Active Learning for maximizing model utility in material science on a small fractographic data set

A thesis
submitted in partial fulfillment of the
requirements for the degree

Master of Science

Faculty of Mathematics
Otto-von-Guericke University Magdeburg

In collaboration with
Bundesanstalt für Materialforschung und -prüfung Berlin Presented by

Kilian Bock
born on 28.10.1994 in Berlin
Course of study Mathematics
Field of study Mathematics

February 28, 2022

Supervised by
Prof. Dr. rer. nat. habil. Sebastian Sager
(Institute for Mathematical Optimization, OvGU)
and
Dr.-Ing. Christoph Völker
(Bundesanstalt für Materialforschung und -prüfung)

Abstract

One of the main challenges in applying learning techniques to classification problems is the large amount of labeled training data required. Especially for microstructural images in material science training data is very expensive in terms of time, effort, and cost to come by. In this study, several active learning approaches to tackle the problem are investigated. Rather than accepting random training examples, active learning iteratively selects unlabeled examples for the user to label, so that the user's energy is focused on labeling the most "informative" examples. The examined strategies belong to Uncertainty sampling, in which the algorithm selects unlabeled examples that it finds hardest to classify, Hypothesis space search, in which selections are made according to the uncertainty in predictions of several machine learning models and a method based on distance measures in the feature space. Further, the impact of the choice of learning algorithm was investigated. Therefore a random forest model and a neural network were compared on various generated feature sets.

Specifically, results for those strategies were demonstrated for two different classification scenarios for a fractographic data set. Namely, those scenarios were fracture classification and material affiliation, which were a binary and multi-class cases, respectively. The proposed methods give large reductions in the number of training examples required over random selection to achieve similar classification accuracy.

Contents

List of Figures

List of Tables

Nomenclature

$\mathcal{Z}$	overall data set
$\mathcal{X}$	feature set
$\mathcal{Y}$	label set
x, y	input data instance and corresponding label
$\hat{y}$	predicted label
$\mathcal{L}$	labeled data set
$\mathcal{U}$	unlabeled data set
$\mathcal{H}$	hypothesis space
h	hypothesis
θ	a particular hypothesis's parameters

1. Introduction

In this chapter, incentives highlighting the importance of this study are presented. Further, an outline of the research methodology is presented, as well as an overview of the primary contribution of this study.

1.1. Motivation

In the past few decades, the significant advancement of machine learning has spread the application of data-driven approaches throughout science, commerce, and industry. Throughout history, the vast majority of technological advancements have been enabled by the discovery of new materials concomitant with enormous societal impacts. Experiments to discover Materials are expensive and often it is not uncommon to take them months to execute. Hence material data is available in small quantities. Other than in the case of "big data", where excessive data sets are available and state-of-the-art algorithms like deep neural networks can be deployed with good outcomes, the same procedure is not without further ado suitable for small data sets.
With millions of potential variants, the potential design space for new materials is huge. This is paired with a relatively poor covering by already investigated materials and the cost of time and resources to conduct an experiment. Further clear relationships between material properties and their features aren't always a given and make it hard and at the same time important to explore the vast search space efficiently. Approaches that rely on trial and error are impractical. It is also not uncommon that experiments have to be done sequentially. Therefore statistical data-driven methods are a great tool to support decision-making. Nevertheless, the importance of domain knowledge remains essential in the field of materials science, which is also reflected by the choice of the initial sampling of experiments. Finding a policy for sequencing experiments, which gives an understanding of the experimental choices and sets rules what experiment to conduct next can help to navigate the material design space, under the premise that it doesn't make any sense to design a tailored material that is un-manufacturable, or only at a prohibitively high

cost. "A consistent pattern in AI research is the need for a great volume of training data. This creates an issue for visual AI, which needs thousands of manually labeled images to learn how to identify and denominate different images. However, there are ways to improve data efficiency." Bangert n.d. "In today's information-drenched society, unlabeled data are often abundant, but the labels required to do supervised learning from these instances are difficult, time-consuming, or expensive to obtain."Settles 2012 By understanding which of the unlabeled images provides the most value when being added to the labeled ones, i.e. provide the most information increase to describe the population of this kind of image one can ensure data efficiency and spare resources. Microstructural images are one use case, where unlabeled images have to be expensively labeled. Here, similar to finding an efficient policy for deciding which experiment to conduct next under the premise of acquiring certain material properties, a policy to query images for labeling is of interest, since "labelling the data, like analyzing images in the first place, is repetitive, time consuming, and pricey" Bangert n.d.
Furthermore the resulting implications are compelling. It is possible to create a by magnitudes smaller data set with equal inherent information compared to a bigger one, which in turn implies shorter training time, or better interpretive power for an algorithm, which in turn has good applicability in other domains, like specifically the discovery process of materials. Such a data set also provides information-rich examples to a human expert, which adds value beyond the improved predictive power of an algorithm.

1.2. Short summary related work

Today's information-drenched society often produces plenty of unlabeled data, but the labels needed to perform supervised learning from these examples are difficult, time-consuming, or prohibitively expensive to obtain. It shows, that by allowing the learners to ask questions and be involved more in the training process, much can be gained. Below is a list of applications, which help to highlight manifold settings in terms of decisions, uncertainty, learning and metrics (Settles 2009):

- Classification and filtering. The process of classifying documents (e.g., articles or web pages) or any other kind of media (e.g., images, audio, and video files) involves annotating each item with particular labels, such as relevant or irrelevant. Electronic resources, such as the Internet, provides a wealth of unlabelled instances, but annotating thousands of them is tedious and even redundant.

- Speech recognition. The labeling of speech utterances requires linguistic training and is time-consuming. X. Zhu and Goldberg 2009 have reported that an annotation at the word level takes ten times longer than the actual audio data, while annotating phonemes takes 400 times longer. The problem is designed for rare or dialectal languages.

- Information extraction. Systems that extract factual information from documents need training with detailed annotations. Entities or relations of interest are highlighted, such as names of organizations or people, or whether individuals work for specific organizations. Even simple newswire stories can take more than half an hour of searching for entities and relationships Settles, Craven, and Friedland 2008. Specific annotations pertaining to specific knowledge domains may require additional expert knowledge, e.g., annotating genetic information in biomedical texts usually requires PhD-level biologists.

- Computational Biology. In the natural sciences, particularly biology, chemistry and material science, the use of machine learning for interpretation and prediction is increasing. Biochemists can, for instance, create models that aid in explaining and predicting enzyme activity from hundreds of synthesized peptide chains Smith et al. 2011. However, there are 20^n possible peptides of length n, which gives for $n = 8$ the possibility of synthesizing and testing about 2.6 billion. In practice, scientists might opt to randomly select sequences, or cherry-pick sequences from interesting proteins, but neither guarantees much information about a chemical activity.

Using traditional supervised learning methods, each of these examples require significant human effort and/or laboratory materials for data collection. When active learning systems are allowed to participate in the data collection process, allowing them to be "curious", the goal is to learn the task better with less training.

1.3. Short summary research method, main contribution

Fractographic classification tasks are highly complex, and thus hard to approach. Applying methods to counteract this issue are sought. Active learning allows an effectiv navigation through the feature space to iteratively accumulate "information-rich" data. Machine learning is used to make predictions in a highly dimensional space, in which the accuracy and uncertainty of the models predictions are of interest. Applying those

methods to small fractographic data can enhance its intrinsic information, and thus help in the classification process of fractured materials.

1.4. Introduction of the working definitions and container words

This section has the intent to give formal definitions and insights of the topic at hand.

Definition 1 (Machine Learning (ML) (Y. Wang et al. 2020)) *A computer program is said to learn from experience E concerning some classes of a task T if its performance measured by performance measure P can improve with E on T.*

A formal definition on Few-Shot Learning (FSL) gets introduced, which naturally connects to the machine learning definition given above. With this definition one can better clarify what the goal of this work and in general the goal of FSL is and how it can be approached.

Definition 2 (Few-Shot Learning (Y. Wang et al. 2020)) *The type of machine learning problems, where experience E contains only a limited number of examples with supervised information for the task T is called Few-Shot Learning (FSL). It is specified by E, T and P.*

Consider a learning task T, FSL deals with a data set $\mathcal{Z} = \{\mathcal{Z}_{train}, \mathcal{Z}_{test}\} = (\mathcal{X}, \mathcal{Y})$ consisting of a training set $\mathcal{Z}_{train} = (\mathcal{X}_{train}, \mathcal{Y}_{train}) = \{x_i, y_i\}_{i \in I}$ where $I \subset \mathbb{N}$ is small, and a test set $\mathcal{Z}_{test} = \{\mathcal{X}_{test}, \mathcal{Y}_{test}\} = \{x_j, y_j\}_{j \in J}$ where $J \subset \mathbb{N}$ and $N = I + J$ describes the index set of the overall data set $\mathcal{Z}$, where in turn $\mathcal{X}$ and $\mathcal{Y}$ denote the feature set and label set, respectively. Let $p(x, y)$ be the ground-truth joint probability distribution of features x and labels y. Furthermore let $\mathcal{H}$ be defined as the hypothesis space. Every $h = h_\theta = h(\cdot; \theta) \in \mathcal{H}$ is determined by the parameter $\theta \in \Theta \subset \mathbb{R}^n$, where $n \in \mathbb{N}$. Specifically, h_θ is called "hypothesis" and is a particular mathematical model, related to a class of models $\mathcal{H}$ which attempts to generalize or explain the training data, and make predictions on new data instances. A strategy that optimizes the θ by searching $\mathcal{H}$ in order to find the optimal $\hat{h} \in \mathcal{H}$ gets ensuing called learner, ML algorithm, or ML model. In our case, a non-parametric model would not be suitable, since it often requires large data sets. h^* is the best hypothesis from x to y. Such an algorithm learns by fitting $\mathcal{Z}_{train}$ and testing on $\mathcal{Z}_{test}$. Its performance is measured by a loss function $l(\hat{y}, y)$, whereas $\hat{y} = h(x; \theta)$ is the prediction and y is the observed output (Y. Wang

et al. 2020). To get more clarity over these terminologies, they are later further specified in section 3.3.

In practice, in the case of Active Learning (AL) $\mathcal{Z}_{test}$ only consists of unlabeled data points, therefore only of the feature set $\mathcal{X}_{test}$, which gets denoted as $\mathcal{U} := \mathcal{Z}_{test} = \mathcal{X}_{test} = \{x_j\}_{j \in J}$. The data set used for training gets denoted as $\mathcal{L} := \mathcal{Z}_{train}$, it is fully labeled. Therefore querying a new data point also consists of determining its corresponding label, usually done by domain experts. Before we dive more deeply into the topic, a formal definition of what AL is gets provided:

Definition 3 (Active Learning (AL)) *A special case of ML, where data points $x \in \mathcal{U}$ are iteratively selected/queried, and labeled to extend the labeled set $\mathcal{L}$.*

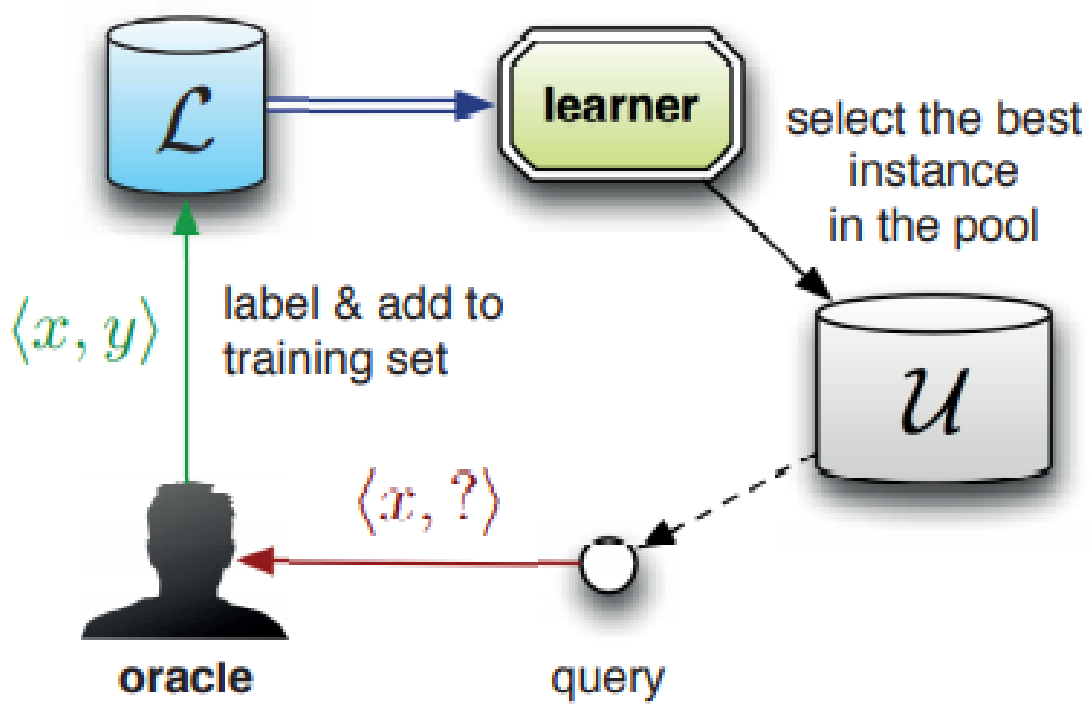


Figure 1.1.: The pool-based active learning approach searches for the most informative instance in a pool of available unlabeled data $\mathcal{U}$ and queries it. (Settles 2012)

Here, due to full information over the data set the labeling process, done by the oracle is instant. Learners can use several specific scenarios when querying, with the main scenarios being query synthesis, stream-based sampling, and pool-based sampling. In

the following, all three scenarios are described in short, where here merely the last case, selecting from a pool of instances is further used. In query Synthesis or membership queries the learner requests "label membership" for any unlabeled data instance in the input space $\mathcal{U}$, also including instances that the learner synthesizes. Query synthesis can be an appropriate approach for some problems, but labeling such arbitrary instances can be problematic, as they often lack natural semantic meaning. An alternative to synthesizing queries is steam-based sampling, where the main assumption is that obtaining an unlabeled instance is inexpensive, so it can first be sampled from the actual distribution, and then the learner can decide whether to discard it or to request its label. These instances are obtained at a time. In contrast, when large collections of unlabeled data can be gathered at once, such as is the case with many real-world learning problems, pool-based sampling is used, where the learner selects the "best" query from the set of unlabeled data $\mathcal{U}$. Pool-based sampling, illustrated in 1.1 assumes that a small set of labeled data $\mathcal{L}$ and a large pool of unlabeled data $\mathcal{U}$ is available. To mention is that this process can be done in batch or sequential, where only the latter gets considered further.

1.5. Outline thesis

This study examines various strategies for acquiring an "information-rich" data set using a small material science data set. Fractographic data was at hand, consisting of fracture images of steel materials. Each image shows a specific fracture type, either ductile or brittle. Also, every image corresponds to a certain type of material.
The result was two classification tasks for the image set, a binary and a multi-class one. Since the number of different materials was too large for this complex task, they were clustered, resulting in a 4-class classification task.
Then, features were extracted based on LBP, Haralick's features, and a novel neural network approach (VGG16). They were applied to a Random Forest model and a shallow neural network, a Multilayer Perceptron. Then, for further use, a feature set was selected that would result in a good performance. In the next step, various active learning approaches applied to the two classification tasks were compared. The impact on performance on a task, of both the model and active learning task was examined.

2. Literature review

Here, a comprehensive summary of previous research on this topic is displayed. After that, research gaps are identified and research questions are established.

2.1. General discussions in the field

The field that involves choosing which experiment to run can be broadly divided into two communities: the classical design-of-experiments (DoE) where a series of experiments are chosen in advance, and then run in batch or sequentially, and sequential design of experiments, where the results of one experiment are used to guide the choice of the next experiment. On the other hand, optimal experimental design in mathematics involves manipulating differential equations by modifying the control parameters. Fischer information and covariance matrix are used to evaluate the quality of parameter estimation in a mathematical model. Based on the stated information gain, the next measurement is made.
Reyes and Powell 2020 aim to provide a foundation in experimental decision making. Fundamental elements of experimental learning were described. Furthermore the important role of belief models: the estimate of relationships between prior research, previous experiments and scientific expertise and its underlying uncertainty, was emphasized. Also the concept of a learning policy got introduced and a policy that aims at maximizing information gain from each experiment was presented, the knowledge gradient. Three policies to reduce the number of experiments needed to achieve a certain objective value are proposed in Ling, Hutchinson, Antono, Paradiso, et al. 2017 and Völker et al. 2021 further augmented these by distance measures to accelerate the discovery process of alkali activated binders with certain properties when processed into cement. They are based on a combination of exploring high uncertainty and exploiting highly estimated target objective regions of experimental design spaces. Also a framework to estimate uncertainty in a random forest machine learning model using a resampling method, namely jackknife bootstrapping was described. These selection strategies and

the framework were applied to four material data sets each containing metadata of the inherent materials and their target values. It was shown, that the proposed methods achieved a significant decrease on the number of measurements required to find the candidate with the optimal objective value compared to randomly picking new designs. Also Lookman et al. 2019 made advances in this field of study, where a surrogate model was used to make predictions, together with a utility function that prioritizes decision making on unexplored data.

Sequential learning problems are often accompanied by small data sets, where machine learning isn't as effective. FSL is proposed in Y. Wang et al. 2020 to tackle this problem by utilizing prior knowledge on the perspective of data, the model and the algorithm.

In material science images of material textures on a microscopic scale are often of interest. On these texture images one can derive information of the material at hand. Also different magnitudes of magnifications show other textural behaviour of the material. There is an inherent link between their processing, structures and properties (Pinomaa et al. 2019).

"Textural and structural features can be regraded as 'two-view' feature sets."(Khawaled, Zibulevsky, and Zeevi 2019) They proposed a two-view classification, where they firstly decomposed fully-textured images into two layers of representation, corresponding to natural stochastic textures and structural layer, respectively. By combining textural and structural features Khawaled, Zibulevsky, and Zeevi 2019 managed to outperform already known techniques by using a shallow neural network. The classification of image features in stochastic and structural ones is common in textural computer vision. Using local binary patterns and gray local co-occurrence matrix Garg and Dhiman 2021 also extracted textural and statistical descriptors of the images in the data set at hand. They were tested on CORAL data set using different machine learning models after selecting features by particle swarm optimization algorithm.

Specifically Bastidas, Prieto-Ortíz, and Espejo-Mora 2016 used computer vision techniques to classify textures on material microstructure data sets. They used descriptors generated by gray level co-occurrence matrix, Haralick's features, and the fractal dimension on 3D fractographic images. Although the classification results were not as good as other classifications on 2D fractographic images they were significant, since experts were consulted to observe the pieces through 2D images, with a correct classification of 83% and 66%, giving the conclusion, that fractographic image classification isn't a trivial task, and a computational tool could support failure analysis. M. Bastidas-Rodriguez, F. Prieto-Ortiz, and Espejo 2016 underlines this by achieving 77% classification compared

to classifications of experts between 70% and 90%, whereas the three classes ductile, fatigue and brittle were classified by a shallow ANN with grey co-occurrence matrix-Haralick's features, texture energy laws and fractal dimension as descriptors. Ling, Hutchinson, Antono, B. DeCost, et al. 2017 and B. L. DeCost, Francis, and Holm 2017 used different depths of the pre-trained VGG16 convolutional neural network to transform images. On these feature maps various functions were applied to extract features, like the maximum, the mean, the gram matrix and a method called VLAD (vector of locally aggregated descriptors), whereas the latter scored better results, but was also more computational expensive. Bastidas-Rodríguez, F.-A. Prieto-Ortiz, and Polanía 2019 also used a pre-trained model (VGG19) for transformation purposes and achieved by further applying handcrafted features an significant increase in performance. Not only for feature extraction, but also as a direct classifier pre-trained neural networks can be fine tuned on specific data sets and utilized. For this purpose different artificial neural network architectures were compared in M. X. Bastidas-Rodriguez et al. 2020 and found by reviewing related work on deep learning used in fractographic applications, that: "The authors concluded that deep neural networks are able to achieve better generalization performance over shallow neural networks and support vector machines.", where the data sets were composed of 484 and 640 images.

2.2. Related work

When it comes to even smaller data sets, one should question where the emphasis should be put. Searching for a model architecture or novel feature generation technique, which results in better predictive power only gives small diminishing returns. Another way to improve model performance, is to extend the data on which the model is trained. One method to do this is Active learning.

Active learning is a relatively newly studied field in material sciences. In contrast to SL it doesn't aim at finding a feasible candidate for the next experimental iteration in regards to its target value, it searches the design/feature space as a whole for collecting a data set iteratively, to improve its model of the system. Since this data set contains more information over the design space, a ML model trained on this iteratively collected data set achieves a better performance. The approach relies on the use of uncertainty measures and making predictions from a surrogate model together with a utility function.

Duros et al. 2017 used machine learning controlled data acquisition for the discovery process of gigantic molecules. Through the algorithm based search it was possible to cover approximately 9 times more design space than a random search and approximately 6 times more than human experts search did and the prediction accuracy on crystallization classes was able to be increased to $82.4 \pm 0.7\%$ in contrast to $77.1 \pm 0.9\%$ for human experimenter data acquisition. Positive examples can appear in less than 1% of the data when the data is used to identify rare classes, detect malicious content, or debug model performance. By looking only at the closest neighbors to the labeled examples in each selection round, Coleman et al. 2021 took advantage of the skew in large training data sets towards classes. The premise for this approach was the presence of a large data set. Li and Guo 2013 applied a novel instance selection policy based on uncertainties output by a Gaussian Process model and a Information Density Measure to 3 different image data sets containing 13 - 102 classes. For image classification the state of the art ML approach - deep learning gets more and more studied in regards to AL. Beck et al. 2021 investigated the positive impact of data augmentation during deep AL process, while using thousands of data instances. They also stated, that: "Surprisingly, when data augmentation is included, there is no longer a consistent gain in using BADGE, a state-of-the-art approach, over simple uncertainty sampling.". Under the usage of CNNs over up to 14000 training examples Beluch et al. 2018 discussed the effect of using ensemble based strategies over single classier uncertainty sampling strategies and came to the conclusion, that ensemble uncertainties outperformed those of single classifiers.

2.3. Gap analysis

In the fractographic-ML research space solely ML models and feature generation techniques are compared (Bastidas, Prieto-Ortíz, and Espejo-Mora 2016; M. Bastidas-Rodriguez, F. Prieto-Ortiz, and Espejo 2016; Ling, Hutchinson, Antono, B. DeCost, et al. 2017; B. L. DeCost, Francis, and Holm 2017). When it comes to micrographs, especially material fractographic cases the explorative aspect of finding information-rich data points, which lies in data acquisition isn't considered. Such data acquisition strategies allow a performance boost for the same model while achieving a reduction in the number of queries needed. In other image classification research fields this is increasingly done by deep learning methods (Beluch et al. 2018; Beck et al. 2021), which are only feasible for larger data. Fractographic data sets in contrast contain smaller data quantities, and hence need to be handled differently. For instance, since the task of fracture classifica-

tion is complex, the model performance of DNN not necessarily outscores that random forests and human experts (M. X. Bastidas-Rodriguez et al. 2020), even though the resources required for training the models were immense.

2.4. Hypothesis, research question, goals for exploration

This thesis investigates the applicability of Active Learning on fractographic data and as a result, the impact of different selection strategies on the performance of machine learning models.

Goals for exploration: After applying transfer learning, LBP, GLCM, and Haralick's to extract features from fractographic data, Random Forest and Multilayer Perceptron models are compared. Selection strategies based on uncertainties in one or several model's predictions are applied. Further, these are augmented by adding a distance measures approach. Benchmarked against random sampling those strategies are examined.

Research questions:

- How does a shallow neural network perform against a random forest model on LBP features, Haralick's features, or features created by transfer learning techniques on VGG16?
- Can Active Learning be applied on fractographic data? What kind of selection strategies produce the best results on fractographic data?
- Could distance measurements on the input space be used to successfully augment those strategies?
- How does applying Random Forest and a Multilayer Perceptron regarding the two classification tasks, fracture classification and material cluster classification impact the effectiveness of those strategies?
- Does the model perform better on both tasks by successfully applying a AL strategies to one of the previous classification tasks?

Hypothesis: Active learning strategies applied to fractographic data have a significant impact on improving model performance on few data. They are further improved when they are extended by utility measures based on the input space, which can be shown by comparison with a baseline. As the data are incorporated with a priori knowledge of their variability, their specificity improves.

3. Research method

Throughout this chapter, the theoretical foundation of this work gets built. Important aspects are discussed and the methods used are explained in detail.

3.1. Machine Learning model and model evaluation

Recap that h_θ is called a Machine Learning model and belongs to a class of models $\mathcal{H}$ that attempts to generalize the training data and make predictions based on new data instances. One wants to know how good an model performs in practice, i.e. the true "risk". This true risk is called expected risk. The expected risk R is the loss measured with respect to the distribution of the data $p(x, y)$. Generally speaking the interest is to predict all true values y by a function h on the corresponding features x measured by a loss function. A loss functions as in Shalev-Shwartz and Ben-David 2014, p. 48 is defined, as any function $l : \mathcal{H} \times \mathcal{Z} \to \mathbb{R}^+$, which maps a set of hypotheses $\mathcal{H}$ and a domain $\mathcal{Z}$ to the set of non-negative real numbers. It describes the amount of failure in predicting labels by a model configuration. Loss functions commonly used for regression and classification tasks are square loss l_{sq} and $0-1$ loss l_{0-1}, respectively:

$$l_{sq}(h, (x, y)) = (h(x) - y)^2 \tag{3.1}$$

$$l_{0-1}(h, (x, y)) = \begin{cases} 0 \text{ if } h(x) \neq y \\ 1 \text{ if } h(x) = y \end{cases} \tag{3.2}$$

Explicitly, the expected risk is defined as:

$$R(h) = \int l(h(x), y)\, dp(x, y) = E[l(h(x), y)]\,.$$

One can not exactly know how good a model performs in practice, since the true distribution, the joint probability distribution $p(x, y)$ of the data is unknown, i.e. the expected risk can't be verified. Therefor the model performance gets approximated by

an estimation of the expected risk $R(h)$ on the training data. Conventionally this is the empirical risk, which is defined as

$$R_I(h) = \frac{1}{I} \sum_{i=1}^{I} l(h(x_i), y_i) \,.$$

Since, one wants to minimize the expected risk R given a set of hypotheses $\mathcal{H}$, the ultimate goal is to find a hypothesis with the lowest empirical risk. That leads to empirical risk minimization (Y. Wang et al. 2020):

$$h_I = arg\, min_{h \in \mathcal{H}}\, R_I(h) \,. \tag{3.3}$$

The minimization of empirical risk is one of the most important concepts in machine learning.

3.1.1. Models for material characterization

This section is devoted to the description of particular hypothesis spaces, which are thus also a restriction to a smaller hypothesis space $\hat{\mathcal{H}}$. By choosing the hypothesis space, which fits the problem at hand one aims to achieve a low empirical risk.

k-Means Because the k-Means algorithm locates the clusters' centers quickly, it is one of the most widely used methods of grouping objects. The clustering algorithm consists of defining a cost function, which is applied over a set of parameterized clusterings. The goal is to determine a partition (clustering) that has a minimal cost. Below, the k-means objective function G, one of the most popular clustering objectives is described. The data in k-Means are divided into disjoint sets $C_1, ..., C_k$ where each C_i is represented by a centroid μ_i. The input set $\mathcal{X}$ is assumed to be embedded in some larger metric space $(\tilde{\mathcal{X}}, dist)$ and that centroids are members of $\tilde{\mathcal{X}}$. The distance squared between the points in X and the centroid of its cluster is measured by the k-Means objective function:

$$G_{k-means}((\mathcal{X}, dist), (C_1, ..., C_k)) = \sum_{i=1}^{k} \sum_{x \in C_i} dist(x, \mu_i)^2 \,, \tag{3.4}$$

where $\mu_i = argmin_{\mu_i \in \tilde{\mathcal{X}}} \sum_{x \in C_i} dist(x, \mu_i)^2$. Due to the fact that k-means is NP-hard, it is usually computationally infeasible to find the optimal k-means solution. The following simple approximation algorithm is often used instead. Commonly, k-means clustering

specifically refers to the results of this algorithm rather than the clustering that minimizes the k-means objective cost.

k-Means

1: **input:** $\mathcal{X}$; Number of clusters k
2: **initialize:** random initial centroids $\mu_1, ..., \mu_k$
3: **repeat** until convergence:
4: for all $i \in [k]$ set cluster $C_i = \{x \in \mathcal{X} : i = argmin_j \, dist(x - \mu_j)\}$
5: for all $i \in [k]$ update $\mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$

Decision Tree A Decision tree recursively partition the feature space by defining piecewise functions on these. A popular splitting rule at it's nodes is thresholding the feature Shalev-Shwartz and Ben-David 2014, p. 250-256. More precise for the binary feature case the tree is propagated to the right or left child of the node depending if a threshold θ is exceeded by the nodes feature x_i ($\mathbb{1}_{x_i \leq \theta}$) or not. The decision tree can be thought of a splitting of the feature space $\mathcal{X}_{train} \subseteq \mathbb{R}^d$ into k instances, where k is the leaf nodes quantity of the tree.
Now the binary feature case is generalized to a real-valued feature one by reducing it to the binary feature case. Let $x_i, \ldots, x_I \in \mathcal{X}_{train} \subseteq \mathbb{R}^d$ be the instances of the training data set. Further all features $p \in [d]$ of the instances are sorted s.t. $x_{1,p} \leq \ldots \leq x_{I,p}$ and the set of thresholds is defined s.t. $\theta_{j,p} \in (x_{j,p}, x_{j+1,p}) \, \forall j \in \{0, \ldots, I\}$, where $x_{0,p} = -\infty$ $x_{I+1,p} = \infty$ are set. Finally for each p and j the binary feature $b_p = \mathbb{1}_{x_p \leq \theta_{j,p}}$ is defined.
In the following an implementation of the recursive building of a decision tree T is presented, which is the "ID3" ("Iterative Dichotomizer 3"), a popular decision tree algorithm.

$ID3(X_{train}, A)$

input: training set $\mathcal{X}_{train}$ and feature subset $A \subseteq [d]$
if all $y_i = 1$ for $(x_i, y_i) \in \mathcal{X}_{train}$:
    return a leaf node 1
if all $y_i = 0$ for $(x_i, y_i) \in \mathcal{X}_{train}$:
    return a leaf node 0
else:
    $i = argmax_{a \in A}\, Gain(\mathcal{X}_{train}, a)$
    return the trees
        $T_1 = ID3(\{\widetilde{\mathcal{X}}_{train} \subseteq X_{train} \,:\, b_i = 1\}, A \backslash \{i\})$;
        $T_2 = ID3(\{\widetilde{\mathcal{X}}_{train} \subseteq \mathcal{X}_{train} \,:\, b_i = 0\}, A \backslash \{i\})$

The algorithm works by recursive calls on the feature set and split the tree based on a **gain measure** $Gain(\mathcal{X}_{train}, i)$ on the current feature. For different algorithms different $Gain(\mathcal{X}_{train}, i)$ are used. It is composed of the cost before splitting the tree on feature i and after:

$$Gain(\mathcal{X}_{train}, i) := C(P_{\mathcal{X}_{train}}(y_i = 1)) \tag{3.5}$$
$$- P_{\mathcal{X}_{train}}(x_i = 1) C(P_{\mathcal{X}_{train}}(y_i = 1 | x_i = 1) \tag{3.6}$$
$$+ P_{\mathcal{X}_{train}}(x_i = 0) C(P_{\mathcal{X}_{train}}(y_i = 1 | x_i = 0))\,, \tag{3.7}$$

and let the first term be the cost before splitting on feature i and the second term the cost after splitting. Here three implementations are presented, where $P_{\mathcal{X}}(\cdot)$ denotes the uniform probability distribution over $\mathcal{X}$:

- **Train Error:**
  The simplest definition is the one of decrease in training error, i.e. $C(a) = min\{a, 1-a\}$.

- **Information Gain:**
  The information gain is the difference of entropy before and after the split and can be achieved by replacing the function $C(a)$ by the entropy function $-a\, log(a) - (1-a)\, log(1-a)$.

- **Gini Index:**
  In case of the gini index $C(a) = 2a(1-a)$.

To mention is, that both the information gain and gini index are smooth concave upper bounds on the train error.
By a so called technique **pruning** the number of nodes in a decision tree gets reduced, since it can be very large. The same result can be achieved by limiting the number of iterations performed in ID3. Pruning is done by bottom-up walk the tree after it is build. Each node may be replaced by a leaf node or one of it's subtrees, based on an estimate of $L_D(h)$ or some bound. A generic tree pruning procedure is shown below in pseudo code.

1: **input:** tree T, bound $f(T, m)$ (where m is the size of the sample)
2: **for each** node $j \in T$ (from leaves nodes to root node):
3: find T' which minimizes $f(T', m)$, where T' is one of the following:
4: T after replacing node j with a leaf, either 0 or 1
5: T after replacing node j with one of its subtrees
6: T
7: T = T'

Hence, if an arbitrary size of the tree is allowed it's prone to overfitting. To avoid overfitting the maximum depth is limited.

Random forest (RF) A random forest is a ensemble method composed of decision trees. Since single decision trees have poor predictive power on nonlinear and noisy data, through the danger of overfitting, an ensemble of trees trained on random draws of training data set is taken. This methodology is further described in detail in the following. First, the algorithm and the distribution of the thresholds has to be defined, these can be chosen by $ID3$ and the uniform distribution. Second, a random sub samples from the training set $\mathcal{X}$, $\tilde{\mathcal{X}}$ are sampled and and a sequence I of features of size k $I_1, I_2, \ldots \subset [d]$ is constructed by sampling uniformly at random elements. Then, the algorithm grows decision trees based on $\tilde{\mathcal{X}}$ and an element of I. If their depth is limited to k, where k is small, overfitting may be prevented on each tree, also this weakness gets overcome by using an ensemble of decision trees. Furthermore for every convex performance metric such as mae it's ensemble mean is not larger then the mean over all ensemble members Rougier 2016, this can be shown under the use of Jensen's inequality. Typically all decision trees predictions are aggregated by taking the mean.
Since the training time of each tree is short and the training time and the number of

trees that make up this model are linearly related, this model trains quickly.

Random forest with uncertainty

The uncertainty estimate presented in Ling, Hutchinson, Antono, Paradiso, et al. 2017 was shown based on empirical testing on for different material data sets to be well calibrated. It includes an explicit bias term, since due to noise in training data or unmeasured degrees of freedom they can be underestimated. The uncertainty is estimated as follows:

$$\sigma(x) = \sqrt{(\sum_{i=1}^{\mathcal{X}} max(\sigma_i^2(x), w)) + \widetilde{\sigma}^2(x)}\,, \tag{3.8}$$

where σ_i^2 denotes the sample-wise variance of the predicted value of the test point x due to training sample i and $w = min_i\sigma_i^2(x)$ is the noise threshold of σ_i^2, consisting of the minimum variance magnitude over the training set predictions. $\widetilde{\sigma}^2(x)$ in 3.8 describes the bias term by a simple decision tree limited to a depth of $\frac{log_2(S)}{2}$ to avoid overfitting. Finally the sample-wise variance σ_i^2 is defined as

$$\sigma_i^2 = Cov_j^2(n_{i,j}, t_j(x)) + (\bar{t}_i(x) - \bar{t}(x))^2 - \frac{ev}{T}\,, \tag{3.9}$$

where Cov_j is the covariance computed over the jth tree, with the number of instances $n_{i,j}$ of the training point x_i and the trees predictions $t_j(x)$, $\bar{t}_i(x)$ is the average of the trees predictions not trained on training sample i, $\bar{t}(x)$ is the average of all tree predictions, e is the euler number, v is the total variance over all trees and T is the number of trees calculated.

Artificial neural network (ANN) The concept of artificial neural networks (ANNs) or short neural networks (NNs) stems from how neurons function in the brain. neurophysiologist Warren McCulloch and mathematician Walter Pitts initially depicted this in their 1943 paper McCulloch and Pitts 1943. As a result, researchers began attempting to translate the ideas behind these networks onto computational systems, which led to the first "Hebbian network" being successfully implemented at MIT in 1954. ANNs consist of several layers, such as an input layer, hidden layers, and an output layer, each of which is composed of one or more "neurons".

The neural network can be seen as a mapping $f : \mathcal{X} \to Y$, where in the following the codomain Y is assumed to be a probability vector $[0, 1]^m$. One can simply imagine a

neuron as a placeholder for a single number that represents an intermediate value in the computation process. The input neurons are placeholders for p_1 to p_n. The output neuron holds the certainty value for each of the m predictable classes. How does the network compute this output neuron? This calculation in the neural network is also called 'forward pass'. It is done layer by layer starting with the first hidden layer. Let the number of hidden layers in this case be k. So the $n = n0$ input neurons are the input for a mapping $f_1 : I^{n_0} \to \mathbb{R}^{n_1}$, where n_1 is the number of neurons in the first hidden layer. Then the output of the first hidden layer is mapped by $f_2 : \mathbb{R}^{n_1} \to \mathbb{R}^{n_2}$ as the computation of the second hidden layer containing n_2 neurons. Analogously $f_{i+1} : \mathbb{R}^{n_i} \to \mathbb{R}^{n_{i+1}}$ then computes n_{i+1} neurons for the other hidden layers, where $i = 2, ..., k$. Finally, the output layer $f_{k+1} : \mathbb{R}^{n_k} \to [0, 1]^{n_{k+1}}$ calculates the output neurons. All in all the layer-wise forward pass can be formalized as a chain of mappings, i.e. $f = f_{k+1} \circ ... \circ f_2 \circ f_1$. In neural networks these chained functions have a specific form. One describes f_i as $f_i = \phi_i \circ a_i$, $i \in \{1, 2, 3, k + 1\}$, where $a_i : \mathbb{R}^{n_{i-1}} \to \mathbb{R}^{n_i}$ is an affine mapping, i.e. $a_i(x) = A_i x + b_i$ with $A_i \in \mathbb{R}^{n_i \times n_{i1}}$, $b_i \in \mathbb{R}^{n_i}$ and $\phi_i : \mathbb{R}^{n_i} \to \mathbb{R}^{n_i}$ is the so called 'activation function' for the neurons of ith layer.
To be precise the value of the j-th neuron in the i-th layer, denoted by $v(i, j)$ is computed as follows:

$$v(i, j) = (\phi_i)_j(< (A_i)_j, v(i - 1) > +(b_i)_j) \,,$$

where $< \cdot, \cdot >$ is denotes the inner product. The i-th layer is dependent on the values of the (i-1)-th layer, which are linearly combined with weights and added by a bias term. The weights and biases are the so called "learnable parameters" of the neural network, they are iteratively adjusted by a process called "training" of the network, where the learnable parameters are optimized in order to get better outputs. Without ϕ, the activation funtions, the forward pass would only be a chained affine mapping, therefor an affine mapping. Besides other reasons activation functions are introduced, for instance the gain of non-linearities to the neural network, which increases the complexity of the model and makes learning of highly complicated tasks like image classification possible. Many distinct activation functions exist, all with their own advantage over the other, for convolutional neural networks (CNN) especially useful in image classification the following are essential:

- Rectified linear unit (ReLU): $relu : \mathbb{R} \to \mathbb{R}$, $relu(x) = max(x, 0)$
- Sigmoid Function: $S : \mathbb{R} \to \mathbb{R}$, $S(x) = \frac{1}{1+e^x}$

- 3. Softmax Function: $softmax : \mathbb{R}^K \to \mathbb{R}^K$, $softmax(x)_j = \frac{e^{x_j}}{\sum_{j=1}^{K} e^{x_j}}$ for $j = 1, ..., K$

Training process

So how does a neural network actually learn? The part of calculating the predictions of the model is done by the already explained forward pass. These predictions get evaluated by a loss funcion, discussed in 3.3. Done by the "backword pass", the gradient of the loss function gets computed with respect to the weights and biases of the model. By using the chain rule an algorithm called "backpropagation" computes the gradients on layer at a time, iterating backwards from the last layer and chains these gradients together. The computed gradients are then used to update the weights. More specifically, during backpropagation the derivatives $\frac{\partial l}{\partial w}$ are determined, where $w \in W$ are the all learnable parameters. Then all learnable parameters of the network are updated as follows:

$$w_{new} := w_{old} - \alpha \frac{\partial l}{\partial w_{old}}\,, \tag{3.10}$$

where α denotes the so called "learning rate" of the network. The forward and backward pass through the network on all training data form one training step or epoch and can be repeated.

There are several learning algorithms for NNs. The above described in 3.10 is the so called gradient decent algorithm. Another commonly used algorithm is the Limited-memory Broyden-Fletcher-Goldfarb-Shannon (L-BFGS) algorithm is a quasi newton method, where an approximation of the hessian by the use of first derivatives ∇l_k of an objective function gets calculated Nocedal and Wright 2006[177-179]. The updating rule of the parameters is as follows:

$$w_{k+1} = w_k - \alpha_k H_k \nabla l_k\,, \tag{3.11}$$

where w_k are the learnable parameters in the kth update iteration, α_k is the learning rate and l_k is the composition of the prediction function h parametrized by w_k and the loss function l. ∇l_k is the gradient of l with respect to the parameter w_k. H_k is then updated in every iteration by:

$$H_{k+1} = V_k^T H_k V_k + \rho_k s_k s_k^T\,, \tag{3.12}$$

where $\rho_k = 1/y_k^T s_k$, $V_k = I - \rho_k y_k s_k^T$, $s_k = w_{k+1} - w_k$, and $y_k = \nabla l_{k+1} - \nabla l_k$. Only the most recent $m \in \mathbb{N}$ vector pairs $\{s_i, y_i\}$ $i = k - m, ..., k - 1$ are remembered. Below a

pseudo code of the algorithm is shown:

L-BFGS algorithm

1: **input:** $m \in \mathbb{N}$
2: **initialize:** $k = 0$, initial parameter w_0 and H_0
3: **repeat** until convergence:
4: $p_k = -H_k \nabla l_k$
5: $w_{k+1} = w_k + \alpha_k p_k$
6: **if** $k > m$:
7: discard $\{s_{km}, y_{km}\}$ from memory storage
8: $s_k = w_{k-1} - w_k$, $y_k = \nabla f_{k+1} - \nabla f_k$
9: $k = k + 1$

α_k in line 5 is chosen to satisfy the Wolfe conditions, but here its not further stated. Another powerful algorithm used for training a NN is the Adam algorithm Kingma and Ba 2017 for first-order gradient-based optimization, and which is based on adaptive estimates of lower-order moments:

Adam algorithm

1: **input:** α, exponential decay rates for the moment estimates β_1 and $\beta_2 \in [0, 1)$, stochastic objective function l, small $\epsilon > 0$
2: **initialize:** $k = 0$, initial parameter w_0, first moment vector $m_0 = 0$, second moment vector $v_0 = 0$
3: **repeat** until convergence:
4: $g_{k+1} = l_k$
5: $m_{k+1} = \beta_1 m_k + (1 - \beta_1) g_{k+1}$
6: $v_{k+1} = \beta_2 v_k + (1 - \beta_2) g_{k+1}^2$
7: $\hat{m}_{k+1} = m_{k+1}/(1 - \beta_1^{k+1})$ (bias corrected m_{k+1})
8: $\hat{v}_{k+1} = v_{k+1}/(1 - \beta_2^{k+1})$ (bias corrected v_{k+1})
9: $w_{k+1} = w_k - \alpha \hat{m}_{k+1}/(\sqrt{\hat{v}_{k+1}} + \epsilon)$
10: $k = k + 1$

Universal approximation theorem

Neural networks are one of the state of the art ML models. To understand better, why this is the case, here the mathematical statement of their great advance is presented. It theoretically shows, that networks are universal function approximators. More pre-

cise, networks with one hidden layer are able to approximate every continuous function. However, the width of the hidden layer is not bounded. To be noted is, that there exists also versions of this theorem, where the network has arbitrary depth, which is here not further discussed. There are many extensions of this theorem.
If there is only one hidden layer and only one output unit, then the set of all functions implemented by such a network with n hidden units is:

$$M_k^{(n)}(\phi) = \{h : \mathbb{R}^k \to \mathbb{R} : h(x) = \sum_{j=1}^{n} \beta_j \phi(a_j^T x - \theta_j)\},$$

where ϕ denotes the activation function. The set of all functions implemented by such a network with an arbitrarily large number of hidden units is:

$$M_k(\phi) = \cup_{n=1}^{\infty} M_k^{(n)}(\phi).$$

Then Kurt Hornik proved in Hornik 1991:

Theorem 1 *If ϕ is continuous, bounded and non-constant, then $M_k(\phi)$ is dense in $C(X)$ for all compact subsets X of $\mathbb{R}^k$, where $C(X)$ is the set of all continuous functions $X \to \mathbb{R}$.*

Convolutional neural network (CNN)
The CNN, an special case of a neural network unites three types of layers - convolutional layers, pooling layers and fully-connected layers Goodfellow, Bengio, and Courville 2016. The conventional ordering of these is that from the beginning of the network convolution and pooling alternate multiple times starting with convolution until several fully connected layers follow to calculate the output. The convolutional layers will determine the output of neurons which are connected to local regions of the input, the pooling layers will simply perform downsampling along the spatial dimensionality and the fully-connected layers work as in traditional NNs. It turned out that the rectified linear unit as activation function in the convolutional and fully-connected layers leads to promising results. Here, in the following these three layer types are discussed in more detail.

Convolutional Layer
The learnable parameters of such a layer are focused around the use of so called 'kernels'. Here each layer type is going to be explained, so lets assume that the input tensor is of size (w, h, c) for a two dimensional image, where c denotes the number of channels of the image, 1 for grayscale and 3 for RGB images and w and h are its spatial dimensions,

width and hight respectively. The kernels are usually small in spatial dimensionality, but spread the entirety of the channel dimension, i.e. a quadratic kernel can be described as a (k, k, c) tensor, with $k \in \mathbb{N}$. During the convolution, the kernel glides over the whole input tensor calculating the frobenius inner product for each position. These terms then form the output for one convolution of the convolutional layer, also called "feature map". One convolution layer can perform several convolutions with distinct learnable kernels, which in turn result in several feature maps. To be more precise, that computation gets mathematically described:
Let $F \in \mathbb{R}^{w \times h \times c}$ be the input tensor and $K_p \in \mathbb{R}^{k \times k \times c}$ be a kernel tensor. Then the corresponding feature map $G_p = (F * K_p) \in \mathbb{R}^{w_{new} \times h_{new}}$ of shape (w_{new}, h_{new}) gets computed as follows:

$$G_p[m, n] = (F * K_p)[m, n] = \sum_{l=1}^{c} \sum_{i=1}^{k} \sum_{j=1}^{k} K_p[i, j, l] F[m + i, n + j, l]$$

for all $m \in \{1, ..., w_{new}\}$ and $n \in \{1, ..., h_{new}\}$ with $w_{new} = w - k + 1$ and $h_{new} = h - k + 1$. This holds for one specific convolution that is part of the whole output of the convolutional layer. Now assume that K is a family of o kernels that are given in the convolutional layer. Then the overall output tensor $G \in \mathbb{R}^{w_{new} \times h_{new} \times o}$ results from G_p with $p = 1, ..., o$.
By the above formulas a convolutional kernel size of $k \geq 2$ would imply shrinking of the output dimensions. Convolution with "padding" and "striking" can counteract, or if desired reinforce this effect. Also, the higher importance, put on the center pixels can be antagonized. Padding means, that the spatial dimensionality of the input volume gets expanded in a certain way, for instance when the borders are padded with zero-values ("zero padding"). The stride of a convolution is the step size in which the kernel glides over the input tensor. In the equations above the pooling and stride were equal to 0 and 1 respectively. By changing this, again the dimensions of the feature maps alter. To generalize even further one could also specify the stride for each spatial dimension. By introducing the padding size p and the stride length s the following formulas for the output dimensions hold:

$$w_{new} = \frac{w - k + 2p}{s} + 1, \text{ and } h_{new} = \frac{h - k + 2p}{s} + 1 \text{ for } p, s \in \mathbb{N}.$$

When the stride is 1 the input and output volume will always have the same spatial dimensions, if one chooses for the padding $\frac{K-1}{2}$.

Pooling Layer

These layers are intended to lower the size of the representation, and thus reduce the number of parameters and as a consequence thereof the computational complexity of the model. A pooling layer operates over each feature map in the input separately, by using a specific kernel. Commonly implemented candidates are max-pooling or average-pooling, where the maximum or average output within a rectangular neighborhood reported, respectively. Normally CNNs use max-pooling layers with 2×2- kernels and a stride of 2 to reduce the feature maps down to 25% of the original size. Overlapping pooling is also possible, for instance by 3×3- kernels and a stride of 2. So, more formally the pooling operation can be described as: Let again $F \in R^{w \times h \times c}$ be the input volume, $k \in \mathbb{N}$ be the kernel size and $s \in \mathbb{N}$ be the stride. For a pooling function f the output tensor $pool(F, k, s, f) \in \mathbb{R}^{w_{new} \times h_{new} \times c}$ of a pooling layer is computed as:

$$pool(F, k, s, f)[m, n, d] = f(F[1+(m-1)s : 1+(m-1)s+k, 1+(n-1)s : 1+(n-1)s+k, d])$$

for all $m \in \{1, ..., w_{new}\}$, $n \in \{1, ..., h_{new}\}$ and $d \in \{1, ..., c\}$ with $w_{new} = \frac{w-k}{s} + 1$ and $h_{new} = \frac{h-k}{s} + 1$. Analogous to its explanation for convolution one can introduce padding strategies into that operation.

Fully Connected Layers

Fully connected layers contain a certain number of neurons that are directly connected to the neurons in the previous layer. It is to note that the input and output of these layers are one dimensional tensors, but the output of a convolution-pooling chain is a three dimensional tensor, which can be dealt with by flattening of the feature volume, i.e. all elements are rolled out into a vector. The final fully connected layer output for instance in a classification task, has as many neurons as the number of to be predicted classes.

Multilayer Perceptron (MLP)

Like CNN MLPs are a type of feedforward ANN, which means connections between its neurons don't form a cycle. There are at least three layers of nodes in a MLP: an input layer, a hidden layer, and an output layer. Except for the input nodes, each node is a neuron with a nonlinear activation function. MLPs are fully connected. Each node in one layer connects to every node in the following layer.

3.1.2. Performance metrics

Performance metrics are used to assess the quality, reliability, and performance of a given model. There are numerous statistical metrics to characterize the performance of machine learning regression and classification models. They are generally based on the comparison of an assumed ground truth y_i and the model predictions $\hat{y}_i$ for a number of instances $i \in \{1, ..., n\}$, which leads to prediction errors $e_i = \hat{y}_i - y_i$. Different performance aspects of a model are illuminated by different metrics. They complement each other when taken together concerning the same ground truth.
In the this section different metrics are discussed in regard to regression and classification tasks Vishwakarma, Sonpal, and Hachmann 2021 Grandini, Bagli, and Visani 2020.

Regression Tasks Regression tasks aim at predicting a continuous outcome variable based on the value of one or multiple variables. Mean absolute error (MAE) and root mean square error ($RMSE$) are the most commonly used error metrics for regression tasks.

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |e_i| \tag{3.13}$$

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^{n} e_i^2} \tag{3.14}$$

The MAE provides information about the average magnitude of errors, whereas the $RMSE$ magnifies larger errors and thus is more sensitive to outliers. When both the MAE and $RMSE$ are considered together information about the homogeneity and heterogeneity of errors is yielded. Since they only consider error magnitudes one should raise a metric which considers the directional distribution around the prediction, like the mean error $ME = \frac{1}{n} \sum_{i=1}^{n} e_i$. Its magnitude corresponds to the directional bias. To further characterize the prediction error distribution its standard deviation σ is considered. A small directional bias indicates an accurate model, whereas a small σ a precise one.

Classification Tasks The aim of classification tasks is the assignment of a new observation to a category among a set of categories. Classification tasks are generally divided into binary or multi-class classifications depending if the set of categories is either of size

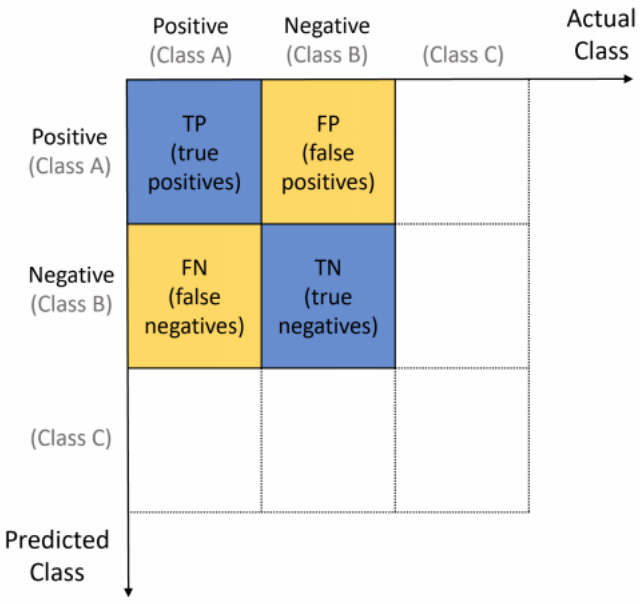


Figure 3.1.: Confusion matrix

two or greater.

For both types of classification a square matrix representing a model's performance, namely confusion matrix is a simple way of visualizing results. In 3.1 a confusion matrix for a binary classification problem (with classes A and B) is presented. In a multivariate classification problem, it can be extended by more classes (exemplarily by class C). From the information depicted in the confusion matrix, several metrics can be derived.

Accuracy (ACC) is defined as the fraction of correct predictions and the total number of predictions, i.e. it states how often the model predicts correctly. For the binary case one can write:

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.15}$$

For the multi-class case, the accuracy gets computed by its average over each class. While this metric is easy to interpret it is not feasible when dealing with imbalanced data, for these cases the so-called balanced accuracy or precision and recall are preferred. Highly populated classes would have more weight than lower populated ones.

Precision ($Prec$) denotes the fraction of class labels predicted correctly and recall (Rec) the fraction of correctly predicted class labels for each class respectively. In the binary case these fractions are only calculated for one class the "positive class" and in the multi-class case for each class. Exemplarily the equations for precision and recall for the binary case are shown below:

$$Prec = \frac{TP}{TP + FP} \tag{3.16}$$

$$Rec = \frac{TP}{TP + FN} \tag{3.17}$$

The balanced accuracy ($Acc_{balanced}$) is the arithmetic mean of the classes recalls, which is for the binary case the arithmetic mean of sensitivity and specificity.

$$Acc_{balanced} = \sum_{k=1}^{K} \frac{Rec_k}{K} \tag{3.18}$$

$$\overset{\text{binary case}}{=} \frac{sensitivity + specificity}{2} = \frac{\frac{TP}{TP+FN} + \frac{TN}{TN+FN}}{2} \tag{3.19}$$

The $F1$ score is a trade-off between precision and recall, it is denoted by their harmonic mean. Deductive, the difference between the two metrics has a huge impact on the $F1$ score, it gets skewed heavily in value towards the smaller metric.

$$F1 = 2 \cdot \frac{Prec \cdot Rec}{Prec + Rec} \tag{3.20}$$

When dealing with a multi-classification problem one can generalize the $F1$ score to its multi-class counterpart, the macro $F1$ score by using the average precision and average recall over all classes. Since the numerator is composed of averages, each class has the same impact on the result.

$$F1_{macro} = 2 \cdot \frac{Prec_{avg} \cdot Rec_{avg}}{Prec_{avg} + Rec_{avg}} = 2 \cdot \frac{(\sum_{k=1}^{K} \frac{Prec_k}{K}) \cdot (\sum_{k=1}^{K} \frac{Rec_k}{K})}{\sum_{k=1}^{K} \frac{Prec_k}{K} + \sum_{k=1}^{K} \frac{Rec_k}{K}} \tag{3.21}$$

When more weight should be assigned to one or the other the weighted F1 score with the weight parameter β should be considered.

$$F1_\beta = (1 + \beta) \cdot \frac{Prec \cdot Rec}{\beta^2 \cdot Prec + Rec} \tag{3.22}$$

Whereas in most classification problems the output of a classifier is the most likely class label, but in certain classification problems the output is a probability distribution over a set of class labels. In this case, categorical cross-entropy L is a good measure. It has to be modified by class weights when the data is unbalanced.

$$L = -log(P(|\hat{y}_i)) = -(y_i log(\hat{y}_i)) + (1 - y_i) log(1 - \hat{y}_i) \tag{3.23}$$

While probability predictions give a more nuanced view of the model performance, distinct class predictions are preferred for most purposes. Changing one to the other is obtained by thresholds. For finding the best threshold the receiver operating curve (ROC) metric, which plots the true positive rate $TPR = \frac{TP}{TP+FN}$ vs. the false positive rate $FPR = \frac{FP}{FP+TN}$ at each threshold value is commonly used. The area under the curve AUC metric can be derived from it, the closer its value is to 1, the better it is. These metrics are insensitive to changes in class distribution.

3.1.3. Validating the model

A good estimation of the true risk of the output predictor of a learning algorithm is desirable. An accurate estimation of the true risk can be obtained by using some of the labeled data as a validation set, over which one can evaluate the success of the algorithm's output predictor, i.e. its performance Shalev-Shwartz and Ben-David 2014. This procedure is called validation. By sampling an additional set of examples independent of the training set, one can estimate the true error of a predictor h by using the empirical error on the validation set as an estimator. Formally, let $V = \{(x1, y1), ..., (x_{m_v}, y_{m_v})\}$ be a set of m_v examples that are sampled randomly.

Theorem 2 *Let $\tilde{\mathcal{H}}$ be an arbitrary set of hypothesis and let the loss function l map to $[0, 1]$. Further, let V be a validation set of size m_v, sampled independently of $\tilde{\mathcal{H}}$. Then, for every $\delta \in (0, 1)$ and with probability of at least $1 - \delta$:*

$$\forall h \in \tilde{\mathcal{H}}, \qquad |R(h) - R_V(h)| \leq \sqrt{\frac{log(2|\tilde{\mathcal{H}}|/\delta)}{2m_v}}$$

This theorem states that the error on the validation set approximates the true error as long as $\tilde{\mathcal{H}}$ is not too large relative to the size of the validation set. However, if the relation counterweights towards a larger $\tilde{\mathcal{H}}$ there is a danger of overfitting, which is the terminology of poor generalization ability of models.

k-Fold Cross Validation So far, the validation procedure assumed plentiful data. But in FSL, data is scarce and one does not want to "waste" it on validation. The k-fold cross validation method gives a good estimate of the true error without wasting too many data points. In k-fold cross validation the original data set is partitioned into k subsets, or folds of size $m/k = m_v$. In each fold, the algorithm is trained on the union of

the other folds before estimating its output error using the fold. Each fold is a validation set. In the end, the true error can be estimated from the average of all these errors.

3.2. Image feature extraction

For most models, a necessary step in the ML pipeline is to extract features from the images on which the model can generate good predictions in regards to a performance metric. In contrast, a deep neural network isn't restricted to this proceeding, it could be trained directly on the image data, but only achieves good performance for suffice data. In this section, different image processing and feature extraction methods are presented. There are many different methods to extract features from images. Here the choice fell on a diversity of approaches, a textural one, a statistical one, and one based on the usage of a pre-trained CNN. In the following, by $f_p = (f_1, ..., f_I) \in \mathbb{R}^I$ the value of the feature f_p over the I training examples gets denoted, and the number of features gets set to m ($p \in \{1, ..., m\}$).

3.2.1. Grey local co-occurence matrix (GLCM) and Haralick's features

In Haralick, Shanmugam, and Dinstein 1973 "some easy computable textural features" were proposed, which are now known as Haralick's features. These features are still commonly used in image processing for classification tasks, due to their good performance. To calculate those, first the grey local co-occurence matrix (GLCM) of the image has to be calculated. The GLCM displays the spatial relationship between two pixels located at a distance d and from an angle γ from a reference pixels. As γ 0, 45, 90, and 135 degrees are used. Since the matrix does not take $-\gamma$ into account, it is not symmetric. Therefor its transposed is added Hall-Beyer 2017, resulting in a matrix V. All angle relations among the adjacent pixels to a reference pixel are calculated, showing in 4 matrices V. Finally, the Vs get normalized. Its entries at the (i, j) position can be described as:

$$G_{i,j} = \frac{V_{i,j}}{\sum_{i,j=1}^{N} V_{i,j}} ,$$

where $V_{i,j}$ is the (i, j)th entry of the matrix V (where $i, j \in \{0, ..., N-1\}$) and N is the number of rows and columns, i.e. the number of gray levels in the image.
Afterwards, based on the GLCM G, 13 Haralick's features can be derived, which can be

divided into :

- Angular Second Moment: $f_1 = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} G_{i,j}^2$
- Contrast: $f_2 = \sum_{n=0}^{N-1} n^2 (\sum_{i,j=0;|i-j|=n}^{N-1} G_{i,j})$
- Correlation: $f_3 = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} G_{i,j} \frac{(i-Mean_i)(j-Mean_j)}{Variance_i Variance_j}$,
  where $Mean_i$, $Mean_j$ and $Variance_i$, $Variance_j$ are the mean and variance of G_i and G_j respectively and are formulated as: $Mean_i = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} iG_{i,j}$ and $Variance_i = \sum_{i,j=1}^{N} G_{i,j}(i - Mean_i)^2$
- Variance: $f_4 = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} (i - Mean_i)^2 G_{i,j}$,
- Homogeneity (Inverse Difference Moment): $f_5 = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \frac{G_{i,j}}{1+(i-j)^2}$
- Sum average: $f_6 = \sum_{i=2}^{2N} iG_{x+y}(i)$, where $G_{x+y}(i) = \sum_{i=1}^{N} \sum_{j=1,i+j=k}^{N} G_{i,j}$
- Sum variance: $f_7 = \sum_{i=2}^{2N} (i - Sum\, average)^2 G_{x+y}(i)$
- Sum entropy: $f_8 = \sum_{i=2}^{2N} G_{x+y}(i) log(G_{x+y}(i))$
- Entropy: $f_9 = -\sum_{i=0}^{N-1} \sum_{j=0}^{N-1} G_{i,j} log(G_{i,j})$
  Here, since $log(0)$ is undefined, $0log(0) = 0$ is assumed.
- Difference variance: $f_{10} = \sum_{i=2}^{2N} G_{x+y}(i) log(G_{x+y}(i))$, where $G_{x-y}(i) = \sum_{i=1}^{N} \sum_{j=1,|i-j|=k}^{N} G_{i,j}$
- Difference entropy: $f_{11} = -\sum_{i=0}^{N-1} G_{x-y}(i) log(G_{x-y}(i))$
- Information measures of correlation:
  $f_{12} = \frac{HXY-HXY1}{max(HX,HY)}$ and $f_{13} = \sqrt{1 - exp(-2(HXY2 - HXY))}$,
  where $HX = -\sum_{i=0}^{N-1} \sum_{j=0}^{N-1} G_{i,j} log(\sum_{j=0}^{N-1} G_{i,j})$,
  $HY = -\sum_{i=0}^{N-1} \sum_{j=0}^{N-1} G_{i,j} log(\sum_{j=0}^{N-1} G_{i,j})$, $HXY = f_9$,
  $HXY1 = -\sum_{i=0}^{N-1} \sum_{j=0}^{N-1} G_{i,j} log(\sum_{j=0}^{N-1} G_{i,j} \sum_{i=0}^{N-1} G_{i,j})$ and
  $HXY2 = -\sum_{i=0}^{N-1} \sum_{j=0}^{N-1} (\sum_{j=0}^{N-1} G_{i,j} \sum_{i=0}^{N-1} G_{i,j}) log(\sum_{j=0}^{N-1} G_{i,j} \sum_{i=0}^{N-1} G_{i,j})$

M. Bastidas-Rodriguez, F. Prieto-Ortiz, and Espejo 2016 used both the average and the range of the features over all four directions, calculated as:

$$Average(f_p) = \frac{1}{4} \sum_{\gamma=1}^{4} f_{p,\gamma}$$

$$Range(f_p) = max_\gamma(f_{p,\gamma}) - min_\gamma(f_{p,\gamma})$$

3.2.2. Local binary patterns (LBP)

In He and L. Wang 1990 local binary patterns (LBP) were introduced, to give an additional statistical descriptor, one, where local texture information of a pixel and its neighborhood is characterized.
Given a neighborhood of 3×3 pixels, which will be denoted by a set containing nine elements: $V = \{V_0, V_1, ...V_8\}$, where V_0, represents the intensity value of the central pixel and V_i, $i \in \{1, 2, ...8\}$ is the intensity value of the neighboring pixel i.

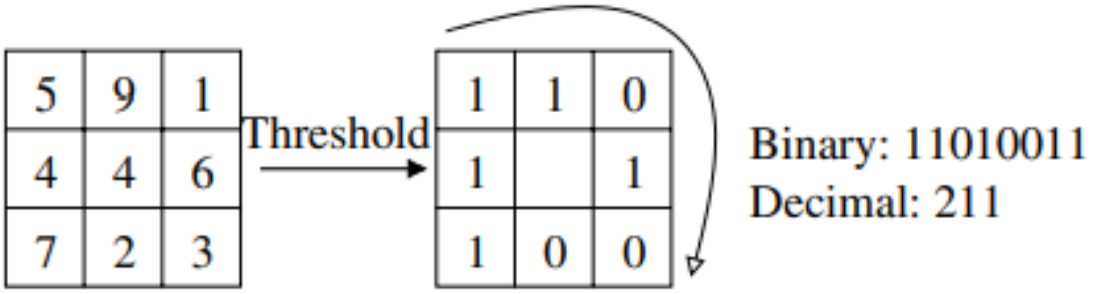


Figure 3.2.: Example of transforming a $(8, 1)$ neighborhood to a texture unit with the decimal number Ahonen, Hadid, and Pietikäinen 2004

The corresponding texture unit is defined by a set containing eighth elements, $TU = \{E_1, E_2, ...E_8\}$,where its elements correspond to the neighboring pixels positions and are determined as:

$$E_i = \begin{cases} 0, & \text{if } V_i < V_0 \\ 1, & \text{if } V_i \geq V_0 \end{cases} .$$

The ordering of the neighbor pixel is not important, but has to be consistent, here in 3.2 it is clockwise starting from the left upper position for the first neighbor pixel and ending at the right position for the eighth neighbor pixel. These texture units transform to a binary number: $BN = E_1E_2...E_8$. In the next step a binary number, corresponding to the central pixel gets converted into an decimal number:

$$Decimal = \sum_{i=1}^{8} 2^{8-i} E_{8-i} \, .$$

By the recipe described above, every pixel gets a decimal number assigned to it. A

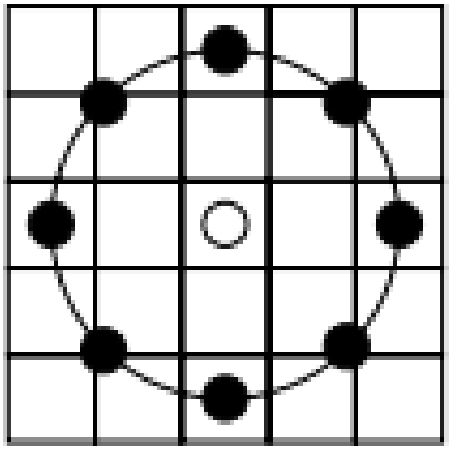


Figure 3.3.: The circular $(8, 2)$ neigbourhood. The pixel values are bilinearly interpolated whenever the sampling point is not in the center of a pixel. Ahonen, Hadid, and Pietikäinen 2004

distribution representing the quantity of local micropatterns occurring in the image, gets approximated by a histogram of the decimal numbers, where its classes build a feature set:

$$f_i = \sum_{p=1}^{P} 1_{Decimal_p=i} \; i = 0, 1, ...n-1 \,,$$

in which n is the number of different decimal numbers, which can get created ($n = 255$ for the above case), p is the number of a pixel and P the total number of pixels.
Analog to the $(8, 1)$ neighborhood, shown above and in 3.2, LBP can be also done for other neighborhoods like $(8, 2)$ shown in 3.3, where the first number corresponds to the quantity of neighbors and the second to their distance to the central pixel.

3.2.3. Transfer learning with VGG16

Transfer learning refers to storing experience, while solving one problem and applying it to another. This paragraph would fit well in the section describing models since knowledge is stored in a model. However, in this case, the model is only used to transform images into features. There were two steps for the feature extraction workflow using pre-trained convolutional neural networks (CNN): the transformation of the image by the CNN, and texture featurization of the transformed images (Ling, Hutchinson, Antono, B. DeCost, et al. 2017). The trained VGG16 neural network was used, see an illustration

in 3.4, where the individual images operate as the input, the red bars indicate convolutional layers, the blue ones Max Pooling layers, the light green ones Fully Connected layers and the dark green bar the output layer. The output of the convolutional layers $C_{1,1}$ - $C_{5,3}$ can be used to transform the images into image feature maps, where the fully connected layers were not used at all. The output from convolutional layers con-

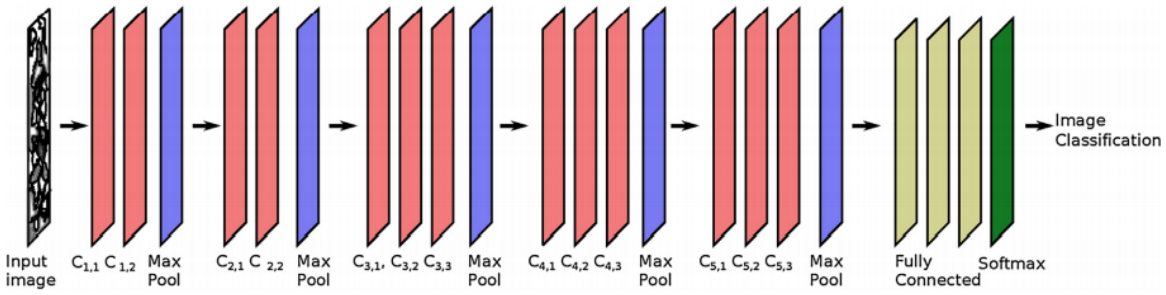


Figure 3.4.: VGG16 CNN architecture (Ling, Hutchinson, Antono, B. DeCost, et al. 2017)

tain information about higher-level features and more complex textures the later they are in the network. There were four translation-invariant feature extraction strategies proposed in Ling, Hutchinson, Antono, B. DeCost, et al. 2017, namely: VLAD, Mean, Max, and Gram. These are presented below, where the features extracted by the Mean strategy enable better predictive power over those of Max. The calculation of VLAD and Gram features comes with the cost of vast memory cost. Also, the large amount of features generated by Gram makes a data-driven model prone to overfitting.

- **VLAD** - In the vector of locally aggregated descriptors encoding strategy the output filters from a given convolutional layer are clustered by a k-Means algorithm with a given number of clusters, in Ling, Hutchinson, Antono, B. DeCost, et al. 2017 it was set to 32. Then the centers of each luster was calculated. The resulting feature-vector for each image consists of all differences between the image output filters and the cluster centroids and is therefor of length $N \cdot M$, where N is the number of filter and M the number of cluster. One downside of this strategy is that the clusters have to be calculated for a given dataset, which can be memory intensive and if this dataset gets updated also the cluster calculation should to be repeated.
- **Mean** - The output from a given convolutional layer $F = (F_1, ..., F_N)$, a tensor

gets averaged over all spatial dimensions j, resulting in a feature vector $F_{mean} = (mean_j\, F_{1,j}, ..., mean_j\, F_{N,j})$ of length N, the number of filters in the output layer.

- **Max** - Similar to the previous strategy, the maximum value of the filter from the given convolutional output layer was taken, resulting in $F_{max} = (max_j\, F_{1,j}, ..., max_j\, F_{N,j})$.
- **Gram** - For each combination of filters from a given convolutional layer the Gram matrix gets computed $G_{i,j} = F_i \cdot F_j^T$ and a feature vector of size N^2 gets calculated by the Gram matrices determinants.

3.2.4. Feature scaling

Usually, the features must be further pre-processed in order to achieve better model performance. For instance, the resulting features of the LBP method have completely different scales. In particular, ML algorithms that work on distances give higher value features more weight and thus degrade their performance. Feature normalization can overcome this problem. It can improve the estimation error and the runtime of the learning algorithm (Shalev-Shwartz and Ben-David 2014). There are many ways to perform feature normalization. Now, two common technique for feature transformation are listed.

Standardization

This transformation converts all features to have a zero mean and unit variance. Formally, let $\bar{f} = \frac{1}{I}\sum_{i=1}^{I} f_i$ and $v = \sum_{i=1}^{I}(f_i - \bar{f})^2$ be the mean and empirical variance of the features over all examples, respectively. Then, the new features f_{new} are set by:

$$f_{new} = \frac{f - \bar{f}}{\sqrt{v}}\,.$$

Min-max normalization

It rescales the range of features based on $[-1, 1]$ or $[0, 1]$ and is the simplest method. The target range is determined by the data, with the general formula for a min-max range of $[0, 1]$ being:

$$f_{new} = \frac{f - \min(f)}{\max(f) - \min(f)}\,.$$

The biggest drawback of this method is that it is prone to outliers.

3.3. Include A-priori knowledge to improve data analysis for small data

In FSL, the size of the available training data is small. Thus FSL problems are much harder in comparison to learning with sufficient training sample sizes. Here, approaches to tackle the FSL problem are represented Y. Wang et al. 2020. These approaches can be summarized under the perspectives of Data, Model, and the learning algorithm.
Let $\hat{h} = arg\,min_{h\in\mathcal{H}} R(h)$ be the hypothesis which minimizes the expected risk and be the best approximation for $h^* = arg\,min_h R(h)$, the optimal hypothesis. Since h^* is unknown and only a limited amount of data is obtainable it is best achieved to be approximated by empirical risk minimization, hence h_I. This leads to the total error stating how well the optimal hypothesis h^* can be approximated by an empirical risk minimizer h_I. It can be decomposed into an approximation error and an estimation error:

$$E[R(h_I) - R(h^*)] = \underbrace{E[R(\hat{h}) - R(h^*)]}_{\varepsilon_{app}(\mathcal{H})} + \underbrace{E[R(h_I) - R(\hat{h})]}_{\varepsilon_{est}(\mathcal{H},I)} . \tag{3.24}$$

The approximation error $\varepsilon_{app}(\mathcal{H})$ states how close the best hypothesis h^* can be approximated by a function in $\mathcal{H}$, and the estimation error $\varepsilon_{est}(\mathcal{H}, I)$ states the effect of minimizing the empirical risk $R_I(h)$ instead of the expected risk $R(h)$. The total error is affected by $\mathcal{H}$ (hypothesis space), I (number of examples in $\mathcal{Z}_{train}$) and the training set $\mathcal{Z}_{train}$ itself.
Hence, learning to reduce the total error can be attempted from the perspectives of data, which provides $\mathcal{Z}_{train}$; model, which determines $\mathcal{H}$; algorithm, which searches for the optimal $h_I \in \mathcal{H}$ that fits $\mathcal{Z}_{train}$. To be pointed out is that the problem specifics have to be considered when applying these perspective inherent methods. Essentially, the reduction of error can be achieved by altering the data, the model, or the learning algorithm. Besides the perspective on data, AL also focuses on the data part, but by iteratively adding new data points to the training set, which in turn improves the hypothesis and its risk. Here, for gaining an overall understanding of these three perspectives, they get discussed in the following and some specific methods get presented.

3.3.1. Data augmentation

These methods use prior knowledge to augment $\mathcal{Z}_{train}$, and increase the number of samples from I to $\widetilde{I}$, where $\widetilde{I} >> I$. In other words the supervised information in experience

E is getting enriched. Standard machine learning models and algorithms can then be used on the augmented data, and a more accurate empirical risk minimizer $h_{\widetilde{I}}$ can be obtained, also though sufficient training data $\varepsilon_{est}(\mathcal{H}, I)$ can be reduced to $\varepsilon_{est}(\mathcal{H}, \widetilde{I})$, and hence the empirical risk minimizer $h_{\widetilde{I}}$ provides a good approximation $R(h_{\widetilde{I}})$ to $R(h^*)$.
In the following data augmentation methods from hand-crafted rules up to more advanced methods are reviewed.

Hand-crafted rules Data augmentation via hand-crafted rules can introduce different kinds of invariance for the model to capture. One can use translation, flipping, shearing, scaling, reflection and rotation on images. Designing these rules is heavily domain knowledge dependent and requires expensive labor cost. Also, it is unlikely that one can picture all possible invariance.

Learned transformation The original examples are duplicated into several samples which are modified by a transformation rule. The transformation is the prior knowledge E. For example each sample can be aligned in correspondence to other samples from a similar class. This results into a set of geometric transformations applied to each sample x_i, y_i forming a large dataset. Similarly, a set of auto-encoders from a similar class can be learned, each representing one class inherent variability. Adding the variation to x_i generates new samples.

Borrow data from other data sets By applying this strategy samples from other data sets are borrowed and are adapted to be like samples from the original D_{train} to which they are augmented. The data sets used for this strategy can be divided into unlabeled and labeled data sets accompanied by specific approaches.
On the other hand dealing with similar data sets sample pairs from other data sets are aggregated. It has to be assumed that the underlying hypothesis applies to all classes. The similarity of all classes has to be computed in regards to all other classes. Later, new samples can be generated by weighted averaging sample pairs of classes, where the weights are the similarity measures.

Taking samples from a weakly labeled or unlabeled data set When dealing with unlabeled data sets, usually large amounts of unlabeled data are available, which can contain enormous variations of samples. The challenge is to label samples correctly or find the samples with desired corresponding label to augment $\mathcal{Z}_{train}$. Label propagation can be used to deal with this matter directly, where a semi-supervised ML algorithm

assigns labels based on already labeled data points. These assignments are not unique and thus don't yield always correct labels. Another way, is manually assigning labels that are known, but not assigned.

3.3.2. Model refinement

These methods use prior knowledge in E to constrain the complexity of $\mathcal{H}$, which results in a much smaller hypothesis space $\hat{\mathcal{H}}$. For this smaller $\hat{\mathcal{H}}$, $\mathcal{Z}_{train}$ is sufficient to learn a reliable h_I.
The hypothesis space should be determined such that the difference between the optimal hypothesis $\hat{h} \in \hat{\mathcal{H}}$ attained in $\hat{\mathcal{H}}$ and the best hypothesis h^* is small. According to prior knowledge the cropped hypothesis space is unlikely to contain the optimal hypothesis. One can choose a small $\hat{\mathcal{H}}$ with only simple models making training on $\mathcal{Z}_{train}$ easier. However, this approach is not feasible, though the often complicated nature of real-world problems resulting in a large $\epsilon_{app}(\hat{\mathcal{H}})$ in 3.24. Hence, a $\hat{\mathcal{H}}$, which is large enough is preferred, leading to a more reliable empirical risk minimizer and risk reduction of overfitting.
Different types of prior knowledge can be used, as methods restricting $\mathcal{H}$ can be classified into the following.

Multitask Learning When multiple related tasks are present, multitask learning learns these tasks simultaneously, by ether **sharing some parameters** or **tying the parameters** by for example regularizing them in regards of their pairwise differences.

Embedding learning When doing embedding learning each sample $x_i \in \mathcal{X} \subset \mathbb{R}^m$ is mapped to a lower-dimensional $\tilde{x}_i \in \tilde{\mathcal{X}} \subset \mathbb{R}^d$, where $d < m$. In the embedded space $\tilde{\mathcal{X}}$ similar samples are closer together and dissimilar ones are more easily kept apart. A smaller hypothesis space $\hat{\mathcal{H}}$ can then be constructed for which fewer training samples are required for finding a good h_I. One can use the same embedding function or two different functions for the training and test samples. In another case a similarity function, which measures the similarities between the two mappings in $\tilde{\mathcal{X}}$ has to be defined.

Learning with external memory When learning with external memory knowledge from $\mathcal{Z}_{train}$ is extracted and stored in an external memory. Each new sample in $\mathcal{X}_{test}$ gets represented by a weighted average from the memory extracted contents and gets therefor limited, thus reducing the size of $\mathcal{H}$.

Generative Modeling With the help of prior knowledge and the observed data $\mathcal{Z}_{train}$ generative modeling methods estimate the probability distribution of x, $p(x)$.

3.3.3. Algorithm optimization

These methods use prior knowledge to search for the $\theta \in \Theta$ which parameterizes the best hypothesis $h^*_\theta \in \mathcal{H}$.

Prior knowledge alters the search strategy by providing a good initialization, or guiding the search steps. For the latter, the resultant search steps are affected by both prior knowledge and empirical risk minimization Y. Wang et al. 2020.
For example at the nth iteration $\theta_n = \theta_{n-1} + \alpha\theta_{n-1}$, where $\alpha\theta_{n-1}$ is the update on parameters θ_{n-1} with the learning rate α. In the special case of gradient decent, θ is updated at the nth iteration as

$$\theta_n = \theta_{n-1} - \alpha_n \nabla_{\theta_{n-1}} l(h(x_n, \theta_{n-1}), y_n)\,, \tag{3.25}$$

where α_n denotes the stepsize. Assume θ_0 is the initialisation of θ 3.25 can be reformulated as

$$\theta_n = \theta_0 \sum_{i=1}^{n} \alpha\theta_{i-1}\,. \tag{3.26}$$

When supervised information is rich, this means there are many design variables x_i with a known outcome y_i, and hence there are enough training samples to update θ and to find a α that is appropriate to do so. However, in FSL, the provided $\mathcal{Z}_{train}$ is not large enough to obtain a reliable empirical risk minimizer h_I. Below methods using prior knowledge for providing a good initial parameter θ_0, or output search steps by learning an optimizer are provided to bypass this problem arising from little data.

Refining existing parameters An initial model parameter θ_0 is learned from similar tasks and refined to θ by the training data $\mathcal{Z}_{train}$. The idea behind this procedure is that θ_0 captures a general structure inherent to the problems data $\mathcal{Z}$ and therefore can be adapted by the much smaller $\mathcal{Z}_{train}$ to $\mathcal{Z}$.
For example a convolutional neural network (CNN) can be trained on a large data set like imageNet and then further fine tuned. That can be done by **regularization** to prevent overfitting on the small training set $\mathcal{Z}_{train}$. This method is popularly used in practice. Some examples for regularization are: the so called **early stopping**, where the existing

data is further divided into a new training set $\widetilde{\mathcal{Z}}_{train}$ and a validation set $\mathcal{Z}_{validation}$ to monitor the learning progress. The learning is stopped when no more improvement in performance can be perceived on $\mathcal{Z}_{validation}$. **Selectively updating** where θ_0 is updated partly, **updating related parts** of θ_0 simultaneously with the same information and **Model regression networks** are also regularization methods.
When many models trained on related tasks are available, one procedure is to just to **aggregate** the existing parameters to one parameter.
Since the pre-trained parameters θ_0 may not be enough to depict $\mathcal{Z}_{train}$, an additional parameter δ is **learned** while the existing θ_0 is **fine-tuned** yielding the model parameter (θ_0, δ).

Refining Meta-Learned Parameter The algorithm learns on different related data sets simultaneously by usage of gradient decent.

Learning the Optimizer In contrast to the other methods it learns directly an optimizer based on specific tasks, which can be directly applied on the new task.

Above specific methods to reduce the complexity of the hypothesis space $\mathcal{H}$ were introduced. While these methods can be useful for improving data analysis by including A-priori knowledge, they weren't used directly in this study due to not blowing its scope.

3.4. Active learning

As opposed to other machine learning methods, AL focuses on preparing an adequate set of labeled images for data analysis, while other approaches try to improve the data analysis directly. AL was already briefly discussed in the Introduction part 1.4 of this study, whereas 1.1 describes one AL algorithm iteration. This section is dedicated to the functioning of an AL algorithm in more detail. A discussion of different query strategies is presented, since the query part of the algorithm is crucial.

3.4.1. Active learning algorithm

A general active learning algorithm is presented below. From this point forward, let X_A^{π} denote the best instance that the utility measure A would select for querying. A

key component of the algorithm below concerning designing an active learning system is line 4, where the instance to query next is chosen. The repetitive applying of a utility measure in an AL algorithm determines a query or selection strategy. Based on a query strategy, an active learner determines which instance is "best" to query. But the best in what sense? The following sections will deal with this issue.

1: **input:** $\mathcal{L}, \mathcal{U}$
2: **for** $t = 1, 2, ...$:
3: $h_\theta = train(\mathcal{L})$
4: select $x^* = X_A^\pi \in \mathcal{U}$
5: query the oracle to obtain label y^*
6: add (x^*, y^*) to $\mathcal{L}$
7: remove x^* from $\mathcal{U}$

Instances can be queried or discarded in a variety of ways. A bias random decision may be made based on a measure of utility or information content by making instances of higher utility more likely to be queried. Another approach is to compute an explicit region of uncertainty, that is, all the instances within the instance space that are still ambiguous to the learner. A further approach is to request instances from a region that is still unknown to the set of hypotheses consistent with the current labeled training set. However, calculating this uncertainty region fully and explicitly is computationally expensive, and it must be maintained after each new query. Thus, in practice, approximations are used. There also exist other computational, but effective approaches like minimizing expected future error of a learner under consideration of possible queries, or indirectly over its variance. This section is devoted to the different kinds of policies of how instances can be queried Settles 2012.

3.4.2. Uncertainty sampling

One needs a way to measure the uncertainty of candidate queries in the pool of unlabeled instances. For binary classification, the "closest to the decision boundary ($probability = 0.5$ for both classes)" heuristic will suffice. But, a more general measure of uncertainty or information content is needed when dealing with problems and models with posterior distributions over three or more labels or with multiple output structures.

Least Confident The simple uncertainty sampling strategy is to examine the instance where the predicted outcome is the least confident:

$$X^{\pi}_{LC} = argmin_x P_{h(\theta)}(\hat{y}|x) = argmax_x 1 - P_{h(\theta)}(\hat{y}|x)\,, \tag{3.27}$$

where $\hat{y} = argmax_y P_{h(\theta)}(y|x)$, the prediction with the highest posterior probability under the model $h(\theta)$, with parameter θ. In other words, this strategy prefers the instance whose most likely labeling is actually the least likely to be labeled among the unlabeled instances available for querying, which are as of now all unlabeled instances $\mathcal{U}$. This uncertainty measure can be interpreted as the expected 0-1 loss, i.e., the model's belief that x was mislabeled. One disadvantage of this approach is that it only takes into account the best prediction. As a result, it effectively discards the other parts of the posterior distribution.

Margin A different uncertainty sampling strategy, which addresses the shortcoming of the least confident strategy by incorporating the second-best labeling in its assessment, is based on the output margin:

$$X^{\pi}_{M} = argmin_x P_{h(\theta)}(\hat{y}_1|x) - P_{h(\theta)}(\hat{y}_2|x) = argmax_x P_{h(\theta)}(\hat{y}_2|x) - P_{h(\theta)}(\hat{y}_1|x)\,, \tag{3.28}$$

where $\hat{y}_1$ and $\hat{y}_2$ are the first and second most likely predictions under the model, respectively. Intuitively, a learner knows how to distinguish between the two most probable alternatives, if their margin is wide. However, instances with small margins are more ambiguous, and knowing the true labeling should help the model distinguish between them more effectively. Yet, the margin approach ignores much of the output distribution for problems with many alternatives.

Entropy Entropy Shannon 1948, usually denoted by H, is perhaps the most general and often found uncertainty sampling strategy:

$$X^{\pi}_{H} = argmax_x H_{h(\theta)}(Y|x) = argmax_x - \sum_y P_{h(\theta)}(y|x) log(P_{h(\theta)}(y|x))\,, \tag{3.29}$$

where y ranges over all possible labelings of x. It measures the average amount of information contained in a variable. Therefore, it is often referred to as a measure of impurity or uncertainty in machine learning. This strategy can be understood as the expected log-loss, meaning the amount of bits required to "encode" the posterior label

distribution of a model.

Remark In practice, uncertainty sampling is probably the most popular active learning strategy. Perhaps the greatest appeal to uncertainty sampling in active learning is its intuitive appeal combined with the ease of implementation since there is no significant engineering overhead involved in most uncertainty-based utility measures. As long as the learner can supply a probability score together with its predictions, any of the measures can be applied with the learner acting as a "black box." It is possible to use standard classification or inference procedures, which leaves the choice of learning algorithm fairly open. On the other hand, uncertainty sampling also comes with its downsides. Despite the speed and simplicity of the approach, the utility scores are based on the output of a single hypothesis. In addition to that, the hypothesis is often trained with very little data and this data is also inherently biased by the active sampling strategy, thus making the learner short-sighted.

3.4.3. Hypothesis space search

Let $V \subseteq \mathcal{H}$ denote the version space, that is, the subset of hypotheses that are consistent with the training data. In more detail, the version space represents the hypotheses that can explain the observed training data equally well. To characterize the data or predict the future, a learning algorithm must choose which hypotheses to use. The area of the version space V shrinks as more data is labeled if one supposes that a function is learned, which can perfectly express the data. Consequently, the set of hypotheses therein will approximate the underlying target function more and more precisely. This suggests that an active learning algorithm should minimize V as quickly as possible. To minimize the number of "legal" hypotheses under consideration, here a directed search through hypothesis space testing unlabeled instances gets conducted. Conceptually, this approach differs from uncertainty sampling, which bases query decisions on the confidence of a single hypothesis. Labeled data instances impose constraints on V in the hypothesis space, leading to a duality between the feature space F and the hypothesis space $\mathcal{H}$. Practically speaking, there are a few issues with this, for example, the version space may be infinite. Active learning based on disagreements is compelling because it queries based on different choices of hypotheses, unlike uncertainty sampling. Next, these assumptions get relaxed, since it can be problematic to measure disagreement among all hypotheses in V, even if those get approximated by a smaller subset. Also, the possibility that V is not properly defined due to noisy data can be a problem, which

should be fixed. Here it is aimed to query the "most informative" instance x.

Query by committee The query by committee (QBC) algorithm in its original formulation sampled two random hypotheses from the version space and relied on a binary concept of disagreement at each iteration. However, QBC has come to refer to any disagreement-based approach that uses a "committee" - or ensemble - of hypotheses, which will be denoted by C. In practice, these can take a variety of forms. All that is needed is a method for gathering hypotheses in the committee, and a heuristic to measure disagreements between the members. As a matter of fact, it is quite natural to think of a committee as being constructed using generic ensemble learning algorithms. There is no general rule of thumb regarding the number of committee members to use. Several heuristics are available for describing disagreement in classification tasks, but only two dominant ones are examined. In the first case, uncertainty measures are essentially generalized by committee. For example, vote entropy is defined as:

$$X_{VE}^{\pi} = argmax_x - \sum_y \frac{vote_C(x,y)}{|C|} log(\frac{vote_C(x,y)}{|C|}) \,, \tag{3.30}$$

where all possible labelings y are considered, $vote_C(x,y) = \sum_{h_\theta \in C} 1_{h_\theta(x)=y}$ is the number of "votes" that the label y receives for x among the hypotheses in committee C, and $|C|$ denotes the committee size. It is also possible to define a "soft" vote entropy that considers the level of confidence of each committee member:

$$X_{SVE}^{\pi} = argmax_x - P_C(y|x) log(P_C(y|x)) \,, \tag{3.31}$$

where $P_C(y|x) = \frac{1}{|C|} \sum_{h_\theta \in C} P_{h_\theta}(y|x)$ is the average probability that y is the correct label according to the committee. As a result, these disagreement measures are basically Bayesian versions of entropy-based uncertainty sampling from 3.29 based on vote $P_C(y|x)$ instead of just a single hypothesis. Analogous committee-based generalizations can be arranged for the least-confident 3.27 and margin 3.28 metrics as well. Another disagreement measure is based on the Kullback-Leibler (KL) divergence Kullback and Leibler 1951, which is an information-theoretic measure of the difference between two probability distributions. In this case, disagreement is defined as the average divergence between each committee member's prediction and the consensus:

$$X_{KL}^{\pi} = argmax_x \frac{1}{|C|} \sum_{h_\theta \in C} KL(P_{h_\theta}(Y|x) | P_C(Y|x)) \,, \tag{3.32}$$

where KL divergence is defined by $KL(P_{h_\theta}(Y|x)|P_C(Y|x)) := \sum_y P_{h_\theta}(y|x)log(\frac{P_{h_\theta}(y|x)}{P_C(y|x)})$. The maximum uncertainty strategy proposed by Ling, Hutchinson, Antono, Paradiso, et al. 2017 can also be categorized as a query by committee selection strategy. As described in 3.1.1 an ensemble of decision trees was created to output a label prediction and an uncertainty in this prediction. Disagreement is defined as the consensus uncertainty:

$$X^{\pi}_{MU} = argmax_x \sigma(x)\,, \tag{3.33}$$

where $\sigma(x)$ is the uncertainty measure over all committee member predictions. Derived from that disagreement mechanism a strategy can be defined as a combination of the consensus uncertainty and committee prediction, and therefore for instance a combination of the least confident method for a committee and maximum uncertainty strategy, denoted as maximum likelihood of improvement:

$$X^{\pi}_{MLI} = argmin_x P_{h(\theta)}(\hat{y}|x) - \sigma(x)\,, \tag{3.34}$$

where again $\hat{y} = argmax_y P_{h(\theta)}(y|x)$.

Remark There is a key conceptual difference between vote entropy and KL divergence in measuring disagreement. The vote entropy method 3.30 only considers PC and cannot distinguish uniform from non-uniform hypothesis spaces, whereas KL divergence can. The consensus label is uncertain, but this is because all committee members are in agreement that it is uncertain. However, KL divergence 3.32 would likely support the notion of disagreement in a situation with uncertain predictions with wildly varying consensus estimates among committee members.

3.4.4. Minimizing expected error and variance

Expected error reduction Perhaps one should not necessarily be concerned about the models' certainty or the correctness of their hypotheses. Now, a shift in attention away from what the learner thinks about instances to what kinds of decisions it might make in the future is made. In particular, one wants a learner to choose instances that, once it knows the corresponding label, are most likely to reduce future error. That has the disadvantage that the learner doesn't know what it will be asked and it doesn't know what its error will be until after it has received an answer and updated its hypothesis. An expectation value is obtained from identifying all the possible outcomes of a series of

actions, determining their values and probabilities, and computing a weighted sum for each one: $E[Y] = \sum_y P(y)y$. Choosing an action that yields the lowest value should be a rational decision. In order to calculate the expected error, one needs two probability distributions, the probability of the oracle's label y in response to query x, and the probability of the learner making an error on another instance x' once the answer is known. Unfortunately, neither of these is really known, but one can use the posterior distribution of the model as a good approximation. If a large unlabeled pool $\mathcal{U}$ is available, the learner can attempt to minimize the expected error over it, assuming that it is representative of the distribution using a part of it as a sort of validation set.
The decision-theoretic utility measure would look like this to minimize the expected classification error (or 0-1 loss) over the unlabeled data:

$$X^{\pi}_{ER} = argmin_x E_{Y|h_\theta,x}\left[\sum_{x'\in\mathcal{U}} E_{Y|h_{\theta+},x'}[y = \hat{y}]\right] \tag{3.35}$$

$$= argmin_x \sum_y P_{h_\theta}(y|x)\left[\sum_{x'\in\mathcal{U}} 1 - P_{h_{\theta+}}(\hat{y}|x')\right], \tag{3.36}$$

where $h_{\theta+}$ refers to the a new model after it has been re-trained using a new labeled set $\mathcal{L}\cup(x, y)$, which results after adding the candidate x and its hypothetical oracle response y. In this case, the goal is to reduce the expected total number of incorrect predictions. Equivalent to the expected total future output uncertainty over U is the minimization of the expected log-loss:

$$X^{\pi}_{LL} = argmin_x E_{Y|h_\theta,x}\left[\sum_{x\in\mathcal{U}} E_{Y|h_{\theta+},x}[-log(p_{h_{\theta+}}(y|x))]\right] \tag{3.37}$$

$$= argmin_x \sum_y P_{h_\theta}(y|x)\left[\sum_{x\in\mathcal{U}} -\sum_{y'} P_{h_{\theta+}}(y'|x)log(P_{h_{\theta+}}(y'|x))\right], \tag{3.38}$$

However, due to concerns of computational complexity, may make these approaches infeasible for a interactive learning deployment.

Variance reduction Minimizing an error function directly is costly. The model has to be re-trained using hypothetical labelings in order to estimate the expected reduction in error, which can be computationally expensive. However, in some cases one may still be able to reduce generalization error indirectly by minimizing output variance, and this approach sometimes does have a closed-form solution. Imagine a regression

problem where the learning objective is to minimize standard error (squared loss). The following result from Geman et al. (1992) shows that a learner's expected error can be decomposed into:

$$E[(\hat{y} - y)^2|x] = E_{Y|x}[y - E_{Y|x}[y|x]] + (E_{\mathcal{L}}[\hat{y}] - E_{Y|x}[y|x])^2 + E_{\mathcal{L}}[(\hat{y} - E_{\mathcal{L}}[\hat{y}])^2]\,, \quad (3.39)$$

where $E_{Y|x}[\cdot]$ is an expectation over the conditional density $P(y|x)$, $E_{\mathcal{L}}[\cdot]$ is an expectation over the training set $\mathcal{L}$, and $E[\cdot]$ is an expectation over both. The first term on the right-hand side of this equation represents noise, that is, the unreliability of the true label y given x. The second term is the bias, which reflects the invariant error caused by the fixed model class itself. The third term is the output variance, which is the remaining component of the learner's squared-loss with respect to the target function. Minimizing the variance, then, is guaranteed to minimize the future generalization error of the model. Since the learner itself can do nothing about the noise or bias components one can attempt to reduce error in the squared-loss sense by labeling instances that are expected to most reduce the model's output variance over the unlabeled instances $\mathcal{U}$:

$$X^{\pi}_{VR} = argmin_x \sum_{x' \in \mathcal{U}} Var_{h_{\theta+}}[Y|x']\,, \quad (3.40)$$

Computing this value is computational expensive. Although there are estimations for variance-reduction methods to reduce its expense, these are still computational complex. Estimating output variance requires inverting and multiplying $K \times K$ matrices, where K is the number of parameters in the model h_θ. Assuming standard implementations, $O(K^3)$ time is required for these operations.

Remark Several principles of active learning were discussed for minimizing the expected error or variance of output on instances in unlabeled distributions. Such approaches are intuitively appealing and have been empirically successful in terms of producing more accurate learners with fewer labeled instances than uncertainty or hypothesis-based approaches. They aim to reduce classification error by minimizing require that models have to be re-trained to account for all possible labelings of all possible queries. The methods that aim to reduce output variance in the squared-loss sense have a long history in the statistics literature, can be computed by mathematical approximations, and apply equally well to classification and regression problems. Yet, these approaches have at least two serious drawbacks. The first is that they are not as general as uncertainty sampling or disagreement-based methods, and can only be easily

applied to hypothesis classes of a certain form. How to use them with decision trees, nearest-neighbor methods, and a variety of other standard machine learning and data mining techniques is less clear. The second drawback is their computational complexity due to the inversion and multiplication of the Fisher information matrix. If the number of parameters or output variables is relatively small, then these methods can still be relatively fast and efficient. However, if these properties of the problem are very large then both the space and time complexities of the algorithms involved might be too high to be usable in practice.

3.4.5. Distance similarity measures

In contrast to other common AL methods discussed previously, which are based on model predictions, the below strategy introduced in Völker et al. 2021 uses distance measurements between the labeled set's features and the instances' features, i.e. distances are measured in the input space. Instances that have a large distance from known instances differ more naturally. The result would be increased model performance due to increased data variability. The idea is to assemble data sets comprised of dissimilar data points.

$$X^{\pi}_{dist} = argmax_x \sum_{i=1}^{I} dist(x, x_i) \tag{3.41}$$

$$= argmax_x \sum_{i=1}^{I} dist(x, (f_{i,1}, ..., f_{i,m})) \,, \tag{3.42}$$

where the distance X^{π}_{dist} selects the point x with the minimum accumulated distance between itself and every point in a set, here $\mathcal{L}$. More specifically, $x_{i \in I}$ are all instances of the labeled set $\mathcal{L}$ and $f_{i,1}, ..., f_{i,m}\, i \in I$ are their features. Again the distance between two points can be defined as a function $dist : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$:

$$dist^k(x, x') = (\sum_{i=1}^{n} |f_i - f'_i|^k)^{1/k} \,,$$

where if $k = 1$ the resulting distance measure would be the Manhattan distance and if $k = 2$ the Euclidean distance measure.

Curse of dimensionality Data becomes sparse in high-dimensional spaces. In high-dimensional space, proximity, distance, or nearest neighbor may not even be qualitatively meaningful Aggarwal, Hinneburg, and Keim 2002.

Theorem 3 *If* $lim_{d\to\infty} Var(\frac{||X_d||_k}{E[||X_d||_k]}) = 0$, *then* $\frac{max\{||X_d||_k\} - min\{||X_d||_k\}}{min\{||X_d||_k\}} \xrightarrow{p} 0$, *where d denotes the dimension,* $max\{||X_d||_k\}$ *and* $min\{||X_d||_k\}$ *the maximum and minimum distance of n points to the origin* $(0, ..., 0)$, *respectively.*

As a remark, a sequence of vectors $x_1, ..., x_d$ converges in probability to a constant c, if: $\forall\epsilon > 0\, lim_{d\to\infty} P(dist_d(x_d, c) \leq \epsilon) = 0$. The result of the theorem 3 shows that the difference between the maximum and minimum distances to a given point (which is without loss of generality the origin) does not increase as fast as the nearest distance to any point in high dimensional space. With regard to the commonly used L_k norms, Aggarwal, Hinneburg, and Keim 2002 analyzed how they behave in high-dimensionality and demonstrate that it is sensitive to the value of k. Thus, Manhattan distance metric is consistently better than Euclidean distance metric for high dimensional data.

Combining Policies Previous AL strategies were introduced. The instance with the highest utility was queried to extend the labeled data set. In the early stages of an AL run, the predictive power of ML algorithms is relatively low due to a small amount of training data. This can lead to a scenario where many candidates perform equally well on the utility function, despite having different feature sets. But what if several instances obtain the highest utility, which one should be chosen? One can simply pick an instance at random. Another and more systematic approach is to use a second utility measure on the selection of instances. As a recap, X_A^π was denoted as the best instances that the utility measure A would select. Now, for the sake of combining two utility measures $X_{A,q}^\pi$ denotes the set of instances, which lay over the percentile q using the utility measure A. Combining the distance measure strategy with another strategy gives:

$$X_{dist,A,q}^\pi = argmax_x \sum_{x' \in X_{A,q}^\pi} dist(x, x')\,. \tag{3.43}$$

3.4.6. Stopping criteria

An important aspect of active learning applications generally is the ability to establish when to stop learning, or at least when to stop querying. This can be thought of as the point at which the expense of acquiring new training data exceeds that of the errors made by the current system. To put it differently, when accuracy levels of a learner reach a plateau it is likely a waste of time and resources to acquire more data to improve accuracy. Therefore, an active learner may decide to stop asking questions in order to save time and resources. Stopping criteria for active learning have been proposed

Ishibashi and Hino 2020; J. Zhu et al. 2010. The real criterion for practical applications is often based on economics or other factors outside the control of the learner, which likely come well before an intrinsic threshold determined by the learner.

3.4.7. Benchmark

The performance metric is an essential part of AL. But as performance metrics only describe how a model performed without context to the benefit of using a policy no real insight is gained.

To tackle this issue a simple and common used benchmark gets introduced, which gets employed to evaluate the performance under the usage of policies. When it comes to the selection of new data points for labeling in regards to the performance gain of an model, one has no real insight which data points are the best choice. Intuitively one would chose those, which provide hopefully the most information to the data set used for training, when added. Since those are often not trival to find, here it is assumed one would chose the next data point randomly. Therefore as an benchmark, the performance of randomly selecting the next data point is used. Random sampling, or a random process at each time step is denoted as X^{π}_{random}, where $x \in \mathcal{U}$ gets randomly chosen with probability $P(x) = \frac{1}{I}$, i.e. following an uniform distribution.

The performance of an AL strategy can be set as a distribution empirically determined from multiple AL runs. Random sampling and other selection strategies can be compared by their performances at each AL iteration. Further, how fast a strategy reaches a specific performance relative to random sampling is another measure of success.

4. Experiments

In this chapter, the data collection and experiments are described. The experiments were performed under the utilization of the programming language python. Frequently used packages were NumPy Harris et al. 2020 and pandas team 2020 data handeling. The AL framework modAL Danka and Horvath n.d. was used. For all plots the packages Matplotlib Hunter 2007 and seaborn Waskom 2021 were applied. Pedregosa et al. 2011 provided a great variety of Models and helper modules. Also Ling, Hutchinson, Antono, Paradiso, et al. 2017 provided a RF regression model, with the additional predictive output of .

4.1. Data description

The data which gets described in the following lay on the Fractographic Online Database website provided by Bundesanstalt für Materialforschung und -prüfung (BAM). Using this website, images were downloaded one by one and then supplemented with their corresponding meta-information in an excel spreadsheet. The data at hand stems from the field of material science, specifically fractography. Fractography is a subfield of material science, which is devoted to the description and assessment of fractured surfaces. In detail, the data set consists of images and their assigned meta-information. These images are scanning electron microscope (SEM) images. An SEM is a type of electron microscope that creates images of a sample by scanning its surface with a focused beam of electrons. Through their interactions with the atoms on a sample surface, electrons create signals that provide information about its surface composition and topography. The electron beam is scanned in a specific pattern, and the intensity of the detected signal is combined with the beam's position to create an image. Those SEM images are two-dimensional consisting of a discrete number of pixels with specific intensity values and are available in grayscale format, i.e. each pixel has an intensity between 0 (black) and 255 (white). Assume, that there are $n^2 \in \mathbb{N}$ pixels for each SEM, then $p = (p_1, ..., p_{n,n}) \in Int^{n \times n}$ is the pixel tupel with possible occurring intensities $Int =$

$\{0, 1, ..., 255\}$, resulting when flattening the image.
The meta-information is composed of the material name, the fracture case on hand and the magnification in which the image was taken. From the material name one can figure out its chemical composition, processing steps and material inherent properties.
A fracture occurs as a result of the separation of an object or material into two or more pieces caused by stress. Here, two fracture cases are of interest, brittle fractures and ductile fracture. A brittle fracture occurs before further deformation is visible, while a ductile fracture occurs with visible deformation.
All images belong to material samples. For example in figure 4.1 is a sample shown of the material 18NiCrMo14-6. Each sample follows determined orientation and adjustment rules for the pocess of taking images under the microscope. For example the brittle fracture always appear at the upper part of the image. In 4.1 marked regions (B1-B5) where chosen by an domain expert. In each region different microscopic structures occur. Consider B4 and B5, shown in 4.2 and 4.3, respectively, where in 4.2 a ductile fracture and in 4.3 a brittle fracture at different magnifications are depicted. Images at different magnification of the same region in a material depict different microstructures.

4.2. Data reporting

The micrographs available span a wide range of magnifications, beginning from $1\mu m$ up to $500\mu m$ magnification.
Since images at different magnification of the same region in a material depict different microstructures, therefore each magnification level would result in a distinct data set. Unfortunately, not all magnifications were equally often available on the website. Hence, only for a magnification of $10\mu m$, $20\mu m$, and $100\mu m$ a sufficient amount of images was available.
Corresponding to these magnifications data sets were constructed containing chemical compositions, kinds of fracture, and micrographs, and were of sizes 120, 120 and 156 respectively. All data sets are composed of about one-third brittle fracture and two-thirds ductile failure 4.4 and hence are unbalanced.

In materials science, identifying the process-structure-property relationship is the ultimate goal. To depict this linkage two tasks were established. The classification of fractures, based on their corresponding images was the first task considered.
The second task in consideration was the classification of the material at hand depicted on the image. Since the number of different materials was 25 in total and therefore

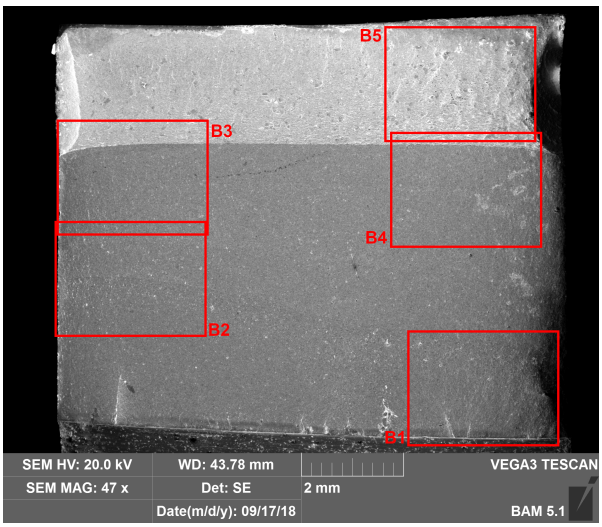
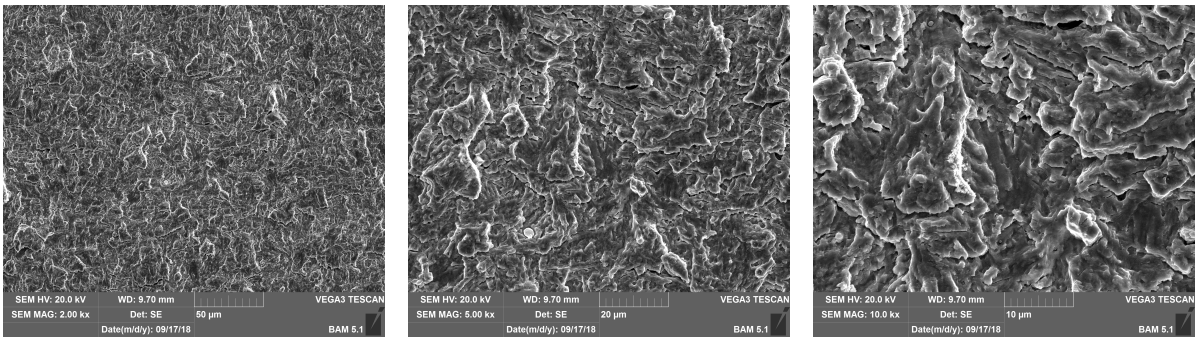
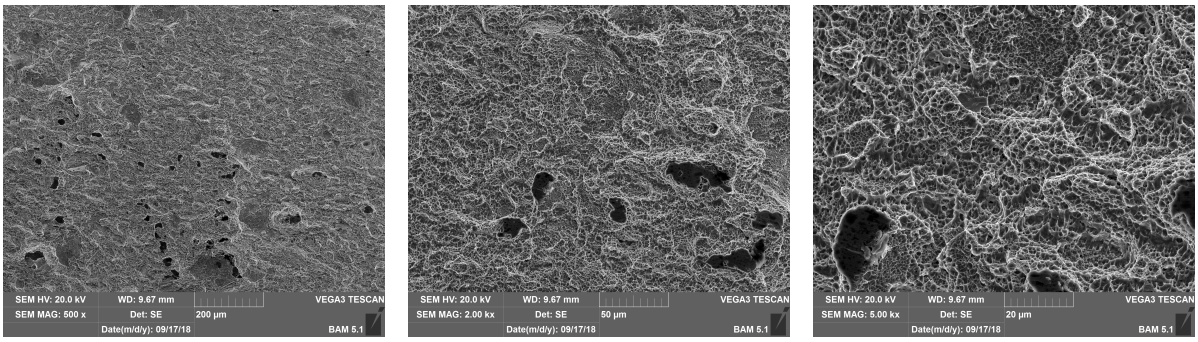


Figure 4.1.: material sample of material 18NiCrMo14-6 with marked regions



(a) ductile failure at 50 μm (b) ductile failure at 20 μm (c) ductile failure at 10 μm

Figure 4.2.: ductile fracture of sample material 18NiCrMo14-6 in region B4 at different magnifications



(a) brittle failure at 200 μm (b) brittle failure at 50 μm (c) brittle failure at 20 μm

Figure 4.3.: brittle fracture of sample material 18NiCrMo14-6 in region B5 at different magnifications

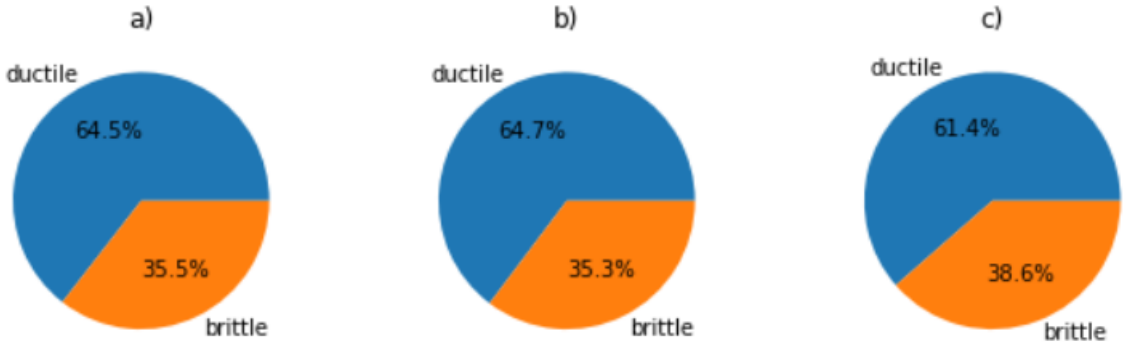


Figure 4.4.: Distribution of the label ductile and brittle in each data set, where a), b) and c) correspond to magnifications $10\mu m$, $20\mu m$ and $100\mu m$, respectively.

too large for deploying classification for this FSL problem, the chemical composition of each material was listed in an excel spreadsheet. The chemical compositions were inferred from the material names. Then the chemical compositions of the materials were used to deploy a k-Means algorithm and cluster the 25 different materials into three clusters. An embedding method known as t-distributed stochastic neighbor embedding (t-SNE) visualizes high-dimensional data by assigning each point a position in a two or three-dimensional map. This nonlinear dimensionality reduction technique models high-dimensional objects with two- or three-dimensional data points in a manner such that similar objects are modeled using nearby points, while dissimilar objects are modeled with distant points. Similar to PCA, it works better since PCA is a linear method, while t-SNE is a non-linear method that minimizes the distance between points in a Gaussian distribution while preserving the local structure of the data. The drawback is that its mapping cannot be replicated. By utilizing t-SNE to map this higher dimensional space, where the materials lay, to a two-dimensional space the k-Means cluster were plotted 4.5. Each coloring indicates a different cluster affiliation. There was also a fourth cluster introduced for the material "Sinterstahl", or sintered steel due to the unique manufacturing process that sets it apart from the other materials. Sintering is the process of compacting and forming a solid mass of material by heat or pressure without melting it to the point of liquefaction. Thus sintered steel can include open pores Rawashdeh, Khraisat, and Borgström 2017.

Besides fracture classification, the goal was to distinguish the newly created material cluster based on their fracture images. The distribution of materials is heavily skewed, shown in 4.6, which had to be considered by choosing a feasible performance metric. The same is true for the fracture classification task.

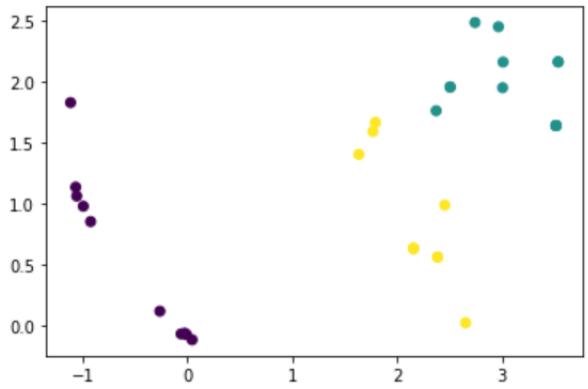


Figure 4.5.: t-SNE plot on material composition clusters resulting from k-Means algorithm

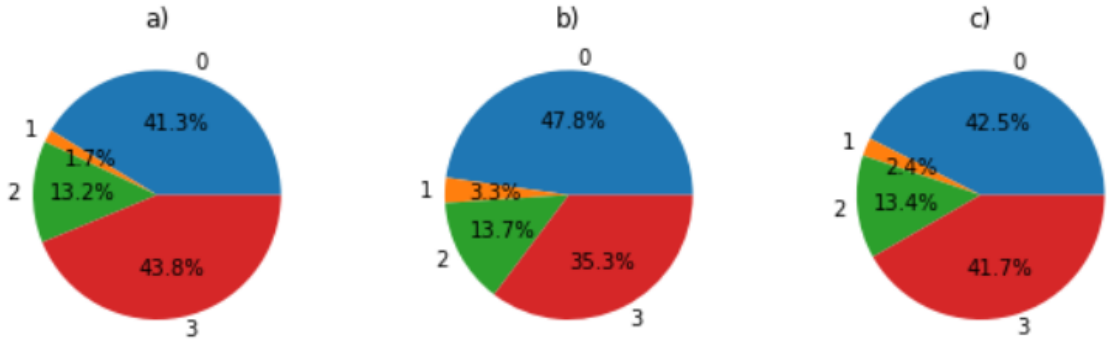


Figure 4.6.: Distribution of material classes 0 to 3 in each data set, where a), b) and c) correspond to magnifications $10\mu m$, $20\mu m$ and $100\mu m$, respectively.

4.3. Data preprocessing

In this section, the process of manipulating reported data before it is used to ensure or enhance performance is described.

4.3.1. Data augmentation

AL is most appropriate when the unlabeled data instances themselves are numerous, can be easily collected or synthesized, and one anticipates having to label many of them to train an accurate system. The data sets described above had only 120 to 156 instances. Comparing different AL strategies only makes sense if those strategies achieve meaningful improvement over the benchmark strategy, the random process. Hence the available data sets were augmented to construct larger ones. Hand-crafted rules mentioned earlier don't alter the textural or statistical properties of the images and are therefore infeasible in this case, since in the further process-feature extraction methods, which describe textural and statistical properties were used. For the deployed third method, this augmentation technique could be used, but due to comparability wasn't. Transformations could easily create samples, which could not possibly be created under a microscope and were thus not considered. No other data sets were regarded.
To obtain a data set with enough data different regions of interest for the images were considered. Twelve 224×224 pixel sub-images were cropped out of each micrograph in the original data sets, expanding the original data sets 12-fold to sizes of 1452, 1836 and 1524 for the magnifications $10\,\mu m$, $20\,\mu m$ and $100\,\mu m$, respectively. The twelve yellow frames in 4.7 indicate exemplarily the image regions used to form larger data sets. During this methodology, the meta-information located at the bottom was automatically cropped. This was important because this area could contain information, which helps an ML model to predict better without utilizing information inherent to the material microstructure. Also, since images were originally available in different sizes, this proceed normalized all data point dimensions for further processing.

4.3.2. Feature generation

In a small data domain, deep learning, that is neuronal networks with many thousand parameters isn't feasible, and traditional ML algorithms are used. Input images have too many extra details that won't be needed for classification. Therefore, feature extraction methods were used. This procedure simplifies and speeds up the process of classifying

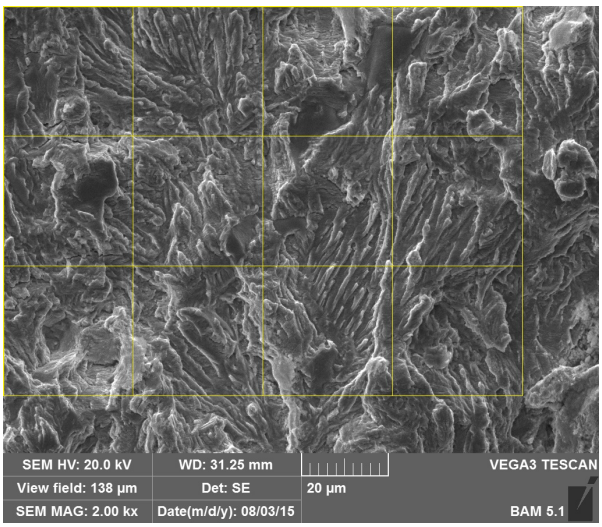


Figure 4.7.: fatigue failure fracture image case 10CrMo9-10, 1.7380 with yellow frames indicating image regions used for feature extraction in each data set

images, which is especially important when dealing with little data. The methods: GLCM and Haralick's features; LBP and transfer learning with VGG16, were utilized and standardization was deployed on the obtained features. The first two methods are commonly used in computer vision, particularly in microstructure classification, including fractographic images. For every image a GLCM was computed on which for four directions 13 Haralick's features were calculated, which in turn translated to the 26 features in total. Those were composed of two times 13 features resulting from the features mean and range over all directions, respectively. The LBP features were created by a $(24, 3)$ neighborhood, meaning 24 neighbors with a distance of 3 to the central pixel. The decimal numbers were calculated, and their distribution was approximated by a histogram with 26 bins, which in turn is the feature set. Extracted features obtained by transfer learning with VGG16 were just recently adapted to fractographic images with good findings. The later convolutional layers are in the network, the more information they contain about higher-level features and more complex textures. Therefore the later layers $C_{3,3}$, $C_{4,3}$ and $C_{5,3}$ were used, since fracture textures are on the complex site. Furthermore, the location of a given feature in an SEM image is not of importance, hence featurization should be translation invariant. The featurization techniques described in 3.2.3 fulfill this criterion, but due to the computational expense of VLAD and the enormous amount of features extracted by Gram, only Mean and Max were feasible. In comparison to Max, Mean evaluates all feature map entries, whereas Max evaluates only one. Concluding information content of Mean over the instances should be greater.

Only the Mean technique was considered. The resulting number of features extracted for each image by the methods GLCM and Haralick's features, LBP, transfer learning with VGG16 layers $C_{3,3}$, $C_{4,3}$ and $C_{5,3}$ as output layer and applying the Mean method on their feature maps with quantities of 256, 512 and 512, respectively. Dimensionality reduction techniques, like principal component analysis, weren't considered, since they may lead to data loss and would also blast the scope of this work.

4.4. Model

When choosing a learner, one should be aware of the number of features since not every model handles high dimensional data equally due to the curse of dimensionality, specified in theorem 3. The models of choice were an ensemble of decision trees and an MLP classifier, which in contrast to support vector machines are suited for multi-class problems. Also, if the input dimension is really large, it seems that all kernel methods for Gaussian processes will fail, and therefore Gaussian processes were not considered.

4.4.1. Random Forest classifier

A single decision tree learner can create over-complex trees that do not generalize the data well. By combining several trees this issue can be addressed. Parameters of the random forest model are the gain measure, number of trees in the ensemble, and max depth of the trees, max features each tree is provided with. For the gain measure, the Gini index was chosen as the gain measure, due to its computational advantage over information gain, where the logarithm has to be computed. To mention is, that this measure can be used to output important features, which is an advantage of an RF. More trees will result in better model performance. More trees, however, also require more computational resources, and the improvement diminishes with an increasing number of trees. Using 29 data sets Oshiro, Perez, and Baranauskas 2012 found no significant improvement after 128 trees, based on their test, suggesting a number of trees between 64 and 128. A single decision tree does need pruning to overcome the over-fitting issue. With regards to tree depth, standard random forest algorithms grow the full tree without pruning, since the issue of over-fitting is eliminated by randomly selecting the variables and averaging the predictions. Therefore the number of trees was restricted to 100, the max depth wasn't limited, thus the default settings from Pedregosa et al. 2011 were used. With these parameters at hand, it is possible to obtain a good balance

between performance, processing time, and memory usage.

4.4.2. Multilayer Perceptron classifier

In contrast, an MLP classifier is an ANN with several hidden, fully connected layers. In the following a simple MLP classifier with its default implementation Pedregosa et al. 2011 was used, which has one input layer, one output layer, and one hidden layer with 100 neurons. The maximum number of iterations was also untouched. This number states how many learning algorithm iterations the network at maximum could have, independent of the learning algorithms convergence. This restricts the computation time of the model, but can also lead to worse performance. Nevertheless a good trade off was opted. For the learning algorithm Adam was chosen over L-BFGS and gradient decent, due to its fast convergence over gradient decent and overall better training loss decrease for shallow NN over the other two Karlsson and Bonde 2020.

4.5. Performance metrics

The performance metrics of choice were the $F1$ score and accuracy, due to their common utilization in the field of micrograph classification. As the labels are highly skewed and equal importance wanted to be assigned to each label class, the $F1$ score was generalized to the macro score, which treats each class equally, regardless of size. In addition to the macro $F1$ score $F1_m$, there was also the weighted $F1$ score $F1_{/beta}$ used, where each label got importance based on its quantity. For the binary classification case the binary $F1$ score was used. A balanced accuracy of ACC_b was used to compensate for the skewness in label distribution.

4.6. Investigated scenarios

4.6.1. General performance

These performances were calculated for each data set on each extracted feature set and for both classification scenarios, namely fracture classification and material cluster classification. This was done under the usage of 6-fold cross-validation with an random seed, which as a consequence let the data always split in exactly the same way. 5/6 and 1/6 of the data sets were used for training and validating the learner, respectively,

repeated 5 times. Also it was considered that every set of 12 cropped images should be either in the validation or training set. Otherwise, if these 12 cropped images were split between those two sets, over-fitting could occur. This is the case, because images used for training a ML model would be very similar to the ones used for validation, since they were extracted from the same sample. Therefore, these extracted features are comparable regarding their performances on the two ML models, RF and MLP.

4.6.2. Active Learning performance

AL strategy implementations were taken from the Danka and Horvath n.d., as well as by own implemented strategies such as uncertainty sampling, distance sampling and random sampling, where random sampling was the benchmark strategy. Unfortunately, it is impossible to know which active learning algorithm will work best for a given problem a priori. There is no consistent winner in terms of query selection heuristics, which suggests that different learning algorithms and applications may result in different optimal strategies. Therefore the above mentioned strategies were plotted beside each other for the two scenarios fracture- and cluster classification, and for both the RF classifier and the MLP classifier. More in detail, the AL process was conducted on the strategies: random sampling X^{π}_{random}, least confident X^{π}_{LC}, entropy X^{π}_{H}, distance measure with the Manhattan distance X^{π}_{dist}, distance measure of all instances over the 95 percentile of the least confident method $X^{\pi}_{dist,LC,95}$,distance measure of all instances over the 95 percentile of the entropy method $X^{\pi}_{dist,H,95}$, vote entropy X^{π}_{VE}, soft vote entropy X^{π}_{SVE} and Kullback-Leibler divergence X^{π}_{KL}.
Also the maximum uncertainty strategy X^{π}_{MU} got applied and the strategy combining the maximum uncertainty strategy and the least confident strategy, the maximum likelihood of improvement strategy X^{π}_{MLI} also got applied. Since the model used for calculating the uncertainty is a RF regression model, X^{π}_{MU}, X^{π}_{MLI} were only used for the binary classification task for RF. For binary classification all uncertainty sampling methods should select the same instances and hence achieve the same result. Consequenty, only the performance of one uncertainty sampling technique, namely entropy was plotted.
Same 6-fold cross-validation as before, when the performances on the whole data sets were computed was chosen. This enabled the comparability between all query strategies and the ML model performance on the whole data set. Although the fractographic data sets were fully labeled, by artificially concealing a portion of labels weakly labeled data sets were created. The initial labeled data set size was arbitrarily set to 8 and got a

fixed random seed assigned to it. Therefore each AL strategy got the same random initial set of instances. A total of 120 AL iterations was set as the stopping criteria. Since the total data set size is 1524, composed of 254 instances for model validation and 1270 for the AL process, after reaching the end of AL about 10% of the available data was used for training a model. Further, in addition to 6-fold cross-validation, the initial data set sampling and the AL procedure of querying 120 times was repeated 5 times and averaged. For each repetition an other initial labeled set for each cross-validation was chose. Averaging those 30 results gave the final results. By this approach the volatility of the performances got minimized, because the information context of the initial labeled sets can vary a lot and therefore the initial generalizability and applicability to selection strategies of the learner.
In addition the minimization of the expected classification error X_{ER}^{π} got applied for the MLP. Due to its computational complexity, the minimization of expected classification error strategy constitutes an exception, by only calculating 30 AL iteration performances for that method and only executing one repetition. This may result in lack of comparability. The data set of choice was the cropped $100\mu m$ data set with $VGG16(C_{3,3})$ generated features, since as discussed later on those features the MLP classifier achieved its best performance and the RF classifier one of its best performances.
Several scenarios were regarded. For once it was distinguished between the two models and secondly between the AL objectives. It is important to mention, that the performances were plotted regarding either fracture classification or cluster classification and are not to be confused with the learning objective of the AL algorithm, i.e. the task over which an instance was queried, which in turn is also either fracture classification or cluster classification.
For simplicity, and similarity of the performances, only balanced accuracy was listed in the results section. The corresponding weighted $F1$ scores can be found in the appendix.
In summary, the investigated situations include the AL selection strategies, which are used in one data set ($100\mu m$), with one feature set ($VGG16(C_{3,3})$, initialized with one initial training set size (8), conducted over 120 AL iterations, under two different AL objectives (fracture classification; material cluster classification), applied to two different ML models (RF classifier; MLP classifier), to be measured by two different performance metrics (balanced accuracy; weighted $F1$ score) on two different classification objectives (fracture; material cluster), respectively, which were statistically analyzed in terms of average performance over 30 repetitions (6-fold cross-validation over 5 repetitions). The table 4.1 gives a detailed overview of the conducted experiments, where the columns

correspond to different scenarios and the rows indicate how the scenarios are parameterized. If a parameter spans over several columns, then it pertains to each scenario. A checkmark indicates that a strategy was applied to a specific scenario and a cross the opposite.

AL learner	RF				MLP			
AL objective (classification)	fracture		cluster		fracture		cluster	
classification task	fracture	cluster	cluster	fracture	fracture	cluster	cluster	fracture
performance metric	ACC_b, $F1$	ACC_b, $F1_\beta$	ACC_b, $F1_\beta$	ACC_b, $F1$	ACC_b, $F1$	ACC_b, $F1_\beta$	ACC_b, $F1_\beta$	ACC_b, $F1$
corresponding figure	5.1 A.1	5.2 A.2	5.3 A.3	5.4 A.4	5.5 B.1	5.6 B.2	5.7 B.3	5.8 B.4
random X^π_{random}	✓	✓	✓	✓	✓	✓	✓	✓
distance measure X^π_{dist}	✓	✓	✓	✓	✓	✓	✓	✓
least confident X^π_{LC}	✗	✓	✓	✗	✗	✓	✓	✗
entropy X^π_H	✓	✓	✓	✓	✓	✓	✓	✓
$X^\pi_{dist,LC,95}$	✗	✓	✓	✗	✗	✓	✓	✗
$X^\pi_{dist,H,95}$	✓	✓	✓	✓	✓	✓	✓	✓
vote entropy X^π_{VE}	✓	✓	✓	✓	✓	✓	✓	✓
soft vote entropy X^π_{SVE}	✓	✓	✓	✓	✓	✓	✓	✓
Kullback-Leibler divergence X^π_{KL}	✓	✓	✓	✓	✓	✓	✓	✓
maximum uncertainty X^π_{MU}	✓	✗	✗	✓	✗	✗	✗	✗
maximum likelihood of improvement X^π_{MLI}	✓	✗	✗	✓	✗	✗	✗	✗
initial data set size	8							
number of AL iterations	120							
k-vold cross-validation	$k = 6$							
number of k-vold CV repetitions	5							

Table 4.1.: AL experimental setup

5. Results

This chapter is devoted to the illustration of the results attained by the done experiments. First, the general performances of the applied ML models on all three data sets and for both, fracture- and material cluster classification are shown. After that, for the $100\mu m$ data, learning curves of AL query strategies are plotted.

5.1. General performance

In all four following tables 5.1, 5.2, 5.3 and 5.4 the performances are shown in regards to the extracted features on the individual data sets under the task of fracture classification and material cluster classification, respectively. As a recapitulation, these performance metric scores are the computed average of the performance metric scores calculated on the validation sets.
5.1 and 5.2 refer to the performances of the random forest classifier and 5.3 and 5.4 to the performance of the MLP classifier. In all tables, rows correspond to the various feature sets calculated from the individual column-corresponding data sets. Entries of the tables show the performances attained by the specific model on the data set- feature set combination. The highest performances in each column are highlighted.
5.1 illustrates the balanced accuracies and binary $F1$ scores achieved by the RF for the fracture classification task. It shows, that the highest accuracy is accompanied by the highest $F1$ score, whereas those are solely achieved at the VGG16 ($C_{4,3}$) Mean features. In contrast, the balanced accuracies and $F1$ scores achieved by the RF for the material cluster classification task illustrated in 5.2 weren't exclusively the highest for the VGG16 ($C_{4,3}$) Mean features. With the VGG16 ($C_{5,3}$) Mean features on the $20\mu m$ data better performances were attained.
5.3 illustrates the balanced accuracies and binary $F1$ scores achieved by the MLP for the fracture classification task. Trained and validated on the VGG16 ($C_{3,3}$) Mean features MLP performed best on the $100\mu m$ data, whereas the VGG16 ($C_{4,3}$) Mean features yielded the best results for the $10\mu m$ and $20\mu m$ data. In 5.4 is shown that one feature

method doesn't result in simultaneous the highest score for each performance metric. For instance, for the $10\mu m$ data the balanced accuracy and weighted $F1$ score were highest for the VGG16 ($C_{5,3}$) Mean features, but the VGG16 ($C_{4,3}$) Mean features gave a slightly better macro $F1$ score.
In neither case were GLCM & Haralick's features or LBP features successful in outscoring the other. Moreover, the MLP has better scores than the RF for nearly every feature set on each data set.

feature extraction method	$10\,\mu m$ image data set	$20\,\mu m$ image data set	$100\,\mu m$ image data set
GLCM & Haralick	$ACC_b = 0.61$ $F1 = 0.46$	$ACC_b = 0.6$ $F1 = 0.44$	$ACC_b = 0.61$ $F1 = 0.5$
LBP	$ACC_b = 0.73$ $F1 = 0.63$	$ACC_b = 0.74$ $F1 = 0.63$	$ACC_b = 0.67$ $F1 = 0.57$
VGG16 ($C_{3,3}$)	$ACC_b = 0.76$ $F1 = 0.67$	$ACC_b = 0.78$ $F1 = 0.7$	$ACC_b = 0.77$ $F1 = 0.7$
VGG16 ($C_{4,3}$)	$\mathbf{ACC_b = 0.8}$ $\mathbf{F1 = 0.72}$	$\mathbf{ACC_b = 0.82}$ $\mathbf{F1 = 0.76}$	$\mathbf{ACC_b = 0.8}$ $\mathbf{F1 = 0.74}$
VGG16 ($C_{5,3}$)	$ACC_b = 0.77$ $F1 = 0.68$	$ACC_b = 0.8$ $F1 = 0.72$	$ACC_b = 0.76$ $F1 = 0.69$

Table 5.1.: **RF** performances in predicting **fracture classes** on different feature sets and magnifications data sets.

feature extraction method	$10\,\mu m$ image data set	$20\,\mu m$ image data set	$100\,\mu m$ image data set
GLCM & Haralick	$ACC_b = 0.61$ $F1_m = 0.56$ $F1_\beta = 0.7$	$ACC_b = 0.57$ $F1_m = 0.54$ $F1_\beta = 0.74$	$ACC_b = 0.63$ $F1_m = 0.55$ $F1_\beta = 0.73$
LBP	$ACC_b = 0.54$ $F1_m = 0.44$ $F1_\beta = 0.65$	$ACC_b = 0.58$ $F1_m = 0.54$ $F1_\beta = 0.72$	$ACC_b = 0.66$ $F1_m = 0.59$ $F1_\beta = 0.79$
VGG16 ($C_{3,3}$)	$ACC_b = 0.61$ $F1_m = 0.55$ $F1_\beta = 0.71$	$ACC_b = 0.6$ $F1_m = 0.57$ $F1_\beta = 0.8$	$\mathbf{ACC_b = 0.71}$ $F1_m = 0.66$ $F1_\beta = 0.85$
VGG16 ($C_{4,3}$)	$ACC_b = 0.64$ $F1_m = 0.57$ $F1_\beta = 0.73$	$\mathbf{ACC_b = 0.61}$ $\mathbf{F1_m = 0.62}$ $\mathbf{F1_\beta = 0.81}$	$\mathbf{ACC_b = 0.71}$ $\mathbf{F1_m = 0.67}$ $\mathbf{F1_\beta = 0.86}$
VGG16 ($C_{5,3}$)	$\mathbf{ACC_b = 0.66}$ $\mathbf{F1_m = 0.62}$ $\mathbf{F1_\beta = 0.76}$	$ACC_b = 0.6$ $F1_m = 0.6$ $F1_\beta = 0.79$	$ACC_b = 0.68$ $\mathbf{F1_m = 0.67}$ $F1_\beta = 0.85$

Table 5.2.: **RF** performances in predicting **material cluster classes** on different feature sets and magnification data sets.

feature extraction method	$10\,\mu m$ image data set	$20\,\mu m$ image data set	$100\,\mu m$ image data set
GLCM & Haralick	$ACC_b = 0.62$ $F1 = 0.5$	$ACC_b = 0.63$ $F1 = 0.51$	$ACC_b = 0.68$ $F1 = 0.59$
LBP	$ACC_b = 0.7$ $F1 = 0.61$	$ACC_b = 0.74$ $F1 = 0.63$	$ACC_b = 0.72$ $F1 = 0.65$
VGG16 ($C_{3,3}$)	$ACC_b = 0.8$ $F1 = 0.72$	$ACC_b = 0.81$ $F1 = 0.75$	$\mathbf{ACC_b = 0.85}$ $\mathbf{F1 = 0.80}$
VGG16 ($C_{4,3}$)	$\mathbf{ACC_b = 0.81}$ $\mathbf{F1 = 0.74}$	$\mathbf{ACC_b = 0.83}$ $\mathbf{F1 = 0.77}$	$ACC_b = 0.83$ $F1 = 0.79$
VGG16 ($C_{5,3}$)	$ACC_b = 0.79$ $\mathbf{F1 = 0.74}$	$\mathbf{ACC_b = 0.83}$ $F1 = 0.76$	$ACC_b = 0.8$ $F1 = 0.75$

Table 5.3.: **MLP** performances in predicting **fracture classes** on different feature sets and magnification data sets.

feature extraction method	$10\,\mu m$ image data set	$20\,\mu m$ image data set	$100\,\mu m$ image data set
GLCM & Haralick	$ACC_b = 0.62$ $F1_m = 0.53$ $F1_\beta = 0.65$	$ACC_b = 0.60$ $F1_m = 0.56$ $F1_\beta = 0.77$	$ACC_b = 0.72$ $F1_m = 0.62$ $F1_\beta = 0.77$
LBP	$ACC_b = 0.59$ $F1_m = 0.48$ $F1_\beta = 0.68$	$ACC_b = 0.64$ $F1_m = 0.59$ $F1_\beta = 0.75$	$ACC_b = 0.72$ $F1_m = 0.66$ $F1_\beta = 0.79$
VGG16 ($C_{3,3}$)	$ACC_b = 0.65$ $F1_m = 0.55$ $F1_\beta = 0.74$	$ACC_b = 0.64$ $F1_m = 0.61$ $F1_\beta = 0.82$	$\mathbf{ACC_b = 0.81}$ $\mathbf{F1_m = 0.71}$ $F1_\beta = 0.87$
VGG16 ($C_{4,3}$)	$ACC_b = 0.65$ $\mathbf{F1_m = 0.59}$ $F1_\beta = 0.75$	$\mathbf{ACC_b = 0.66}$ $\mathbf{F1_m = 0.62}$ $\mathbf{F1_\beta = 0.83}$	$ACC_b = 0.77$ $\mathbf{F1_m = 0.71}$ $\mathbf{F1_\beta = 0.88}$
VGG16 ($C_{5,3}$)	$\mathbf{ACC_b = 0.67}$ $F1_m = 0.58$ $\mathbf{F1_\beta = 0.76}$	$ACC_b = 0.63$ $F1_m = 0.59$ $F1_\beta = 0.8$	$ACC_b = 0.72$ $F1_m = 0.64$ $F1_\beta = 0.84$

Table 5.4.: **MLP** performances in predicting **material cluster classes** on different feature sets and magnification data sets.

5.2. AL performance

In this section different learning curves of selection strategies regarding the objectives of fracture classification and material cluster classification are plotted. The overall general performance, under the usage of 100% of the available data was plotted as a horizontal black line. In contrast, the AL strategies only accumulate about 10% of available data. The abscissa for each plot refers to the number of AL iterations already performed, and the ordinate represents the score of the performance metric achieved by the model's prediction on the validation set. For all plots shown in this section, the performance metric is the balanced accuracy. The random forest strategy is in every plot depicted as a blue line. To keep the following as simple as possible, each line will be referred to using its corresponding strategy rather than its color. 5.1, 5.2, 5.4 and 5.3 show RF performances, where the RF was trained on data acquired by AL strategies with RF learner, whereas 5.5, 5.6, 5.8 and 5.7 show MLP performances, where the MLP was trained on data acquired by AL strategies with MLP learner. The same plots, but with $F1$ scores as performance metrics are in the appendix chapter.
Further, each plot is accompanied by a table, where the number of learning iterations needed to achieve a certain performance threshold for the first time, for each strategy is illustrated. / marks when a certain threshold wasn't reached. These thresholds are 70%, 75%, 80%, 85%, 90%, 95% and 100% of the general performance, and correspond to each column, respectively. The rows of the tables correspond to the different strategies. These tables should help to quantify the strategies better. All entries in each column, which contain a smaller number than the entry corresponding to random are highlighted. Those reflect the acquisition of less data, while simultaneously resulting in the same model performance.

5.2.1. Random Forest learner

Here, several results for Al selection techniques using a RF learner are presented.

Fracture classification The plots 5.1 and 5.2 show RF performances after the acquisition of different training set sizes in regards to the AL objective of fracture classification for fracture classification and material cluster classification, respectively. Figure 5.1 depicts RF fracture classification performance for different selection strategies after the acquisition of different training set sizes based on the AL objective of fracture classification. It shows maximum uncertainty and maximum likelihood of improvement are constantly over random, whereas maximum uncertainty reaches even higher and even nearly reaching the general performance. After about 37 iterations the combination of distance and entropy outperforms random and is constantly as good as maximum likelihood of improvement. In contrast, the other strategies, with the exception of vote entropy and Kullback-Leibner divergence, for the iterations 19 to 34 didn't perform better than random, or even worse. For instance, distance constantly underperformed random.

Figure 5.2 depicts RF cluster classification performance for different selection strategies after the acquisition of different training set sizes based on the AL objective of fracture classification. Entropy and the combination of distance and entropy performed after about 6 better than random, where entropy outperformed the latter. Vote entropy and Kullback-Leibner divergence even lost performance during the first 38 iterations, then they rapidly improved and even outperformed random after 57 and 100 iterations, respectively. Again, distance underperformed, and soft vote entropy too. Maximum uncertainty was only slightly better than random after 45 iterations and maximum likelihood of improvement equaled randoms performance after 60 iterations.
It shows, for the learning objective of fracture classification maximum uncertainty performed the best on fracture classification, and also outperformed random after 45 iterations on cluster classification. The combination of a distance measure and entropy was nearly as effective for the feature classification task as maximum uncertainty, while outperforming it slightly on the cluster classification task. They reached over 73% balanced accuracy for feature classification, which is over 95% of the performance reached on the whole data set. Combining maximum uncertainty with the least confident strategy didn't improve the RF performance.

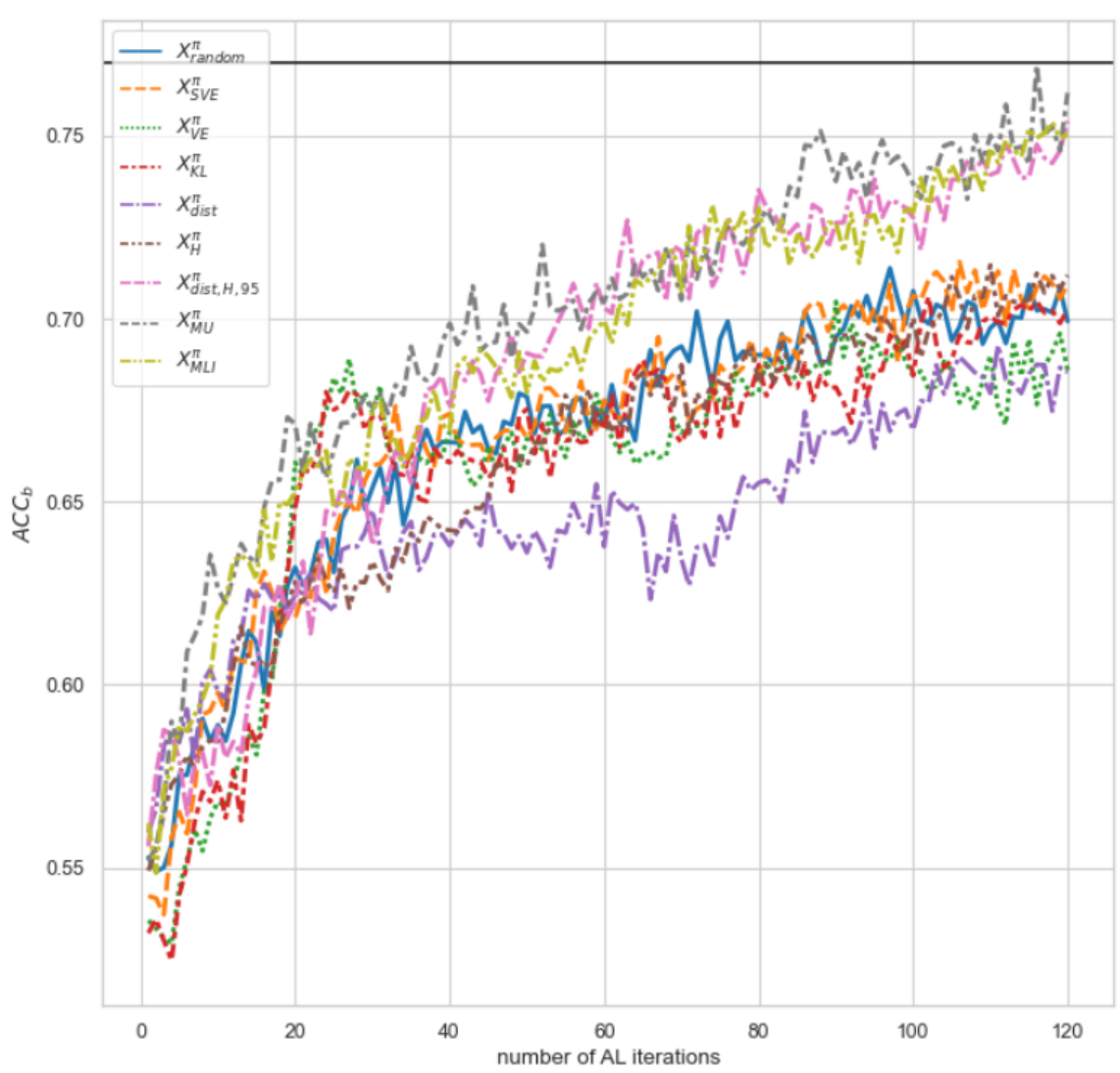


Figure 5.1.: **Fracture classification** learning curves for AL strategies under the **learning objective of fracture classification**.

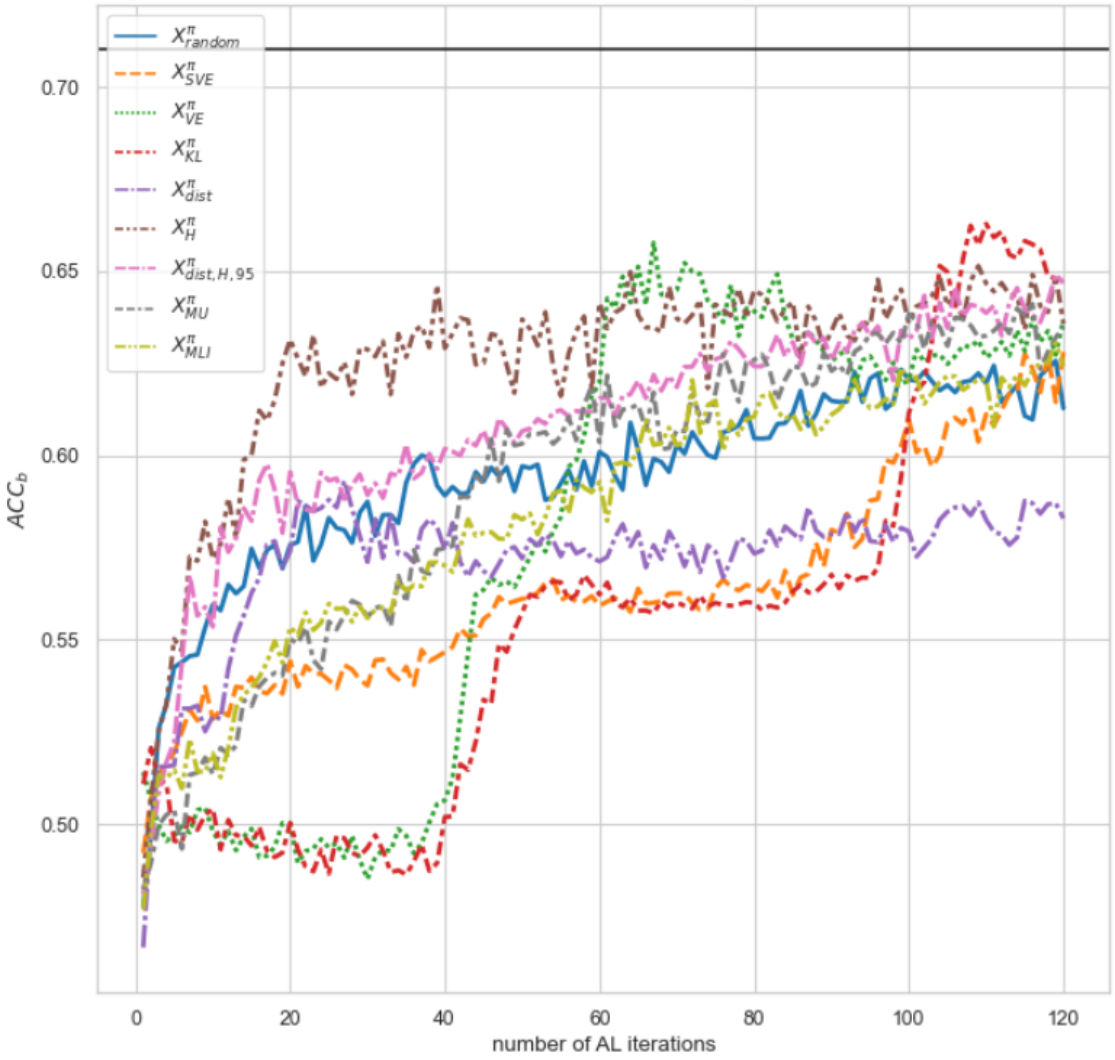


Figure 5.2.: **Cluster classification** learning curves for AL strategies under the **learning objective of fracture classification**.

Performance thresholds	0.5390	0.5775	0.6160	0.6545	0.6930	0.7315	0.7700
X^{π}_{random}	1	7	17	28	72	/	/
X^{π}_{SVE}	1	8	**15**	29	**67**	/	/
X^{π}_{VE}	5	13	18	**20**	90	/	/
X^{π}_{KL}	5	14	18	21	97	/	/
X^{π}_{dist}	1	**3**	**14**	59	/	/	/
X^{π}_{H}	1	**6**	18	46	83	/	/
$X^{\pi}_{dist,H,95}$	1	**3**	**16**	28	**48**	**80**	/
X^{π}_{MU}	1	**4**	**8**	**17**	**39**	**84**	/
X^{π}_{MLI}	1	**5**	**10**	**21**	**58**	**101**	/

Table 5.5.: Learning iterations for AL strategies needed to exceed a certain threshold corresponding to 75% - 100% of the **fracture classification** performance achieved on the whole data set.

Performance thresholds	0.4970	0.5325	0.5680	0.6035	0.6390	0.6745	0.7100
X^{π}_{random}	3	5	15	64	/	/	/
X^{π}_{SVE}	**2**	9	84	100	/	/	/
X^{π}_{VE}	**1**	43	50	**58**	**61**	/	/
X^{π}_{KL}	**1**	45	96	100	**104**	/	/
X^{π}_{dist}	**2**	12	17	/	/	/	/
X^{π}_{H}	**2**	**4**	**7**	**16**	**39**	/	/
$X^{\pi}_{dist,H,95}$	3	6	**11**	**43**	**102**	/	/
X^{π}_{MLI}	**2**	14	38	66	/	/	/
X^{π}_{MU}	3	14	36	46	**114**	/	/

Table 5.6.: Learning iterations for AL strategies needed to exceed a certain threshold corresponding to 75% - 100% of the **cluster classification** performance achieved on the whole data set.

Cluster classification The plots 5.3 and 5.4 show RF performances after the acquisition of different training set sizes in regards to the AL objective of material cluster classification for material cluster classification and fracture classification, respectively.
Figure 5.3 depicts RF cluster classification performance for different selection strategies after the acquisition of different training set sizes based on the AL objective of cluster classification. Entropy, least confident and soft vote entropy outscored random after 6 iterations, the combination of distance and least confident outscored random after 10 iterations and the combination of distance and entropy after 20 iterations, it scored for the rest 100 iterations worse than the other 4 strategies. Also the combination of distance and least confident was slightly worse than the remaining 3 strategies. Least confident outperformed all other strategies between the 30th and 70th iterations. The last 50 iterations least confident, entropy, and soft vote entropy performed about equally well, and even reached partly the general performance. The distance underperformed random, and vote entropy and Kullback-Leibner divergence performed equally, with rapid performance boost after 40 and 95 iterations, leading to a better score than random after 100 iterations. No a single strategy was better than random for both classification tasks.

Figure 5.4 depicts RF fracture classification performance for different selection strategies after the acquisition of different training set sizes based on the AL objective of cluster classification.
It shows that for this case no strategy outscored random, except for iterations 18 to 30, where vote entropy and Kullback-Leibner divergence had a spike in performance, which then even decreased. Also least confident was better for the iterations 35 to 65. It is to mention that for the first 45 iterations soft vote entropy did the worst.
For cluster classification as the learning objective soft vote entropy, least confident and entropy did great for cluster classification, but for fracture classification no real improvement can be seen. Under the utilization of soft vote entropy and least confident strategy the RF even reached the same performance as it would on the whole data set, while only acquiring 10% of the data. Augmenting least confident and entropy with the distance measure approach resulted in no improvement.

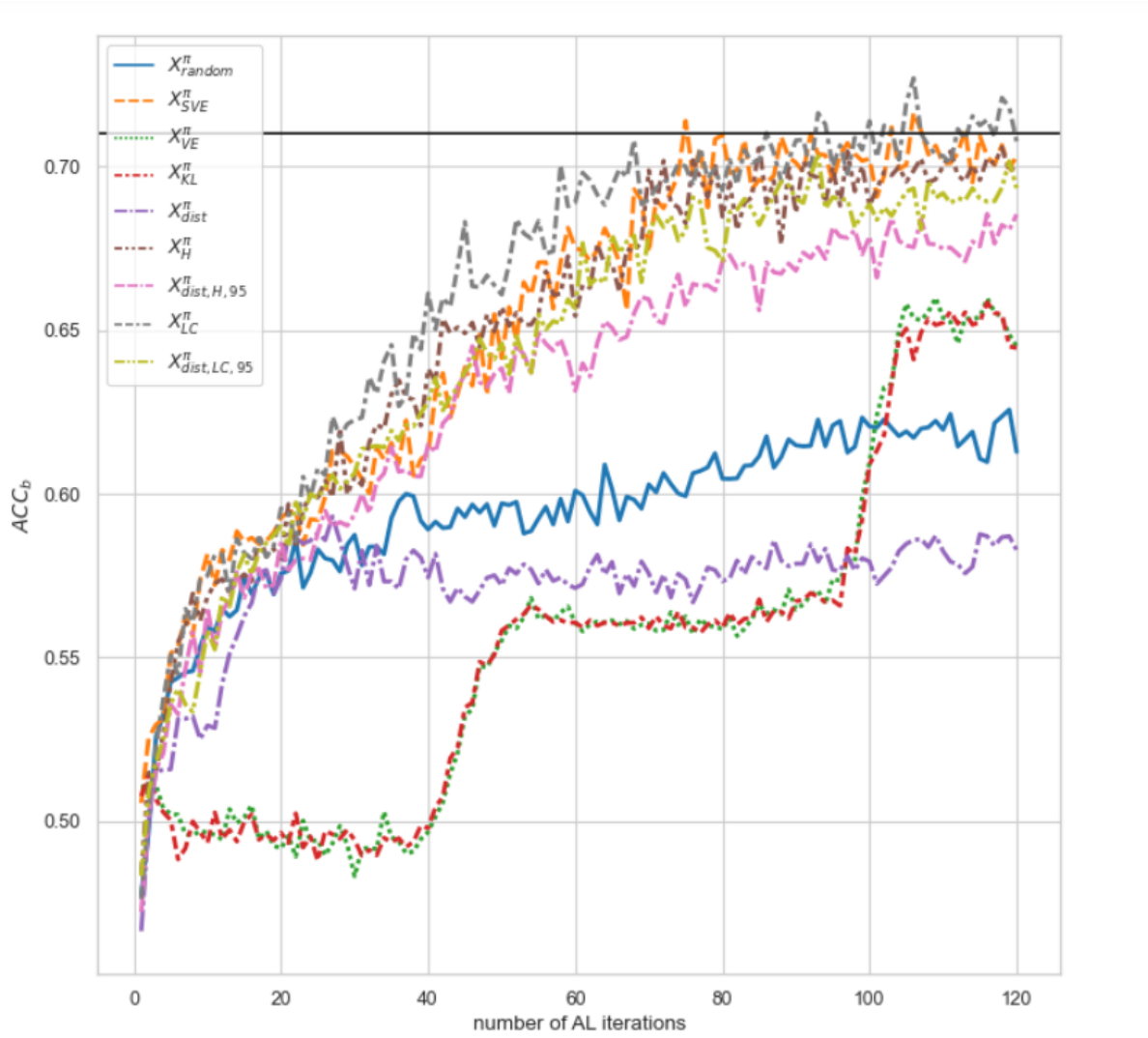


Figure 5.3.: **Cluster classification** learning curves for AL strategies under the **learning objective of cluster classification**.

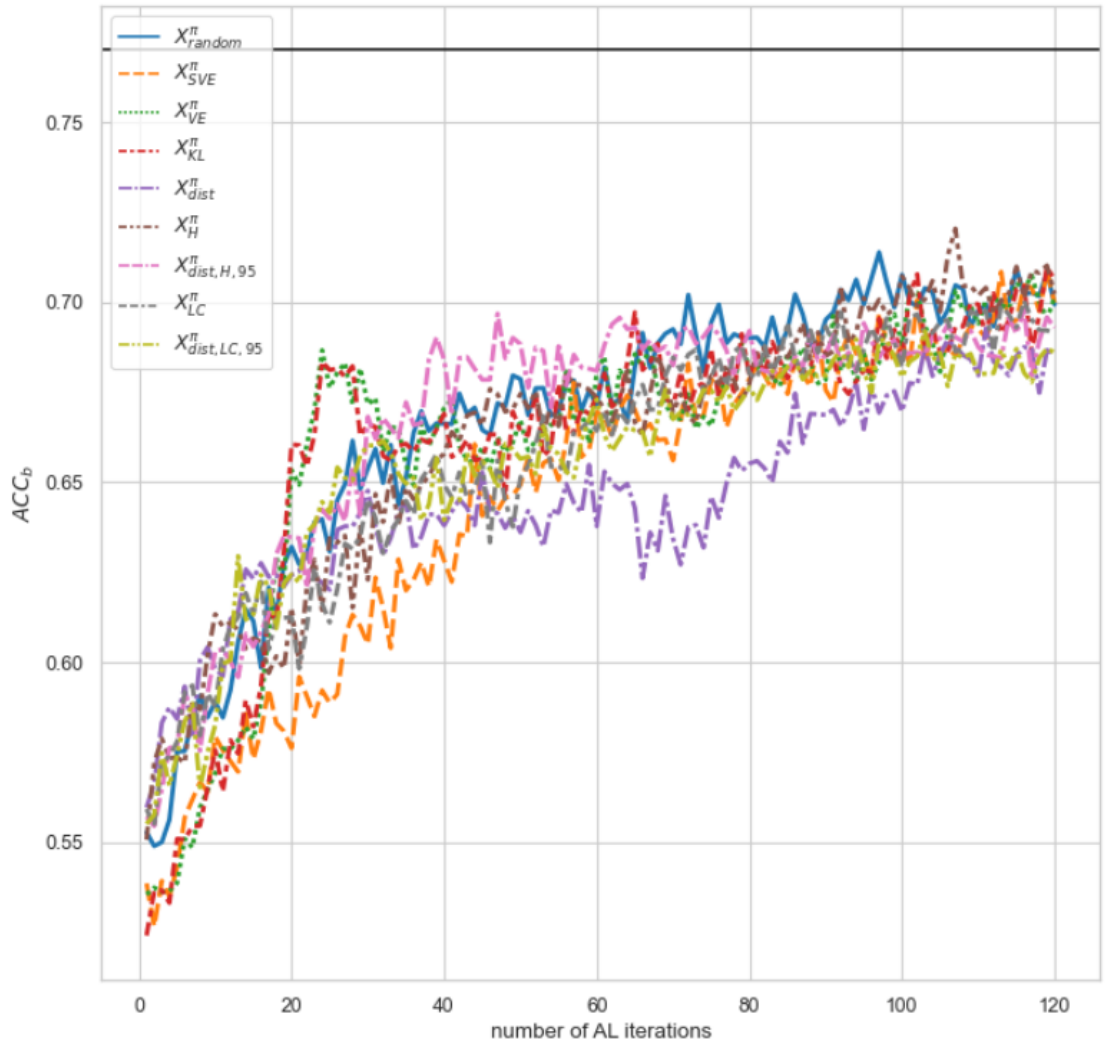


Figure 5.4.: **Fracture classification** learning curves for AL strategies under the **learning objective of cluster classification**.

Performance thresholds	0.4970	0.5325	0.5680	0.6035	0.6390	0.6745	0.7100
X^{π}_{random}	3	5	15	64	/	/	/
X^{π}_{SVE}	**1**	5	**9**	**27**	**47**	**59**	**75**
X^{π}_{VE}	**1**	46	54	100	**104**	/	/
X^{π}_{KL}	**1**	45	92	100	**104**	/	/
X^{π}_{dist}	**2**	12	17	/	/	/	/
X^{π}_{H}	**2**	5	**8**	**26**	**42**	**63**	/
$X^{\pi}_{dist,H,95}$	**2**	5	**14**	**33**	**46**	**93**	/
X^{π}_{LC}	**2**	**4**	**9**	**23**	**34**	**45**	**86**
$X^{\pi}_{dist,LC,95}$	**2**	5	**12**	**27**	**46**	**61**	/

Table 5.7.: Learning iterations for AL strategies needed to exceed a certain threshold corresponding to 75% - 100% of the **cluster classification** performance achieved on the whole data set.

Performance thresholds	0.5390	0.5775	0.6160	0.6545	0.6930	0.7315	0.7700
X^{π}_{random}	1	7	17	28	72	/	/
X^{π}_{SVE}	3	10	31	44	97	/	/
X^{π}_{VE}	6	13	18	**22**	91	/	/
X^{π}_{KL}	5	12	19	**20**	**65**	/	/
X^{π}_{dist}	1	**3**	**14**	59	/	/	/
X^{π}_{H}	1	**3**	22	39	87	/	/
$X^{\pi}_{dist,H,95}$	1	**6**	18	30	**47**	/	/
X^{π}_{LC}	1	**5**	**14**	39	85	/	/
$X^{\pi}_{dist,LC,95}$	1	**6**	**13**	29	/	/	/

Table 5.8.: Learning iterations for AL strategies needed to exceed a certain threshold corresponding to 75% - 100% of the **cluster classification** performance achieved on the whole data set.

5.2.2. Multilayer Perceptron learner

Fracture classification as AL objective The plots 5.5 and 5.6 show MLP performances after the acquisition of different training set sizes in regards to the AL objective of fracture classification for fracture classification and material cluster classification, respectively.

Figure 5.5 depicts MLP fracture classification performance for different selection strategies after the acquisition of different training set sizes based on the AL objective of fracture classification. It shows, that entropy and combination of distance and entropy outscore random after 17 iterations. Entropy was through the entire learning process slightly better than the combination of distance and entropy, while both nearly converge towards the general performance. Kullback-Leibner divergence surpass random's score after 44 iterations, whereas underperformed the latter two strategies. All the other strategies scored worse than random. Figure 5.6 shows MLP cluster classification performance for different selection strategies after the acquisition of different training set sizes based on the AL objective of fracture classification. Entropy and combination of distance and entropy performed about equally better than random, whereas the latter outperformed for the first 26 iterations and then the former till iteration 76. Distance also outscored random till iteration 63. Vote entropy and Kullback-Leibner divergence stagnated and then abruptly improved, gained a better score than random at iteration 49 and 99, respectively,and abruptly stagnated again. Soft vote entropy didn't improve the model performance at all.

Entropy and combination of distance and entropy obtained better performance on both tasks. Also Kullback-Leibner divergence obtained for the last 21 iterations high scores on both tasks.

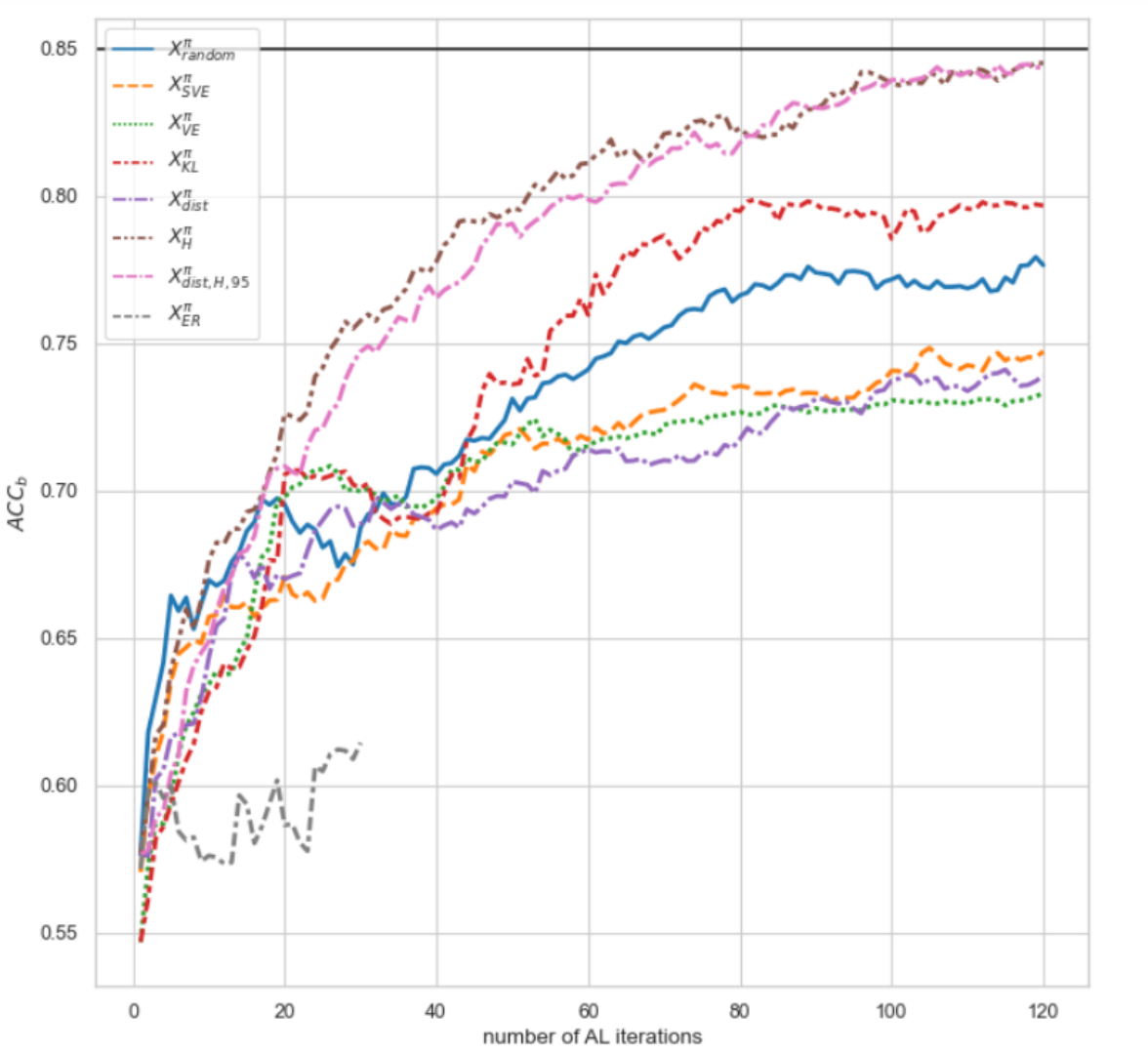


Figure 5.5.: **Fracture classification** learning curves for AL strategies under the **learning objective of fracture classification**.

Performance thresholds	0.5950	0.6375	0.6800	0.7225	0.7650	0.8075	0.8500
X^{π}_{random}	2	4	15	49	76	/	/
X^{π}_{SVE}	2	6	30	64	/	/	/
X^{π}_{VE}	6	11	18	52	/	/	/
X^{π}_{KL}	6	12	20	**46**	**59**	/	/
X^{π}_{dist}	3	10	24	84	/	/	/
X^{π}_{H}	2	5	**11**	**20**	**35**	**56**	/
$X^{\pi}_{dist,H,95}$	5	8	16	**26**	**38**	**66**	/

Table 5.9.: **fracture classification** Learning iterations for AL strategies needed to exceed a certain threshold corresponding to 75% - 100% of the **fracture classification** performance achieved on the whole data set.

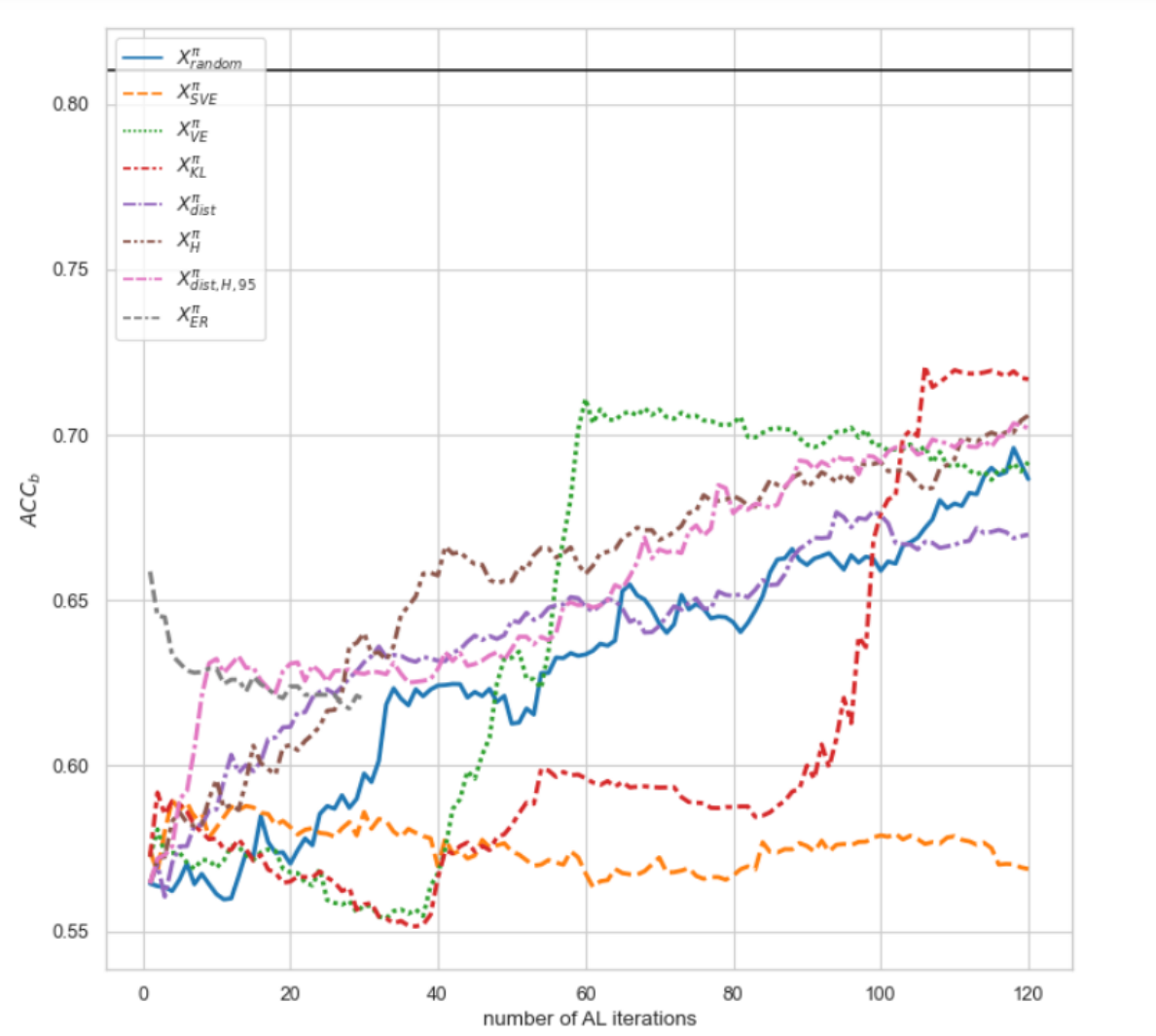


Figure 5.6.: **Cluster classification** learning curves for AL strategies under the **learning objective of fracture classification**.

Performance thresholds	0.5670	0.6075	0.6480	0.6885	0.7290	0.7695	0.8100
X^{π}_{random}	6	33	65	115	/	/	/
X^{π}_{SVE}	**1**	/	/	/	/	/	/
X^{π}_{VE}	**1**	47	**56**	**59**	/	/	/
X^{π}_{KL}	1	94	99	**103**	/	/	/
X^{π}_{dist}	**2**	**17**	**56**	/	/	/	/
X^{π}_{H}	**2**	**22**	**36**	**92**	/	/	/
$X^{\pi}_{dist,H,95}$	**2**	**8**	**57**	**89**	/	/	/

Table 5.10.: Learning iterations for AL strategies needed to exceed a certain threshold corresponding to 75% - 100% of the **cluster classification** performance achieved on the whole data set.

Cluster classification as AL objective The plots 5.7 and 5.8 show MLP performances after the acquisition of different training set sizes in regards to the AL objective of material cluster classification and fracture classification, respectively. Figure 5.7 depicts MLP cluster classification performance for different selection strategies after the acquisition of different training set sizes based on the AL objective of fracture classification. It is shown, that soft vote entropy, least confident, entropy, combination of distance and least confident, and combination of distance and entropy performed all right from the start similarly better than random, whereas the combination of distance and entropy performed during most of the learning process best, while even exceeding the general performance for the last 12 iterations.The combination of distance and least confident also outscored the other methods for the last 28 iterations. Distance outperformed random for the first 62 iterations. Kullback-Leibner divergence and vote entropy had the same scores. They increased in performance at iteration 90, and surpassed random after 100 iterations. Figure 5.7 depicts MLP cluster classification performance for different selection strategies after the acquisition of different training set sizes based on the AL objective of fracture classification.

Only Kullback-Leibner divergence and vote entropy outperformed random for the last iterations, whereas after 45 iterations they outscored random. Before about 20 learning steps not a single strategy performed better than random. After that the performances of the strategies exceeded those of random for some iterations, and can be ranked by: first least confident, second entropy, third distance with entropy, fourth distance with least confident and fifth soft vote entropy. Distance performed worst.

Only Kullback-Leibner divergence and vote entropy obtained better performance on both tasks, but only for the last 21 iterations.

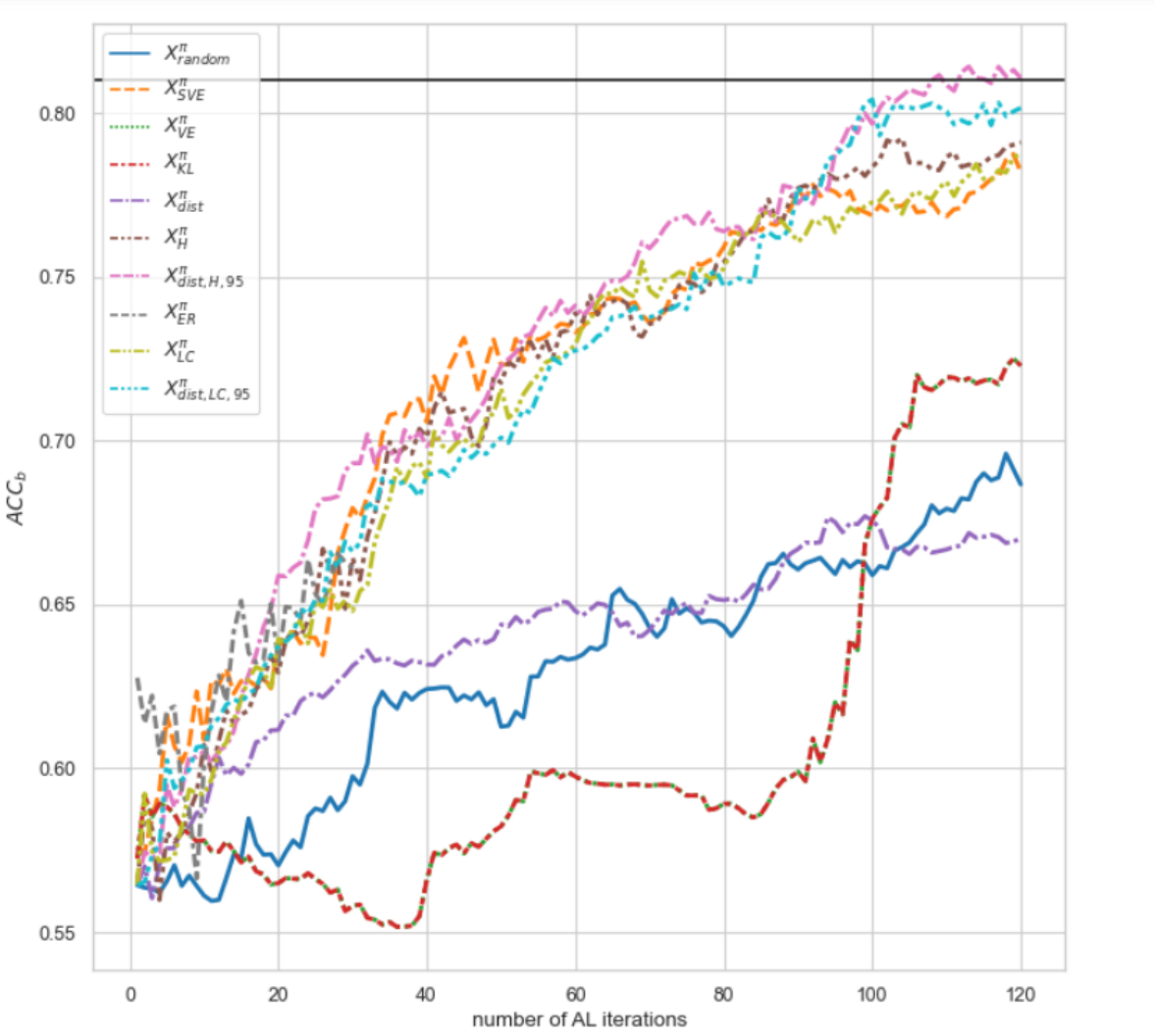


Figure 5.7.: **Cluster classification** learning curves for AL strategies under the **learning objective of cluster classification**.

Performance thresholds	0.5670	0.6075	0.6480	0.6885	0.7290	0.7695	0.8100
X^{π}_{random}	6	33	65	115	/	/	/
X^{π}_{SVE}	**1**	**5**	**27**	**33**	**45**	**89**	/
X^{π}_{VE}	**1**	92	99	**103**	/	/	/
X^{π}_{KL}	**1**	92	99	**103**	/	/	/
X^{π}_{dist}	**2**	**17**	**56**	/	/	/	/
X^{π}_{H}	**2**	**12**	**25**	**35**	**53**	**86**	/
$X^{\pi}_{dist,H,95}$	**2**	**14**	**19**	**29**	**53**	**78**	**109**
X^{π}_{LC}	**2**	**13**	**25**	**36**	**60**	**86**	/
$X^{\pi}_{dist,LC,95}$	**3**	**11**	**23**	**40**	**62**	**90**	/

Table 5.11.: Learning iterations for AL strategies needed to exceed a certain threshold corresponding to 75% - 100% of the **cluster classification** performance achieved on the whole data set.

Performance thresholds	0.5950	0.6375	0.6800	0.7225	0.7650	0.8075	0.8500
X^{π}_{random}	2	4	15	49	76	/	/
X^{π}_{SVE}	4	9	18	**45**	/	/	/
X^{π}_{VE}	6	12	20	**46**	**61**	/	/
X^{π}_{KL}	6	12	20	**46**	**61**	/	/
X^{π}_{dist}	3	10	24	84	/	/	/
X^{π}_{H}	2	11	16	**30**	**59**	/	/
$X^{\pi}_{dist,H,95}$	2	9	16	**28**	/	/	/
X^{π}_{LC}	2	5	**14**	**26**	/	/	/
$X^{\pi}_{dist,LC,95}$	3	5	**14**	**33**	/	/	/

Table 5.12.: Learning iterations for AL strategies needed to exceed a certain threshold corresponding to 75% - 100% of the **fracture classification** performance achieved on the whole data set.

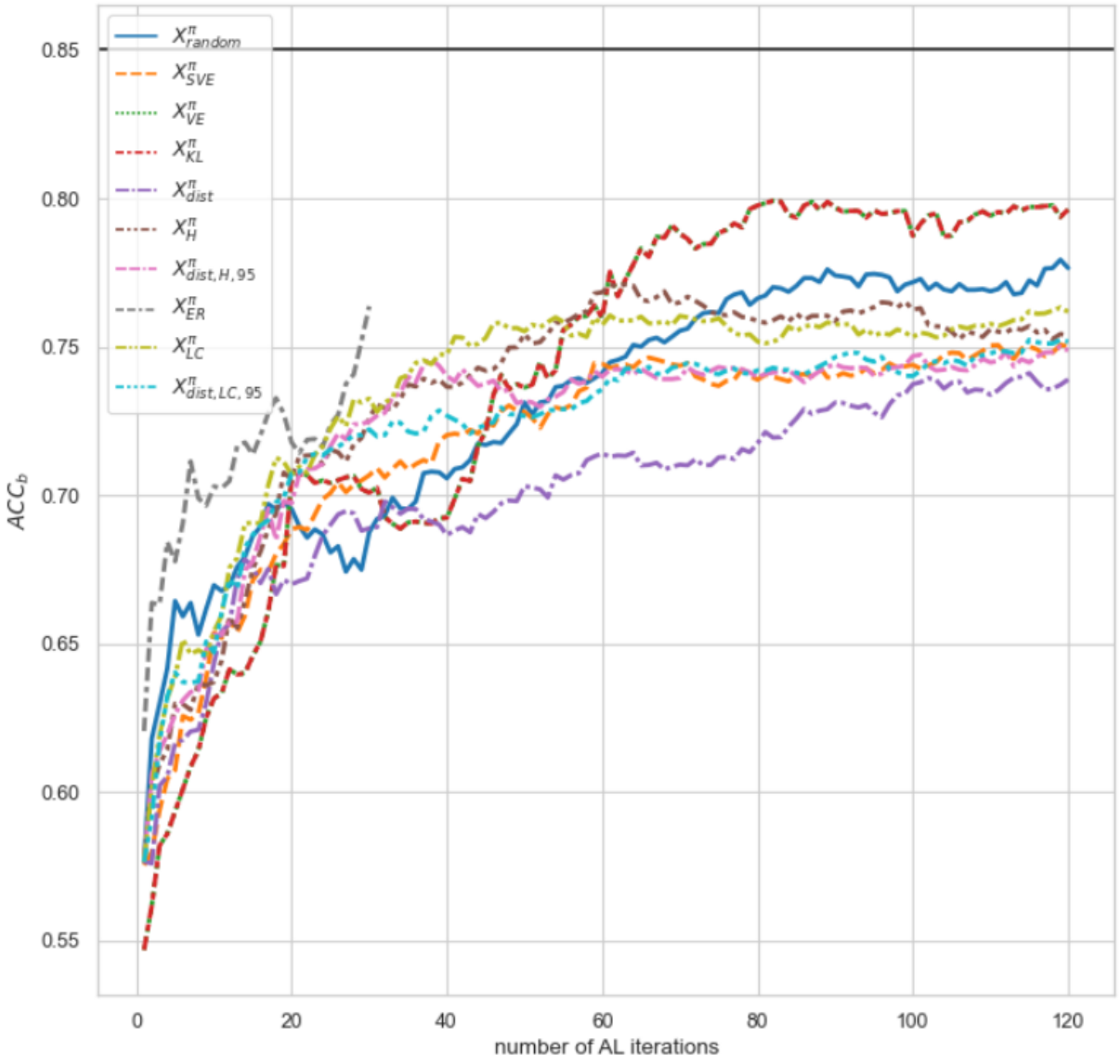


Figure 5.8.: **Fracture classification** learning curves for AL strategies under the **learning objective of cluster classification**.

6. Discussions

In the following the major findings of my study, the meaning of those findings, and how these findings relate to related work.

6.1. Main contributions

General Performance of RF and MLP on various features RF achieved for the VGG16 ($C_{4,3}$) Mean features the best performances in the fracture classification task. In contrast, the performances achieved by the RF for the material cluster classification task illustrated in weren't exclusively the highest for the VGG16 ($C_{4,3}$) Mean features. With the VGG16 ($C_{5,3}$) Mean features on the $20\mu m$ data better performances were attained, which makes sense because different layer of the VGG16 recognize different patterns in the image. It shows that, for fracture classification and material classification, and for different magnifications of the fracture image different textural patterns are of significance. For lower magnifications, for instance $10\mu m$, complex pattern features give better performance, which are recognized at the end of a CNN (Ling, Hutchinson, Antono, Paradiso, et al. 2017). On the other hand, for higher magnifications lower complexity patterns are of importance. This can also be highlighted by the MLP performances. It also shows, that the two classification tasks perform differently well on different VGG16 feature sets. This suggests, both tasks require divers features, which reflect differently complex features. In neither case were GLCM Haralick's features or LBP features successful in outscoring the VGG16 features, this is due to loss of information in the data when applying these methods. Also, the MLP outperformed the Random Forest on all three data sets and on nearly every feature set of those.

AL Performance It shows that except for applying RF under the AL objective of fracture classification the uncertainty sampling strategies entropy and least confident performed well for both, the RF and the MLP model under the learning objectives fracture classification and cluster classification. The above-mentioned exception is surprising

because for the case of applying RF under the AL objective of fracture classification the augmentation of the entropy by the distance measure increased the RF performance on both tasks. Also, the MLP model obtained for the augmented entropy optimized over the fracture task better performance on both tasks, whereas the entropy strategy by itself didn't do worse. Also for the RF under the AL objective of cluster classification combining uncertainty sampling strategies with distance measures didn't result in an improvement over those strategies. In the case of utilizing a combination of uncertainty sampling and distance measures MLP under the AL objective of cluster classification, on the other hand, reached better performances than utilizing those strategies without distance measures.
This shows that utilizing distance measures to augment a strategy can help to improve model performance even further than the strategy alone does, for instance for the cluster classification task the utilization of distance measure combined with entropy over the learning objective of cluster classification even resulted in the same MLP performance as when trained on the whole data set, while only acquiring about 1/10 of the data. The distance measure strategy on its own did poorly for all AL tasks.
Using committee querying strategies can also help to improve model performance, like the maximum uncertainty method, which did great on the RF for fracture classification on the fracture classification learning objective. However, the results for the other three used committee querying strategies soft vote entropy, vote entropy, and Kullback-Leibner divergence were inconclusive. The number of committee learners used for those was set to 2, therefore the utility measures were not sensitive in choosing the next data point to query.
In general, I found that uncertainty sampling methods performed the best on the fractographic data. Augmenting those strategies by distance measures, to improve their success in terms of model performance highly depends on the learning task and model at hand.

6.2. Relate to Literature

The transfer learning method of using VGG16 to extract features as demonstrated in Ling, Hutchinson, Antono, B. DeCost, et al. 2017 and B. L. DeCost, Francis, and Holm 2017 created reliable features that outperformed both LBP and Haralick's features when applied, whereas they used Random Forest models, the here applied MLP outperformed a Random Forest model, even if smaller data was used. Similarly to other classifica-

tion tasks, for instance Beluch et al. 2018 and Beck et al. 2021, where large data sets containing various kinds of images were used, uncertainty sampling also performs well for fractographic classification tasks. The sequential learning strategy maximum uncertainty and the uncertainty estimation framework proposed in Ling, Hutchinson, Antono, Paradiso, et al. 2017 can also be applied to active learning. Also utility measures based on distances in the input space, successfully applied to sequential learning in Völker et al. 2021 can be in some cases transferred to active learning.

6.3. Practical value, use, contributions to the body of knowledge

Through the application of active learning, fractographic data sets of smaller size were able to hold the information needed to properly classify fractographic tasks as accurately as a much larger data set. By implication, resources like time and money to obtain image labelings can be saved. As the information of an extensive data set is consolidated into a smaller one, human experts are more likely to gain an insight into specific fracture behaviors.

6.4. What does future research need to do

Due to the lack of comparability, results for minimization of the expected classification error were not discussed; in the future, strategies for minimization of the expected error and for reducing variance will need to be explored. Also this study has only discussed active learning on steel fractographs, active learning for other material cases have to be investigated.

7. Conclusion

It was shown that an MLP, a shallow neural network can outperform a Random Forest model on small data. The performance of selection strategies for active learning with an RF and MLP on image classification tasks was compared. It was shown that uncertainty sampling methods consistently outperform other methods in particular hypothesis space search methods. It was also shown that incorporating distance measures in the input space to uncertainty sampling strategies can in some cases improve data quality further. It concludes by showing results on a real-world application in fractography fracture and material classification that active learning with uncertainty sampling works well in both fractographic classification scenarios, and can even produce data from which a model is much more effective on both tasks than a similar model trained on randomly sampled data. Through the compression of information over data distribution, machine learning models converge faster and are less computationally expensive. However, since the data was not plentiful, further data points had to be artificially generated from the existing ones, which leads to unwanted similarity in the data, which may give active learning methods an edge over random sampling.

Bibliography

[1] Charu Aggarwal, Alexander Hinneburg, and Daniel Keim. "On the Surprising Behavior of Distance Metric in High-Dimensional Space". In: *First publ. in: Database theory, ICDT 200, 8th International Conference, London, UK, January 4 - 6, 2001 / Jan Van den Bussche ... (eds.). Berlin: Springer, 2001, pp. 420-434 (=Lecture notes in computer science ; 1973)* (Feb. 2002).

[2] Timo Ahonen, Abdenour Hadid, and Matti Pietikäinen. "Face Recognition with Local Binary Patterns". In: *Computer Vision - ECCV 2004*. Ed. by Tomás Pajdla and Jiří Matas. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 469–481.

[3] Patrick Bangert. *Data efficiency: solving a problem of visual analytics.* URL: https://www.linkedin.com/pulse/data-efficiency-solving-problem-visual-analytics-patrick-bangert/. accessed: 22.07.2021.

[4] Ximena Bastidas, Flavio Prieto-Ortíz, and Édgar Espejo-Mora. "Fractographic classification in metallic materials by using 3D processing and computer vision techniques". In: *Revista Facultad de Ingeniería* 25 (Dec. 2016), pp. 83–96.

[5] M.X. Bastidas-Rodriguez, F.A. Prieto-Ortiz, and Edgar Espejo. "Fractographic classification in metallic materials by using computer vision". In: *Engineering Failure Analysis* 59 (2016), pp. 237–252. ISSN: 1350-6307. DOI: https://doi.org/10.1016/j.engfailanal.2015.10.008. URL: https://www.sciencedirect.com/science/article/pii/S1350630715301084.

[6] María X. Bastidas-Rodriguez, Luisa Polania, Adrien Gruson, and Flavio Prieto-Ortiz. "Deep Learning for fractographic classification in metallic materials". In: *Engineering Failure Analysis* 113 (2020), p. 104532. ISSN: 1350-6307. DOI: https://doi.org/10.1016/j.engfailanal.2020.104532. URL: https://www.sciencedirect.com/science/article/pii/S1350630720300364.

[7] Maria-Ximena Bastidas-Rodríguez, Flavio-Augusto Prieto-Ortiz, and Luisa F. Polanía. "A Textural Deep Neural Network Combined With Handcrafted Features for Mechanical Failure Classification". In: *2019 IEEE International Conference on Industrial Technology (ICIT)*. 2019, pp. 847–852. DOI: 10.1109/ICIT.2019.8755046.

[8] Nathan Beck, Durga Sivasubramanian, Apurva Dani, Ganesh Ramakrishnan, and Rishabh Iyer. *Effective Evaluation of Deep Active Learning on Image Classification Tasks*. 2021. arXiv: 2106.15324 [cs.CV].

[9] William H. Beluch, Tim Genewein, Andreas Nürnberger, and Jan M. Köhler. "The Power of Ensembles for Active Learning in Image Classification". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2018.

[10] Cody Coleman, Edward Chou, Sean Culatana, Peter Bailis, Alexander C. Berg, Roshan Sumbaly, Matei Zaharia, and I. Zeki Yalniz. *Similarity Search for Efficient Active Learning and Search of Rare Concepts*. 2021. URL: https://openreview.net/forum?id=G67PtYbCImX.

[11] Tivadar Danka and Peter Horvath. "modAL: A modular active learning framework for Python". In: (). available on arXiv at https://arxiv.org/abs/1805.00979. URL: https://github.com/modAL-python/modAL.

[12] Brian L. DeCost, Toby Francis, and Elizabeth A. Holm. "Exploring the microstructure manifold: Image texture representations applied to ultrahigh carbon steel microstructures". In: *Acta Materialia* 133 (2017), pp. 30–40. ISSN: 1359-6454. DOI: https://doi.org/10.1016/j.actamat.2017.05.014. URL: https://www.sciencedirect.com/science/article/pii/S1359645417303919.

[13] Vasilios Duros, Jonathan Grizou, Weimin Xuan, Zied Hosni, De-Liang Long, Haralampos N. Miras, and Leroy Cronin. "Human versus Robots in the Discovery and Crystallization of Gigantic Polyoxometalates". In: *Angewandte Chemie International Edition* 56.36 (2017), pp. 10815–10820. DOI: https://doi.org/10.1002/anie.201705721. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/anie.201705721. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/anie.201705721.

[14] Meenakshi Garg and Gaurav Dhiman. "A novel content-based image retrieval approach for classification using GLCM features and texture fused LBP variants". In: *Neural Computing and Applications* 33 (2021), pp. 1311–1328. URL: https://doi.org/10.1007/s00521-020-05017-z.

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

[16] Margherita Grandini, Enrico Bagli, and Giorgio Visani. *Metrics for Multi-Class Classification: an Overview*. 2020. arXiv: 2008.05756 [stat.ML].

[17] Mryka Hall-Beyer. *GLCM Texture: A Tutorial v. 3.0 March 2017*. Mar. 2017. DOI: 10.13140/RG.2.2.12424.21767.

[18] Robert M. Haralick, K. Shanmugam, and Its'Hak Dinstein. "Textural Features for Image Classification". In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-3.6 (1973), pp. 610–621. DOI: 10.1109/TSMC.1973.4309314.

[19] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[20] Dong-chen He and Li Wang. "Texture Unit, Texture Spectrum, And Texture Analysis". In: *IEEE Transactions on Geoscience and Remote Sensing* 28.4 (1990), pp. 509–512. DOI: 10.1109/TGRS.1990.572934.

[21] Kurt Hornik. "Approximation capabilities of multilayer feedforward networks". In: *Neural Networks* 4.2 (1991), pp. 251–257. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(91)90009-T. URL: https://www.sciencedirect.com/science/article/pii/089360809190009T.

[22] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[23] Hideaki Ishibashi and Hideitsu Hino. "Stopping criterion for active learning based on deterministic generalization bounds". In: *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*. Ed. by Silvia Chiappa and Roberto Calandra. Vol. 108. Proceedings of Machine Learning Research. PMLR, 26–28 Aug 2020, pp. 386–397. URL: https://proceedings.mlr.press/v108/ishibashi20a.html.

[24] Ludvig Karlsson and Oskar Bonde. *A Comparison of Selected Optimization Methods for Neural Networks*. 2020.

[25] Samah Khawaled, Michael Zibulevsky, and Yehoshua Y. Zeevi. *Texture and Structure Two-view Classification of Images*. 2019. arXiv: 1908.09264 [cs.CV].

[26] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[27] S. Kullback and R. A. Leibler. "On Information and Sufficiency". In: *The Annals of Mathematical Statistics* 22.1 (1951), pp. 79–86. DOI: 10.1214/aoms/1177729694. URL: https://doi.org/10.1214/aoms/1177729694.

[28] Xin Li and Yuhong Guo. "Adaptive Active Learning for Image Classification". In: *2013 IEEE Conference on Computer Vision and Pattern Recognition*. 2013, pp. 859–866. DOI: 10.1109/CVPR.2013.116.

[29] Julia Ling, Maxwell Hutchinson, Erin Antono, Brian DeCost, Elizabeth A. Holm, and Bryce Meredig. "Building data-driven models with microstructural images: Generalization and interpretability". In: *Materials Discovery* 10 (2017), pp. 19–28. ISSN: 2352-9245. DOI: https://doi.org/10.1016/j.md.2018.03.002. URL: https://www.sciencedirect.com/science/article/pii/S235292451730042X.

[30] Julia Ling, Maxwell Hutchinson, Erin Antono, Sean Paradiso, and Bryce Meredig. "High-Dimensional Materials and Process Optimization Using Data-Driven Experimental Design with Well-Calibrated Uncertainty Estimates". In: *Integrating Materials and Manufacturing Innovation* 6.3 (July 2017), pp. 207–217. ISSN: 2193-9772. DOI: 10.1007/s40192-017-0098-z. URL: http://dx.doi.org/10.1007/s40192-017-0098-z.

[31] Turab Lookman, Prasanna V. Balachandran, Dezhen Xue, and Ruihao Yuan. "Active learning in materials science with emphasis on adaptive sampling using uncertainties for targeted design". In: *npj Computational Mathematics* 5, 21 (Feb. 2019), p. 21. DOI: 10.1038/s41524-019-0153-8.

[32] Warren S. McCulloch and Walter Pitts. "A Logical Calculus of the Ideas Immanent in Nervous Activity". In: *The Bulletin of Mathematical Biophysics* 5.4 (1943), pp. 115–133. DOI: 10.1007/bf02478259.

[33] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. second. New York, NY, USA: Springer, 2006.

[34] Thais Oshiro, Pedro Perez, and José Baranauskas. "How Many Trees in a Random Forest?" In: vol. 7376. July 2012. ISBN: 978-3-642-31536-7. DOI: 10.1007/978-3-642-31537-4_13.

[35] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[36] Tatu Pinomaa, Ivan Yashchuk, Matti Lindroos, Tom Andersson, Nikolas Provatas, and Anssi Laukkanen. "Process-Structure-Properties-Performance Modeling for Selective Laser Melting". In: *Metals* 9.11 (2019). ISSN: 2075-4701. DOI: 10.3390/met9111138. URL: https://www.mdpi.com/2075-4701/9/11/1138.

[37] Nathir Rawashdeh, Walid Khraisat, and Henrik Borgström. "Pinning Effect of Pores on Grain Growth in Sintered Steel". In: *Jordan Journal of Mechanical and Industrial Engineering* 11 (Apr. 2017), pp. 73–78.

[38] Kristopher Reyes and Warren B Powell. *Optimal Learning for Sequential Decisions in Laboratory Experimentation*. 2020. arXiv: 2004.05417 [cs.LG].

[39] Jonathan Rougier. "Ensemble Averaging and Mean Squared Error". In: *Journal of Climate* 29.24 (2016), pp. 8865–8870. DOI: 10.1175/JCLI-D-16-0012.1. URL: https://journals.ametsoc.org/view/journals/clim/29/24/jcli-d-16-0012.1.xml.

[40] Burr Settles. *Active Learning Literature Survey*. Computer Sciences Technical Report 1648. University of Wisconsin–Madison, 2009. URL: http://axon.cs.byu.edu/~martinez/classes/778/Papers/settles.activelearning.pdf.

[41] Burr Settles. *Active learning*. USA: Morgan Claypool Publishers, 2012. ISBN: 956875.

[42] Burr Settles, Mark Craven, and Lewis Friedland. "Active learning with real annotation costs". In: (Jan. 2008).

[43] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. USA: Cambridge University Press, 2014. ISBN: 1107057132.

[44] Claude E. Shannon. "A mathematical theory of communication." In: *Bell Syst. Tech. J.* 27.3 (1948), pp. 379–423. URL: http://dblp.uni-trier.de/db/journals/bstj/bstj27.html#Shannon48.

[45] Brian C. Smith, Burr Settles, William C. Hallows, Mark W. Craven, and John M. Denu. "SIRT3 Substrate Specificity Determined by Peptide Arrays and Machine Learning". In: *ACS Chemical Biology* 6.2 (2011). PMID: 20945913, pp. 146–157. DOI: 10.1021/cb100218d. eprint: https://doi.org/10.1021/cb100218d. URL: https://doi.org/10.1021/cb100218d.

[46] The pandas development team. *pandas-dev/pandas: Pandas*. Version latest. Feb. 2020. DOI: 10.5281/zenodo.3509134. URL: https://doi.org/10.5281/zenodo.3509134.

[47] Gaurav Vishwakarma, Aditya Sonpal, and Johannes Hachmann. "Metrics for Benchmarking and Uncertainty Quantification: Quality, Applicability, and Best Practices for Machine Learning in Chemistry". In: *Trends in Chemistry* 3.2 (Feb. 2021), pp. 146–156. ISSN: 2589-5974. DOI: 10.1016/j.trechm.2020.12.004. URL: http://dx.doi.org/10.1016/j.trechm.2020.12.004.

[48] Christoph Völker, Rafia Firdous, Dietmar Stephan, and Sabine Kruschwitz. *Sequential learning to accelerate discovery of alkali-activated binders*. June 2021. DOI: 10.13140/RG.2.2.18388.94087/1.

[49] Yaqing Wang, Quanming Yao, James Kwok, and Lionel M. Ni. *Generalizing from a Few Examples: A Survey on Few-Shot Learning*. 2020. arXiv: 1904.05046 [cs.LG].

[50] Michael L. Waskom. "seaborn: statistical data visualization". In: *Journal of Open Source Software* 6.60 (2021), p. 3021. DOI: 10.21105/joss.03021. URL: https://doi.org/10.21105/joss.03021.

[51] Jingbo Zhu, Huizhen Wang, Eduard Hovy, and Matthew Ma. "Confidence-Based Stopping Criteria for Active Learning for Data Annotation". In: *ACM Trans. Speech Lang. Process.* 6.3 (Apr. 2010). ISSN: 1550-4875. DOI: 10.1145/1753783.1753784. URL: https://doi.org/10.1145/1753783.1753784.

[52] Xiaojin Zhu and Andrew Goldberg. *Introduction to Semi-Supervised Learning*. 2009.

Acknowledgements

I have received a great deal of support and assistance while writing this thesis. I would first like to thank my supervisor, Dr.-Ing. Christoph Völker. He was instrumental in formulating my research question and methodology. The insightful feedback you gave me pushed me to sharpen my thinking and elevated my work to a higher level.

I also want to thank my supervisor Professor Sebastian Sager for always being available for any questions regarding this study.

Furthermore, I would like to thank my dear friend Jan Krause for his emotional support and excellent feedback on my writing and programming.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe.

Ort, Datum, Unterschrift

Appendices

A. Active Learning with Random Forest classifier - $F1$ scores

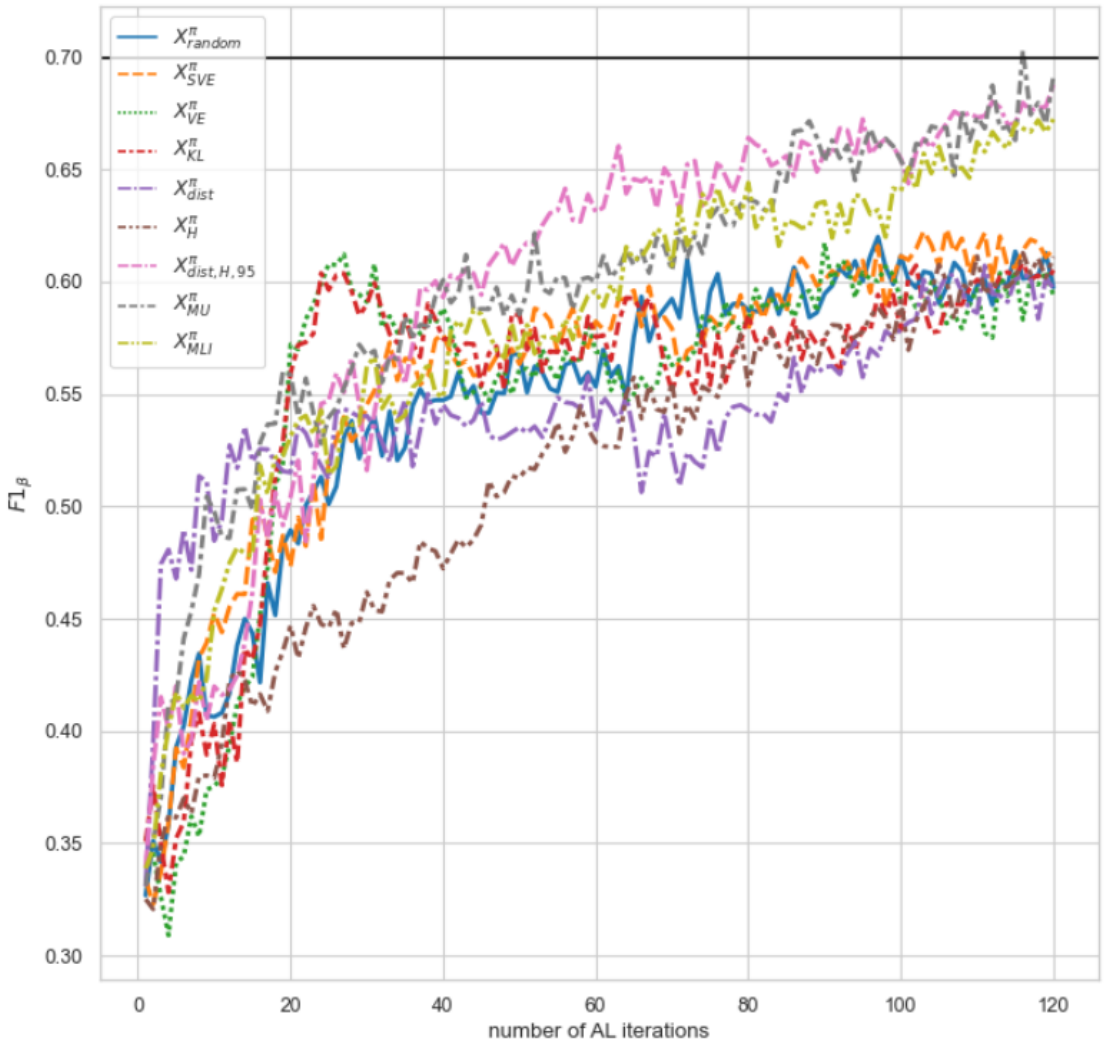


Figure A.1.: Fracture classification learning curves for AL strategies under the learning objective of fracture classification

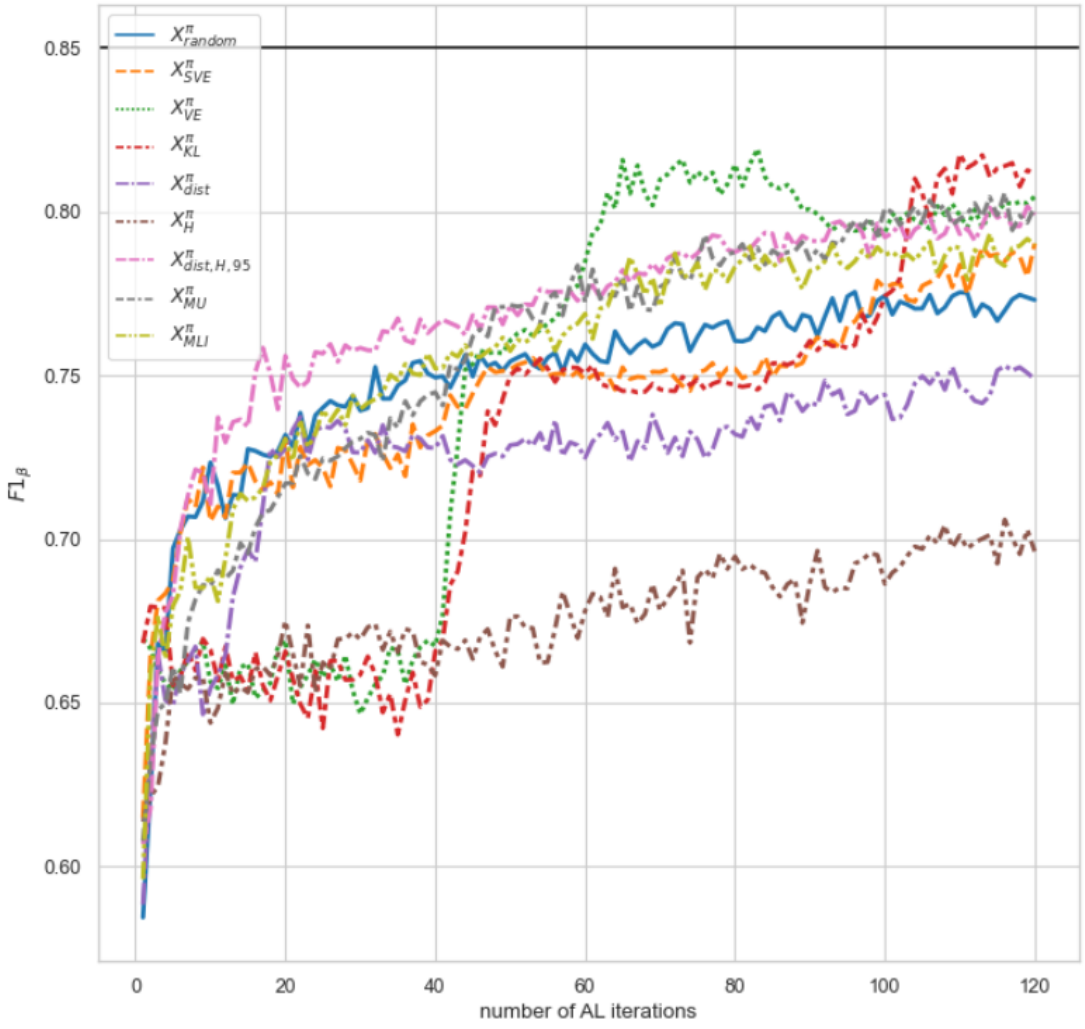


Figure A.2.: Cluster classification learning curves for AL strategies under the learning objective of fracture classification

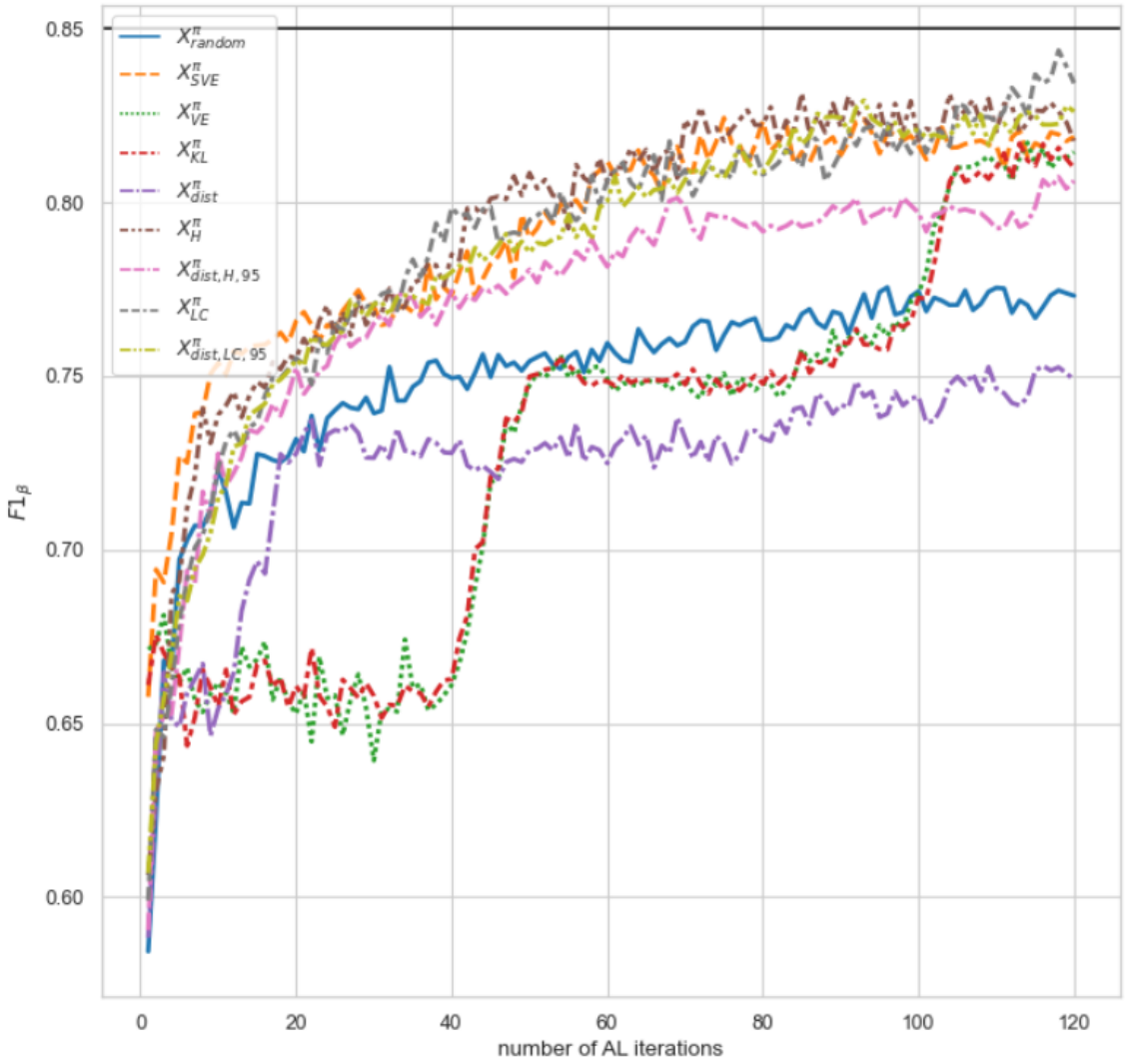


Figure A.3.: Cluster classification learning curves for AL strategies under the learning objective of cluster classification

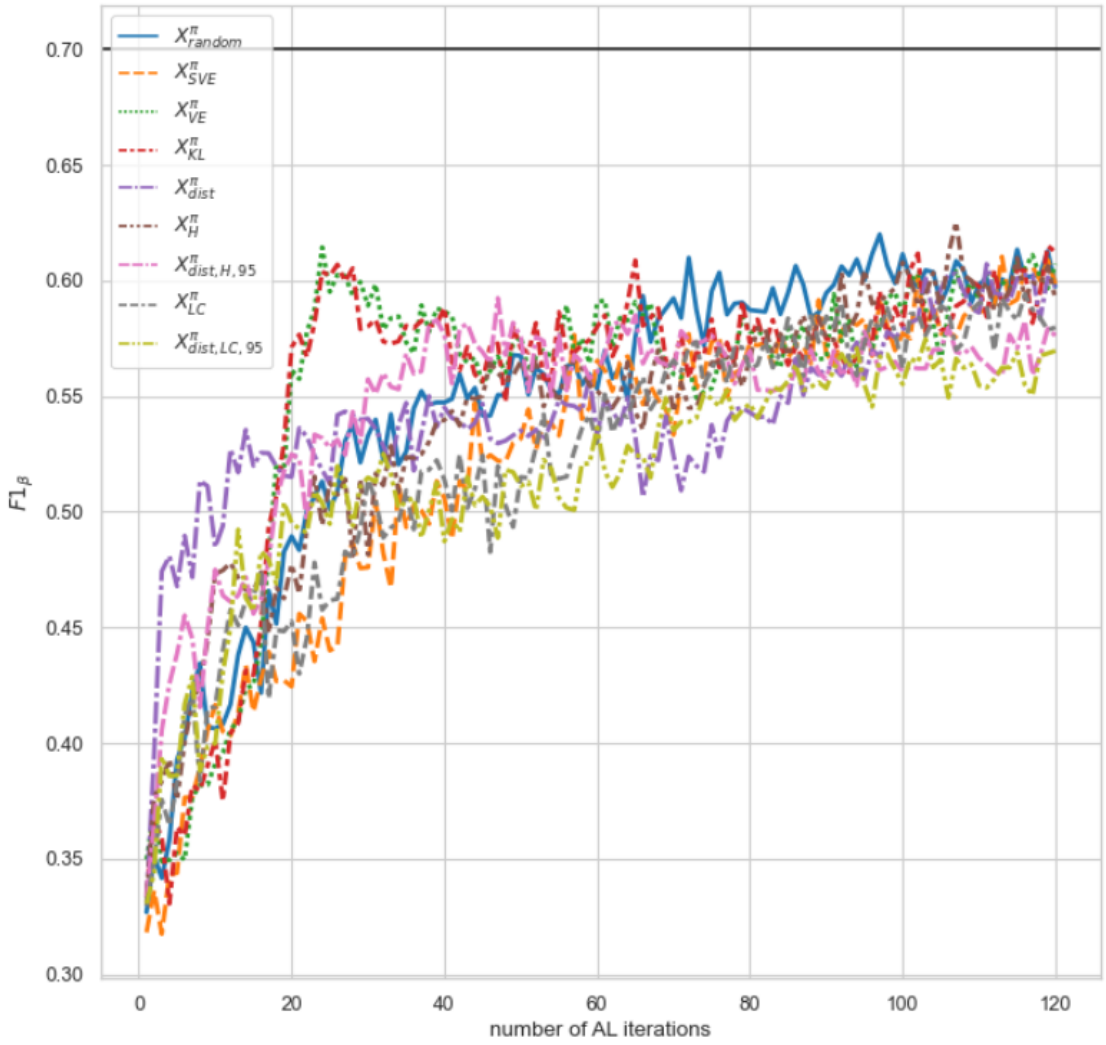


Figure A.4.: Fracture classification learning curves for AL strategies under the learning objective of cluster classification

B. Active Learning with Multilayer Perceptron classifier - $F1$ scores

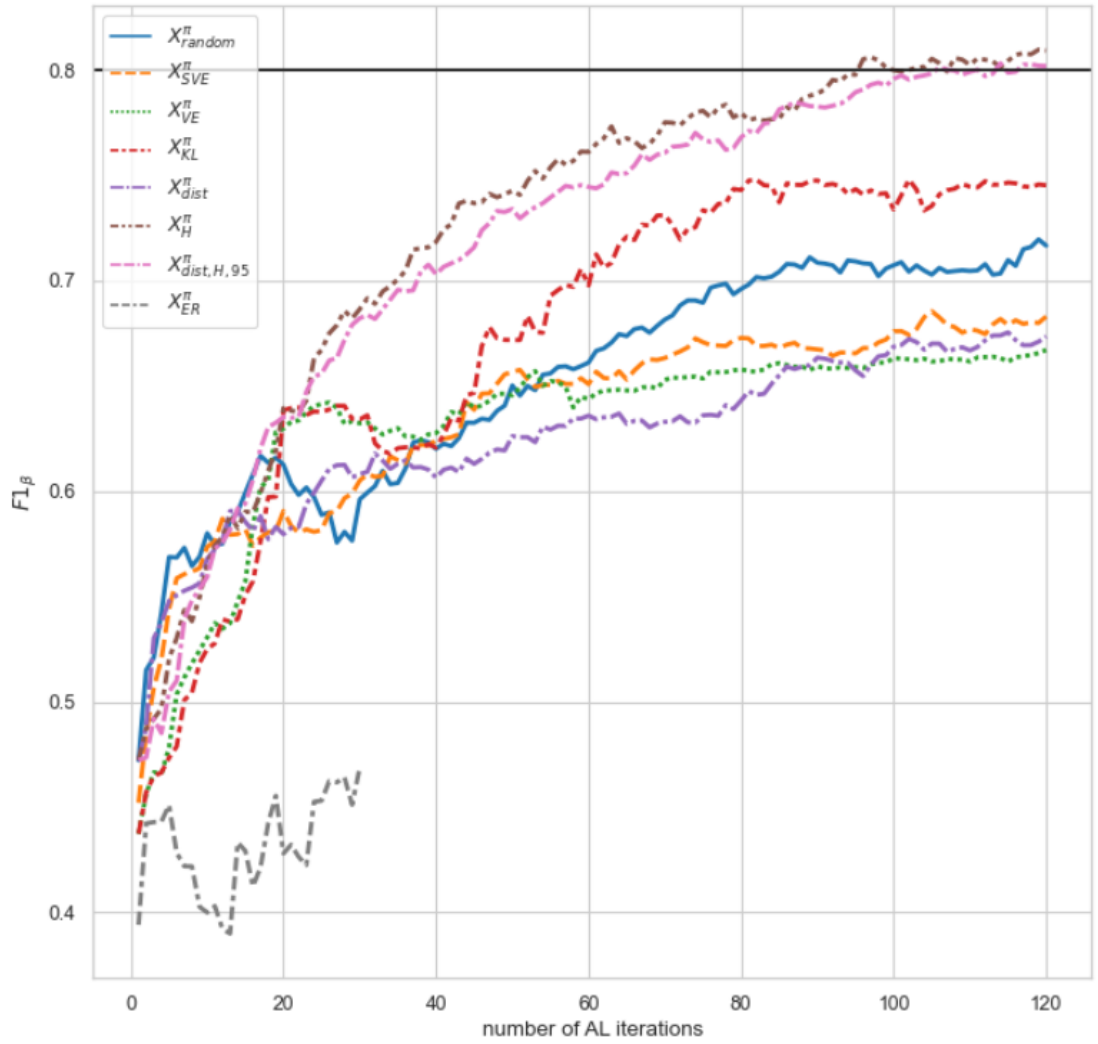


Figure B.1.: Fracture classification learning curves for AL strategies under the learning objective of fracture classification

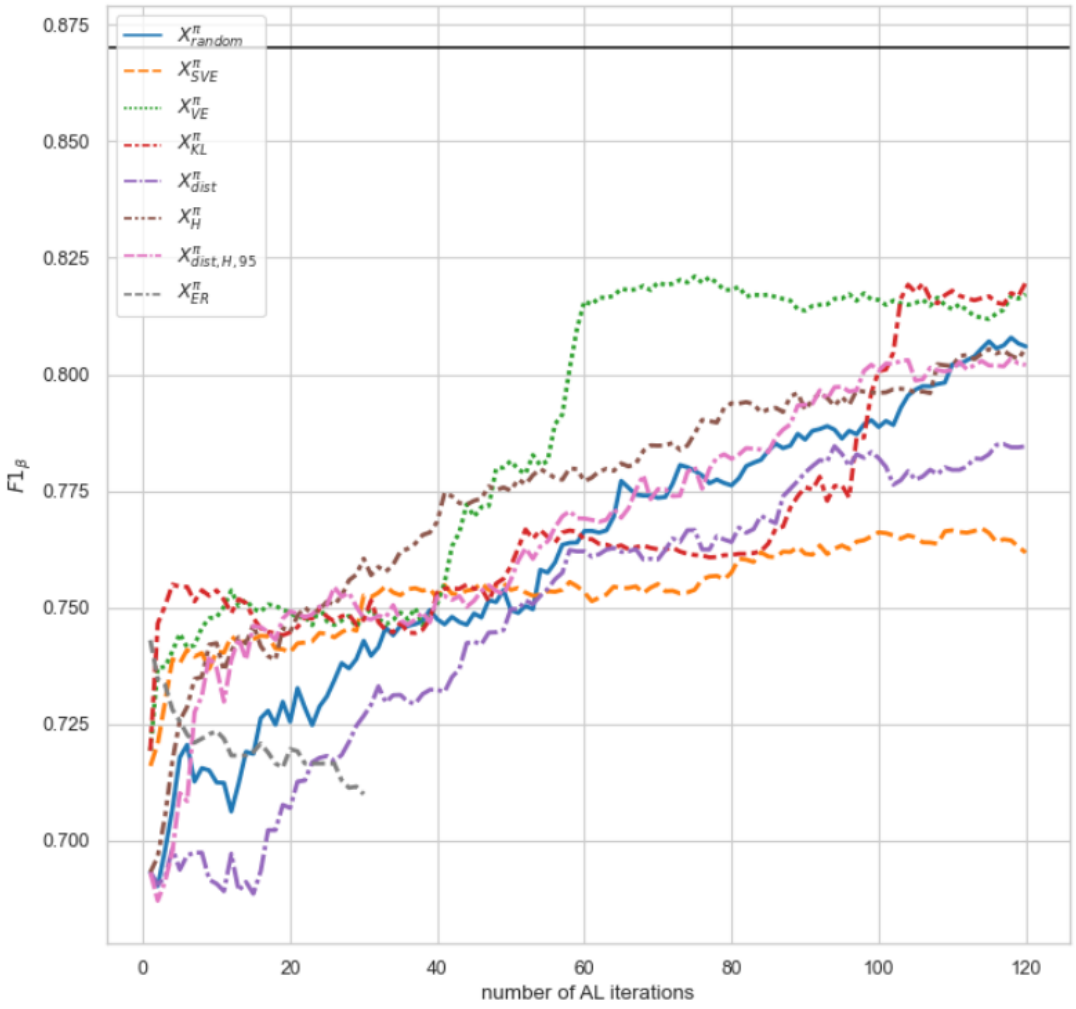


Figure B.2.: Cluster classification learning curves for AL strategies under the learning objective of fracture classification

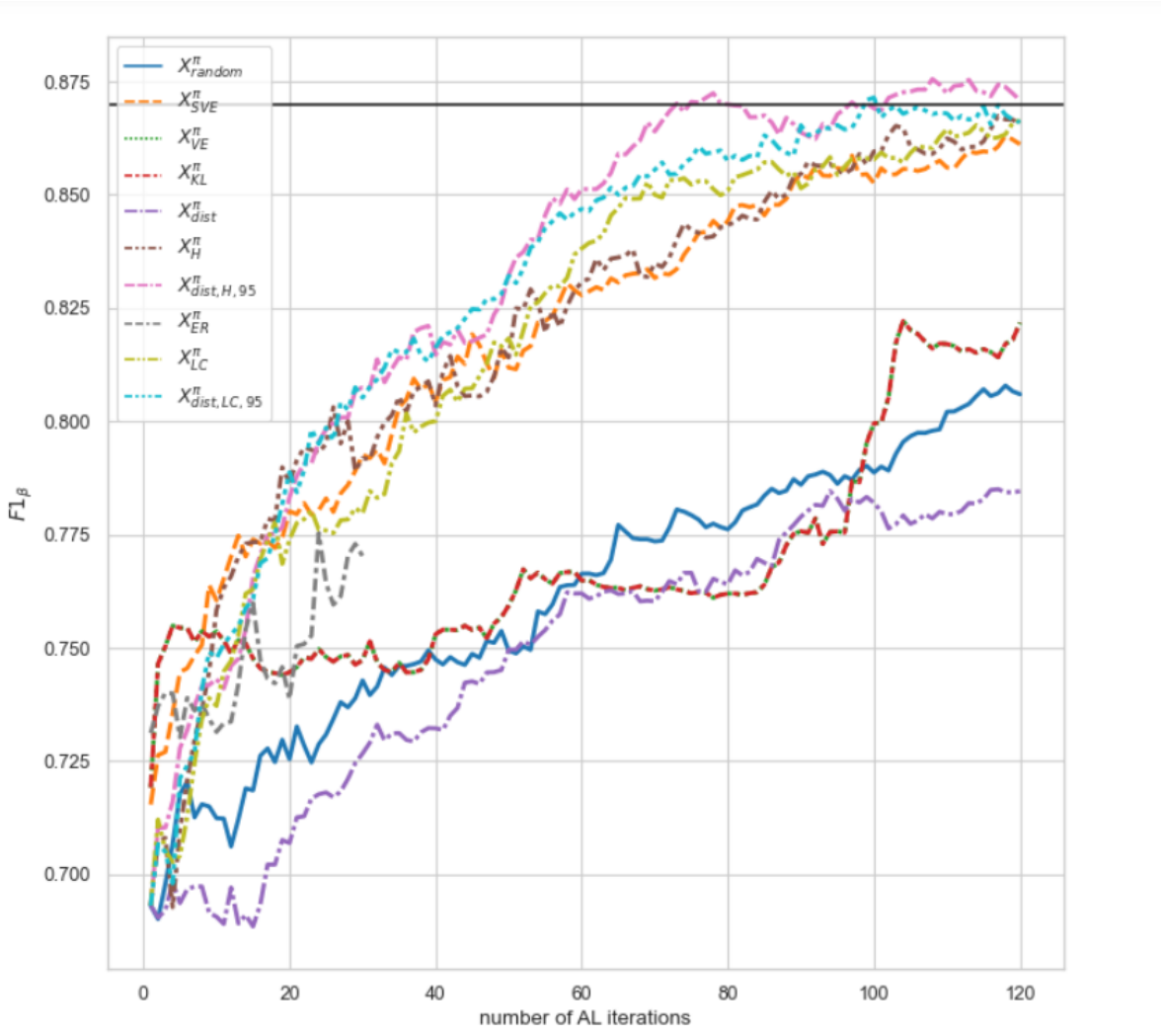


Figure B.3.: Cluster classification learning curves for AL strategies under the learning objective of cluster classification

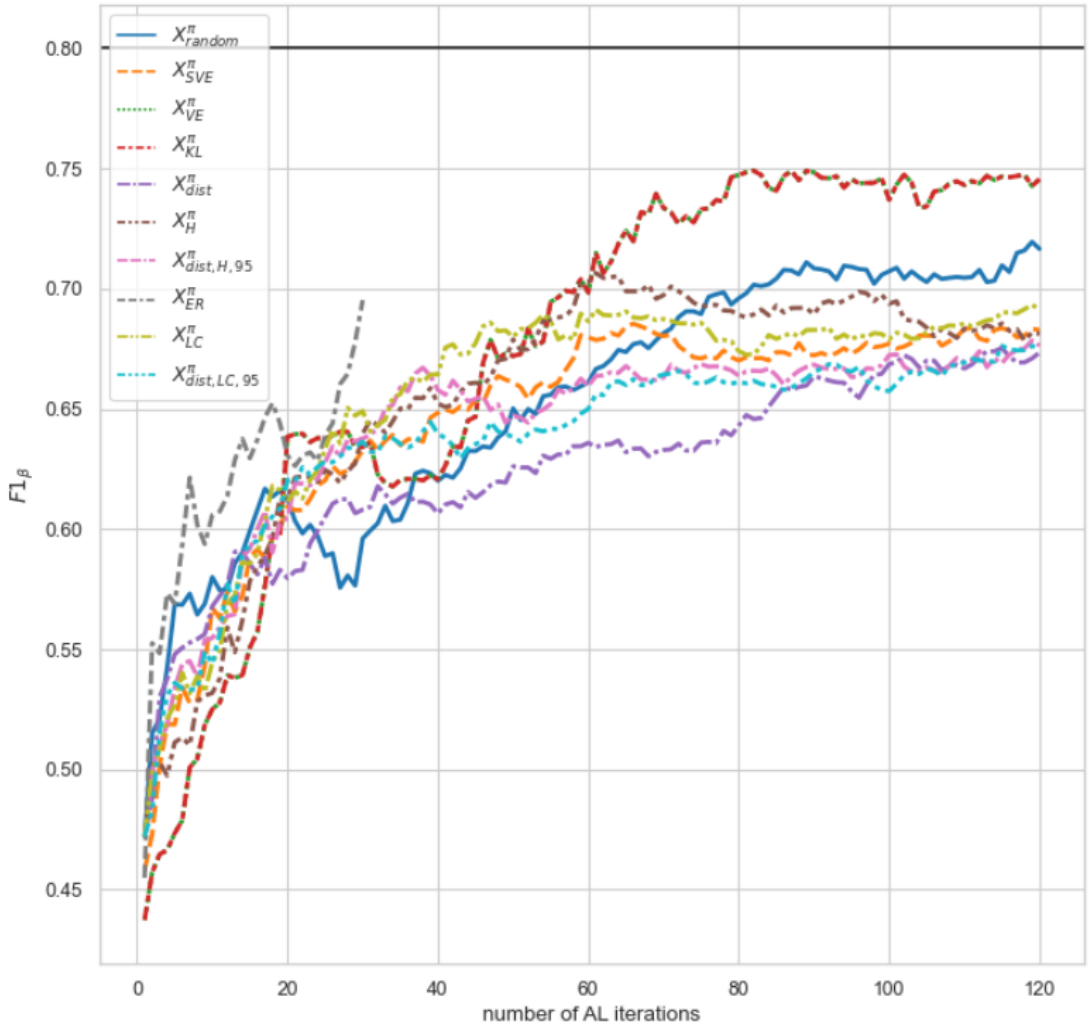


Figure B.4.: Fracture classification learning curves for AL strategies under the learning objective of cluster classification