Universiteit Leiden

Master Computer Science

Leveraging Uncertainty for Improved Model Performance: Optimizing Generative Models through Hyperparameter Optimization

Name:	Luuk de Jong
Student ID:	3372944
Date:	31-07-2024
Specialisation:	Artificial intelligence
Daily supervisor:	Inês Gomes (University of Porto)
1st supervisor:	Jan N. van Rijn (Leiden University)
2nd supervisor:	Carlos Soares (University of Porto)

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

In modern-day machine learning models, effective decision-making is essential. Incorrect medical diagnoses can have significant consequences, highlighting models' need to convey their confidence levels. This paper investigates the integration of a reject option in machine learning models to enhance reliability and explainability. By rejecting uncertain predictions, we can mitigate risks associated with low-confidence decisions, meaning that the model will be more reliable. The core contribution of this work is the development and evaluation of AutoGASTeN. This automated pipeline leverages generative adversarial stress test networks (GASTeN) to generate realistic images at the decision boundary of two classes. We can create custom datasets with these low-confidence instances for training machine learning models with a reject option. We first examine the impact of various hyperparameter optimization techniques on GASTeN's performance. Bayesian optimization and BOHB demonstrate better performance than the baseline results. This, for example, resulted in a Fréchet inception distance value of 10.12 and an average confusion distance value of 0.34 on the binary MNIST dataset 8 versus 0, where the original work scored 12.71 and 0.42, respectively. Subsequently, we employ AutoGASTeN to generate custom datasets with low-confidence instances, training a convolutional neural network to reject uncertain predictions. The results indicate that the convolutional neural network, trained with these custom datasets, maintains the same precision, recall, and F1 scores as models trained without the reject option, thereby enhancing overall decision reliability without sacrificing performance.

Acknowledgements

First, I would like to thank my supervisor, Jan N. van Rijn. Our meetings, filled with exciting discussions about this idea's potential and the refinement of my ideas, have been incredibly helpful. I sincerely appreciate your assistance in writing my proposal for the TAILOR fund and introduction to the ADA group, where I could meet other master students going through the same process. Second, I would like to thank my daily supervisor, Inês Gomes, and supervisor, Carlos Soares. Our meetings in Porto, where we shared stories and laughs before getting down to thesis discussions, have been a highlight of this process. Also, your feedback has been incredible in finishing my thesis. Last, I would like to thank my family and friends; your support has been fundamental. Many of you may not even know if you supported me, but being there when I wanted to have a drink or just a fun time when I was not writing my thesis has been amazing. Thank you all!

Project funding

This project was funded by the TAILOR connectivity fund, [1] with proposal number 90. This fund is a financial initiative under the European Union's Digital Europe program. It provides funding for projects that enhance research and development in trustworthy AI, facilitating collaborations and infrastructure development across Europe.

[1] https://tailor-network.eu/connectivity-fund/

Contents

1 Introduction **3**

2 Background **5**
2.1 Hyperparameter optimization 5
2.2 Machine learning with a reject option 6
2.2.1 Separated rejector architecture 7
2.2.2 Dependent rejector architecture 7
2.3 Explainable AI 8
2.4 Generative AI 9

3 Related Work **11**
3.1 Generative adversarial networks 11
3.2 AmbiGuess 12
3.3 Dirty MNIST dataset 13

4 Methodology **14**
4.1 Hyperparameter optimization techniques 14
4.1.1 Random grid search 14
4.1.2 Random search 15
4.1.3 Bayesian optimization 15
4.1.4 Hyperband 15
4.1.5 BOHB (Bayesian optimization and hyperband) 16
4.2 AutoGASTeN 16
4.2.1 Optimization criteria 17
4.3 Multi-objective decision making 17
4.4 Clustering and image generation 18
4.5 Integrated rejector architecture 19
4.6 Practicalities 19

5 RQ1: Hyperparameter optimization **20**
5.1 Experimental setup 20
5.1.1 Dataset 20
5.1.2 Evaluation 20
5.2 Results 21
5.2.1 Baseline 22
5.2.2 Hyperparameter optimization technique comparison 23
5.3 Limitations 26

6 RQ2: Machine learning with a reject option **27**
6.1 Experimental setup 27
6.1.1 Dataset 27

6.1.2 Evaluation 27
6.2 Results 28
6.2.1 Baseline 28
6.2.2 Custom dataset with low-confidence instances 29
6.3 Limitations 32

7 Conclusion **33**

A Additional results **38**
A.1 Binary MNIST data 8 versus 0 38
A.2 Binary MNIST data 7 versus 1 39
A.3 Binary MNIST data 5 versus 3 45
A.4 Binary MNIST data 9 versus 4 51
A.5 Binary fashion-MNIST data dress versus T-shirt 57

Chapter 1 Introduction

In modern-day machine learning models, effective decision-making is essential when a machine learning model is uncertain. Machine learning models can identify diseases such as Alzheimer's, heart failure, breast cancer, and pneumonia [Ahsan and Siddique, 2021]. However, an incorrect diagnosis can have serious consequences for patient treatment. For example, while a machine learning model may accurately identify Alzheimer's disease in most cases, there are instances where the model is uncertain about its decision. It is imperative to always involve a human expert in the decision-making process. When the model is uncertain, it is safer to abstain from predicting and deferring the decision to a human expert to avoid harmful consequences. Putting a human in the loop for the decision-making process is essential in healthcare. This example highlights the critical importance of confidence in machine learning predictions.

The problem is that even if a model is not confident in its decision, it still provides class predictions without indicating its uncertainty [de Menezes e Silva Filho et al., 2023]. For instance, consider a binary classification scenario where the model encounters an image closely resembling both classes. The model might predict a given class with low-confidence but still make a definitive prediction. In such cases, the probabilities alone are unreliable indicators of the model's certainty. It would be advantageous if the model could abstain from predicting when faced with such uncertain instances. Rejecting an instance or abstaining when predicting is typically done with machine learning with a reject option technique [Hendrickx et al., 2021]. Machine learning with a reject option refers to a classification scenario where the model not only predicts the class but also has the option to abstain from making a prediction when it detects high uncertainty. By incorporating this option, machine learning models can enhance their reliability and explainability. Explainability, as defined by Arrieta et al. [2020], refers to the capability of an AI system to provide transparent reasons for its decisions. If the model abstains from predicting a certain instance, users can examine the underlying reasons for this choice, contributing to the model's transparency. Reliability, on the other hand, refers to the ability of an AI system to consistently produce accurate and trustworthy results under various conditions [Arrieta et al., 2020]. Both explainability and reliability are essential components of Trustworthy AI, helping to mitigate the risk of incorrect predictions and enhancing the overall accuracy and dependability of the model.

Specifically, which datasets are suitable for training a machine learning model to abstain from decision-making? We can utilize a state-of-the-art approach outlined by Cunha et al. [2023] to address this. The generative adversarial stress test network (GASTeN) is a variation of a generative adversarial network. GASTeN generates realistic data that lies on the decision boundary of two classes. It identifies instances classified with low-confidence using a given classifier. A generative adversarial network typically uses the difference between the generator's output and the actual image as an objective. GASTeN extends this with the classifier's output on the generated images as part of a new optimization objective for the generator. With GASTeN, we can create custom datasets with low-confidence instances, which can be used to train a machine learning model to abstain from decision-making. One of the limitations of GASTeN

is that the number of hyperparameter combinations tested may not fully represent its behavior, as stated by Cunha et al. [2023]. Generative adversarial networks are known to be highly sensitive to hyperparameter choices such as learning rates, batch sizes, and network architectures (Lucic et al. [2018]; Sharma et al. [2019]). This sensitivity can significantly influence the model's performance and stability. For instance, small changes in hyperparameters can lead to considerable differences in the quality of the generated data. Therefore, there is room for improvement, and we can use hyperparameter optimization techniques to improve GASTeN's performance. We will use automated machine learning to automate this process [Hutter et al., 2019].

With hyperparameter optimization techniques such as hyperband introduced by Li et al. [2017], we can optimize the performance of GASTeN. Hyperband is an automated algorithm configuration method for hyperparameter optimization that employs successive halving and early-stopping strategies. It efficiently allocates resources to promising hyperparameter configurations while discarding poor ones, ultimately identifying a good configuration within a specified resource budget [Li et al., 2017]. Firstly, we can automate the entire training and optimization process for GASTeN by creating a pipeline. We call this AutoGASTeN. The pipeline only needs input on which dataset and classes we want GASTeN to train on. Secondly, with AutoGASTeN, we can produce images on the decision boundary of two classes. We can use these instances to fine-tune or train other machine learning models with machine learning with a reject option techniques to abstain from predicting these instances.

In this work, we will explore a new dimension to the previously mentioned problem: generative models that can generate realistic images on the decision boundary of two classes. GASTeN is a generative adversarial network that does precisely this and enables us to develop the images that the model should abstain from predicting [Cunha et al., 2023]. With AutoGASTeN, we can effortlessly generate new images on the decision boundary of two classes on new datasets. Researchers can use these images in training (i.e., informing the model that this is an outlier class) and testing context (i.e., verifying whether machine learning models correctly abstain from prediction on complex classes). Moreover, we studied the following research questions:

- **RQ1:** How do varying hyperparameter optimization techniques impact the performance, expressed in Fréchet inception distance and average confusion distance, of GASTeN across small gray-scale datasets?
- **RQ2:** What is the impact of the integrated rejection architecture on the performance, expressed in precision, recall, and F1 scores of a convolutional neural network when handling binary subsamples of small gray-scale datasets with low-confidence instances?

The remainder of this paper is organized as follows: Chapter 2 gives a background on the relevant topics to form a basic understanding of the problem. Chapter 3 describes the most important related works, and Chapter 4 describes the proposed methods. In Chapter 5, we report the experimental setup, results, and discussion for RQ1. Chapter 6 details the experimental setup, results, and discussion for RQ2. Finally, Chapter 7 addresses our research questions, concluding remarks, and future work.

Chapter 2 Background

This chapter provides the background needed to understand the research objectives. It covers a background on neural networks, hyperparameter optimization, and machine learning with a reject option.

2.1 Hyperparameter optimization

Automated machine learning (AutoML) automates the selection of machine learning pipeline components and hyperparameter optimization, democratizing machine learning by making high-performance models accessible with minimal manual effort [Baratchi et al., 2024]. As described by Baratchi et al. [2024], machine learning algorithms generate machine learning models by optimizing all the parameters in the model. Typically, machine learning algorithms have hyperparameters that need to be set externally by a human. These hyperparameters control various aspects of the training process and the model. Examples of hyperparameters include the learning rate in gradient descent algorithms, the number of hidden layers and neurons in a neural network, and the decision tree depth. To illustrate, modern-day machine learning models, such as a large language model, can have billions of parameters, such as weights [Wei et al., 2022] and only ten to fifty hyperparameters. Hyperparameter optimization is fine-tuning a machine learning algorithm's hyperparameters to improve its performance [Baratchi et al., 2024]. The goal is to find the set of hyperparameters, also called a configuration, that maximizes the machine learning model's performance on a given dataset or task. A general formulation for the hyperparameter optimization problem, as outlined in Baratchi et al. [2024], can be defined as follows:

$$\lambda^* \in \arg\min_{\lambda \in \Lambda} \frac{1}{k} \cdot \sum_{i=1}^{k} \mathcal{L}\left(A_\lambda, \mathcal{D}_{\mathsf{train}}^{(i)}, \mathcal{D}_{\mathsf{valid}}^{(i)}\right) \tag{2.1}$$

Where A is the learning algorithm, λ is the hyperparameter configuration, and A_λ denotes that A with the hyperparameter vector. $\mathcal{L}$ is the loss measured using a performance metric, with $\mathcal{D}_{\mathsf{train}}$ being the training dataset and $\mathcal{D}_{\mathsf{valid}}$ the validation dataset. The hyperparameter optimization problem can be defined as finding λ^* that yields the model with the best performance on the validation set.

Grid search is one of the easiest and simplest hyperparameter optimization techniques. It systematically searches through a predefined grid of hyperparameter values and evaluates every possible hyperparameter configuration until none is left to assess Baratchi et al. [2024]. Continuous hyperparameters need to be mapped to discrete values in the grid for grid search to work. Also, continuous hyperparameters become problematic in a high-dimensional search space. With every addition of another hyperparameter, the completion time scales linearly $\mathcal{O}(N^m)$, where N is the number of values for each hyperparameter, and m is the number of hyperparameters.

Other hyperparameter optimization techniques are evolution strategies and genetic algorithms; these algorithms are a part of the subset of evolutionary algorithms. An evolutionary algorithm is a population-based optimization method that uses mechanisms inspired by biological evolution, such as selection, crossover, and mutation, to find Pareto-optimal solutions for multi-objective problems by simultaneously exploring multiple trade-offs among the objectives [Deb et al., 2002]. To illustrate, evolution strategies and genetic algorithms are beneficial in black-box settings such as hyperparameter optimization because they do not require gradients. An individual in the evolutionary algorithm terminology represents a single hyperparameter configuration; the population is the set of currently maintained configurations. The performance score of an individual typically measures the fitness of the individual. For instance, if a model's generalization error is 0.1, the fitness score could be 10 (i.e., the reciprocal of the error). The NSGA-II algorithm is an example of a genetic algorithm. NSGA-II improves upon its predecessor by incorporating fast, non-dominated sorting, elitism, and a crowding distance mechanism, thereby efficiently finding and preserving a diverse set of Pareto-optimal solutions [Deb et al., 2002]. It utilizes a binary tournament selection based on rank and crowding distance to balance convergence and diversity in the population.

2.2 Machine learning with a reject option

Machine learning models with a reject option can assess the confidence in each prediction; these models can abstain from making a prediction when they are likely to make a mistake [Hendrickx et al., 2021]. Machine learning with rejection extends the output space with $\textcircled{\text{R}}$. This symbol is a new class the classifier can predict or where the classifier abstains from making a prediction. Formally, Hendrickx et al. [2021] describes that a model with rejection $m : \mathcal{X} \to \mathcal{Y} \cup \textcircled{\text{R}}$ is represented by a pair (h, r), where $h : \mathcal{X} \to \mathcal{Y}$ is the predictor and $r : \mathcal{R} \to \mathbb{R}$ is the rejector. The function $m(x)$ outputs the symbol $\textcircled{\text{R}}$ if it abstains from making a prediction. Abstaining happens when the rejector r determines that the predictor is not confident enough in their decision. If this is not the case, $m(x)$ returns the predictor's output:

$$m(x) = \begin{cases} \textcircled{\text{R}} & \text{if the prediction is } \textit{rejected} \\ h(x) & \text{if the prediction is } \textit{accepted} \end{cases} \tag{2.2}$$

There are two types of rejections: ambiguity rejection and novelty rejection. Ambiguity rejection occurs when x falls in a region where the target y is ambiguous [Hendrickx et al., 2021]. This often happens near the decision boundary in classification tasks, where classes overlap, or in regression, where the target variable has high variance. For example, using linear models for a non-linear relationship can result in significant prediction errors. Novelty rejection occurs when x falls in a region with little or no training data. This situation arises when the sampling distribution differs from the actual distribution. It also includes situations with significant class distribution skew or when new distributions emerge after training, like in medical applications with limited data specific to certain demographics [Hendrickx et al., 2021]. For example, a model trained on diseases common in adults may not perform well on children or rare condi-

tions.

There are also three common architectural principles that use the function $m(x)$ differently. In this section, we talk about the separated rejector architecture and dependent rejector architecture. We will elaborate on the integrated rejector architecture in Section 4.

2.2.1 Separated rejector architecture

Chapter 2.2 mentions the separated rejector architecture. Specifically, separated rejectors filter out unlikely examples without relying on the predictor's output. This architecture uses a threshold τ, and the rejector r independently learns to reject certain instances. Setting the threshold τ is essential and can be set with domain knowledge or heuristic methods. Although this architecture offers flexibility and simplicity, its independent nature can lead to suboptimal rejection performance and is generally more suited for novelty rejection than ambiguity rejection.

$$m(x) = \begin{cases} \text{®} & \text{if } r(x) < \tau \\ h(x) & \text{otherwise.} \end{cases} \tag{2.3}$$

Homenda et al. [2014] describes an advanced pattern recognition method that uses the separated rejection architecture using ensembles of support vector machines (SVMs). With this method, Homenda et al. [2014] tries to classify known and foreign elements, where the known elements are in the dataset but foreign elements are not. They want to improve the accuracy and reliability of the classification system. By employing two- and one-class SVMs, the approach effectively minimizes misclassification of instances not in the original dataset. The study underscores the significance of rejection in maintaining system integrity. However, it also highlights the importance of sharing information between the predictor and the rejector results, often resulting in sub-optimal rejection performance.

2.2.2 Dependent rejector architecture

Chapter 2.2 also mentions the dependent rejector architecture. The dependent rejector architecture involves an integrated approach where the rejector uses the predictor's output to make rejection decisions. In this architecture, the rejector is not an independent function of the input X but rather a function of both the input X and the predictor's output. Formally, we can represent the model as follows:

$$m(x) = \begin{cases} \text{®} & \text{if } r(x; h) < \tau \\ h(x) & \text{otherwise.} \end{cases} \tag{2.4}$$

Here, the rejector r considers both the input x and the prediction $h(x)$ to determine whether to accept or reject an example. This integration allows the rejector to use information about the prediction confidence, uncertainty, or other relevant metrics the predictor provides. The learning process for a dependent rejector involves training the rejector alongside the predictor, leveraging the predictor's output to inform rejection decisions. For instance, if a classifier outputs a probability distribution over classes, the rejector can reject examples where the highest

probability is below a certain threshold, indicating low-confidence in the prediction. Zou et al. [2011] demonstrate that their multi-class SVM with the dependent rejector architecture option effectively handles novelty and ambiguity. This method enhances the reliability of fault diagnosis in steam turbine generators by abstaining from low-confidence predictions. However, this integration increases system complexity, necessitating more sophisticated training techniques to maintain optimal performance.

2.3 Explainable AI

As discussed by Arrieta et al. [2020], explainable artificial intelligence (XAI) focuses on making the decision-making processes of AI models transparent and understandable to human users. Often, machine learning models act as a black box, offering no clarity on how they reach specific outcomes. Explainable artificial intelligence seeks to address this by developing methods to explain these models' workings. These methods can be broadly classified into local and global explainable artificial intelligence. We acknowledge that there are also other explainable artificial intelligence methods like counterfactual methods; this is visualized in Figure 2.1.

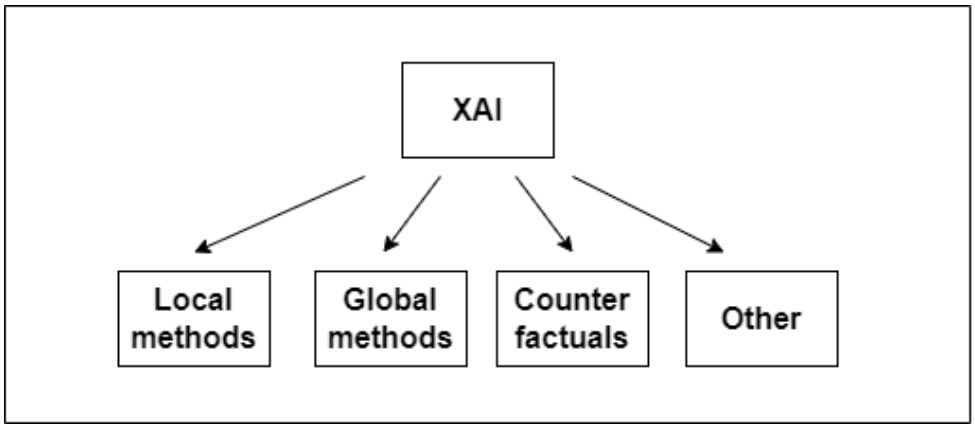

Figure 2.1: Schematic overview of explainable artificial intelligence methods.

Local explainable artificial intelligence focuses on explaining individual predictions made by a model, providing insights into why a specific decision was made for a particular instance rather than the model's overall behavior. One such method is local interpretable model-agnostic explanations (LIME), which creates a simple, interpretable model, like a small decision tree, to approximate the black-box model's behavior for a specific data point. The optimization problem can be formulated as follows, as described by Ribeiro et al. [2016]:

$$\arg\min_{g \in G} \mathcal{L}(f, g, \pi_x) + \Omega(g) \tag{2.5}$$

Where G is the family of interpretable models, $\mathcal{L}(f, g, \pi_x)$ measures the fidelity of the surrogate model g to the original model f in the vicinity of the instance x, weighted by the locality kernel π_x and $\Omega(g)$ is a regularization term to ensure the interpretability of the model g.
Global explainable artificial intelligence aims to understand a model's behavior comprehensively. It seeks to explain how the model makes decisions across all instances, providing insights into the model's structure and logic. A commonly used method in global explainable

artificial intelligence is the calculation of feature importance, which quantifies the contribution of each feature to the model's predictions. A technique for computing feature importance is permutation feature importance, formulated from Breiman [2001]:

$$FI_i = \frac{1}{n} \sum_{j=1}^{n} \left[\mathcal{L}(f, D) - \mathcal{L}(f, D^{\mathsf{perm}(i)}) \right] \tag{2.6}$$

Where $\mathcal{L}$ is a chosen loss function, for example, mean squared error, D represents the original dataset, $D^{\mathsf{perm}(i)}$ denotes the dataset with the values of the i-th feature permuted, n is the number of samples. We propose to categorize machine learning techniques with a reject option under the category of global explainable artificial intelligence. This is because such techniques learn to make decisions for all instances, explaining why certain instances are rejected.

2.4 Generative AI

Generative methods are a subset of artificial intelligence techniques focused on creating new data. These methods, including generative adversarial networks [Goodfellow et al., 2014] and variational autoencoders [Kingma and Welling, 2014], model the underlying distribution of data to generate new instances such as images. These models are adept at producing novel outputs that maintain the characteristics of the original data, offering applications in creative industries and data augmentation [Murphy, 2012]. In contrast, predictive AI focuses on analyzing existing data to make informed predictions and decisions and does not create new data. Techniques such as regression analysis and decision trees are employed in predictive AI to predict or classify data [Bishop, 2007].

As mentioned in Chapter 1, GASTeN is a framework and another method that can generate realistic images on the decision boundary of two classes. It incorporates a target classifier's output into the generator's loss function. Figure 2.2 shows GASTeN's architecture and how GASTeN uses the given classifier. The GASTeN framework consists of three main components:

- **Generator (G)**: Similar to a standard generative adversarial network, the generator tries to produce realistic images ($\hat{x}$) from random noise vectors, in our case, the vector z. To address issues such as mode collapse that are common in generative adversarial networks, GASTeN uses the deep convolutional generative adversarial network architecture introduced by Radford et al. [2016]. This architecture incorporates fractionally-strided convolutional layers, which help in generating high-quality images. The generator aims to produce realistic images on the decision boundary of two classes. It achieves this by mapping the noise vector (z) to the data distribution of real images (x), guided by the loss from the discriminator and the classifier.

- **Discriminator (D)**: The discriminator distinguishes between real images (x) from the dataset S and generated images $\hat{x}$. The discriminator uses normal convolutional layers. Moreover, the objective is to classify real and fake images correctly. The output of the discriminator is a probability indicating whether an image is real or fake. This output helps the generator learn to generate images similar to images from the dataset (S).

- **Classifier (C)**: The classifier is the addition to the GASTeN framework. The classifier is only used to classify the images of the generator and is trained on the binary dataset the generator tries to generate. This output can optimize the generator for a new optimization objective.

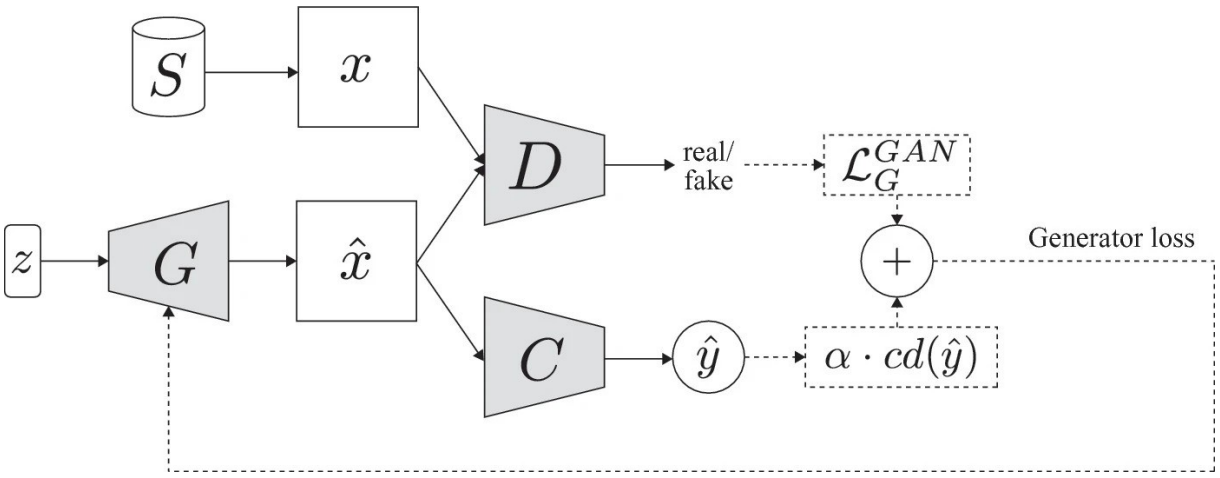

Figure 2.2: Schematic overview of the architecture of GASTeN. Figure taken from Cunha et al. [2023]

The generator's loss function in GASTeN is modified to incorporate the classifier's output. The new loss function is given by:

$$\mathcal{L}_G^{GASTeN} = \mathcal{L}_G^{GAN} + \alpha \cdot cd(C(G(z))) \tag{2.7}$$

Where $\mathcal{L}_G^{GAN}$ is the standard loss from the generator, which aims to deceive the discriminator D, alpha (α) is a hyperparameter that scales the contribution of the confusion distance term. cd denotes the confusion distance, which measures the deviation of the classifier's output from the decision boundary. With the confusion distance, we can calculate the average confusion distance (ACD) by calculating the mean of all the confusion distance values across all instances. Cunha et al. [2023] introduced this metric and measures a sample's closeness to the decision boundary.

Chapter 3 Related Work

This chapter will discuss related approaches to generating ambiguous or low-confidence instances. Additionally, we will review relevant work on other hyperparameter optimization techniques. Finally, we will examine techniques designed to avoid predicting when dealing with datasets containing low-confidence instances.

3.1 Generative adversarial networks

A generative adversarial network is a framework where two neural networks, a generator G and a discriminator D, are trained in an adversarial process to produce data resembling a given dataset [Goodfellow et al., 2014]. The generator tries to confuse the discriminator into misclassification by generating images that seem real. The discriminator needs to guess which image is actual and which is fake. The generator aims to minimize the probability that the discriminator can correctly distinguish samples. The discriminator seeks to maximize that same probability. The discriminator and generator play the following two-player minimax game with value function $V(G, D)$, taken from Goodfellow et al. [2014]:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[logD(X)] + \mathbb{E}_{z \sim p_z(z)}[log(1 - D(G(z)))] \tag{3.1}$$

Where G and D represent the generator and the discriminator, respectively, and $V(D, G)$ is the value function we try to optimize. The term $\mathbb{E}_{x \sim p_{data}(x)}[logD(X)]$ is the expected value of $logD(X)$, representing the average log probability that the discriminator correctly identifies real data samples when x is drawn from the real data distribution $p_{data}(x)$. $\mathbb{E}_{z \sim p_z(z)}[log(1 - D(G(z)))]$ is the expected value of $log(1 - D(G(z)))$ representing the average log probability that the discriminator correctly identifies generated samples as fake when z is drawn from the noise distribution.

Training a generative adversarial network can be challenging due to mode collapse and the inherent instability of the network. Mode collapse happens when the generator is trained too much without updating the discriminator. When this happens, the generator will generate samples with limited diversity [Goodfellow et al., 2014]. The training process becomes much more stable with convolutional layers. Deep convolutional generative adversarial network (DCGAN) extends the generative adversarial network framework by using convolutional layers instead of fully connected ones [Radford et al., 2016]. For example, the generator consists of fractionally-strided convolutional layers. Fractionally-strided convolutional layers apply a filter to the input, but instead of reducing spatial dimensions, they increase them. These fractionally-strided convolutional layers allow the generator to learn its upsampling process. Furthermore, the discriminator uses standard convolutional layers to downsample the images. Figure 3.1 shows a deep convolutional generative adversarial network. The figure illustrates how the generator increases the spatial dimensions of the initial vector to generate a feature map with more significant spatial dimensions than the previous one.

In addition to generative adversarial networks, variational autoencoders provide another robust framework for generative modeling. Variational autoencoders, introduced by Kingma and Welling [2014], combine principles from variational inference and deep learning to generate new data points similar to a given dataset. Unlike generative adversarial networks, variational autoencoders use an encoder to map input data to a latent space, from which a decoder reconstructs the input data. This probabilistic approach allows variational autoencoders to capture uncertainty and generate diverse outputs. The variational autoencoders' reparameterization trick ensures we can train the model using standard gradient-based optimization methods. This trick makes it an efficient alternative to the adversarial training in GANs.

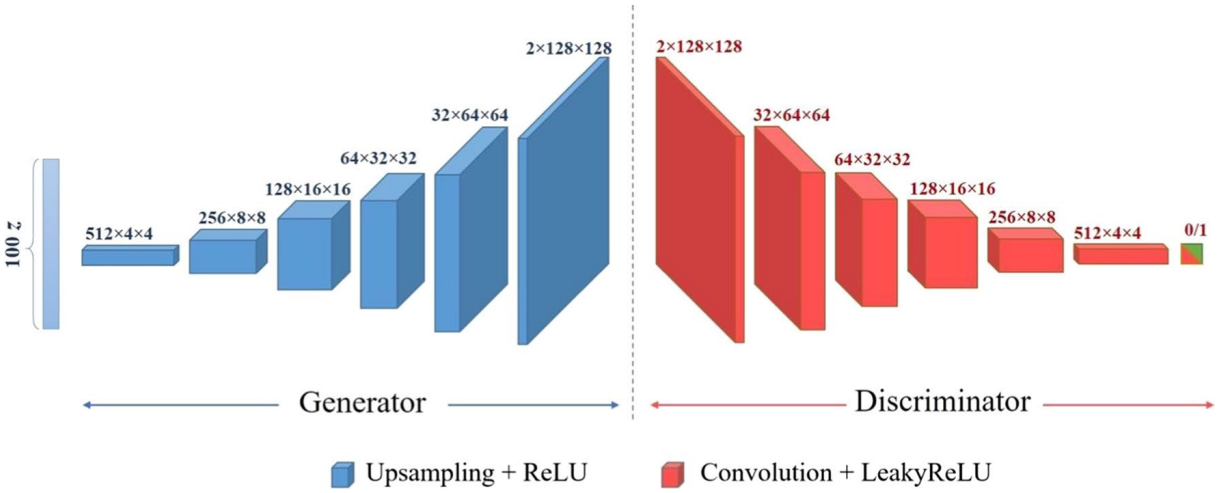

Figure 3.1: Example of a deep convolutional generative adversarial network, where the generator uses fractionally-strided convolutional layers and the discriminator uses normal convolutional layers. Figure taken from Karimi-Bidhendi et al. [2020]

3.2 AmbiGuess

Weiss et al. [2023] introduce AmbiGuess, a novel technique for creating labeled ambiguous images tailored for image classification tasks. Unlike other approaches, AmbiGuess prioritizes creating ambiguous data that not even a human domain expert can confidently predict the class. AmbiGuess uses an autoencoder and an encoder-decoder architecture. The encoder compresses the image into a smaller latent space. After this, a decoder reconstructs the compressed information back into an image. The discriminator receives the encoded image and decides if the image is ambiguous enough. Furthermore, the discriminator learns to differentiate encoded images from samples drawn from latent space while the encoder attempts to fool the discriminator. Weiss et al. [2023] train the autoencoder and the discriminator on only two classes from a dataset. For example, in vast real-world datasets, ambiguity is much more prevalent between some combination of two classes than between multiple others, so not all pairwise combinations are equally attractive for the generation [Weiss et al., 2023]. For example, when a medical diagnostic system tries to classify various skin conditions. Psoriasis is a chronic autoimmune condition that causes rapid skin cell growth and scaly patches. In some

cases, psoriasis can be challenging to distinguish from eczema because the conditions look similar, hence having actual ambiguity. There are similarities between eczema and a common birthmark. So, a genuinely ambiguous image between those two conditions is hard to imagine.

3.3 Dirty MNIST dataset

The dirty MNIST dataset, introduced by Mukhoti et al. [2023], is a modified version of the standard MNIST dataset to illustrate the challenges associated with uncertainty in machine learning models. While MNIST consists of well-defined handwritten digits, dirty MNIST includes additional ambiguous digits, also called ambiguous MNIST, created to have multiple plausible labels. This modification makes the dataset particularly useful for testing and evaluating models' ability to handle and quantify uncertainty in their predictions. The dataset is a benchmark for assessing models designed to disentangle aleatoric and epistemic uncertainty [Hüllermeier and Waegeman, 2021]. Aleatoric uncertainty arises from inherent noise in the data, such as the ambiguity in labeling the digits, while epistemic uncertainty comes from the model's lack of knowledge about the data. For instance, Mukhoti et al. [2023] proved that the deep deterministic uncertainty method effectively distinguishes between these two types of uncertainty. Deep deterministic uncertainty outperforms other models in handling both in-distribution and out-of-distribution data by employing spectral normalization and residual connections, thereby validating its approach in a practical setting.

Chapter 4 Methodology

This chapter describes the algorithms and techniques used. First, we discuss the hyperparameter optimization techniques. Next, we explain how GASTeN works and the process of generating a custom dataset using clustering techniques with AutoGASTeN and a clustering technique. Finally, we detail the architecture used for the machine learning model with a reject option so that another neural network can reject the low-confidence instances.

4.1 Hyperparameter optimization techniques

First, we define our hyperparameter optimization problem more precisely. In the next section, we will explain that GASTeN's training process consists of two steps. This is why we face two optimization challenges: one for the first training step and another for the second training step. The problem for the first training step is to find the optimal hyperparameters by minimizing the Fréchet inception distance across the training dataset, as shown below:

$$\lambda^* \in \arg\min_{\lambda \in \Lambda} \mathcal{L}_{\mathsf{FID}} \left(A_\lambda(\mathcal{D}^{(i)}_{\mathsf{train}}) \right) \tag{4.1}$$

For the second training step, the goal is to generate a Pareto front of Pareto optimal solutions by simultaneously minimizing two objectives, the Fréchet inception distance and average confusion distance, across the same datasets:

$$\lambda^* \in \mathsf{Pareto} \left\{ \left(\mathcal{L}_{\mathsf{FID}}(A_\lambda, \mathcal{D}^{(i)}_{\mathsf{train}}), \mathcal{L}_{\mathsf{ACD}}(A_\lambda, \mathcal{D}^{(i)}_{\mathsf{train}}) \right) : \lambda \in \Lambda \right\} \tag{4.2}$$

We will explain these training steps and the Pareto front in more detail in the following sections.

4.1.1 Random grid search

As mentioned in Chapter 2.1, grid search is a straightforward search technique for hyperparameter optimization. In contrast to grid search, random grid search is an alternative method for hyperparameter optimization that can be more efficient, especially in high-dimensional spaces. Random grid search involves sampling a fixed number of hyperparameter combinations from a predefined grid rather than evaluating every possible combination [Bergstra and Bengio, 2012]. This approach helps reduce the computational resources needed while exploring diverse configurations. However, the predefined grid can limit the exploration of the hyperparameter space, as it restricts the sampled values to those specified in the grid. This problem can lead to missing configurations if they lie outside the predefined values. Despite this limitation, random grid search can still provide diverse configurations within a computationally feasible framework, making it a practical choice in many scenarios. Our implementation for random grid search is a simple algorithm that samples random discrete values from a predefined grid for a random grid search. We use the same algorithm for random search but sample values uniformly within the specified range. This algorithm can easily create the Pareto front based on all the return values of random grid search and random search.

4.1.2 Random search

Random search, in contrast to grid search, does not rely on a predefined grid of hyperparameter values. Instead, it samples hyperparameters directly from specified ranges, allowing for a more flexible exploration of the hyperparameter space. This method can sample from continuous distributions, providing a broader and potentially more optimal set of hyperparameter configurations. By not being constrained to a predefined grid, random search increases the likelihood of discovering high-performing configurations that grid-based methods might miss. Studies have demonstrated that random search is often more effective than grid search in finding optimal hyperparameters, particularly in high-dimensional spaces. For instance, Bergstra and Bengio [2012] highlighted that random search could outperform grid search in many scenarios by efficiently exploring the hyperparameter space. This flexibility makes random search a valuable tool for hyperparameter optimization in machine learning.

4.1.3 Bayesian optimization

Bayesian optimization is a hyperparameter optimization technique for optimizing expensive-to-evaluate black-box functions [Hutter et al., 2011]. Unlike random or grid search, Bayesian optimization builds a surrogate model to estimate the objective function and systematically explore the hyperparameter space. This method involves selecting the most promising hyperparameter configurations based on the surrogate model and iteratively refining the model with new data points. Snoek et al. [2012] demonstrated that Bayesian optimization could outperform traditional methods by efficiently balancing exploration and exploitation. By incorporating prior knowledge and updating the model with observed results, Bayesian optimization can focus on the most promising regions of the hyperparameter space, thereby finding optimal configurations with fewer evaluations. This efficiency makes it particularly useful for high-dimensional and expensive-to-evaluate functions, making Bayesian optimization a powerful tool for hyperparameter tuning in machine learning.
Since our second hyperparameter optimization problem involves two objectives, we require an algorithm capable of minimizing both objectives. For this purpose, we use the Python package SMAC3, developed by Lindauer et al. [2022], which offers a multi-objective optimization algorithm. Currently, the algorithm uses a mean aggregation strategy to combine multiple objectives into a single scalar objective. This aggregated objective is then passed to the surrogate model for the optimizing process. This approach is similarly applied in hyperband and BOHB, where both methods select the best configurations based on the scalarized single objective.

4.1.4 Hyperband

Hyperband is an innovative method for hyperparameter optimization, extending the successive halving algorithm. It improves traditional hyperparameter optimization techniques by using a resource allocation approach. This resource allocation approach is called successive halving. Successive halving uniformly allocates a budget to a set of hyperparameter configurations. For example, given a total budget of 80 and 8 hyperparameter configurations, each configuration is initially allocated a budget of $80/8 = 10$. This budget can be interpreted as the number of epochs or the amount of runtime available. Hyperband does not evaluate all configurations

equally; it dynamically allocates more resources to promising configurations and early-stops those that perform poorly. This technique aims to maximize the efficiency of hyperparameter searches. Hyperband ranks the configurations based on performance, eliminates the worst half, and reallocates resources to the remaining configurations. This process continues iteratively until only one configuration remains. Hyperband employs successive halving in the algorithm's inner loop with a fixed number of configurations n and a fixed budget r. The outer loop iterates over different values of n and r, reducing n by a factor of η in each iteration. Li et al. [2017] introduced hyperband, demonstrating its ability to drastically reduce the computational resources required compared to standard grid and random search methods. By rapidly narrowing the search space and focusing on the most promising configurations, hyperband is particularly effective in high-dimensional settings where exhaustive search is impractical. This method's ability to balance exploration and exploitation efficiently makes it a valuable tool for hyperparameter optimization.

4.1.5 BOHB (Bayesian optimization and hyperband)

BOHB (Bayesian optimization and hyperband) combines the strengths of Bayesian optimization and hyperband to create a robust hyperparameter optimization method. BOHB uses Bayesian optimization to model the objective function and predict promising configurations, while hyperband's resource allocation strategy guarantees efficient exploration and exploitation of the hyperparameter space. This approach allows BOHB to leverage the benefits of both methods, providing a balance between the thoroughness of Bayesian optimization and the resource efficiency of hyperband. Falkner et al. [2018] demonstrated that BOHB could outperform both Bayesian optimization and hyperband individually by achieving better performance with fewer computational resources. By integrating these two techniques, BOHB offers an efficient solution for hyperparameter tuning, making BOHB suitable for complex machine learning tasks with large hyperparameter spaces.

4.2 AutoGASTeN

The training of GASTeN begins with initializing the generator and discriminator with random weights. In the first training step, we train the generative adversarial network using the standard loss function for ten epochs to generate realistic images. In the second training step of GASTeN, we introduce the classifier. After this, we can further train the generator and discriminator from the first training step, using the GASTeN loss function for an additional forty epochs. During each iteration of this phase, synthetic images ($\hat{x}$) are generated by the generator from the random noise vector (z). We then update the discriminator with these images and compute the confusion distance. Finally, we calculate the generator loss ($\mathcal{L}_G^{GASTeN}$) by combining the standard generative adversarial network loss with the scaled confusion distance term, and the generator is updated accordingly. We can automate the process using hyperparameter optimization techniques by providing the two classes and the dataset. This automated process ensures that GASTeN consistently generates realistic, low-confidence images. We will refer to this automated approach as AutoGASTeN. We want AutoGASTeN to perform at its best to produce realistic images of the decision boundary between two classes. With the

previously mentioned hyperparameter optimization techniques, we can improve GASTeN's performance. By evaluating various hyperparameter optimization techniques, we identify the one that most increases the performance. Subsequently, the automated pipeline consists of the training process plus the hyperparameter optimization technique.

4.2.1 Optimization criteria

We evaluate GASTeN with two metrics: the Fréchet inception distance and the previously mentioned average confusion distance. Introduced by Heusel et al. [2017], the Fréchet inception distance compares the generated and real image distribution using deep features extracted from a pre-trained inception v3 network. Specifically, it calculates the Fréchet distance between two multivariate Gaussian distributions representing the activations of the real and generated images in the Inception network's feature space. The formula for the Fréchet inception distance is:

$$\text{FID} = \|\mu_r - \mu_g\|^2 + \text{Tr}(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{1/2}) \tag{4.3}$$

Where μ_r and μ_g are the mean vectors, Σ_r and Σ_g are the covariance matrices of the real and generated image feature vectors, respectively, and Tr stands for the trace of the matrix, which is the sum of the elements on the main diagonal. Lower Fréchet inception distance values suggest that the generated images are more similar to the actual images, thus reflecting higher image quality and more significant similarity in data distributions.
The confusion distance, introduced by Cunha et al. [2023], guides the generator to produce images near the classifier's decision boundary. In the context of binary classification, the classifier produces a scalar value between 0 and 1. The confusion distance measures the deviation of the classifier's prediction from a decision threshold, typically set at 0.5. For a given prediction from the classifier $C(G(z))$, where $G(z)$ represents the generated image, we calculate the confusion distance as $cd(C(G(z))) = |C(G(z)) - 0.5|$. This value quantifies how close the classifier's output is to the decision boundary, indicating the level of uncertainty in the prediction.

4.3 Multi-objective decision making

We need to determine the optimal hyperparameter configuration for AutoGASTeN. This involves minimizing two objectives: the Fréchet inception distance and the average confusion distance. This process generates a Pareto front of hyperparameter configurations after the second training step. A Pareto front represents a set of optimal solutions in a multi-objective optimization problem, where no single solution is better in all objectives. In our context, this illustrates the trade-offs between the Fréchet inception distance and the average confusion distance. Ultimately, we aim to produce images with low Fréchet inception distance and low average confusion distance. However, due to the trade-off mechanism, if the Fréchet inception distance is low, the average confusion distance tends to be higher, and vice versa. The Pareto front, shown in Figure 4.1, consists of synthetic hyperparameter configurations that balance

these competing objectives, visually representing the trade-offs involved.

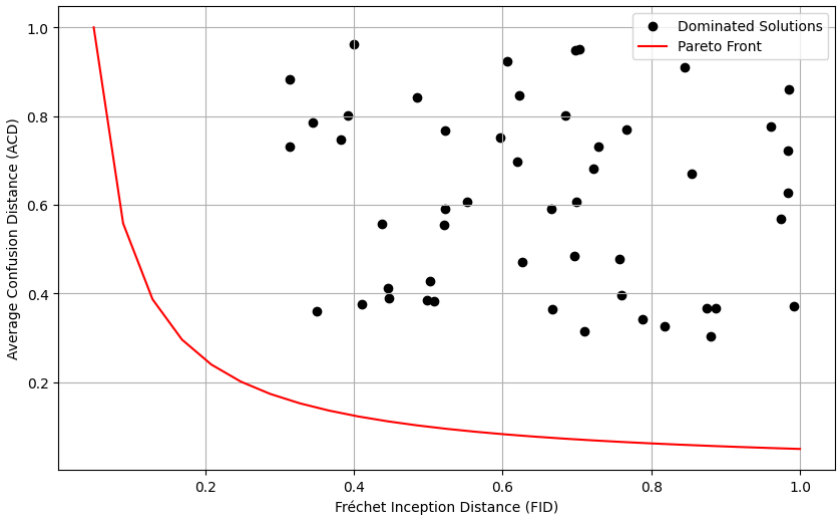

Figure 4.1: This illustration of the Pareto front serves as an example of the trade-off mechanism between the Fréchet inception distance and average confusion distance using hypothetical data points. Black marks represent dominated solutions, while red marks highlight the Pareto front.

We decided on the knee point in the Pareto Front to choose the optimal hyperparameter configuration. The knee or elbow point is a concept used in optimization and data analysis to identify a point on a curve where the curve begins to flatten Satopaa et al. [2011]. This point represents a trade-off between two competing objectives. It is often chosen as an optimal point because it balances improving one objective without significantly worsening the other.

4.4 Clustering and image generation

With the optimal hyperparameter configuration, we can generate low-confidence images and cluster them. Gomes et al. [2024] introduced a clustering technique that involves using classifier embeddings to cluster similar images and extract the medoid of each cluster. We need this clustering to generate the custom dataset. The medoid represents the type of image that challenges the evaluated model. However, for our purposes, we only require the clustering component of this technique. We use AutoGASTeN to generate the images needed. Then, we can cluster these images, hand-pick the low-confidence clusters, and use them to create a custom dataset. AutoGASTeN automates the image generation process, which integrates well with the clustering approach. The custom dataset consists of the binary dataset and the low-confidence instances generated by AutoGASTeN. We use this dataset to train another neural network using machine learning with a reject option technique.

4.5 Integrated rejector architecture

The integrated rejector architecture in machine learning with a reject option combines a predictor and a rejector to improve decision-making reliability. This architecture requires unique algorithms for the simultaneous learning of predictor and rejector functions. Formally, the model with an integrated reject option acts as:

$$m(x) \in \mathcal{Y} \cup \textcircled{R} \tag{4.4}$$

The integrated rejector architecture has two main approaches: model-agnostic and model-specific. To begin with, model agnostic integrates rejection into the predictor's decision-making process. Model-specific designs have an objective function of penalizing mispredictions and rejection. Practical implementations, such as neural networks, have demonstrated the value of integrated rejectors in tasks such as medical diagnostics and multi-label classification. They can effectively handle ambiguous cases by rejecting uncertain predictions [Geifman and El-Yaniv, 2019]. Despite their advantages, integrated rejectors demand extensive domain knowledge of the predictor for effective integration. Also, they require intensive computation due to the need for retraining when adjusting the rejector component.
In our implementation, we have adopted a model-agnostic approach to the integrated rejector architecture. This means that the rejection mechanism is incorporated into the decision-making process without being tied to a specific type of model. Our method can be applied across different predictive models, including convolutional neural networks, by adding an additional class for rejection. This flexibility allows the model to reject uncertain predictions based on a predefined rejection threshold, ensuring that the system can handle ambiguous cases effectively. By using a model-agnostic approach, we maintain the versatility of our model while still benefiting from the improved reliability that the integrated rejector architecture offers. We have implemented the integrated rejector architecture by treating the rejection option as an additional class that the model can select. With a binary dataset, this results in three classes: the two normal classes from the dataset and the rejection class (Ⓡ). For a convolutional neural network, this setup leads to three output possibilities. We then set a rejection percentage, meaning if the confidence score for the rejection class reaches 0.1 and the rejection threshold is also 0.1, the model will always reject that instance.

4.6 Practicalities

The majority of this work was produced using the ALICE cluster used by the ADA research group[1] All code used in this work can be downloaded from and is available at: `https://github.com/LuukdeJong123/gasten`

[1] `https://ada.liacs.nl/`

Chapter 5 RQ1: Hyperparameter optimization

This chapter describes the datasets and the precise process for the first research question. We also explain how we evaluate these results and discuss their limitations. Lastly, we note any practicalities regarding this research's experimental setup.

5.1 Experimental setup

This section outlines the datasets we used and explains how we evaluated the first research question.

5.1.1 Dataset

The MNIST dataset is an extensive database of handwritten digits introduced by LeCun et al. [1998]. Image processing systems use this for training and testing purposes. It consists of 60,000 training and 10,000 testing images, each of which is a grayscale image of 28x28 pixels. The digits fall into ten classes, from zero to nine. Subsequently, the fashion-MNIST is a dataset of Zalando's article images created by Xiao et al. [2017]. The intent is to be a direct drop-in replacement for the original MNIST dataset for benchmarking machine learning algorithms. Fashion-MNIST consists of the same number of training and testing images as MNIST, also in grayscale and of the same size. However, the images in fashion-MNIST represent different clothing items, which fall into ten classes, such as T-shirts, trousers, and shoes. Table 5.1 shows the training and testing instances distribution for every class of the MNIST and fashion-MNIST datasets. Additionally, the MNIST and fashion-MNIST datasets are easy to understand and do not require extensive domain knowledge.

5.1.2 Evaluation

In the first research question, we want to measure the impact of hyperparameter optimization techniques on GASTeN's performance. GASTeN's training process consists of two steps. To evaluate every hyperparameter optimization technique (random grid search, random search, Bayesian optimization, hyperband, and BOHB) equally, we give each technique twenty hours of runtime for the first training step and eighty hours of runtime for the second training step. The second training step gets more runtime because the training process uses four times as many epochs. This amount of runtime ensures that each technique can evaluate a sufficient number of hyperparameter configurations, which provides an understanding of the technique's performance. The metrics we use to evaluate the performance are the Fréchet inception distance and average confusion distance. We focus only on the Fréchet inception distance in the first training step because the classifier is not present yet. A lower Fréchet inception distance value indicates a better performance for the hyperparameter optimization technique. We evaluate the Fréchet inception distance and average confusion distance metrics during the second

Table 5.1: Distribution of training and testing instances per class in MNIST and fashion-MNIST datasets

Class	MNIST		Fashion-MNIST	
	Train	Test	Train	Test
0	5 923	980	6,000	1,000
1	6,742	1,135	6,000	1,000
2	5,958	1,032	6,000	1,000
3	6,131	1,010	6,000	1,000
4	5,842	982	6,000	1,000
5	5,421	892	6,000	1,000
6	5,918	958	6,000	1,000
7	6,265	1,028	6,000	1,000
8	5,851	974	6,000	1,000
9	5,949	1,009	6,000	1,000
Total	60,000	10,000	6,0000	10,000

training step. Given the long-lasting runtime in the second training step, resource efficiency is also a consideration. For example, the optimal hyperparameter optimization technique quickly achieves a low Fréchet inception distance in the first training step. It maintains a low Fréchet inception distance alongside a low average confusion distance in the second step while not adding noise or sacrificing the realness. Additionally, we manually inspect generated clusters through visualizations to confirm that the instances are, in fact, low-confidence and look real. We then select the best-performing hyperparameter optimization technique based on these metrics.

As discussed in Section 4.1, we use both the simple algorithms for random grid search and random search and SMAC3 for the other hyperparameter optimization techniques in this experiment. The selected hyperparameters are shown in Table 5.2. During the first training step, the hyperparameters α and the number of filters (*nf*) are excluded from the hyperparameter grid because AutoGASTeN does not use the classifier yet. So, α and the number of filters are not applicable. The second training step utilizes all hyperparameters except for the number of blocks (*n_blocks*). The number of blocks is not used because the architecture of the best generator from the first training step is fixed and cannot change in the second training step. Our baseline for the first research question is training GASTeN with the hyperparameter configuration given by Cunha et al. [2023]. We compare the baseline against the results of the hyperparameter optimization techniques.

5.2 Results

In this section, we present the baseline results and the outcomes of all the hyperparameter optimization techniques, followed by a discussion of these results.

Parameter	Description	Range
G_lr	Generator learning rate	1e-4 to 1e-3
D_lr	Discriminator learning rate	1e-4 to 1e-3
G_beta1	Generator beta1	0.1 to 1
D_beta1	Discriminator beta1	0.1 to 1
G_beta2	Generator beta2	0.1 to 1
D_beta2	Discriminator beta2	0.1 to 1
n_blocks	Number of blocks	1 to 5
α	Factor for the confusion distance	1 to 30
nf	Number of filters for the classifier	[1,2,4,8]

Table 5.2: Parameter grid for hyperparameter optimization

5.2.1 Baseline

Figure 5.1 shows the baseline results of the first training step of GASTeN with the binary MNIST dataset 8 versus 0. The figure shows a stable training process over three runs, with a 95 percent confidence interval, represented by the shaded area. The three runs narrowed considerably after the sixth epoch, indicating reduced variability and consistent performance. After ten epochs, the Fréchet inception distance reaches a value of 17.74.

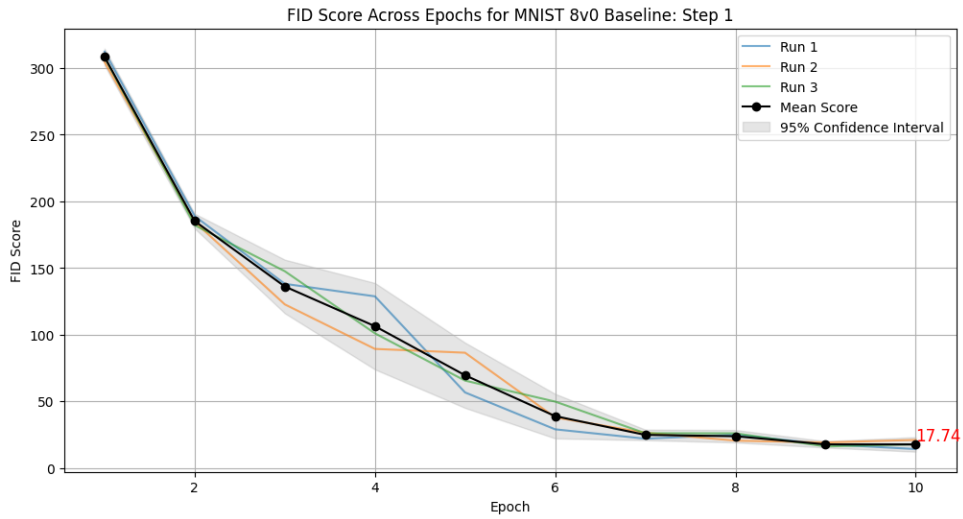

Figure 5.1: Baseline results for the MNIST 8 versus 0 dataset, step 1: Mean and standard deviation across three runs.

Figure 5.2 shows the baseline results of training step 2 of GASTeN with the binary MNIST dataset 8 versus 0. The figure shows a less stable training process over three runs compared to the first training step. Moreover, the 95 percent confidence interval varies greatly in the first ten epochs. Following this point, the training process stabilizes with no further improvement. After forty epochs, the Fréchet inception distance and average confusion distance values reach 12.71 and 0.42, respectively.

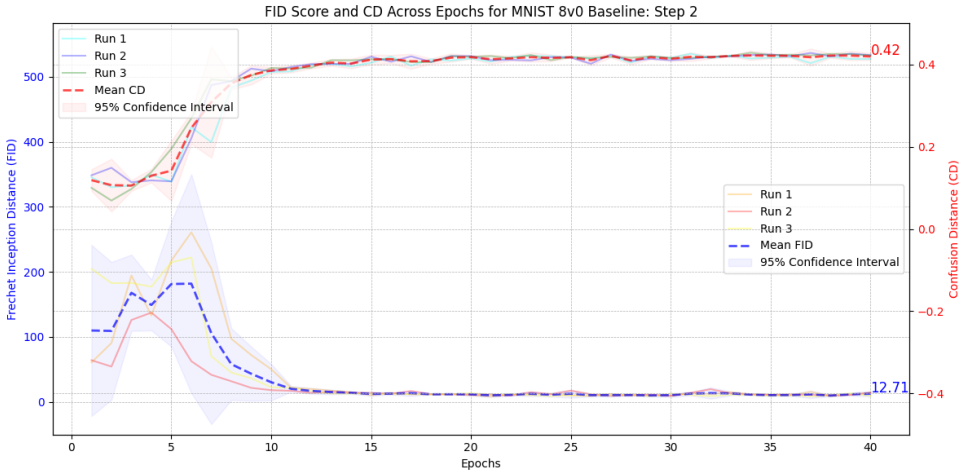

Figure 5.2: Baseline results for the MNIST 8 versus 0 dataset, step 2: Mean and standard deviation across three runs.

5.2.2 Hyperparameter optimization technique comparison

Figure 5.3 shows a convergence graph for every hyperparameter optimization technique with the Fréchet inception distance on a log scale y-axis and iterations on the x-axis. One iteration on the x-axis means an evaluation of a hyperparameter configuration. Every hyperparameter optimization technique has twenty hours of runtime, but random grid search and random search have more iterations. This is because of the more straightforward implementation, thus reducing the overhead code that can slow down the process. All hyperparameter optimization techniques significantly reduced the Fréchet inception distance value within the first fifty iterations. Specifically, Bayesian optimization and BOHB performed better, stabilizing with a lower Fréchet inception distance value of around ten iterations. While also effective, random grid search, hyperband, and random search are slower in stabilizing. Also, every hyperparameter optimization technique reaches a better Fréchet inception distance value than the baseline results, with Bayesian optimization reaching a Fréchet inception distance value of 8.69.
Figure 5.4 shows a cumulative distribution function for every hyperparameter optimization technique for the first training step. The y-axis represents the cumulative percentage of iterations that have achieved the Fréchet inception distance value up to a certain point. For example, at fifty percent, a Fréchet inception distance value of around 15 is marked, which means that fifty percent of the iterations have achieved a Fréchet inception distance value less than or equal to that specific value. This graph shows that Bayesian optimization, hyperband, and BOHB can find many low Fréchet inception distance values more than random grid search and random search.
Figure 5.5 shows a scatter plot plus a hexbin plot for Bayesian optimization in the second training step. The graphs depict the relationship between the Fréchet inception distance and average confusion distance for Bayesian optimization. Both the scatter and hexbin plots reveal two distinct groups. The first group, located on the left with lower Fréchet inception distance

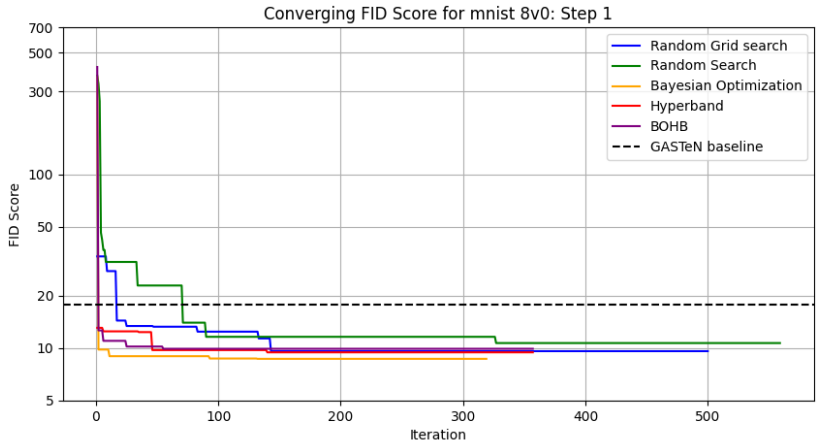

Figure 5.3: Comparison of hyperparameter optimization techniques on the binary MNIST dataset 8 versus 0: Convergence of Fréchet inception distance value for the first training step.

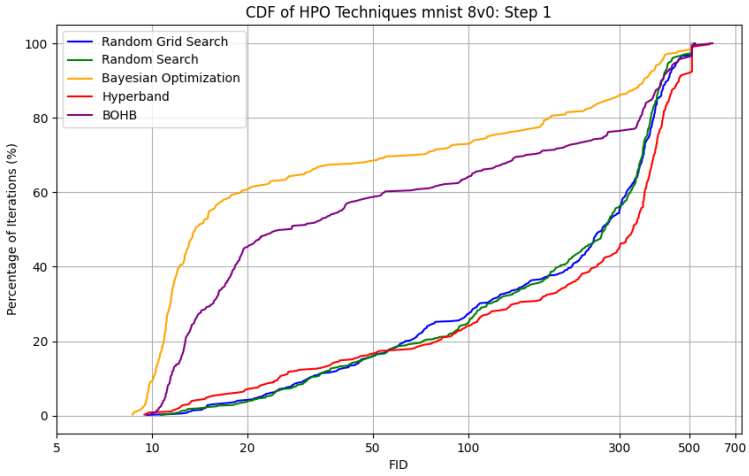

Figure 5.4: Cumulative distribution function of different hyperparameter optimization techniques for the binary MNIST dataset 8 versus 0 for the first training step.

and varying average confusion distance values, shows a tighter clustering of points. In contrast, the second group on the right shows higher Fréchet inception distance values and lower average confusion distance values. This pattern is consistent with BOHB but is not observed in random grid search, random search, and hyperband. These methods do not exhibit the two distinct groups. This indicates that Bayesian optimization and BOHB can find these two groups because they can satisfy the second objective with a surrogate model. Random search methods can not find these patterns. The graphs for BOHB, random grid search, random search, and hyperband are available in the appendix.
Figure 5.6 shows the visual inspection of instances with a lower average confusion distance.

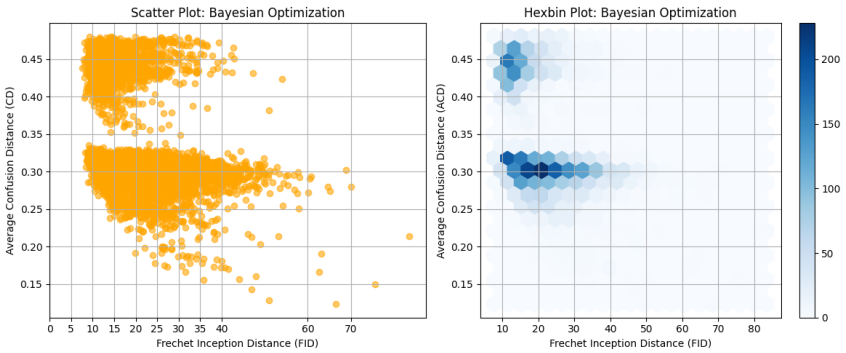

Figure 5.5: Evaluation of BOHB on the binary MNIST dataset 8 versus 0: scatter and hexbin plots of Fréchet inception distance versus average confusion distance

By visually inspecting the instances, we can see that AutoGASTeN added noise to the images, leading to a lower average confusion distance and higher Fréchet inception distance.

Figure 5.6: Visual inspection of instances where the average confusion distance is around 0.25.

Figure 5.7 shows a cumulative distribution function for each hyperparameter optimization technique during the second training step, using the same y-axis as Figure 5.4. This graph shows that Bayesian optimization performs worse than other hyperparameter optimization techniques when considering only the Fréchet inception distance, as it finds a Fréchet inception distance value of 10.12 and an average confusion distance of 0.34. As previously noted, a trade-off mechanism between Fréchet inception distance and average confusion distance indicates that Bayesian optimization struggles to achieve lower Fréchet inception distance values because it also aims to maintain a low average confusion distance. Consequently, optimizing for a low average confusion distance leads to higher Fréchet inception distance values. We can see these results by taking some points from the scatter plot. For example, we have the points $(29.73, 0.29)$ and $(11.99, 0.43)$. From this, we can see that Bayesian optimization can find lower Fréchet inception distance values but also focuses on the other objective, thus finding lower average confusion distance values sacrificing the Fréchet inception distance. This results in AutoGASTeN adding noise to the instances to satisfy the average confusion distance objective.

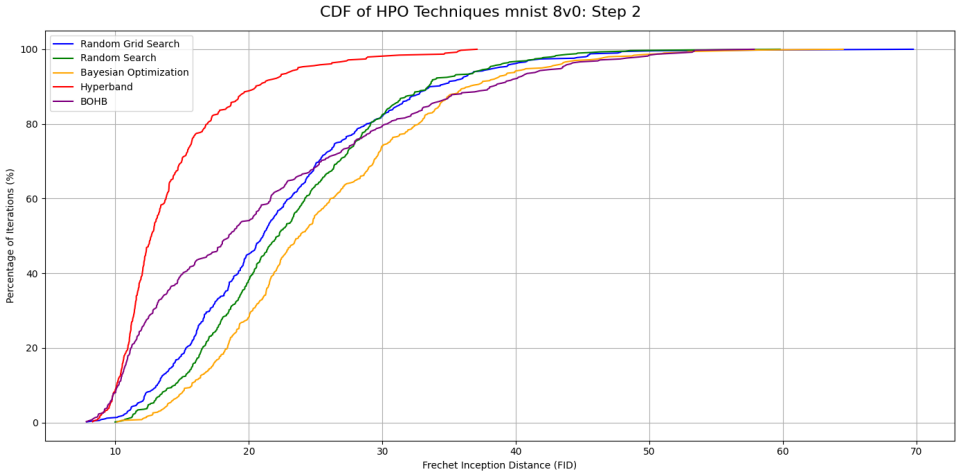

Figure 5.7: Cumulative distribution function of different hyperparameter optimization techniques for the binary MNIST dataset 8 versus 0 for the second training step.

The convergence plot and cumulative distribution function indicate that Bayesian optimization, hyperband, and BOHB are more efficient in identifying optimal hyperparameters, resulting in faster convergence and better overall performance. The rapid decrease and subsequent stabilization of Fréchet inception distance value suggest that these techniques can significantly enhance generative models' training efficiency and quality. Also, every hyperparameter optimization technique finds a better Fréchet inception distance value in the first and second training steps than the baseline results. Even though there is not a significant decrease in performance for the Fréchet inception distance values in the second training step, we still find lower average confusion distance values, suggesting we can discover more low-confidence instances with hyperparameter optimization techniques. We only highlighted the results for the binary MNIST dataset 8 versus 0, but only some of the results. The appendix shows additional results for the MNIST datasets, including 7 versus 1, 5 versus 3, and 9 versus 4 comparisons. Additionally, results for the binary fashion-MNIST dataset, specifically dress versus T-shirt, are provided. Similar outcomes are observed across all other datasets.

5.3 Limitations

Scale benchmark: Another limitation is that these datasets might only partially represent AutoGASTeN's behavior. As datasets become more complex, with color or objects that do not appear similar, the decision boundaries become less clear-cut. For instance, while digits like seven and one have some similarities, it is challenging to draw similarities between objects like a car and a plane. This lack of common ground in decision boundaries can hinder evaluating AutoGASTeN's effectiveness across different datasets. More comprehensive benchmarks involving a variety of complex datasets are needed to understand the limitations.

Chapter 6 RQ2: Machine learning with a reject option

This chapter describes the datasets and the precise process for the second research question. We also explain how we evaluate these results and discuss them and their limitations. Lastly, we note any practicalities regarding this research's experimental setup.

6.1 Experimental setup

This section outlines the datasets we used and explains how we evaluated the second research question.

6.1.1 Dataset

As mentioned in Chapter 4 we generate custom datasets with low-confidence instances to test our second research question. Table 6.1 shows the training and testing distribution for the binary dataset 8 versus 0 of the MNIST dataset with low-confidence instances. There are $5,923$ training instances and 980 testing instances for class 0, $5,851$ training instances and 974 testing instances for class 8, and $5,822$ training instances and $1,028$ testing instances for class . Where if the model classifies the class , it rejects the instance. The appendix shows all the other datasets for the MNIST datasets: 7 versus 1, 5 versus 3, and 9 versus 4. Also, the results are shown for the binary fashion-MNIST dataset dress versus T-shirt. We focused on the binary dataset 8 versus 0, as Cunha et al. [2023] focused on the binary dataset 7 versus 1. Using this dataset, we can demonstrate that any subset of the MNIST dataset applies for hyperparameter optimization.

Table 6.1: Distribution of training and testing instances for the custom MNIST binary dataset 8 versus 0

Class	**Train**	**Test**
Class 0	5,923	980
Class 8	5,851	974
Class ®	5,822	1,028

6.1.2 Evaluation

In the second research question, we want to measure the impact on performance when we train a convolutional neural network with integrated rejector architecture and a dataset that contains low-confidence instances. We give the convolutional neural network one and three epochs to train on the dataset. The metrics we use to evaluate the performance are precision, recall, and F1 score. Ideally, adding low-confidence instances to a dataset helps the performance of a convolutional neural network. It should have the same or a better performance.

To see the impact, we will compare the custom binary datasets to a normal binary dataset without low-confidence instances.

For the implementation, we created a convolutional neural network to classify the classes from the binary dataset and the reject class. We use an architecture that starts with two convolutional layers with the rectified linear unit activation, extracting features from images, followed by max-pooling layers. The feature maps are then flattened into a one-dimensional vector, passed through a dense layer with 128 neurons, and rectified linear unit activation for further processing. Finally, the output layer uses a dense layer with three neurons and softmax activation to classify the input into one of three classes. Our baseline for the second research question is taking a dataset without low-confidence instances versus a custom dataset with low-confidence instances. We will compare the results of a convolutional neural network trained for one and three epochs to determine whether it can accurately reject low-confidence instances without compromising precision, recall, and F1 score.

6.2 Results

In this section, we present the baseline results and the outcomes integrated rejector architecture with a custom dataset that contains low-confidence instances, followed by a discussion of these results.

6.2.1 Baseline

Table 6.2 shows the baseline results for a convolutional neural network trained on the binary MNIST dataset 8 versus 0. Both tables report the precision, recall, and F1 score after one and three training epoch (s), averaged over three runs. They consistently perform with all metrics at or above 0.99 for both classes. This indicates that this binary dataset is easy to classify for a convolutional neural network. Overall, the convolutional neural network can accurately classify the digits '0' and '8', achieving near-perfect scores after just one epoch.

Class	**Epoch 1**			**Epoch 3**		
	Precision	**Recall**	**F1 score**	**Precision**	**Recall**	**F1 score**
0	0.99	0.99	0.99	0.99	0.99	1.00
8	0.99	0.99	0.99	1.00	1.00	1.00

Table 6.2: Evaluation of a convolutional neural network on the binary MNIST dataset 8 versus 0: precision, recall, and F1 score after one epoch and three epochs, averaged over three runs.

Table 6.3 shows the baseline results for the same convolutional neural network trained on the binary fashion-MNIST dataset dress versus T-shirt. The table reports the precision, recall, and F1 score after one and three training epoch(s) averaged over three runs. Table 6.3 shows that after one epoch, the model achieves a precision, recall, and F1 score of 0.95 for both classes. The table also demonstrates slight improvements after three epochs, with precision, recall, and F1 scores rising to 0.97 for both classes. These results suggest that this dataset is more

challenging for a convolutional neural network to classify. Although the model performs well, and additional training epochs further enhance its accuracy, it does not perform better than the MNIST dataset. Overall, the convolutional neural network effectively classifies dresses and T-shirts in the binary fashion-MNIST dataset.

Class	Epoch 1			Epoch 3		
	Precision	Recall	F1 score	Precision	Recall	F1 score
T-shirt	0.95	0.95	0.95	0.97	0.96	0.97
Dress	0.95	0.95	0.95	0.97	0.97	0.97

Table 6.3: Evaluation of a convolutional neural network on the binary fashion-MNIST dataset dress versus T-shirt: precision, recall, and F1 score after one epoch and three epochs, averaged over three runs.

6.2.2 Custom dataset with low-confidence instances

Table 6.4 shows the results for a convolutional neural network with integrated rejector architecture trained on the custom binary MNIST dataset 8 versus 0. In Table 6.4, we can see that the convolutional neural network has a harder time classifying this custom dataset than the baseline dataset. After one epoch, all the classes perform slightly worse than the baseline results. The recall for the reject class is lower than the other classes because the convolutional neural network rejects instances that do not need to be rejected, resulting in lower precision for classes 0 and 8. This indicates that the convolutional neural network has trouble distinguishing low-confidence from real instances. However, after three epochs, the convolutional neural network performed way better. The performance is almost the same as the baseline results. This indicates that the performance increases with more epochs, and the convolutional neural network can better distinguish between the low-confidence and MNIST instances with more training time. These results are obtained without a rejection threshold, relying solely on taking the maximum value of all the probabilities and then classifying or rejecting the instance.

Class	Epoch 1			Epoch 3		
	Precision	Recall	F1 score	Precision	Recall	F1 score
0	0.96	0.99	0.97	0.99	1.00	0.99
8	0.93	0.96	0.94	1.00	0.99	0.99
Ⓡ	0.97	0.90	0.93	0.99	1.00	1.00

Table 6.4: Evaluation of integrated rejector architecture with a convolutional neural network on the custom binary MNIST dataset 8 versus 0: precision, recall, and F1 score after one epoch and three epochs.

Table 6.5 shows the results for a convolutional neural network with integrated rejector architecture trained on the custom binary fashion-MNIST dataset dress versus T-shirt, or dress versus T-shirt. We have already seen that a convolutional neural network performs slightly worse if the complexity of the dataset increases. This case holds because it is even harder to classify the

fashion-MNIST custom dataset than the baseline dataset, shown in Table 6.5. After one epoch, all the classes perform worse than the baseline results. In this case, the convolutional neural network rejects instances that are not low-confidence and classifies instances that the model needs to reject, resulting in lower precision, recall, and F1 scores for all classes. This indicates that the convolutional neural network has more trouble distinguishing low-confidence from real instances with a more complex dataset. However, after three epochs, the convolutional neural network's performance improved significantly, closely matching the baseline results. This indicates that the model's performance increases with more training epochs, allowing the convolutional neural network to better distinguish between low-confidence and MNIST instances with additional training time. These results are obtained without a rejection threshold, relying solely on taking the maximum value of all the probabilities and then classifying or rejecting the instance. Of course, in practice, we would need to assess the most appropriate value using a holdout set. This can be seen as a limitation. The appendix shows all the other results for the custom dataset MNIST with low-confidence instances: 7 versus 1, 5 versus 3, and 9 versus 4. Every other custom dataset with low-confidence instances shows similar results with the integrated rejector architecture.

Class	**Epoch 1**			**Epoch 3**		
	Precision	**Recall**	**F1 score**	**Precision**	**Recall**	**F1 score**
T-shirt	0.94	0.89	0.94	0.94	0.94	0.94
Dress	0.83	0.92	0.87	0.94	0.94	0.94
®	0.91	0.87	0.89	0.98	0.98	0.98

Table 6.5: Evaluation of integrated rejector architecture with a convolutional neural network on the custom binary fashion-MNIST dataset dress versus T-shirt: precision, recall, and F1 score after one epoch and three epochs.

Figures 6.1 and 6.2 show the reject percentage and F1 score versus the reject threshold on the task of rejecting low-confidence instances from custom binary MNIST dataset 8 versus 0 after one and three epoch(s) of training. Both figures show that if the rejection threshold is 0.0, the F1 score decreases significantly. If the rejection threshold is 0.0, the model will always reject the instance if the probability of rejection is above 0.0. After one epoch of training, we can see that with a reject threshold of 0.1, the F1 score is lower than with a higher reject threshold. This indicates that the convolutional neural network is not confident enough because it assigns a probability equal to or higher than 0.1 to the reject class. The rejection percentage is also higher at a reject threshold of 0.1. This result occurs because the overall rejection percentage increases when more instances have the potential to be rejected. We can see that this pattern disappears after three epochs of training. With three epochs of training time and a reject threshold of 0.1, the F1 score is the same as a reject threshold of 1.0, meaning the convolutional neural network is more confident in its decision.
Figures 6.3 and 6.4 show the reject percentage and F1 score versus the reject threshold on the task of rejecting low-confidence instances from the custom binary fashion-MNIST dataset dress versus T-shirt, or dress versus T-shirt, after one and three epoch(s) of training. Both figures show the same pattern as the Figures 6.1 and 6.2. The only difference is at a rejection threshold of 0.3 and 0.4. The higher F1 score at rejection thresholds of 0.3 and 0.4 suggests that

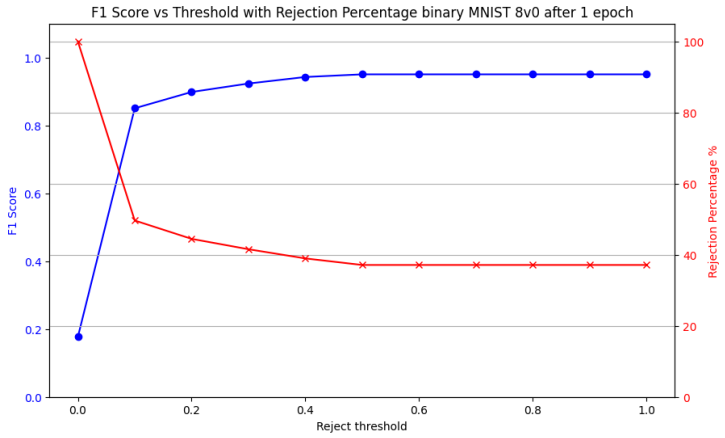

Figure 6.1: Evaluation of reject threshold impact on the custom binary MNIST dataset 8 versus 0: F1 score and Rejection Percentage after one training epoch.

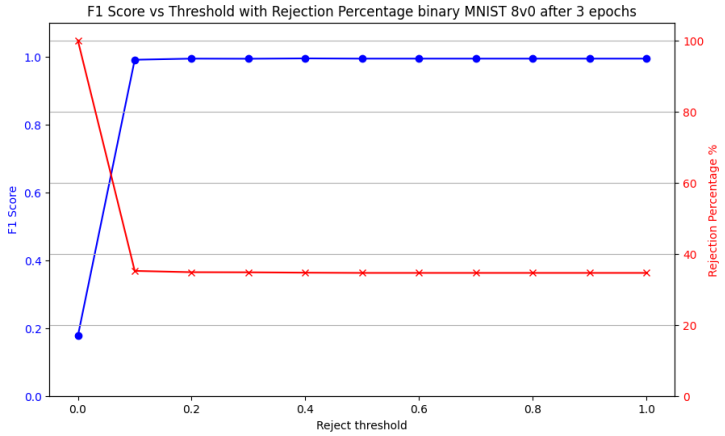

Figure 6.2: Evaluation of reject threshold impact on the custom binary MNIST dataset 8 versus 0: F1 score and Rejection Percentage after three training epochs.

these thresholds are more effective at balancing the trade-off between precision and recall. At these levels, the model rejects enough uncertain instances to avoid many incorrect predictions while still making enough correct predictions to maintain a high F1 score. In contrast, higher thresholds reduce the rejection rate too much, leading to more incorrect predictions and a lower F1 score. This pattern also disappears after three epochs of training time, suggesting that the model is more confident in its decision and that the precision and recall are always balanced.

These results show that training another neural network with a custom dataset containing low-confidence instances is possible. With enough training time, a convolutional neural network can find the same results on both the MNIST and fashion-MNIST datasets. Both the MNIST and fashion-MNIST datasets are datasets where almost every convolutional neural network can find high F1 scores, so obtaining higher performance, in this case, is impossible. Investigating whether a convolutional neural network can achieve comparable or superior results with more complex datasets would be interesting. The appendix shows all the other results for the custom

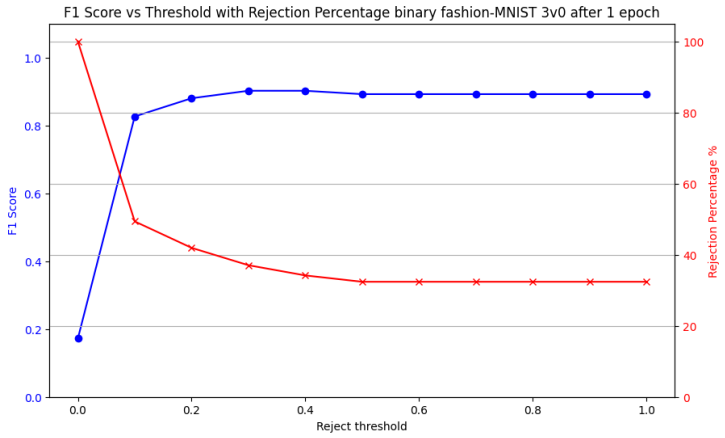

Figure 6.3: Evaluation of reject threshold impact on the custom binary fashion-MNIST dataset dress versus T-shirt: F1 score and Rejection Percentage after one training epoch

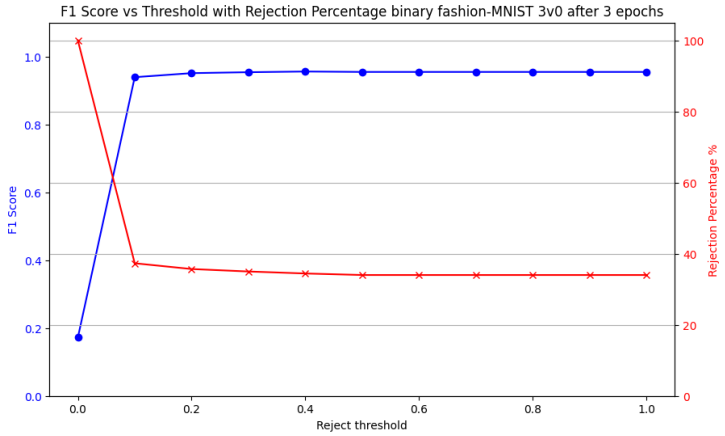

Figure 6.4: Evaluation of reject threshold impact on the custom binary fashion-MNIST dataset dress versus T-shirt: F1 score and Rejection Percentage after three training epochs

dataset MNIST with low-confidence instances: 7 versus 1, 5 versus 3, and 9 versus 4. Every other custom dataset with low-confidence instances shows similar results with the integrated rejector architecture.

6.3 Limitations

Subjectivity: A limitation of this work is requiring humans to decide what low-confidence is and is not. The difference between an eight and a zero is subjective. Some will view an image as clearly an eight or zero, and others can disagree. The same holds for every dataset, becoming even more pronounced as it grows in complexity. For example, in the fashion-MNIST dataset, what one person identifies as a T-shirt, another might see as a dress. This variability highlights the challenge of creating universally applicable low-confidence thresholds.

Chapter 7 Conclusion

This paper explores ways to improve GASTeN's performance by generating realistic images on the decision boundary of two classes and integrating a reject option in machine learning models to improve decision-making accuracy and reliability. By enabling models to abstain from deciding on low-confidence instances, we can enhance their trustworthiness and not worsen their performance. Our primary contribution is the development of AutoGASTeN. This automated pipeline uses GASTeN to generate realistic images on the decision boundary of two classes.

The objective of the first research question was to see if hyperparameter optimization techniques can improve GASTeN's performance across small gray-scale datasets. Our findings indicate that even basic hyperparameter optimization techniques, such as random grid search and random search, yield better results compared to the baseline in step 1 of the training process, see Figure 5.3. Bayesian optimization improves GASTeN performance the most with a Fréchet inception distance value of 8.69. We observe similar improvements in step two of the training process. Bayesian optimization achieved a Fréchet inception distance value of 10.12 and an average confusion distance value of 0.34 on the binary MNIST dataset 8 versus 0, where the original work scored 12.71 and 0.42, respectively. Although the improvement of the Fréchet inception distance is modest, the average confusion distance shows more improvement. Figures 5.5 and 5.6 illustrate that Bayesian optimization can achieve lower Fréchet inception distance values but also prioritizes the other objective, leading to an improvement of the average confusion distance. As a result of this improvement, AutoGASTeN introduces noise to the instances to better satisfy the average confusion distance objective. So, we need to be cautious when selecting a hyperparameter configuration. With visual inspection, we can select the hyperparameter configuration for AutoGASTeN, which can generate more instances of low confidence.

The objective of the second research question was to see the impact of the integrated rejection architecture on the performance of a convolutional neural network when handling binary subsamples of small gray-scale datasets with low-confidence instances. First, we created custom datasets with low-confidence instances, which are essential for training models to handle ambiguous scenarios effectively. Subsequently, we use these custom datasets to train a convolutional neural network with an integrated rejector architecture. The results demonstrate that the convolutional neural network trained with these datasets achieves similar precision, recall, and F1 scores, effectively rejecting low-confidence instances without compromising overall performance. We can see that by looking at Table 6.4. After three epochs, a convolutional neural network reaches the same precision, recall, and F1 score as the baseline results, indicating that the network can correctly reject low-confidence instances. The same results hold for the fashion-MNIST dataset. Also, looking at the rejection threshold in Figure 6.1 and 6.2 reveals that after one epoch of training, an F1 score is lower with a reject threshold of 0.1 compared to a higher threshold. This suggests that the convolutional neural network lacks confidence, assigning a probability of 0.1 or higher to the reject class. The rejection percentage is also higher at a threshold of 0.1 because more instances are likely to be rejected, but this pattern

disappears after three epochs, where the F1 score at a 0.1 threshold matches that of a 1.0 threshold, indicating increased confidence in the network's decisions.

Future work

Scale benchmark: Future research should focus on extending the application of AutoGASTeN to more complex and diverse datasets beyond MNIST and fashion-MNIST. Datasets such as CIFAR-10, ImageNet, and domain-specific datasets in fields such as medical imaging can provide insights into the scalability and robustness of the proposed methods in more challenging and varied environments.
Metric development: Additionally, future research should aim to develop a metric for measuring low-confidence. This work uses the Fréchet inception distance and average confusion distance as low-confidence indicators. We then supplement these metrics with visual inspection to determine whether an instance is truly low-confidence. As previously mentioned, this approach keeps humans in the loop. Ideally, we aim to remove human involvement and allow AutoGASTeN to autonomously make the best low-confidence decisions. Introducing an additional metric could enhance AutoGASTeN and the quality of the instances it generates.
Network architecture: Further research should also investigate other network architectures to enhance AutoGASTeN's performance. This involves experimenting with various neural network models to identify the most effective structure for different data types and applications. Exploring alternative architectures could broaden the range of potential applications and enhance the robustness of AutoGASTeN in various contexts.

Bibliography

M. M. Ahsan and Z. Siddique. Machine learning based disease diagnosis: A comprehensive review. *CoRR*, abs/2112.15538, 2021.

A. B. Arrieta, N. D. Rodríguez, J. D. Ser, A. Bennetot, S. Tabik, A. Barbado, S. García, S. Gil-Lopez, D. Molina, R. Benjamins, R. Chatila, and F. Herrera. Explainable artificial intelligence (XAI): concepts, taxonomies, opportunities and challenges toward responsible AI. *Inf. Fusion*, 58:82–115, 2020.

M. Baratchi, C. Wang, S. Limmer, J. N. van Rijn, H. H. Hoos, T. Bäck, and M. Olhofer. Automated machine learning: past, present and future. *Artif. Intell. Rev.*, 57(5):122, 2024.

J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305, 2012.

C. M. Bishop. *Pattern recognition and machine learning, 5th Edition*. Information science and statistics. Springer, 2007. ISBN 9780387310732.

L. Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, 2001.

L. Cunha, C. Soares, A. Restivo, and L. F. Teixeira. Gasten: Generative adversarial stress test networks. In *Advances in Intelligent Data Analysis XXI - 21st International Symposiumon Intelligent Data Analysis*, volume 13876, pages 91–102. Springer, 2023.

T. de Menezes e Silva Filho, H. Song, M. Perelló-Nieto, R. Santos-Rodríguez, M. Kull, and P. A. Flach. Classifier calibration: a survey on how to assess and improve predicted class probabilities. *Mach. Learn.*, 112(9):3211–3260, 2023.

K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.*, 6(2):182–197, 2002.

S. Falkner, A. Klein, and F. Hutter. BOHB: robust and efficient hyperparameter optimization at scale. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1436–1445. PMLR, 2018.

Y. Geifman and R. El-Yaniv. Selectivenet: A deep neural network with an integrated reject option. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2151–2159. PMLR, 2019.

I. Gomes, L. F. Teixeira, J. N. van Rijn, C. Soares, A. Restivo, L. Cunha, and M. Santos. Finding patterns in ambiguity: Interpretable stress testing in the decision boundary. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2024.

I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing*

Systems 27: Annual Conference on Neural Information Processing Systems 2014, pages 2672–2680, 2014.

K. Hendrickx, L. Perini, D. V. der Plas, W. Meert, and J. Davis. Machine learning with a reject option: A survey. *CoRR*, abs/2107.11277, 2021.

M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*, pages 6626–6637, 2017.

W. Homenda, M. Luckner, and W. Pedrycz. Classification with rejection based on various SVM techniques. In *2014 International Joint Conference on Neural Networks*, pages 3480–3487. IEEE, 2014.

E. Hüllermeier and W. Waegeman. Aleatoric and epistemic uncertainty in machine learning: an introduction to concepts and methods. *Mach. Learn.*, 110(3):457–506, 2021.

F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization - 5th International Conference*, volume 6683 of *Lecture Notes in Computer Science*, pages 507–523. Springer, 2011.

F. Hutter, L. Kotthoff, and J. Vanschoren, editors. *Automated Machine Learning - Methods, Systems, Challenges*. The Springer Series on Challenges in Machine Learning. Springer, 2019. ISBN 978-3-030-05317-8.

S. Karimi-Bidhendi, A. Arafati, A. L. Cheng, Y. Wu, A. Kheradvar, and H. Jafarkhani. Fully-automated deep-learning segmentation of pediatric cardiovascular magnetic resonance of patients with complex congenital heart diseases. *Journal of cardiovascular magnetic resonance*, 22(1):80, 2020.

D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *2nd International Conference on Learning Representations*, 2014.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, 1998.

L. Li, K. G. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *J. Mach. Learn. Res.*, 18:185:1–185:52, 2017.

M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, D. Deng, C. Benjamins, T. Ruhkopf, R. Sass, and F. Hutter. SMAC3: A versatile bayesian optimization package for hyperparameter optimization. *J. Mach. Learn. Res.*, 23:54:1–54:9, 2022.

M. Lucic, K. Kurach, M. Michalski, S. Gelly, and O. Bousquet. Are gans created equal? A large-scale study. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018*, pages 698–707, 2018.

J. Mukhoti, A. Kirsch, J. van Amersfoort, P. H. S. Torr, and Y. Gal. Deep deterministic uncertainty: A new simple baseline. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 24384–24394. IEEE, 2023.

K. P. Murphy. *Machine learning - a probabilistic perspective*. Adaptive computation and machine learning series. MIT Press, 2012. ISBN 0262018020.

A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *4th International Conference on Learning Representations, Conference Track Proceedings*, 2016.

M. T. Ribeiro, S. Singh, and C. Guestrin. "why should I trust you?": Explaining the predictions of any classifier. *CoRR*, abs/1602.04938, 2016.

V. Satopaa, J. R. Albrecht, D. E. Irwin, and B. Raghavan. Finding a "kneedle" in a haystack: Detecting knee points in system behavior. In *31st IEEE International Conference on Distributed Computing Systems Workshops (ICDCS 2011 Workshops)*, pages 166–171. IEEE Computer Society, 2011.

A. Sharma, J. N. van Rijn, F. Hutter, and A. Müller. Hyperparameter importance for image classification by residual neural networks. In *International Conference on Discovery Science*, pages 112–126. Springer, 2019.

J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. *CoRR*, abs/1206.2944, 2012.

J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, E. H. Chi, T. Hashimoto, O. Vinyals, P. Liang, J. Dean, and W. Fedus. Emergent abilities of large language models. *Trans. Mach. Learn. Res.*, 2022, 2022.

M. Weiss, A. G. Gómez, and P. Tonella. Generating and detecting true ambiguity: a forgotten danger in DNN supervision testing. *Empir. Softw. Eng.*, 28(6):146, 2023.

H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017.

C. Zou, E.-h. Zheng, H.-w. Xu, and L. Chen. Cost-sensitive multi-class svm with reject option: A method for steam turbine generator fault diagnosis. *International Journal of Computer Theory and Engineering*, 3(1):77, 2011.

Chapter A Additional results

All additional results, including figures and tables, will be presented in this section.

A.1 Binary MNIST data 8 versus 0

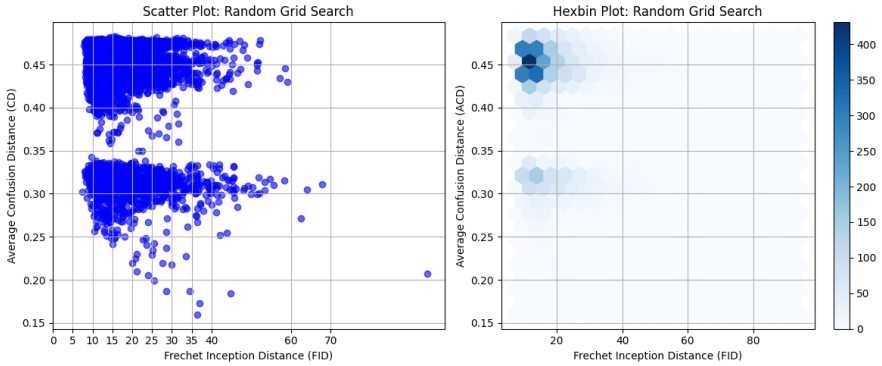

Figure A.1: Evaluation of random grid search on the binary MNIST dataset 8 versus 0: scatter and hexbin plots of the Fréchet inception distance versus average confusion distance.

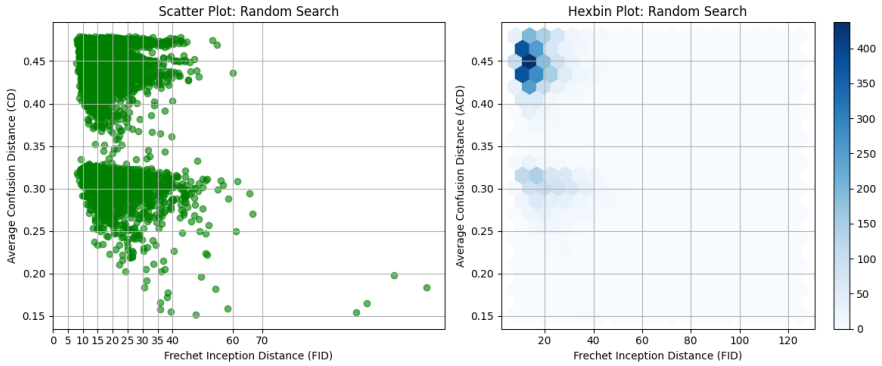

Figure A.2: Evaluation of random search on the binary MNIST dataset 8 versus 0: scatter and hexbin plots of the Fréchet inception distance versus average confusion distance

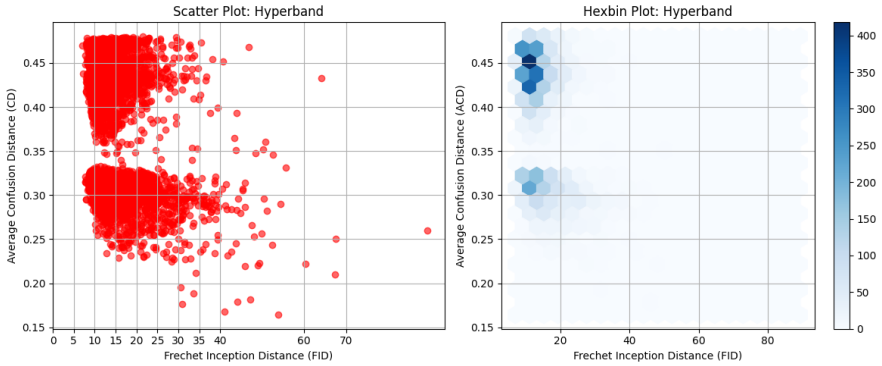

Figure A.3: Evaluation of hyperband on the binary MNIST dataset 8 versus 0: scatter and hexbin plots of the Fréchet inception distance versus average confusion distance

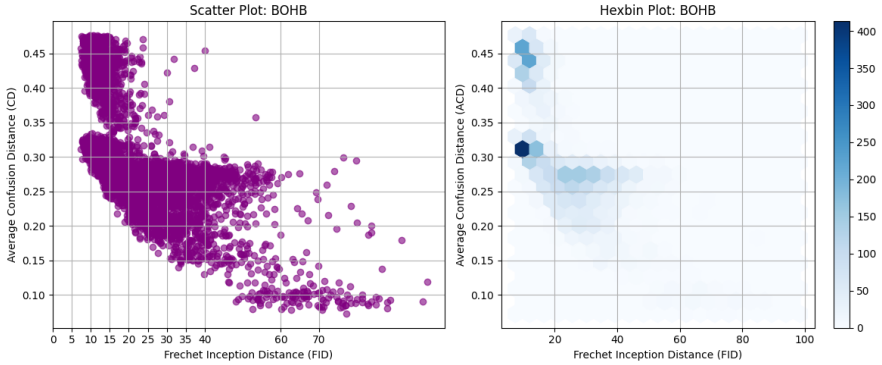

Figure A.4: Evaluation of BOHB on the binary MNIST dataset 8 versus 0: scatter and hexbin plots of the Fréchet inception distance versus average confusion distance

A.2 Binary MNIST data 7 versus 1

Table A.1: Distribution of training and testing instances for the custom MNIST binary dataset 7 versus 1

Class	**Train**	**Test**
Class 1	6742	1135
Class 7	6265	1028
Class ®	5822	1028

Class	**Precision**	**Recall**	**F1 score**
1	0.99	0.99	0.99
7	1.00	1.00	0.99

Table A.2: Evaluation of a convolutional neural network on the binary MNIST dataset 7 versus 1: precision, recall, and F1 score after one epoch averaged over three runs.

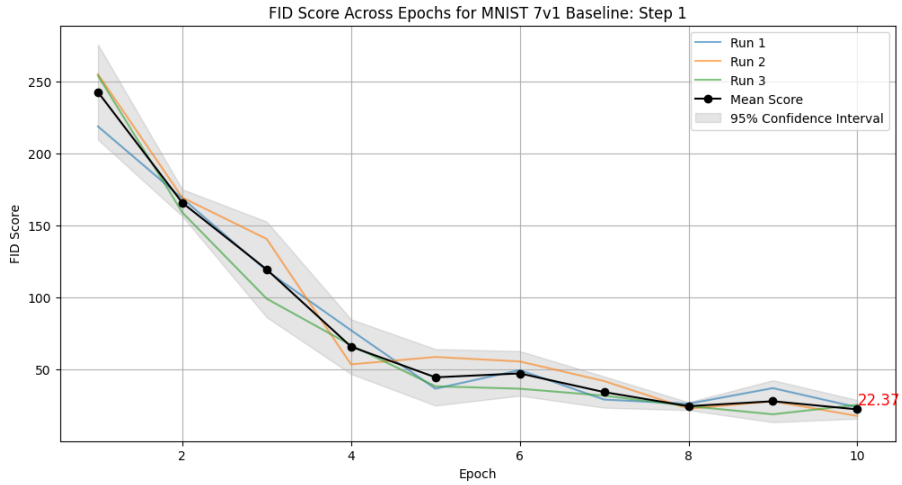

Figure A.5: Baseline results for the MNIST 7 versus 1 dataset, step 1: Mean and standard deviation across three runs.

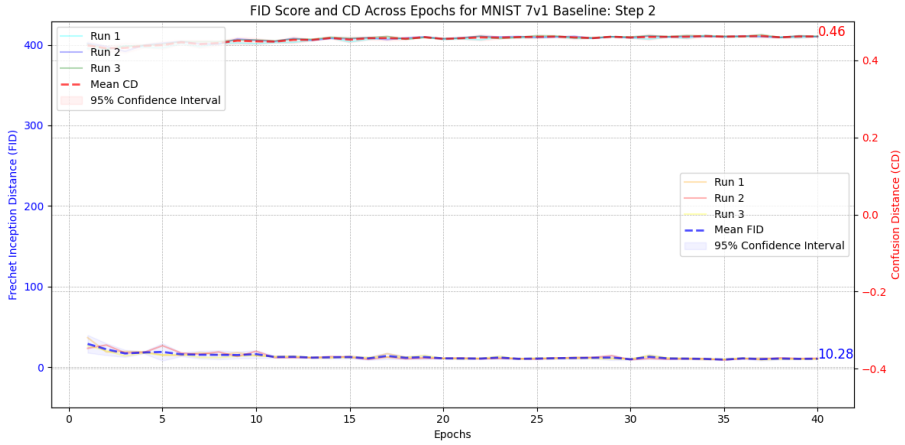

Figure A.6: Baseline results for the MNIST 7 versus 1 dataset, step 2: Mean and standard deviation across three runs.

Class	Precision	Recall	F1 score
1	1.00	1.00	1.00
7	1.00	1.00	1.00

Table A.3: Evaluation of a convolutional neural network on the binary MNIST dataset 5 versus 3: precision, recall, and F1 score after three epochs averaged over three runs.

Class	Precision	Recall	F1 score
1	0.98	0.99	0.99
7	1.00	0.94	0.97
Ⓡ	0.94	0.99	0.96

Table A.4: Evaluation of integrated rejector architecture with a convolutional neural network on the custom binary MNIST dataset 7 versus 1: precision, recall, and F1 score after one epoch.

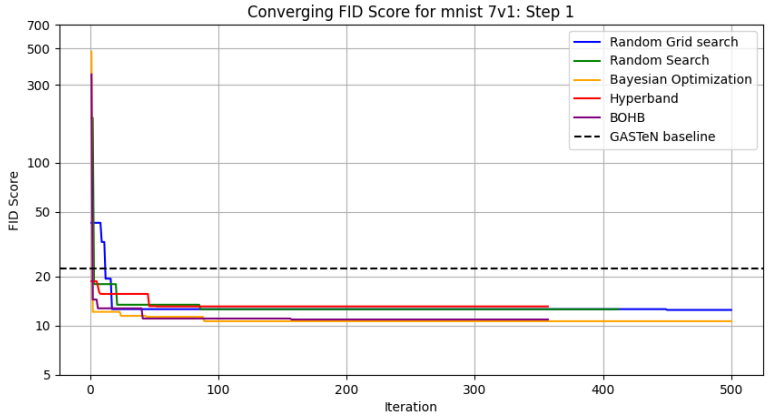

Figure A.7: Comparison of hyperparameter optimization techniques on the binary MNIST dataset 7 versus 1: Convergence of the Fréchet inception distance score for the first training step.

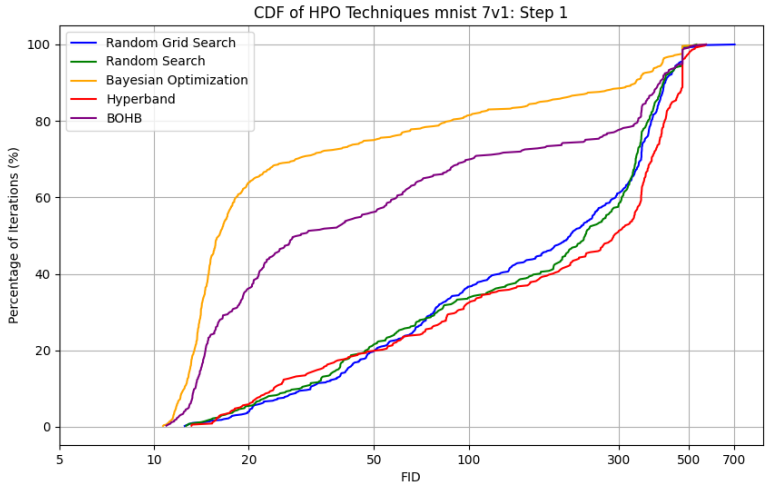

Figure A.8: Cumulative distribution function of different hyperparameter optimization techniques for the binary MNIST dataset 7 versus 1 for the first training step.

Class	Precision	Recall	F1 score
1	0.98	1.00	0.99
7	0.98	0.98	0.98
Ⓡ	0.99	0.97	0.98

Table A.5: Evaluation of integrated rejector architecture with a convolutional neural network on the custom binary MNIST dataset 7 versus 1: precision, recall, and F1 score after three epochs.

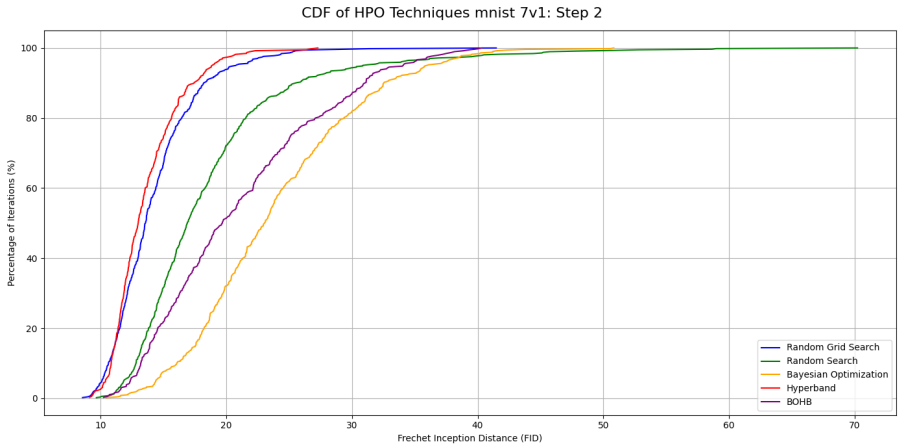

Figure A.9: Cumulative distribution function of different hyperparameter optimization techniques for the binary MNIST dataset 7 versus 1 for the second training step.

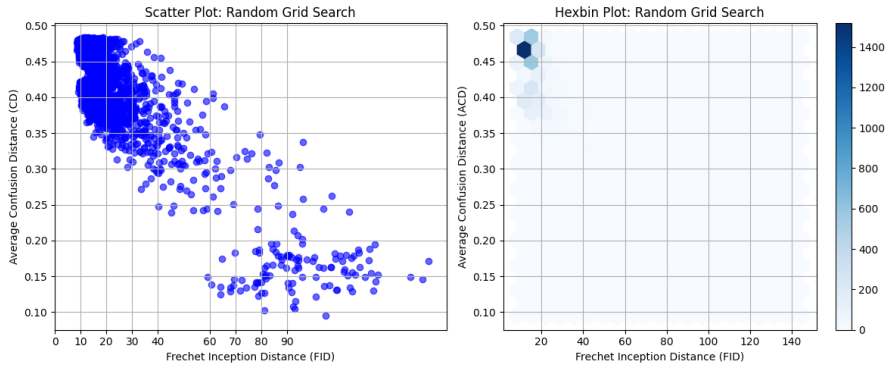

Figure A.10: Evaluation of random grid search on the binary MNIST dataset 7 versus 1: scatter and hexbin plots of the Fréchet inception distance versus average confusion distance.

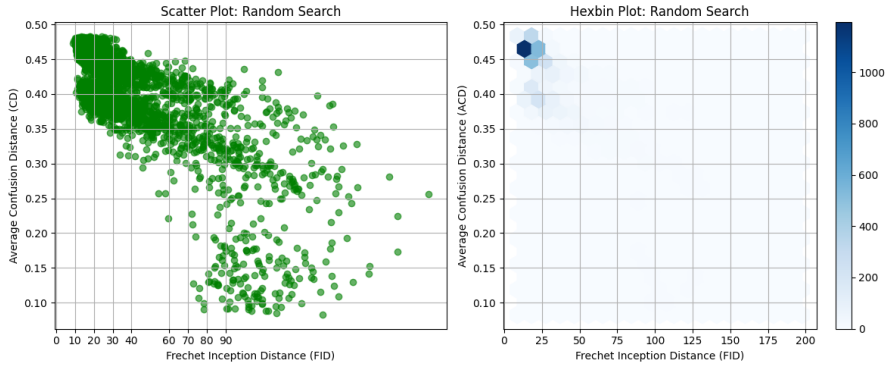

Figure A.11: Evaluation of random search on the binary MNIST dataset 7 versus 1: scatter and hexbin plots of the Fréchet inception distance versus average confusion distance

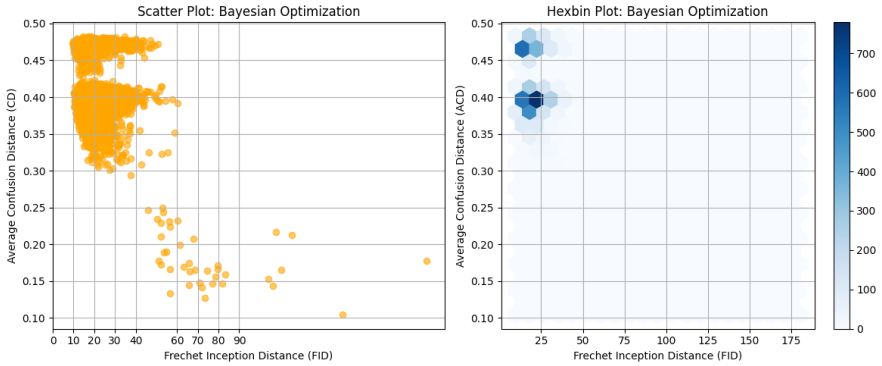

Figure A.12: Evaluation of Bayesian optimization on the binary MNIST dataset 7 versus 1: scatter and hexbin plots of the Fréchet inception distance versus average confusion distance

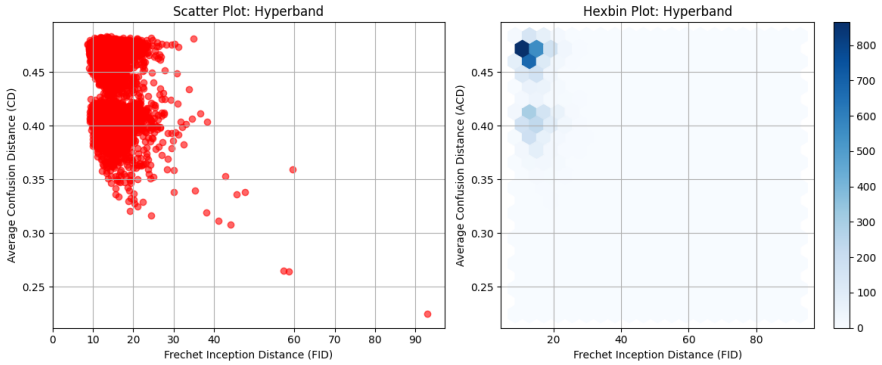

Figure A.13: Evaluation of hyperband on the binary MNIST dataset 7 versus 1: scatter and hexbin plots of the Fréchet inception distance versus average confusion distance

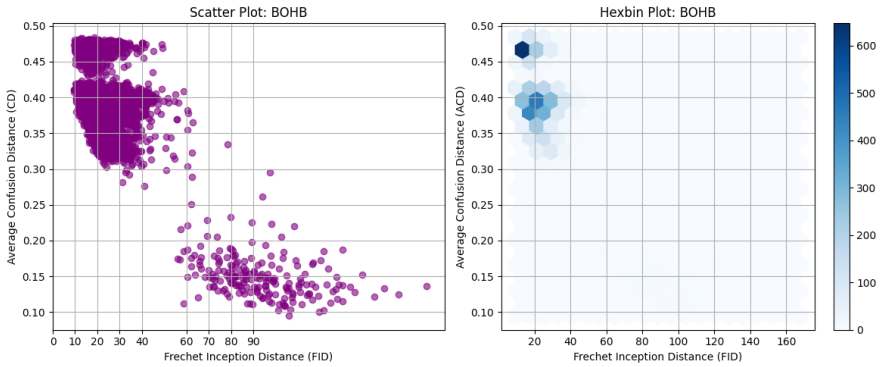

Figure A.14: Evaluation of BOHB on the binary MNIST dataset 7 versus 1: scatter and hexbin plots of the Fréchet inception distance versus average confusion distance

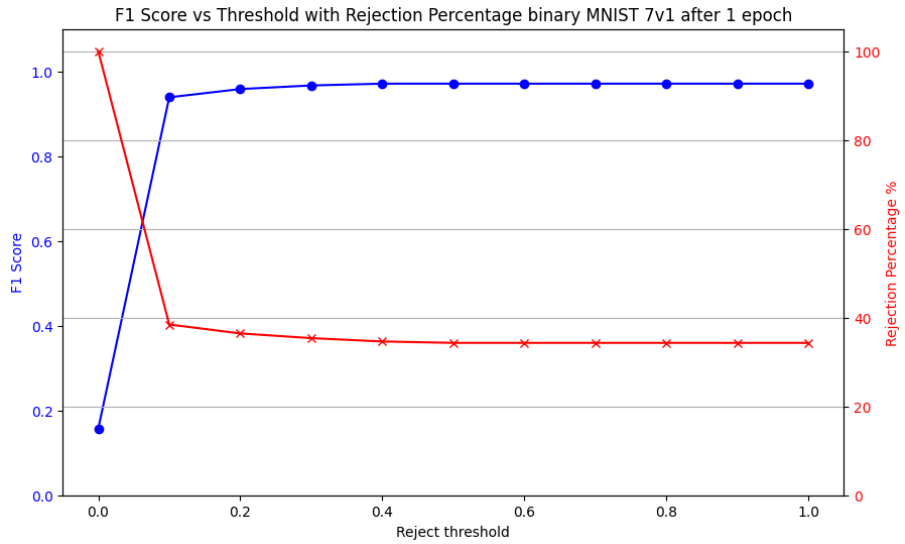

Figure A.15: Evaluation of reject threshold impact on the custom binary MNIST dataset 7 versus 1: F1 score and Rejection Percentage after one training epoch

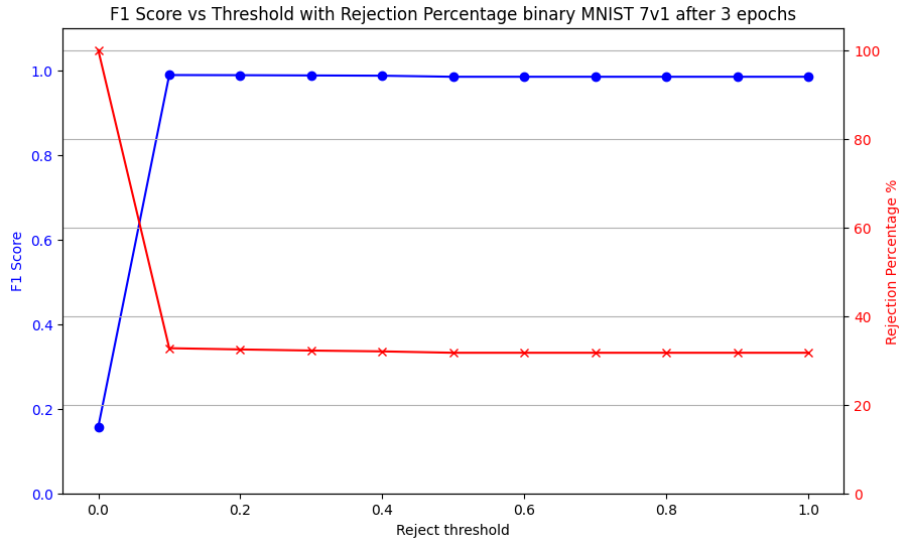

Figure A.16: Evaluation of reject threshold impact on the custom binary MNIST dataset 7 versus 1: F1 score and Rejection Percentage after three training epochs

A.3 Binary MNIST data 5 versus 3

Table A.6: Distribution of training and testing instances for the custom MNIST binary dataset 5 versus 3

Class	Train	Test
Class 3	6131	1010
Class 5	5421	892
Class ®	5822	1028

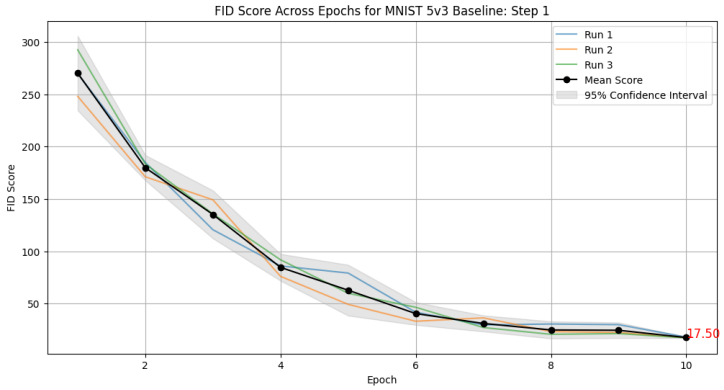

Figure A.17: Baseline results for the MNIST 5 versus 3 dataset, step 1: Mean and standard deviation across three runs.

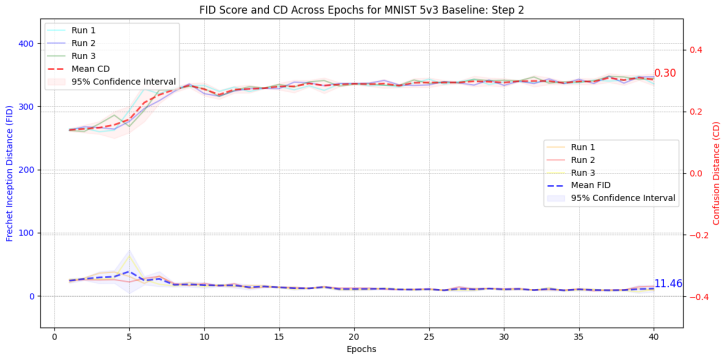

Figure A.18: Baseline results for the MNIST 5 versus 3 dataset, Step 2: Mean and standard deviation across three runs.

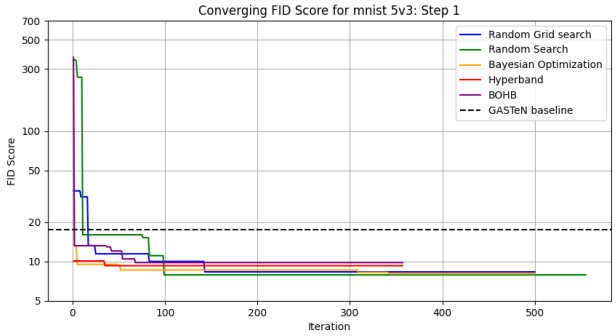

Figure A.19: Comparison of hyperparameter optimization techniques on the binary MNIST dataset 5 versus 3: Convergence of the Fréchet inception distance score for the first training step.

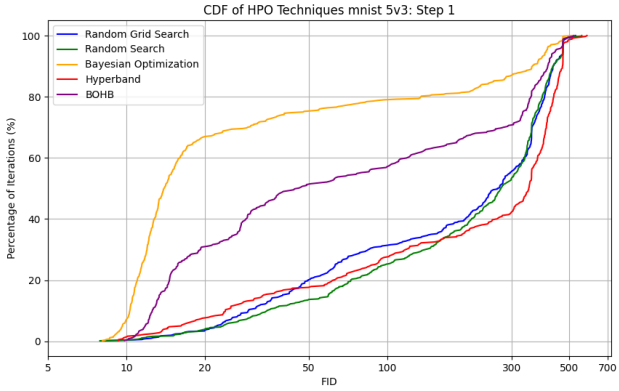

Figure A.20: Cumulative distribution function of different hyperparameter optimization techniques for the binary MNIST dataset 5 versus 3 for the first training step.

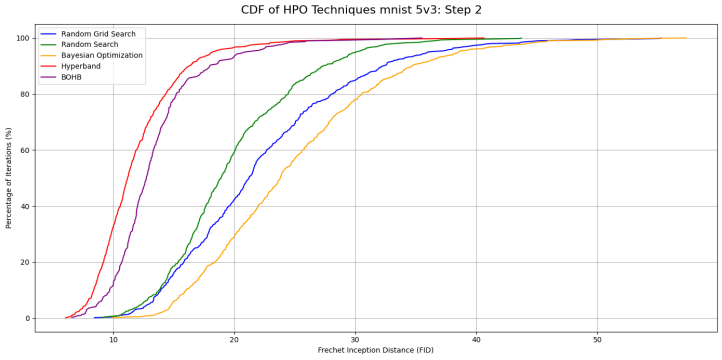

Figure A.21: Cumulative distribution function of different hyperparameter optimization techniques for the binary MNIST dataset 5 versus 3 for the second training step.

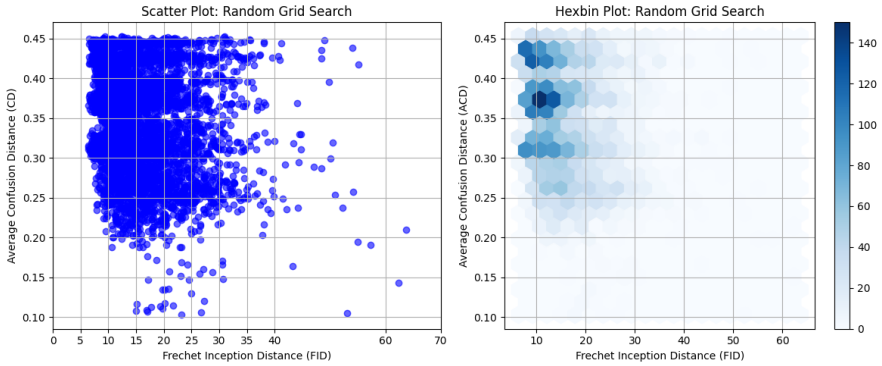

Figure A.22: Evaluation of random grid search on the binary MNIST dataset 5 versus 3: scatter and hexbin plots of the Fréchet inception distance versus average confusion distance.

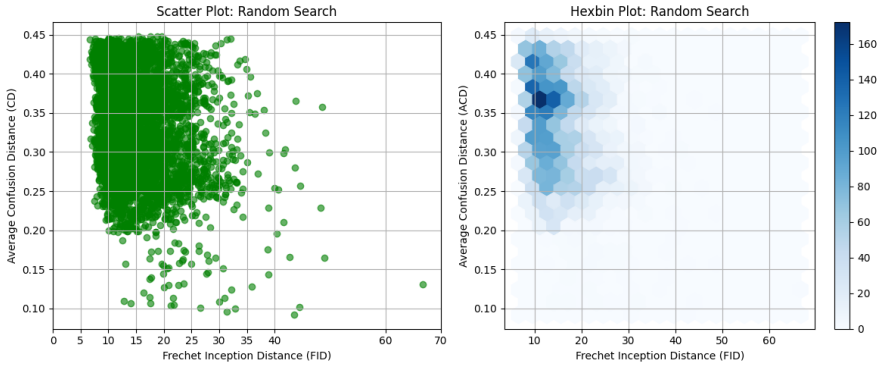

Figure A.23: Evaluation of random search on the binary MNIST dataset 5 versus 3: scatter and hexbin plots of the Fréchet inception distance versus average confusion distance

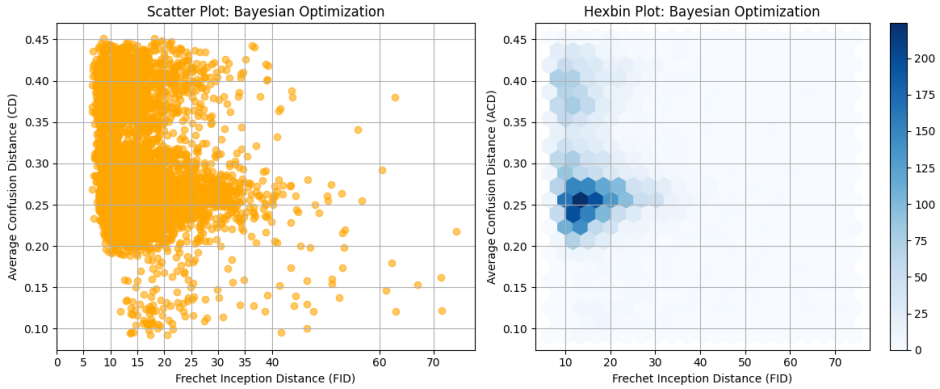

Figure A.24: Evaluation of Bayesian optimization on the binary MNIST dataset 5 versus 3: scatter and hexbin plots of the Fréchet inception distance versus average confusion distance

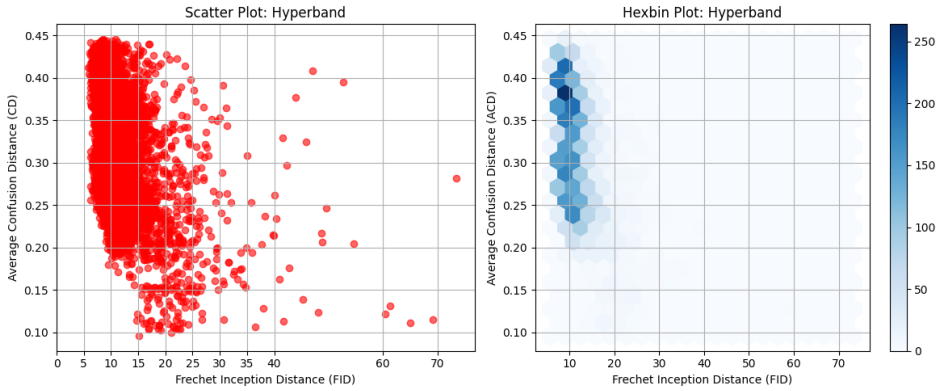

Figure A.25: Evaluation of hyperband on the binary MNIST dataset 5 versus 3: scatter and hexbin plots of the Fréchet inception distance versus average confusion distance

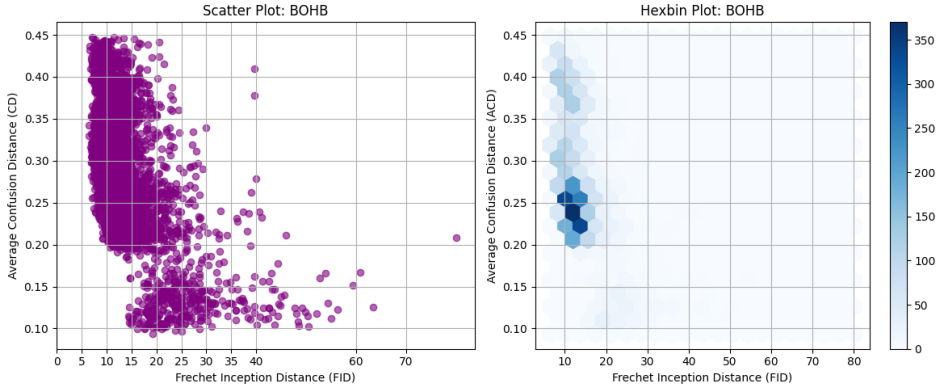

Figure A.26: Evaluation of BOHB on the binary MNIST dataset 5 versus 3: scatter and hexbin plots of the Fréchet inception distance versus average confusion distance

Class	Precision	Recall	F1 score
3	1.00	0.99	0.99
5	0.99	1.00	0.99

Table A.7: Evaluation of a convolutional neural network on the binary MNIST dataset 5 versus 3: precision, recall, and F1 score after one epoch averaged over three runs.

Class	Precision	Recall	F1 score
3	1.00	1.00	1.00
5	0.99	1.00	1.00

Table A.8: Evaluation of a convolutional neural network on the binary MNIST dataset 5 versus 3: precision, recall, and F1 score after three epochs averaged over three runs.

Class	Precision	Recall	F1 score
3	0.99	0.95	0.97
5	0.95	0.95	0.95
Ⓡ	0.91	0.96	0.93

Table A.9: Evaluation of integrated rejector architecture with a convolutional neural network on the custom binary MNIST dataset 5 versus 3: precision, recall, and F1 score after one epoch.

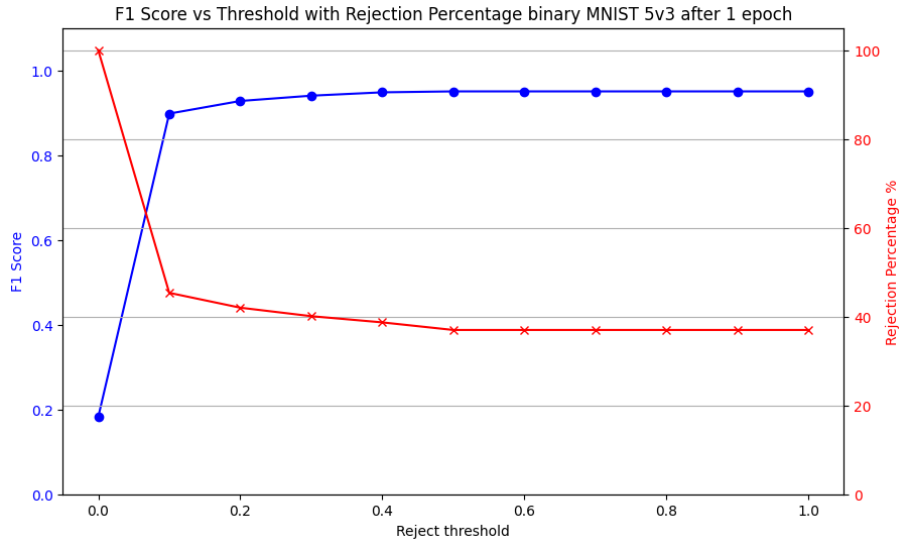

Figure A.27: Evaluation of reject threshold impact on the custom binary MNIST dataset 5 versus 3: F1 score and Rejection Percentage after one training epoch

Class	Precision	Recall	F1 score
3	0.99	0.99	0.99
5	0.98	0.99	0.99
Ⓡ	1.00	0.98	0.99

Table A.10: Evaluation of integrated rejector architecture with a convolutional neural network on the custom binary MNIST dataset 5 versus 3: precision, recall, and F1 score after three epochs.

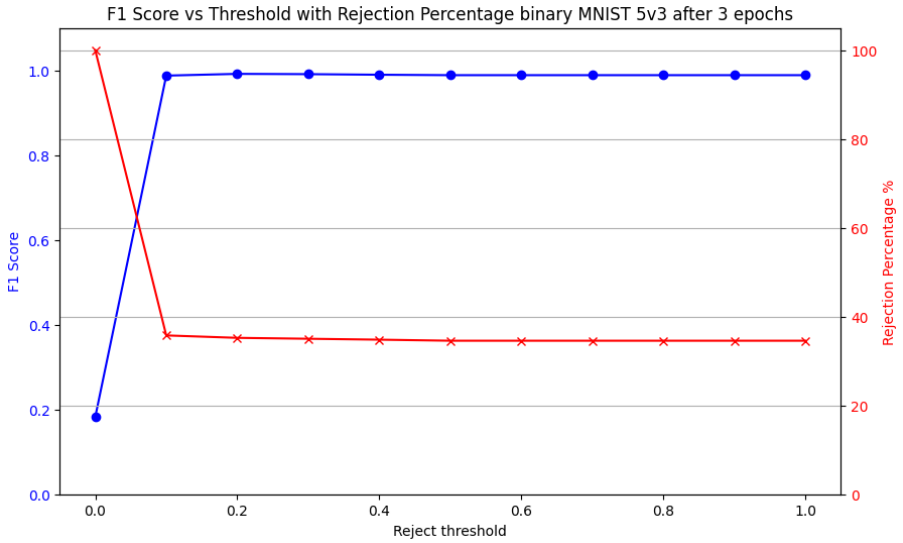

Figure A.28: Evaluation of reject threshold impact on the custom binary MNIST dataset 5 versus 3: F1 score and Rejection Percentage after three training epochs

A.4 Binary MNIST data 9 versus 4

Table A.11: Distribution of training and testing instances for the custom MNIST binary dataset 9 versus 4

Class	Train	Test
Class 4	5842	982
Class 9	5949	1009
Class ®	5822	1028

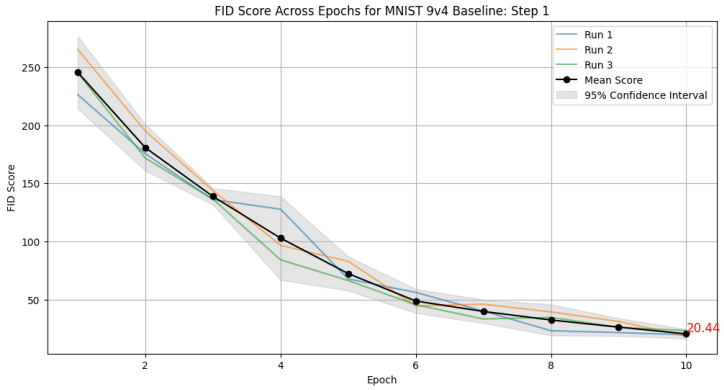

Figure A.29: Baseline results for the binary MNIST 9 versus 4 dataset, step 1: Mean and standard deviation across three runs.

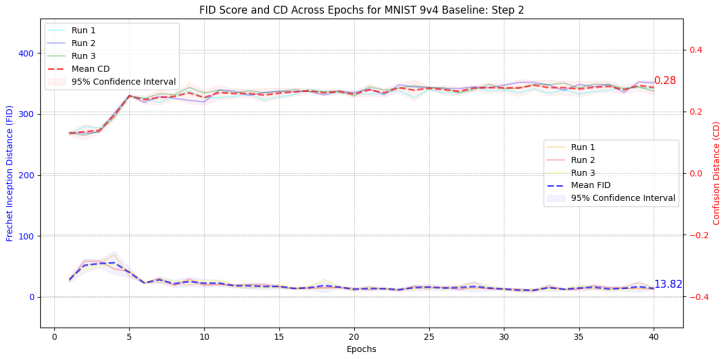

Figure A.30: Baseline results for the binary MNIST 9 versus 4 dataset, step 2: Mean and standard deviation across three runs.

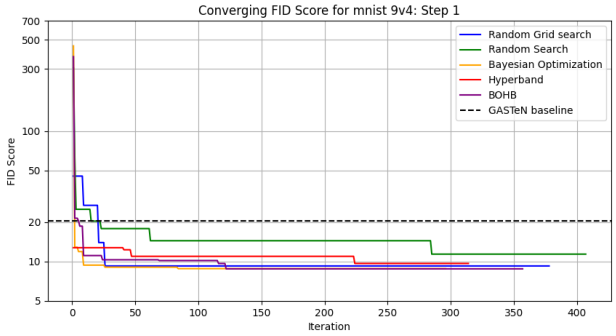

Figure A.31: Comparison of hyperparameter optimization techniques on the binary MNIST dataset 9 versus 4: Convergence of the Fréchet inception distance score for the first training step.

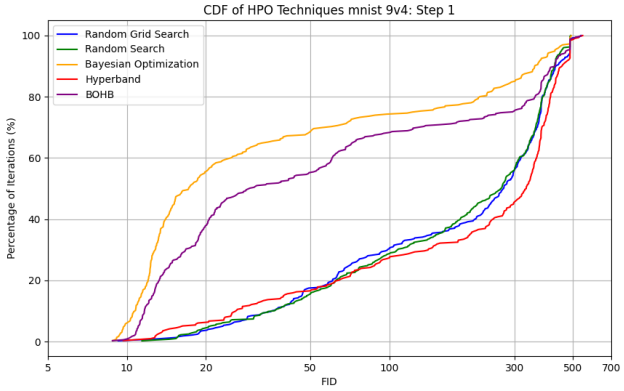

Figure A.32: Cumulative distribution function of different hyperparameter optimization techniques for the binary MNIST dataset 9 versus 4 for the first training step.

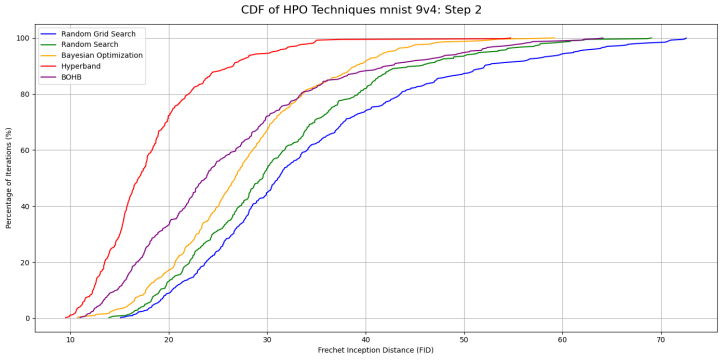

Figure A.33: Cumulative distribution function of different hyperparameter optimization techniques for the binary MNIST dataset 9 versus 4 for the second training step.

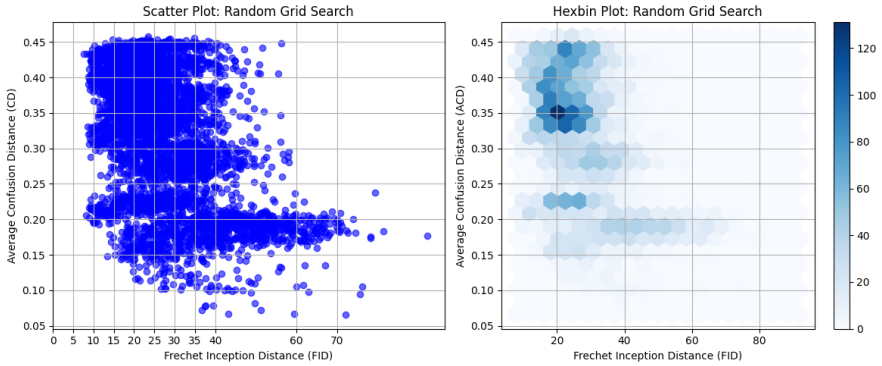

Figure A.34: Evaluation of random grid search on the binary MNIST dataset 9 versus 4: scatter and hexbin plots of the Fréchet inception distance versus average confusion distance.

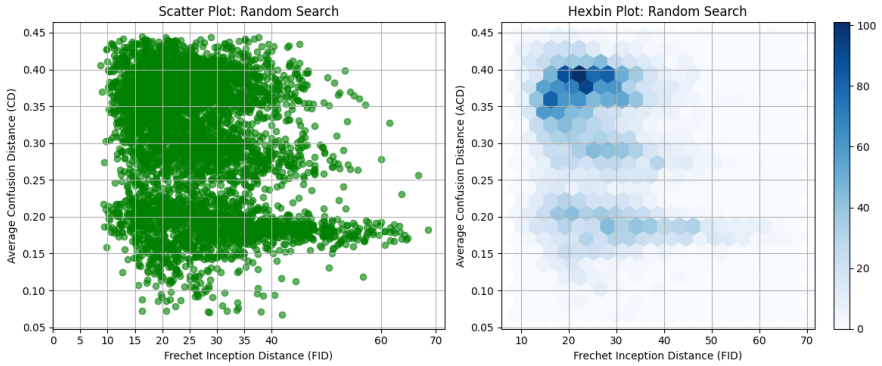

Figure A.35: Evaluation of random search on the binary MNIST dataset 9 versus 4: scatter and hexbin plots of the Fréchet inception distance versus average confusion distance

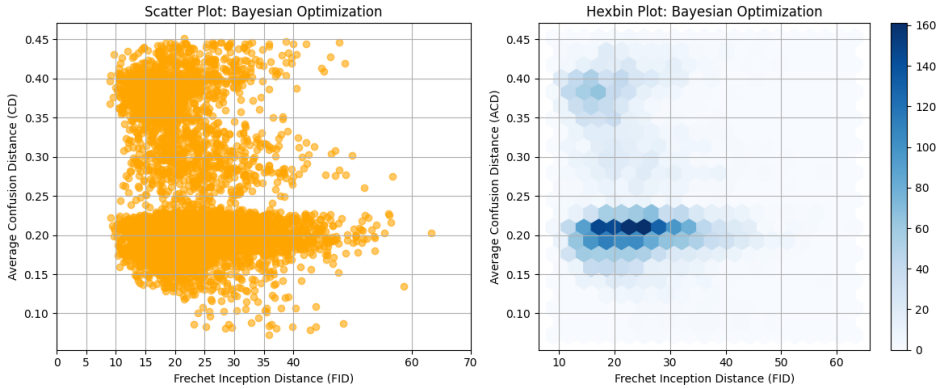

Figure A.36: Evaluation of Bayesian optimization on the binary MNIST dataset 9 versus 4: scatter and hexbin plots of the Fréchet inception distance versus average confusion distance

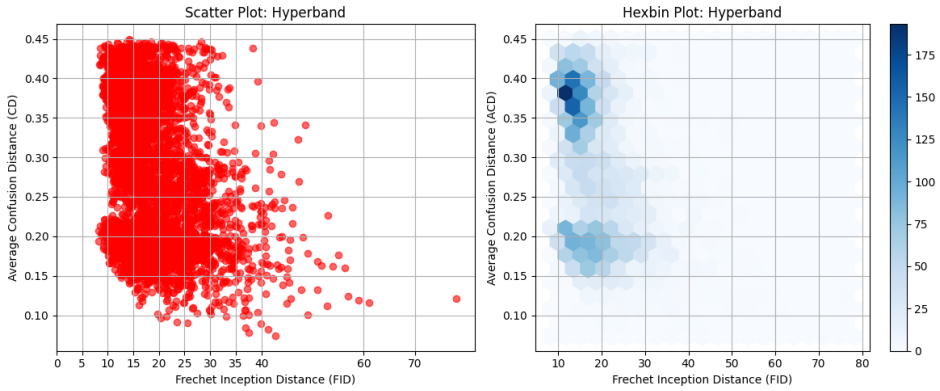

Figure A.37: Evaluation of hyperband on the binary MNIST dataset 9 versus 4: scatter and hexbin plots of the Fréchet inception distance versus average confusion distance

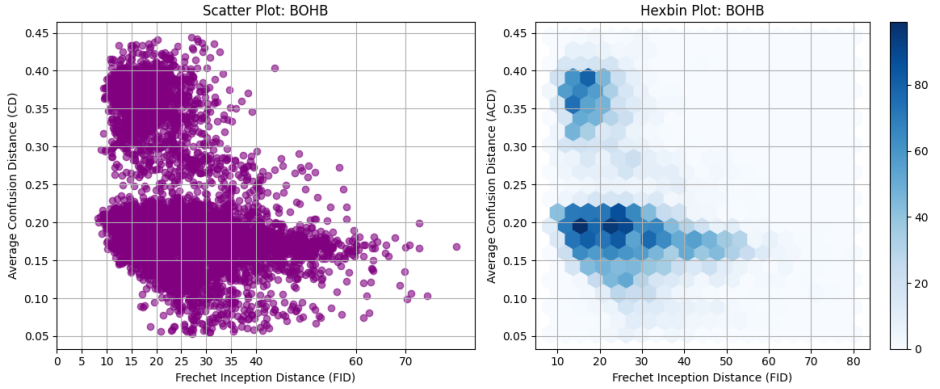

Figure A.38: Evaluation of BOHB on the binary MNIST dataset 9 versus 4: scatter and hexbin plots of the Fréchet inception distance versus average confusion distance

Class	Precision	Recall	F1 score
4	1.00	0.99	0.99
9	0.99	1.00	1.00

Table A.12: Evaluation of a convolutional neural network on the binary MNIST dataset 9 versus 4: precision, recall, and F1 score after one epoch averaged over three runs.

Class	Precision	Recall	F1 score
4	1.00	1.00	1.00
9	1.00	1.00	1.00

Table A.13: Evaluation of a convolutional neural network on the binary MNIST dataset 9 versus 4: precision, recall, and F1 score after three epochs averaged over three runs.

Class	Precision	Recall	F1 score
4	0.98	0.89	0.93
9	0.96	0.95	0.95
Ⓡ	0.87	0.95	0.91

Table A.14: Evaluation of integrated rejector architecture with a convolutional neural network on the custom binary MNIST dataset 9 versus 4: precision, recall, and F1 score after one epoch.

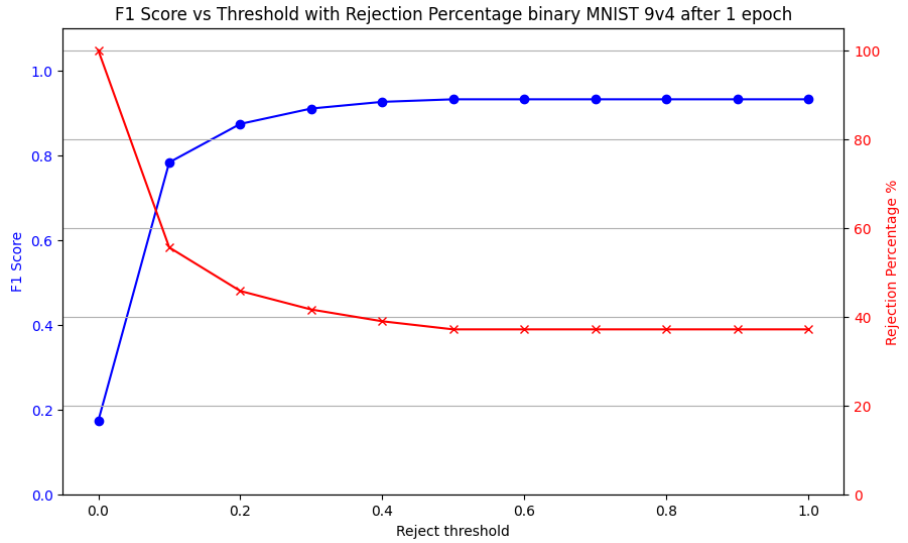

Figure A.39: Evaluation of reject threshold impact on the custom binary MNIST dataset 9 versus 4: F1 score and Rejection Percentage after one training epoch

Class	Precision	Recall	F1 score
4	0.99	0.99	0.99
9	0.95	1.00	0.97
Ⓡ	1.00	0.95	0.98

Table A.15: Evaluation of integrated rejector architecture with a convolutional neural network on the custom binary MNIST dataset 9 versus 4: precision, recall, and F1 score after three epochs.

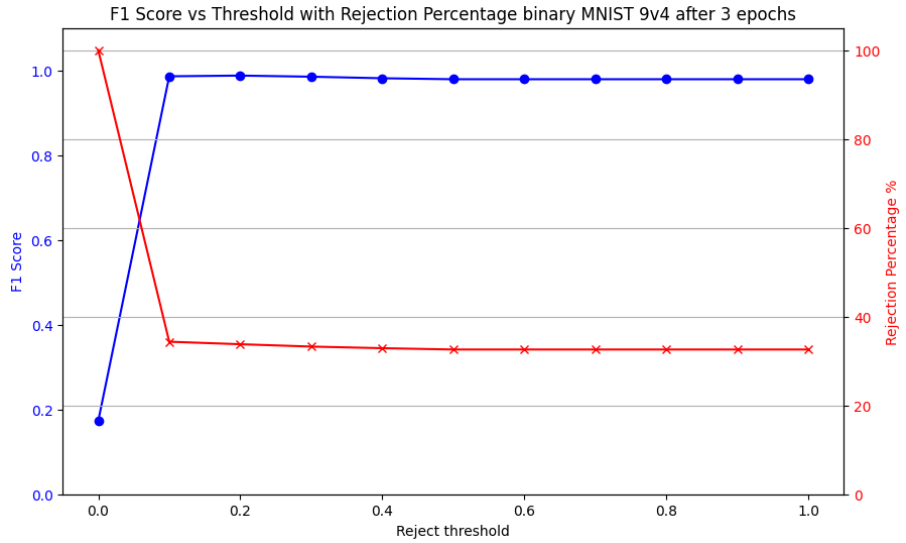

Figure A.40: Evaluation of reject threshold impact on the custom binary MNIST dataset 9 versus 4: F1 score and Rejection Percentage after three training epochs

A.5 Binary fashion-MNIST data dress versus T-shirt

Table A.16: Distribution of training and testing instances for the custom fashion-MNIST binary dataset dress versus T-shirt (dress versus top)

Class	Train	Test
Class 0	6000	1000
Class 3	6000	1000
Class ®	5822	1028

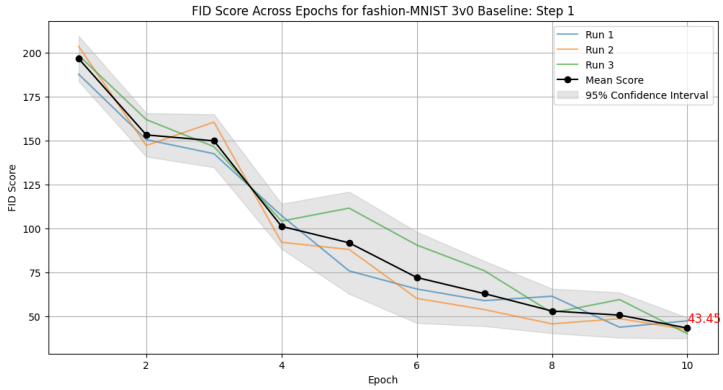

Figure A.41: Baseline results for the fashion-MNIST dress versus T-shirt dataset, step 1: Mean and standard deviation across three runs.

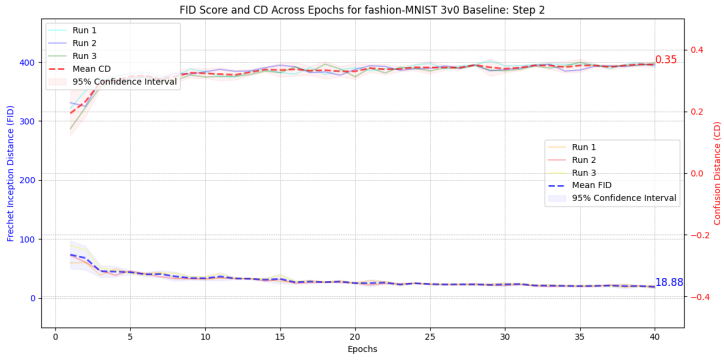

Figure A.42: Baseline results for the fashion-MNIST dress versus T-shirt dataset, step 2: Mean and standard deviation across three runs.

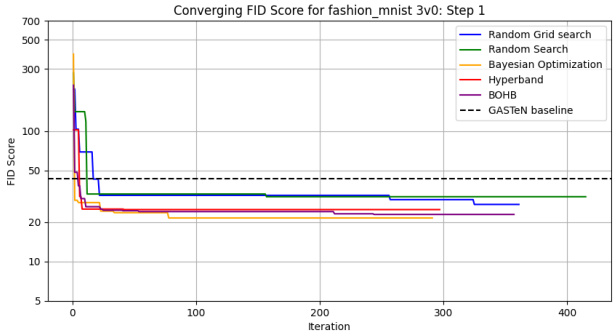

Figure A.43: Comparison of hyperparameter optimization techniques on the binary fashion-MNIST dataset dress versus T-shirt: Convergence of the Fréchet inception distance score for the first training step.

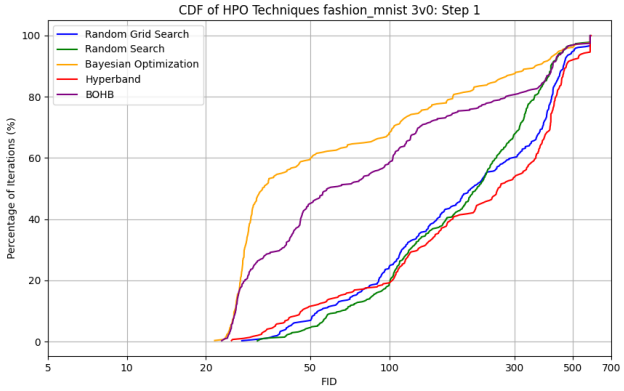

Figure A.44: Cumulative distribution function of different hyperparameter optimization techniques for the binary fashion-MNIST dataset dress versus T-shirt for the first training step.

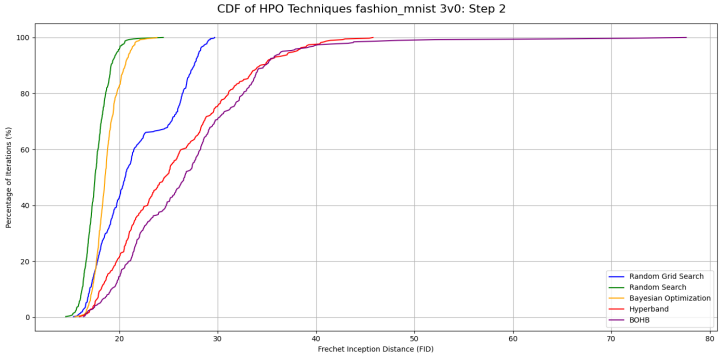

Figure A.45: Cumulative distribution function of different hyperparameter optimization techniques for the binary fashion-MNIST dataset dress versus T-shirt for the second training step.

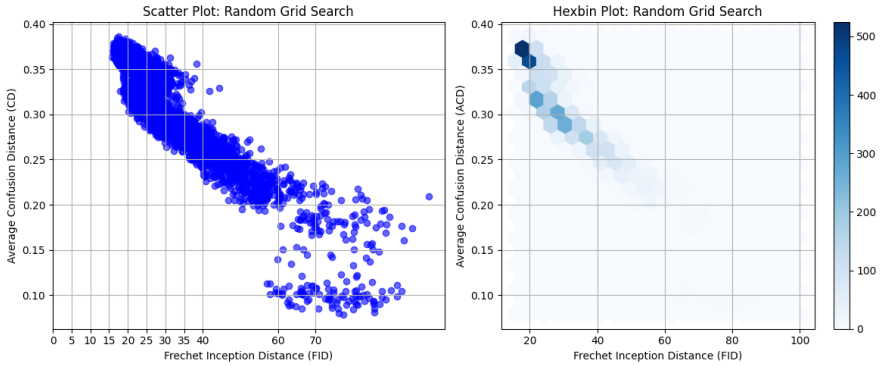

Figure A.46: Evaluation of random grid search on the binary fashion-MNIST dataset dress versus T-shirt: scatter and hexbin plots of the Fréchet inception distance versus average confusion distance.

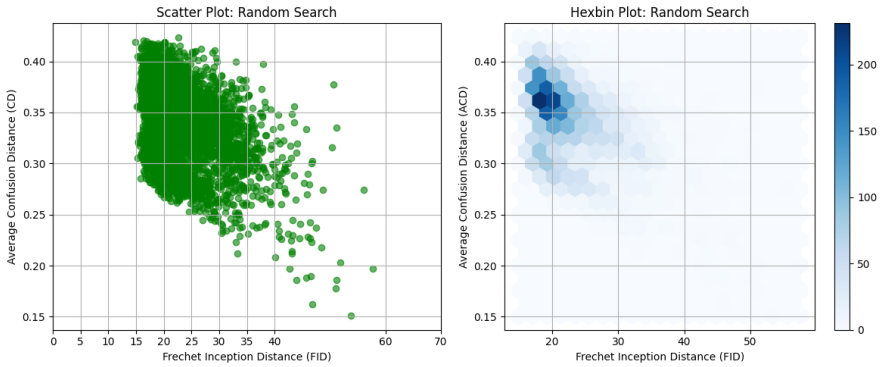

Figure A.47: Evaluation of random search on the binary fashion-MNIST dataset dress versus T-shirt: scatter and hexbin plots of the Fréchet inception distance versus average confusion distance

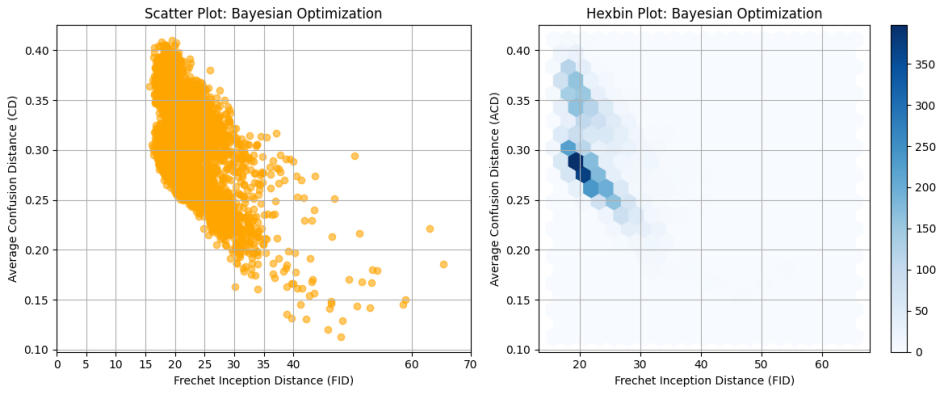

Figure A.48: Evaluation of Bayesian optimization on the binary fashion-MNIST dataset dress versus T-shirt: scatter and hexbin plots of the Fréchet inception distance versus average confusion distance

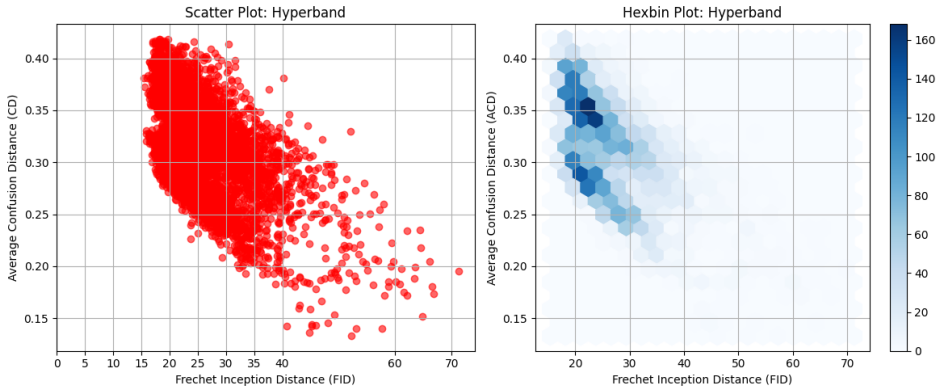

Figure A.49: Evaluation of hyperband on the binary fashion-MNIST dataset dress versus T-shirt: scatter and hexbin plots of the Fréchet inception distance versus average confusion distance

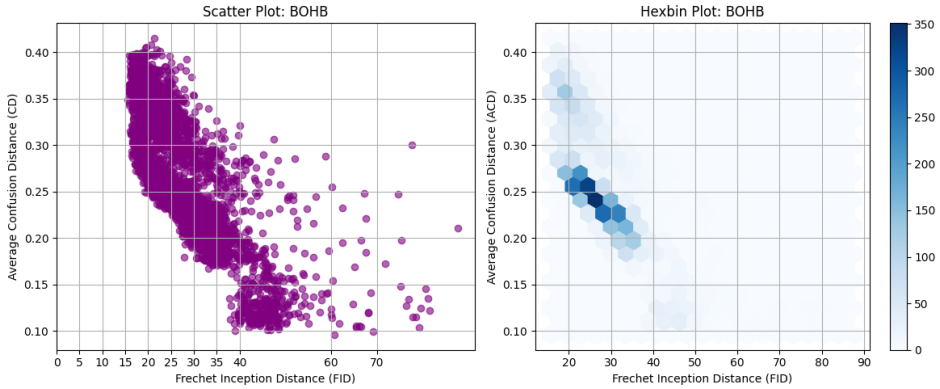

Figure A.50: Evaluation of BOHB on the binary fashion-MNIST dataset dress versus T-shirt: scatter and hexbin plots of the Fréchet inception distance versus average confusion distance