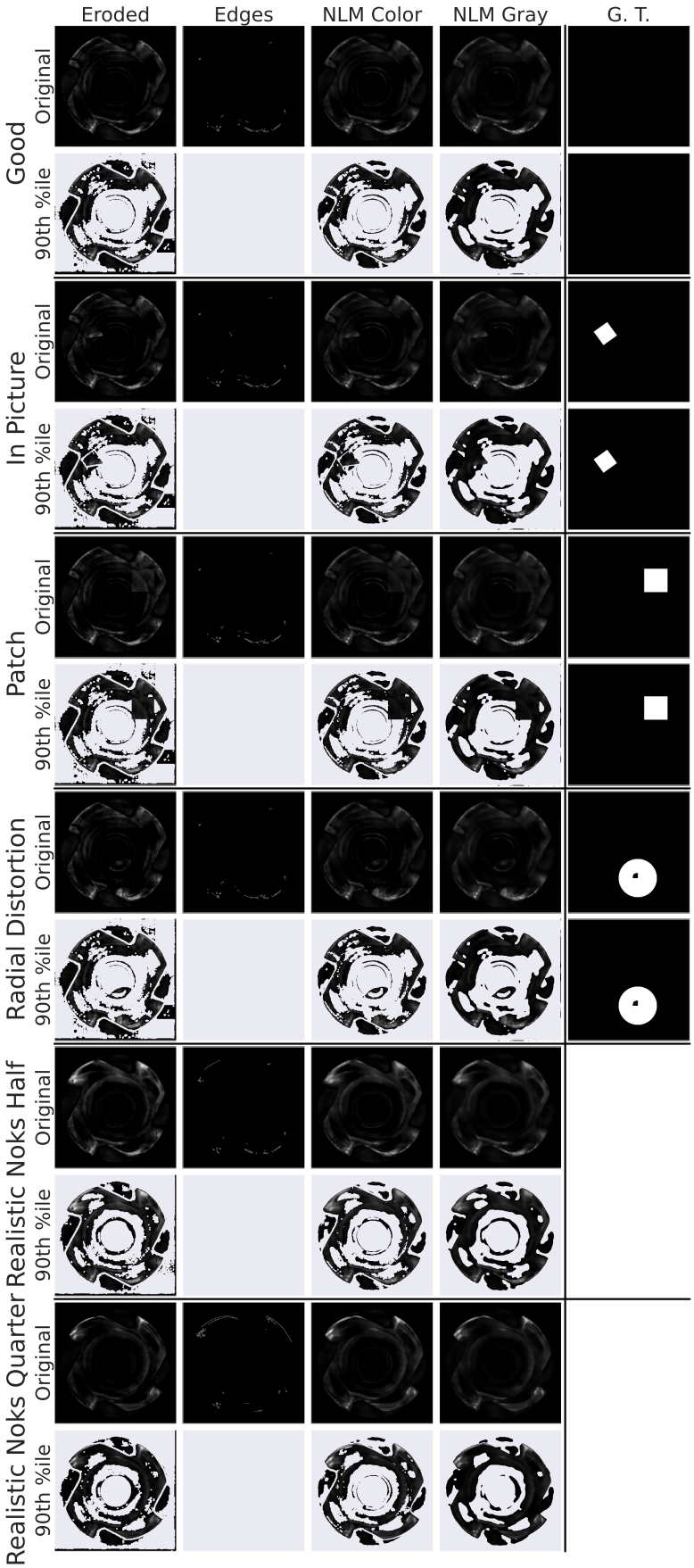


Figure 125: Second batch of filtering techniques. It includes the erosion, edge detection and **NLM** techniques. The 90th percentile mask and the ground truth mask are also shown.

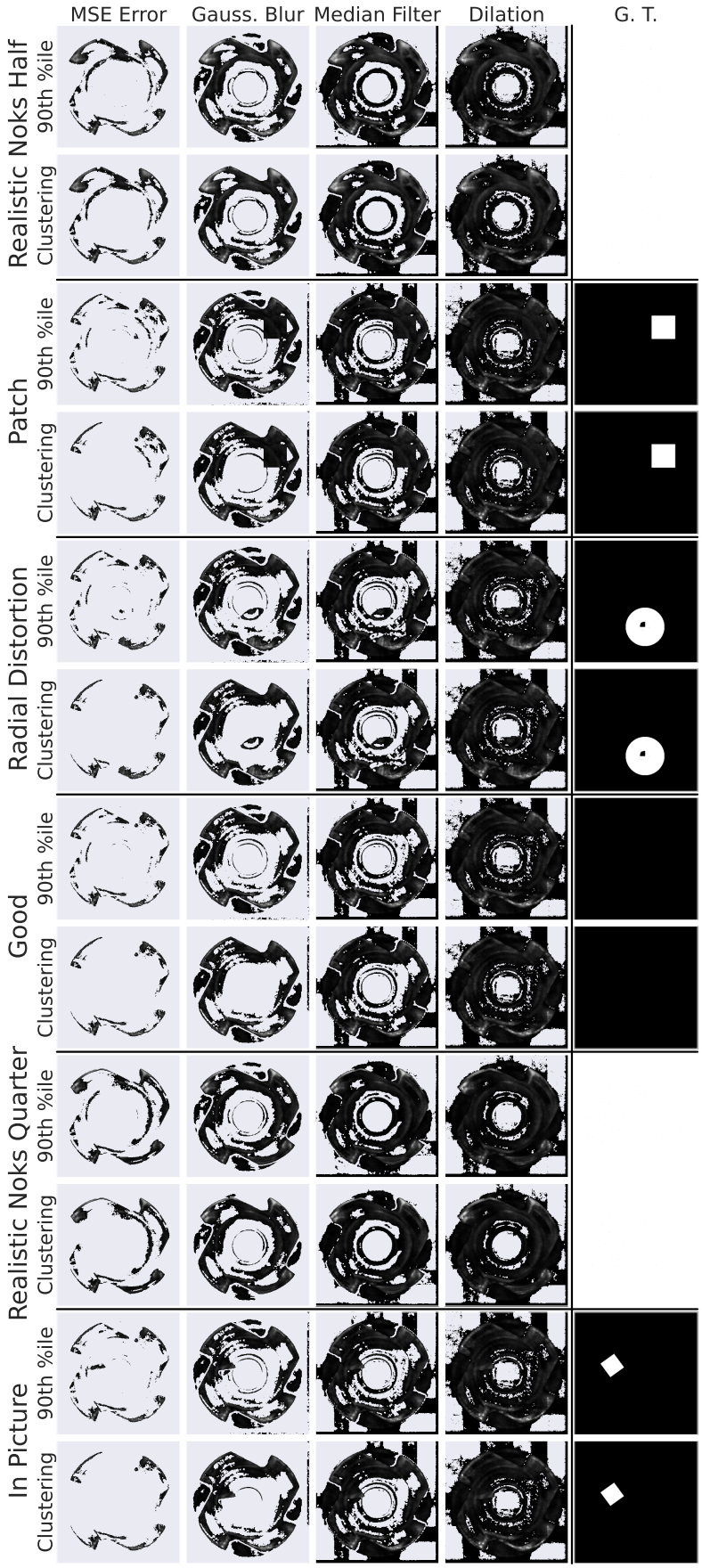


Figure 126: First batch of clustering filtering techniques. It includes the original error image, gaussian blur, median filter, and dilation. The 90th percentile mask and the ground truth mask are also shown.

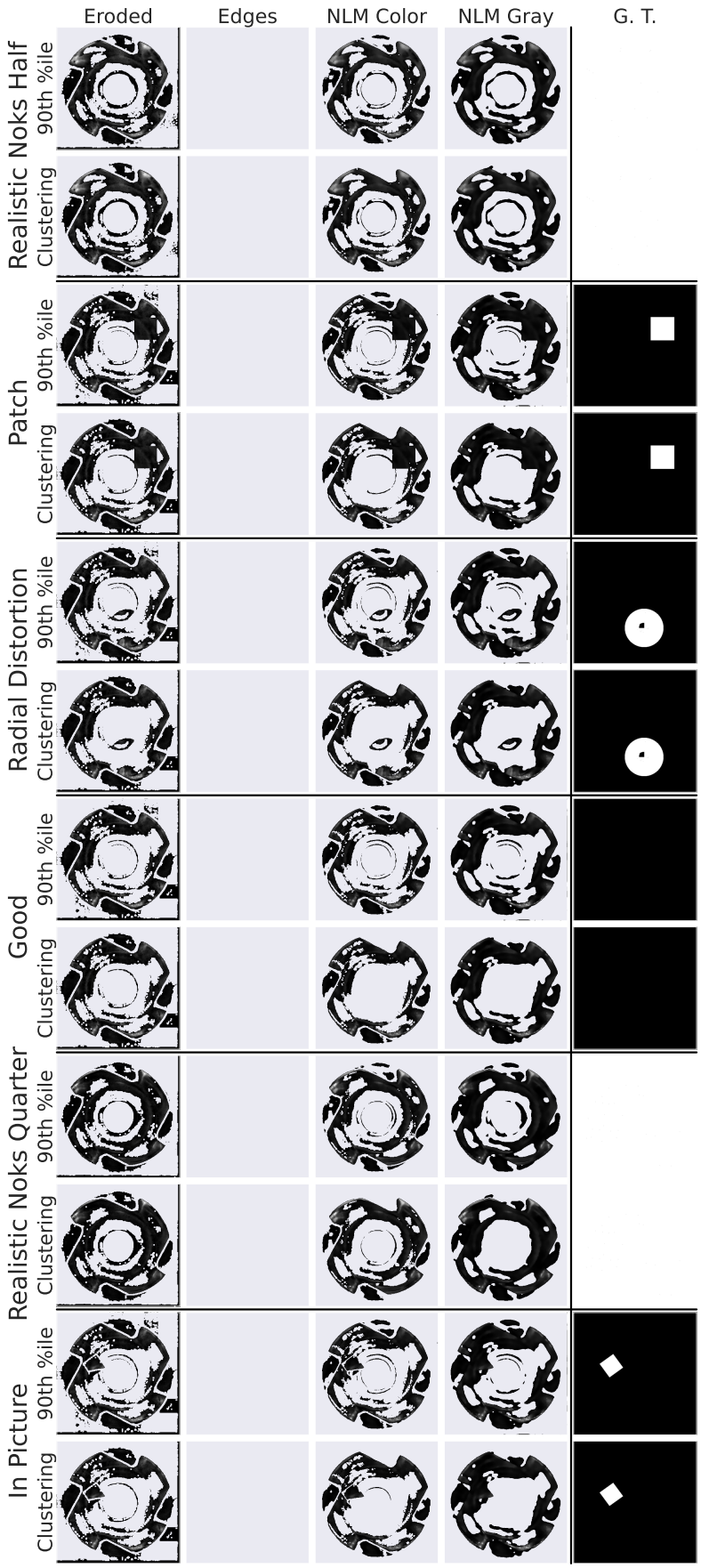


Figure 127: Second batch of clustering filtering techniques. It includes the erosion, edge detection and **NLM** techniques. The 90th percentile mask and the ground truth mask are also shown.

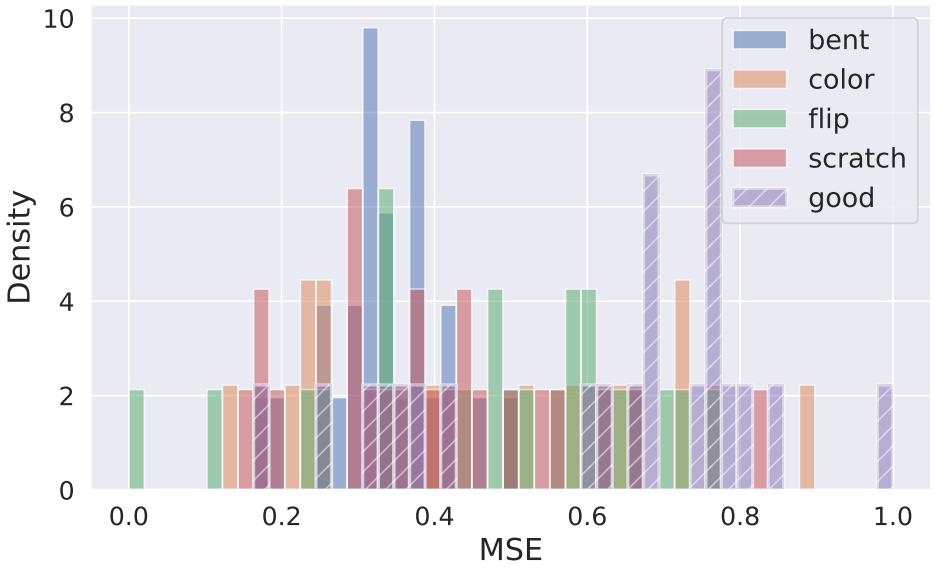


Figure 128: Histogram of the **MSE** reconstruction loss for the test dataset using the **VAE** architecture.

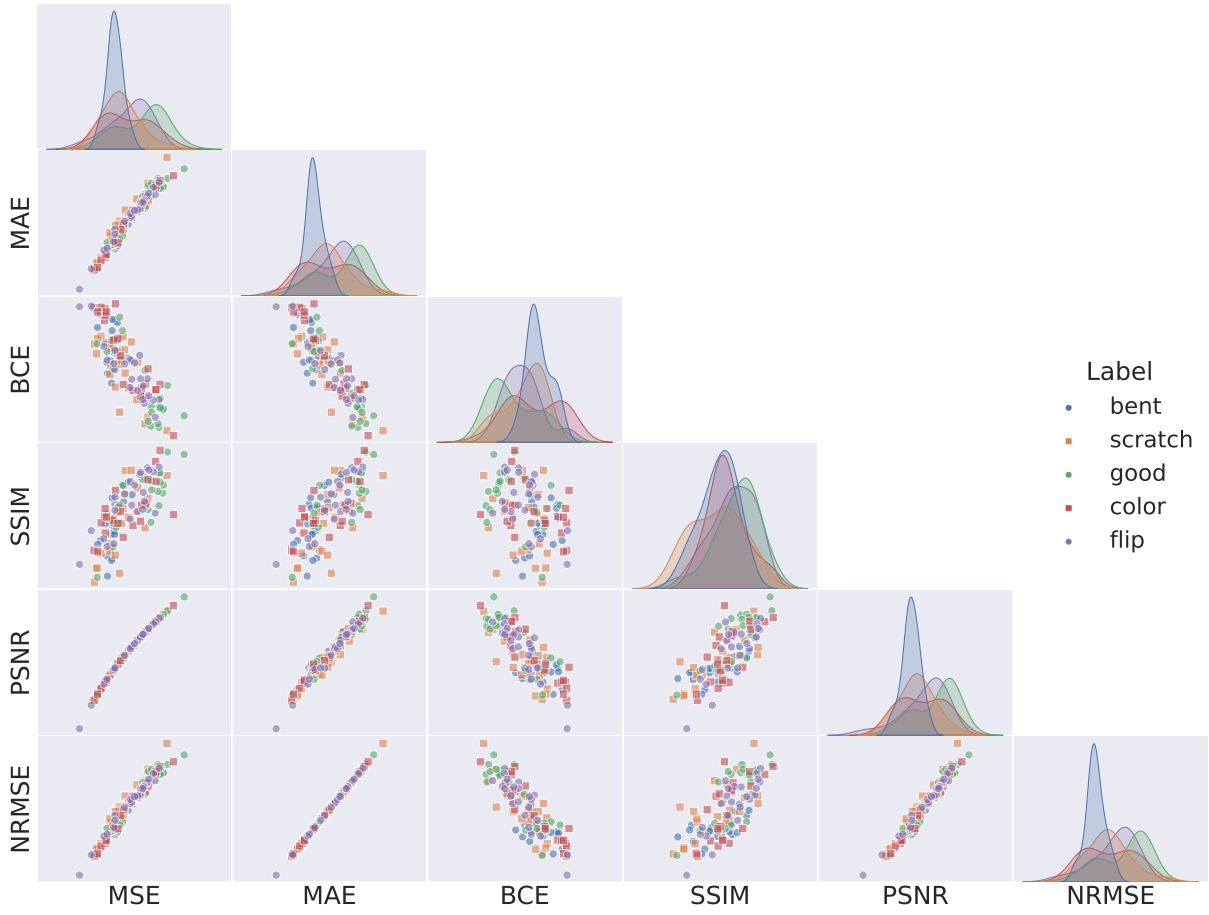


Figure 129: Pairplot of the validation dataset using the **VAE** architecture.

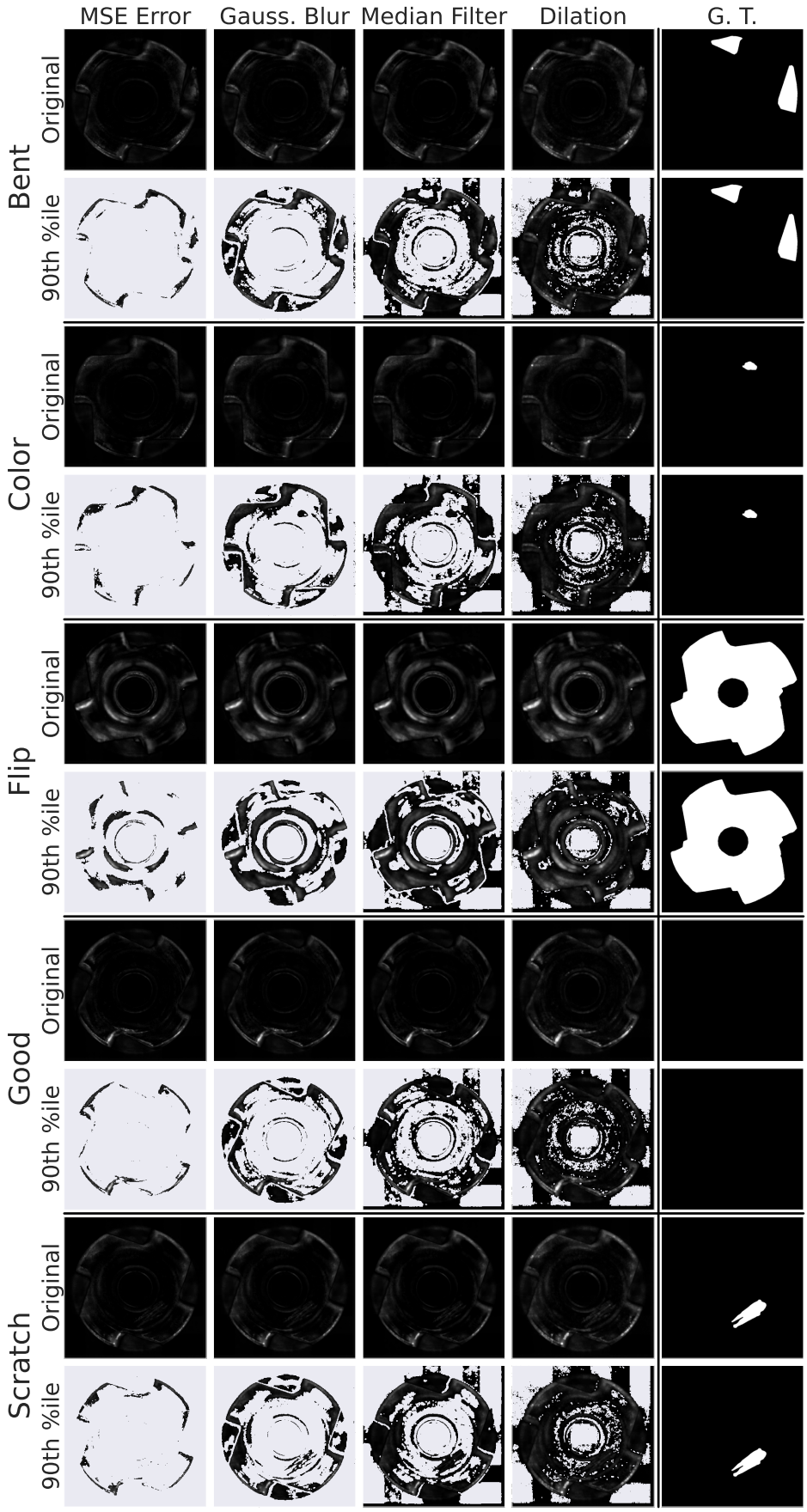


Figure 130: First batch of filtering techniques. It includes the original error image, gaussian blur, median filter, and dilation. The 90th percentile mask and the ground truth mask are also shown.

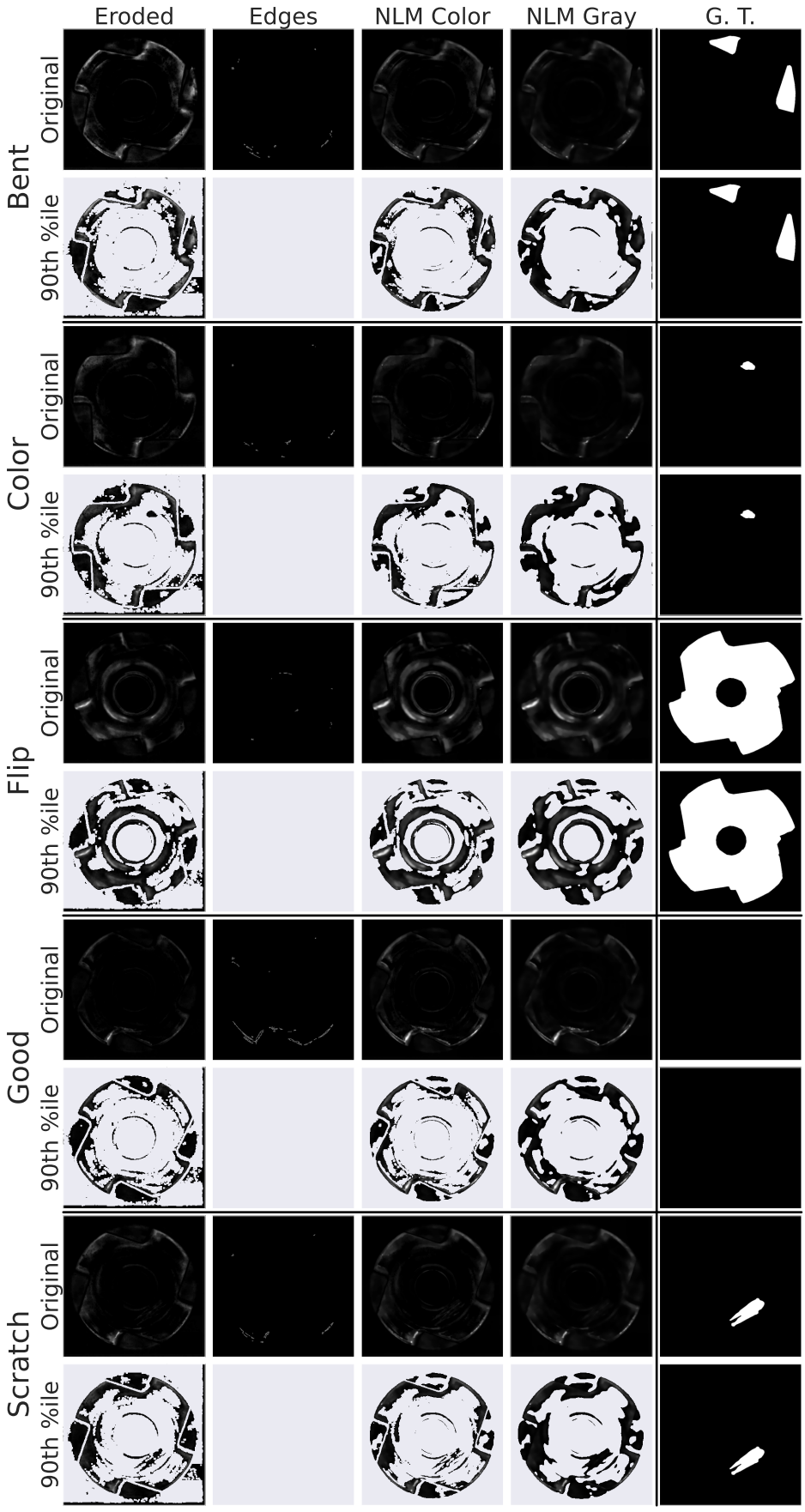


Figure 131: Second batch of filtering techniques. It includes the erosion, edge detection and **NLM** techniques. The 90th percentile mask and the ground truth mask are also shown.

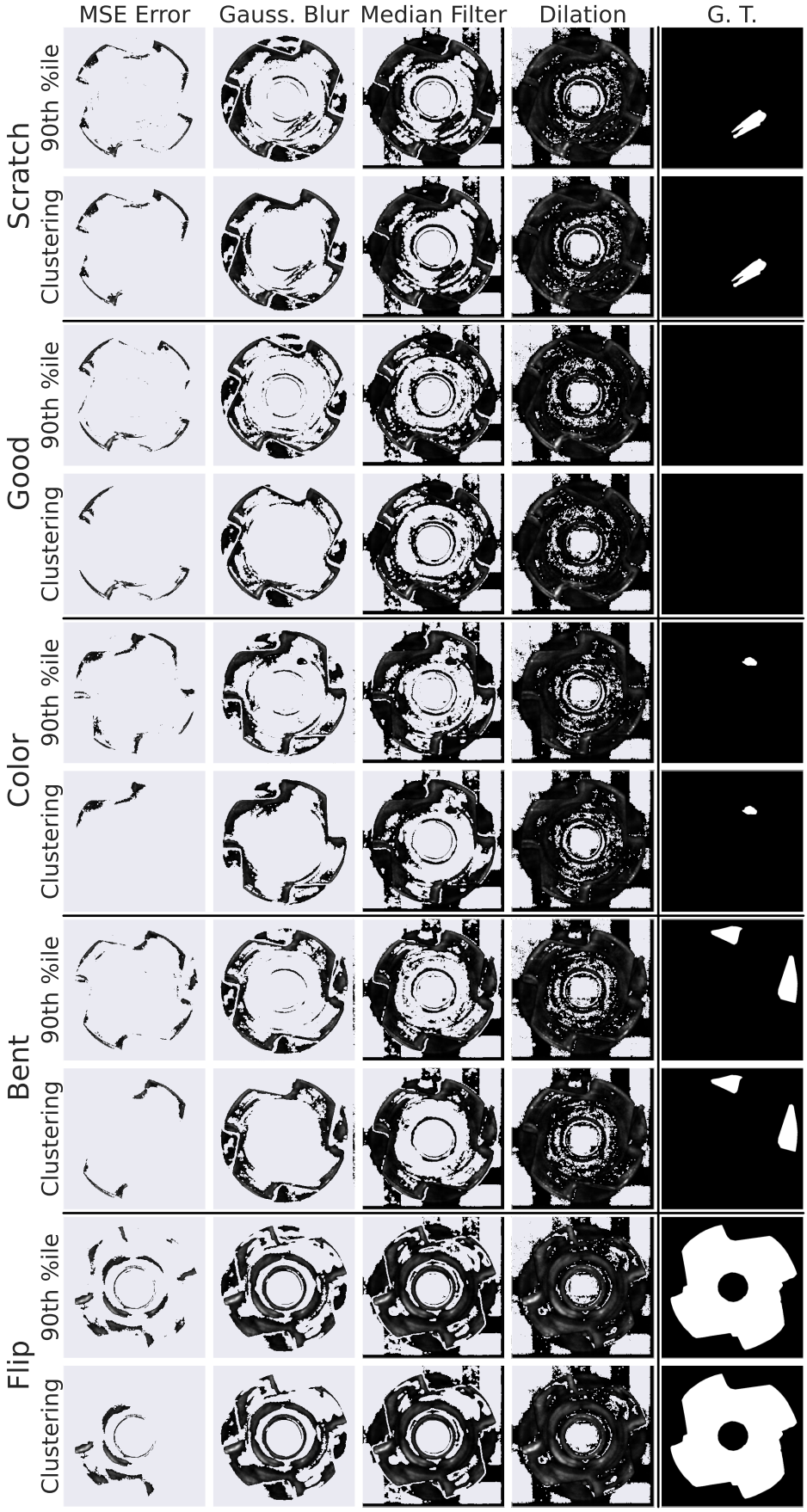


Figure 132: First batch of clustering filtering techniques. It includes the original error image, gaussian blur, median filter, and dilation. The 90th percentile mask and the ground truth mask are also shown.

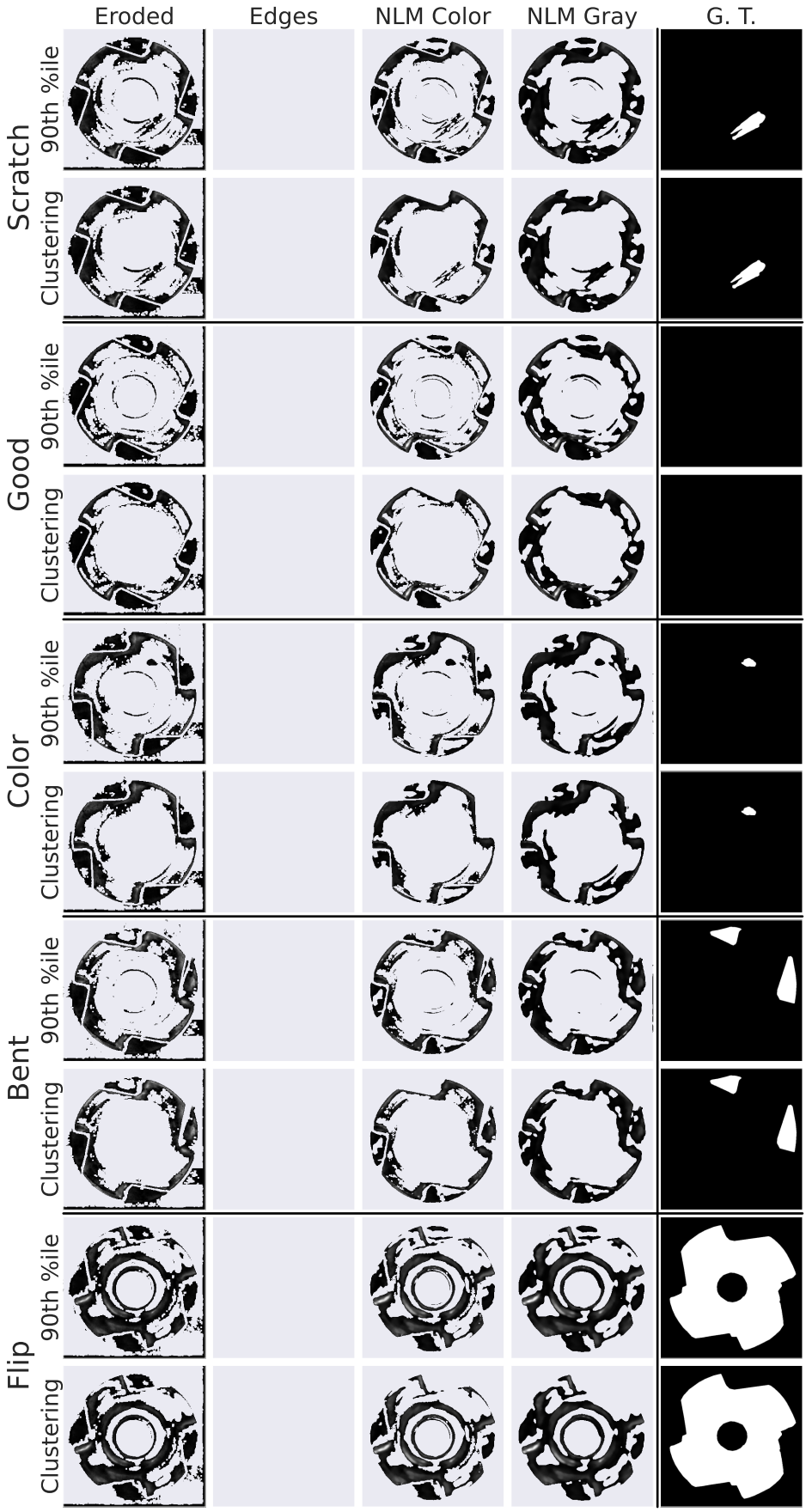


Figure 133: Second batch of clustering filtering techniques. It includes the erosion, edge detection and **NLM** techniques. The 90th percentile mask and the ground truth mask are also shown.

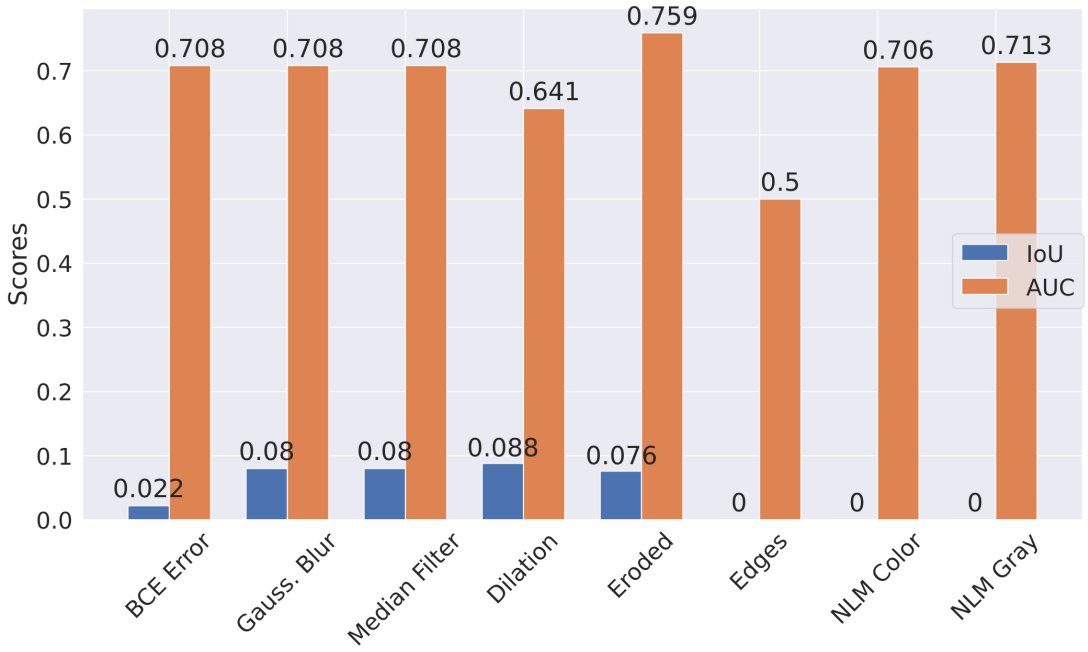


Figure 134: **BCE** performance on the test dataset using different filtering techniques.

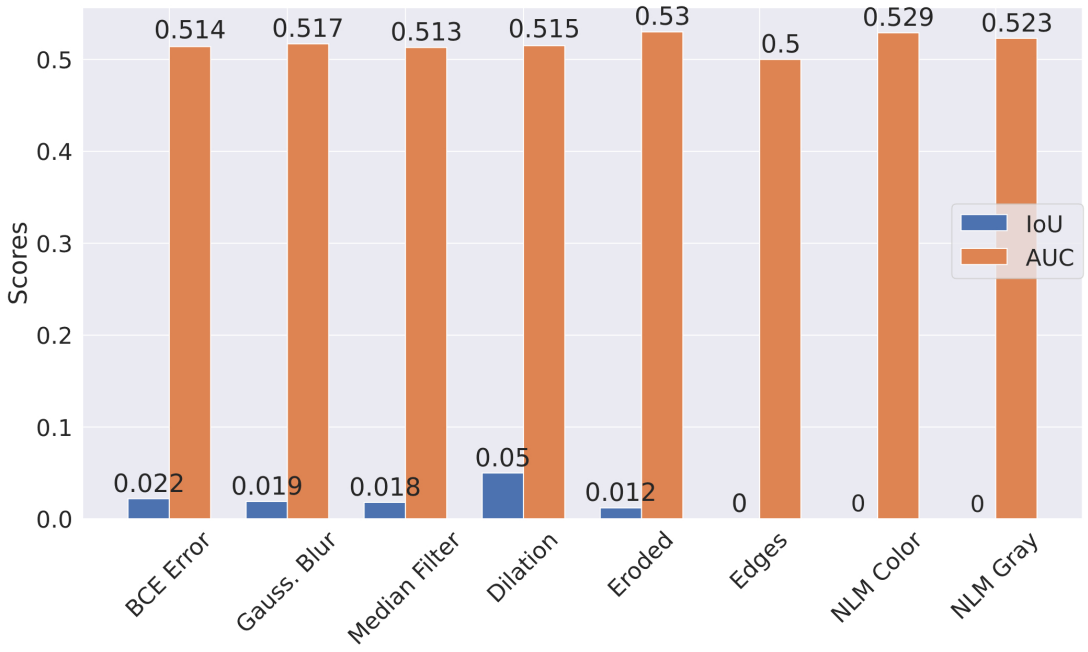


Figure 135: **BCE** performance on the test dataset using different clustering techniques.

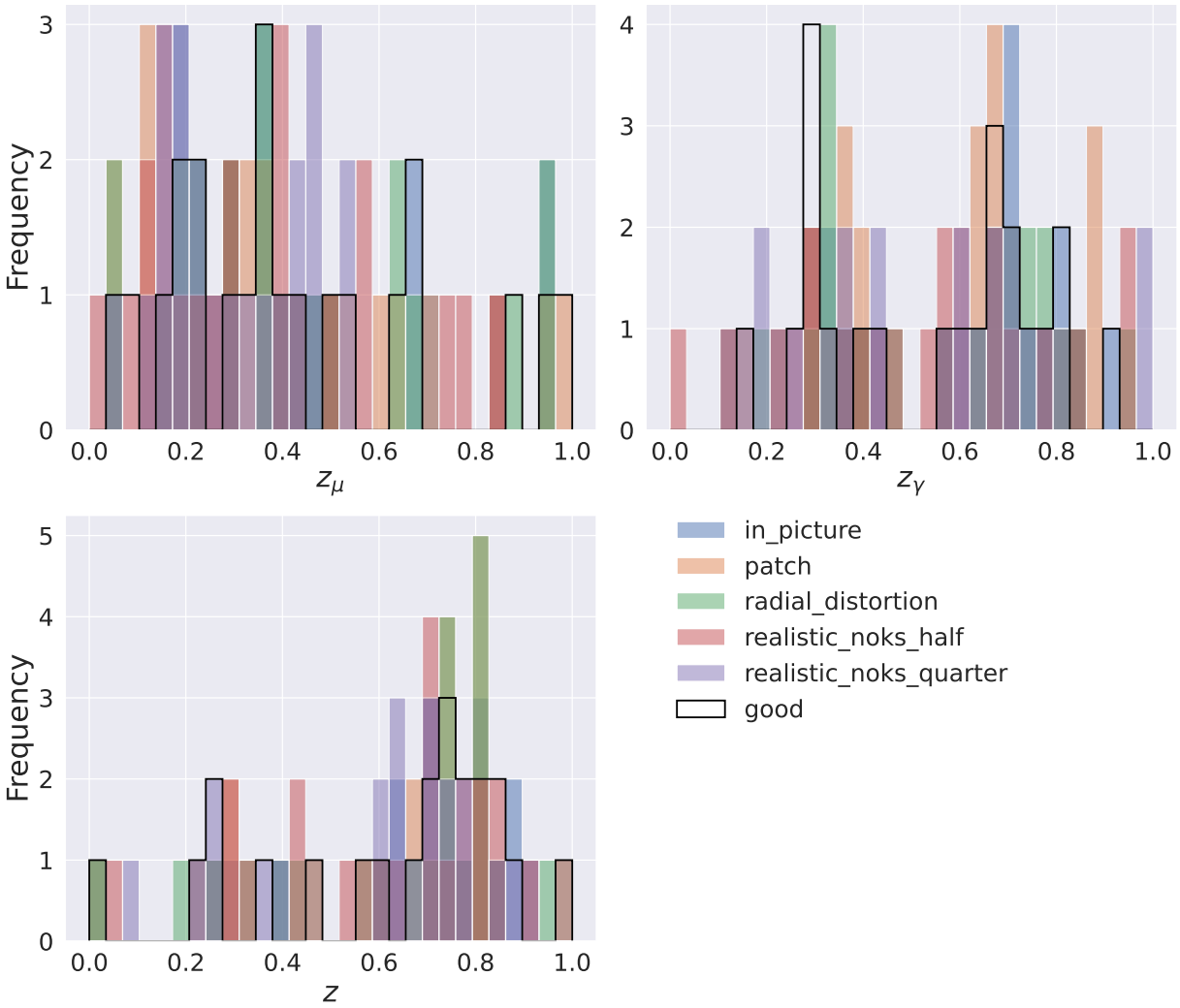


Figure 136: Histograms of the latent space parameters for the validation dataset.

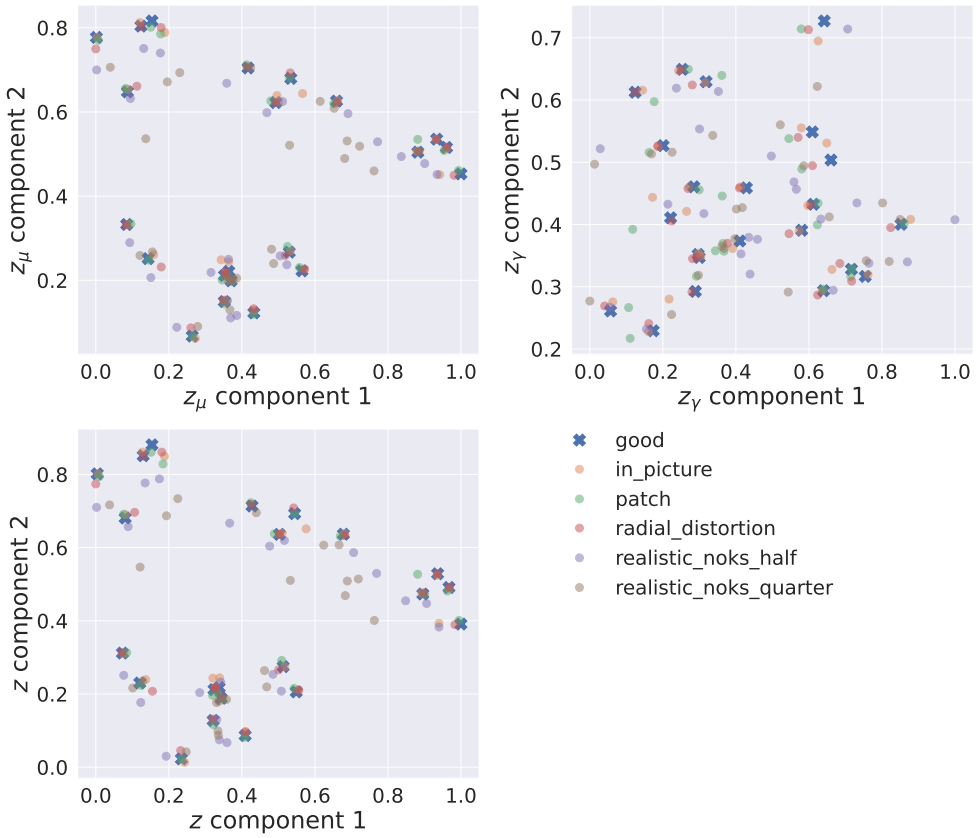


Figure 137: 2D **PCA** projection of the latent space for the validation dataset.