Institute for Data Science in Mechanical Engineering

RWTH AACHEN UNIVERSITY

The present work was submitted to the Institute for Data Science in Mechanical Engineering.
Diese Arbeit wurde vorgelegt am Institut für Data Science im Maschinenbau.

Active Learning of Image Classifier for Application In Robotics

Aktives Lernen von Bildklassifikatoren für Anwendungen in der Robotik

Master Thesis
Masterarbeit

Presented by / Vorgelegt von

Jyotirmaya Patra
Matr.Nr.: 415710

Supervised by / Betreut von
Jongseok Lee M.Sc.
(Institute of Robotics and Mechatronics,
German Aerospace Agency)

Paul Brunzema M.Sc.
(Institute for Data Science in Mechanical Engineering)

1st Examiner / 1. Prüfer
Univ.-Prof. Dr. sc. Sebastian Trimpe
(Institute for Data Science in Mechanical Engineering)

2nd Examiner / 2. Prüfer
Dr. Friedrich Solowjow
(Institute for Data Science in Mechanical Engineering)

Aachen, January 02, 2024

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne die Benutzung anderer als der angegebenen Hilfsmittel selbstständig verfasst habe; die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe des Literaturzitats gekennzeichnet.

Aachen, den Januar 02, 2024

Abstract

The last decade has increasingly witnessed the use of robots in varied applications, ranging from industrial automation to space exploration. A key component in these robotics systems is their ability to extract semantic information regarding the objects they interact with when deployed into the real world. The field of object recognition has seen significant advances in accuracy and robustness using new artificial intelligence methods and machine learning algorithms. However, these methods partially address the cost and sample efficiency issue for acquiring the information required to learn such algorithms successfully. The use of active learning becomes essential to address these issues.

Prior active learning methods assume offline access to the entire dataset ahead of time. However, in the context of mobile robotics, where robots learn continuously by interacting with new objects and environments, the study of active learning in a streaming setting is more relevant. Motivated by these observations, we develop a data-driven framework that combines reinforcement learning with active learning for streaming scenarios. We propose our framework in two steps: first, given the abundance of labeled datasets for object recognition, we learn an active learning policy from these datasets using reinforcement learning, and in the second step, we apply this learned policy to select the most informative samples encountered in a stream of unlabeled data.

In this work, we model the elements of reinforcement learning in a stream-based active learning scenario. We then systematically investigate the performance of our proposed data-driven framework using diverse state representations for our reinforcement learning agent. We further provide insights into the advantages and challenges of each representation and also compare our method with relevant active learning baselines. Finally, our proposed methods significantly improve the selection of the most informative data over the considered streaming baseline, however, demonstrate on-par performance with the pool-based baselines.

Kurzzusammenfassung

In den letzten zehn Jahren wurden zunehmend Roboter in verschiedenen Anwendungen eingesetzt, die von der industriellen Automatisierung bis zur Weltraumforschung reichen. Eine Schlüsselkomponente dieser Robotersysteme ist ihre Fähigkeit, semantische Informationen über die Objekte zu extrahieren, mit denen sie beim Einsatz in der realen Welt interagieren. Auf dem Gebiet der Objekterkennung wurden durch neue Methoden der künstlichen Intelligenz und Algorithmen des maschinellen Lernens erhebliche Fortschritte in Bezug auf Genauigkeit und Robustheit erzielt. Diese Methoden gehen jedoch nur teilweise auf das Problem der Kosten und der Stichprobeneffizienz bei der Erfassung der Informationen ein, die für das erfolgreiche Erlernen solcher Algorithmen erforderlich sind. Der Einsatz von aktivem Lernen ist daher unerlässlich, um diese Probleme zu lösen.

Frühere Methoden des aktiven Lernens setzen voraus, dass der gesamte Datensatz im Voraus offline zur Verfügung steht. Im Kontext der mobilen Robotik, wo Roboter durch die Interaktion mit neuen Objekten und Umgebungen kontinuierlich lernen, ist die Untersuchung des aktiven Lernens in einer Streaming-Umgebung jedoch von größerer Bedeutung. Motiviert durch diese Beobachtungen entwickeln wir einen datengesteuerten Rahmen, der Verstärkungslernen mit aktivem Lernen für Streaming-Szenarien kombiniert. Wir schlagen unseren Rahmen in zwei Schritten vor: Zunächst lernen wir angesichts der Fülle von beschrifteten Datensätzen für die Objekterkennung eine aktive Lernstrategie aus diesen Datensätzen mit Hilfe von Verstärkungslernen, und im zweiten Schritt wenden wir diese gelernte Strategie an, um die informativsten Proben auszuwählen, die in einem Strom von unbeschrifteten Daten vorkommen.

In dieser Arbeit modellieren wir die Elemente des Verstärkungslernens in einem strombasierten aktiven Lernszenario. Anschließend untersuchen wir systematisch die Leistung unseres vorgeschlagenen datengesteuerten Rahmens unter Verwendung verschiedener Zustandsdarstellungen für unseren Verstärkungslernagenten. Darüber hinaus geben wir Einblicke in die Vorteile und Herausforderungen der einzelnen Repräsentationen und vergleichen unsere Methode mit relevanten aktiven Lerngrundlagen. Schließlich verbessern die von uns vorgeschlagenen Methoden die Auswahl der informativsten Daten im Vergleich zu den betrachteten Streaming-Baselines erheblich und zeigen eine gleichwertige Leistung wie die Pool-basierten Baselines.

Contents

1. Introduction

Robots engaging with humans in real-world settings encounter a wide range of object instances. A key component in robotics systems is their ability to extract semantic information regarding these objects and use this information to enable further downstream tasks. The field of object instance recognition, where significant advancements were made in terms of accuracy and robustness, partially addresses the issue of cost and sample efficiency for acquiring the information required for the successful learning of algorithms.

A significant aspect of training machine learning models is obtaining labeled datasets, which can be laborious and costly. Access to diverse and informative data beyond the available bench-marking datasets becomes particularly important in robotics applications. A bulk of current machine learning approaches demand prior information in the form of annotated data for each object class or instance under consideration. To tackle this, we plan to focus on scenarios where no such supervisory signals are readily available, i.e., the classification of unknown objects. Such a learning approach requires human annotators in the loop; hence, sample efficiency of the overall framework can be of practical importance. One such well-known framework for learning machine learning models is *active learning*, where a model learns from a small amount of data by selecting the data it wants to learn. This can be done using stream-based or pool-based active learning.

When implementing Active Learning (AL), it is essential to identify the type of unlabeled data source. Unlabeled data may be available through two means: either all data is obtained beforehand and is available as a pool, or the data is accessible continuously, but not simultaneously, and is therefore accessible as a stream. In the context of robotics, especially mobile robotics, where robots learn continuously by interacting with new objects or environments, the study of stream-based AL methods is more relevant.

AL has demonstrated significant potential in lowering labeling costs [47]. Nonethe-

less, with the emergence of high-dimensional data and deep Neural Network (NN), efficiently training such models with a limited number of significant samples continues to be a key challenge. The transition from classical AL to Active Deep Learning (ADL) promises to address the challenges of this new era of machine learning [29]. The current emphasis of ADL has shifted from model-driven to data-driven approaches.

Model-driven ADL techniques depend on handcrafted features from unlabeled data and the model's assessment of that data to choose the most informative sample for training. Model-driven approaches encompass a range of strategies varying from uncertainty-based [21], [47] to diversity-based [46]. While these methods hold clear statistical and physical significance, a serious disadvantage is their dependence on human experience and prior assumptions. In comparison, data-driven methods focus on selectors with deep architectures, where their features are automatically learned based on the design and structure of the data-driven mechanism. Although recent studies in data-driven ADL have utilized meta-learning [42], there has also been a heightened emphasis on Deep Reinforcement Learning [7], [13], [28], [30], [38], [50]. While data-driven ADL can overcome their reliance on prior knowledge and human experience, they present a non-trivial design challenge with the additional drawback of unclear physical and statistical meaning.

This thesis aims to develop a method for actively learning an image classifier using reinforcement learning from a stream-based unlabeled image data source. We assume that there is access to labeled data that is similar to or close to the distribution of the unlabeled data. Our proposed AL methodology relies on the available labeled data to learn an RL agent. Once trained, the agent can then be used on an unlabeled image stream of a different distribution to ascertain whether to request its labels from an oracle or not.

Here are the key contributions of this thesis:

1. Formulation of the Markov decision process for AL;
2. Proposal of two novel state representations for the RL agent to learn an effective AL selection policy for image classification;
3. Propose an RL-based data-driven framework for stream-based AL that learns a selection policy on a highly labeled dataset and applies the learned policy on to a low labeled dataset for image classification;

4. Analysis of each of the new methods and insights on their performance against baseline AL algorithms and faced challenges;

Outline: This thesis is organized as follows: In Chapter 2, we review the current state-of-the-art in ADL, with a primary focus on data-driven approaches. We then introduce the theoretical background of AL in Chapter 3, where we present the query types in AL and a general framework for stream-based AL. We also dive into the selection strategies relevant to the context of our proposed method. Additionally, we cover the foundations of RL and deep Q-learning relevant to our later method. Lastly, we discuss the foundational vision model for our use as powerful feature extractors. In Chapter 4, we provide a detailed overview of our RL-based AL method. Here, we first introduce the components of our classifier model, followed by the formulation of our AL problem as a Markov decision process. Finally, we introduce our proposed AL framework in detail. Then, the results of our AL framework are discussed in Chapter 5. Finally, in Chapter 6, we conclude our result and present the future scope of this work.

2. Related Work

The primary goal of this thesis is to apply AL for an image classifier intended for downstream robotics applications where the unlabeled information is obtained as a stream. In the context of robotic systems, one of the important problem statements is extracting semantic information from their observed input data. The extracted semantic information can range from object recognition Narr *et al.* [33] and Saran *et al.* [44], object detection Choi *et al.* [9] and Wu *et al.* [59], or semantic segmentation Casanova *et al.* [7]. Out of the numerous facets of semantic information extraction, this thesis is focused on object recognition, a fundamental capability required in many perception systems for mobile robots. Though significant progress has been made in improving the robustness and accuracy of these models, fundamental questions persist about the sampling efficiency and cost associated with acquiring informative datasets for training them. Consequently, the focus here is on exploring and applying an AL framework to address the aforementioned challenges.

2.1. Active Deep Learning

AL considers a supervised learning problem where there is an abundance of unlabelled data and acquiring their labels is expensive. A good overview of the AL framework is given by Settles [47]. With the advent of deep learning, the emphasis has shifted towards active deep learning Liu *et al.* [29] and Ren *et al.* [43], aiming to train a deep NN by selectively labeling a small number of samples. The unlabelled data for AL can be available in two ways: in the first case, we have a fixed *pool* of data, and the AL algorithm can pick good samples to query from this pool, which is usually very large. This is the primary use of AL, and a few noteworthy AL works have been conducted for pool-based object recognition tasks, including Gal *et al.* [15], Kirsch *et al.* [23], and Sener and Savarese [46]. Additionally, unlabelled data can be accessible as a continuous stream, where the data is not all available

at the same time. This thesis focuses on the case of stream-based AL as it aligns better with the limitations of mobile robotics, especially for robots that need to continuously learn semantics. Several works related to this topic include Narr *et al.* [33], Saran *et al.* [45], and Sun and Gong [50].

2.2. Model-Driven to Data-Driven Active Deep Learning - Reinforcement Learning

In Liu *et al.* [29] survey of ADL, two categories are distinguished: model-driven and data-driven methods. The primary difference lies in the strategy of selecting data. Prominent examples of model-driven ADL, like the ones proposed by Gal *et al.* [15] and Kirsch *et al.* [23], aim to actively train a Bayesian neural network (BNN) by selecting unlabelled data points that are likely to be mispredicted by the BNN's different samples. Another model-based method by Sener and Savarese [46] employed diversity sampling as a sample selection strategy. These methods have demonstrated noteworthy performance gains while utilizing significantly fewer samples. However, it should be noted that these methods fall under the category of pool-based ADL. In the stream-based setting, following the work at *Deutsches Zentrum für Luft- und Raumfahrt* (DLR), Narr *et al.* [33] propose an online AL framework utilizing a variant of decision trees for a mobile robotics scenario. However, this method would fall under the category of AL, while our focus is on ADL. Another recent work in the stream-based model-driven ADL, Saran *et al.* [45], explores a sampling strategy that trades off between uncertainty and diversity of queried samples to match a desired query rate.

In recent years, Reinforcement Learning (RL) based ADL methods have gained traction in the application to language [13], [28] and computer vision [7], [30], [38], [50], [58] tasks. Fang *et al.* [13] propose ADL using for classification-based language tasks. Their method involves learning an AL *policy* on a highly labeled language and then transferring the AL policy using cross-lingual word embeddings [22] to learn a predictor on a low labeled language actively. They demonstrate the feasibility of their approach in comparison to model-driven ADL baselines for a stream-based scenario. However, such embedding does not exist for the domain of images, which would allow a policy learned on a particular image dataset to be applied to datasets

of different domains and distributions.

Casanova *et al.* [7] propose a pool-based RL technique; however, their method deals with pixel-based instance segmentation. Sun and Gong [50] formulate an RL-based ADL approach for object recognition. Notably, [50] defined the reward for their RL agent using entropy loss based on image embeddings from the penultimate layer of the classifier with respect to the class centers of previously labeled sets. However, fine-tuning the classifier every time a new image instance is labeled leads to a non-stationary embedding space, which can result in unstable RL rewards. We take inspiration from this line of work and instead use pre-trained foundational vision models for feature extraction to overcome the challenge of non-stationary embedding space.

3. Background

This section provides a thorough overview of ADL for object recognition, focusing on the theoretical framework. Related works in AL and ADL for object recognition in robotics are also explored in detail. In subsequent sections, we delve into reinforcement learning, an important data-driven ADL technique, followed by foundational vision models used as feature extractors for vision tasks.

3.1. Active Learning

In the machine learning community, AL is a sub-field that fundamentally studies the question of how to obtain as much performance gains as possible by labeling as few select samples as possible. A fundamental ingredient in achieving a machine-learning model with high accuracy and robustness is access to a labeled dataset that is representative of the task at hand. Obtaining a large number of unlabeled datasets is relatively simple in the real world. However, manual labeling of datasets incurs a significant cost. The aim of AL is to learn a machine learning model by selecting the most informative samples from the unlabeled dataset and handing it over to the oracle for labeling [47]. This allows for minimizing the cost of labeling as much as possible while still maintaining the performance of the model being actively learned. While AL is widely known in machine learning, it is often referred to as optimal experimental design [14] in the field of statistics. Figure 3.1 illustrates the AL procedure. An oracle can be a human being or a machine with the task of labeling the selected instances, and the learner, also known as the selector, will, in the following, be considered as the framework or algorithm that aims to find the most informative instances for the learning process. The model being actively learned is called the predictor.

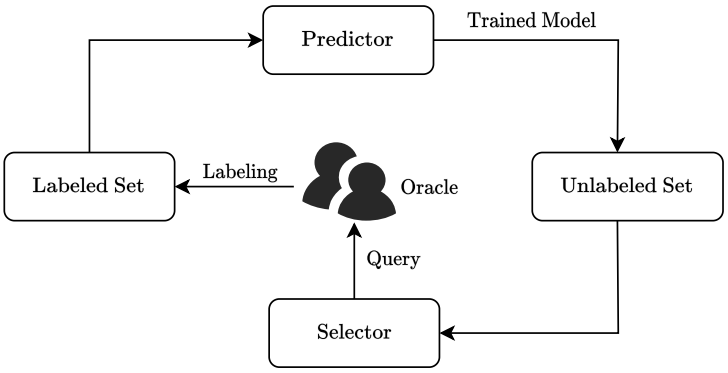


Figure 3.1.: Active learning trains the predictor with initial training samples and uses its selector to select a few of unlabeled samples, labels them, adds them into the training dataset, and then re-trains the predictor.

3.1.1. Query Types in Active Learning

Based on Settles [47], there are three query types in AL based on the source of unlabeled data. The considered query types are: membership query synthesis, pool-based sampling, and stream-based selective sampling. All three scenarios assume the availability of an oracle to query the ground truth of a given unlabeled data.

Membership Query Synthesis (MQS): MQS dates back to one of the earliest AL methods [1]. In MQS, the learner generates new data points based on the unlabeled data and the current status of the actively learned model. However, annotating the generated samples for a human becomes challenging. Therefore, we do not further consider this scenario for our given task.

Pool-based Active Learning: Pool-based AL assumes access to a large collection of unlabeled data prior to the beginning of AL. The first pool-based AL method was proposed by Lewis and Gale [27]. They propose a selector to rank all unlabeled data based on an informative measure and then selectively acquire the highest-rated N samples to be annotated by the oracle.

Stream-based Active Learning: In this thesis, we focus on the streaming setting, which considers the case where the complete unlabeled data is not simultaneously available at the time of AL. In this AL scenario, it is desirable to make a decision about whether to label a given data point as soon as it is encountered rather than

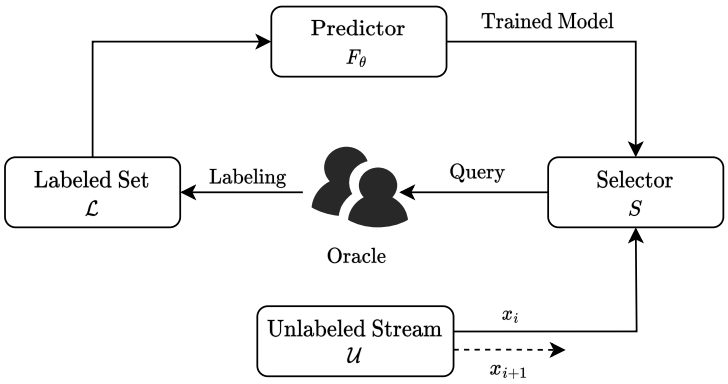


Figure 3.2.: Active learning on a stream of unlabeled data. For each incoming instance, the selector decides whether it should query its label or not. If labeled, the instance is added to the labeled set, and the predictor is updated for the next round.

after the stream has ended. The decision of whether an incoming unlabeled instance has to be labeled by the oracle can be made with the help of an informative measure. Considering the streaming condition, Argamon and Dagan [2] proposes to use a threshold of an informative measure as a selector, where the incoming instance with the informative measure greater than the defined threshold is queried by the oracle.

3.1.2. Active Learning Framework

Considering the relevance of the streaming scenario for our work, here we formally introduce the elements of a general stream-based AL framework. The pseudo-code for this scenario is given in Algorithm 1. Nonetheless, it can be adapted to handle a pool-based AL scenario with slight modifications, as shown in Appendix A.1. Namely, as shown in the described framework, a stream of unlabeled instances $\mathcal{U}$ is considered. Then, for each incoming instance x from the stream, the selection criterion S, based on an informative measure, decides whether to query the label for that instance from an oracle or not. The labeled instance (x, y) is added to the labeled set $\mathcal{L}$, on which the predictor F_θ with a set of parameters θ is either *fine-tuned* or retrained from scratch. The iterative process continues until either the predefined budget B is exhausted or the unlabeled stream $\mathcal{U}$ has become empty. This interaction is additionally illustrated in Figure 3.2.

Algorithm 1 Active Learning Algorithm For a General Stream-based Scenario

Require: initial labeled dataset $\mathcal{L}$; unlabeled stream $\mathcal{U}$; query budget B; untrained or pre-trained predictor $F_\theta(\cdot)$; active learning criteria $S(\cdot)$
Ensure: trained predictor network F_θ, updated labeled set $\mathcal{L}$

procedure ActiveLearning($\mathcal{L}$, $\mathcal{U}$, B, F_θ, S)
    queried count $C \leftarrow 0$
    train F_θ on $\mathcal{L}$
    for each new instance x from the stream $\mathcal{U}$ **do**
        if active learning criteria $S(x)$ **then**
            query label y for x
            $\mathcal{L} \leftarrow \mathcal{L} \cup (x, y)$
            retrain or fine-tune F_θ on $\mathcal{L}$
            $C \leftarrow C + 1$
        end if
        if $C = B$ **then**
            break ▷ End learning when budget is reached
        end if
    end for
    return F_θ and $\mathcal{L}$ ▷ The trained model and the labeled set are returned
end procedure

While a few AL selection criteria have been briefly introduced in Section 3.1.1, a more detailed discussion is given in Section 3.1.4.

3.1.3. Active Deep Learning - Model-Driven to Data-Driven

With the advent of deep learning, the emphasis has shifted towards ADL, where the goal is to train a deep neural network by selectively labeling a small number of unlabeled data. ADL follows the same iterative interaction of the selector S and the predictor F_θ as in the case of AL; however, in the case of ADL, the predictor is invariably a deep NN model [29]. On the other hand, the selector does not have to be a NN model; rather, it primarily depends on the basis of the selection strategy employed.

Three key factors establish the fundamental design of the selector: the intrinsic structure of the data as used in Sener and Savarese [46], the data's impact on the predictor as demonstrated in Sourati *et al.* [49], or a metric-based measure, such as the predictor's uncertainty on the data as shown in Joshi *et al.* [21] and Gal *et*

al. [15]. The Core-set approach from Sener and Savarese [46] queries data points such that when added to the training set, the distance between a data point in the unlabeled pool and its closest data point in the training set will be minimized. Sourati *et al.* [49] use Fisher Information to identify the unlabeled samples that create the highest impact on the model parameters of the predictor. Joshi *et al.* [21] propose a strategy called Best-versus-Second Best (BvSB) where samples chosen for labeling have the lowest difference between the predicted probabilities of the top two predictions. A hybrid selector that uses a combination of these factors can also be designed. Gao *et al.* [16] propose an effective selection metric that considers sample consistency by implicitly balancing sample uncertainty and diversity during the sample selection phase. ADL methods that define selection criteria as explicit models composed of handcrafted features or metrics fall under the class of *model-driven* ADL [29]. In this thesis, a model-driven selector is represented by S.

At the other end of the spectrum are *data-driven* ADL approaches, which employ NN as selector models and rely on the annotated data to learn the selector (or can be called a selection policy). They learn to identify the data points that are the most beneficial for training a predictor, given the current state of the predictor and the unlabeled instance. Since the focus of data-driven is not on the explicit use of hand-designed features as in the case of model-driven ADL approaches, the focus in this class of methods is on defining a structure and a mechanism for learning a selector from data. Liu *et al.* [28] has classified research in this type of ADL into four categories: ADL with meta-learning [42], ADL with reinforcement learning [13], ADL with uncertainty [60] and lastly ADL with data augmentation [36]. Although we will address the function of reinforcement learning in ADL in the following Section 3.1.4, for a comprehensive overview of the other data-driven ADL methods, one can refer to the surveys by Liu *et al.* [28], Budd *et al.* [4] and Ren *et al.* [43]. These sources offer comprehensive coverage of the different problem formulations for the other specific types of data-driven ADL methods.

3.1.4. Selector Strategies

In AL, the role of the selector S is to select an instance x from a set of unlabeled instances $\mathcal{U}$ that is most beneficial for improving the performance of the deep predictor F_θ. This is done by designing S that evaluates the informativeness of

unlabeled instances. Based on this measure, a decision is then made to request a label from an oracle. The AL literature proposes numerous selection strategies, and in the subsequent subsection, we provide a review of considered model-driven AL baselines for the benchmarking of our data-driven ADL method. The baselines include random sampling, entropy, and BvSB for uncertainty-based sampling, and the core-set method for diversity/representative-based sampling. We use the notation x_S^* to refer to the most informative instance according to the selection criteria S for model-driven ADL.

Random Sampling

A naive random sampling strategy uniformly chooses instances to be labeled from a given stream or pool of unlabeled data. This form of a selector is often used as a baseline to compare the performance of different strategies [45], [46].

Uncertainty Sampling

Another class of AL baselines uses uncertainty measures as the active sampling strategy. In uncertainty-based sampling strategies, the selector chooses samples with the highest level of uncertainty. This uncertainty measure is based on the output of the predictor that is generated while considering an unlabeled sample x. Different approaches exist for measuring the uncertainty of a sample, and one of them involves utilizing an NN to predict its class probabilities. To formalize the procedure mathematically, the predictor's output for a given input x is defined as $F_\theta(x) := \{\hat{y}_c \mid \forall\, c \in \mathcal{C}\}$, where $\mathcal{C} = \{1, \dots C\}$ and $\hat{y}_c$ represents the probability of x belonging to class c, i.e., $\hat{y}_c = p(y = c \mid x)$.

BvSB, also known as minimum margin uncertainty [21] and entropy uncertainty [47] are two well-known AL sampling strategies used in conjecture with deep architectures. The minimum margin uncertainty is defined as:

$$\begin{gathered} x_S^* = \arg\max_{x \in \mathcal{U}} \; (\hat{y}_{c_1} - \hat{y}_{c_2}) \\ \text{with } c_1 = \arg\max_{c \in \mathcal{C}} \; p(y = c \mid x), \text{ and } c_2 = \arg\max_{c \in \mathcal{C} \setminus c_1} \; p(y = c \mid x) \end{gathered} \tag{3.1}$$

and the entropy uncertainty is defined as:

$$x_S^* = \arg\max_{x \in \mathcal{U}} - \sum_{c \in \mathcal{C}} p(y = c \mid x) \log(p(y = c \mid x)) \tag{3.2}$$

BvSB measures the class probabilities between the two highest class probabilities and chooses the instances that have the smallest margin for labeling. Cao *et al.* [5] apply BvSB uncertainty sampling for hyperspectral image classification using Convolutional Neural Networks (CNNs). On the other hand, entropy, as a measure in information theory, quantifies the information required to encode a distribution. A high entropy suggests uncertainty in the predictor's prediction. In an AL strategy based on entropy, the selector chooses unlabeled instances with the highest level of information entropy for labeling.

Core-Set

To benchmark the method proposed in this thesis against representative-based AL algorithms, we consider the Core-Set approach as a baseline. Core-Set is a model-driven ADL method designed to select instances based on the intrinsic characteristic of the unlabeled data. Due to the way this selection criterion is defined, it is more often associated with a pool-based AL scenario. Sener and Savarese [46] view the Core-Set as a representative-based method, where the goal is to find a representative subset of the unlabeled data pool such that the performance of the predictor learned on that subset is as close as possible to a predictor that could have been learned on the entire dataset. This method relies on solving a k-center problem on a set of points represented by the embeddings obtained from the penultimate layer of the predictor for a given unlabeled set $\mathcal{U}$. The embeddings derived from the penultimate layer represent the extracted features from the high-dimensional unlabeled instances.

3.2. Reinforcement Learning

In recent years, RL-based AL has gained traction as a suitable data-driven ADL method. To provide a comprehensive understanding of the RL methodology used in our AL pipeline, we begin with an overview of key RL concepts.

In RL, an agent learns to perform a task by interacting with its environment. The

goal in RL is to train an agent that maximizes the expected sum of future rewards discounted over time. In the context of RL in AL, a trained agent must be capable of assessing whether it would be informative to request the label of an instance based on the present state of AL. This section covers the relevant fundamentals of RL, laying the groundwork for our proposed data-driven AL method.

3.2.1. Basic Concepts for Reinforcement Learning

A wide range of artificial intelligence tasks can be formalized as sequential decision-making processes. The decision-making entity is referred to as the *agent*, and everything outside of the agent is its *environment*. Reinforcement learning is based on the idea that the agent learns its *policy* by trial and error through interaction with its environment. As illustrated in Figure 3.3, at each time-step i, the agent receives observation $s_i \in \mathcal{S}$ and executes an action $a_i \in \mathcal{A}$ according to its policy. Following the action a_i in the state s_i, the environment provides a feedback signal in the form of a reward $r(s_i, a_i) \in R$. This series of actions, observations, and rewards defines the agent's *experience*.

The goal of reinforcement learning is to learn a policy that improves the agent's future reward, given its past experience. Two primary approaches to achieving such a policy include policy search and value iteration [52]. A policy, $\pi(s_i, a_i)$, functions as a mapping from states to probabilities of selecting each action, and policy search involve finding the *optimal policy* π_*, which maximizes the sum of expected future rewards.

On the other hand, the value, $V(s_i)$, is the expected sum of future rewards for the state s_i when following a policy π. It can also be seen as an estimate of how good it is for the agent to be in a given state s_i. The objective of value iteration is to find the *optimal value function* V_* that maximizes the sum of future expected values and from which the optimal policy can be derived.

$$\begin{aligned} V(s) &= \mathbb{E}_\pi \left[G_t \mid s_i = s \right], \text{ for all } s \in \mathcal{S} \\ V_* &= \arg\max_\pi V(s) \end{aligned} \tag{3.3}$$

$G_t = \sum_{t=i}^{T} \gamma^{t-i} r_t$ is the sum of discounted future reward, and $\gamma \in [0, 1]$ is the discount factor that is used for discounting the value of the future rewards. Here,

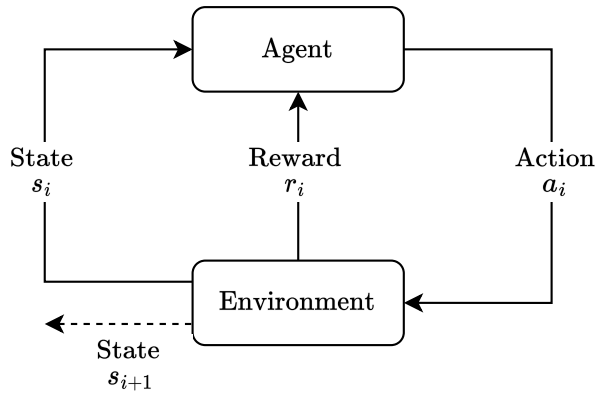


Figure 3.3.: A general idea of reinforcement learning: The agent interacts with the environment to learn from experiences. At each time-step i, the agent is in a certain state $s_i \in \mathcal{S}$ and takes action $a_i \in \mathcal{A}$. As a result, it receives a reward r_i and switches to a new state s_{i+1}.

the expectation is taken over all possible actions in state s.

Similarly, we also have the value of taking an action a_i in state s_i denoted by $Q(s_i, a_i)$. Optimal policies π_* share the same *optimal action-value function*, which is denoted by Q_*.

$$\begin{gathered} Q(s,a) = \mathbb{E}_\pi \left[G_t \mid s_i = s, a_i = a \right], \text{ for all } s \in \mathcal{S} \text{ and } a \in \mathcal{A} \\ \pi_* = \arg\max_a Q_*(s,a) \end{gathered} \tag{3.4}$$

Here, the expectation is taken over all transitions involving state s and action a.

To apply RL to a given task, it is necessary to formulate the task as a Markov Decisions Processes (MDP). However, the task must satisfy the *Markovian property* to be formulated as an MDP. The Markovian property assumes independence of past and future states and actions, implying that the system has no memory of previous states and actions. That is, the probability of transitioning to state s_{i+1} after taking action a_i depends only on the current state s_i, not on past states and actions. The dynamics of an MDP are characterized by the state transition probability $p(s_{i+1} \mid s_i, a_i)$, which denotes the likelihood of transitioning to state s_{i+1} when the agent performs action a_i in state s_i. *Model-based* RL methods, like dynamic programming, presuppose knowledge of the MDP's dynamics in the training of the

RL agent [55]. However, in scenarios where the dynamics of the environment are complex or unknown, *model-free* RL methods become particularly valuable. Model-free methods, such as Monte-Carlo evaluation [48] or temporal-difference learning [51], directly learn from experience without assuming a prior knowledge of the environment's dynamics. This makes them well-suited for real-world applications where obtaining an accurate model of the environment is challenging or impractical. For a deeper understanding of the ideas behind model-based and model-free RL methods, refer Sutton and Barto [52].

3.3. Deep Reinforcement Learning

When dealing with classical RL and a limited, finite state and action space, it is feasible to create approximations of value functions and policies using arrays or tables. This scenario is referred to as the *tabular* case, with corresponding RL methods referred to as tabular methods [52]. However, in many practical cases, the state space can grow rapidly, making it unrealistic to visit all possible states to learn the value of every combination of state-action pairs. Additionally, the memory available for storing such large tables is also a significant constraint. At the same time, there is also a lack of shared knowledge about similar states, which may lead to improved representation and reduced training times. In such cases, the value functions and policies must be approximated using some compact parameterized function representations.

A common approach is to replace these tabular representations with deep NN that act as function approximators. This leads to deep reinforcement learning (deepRL), and in doing so, we obtain the following components of RL: value function $V(s_i; \phi)$ or $Q(s_i, a_i; \phi)$ and the policy $\pi(s_i, a_i; \phi)$. Here, the parameters ϕ are the weights of the NN. Additionally, NN has the ability to approximate non-linear functions and capture relevant features from raw inputs, enabling generalization over unseen states.

In deep RL, there are two paradigms to training an agent: Value-based methods [32] and policy gradient-based methods [53], [57]. In this thesis, we focus on a value-based model-free method, deep Q-Learning [32], which builds on the combination of Q-learning [56] and non-linear function approximation.

3.3.1. Deep Q-Learning

Deep Q-Learning specifically applies NNs to the Q-Learning algorithm. This network is also referred to as the Deep Q-Network (DQN), given a state s_i, outputs the Q values for all possible actions, i.e., $Q(s_i, a_i; \phi)$ for $a_i \in \mathcal{A}$. Here, ϕ are the parameters of the DQN. NNs possess a robust capacity for learning various levels of abstraction from data [18], [25], making them a suitable choice for making generalizations about the Q-function based on these learned abstractions. These abstractions may potentially offer better representations than features designed by humans [32].

Over the past few decades, researchers have explored the combination of Q-learning and nonlinear function approximation. However, this approach encountered challenges, particularly in terms of unstable learning. A notable advancement was made by Mnih *et al.* [32], who introduced two additional mechanisms known as *experienced replay* and *frozen target network* to address the instability issue. The concept of experienced replay involves storing the agent's experiences (s_i, a_i, r_i, s_{i+1}) in a buffer M. During each training step, a batch of experiences is uniformly sampled from M to update the DQN. This strategy effectively mitigates correlations in the data sequences, providing the network with independent data and contributing to more stable learning processes.

The DQN is updated according to the following loss function:

$$L(\phi) = \mathbb{E}_{s_i, a_i, r_i, s_{i+1}}[(y(r_i, s_{i+1}) - Q(s_i, a_i; \phi))^2] \tag{3.5}$$

where $y(r_i, s_{i+1})$ is:

$$y(r_i, s_{i+1}) \coloneqq r_i + \gamma \max_a Q(s_{i+1}, a; \phi^-)$$

However, introducing an experience replay buffer brings in the deadly triad of function approximation, bootstrapping, and off-policy learning as identified in Sutton and Barto [52]. The combination of these three properties could diverge the training process, leading to unbounded value estimates. To avoid such conditions, the frozen target network mechanism is presented in equation (3.5). Two networks with the same structure but different weights are used: ϕ for the Q-network and ϕ^- for the target network. The Q-network is regularly updated via the loss function

in equation (3.5). In contrast, the target network ϕ^- is updated by periodically copying Q-network weights ϕ after a fixed number of training steps. This serves to improve the stability of the Q-value estimation by using a fixed target network to evaluate states during the training step.

Throughout the training process, it is assumed that the agent's optimal strategy is always to select the action with the highest Q-value. However, during the early stages of training, the Q-value estimates are a result of the model's random initialization rather than learned experience. Consequently, relying solely on the model's selected actions may not generate meaningful agent experiences to further improve the Q-network. To address this, DQN employs an ϵ-greedy strategy [32] that embodies the exploration-exploitation trade-off inherent in reinforcement learning. Introducing this ϵ-greedy strategy makes DQN an *off-policy* algorithm. Initially, during the early stages of training, when Q-value estimates are less reliable, the agent emphasizes exploration by selecting a random action with probability ϵ and the action with the highest Q-value with probability $1 - \epsilon$. Such a strategy allows the agent to explore the state and action space and potentially gain better rewards in the long run. As the training progresses and the Q-network becomes more accurate, the epsilon value is systematically reduced over time. This reduction is controlled by an epsilon decay schedule, with the initial high value encouraging exploration and the gradual decrease shifting the balance toward exploitation. Through this dynamic adjustment, the agent intensifies exploitation as it gains confidence in the Q-value estimates. This balanced strategy enables the DQN agent to adjust to the evolving dynamics of the environment, resulting in a Q-network that is more precise.

3.4. Foundational Vision Models

The primary goal of foundation models is to learn universal representations that work seamlessly for any downstream task. Large language models have displayed remarkable achievements in natural language processing [3], [35], [40]. With this in mind, concerted endeavors have been made to extend this paradigm to computer vision [6], [39]. Foundational Vision Models (FVMs) are typically designed using structures such as CNNs or Transformers. These models have parameters in the order of tens to hundreds of millions, giving them greater representational capacity than smaller models. The emergence of significant pre-trained FVMs, such as the

DINOv2 developed by Oquab *et al.* [34], has demonstrated their remarkable ability to learn robust representations and to provide better generalization capabilities for downstream vision tasks.

We use these FVMs in our work in order to optimize the training and inference process so that the input to the classifier is not a whole image with a huge amount of pixels but a reduced number of relevant features extracted from it. The DINOv2 model is capable of extracting powerful image features and performs well across various tasks. The output of DINOv2 can be directly used as input for any other purpose, for instance, semantic segmentation, instance retrieval, or object recognition. Such models responsible for feature extraction from input data, an image in this case, are referred to as the *backbone*. There are different vision transformer models available, the main one with a billion parameters, while the rest are distilled from it. The input image for the model has to be 224x224 pixels, or a multiple of the patch size, which is 14. The output of DINOv2 returns a class token and patch tokens, and the dimension depends on the model, ranging from 384 for the smallest to 1536 for the biggest one. These extracted features can then be passed to a *base network*, which is selected based on the specific image task requirements. In DINOv2 documentation [34], it is stated that the output of this FVM can just be used with downstream base networks such as linear layers, yielding competitive results. This adaptability and performance make DINOv2 and its foundational vision models indispensable in advancing various image-processing applications.

4. Method

The objective of this thesis is to develop an algorithm for AL that can learn semantic information, specifically for image classification. This chapter discusses the approach taken to actively learn the image classifier using a data-driven ADL approach for a stream-based scenario. Traditional methods often rely on hand-crafted selectors and hence follow a model-driven ADL approach. In contrast, we propose a data-driven approach to ADL using RL. We then explain how the learned policy of the RL agent can be applied to other datasets for actively learning a classifier.

The following sections present a detailed explanation of the research methodology. This includes the usage of foundational models, MDP formulations for AL, and the proposed algorithm for RL-based ADL.

4.1. Classifier: Backbone and Base Network

CNNs are the most commonly used deep learning method for processing visual data. Consequently, given that our AL task deals with image classification, several options are available for the classifier model. Among the most popular networks in Computer Vision (CV) for image classification are the ResNets [19] or Residual Networks. The ResNet architecture was designed to address the degradation problem encountered when using standard CNNs. This degradation occurs as the model accuracy diminishes with the increase in complexity and depth of the model. They mitigate this issue by using identity-mapping layers, resulting in an NN that is easier to optimize while also gaining accuracy with increased depth. ResNets are designed using the same core building blocks called the *residual blocks* and are available in variants depths, such as ResNet-34, ResNet-50, or ResNet-101. In Pytorch [37], these models are available and pre-trained with IMAGENET [12], an image database organized according to the WordNet – an extensive lexical database of English – hierarchy, in which multiple images depict each node. Therefore, we can use these pre-trained

network models directly and only train or fine-tune their last layer. With this, we take advantage of the ResNet architecture and the pre-training, allowing the network to extract meaningful features and only modify the last part to fit other datasets. In this case, ResNet would function both as a base network, since we would be training at the final layers given a new dataset, and also as a backbone, in the sense that the frozen layers of the network before the final layers would serve as a powerful feature extractor.

On the other hand, the potential effectiveness of FVMs for extracting features from unseen data, as described in Section 3.4, motivates our use of DINOv2 as the backbone for the classifier. Hence, in Section 5.1, we compare the performance of both backbones, ResNets, and DINOv2, to see if our decision to use a state-of-the-art tool, DINOv2, is better than a former consolidated option, such as ResNet. In addition, the DINOv2 literature [34] states that the class token generated by the model for a given image data can be directly used for image classification using a single linear layer. The class token serves as the representation of the entire input image and can also referred to as the image feature.

Both the base classifier and the backbone can be modified for any other combination, depending on the application and the nature of the dataset. For the purpose of this work, we chose DINOv2 as the backbone. For the base classifier, we use a simple two-layer NN with a single hidden layer, ReLU activation, defined as $\text{ReLU}(x) := \max(0, x)$, and a final linear layer with nodes equivalent to the image classes in the dataset. The output of this base classifier is *logits* from which the class probabilities are computed using a softmax function defined as

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j \in \mathcal{C}} \exp(x_j)} \tag{4.1}$$

where, x_i are the predicted logits for each class. The input to the base network is the output of the backbone, so it will depend on the DINOv2 model that is used. To validate the choice made in this thesis, we will test different configurations for the classifier, including a base network for the classifier with only one layer, in Section 5.1 and obtain their results on the performance so that our initial idea is supported with results. However, the intuition behind using DINOv2 is that it is a versatile and powerful feature extractor; therefore, we expect a simple two-layer classification model as the base network to be sufficient.

4.2. Active Learning as a Markov Decision Process

The goal of AL is to iteratively train a classifier for image classification by strategically selecting a subset of the most informative samples from a stream of unlabeled data and obtaining labels for them. The intended outcome of this class of methods is to find instances on which the classifier is most likely to make errors. By labeling and adding these instances to the training set, the classifier becomes more resilient to such errors when processing new data. As described for a stream-based scenario in Section 3.1.2, this iterative process continues until the annotation budget is exhausted, which also acts as the termination criterion of this learning framework.

The iterative steps involved in AL can be conceptualized as a decision-making process, where at each step, the selector decides whether to label an instance based on the current state of the classifier and the unlabeled instance. This decision process is well suited to be modeled as an MDP, which facilitates the application of deepRL as a data-driven method for ADL.

In the context of AL as an MDP, the interaction between the RL agent and the environment is illustrated in Figure 4.1. Here, the RL agent serves as both the selector and the policy, denoted as π_ϕ with parameters ϕ. Conversely, the environment comprises the classifier and the ongoing unlabeled data stream $\mathcal{U}$. At each time-step i, the agent (selector) observes the current state s_i from the environment, which includes the unlabeled image instance x_i and the learned classifier (predictor) F_θ. The agent then takes an action a_i with the aim of maximizing the reward r_i. The action a_i decides whether the unlabeled sample needs to be sent to an oracle for the label. If the image is labeled, it is added to the existing pool of labeled data $\mathcal{L}$, and the classifier F_θ is updated to include this new training point. This iterative process continues until the data stream is exhausted or a predetermined annotation budget B is reached. At each step, a reward r_i is computed, linked to the action taken by the agent and the empirical performance of the classifier. This MDP framework can be succinctly represented as a tuple $(\mathcal{S}, \mathcal{A}, p(s_{i+1} \mid s_i, a_i), R)$, where $\mathcal{S}$ is the space of all possible states, $\mathcal{A}$ is the set of actions, R is the reward function and $p(s_{i+1} \mid s_i, a_i)$ is the transition function. In the following subsections, we explicitly define our proposed state, action, and reward formulation, which is well suited for our AL task of learning an image classifier in a streaming scenario.

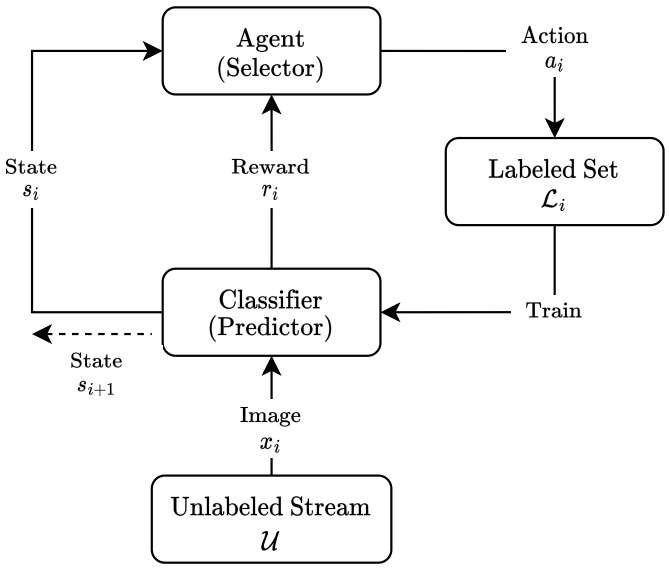


Figure 4.1.: Formulation of AL as MDP for RL. In our setting, the RL agent acting as the selector, makes decisions on whether to label an image x_i from the unlabeled stream $\mathcal{U}$. If the label is acquired from an oracle, the labeled set $\mathcal{L}$ is updated with this image and its acquired label. Subsequently, the classifier is retrained on $\mathcal{L}$ before proceeding to the next step in the RL episode.

4.2.1. State

The RL agent requires relevant information about the state of the environment in order to perform the AL task successfully. We use the following data extracted from the candidate image x_i considered for annotation and the labeled set of images $\mathcal{L}$ collected during each step of the RL episode using the classifier model.

Image Features (x_i^f)**:** The RL agent relies on information about the candidate image, and to capture this, we employ the DINOv2 model to generate a meaningful image feature representation. Firstly, using a pre-trained feature extractor optimizes the training and inference process since the input to the base network of the classifier is not a whole image with a huge number of pixels but a reduced representation of the image features extracted from it. Second, this strategy allows us to leverage the powerful and informative features from images without needing to train a separate

backbone, as we qualitatively demonstrated in Section 5.1. Accordingly, for a given image x_i and DINOv2 as the backbone, we refer to the extracted feature vector as $x_i^f \in \mathbb{R}^{384}$.

Classifier's Prediction Probabilities: The other key piece of useful information for the RL agent is the prediction scores of the classifier. Given a classifier F_θ and an image instance x_i, we define the predicted class probabilities as $F_\theta(x_i) := \{\hat{y}_{i,c} \mid \hat{y}_{i,c} = p\left(y_i = c \mid x_i\right), \forall\, c \in \mathcal{C}\}$. These predicted class probabilities from the classifier can be further processed to obtain uncertainty measures, such as the entropy and the margin score of the classifier model for the given unlabeled image instance. Using this information as a part of the RL state would allow the agent to learn to select images in the stream for annotation where the classifier model is the most uncertain.

The above information is used to construct a state representation that is useful for learning an RL agent for AL. Therefore, in this thesis, we investigate two different representations, which are presented in the following subsections. We also summarize them in Table 4.1

State Representation 1

We define the State Representation 1 (SR1) for the RL agent as a composition of the following

- Cosine similarity, which measures the similarity between the feature vector x_i^f of the unlabeled instance x_i and the corresponding class centers $\mathcal{M} = \{m_c \mid \forall\, c \in \mathcal{C}\}$, in an inner product space. This similarity is computed using the cosine of the angle between two vectors, which indicates whether they point in approximately the same direction. We define the cosine similarity s^{cs} of the feature vector x_i^f with respect to the class centers m_c for the agent's state as

$$\begin{aligned} s^{cs} &= \Big(\mathrm{sim}(x_i^f, m_1), \ldots, \mathrm{sim}(x_i^f, m_c), \ldots, \mathrm{sim}(x_i^f, m_C)\Big) \\ \text{where, } \mathrm{sim}(x_i^f, m_c) &= \frac{x_i^f \cdot m_c}{\|x_i^f\|\,\|m_c\|} \text{ and, } s^{cs} \in \mathbb{R}^C \end{aligned} \tag{4.2}$$

  The class center m_c is computed using the image features extracted from the

labeled set $\mathcal{L}$ using the backbone network. For a given class c, we define m_c as

$$m_c = \frac{1}{n_c} \sum_{(x_k, y_k) \in \mathcal{L}} x_k^f \cdot \mathbb{1}\{y_k = c\} \tag{4.3}$$

where, n_c is the number of the samples belonging to class c and x_k^f is the feature extracted from the image x_k.

- Entropy of the classifier F_θ given an unlabeled image. The entropy $s^e \in \mathbb{R}$ measures the classifier uncertainty in its prediction and is defined as

$$s^e = -\sum_{c \in \mathcal{C}} \hat{y}_{i,c} \cdot \log(\hat{y}_{i,c}) \tag{4.4}$$

- Margin score, another uncertainty measure, computes the difference between the highest two predicted class probabilities by the classifier F_θ. We denote this by $s^m \in \mathbb{R}$, and it is defined as

$$\begin{gathered} s^m = (\, \hat{y}_{i,c_1} - \, \hat{y}_{i,c_2}) \\ \text{with } c_1 = \arg\max_{c \in \mathcal{C}} \, \hat{y}_{i,c} \,, \text{ and } c_2 = \arg\max_{c \in \mathcal{C} \setminus c_1} \, \hat{y}_{i,c} \end{gathered} \tag{4.5}$$

State Representation 2

In comparison to the SR1 for the agent, we also formulate a relatively straightforward state representation 2 (SR2), which is composed of the following information regarding our AL environment.

- Classifier's Prediction Probabilities $F_\theta(x_i)$. We denote this component of SR2 as $s^{cp} \in \mathbb{R}^C$, where

$$s^{cp} = (\hat{y}_{i,1}, \ldots, \hat{y}_{i,c}, \ldots, \hat{y}_{i,C}) \tag{4.6}$$

- Entropy of the classifier s^e computed according to equation (4.4)

- Margin score s^m computed according to equation (4.5)

Components of State Representation s_i given image x_i		
State Representation 1 (SR1) $s_i = (s_i^{cs}, s_i^e, s_i^m)$	s_i^{cs}:	cosine similarity
	s_i^e:	entropy score
	s_i^m:	margin score
State Representation 2 (SR2) $s_i = (s_i^{cp}, s_i^e, s_i^m)$	s_i^{cp}:	predicted probabilities
	s_i^e:	entropy score
	s_i^m:	margin score

Table 4.1.: Summary of the state representations of the AL MDP for image classification

4.2.2. Action

The action represents whether the oracle must annotate the current observed unlabeled image x_i. We use a binary action space $\mathcal{A} = \{0, 1\}$, where the action $a_i = 1$ signifies that the RL agent chooses to annotate x_i from the oracle, and $a_i = 0$ signifies otherwise. If an image is annotated, the newly labeled image is added to the existing set of labeled images $\mathcal{L}$, and the parameters θ of the classifier F_θ are updated by retraining.

4.2.3. Reward

In RL, the reward function is critical in successful learning. It also serves as an indicator of the quality or effectiveness of the action selected by the agent. For this, we assign a reward to the agent using a held-out labeled set. At each time-step i, we define the reward r_i as the change in the classifier's performance on the held-out dataset $\mathcal{H}$, i.e.,

$$r_i = \text{acc}\left(F_{\theta_i}\left(\mathcal{H}\right)\right) - \text{acc}(F_{\theta_{i-1}}(\mathcal{H})) \tag{4.7}$$

where acc denotes the ratio between the number of correct predictions by the classifier and the total number of predicted samples, and F_{θ_i} is the trained classifier after action a_i has taken place. The reward r_i can be negative or positive, indicating either a detrimental or a beneficial effect on the performance.

4.2.4. Budget

We define a fixed budget B for the total number of instances to be annotated. The terminal state is reached when no further data remains, or the annotation budget is exhausted, which concludes the AL run. A single run is referred to as an episode. The budget is a predetermined number that is decided before learning starts based on the time complexity of training the RL agent. Once the budget is exhausted or the unlabeled stream is empty, the final classifier is trained on the resulting labeled set of images.

4.3. Active Learning Using Reinforcement Learning

This section lays out the details of our ADL pipeline using deepRL. We break the complete pipeline into two steps. The first subsection explains the training of the RL agent using deep Q-learning to learn a meaningful policy for AL, followed by the next subsection, where we describe the classifier's training by actively selecting samples using the policy learned by the RL agent.

4.3.1. Agent Training

The goal is to train the RL agent so the policy it learns can choose informative images from a data stream to learn a classifier further. However, training an RL agent remains a data-intensive task, so we rely on the availability of a labeled image dataset to train the agent. This labeled dataset can be referred to as the *source* dataset. For the deepRL approach, we use the deep Q-learning described in Section 3.3.1 to train our RL agent, which we will refer to interchangeably as the DQN.

Training the DQN is an episodic task, and the agent's training process consists of two steps executed iteratively at every step of each episode: training of the classifier F_θ and training of the DQN $Q(s_i, a_i; \phi)$. The DQN takes as input the state representation SR1 (s^{cs}, s^e, s^m) or SR2 (s^{cp}, s^e, s^m) defined in Section 4.2.1, and outputs two scalar values corresponding to the values $Q(s_i, a_i; \phi)$ for $a \in \{0, 1\}$. The parameters in the DQN are learned by minimizing a loss function, i.e., equation (3.5) w.r.t to its network's weights ϕ, using a stochastic gradient optimizer and a batch of randomly selected (s_i, a_i, r_i, s_{i+1}) transitions from the experience replay buffer M.

Algorithm 2 Train DQN Agent for AL (Learn AL selection policy)

Require: labeled image data $\mathcal{D}$ (source dataset); query budget B
Ensure: learned DQN policy π_ϕ

procedure TrainDQNPolicy($\mathcal{D}$, B)
  $\mathcal{U}$, $\mathcal{H} \leftarrow$ RandomSplit($\mathcal{D}$) ▷ Create unlabeled stream $\mathcal{U}$ and reward set $\mathcal{H}$
  for *episode* : $1, 2, \ldots, N$ **do**
    queried count $C \leftarrow 0$
    $\mathcal{L} \leftarrow \emptyset$ and shuffle $\mathcal{U}$
    initialize $\mathcal{L}$ from $\mathcal{U}$ and train classifier F_θ
    for $i \in \{0, 1, 2, \ldots, |\mathcal{U}|\}$ **do**
      construct state s_i using x_i and F_θ
      with probability ϵ choose random action a_i
      otherwise select $a_i \leftarrow \arg\max Q(s_i, a_i; \phi)$
      if $a_i = 1$ **then**
        obtain annotation y_i
        $\mathcal{L} \leftarrow \mathcal{L} \cup (x_i, y_i)$
        $C \leftarrow C + 1$
        update classifier F_θ based on $\mathcal{L}$
      end if
      receive a reward r_i using reward set $\mathcal{H}$
      if $C = B$ **then**
        store $(s_i, a_i, r_i, \text{Terminate})$ in replay buffer M
        break ▷ End episode when budget is exhausted
      end if
      construct new state s_{i+1}
      store transition (s_i, a_i, r_i, s_{i+1}) in M
      sample random mini-batch of transitions $\{(s_i, a_i, r_i, s_{i+1})\}$ from M, and perform gradient descent step on $L(\phi)$ (equation (3.5))
    end for
  end for
  return the latest policy π_ϕ ▷ The trained DQN is returned
end procedure

The algorithm for training the DQN is summarized in Algorithm 2. Given a source image dataset $\mathcal{D}$, we simulate a streaming scenario to train the DQN agent over multiple episodes. At the start of every episode, we shuffle the order of the data and hide the known labels, which are revealed when requested by the DQN agent. Additionally, we randomly select a few images from the stream for annotation at the start of each episode. Once annotated, these images are added to the labeled set $\mathcal{L}$

and subsequently used to initialize the classifier's parameters. Therefore, it is crucial to ensure that the initial labeled set contains samples from each class. Between each episode, the classifier is reset and reinitialized on the randomly annotated data at the beginning of the episode, with the main changes being the different (random) data ordering and the evolving DQN policy function.

4.3.2. Classifier Training

With the extensive use of the training dataset to learn the DQN, the DQN policy application only makes sense when employed in a different data setting, e.g., where the image domain is different, to learn a classifier actively. To this end, as described in Section 4.2.1, our state and actions are defined such that they are independent of the specific raw feature representations of a given data stream. The dataset on which the classifier is actively learned using the DQN agent can be referred to as the *target* dataset.

Algorithm 3 gives a concise description of the steps involved in selecting samples for annotation using the trained agent. This algorithm differs from DQN agent training in the following ways: First, Algorithm 3 makes only one pass over the new image stream, and second, the DQN policy is used only for inference, that is, to select the most informative sample from the data stream.

Algorithm 3 AL with DQN Agent (Use learned AL Selection Policy)

Require: unlabeled stream $\mathcal{U}$ (target dataset); query budget B; trained DQN agent policy π_ϕ
Ensure: labeled dataset $\mathcal{L}$ and trained classifier F_θ

procedure ActiveLearning($\mathcal{U}$, B, π_ϕ)
    queried count $C \leftarrow 0$
    $\mathcal{L} \leftarrow \emptyset$
    initialize $\mathcal{L}$ from $\mathcal{U}$ and train classifier F_θ
    for $i \in \{0, 1, 2, \ldots, |\mathcal{U}|\}$ **do**
        construct state s_i using x_i and F_θ
        select action $a_i \leftarrow \arg\max Q(s_i, a_i; \phi)$
        if $a_i = 1$ **then**
            obtain annotation y_i
            $\mathcal{L} \leftarrow \mathcal{L} \cup (x_i, y_i)$
            $C \leftarrow C + 1$
            update classifier F_θ based on $\mathcal{L}$
        end if
        if $C = B$ or $\mathcal{U} = \emptyset$ **then**
            break
        end if
    end for
    return $\mathcal{L}$ and F_θ ▷ The labeled set and trained classifier is returned
end procedure

5. Experiments and Results

In this chapter, our experiments are designed to assess and compare the effectiveness of the RL-based AL method proposed in the preceding chapter. The empirical evaluation focuses on the predictive performance of the actively learned classifier using RL agents and its comparison against select AL baselines. Before delving into the AL experiments, we present the experiments that validate the chosen classifier setup for AL. In addition, we also present an ablation study to evaluate the effectiveness of the various components that comprise the state representation of our AL environment.

All experiments are implemented in PyTorch [37]. Our RL algorithm, DQN, is implemented using Stable-Baselines3 [41], an open-source project containing reliable implementations of RL algorithms in PyTorch. Details on the DQN training parameters can be found in the Appendix A.3. The environment for our AL method is simulated using Gymnasium [54], another open-source library designed for RL algorithm development, which provides a standardized interface for communication between the RL algorithm and the environment.

5.1. Classifier

Datasets: In this experiment, we compare the accuracy of the three configurations for the classifier composed of a backbone and a base network. The base network is trained on varying fractions of the training data across the following academic image datasets: MNIST [26], KMNIST [10], and CIFAR-10 [24]. These datasets are also used in our subsequent AL experiments. MNIST and KMNIST datasets consist of black-and-white images of size 28×28, while the CIFAR-10 dataset consists of RGB images of size $3 \times 32 \times 32$. MNIST represents the ten handwritten digits, KMNIST 10 Japanese letters, and CIFAR-10 everyday objects with 10 different classes. A summary of these datasets is provided in Table 5.1.

Dataset	Domain	Shape	Number of samples	
			Train	Test
MNIST	handwritten digits	$1 \times 28 \times 28$	$60,000$	$10,000$
KMNIST	Japanese letters	$1 \times 28 \times 28$	$60,000$	$10,000$
CIFAR-10	real-world photos	$3 \times 32 \times 32$	$50,000$	$10,000$

Table 5.1.: The datasets used in this thesis.

Experimental Setup: As described in Section 4.1, for the classifier's backbone, we use DINOv2, a state-of-the-art FVM capable of extracting features from previously unseen datasets. These features can then be passed to a simple base network to obtain their class labels. To validate the efficacy of our choice of the state-of-the-art tool, we compare the performance of DINOv2 with that of ResNet, a well-established alternative. To do so, we evaluate three configurations: a ResNet-50 (pre-trained on IMAGENET [12] dataset) while only training its last linear layer as the classifier, DINOv2 with a single-layer NN classifier, and DINOv2 with a two-layer NN using ReLU activation.

The choice of a single layer as the base network is inspired by the DINOv2 literature [34], which suggests that the extracted features can be directly employed for image classification using a linear layer. However, we extend our evaluation using a slightly more complex base network consisting of two linear layers and ReLU activation. This is to investigate whether the model can benefit from more sophisticated transformations of the features by adding an additional layer and non-linear activation such as ReLU. It should be noted that the backbone parameters are fixed while the base network undergoes training.

Preprocessing: To ensure compatibility of the datasets with DINOv2, images must have three channels, dimensions larger than 28×28, and a size multiple of 14. Hence, for black-and-white images, we repeat the first channel and scale the images to a size of $3 \times 210 \times 210$. For CIFAR-10, we scale the images to match the same size. Similar steps are followed for ResNet, with images scaled to $3 \times 224 \times 224$.

Results: In the Figure 5.1, we present the accuracy on the test-split for the three configurations after training on different fractions of the respective datasets. Notably, in the case of the CIFAR-10, we observed that both DINOv2 configurations outperform the ResNet50 configuration, even when using only 0.01 fraction (1%)

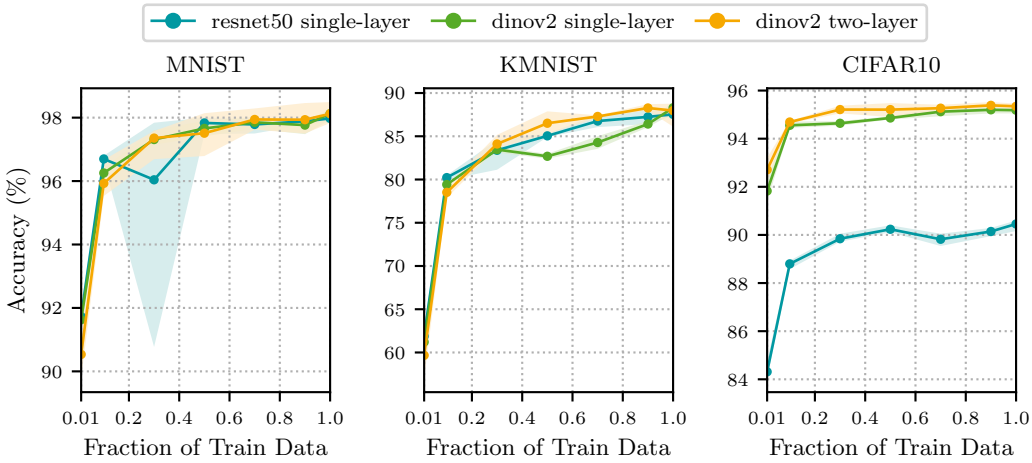


Figure 5.1.: Test accuracy of the three configurations of the backbone and base network, trained on the respective datasets. The plots depict mean accuracy across four random seeds, accompanied by standard deviation in a lighter shade, for different fractions of the training data.

of the train-split for training. Furthermore, the DINOv2-based configurations consistently show higher performance over ResNet as the fraction of training data increases. We observe a slight performance difference between the simpler single-layer classifier and the multi-layer neural network. The latter demonstrated a modest but consistent advantage. In the case of MNIST and KMNIST, a marginal difference in performance is observed, suggesting that the quality of features extracted by the two backbones for these datasets is most likely comparable.

To further assess the effectiveness of the image features from the two backbones, we experiment using the t-SNE [31] technique for dimensionality reduction. This method allows us to visualize the features of each image dataset computed using the respective backbones in a 2-dimensional space. t-SNE, or t-Distributed Stochastic Neighbor Embedding, projects multidimensional feature embeddings to a lower-dimensional space while preserving the proximity of points in the resulting projection. The results of applying t-SNE on features from both DINOv2 and ResNet for CIFAR-10 are presented in Figure 5.2. In this visualization, we observe that the features extracted by DINOv2 form more distinct clusters than those extracted by ResNet, which could potentially contribute to the performance difference observed in Figure 5.1. For MNIST and KMNIST, the t-SNE results can be found in Ap-

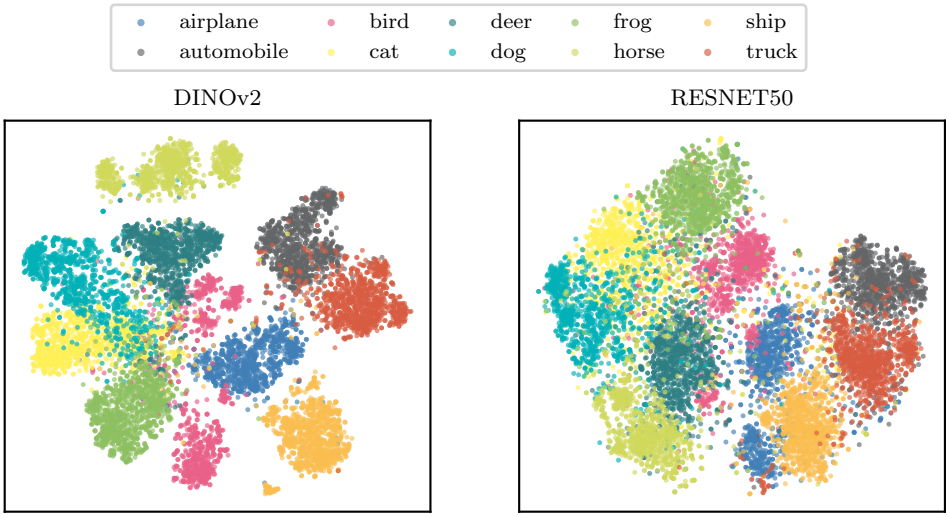


Figure 5.2.: t-SNE visualization of features extracted from 10,000 images of CIFAR-10 dataset using DINOv2 (left) and ResNet-50 (right)

pendix A.4.1. We observe that both backbones present equally distinctive clusters for MNIST in Figure A.1a, whereas for KMNIST in Figure A.1b, visual discernment of distinct clusters is challenging.

Summary: Based on our evaluation of DINOv2 and ResNet configurations, we opt for the DINOv2 model as the backbone for image feature extraction coupled with a two-layer NN with ReLU activation as the base network for classification in our subsequent AL experiments. The decision to use a two-layer NN with nonlinearly instead of a single-layer one is based on the observation that it performs better than the single-layer classifier, albeit by a slight margin. The ReLU activation, in particular, introduces non-linearity and captures complex patterns in the data that a simple linear layer may struggle to represent effectively. Additionally, DINOv2 can produce effective image features. With these results, we validate our use of the state-of-the-art FVM, DINOv2, for feature extraction and then use them with a downstream classifier of multi-layer NN.

5.2. Active Learning

Using DINOv2 as our backbone and a two-layer NN with ReLU activation as the base network for our classifier model, this section presents the results of our RL-based AL method using various state representations as described in Section 4.2.1.

Datasets: We evaluate our AL method and all baselines on three academic image benchmarks summarized in Table 5.1. To ensure compatibility with DINOv2, the images were processed using the steps described in Section 5.1. It can be seen that MNIST and KMNIST comprise black-and-white images, while CIFAR-10 consists of RGB images. This selection of datasets allows us to evaluate our RL-based AL method on datasets with distinct input distributions and varying domains. Specifically, MNIST and KMNIST consist of digits and letters, and CIFAR-10 consists of images of everyday objects.

Experimental Setup: The proposed RL-based AL algorithm consists of two key steps. In the first step, we conducted exhaustive training of the DQN agent, utilizing the source dataset. Repeated simulations of AL are performed using this source dataset, following the procedure outlined in Algorithm 2. We use the MNIST dataset as the source dataset. Moving on to the second step, the trained DQN agent is then applied to the target dataset for the purpose of selecting samples for annotation and actively learning a classifier. To evaluate the performance of the proposed method, KMNIST and CIFAR-10 serve as the target, effectively mimicking an actual stream-based AL scenario. It is important to note that this second step involves only a single episode of AL as detailed in Algorithm 3.

We begin by randomly dividing the train-split of the source dataset, i.e., MNIST, into two distinct parts: `train` and `eval`. The `train` subset is used for training the DQN agent, while the `eval` subset is reserved for later evaluation of the learned agent when actively training a classifier. This is done to evaluate the predictive accuracy of the actively learned classifier using the DQN agent on the same dataset it was originally learned on before progressing to the target dataset. For AL on the target dataset, the train-split is used to simulate an unlabeled stream. Throughout both steps of our AL method, we measure the performance by evaluating the predictive accuracy of the learned classifier on their respective test splits. The predictive accuracy measures the percentage of correctly classified samples.

AL Baseline	AL Type	Query Type	Abbreviation
Random Stream	random	Stream-based	`rand-stream`
Entropy	uncertainity	Pool-based	`entropy`
Margin (BvSB)	uncertainity	Pool-based	`margin`
Core-Set	diversity	Pool-based	`coreset`

Table 5.2.: List of the AL methods that are considered as baselines in this thesis. Each version is given an abbreviation for easier notation.

We now outline the parameter settings of the classifier model and the DQN. The classifier uses DINOv2 as the backbone and two-layer NN as the base classifier with ReLU for nonlinearity, as discussed in Section 5.1. Further details of the base classifier training can be found in Appendix A.2. During each AL step of the RL episode, the base classifier is retrained on the acquired labeled set with a batch size of 64 for 30 epochs. As for the DQN agent, it is a NN with two fully connected hidden layers. We list all the details of the DQN and agent training in Appendix A.3.

In the experimental runs, we define an AL budget of 200 for training a DQN agent on the source dataset using Algorithm 3. At the start of each AL episode, 20 images from every class are randomly chosen, and their labels are obtained to initialize the classifier's parameters. These labeled images are added to the labeled set $\mathcal{U}$. At the beginning of each AL episode, we randomly select 20 images from each class from the unlabeled stream and initialize the labeled set. To report the performance, we apply the trained agent to the `eval` subset of the source data set or the train-split of the target dataset and measure the predictive accuracy of the actively learned classifier. Although the DQN agent is trained on a budget of 200, we also evaluate its performance on a higher budget size than its initial training and report the classifier's accuracy on these respective datasets.

Baselines: We compare our method with the set of AL methods presented Table 5.2, most of which are for pool-based AL scenarios. This means these AL algorithms have access to all unlabeled data when selecting samples to query from the oracle. We briefly introduce these pool-based methods in Section 3.1.4. For each round of the pool-based AL algorithms, the AL selector samples instances for labeling in batches of 5 from the unlabeled pool, and the learning is complete when the predefined budget is reached. All baselines use the same classifier model and training

parameters as our proposed method.

We also introduce a streaming baseline, `rand-stream`, which randomly chooses to annotate an image as it is presented one by one from an unlabeled stream. At each round of AL, streaming algorithms are only permitted to see each unlabeled image once, at whatever time it is presented. In a streaming scenario, the AL algorithms cannot access instances from previous rounds, therefore not allowing them to refine their previous decisions as more data is seen. This makes the streaming approaches rather challenging compared to the pool-based methods.

5.2.1. State Representation 1

Results: In the following, we present the results of the agent training phase of our proposed method using SR1, defined in Section 4.2.1, on the source dataset. In particular, we assess the accuracy of the classifier model. Next, we present the results of AL on the target datasets.

Agent Training on Source Dataset. Figure 5.3 shows the agent learning in terms of the test accuracy of the actively learned classifier. The comparison is drawn between the DQN agent's performance in learning a classifier at the initial stage of training (episode 0) and its subsequent performance after training. As expected, the DQN agent learned a selection policy that achieved a mean predictive accuracy of 92.3% for a budget of 200 over five random seeds.

Upon completion of agent training, it is evaluated on the `eval` subset of the source dataset, MNIST. On the left of Figure 5.4 is a comparison of the agent's performance in terms of the classifier's accuracy on MNIST in an episode of AL against the baselines detailed in Table 5.2. In this evaluation, we extend the AL budget to 400, which is greater than the value set for the agent's training. The agent actively learned a classifier with significant improvement in its predictive accuracy compared to the streaming baseline `rand-stream` for $B = 200$. Additionally, the agent was able to sustain this gain over the larger budget, $B = 400$. In comparison to the best performing pool-based AL algorithm, `margin`, the DQN agent shows a marginal improvement.

Next, the trained agent is applied to the target datasets for AL. First, we consider the KMNIST dataset with images from a domain and distribution that are different from the source dataset. On the right of Figure 5.4 are the results of the agent's per-

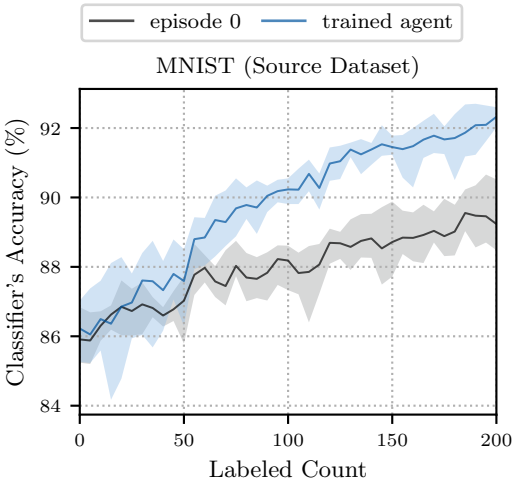


Figure 5.3.: Comparison between the test accuracy of the classifier over an episode of AL using an RL agent before and after it is trained. The state of the AL environment is described using SR1. The plots depict the mean accuracy across five random seeds, accompanied by the standard deviation in a lighter shade.

formance in terms of the test accuracy of the actively learned classifier on KMNIST. The agent shows an improvement in the classifier's accuracy compared to both the streaming baseline, `rand-stream` and the best performing pool-based AL baseline, `margin`, for the budget it was trained on initially, $B = 200$. We also note that the agent shows improved performance over the baselines for a larger budget, $B = 400$.

However, the trained agent failed to select any samples for labeling when applied to a second target dataset, CIFAR-10. Further investigation revealed that the class similarity values, s^{cs}, in SR1 for the CIFAR-10 images differed in the range from the other two datasets, particularly MNIST, on which the agent was trained for AL. In Table 5.3 are the computed class similarities for an image with the class centers of the dataset as per equation (4.2), averaged over 100 images. We notice a difference in s^{cs} values between CIFAR-10 and the other two datasets. Given that the agent was trained on MNIST, this shift in s^{cs} from the agent's perspective could be interpreted as an unexplored region in its state space. Consequently, the agent fails to recognize the significance of images from the CIFAR-10 dataset, resulting in its inability to actively select a sample.

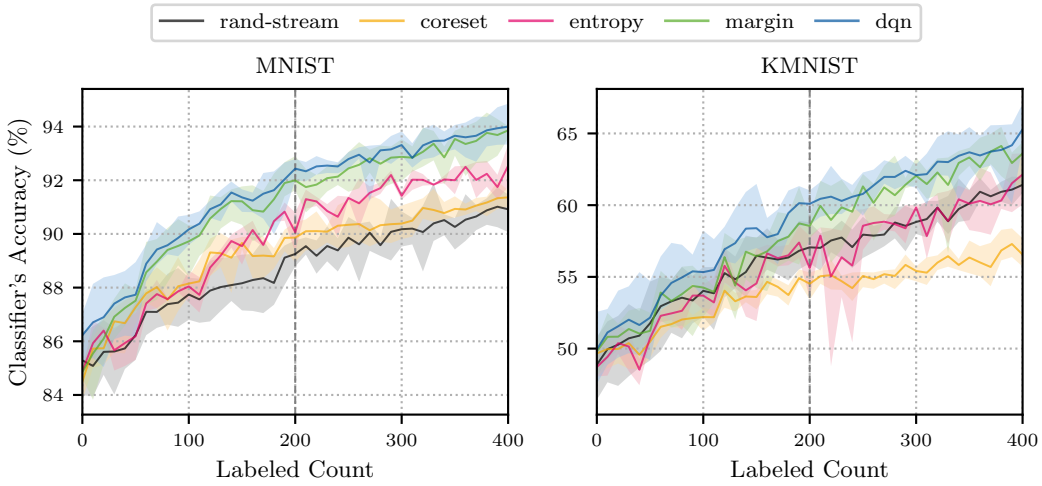


Figure 5.4.: The performance of DQN agent trained using SR1 on MNIST, with respect to baseline AL methods on different datasets. The vertical dashed line marks the budget for which the DQN agent was trained. The plots show the mean test accuracy of the classifier across five random seeds with the standard deviation in a lighter shade.

	Class Similarity s^{cs}
MNIST	$(0.87, 0.82, 0.88, 0.86, 0.88, 0.88, 0.89, 0.88, 0.86, 0.89)$
KMNIST	$(0.80, 0.83, 0.84, 0.80, 0.83, 0.80, 0.82, 0.81, 0.83, 0.83)$
CIFAR-10	$(0.24, 0.28, 0.27, 0.29, 0.29, 0.28, 0.24, 0.27, 0.22, 0.28)$

Table 5.3.: Computed average of class similarities for 100 unlabeled images from the respective data streams according to equation (4.2)

5.2.2. State Representation 2

Results: Here, we present the results of AL with the agent trained using SR2, as outlined in Section 4.2.1. Following a structure similar to the previous section, we present the results from the agent training phase on the source dataset. Subsequently, we assess the accuracy of the classifier model. Next, we present the results of AL on the target datasets.

Agent Training on Source Dataset. The performance of the DQN agent during training is visualized in Figure 5.5, showcasing the test accuracy of the classifier in an AL episode at both the beginning (episode 0) and the end of its training. The

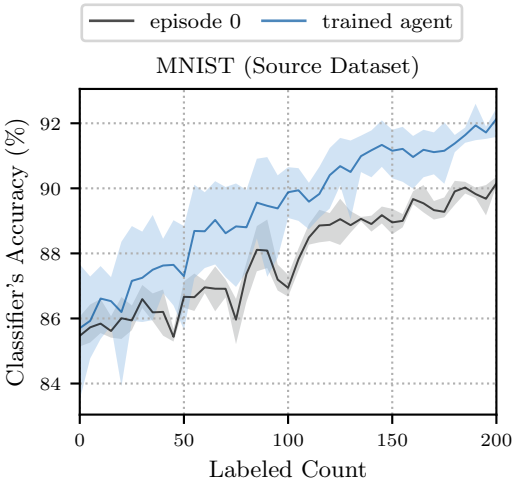


Figure 5.5.: Comparison between the test accuracy of the classifier over an episode of AL using an RL agent before and after it is trained. The state of the AL environment is described using SR2. The plots depict the mean accuracy across five random seeds, accompanied by the standard deviation in a lighter shade.

AL policy learned by the agent achieves a mean accuracy of 92.13% for a budget of 200 over five random seeds.

The trained agent is then applied to the `eval` subset of MNIST. On the top left of Figure 5.6, we observe the accuracy results of the classifier actively learned with the trained agent. Compared to the streaming baseline, `rand-stream`, the agent's classifier shows a significant improvement in accuracy for the budget the agent was initially trained on, as well as for a larger budget of $B = 400$. However, the agent performs on par with the best algorithm among the pool-based AL baselines, `margin`.

AL on Target Datasets. Subsequently, the agent trained on the source dataset for $B = 200$ is applied to the target datasets: KMNIST and CIFAR-10. The accuracy results for KMNIST and CIFAR-10 are presented in the top right and bottom sections of Figure 5.6, respectively, and are compared to the considered baselines. The agent shows a significant improvement in classifier accuracy compared to `rand-stream` for both target datasets. Compared to the best performing pool-based AL baseline, `margin`, the DQN agent exhibits similar performance in the case of KMNIST, with a slight edge in performance for CIFAR-10.

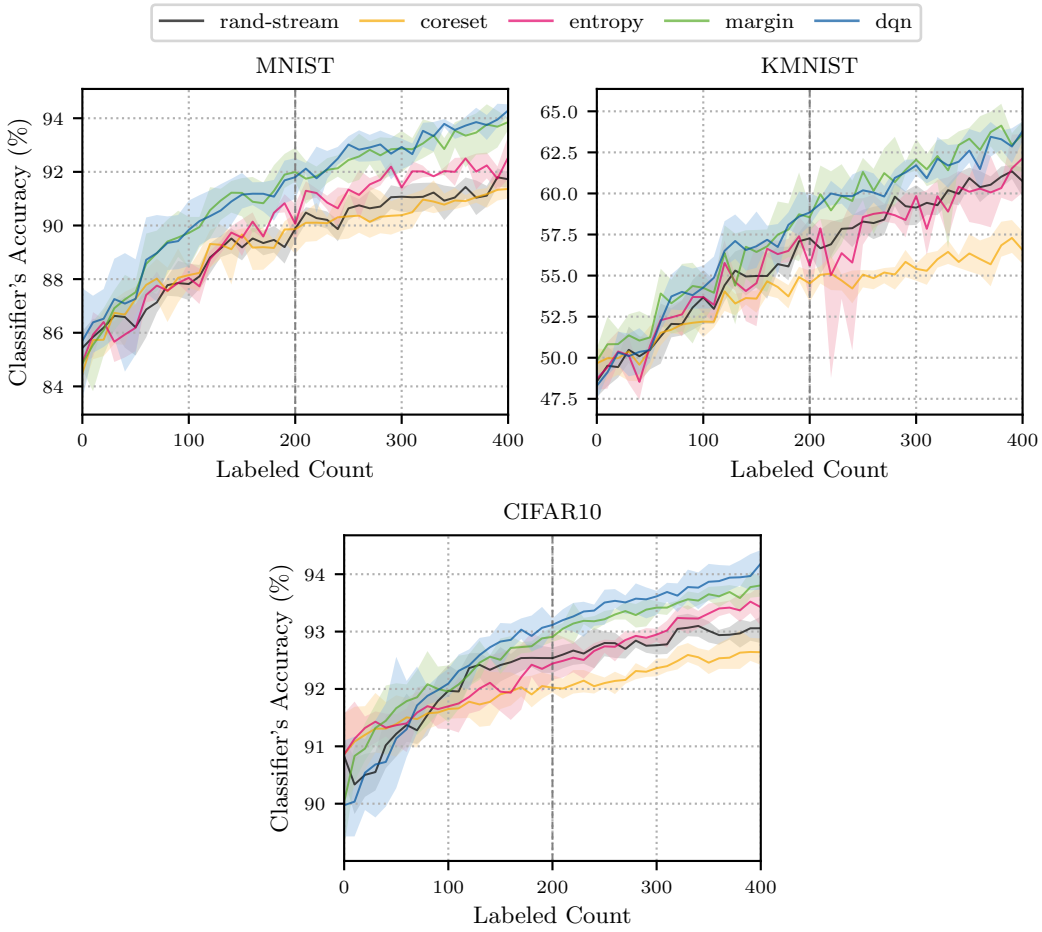


Figure 5.6.: The performance of DQN agent trained using SR2 on MNIST, with respect to baseline AL methods on different datasets. The vertical dashed line marks the budget for which the DQN agent was trained. The plots depict the mean test accuracy of the classifier across five random seeds accompanied by the standard deviation in a lighter shade.

5.2.3. Ablation Study

Looking back at the components of SR1 and SR2, as summarized in Table 4.1, it is evident that both representations for the RL agent incorporate the margin score, s^m, computed for an unlabeled instance using the classifier. Additionally, the results discussed previously highlight the comparable performance of the agent to the baseline `margin`, and, in some cases, a slight improvement in the agent's

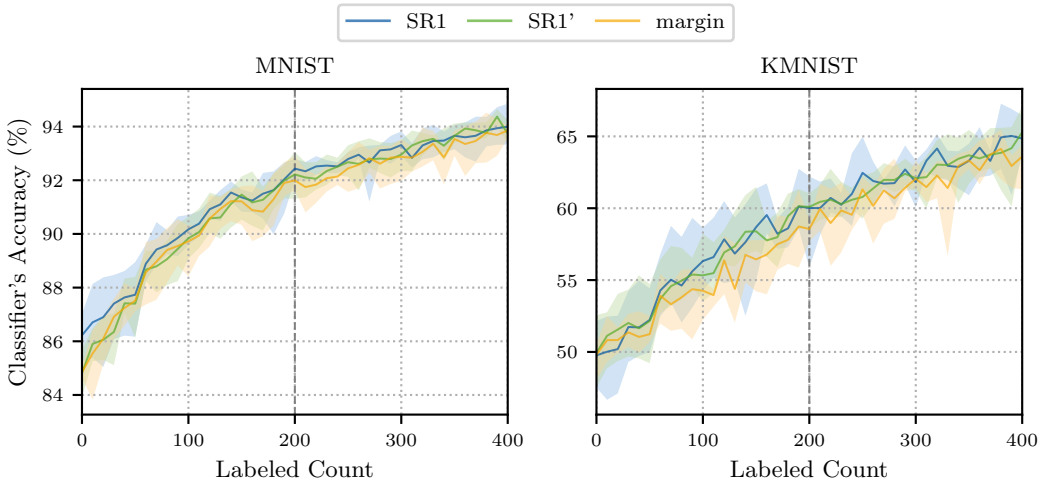


Figure 5.7.: The performance of the DQN agent trained using SR1' on the MNIST dataset is compared against the performance of the agent trained with SR1 and the AL baseline `margin`. The vertical dashed line indicates the budget for which the DQN agent was trained. The plots displays the mean test accuracy of the classifier across five random seeds, along with the standard deviation in a lighter shade.

performance. Consequently, these observations prompt an ablation study in the following subsections to investigate the significance of the margin score on the AL performance of an agent trained using SR1 or SR2.

State Representation 1

Results: The SR1 ablation study modifies the original state representation, (s^{cs}, s^{e}, s^{m}) to introducing a new state representation, SR1' $:= (s^{cs}, s^{e})$, which excludes the margin score s^{m}. Following the same agent training steps as in the previous experiments, we train the agent using SR1' and apply it to the respective source (MNIST) and target (KMNIST) datasets. The classifier's AL accuracy is then evaluated in comparison to the original state SR1 and the baseline `margin`. The results obtained are illustrated in Figure 5.7.

The results show a similarity in classifier accuracy between the original state SR1 and the ablated state SR1'. This leads to the insight that, for SR1, the DQN agent relies primarily on the class similarities and the entropy score of a given unlabeled

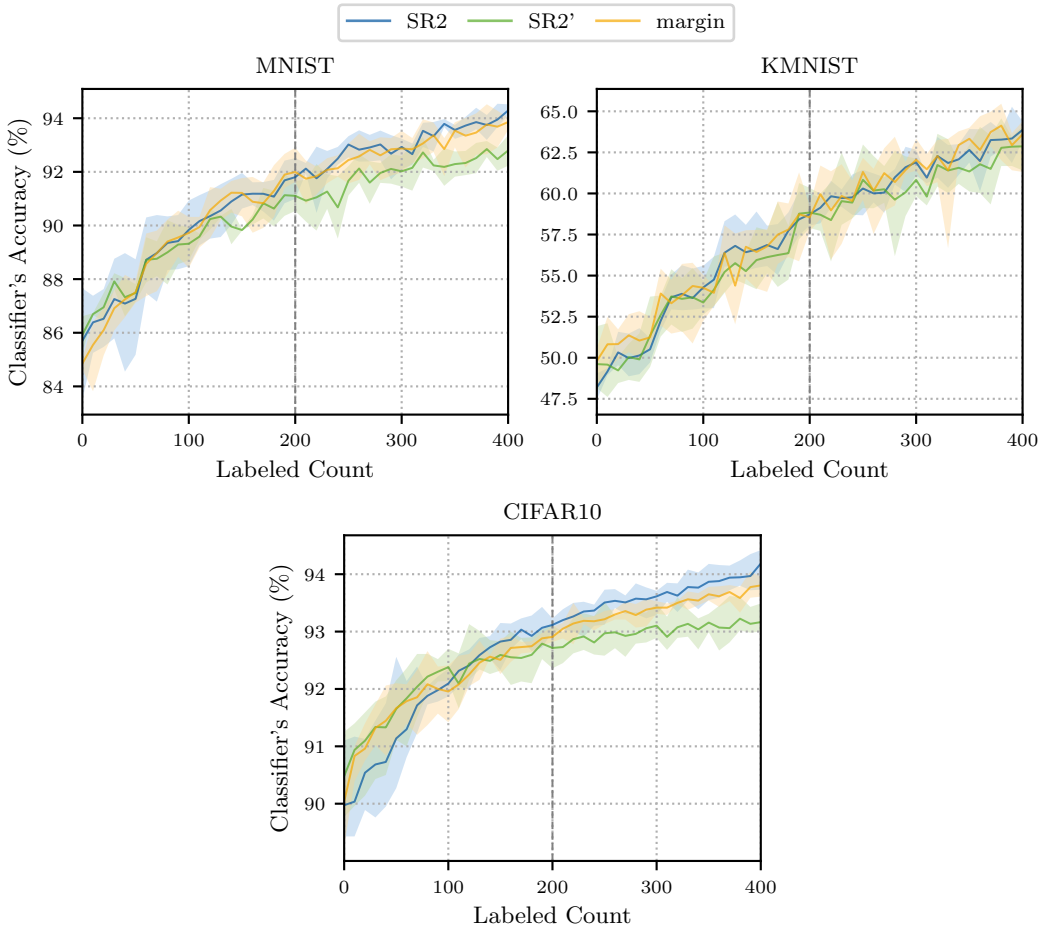


Figure 5.8.: The performance of the DQN agent trained using SR2' on the MNIST dataset is compared against the performance of the agent trained with SR2 and the AL baseline `margin`. The vertical dashed line indicates the budget for which the DQN agent was trained. The plots displays the mean test accuracy of the classifier across five random seeds, along with the standard deviation in a lighter shade.

image to learn the classifier actively. The margin score, while present, does not appear to influence the agent's learning process for SR1 significantly.

State Representation 2

Results: Here, the ablation study introduces SR2' := (s^{cp}, s^{e}), by excluding the margin score, s^{m}, from SR2. We consider the MNIST for the source dataset, and given that the agent can actively learn on CIFAR-10 using SR2, we consider KMNIST and CIFAR-10 for the target datasets. In Figure 5.8, the prediction accuracy of the classifier learned using agent SR2' is then compared with the original agent SR2 and the baseline `margin`.

The results show that the prediction accuracy of the classifier learned using the agent trained with SR2' falls relative to the original SR2 agent. Therefore, these findings inform us that the agent trained using SR2 relies not only on the class probabilities and the entropy score of a given unlabeled image but also on the margin score.

5.2.4. Summary

In total, we have conducted three experiments to evaluate the performance of our RL-based AL method in this thesis - one for each of the two state representations and an ablation study.

Our first proposed state representation, SR1, incorporates class similarity, entropy, and margin score for a given unlabeled image. By leveraging this representation, the agent achieves a superior AL policy compared to the streaming baseline, `rand-stream`, especially on the source dataset MNIST - the dataset on which the agent was trained. However, when the agent is applied to a stream of CIFAR-10, characterized by a distinct input distribution (RGB images) compared to the source dataset, the agent fails to select samples from this unlabeled stream. On the contrary, when presented with KMNIST as the target dataset, sharing a similar distribution (both containing black-and-white images), the agent actively learns a classifier that outperforms its corresponding streaming baseline. The agent exhibits a slight edge compared to the best-performing pool-based AL algorithm, `margin`. On the other hand, the RL agent outperforms the other two pool-based AL approaches, `entropy` and `coreset`.

The second state representation, SR2, comprises the class predictions, entropy, and margin score computed from the classifier for the given unlabeled image from a

stream. Similar to the results from SR1, the agent trained using SR2 outperforms the streaming baseline in the source dataset MNIST. When applied to the target datasets KMNIST and CIFAR-10, the agent outperforms the streaming baseline on both datasets. Unlike the SR1 agent, the SR2 agent can perform AL on CIFAR-10. This can be attributed to the fact that we use class predictions in SR2 rather than class similarities computed in the feature space of the respective datasets. We find that using such a state representation makes the RL agent robust against differences in the source and target data distributions. The SR2 agent exhibits a similar performance or slight edge compared to the best-performing pool-based AL algorithm, `margin`. On the other hand, the RL agent outperforms the other two pool-based AL approaches, `entropy` and `coreset`.

The two experiments above demonstrate that the RL agent's performance is comparable to the pool-based AL baseline, `margin`, although it outperforms `entropy` and `coreset`. Hence, we analyze the significance of the margin score in SR1 and SR2 for the learned RL agent. We find that out of the components of the state SR1, the agent is reliant on the class probabilities and entropy compared to the margin score. This observation is based on the result that excluding the margin score from SR1 does not affect the RL agent's performance in AL. However, in the case of SR2, we note a drop in the agent's performance in AL when the margin score is excluded from the state. In essence, these findings inform us that the margin score plays a varying significance in the state representations, SR1 and SR2, and, therefore, the respective RL agent's performance for our AL scenario.

6. Conclusion

In this thesis, we propose a stream-based AL method that focuses on using RL, a data-driven approach. We apply our proposed data-driven AL framework in two stages: first, we train the RL agent on a given source dataset to learn a suitable selection strategy, and then, second, the policy learned on the source dataset is applied to a target dataset for actively learning a classifier. The benefit of this approach is that the AL selection strategy decides to query the label for an unlabeled instance from a stream, unlike traditional AL methods, which assume access to the entire unlabeled dataset ahead of time. We propose two state representations for our RL agent in the context of image classification. Our experimental results show that for the RL agent to be robust from the difference between the source and target data distributions, a state representation that is also robust from these differences is required. For a given unlabeled image, the first state we propose comprises information from the actively learned classifier and the image feature space. We observe a successful performance of the RL agent when both the source and target datasets have a similar distribution. However, the agent fails if the distribution is vastly different. Hence, we propose a second state composed of information from only the actively learned classifier about a given unlabeled image. In this case, we observe that a difference in the source and target datasets does not affect the performance of the RL agent in AL. Another insight we gather in this thesis is that each component that comprises the state representation of the RL agent has a varied effect on the AL performance. However, the exact contribution of each component remains an open research question. The performance of the proposed methods is evaluated against a streaming and three relevant pool-based AL baselines. We evaluate their performances using MNIST as the source dataset and KMNIST and CIFAR-10 as the target datasets. The benchmark results show that our proposed methods outperform the streaming baseline; however, we observe on-par performance to the best of the three considered pool-based AL baselines.

Future Scope: This section aims to outline potential directions for further research, improvements, or advancements related to the developed method in the thesis.

In robotic applications, the lack of large amounts of annotated, task-related training data has paved a path towards the idea of learning from synthetic data, where large amounts of task-relevant annotated data can be obtained from simulation with relatively less time and manual efforts [20]. However, another set of challenges exists, the so-called *Sim-to-Real gap*, which is the main barrier to transferring a model learned on simulation data to real-world robotics applications. Therefore, despite the availability of simulation data, there is still a need for annotated real-world data, albeit in small amounts. An exciting application of our AL method can be to reduce this Sim-to-Real gap. Here, the simulation data can be treated as the source dataset on which the RL agent learns an AL selection policy, and the corresponding learned policy is then applied to the real-world dataset, which would be the target dataset.

A well-known limiting factor of NN is their confidence calibration, which refers to the difficulty of predicting probability estimates that accurately reflect the likelihood of correctness [17]. In our experiments for image classification, we work with relatively simple image datasets, allowing us to use shallow NN for classification. However, more complex real-world datasets may require the use of a deeper NN model, which often exhibits poor confidence calibration. This could, in turn, lead to a weaker state representation for the RL agent. Hence, identifying a state representation robust to these challenges remains an open research question for more complex datasets.

The experiments we demonstrate explore the performance of our method on an i.i.d (independent and identically distributed) data stream. However, this may not always stand true in real-world applications. Therefore, investigating the robustness of our method in a non-i.i.d stream compared to other baselines AL methods can be another interesting extension to this presented work.

A. Appendix

A.1. Pool-based Active Learning

Algorithm 4 General Active Learning Algorithm for Pool-based Scenario

Require: unlabeled data pool $\mathcal{U}$; number of AL rounds $R \in \mathbb{N}$ and number of samples added per round $k \in \mathbb{N}$; untrained or pre-trained predictor network F_θ; AL strategy, the selector S

Ensure: Trained predictor network F_θ, labeled set $\mathcal{L}$

procedure ActiveLearning($\mathcal{U}$, R, k, F_θ, S)
  Initialize $\mathcal{L}$ from $\mathcal{U}$ and train classifier F_θ ▷ Set of labeled samples
  for `round in` $1 \dots R$ **do**
    select k instances $\{x_1, \dots x_k\}$ from $\mathcal{U}$ using active learning strategy S
    query respective labels $\{y_1, \dots y_k\}$ from the oracle
    $L \leftarrow L \cup \{(x_1, y_1), \dots, (x_k, y_k)\}$
    $U \leftarrow U \setminus \{x_1, \dots x_k\}$
    retrain F_θ on $\mathcal{L}$
  end for
  return F_θ and labeled set L
end procedure

A.2. Parameters for Classifier Training

Base Network	Layer	Type	Activation	Size
single-layer	1	Fully-Connected	Linear	Output Size
two-layer NN	1	Fully-Connected	ReLU	32
	2	Fully-Connected	Linear	Output Size

Table A.1.: Single layer and two-layer base network configurations for ResNet-50 and DINOv2 backbone

Parameter name		Parameter	Value
Adam optimizer	weight decay	λ	$1 \cdot 10^{-8}$
	learning rate	γ	0.002
Batch Size		batch_size	64
Training Epochs		n_epochs	30

Table A.2.: Used parameters for the training of the base classifier.

A.3. Parameters for DQN Training

Layer	Type	Activation	Size
1	Fully-Connected	ReLU	256
2	Fully-Connected	ReLU	256
3	Fully-Connected	Linear	number of actions

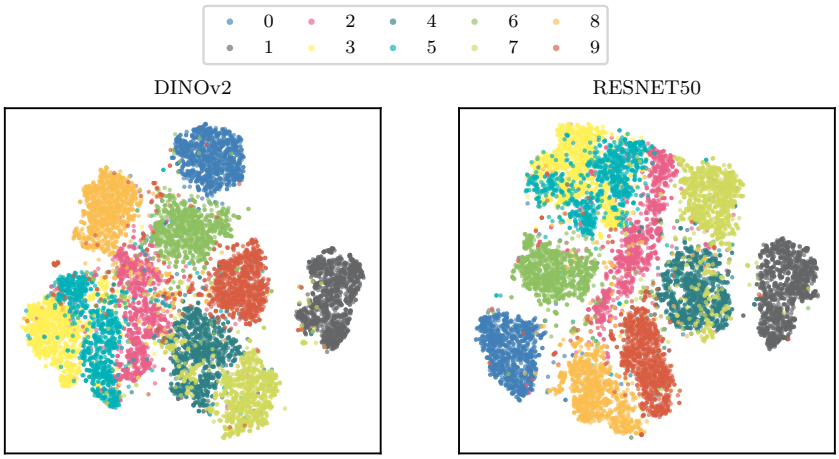
Table A.3.: Neural network model for the deep Q-network

Parameter Name	Parameter	Value
total timesteps	total_timesteps	$100,000$
discount factor	γ	0.99
replay buffer size	buffer_size	$50,000$
timesteps before learning starts	learning_starts	0
training frequency	train_freq	4
number of gradient steps every train_freq	gradient_steps	1
target network update interval	target_update_interval	$1,000$
batch size	batch_size	64
exploration fraction	exploration_fraction	0.06
exploration initial epsilon	exploration_initial_eps	1.0
exploration final epsilon	exploration_final_eps	0.02

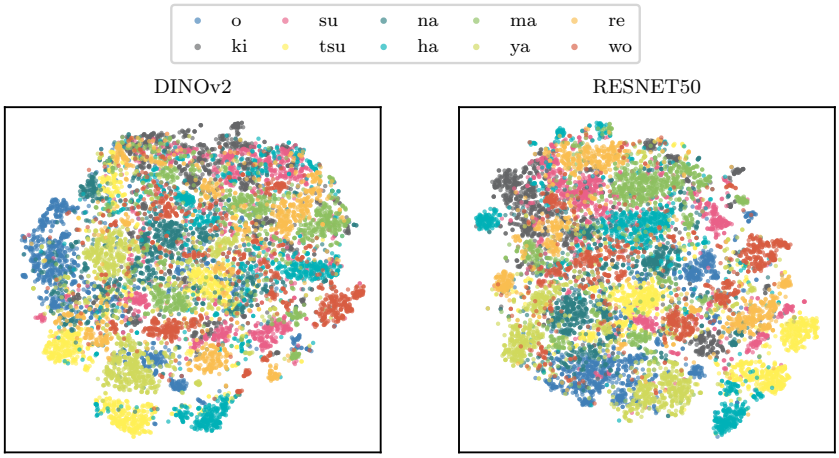
Table A.4.: Used DQN parameters in the stable baselines3 library [41].

A.4. Further Results

A.4.1. t-SNE Plots - MNIST, KMNIST



(a) MNIST



(b) KMNIST

Figure A.1.: t-SNE visualization of features extracted from 10,000 images from respective datasets using DINOv2 (left) and ResNet-50 (right)

A.5. On the Use of AI Tools

For this these we use the following tools to evaluate and improve the quality of the writing. In the following, we briefly discuss the specifics of how they were used.

ChatGPT [8]: Following are the prompts or variations thereof used for the writing of this thesis:

- Evaluation of Text Consistency and grammar:
  - "In terms of the flow of the main ideas presented in the following paragraphs, do you think they are consistent?: ..."
  - "Check the punctuation for the following paragraphs: ..."
- Automate minor tasks:
  - "Can you improve the following short caption <...> for a latex figure using the following long caption: ..."
  - "Generate `\newglossaryentry` for the following macros defined for math symbols: ..."
  - "Clean up the following latex pseudocode: ..."

None of the text present in this thesis is a forthright copy of the chatGPT response; however, it has been used as a feedback mechanism to improve the text by oneself if deemed beneficial.

DeepL Write [11]: DeepL Write is an AI writing tool that has been used to improve the written text by checking the grammar, punctuation, and style.

Acronyms

ADL Active Deep Learning
AL Active Learning

BvSB Best-versus-Second Best

CNN Convolutional Neural Network
CV Computer Vision

DQN Deep Q-Network

FVM Foundational Vision Model

MDP Markov Decisions Processes
MQS Membership Query Synthesis

NN Neural Network

RL Reinforcement Learning

List of Mathematical Symbols

Sets

$\mathbb{N}$	Natural numbers
$\mathbb{R}$	Real numbers

Probability Theory

$p(\cdot)$	Probability distribution
$\mathbb{E}[\cdot]$	Expectation value

Machine Learning

$\mathcal{C}$	Set of classes
c	Specific class in $\mathcal{C}$
x	Labeled or unlabeled sample instance
x^f	Feature representation of sample x
y	Label associated with the sample x
$\hat{y}_c$	Predicted probability of sample x belonging to class c
$F_\theta(\cdot)$	Predictor or classifier model
θ	Predictor or classifier model weights

Active Learning

$\mathcal{U}$	Unlabeled pool or stream
$\mathcal{L}$	Labeled set
$\mathcal{H}$	Reward Set

B	Budget
$S(\cdot)$	Active learning selector
$\mathcal{M}$	Set of class centers in the feature space

Reinforcement Learning

$\mathcal{S}$	State space
$\mathcal{A}$	Action space
R	Reward space
M	Memory buffer
i	Episode time step
s_i	State at time step i
a_i	Action at time step i
r_i	Reward at time step i
γ	Discount factor
$p(s_{i+1} \mid s_i, a_i)$	State transition probability
$\pi(s_i, a_i)$	Reinforcement learning policy
$V(s_i)$	State value function
$Q(s_i, a_i)$	Q-value function
ϵ	Exploration parameter
ϕ	Reinforcement learning agent's model weights

List of Figures

List of Tables

Bibliography

[1] D. Angluin, "Queries and concept learning," *Machine Learning*, vol. 2, pp. 319–342, 1988.

[2] S. E. Argamon and I. Dagan, "Committee-based sample selection for probabilistic classifiers," *J. Artif. Intell. Res.*, vol. 11, pp. 335–360, 1999.

[3] T. B. Brown, B. Mann, N. Ryder, *et al.*, "Language models are few-shot learners," *ArXiv*, vol. abs/2005.14165, 2020.

[4] S. Budd, E. C. Robinson, and B. Kainz, "A survey on active learning and human-in-the-loop deep learning for medical image analysis," *Medical image analysis*, vol. 71, p. 102 062, 2019.

[5] X. Cao, J. Yao, Z. Xu, and D. Meng, "Hyperspectral image classification with convolutional neural network and active learning," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 58, pp. 4604–4616, 2020.

[6] M. Caron, H. Touvron, I. Misra, *et al.*, "Emerging properties in self-supervised vision transformers," *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 9630–9640, 2021.

[7] A. Casanova, P. O. Pinheiro, N. Rostamzadeh, and C. J. Pal, *Reinforced active learning for image segmentation*, 2020. arXiv: `2002.06583 [cs.CV]`.

[8] ChatGPT, "Gpt-3.5," 2023. [Online]. Available: `https://www.openai.com`.

[9] J. Choi, I. Elezi, H.-J. Lee, C. Farabet, and J. M. Alvarez, "Active learning for deep object detection via probabilistic modeling," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

[10] T. Clanuwat, M. Bober-Irizar, A. Kitamoto, A. Lamb, K. Yamamoto, and D. Ha, "Deep learning for classical japanese literature," *ArXiv*, vol. abs/1812.01718, 2018.

[11] *Deepl write: Better texts in no time.* [Online]. Available: `https://www.deepl.com/write`.

[12] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.

[13] M. Fang, Y. Li, and T. Cohn, "Learning how to active learn: A deep reinforcement learning approach," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017.

[14] V. Fedorov, "Optimal experimental design," *WIREs Computational Statistics*, vol. 2, no. 5, pp. 581–589, 2010. DOI: `https://doi.org/10.1002/wics.100`.

[15] Y. Gal, R. Islam, and Z. Ghahramani, "Deep bayesian active learning with image data," ser. ICML'17, 2017, pp. 1183–1192.

[16] M. Gao, Z. Zhang, G.-D. Yu, S. Ö. Arik, L. S. Davis, and T. Pfister, "Consistency-based semi-supervised active learning: Towards minimizing labeling cost," in *European Conference on Computer Vision*, 2019.

[17] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," in *International Conference on Machine Learning*, 2017.

[18] M. J. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," *ArXiv*, vol. abs/1507.06527, 2015.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2015.

[20] J. L. Jianxiang Feng, M. Durner, and R. Triebel, "Bayesian active learning for sim-to-real robotic perception," in *IROS*, 2022.

[21] A. J. Joshi, F. M. Porikli, and N. Papanikolopoulos, "Multi-class active learning for image classification," *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2372–2379, 2009.

[22] Y. Kim, "Convolutional neural networks for sentence classification," in *Conference on Empirical Methods in Natural Language Processing*, 2014.

[23] A. Kirsch, J. van Amersfoort, and Y. Gal, "Batchbald: Efficient and diverse batch acquisition for deep bayesian active learning," in *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[24] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009.

[25] Y. LeCun, Y. Bengio, and G. E. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, 2015.

[26] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, pp. 2278–2324, 1998.

[27] D. D. Lewis and W. A. Gale, "A sequential algorithm for training text classifiers," in *Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1994.

[28] M. Liu, W. Buntine, and G. Haffari, "Learning how to actively learn: A deep imitation learning approach," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2018.

[29] P. Liu, L. Wang, R. Ranjan, G. He, and L. Zhao, "A Survey on Active Deep Learning: From Model Driven to Data Driven," *ACM Computing Surveys*, 2022.

[30] Z. Liu, J. Wang, S. Gong, D. Tao, and H. Lu, "Deep reinforcement active learning for human-in-the-loop person re-identification," *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 6121–6130, 2019.

[31] L. van der Maaten and G. E. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008.

[32] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

[33] A. Narr, R. Triebel, and D. Cremers, "Stream-based active learning for efficient and adaptive classification of 3d objects," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016.

[34] M. Oquab, T. Darcet, T. Moutakanni, *et al.*, *Dinov2: Learning robust visual features without supervision*, 2023.

[35] L. Ouyang, J. Wu, X. Jiang, *et al.*, "Training language models to follow instructions with human feedback," *ArXiv*, vol. abs/2203.02155, 2022.

[36] A. Parvaneh, E. Abbasnejad, D. Teney, G. R. Haffari, A. van den Hengel, and J. Q. Shi, "Active learning by feature mixing," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 12 237–12 246.

[37] A. Paszke, S. Gross, F. Massa, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Neural Information Processing Systems*, 2019.

[38] U. Patel and V. Patel, "Active learning-based hyperspectral image classification: A reinforcement learning approach," *The Journal of Supercomputing*, 2023.

[39] A. Radford, J. W. Kim, C. Hallacy, *et al.*, "Learning transferable visual models from natural language supervision," in *International Conference on Machine Learning*, 2021.

[40] A. Radford and K. Narasimhan, "Improving language understanding by generative pre-training," 2018.

[41] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.

[42] S. Ravi and H. Larochelle, "Meta-learning for batch mode active learning," in *International Conference on Learning Representations*, 2018.

[43] P. Ren, Y. Xiao, X. Chang, *et al.*, "A survey of deep active learning," *ACM Comput. Surv.*, 2021.

[44] A. Saran, S. Yousefi, A. Krishnamurthy, J. Langford, and J. T. Ash, "Streaming active learning with deep neural networks," in *Proceedings of the 40th International Conference on Machine Learning*, ser. ICML'23, 2023.

[45] A. Saran, S. Yousefi, A. Krishnamurthy, J. Langford, and J. T. Ash, "Streaming active learning with deep neural networks," *arXiv preprint arXiv:2303.02535*, 2023.

[46] O. Sener and S. Savarese, "Active learning for convolutional neural networks: A core-set approach," in *International Conference on Learning Representations*, 2018.

[47] B. Settles, "Active Learning Literature Survey," University of Wisconsin-Madison Department of Computer Sciences, Technical Report, 2009.

[48] S. P. Singh and R. S. Sutton, "Reinforcement learning with replacing eligibility traces," *Mach. Learn.*, pp. 123–158, 1996, ISSN: 0885-6125.

[49] J. Sourati, A. Gholipour, J. G. Dy, X. Tomas-Fernandez, S. Kurugol, and S. Warfield, "Intelligent labeling based on fisher information for medical image segmentation using deep learning," *IEEE Transactions on Medical Imaging*, vol. 38, pp. 2642–2653, 2019.

[50] L. Sun and Y. Gong, "Active learning for image classification: A deep reinforcement learning approach," in *2019 2nd China Symposium on Cognitive Computing and Hybrid Intelligence (CCHI)*, 2019.

[51] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, vol. 3, pp. 9–44, 1988.

[52] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. A Bradford Book, 2018, ISBN: 0262039249.

[53] R. S. Sutton, D. A. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Neural Information Processing Systems*, 1999.

[54] M. Towers, J. K. Terry, A. Kwiatkowski, *et al.*, *Gymnasium*, Mar. 2023. DOI: 10.5281/zenodo.8127026.

[55] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, University of Cambridge, 1989.

[56] C. Watkins and P. Dayan, "Technical note: Q-learning," *Machine Learning*, vol. 8, pp. 279–292, 1992.

[57] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, pp. 229–256, 1992.

[58] M. P. Woodward and C. Finn, "Active one-shot learning," *ArXiv*, vol. abs/1702.06559, 2017.

[59] J. Wu, J. Chen, and D. Huang, "Entropy-based active learning for object detection with progressive diversity constraint," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[60] D. Yoo and I.-S. Kweon, "Learning loss for active learning," *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 93–102, 2019.