Deep Active Learning for Classification Tasks

Technische Universität München

Department of Mathematics

Master's Thesis

Moritz Spielvogel

Supervisor: Prof. Ph.D. Hans-Joachim Bungartz

Advisor: M.Sc. Mathias Sundholm, M.Sc. Ionut-Gabriel Farcas

Submission Date: 03/03/2020

I hereby declare that this thesis is my own work and that no other sources have been used except those clearly indicated and referenced.

München, 03/03/2020

Acknowledgements

First of all, I would like to thank Prof. Ph. D. Bungartz for supervising me on this thesis.

Then, I would especially like to thank my advisor Mathias Sundholm from PreciBake GmbH for all his support and the inspiring discussions I have had with him, for his time and ideas.

Also, I would like to thank my advisor Ionut-Gabriel Farcas from Prof. Bungartz' chair for all his help, his different views onto the topic on Machine Learning and his time to answer all my questions.

Lastly but not less important, I would like to thank my girlfriend Sarah Dörr and my family for their constant motivation and ongoing support.

I dedicate the master thesis to my grandfather Horst († 02.02.20) and my brother Toni (* 02.02.20).

Abstract

Deep neural networks need in general high amounts of data to gain high performance on a test set. Depending on the task, labeling such a big data set is very expensive whereas it is often easy to get huge amounts of unlabeled data. The purpose of this thesis is to find a much smaller subset on which the network performs similarly well. Thus, an active learning framework is built, which gradually samples from the given pool of unlabeled data. For the sampling process, different sampling methods such as uncertainty sampling, class balance sampling, and representation sampling are compared and combined in order to find the best method. By using Bayesian neural network, the uncertainty of the network's parameters is extracted. Since they are intractable, different approximation methods are tested. Neural networks tend to overfit on small data sets. By applying regularization methods throughout the training, this can be reduced. From all these sampling methods, the combination of uncertainty sampling and class prediction methods yielded the best results. With the previous sampling method, the framework yielded a similar performance on the data set provided by PreciBake GmbH, but only used a tenth of the data for training.

Zusammenfassung

Tiefe Neuronale Netze (Deep neural networks) brauchen im Allgemeinen große Mengen annotierte Daten, um hochperformante Ergebnisse erzielen zu können. Die Idee von Active Learning (Aktives Lernen) ist hingegen die Daten, welche annotiert werden sollen, auszuwählen, damit das Neuronale Netz auch mit diesem Datensatz eine vergleichbare Performanz erreicht. Hierfür wird ein Algorithmus entwickelt, welcher von diesem Datensatz schrittweise Teilmengen einer bestimmten Größe auswählt, diese annotieren lässt und das Neuronale Netz auf dem annotierten Teildatensatz trainiert. In dieser Masterarbeit werden verschiedene Methoden eingeführt und miteinander verglichen, welche in jeder Runde eine Teilmenge des Datensatzes auswählen. Eine der am vielversprechensten Methoden ist die Unsicherheit des Models über die Klassenzugehörtigkeit der nicht-annotierten Daten zu betrachten und Elemente auszuwählen auf denen das Model am unsichersten ist. Hierfr̈ sind Bayes Neuronale Netze von Nöten. Andere Methoden beachten etwa die Balanciertheit des Teildatensatzes oder aber dessen Diversität. Nicht zuletzt werden andere Neuronale Netze verwendet, um die Elemente des gesamten Datensatzes mit dem Teildatensatz repräsentieren oder rekonstruieren zu können. All diese Methoden werden evaluiert und die Performanz des Neuronalen Netzes wird mit der zufälligen Auswahl eines Datensatzes der gleichen Größe genauso wie mit dem Training auf dem gesamten Datensatz verglichen. Außerdem werden verschiedene Regulierungsmethoden (Regularization) auf das Training des Netzes angewandt, damit es sich nicht einfach die gelabelten Daten merken kann. Das Hauptresultat ist, dass sich durch die Kombination der Konstruktion eines balancierten Teildatensatzes und eines mit Elementen, auf welchem das Neuronale Netz am unsichersten ist, die wenigsten Daten (zehn Prozent) braucht, um eine genauso gute Genauigkeit durch das Training auf dem Testdatensatz zu haben, wie durch das Training auf dem gesamten Datensatz.

Table of contents

1 Introduction

In this chapter, active learning is introduced. First, in Section 1.1, the problem is stated. To understand the idea of active learning, it is compared with active learning in schools in Sections 1.1.1 and 1.1.2. Then, in Section 1.2 the use case of the company PreciBake GmbH, with which the thesis was written, is explained. Finally, in Section 1.3, the related work is presented.

The remainder of this work is organized as follows. In Section 2, the reader is provided with the notation, which is used in this work. Then, in Section 3 the active learning framework and the different subset selection methods are presented. In Section 4, these methods are evaluated and learning curves are presented in order to indicate the success of each subset selection method.

1.1 Motivation

Deep neural networks often need huge labeled data sets to perform well on unseen data. However, annotating the whole data set gets time and money consuming, whereas unlabeled data sets can be generated without much effort. For example, in computational pathology expert knowledge is needed to be able to annotate the given data. This expert may need a lot of time to label each data point correctly. Since such experts are expensive annotating a huge data sets gets also very expensive. Moreover, similar samples may be in the pool of unlabeled data and are therefore redundant to label. The unlabeled pool may also be class unbalanced, i.e. there are classes with considerably more samples than in other classes. To overcome these problems, active learning is introduced. Active learning is an algorithm (or framework), which chooses the data points on which the given deep neural network is trained. It aims to use fewer annotated data points while achieving similar test results (i.e. a similar accuracy on the test set). [58]

In order to understand the idea behind active learning, the application of active learning in the educational system will be introduced to compare it afterwards with active learning for Machine Learning.

1.1.1 Active Learning in School

As explained in [13] and [50], there exist passive and active learning as learning techniques in the educational system.

Passive Learning In passive learning the student or pupil does nothing or little in order to gain the knowledge. This means that the student is only observing the presented knowledge. The most common example for passive learning is lecture-style teaching. Especially when the student does not have any prior knowledge, passive learning may be more efficient than active learning. Also in the case in which the student has to gain lots of knowledge passive learning might be the best choice.

Active Learning Opposing to passive learning the student has to do something in active learning in order to gain the knowledge. Examples for active learning strategies are project-based learning, flipped classroom or any form of discussion in the classroom. The philosophy behind active learning is constructivism, which means that the student uses its current knowledge in order to build upon it new knowledge and to deepen the already acquired knowledge. In a project a student may use its current knowledge (combined with the knowledge of other students) and build upon it new knowledge or deepen the current knowledge in order to achieve the goal of the project. In the case of flipped-classrooms students have to prepare the knowledge, which they would otherwise obtain passively by frontal learning, before class such that they can use this knowledge for other activities during class. Thus, instead of getting only the knowledge in school, they have to prepare it on their own in order to apply it. Such activities reduce the lecturing time and yield to more interaction between the student, the task and especially the content.

For active learning it may be more difficult to understand what a student has learned since the knowledge differs from student to student because of its constructivism. Another problem of active learning is that it consumes more time and needs much more effort to be prepared by the teacher. But with active learning a student can learn how to learn and even deepen the knowledge much better than in passive learning.

1.1.2 From School to Machine Learning

Following the definition of [26] there are the same types of learning in Machine Learning.

Passive Learning In Machine Learning passive learning means that the network gets the whole set of data and does not need to do anything in order to acquire the labeled data. To query randomly instances, which need to be labeled by the oracle, can also be seen as passive learning since the model is not using any information about the data.

Active Learning In active learning the algorithm chooses the data it wants to learn from. It therefore constructs this data set step-wise. Active learning is, similar to the case of active learning in school, more time consuming because the network has be retrained after each step of the construction. Moreover, it has to acquire the instances on which it wants to be trained next, too. This means that the learner has to "understand" the content (i.e. the data), which it already knows (or has been trained on), better before it gets new information. This knowledge is necessary to choose those samples, which are most valuable for the learner. In lots of cases the time of the machine needed to train the network and to figure out which instances are most valuable to be labeled is not as important as the time of the annotator. Therefore, it is more important to save the annotator's time by providing fewer samples to label. The learner has to choose the data it wants to be labeled by an annotator (from a pool or a sequence) or it needs to create the instances which have to be labeled on its own (see section 3.1). Thus, similar to active learning in school also active learning for machine learning can be understood in the philosophy of constructivism.

The main difference between active and passive learning is the queuing component [65].

1.2 Application - The Company

This master's thesis is written in cooperation with the company PreciBake GmbH. PreciBake is an artificial intelligence (AI) and sensor technology company, which provides software and hardware solutions to the professional food and baking industry. To achieve this goal, PreciBake came up with the concept of a virtual baker, which functions as an AI-empowered assistance system for ovens [1]. This system is used by customers like grocery stores because bakery products build an important sector. The virtual baker provides solutions for baking sessions in order to receive a high quality of baking products and to automatize the baking process. This means that the virtual baker detects the baking products once they are loaded into the oven and chooses the corresponding baking program.

The detection process is a classification task. A model classifies the oven's input by a deep neural network which gives as its output a prediction of the corresponding class or in this case the name of the baking good. PreciBake gets the oven's input as image from its costumers. Thus, the company gets the unlabeled samples for free but needs to label them on its own. Because of the time and money consumption of the labeling process, PreciBake wants to reduce the amount of labeled data needed to achieve a high model performance. Moreover, in the given unlabeled data sets, there are often samples, which are similar to each othere annd the classes are not equally distributed. Therefore, active learning is needed in order to construct a suitable data set.

1.3 Related Work

In this thesis, an active learning framework is designed to overcome the stated problem. The presented framework is an extension of the one presented in [30]. It samples sequentially $k \in \mathbb{N}$ samples from a given pool of unlabeled samples $\mathcal{U}$ and annotates these samples.

The focus of this thesis lies in the sampling process. Different methods are tested, which can be grouped into uncertainty sampling, class balance sampling and representation sampling. Uncertainty sampling can be split into two classes: direct uncertainty sampling (known as uncertainty sampling [42]) and uncertainty by committee sampling (known as query by committee sampling [59]). Direct uncertainty sampling is using the probabilistic output of a classification network while uncertainty by committee sampling needs a committee of models with different hypothesis for the label prediction. In order to generate such a network, Bayesian neural networks are introduced. Since Bayesian neural networks are intractable, approximations are needed. They can get approximated by *Monte Carlo Markov Chain* methods [61]. These methods have a low bias but a high variance and therefore converge slowly. They are therefore not suitable for the application on deep neural networks. Thus, *Variational Inference* methods are introduced [25]. This method has been optimized for deep neural network, by taking advantage of the way the gradient is computed. This optimization is called *Bayes by Backprop* [8]. For convolutional neural networks, the local reparameterization trick yields an approximation which saves a lot computations [41]. Since these methods approximate the posterior distribu-

tion of the Bayesian neural network, it is still necessary to sample from this distribution to obtain a committee. Calculating a specific uncertainty directly from this distribution was shown to be infeasible [41]. In [17] a method is introduced, which creates a committee of realizations of a Bayesian neural network directly. This method uses stochastic noise (i.e. Dropout) during inference time, which is usually only enabled during training. Thus, a committee of different hypothesis is built by predicting the outcome of a network on a given sample several times. [64] proposes to enable Batch Normalization instead of Dropout during inference. Thus, the output of a mini batch of samples needs to be computed simultaneously to be able to apply this technique. Finally, ensembles of artificial neural networks can be used, to create a committee. Here, each member of the ensemble is a member of the committee [37].

For class balance sampling the unlabeled data set is divided into clusters. In order to do so the label can be predicted or a hash can be calculated according to specific properties of the input sample. In this thesis hashs are computed with *Local-Sensitivity Hashing* [14] using the feature vector of the sample.

Because the framework selects in every round not only one, but k samples, samples within the selected subset might be similar to each other. This is especially true for uncertainty sampling. Thus, [30] presented methods to find the most diverse or the most representative subset of a given set.

Since active learning deals with small data sets, it is important to prevent the classification network from overfitting. Thus, different regularization methods can be used for active learning. Stochastic noise can, for example, be injected to neural networks. This is done by *Dropout* [62] or *Batch Normalization* [29]. Moreover, data augmentation techniques [63] can increase the variation within the same data set and thus yield a more general representation. Finally, one can find the most vulnerable direction of an artificial neural network on a specific sample, the adversarial direction [20]. *Adversarial Training* trains the network in order to predict for each data point of the input space a label, which is similar to the labels of its neighbors in the adversarial direction. In *Virtual Adversarial Training* [46], the labels of the neighbors in the adversarial direction are predicted. This enables the use of unlabeled data sets as well. Since in active learning scenarios, the pool of unlabeled data is usually huge, this method is of particular interest.

2 Background

In this section, the reader is provided with the theoretical background, which will be used in the following chapters. In Section 2.1, the background about the basic statements and definitions of probability theory and statistics is given in order to apply these statements on artificial neural networks (introduced in Section 2.2) to introduce Bayesian neural networks (Section 2.3). Additionally, in Section 2.2 an introduction into optimization methods of artificial neural networks is given. Bayesian neural networks are necessary because they enable the user to obtain the networks uncertainty. Since Bayesian neural networks are intractable, approximations are introduced in Section2.3 as well. Having computed the uncertainty of an artificial neural network on a specific input image, this information can be used for active learning (see Chapter 3).

2.1 Probability Theory

In order to be able to define a probability measure and to later state Bayes' theorem, a few definitions need to be introduced. The theoretical foundation of a probability measure is a measurable space defined by [9]:

Definition 2.1. A system $\mathcal{F}$ of subsets of a nonempty set $\mathcal{X}$ is called **σ-algebra** if the following holds:

1. $\mathcal{X} \in \mathcal{F}$
2. $F \in \Omega \Rightarrow F^c := \mathcal{X} \backslash F \in \mathcal{F}$
3. $F_1, F_2, \ldots \in \mathcal{F} \Rightarrow \bigcup_{i=1}^{\infty} F_i \in \mathcal{F}$

The pair $(\mathcal{X}, \mathcal{F})$ is called **measurable space** or **event space**.

Using the definition of a measurable space, one can define the probability space in the following way:

Definition 2.2. Let $(\mathcal{X}, \mathcal{F})$ be a measurable space and let $F \in \mathcal{F}$. A map μ with μ(F) ≥ 0 or μ(F) $= \infty$ is called a **measure** if it satisfies:

1. $\mu(\varnothing) = 0$
2. $\mu\left(\bigcup_{i\geq 0} F_i\right) = \sum_{i\geq 0} \mu(F_i)$, $F_i \in \Omega$,

for every finite or infinite sequence $F_i \in \mathcal{F}$ with $F_1, F_2, ... \subset F$ being pairwise disjoint. The tripe (Ω, F, μ) is then called **measure space**. If $\mu(F) = 1$, then $p := \mu$ is called a **probability measure** and the measure space ($\mathcal{X}$, $\mathcal{F}, p$) is called **probability space**.

Using the definition of a probability measure, a random variable can be defined as done in [54].

Definition 2.3. Let $(\mathcal{X}, \mathcal{F})$, $(\Sigma, \mathcal{S})$ be measurable spaces. A map $X : \mathcal{X} \to \Sigma$ is called a measurable function or a **random variable** if it satisfies:

$$X^{-1}(S) = \{\chi \in \mathcal{X} : X(\chi) \in S\} \in \mathcal{F} \quad \forall S \in \mathcal{S}.$$

For simplicity, the following abbreviations are introduced:

Notation 2.4. Let $(\mathcal{X}, \mathcal{F}, p)$ be a probability space, $(\Sigma, \mathcal{S})$ a measurable space, $X : \mathcal{X} \to \Sigma$ be a random variable and $A \in \mathcal{S}$. Then

$$p(x) := p(\{\chi \in \mathcal{X} : X(\chi) = x\}),$$
$$p(A) := p(A) := p(\{\chi \in \mathcal{X} : X(\chi) \in A\}).$$

Using the definition of a probability measure and the previous notation, the conditional probability distribution can be defined as done in [54].

Definition 2.5. Consider a probability space $(\mathcal{X}, \mathcal{F}, p)$ and two events $A, B \in \mathcal{F}$ with $p(B) > 0$. Then, the **conditional probability distribution** of A given B is defined by

$$p(A \mid B) := \frac{p(A \cap B)}{p(B)}. \tag{2.1}$$

Having defined the conditional probability of one event, given another, one can finally state Bayes' Theorem:

Theorem 2.6 (Bayes' Theorem)**.** *Let $(\mathcal{X}, \mathcal{F}, p)$ be a probability space and $A, B \in \mathcal{F}$ with $p(B) > 0$. Then,*

$$p(A \mid B) = \frac{p(B \mid A) \cdot p(A)}{p(B)} \tag{2.2}$$

Proof. Chapter 2.1 of [54]. □

Given the conditional probability of an event X, given y and its probability. The probability of the event X can be obtained by marginalization. The following theorem will therefore be important for the calculation of the evidence of a Bayesian neural network (Section 2.3).

Theorem 2.7 (Marginalization)**.** *Let $(\mathcal{X}, \mathcal{F}, p_1)$, $(\mathcal{Y}, \mathcal{S}, p_2)$ be two probability spaces. Let $X \in \mathcal{F}$. Then it applies*

$$p_1(X) = \int_{\mathcal{Y}} p_1(X \mid y) p_2(y) \mathrm{d}y. \tag{2.3}$$

Proof. Chapter 5.1 of [54]. □

In order to train an artificial neural network, data needs to be observed, which is then used to optimize the artificial neural network's parameters. The foundation of such data observations and parameter optimization is statistics. Thus, the following definitions (founded on [35]) are of importance for artificial neural networks.

Definition 2.8. Let $(\mathcal{X}, \mathcal{F})$ be a measurable space. Let $(p_\omega)_{\omega\in\Omega}$ be a family of probability measures. Then the tripe $(\mathcal{X},\ \mathcal{F},\ (p_\omega)_{\omega\in\Omega})$ is called **statistical model**. $\mathcal{X}$ is called **sample space**.

Definition 2.9. Let $(\mathcal{X}, \mathcal{F}, (p_\omega)_{\omega\in\Omega})$ be a statistical model. A **statistic** is a measurable function $S : \mathcal{X} \to \Sigma$. Let $\tau : \Omega \to \Sigma$ be a map. Then a statistic $T : \mathcal{X} \to \Sigma$ is called **estimator** for τ.

Definition 2.10. Let $(\mathcal{X}, \mathcal{F}, (p_\omega)_{\omega\in\Omega})$ be a statistical model with probability mass function p_ω. The function $p : \mathcal{X} \times \Omega \to [0, 1]$ with $p(\mathcal{D}, \omega) := p_\omega(\mathcal{D})$ is called **likelihood function**. The map $p_\mathcal{D} : \Omega \to [0, 1]$ with $\mathcal{D} \mapsto p(\mathcal{D}, \omega)$ is called **likelihood function** to the event $\mathcal{D} \in \mathcal{F}$. An estimator $T : \mathcal{X} \to \Omega$ is called **maximum likelihood estimator**

$$\text{if } p(\mathcal{D}, T(\mathcal{D})) = \max_{\omega\in\Omega} p(\mathcal{D}, \omega)\ \mathcal{D} \in \mathcal{F}, \tag{2.4}$$

i.e. $p_\mathcal{D}$ is maximal at $T(\mathcal{D})$.

Having observed data X and its corresponding labels Y, the aim of the maximum likelihood estimator is to find parameters $\omega \in \Omega$ such that the given labels $y \in Y$ are most likely to belong to the corresponding samples $x \in X$. $\mathcal{D}$ is written instead of Y given X. Thus, $\mathcal{D}$ represents the observed data and $p(\mathcal{D})$ its corresponding probability.

The probability of the likelihood function can, especially in the case of a classification task, be seen as the output of an artificial neural network.

The aim of probability theory is to find the properties of the outcomes of a given probability distribution, whereas the inverse problem is to find the properties of a probability distribution if the outcomes are given. This problem is known as statistical inference. This shifts the problem from maximizing $p(\mathcal{D}|\omega)$ w.r.t. ω to the problems of calculating the posterior distribution $p(\omega|\mathcal{D})$ and use the posterior to predict the outcome of the model. Also, this leads to distributed parameters and therefore Bayesian neural networks.

The following definition names the conditional distribution of the data given the parameters (likelihood) as well as the probability distribution of the parameters (prior) and the probability of the given data (evidence).

Definition 2.11. Let $(\mathcal{X}, \mathcal{F}, (p_\omega)_{\omega\in\Omega})$ be a statistical model. Let $\mathcal{D} \in \mathcal{F}$ and $\omega \in \Omega$. Then, the probability density function $p : \Omega \times \mathcal{X} \to [0, 1]$ with $p(\omega, \mathcal{D}) := p(\omega|\mathcal{D})$ is called **posterior distribution**. The probability density function p of ω, $p : \Omega \to [0, 1]$, is called **prior distribution** of ω. And the probability density function of X $p : \mathcal{X} \to [0, 1]$, $p(\mathcal{D})$ is called **evidence** or normalizer.

Onto those probability distributions Bayes' theorem can be applied:

Theorem 2.12. *Let $(\mathcal{X}, \mathcal{F}, (p_\omega)_{\omega\in\Omega})$ be a statistical model. Let $\mathcal{D} \in \mathcal{F}$. Then, the posterior distribution can be computed using the likelihood, prior and evidence:*

$$p(\omega|\mathcal{D}) = \frac{p(\mathcal{D}|\omega) \cdot p(\omega)}{p(\mathcal{D})} \tag{2.5}$$

Proof. This is an application of Bayes' Theorem 2.6. □

Having a prior distribution applied on the neural networks parameters, their posterior distribution can be computed by the application of Theorem 2.12. Having computed the posterior distribution, the probability of a label y^*, given a new instances x^* to the neural network, can be determined by Bayesian inference:

Theorem 2.13. *Let $(\mathcal{X}, \mathcal{F}, (p_\omega)_{\omega\in\Omega})$ be a statistical model and let $\mathcal{D} \in \mathcal{F}$. Then, integrating over all possible $\omega \in \Omega$ (Marginalization 2.7) gives the probability for a new data point x^* belonging to label y^**

$$p(y^*|x^*, \mathcal{D}) = \int_\Omega p(y^*|x*, \omega)p(\omega|\mathcal{D})\mathrm{d}\omega.$$

*This process is called **inference** or posterior predictive distribution.*

Proof.

$$\begin{aligned} p(y^*|x^*\mathcal{D}) &\overset{2.7}{=} \int_\Omega p(y^*|x^*, \omega, \mathcal{D})p(\omega|\mathcal{D})\mathrm{d}\omega \\ &= \int_\Omega p(y^*|x^*, \omega)p(\omega|\mathcal{D})\mathrm{d}\omega \end{aligned} \tag{2.6}$$

in Equation 2.6 x^* is assumed to be conditionally independent to X [19]. □

Instead of summing over all possible parameters ω with their corresponding probability, a point estimate of the parameters can be used as well, given the posterior. This point estimate is called maximum a posteriori estimate [23]. But as it is only a point estimate, it is not often used in practice.

Definition 2.14. Let $(\mathcal{X}, \mathcal{F}, (p_\omega)_{\omega\in\Omega})$ be a statistical model. An estimator $T : \mathcal{X} \to \Omega$ is called **maximum a posteriori** estimator, if

$$p(T(\mathcal{D}), \mathcal{D}) = \max_{\omega\in\Omega} p(\omega, \mathcal{D}) \quad \forall \mathcal{D} \in \mathcal{F}.$$

2.2 Artificial Neural Networks

For the classification of for example an oven's input, a classification model is needed. The best classification results, according to the model's accuracy, were achieved by deep neural networks, which are layered feed forward neural networks with a high number of layers. Thus, both feed forward neural networks, and more generally, artificial neural networks will be introduced in this chapter. At the end of this chapter Bayesian neural networks are introduced, which are needed to be able to calculate the model's uncertainty.

2.2.1 Introduction

Artificial neural networks aim to mimic the human brain. Since the human brain is a network consisting of neurons or nerve cells (see Figure 1), an artificial neural network is a network of artificial neurons or nodes (see Figure 2). Artificial neural networks can be used for classification tasks.

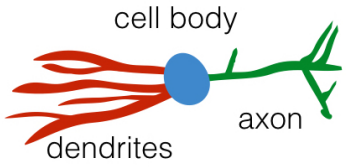


Figure 1: A sketch of a biological **neuron**.

Definition 2.15. Let $N \in \mathbb{N}$, $x \in \mathbb{R}^N$. Then an **artificial neuron** is defined as the function

$$n\colon \mathbb{R}^N \to \mathbb{R},\ n(x) = \sigma(w^T x + b),$$

where $\sigma : \mathbb{R} \to \mathbb{R}$ is a non-linear function (called activation function), $w \in \mathbb{R}^N$ and $b \in \mathbb{R}$ are the parameters of the neuron.

Following the introduction of [68], the combination and composition of artificial neurons results in an artificial neural network. It can be described by a weighted, directed graph $G = (V, E)$ with vertices corresponding to neurons, input or output nodes. If the graph G is acyclic, it represents a feedforward neural network, otherwise it represents a recurrent neural network. If the neurons of a feedforward neural network can be organized in distinct layers and there only exist connections between neurons of two neighboring layers, then this network is called layered feedforward neural network (see Figure 3). Usually, the activation functions are in all hidden layers the same and only at the output a different activation function is chosen depending on the task the model should be trained on.

Definition 2.16. Let $N \in \mathbb{N}$, $x \in \mathbb{R}^N$ be the input vector. Let l_i be the i-th layer of artificial neurons n_{ij}, $j \in \{1, ..., i_{k_i}\}$, where $k_i \in \mathbb{N}$ is the number of neurons of l_i. Let

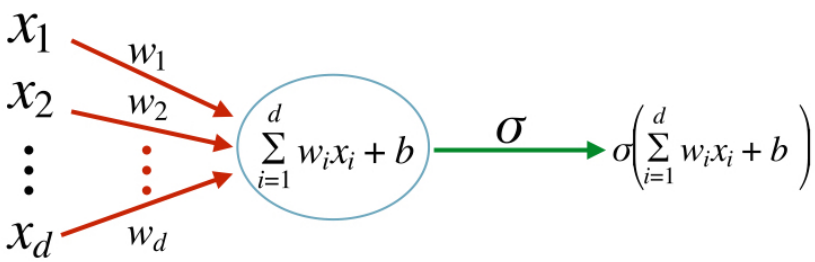


Figure 2: An **artificial neuron** with input $x \in \mathbb{R}^d$, parameters $w \in \mathbb{R}^d, b \in \mathbb{R}$ and a nonlinear activation function σ.

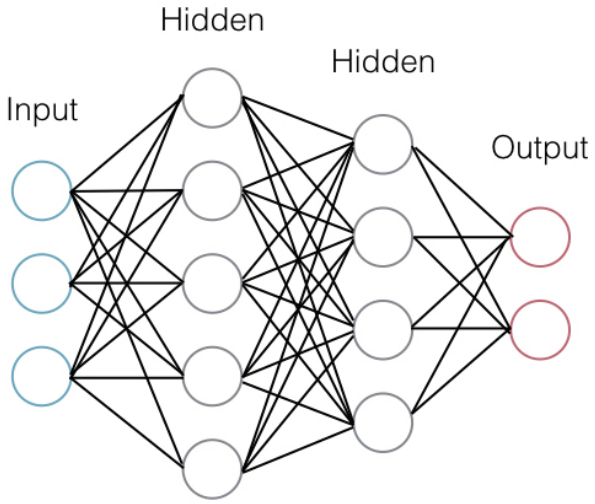


Figure 3: An artificial **neural network** consisting of an input layer, two hidden layers and an output layer. The first hidden layers consists of five artificial neurons and the second one of four.

$w_{ij} \in \mathbb{R}^k$, $b_{ij} \in \mathbb{R}$ be the parameters of such a neuron. Let

$$W_i := \begin{pmatrix} w_{i1}^T \\ w_{i2}^T \\ ... \\ w_{ik_i}^T \end{pmatrix}, \; B_i := \begin{pmatrix} b_{i1} \\ b_{i2} \\ ... \\ b_{ik_i} \end{pmatrix},$$

Then a **fully connected** layered feedforward **neural network** (FC-NN) f with m layers is defined as the function

$$f : \mathbb{R}^N \to \mathbb{R}^{k_m}, f(x) = \sigma_{out}(W_m(...(\sigma(W_2(\sigma(W_1 x + B_1)) + B_2))...) + B_m),$$

where $\sigma, \sigma_{out} : \mathbb{R}^{k_i} \to \mathbb{R}^{k_i}$ are non-linear functions (called activation functions), $W_i \in \mathbb{R}^N$ and $B_i \in \mathbb{R}^{k_i}$ are the parameters of the neuron.

https://www.overleaf.com/project/5d9db704e8ac710001ea8333 Fully connected layered feedforward neural networks will in the following be called fully connected neural networks to avoid the long name. Moreover, for classification tasks the output dimension k_m of a neural network is equal to the number of classes C.

For image classification tasks, artificial neural networks are used with images as input. Using only fully connected layers leads to a very high number of parameters of the artificial neurons in the network. This is especially the case for a high image resolution. Thus, convolutional neural networks will now be introduced.

Convolutional Neural Network Since the input of convolutional neural networks (CNN) is always an image, a specific property of images can be used in order to have less computations. Namely, this property is that in a specific region of the image, the pixels do not differ much. Images are of the form $h \times w \times c$, (e.g. $128 \times 128 \times 3$) with h being

its height, w its width and c the number of its channels. RGB images have 3 channels. A convolutional layer consists of $K \in \mathbb{N}$ filters applied on the channels of the previous layer. Each filter is applied on all regions of the filter's size. A filter consists of a weight matrix $\omega \in \mathbb{R}^{h \times w}$ and a bias $b \in \mathbb{R}$. Therefore, only $K \times h \times w + 1$ learnable parameters are used in each layer. As can be seen in the appendix, the kernel size (dimension of the filter) is most of the time less or equal than four. With less than 256 filters, the number of learnable parameters is reduced enormously. The architecture of a convolutional neuronal network consists of convolutional layers, followed by non-linear functions (an activation function) and pooling layers and at its end it consists of fully-connected layers.

Convolutional Layer A convolutional layer is as described in [17] a linear transformation, which preserves spatial information of the input image. The convolution of a sequence of K_i kernels $\mathcal{K}_{k'}$ with $k' \in \{1, ..., K_i\}$ is a convolutional layer. Each kernel $\mathcal{K}_k$ consists of a filter $W_{k,k'} \in \mathbb{R}^{h \times w \times K_{i-1}}$ and a bias $b_k \in \mathbb{R}$. Usually, the bias is set to 0. Let $x \in \mathbb{R}^{H_{i-1} \times W_{i-1} \times K_{i-1}}$ be the input of a convolutional laye and the output of the layer is $y \in \mathbb{R}^{H_i \times W_i \times K_i}$. Each $y_{i,j,k}$ is the sum over all n patches of the element wise product of the filter $W_{k,k'}$ and a patch of the input of size $h \times w \times K_{i-1}$.

$$y_k = \sum_{k'=1}^{K_{i-1}} W_{k,k'} * x_{k'} + b_k \tag{2.7}$$

with the convolutional operation $W_{j,k} * x_{i,k}$ being:

$$(W_{k,k'} * x_k)_{i,j} = \sum_{o=1}^{h} \sum_{l=1}^{w} \omega_{o,l} \cdot x'_{si+o-1,sj+l-1}, \tag{2.8}$$

with $x' = x_k$ with padding p, weigths $\omega_{o,l} \in \mathbb{R}^{h \times w}$(and $\omega_{o,l} \in W_{k,k'}$)) and stride s. Padding is the addition of pixels with value 0 to the edges of the input x. The factor p describes how many rows and columns of zeros are added to the input. The stride is defined by the amount of movement between two applications of the filter to the input. Usually, it has the same value for the horizontal and vertical direction.

The convolution can be seen as a linear operation as well. It can be composed to the matrix product

$$\hat{x} * W + b = \hat{y} \in \mathbb{R}^{n \times K_i} \tag{2.9}$$

with $\hat{x} \in \mathbb{R}^{n \times hwK_{i-1}}$, $W \in \mathbb{R}^{hwK_{i-1} \times K_i}$ and $b \in \mathbb{R}^{K_i}$, where $\hat{x}$ is a matrix having in each row a vectorized patch of the input. As we have n such patches this yields $\hat{x}$ consisting of n rows. The weight matrix W has in its columns each filter vectorized. It therefore consists of K_i such vectors. Since the number of patches depends on H_{i-1} and W_{i-1} and $n = H_i * W_i$ with

$$H_i = \left\lfloor \frac{H_{i-1} + 2 * p - h - 1 - 1}{s} + 1 \right\rfloor \text{ and } W_i = \left\lfloor \frac{W_{i-1} + 2 * p - w - 1 - 1}{s} + 1 \right\rfloor,$$

$\hat{y}$ can be rearranged to $y \in \mathbb{R}^{H_i \times W_i \times K_i}$.

As already mentioned, a convolutional neural network's architecture can furthermore contain the following layers:

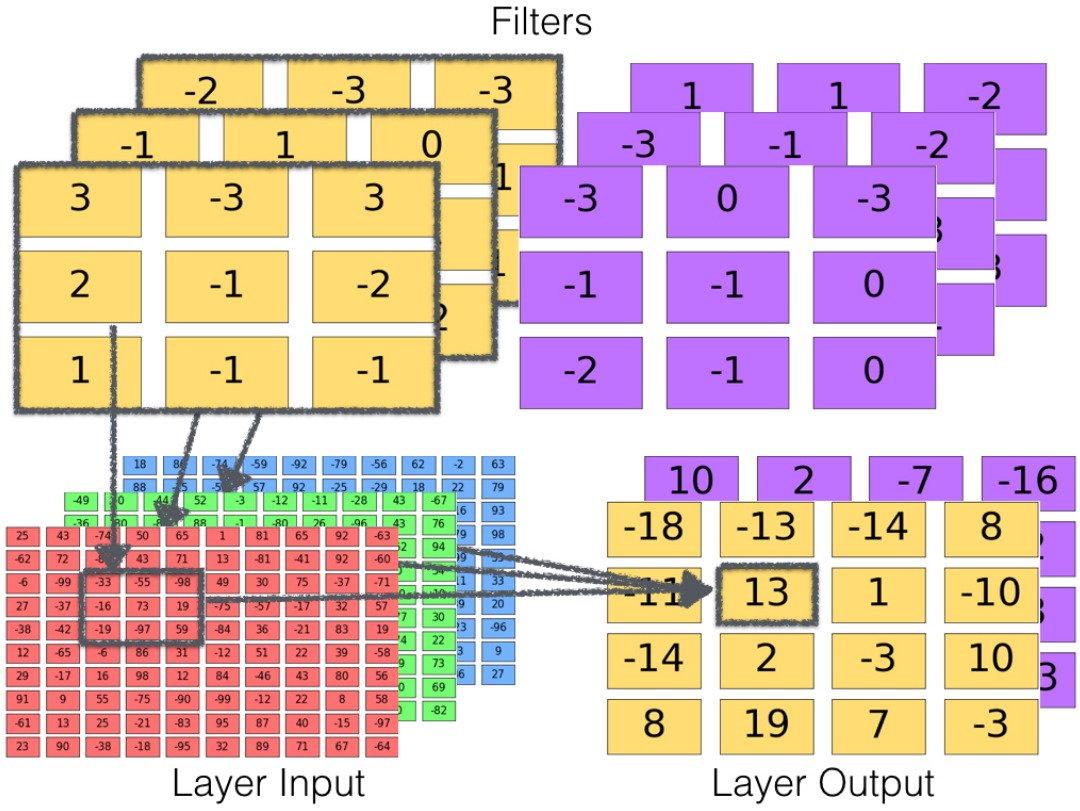


Figure 4: Visualization of a **convolutional layer** with an input of size $10 \times 10 \times 3$ with 3 being the number of channels, two $3 \times 3 \times 3$ filters and an output of size $4 \times 4 \times 2$, where 2 is the number of channels of the output which equals the number of filters.

Pooling Layer In this layer the dimension of a layer is reduced by the application of a nonlinear function on a region of the output of the layer. These regions are usually quadratic and distinct. Applied nonlinear functions are for example the maximum or the average of the region, e.g. a 2×2 kernel.

Noise Layer Noise layers inject stochastic noise to a previous layer. These layers will be of Importance for the approximation of Bayesian neural networks. Two examples for noise layers are Batch Normalization layers and Dropout layers. These examples are described in the following.

Batch Normalization Layer Batch normalization is a unit-wise operation, which was proposed by [29]. Its aim is to standardize the distribution of the input of each unit of the previous layer. It is applied before the activation function and after the linear transformation, i.e.

$$\sigma(\mathcal{BN}_{\beta,\gamma}(Wx + B)). \tag{2.10}$$

The output of a node in a fully connected layer is 1 dimensional. Thus, the mean and variance of a mini batch (introduced in Section 2.2.2) $\mathcal{B}$ are calculated in the following

way:

$$\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} h_i \tag{2.11}$$

$$\sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{i=1}^{m}(h_i - \mu)^2 \tag{2.12}$$

On the other hand, for a convolutional layer as the previous layer, batch normalization is calculated as follows:

$$h_i^{j,k} := h_i^{j+k*W_i} \tag{2.13}$$

$$\mu_{\mathcal{B}} = \frac{1}{m + H_i * W_i}\sum_{i=1}^{m}\sum_{j=1}^{W_i}\sum_{k=1}^{H_i} h_i^{j,k} \tag{2.14}$$

$$\sigma_{\mathcal{B}}^2 = \frac{1}{m + H_i * W_i}\sum_{j=1}^{W_i}\sum_{k=1}^{H_i}(h_i^{j,k} - \mu)^2 \tag{2.15}$$

Thus, Batch normalization is computed by using algorithm 1.

Data: Mini batch $\mathcal{B} = \{x_1, ..., x_m\}$, hyperparameters γ, β
Result: Batch normalized y
1: $\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} h_i$ or $\mu_{\mathcal{B}} = \frac{1}{m+H_i*W_i}\sum_{i=1}^{m}\sum_{j=1}^{W_i}\sum_{k=1}^{H_i} h_i^{j,k}$
2: $\sigma_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m}(h_i - \mu)^2$ or $\sigma_{\mathcal{B}} = \frac{1}{m+H_i*W_i}\sum_{j=1}^{W_i}\sum_{k=1}^{H_i}(h_i^{j,k} - \mu)^2$
3: $\hat{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$
4: $y_i = \gamma\hat{x}_i + \beta$

Algorithm 1: Batch Normalization by [29]

Batch normalization is during training time a stochastic process due to the fact that the samples in the mini-batches are chosen randomly. This randomness influences μ and σ in a way that having different samples in a batch leads to different μ and σ.
During evaluation time the batch becomes the whole training set.

Dropout Layer In this layer, it is with probability of $p \in (0, 1)$ decided whether an artificial neuron or a kernel of the previous layer should be set to zero (i.e. dropped out). This is done in order to prevent the network from overfitting, but it can be used in order to make the model stochastic. The model becomes stochastic because each neuron is randomly set to zero during training time. During test time no neuron will be set to zero.

2.2.2 Optimization

Classification There are two known tasks for artificial neural networks. On one hand, there exists the regression task for which the output of the artificial neural network is a

continuous value $y \in \mathbb{R}$. On the other hand, for the classification task the neural network has values of the output of the form of a class label $y \in \mathcal{C}$ with $\mathcal{C}$ being a discrete set. In this thesis the methods are only applied on classification tasks.

Loss and risk function In order to be able to optimize an artificial neural network's parameters it is necessary to measure how well the artificial neural network's prediction approximates the real labels. Therefore, a loss function is needed, which quantifies the distance between the prediction and the ground truth. For classification tasks, the cross-entropy loss is used.

Definition 2.17. Let $f_\omega : \mathcal{X} \to \mathcal{Y}$ with f_ω being a realization of an artificial neural network. Let ω be its parameters. Then the **loss function** is defined in $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$. And the **Cross Entropy** loss can be defined as

$$L_{CE}(x, y) = -log\left(\frac{exp(f_\omega(x)_y)}{\sum_{c=1}^{C} exp(f_\omega(x)_c)}\right) \tag{2.16}$$

.

Having a loss function the risk function is defined as follows:

Definition 2.18. Let $(\mathcal{X} \times \mathcal{Y}, \mathcal{F}, p)$ be a probability space, $f_\omega : \mathcal{X} \to \mathcal{Y}$ with f_ω being a realization of an artificial neural network. Let ω be its parameters and $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ be a loss function. The **risk function** $R : \{\mathcal{C}^1(\mathcal{X}; \mathcal{Y})\} \to [0, \infty)$ is defined as the average loss, i.e.

$$R(h) := \int_{\mathcal{X}\times\mathcal{Y}} L(y, f_\omega(x))\mathrm{d}p(x, y) \tag{2.17}$$

The **empirical risk function** $\hat{R} : \{\mathcal{C}^1(\mathcal{X}; \mathcal{Y})\} \to [0, \infty)$ is defined as its empirical approximation, i.e.

$$\hat{R}(h) := \frac{1}{n}\sum_{i=1}^{n} L(y_i, f_\omega(x_i)) \tag{2.18}$$

with $(x_i, y_i) \in (\mathcal{X}, \mathcal{Y})\ \forall i \in \{1, ..., n\}$.

Gradient Descent Having the risk function, the aim is to find the parameters of the artifical neural network which yield the smallest risk function value. In general, deep neural networks are high-dimensional and non-convex. Thus, the global minimum of the risk function is not easy to obtain. In order to find a local minimum an algorithm is used which aims to descent from the given starting point to a minimum. A reasonable direction in which one should descent is the steepest one.

Definition 2.19. Let $f \in \mathcal{C}^1(\mathbb{R}^n; \mathbb{R}),\ x \in \mathbb{R}^n$ with $\nabla f(x) \neq 0$. Then, the solution d^* of

$$\arg\min_{d\in\mathbb{R}^n} \nabla f(x)^T d \tag{2.19}$$

is called **steepest descent direction** of f in x.

Theorem 2.20. *Let $f \in \mathcal{C}^1(\mathbb{R}^n;\mathbb{R})$, $x \in \mathbb{R}^n$ with $\nabla f(x) \neq 0$. Then*

$$d^* = -\frac{\nabla f(x)}{|| \nabla f(x) ||} \tag{2.20}$$

is the unique solution of 2.19.

Proof. see Figure Chapter 7.1 of [51]. □

In order to have a step size, which is neither to small nor to wide, the following rule for updating the step size is introduced.

Definition 2.21 (Armijo rule)**.** Let $\gamma, \beta \in [0,1]$, $f \in \mathcal{C}^1(\mathbb{R}^n;\mathbb{R})$. Take the highest $\sigma_k \in \{1, \beta, \beta^2, ...\}$ with

$$f(x^k + \sigma_k s^k) - f(x^k) = \sigma_k \gamma \nabla f(x^k)^T s^k. \tag{2.21}$$

This leads to Algorithm 2 (Gradient Gescent).

Data: starting point x^0, hyperparameters γ, β for 2.21
Result: sequence $\{x^k\}_{k\geq 0}$
1: **for** $k \in \{1, 2, ...\}$ **do**
2: **if** $\nabla f(x^k) = 0$ **then**
3: STOP
4: **end if**
5: $s^k = -\nabla f(x^k)$
6: define σ_k with definition 2.21
7: set $x^{k+1} = x^k + \sigma_k s^k$
8: **end for**

Algorithm 2: Gradient Descent

Algorithm 2 converges to a stationary point, which is in some cases a local minimum.

Theorem 2.22 (Convergence of Gradient Descent)**.** *Let $\gamma, \beta \in [0,1]$, $f \in \mathcal{C}^1(\mathbb{R}^n;\mathbb{R})$. Then, Algorithm 2 is terminating with $\nabla f(x^k) = 0$ or with a sequence with the following properties:*

1. *$\forall k : f(x^{k+1}) < f(x^k)$,*
2. *For all accumulation point $\bar{x}$ of the sequence it holds: $\nabla f(\bar{x}) = 0$*

Proof. see Figure Chapter 7.3 of [51]. □

There are more efficient algorithms to optimize the parameters of an artificial neural network, which have for example a different learning rate for each parameter. Adam [32] is one algorithm of this kind.
In order to apply Gradient Descent or Adam it is necessary to compute the gradient of the artificial neural network's risk function with respect to its parameters. As this function gets especially for deep neural networks very complex the following trick is used:

Back-propagation The idea of back-propagation [55] is to use the chain rule in order to divide the complex derivative of the empirical risk function into simple subfunctions of which a closed derivative exits. This is, for simplicity, only demonstrated with a fully connected neural network f_ω. Let $h^m = f_\omega(x) = \sigma_{out}(z^m)$, $z^m = w^{mT}h^{m-1} + b^m$ $h^l = \sigma(z^l)$, $z^l = w^{l^T}h^{l-1} + b^l$, $l \in \{0, ..., m-1\}$ and m being the number of layers. Then, one gets by back-propagation the following derivatives:

$$\frac{h_j^m}{\partial z_j^m} = \sigma'_{out}(z_j^m) \tag{2.22}$$

$$\delta_j^m := \frac{\partial \hat{R}}{\partial z_j^m} = \sum_{k=1}^{N_m} \frac{\partial \hat{R}}{\partial h_k^m}\frac{\partial h_k^m}{\partial z_j^m} = \frac{\partial \hat{R}}{\partial h_j^m}\sigma'_{out}(z_j^m) \tag{2.23}$$

$$\delta_j^l := \frac{\partial \hat{R}}{\partial z_j^l} = \sum_{k=1}^{N_l} \frac{\partial \hat{R}}{\partial z_k^{l+1}}\frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_{k=1}^{N_l} \frac{\partial \hat{R}}{\partial z_k^{l+1}} w_{k,j}^{l+1}\sigma'(z_j^l) = \sum_{k=1}^{N_l} \delta_k^{l+1} w_{k,j}^{l+1}\sigma'(z_j^l) \tag{2.24}$$

$$\frac{\partial \hat{R}}{\partial w_{i,j}^l} = \sum_{k=1}^{N_l} \frac{\partial \hat{R}}{\partial z_k^l}\frac{\partial z_k^l}{\partial w_{i,j}^l} = \delta_j^l h_j^{l-1} \tag{2.25}$$

$$\frac{\partial \hat{R}}{\partial b_i^l} = \sum_{k=1}^{N_l} \frac{\partial \hat{R}}{\partial z_k^l}\frac{\partial z_k^l}{\partial b_i^l} = \delta_j^l \tag{2.26}$$

Thus, to compute the partial derivative with respect to a parameter of layer l, δ_k^l has to be computed for every neuron k of the layer. To compute δ_k^l of layer l and neuron k the value of z_k^l has to be stored from the forward pass and δ_j^{l+1} has to be used from the following layer $l+1$. Hence, back-propagation is done recursively in the reverse direction of the forward pass and therefore also known as backward pass.

Mini Batch In the gradient descent algorithm the gradient of the empirical risk function is calculated (also called batch gradient descent). This computation is too expensive if the number of samples in the training set is high. On top, this may converge to a local minimum or saddle point and does not find the global minimum.

Instead, one may calculate the gradient of the empirical risk function over only one sample and then update the parameters. This leads to less computational costs per iterations (one update of the parameters) but there are lots of iterations and therefore updates of the parameters per epoch (one full training circle) needed. This is called stochastic gradient descent. It has in average the same empirical risk, but it is more noisy and may overjump a local minimum or saddle point but also may not stop at a global minimum.

Mini-batch gradient descent is a mixture of both methods and uses a mini-batch of size 2^k with k chosen depending on the size of the data and its resolution.

Validation Set Gradient Descent (Algorithm 2) or a similar algorithm is applied on the training set. In order to find the optimal number of steps e (called epochs) a validation set is used. Thus, the training stops, when the value of the risk function on the validation

set converges or before it starts to get higher again. This set consists of 10 to 30 percent of the labeled data.

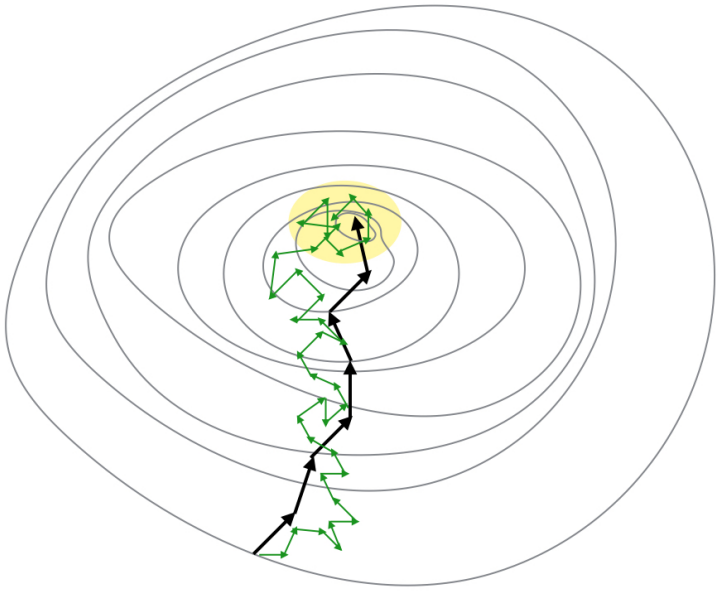


Figure 5: **Gradient descent** (black) converges to an accumulation point (see Theorem 2.22) whereas **stochastic gradient descent** (green) only converges to the yellow region. Mini Batch Gradient Descent is a mixture of both methods.

2.2.3 Bayesian Neural Networks

A Bayesian neural network is an artificial neural network with a probability distribution placed on its parameters. In order to be Bayesian this is achieved by placing a prior distribution on every weight and after observing the labeled data, a posterior distribution is calculated for every weight.

Definition 2.23. Let $(\mathcal{X}, \mathcal{F}, (p_\omega)_{\omega\in\Omega})$ be a statistical model. Let $\mathcal{D} \in \mathcal{F}$, f_ω be a fully-connected neural network and ω be the network's parameters (i.e. the weigths and biases). If the parameter's distribution is given by the posterior (Theorem 2.12), then f_ω is a **Bayesian neural network** (BNN).

Bayesian Convolutional Neural Network A Bayesian convolutional neural network is a convolutional neural network with its filters being additionally threatened in a Bayesian manner.

Definition 2.24. Let $(\mathcal{X}, \mathcal{F}, (p_\omega)_{\omega\in\Omega})$ be a statistical model. Let $\mathcal{D} \in \mathcal{F}$, f_ω be a convolutional neural network and ω be the network's parameters (i.e. the filters, biases and weights). If the parameter's distribution is given by the posterior (Theorem 2.12), then f_ω is a **Bayesian convolutional neural network** (BCNN).

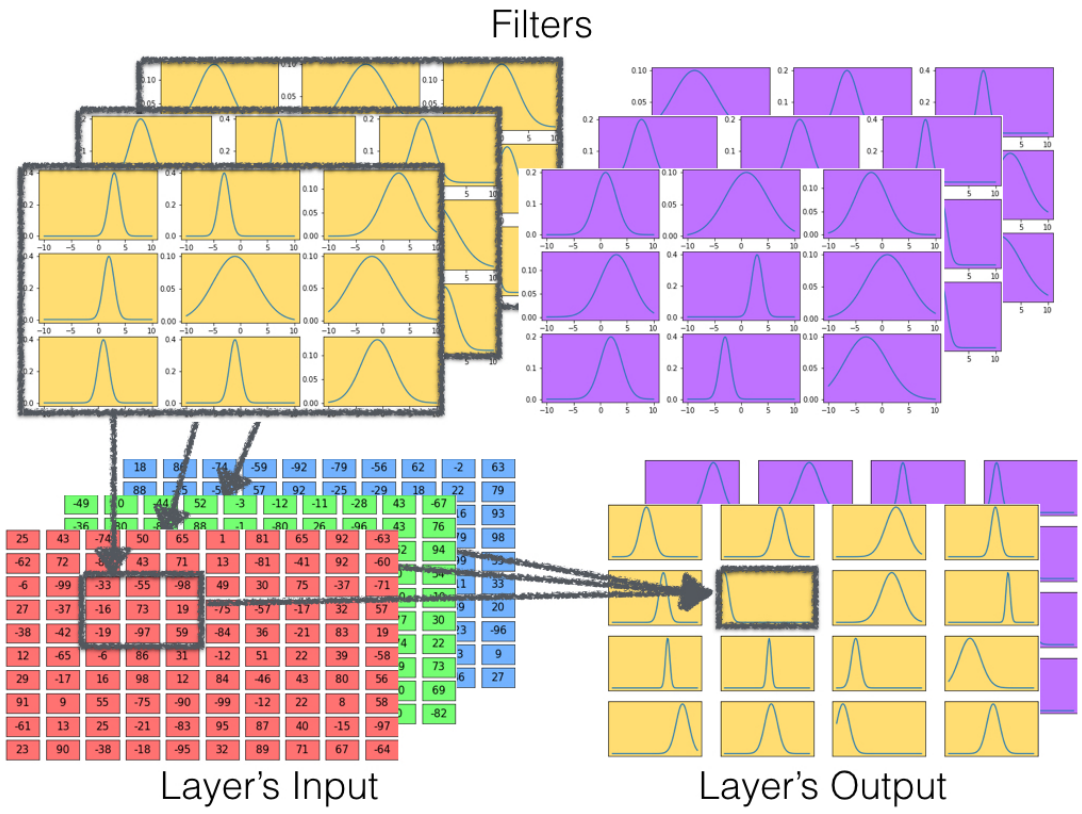


Figure 6: Visualization of a **Bayesian convolutional layer** with an input of size $10 \times 10 \times 3$ with 3 being the number of channels, 2 $3 \times 3 \times 3$ filters and an output of size $4 \times 4 \times 2$, where 2 is the number of channels of the output which equals the number of filters.

The given observation of data $\mathcal{D}$ is in the supervised case the same as observing the samples X and the corresponding labels Y. The probability of observing $\mathcal{D}$ is the same as observing the labels Y given the samples X, i.e.:

$$P(\mathcal{D}) = P(X \mid Y). \tag{2.27}$$

Theorem 2.12 can therefore be rewritten as:

$$p(\omega \mid X, Y) = \frac{p(Y \mid X, \omega)p(\omega)}{p(Y \mid X)}, \tag{2.28}$$

with $p(Y \mid X, \omega)$ being a neural network (convolutional neural network) with fixed parameters ω. The underlying prior distribution of the parameters is represented by $p(\omega)$ and $p(Y \mid X)$ is the model's evidence.

Theorem 2.25. *Let $(\mathcal{Y}, \mathcal{F}, (p_\omega)_{(\omega \in \Omega)})$ be a statistical model, $(\Omega, \mathcal{S}, p)$ be a probability space and let $Y \in \mathcal{F}, \omega \in \Omega$. Then, the evidence can be determined by integration over all possible model parameters $\omega \in \Omega$.*

$$p(Y|X) = \int_\Omega p(Y|X, \omega)p(\omega) \, \mathrm{d}\omega$$

Proof. Marginalization, theorem 2.7. □

The evidence is therefore the probability of observing the data under all possible parameter values, which are weighted by their respective prior probabilities.

2.3 Approximation of Bayesian Neural Network

Intractability of the Evidence In higher dimensions (and this is the case in Machine Learning, especially for computer vision tasks) calculating the evidence becomes intractable. Thus, the exact computation of the posterior is infeasible and cannot analytically be done. Therefore, approximation techniques like Markov Chain Monte Carlo or Variational Inference are needed.

2.3.1 Markov Chain Monte Carlo

As stated in [53], Markov Chain Monte Carlo (MCMC) consists of two parts, namely Markov Chains and Monte Carlo sampling. To generate sample (or states) from a distribution, Monte Carlo sampling is used. Markov Chains are used to determine the probabilities for the sampling of a new state, given the current state. A Markov Chain is chosen, which is insensitive to a normalization factor (namely a reversible Markov Chain) [53]. If the Markov Chain's stationary distribution is the posterior distribution (likewise the non-normalized posterior distribution), samples generated after a certain amount of time (burn-in B) are distributed according to the stationary distribution. Such a Markov Chain can be generated using the Metropolis-Hasting or the Gibbs Sampling algorithm [53]. In order to get independent samples from the posterior distribution, not directly subsequent samples should be taken but rather samples should be chosen after a lag L (see Algorithm 3). (The lag L can be estimated with the help of the autocorrelation function.) An initial sample can be generated using the prior distribution and Monte Carlo sampling.

Data: Markov Chain's transition probabilities q, prior distribution p, burn-in B, Lap L
Result: samples $z_0, ..., z_N$ from arbitrary close approximation of normalized posterior distribution
1: draw $s_0 \sim p(s_0)$
2: **for** i in {1,...,L*N+B} **do**
3: draw $s_i \sim q(s_i|s_{i-1})$
4: **end for**
5: **for** i in {0,...,N} **do**
6: choose $z_i = s_{i*L+B}$
7: **end for**

Algorithm 3: Markov Chain Monte Carlo sampling

"This sequence is constructed so that, although the first sample may be generated from the prior, successive samples are generated from distributions that probably get closer and closer to the desired posterior." [36]

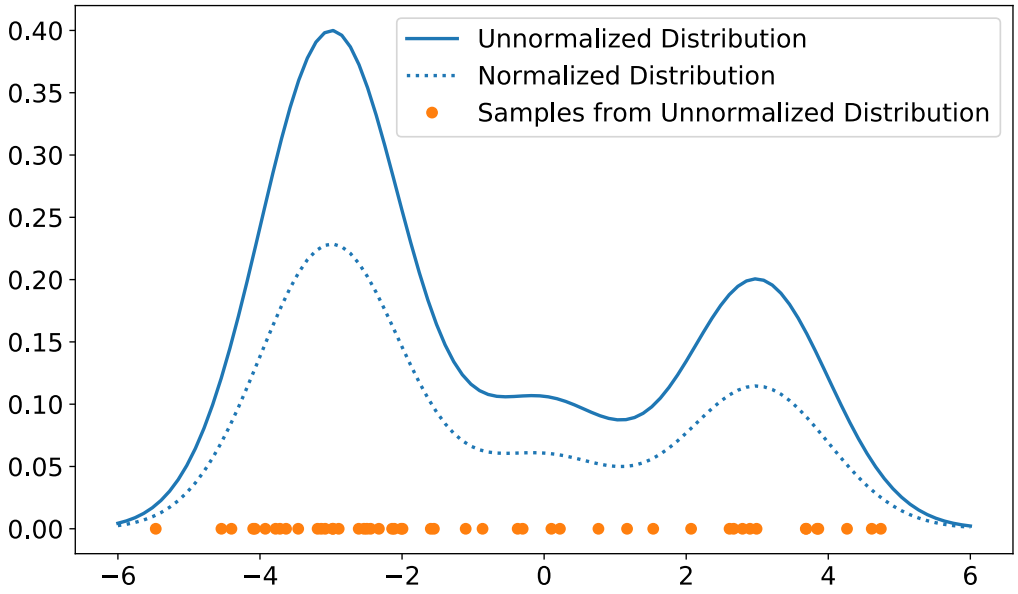


Figure 7: Samples generated by the **Markov Chain Monte Carlo** algorithm after a burn-in B and using a lap L.

Having generated samples from the posterior distribution, those samples can be used in order to calculate statistics of the posterior distribution (e.g., its mean, variance).

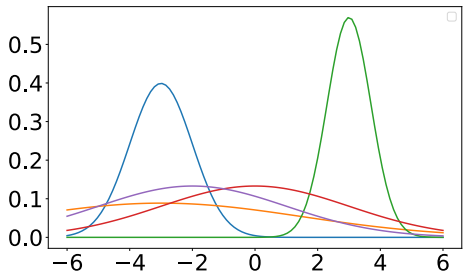
For the MCMC approach there is no model assumption for the posterior distribution needed. It therefore has a low bias but a high variance. The problem with this method is that it is not a-priori known how many iteration are needed to approximate the posterior well (the burn-in B is unknown). Moreover, since the variance of such sampling methods is high, it takes a lot of time to get close to the burn-in B. Therefore, MCMC is not suitable for deep Bayesian convolutional neural networks. [56]

2.3.2 Variational Inference

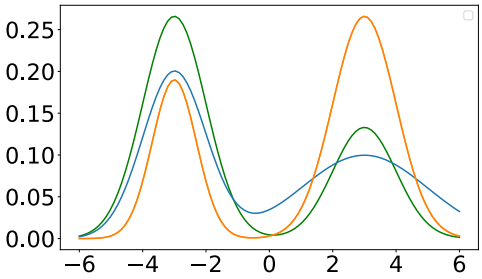
The solution of a Variational Inference method is the best approximation of the posterior distribution from a parametrized family. Thus, $q^* := \min_{\theta \in \Theta} D(q_\theta, p)$, where Θ is a parameter family, p is the posterior distribution and D is a divergence measure between two distributions. The aim of Variation Inference is to replace the marginalization of the evidence (see Theorem 2.7) with an optimization problem. Hence, the task is shifted away from an integration problem towards a derivation problem.

Parameterization Family A parameterization family should not be too small nor too big. Two parameterization families are shown in Figure 8. The smaller the family, the simpler is the optimization process but the higher the bias (see in Figure 9 the difference between the orange line and one of the blue lines). If the family is bigger it may yield to

an intractable optimization problem and to overfitting to the training data. Thus, smaller distributions with less hyperparameters are prefered, if the bias is low enough.



(a) Family 1: Different Gaussian distributions



(b) Family 2: Different mixtures of two Gaussian distributions

Figure 8: Two parameterization families. The first one has two parameters to optimize: μ, σ. The second family has four parameters to optimize: μ_1, μ_2 and σ_1, σ_2.

Divergence Having found a suitable parametrization family, the task is to find the best approximation of the posterior distribution up to its normalization factor among this family. Thus, an error measurement needs to be found which is insensitive to a normalization factor.

Definition 2.26. Let $(\Omega, \mathcal{S}, p)$ and $(\Omega, \mathcal{S}, q)$ be two probability space. Then, the **Kullback-Leibler (KL)** divergence is defined as

$$\mathbb{KL}(p||q) = \int_\Omega p(\omega) log\left(\frac{p(\omega)}{q(\omega)}\right) \mathrm{d}\omega \tag{2.29}$$

Theorem 2.27. *Let $((\Omega, \mathcal{S}, (q_\theta)_{\theta\in\Theta})$ be a statistical model and $(\Omega, \mathcal{S}, p)$ be a probability space. Then, minimizing the KL-divergence w.r.t. $\theta \in \Theta$ is invariant to a factor of p, i.e.*

$$\arg\min_{\theta in \Theta} \mathbb{KL}(q_\theta||Cp) = \arg\min_{\theta\in\Theta} \mathbb{KL}(q_\theta||p) \tag{2.30}$$

Proof. Let $C \in \mathbb{R}$.

$$\begin{aligned}
\mathbb{KL}(q_\theta||Cp) &= \int_\Omega q_\theta(\omega) log\left(\frac{q_\theta(\omega)}{Cp(\omega)}\right) \mathrm{d}\omega \\
&= \int_\Omega q_\theta(\omega) log\left(\frac{q_\theta(\omega)}{p(\omega)}\right) - log(C)\mathrm{d}\omega \\
&= \int_\Omega q_\theta(\omega) log\left(\frac{q_\theta(\omega)}{p(\omega)}\right) \mathrm{d}\omega - log(C) \\
&= \mathbb{KL}(q_\theta||p) - log(C)
\end{aligned}$$

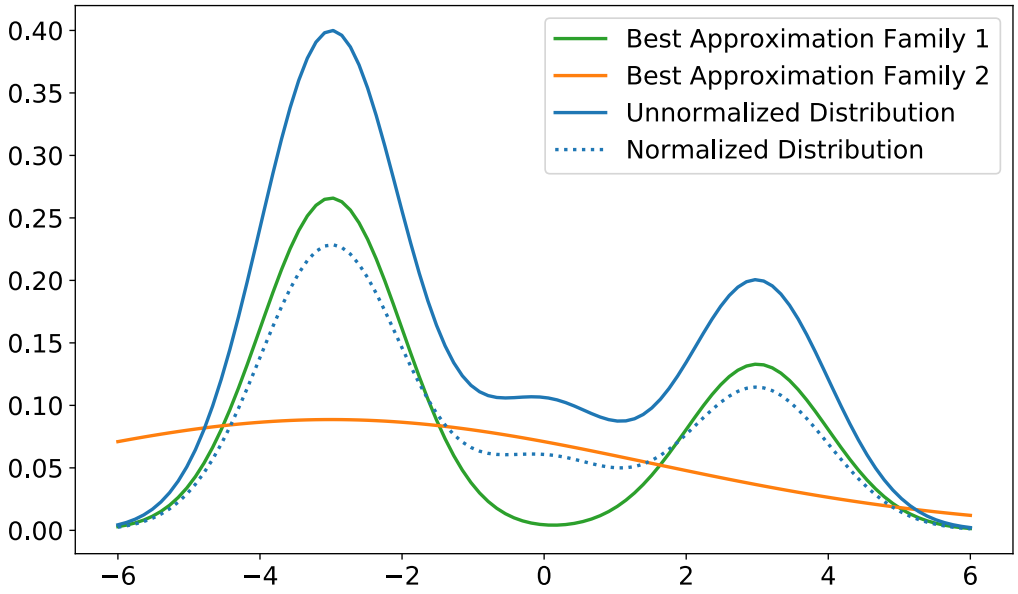


Figure 9: The non-normalized posterior distribution as an non-normalized mixture of three Gaussian distributions and the best approximations of both parameterization families. While the best approximation of the family of Gaussians has a high bias, the closest distribution of the family of mixtures of two Gaussians approximates the normalized posterior distribution well.

$$\begin{aligned} \implies \arg\min_{\theta in \Theta} \mathbb{KL}(q_\theta || Cp) &= \arg\min_{\theta \in \Theta} \mathbb{KL}(q_\theta || p) - log(C) \\ &= \arg\min_{\theta \in \Theta} \mathbb{KL}(q_\theta || p) \end{aligned}$$

□

Another divergence measure, which does not consider the model's evidence is the Evidence lower bound:

Definition 2.28. Let $(\mathcal{X}, \mathcal{F}, (p_\omega)_{\omega \in \Omega})$, $((\Omega, \mathcal{S}, (q_\theta)_{\theta \in \Theta})$ be statistical models, $(\Omega, \mathcal{S}, p)$ be a probability space and let $\mathcal{D} \in \mathcal{F}$, $\omega \in \Omega$, $\theta \in \Theta$ with Θ being a parametrization family. Then the **Evidence Lower Bound (ELBO)** of the posterior $p(\omega \mid \mathcal{D})$ and its approximation q_θ is defined as

$$L_{EL}(q_\theta(\omega), p(\omega \mid \mathcal{D})) := \mathbb{E}_{q_\theta(\omega)}[log(p(\mathcal{D}, \omega))] - \mathbb{H}_{q_\theta}(\omega) \tag{2.31}$$

Theorem 2.29. *Let $(\mathcal{X}, \mathcal{F}, (p_\omega)_{\omega \in \Omega})$, $(\Omega, \mathcal{S}, (q_\theta)_{\theta \in \Theta})$ be statistical models, $(\Omega, \mathcal{S}, p)$ be a probability space and let $\mathcal{D} \in \mathcal{F}, \omega \in \Omega$. Then, minimizing the KL divergence of the posterior $p(\omega|\mathcal{D})$ and its approximation $q_\theta(\omega)$ is equivalent to the maximization of the corresponding ELBO.*

Proof.

$$\begin{aligned}
\arg\min_{\theta\in\Theta} \mathbb{KL}(q_\theta(\omega)|p(\omega|\mathcal{D})) &\overset{2.27}{=} \arg\min_{\theta\in\Theta} \mathbb{KL}(q_\theta(\omega)|p(\omega,\mathcal{D})) \\
&= -\arg\max_{\theta\in\Theta} \mathbb{KL}(q_\theta(\omega)|p(\omega,\mathcal{D})) \\
&= \arg\max_{\theta\in\Theta} - \mathbb{KL}(q_\theta(\omega)|p(\omega,\mathcal{D})) \\
&= \arg\max_{\theta\in\Theta} - \int_\Omega q_\theta(\omega) log\left(\frac{q_\theta(\omega)}{p(\omega,\mathcal{D})}\right) \mathrm{d}\omega \\
&= \arg\max_{\theta\in\Theta} - \int_\Omega q_\theta(\omega) log(q_\theta(\omega)) - log(p(\mathcal{D},\omega))\mathrm{d}\omega \\
&= \arg\max_{\theta\in\Theta} \int_\Omega q_\theta(\omega) log(p(\mathcal{D},\omega))\mathrm{d}\omega - \int_\Omega q_\theta(\omega) log(q_\theta(\omega))\mathrm{d}\omega \\
&= \arg\max_{\theta\in\Theta} \mathbb{E}_{q_\theta(\omega)}[log(p(\mathcal{D},\omega))] - \mathbb{H}_{q_\theta}(\omega) = L_{VI}(q_\theta(\omega), p(\omega \mid \mathcal{D}))
\end{aligned}$$

□

Optimizing the ELBO yields therefore a variational distribution with a high probability mass on regions where the non-normalized posterior distribution is high (first term). The second term is the entropy of the variational distribution $q(\omega)$ and is by the maximization encouraged to get as high as possible [23]. This leads a more general distribution and the second term can therefore be seen as a regularization term as well.

Theorem 2.30. *The ELBO can also be expressed in terms of the prior distribution and the likelihood function:*

$$L_{VI}(q_\theta(\omega), p(\omega \mid \mathcal{D})) = -\mathbb{E}_{q_\theta(\omega)}[log(p(\mathcal{D} \mid \omega))] + \mathbb{KL}(q_\theta(\omega)||p(\omega))) \tag{2.32}$$

Proof.

$$\begin{aligned}
&\arg\max_{\theta\in\Theta} \mathbb{E}_{q_\theta(\omega)}[log(p(\mathcal{D},\omega))] - \mathbb{H}_{q_\theta}(\omega) \\
&= \arg\max_{\theta\in\Theta} \int_\Omega q_\theta(\omega) log(p(\mathcal{D},\omega))\mathrm{d}\omega - \int_\Omega q_\theta(\omega) log(q_\theta(\omega))\mathrm{d}\omega \\
&= \arg\max_{\theta\in\Theta} \int_\Omega q_\theta(\omega) log(p(\mathcal{D} \mid \omega)p(\omega))\mathrm{d}\omega - \int_\Omega q_\theta(\omega) log(q_\theta(\omega))\mathrm{d}\omega \\
&= \arg\max_{\theta\in\Theta} \int_\Omega q_\theta(\omega) log(p(\mathcal{D} \mid \omega))\mathrm{d}\omega - \int_\Omega q_\theta(\omega) log\left(\frac{q_\theta(\omega)}{p(\omega)}\right) \mathrm{d}\omega \\
&= \arg\min_{\theta\in\Theta} - \int_\Omega q_\theta(\omega) log(p(\mathcal{D} \mid \omega))\mathrm{d}\omega + \int_\Omega q_\theta(\omega) log\left(\frac{q_\theta(\omega)}{p(\omega)}\right) \mathrm{d}\omega \\
&= \arg\min_{\theta\in\Theta} - \mathbb{E}_{q_\theta(\omega)}[log(p(\mathcal{D} \mid \omega))] + \mathbb{KL}(q_\theta(\omega)||p(\omega)))
\end{aligned}$$

□

Inserting the data X and its labels Y from the set of labeled data $\mathcal{L}$ instead of the placeholder $\mathcal{D}$ yields an ELBO equality of the form:

$$L(q_\theta(\omega), p(\omega \mid \mathcal{L})) = -\mathbb{E}_{q_\theta(\omega)}[log(p(Y \mid X, \omega))] + \mathbb{KL}(q_\theta(\omega)||p(\omega))) \tag{2.33}$$

Inserting the data points of $(x_i, y_i) \in (X, Y)$ yields:

$$L_{VI}(q_\theta(\omega), p(\omega \mid \mathcal{L})) = -\sum_{i=1}^{N} \mathbb{E}_{q_\theta(\omega)}[log(p(y_i \mid x_i, \omega))] + \mathbb{KL}(q_\theta(\omega)||p(\omega))) \tag{2.34}$$

Using only a mini batch the term can be approximated by:

$$\hat{L}_{VI}(q_\theta(\omega), p(\omega \mid \mathcal{L})) = -\frac{N}{M}\sum_{i=1}^{M} \mathbb{E}_{q_\theta(\omega)}[log(p(y_i \mid x_i, \omega))] + \mathbb{KL}(q_\theta(\omega)||p(\omega))) \tag{2.35}$$

A method optimizing the ELBO w.r.t. the parameters ω is Bayes by Backprop (see Section D.1 in the Appendix). It considers a Gaussian parameterization family for the parameters. This method has not been tested on PreciBake's data set yet, but this will be done in the future. In the following subsections, methods are presented, which use Monte Carlo Variational Inference.

Monte Carlo Variational Inference Monte Carlo Variational Inference samples from the parameterized distribution of the network's parameters $q_\theta(\omega)$. Thus, $\hat{\omega}_i$ is sampled T times from $q_\theta(\omega)$ and yields the Monte Carlo estimator:

$$\hat{L}_{MC}(q_\theta(\omega), p(\omega \mid \mathcal{D})) = -\frac{1}{T}\frac{N}{M}\sum_{t=1}^{T}\sum_{i=1}^{M} log(p(y_i|x_i, \hat{\omega}_t) + \mathbb{KL}(q_\theta(\omega)||p(\omega))) \tag{2.36}$$

q_θ can therefore be approximated by its unbiased estimator $\bar{q}_\theta$:

$$\bar{q}_\theta(y_i|x_i) = \sum_{t=1}^{T} p(y_i|x_i, \hat{\omega}_t) \tag{2.37}$$

This converges for $T \to \infty$ to the estimator of the model's posterior distribution:

$$\bar{q}_\theta(y_i|x_i) = \sum_{t=1}^{T} p(y_i|x_i, \hat{\omega}_t) \overset{T\to\infty}{\to} \int p(y_i|x_i, \omega) q_\theta(\omega) \mathrm{d}\omega. \tag{2.38}$$

The advantage of Monte Carlo Variational Inference is, that architectures of convolutional neural networks can be used. For Monte Carlo Variational Inference methods, stochastic noise (e.g. Dropout or Batch Normalization) may be introduced, which makes it computationally cheap to sample from the distribution. But the distribution is not Gaussian. Another technique is to initialize an ensemble of T differently initialized models, which are initialized according to Gaussian prior. Thus, their posterior is a Gaussian as well, but the optimization takes more time.

2.3.3 Monte Carlo Dropout and Batch Normalization

Approximation with Dropout As it is computationally expensive to parameterize every parameter of an artificial neural network, one needs to find a way to approximate a Bayesian neural network without dealing with two many computations. One way to approximate a Bayesian neural network is to use dropout during the inference process. The intuition of dropout is to set an artificial neuron or a kernel of the previous layer with probability $p \in (0, 1)$ to zero. This can be used to model a prior distribution of the artificial neuron, kernels and even of its parameters. One therefore gets a Bernoulli prior distribution.

Let $\epsilon_i \sim Bernoulli(p), p \in (0, 1)$, let $\hat{\epsilon_i}$ being a realization of ϵ_i and let $\epsilon = (\epsilon_1, \epsilon_2, ..., \epsilon_k)^T$, $\hat{\epsilon_i}$ analogously.

Applying dropout to a fully connected neural networks yields, with $Z_k = W_k A_{k-1} + B_k$ and W_k, B_k being the parameters of the layer k (W weights, B biases) and A_{k-1} being the output of the previous layer:

$$
\begin{aligned}
\hat{y} &= \sigma_{out}(Z_k) = \sigma_{out}(W_k \hat{A}_{k-1} + B_k) = \sigma_{out}(W_k(A_{k-1} \odot diag(\hat{\epsilon})) + B_k) \\
&= \sigma_{out}((W_k \odot diag(\hat{\epsilon}))A_{k-1} + B_k) = \sigma_{out}(\hat{W}_k A_{k-1} + B_k) = \sigma_{out}(\hat{W}_k \sigma(Z_{k-1}) + B_k) \\
&= \sigma_{out}(\hat{W}_k \sigma(W_{k-1} \hat{A}_{k-2} + B_{k-1}) + B_k) = ... \\
&= \sigma_{out}(\hat{W}_k \sigma(\hat{W}_{k-1} \sigma(\hat{W}_{k-2} \sigma(...) + B_{k-2}) + B_{k-1}) + B_k
\end{aligned}
$$

with $\hat{A}_i = A_i \odot diag(\hat{\epsilon}), \hat{W}_i = W_i \odot diag(\hat{\epsilon})$. Thus, the FCNN with dropout layers can also be written as a Bayesian neural network with a Bernoulli prior distribution on the network's parameters.

To illustrate how convolutional neural networks work with dropout, the input A_{k-1} of each layer needs to be transformed to a vector of patches as described in Section 2.2.1. Let $\bar{A}$ be such a transformation of A. For a convolutional layer applying dropout means that a kernel is dropped with a probability of $p \in (0, 1)$, i.e. set to zero. Or in Bayesian terms: a prior is placed over each kernel. Let $\epsilon_{i,j,l} \sim Bernoulli(p), p \in (0, 1)$, and let $\hat{\epsilon}_{i,j,l}$ being a realization of $\epsilon_{i,j,l}$ where i is the layer, j the kernel and l the patch. $\epsilon_{i,j,l}$ denotes the probability of inputs vaules of patch l being dropped for kernel j in layer i. Let $E_{i,l} = diag([\epsilon_{i,j,l}]_{j=1}^{K-i})$ and $D_i = diag([E_{i,l}]_{l=1}^{n})$. Then, dropout applied to a convolutional layer can be seen as:

$$
\bar{A}_i := \begin{pmatrix} A_i^1 \\ A_i^2 \\ ... \\ A_i^n \end{pmatrix}, \text{ with } A_i^j \in \mathbb{R}^{hwK_{i-1}} \text{ being the } j\text{th patch of } A_i.
$$

$$\bar{\hat{A}}_i W_i + b_i = \begin{pmatrix} \hat{A}_i^1 \\ \hat{A}_i^2 \\ \dots \\ \hat{A}_i^n \end{pmatrix} W_i + b_i = \begin{pmatrix} A_i^1 diag([\epsilon_{i,j,l}]_{j=1}^{K-i}) \\ A_i^2 diag([\epsilon_{i,j,l}]_{j=1}^{K-i}) \\ \dots \\ A_i^n diag([\epsilon_{i,j,l}]_{j=1}^{K-i}) \end{pmatrix} W_i + b_i = \begin{pmatrix} A_i^1 E_{i,1} \\ A_i^2 E_{i,2} \\ \dots \\ A_i^n E_{i,n} \end{pmatrix} W_i$$
$$= \begin{pmatrix} A_i^1 \\ A_i^2 \\ \dots \\ A_i^n \end{pmatrix} D_i W_i + b_i = \bar{A}_i D_i W_i + b_i = \bar{A}_i \hat{W}_i + b_i$$

with $\hat{W}_i := diag([E_{i,l} W_i]_{l=1}^n)$
Thus, applying dropout after a convolutional layer is equivalent to parameterizing the weights of the convolutional layer.

Having noise injected, which is Bernoulli distributed, the weights will also get Bernoulli distributed. This does not approximate a Gaussian distribution well.

One gets:

$$p(y^* \mid x^*, \mathrm{X}, \mathrm{Y}) \approx \int_{\{0,1\}^d} \mathrm{Cat}(y^* \mid f_{\omega\epsilon}(x^*)) \mathcal{B}(\epsilon \mid p) \mathrm{d}\epsilon \tag{2.39}$$

$$= \int_{\{0,1\}^d} \prod_{c=1}^{C} f_{\omega\epsilon}(x^*)^{y_c^*} p(\epsilon) \mathrm{d}\epsilon \text{ with } \begin{cases} p(\epsilon_t^i = 0) = p \\ p(\epsilon_t^i = 1) = 1 - p \end{cases} \tag{2.40}$$

$$\overset{2.50}{\approx} \sum_{t=1}^{T} \prod_{c=1}^{C} f_{\hat{\omega}_t}(x^*)^{y_c^*} \text{ with } \hat{\omega}_t^i = \omega_t^i \hat{\epsilon}_t^i \text{ and } \hat{\epsilon}_t \sim p(\epsilon \mid p) \tag{2.41}$$

$$\text{with } \begin{cases} p(\epsilon_t^i = 0) = p \\ p(\epsilon_t^i = 1) = 1 - p \end{cases}$$

Equation 2.40 can neither be solved with a closed form solution and thus, the parameters ϵ need to be sampled during inference time. This leads to Equation 2.41. During training time these noise injections were not used to approximate a distribution but to regularize the network.

Problems with this approach In practice, dropout is not applied to every layer. Thus, not every layer has weights with an underlying distribution. Also a Bernoulli distribution does not approximate a Gaussian distribution well. Thus, the results may not be the best approximation of the posterior distribution.

Approximation with Batch Normalization As stated in [64], the same approach is possible using Batch Normalization as noise injection. Let $\omega \in \mathbb{R}^d$ be all parameters of the network and let $W_k = \omega_l, ..., \omega_{l+o_w}$ be the parameters of the weight or filter matrix of the k^{th} layer, $B_k = \omega_{l+o_w+1}, ..., \omega_{l+o_w+o_b}$ be the parameters of the bias of the k^{th} layer. Then, it holds for a network only consisting of fully connected layers:

$$\begin{aligned}
\hat{y} &= \sigma_{out}(\hat{Z}_k) = \sigma_{out}(\mathcal{BN}_{\gamma,\beta}(Z_k)) = \sigma_{out}\left(\gamma\frac{Z_k - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} + \beta\right) = \sigma_{out}\left(\gamma\frac{W_k A_{k-1} + B_k - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} + \beta\right) \\
&= \sigma_{out}\left(\gamma\frac{W_k A_{k-1} + (B_k - \mu_\mathcal{B})}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} + \beta\right) = \sigma_{out}\left(\frac{\gamma W_k A_{k-1}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} + \frac{\gamma(B_k - \mu_\mathcal{B})}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} + \beta\right) \\
&= \sigma_{out}\left(\frac{\gamma W_k}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} A_{k-1} + \frac{\gamma(B_k - \mu_\mathcal{B})}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} + \beta\right) = \sigma_{out}(\hat{W}_k A_{k-1} + \hat{B}_k) = \sigma_{out}(\hat{W}_k \sigma(\hat{Z}_{k-1}) + \sigma B_k) \\
&= ... = \sigma_{out}(\hat{W}_k \sigma(\hat{W}_{k-1}\sigma(\hat{W}_{k-2}\sigma(...) + \hat{B}_{k-2}) + \hat{B}_{k-1}) + \hat{B}_k)
\end{aligned}$$

And for a convolutional layer it holds:

$$\begin{aligned}
\hat{y} &= \sigma(\hat{Z}_i) = \sigma(\mathcal{BN}_{\gamma,\beta}(Z_i) = \sigma\left(\gamma(\frac{Z_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}}) + \beta\right) = \sigma\left(\gamma(\frac{\bar{A}_i W_i + B_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}}) + \beta\right) \\
&= \sigma\left(\frac{\gamma \bar{A}_i W_i}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} + \frac{\gamma(B_i - \mu_\mathcal{B})}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} + \beta\right) = \sigma\left(\bar{A}^i \frac{\gamma W_i}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} + \frac{\gamma(B_i - \mu_\mathcal{B})}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} + \beta\right) \\
&= \sigma_{out}(\bar{A}^i \hat{W}_i + \hat{B}_i)
\end{aligned}$$

As described in the previous equations the noise, which is injected by Batch Normalization to the input of the non-linearity, can be considered to belong to the parameters. Thus, the function $\mathcal{BN}(\omega)$ is justified. It holds:

$$p(y^* \mid x^*, \mathrm{X}, \mathrm{Y}) \approx \int_\Theta \mathrm{Cat}(y^* \mid f_{\mathcal{BN}^\theta_{\beta,\gamma}(\omega)}(x^*)) p(\theta \mid X, Y) \mathrm{d}\theta \tag{2.42}$$

$$\overset{2.50}{\approx} \sum_{t=1}^{T} \prod_{c=1}^{C} f_{\hat{\omega}_t}(x^*)^{y_c^*} \text{ with } \hat{\omega}_t = \mathcal{BN}^{\hat{\theta}_t}_{\beta,\gamma}(\omega) \text{ and } \hat{\theta}_t \sim p(\theta \mid X, Y) \tag{2.43}$$

Problems with this approach In practice, Batch Normalization is not applied to every layer. Thus, not every layer has weights with an underlying prior distribution. Moreover, since θ depends on the data, whose distribution is not known, the distribution of $\hat{\omega}$ is unknown, i.e. the prior distribution of the parameters is unknown. This may result in a good or a bad approximation.

2.3.4 Ensemble Methods

Ensemble methods are in Machine Learning used because it may be better to not only rely on one model's prediction but on a myriad of model's predictions since one does

not know if the prediction of one model is the most accurate. Having the same architecture of all models and training the models on the same data set should lead to slightly different models, if they are initialized differently. Training models with all possible initializations (according to a prior distribution) would therefore lead to every deterministic representation of a Bayesian neural network. Thus, using $T \in \mathbb{N}$ ensembles with different initializations approximates a Bayesian neural network. Let $\{\mathcal{M}_1 ..., \mathcal{M}_T\}$ be the ensemble of T models.

$$p(y^* \mid x^*, \mathrm{X}, \mathrm{Y}) = \int_\Omega \mathrm{Cat}(y^* \mid f_\omega(x^*)) p(\omega \mid X, Y) \mathrm{d}\omega \tag{2.44}$$

$$\overset{2.50}{\approx} \sum_{m=1}^{T} \prod_{c=1}^{C} f_{\hat{\omega}_m}(x^*)^{y_c^*} \text{ with } \hat{\omega}_m,\ hat\omega_m \sim p(\omega \mid X, Y), \tag{2.45}$$

Ensemble by Cyclic Learning Instead of training different models, one model is in this method trained in the following way: The model is trained in a normal manner and after finishing the training process the learning rate is reset to its starting value and the model is trained again. This process is done T times and each trained model is saved in order to get an ensemble of T different models. This method is inspired by [27].

Deep Probabilistic Ensembles As stated in [10] ensembles are easy to optimize and fast to execute. But the model's uncertainty is not approximated in the same matter as it is by using Bayesian neural networks. This is due to complexity of deep neural networks and the fact that a parameter may serve a different purpose for different members of the ensemble. Thus, the variance of these parameters cannot be compared to the variance, which would have been obtained in a Bayesian neural network with the same architecture. The KL divergence term of Equation 2.32 needs to be applied as a regularization penalty to the set of values that a given parameter takes over all the members of the ensemble. By assuming that the parameters of the model are mutually independent (which is in general not true) and that they are Gaussian, the KL divergence term of the Equation 2.32 can be computed analytically by:

$$\mathbb{KL}(p||q) = \frac{1}{2}\left(log\left(\frac{\sigma_q^2}{\sigma_p^2}\right) + \frac{\sigma_p^2 + (\mu_p - \mu_q)^2)}{\sigma_q^2} - 1\right) \tag{2.46}$$

This leads to the following optimization problem:

$$\omega^* = \arg\min_\omega \sum_{i=1}^{N} \sum_{m=1}^{T} L(y_i, f_{\hat{\omega}_m}(x_i)) + \beta R(\omega), \tag{2.47}$$

with $\{(x_i, y_i)\}$ labeled data points, $\{\mathcal{M}_1 ..., \mathcal{M}_T\}$ ensemble of E models, ω_m parameters of model $\mathcal{M}_m$ represented by f_ω and ω being the parameters of all models. R describes a regularization term, i.e. Equation 2.46 and L is the first term of Equation 2.32.

2.4 Uncertainty of Bayesian Neural Networks

Once an approximation method of a Bayesian neural network is found, it can be used in order to determine the network's uncertainty.

For active learning a specific type of uncertainty can be used in order to find the instances, on which the model learns the most (as it is not yet certain about them). This type is called epistemic uncertainty and will be introduced in this chapter. Moreover, two uncertainty decomposition techniques are presented in this chapter in order to find the model's epistemic uncertainty of a given input sample.

There exist two types of uncertainty, namely aleatoric and epistemic uncertainty. Aleatoric uncertainty is the uncertainty of the data (measurement errors, etc.) and epistemic uncertainty is the uncertainty of the model, more precisely of its parameters [31].

Aleatoric Uncertainty The English word aleatoric comes from the Latin word āleae which means rolling a die, gambling or game of chance. Therefore, by aleatoric uncertainty the natural randomness of an event is meant. There are two types of aleatoric uncertainty: heteroscedastic and homoscedastic uncertainty. The first one describes input-dependent uncertainties while the latter one describes input-independent uncertainties.

Epistemic Uncertainty The English word epistemic comes from the Greek word epistēmē which means knowledge. In the context of artificial neural networks, it therefore tells how much a model knows. The knowledge of a network can be improved with more data and thus, the epistemic uncertainty decreases as the model learns from more data. Moreover, a network can gain more knowledge from some data than from others. To model the epistemic uncertainty, a distribution needs to be placed over every weight of the model. In the Bayesian approach this is done by placing a prior distribution over each weight of the network and then update the distribution by determining the posterior distribution using Bayes' theorem (see Theorem 2.6), which was already shown in Section 2.3.

Let $q(\omega|X,Y)$ be an approximation of the underlying weight distribution $p(w \mid X,Y)$. By marginalization if follows:

$$p(y^* \mid x^*, \mathrm{X}, \mathrm{Y}) = \int_\Omega p(y^* \mid x^*, \omega) p(\omega \mid X, Y) \mathrm{d}\omega \tag{2.48}$$

$$\approx \int_\Omega p(y^* \mid x^*, \omega) q(\omega \mid X, Y) \mathrm{d}\omega =: q(y^* \mid x^*, X, Y) \tag{2.49}$$

$$\approx \sum_{t=1}^{T} p(y^* \mid x^*, \hat{\omega}_t) =: \hat{q}(y^* \mid x^*, X, Y) \tag{2.50}$$

with $\hat{\omega}_t \sim q(\omega \mid X,Y)$. With $\hat{q}(y^* \mid x^*, X, Y)$ being an unbiased estimator of $q(y^* \mid x^*, X, Y)$:

$$q(y^* \mid x^*, X, Y) = \mathbb{E}_{q(\omega)}[p(y^* \mid x^*, \omega)].$$

Having a measurement of uncertainty, these definitions help in order to distinguish the uncertainty into epistemic and aleatoric uncertainty.

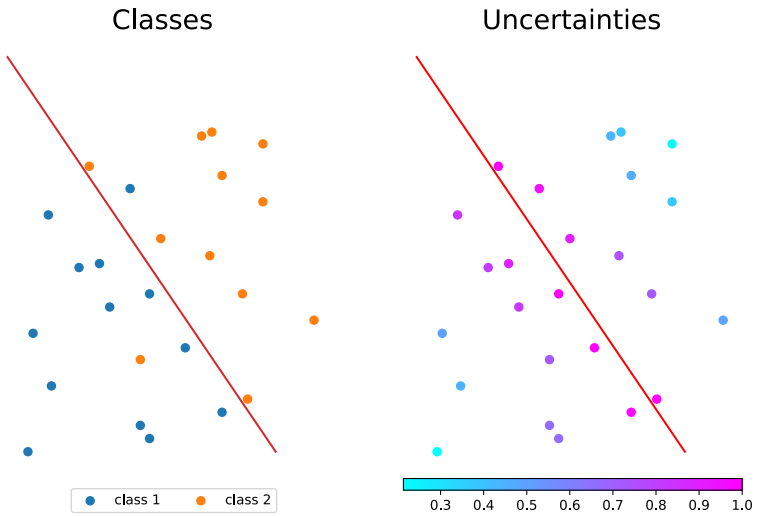


Figure 10: Visualization of the model's **epistemic uncertainty** on given samples. The epistemic uncertainty is given by the uncertainty of the model's parameters, represented by the red decision boundary. Being close to the model's decision boundary means that the model is uncertain about a specific instance because a small change in the parameters would lead to a slightly different decision boundary, which may yield different predictions for samples close to the decision boundary.

For simplicity following notations will be used in the following theorem:

Notation 2.31. Let $(\mathcal{X}, \mathcal{F}, p)$, $(\mathcal{X}, \mathcal{F}, q)$ and $(\Omega, \mathcal{S}, q)$ be probability spaces, $(y^*, x^*) \in \mathcal{X}, (X, Y) \in \mathcal{F}$ and $\omega \in \Omega$. Then it is written:

$$\begin{aligned} q(\omega) &:= q(\omega \mid X, Y) \\ q(y^* \mid x^*) &:= q(y^* \mid x^*, X, Y) \\ p(y^* \mid x^*) &:= p(y^* \mid x^*, X, Y) \end{aligned}$$

2.4.1 Uncertainty Decomposition using Predictive Variance

The task is to determine the epistemic uncertainty of the prediction of y^*, while x^* is given. Using the variance of the prediction of y^*, given x^* this leads to the following decomposition:

Theorem 2.32 (Predictive Variance Decomposition Theorem)**.** *Let $(\mathcal{X}, \mathcal{F}, (p_\omega)_{\omega \in \Omega})$ be a statistical model, $(\mathcal{X}, \mathcal{F}, q)$ and $(\Omega, \mathcal{S}, q)$ be probability spaces, $(y^*, x^*) \in \mathcal{X}, (X, Y) \in \mathcal{F}$ and $\omega \in \Omega$. Then, the predictive variance of the Bayesian neural network approximation*

q on y^* can be decomposed into its aleatoric and epistemic uncertainty on y^*. [39]

$$\begin{aligned}\mathrm{Var}_{q(y^*|x^*)}[y^*] &= \mathbb{E}_{q(y^*|x^*)}[y^* y^{*T}] - \mathbb{E}_{q(y^*|x^*)}[y^*]\mathbb{E}_{q(y^*|x^*)}[y^*]^T \qquad (2.51)\\ &= \underbrace{\int_\Omega \left[\mathrm{diag}(\mathbb{E}_{p(y^*|x^*,\omega)}[y^*]) - \mathbb{E}_{p(y^*|x^*,\omega)}[y^*]\mathbb{E}_{p(y^*|x^*,\omega)}[y^*]^T\right] q(\omega)\ \mathrm{d}\omega}_{aleatoric}\\ &+ \underbrace{\int_\Omega \left[\mathbb{E}_{p(y^*|x^*,\omega)}[y^*] - \mathbb{E}_{q(y^*|x^*)}[y^*]\right]\left[\mathbb{E}_{p(y^*|x^*,\omega)}[y^*] - \mathbb{E}_{q(y^*|x^*)}[y^*]\right]^T q(\omega)\ \mathrm{d}\omega}_{epistemic},\end{aligned} \qquad (2.52)$$

where $diag(\nu)$ is a diagonal matrix with the entries of the vector ν being on its diagonal.

Proof.

$$\begin{aligned}\mathbb{E}_{q(y^*|x^*)}[y^* y^{*T}] &= \int y^* y^{*T} q(y^* \mid x^*)\mathrm{d}y^*\\ &= \int y^* y^{*T} \int_\Omega p(y^* \mid x^*, \omega) q(\omega)\mathrm{d}\omega\mathrm{d}y^*\\ &= \int \int_\Omega y^* y^{*T} p(y^* \mid x^*, \omega) q(\omega)\mathrm{d}\omega\mathrm{d}y^*\\ &\overset{Fubini}{=} \int y^* y^{*T} \int_\Omega p(y^* \mid x^*, \omega) q(\omega)\mathrm{d}\omega\mathrm{d}y^*\\ &= \int_\omega \int y^* y^{*T} p(y^* \mid x^*, \omega)\mathrm{d}y^* q(\omega)\mathrm{d}\omega\\ &= \int_\Omega \mathbb{E}_{p(y^*|x^*,\omega)}[y^* y^{*T}] q(\omega)\mathrm{d}\omega\\ &\overset{y^* one-hot-coded}{=} \int_\Omega \mathbb{E}_{p(y^*|x^*,\omega)}[diag(y^*)] q(\omega)\mathrm{d}\omega\\ &= \int_\Omega diag(\mathbb{E}_{p(y^*|x^*,\omega)}[y^*]) q(\omega)\mathrm{d}\omega\\ &\overset{+0}{=} \int_\Omega diag(\mathbb{E}_{p(y^*|x^*,\omega)}[y^*]) - \mathbb{E}_{p(y^*|x^*,\omega)}[y^*]\mathbb{E}_{p(y^*|x^*,\omega)}[y^*]^T\\ &+ \mathbb{E}_{p(y^*|x^*,\omega)}[y^*]\mathbb{E}_{p(y^*|x^*,\omega)}[y^*]^T q(\omega)\mathrm{d}\omega\\ &= \int_\Omega diag(\mathbb{E}_{p(y^*|x^*,\omega)}[y^*]) - \mathbb{E}_{p(y^*|x^*,\omega)}[y^*]\mathbb{E}_{p(y^*|x^*,\omega)}[y^*]^T q(\omega)\mathrm{d}\omega\\ &+ \int_\Omega \mathbb{E}_{p(y^*|x^*,\omega)}[y^*]\mathbb{E}_{p(y^*|x^*,\omega)}[y^*]^T q(\omega)\mathrm{d}\omega\end{aligned} \qquad (2.53)$$

$$\begin{aligned}\mathbb{E}_{q(y^*|x^*)}[y^*] &= \int y^*\ q(y^* \mid x^*)\ \mathrm{d}y^* = \int y^* \int_\Omega p(y^* \mid x^*, \omega) q(\omega)\ \mathrm{d}\omega\ \mathrm{d}y^*\\ &= \int_\Omega \int y^*\ p(y^* \mid x^*, \omega)\ \mathrm{d}y^* q(\omega)\ \mathrm{d}\omega = \int_\Omega \mathbb{E}_{p(y^*|x^*,\omega)}[y^*]\ q(\omega)\ \mathrm{d}\omega.\end{aligned}$$

$$
\begin{aligned}
-\mathbb{E}_{q(y^*|x^*)}[y^*]\mathbb{E}_{q(y^*|x^*)}[y^*]^T &= \mathbb{E}_{q(y^*|x^*)}[y^*]\mathbb{E}_{q(y^*|x^*)}[y^*]^T - 2\mathbb{E}_{q(y^*|x^*)}[y^*]\mathbb{E}_{q(y^*|x^*)}[y^*]^T \\
&= \mathbb{E}_{q(y|x)}[y^*]\mathbb{E}_{q(y^*|x^*)}[y^*]^T - 2\mathbb{E}_{q(y^*|x^*)}[y^*]\mathbb{E}_{q(y^*|x^*)}[y^*]^T \\
&\overset{2.54}{=} \mathbb{E}_{q(y|x)}[y^*]\mathbb{E}_{q(y^*|x^*)}[y^*]^T \\
&\quad - 2\int_\Omega \mathbb{E}_{p(y^*|x^*,\omega)}[y^*]\ q(\omega)\mathrm{d}\omega\mathbb{E}_{q(y^*|x^*)}[y^*]^T \\
&= \mathbb{E}_{q(y|x)}[y^*]\mathbb{E}_{q(y^*|x^*)}[y^*]^T && (2.54) \\
&\quad - 2\int_\Omega \mathbb{E}_{p(y^*|x^*,\omega)}[y^*]\ q(\omega)\mathbb{E}_{q(y^*|x^*)}[y^*]^T\mathrm{d}\omega && (2.55) \\
& && (2.56)
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{Var}_{q(y^*|x^*)}[y^*] &= \mathbb{E}_{q(y^*|x^*)}[y^*{y^*}^T] - \mathbb{E}_{q(y^*|x^*)}[y^*]\mathbb{E}_{q(y^*|x^*)}[y^*]^T && (2.57) \\
&\overset{2.53}{=} \int_\Omega diag(\mathbb{E}_{p(y^*|x^*,\omega)}[y^*]) - \mathbb{E}_{p(y^*|x^*,\omega)}[y^*]\mathbb{E}_{p(y^*|x^*,\omega)}[y^*]^T q(\omega)\mathrm{d}\omega \\
&\quad + \int_\Omega \mathbb{E}_{p(y^*|x^*,\omega)}[y^*]\mathbb{E}_{p(y^*|x^*,\omega)}[y^*]^T q(\omega)\mathrm{d}\omega - \mathbb{E}_{q(y^*|x^*)}[y^*]\mathbb{E}_{q(y^*|x^*)}[y^*]^T && (2.58) \\
&\overset{2.55}{=} \int_\Omega diag(\mathbb{E}_{p(y^*|x^*,\omega)}[y^*]) - \mathbb{E}_{p(y^*|x^*,\omega)}[y^*]\mathbb{E}_{p(y^*|x^*,\omega)}[y^*]^T q(\omega)\mathrm{d}\omega \\
&\quad + \int_\Omega \mathbb{E}_{p(y^*|x^*,\omega)}[y^*]\mathbb{E}_{p(y^*|x^*,\omega)}[y^*]^T q(\omega)\mathrm{d}\omega - \mathbb{E}_{q(y|x)}[y^*]\mathbb{E}_{q(y^*|x^*)}[y^*]^T \\
&\quad - 2\int_\Omega \mathbb{E}_{p(y^*|x^*,\omega)}[y^*]\ q(\omega)\mathbb{E}_{q(y^*|x^*)}[y^*]^T\mathrm{d}\omega \\
&= \int_\Omega \left[\mathrm{diag}(\mathbb{E}_{p(y^*|x^*,\omega)}[y^*]) - \mathbb{E}_{p(y^*|x^*,\omega)}[y^*]\mathbb{E}_{p(y^*|x^*,\omega)}[y^*]^T\right] q(\omega)\ \mathrm{d}\omega \\
&\quad + \int_\Omega \left[\mathbb{E}_{p(y^*|x^*,\omega)}[y^*] - \mathbb{E}_{q(y^*|x^*)}[y^*]\right]\left[\mathbb{E}_{p(y^*|x^*,\omega)}[y^*] - \mathbb{E}_{q(y^*|x^*)}[y^*]\right]^T q(\omega)\ \mathrm{d}\omega
\end{aligned}
$$

□

Epistemic and Aleatoric Uncertainty Let $p(y^* \mid x^*, \omega) = \mathrm{Cat}(y^* \mid f_\omega(x^*))$ with $f_\omega(x)$ representing a model with input x. This gives the following for the expectation of y^* under $p(y^* \mid x^*, \omega)$:

$$
\begin{aligned}
\mathbb{E}_{p(y^*|x^*,\omega)}[y^*] &\overset{y^* \text{ categorial}}{=} \sum_{y^* \in \mathcal{C}} y^* p(y^* \mid x^*, \omega) \overset{y^* \text{ categorial}}{=} \sum_{y^* \in \mathcal{C}} y^* \mathrm{Cat}(y^* \mid f_\omega(x^*)) \\
&\overset{y^* \text{ one hot coded}}{=} \sum_{c'=1}^{C} \mathbf{e}_c \prod_{c=1}^{C} (f_\omega(x^*))_c^{y_c^*} = f_\omega(x^*) && (2.59)
\end{aligned}
$$

with $\mathbf{e}_i$ being the canonical vector with its i^{th} entry being 1.

$$\mathbb{E}_{q(y^*|x^*)}[y^*] \overset{y^* \text{ categorial}}{=} \sum_{y^* \in \mathcal{C}} y^* q(y^* \mid x^*, \omega) \overset{2.49}{=} \sum_{y^* \in \mathcal{C}} y^* \int_\Omega p(y^* \mid x^*, \omega) q(\omega) \mathrm{d}\omega$$

$$= \int_\Omega \sum_{y^* \in \mathcal{C}} y^* p(y^* \mid x^*, \omega) q(\omega) \mathrm{d}\omega = \int_\Omega \mathbb{E}_{p(y^*|x^*,\omega)}[y^*] q(\omega) \mathrm{d}\omega \overset{2.59}{=} \int_\Omega f_\omega(x^*) q(\omega) \mathrm{d}\omega \tag{2.60}$$

(2.61)

Thus, it applies for the aleatoric uncertainty:

$$\begin{aligned}
&\int_\Omega \left[\mathrm{diag}(\mathbb{E}_{p(y^*|x^*,\omega)}[y^*]) - \mathbb{E}_{p(y^*|x^*,\omega)}[y^*] \mathbb{E}_{p(y^*|x^*,\omega)}[y^*]^T\right] q(\omega) \mathrm{d}\omega \\
&\overset{2.59}{=} \int_\Omega \left[\mathrm{diag}(f_\omega(x^*)) - f_\omega(x^*) f_\omega(x^*)^T\right] q(\omega) \mathrm{d}\omega \\
&= \mathbb{E}_{q(\omega)} \left[\mathrm{diag}(f_\omega(x^*)) - f_\omega(x^*) f_\omega(x^*)^T\right] \qquad (2.62) \\
&\qquad (2.63)
\end{aligned}$$

Having Equation 2.62, it can be seen that the aleatoric uncertainty captures randomness in the observed data. The aleatoric uncertainty would be zero if $f_\omega(x^*)$ is a one-hot coded vector and it reaches its maximum if each value of $f_\omega(x^*)$ equals $\frac{1}{C}$. This coincides with the noise of an instance as the network would not be able to classify a noisy instance to a one-hot coded vector and if so, the noise would at least not affect the classifiers performance.

For the epistemic uncertainty it applies:

$$\begin{aligned}
&\int_\Omega \left[\mathbb{E}_{p(y^*|x^*,\omega)}[y^*] - \mathbb{E}_{q(y^*|x^*)}[y^*]\right] \left[\mathbb{E}_{p(y^*|x^*,\omega)}[y^*] - \mathbb{E}_{q(y^*|x^*)}[y^*]\right]^T q(\omega)\ \mathrm{d}\omega \\
&\overset{2.59}{=} \int_\Omega \left[f_\omega(x^*) - \mathbb{E}_{q(y^*|x^*)}[y^*]\right] \left[f_\omega(x^*) - \mathbb{E}_{q(y^*|x^*)}[y^*]\right]^T q(\omega)\ \mathrm{d}\omega \\
&\overset{2.60}{=} \int_\Omega \left[f_\omega(x^*) - \int_\Omega f_{\omega'}(x^*) q(\omega') \mathrm{d}\omega'\right] \left[f_\omega(x^*) - \int_\Omega f_{\omega'}(x^*) q(\omega') \mathrm{d}\omega'\right]^T q(\omega)\ \mathrm{d}\omega \\
&= \mathbb{E}_{q(\omega)} \left[\underbrace{\left(f_\omega(x^*) - \mathbb{E}_{q(\omega)}[f_{\omega'}(x^*)]\right) \left(f_\omega(x^*) - \mathbb{E}_{q(\omega)}[f_{\omega'}(x^*)]\right)^T}_{\text{variability of the model depending on } \omega}\right] \qquad (2.64)
\end{aligned}$$

Having Equation 2.64, it can be seen that the epistemic uncertainty measures the variability of the model's output depending on its parameters ω.

To summarize, the aleatoric uncertainty can be determined for each model instance separately whereas the epistemic uncertainty needs all model instances in order to determine the variability of the parameters.

Computation of Epistemic Uncertainty Now, q is approximated with $\hat{q}$ being the Monte Carlo estimator of q. This gives with $p_t := f_{\hat{\omega}_t}(x^*)$ and $\bar{p} := \sum_{t=1}^{T} f_{\hat{\omega}_t}(x^*)$ with $\hat{\omega}_t \sim q(\omega \mid X, Y)$:

$$\begin{aligned} 2.64 \approx \frac{1}{T}\sum_{t=1}^{T}(p_t - \bar{p})(p_t - \bar{p})^T &= \frac{1}{T}\sum_{t=1}^{T} p_t p_t^T - p_t \bar{p}^- \bar{p} p_t^T + \bar{p}\bar{p}^T = \frac{1}{T}\sum_{t=1}^{T} p_t p_t^T - 2 p_t \bar{p}^T + \bar{p}\bar{p}^T \\ = \bar{p}\bar{p}^T + \frac{1}{T}\sum_{t=1}^{T} p_t p_t^T - 2\frac{1}{T}\sum_{t=1}^{T} p_t \bar{p}^T &= \bar{p}\bar{p}^T + \frac{1}{T}\sum_{t=1}^{T} p_t p_t^T - 2\bar{p}\bar{p}^T = \sum_{t=1}^{T} p_t p_t^T - \bar{p}\bar{p}^T \end{aligned} \tag{2.65}$$

Thus, using Monte Carlo Varational Inference estimator, the epistemic uncertainty can be approximated by Equation 2.65.

2.4.2 Uncertainty Decompostion using Information Theory

Definition 2.33. Let $(\mathcal{X}, \mathcal{F}, p_X)$, $(\mathcal{Y}, \mathcal{G}, p_Y)$ be a probability spaces, $(\Sigma, \mathcal{S})$ a measurable space, $X : \mathcal{X} \to \Sigma$ and $Y : \mathcal{Y} \to \Sigma$ be random variables. Then, the **Shannon Entropy** of X is defined as

$$\mathbb{H}_p(X) = -\sum_{x \in \mathcal{X}} p_X(x) log(p_X(x)) \tag{2.66}$$

This definition can be generalized to continuous probability densities and is called **differential entropy**:

$$h_p(X) = -\int_{\mathcal{X}} log(p_X(x)) p_X(x) \mathrm{d}x \tag{2.67}$$

The **conditioned entropy** of X, given Y is defined as

$$\mathbb{E}_{p_Y}\left[\mathbb{H}_{p_X}(X|Y)\right] = -\sum_{y \in \mathcal{Y}} \sum_{x \in \mathcal{X}} log(p_X(x|y)) p_X(x|y) p_Y(y) \tag{2.68}$$

And the **conditioned differential entropy** of X, given Y is defined as

$$\mathbb{E}_{p_Y}[h_{p_X}(X|Y)] = -\int_{\mathcal{Y}} \int_{\mathcal{X}} log(p_X(x|y)) p_X(x|y) \mathrm{d}x \; p_Y(y) \mathrm{d}y \tag{2.69}$$

Let moreover $(\mathcal{X} \times \mathcal{Y}, \mathcal{F} \times \mathcal{G}, p_{(X,Y)})$ be the joint probability space of the random variables X and Y. Then their **mutual information** is defined as

$$\mathbb{I}(X;Y) := \mathbb{KL}(p_{(X,Y)} || p_X \otimes p_Y) \tag{2.70}$$

Theorem 2.34 (Entropy Decomposition Theorem)**.** *Let X, Y be random variables as defined in Definition 2.33. Then, the differential entropy of X can be decomposed into the conditioned differential entropy of X given Y and their mutual information, i.e.*

$$h_{p_X}(X) = \mathbb{E}_{p_Y}[h_{p_X}(X|Y)] + \mathbb{I}(X;Y). \tag{2.71}$$

Proof.

$$
\begin{aligned}
\mathbb{I}(X;Y) &= \mathbb{KL}\left(p_{(\mathcal{X},\mathcal{Y})} || p_{\mathcal{X}} \otimes p_{\mathcal{Y}}\right) \\
&= \int_{\mathcal{X}\times\mathcal{Y}} log\left(\frac{p(x,y)}{p(x)p(y)}\right) p(x,y)\mathrm{d}(x,y) \\
&= \int_{\mathcal{X}\times\mathcal{Y}} log\left(\frac{p(x \mid y)}{p(x)}\right) \frac{p(x,y)}{p(y)} p(y)\mathrm{d}(x,y) \\
&= \int_{\mathcal{X}\times\mathcal{Y}} log\left(\frac{p(x \mid y)}{p(x)}\right) p(x \mid y)p(y)\mathrm{d}(x,y) \\
&= \int_{\mathcal{X}\times\mathcal{Y}} \left(log(p(x \mid y)) - log(p(x))\right) p(x \mid y)p(y)\mathrm{d}(x,y) \\
&= \int_{\mathcal{X}\times\mathcal{Y}} (log(p(x \mid y))p(x \mid y)p(y)\mathrm{d}(x,y) \\
&- \int_{\mathcal{X}\times\mathcal{Y}} log(p(x)))p(x \mid y)p(y)\mathrm{d}(x,y) \\
&\overset{Fubini}{=} \int_{\mathcal{Y}}\int_{\mathcal{X}} (log(p(x \mid y))p(x \mid y)p(y)\mathrm{d}x\mathrm{d}y \\
&- \int_{\mathcal{X}}\int_{\mathcal{Y}} log(p(x)))p(x \mid y)p(y)\mathrm{d}y\mathrm{d}x \\
&= \int_{\mathcal{Y}}\int_{\mathcal{X}} (log(p(x \mid y))p(x \mid y)\mathrm{d}x p(y)\mathrm{d}y \\
&- \int_{\mathcal{X}} log(p(x)))p(x)\mathrm{d}x \\
&= h_{p_X}(X) - \mathbb{E}_{p_Y}[h_{p_X}(X|Y)]
\end{aligned} \tag{2.72}
$$

□

Epistemic and Aleatoric Uncertainty The task is to determine the uncertainty of the prediction of y^*, while x^* is given. Using Equation 2.48 and Equation 2.49, the uncertainty can be modeled by the entropy $h_q(y^* \mid x^*)$. and can be decomposed into aleatoric and epistemic uncertainty, by using Equation 2.71.

$$
h_q(y^* \mid x^*) = \mathbb{E}_{q(w)}[h_q(y^*|x^*,\omega)] + \mathbb{I}(y^* \mid\mid \omega) \tag{2.73}
$$

This equals the Shannon Entropy since y^* is a discrete random variable:

$$
\mathbb{H}_q(y^* \mid x^*) = \underbrace{\mathbb{E}_{q(\omega)}\left[\mathbb{H}_q(y^*|x^*,\omega)\right]}_{aleatoric} + \underbrace{\mathbb{I}(y^* \mid\mid \omega)}_{epistemic} \tag{2.74}
$$

Since the aleatoric uncertainty $\mathbb{E}_{q(\omega)}[\mathbb{H}_q(y^*|x^*,\omega)$ has fixed weights ω, it does not de-

pend on the variability of ω. The epistemic uncertainty $\mathbb{I}(y^* \mid\mid \omega)$ can be written as:

$$\begin{aligned}
\mathbb{I}(y^* \mid\mid \omega) &= \int_\Omega \sum_{c=1}^{C} log\left(\frac{q(y^* = c, \omega \mid x^*)}{q(y^* = c \mid x^*)q(\omega)}\right) q(y^* = c, \omega \mid x^*)\mathrm{d}\omega \\
&= \int_\Omega \sum_{c=1}^{C} log\left(\frac{q(y^* = c \mid x^*, \omega)q(\omega)}{q(y^* = c \mid x^*)q(\omega)}\right) q(y^* = c \mid x^*, \omega)q(\omega)\mathrm{d}\omega \\
&= \int_\Omega \sum_{c=1}^{C} log\left(\frac{q(y^* = c \mid x^*, \omega)}{q(y^* = c \mid x^*)}\right) q(y^* = c \mid x^*, \omega)q(\omega)\mathrm{d}\omega \\
&= \int_\Omega \sum_{c=1}^{C} \underbrace{log(q(y^* = c \mid x^*, \omega) - log(q(y^* = c \mid x^*))}_{\text{variability depending on } \omega} q(y^* = c \mid x^*, \omega)q(\omega)\mathrm{d}\omega
\end{aligned}$$

Computation of Epistemic Uncertainty The epistemic uncertainty using the information theoretic approach can by Theorem 2.34 be calculated by:

$$\mathbb{I}(y^* \mid\mid \omega) = h_q(y^* \mid x^*) - \mathbb{E}_{q(w)}[h_q(y^* | x^*, \omega)] \tag{2.75}$$

This equals the Shannon Entropy as y^* is a discrete random variable:

$$\mathbb{I}(y^* \mid\mid \omega) = \mathbb{H}_q(y^* \mid x^*) - \mathbb{E}_{q_{(\omega)}}\left[\mathbb{H}_q(y^* | x^*, \omega)\right] \tag{2.76}$$

Which can again be approximated using the tractable estimator $\hat{q}$ of q and an unbiased approximation of the expectation with $\hat{\omega}_t \sim q(\omega)$:

$$\mathbb{I}(y^* \mid\mid \omega) \approx \mathbb{H}_{\hat{q}}(y^* \mid x^*) - \frac{1}{T}\sum_{t=1}^{T} \mathbb{H}_q(y^* \mid x^*, \hat{\omega}_t). \tag{2.77}$$

A tractable estimator $\hat{q}$ is for example a Monte Carlo Variational Inference estimation.

3 Active Learning

As already mentioned, the aim of active learning is to find the most label-efficient data set. Therefore, in Section 3.1 different scenarios are discussed in order to find the most appropriate scenario for the given use case. Then, in Section 3.2 a general active learning framework is developed for the selected scenario. Finally, from Section 3.3 to Section 3.6 different query strategies are presented to be used in this framework. And in Section 3.7 are regularization methods for active learning presented in order to prevent the used network from overfitting on the small selected data sets.

By [58] active learning (other names in the literature are "query learning" or "optimal experiment design") is a subfield of Machine Learning, which is a subfield of Artificial Intelligence. In some Machine Learning tasks it is expensive to label all the data. To overcome this bottleneck active learning uses queries of unlabeled instances, which the oracle has to label. The goal of active learning is to use as few labeled instances as possible while the active learner still wants the used network to achieve a high test accuracy.

An oracle can be a human being or a machine with the task to label the chosen instances and a learner will in the following be considered as the framework or algorithm which aims to find or create the best fitting instances for the learning process.

3.1 Scenarios of Active Learning

By [58] there exist various problem scenarios for the context of active learning. In the literature three scenarios have mainly been considered: membership query synthesis, stream-based selective sampling and pool-based sampling. All these scenarios assume that unlabeled instances are queued in order to be labeled by an oracle.

Membership Query Synthesis The learner requests labels for any unlabeled instances [2]. Those unlabeled instances can be generated by the learner itself or sampled from an underlying natural distribution, but the first case is mostly considered. The application of membership query synthesis is in some cases reasonable, but it gets difficult to annotate the generated sample for a human. Thus, this scenario is, except for augmentations (see Section 3.7) not further considered for the given task. But in the case of augmentation an instance from a natural distribution is labeled first and then it gets augmented. Thus, there is no need to annotate augmented samples.

Stream-Based Selective Sampling Stream-Based selective sampling introduced by [4] (also known as Stream-Based active learning or Sequential active learning) underlies the assumption that obtaining an unlabeled instance is for free. Thus, after obtaining such an instance, the learner decides whether this instance needs to be labeled by the oracle. Opposing to Membership Query Synthesis such instances only come from an underlying natural distribution. The decision whether an instance has to be labeled by the oracle can be made with the help of an informative measure [12]. Such an informative measure can be used in a way that instances, which are according to that measure more informative,

will be more likely to be labeled. An approach for this idea is the following: If the value of such an informative measure on the instance is higher than a specific threshold, then the instance is given to the oracle. Another approach would be to find the *region of uncertainty* [11] which is a part of the *version space* (the space, which is consistent with the labeled data set [45]) in which the instances are most ambiguous to the learner. I.e. if any two models of the same model class agree on all labeled data but disagree on the some unlabeled instance, then that instance lies within the region of uncertainty.

Pool-Based sampling The assumption for Pool-Based sampling, which was proposed by [42] is that a large collection of unlabeled instances can be gathered at once (or before the active learning process). Pool-Based sampling therefore has the following setting: A small labeled set $\mathcal{L}$, which has initially size zero, and a large unlabeled set $\mathcal{U}$. Queries are then selectively drawn from the pool of unlabeled instances [58]. An approach is to query the instances in a greedy fashion according to the value of each instance of the informativeness measurement. Thus, the learner decides which instances are the most informative ones of the pool and therefore have to be labeled by the oracle.

To resume, Stream-Based Selective Sampling scans sequentially through the data and makes query decisions individually whereas Pool-Based Sampling ranks the entire unlabeled set or a subset of it before selecting the best query. For the given task, a large pool of unlabeled data $\mathcal{U}$ is easily obtained. Thus, Pool-Based sampling is the most appropriate scenario for the given task.

3.2 Active Learning Framework

Once the Pool-Based scenario is chosen, Algorithm 4 can be implemented. Nonetheless, Algorithm 4 can handle a Stream-Based scenario as well. Namely, if new unlabeled instances are obtained, it adds them to the pool of unlabeled data $\mathcal{U}$. Then, they are taken into consideration in the next round. Let the labeled set be $\mathcal{L}$ with samples X_L and the corresponding labels Y_L. Let $\mathcal{U}$ be the pool of unlabeled samples and let X_U be the unlabeled samples. Then, the classifier $\mathcal{M}$ is trained by the following active learning framework:

Data: unlabeled instances $\mathcal{U} = (X_U)$, number of rounds $R \in \mathbb{N}$, amount of roundly added samples $k \in \mathbb{N}$, untrained or pretrained classification network $\mathcal{M}$
Result: trained classification network $\mathcal{M}$, labeled set $\mathcal{L} = (X_L, Y_L)$
1: initialize small set $X_I \in X_U$, label it and add (X_I, Y_I) to $\mathcal{L}$
2: train $\mathcal{M}$ on $\mathcal{L}$
3: **for** round in $\{1, ..., R\}$ **do**
4: select k instances X^* X_U
5: find labels Y^* from X^*, add (X^*, Y^*) to $\mathcal{L}$
6: (optional: augment X^* to $(\hat{X}^*, \hat{Y}^*)$, add $(\hat{X}^*, \hat{Y}^*)$ to $\mathcal{L}$)
7: retrain $\mathcal{M}$ on $\mathcal{L}$
8: **end for**

Algorithm 4: General Active Learning Algorithm

This algorithm builds the foundation of the active learning framework. In line 4, Algorithm 4 uses the in the upcoming subsections presented sample selection methods. Some of these methods make use of acquisition function to find the most valuable samples others take randomly a specific number of samples from clusters. This line is also represented by the green box in Figure 11.

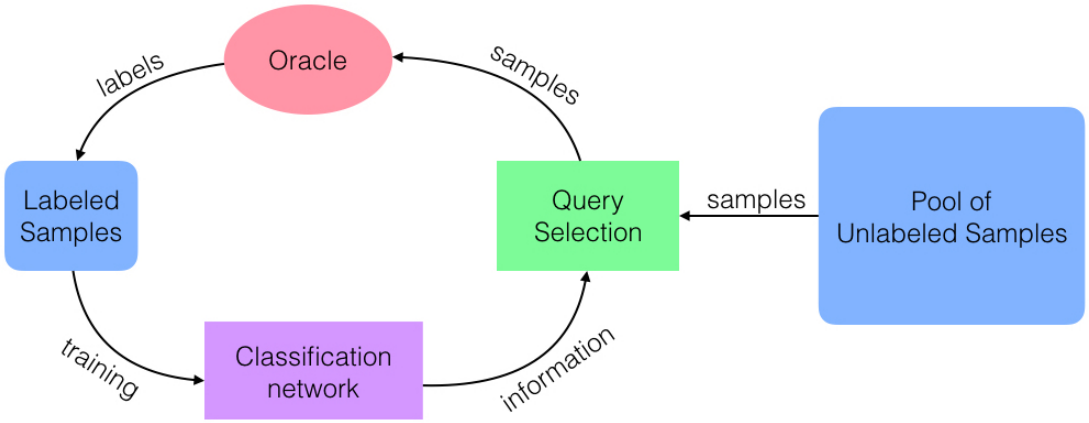


Figure 11: Visualization of the Active Learning Framework

Inspired by [18] an acquisition function is defined as follows:

Definition 3.1. Let $\mathcal{M}$ be a classification network, $\mathcal{U}$ be the pool of unlabeled instances.Let $a : X_U \to \mathbb{R}_+$ Then, a is an **acquisition function** of x, if the active learning framework can use is it to decide which instances $x^* \in X_U$ it has to query next, i.e.

$$x^* = \underset{x \in X_U}{\arg\max}\ a_{\mathcal{M}}(x) \tag{3.1}$$

3.3 Uncertainty Selection Strategies

In these query selection strategies the learner (i.e. the active learning framework) selects the samples with the highest uncertainty. On a sample with a high uncertainty, the network is uncertain about. This can have, depending on the measurement of uncertainty, different meanings. In general, it can be said that a network is uncertain about a sample, if it likely to predict different labels for it on different runs. There are different kinds of methods to measure the uncertainty. One kind uses the output of an artificial neural network whereas another kind uses the outputs of different realizations of a Bayesian neural networks (a committee of networks) to determine the uncertainty. Both kind of methods as well as different kind of measurements of the uncertainty are in the following presented.

3.3.1 Direct Uncertainty Sampling

In the case of a classification task, the output of a neural network is probabilistic. If the output vector of the classification model is not one-hot coded, it is not totally confident

about its class-label. If the vector's highest entries are almost equal, the input sample is close to a decision boundary of the artificial neural network. Thus, the probabilities of the artificial neural network can be used in order to determine its uncertainty. This is quite straightforward for the binary case (i.e. the case of only two classes). Here, the samples with the higher probability being closest to 0.5 are chosen. For classification with more than two classes there are different approaches:

Measurements of Uncertainty The method of the binary case can also be applied to the case of multiple classes. Again, the value of the highest class probability is considered. This leads to the **misclassification rate** or least confident measurement as acquisition function:

$$x^*_{LC} = \arg\max_{x \in X_U} 1 - p_\theta(\hat{y}|x), \text{ with } \hat{y} = \arg\max_{y \in \mathcal{C}} p_\theta(y|x), \tag{3.2}$$

which is equivalent to the expectation of the 0/1-loss because it only considers information about the most probable label.

One can also consider more than one class probability. The difference of the two highest class probabilities is called margin because it is the space between the prediction and the second most probable prediction. Thus, sampling the instance with the smallest margin is called **margin sampling**. The corresponding acquisition function is given by:

$$x^*_M = \arg\max_{x \in X_U} p_\theta(\hat{y}_2|x) - p_\theta(\hat{y}_1|x), \text{ with } \hat{y}_1 = \arg\max_{y \in \mathcal{C}} p_\theta(y|x), \text{ and } \hat{y}_2 = \arg\max_{y \in C \setminus \{y_1\}} p_\theta(y|x) \tag{3.3}$$

The information theoretic approach, which considers all class probabilities and which represents the amount of information needed to encode a distribution, is called Shannon entropy. It is equivalent to the expectation of the log-loss and generalizes better to multiple classes. Those instances are chosen, which have the highest **Shannon entropy**:

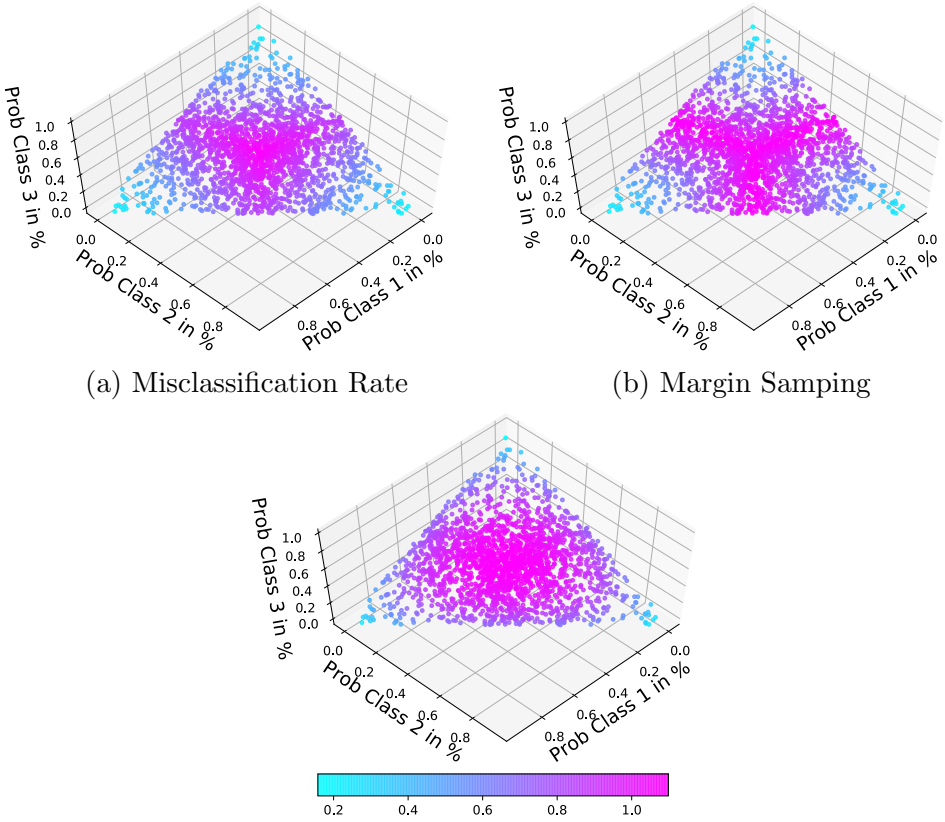
$$x^*_H = \arg\max_{x \in X_U} -\sum_i p_\theta(\hat{y}_i|x) log(p_\theta(\hat{y}_i|x)) \tag{3.4}$$

The difference of the measurements is shown in Figure 12.

Since neural networks tend to be overconfident, their probability estimates may not provide reliable information. In contrast, Bayesian models provide a principled approach to estimate uncertainties of the model [10].

3.3.2 Uncertainty by Committee

The in [16], with the name Query-by-Committee, introduced method consists of a committee of T models $\{\mathcal{M}_1, \mathcal{M}_2, ..., \mathcal{M}_T\}$ with parameters $\{\theta_1, \theta_2, ..., \theta_T\}$, $T \in \mathbb{N}$, which are all trained on the labeled set $\mathcal{L}$, but represent different, competing hypotheses. Each committee member can then vote for a label of an instance. A vote can be hard (i.e. the member of the committee chooses one class) or it can be soft (i.e. each member gives



(a) Misclassification Rate (b) Margin Samping

(c) Shannon Entropy

Figure 12: The uncertainty of 2.000 samples is displayed depending on its probability belonging to one of the three different classes. Different acquisition functions consider different samples as uncertain. According to a given acquisition function, the network is on pink samples very uncertain and on turquoise ones certain. The acquisition functions misclassification rate and margin sampling take, in contrast to entropy, samples into consideration, which have a very low class probability for one class.

a probability for the instance belonging to each class). This yields the following vote functions:

$$\text{Hard votes: } V_h : \{1, ..., C\} \to \{1, ..., T\},\ V_h(c) = \sum_{i=1}^{T} \begin{cases} 1, \text{if } c = \underset{y \in \{1,...,C\}}{\arg\max}\ p_{\theta_i}(y|x) \\ 0, \text{ else,} \end{cases} \tag{3.5}$$

$$\text{and soft votes: } V_s : \{1, ..., C\} \to [1, ..., T],\ V_s(c) = \sum_{i=1}^{T} p_{\theta_i}(y = c|x). \tag{3.6}$$

Both functions will in the following be summarized to $V(c)$ and both can be substituted whenever $V(c)$ is written.

As described in Section 3.1, the aim of the Uncertainty by Committee method is to minimize the region of uncertainty with as few labeled instances as possible. Therefore, the active learning framework chooses the instances on which the members of the committee

most disagree on. This means that different votes are made by the committee members. Such samples should be close to the decision boundary of the Bayesian neural network, of which the committee members are realizations (see Figure 10).

Measurements of Disagreement Again, there are different ways to measure the disagreement in the committee or the uncertainty of the Bayesian neural network respectively. The methods from Section 3.3.1 can be generalized in order to be used for the committee. Define the vote probability as $p_V(c) = \frac{V(c)}{T}$. The generalization of the misclassification rate is then the **variation rate** with the following acquisition function:

$$x^*_{Var} = \underset{x \in X_U}{\arg\max}\ 1 - p_V(\hat{y} \mid x), \text{ with } \hat{y} = \underset{y \in \{1,\dots,C\}}{\arg\max}\ p_V(y \mid x). \tag{3.7}$$

The Shannon entropy will be generalized to the **vote entropy**:

$$x^*_{VE} = \underset{x \in X_U}{\arg\max} \sum_{c=1}^{C} -p_V(y = c \mid x) log(p_V(y = c \mid x)). \tag{3.8}$$

The vote entropy can be decomposed into the **mutual information** $\mathcal{I}(p_{\mathcal{M}_1,\dots,\mathcal{M}_T}; p_V(c))$ of the probabilistic outcomes of the committee members and the vote probability, and their conditioned entropy as shown in Section 2.4.2. Since the mutual information gives an epistemic uncertainty of the Bayesian neural network, represented by the committee members, this approach is theoretically best founded. The corresponding acquisition function is given by:

$$x^*_{MI} = \underset{x \in X_U}{\arg\max}\ \mathbb{I}[(y \mid x); w] = \underset{x \in X_U}{\arg\max}\ \mathbb{H}_{p_V}(y \mid x) - \frac{1}{T} \sum_{t=1}^{T} \mathbb{H}_{p_{\theta_t}}(y \mid x). \tag{3.9}$$

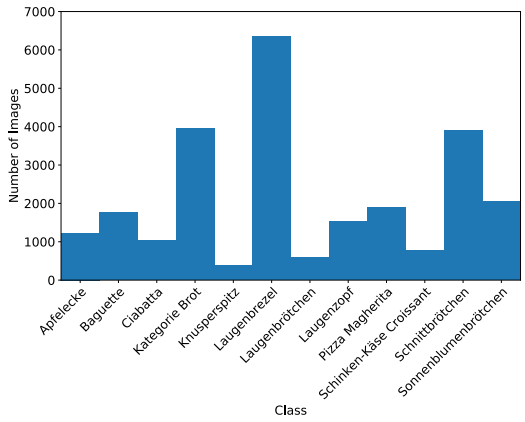
Another way to obtain the approximation of an epistemic uncertainty of the Bayesian neural network is to decompose the **predictive variance** into epistemic and aleatoric uncertainties. This is described in detail in Section 2.4.1. As the predictive variance is given in the form of a matrix, this is also the case for the decomposed epistemic uncertainty. Thus, to get scalar value, which are necessary for an acquisition function's output, the sum is taken over all its entries. This yields the following acquisition function:

$$x^*_{PV} = \underset{x \in X_U}{\arg\max} \sum_{i,j=1}^{C} \left(\frac{1}{T} \sum_{t=1}^{T} p_{\theta_t}(y \mid x) p_{\theta_t}(y \mid x)^T - p_V(y \mid x) p_V(y \mid x)^T \right)_{i,j}. \tag{3.10}$$

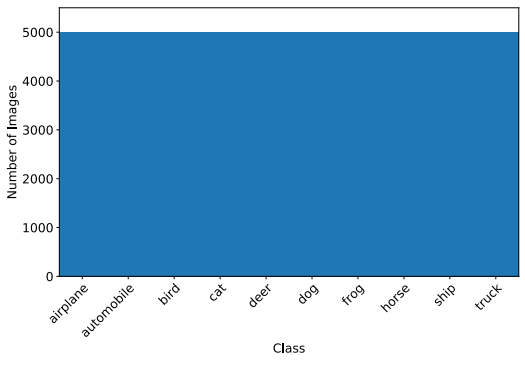
3.4 Class Balancing

Balanced Data Set For a classification task, a balanced data set contains for every class the same amount of instances. This is in most applications not the case because in the real world not every class has the same probability. The data set from Precibake contains for example different amounts of images per class because more Laugenbrezels are sold than other baking products. Thus, balanced data sets are artificially generated. This is done for the CIFAR10 data set or for the balanced test set of bakery products as can be seen in Section 4.1.

Importance of Balanced data sets Artificial neural networks are sensitive to the proportion of the classes. This is due to the fact that the value of the risk function is influenced by every instances with the same weight. Thus, having a high proportion of a specific class (major class), forces the network to prefer the major class. Which means that they have a higher probability than they would have on a balanced data set. This gets problematic if the task is to predict a class with a smaller proportion (minor class) of the data set. An example would be to predict the occurrence of cancer. Usually, the proportion of the society having cancer is lower than the proportion of people not having cancer. Thus, an imbalanced data set is easily created. One assumes that 99% of the people included in the data set do not have cancer. This means that if the network always predicts to "no cancer", its accuracy would be 99% as well. If the task is now to find people having cancer, the network could not perform worse. Having a balanced data set is a good way to overcome this issue. Other examples are given in [5]. [5] also describes ways to overcome this issue. In the following there are different methods presented to overcome this issue in the active learning framework.



(a) **Unbalanced data set** provided by PreciBake



(b) CIFAR10: **balanced data set**

Figure 13: Histograms of an unbalanced and a balanced data set

3.4.1 Class Prediction

In order to get a balanced data set, the label of each sample of the unlabeled pool $\mathcal{U}$ can be predicted by the current model $\mathcal{M}$. Then, the samples from the pool are clustered according to their labels. Afterwards, the same number of samples is, if possible, randomly chosen from every cluster. Supposing that k samples are added in every round of Algorithm 4 and that C is the number of classes. It is possible to add an equal amount of samples to $\mathcal{L}$, if k is divisible by C and the model $\mathcal{M}$ has at least predicted $\frac{k}{C}$ samples for each cluster. If k is not divisible by C, each needs to contain at least $\lfloor \frac{k}{C} \rfloor$ samples in order to add an almost balanced subset of $\mathcal{U}$ to $\mathcal{L}$. If a cluster does not contain $\frac{k}{C}$ samples, all its samples are taken and from the other clusters is an equal amount of samples taken. Assuming that the model $\mathcal{M}$ predicts perfectly (100% accuracy on a test sets), this method would yield a as perfectly as possible balanced subset of $\mathcal{U}$. But if this would be the case, no active learning framework would be needed because the model is already performing perfectly. Still, this method balances the model more than a random selection (assuming that the classes are not already equally distributed) of the subset because the model's accuracy is better than a random prediction. If it would be worse, the model would not have learned from the data.

Consider labeled set $\mathcal{L}$ Since the approach above only takes the pool of unlabeled data $\mathcal{U}$ into consideration, this approach can only asymptotically and not directly balance the labeled set $\mathcal{L}$. In order to obtain a better balanced data set $\mathcal{L}$ the labels from the previous step need to be considered as well. The approach is then to take into account the amount of samples per class and only choose samples from the less represented classes until all classes have an equal amount of instances. Then, the above presented method can be applied.

3.4.2 Local-Sensitivity Hashing

Hash table As stated in [21], a hash table consists of a bucket array and a hash function. The hash function $h : \mathbb{R}^n \to \{0, ..., K-1\}$ gives the index of each sample in the bucket array with K being the capacity of the bucket array. Every bucket of the array can contain multiple instances.

By [38] and [22] Locality-sensitive hashing (LSHash) is an algorithmic technique using hash functions in order to configure buckets such that similar inputs belong to the same bucket with high probability.

Feature Vector A feature vector $fv \in \mathbb{R}^n$ to the sample x is a n-dimensional representation of the sample. In Machine Learning, a feature vector is given as the output of a specific layer of an artificial neural network [6]. In the application of active learning, LSHash is used in order to minimize the amount of similar instances in the training set. This is done in the following way: First, k hyperplanes are randomly generated in the space of the samples feature vectors. Then, of every sample is the feature vector determined to get a lower dimensional vector of the input sample. Afterwards, it is checked

whether the feature vector is above or below the hyperplane. This yields a vector of zeroes (the feature vector is below a hyperplane) and ones (the feature vector is above a hyperplane) as the output of the hash function h. The stated process is formulated in Algorithm 5.

Data: Unlabeled instances (most informative or a random subset) X_U
Result: B instances, which are the first of each bucket
for $j \in \{1, ..., k\}$ **do**
  Generate hyperplane w_j randomly
end for
for $i \in \{1, ..., N\}$ **do**
  sample $x_u \sim X_U$
  determine feature vector fv of x_u
  for $j \in \{1, ..., k\}$ **do**
    Create hash for fv:

$$h(fv)_j^i = \begin{cases} 1, \text{ if } w_i^T a > 0 \\ 0, \text{ else} \end{cases}$$

  end for
  add hash h^i to the hashtable H^i
  choose B instances, which are the first $\lfloor \frac{B}{k} \rfloor$ of each bucket plus the remaining ones
end for

Algorithm 5: Algorithm of **Local-sensitivity Hashing**

In the case of class balancing, the number of different hashes 2^k should equal the amount of different classes C. In the best case, if the feature vectors of the classes differ enough and if the hyperplanes are close to the class boundaries, the LSHash approach finds in each round a balanced subset of the given data set.

Consider labeled set $\mathcal{L}$ The in Section 3.4.1 presented idea to consider the labeled set $\mathcal{L}$ can also be applied to *LSHashs*. In order to apply the idea, a hash of each labeled sample has to be determined first.

Nonetheless, since the feature vectors are generated by the given classification model $\mathcal{M}$, *LSHash* only approximates the label prediction method. If 2^k is considerably higher than the amount of classes C, *LSHash* can be seen as a representation or diversity selection methods, which are introduced in the next subsection.

3.5 Representation and Diversity Selection Strategies

Both methods use a distance metric to find a balanced data set which needs to be labeled. They can both be described by the k-center problem with $k \in \mathbb{N}$ being the number of instances to be labeled. But there are other methods as well to find a most representative subset. One considered method adds the samples to the labeled set, which cannot get by an encoder-decoder network, which is trained on the labeled samples $\mathcal{L}$, reconstructed well.

Representation The aim is to find the most representative subset X^* of a given set V. This means a subset which represents all other samples well. This can be interpreted in mathematical terms by the subset with the lowest sum over all elements which are not in X over the minimal distance to an element of x. Or as presented in [30]:

$$X^* = \underset{X \subset V, |X|=k}{\arg\min} \sum_{y \in V} \min_{x \in X} dist(x, y) \tag{3.11}$$

Theorem 3.2.

$$\underset{X \subset V, |X|=k}{\arg\min} \sum_{y \in V} \min_{x \in X} dist(x, y) = \underset{X \subset V, |X|=k}{\arg\min} \sum_{y \in V \setminus X} \min_{x \in X} dist(x, y) \tag{3.12}$$

Proof.

$$\begin{aligned}
&\underset{X \subset V, |X|=k}{\arg\min} \sum_{y \in V} \min_{x \in X} dist(x, y) \\
&= \underset{X \subset V, |X|=k}{\arg\min} \sum_{y \in V \setminus X} \min_{x \in X} dist(x, y) + \underset{X \subset V, |X|=k}{\arg\min} \sum_{y \in X} \min_{x \in X} dist(x, y) \\
&\overset{dist(x,x)=0}{=} \underset{X \subset V, |X|=k}{\arg\min} \sum_{y \in V \setminus X} \min_{x \in X} dist(x, y) + 0 \\
&= \underset{X \subset V, |X|=k}{\arg\min} \sum_{y \in V \setminus X} \min_{x \in X} dist(x, y)
\end{aligned}$$

□

Diversity The aim is to find the most diverse subset $X \subset V$. That is a subset X in which all elements are as different from each other as possible. This means in mathematical terms that the subset with the minimal value of the minimum mutual distance $dist(x, y)$ of elements $x, y \in X$ is chosen. Or as presented in [30]:

$$X^* = \underset{X \subset V, |X|=k}{\arg\max} \min_{x,y \in X, x \neq y} dist(x, y). \tag{3.13}$$

Another way to interpret a most diverse subset was presented in [70] and is of the form:

$$X^* = \underset{X \subset V, |X|=k}{\arg\max} \frac{1}{|X|(|X|-1)} \sum_{x \in X} \sum_{x' \in X, t' \neq t} dist(x, x'). \tag{3.14}$$

The most diverse set is in this form a set in which the sum over all mutual distances is the highest.

3.5.1 k-Center Problem

The aim of k-center problem is to find a subset of k data points, k centers, which represents the given data set best. The idea is to minimize the maximum distance of a data point

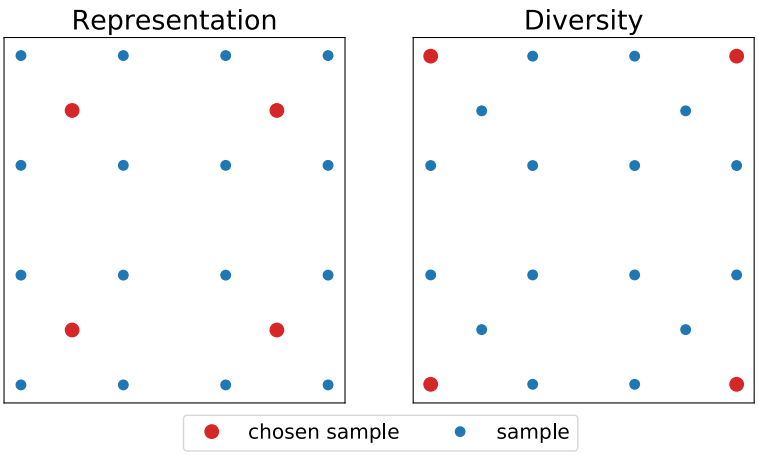


Figure 14: Visualization of the most **representative** (using formula (3.11)) and the most **diverse subset** (using formula (3.13)) of a given set.

of the given data set to a center [57], [43]. This idea has already been mathematically formulated in Equation 3.11. A similar problem can be analogously formulated for a most diverse subset consisting of k data points (see Equation 3.13).

In order to find an optimal subset for representation or diversity, every subset of the size k of a set of size n needs to be evaluated according to a measurement of its representation or diversity. The computational cost of it is $\mathcal{O}(\binom{n}{k}))$. Having high numbers of the size n of the set as well as the size k of the subset, it is intractable to find a solution of the k-center problem. This is stated in the following theorem.

Theorem 3.3 (NP Hardness k-Center Problem)**.** *Approximating the k-Center Problem with any factor ϵ is $\mathbb{NP}$-hard.*

Proof. Pages 3-8 from [57]. □

Greedy Algorithm An algorithm which does not need to consider every possible subset X but aims to construct a set, which approximates the optimal solution, is the greedy algorithm. Let $f(j|X^i) = f(X^i \cup \{j\}) - f(X^i)$ with f being the utility function of the subset (e.g. for representation Formula 3.11 and for diversity Formula 3.13), then the greedy algorithm for the k-center problem can be formulated as:

Data: k, images, similarity S
Result: subset of size k
1: $X^0 = \{\}$
2: **while** $X^i < k$ **do**
3: $\quad X^{i+1} = X^i \cup \underset{j \in V \setminus X^i}{\arg\max}\ f(j|X^i)$
4: **end while**

Algorithm 6: k-Center Greedy Algorithm

To formulate how well the greedy algorithm approximates the k-Center problem, the

bottleneck distance is introduced.

Definition 3.4. Let $\mathcal{S}$ be a set and $\mathcal{C}$ be the set of chosen centers. Let $\mathcal{N}_c\partial\mathcal{S}$ be a set of points to which the center point c is the closest one. The bottleneck of a center point $c \in \mathcal{C}$ is defined as the maximum distance of a point $x \in \mathcal{N}_c$ to c. Then, the **bottleneck distance** of the set of centers is defined as the maximum bottleneck over all center points.

Theorem 3.5. *Let $\mathcal{S}$ be a set, let $\mathcal{C}$ be the set of the optimal solution of the k-center problem. Then, the bottleneck distance of the greedy algorithm is at most two times bigger than the bottleneck distance of the optimal solution.*

Proof. Pages 68-72 from [47] (Greedy Approximation Algorithms: The k-Center-Problem). □

Thus, with a growing number of selected samples, the difference between the optimal solution and the by the greedy algorithm constructed solution is converging to zero.

Distance Metrics In order to be able to apply the k-center problem, a distance metric has to be used. Those metrics can be directly applied to two samples directly or to their feature vectors. Direct distance metrics can for example be the following: In a normed space a distance measurement can be defined as $dist(x, y) = ||y - x||_p$ of two vectors $x, y \in \mathbb{R}^n$. In the case of computer vision, images can be vectorized and afterwards they can be proceeded in the common way (i.e., taking the element-wise difference and then the p-norm of it).

Depending on the layer of which the feature vector is taken, different representations of the sample are given, which leads to different distances. A feature vector of the first layer would be close to the direct difference whereas the feature vector of the last layer would very likely differ for samples from different classes. Hence, taking the feature vectors of the last layer into consideration may lead to class balancing, which was introduced in Chapter 3.4. Taking the feature vectors of one of the first layers into consideration would yield distances due to differences in more general features of the samples, whereas the distance of feature vectors of later layers would be due to differences in more specific features of the samples.

In the tested methods, the negative cosine similarity was used with the feature vectors of the penultimate layer as a distance measure, i.e.

$$dict(x, y) = -cos(x, y) = -\frac{x^T y}{||x|| \cdot ||y||}. \tag{3.15}$$

A similarity of 1 means in this case that the vectors are similar, while -1 indicates, that they are the exact opposite.

As artificial neural network are used in order to determine the feature vectors, the distance, which uses feature vectors, depends on the data on which the artificial neural network was trained on in previous rounds. Therefore, the distance between samples of different classes should grow with a growing number of labeled samples, on which the network was trained on before the evaluation on the unlabeled samples.

Problem with the Greedy Algorithm In each step the greedy algorithm has to consider all possible set with one more sample. For a big size of V the value of the utility function of a lot sets needs to be considered. As the size of the set X is increasing with every step of the greedy algorithm, the calculation of the function's value is taking much more time as well. This makes the greedy algorithm too slow to be used in practice. Thus, a faster method is needed.

LSHash as Approximation With a high number of different hashes (each containing at least one instance) - optimally as high as the size of chosen subset is - *LSHash* can be used to approximate the construction of the most diverse subset. This is due to the fact that the hyperplanes, which are constructed by *LSHash* divide the feature space, in which the feature vectors of the considered samples lie. Thus, having the feature space divided in the number of samples which are chosen, the diversity and the representation problem can be approximated. With a higher number of chosen instances the approximation becomes better.

There are two options how to use *LSHash*. The first option is to only consider instances of the set, of which the instances should be chosen (i.e. a subset of the unlabeled pool $\mathcal{U}$ from. Then, from every bucket the first instances are chosen until the number of chosen instances is reached. If one bucket does not contain any element, it is not considered. The other option is to determine the hash of the already labeled instances first and to add them to the corresponding bucket in the hash table. Then, determine the hash of the set of which the instances should be chosen and to add them as well. As before the firstly added elements of the each bucket are taken. But if an element is already labeled then this element is not taken. Instead, the bucket is skipped. This yields to a more diverse labeled data set.

Advantages of LSHash Let N be the size of the set of which a subset needs to be chosen. Then, from every sample a hash is only once determined and then a specific amount of samples is chosen from every hash. Therefore, the computational effort is with $\mathcal{O}(N)$ much less than the computational effort of the greedy algorithm (Algorithm 6), which has a computational effort of $\mathcal{O}(N^2)$.

Which set should get represented? The arising question is whether to take into account the labeled set and the unlabeled set or only consider the unlabeled set as the set to get a representative subset from. Assume to take into account the union of the labeled and unlabeled set as the one to find a representative subset from. In the general framework (Algorithm 4), a specific number k of instances needs to be annotated in every round. Let L be the size of the labeled set $\mathcal{L}$ and K be the number of centers needed to find for the k-Center problem. Thus, with an increasing number of rounds the number $K = L + k$ of centers increases as well. Finding K centers would also differ in every round. Instances, which were considered as a center in a previous round may not be a center in the next round. Thus, in every round $L + k$ centers have to be found, which can get intractable for the greedy algorithm. If only the unlabeled set has to be represented, this can be done in every round with a non-changing amount of centers $K = k$. Hence, only the unlabeled

set was considered in the experiments. Using both sets does not become intractable for *LSHash* as mentioned in the previous paragraph. How to apply the k-Center problem for both data sets was already mentioned in Section 3.4.

3.5.2 Reconstruction Methods

Reconstruction Error The idea of this method is the following: If the reconstruction error of an encoder-decoder model, which is trained on the labeled set $\mathcal{L}$, is for an instance low, then this instances is well represented by the other instances from $\mathcal{L}$. This means that the encoder-decoder is trained on samples, which are similar (or "close") to the current instance. Thus, there is no need to label it. Contrary, if an instance has a high reconstruction error, it is not represented well by $\mathcal{L}$. Such instances are chosen by the, in the following, introduced acquisition function. An encoder maps the given high-dimensional input to a low-dimensional latent representation and a decoder aims to map a low-dimensional latent representation to the high-dimensional input. This is visualized in Figure 15. Let e be an encoder network and d be the corresponding decoder network. Then the reconstruction loss is defined as:

$$L_{rec}(x) = ||x - x_{rec}||_2, \text{ with } x_{rec} = d(e(x)). \tag{3.16}$$

The corresponding acquisition function is therefore defined as:

$$x^*_{rec} = \underset{x \in X_U}{\arg\max}\ L_{rec}(x). \tag{3.17}$$

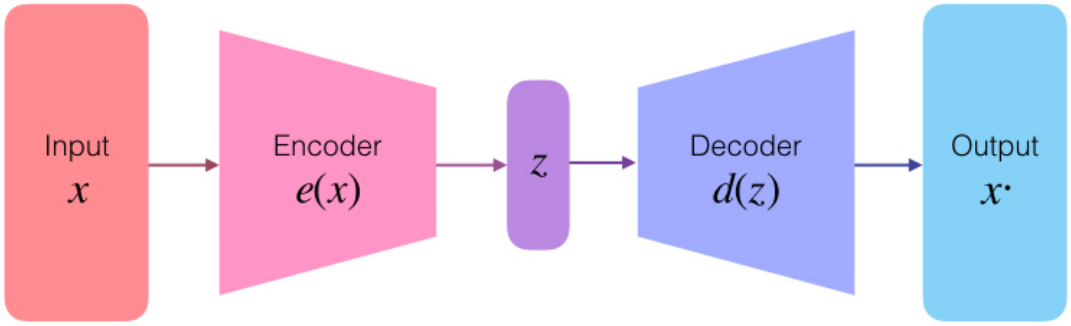


Figure 15: Visualization of the general idea of **Decoder-Encoder** networks.

For the experiments a Variational Autoencoder (VAE) (introduced by [34]) is used as an encoder-decoder model. As explained in [15], the goal of Virtual Autoencoders is to get a good latent representation z given the observed data $\mathcal{L}$. Thus, the goal is to find the underlying distribution $p(z \mid \mathcal{L})$. Using Bayes' Theorem (Theorem 2.12, this equation can be expressed as:

$$p(z \mid \mathcal{L}) = \frac{p(\mathcal{L} \mid z)p(z)}{p(\mathcal{L})}. \tag{3.18}$$

Again, $p(\mathcal{L})$ is intractable. Thus, Variational Inference needs to be performed as shown in Section 2.3.2. $p(z \mid \mathcal{L})$ is assumed to be Gaussian. Therefore the Gaussian parameterization family is used in order to find the best approximation $q_\theta(z \mid \mathcal{L})$ of $p(z \mid \mathcal{L})$. In

order to find a tractable solution ELBO is used (see Section 2.3.2). This finally yields (according to Equation 2.32) the following risk function:

$$R_{VI}(q_\theta(z \mid \mathcal{L}), p(z \mid \mathcal{L})) = -\mathbb{E}_{q_\theta(z|\mathcal{L})}[log(p(\mathcal{L} \mid z))] + \mathbb{KL}(q_\theta(z \mid \mathcal{L})||p(z))) \tag{3.19}$$

$p(\mathcal{L} \mid z)$ represents the decoder and $q_\theta(z \mid \mathcal{L})$ the encoder. The first term of Equality 3.19 encourages the decoder to learn to reconstruct the observed data $\mathcal{L}$, whereas the second terms aims to keep the representation of the latent variable z diverse.

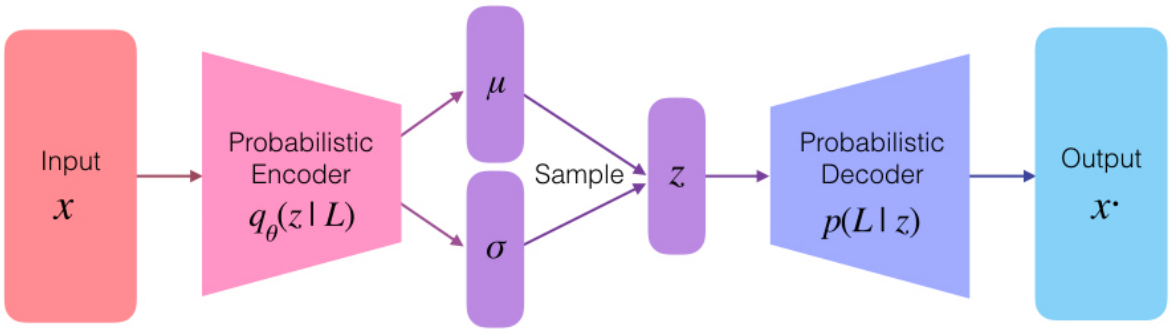


Figure 16: Visualization of the general idea of **Virtual Autoencoder**s.

The general idea of VAE is visualized in Figure 16 and the used model architectures are shown in Section A.4. For the encoder, the actual model of Algorithm 4 could have been used as well.

Virtual Adversarial Active Learning As described in [60], the Virtual Adversarial Active Learning (VAAL) framework consists of a Variational Autoencoder and a discriminator. The discriminator (also known as adversarial network) classifies to which pool the instances belong to (i.e., the unlabeled pool $\mathcal{U}$ or the labeled pool $\mathcal{L}$). The discriminator is therefore an artificial neural network with two artificial neurons in the output layer. The VAE learns a latent representation such that sets of labeled and unlabeled instances are mapped into one common embedding. As the discriminator aims to distinguish between labeled and unlabeled instances, this leads to a min-max game, which is usually known in GAN-architectures [60]. This, is due to the fact that the VAE network wants the adversarial network to classify all instances as labeled, whereas the discriminator wants to distinguish between labeled and unlabeled instances. This can be confirmed by taking a look at the risk function:

$$\begin{aligned} R^{trd}_{VAE} = &\mathbb{E}[log(p_\theta(x_L|z_L))] - \beta\mathbb{KL}(q_\theta(z_L|x_L)||p(z_L)) \\ &+ \mathbb{E}[log(p_\theta(x_U|z_U))] - \beta\mathbb{KL}(q_\theta(z_U|x_U)||p(z_L)) \end{aligned} \tag{3.20}$$

The aim of this risk function is to measure how well the instances can be reconstructed.

$$R^{adv}_{VAE} = -\mathbb{E}[log(D(q_\theta(z_L|x_L)))] - \mathbb{E}[log(D(q_\theta(z_U|x_U)))] \tag{3.21}$$

This risk function is low if all images are classified by the discriminator as belonging to the labeled pool.

$$R_{VAE} = \lambda_1 R^{trd}_{VAE} + \lambda_2 R^{adv}_{VAE} \tag{3.22}$$

This risk function is a composition of the previous ones with hyperparameters $\lambda_1, \lambda_2 \in \mathbb{R}$, which is used to train the VAE network.

$$R_D = -\mathbb{E}[log(D(q_\theta(z_L|x_L)))] - \mathbb{E}[log(1 - D(q_\theta(z_U|x_U)))] \tag{3.23}$$

This risk function is low if the labeled samples are classified as labeled samples and the unlabeled ones as unlabeled.

To select the most representative subset of $\mathcal{U}$, the learner selects instances, which are sufficiently different from the latent space. This sample selection process is done by a discriminator and is done as follows:

$$X_S = \underset{|X_S|=k}{\arg\min}\{D(q_\theta(z_U))\} \tag{3.24}$$

Thus, those instances are selected on which the discriminator is least confident about. This leads to the following algorithm for the selection of unlabeled data points to get annotated:

Data: labeled instances (X_L, Y_L), unlabeled instances (X_U)
Result: $X^* \subset X_U$
for epoch in epochs **do**
  sample batch $(x_L, y_L) \sim (X_L, Y_L)$
  sample batch $(x_U) \sim (X_U)$
  compute risk functions R_{VAE} and R_D
  update network parameters:
  $\theta'_{VAE} = \theta_{VAE} - \alpha_1 \nabla R_{VAE}$
  $\theta'_D = \theta_D - \alpha_2 \nabla R_D$;
end for
select samples X^* using equation 3.24

Algorithm 7: Data Selection with VAAL

Similarities and Differences between Uncertainty, Diversity and Representation Sampling The aim of uncertainty sampling is to find samples, on which the model is most uncertain about. While the model is trained on the labeled data set and aims to find instances, on which it is uncertain, this may yield, in the case of uncertainty, to sampling samples, which differ from the labeled set. This is comparable to finding a subset, which is diverse, if both the unlabeled and the labeled data set are considered. But for uncertainty sampling, the mutual diversity within the selected subset is not considered.

In the case of uncertainty sampling the model is trained on the labeled data and it aims to find instances on which it is uncertain about, those instances should be instances which the model has not "seen" before or which are badly represented by the ones it has learned from. Thus, the aim is to find a representative subset of the unlabeled samples. Since the mutual similarity within the selected subset is, again, not considered for uncertainty sampling, this approach can be better compared with the reconstruction methods than the other representation sampling methods.

But since in all cases it is dealt with approximations of the problem and the solutions differ in practice. Therefore, they are presented in different subsections and are seen as different approaches.

3.6 Combination of different Methods

In this subsection, fours ways are presented in order to combine different subset selection methods on the unlabeled data set $\mathcal{U}$. In the presented methods either an acquisition function was used to select the most valuable (*Weighting*) samples, $\mathcal{U}$ was divided into clusters (*Dividing*) or a subset was constructed (*Constructing*).

Weighting and Constructing Weighting and Constructing can be combined in the following way: Let k be the size of the selected subset. Then, $\mathcal{U}$ is weighted according to an acquisition function. Afterwards, $h > k$ samples of $\mathcal{U}$ are chosen for which the acquisition function has the highest values. Finally, from this set a subset of size k is constructed.

An application of this combination method was proposed by [30]. Here, h samples with the highest direct uncertainty were chosen and a subset was constructed using the Greedy algorithm (see Algorithm 6) or a variation of the algorithm, which is faster.

Weighting and Clustering *Clustering and Weighting* can be combined in two ways. The first one is to first divide $\mathcal{U}$ into clusters and then apply weighting on every cluster, i.e. take from every cluster the samples with the highest acquisition value. The second way is similar to the one of *Weighting and Constructing*, i.e. first select a subset of size $h > k$ and then cluster this subset and select from every cluster the same amount of samples randomly.

Weighting and Weighting The idea is to combine two acquisition functions a_1 and a_2 with a hyperparameter λ:

$$a_3(x) = a_1(x) + \lambda a_2(x), \tag{3.25}$$

An application of this method was proposed by [52]. Its aim is to select samples, on which the model is not only uncertain but which are representative as well. The corresponding framework is called informative density framework. a measures the uncertainty and the second term how well a sample can represents the other samples from $\mathcal{U}$. Applying $exp(a_3)$ yields with $a(x) = e_1^a(x)$, $\left(\frac{1}{|X_U|}\sum_{x' \in X_U} sim(x, x')\right)$ and $\beta = e^\lambda$:

$$x_{ID}^* = \underset{x \in X_U}{\arg\max}\ a(x) * \left(\frac{1}{|X_U|}\sum_{x' \in X_U} sim(x, x')\right)^\beta, \text{ with } sim = -dist. \tag{3.26}$$

As stated in [58], this method is superior to methods which do not include a measure to find representative samples. Moreover, if the distances can be precomputed efficiently for

later use, the time required to select the next query would not be essentially different to the time required when only the acquisition function a is considered. This is due to the fact that the distance of two samples does not change with respect to the training of the model. Therefore, distances are not allowed to be computed with the help of the feature vectors of the current model.

Clustering and (Clustering or Constructing) Finally, it is also possible to divide $\mathcal{U}$ first into clusters and then construct within each cluster a subset or divide it again into clusters and select from each of those cluster randomly samples. Thus, class balance methods can be combined with representation methods in order to get a class balanced subset, which is representative for every class.

The only applied combination is, until now, the first way of *Weighting and Clustering*.

3.7 Regularization in Active Learning

As the labeled set $\mathcal{L}$ is, especially during the first rounds of the active learning framework, quite small and the artificial neural network has a lot of parameters, it is prone to overfit to the training data. This means that it could easily remember every data point of the training set exactly but it would still perform badly at the test set because it is not able to generalize well. But the goal of active learning is to perform well on unseen data with only as many data points as needed. Therefore it is helpful to use regularization techniques to generalize better.

Early Stopping of Training While the training error keeps reducing with more epochs, the validation error may reach a point after which it is increasing (see Figure 18). At this point the model starts to overfit. A way to overcome this problem is to take the model with the lowest validation error. This can be done in the following way: A copy of the model is stored every time the error on the validation set is lower than the errror of the previous best model. After training the parameters of the model with the lowest validation error are then loaded in order to get the most general model. This regularization technique is used in all experiments.

Noise Injection Adding or multiplying, during training time, noise to the network's hidden units is, as described in [48], another regularization technique for neural networks. Noise injection methods aim to find the best trade-off between the data fitting process (having the lowest value of the risk function on the training data) and model regularization [48]. Two examples of noise injection were described in Section 2.2 and Section 2.3, namely having a Batch normalization layer or a Dropout layer. Both are applied in the convolutional neural networks in the experiments since injected noise can also be used in order to approximate a Bayesian neural network as shown in section 2.3.

Adversarial Training Artificial neural networks are vulnerable to small perturbations in a specific direction [46], i.e. the adversarial direction. The adversarial direction is the

direction in the input space, in which the label probability $p_\omega(y = k|x)$ of the model is most sensitive.

Thus, [20] proposed *Adversarial Training*, in which the model is trained in order to assign to each data point of the input space a label that is similar to the labels, which are assigned to its neighbors in the adversarial direction. Let D be a non-negative divergence function between two probability distributions, let $x_l \in X_L$ be a instance of the labeled samples $\mathcal{L}$ and let $q(y|x_l)$ be the true distribution of the output label, which is unknown and can be approximated by a one-hot coded vector (which is one at the true label of x_l). Then, the adversarial loss on x_l, given the networks parameters ω, is defined as:

$$L_{adv}(x_l, \omega) := D[q(y|x_l), p_\omega(y|x_l + r_{adv})], \tag{3.27}$$

$$\text{where } r_{adv} := \arg\max_{r;||r||\leq\epsilon} D[q(y|x_l), p_\omega(y|x_l + r)] \tag{3.28}$$

Virtual Adversarial Training The in [46] proposed method *Virtual Adversarial Training (VAT)* is a semi-supervised *Adversarial Training* method. Let x_* be either a labeled instance $x_l \in X_L$ or an unlabeled instance $x_u \in X_U$. To be able to use the unlabeled instances as well, "virtual" labels are introduced. "Virtual" labels $\hat{y}$ are sampled from the current model distribution $p_\omega(y; x)$. They are used in order to approximate the true distribution of the output label $q(y \mid x)$.

$$L_{vadv}(x_*, \omega) := D[p_\omega(\hat{y}|x_*), p_\omega(\hat{y}|x_* + r_{vadv})], \tag{3.29}$$

$$r_{vadv} := \arg\max_{r;||r||_2\leq\epsilon} D[p_\omega(\hat{y}|x_*), p_\omega(\hat{y}|x_* + r)] \tag{3.30}$$

[46] proposes to use a regularization term of the form:

$$R_{vadv}(\omega) = \frac{1}{\mid X_L \mid + \mid X_U \mid} \sum_{x_* \in X_L, X_U} L_{vadv}(x_*, \omega), \tag{3.31}$$

which leads to a regularized risk function of the form $R_{reg}(\omega) = R(\omega) + \alpha R_{vadv}(\omega)$. Thus, in contrast to generative models, only two hyperparameters (namely ϵ and α) need to be set.

This method is of interest for active learning as it takes, during training, the pool of unlabeled into consideration. Since the pool of unlabeled data $\mathcal{U}$ is usually big in comparison to the set of labeled instances $\mathcal{L}$, it contains a lot of additional information and should therefore yield better classification boundaries.

Supervised *VAT* is almost similar to *Adversarial Training*, except that the actual labels are replaced by virtual labels. Semi-supervised and supervised *VAT* regularization methods are applied in the experiment section.

Augmentation Random image augmentation is a variant of random perturbation that simply augments the data set consisting of images by perturbation using regular deformation [46].

As stated in [67], instead of using a large data set, augmentation can be used in order to create an artificially large data set. Data set augmentation is applied in computer

vision tasks, especially in classification tasks. This is due to the fact that images are high dimensional and can include factors of variation which can be easily simulated. Examples are operations like translations, rotation, cropping, etc.

Two different augmentation techniques are test in the experiment section. The used deformation techniques are displayed in the appendix (see Section B of the Appendix).

A whole active learning framework, which takes into consideration both uncertainty by committee sampling and augmentation of the sampled images is described by [66] and explained in more detail in Section D.2 of the appendix because it seems to be promising for further testing.

4 Experiments

In this chapter are the experimental results presented. In Section 4.1, the experimental setup is explained. Then, in Section 4.2 are the query selection methods evaluated. And in Section 4.3 are learning curves shown in order to compare the performance of different methods with random sampling and with the training on the whole data set. A learning curve, which reaches the test accuracy of the whole data set before having labeled all the data, is said to be one solution of the problem statement of Section 1.1. The best solution is the one, which reaches the test accuracy of the whole data set first.

Table 1: In this table are the used **experiment types** displayed. MC is the abbreviation of Monte Carlo and therefore describes the tractable Bayesian neural network approximations of Section 2.3, i.e. MC Ensemble Cyclic is the Monte Carlo Variational Inference method, which uses ensembles with a cyclic learning rate.

Query Method	Specification	Specification	Abbreviation
Direct Uncertainty	Misclassification Rate		DU-MR
	Shannon Entropy		DU-SE
Uncertainty by Committee	MC Dropout	Mutual Information	UC-D-MI
		Predictive Variance	UC-D-PV
	MC Batchnorm	Mutual Information	UC-B-MI
		Predictive Variance	UC-B-PV
	MC Ensemble	Mutual Information	UC-E-MI
		Predictive Variance	UC-E-PV
	MC Ensemble Cyclic	Mutual Information	UC-EC-MI
		Predictive Variance	UC-EC-PV
	MC Ensemble DPE	Mutual Information	UC-ED-MI
Class Balance	Class predicitions		CB-LS-L
	Class predictions	Including labeled set $\mathcal{L}$	CB-LS-L
	LSHash		CB-LS-LV
	LSHash	Including labeled set $\mathcal{L}$	CB-LS-L
Representation	Representation		RP-RP
	Diversity		RP-DV
	Reconstruction Err.	VAE with FC-NN	RP-RE-FC
	Reconstruction Err.	VAE with CNN	RP-RE-CN
	Virtual Advers.	Active Learning	RP-VA-AL
Regularization	Virtual Advers. Train.	Semisupervised	RG-VA-SS
	Virtual Advers. Train.	Supervised	RG-VA-SV
	Augmentation	1	RG-AU-1
	Augmentation	2	RG-AU-2
Combinations	CB-PB	DU-MR	CO-CB-DU-CP-MR
	CB-PB	DU-SE	CO-CB-DU-CP-SE
	CB-PB	UC-D-MI	CO-CB-UC-CP-D-MI
	CB-PB	UC-B-MI	CO-CB-UC-CP-B-MI
	CB-PB	UC-D-MI, RG-VA	CO-CB-UC-CP-D-MI-VA
	CB-PB	UC-D-PV	CO-CB-UC-CP-D-PV
	CB-PB	UC-B-PV	CO-CB-UC-CP-B-PV

4.1 Experimental Setup

Now, the setup of Algorithm 4 is explained. In Algorithm 4, an artificial neural network is trained by an initially selected subset of the given pool of unlabeled data $\mathcal{U}$. Then, the active learning process is done for $R \in \mathbb{N}$ rounds. In every round the following steps are done: Query selection is applied, i.e. a subset from $\mathcal{U}$ is selected. Then, these samples are labeled by an oracle (a human annotator for instance). Finally, the network is retrained on the labeled samples $\mathcal{L}$.

Since the query selection methods have already been explained in detail in Section 3, the experimental setup for the following steps needs to get specified in the following: the oracle, the data set, the classification network and the training process.

4.1.1 Oracle

For the purpose of this thesis, data set are chosen, which are already labeled. But samples, which have not been labeled according to the framework, are treated as if they have not been labeled yet. Thus, there is no human annotator needed for this experimental session.

4.1.2 Data Sets

Different computer vision data sets are used, to get more empirical evidence about the performance of the different methods. Moreover, the selected data sets have different characteristics regarding the resolution of the images and the class balance of the data sets. Thus, if a method is performing well on both data sets, there is a higher chance that it may also perform well on another data set.

PreciBake's Data Set This data set contains 12 classes of bakery products from four different ovens. The images are made during the loading process. As can be seen in Table 20 the classes have different amounts of images and the data set is therefore not class balanced. This makes it less easy to find a class balanced labeled set and it is therefore less probable that the random sampling method finds a class balanced labeled set. This data set is not artificially created and it therefore may represent the real world distribution of baking products. The images have size $224 \times 224 \times 3$ and are therefore colored.

CIFAR10 CIFAR10 contains of 60,000 images of 10 classes (see Table 21). Each image has size $32 \times 32 \times 3$, thus they are colored low resolution images. Since every class contains the same amount of images, the data set is class balanced. Thus, the random sample selection is less prone to construct an unbalanced data set. Therefore, it is more difficult to find better samples for the Active Learning framework.

4.1.3 Classification Network

For each data set a different classification model is used because the input size of the images differs. But for all experiments for a data set the same model is trained and evaluated in order to be able to compare different selection methods. That does not mean that a different method would not outperform the best network in this setting when another network is used instead. Prerequisite for the model are that it has Batch normalization and Dropout layers because there are methods, which need at least one of these two methods. A state of the art network which satisfies both requirements is *DenseNet* [28].

In the experiments, the active learning framework uses for the training with the data set provided by PreciBake the *DenseNetSmall-128* architecture (see Section A.2, which consists of the *DenseNet121* architecture (see Section A.1), except that the third *Transition layer* and the fourth *Dense block* are removed from the *DenseNetSmall-128* model. For the training on the CIFAR10 set, the *DenseNetSmall-32* architecture (see Section A.3) is used. It consists as well of the *DenseNet121* architecture but additionally, the second *Transition layer* and the third *Dense block* are removed. By the initialization of the model and also all the other randomized processes in the framework seeds has been used for the purpose of reproducibility.

4.1.4 Training Process

Initialization Sets As the network is not yet trained, the initialization is done randomly. To be able to train the network, it is necessary to have one instance in each class. Thus, the initialization process is stopped when every class of the training and every class of the validation set has at least one instance. To have a more significant result, the experiments are done multiple times with different seeds. Different seed lead to different initialization sets. This leads to a different size of those sets. Thus, it is not possible to compare one seed with another (i.e. to determine the mean of the test accuracies with respect to the part of the data set or the number of instances already included). Therefore, only one instances per class is chosen in order to be able to compare different seeds with each other.

Way to Query As querying single samples and retraining the model after adding this single sample is not feasible for deep neural networks [10] it is likely that it does not have a statistical impact on the model's performance. Moreover, adding only one sample is computationally intractable as the data sets are in general huge and the networks has to be retrained as often as an instance is added. Thus, as can be seen in Section B of the appendix, 100 (PreciBake's data set) or 200 (CIFAR10) instances were added after every round. The number of selected instances is higher for the data set provided by PreciBake because its samples have a higher resolution. Moreover, CIFAR10 contains more samples. Adding a subset of a least size 100 is done because this results in significant changes in the test accuracy between different rounds.

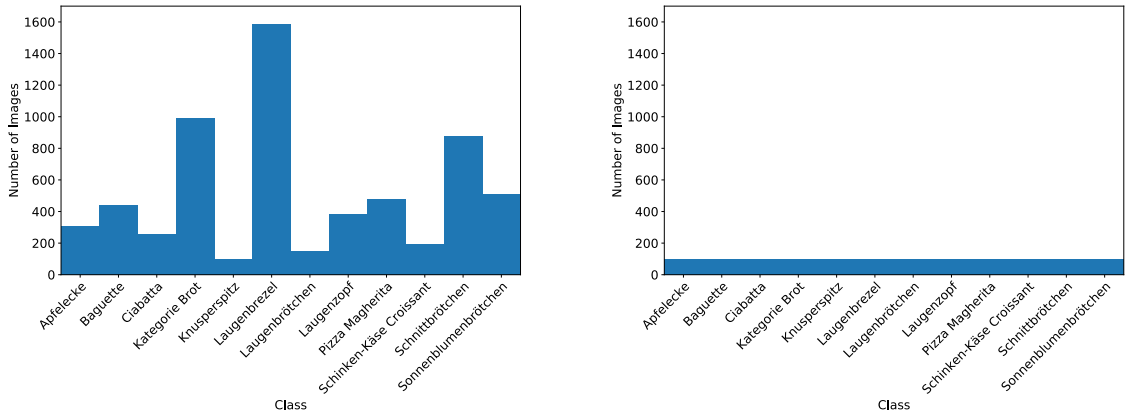
Validation Set Once the instances are labeled, the network needs to be trained on them. For the purpose of training, it is not only a training set but also a validation set needed.

Active Learning for Validation Set Since a good performance on the validation set should lead to a good performance on the test set, it is crucial to select data points for the validation set wisely. In order to approximate the distribution of the test set well, different approaches might be of interest. The in Section 3 presented subset sampling methods can be taken into consideration for the construction of a validation set as well. A subset, which represents the test set well might be of interest. Since the test set is not a-priori known, one may try to represent the pool of unlabeled samples well. Another alternative is to find a class balanced validation set. Even uncertainty sampling may yield to a better validation set. Nonetheless, these methods are out of the scope of this thesis and might be interesting for future work. In the experiments it is assumed that a randomly selected validation set approximates the test set distribution well.

Weight Initialization The retraining process of line 7 (of Algorithm 4) can be done in one of the two following ways: The network's weights are after every round initialized with the values the same values as in line 2 of the algorithm. Or the network uses the weights of the best previous validation epoch as this can be seen as the probably best weight initialization. The second method is used in the experiments.

Learning Rate Initialization In every round the network has to be retrained on the labeled data (see the Algorithm 4). For the learning rate initialization there exist different options: The learning rate can be initialized with the same value after every round (cyclic learning rate) or it gets the value of the last learning rate of the last round. Taking the value of the last rounds may not converge to a local minimum because the step size may be too small. This is due to the fact that the learning rate is decreasing during the training (at least within a round). For the first setting is the optimization process may overshoot a local minimum but as the learning rate is decreasing during the training it can still end up at a minimum. Thus, a cyclic learning rate is used in the following experiments.

Test Set The problem described in Section 3.4 is also of importance for the testing of the model. Testing a model on an unbalanced data set prefers models, which favor overrepresented classes. As a result the model with the best test accuracy is not able to detect samples of underrepresented classes. An unrepresented class is in Histogram 17a the class Knusperspitz. This data set is balanced by only taking an equal amount of samples of each class for testing. This yields Histogram 17b. Thus, the unbalanced test set from PreciBake is distributed in the same way as the training set, whereas the balanced test set contains now of the same amount of images per class.



(a) **Unbalanced test set** (b) **Balanced test set**

Figure 17: Histograms of an unbalanced and a balanced test set, provided by PreciBake

Training In Figure 18 the validation and training losses are displayed per epoch of the training process. Each round is separated from another by a vertical black line. (This means that at every black line selected samples are added to the training and validation set. In the cases of random selection, those samples are randomly selected. After the training process of a round is finished, the model with the lowest validation loss per epoch is chosen and the model is retrained having this model as its initialization. The learning rate is also reset to its starting value. As the number of rounds increase, so does the number of data points in the training and validation sets. This leads to more iterations per epoch. Thus, the total number of iterations increases with a higher number data points in the set and would therefore overfit earlier. Hence, with more data points in the data sets it is valid to train with less epochs. It can be seen that the difference between the training and validation loss is decreasing with an increasing number of rounds, i.e. more training and validation samples added. Thus, the network is less prone to overfit to the training set with bigger training and validation sets.

4.2 Evaluation of Methods

In this subsection, the methods are evaluated. In Section 4.2.1, two uncertainty sampling methods are evaluated, which is followed by the evaluation of the class balance of the selected subset (Section 4.2.2) and of the labeled data set $\mathcal{L}$. In Section 4.2.2 only methods were included, which directly aim to have a diverse or balanced subset. In Section 4.2.3 all methods from Table 1 are evaluated, except regularization methods, because regularization methods are not query methods (they only regularize the model's parameters). All evaluations are done on the data set provided by PreciBake.

4.2.1 t-distributed stochastic embedding

t-distributed stochastic embedding (tsne) is a technique for dimensionality reduction, which is well suited for the visualization in a low embedded space (e.g. two or three dimensions)

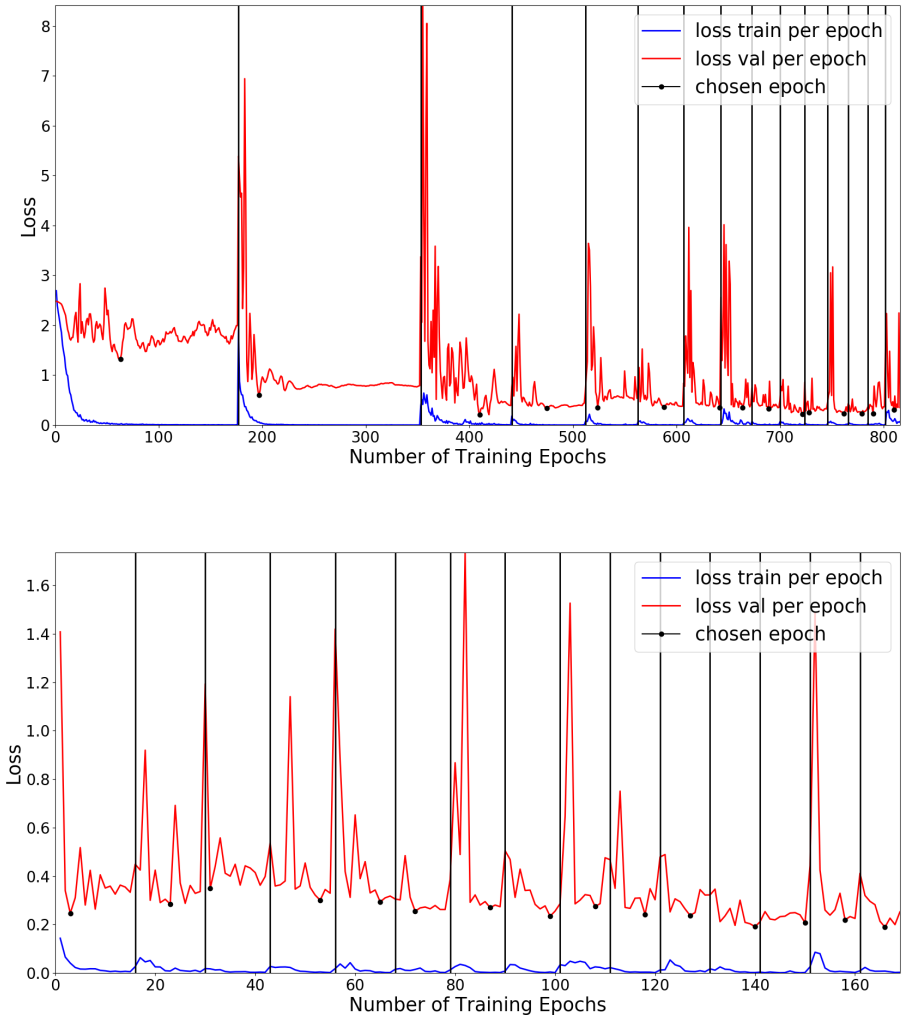


Figure 18: **Training Process with Random Selection:** loss values of training and validation per epoch and round.

of high dimensional embedded data sets (e.g. image data) [44]. It is a variation of the *Stochastic Neighbor Embedding* technique presented by [24] but it produces significantly better visualization results because it is less prone to crowd points in the center of the map. Moreover, it reveals the structure of the data set at many scales [44]. In Figure 19 the application of the tsne technique on feature vectors of all training samples of PreciBake's data set is shown for round one until five. The feature vectors are determined using the model trained on the labeled images after a specific round.

Mutual Information with MC Dropout (Figure 19) After training the classifier on the data selected in the first round, the most samples were selected from the classes Sonnenblumenbrötchen (red, 29) and Schnittbrötchen (lighter red, 30). This can be backed by the facts that there is a lot confusion in the center of the left-hand tsne plot, i.e. clusters of these two different classes are mixed. And regarding the right-hand plot, in

the regions, which are dominated by the named classes, are purple points, which indicate sample points with a high uncertainty. Nonetheless, considering the right-hand tsne plot, one might expect more than two Laugenbrezels being sampled from the method.

After the training of round two, most samples are selected from Ciabatta (20, blue), Schnittbrötchen (19, lighter red) and Baguette (16, dark blue). Since those samples are mixed in the left tsne plot for round two, this result is backed. At the right plot, it can be seen that samples with a high mutual information are all over the plot and from this plot it cannot be indicated which class provides the sample with the highest mutual information. After the training of round three, most samples are taken from Apfelecke (purple, 23) and Schinken-Käse (orange, 11). Both lie at the lower centers of the tsne plots. In this region, there are not many purple dots in the right-hand tsne plot showing the uncertainty. For the class Laugenbreze, the framework selects the second-most samples (green, 13). This can also be observed in the right-hand plot.

The training of round four results in tsne plots, which show among others the classes of Kategorie-Brot (lighter blue, 30), Sonnenblumenbrötchen and Schnittbrötchen being not well separated. Thus, most samples are chosen from those classes. This can be seen in the right tsne plot, as there are more purple plots at the left-hand side of the plot.

The training on the images labeled after round four, yields the most images being chosen from the classes Pizza (lighter orange, 28), Kategorie Brot (lighter blue, 23) and Schnittbrötchen (lighter red, 12). This can be backed by lots of purple plots in the tsne plot on the right-hand side as well as the fact that there is a confusion of different classes at the left-hand side of the left tsne plot.

Shannon Entropy (Figure 20) After the training on the data set with the first chosen samples by the direct uncertainty method, there are, except for Laugenbreze, no class boundaries to detect (see left tsne plot). Surprisingly, Laugenbreze has a lot of samples with a high uncertainty. Nonetheless, only one of the 100 chosen samples is from this class. The most samples were chosen from the class Kategorie Brot (lighter blue, 68 images chosen) even if only some samples are marked purple in the right-hand plot. The second-most samples are taken from the class Sonnenblumenbrötchen (lighter red, 18). For this class more purple dots can be indicated.

For the tsne plot of the feature vectors based on the network trained on samples chosen after round one, it can be stated that the uncertainty is better distributed over the whole set of data, i.e. there are purple points all over the right-hand tsne plot. This is also true for samples of the classes Laugenzopf (lighter green, 32) and Apfelecke (purple, 30), which represent the most chosen samples after round two.

After the training in round three, the most images were selected from the classes Apfelecke (purple, 32) Ciabatta (blue, 25) and Baguette (dark blue, 14). This is backed by the fact that in the region of these blue/purple classes, a lot classes are mixed (see left tsne plot). The right-hand plot also shows many purple dots in this region.

Training on images selected during the first three rounds yields a classifier being mostly uncertain about samples from the classes Pizza (lighter orange, 27) and Ciabatta (blue, 14). At the left-hand tsne plot, it can be seen that Pizza is clustered in a region but

there are also samples from Pizza, which do not belong to this cluster. Those samples are close to samples of other classes and therefore have a high mutual information (right-hand plot). Thus, they are selected by the framework. Images from the class Ciabatta are also mixed with images from other classes and have a high uncertainty on the right-hand plot as well.

The image selection after round five reveals, that the class Apfelecke (purple, 24) has the most samples from the 100 most uncertain samples. Baguette (darker blue, 22) has the second-most. Even if the images are better clustered according to the left-hand plot, there are samples with higher uncertainties at the borders of the named classes.

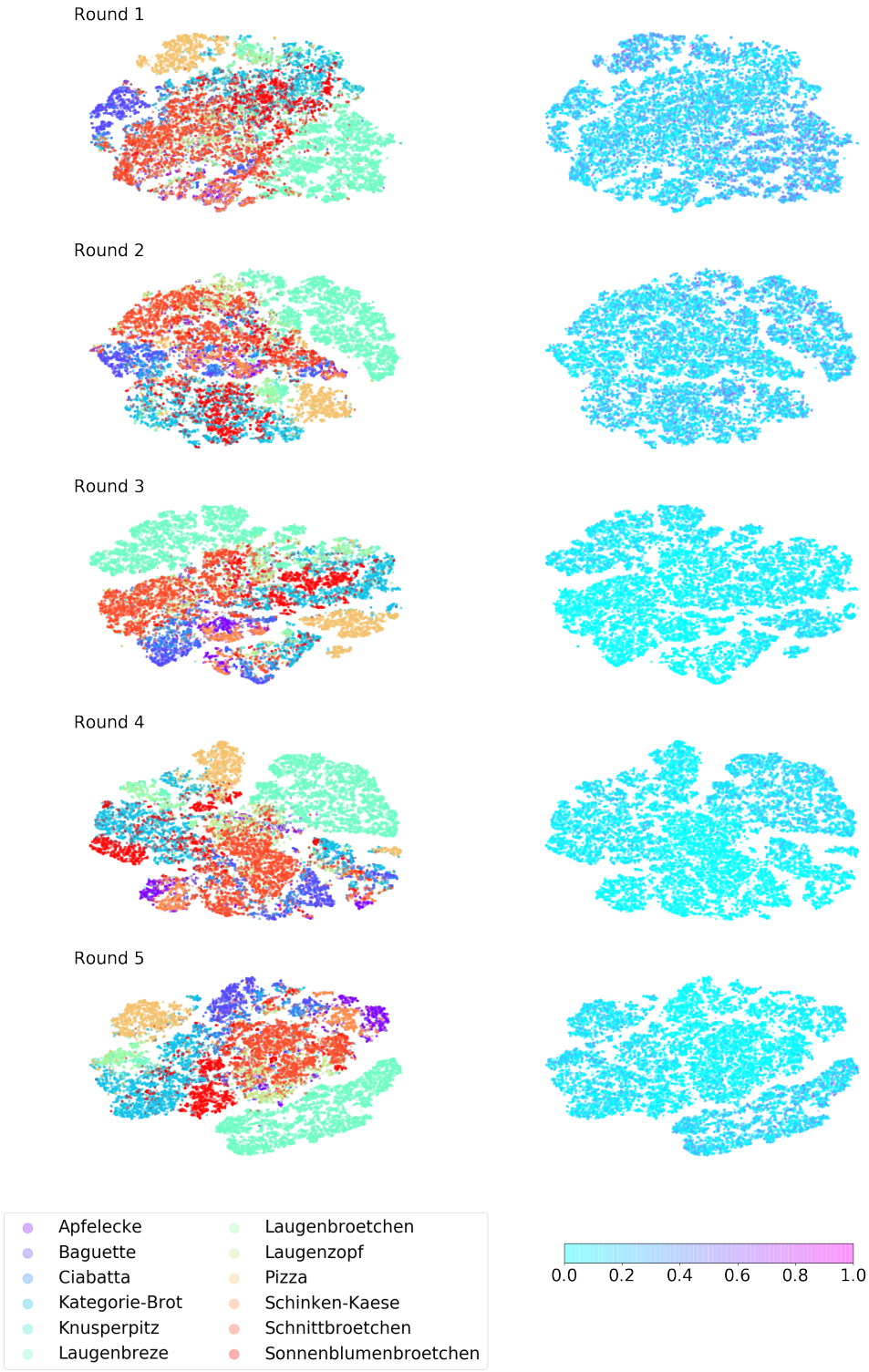


Figure 19: tsne plots showing the subset selection for MC Dropout with **mutual information** for five rounds. Left: tsne plot colored by labels, right: tsne plot colored by uncertainties.

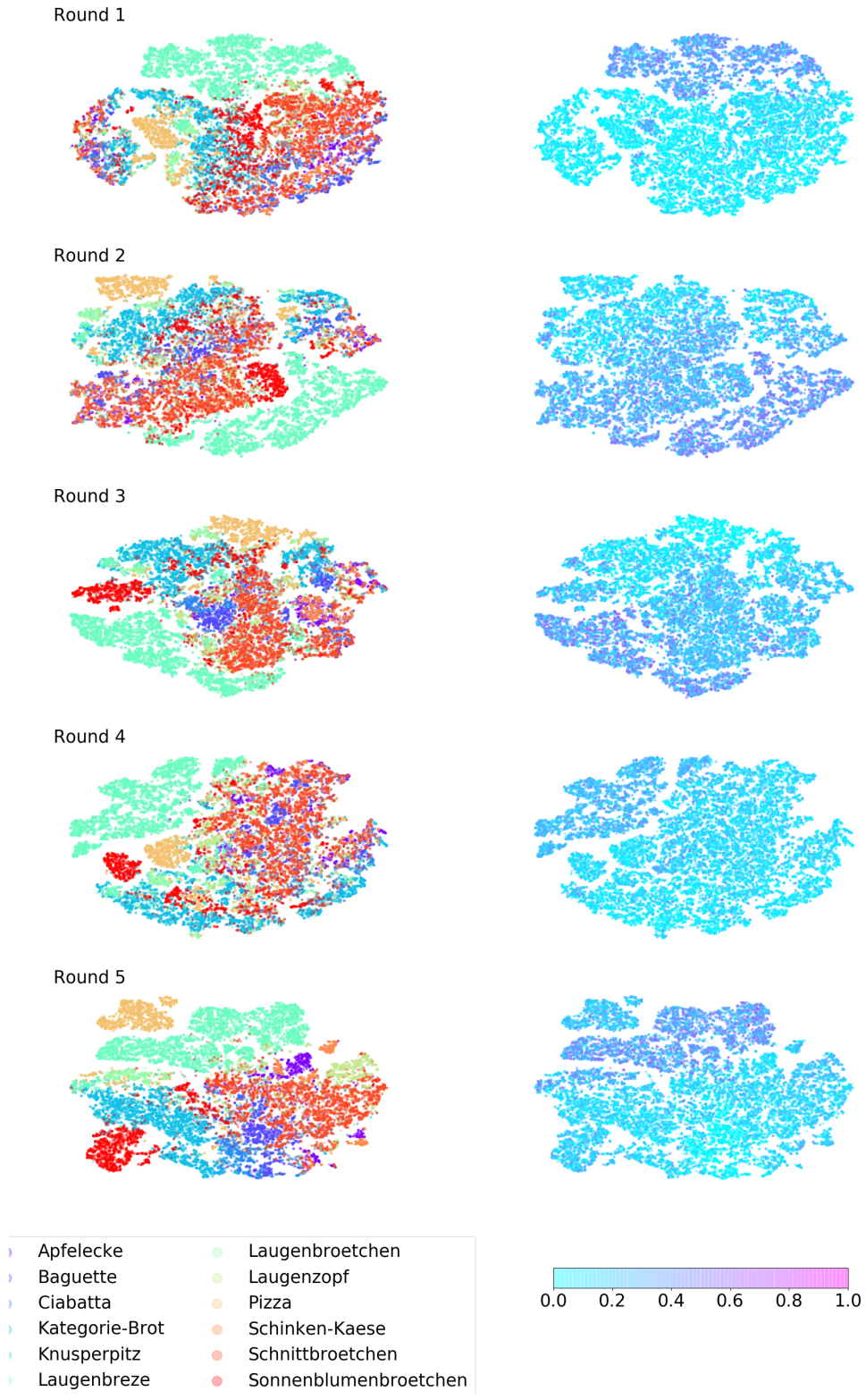


Figure 20: tsne plots showing the subset selection for direct uncertainty with **Shannon Entropy** for five rounds. Left: tsne plot colored by labels, right: tsne plot colored by uncertainties.

4.2.2 Minimal number of images per class

This evaluation method visualizes how well the considered selection method can balance the selected subset of unlabeled images, which is added to the labeled set. The minimum amount of images per class in each selected set is displayed on the y-axis where on the x-axis is the number of rounds is displayed. The maximal amount of images in the class with the lowest amount is for data set provided by PreciBake given by $\lfloor \frac{k}{C} \rfloor = \lfloor \frac{100}{12} \rfloor = 8$ with $k = 100$ being the number of samples, which is annotated per round and $C = 12$ is the number of classes.

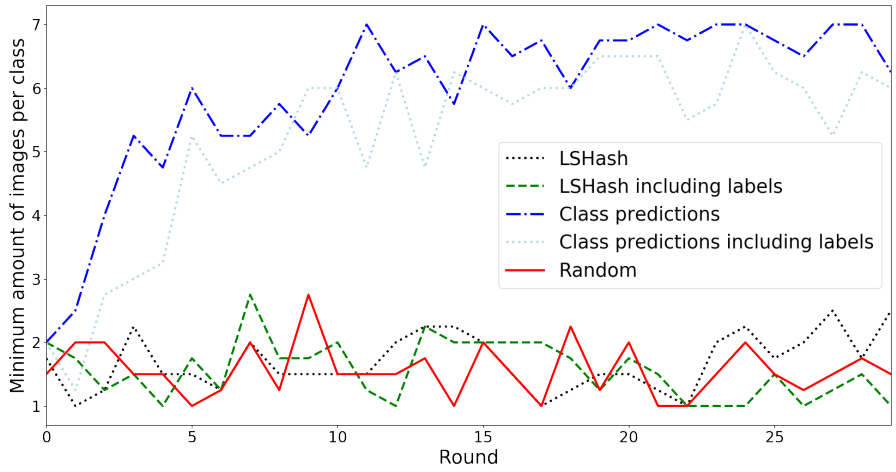


Figure 21: **Class balance** of the selected **subset** of unlabeled data, selected by **class balance methods**. Both class prediction methods outperform random subset selection and both LSHash methods by four to five images per round. Class prediction adds after eight rounds between five and seven images per round to the class with the lowest amount of images. By a possible maximum of eight images for the lowest class, this method adds an almost perfectly balanced subset to the labeled set, which is the aim of this method. Both LSHash methods do not have an effect on the minimal number of images per class. The effect of both methods which include the already labeled set $\mathcal{L}$ is slightly smaller since it does not only consider the subset.

Observing Figure 21, it can be seen that both class prediction methods yield a more balanced selected subset. Since class prediction only considers the subset and not the labeled data, it achieves higher results for this evaluation method. Thus, the methods perform as intended. For LSHash, there is no big difference to see in the amount of images for the smallest class in comparison to random sampling.

In Figure 22, it can be seen that especially diversity sampling but also representation sampling have higher peaks than random subset selection. Thus, these methods lead in specific rounds to a better balanced selected subset than a randomly selected one.

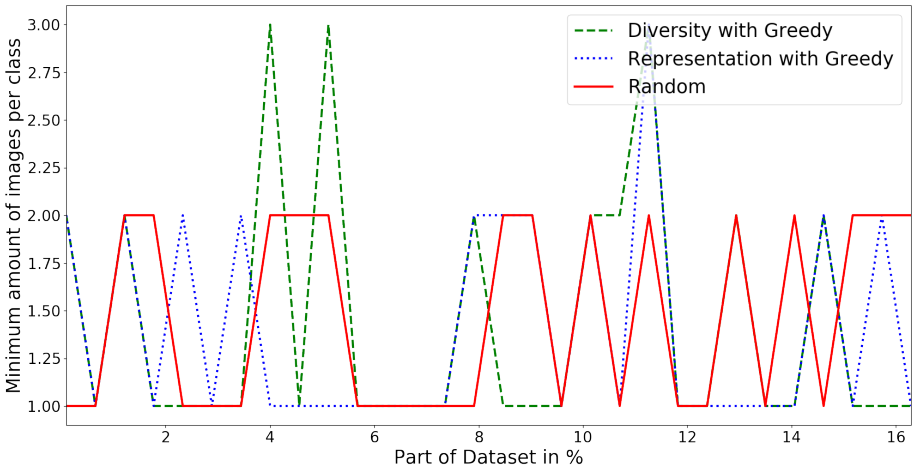


Figure 22: **Class balance** of the selected **subset** of unlabeled data, selected by **representation and diversity methods**. The diversity selection methods has higher peaks than random subset selection as well as representativeness subset selection. Therefore, all in all, the minimum number of images per class is slightly higher for diversity selection.

4.2.3 Class Balance

This evaluation visualizes how the presented methods balance the set of data on which the classifier is trained. The mean of the outcomes of the four different seeds is here visualized (except for the diversity/representation methods, where only the first seed was used). If a data set achieves a balance of 100%, the class with the smallest amount of images has as many images as the one with the most. A class balance of 0 means that the smallest class does not posses any image. All methods are evaluated only on the data set provided by PreciBake since it is unbalanced.

First, the class balance methods are evaluated and the expected outcome can be observed (see Figure 23). I.e. the methods, which take the already labeled set into consideration achieve a higher class balance. Moreover, the class prediction methods achieve a higher class balance than the LSHash methods, as class prediction aims to cluster the selected set into the different classes whereas LSHash only aims to cluster the samples.

Considering the class balance of the direct uncertainty sampling methods (see Figure 24), both methods achieve a higher class balance after the third round of training than random sampling achieves. After round 10, both methods converge to a class balance of 50%. Both methods do not differ much in terms of the class balance.

To the class balance of the uncertainty by committee sampling methods (see Figure 25 and Figure 26) it can be said that all methods achieve higher class balances than random sampling for mutual information as acquisition function while MC Batchnorm and MC Dropout only achieve lower class balances after round nine or 19. The MC Ensemble methods achieve almost equal class accuracies for both acquisition functions. But except MC Ensemble Cyclic, all methods seem to converge to the same class balance as random

sampling (around 40 percent).

The class balances of the methods using the reconstruction error or the VAAL framework (see Figure 27), are only between 3 and 15 rounds higher than the of random sampling. Afterwards, they seem to converge to the same value (40 percent).

The diversity and representation subset selection methods jump to its highest class balance and keep this balance afterwards (see Figure 28). But since the methods were only tested using one seed, it cannot be said much about the resulting class balance.

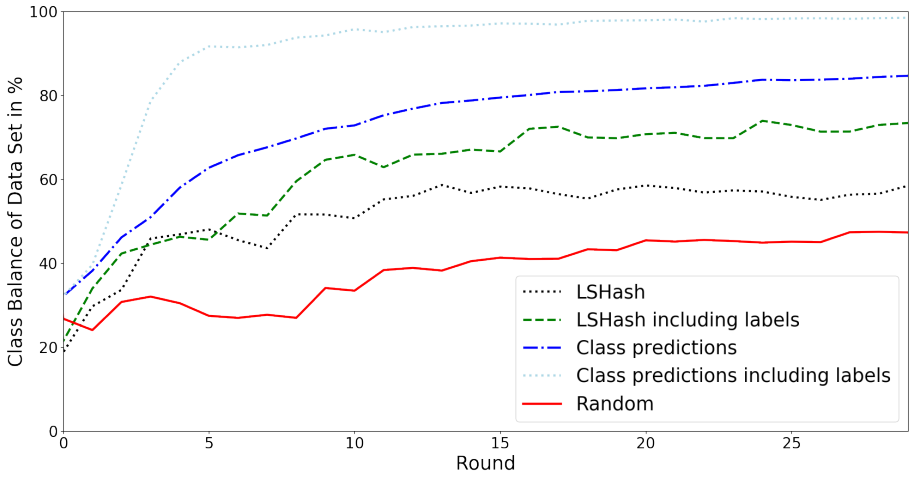


Figure 23: Comparison of the **class balance** of the **class balance** sampling strategies and random sampling. Class predictions, which takes the labeled set $\mathcal{L}$ into consideration, yields an almost perfectly balanced labeled set after 10 rounds of training. Class prediction, which does not consider the labeled set, achieves a class balance of 80% after 30 rounds. Both LSHash methods lead to a better class balance as random subset selection. LSHash achieves after 30 rounds a class balance of 50% and LSHash considering $\mathcal{L}$ of 70%. This is the expected result.

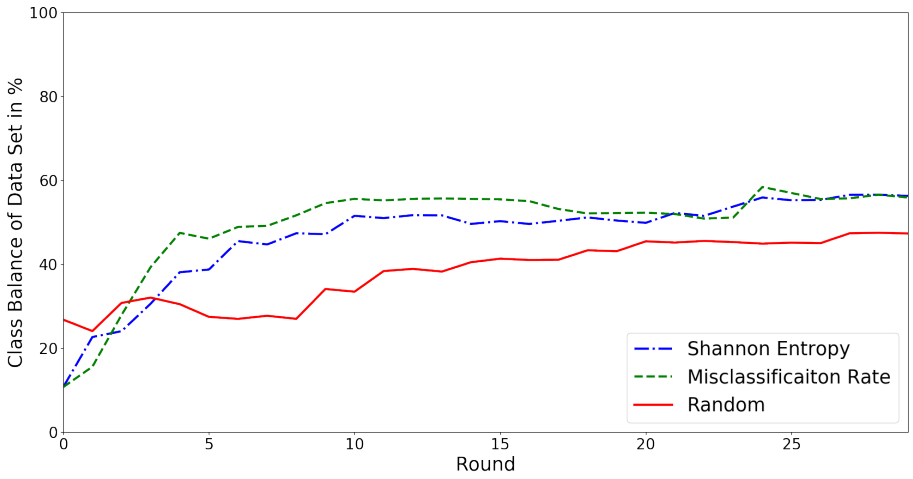


Figure 24: Comparison of the **class balance** of the **direct uncertainty** sampling strategies and random sampling. Both direct uncertainty selection methods yield a higher balanced data set. Misclassification rate yields to the highest class balance between round two and 20. Afterwards, both methods are more or less equal.

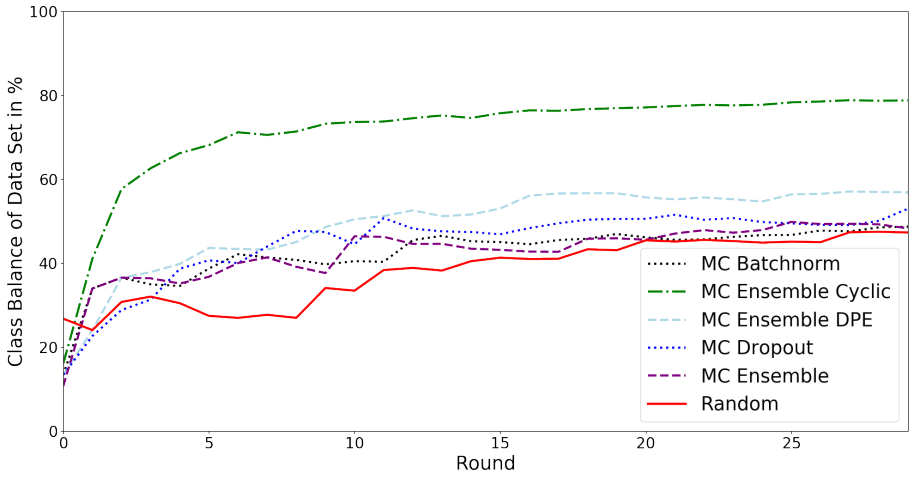


Figure 25: Comparison of the **class balance** of the uncertainty by committee sampling strategies using **mutual information** and random sampling. MC Dropout sampling has after seven rounds a higher class balance then MC Batchnorm and random sampling, MC Batchnorm is after seven rounds higher than random selection. After 20 rounds all methods, except MC Ensemble Cyclic and MC Ensemble DPE seem to converge to the same class balance. MC Ensemble Cyclic yields the highest class balance.

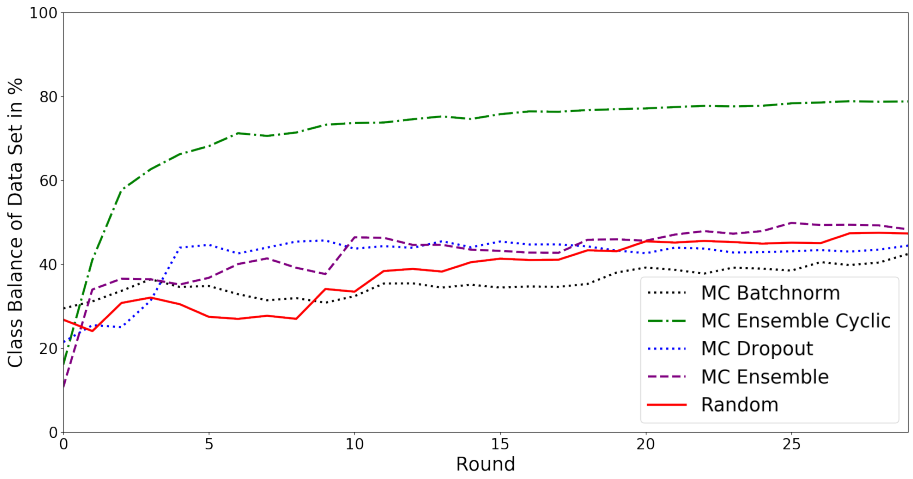


Figure 26: Comparison of the **class balance** of the uncertainty by committee sampling strategies using the **epistemic uncertainty** of the predictive variance and random sampling. MC Batchnorm has a higher class balance than random selection until nine rounds and MC Dropout until 19 rounds. Afterwards both methods yield a less balanced data set but seem to converge to the same class balance. MC Ensemble and MC Ensemble Cyclic both achieve almost the same values for the class balance as for mutual information. Moreover, both yield after round one a higher class balance than random sampling.

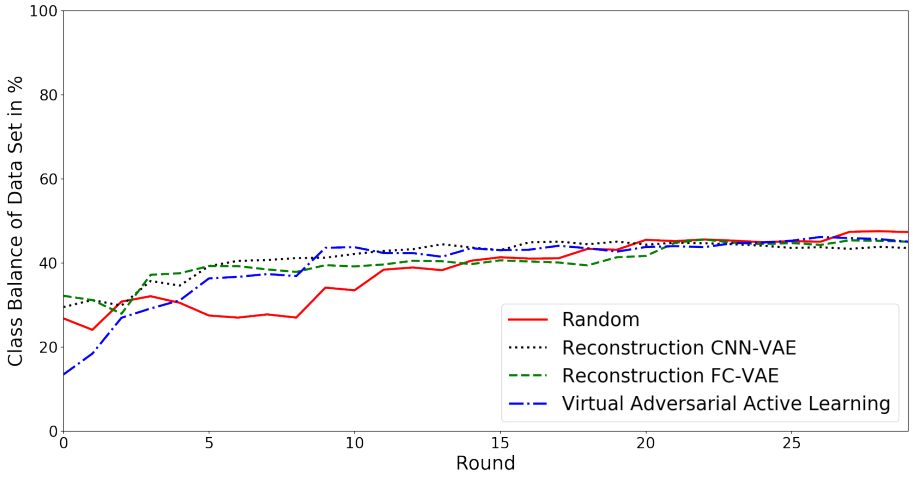


Figure 27: Comparison of the **class balance** of the sampling strategies using the **reconstruction error** of different artificial neural network architectures, the VAAL sampling method and random sampling. Until 14 rounds (FC-VAE), 17 rounds (VAAL) and 19 rounds (CNN-VAE) all methods yield a more balanced labeled data set than random selection. Afterwards, the methods yield a slightly less balanced labeled set but tend to converge to the same class balance.

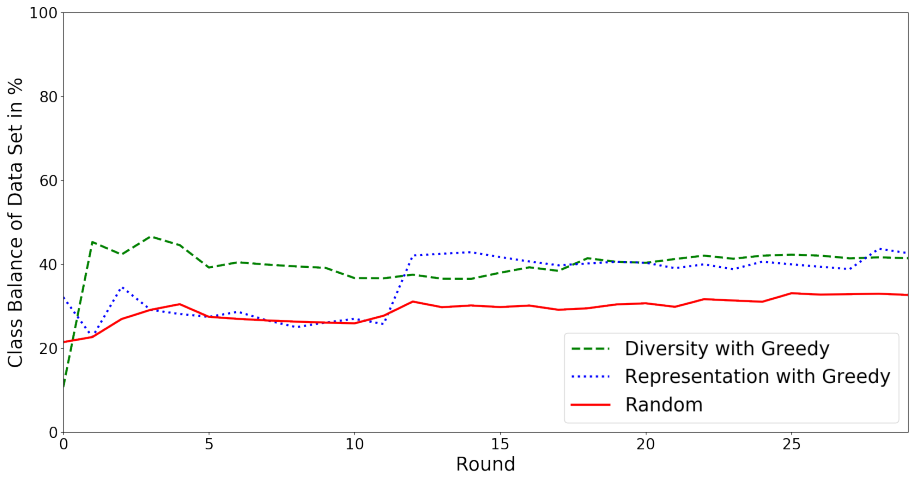


Figure 28: Comparison of the **class balance** of the **representation** and **diversity** sampling strategies and random sampling. Diversity selection jumps after one round to its maximal balanced data set and the balance remains more or less stable after this round. The same happens for representation selection after 12 rounds. Both methods yield after 17 rounds equally balanced labeled sets while both yield a more than 10% more balanced labeled set than the set generated by random selection.

4.3 Results

In this subsection, learning curves are shown for the methods from Table 1 and both data sets. First, the learning curves are shown for the methods on the data set provided by PreciBake for an unbalanced and a balanced test set. To confirm the results, the methods were, if they achieved good results on the data set provided by PreciBake, tested on the CIFAR10 set as well. Thus, such learning curves are in such cases shown after those on PreciBake's data set. At the end of this section, the results are summarized in a table.

Learning curves were originally proposed by [69] and they "provide a mathematical representation of the learning process that takes place as task repetition occurs." [3] Here, the repeated task is to classify the images from the test set while the network is trained on a gradually increasing amount of labeled samples. Thus, the learning curve represents the test accuracy with respect to the portion of the data set, which is labeled. If the learning curve intersects the horizontal line, which represents the test accuracy of the training on the whole data set, it means that the method has selected a subset, which has an equally good test accuracy. Thus, having intersected this line means that a solution for the problem is found. Random sampling can be seen as the way the pool of unlabeled data was constructed since the samples are randomly selected from this pool. Therefore, a learning curve, which is above the learning curve of random sampling, means the method constructs a better set.

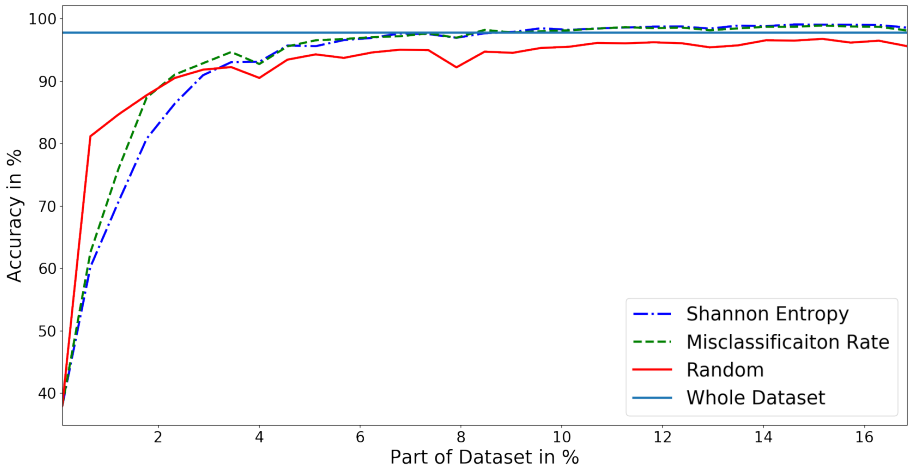
4.3.1 Direct Uncertainty Sampling

In this experimental setup random sampling is compared to sampling, which takes the instances, which have the highest uncertainty. The uncertainties are computed using the output of the artificial neural network with the misclassification rate or Shannon entropy as acquisition function. This method is described in Section 3.3.1.

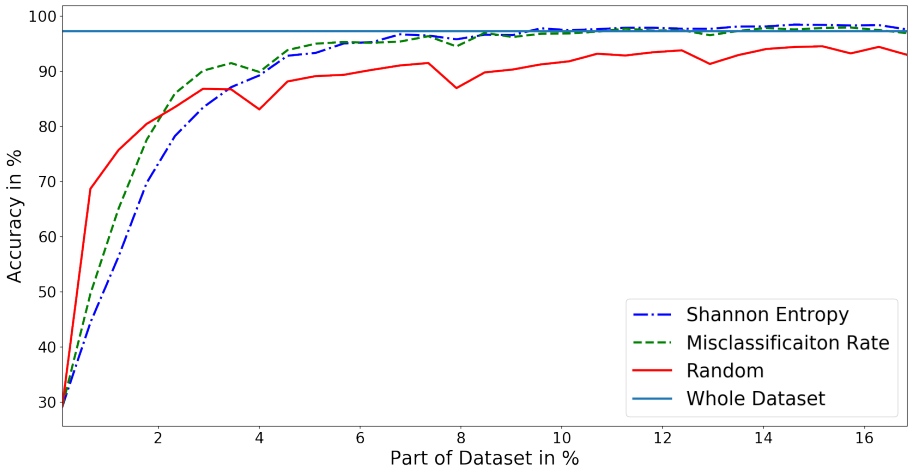
As Figure 30 shows, sampling by using the misclassification rate achieves a higher accuracy than random sampling after two percent of the data being labeled and Shannon entropy achieves this after less four percent for the unbalanced test set. For the balanced test set both methods achieve a higher accuracy after two percent. A higher or equal accuracy is achieved compared to the training on the whole data set after seven percent for misclassification rate and for Shannon entropy after eight percent for the unbalanced data set and for the balanced after 10 and 8.5 percent respectively.

Both acquisition functions yield to an improvement compared to random sampling. The misclassification rate seems to be better than Shannon entropy at the beginning of the training, while Shannon entropy is later better. Both methods perform nonetheless worse than random sampling in the first rounds.

Figure 29: Comparison of Random Sampling and **direct uncertainty sampling** using Shannon entropy and misclassification rate as acquisition functions. The method using misclassification rate has a higher test accuracy than random sampling after five percent of the data is labeled whereas the method using Shannon entropy achieves this after ten percent. After twelve percent, Shannon entropy is slightly better than misclassification rate. They keep achieving higher test accuracies until 60 percent of the data is labeled but do not achieve a test accuracy, which is as high as training on the whole labeled data set.



(a) Unbalanced



(b) Balanced

Figure 30: Comparison of Random Sampling and **direct uncertainty sampling** using Shannon entropy and misclassification rate as acquisition functions. Using the **unbalanced test set**, the method using the misclassification rate is until four percent of the data set are labeled better than Shannon entropy. After three percent both methods yield a higher test accuracy than random sampling. And after eight percent both methods are better than or equal to training ont the whole labeled data set. For the **balanced test set**, the difference in the accuracy of the methods and random sampling is bigger. Again, both methods are after three percent better than random sampling, but here they equal the test accuracy of the whole data set after eleven percent.

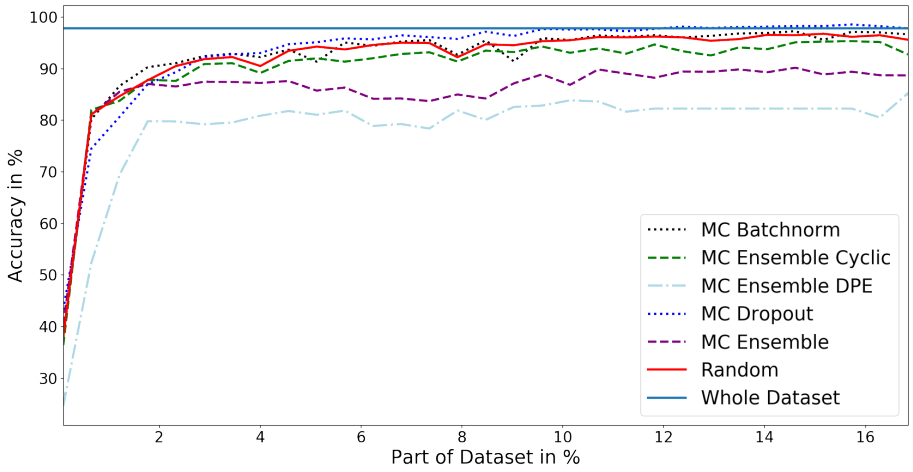
4.3.2 Uncertainty by Committee Sampling

The images with the highest uncertainty are sampled according to measurements described in Section 3.3.2. Namely, the epistemic uncertainty of the entropy (mutual information)

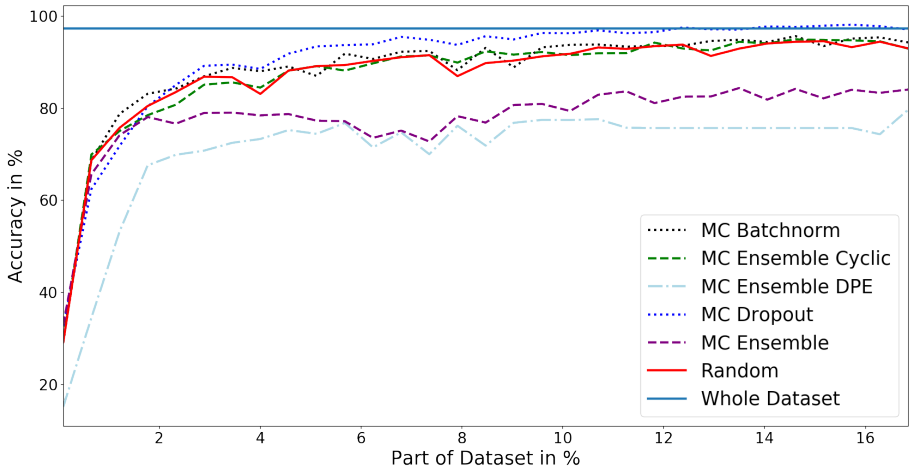
and the epistemic uncertainty of the predictive variance are used as acquisition functions. For both sampling methods the test accuracy of a classification network is compared between different ways to approximate a Bayesian neural network using the given classification network and with random sampling. Finally, the accuracy is compared to the test accuracy if the classification network is trained on the whole data set.

Mutual Information In Figure 31 it can be seen, for the unbalanced test set of PreciBake, that the approximation with MC Batchnorm is after one percent of the labeled data set better than the other methods. After three percent MC Dropout is the highest and after ten percent equal to the training on the whole data set. MC Batchnorm is after three percent equal to random sampling, while the ensemble methods are worse than random sampling. For PreciBake's balanced test set, the difference between the accuracy of MC Dropout and random sampling is bigger, but MC Dropout reaches the accuracy of the training on the whole data set not before 13 percent. MC Ensemble Cyclic is now reaching an equal accuracy as random selection, whereas the difference to MC Ensemble is even bigger. And for the balanced CIFAR10 data set (Figure 32), the method using MC Ensemble as a Bayesian neural network approximation is, in terms of the test accuracy, again performing much worse than random sampling, whereas MC Dropout and MC Batchnorm perform better than random sampling. MC Dropout's accuracy is more stable than the one from MC Batchnorm and all in all a bit higher. No method with a labeled data set smaller than 60 percent is better or equal to the training on the whole data set.

Predictive Variance Using the epistemic uncertainty of the predictive variance, the following can be said about its performance on the data set provided by PreciBake (see Figure 33):On the unbalanced test set, MC Dropout achieves a slightly higher accuracy than random sampling since two percent of the data being labeled, while MC Batchnorm is more or less equal to random sampling and MC Ensemble Cyclic performs a bit worse. MC Ensemble however performs a lot worse than random sampling. For the balanced test set, the difference between MC Dropout and random sampling is bigger. MC Ensemble Cyclic is, as in the case of mutual information as acquisition function, now performing more or less equal to random sampling. MC Ensemble is again performing even worse. To the performance on the CIFAR10 set, it can be said that only the approximation using Batch Normalization (MC Batchnorm) is yielding a better test accuracy than random sampling (since four percent). MC Dropout performs slightly worse, whereas MC Ensemble performs a lot worse (see Figure 34).



(a) Unbalanced



(b) Balanced

Figure 31: Comparison of random sampling and sampling using **mutual information** as acquisition function and different Bayesian neural network approximations on PreciBake's data set. For the **unbalanced test set**, the approximation with MC Batchnorm is after one percent of the labeled data set better than the other methods. After three percent MC Dropout is the highest and after ten percent equal to the training on the whole data set. For the **balanced test set**, the difference between the accuracy of MC Dropout and random sampling is bigger, but MC Dropout reaches the accuracy of the training on the whole data set not before 13 percent.

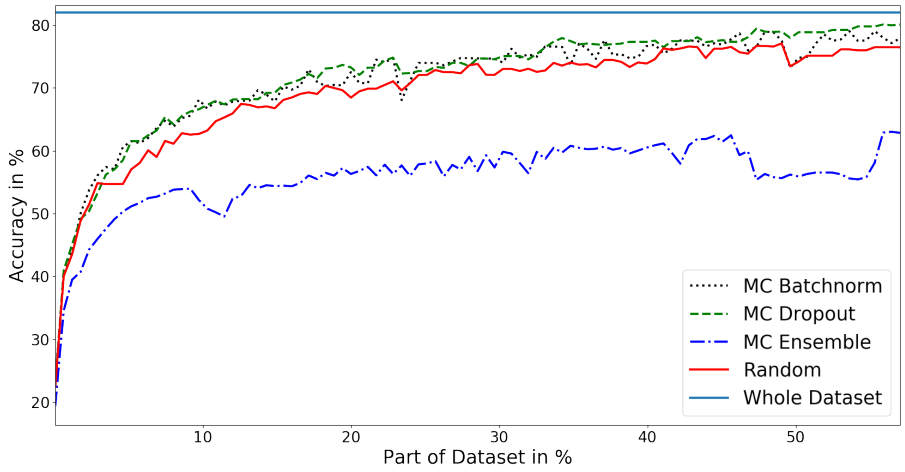
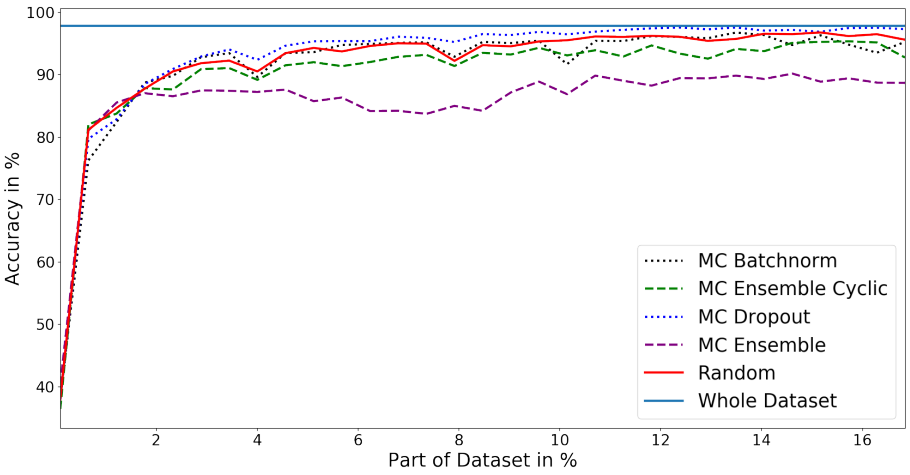
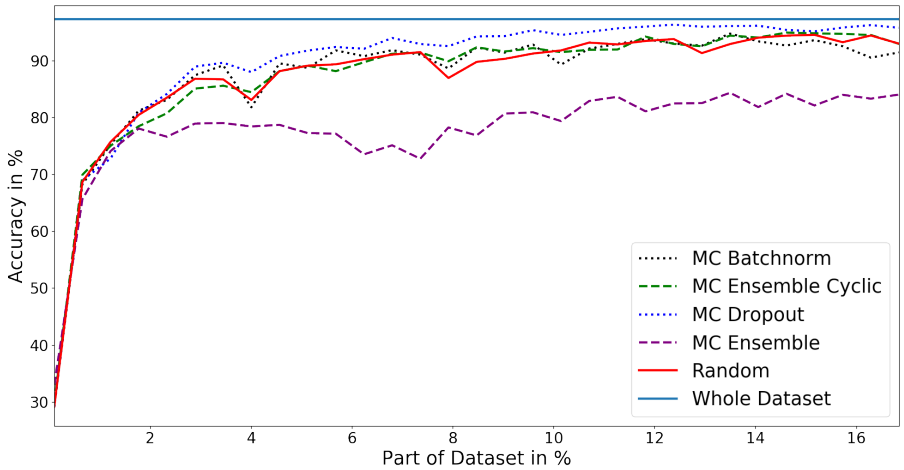


Figure 32: Comparison of random sampling and sampling using **mutual information** as aquisition function and different Bayesian neural network approximations on the CIFAR10 data set. The method using MC Ensemble as a Bayesian neural network approximation is, in terms of the test accuracy, again performing much worse than random sampling, whereas MC Dropout and MC Batchnorm perform better than random sampling. MC Dropout's accuracy is more stable than the one from MC Batchnorm and all in all a bit higher. No method with a labeled data set smaller than 60 percent is better or equal to the training on the whole data set.

From the methods presented for uncertainty by committee sampling, MC Dropout was the Bayesian neural network approximation with the highest test accuracies. Thus, it can be assumed that this method approximates a Bayesian neural network better than the other methods. Compared to the presented Ensemble methods, an explanation is that the committee is larger for MC Dropout and MC Batchnorm (50) than for MC Ensemble (8), MC Ensemble Cyclic (8) or MC Ensemble DPE (4). MC Dropout and MC Batchnorm achieved higher test results with mutual information as acquisition function. Therefore, MC Dropout with mutual information can be considered to be the best method of the given ones.



(a) Unbalanced



(b) Balanced

Figure 33: Comparison of random sampling and sampling using the **epistemic uncertainty of the predictive variance** with different Bayesian neural network approximations on the data set provided by PreciBake. On the **unbalanced test set**, MC Dropout achieves a slightly higher accuracy than random sampling since two percent of the data being labeled, while MC Batchnorm is more or less equal to random sampling and MC Ensemble Cyclic performs a bit worse. MC Ensemble is however performing a lot worse than random sampling. For the **balanced test set**, the difference between MC Dropout and random sampling is bigger.

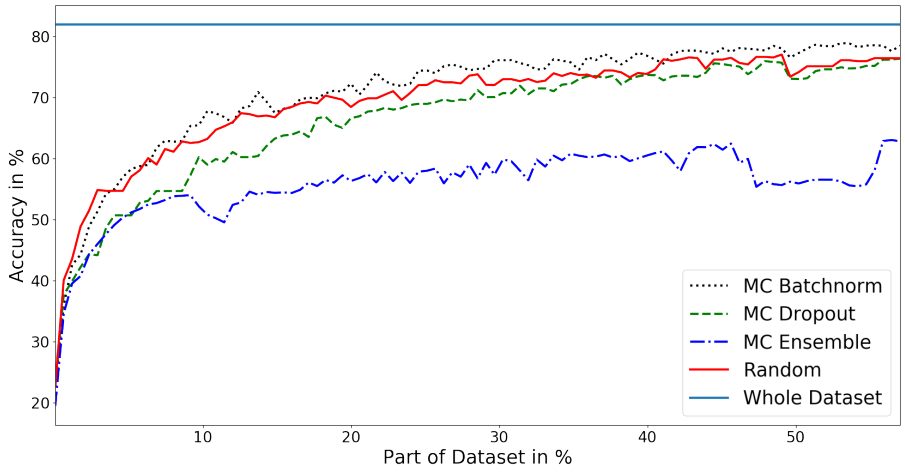


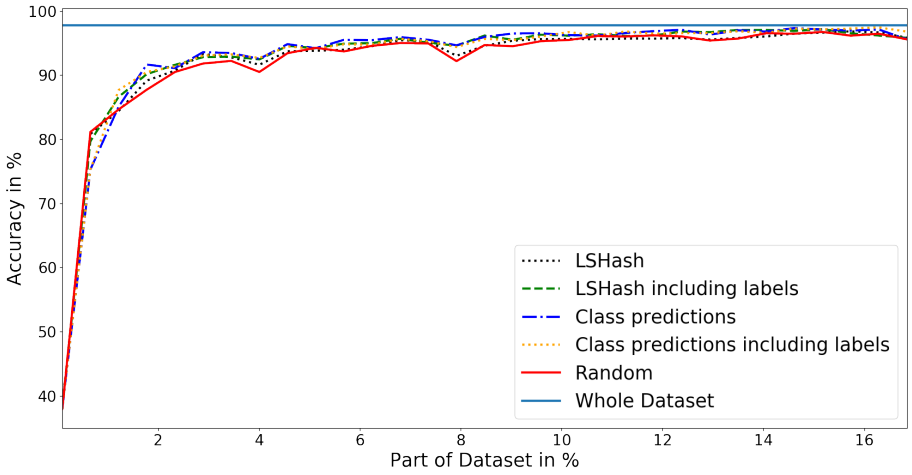
Figure 34: Comparison of random sampling and sampling using the **epistemic uncertainty of the predictive variance** with different Bayesian neural network approximations on the CIFAR10 set. Only the approximation using Batch Normalization (MC Batchnorm) is yielding a better test accuracy than random sampling (since four percent). MC Dropout performs slightly worse, whereas MC Ensemble performs a lot worse.

4.3.3 Class Balance

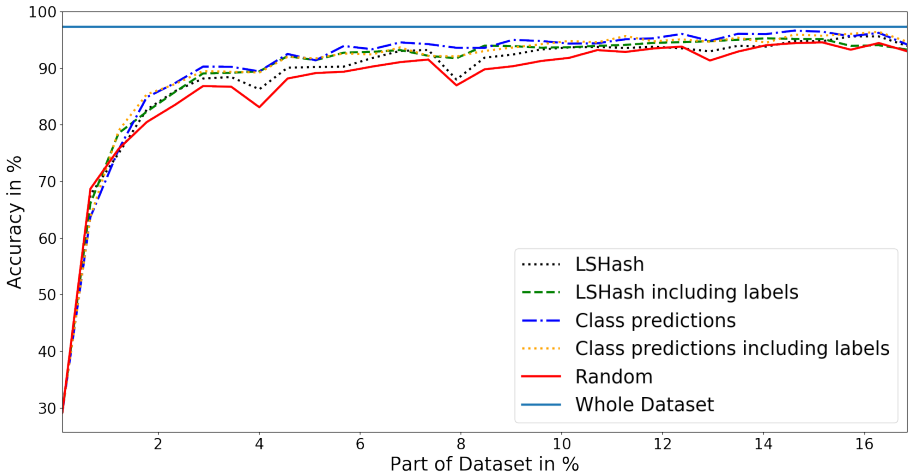
In this experiment, the sampling methods, which aim to sample a class balanced set or to yield a class balanced labeled set $\mathcal{L}$, are compared with the random sampling method as well as with the training on the whole data set. The class prediction method describes clustering by a prediction of the corresponding class label, whereas the clustering is done for the LSHash methods by the corresponding LSHashs. "including labels" means that the LSHashs of the labeled instances are considered first.

Figure 35 displays their respective test accuracies according to the part of the data set on which the model is trained on for the balanced an unbalanced test set from PreciBake. It can be seen that each method yields a slightly higher test accuracy (with "class prediction" higher than "class predictions including labels" higher than "LSHash including labels" higher than "LSHash") for the unbalanced test set. On the balanced set these differences in the test accuracies are intensified.

Due to the fact that the CIFAR10 data set is already balanced, the presented methods do not lead to a better performing network. The class predictions method, which takes the already labeled samples into consideration, even performs worse than random sampling. This can be seen in Figure 36.



(a) Unbalanced



(b) Balanced

Figure 35: Comparison of random sampling and **class balance** sampling on the data set provided by PreciBake. For the **unbalanced test set**, all methods achieve, after the first percent of data added to the labeled set, a slightly higher test accuracy than random sampling. For the **balanced test set**, this effect is even higher. In general, it can be concluded that for both test sets class prediction is a bit better than class prediction, which considers the already labeled data, which is better than the method using LSHash. The method using LSHash performs worse than the one which considers the already labeled samples.

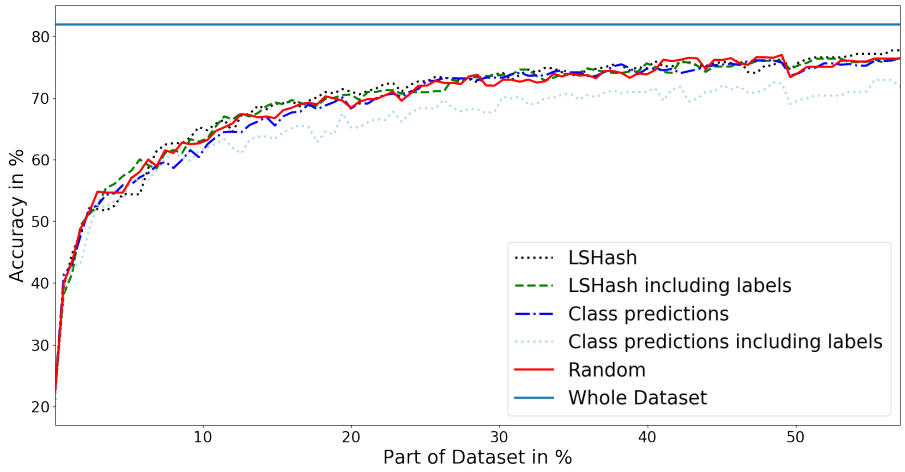


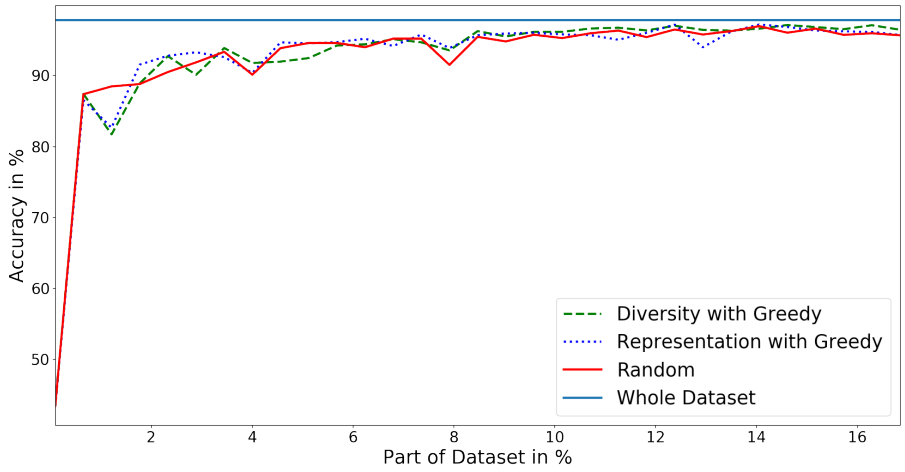
Figure 36: Comparison of random sampling and **class balance** strategies on the CIFAR10 set. On the CIFAR10 test set, the methods, except class prediction taking the labeled data into consideration, do not really seem to differ from random sampling in terms of the test accuracy. This method performs worse than random sampling.

4.3.4 Representation Sampling

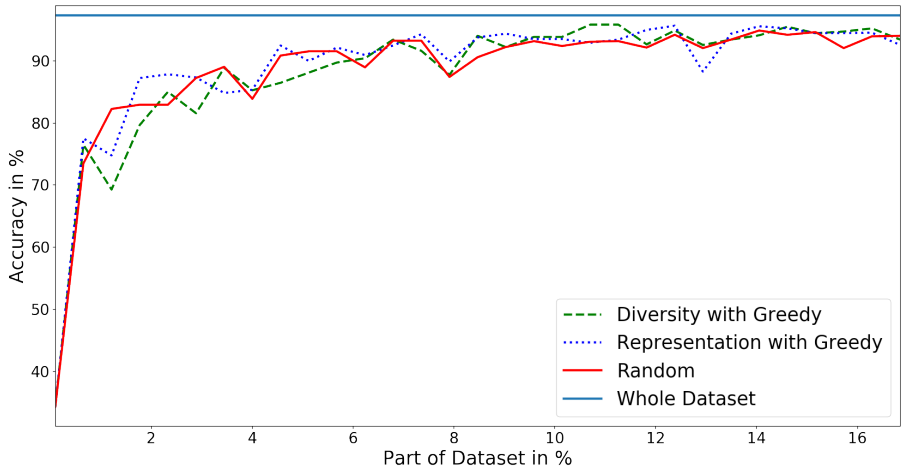
In this experiment, only the first seed was used since the methods took too much time to run. The representation utility function of Equation 3.11 and the diversity utility function of Equation 3.13 are used in combination with the greedy algorithm (Algorithm 6) to approximate the optimal solution of the k-Center problem.

In Figure 37 there are, for both test sets, no big differences in the accuracy of the two models in comparison to random sampling to detect. Diversity selection seems to perform slightly better for more than eight percent of the data being labeled, whereas representation selection seems to perform slightly better between two and ten percent of PreciBake's data set being labeled.

Due to the long sampling times, there were not tested any greedy k-Center method on the CIFAR10 data set. And due to the results given by PreciBake's data set, this does not seem to be necessary.



(a) Unbalanced



(b) Balanced

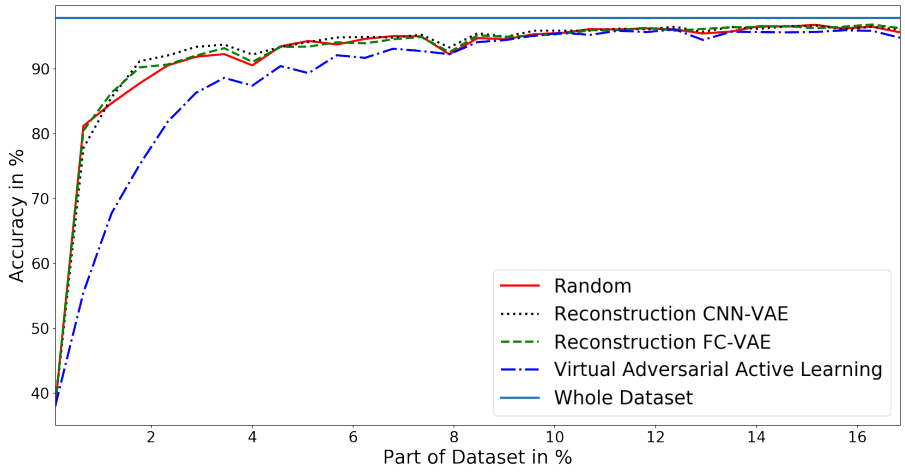
Figure 37: Comparison of random sampling and **diversity** and **representation** sampling methods on the data set provided by PreciBake. The diversity subset selction method has after eight percent of the data set a slightly better test accuracy for the **unbalanced test set** than achieved by the random sampling method. The representation sampling method is more or less equal to random sampling. This is also observed for the **balanced test set**.

4.3.5 Reconstruction Error

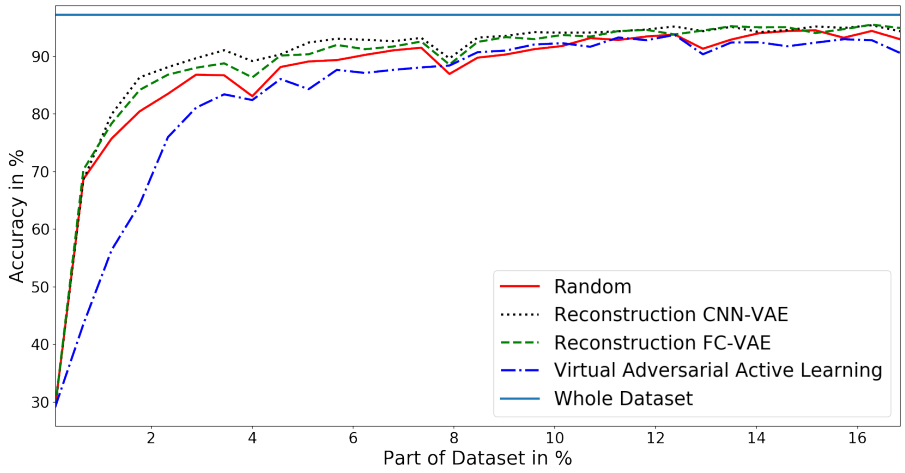
Two reconstruction methods are, in this experiment, compared with the VAAL method [60] and random sampling. For "Reconstruction FC-VAE" the network A.4 is used. For the VAAL method as well as for "Reconstruction CCN-VAE" is the network A.5 used in order to reconstruct the unlabeled samples.

Figure 38 displays that for the data set provided by PreciBake, the best result is achieved by the reconstruction method using a convolutional neural network. Comparing this to the results of the reconstruction method using a fully connected neural network, the difference can be explained by the fact that convolutional neural networks are better suited for the task of image reconstruction and the considered neural network does also have more learnable parameters than the considered fully connected neural network. The VAAL framework performs in terms of the achieved accuracy on a test set worse than expected as it does not even outperform the random sampling method. Thus, the subset is worse sampled than the randomly sampled subset. To achieve a better performance, the hyperparameters of the VAAL framework may have to be tuned more. This explanation is true for tests on the unbalanced and balanced data set provided by PreciBake, but the described effects are even stronger on the balanced test set.

The methods, which take the reconstruction error into consideration, are as well tested on CIFAR10 (see Figure (39)). Surprisingly, the opposite effect can be observed. Namely, the reconstruction error sampling method, which is based on a convolutional neural network was in general worse on the test set than the method based on a fully connected network, which achieved worse test results than random sampling.



(a) Unbalanced



(b) Balanced

Figure 38: Comparison of random sampling and sampling methods using the **reconstruction error** of Variational Autoencoders on the data set provided by PreciBake. For the **unbalanced test set** a small improvement of the test accuracy between 0.5 percent and four percent. Afterwards, there is no difference to notice. For the **balanced test set** there is a difference between the reconstruction methods and random sampling until the 16 percent of the training. This difference is bigger than the difference noted for the unbalanced test set. The test accuracy is for the method using a convolutional variational autoencoder higher than for the method using a fully connected variational autoencoder. This is especially true for the first six percent of the test sets. The VAAL framework could not yield better test results than random sampling.

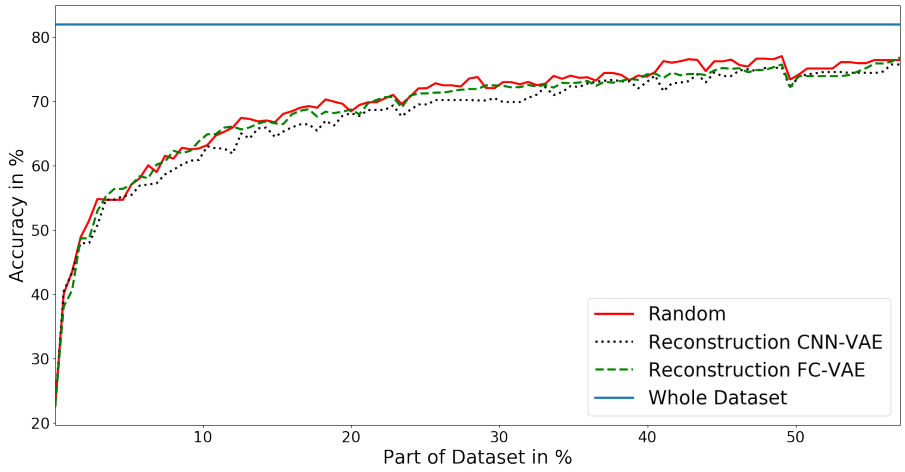


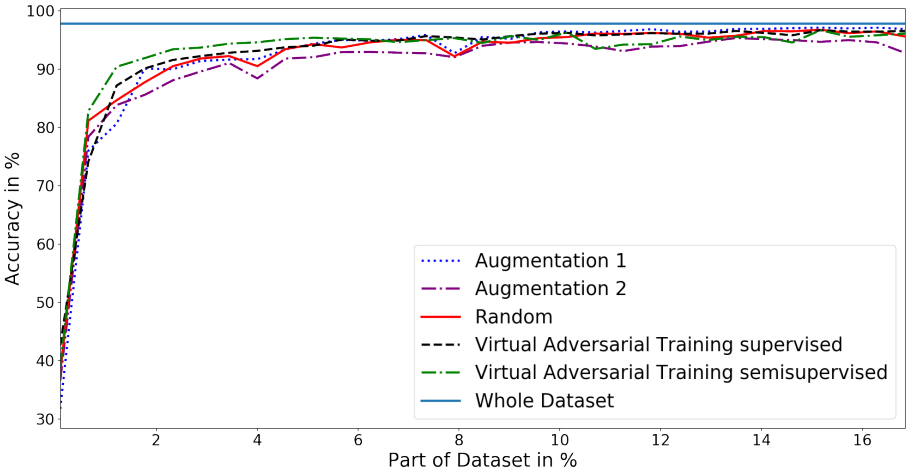
Figure 39: Comparison of random rampling and sampling methods using the **reconstruction error** of Variational Autoencoders on the CIFAR10 set. Here, the opposing effect can be noted. Namely, the reconstruction error samplig method using a variational autoencoder consisting of convolutional layers (CNN-VAE) performs worse than the method using a varitional autoencoder consisting only of fully connected layers (FC-VAE), which achieves a lower test accuracy than random sampling.

4.3.6 Regularization

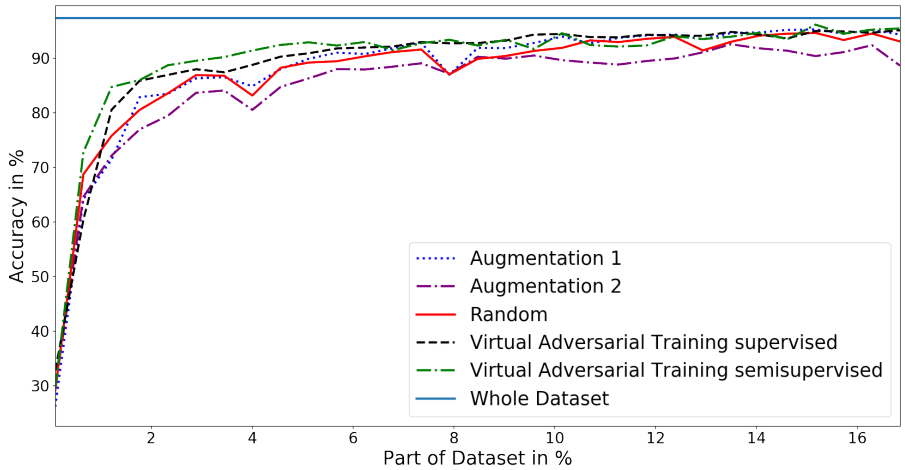
The classifier's test accuracies of different regularization methods for random sampling (Virtual Adversarial Training and augmentation) are in this experimental setup compared with random sampling using no regularization method. Two different Virtual Adversarial Training methods are used. The first one aims to find the adversarial direction using the unlabeled instances and the label instances (semi-supervised) while the second only considers the labeled instances (supervised) (see Section 3.7). Moreover, two different augmentations are tested.

In Figure 40 it can be seen that the Virtual Adversarial Training methods achieved both a higher accuracy until 10 percent of the data being labeled on PreciBake's data sets. Afterwards random sampling performs equally well considering both test sets. The augmentations did not lead to an improvements in terms of the test accuracy. It is important to note that especially at the beginning of the training, when only one or two percent of PreciBake's data set is used, the VAT methods and in particular the one using both labeled and unlabeled images, are learning faster (i.e. achieve a higher test accuracy).

On the CIFAR10 set (see Figure 41) all regularization methods except the first augmentation method (Augmentation1) achieved a lower test accuracy than random sampling. Augmentation1 only achieved more or less equal results than random sampling.



(a) Unbalanced



(b) Balanced

Figure 40: Comparison of random sampling and random sampling with different **regularization methods** on the data set provided by PreciBake. For the **unbalanced test set**, Virtual Adversarial Training improves the accuracy especially at the beginning of the training and leads to - compared to other methods on this test set - a big improvement in the first rounds (between 0.5 and 6%. Until ten percent, the semi-supervised method achieved the highest test accuracies. Afterwards, this can be stated for the supervised method. Contrary to the VAT methods, the experiment with both augmentations did not lead to an improvement in terms of the test accuracy. It is less or equally accurate than random sampling. Again, for the **balanced test set** the difference is bigger between the different methods.

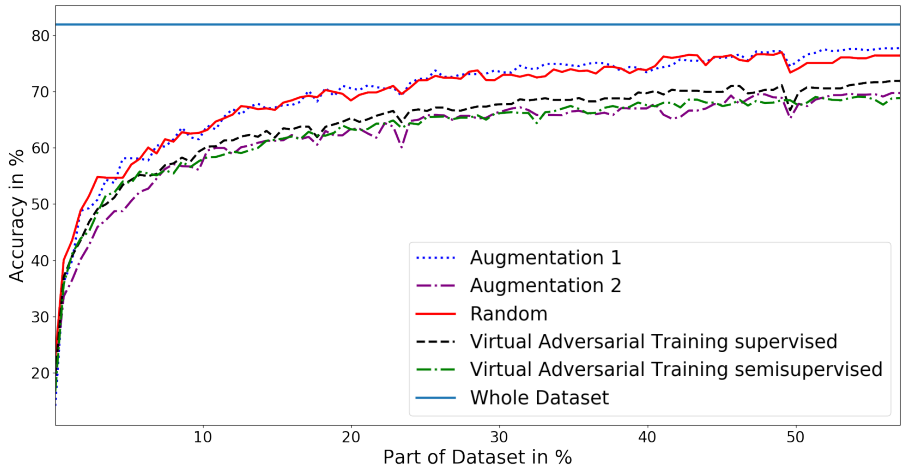


Figure 41: Comparison of random sampling and random sampling with different **regularization methods** on the CIFAR10 set. For the test set of CIFAR10, the accuracy of random sampling is for all rounds higher than or equal to the ones of the different regularization methods. Supervised Virtual Adversarial Training performs less worse than the semi-supervised approach, which performed better than Augmentation2. Augmentation1 does not perform worse than random sampling but neither does it perform better.

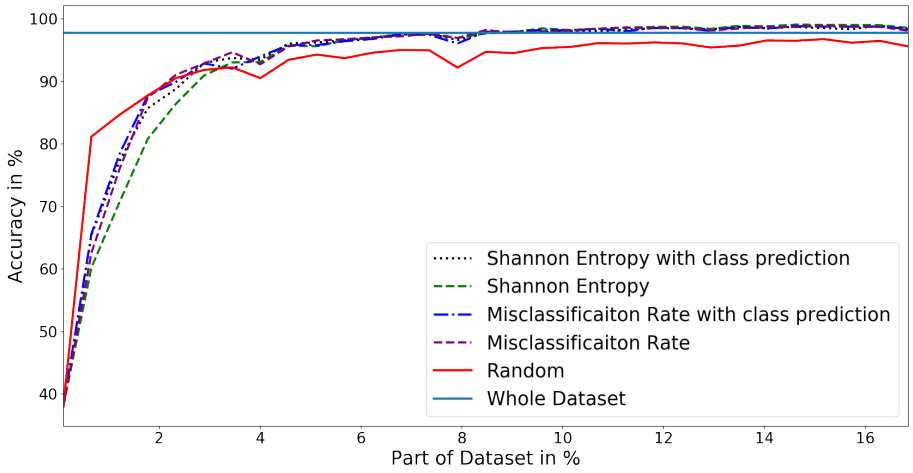
4.3.7 Combinations of Class Prediction and Uncertainty Sampling

The combination methods are only tested on the unbalanced and balanced data set from PreciBake, because class predictions did not have an effect on the balanced CIFAR10 data set. Class prediction was selected as class balance method as it yielded the biggest improvement during the experiments in Section 4.3.3.

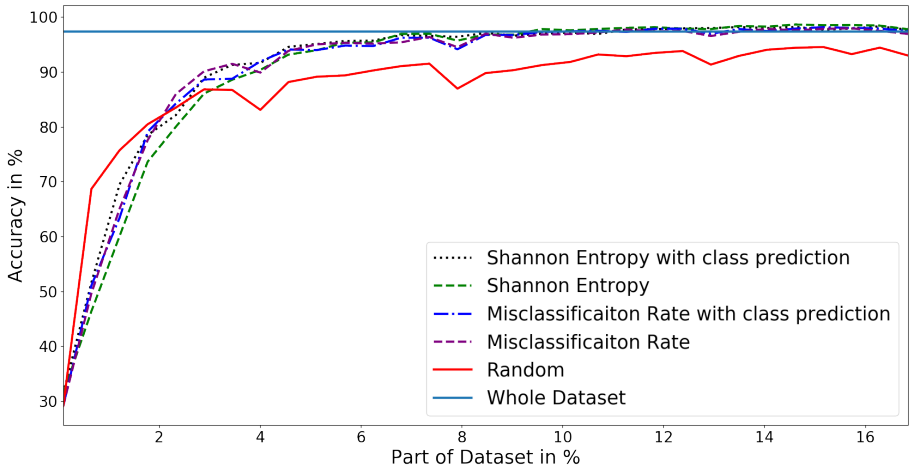
For the direct uncertainty sampling methods, the prediction of the class labels for each instance yields an improvement in the first rounds for Shannon entropy but also yields a deterioration during the first rounds for the misclassification rate. This deterioration is smaller than the improvement for Shannon entropy (see Figure 42).

For the combination of uncertainty by committee sampling and class predictions, the best two Bayesian neural network approximations (according to the experiments in Section 4.3.2) were tested with class predictions (see Figure 43 and Figure 44). The combination with class predictions leads to an improvement (especially during the first rounds) for all acquisition functions and Bayesian neural network approximations. Only for MC Batchnorm with mutual information as acquisition function there was a small range after ten percent of the data being labeled at which the combination with class predictions had a negative effect on the test accuracy.

All in all, class balance seems to improve the performance the active learning framework using uncertainty sampling methods on an unbalanced unlabeled pool of data.

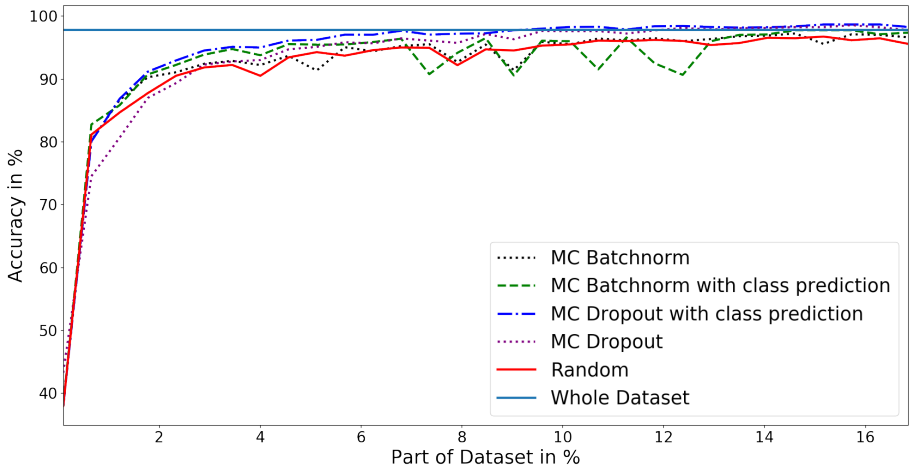


(a) Unbalanced

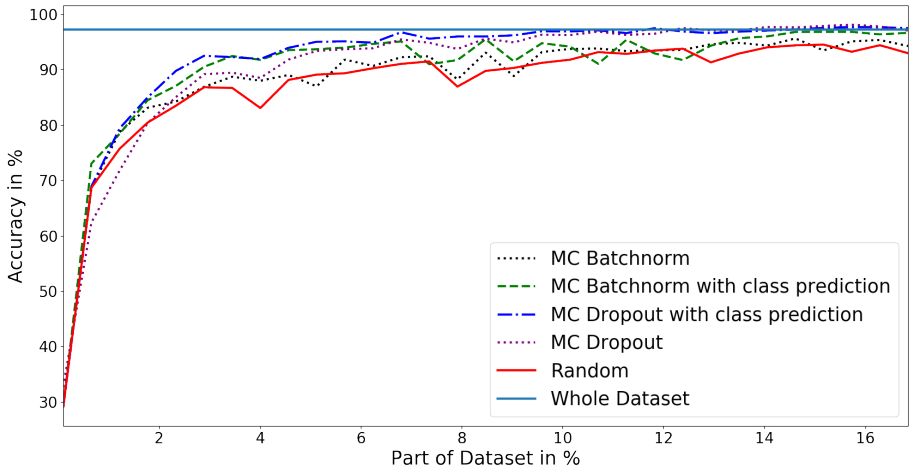


(b) Balanced

Figure 42: Comparison of Random Sampling and the **combination of direct uncertainty sampling and class prediction** and direct uncertainty sampling methods on the data set provided by PreciBake. For the **unbalanced test set**, class prediction has only a small negative effect between 1.5 and 2% on the test accuracy of the method using the misclassification rate as acquisition function but a bigger positive effect for Shannon entropy until four percent. For the **balanced test set**, especially the negative effect on the method with the misclassification rate is stronger.

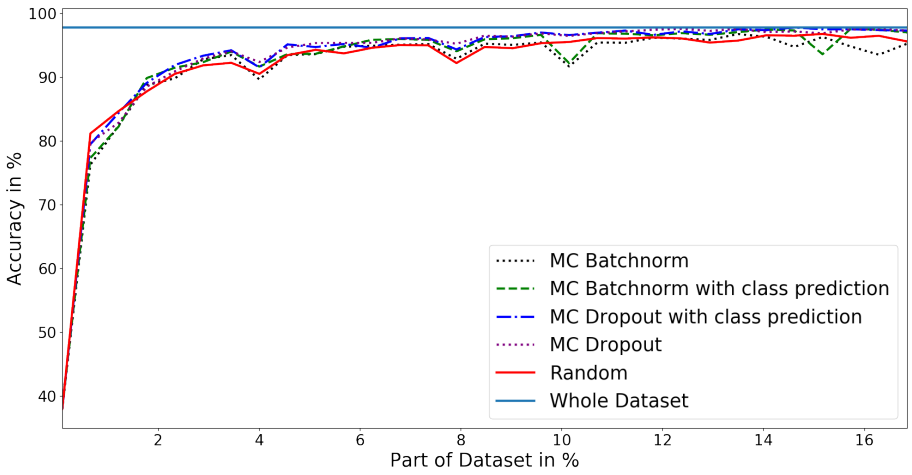


(a) Unbalanced

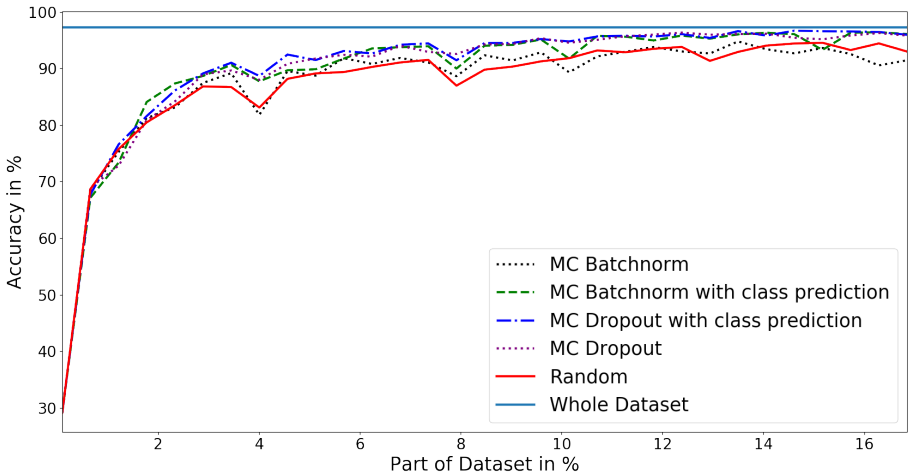


(b) Balanced

Figure 43: Comparison of random sampling and **mutual information sampling** with a **combination** of mutual information sampling and class prediction. For the **unbalanced test set**, class prediction yields a relatively big improvement for MC Dropout in terms of the test accuracy of the model trained on the selected data set. An improvement can be noted for MC Batchnorm everywhere except in the range between ten and 13 percent. For the **balanced test set**, the improvement is bigger and the negative effect in the interval from to 13 percent is not as big.



(a) Unbalanced



(b) Balanced

Figure 44: Comparison of random sampling and sampling using the **epistemic uncertainty of the predictive variance** with a **combination** of it and class prediction. For the **unbalanced test set**, class prediction yields a small improvement for both, MC Dropout and MC Batchnorm. For the **balanced test set**, the improvement is relatively small for MC Dropout but big for MC Batchnorm.

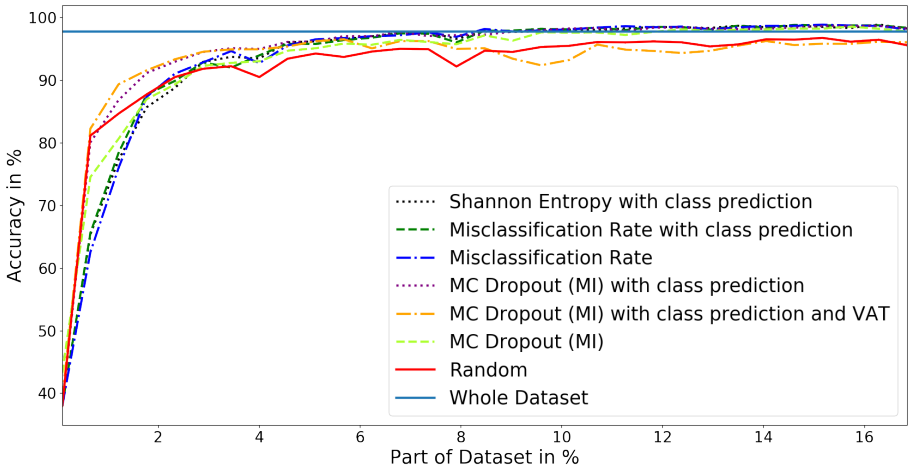
4.3.8 Best Results

Comparing the best results for the data sets from PreciBake (see Figure 46), it can be observed, that MC Dropout with mutual information as acquisition function yields the highest improvement for the model's test accuracy. This is especially true during the first rounds (between zero and four percent of the data being labeled). MC Dropout using mutual information is performing until two percent of the data is labeled better than the direct uncertainty methods. Nonetheless, all of them achieve, in this interval, worse test accuracy results than random sampling. After eight percent all of these methods are equally better than random sampling and achieve equally high test accuracies on both test sets as the training on the whole data set achieves.

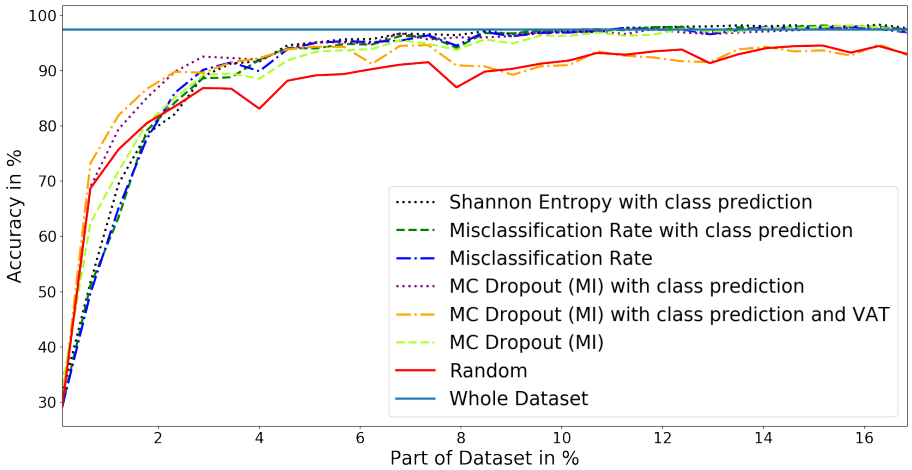
The bad test results in the first rounds can be improved by combining "MC Dropout with mutual information" with "class predictions". This combination yields test results, which perform at all stages better or equal to random sampling. Applying on this method the semi-supervised regularization technique Virtual Adversarial Training, yields only during the first round a further improvement. After eight percent of the data is labeled, it worsens the test results.

For the CIFAR10 data set (see Figure 45), the best results are achieved, except in the interval between 25 and 35 percent, for the uncertainty by committee method, MC Dropout sampling using mutual information, which performs better than both direct uncertainty methods, which outperform random sampling.

Figure 45: Comparison of random sampling and the **best methods** for CIFAR10. All methods achieve after three percent better test accuracies than random sampling. Before, only MC Dropout with mutual information as acquisition function reached equal accuracies. Except in the interval between 25 and 35 percent, MC Dropout with mutual information outperformed both direct uncertainty sampling methods.



(a) Unbalanced



(b) Balanced

Figure 46: Comparison of random sampling and the **best methods** for PreciBake's sets. On the **unbalanced test set**, MC Dropout (MCD) with class prediction (CP) achieves at all stages a higher or equal test accuracy than the other methods, except MCD with CP and VAT until two percent. The latter method becomes after percent worse than or equal to random sampling. Both methods are especially during the first rounds of the active learning framework (between 0 and 4 percent) better than the other sampling methods. Afterwards, all methods (except MCD with CP and VAT) converge to the test accuracy of the classifier trained on the whole data set. The **balanced test set** fortifies the difference between the methods and random sampling after two percent of the data were labeled. All other results are also true for the balanced test.

4.4 Summary

Table 2: **Difference of every tested sampling method and random sampling** on the unbalanced (unbal.) and balanced (bal.) test set from **PreciBake** with respect to the part of data set, which was labeled. It can be observed, that the test accuracies of the combination of MC Dropout with mutual information and class prediction (CO-CB-UC-CP-D-MI) are always higher than the of random sampling. This is especially not true for the direct uncertainty sampling methods. Nonetheless, the combination of direct uncertainty sampling methods with class prediction yields higher test results for the training on one percent of the data, too. Moreover, using Virtual Adversarial Training (RG-VA-SS or RG-VA-SV) yielded higher test results than random sampling during the first rounds (here one percent). But it had, at 16 percent, the same test results as random sampling. The combination with the best method (CO-CB-UC-CP-D-MI-VA) was only improved at the beginning and later (after eight percent) not better than random sampling.

Part of data in %	1		8		16	
	unbal.	bal.	unbal.	bal.	unbal.	bal.
DU-MR	-8.68	-10.63	4.74	7.56	2.23	3.05
DU-SE	-13.96	-19.33	4.74	8.85	**2.51**	**3.95**
UC-D-MI	-4.02	-3.95	3.55	6.79	1.76	3.37
UC-B-MI	1.98	3.03	0.50	1.29	0.54	0.95
UC-E-MI	0.85	-1.61	-7.21	-8.70	-7.75	-11.10
UC-EC-MI	-0.91	-0.69	-0.83	2.94	-1.31	0.09
UC-ED-MI	-15.23	-22.53	-10.31	-10.78	-15.95	-20.10
UC-D-PV	-1.82	-2.92	3.02	5.61	1.03	1.83
UC-B-PV	-2.27	-0.47	0.67	1.68	-2.96	-3.89
UC-E-PV	0.85	-1.61	-7.21	-8.70	-7.75	-11.10
UC-EC-PV	-0.91	-0.69	-0.83	2.94	-1.31	0.09
CB-CP	0.42	-0.13	2.47	6.64	0.66	1.80
CB-CP-L	3.02	3.22	2.19	5.11	0.98	1.98
CB-LS	-0.26	-0.79	0.84	1.03	0.21	1.16
CB-LS-L	1.96	2.77	2.41	4.79	-0.29	-0.39
RP-RP	-2.01	-0.97	1.69	2.92	-0.31	0.09
RP-DV	-2.95	-6.46	1.36	0.77	0.63	0.77
RP-RE-FC	1.63	2.62	0.33	1.63	0.34	1.10
RP-RE-CN	0.91	4.21	1.06	2.71	-0.04	0.99
RP-VA-AL	-16.97	-19.37	0.03	1.46	-0.63	-1.61
RG-VA-SS	**5.73**	**8.93**	3.12	6.34	-0.68	0.71
RG-VA-SV	2.51	4.77	3.23	5.69	-0.04	0.15
RG-AU-1	-0.83	-3.63	-0.15	0.11	-1.88	-2.13
RG-AU-2	-4.03	-4.36	0.55	-0.06	0.64	0.67
CO-DU-CP-MR	-6.26	-12.33	3.85	7.17	2.30	3.48
CO-DU-CP-SE	-7.38	-6.23	4.42	**9.45**	2.44	3.88
CO-CB-UC-CP-D-MI	2.18	3.67	**5.01**	9.04	2.21	3.26
CO-CB-UC-CP-B-MI	1.16	2.71	1.97	4.79	0.66	1.95
CO-CB-UC-CP-D-MI-VA	4.70	6.19	2.81	3.97	-0.32	0.30
CO-CB-UC-CP-D-PV	-0.29	0.84	2.14	4.49	0.88	2.02
CO-CB-UC-CP-B-PV	-2.43	-2.38	1.84	3.03	0.94	1.98

Table 3: **Difference of every tested sampling method and random sampling** on the balanced set **CIFAR10** with respect to the part of data set, which was labeled. It can be observed, that except MC Ensemble (UC-E-MI, UC-E-PV) and MC Dropout with the epistemic uncertainty of the predictive varaince (UC-D-PV), all uncertainty sampling methods yielded higher test accuracies than random sampling. All remaining sampling methods did not yield better test accuracies than random sampling.

Part of data in %	14	28	56
DU-MR	2.51	0.06	2.80
DU-SE	3.01	**1.34**	2.92
UC-D-MI	1.29	-0.09	**3.49**
UC-B-MI	2.75	1.13	0.67
UC-E-MI	-12.84	-14.54	-13.42
UC-D-PV	-6.50	-4.06	-0.23
UC-B-PV	**4.00**	0.90	1.15
UC-E-PV	-12.84	-14.54	-13.42
CB-CP	-0.66	-0.36	-0.31
CB-CP-L	-3.11	-5.33	-3.46
CB-LS	1.91	-0.46	1.30
CB-LS-L	0.72	-0.15	0.00
RP-RE-FC	-0.35	-1.65	-0.24
RP-RE-CN	-1.09	-3.37	-0.69
RG-VA-SS	-6.89	-8.19	-7.65
RG-VA-SV	-4.92	-6.52	-4.52
RG-AU-1	-5.98	-7.86	-6.68
RG-AU-2	0.98	-0.43	1.26

5 Conclusion

This thesis aimed to label a significantly smaller subset of a given unlabeled pool of data. On this subset, the chosen artificial neural network should achieve similar test results as on the whole labeled pool. Therefore, an active learning framework was built, which gradually selects subsets from the pool of unlabeled data. This framework makes use of different subset selection methods, which can be grouped into uncertainty sampling, class balance sampling, and representation sampling. Additionally, the framework can combine different sampling methods. Since deep neural networks tend to overfit on small data sets, different regularization methods were applied as well.

From the three presented sampling groups, the methods from uncertainty sampling achieved the best test results and they improved the accuracy on all test sets. Class balance sampling methods improved the test results (compared to random sampling) for the data set provided by PreciBake, especially on the balanced test set, but they had a negligible effect on the test results on the balanced CIFAR10 data set. Representation sampling has varying results on the data set provided by PreciBake and even deteriorates the test results on the CIFAR10 set (compared to random sampling). Some regularization methods improved the performance of random sampling for the data set provided by PreciBake but they worsened the test accuracy for CIFAR10. Nevertheless, their hyperparameter may have to be tuned further. Thus, for each data set, it must be decided separately how useful a regularization method is for the framework.

For uncertainty sampling, the best methods were the uncertainty by committee selection methods MC Dropout and MC Batchnorm with mutual information as acquisition function. They had higher test accuracies especially during the first rounds of the training on PreciBake's data set. MC Dropout with mutual information achieves, all in all, higher test accuracies than the uncertainty sampling methods for CIFAR10.

All class balance sampling methods yielded higher test accuracies than random sampling, while class predictions improved the results the most. However, the methods, except class predictions, which included the already labeled set, had negligible changes of the test accuracies on CIFAR10. The latter yielded worse results. This can be explained by the fact that CIFAR10 is a class balanced data set. While for representation sampling the methods representation and diversity did not significantly improve the test accuracy in comparison to random sampling, sampling methods using the reconstruction error as acquisition function yielded higher test accuracy for the data set provided by PreciBake but lower for CIFAR10. The combination of the best class balance method and the best uncertainty sampling methods did not significantly change the test accuracies on PreciBake's data set. The best combination of methods is, as for uncertainty sampling, class predictions with MC Dropout and mutual information as acquisition function. This method yielded especially during the first rounds the best results. Since the semi-supervised Virtual Adversarial Training regularization method yielded better results for random sampling on PreciBake's data set, it was also tested on the best method. While this combination improved again the test results during the first rounds, its test accuracy was after eight percent (of the data set were labeled) worse than the uncertainty sampling methods and achieved similar results as random sampling.

It can be concluded, that the problem could be solved with different methods with a similar test accuracy as the training on the whole labeled data set would have yielded by only using a subset of the tenth size. Overall, the best performance is achieved by a combination of the class balance sampling method class prediction and the uncertainty sampling method MC Dropout with mutual information as acquisition function. Hence, existing uncertainty sampling methods have been improved by combining it with class balance sampling methods. Moreover, Virtual Adversarial Training as regularization method should only be used in the first few rounds of the active learning framework.

Future Work Since the most positive effect on the model's accuracy was scored by uncertainty by committee methods, it is most promising to improve these methods to get a better model accuracy with even less data. The performance can be improved by a better Bayesian neural network approximation. Using a higher number of committee members or a different approximation method like Bayes by Backprop [8], [41] may yield better approximations. Thus, it is advisable to test especially the latter method in the future.

Since the methods using the reconstruction error as acquisition function yielded improvements for PreciBake's data set, these methods may get improved by using different encoder or decoder networks. For example, the classification network (or a part of it up to a specific layer) can be used as the encoder.

For regularization, the Bayesian Generative Active Learning framework [66], which augments the samples directly after they are labeled, can be tested during further research.

References

[1] Precibake homepage. `http://www.precibake.com/`. Accessed: 2020-01-25.

[2] Dana Angluin. Queries and concept learning. *Machine learning*, 2(4):319–342, 1988.

[3] Michel Jose Anzanello and Flavio Sanson Fogliatto. Learning curve models and applications: Literature review and research directions. *International Journal of Industrial Ergonomics*, 41(5):573–583, 2011.

[4] Les E Atlas, David A Cohn, and Richard E Ladner. Training connectionist networks with queries and selective sampling. In *Advances in neural information processing systems*, pages 566–573, 1990.

[5] Gustavo EAPA Batista, Ronaldo C Prati, and Maria Carolina Monard. A study of the behavior of several methods for balancing machine learning training data. *ACM SIGKDD explorations newsletter*, 6(1):20–29, 2004.

[6] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[7] Marcus D Bloice, Christof Stocker, and Andreas Holzinger. Augmentor: an image augmentation library for machine learning. *arXiv preprint arXiv:1708.04680*, 2017.

[8] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*, 2015.

[9] Kersting Götz Brokate, Martin. *Maß und Integral*. Birkhaüser, 2011.

[10] Kashyap Chitta, Jose M. Alvarez, and Adam Lesnikowski. Large-scale visual active learning with deep probabilistic ensembles, 2018.

[11] David Cohn, Les Atlas, and Richard Ladner. Improving generalization with active learning. *Machine learning*, 15(2):201–221, 1994.

[12] Ido Dagan and Sean P Engelson. Committee-based sampling for training probabilistic classifiers. In *Machine Learning Proceedings 1995*, pages 150–157. Elsevier, 1995.

[13] Dr. Darrin. Passive vs Active Learning. `https://educationalresearchtechniques.com/2018/02/14/passive-vs-active-learning/`. Accessed: 2019-12-31.

[14] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262, 2004.

[15] Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

[16] Gregory Druck, Gideon Mann, and Andrew McCallum. Learning from labeled features using generalized expectation criteria. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 595–602. ACM, 2008.

[17] Yarin Gal. *Uncertainty in deep learning*. PhD thesis, PhD thesis, University of Cambridge, 2016.

[18] Yarin Gal, Riashat Islam, and Zoubin Ghahramani. Deep bayesian active learning with image data. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1183–1192. JMLR. org, 2017.

[19] Andrew Gelman, John B Carlin, Hal S Stern, David B Dunson, Aki Vehtari, and Donald B Rubin. *Bayesian data analysis*. Chapman and Hall/CRC, 2013.

[20] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[21] Michael T Goodrich and Roberto Tamassia. *Algorithm design: foundation, analysis and internet examples*. John Wiley & Sons, 2006.

[22] David Gorisse, Matthieu Cord, and Frederic Precioso. Locality-sensitive hashing for chi2 distance. *IEEE transactions on pattern analysis and machine intelligence*, 34(2):402–409, 2011.

[23] Stephan Guünemann. Introduction to machine learning. University Lecture at Techinical University of Munich, 2019.

[24] Geoffrey E Hinton and Sam T Roweis. Stochastic neighbor embedding. In *Advances in neural information processing systems*, pages 857–864, 2003.

[25] Matthew D Hoffman, David M Blei, Chong Wang, and John Paisley. Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347, 2013.

[26] Stefan Hosein. Active learning: Curious ai algorithms. `https://www.datacamp.com/community/tutorials/active-learning`, 2018.

[27] Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E. Hopcroft, and Kilian Q. Weinberger. Snapshot ensembles: Train 1, get m for free, 2017.

[28] Forrest Iandola, Matt Moskewicz, Sergey Karayev, Ross Girshick, Trevor Darrell, and Kurt Keutzer. Densenet: Implementing efficient convnet descriptor pyramids. *arXiv preprint arXiv:1404.1869*, 2014.

[29] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[30] Vishal Kaushal, Rishabh Iyer, Suraj Kothawade, Rohan Mahadev, Khoshrav Doctor, and Ganesh Ramakrishnan. Learning from less data: A unified data subset selection and active learning framework for computer vision. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1289–1299. IEEE, 2019.

[31] Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? *CoRR*, abs/1703.04977, 2017.

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

[33] Diederik P. Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick, 2015.

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

[35] Claudia Kluppelberg. Einführung in die statistik. University Lecture at Techinical University of Munich, 2016.

[36] D. Koller and N. Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press, 2009.

[37] Anders Krogh and Jesper Vedelsby. Neural network ensembles, cross validation, and active learning. In *Advances in neural information processing systems*, pages 231–238, 1995.

[38] Brian Kulis and Kristen Grauman. Kernelized locality-sensitive hashing for scalable image search. In *ICCV*, volume 9, pages 2130–2137, 2009.

[39] Yongchan Kwon, Joong-Ho Won, Beom Joon Kim, and Myunghee Cho Paik. Uncertainty quantification using Bayesian neural networks in classification: Application to ischemic stroke lesion segmentation, 2018. Online: `https://openreview.net/pdf?id=Sk_P2Q9sG`.

[40] Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, Hugo Larochelle, and Ole Winther. Autoencoding beyond pixels using a learned similarity metric. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1558–1566, New York, New York, USA, 20–22 Jun 2016. PMLR.

[41] Felix Laumann, Kumar Shridhar, and Adrian Llopart Maurin. Bayesian convolutional neural networks. *CoRR*, abs/1806.05978, 2018.

[42] David D Lewis and William A Gale. A sequential algorithm for training text classifiers. In *SIGIR'94*, pages 3–12. Springer, 1994.

[43] Andrew Lim, Brian Rodrigues, Fan Wang, and Zhou Xu. k-center problems with minimum coverage. *Theoretical Computer Science*, 332(1-3):1–17, 2005.

[44] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[45] Tom M Mitchell. Generalization as search. *Artificial intelligence*, 18(2):203–226, 1982.

[46] Takeru Miyato, Shin-Ichi Maeda, Masanori Koyama, and Shin Ishii. Virtual adversarial training: A regularization method for supervised and semi-supervised learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(8):1979–1993, Aug 2019.

[47] David M Mount. Cmsc 451 design and analysis of computer algorithms. *Dept. of Computer Science, University of Maryland*, pages 1–135, 2003.

[48] Hyeonwoo Noh, Tackgeun You, Jonghwan Mun, and Bohyung Han. Regularizing deep neural networks by noise: its interpretation and optimization. In *Advances in Neural Information Processing Systems*, pages 5109–5118, 2017.

[49] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier gans. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2642–2651. JMLR. org, 2017.

[50] Michael Prince. Does active learning work? a review of the research. *Journal of engineering education*, 93(3):223–231, 2004.

[51] Prof. Dr. Stefan Ulbricht Prof. Dr. Michael Ulbricht. *Nichtlineare Optimierung*. Birkhaüser, 2012.

[52] Rouhollah Rahmani and Sally A Goldman. Missl: Multiple-instance semi-supervised learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 705–712. ACM, 2006.

[53] Joseph Rocca. Bayesian inference problem, MCMC and variational inference. `https://towardsdatascience.com/bayesian-inference-problem-mcmc-and-variational-inference-25a8aa9bce29`, 2019. Accessed: 2019-11-26.

[54] Silke Rolles. Einführung in die wahrscheinlichkeitstheorie. University Lecture at Techinical University of Munich, 2014.

[55] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[56] Tim Salimans, Diederik P. Kingma, and Max Welling. Markov chain monte carlo and variational inference: Bridging the gap, 2014.

[57] Peter Sanders, Rob van Stee, and P Sanders. Approximations- und online-algorithmen. University Lecture at Karlsruher Institut fuï Technologie, `https://algo2.iti.kit.edu/vanstee/courses/kcenter.pdf`, 2007.

[58] Burr Settles. Active learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2009.

[59] H Sebastian Seung, Manfred Opper, and Haim Sompolinsky. Query by committee. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 287–294, 1992.

[60] Samarth Sinha, Sayna Ebrahimi, and Trevor Darrell. Variational adversarial active learning. *CoRR*, abs/1904.00370, 2019.

[61] Adrian FM Smith and Gareth O Roberts. Bayesian computation via the gibbs sampler and related markov chain monte carlo methods. *Journal of the Royal Statistical Society: Series B (Methodological)*, 55(1):3–23, 1993.

[62] Nitish Srivastava. Improving neural networks with dropout. *University of Toronto*, 182(566):7, 2013.

[63] Martin A Tanner and Wing Hung Wong. The calculation of posterior distributions by data augmentation. *Journal of the American statistical Association*, 82(398):528–540, 1987.

[64] Mattias Teye, Hossein Azizpour, and Kevin Smith. Bayesian uncertainty estimation for batch normalized deep networks, 2018.

[65] Simon Tong and Daphne Koller. Support vector machine active learning with applications to text classification. *Journal of machine learning research*, 2(Nov):45–66, 2001.

[66] Toan Tran, Thanh-Toan Do, Ian Reid, and Gustavo Carneiro. Bayesian generative active deep learning. *arXiv preprint arXiv:1904.11643*, 2019.

[67] Yash Upadhyay. Regularization techniques for Neural Networks. `https://towardsdatascience.com/regularization-techniques-for-neural-networks-e55f295f2866`, 2019. Accessed: 2019-11-14.

[68] Michael M. Wolf. Mathematical foundations of machine learning. University Lecture at Techinical University of Munich, 2018.

[69] Theodore P Wright. Factors affecting the cost of airplanes. *Journal of the aeronautical sciences*, 3(4):122–128, 1936.

[70] Kaiyang Zhou, Yu Qiao, and Tao Xiang. Deep reinforcement learning for unsupervised video summarization with diversity-representativeness reward, 2017.

Appendices

A Neural Network Architectures

A.1 DenseNet121

Table 4: **DenseNet121:** the convolutional layer has stride = 2 and padding = 3, the max pooling layer has stride = 2 and padding = 1.

Layer Type	Output Channels	Input Shape	Layer Specs	NonLinearity
Convolution	64	$N \times 3 \times 112 \times 112$	ker = 7×7, pad = 3	
Batch Norm	64	$N \times 64 \times 112 \times 112$		ReLU
Max Pooling	64	$N \times 64 \times 112 \times 112$	ker = 3×3	
Dense Block (1)	256	$N \times 64 \times 56 \times 56$	6× Dense Layer	
Transition Layer	128	$N \times 256 \times 56 \times 56$		ReLU
Dense Block (2)	512	$N \times 128 \times 28 \times 28$	12× Dense Layer	
Transition Layer	256	$N \times 512 \times 28 \times 28$		ReLU
Dense Block (3)	1024	$N \times 256 \times 14 \times 14$	24× Dense Layer	
Transition Layer	512	$N \times 1024 \times 14 \times 14$		ReLU
Dense Block (4)	1024	$N \times 512 \times 7 \times 7$	16× Dense Layer	
Batch Norm	1024	$N \times 1024 \times 7 \times 7$		
Avg Pooling	1024	$N \times 1024 \times 7 \times 7$	ker = 1×1	
FC Layer	1000	$N \times 1024$		Softmax
Output		$N \times 1000$		

Table 5: **Dense Layer:** the first convolutional layer has padding = 0 and stride = 1, the second has padding = 1 and stride = 1.

Layer type	Input Shape	Output Channels	Kernel Size	Non-Linearity
Batch Normalization	$N \times c_{in} \times h \times w$	c_{in}		ReLU
Convolution	$N \times c_{in} \times h \times w$	128	1×1	
Batch Normalization	$N \times 128 \times h \times w$	128		ReLU
Convolution	$N \times 128 \times h \times w$	32	3×3	
Dropout ($p = 0.5$)	$N \times 32 \times h \times w$	32		
Output	$N \times 32 \times h \times w$			

Table 6: **Transition Layer:** the convolutional layer has stride = 1 and padding = 0, the pooling layer has stride = 2 and padding = 0

Layer type	Input Shape	Output Channels	Kernel Size	Non-Linearity
Batch Normalization	$N \times c_{in} \times h \times w$	c_{in}		ReLU
Convolution	$N \times c_{in} \times h \times w$	$\lfloor \frac{c_{in}}{2} \rfloor$	1×1	
Average Pooling	$N \times \lfloor \frac{c_{in}}{2} \rfloor \times h \times w$	$\lfloor \frac{c_{in}}{2} \rfloor$	2×2	
Output	$N \times \lfloor \frac{c_{in}}{2} \rfloor \times \lfloor \frac{h}{2} \rfloor \times \lfloor \frac{w}{2} \rfloor$			

A.2 DenseNetSmall-128

Table 7: **DenseNetSmall-128:** for the convolutional layer is stride = 2 and padding = 3, for the max pooling layer is stride = 2 and padding = 1.

Layer Type	Output Channels	Input Shape	Layer Specs	NonLinearity
Convolution	64	$N \times 3 \times 128 \times 128$	ker = 7×7, pad = 3	
Batch Norm	64	$N \times 64 \times 128 \times 128$		ReLU
Pooling	64	$N \times 64 \times 128 \times 128$	ker = 3×3	
Dense Block (1)	256	$N \times 64 \times 64 \times 64$	6× Dense Layer	
Transition Layer	128	$N \times 256 \times 64 \times 64$		ReLU
Dense Block (2)	512	$N \times 128 \times 32 \times 32$	12× Dense Layer	
Transition Layer	256	$N \times 512 \times 32 \times 32$		ReLU
Dense Block (3)	1024	$N \times 256 \times 16 \times 16$	24× Dense Layer	
Batch Norm	1024	$N \times 1024 \times 16 \times 16$		
Pooling	1024	$N \times 1024 \times 16 \times 16$	ker = 16×16, avg	
FC Layer	12	$N \times 1024$		Softmax
Output		$N \times 12$		

A.3 DenseNetSmall-32

Table 8: **DenseNetSmall-32:** for the convolutional layer is stride = 2 and padding = 3, for the max pooling layer is stride = 2 and padding = 1.

Layer Type	Output Channels	Input Shape	Layer Specs	NonLinearity
Convolution	64	$N \times 3 \times 32 \times 32$	ker = 7×7, pad = 3	
Batch Norm	64	$N \times 64 \times 32 \times 32$		ReLU
Pooling	64	$N \times 64 \times 32 \times 32$	ker = 3×3	
Dense Block (1)	256	$N \times 16 \times 16 \times 64$	6× Dense Layer	
Transition Layer	128	$N \times 256 \times 16 \times 16$		ReLU
Dense Block (2)	512	$N \times 128 \times 8 \times 8$	12× Dense Layer	
Batch Norm	512	$N \times 512 \times 8 \times 8$		
Pooling	512	$N \times 512 \times 8 \times 8$	ker = 8×8, avg	
FC Layer	10	$N \times 512$		Softmax
Output		$N \times 10$		

A.4 Variational Autoencoder

Table 9: **Variational Autoencoder** consisting of fully connected layers (FC-VAE)

(a) encoder

Layer Type	Input Size	Output Size	Non-Linearity
Fully Connected	$N \times 3 \times h * w$	400	ReLU
Fully Connected	$N \times 3 \times 400$	20	
Output	$N \times 3 \times 20$		

(b) decoder

Layer Type	Input Size	Output Size	Non-Linearity
Fully Connected	$N \times 3 \times 20$	400	ReLU
Fully Connected	$N \times 3 \times 400$	h * w	Sigmoid
Output	$N \times 3 \times h \times w$		

A.5 Convolutional Variational Autoencoder

Table 10: **Convolutional Variational Autoencoder** with convolutional layers for the CIFAR10 data set (CNN-VAE). All (transpose) convolutional layers have stride = 2 and padding = 0.

(a) encoder

Layer Type	Kernel Size	Input Shape	Output Channels	Non-Linearity
Convolution	4×4	$N \times 3 \times 32 \times 32$	32	ReLU
Convolution	4×4	$N \times 32 \times 15 \times 15$	64	ReLU
Fully Connected		$N \times 64 * 6 * 6$	32	
Output		$N \times 32$		

(b) decoder

Layer Type	Kernel Size	Input Shape	Output Channels	Non-Linearity
Fully Connected	6×6	$N \times 32$	64	
Transp. Convolution	6×6	$N \times 64 \times 1 \times 1$	32	ReLU
Transp. Convolution	6×6	$N \times 32 \times 6 \times 6$	16	ReLU
Transp. Convolution	2×2	$N \times 16 \times 16 \times 16$	3	Sigmoid
Output		$N \times 3 \times 32 \times 32$		

Table 11: **Convolutional Variational Autoencoder** with Convolutional layers for the data set provided by PreciBake (CNN VAE). All convolutional and transpose convolutional layers have stride = 2 and padding = 0.

(a) encoder

Layer Type	Kernel Size	Input Shape	Output Channels	Non-Linearity
Convolution	4×4	$N \times 3 \times 128 \times 128$	32	ReLU
Convolution	4×4	$N \times 32 \times 63 \times 63$	64	ReLU
Convolution	4×4	$N \times 64 \times 30 \times 30$	128	ReLU
Convolution	4×4	$N \times 128 \times 14 \times 14$	256	ReLU
Fully Connected		$N \times 256 * 6 * 6$	32	
Output		$N \times 32$		

(b) decoder

Layer Type	Kernel Size	Input Shape	Output Channels	Non-Linearity
Fully Connected		$N \times 32$	256	
Transp. Convolution	5×5	$N \times 256 \times 1 \times 1$	128	ReLU
Transp. Convolution	6×6	$N \times 128 \times 5 \times 5$	64	ReLU
Transp. Convolution	6×6	$N \times 64 \times 13 \times 13$	32	ReLU
Transp. Convolution	6×6	$N \times 32 \times 30 \times 30$	16	ReLU
Transp. Convolution	2×2	$N \times 16 \times 64 \times 64$	3	Sigmoid
Output		$N \times 3 \times 128 \times 128$		

Table 12: **Discriminator** for Variational Adversarial Active Learning

Layer Type	Output Size	Input Shape	NonLinearity
Fully Connected	512	$N \times 32$	ReLU
Fully Connected	512	$N \times 512$	ReLU
Fully Connected	512	$N \times 512$	
Fully Connected	1	$N \times 512$	Sigmoid
Output		$N \times 1$	

B Training Hyperparameters

Table 13: Training Hyperparameters for both data sets. T and E are only necessary when the corresponding Bayesian neural network approximation is used.

Hyperparameter	Value
loss	L_{CE} (Equation 2.3.2)
optimizer	Adam [32]
learning rate type	exponential
learning rate start	0.001
learning rate decay	0.997
initialization set train	one image per class
initialization set val	one image per class
MC samples (T)	50 (Dropout, Batchnorm) 8 (Ensemble, Ensemble Cyclic) 4 (Ensemble DPE)

Table 14: Hyperparameters for the training with the data set provided by **PreciBake**

Hyperparameter	Value
minimum number of iterations	250
minimum number of epochs	10
maximum batch size	64
labeled images per round train (B)	100
preselected images per round (β)	400
labeled images per round val	42
rounds	30
seeds	0, 123, 999, 5
dropout rate	0.5

Table 15: Hyperparameters for the training with the **CIFAR10** data set

Hyperparameter	Value
minimum number of iterations	150
minimum number of epochs	20
maximum batch size	1024
labeled images per round train (B)	200
preselected images per round (β)	800
labeled images per round val	85
rounds	100
seeds	0, 123

Table 16: Hyperparameters for all **Variational Auto-Encoder** networks

Hyperparameter	Value
epochs	40 (FC), 80 (CNN)
loss	$L = MSE + \mathbb{KL}$
optimizer	Adam [32]
learning rate decay type	exponential
learning rate start	0.001
learning rate decay	0.997

Table 17: Hyperparameters for the **Variational Adversarial Active Learning**

Hyperparameter	Value
iterations	10,000
maximum batch size	1024
optimizers	Adam [32]
learning rates' decay type	step decay
learning rates' decay step	400
learning rates' decay	0.9
L_{VAE}^{trd}	MSE -$\beta\mathbb{KL}$
β	1
L_{VAE}^{adv}	binary cross entropy
λ_1	1
λ_2	1
number steps VAE	2
L_D	binary cross entropy
number steps D	1

B.1 Augmentation

Table 18: Hyperparameters for **Augmentation 1**, augmentation methods used from [7].

Augmentation Method	Probability	Parameter	Parameter
rotation	0.3	max. rot. left: 10	max. rot. right: 10
flip top bottom	0.25		
flip left right	0.25		
crop random	0.25	percentage area = 0.9	
random contrast	0.5	min factor = -0.5	max factor = 0.5
resize	1	height: 128 or 32	width = 128 or 32

Table 19: Hyperparameters for **Augmentation 2**, augmentation methods used from [7].

Augmentation Method	Probability	Parameter	Parameter
rotation	1	maximal rot. left: 5	maximal rot. right: 5
flip top bottom	0.5		
zoom random	1	percentage area: 0.5	

C Data sets

Table 20: Data set provided by **PreciBake** with the number of images per class for the training, validation and testing of the model

Class Name	Train + Val	Test
Apfelecke	1,225	306
Baguette	1,756	439
Ciabatta	1,030	257
Kategorie Brot	3,961	989
Knusperspitz	387	97
Laugenbrezel	6,353	1,588
Laugenbrötchen	593	148
Laugenzopf	1,524	381
Pizza Magherita	1,900	476
Schinken-Käse Croissant	764	191
Schnitt-brötchen	3,893	974
Sonnenblumen-brötchen	2,041	510
total	25,439	6,367

Table 21: **CIFAR10** with 10 classes and the corresponding number of images per class and for training and validation or testing of the model

Class Name	Train + Val	Test
airplane	5,000	1,000
automobile	5,000	1,000
bird	5,000	1,000
cat	5,000	1,000
deer	5,000	1,000
dog	5,000	1,000
frog	5,000	1,000
horse	5,000	1,000
ship	5,000	1,000
truck	5,000	1,000
total	50,000	10,000

D Additional Methods

D.1 Bayes by Backprop

This method is another technique to get an approximation of the real underlying distribution of a Bayesian neural network. Bayes by Backprop is a Variational Inference approach. It aims to find the best parameters of a given parametrization family with backpropagation and the local reparameterization trick.

As stated in [8] it applies:

Theorem D.1. *Let $\epsilon \sim q(\epsilon)$ be a random variable with $q(\epsilon)\mathrm{d}\epsilon = q(w|\theta)\mathrm{d}w$ and let $w = t(\theta, \epsilon)$ with t being a deterministic function. Then for a function f with derivatives in w it applies:*

$$\frac{\partial}{\partial\theta}\mathbb{E}_{q(w|\theta)}[f(w,\theta)] = \mathbb{E}_{q(\epsilon)}\left[\frac{\partial f(w,\theta)]}{\partial w}\frac{\partial w}{\partial\theta} + \frac{\partial f(w,\theta)]}{\partial\theta}\right] \tag{D.1}$$

Proof. In section 3.1 of [8]. □

Using this theorem, $\nu = \frac{\partial f(w,\theta)]}{\partial w}$ only needs to be determined once. This yields for gaussian distributed weights with $w = \mu + \sigma * \epsilon$ to the following equations:

$$\begin{aligned}\nabla_\mu &= \frac{\partial f(w,\theta)]}{\partial w}\frac{\partial w}{\partial\theta} + \frac{\partial f(w,\mu)}{\partial\mu} \\ &= \nu\frac{\partial w}{\partial\mu} + \frac{\partial f(w,\mu)}{\partial\mu} \\ &\overset{\frac{\partial w}{\partial\mu}=1}{=} \nu + \frac{\partial f(w,\mu)}{\partial\mu}\end{aligned}$$

and

$$\begin{aligned}\nabla_\sigma &= \frac{\partial f(w,\theta)]}{\partial w}\frac{\partial w}{\partial\theta} + \frac{\partial f(w,\sigma)}{\partial\sigma} \\ &= \nu\frac{\partial w}{\partial\sigma} + \frac{\partial f(w,\sigma)}{\partial\sigma} \\ &\overset{\frac{\partial w}{\partial\sigma}=\epsilon}{=} \nu\epsilon + \frac{\partial f(w,\mu)}{\partial\mu}\end{aligned}$$

This leads to the following algorithm.

Data: non-normalized probability distribution
Result:
1: draw $\epsilon \sim \mathcal{N}(0,1)$
2: $w = \mu + log(1 + exp(\rho)) * \epsilon$
3: $\theta = (\mu, \rho)$
4: $f(w,\theta) = log(q(w|\theta)) - log(p(w)p(Y|X,w))$ (ELBO)
5: $\nu = \frac{\partial f(w,\theta)]}{\partial w}$
6: $\nabla_\mu = \nu + \frac{\partial f(w,\theta)}{\partial \mu}$
7: $\nabla_\rho = \nu \frac{\epsilon}{1+exp(-\rho)} + \frac{\partial f(w,\sigma)}{\partial \sigma}$
8: $\mu \leftarrow \mu - \alpha \nabla_\mu$
9: $\rho \leftarrow \rho - \alpha \nabla_\rho$

Algorithm 8: Bayes by Backprop [8]

Deep neural networks have lots of parameters. The need of sampling errors gets computationally very costly for lots of parameters in network architectures with a high amount of layers. Thus, assume for example a layer with m neurons with inputs $x \in \mathbb{R}^{1000}$. For only this layer ϵ has to be sampled m*1000 times. Thus, this method is not suitable for deep neural networks. The local reparameterization trick overcomes this issue by sampling only the activation functions.

Local reparameterization trick [33] As the input of an activation function is of a lower dimension then the product of the weight matrix of the layer and its input, it is less costly to sample for each output of the activation function instead of sampling for each summand of the matrix product. Assuming a gaussian distributions of the parameters this yields to the following sampling methods for fully connected and convolutional layers respectively. For a fully connected layer the input $z_{i,j}$ of the activation function is with input values $x_{i,j}$:

$$z_{i,j} = \sum_{k=1}^{M_{i-1}} \omega_{i,j,k} x_{k,j} + \omega_{i,j,0}. \tag{D.2}$$

It applies with $\epsilon_{i,j} \sim \mathcal{N}(0,1)$ and

$$z_{i,j} = \gamma_{i,j} + \delta_{i,j} * \epsilon_{i,j} \tag{D.3}$$

$$= \sum_{k=1}^{M_{i-1}} \mu_{i,j,k} x_{k,j} + \mu_{i,0} + \sqrt{\sum_{k=1}^{M_{i-1}} \sigma_{i,j,k}^2 x_{k,j}^2 + \sigma_{i,0}} \epsilon_{i,j}. \tag{D.4}$$

Equation D.3 yields $z_{i,j} \sim \mathcal{N}(\gamma_{i,j}, \delta_{i,j})$. And finally, Equation D.4 yields the weights $\omega_{i,j}$ of the j^{th} neuron of the i^{th} layer to be distributed with $q_\theta(\omega_{i,j,k}|X,Y) = \mathcal{N}(\mu_{i,j,k}, \sigma_{i,j,k})$.

And for a convolutional layer the input z_k of the nonlinear function is with input values $x_{k'}$:

$$z_{i,k} = \sum_{k'=1}^{K_{i-1}} W_{i,k,k'} * x_{k'} + W_{k,0} \tag{D.5}$$

It applies with $\epsilon_k \sim \mathcal{N}(0,1)$:

$$z_{i,k} = \gamma_{i,k} + \epsilon_{i,k}\delta_{i,k} \tag{D.6}$$

$$= \sum_{k'=1}^{K_{i-1}} \mu_{i,k,k'} * x_{k'} + \sigma_{i,k,0} + \epsilon_k \sqrt{\sum_{k'=1}^{K_{i-1}} \sigma^2_{i,k,k'} * x_{k'} + \sigma^2_{i,k,0}} \tag{D.7}$$

Equation D.6 yields $z_k \sim \mathcal{N}(\gamma_k, \delta_k)$
And finally, Equation D.7 yields the weights $\omega_{i,k,k'} \in W_{i,k,k'}$ of the k^{th} filter of the i^{th} layer to be distributed with $q_\theta(\omega_{i,k,k',h,w}|X,Y) = \mathcal{N}(\mu_{i,k,k',h,w}, \sigma_{i,k,k',h,w})$.

The local reparameterization trick is especially important for convolutional layers as the weights $\epsilon_{k,k',h,w}$ would be sampled for every $\omega_{k,k',h,w}$ $H_i * W_i$ times as this is the number of patches obtained from the input x.

Estimating Uncertainties using Bayes by Backprop Usually, in Bayes by Backprop it is assumed $q_\theta(\omega|X,Y) \sim \mathcal{N}(\omega \mid \mu, \Sigma)$ with $\theta = \{\mu, \Sigma\}$.
Moreover, Bayes by Backprop assumes the weights independent of each other.
Thus: $\Sigma = \begin{pmatrix} \sigma_1^2 & 0 & \cdots & 0 \\ 0 & \sigma_2^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_d^2 \end{pmatrix}$ with d being the number of parameters in the network.
Using Equation 2.49 leads to a predictive posterior distribution of the form:

$$p(y^* \mid x^*, \mathrm{X}, \mathrm{Y}) \approx \int_\Omega p(y^* \mid x^*, \omega)\mathcal{N}(\omega \mid \mu, \Sigma)\mathrm{d}\omega \tag{D.8}$$

As the task is a classification problem, the label y^* of the unseen instance x^* is assumed to be categorical distributed, which leads to:

$$p(y^* \mid x^*, \mathrm{X}, \mathrm{Y}) \approx \int_\Omega \mathrm{Cat}(y^* \mid f_\omega(x^*))\mathcal{N}(\omega \mid \mu, \Sigma)\mathrm{d}\omega \tag{D.9}$$

$$= \int_\Omega \prod_{c=1}^{C} f_\omega(x^*)^{y_c^*} \frac{1}{\sqrt{(2\pi)^k \det(\Sigma)}} e^{-\frac{(\omega-\mu)^T\Sigma^{-1}(\omega-\mu)}{2}} \mathrm{d}\omega$$

$$= \int_\Omega \prod_{c=1}^{C} f_\omega(x^*)^{y_c^*} \prod_{i=1}^{d} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(\omega^i-\mu_i)^2}{2\sigma_i^2}} \mathrm{d}\omega \tag{D.10}$$

For Equation D.10 there does not exist a closed-form solution [41]. Thus, the unbiased estimator of the form of Equation 2.50 has to be used and one obtains:

$$\int_{\Omega} \prod_{c=1}^{C} f_{\omega}(x^*)^{y_c^*} \prod_{i=1}^{d} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(\omega^i - \mu_i)^2}{2\sigma_i^2}} \mathrm{d}\omega \tag{D.11}$$

$$\approx \sum_{t=1}^{T} \prod_{c=1}^{C} f_{\hat{\omega}_t}(x^*)^{y_c^*} \text{ with } \hat{\omega}_t^i \sim \mathcal{N}(\mu_i, \sigma_i^2) \tag{D.12}$$

Epistemic Uncertainty To approximate the epistemic uncertainty of the predictive variance Equation 2.65 can be used and in order to approximate the epistemic uncertainty of the entropy Equation 2.77 can be used.

D.2 Bayesian Generative Active Learning

The idea of Bayesian Generated Active Learning (BGAL) is as described in [66] to combine data augmentation with active learning. Data augmentation generates artificially new instances, which increases the size of the training set. This training set is especially low for active learning. Some of the previously considered active learning methods select instances by an acquisition function $a_{\mathcal{M}}(x)$. Assume that an acquisition function is used, then active learning may avoid that the data augmentation generates instances which waste computational resources but may generate them in a way that their information gain for the network is closely as good as the one from the most informative instances.

Theorem D.2. *Assume that there exists the gradient $\nabla_x a_{\mathcal{M}}(x)$ of the acquisition function $a_{\mathcal{M}}(x)$ with respect to the instance x and that x^* is an interior point of the unlabeled pool $\mathcal{U}$. Let x' be a generated point from then x^* by line 6 from algorithm 9, then*

$$a_{\mathcal{M}}(x') \approx a_{\mathcal{M}}(x^*). \tag{D.13}$$

Proof. In section 3.3 of [66]. □

This means that a generated instances is also informative.

The in [66] proposed procedure is therefore following: First, select the most informative instances, then label them and finally produce new artificial samples ($k \in \mathbb{N}$ each) of those instances by a generator. The preferred model is a VAE-ACGAN network, which is a combination of a VAE-GAN [40] network and an ACGAN [49] network. The encoder network E consists of the VAE part of the network whereas the generator/decoder network G and the descriminator network D are included in the ACGAN architecture. The loss

function L of this network consists of the following functions:

$$L_{VAE} = L_{rec} + L_{prior} \tag{D.14}$$

$$L_{ACGAN} = log(d_{\theta_D}(x)) + log(1 - d_{\theta_D}(g_{\theta_G}(z))) + log(1 - d_{\theta_D}(g_{\theta_G}(u))) \tag{D.15}$$

$$+ log(\sigma(c(x))) + log(\sigma(c(g(z)))) + log(\sigma(c(g_{\theta_G}(u)))) \tag{D.16}$$

with $u \sim \mathcal{N}(0, I)$ and σ being the softmax activation function.

$$L = L_{VAE} + L_{ACGAN} \tag{D.17}$$

Data: selected instances (most informative) $X_U^* \subset X_U$
Result: labeled selected instances augmented instances (X'_L, Y_L)

for $x_i \in X_U^*$ **do**
  ask to oracle to find label y_i for x_i
  add x_i, y_i to $\mathcal{L}$
  $z_i \leftarrow e_{\theta_E}(x_i)$
  for $j \in \{1, ..., k\}$ **do**
    $x'_{i,j} \leftarrow g_{\theta_G}(z_i)$
    compute loss L_{VAE}
    sample $u \sim \mathcal{N}(0, I)$
    compute loss L_{ACGAN}
    update network parameters:
    $\theta_E \leftarrow \theta_E - \alpha_1 \nabla_{\theta_E} L_{VAE}$
    $\theta_G \leftarrow \theta_G - \alpha_1 \nabla_{\theta_G} (\gamma L_{rec} - L_{ACGAN})$
    $\theta_D \leftarrow \theta_D - \alpha_2 \nabla_{\theta_D} L_{ACGAN}$
    add augmented instances $(x'_{i,j}, y_i)$ to $\mathcal{L}$
  end for
end for

Algorithm 9: Data Augmentation Algorithm