Karlsruhe Institute of Technology

Intelligent System Security
Karlsruhe Institute of Technology

Continual Active Learning for Effective Model Stealing Attacks

Bachelor's Thesis of

Felix Möller

at the Department of Informatics
KASTEL – Institute of Information Security and Dependability

Reviewer:	Jun.-Prof. Dr. Christian Wressnegger
Second reviewer:	Prof. Dr. Thorsten Strufe
Advisor:	M.Sc. Yilin Ji

10. January 2023 – 10. May 2023

I declare that I have developed and written the enclosed thesis completely by myself. I have submitted neither parts of nor the complete thesis as an examination elsewhere. I have not used any other than the aids that I have mentioned. I have marked all parts of the thesis that I have included from referenced literature, either in their original wording or paraphrasing their contents. This also applies to figures, sketches, images and similar depictions, as well as sources from the internet.

Karlsruhe, May 10th,2023

. . . . . .

(Felix Möller)

Abstract

The advancement of machine learning and especially deep learning has paved the way for remarkable breakthroughs in the computer vision domain [1, 2, 3]. Despite its benefits, deep learning is a demanding task that requires significant resources due to the increasingly complex architecture of state-of-the-art models and the need for extensive training with labeled data. Technology companies like Google, Amazon and Microsoft, however, have access to a lot of computing power, which they monetize by offering machine learning as a service (MLaaS). MLaaS platforms allow customers to train their models in the cloud and make them publicly accessible via prediction application programming interfaces (APIs). While MLaaS makes machine learning more attainable, it poses a security risk. Previous works have shown that it is possible to infer numerous properties (e.g., training data [4] and architecture [5]) of the model behind the prediction API [6, 7]. However, we purely focus on stealing the *functionality* of the target model, i.e., its ability to predict the correct class of a given input image. In this thesis, we build upon ActiveThief [8], a model extraction framework using active learning strategies to steal deep neural networks. First, we propose a novel approach named continual active learning, which combines active learning and continual learning to improve the trade-off between accuracy and efficiency in the active learning process. Our evaluation demonstrates that continual active learning reduces validation accuracy by 10-30% while reducing execution time by 60-98%. Next, we incorporate continual active learning into the ActiveThief framework and investigate whether our findings from traditional continual active learning are transferable to model stealing. We find that continual active learning falls short of the performance achieved by ActiveThief, specifically when training with the predicted label of the target model. Furthermore, we show that continual active learning benefits from training with the softmax output of the target model, which has been observed for previous model stealing attacks [8, 9].

Zusammenfassung

Der Fortschritt von Maschinellem Lernen und insbesondere von Deep Learning hat den Weg für bemerkenswerte Durchbrüche im Bereich der Bilderkennung geebnet [1, 2, 3]. Trotz seiner offensichtlichen Vorteile ist Deep Learning ressourcenintensiv, da die Architektur moderner Modelle immer komplexer wird und diese Modelle auf großen Mengen annotierter Daten trainiert werden müssen. Technologieunternehmen wie Google, Amazon und Microsoft haben jedoch Zugang zu einer großen Menge an Rechenleistung, die sie monetarisieren, indem sie Machine Learning as a Service (MLaaS) anbieten. MLaaS-Plattformen ermöglichen es Kunden, ihre Modelle in der Cloud zu trainieren und über Vorhersage-APIs öffentlich zugänglich zu machen. Während MLaaS Machine Learning zugänglicher macht, birgt es zugleich Sicherheitsrisiken. Vorangegangene Forschungsarbeiten haben gezeigt, dass es möglich ist, eine Vielzahl an Eigenschaften (z.B. Trainingsdaten [4] und Architektur [5]) des Modells hinter der Vorhersage-API zu inferieren [6, 7]. Wir konzentrieren uns jedoch ausschließlich auf das Stehlen der *Funktionalität* des Zielmodells, also seine Fähigkeit, die richtige Klasse eines gegebenen Eingabebildes vorherzusagen. In dieser Arbeit bauen wir auf ActiveThief [8] auf. ActiveThief ist ein Framework zum Stehlen von Deep Neural Networks mittels Active Learning. Zunächst schlagen wir einen neuen Ansatz namens Continual Active Learning vor, der Active Learning und Continual Learning kombiniert, um den Trade-off zwischen Genauigkeit und Effizienz im Active-Learning-Prozess zu verbessern. Unsere Evaluation ergibt, dass Continual Active Learning die Validierungsgenauigkeit um 10-30% und zugleich die Ausführungszeit um 60-98% reduziert. Darüber hinaus integrieren wir Continual Active Learning in das ActiveThief Framework und untersuchen, ob unsere Erkenntnisse aus dem traditionellen Continual Active Learning auf die Model-Stealing-Domäne übertragbar sind. Wir stellen fest, dass Continual Active Learning nicht die Genauigkeit von ActiveThief erreicht, insbesondere wenn mit dem vorhergesagten Label des Zielmodells trainiert wird. Zuletzt zeigen wir, dass Continual Active Learning davon profitiert, wenn mit den Vorhersagewahrscheinlichkeiten des Zielsmodells trainiert wird, was bereits bei vorherigen Model-Stealing-Angriffen beobachtet wurde [8, 9].

Contents

List of Figures

List of Tables

Glossary

A-GEM averaged gradient episodic memory

ALASSO asymmetric loss approximation by single-side overestimation

API application programming interface

BADGE batch active learning by diverse gradient embeddings

BALD bayesian active learning by disagreement

CNN convolutional neural network

DFAL deepfool active learning

EWC elastic weight consolidation

FIM Fisher information matrix

GAN generative adversarial network

GEM gradient episodic memory

HPC high performance computing

IMM incremental moment matching

KL Kullback-Leibler

LC least confidence

MAD maximizing angular deviation

MAS memory aware synapses

MLaaS machine learning as a service

NNPD natural non-problem domain

PRADA protecting against DNN model stealing attacks

SGD stochastic gradient descent

SI synaptic intelligence

SNPD synthetic non-problem domain

VAAL variational adversarial active learning

VAE variational autoencoder

Notation

In this chapter, we introduce the notation used throughout this thesis. The variables introduced in the table below are used in the following chapters to describe the algorithms and experiments. They have the meaning assigned to them in the table below unless explicitly stated otherwise.

Symbol	Definition
x_i	The i^{th} data point in a dataset
y_i	The label of the given data point x_i, also known as $y(x_i)$
X	The set of data points in a dataset
Y	The set of labels in a dataset
b	The number of samples within a batch in active learning
U	The unlabeled pool in pool-based active learning
L	The labeled pool in pool-based active learning
$\mathcal{L}$	The loss function
P	Patterns per experience, a hyperparameter of the A-GEM algorithm
S	Samples drawn from memory to compute the reference gradients in A-GEM
θ	The parameters of a neural network
O	The oracle in active learning

1 Introduction

Research in deep learning has produced an abundance of highly influential work in recent years, like convolutional neural networks (CNNs) [10], transformers [11] and generative adversarial networks (GANs) [1]. Machine learning models are becoming more accurate, achieving or surpassing human-level performance in visual perception and natural language understanding. Because of their high innovation potential, machine learning models are increasingly being used in real-world applications. To profit from this, many technology enterprises offer services to ease training, development, and deployment of machine learning applications. These services can be grouped under the term MLaaS.

While MLaaS is crucial to increase the accessibility of machine learning, it also poses a security threat. In recent years, numerous research papers have been published that demonstrate how easily machine learning models can be extracted from MLaaS services [6, 7, 12]. The standard procedure to steal such a machine learning model is to train a local clone (also called substitute model) of the model hosted on the MLaaS service (also called target model). The substitute model is trained using a dataset collected by the attacker. This dataset is also called the thief dataset. Labels for the thief dataset are obtained by querying the target model on the thief dataset. The process of extracting machine learning models from MLaaS services is also called model stealing.

Because of the dramatic consequences of model stealing attacks, researchers have investigated how to defend against them. Two recent model stealing defenses are prediction poisoning [13] and protecting against DNN model stealing attacks (PRADA) [14]. Researchers have proposed attacks that evade these defense strategies, despite the effort to defend against model stealing attacks. One notable example is the model extraction framework ActiveThief [8], which successfully evades the defense strategy PRADA. ActiveThief utilizes active learning to determine which samples of the thief dataset it should query the target model on. Active learning is an intensively studied research field that aims to minimize the labeling effort in the training process of machine learning models. However, the problem with active learning is that it needs large amounts of computing resources. In this work, we extend the ActiveThief framework by using continual active learning in the model extraction process.

Continual learning is a research field that aims to make machine learning models more robust against the advent of new data. The major problem of the classic training procedure is that a model trained on a new task rapidly loses the ability to perform any previous tasks it was trained on. Research in continual learning aims to develop methods that allow machine learning models to learn new tasks without forgetting the knowledge of old tasks.

Delimitation We conduct all work in this thesis in the context of image classification. Furthermore, we conduct all experiments with CNNs. Therefore, our findings only apply to deep learning within the computer vision domain and may not generalize to other settings.

Contributions The objective of this thesis is to combine existing approaches from continual learning with approaches in the active learning domain. More specifically, we focus

on regularization-based continual learning methods, whereas we use uncertainty-based and diversity-based active learning methods. The continual learning methods we use are EWC [15], MAS [16], IMM [17] and ALASSO [18]. Concerning active learning, we compare random sampling, LC [19], BADGE [20], BALD [21] and CoreSet [22]. We analyze how the continual learning methods can speed up the active learning process, which combinations of continual learning and active learning methods are most effective, and the trade-off between accuracy and speed-up. While the focus of this thesis is on regularization-based continual earning methods, we briefly explore the effectiveness of exemplar rehearsal continual earning using A-GEM [23] combined with representation-based active learning in form of VAAL [24]. Furthermore, we evaluate the performance of a custom Replay strategy.

After exploring the effectiveness of the different combinations of continual and active learning methods, we apply these combinations in the model stealing domain. More specifically, we build upon the model extraction framework ActiveThief [8] and investigate the performance of continual active learning methods in the model stealing domain.

2 Background

In this chaper, we provide an overview of the background knowledge necessary to understand the following chapters. We start with active learning, describing common active learning settings. Next, we cover continual learning, including conventional continual learning scenarios and a taxonomy of continual learning methods. Finally, we describe model stealing attacks, introduce the most relevant terminology, and outline model stealing defenses.

2.1 Active Learning

Active learning is a subfield of machine learning that focuses on the problem of how to select the most informative data points to label. Research in active learning is motivated by the contrast between low-effort data acquisition and resource-intensive labeling. Therefore, the aim of active learning research is to create techniques that optimize model performance while minimizing the amount of labeled data required. A typical active learning scenario comprises a learner, an oracle, unlabeled samples, and labeled data. The learner itself is then the machine learning model. Let I be the instance space, i.e., the set of all possible data points, and L be the label space, i.e., the set of all possible labels. The oracle O represents the function

$$O : I \rightarrow L, x \mapsto y(x), \tag{2.1}$$

where $y(x)$ is the true label of the data point x. In the following, we will use y as a short form of $y(x)$. U, the set of unlabeled data, is a subset of I just as the set of labeled data L. In the beginning, there are no labeled data points. Often a few labeled data points are randomly sampled from U and labeled by the oracle to initialize the learning process. A detailed overview of research activities within the active learning domain is presented by Settles [25]. According to Settles, active learning methods can be divided into three categories:

- Query synthesis active learning
- Pool-based active learning
- Stream-based active learning

While this taxonomy is data-centric, it is worth noting that the type of machine learning model used, e.g., (convolutional) neural networks or support vector machines, and the type of data used, e.g., images or text, plays a decisive role in the effectiveness of an active learning strategy.

2.1.1 Query Synthesis Active Learning

Query synthesis active learning, also known as membership query synthesis active learning, was among the first active learning scenarios proposed [26]. Query synthesis methods synthesize

data points from the input space instead of sampling real data points. Modern query synthesis methods train a generative model, e.g., a GAN [27], which learns the distribution of unlabeled data. However, earlier works relied on statistical models such as Gaussian mixture models [28]. The generated queries are labeled by the oracle and can be used to train the machine learning model. Note that query synthesis is not limited to classification tasks. For example, Cohn et al. [28] proposed a method to predict the absolute coordinates of a robot hand when given the joint angles of its mechanical arms as inputs. When the oracle is a human annotator, query synthesis active learning researchers have encountered problems in labeling them. Human annotators struggled to assign any class to them in the survey of Baum et al. [29] because the generated queries do not show any class-discriminative features.

2.1.2 Pool-based Active Learning

Pool-based active learning is the most widely studied and used type of active learning. The idea behind pool-based active learning approaches is to iteratively select the most informative data points from the current unlabeled pool, query the oracle for their labels and add them to the labeled pool. Next, the machine learning model is trained on the current labeled pool. This process repeats itself until the query budget is exhausted. A more detailed explanation can be found in algorithm 1. More formally, the pool-based active learning problem can be defined as:

$$\min_{s^1 \subseteq U : |s^1| \leq b} E_{x,y \sim P_{(X,Y)}} [\mathcal{L}(x, y; L \cup b)], \tag{2.2}$$

where U is the unlabeled pool, b is the query batch size and L is the labeled pool. In other words, a pool-based active learning algorithm aims to select the points that minimize the expected loss of the model on the labeled pool when being added to it. As can be seen from algorithm 1, the structure across pool-based active learning strategies is similar. The main difference between these is the informative measure, i.e., the criterion with which they select the data points to label next. Within pool-based active learning, there are two subcategories: uncertainty-based sampling and diversity-based sampling. Uncertainty-based sampling strategies select the data points the model is most uncertain about. Diversity-based sampling strategies, on the other hand, aim to select data points that best represent the data distribution in the unlabeled pool.

Algorithm 1 Pool-based active learning

Input: unlabeled data U, labeled data $L = \emptyset$, oracle O, model M, budget B

1: Select k data points from U at random, obtain labels by querying O and set $L = \{x_1, \ldots, x_1\}$ and $U = U \setminus \{x_1, \ldots, x_1\}$ ▷ Initialization
2: Train M on initial labeled set L
3: **while** Label budget B not exhausted **do**
4: Select l data points from U predicted to be the most informative by the active learning strategy
5: Obtain labels by querying O for $x_i, \ldots, x_l$
6: Set $L = L \cup \{x_i, \ldots, x_l\}$ and $U = U \setminus \{x_i, \ldots, x_l\}$
7: Train M on labeled set L
8: **end while**

2.1.3 Stream-based Active Learning

Stream-based active learning, introduced by Cohn et al. [30], is closer to pool-based active learning than query synthesis active learning. The main difference between stream-based active learning and pool-based active learning is that data arrives sequentially. In the stream-based active learning scenario, the learner draws a data point from the data source one at a time. For each data point, the learner can then decide to query the oracle for its label or to discard it. The decision whether to label a data point can either be made based on its location within the instance space [30] or its informativeness [31]. In the former case, the learner would label the data point if it in a region of the instance space that the learner is not confident about.

2.2 Continual Learning

Continual learning is a subfield of machine learning that aims to solve the problem of catastrophic forgetting. Nowadays, most machine learning services are deployed in an environment where constant changes occur. To adapt to their environment, machine learning models need to learn new tasks without forgetting the knowledge they have acquired in the past. In contrast to human behavior, machine learning models rapidly decrease their performance on old task when trained on new ones. This phenomenon is known as catastrophic forgetting and was already discovered in the early days of machine learning research [32]. Generally speaking, research in continual learning aims not only to alleviate catastrophic forgetting but to solve the "stabiliy/plasticity dilemma" [33]. The stability/plasticity dilemma refers to the fact that machine learning models should be stable enough to retain their performance on old tasks while simultaneously being plastic enough to adapt to new ones. This is a dilemma because both properties are desirable even though they conflict with each other. In practice, machine learning models tend to be more plastic than stable. This is especially true for deep neural networks, but generally for all machine learning models trained by greedily updating their parameters using gradient descent [34].

Continual learning is employed widely in scenarios where new tasks arrive or the data distribution changes over time. Despite being useful in these classic scenarios, continual learning approaches can also be used in cases where storing data is impossible for legal reasons or due to memory constraints, i.e., when batch learning is inapplicable.

2.2.1 Continual Learning Scenarios

Continual learning is a rapidly evolving research field. Therefore, terminology, as well as taxonomy, are currently being established. Among the most important factors to distinguish is how new tasks arrive. In the following, we will introduce the three typical continual learning scenarios presented by van de Ven et al. [35]. These are task-incremental continual learning, domain-incremental continual learning, and class-incremental continual learning.

Task-Incremental Continual Learning

In the task-incremental setting, a model is informed about the task it will be trained on or the task the data whose label it is supposed to predict belongs to. Task information is delivered

through an integer task identifier. Because the model does not have to infer the task it is supposed to predict or learn, it is possible to have task-specific components in the model. For neural networks, this entails one output layer per task. The corresponding output layer is utilized for each task, while the remaining layers are shared among all tasks. These classifiers are known as multi-head classifiers [36]. An example would be the following: The first task is to classify images of cats and dogs. The second task is to classify cows and sheep. The model would have one output layer to classify cats and dogs and a second to classify cows and sheep. When training the first task, only the first output layer is used. Upon the arrival of the second task, the model is retrained with the second output layer. The mapping learned is

$$f : X \times T \rightarrow Y \tag{2.3}$$

Where $X = \mathbb{R}^{NxNxC}$ is the input space (i.e., all possible input images of size NxN with C channels), $T = \{1, 2, \ldots\}$ is the task space (i.e., all possible tasks that the model can be trained on) and $Y = \mathbb{Z}_k$ is the label space with k being the number of possible classes.

Domain-Incremental Continual Learning

In the domain-incremental setting, task identities are not passed to the classifier during evaluation. Since the underlying task does not change, this is not necessary either. While the structure of the task stays the same, the distribution of the data changes. An example of domain-incremental continual learning would be the classification of digits between zero and nine (such as those in the MNIST [37] dataset), where the digits for one subtask are green and blue for the other. The mapping learned is

$$f : X \rightarrow Y \tag{2.4}$$

Where $X = \mathbb{R}^{NxNxC}$ is the input space, and $Y = \mathbb{Z}_k$ is the label space with k being the number of possible classes.

Class-Incremental Continual Learning

Class-incremental continual learning is the most challenging scenario. Like in the domain-incremental setting, task identities are not provided at evaluation time. Instead, they need to be inferred by the model. In this scenario, a classifier is exposed to multiple tasks containing different classes. An example would be again the classification of digits between zero and nine. Contrary to the previous setting, each task comprises a disjoint subset of digits. The classifier would be trained on the first task, containing the digits zero and one. The second task would contain the digits two and three, and so on. During the evaluation, the classifier would have to classify the digits correctly and infer which task they belong to simultaneously. The mapping learned is

$$f : X \rightarrow Y \times T \tag{2.5}$$

Where $X = \mathbb{R}^{NxNxC}$ is the input space, $T = \{1, 2, \ldots\}$ is the task space (i.e., all possible tasks that the model can be trained on) and $Y = \mathbb{Z}_k$ is the label space with k being the number of possible classes.

2.2.2 Continual Learning Approaches

The continual learning approaches proposed so far can be grouped into three categories according to Mundt et al. [34], Parisi et al. [38] and Zenke et al. [39]. Parisi et al. propose to group continual learning approaches into regularization, rehearsal, and architectural approaches whereas Zenke et al. [39] group continual learning approaches into architectural, functional and structural approaches. In the following, we will stick to the categorization proposed by Parisi et al. because it is broader and fully encompasses the categorization by Zenke et al. Furthermore, Parisi et al.'s classification has been adopted by recent continual learning reviews [34].

Regularization Approaches

Regularization-based approaches to continual learning aim to prevent the forgetting of previous tasks by adding a regularization term to the model's loss function. The regularization term is a proxy for how much model performance on previous tasks will decrease. A high regularization term indicates that the model will perform poorly on the old tasks with the current weights. Consequently, a low regularization term indicates that the model has not lost much knowledge of the old tasks. Regularization-based continual learning approaches are further divided into two subgroups based on the method by which the regularization term is calculated.

There are structural approaches that regularize based on weight changes to the model, and there are functional approaches that regularize based on model output. Structural approaches determine an importance weight for each model parameter. Popular approaches either use Fisher information [15, 17] or gradient information [16, 39]. The idea of structural regularization approaches is visualized in figure 2.1.

Functional regularization approaches are inspired by knowledge distillation [40]. They add a distillation loss to the objective function, which is computed based on the prediction of a data sample stored for future use. These data samples are called soft targets. Li et al. [41] compute the distillation loss by using the output of the newly arrived task given by the model trained on the old tasks. The distillation loss they introduce aims to retain the prediction of the old model on the new task, even if the prediction itself may be inaccurate. Another approach uses autoencoder reconstructions of old tasks to compute the distillation loss [42].

Rehearsal Approaches

Rehearsal approaches aim to prevent catastrophic forgetting by fitting a model's parameters to the distribution of an incoming task and all previous tasks. Within rehearsal approaches, a trade-off between model performance and computational cost exists. Generally, using more data from previous tasks to train the model will increase the accuracy. However, to train on more data, the data has to be stored and fed into the training process, which is costly in terms of memory. The amount of training data can be increased by replaying stored data from previous tasks or by retraining on generated data drawn from the distribution of previous tasks. Continual learning methods that rely on the former idea are categorized as exemplar rehearsal approaches, while those relying on the latter idea are generative replay approaches.

Exemplar rehearsal techniques store data from previous tasks in a so-called replay buffer [23]. When training on a new task T_N, samples from the replay buffer are drawn and fed to

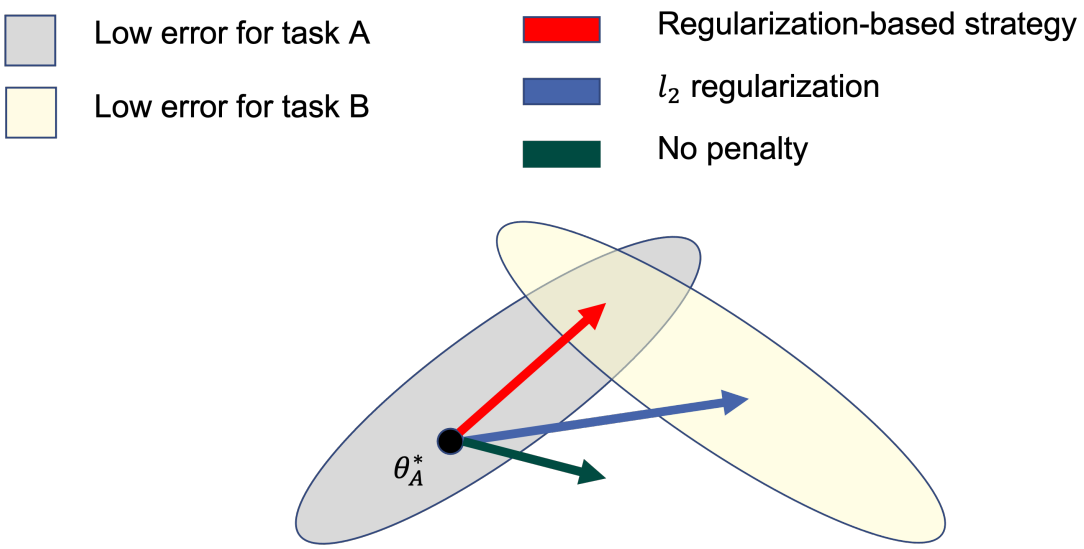

Figure 2.1: Idea of regularization-based continual learning. Compared to l_2 regularization, regularization-based continual learning strategies apply dynamic regularization to the model's parameters with the aim of finding model weights which perform well on multiple tasks. The figure was inspired by Kirkpatrick et al. [15].

the model while training on T_N. Approaches falling in this category vary by replay buffer construction and by the way samples are replayed during training of a new task.

Generative replay approaches use generative models to generate artificial samples from previous tasks [43]. These samples are then replayed to the model when training on a new task T_N. Contrary to exemplar rehearsal approaches, generative replay approaches do not require a replay buffer. However, the generative model itself needs to be stored.

Architectural Approaches

Architectural approaches aim to tackle continual learning by changing the underlying model architecture. Within this category, there are two subcategories. Methods falling into the first category are fixed capacity methods, while the second category is called dynamic growth.

Fixed capacity methods do not change the number of model parameters. Instead, they change the parameters used for each task. They assume that a subset of the model's parameters is redundant and reusable for different tasks. When training on a new task T_N, those parameters that are important for the task (how importance is defined depends on the specific approach) are used to form a path through the model only used for T_N. All other parameters are frozen, i.e., their weight and bias are set to zero.

On the other hand, dynamic growth approaches allow adding new layers or single neurons to the model when training on a new task. The difference between approaches from this category is their decision on when to add new layers or neurons. A popular criterion for deciding when to add new model parameters is when the loss function plateaus [44].

2.3 Model Stealing

With the advent of MLaaS, numerous machine learning models are exposed to the public via prediction APIs. The idea of these prediction APIs is that a user can request a prediction from a model by sending a request containing an unlabeled data point to the API. The response to the request is the model's prediction for the data point. Prediction APIs are monetized by charging the user on a pay-per-query principle. That means each request has a fixed price for each query, which is subtracted from the account balance every month. Providing public access to a machine learning model is a win-win situation. The model developer receives compensation for his efforts in gathering and labeling appropriate training data. The developer also chooses a proper model architecture, trains the model, and fine-tunes its hyperparameters. On the other hand, the user of the prediction API can benefit from the model's predictions without having to train the model.

Since the developed model is the intellectual property of the model developer, it is crucial that only the model's predictions are exposed to the public and not the model (i.e., its architecture and the model weights) itself. However, in the last years, numerous research papers have been published that demonstrate how to extract multiple features of a machine learning model, i.e., its training data [4], its architecture [5] and its functionality [7]. This newly created field of research is called model stealing or model extraction, and it is a subfield of adversarial machine learning according to Oliynyk et al. [45]. Because model extraction attacks are so effective, further research concerning model stealing defense mechanisms has been published. In the following, we will use the taxonomy provided by Oliynyk et al. [45] as a basis for our categorization of model stealing attacks and defenses. We present an overview of this taxonomy in figure 2.2.

2.3.1 Terminology

Since model stealing is a fairly new field it comes with numerous terms that are not commonly used in other fields of machine learning and others that are used synonymously. To avoid confusion, we will define and explain the terms we use in this thesis.

Model Stealing Attack: The process of maliciously querying a machine learning model to extract some or all of the model's features, such as its architecture, its weights or its training data. Model stealing attacks are also known as model extraction attacks or model inference attacks among others.

Target Model: The model queried via the prediction API and whose features the attacker aims to extract. The target model is also referred to as oracle, victim model, or secret model.

Substitute Model: The model used by the attacker to extract the target model's features. Synonyms for substitute model are adversarial classifier, knockoff model or inferred model.

Target Model Dataset: The dataset that the target model has been trained on. The term target model dataset has not been used in the literature before (to the best of our knowledge). However, we choose this term over the synonymous term secret dataset because we believe it is more expressive. During model stealing attacks, the attacker has no access to the target model's dataset. For research purposes, however, a test set of the target model dataset is used to evaluate the performance of the substitute model.

Thief Dataset: The dataset used by the attacker to train the substitute model. The thief dataset is also known as fake dataset, attacker set or transfer set. When constructing the thief dataset, the attacker can either use natural non-problem domain (NNPD) or synthetic non-problem domain (SNPD) data. NNPD data is data that is collected from the real world but does not (knowingly) overlap with the target model dataset, whereas SNPD data is artificially created, usually by drawing from a given distribution such as a multidimensional uniform distribution [8]. Empirical observations show that using NNPD data yields significantly better results than using SNPD data. Even when using NNPD data, significant differences between different datasets have been observed [8]. In general, it is wise to choose a dataset that is as diverse as possible. Regardless of using SNPD or NNPD data, the adversary should keep the dimensions of the target model dataset, i.e., image size and especially the number of channels for image datasets, in mind and adapt the thief dataset accordingly.

Agreement: The relative agreement between the predictions of the substitute model and the target model. The agreement is computed on a test set of the target model dataset, meaning that it is not accessible by the attacker. The agreement is also known as extraction accuracy, label prediction match, or similarity. More formally, the agreement between target model f and substitute model $\tilde{f}$ is defined as follows:

$$Agreement(f, \tilde{f}) = \frac{1}{|X_{tm}^{test}|} \sum_{x \in X_{tm}^{test}} \mathbb{1}(f(x) = \tilde{f}(x)), \tag{2.6}$$

where X_{tm}^{test} is the test set of the target model dataset and $\mathbb{1}$ is the indicator function.

2.3.2 Model Stealing Attacks

Model stealing attacks can be performed in a variety of ways. The method used to execute such an attack depends on the attacker's goal. If the attacker aims to extract training hyperparameters, they can use an equation-solving approach [46] or train a meta-model to predict the hyperparameters [5]. Oh et al. 's meta-model approach can also be used to extract the model architecture. It is the only approach that does not rely on either software or hardware access to the target model. Other methods, such as the one proposed by Yan et al. [47], use side-channel attacks to extract the architecture of the target model and therefore require software or hardware access to the target model.

Apart from extracting training hyperparameters or model architecture, the attacker might also be interested in the target model's weights, especially if the model architecture and/or training hyperparameters are already known. These can be apparent because they are not hidden but also by using the aforementioned attacks to extract them. We highlight that different model stealing attacks are not mutually exclusive but rather a toolkit that, while often used individually, can be combined to extract a holistic view of the target model. So far, attacks have been proposed to extract the weights of simple neural networks [7], logistic regression [7], support vector regression [12] and a binary classifier [48].

Many attackers are not interested in specific model features, such as architecture or (hyper-) parameters, but rather in reproducing the target model's functionality, i.e., its ability to predict a label for a given input. When stealing model functionality, attackers either aim to maximize the accuracy of the substitute model on the target model's test set or maximize the agreement

between the predictions of the substitute model and the target model. While Oliynyk et al. list these as separate attacks, we believe it is rather a matter of the optimization objective. Regardless of whether the attacker aims to maximize the accuracy or agreement, he has to train a substitute model and choose an appropriate thief dataset. Empirical observations have shown that choosing a rather complex model architecture for the substitute model yields better results if the model architecture is unknown, whereas choosing a model architecture that is as similar to the target model's architecture as possible yields better results if the model architecture is known [8, 9].

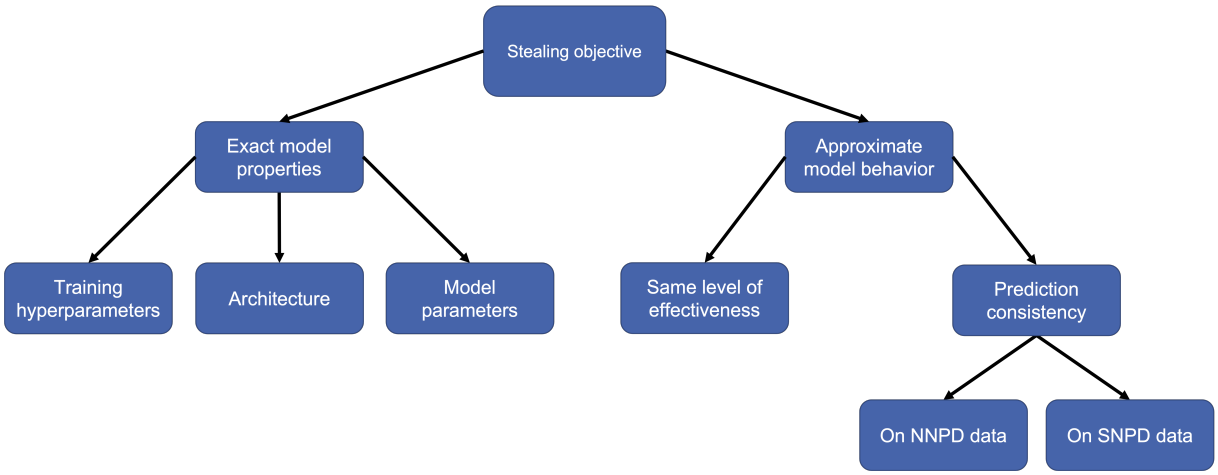

Figure 2.2: Taxonomy of model stealing attacks by Oliynyk et al. [45].

2.3.3 Model Stealing Defenses

Because of the detrimental effect of model stealing attacks, many defenses have been proposed to protect against them. We divide defenses into proactive and reactive defenses. While proactive defenses actively aim to prevent or at least complicate model stealing attacks, reactive approaches purely aim to detect them.

Reactive Model Stealing Defenses (Detection)

Reactive model stealing defenses aim to detect model stealing attacks, not prevent them. So far, model stealing detection approaches rely either on watermarking or monitoring. Watermarking is a way to prove ownership of a machine learning model. This is usually achieved by "hiding" information in the model which can only be restored by the original owner. A watermark can be a predefined prediction value for outlier samples that are unlikely to be part of the thief dataset. Watermarking can be applied during training [49] or afterward [50]. Monitoring is a way to detect model stealing attacks by investigating a user's queries to the target model. Kesarwani et al. [51] proposed a method that estimates the relative amount of the data space a user has covered with his queries. Based on this metric, they can infer the status of a possible extraction attack. Another approach named PRADA [14] analyses the distribution of the samples that

a user has queried so far. They assume that the distance between queries of a benign user follows a normal distribution. Consequently, any user whose query distances are not normally distributed is considered an attacker.

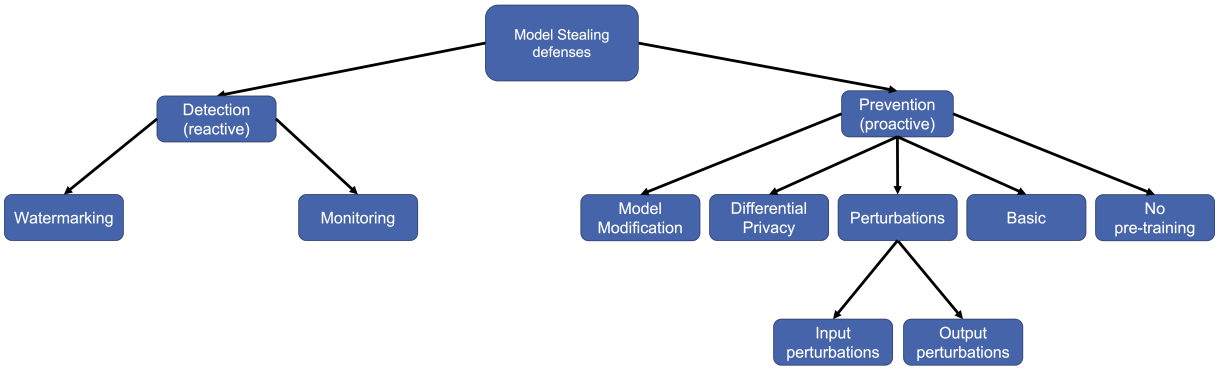

Figure 2.3: Taxonomy of model stealing defenses by Oliynyk et al. [45].

Proactive Model Stealing Defenses (Prevention)

Proactive model stealing defenses aim to complicate or prevent model stealing attacks. We highlight that, compared to breaking into a house, the success of a model stealing attack is not binary. Model stealing attacks will always be successful to some degree. However, their success mainly depends on the accuracy the substitute model can achieve (or the achieved model agreement in case the attacker tries to optimize for that). So the quality of a proactive model stealing defense should be measured by how much it can decrease the accuracy of the substitute model compared to a model stealing attack with no defense. A key challenge in developing proactive model stealing defenses is that extracting the target model should be challenging, on the one hand, and the user experience of benign users should not be affected on the other hand.

Early on, a few basic approaches against model stealing attacks were proposed. These include training several models and randomly choosing the output of one of them [52]. An even simpler albeit effective approach is to output only the predicted label instead of the vector of output probabilities [7]. A downside to this is that a benign user loses a lot of information about the output, and the model is less interpretable. Another approach of similar complexity is renouncing using pre-trained models [53] and instead training the target model from scratch.

To prevent model stealing attacks, researchers have proposed to use data perturbation. One can perturb the data fed to the target model or its prediction. Input perturbation can be performed by adding noise to unimportant pixels determined via gradient class activation maps [54]. To perform output perturbation, the developer of the target model can either use maximizing angular deviation (MAD) [13] or flip a few labels to obfuscate the model's decision boundary [55].

Furthermore, a few approaches which bring differential privacy into the model stealing domain have been proposed. Differential privacy is a way to protect the privacy of a dataset so that it is impossible for an adversary to determine if a particular individual is part of the dataset.

One of these approaches was proposed by Zheng et al. [56], who add a so-called boundary differential privacy layer to the target model.

The final type of proactive model stealing defense is modifying the target model's architecture. Approaches falling into this category add redundant layers to the target model [57], propose novel model layers [58], or increase model sensitivity [59].

3 Related Work

We present related work for this thesis in this chapter. First, we outline the basic idea of each active learning strategy we use in our experiments. Next, we give an overview of the continual learning approaches we use. Finally, we introduce ActiveThief, the model stealing framework we extend in this thesis.

3.1 Active Learning

In this section, we present the active learning strategies we used in our experiments. These are least confidence (LC), CoreSet, bayesian active learning by disagreement (BALD), batch active learning by diverse gradient embeddings (BADGE), and variational adversarial active learning (VAAL). While we aim to give a detailed explanation of these approaches, we refer to the original papers for any further details.

3.1.1 Least Confidence

Least confidence (LC) is one of the earlier approaches to active learning. LC is an uncertainty-based active learning approach proposed by Lewis et al. [19]. The idea behind LC is that the next points to label should be the ones with the lowest prediction probability. More formally, the data queried will be the following:

$$\underset{x \in U}{\operatorname{argmin}} \max_{y \in Y} p(y|x) \tag{3.1}$$

where U is the set of unlabeled data points and Y is the set of possible labels.

3.1.2 CoreSet

CoreSet, sometimes named k-center strategy, is an active learning method specifically proposed for CNNs [22]. Sener and Savarese redefine the pool-based active learning problem (for the original definition, see section 2.1.2) as

$$\min_{s^1 \subseteq U : |s^1| \leq b} \left| \frac{1}{|L \cup U|} \sum_{x_i, y_i \in L \cup U} \mathcal{L}(x_i, y_i; L \cup s^1) - \frac{1}{|L \cup s^1|} \sum_{j \in L \cup s^1} \mathcal{L}(x_j, y_j, L \cup s^1) \right|, \tag{3.2}$$

where L is the labeled set, U is the unlabeled set and b is the batch size. Informally, we select the data points to add to the labeled pool, which ensure the best loss approximation of a model

trained on the labeled set compared to a model trained on the whole dataset. The authors assume zero training error for the core-set, which simplifies the equation above to

$$\min_{s^1 \subseteq U : |s^1| \leq b} \frac{1}{|L \cup U|} \sum_{x_i, y_i \in L \cup U} \mathcal{L}(x_i, y_i; L \cup s^1). \tag{3.3}$$

Sener and Savarese show that minimizing this equation is equivalent to solving the k-Center problem [60], i.e.,

$$\min_{s^1 : |s^1| \leq b} \max_i \min_{j \in s^1 \cup L} \Delta(x_i, x_j). \tag{3.4}$$

This problem is NP-hard but can be solved by the 2-OPT greedy algorithm presented in algorithm 2. The authors propose a more sophisticated algorithm that performs marginally

Algorithm 2 k-Center-Greedy

Input: Data x_i, labeled pool L and budget b
  Initialize s = L
  repeat
    $u = \text{argmax}_{x_i \in U \setminus s} \min_{x_j \in s} \Delta(x_i, x_j)$
    $s = s \cup \{u\}$
  until $|s| = b$
return $s \setminus L$

better than the greedy solution but is approximately four times slower. Therefore, we omit the proposed algorithm in this section.

3.1.3 Bayesian Active Learning by Disagreement

Bayesian active learning by disagreement (BALD) is an uncertainty-based active learning strategy proposed by Houlsby et al. [21]. BALD uses Shannon's entropy [61] to determine a model's prediction uncertainty for a given unlabeled data point. More precisely, a learner using BALD will query the sample(s) fulfilling the following condition

$$\underset{x \in U}{\text{argmax}}\, H[y \mid x, L] - \mathbb{E}_{\theta \sim p(\theta \mid L)}[H[y \mid x, \theta]] \tag{3.5}$$

where H is the entropy, U is the set of unlabeled data points, and L is the set of labeled data points. Informally, BALD selects those data points with the largest gap between the model's actual prediction uncertainty and the model's expected prediction uncertainty. Since the expected prediction uncertainty cannot be computed given that the parameter distribution conditioned on the previously observed data is unknown, BALD uses Monte Carlo sampling to approximate the expected prediction uncertainty. For neural networks, Monte Carlo dropout [62] can be applied.

3.1.4 Batch Active Learning by Diverse Gradient Embeddings

Batch active learning by diverse gradient embeddings (BADGE) [20] is a batch active learning strategy that combines both model uncertainty and batch diversity to select the next batch

of data points to label. The authors were inspired to develop BADGE by the observation that diversity-based active learning strategies perform well in the first few iterations, while uncertainty-based approaches perform well in the later stages of active learning. BADGE therefore incorporates both uncertainty and diversity in its selection strategy. Furthermore, BADGE uses the gradient with respect to the output layer as a measure of model uncertainty, similar to the ideas of Zhang et al. [63] and Settles et al. [64]. Since BADGE works on unlabeled data points, the gradient cannot be computed analytically because the label is unknown. Therefore, BADGE uses the hypothetical label $\hat{y}(x)$ as a proxy for the true label and computes the gradient of the cross-entropy loss on the hypothetical label as an approximation of the actual gradient. To incorporate diversity, BADGE uses k-means++ seeding [65] for the gradient embeddings to sample the next batch of data points to label. The authors demonstrate BADGE's agnosticism towards model architecture, training hyperparameters, and datasets in mutliple experiments.

Algorithm 3 BADGE

Input: Neural network $f(x;\theta)$, unlabeled pool U, labeled pool $L = \emptyset$, number of iterations T, batch size b

1: Labeled dataset $L \leftarrow k$ random samples from U together with their labels
2: Train initial model θ_1 on L
3: **for** $t = 1, 2, \ldots, T$ **do**
4: For all examples x in $U \setminus L$:

1. Compute its hypothetical label $\hat{y}(x) = h_{\theta_t}(x)$
2. Compute gradient embeddings $g_x = \frac{\partial}{\partial \theta_{out}} \mathcal{L}_{\text{CE}}(f(x;\theta_t), \hat{y}(x))$, where θ_{out} refers to parameters of the final (output) layer.

5: Compute S_t, a subset of $U \setminus L$ of size b, using the k-MEANS++ seeding algorithm on $\{g_x : x \in U \setminus L\}$ and query for their labels.
6: Update labeled pool $L \leftarrow L \cup S_t$
7: Train new model θ_{t+1} on L
8: **end for**

return Final model θ_T

3.1.5 Variational Adversarial Active Learning

Variational adversarial active learning (VAAL) is a representation-based active learning strategy proposed by Sinha et al. [24]. VAAL uses a β-variational autoencoder (VAE) [66] and a discriminator to select the most informative samples to query. Thereby, the encoder learns a low-dimensional space for the distribution of the labeled and the unlabeled data reconstructed by the decoder. The discriminator is trained to distinguish between the labeled and the unlabeled data and given the encoder's latent representation of a data point as an input. We provide a visual summary of the approach in figure 3.1.

More formally, the encoder learns the distribution of X_U and X_L where X_U is the unlabeled pool and (X_L, Y_L) is the labeled pool. The encoder aims to minimize the Kullback-Leibler (KL) divergence [67] between the reconstructed distribution $q_\Phi(z_L|x_L)$ and the real distribution

$p(z)$ of the labeled data, which we assume to be a unit Gaussian. Simultaneously, the encoder aims to minimize the KL divergence between $q_\Phi(z_U|x_U)$ and $p(z)$. This results in the following objective function for the VAE:

$$\mathcal{L}_{VAE}^{trd} = \mathbb{E}[\log p_\theta(x_L|z_L)] - \beta D_{KL}(q_\Phi(z_L|x_L)||p(z)) + \mathbb{E}[\log p_\theta(x_U|z_U)] - \beta D_{KL}(q_\Phi(z_U|x_U)||p(z)) \tag{3.6}$$

where p_θ and q_Φ are the decoder and the encoder. The discriminator D is trained to distinguish between the latent space representation of labeled and unlabeled data. Additionally, the VAE is trained to make samples from the labeled and unlabeled data indistinguishable to complicate the classification of the discriminator. The adversarial objective function for the discriminator is given by:

$$\mathcal{L}_{VAE}^{adv} = -\mathbb{E}[\log(D(q_\theta(z_L|x_L)))] - \mathbb{E}[\log(D(q_\theta(z_U|x_U)))]. \tag{3.7}$$

Both equations above result in the following final loss function for the VAE:

$$\mathcal{L}_{VAE} = \lambda_1 \mathcal{L}_{VAE}^{trd} + \lambda_2 \mathcal{L}_{VAE}^{adv} \tag{3.8}$$

where λ_1 and λ_2 are hyperparameters, trading off between adversarial training and optimal reconstruction.

The discriminator loss is given by:

$$\mathcal{L}_D = -\mathbb{E}[\log(D(q_\theta(z_L|x_L)))] - \mathbb{E}[\log(1 - D(q_\theta(z_U|x_U)))]. \tag{3.9}$$

To query a batch of samples of size b, the VAE and discriminator are first trained on the entire pool of labeled and unlabeled data for a pre-specified number of epochs e. The samples chosen by VAAL are the ones where the discriminator is most confident about their belonging to the unlabeled pool or, in other words, least confident about their belonging to the labeled pool.

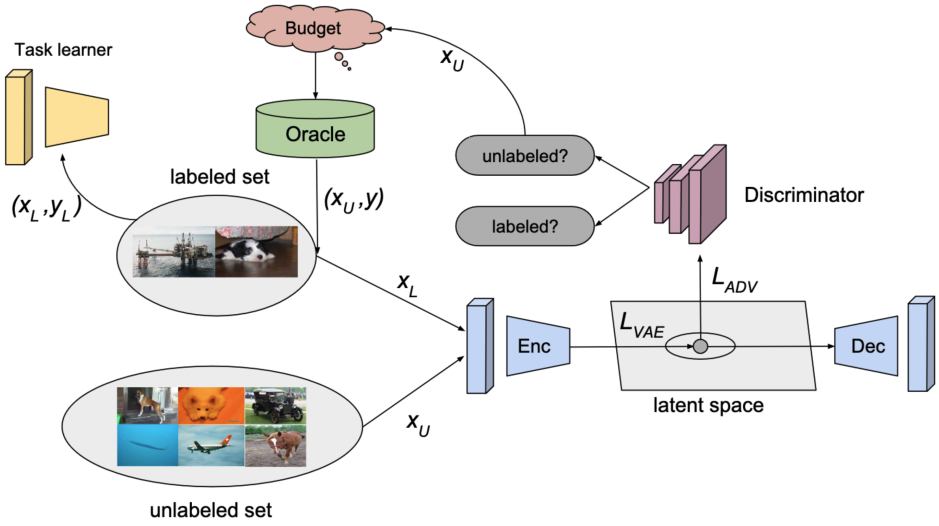

Figure 3.1: Illustration of the idea of VAAL taken from the paper of Sinha et al. [24].

3.2 Continual Learning

In this section, we present the continual learning algorithms used in our experiments. These are elastic weight consolidation (EWC) [15], memory aware synapses (MAS) [16], asymmetric loss approximation by single-side overestimation (ALASSO) [18], incremental moment matching (IMM) [17] and averaged gradient episodic memory (A-GEM) [68]. While we aim to give a detailed explanation of these approaches, we refer to the original papers for any further details.

3.2.1 Elastic Weight Consolidation

Elastic weight consolidation (EWC) [15] is a structural regularization approach proposed by Kirkpatrick et al. EWC was the first approach aiming to overcome catastrophic forgetting by regularizing model parameters. This is done by adding a regularization term to the loss function, which resembles l_2 regularization. The regularization loss is introduced to constrain important parameters for task $N-1$ to stay close to their previous values when training on task N. To come up with an importance measure for the parameters, Kirkpatrick et al. view neural network training from a probabilistic perspective, modeling the weight distribution using the Bayes rule:

$$p(\theta \mid D) = \frac{p(D \mid \theta) \cdot p(\theta)}{p(D)}. \tag{3.10}$$

When applying the logarithm, this can be rewritten as

$$\log p(\theta \mid D) = \log p(D \mid \theta) + \log p(\theta) - \log p(D), \tag{3.11}$$

and when splitting the whole dataset D into $D_{1:N-1}$ and D_N, this equates to

$$\log p(\theta \mid D) = \log p(D_N \mid \theta) + \log p(\theta \mid D_{1:N-1}) - \log p(D_N), \tag{3.12}$$

meaning that only the term $\log p(\theta \mid D_{1:N-1})$ is dependent on the previous tasks. Therefore, it must contain the full information about all previous tasks. Since $\log p(\theta \mid D_{1:N-1})$ cannot be computed, Kirkpatrick et al. approximate it using Laplace's approximation [69], obtaining a Gaussian distribution with mean of $\theta^*_{1:N-1}$ and covariance matrix $[\mathbb{I}_{D_{1:N-1}}]^{-1}$, where

$$\mathbb{I}_{D_{1:N-1}} = \mathbb{E}[-\frac{\partial^2 \theta(\log(p(\theta \mid D_{1:N-1})))}{\partial^2 \theta} \mid_{\theta^*_{D_{1:N-1}}}] \tag{3.13}$$

which is the Fisher information matrix (FIM), as shown by Aich [70]. A large value of F_i, the i^{th} diagonal of the FIM, indicates a small variance in the posterior distribution of the parameter, meaning that it is important for previous tasks. Therefore, F_i, is used as an importance measure for the i^{th} parameter. To conclude, the loss function for a task N is altered to

$$\mathcal{L}(\theta) = \mathcal{L}_B(\theta) + \sum_i \frac{\lambda}{2} F_i \cdot (\theta_i - \theta^*_{N-1,i})^2, \tag{3.14}$$

where $\mathcal{L}_B(\theta)$ is the loss function of the model on the current task (e.g. cross-entropy loss), θ_i is the i^{th} parameter of the model, and $\theta^*_{N-1,i}$ is the i^{th} parameter of the model trained on the previous task. The hyperparameter λ is used to trade-off between learning new tasks and retaining knowledge of previous tasks.

3.2.2 Memory Aware Synapses

Memory aware synapses (MAS), like EWC, is a structural regularization method proposed by Aljundi et al. [16]. In line with EWC, MAS aims to prevent catastrophic forgetting by regularizing model parameters. The main difference between EWC and MAS the composition of their importance measure. While EWC relies on Fisher information to compute parameter importances, MAS uses gradient magnitude to estimate the importance of a parameter for previous tasks. To derive the importance measure, the authors first observe that the effect of small changes to the network parameters on the prediction can be approximated as follows

$$f(x_k; \theta + \delta) - f(x_k; \theta) \approx= \sum_i g_i(x_k) \cdot \delta_i \tag{3.15}$$

where $g_i(x_k) = \frac{\partial(f(x_k;\theta))}{\partial \theta_i}$ is the gradient with respect to parameter θ_i when evaluating the model at input x_k. Since most classification models, like neural networks, for example, have a multi-dimensional output Aljundi et al. propose to use the gradient of the squared l_2 norm in this setting, i.e., $g_i(x_k) = \frac{\partial[l_2^2(f(x_k;\theta))]}{\partial \theta_i}$. With the approximation given by the equation above, the authors claim that the importance of a parameter θ_i can be computed as

$$\Omega_i = \frac{1}{n} \sum_{k=1}^{n} \|g_i(x_k)\| \tag{3.16}$$

where n is the number of data points observed so far. The advantage of computing the importance measure as stated above is that it can be updated when- ever a new data point is observed. However, it is more efficient to update the Ωs batch-wise. The overall loss function for a task N is then altered for the standard setting as follows

$$\mathcal{L}(\theta) = \mathcal{L}_N(\theta) + \lambda \sum_i \Omega_i \cdot (\theta_i - \theta^*_{N-1,i})^2 \tag{3.17}$$

where λ again is the hyperparameter used to trade-off between learning new tasks and retaining the knowledge of previous tasks.

3.2.3 Asymmetric Loss Approximation by Single-Side Overestimation

Asymmetric loss approximation by single-side overestimation (ALASSO) is a structural regularization method proposed by Park et al. [18]. ALASSO is an extension of synaptic intelligence (SI) proposed by Zenke et al. [39]. Park et al. claim that ALASSO "mitigates the limitations of SI". The main observation they made about SI is that it underestimates the true loss on the unobserved side of the loss function because it assumes that the loss function is symmetric. Park et al. first show that this assumption is not always correct through empirical observation and then present a new approach to approximate the loss function on the unobserved side. We present the fundamental observation and idea of ALASSO in figure 3.2. Since ALASSO extends the SI framework, we strongly recommend familiarizing with SI before trying to understand ALASSO. In the following, we will assume that the reader is familiar with SI and only explain the extension that ALASSO brings to SI.

Since ALASSO is a structural regularization method like MAS and EWC, it alters the loss function as follows

$$\tilde{\mathcal{L}}^N = \mathcal{L}^N + c \sum_k \mathcal{L}_s^{N-1}(\theta_k, a), \tag{3.18}$$

meaning that it adds a surrogate loss to the loss function, which is controlled via the hyperparameter c. To understand how the surrogate loss is composed ALASSO first introduces $\alpha(\theta_k)$, which determines for a model parameter if it is on the observed or unobserved side of the loss function. $\alpha(\theta_k)$ is defined as follows

$$\alpha(\theta_k) = (\theta_k - \hat{\theta}_k^N)(\hat{\theta}_k^{N-1} - \hat{\theta}_k^N) \tag{3.19}$$

The surrogate loss $\mathcal{L}_s^N(\theta_k, a)$ is then defined as follows

$$\mathcal{L}_s^n(\theta_k, a) = \begin{cases} \hat{\Omega}_k^N(\theta_k - \hat{\theta}_k)^2 & \text{if } \alpha(\theta_k) > 0 \\ (a\hat{\Omega}_k^N + \epsilon)(\theta_k - \hat{\theta}_k)^2 & \text{if } \alpha(\theta_k) \leq 0 \end{cases}. \tag{3.20}$$

Again, the variable $a(> 1)$ is a hyperparameter used to control the amount of overestimation if the parameter is on the unobserved side of the loss function. The constant ϵ is used to make sure that the loss on the unobserved side is always overestimated, i.e., $(a\hat{\Omega}_k^N + \epsilon) > \hat{\Omega}_k^N$. The second contribution of ALASSO is its derivation of the optimal coefficient $\hat{\Omega}_k^N$ for the approximation of the loss function on the *observed* side. According to Park et al., $\hat{\Omega}_k^N$ is determined by the following equation

$$\hat{\Omega}_k^N = \frac{\omega_k^N + \omega_k^{1:N-1}}{(\hat{\theta}_k^N - \hat{\theta}_k^{N-1})^2}, \tag{3.21}$$

where ω_k^N and $\omega_k^{1:N-1}$ are given by

$$\omega_k^N = \mathcal{L}^N(\hat{\theta}_k^{N-1}) - \mathcal{L}^N(\hat{\theta}_k^N) \tag{3.22}$$

$$\omega_k^{1:N-1} = -c\mathcal{L}_s^{N-1}(\hat{\theta}_k^N, a) \tag{3.23}$$

Because of the ambiguous influence of the hyperparameters c and a to the approximation of the loss function, the authors of ALASSO propose to decouple them. More precisely, they suggest using c' and a' instead of c and a in equation 3.23.

3.2.4 Incremental Moment Matching

Incremental moment matching (IMM) is a structural regularization approach, proposed by Lee et al. [17]. IMM deviates from approaches like MAS and EWC in that it should not be viewed as a single method but rather a framework of multiple methods, many of which can be combined. The main idea of IMM is to match the first and second moments of the posterior distribution $p(\theta \mid D_{1:N})$ of the model parameters given all tasks up to the current one. As in EWC, the posterior distribution cannot be computed. Therefore, it is approximated via a Gaussian distribution which we call $q_{1:N}$. The first approach to matching these moments is called mean-based incremental moment matching, also known as mean-IMM. The weight

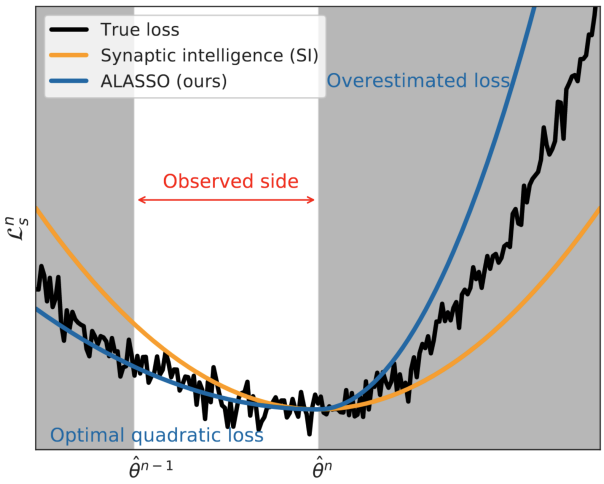

Figure 3.2: Illustration of the idea of ALASSO taken from the paper of Park et al. [18]. The authors of ALASSO state that, SI underestimates the unobserved side of the loss function and hence a better approximation of the loss function is achieved by overestimating it.

update of mean-IMM is the analytical solution of the problem to minimize the weighted sum of KL divergences between each q_i and $q_{1:N}$ [67], i.e. the solution to

$$\underset{\mu^*_{1:N},\Sigma^*_{1:N}}{\operatorname{argmin}} \sum_{i=N-k}^{N} \alpha_i \cdot KL(q_i \,||\, q_{1:N}) \tag{3.24}$$

which is given by

$$\mu^*_{1:N} = \sum_{i=N-k}^{N} \alpha_i \cdot \mu_i \tag{3.25}$$

$$\Sigma^*_{1:N} = \sum_{i=N-k}^{N} \alpha_i (\Sigma_i + (\mu_i - \mu^*_{1:N})(\mu_i - \mu^*_{1:N})^T) \tag{3.26}$$

The α_i are the mixing coefficients that weigh the previous k tasks and have the constraint $\sum_{i=N-k}^{N} \alpha_i = 1$.
The second approach to matching the moments is called mode-based incremental moment matching, also known as mode-IMM. mode-IMM further incorporates covariance information from previous tasks to match the first and second moments of the posterior distribution. The mean and covariance update for mode-IMM is given by

$$\mu^*_{1:N} = \sum_{i=N-k}^{N} \alpha_i \cdot \Big(\sum_{i=N-k}^{N} \alpha_i \Sigma_i^{-1} \mu_i \Big) \tag{3.27}$$

$$\Sigma^*_{1:N} = \Big(\sum_{i=N-k}^{N} \alpha_i \Sigma_i^{-1} \Big)^{-1} \tag{3.28}$$

To approximate the covariance matrix, mode-IMM uses the inverse of the FIM like EWC. Furthermore, the authors assume that the model parameters are pairwise independent, making the covariance matrix diagonal and therefore saving computation.

Apart from moment matching methods, IMM also includes approaches to transfer model parameters from previous tasks. The first approach is called weight transfer. When using continual learning with weight transfer, the parameters of the previous task are used as initialization for the current task. This approach is very similar to the term warm start commonly used in active learning. l_2-transfer, which is a special form of l_2 regularization, is the next transfer technique. l_2-transfer can be seen as a special form of EWC where all the F_i are set to one. With l_2-transfer, the loss function is altered to

$$\log p(y_N \mid X_N, \mu_N) - \lambda \cdot \|\mu_N - \mu_{N-1}\|_2^2 \tag{3.29}$$

with λ being a hyperparameter. The third transfer technique is drop-transfer. Drop-transfer

$$\hat{\mu}_{N,i} = \begin{cases} \mu_{N,i}, & \text{if the } i\text{th node is turned off} \\ \frac{1}{1-p} \cdot \mu_{N,i} - \frac{p}{1-p} \cdot \mu_{N-1,i}, & \text{otherwise} \end{cases} \tag{3.30}$$

where p is the dropout ratio. Drop-transfer can be seen as a further regularizer for continual learning, with similar effect to l_2-transfer albeit being orthogonal to it. In algorithm 4, we show the pseudocode for IMM with weight-transfer and l_2-transfer.

Algorithm 4 IMM with weight-transfer, l_2-transfer

Input: data $f\{(X_1, Y_1), \ldots, (X_N, Y_N)\}$, balancing hyperparameter α with $\sum_{i=1}^{k} \alpha_i = 1$, regularization hyperparameter λ
return $w_{1:N}$
  $w_0 \leftarrow$ InitializeNN()
  for $i = 1 : N$ **do**
    $w_{i*} \leftarrow w_{i-1}$
    Train(w_{i*}, X_i, Y_i) with $L(w_{i*}, X_i, Y_i) + \lambda \cdot (\|w_{i*} - w_{i-1}\|)_2^2$
    $m = \max(0, i - k)$
    if type is mean-IMM **then**
      $w_{i*} \leftarrow \sum_{t=max(0,i-k)}^{i} \alpha_t w_t$
    else if type is mode-IMM **then**
      $F_{i*} \leftarrow$ CalculateFisherMatrix(w_{i*}, X_i, Y_i)
      $\Sigma_{1:i} \leftarrow (\sum_{t=max(0,i-k)}^{i} \alpha_t F_{t*} w_{t*})^{-1}$
      $w_{i*} \leftarrow \Sigma_{1:i} \cdot (\sum_{t=max(0,i-k)}^{i} \alpha_t F_{t*} w_{t*})$
    end if
  end for

3.2.5 Averaged Gradient Episodic Memory

Averaged gradient episodic memory (A-GEM) is an exemplar rehearsal approach to continual learning, proposed by Chaudhry et al. [23]. A-GEM is based on gradient episodic memory (GEM)

[68], aiming to make it more efficient while maintaining similar or even better performance. We recommend reading the original paper for a more detailed explanation of GEM.

Like GEM, A-GEM uses a so-called episodic memory M for all previous tasks $M_k (k < N)$. A-GEM aims to avoid catastrophic forgetting by ensuring that the average loss for the previous tasks does not increase while simultaneously aiming to minimize the loss on the current task. Formally, A-GEM proposes a solution to the following objective

$$\text{minimize}_\theta \ \mathcal{L}(f_\theta, T_N) \text{ s.t. } \mathcal{L}(f_\theta, T) \leq l(f_\theta^{N-1}, T) \text{ where } T = \bigcup_{i<N} T_i, \tag{3.31}$$

where f_θ^{N-1} is the model trained for the previous task. The optimization problem for this objective is given as

$$\text{minimize}_{\tilde{g}} \ \frac{1}{2}\|g - \tilde{g}\|_2^2 \text{ s.t. } \tilde{g}^T g_{ref} \geq 0 \ \forall k < N, \tag{3.32}$$

where g_{ref} is a gradient computed by randomly sampling from the episodic memory of all past tasks. The optimization problem has the solution

$$\tilde{g} = g - \frac{g^T g_{ref}}{g_{ref}^T g_{ref}} g_{ref}, \tag{3.33}$$

which is the gradient update used by A-GEM.

When running A-GEM, we need to decide on two hyperparameters: The first one is S, the number of samples from the episodic memory M to compute g_{ref}. The second one is P, the number of patterns (or data points) stored in the episodic memory after each task.

3.3 Model Stealing

In this section, we will present related work from the model stealing domain. This consists of ActiveThief [8], a model stealing framework using active learning.

3.3.1 ActiveThief

ActiveThief is a novel model stealing approach proposed by Pal et al. [8]. Inspired by prior theoretical [71] and practical work [72], they use active learning to choose which samples to query the target model. Using active learning in the model stealing domain provides three major benefits: First, it makes it easier for the attacker to create a thief dataset. Data is abundant nowadays, whereas labeling large amounts of data remains a time-intensive task. By using active learning, which works on unlabeled data, creating a thief dataset is sped up immensely. Second, active learning aims to query the most informative samples, yielding the most performant model with as little data as possible. Third, by querying the target model with samples selected by active learning, the model stealing attack can successfully be disguised as a benign query process. Pal et al. demonstrate this by showing that ActiveThief successfully evades PRADA [14], a state-of-the-art model stealing defense which detects attacks by analyzing the distribution of distances between queries.

The authors of ActiveThief evaluate their framework using the active learning strategies Random, Uncertainty [19], CoreSet [22], deepfool active learning (DFAL)[73] and a custom combination of DFAL and CoreSet. They use a custom CNN architecture for both the target and substitute model and show that ActiveThief can successfully steal the target model for multiple benchmark datasets such as MNIST and CIFAR-10. A visualization of the model stealing workflow proposed by Pal et al. is shown in figure 3.3.

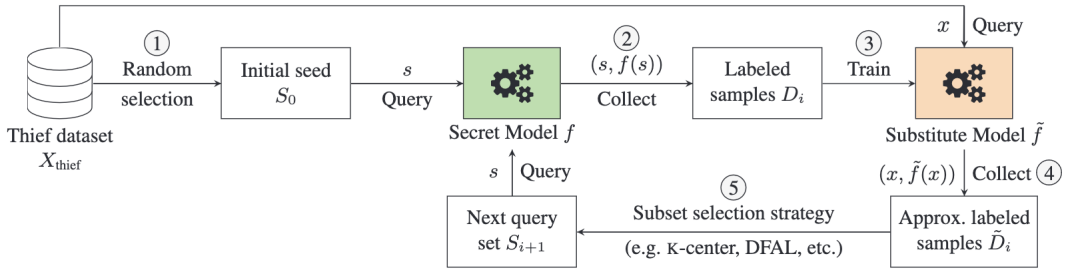

Figure 3.3: Illustration of the model stealing workflow proposed by Pal et al. [8]. Inspired by Chandrasekaran et al. [71], Pal et al. use active learning to select the next samples to query the target model.

4 Methodology

Since our continual active learning approach is, to the best of our knowledge, the first approach to combine pool-based active learning with continual learning, we explain it in detail in this chapter, including its motivation. We then transfer our approach to the model stealing domain. Because we build upon the framework of ActiveThief, we describe how our method compares to the original one in detail.

4.1 Continual Active Learning

The main contribution of this thesis is combining the two learning paradigms continual learning and active learning. To motivate the idea of combining these two paradigms, we will first outline the classic continual learning setting and the classic active learning setting. Next, we explain common issues with these two learning paradigms and how we aim to overcome these by combining both paradigms. Finally, we describe a custom Replay strategy, which we use in our experiments.

4.1.1 Classic Continual Learning Setting

In the typical continual learning setting, the model is trained on a sequence of tasks. Each task $T_i = \{(x_k, y_k) | k \in \{1, \ldots, n\}\}$ is a set of instances with their respective label. Together, the tasks form a dataset $D = \bigcup_{i=1}^{N} T_i$. Note that the distributions of two distinct tasks $P(T_i)$ and $P(T_j)(i \neq j)$ are not necessarily the same. Often the tasks are independent, which is why neural networks struggle to perform well on multiple tasks simultaneously. This also means that the size of two distinct datasets can be different, i.e., $|T_i| \neq |T_j|$ and so can the number of classes $|\{y_k | \exists x_k : (x_k, y_k) \in T_i\}| \neq |\{y_l | \exists x_l : (x_l, y_l) \in T_j\}|$. When training a model on a sequence of tasks, the model is first fed with the data of the first task T_1 and then trained on it. After the model was trained on the first task, it can either be trained on the next task or deployed to classify samples stemming from the distribution of the first task. Next, the model is trained on the second task. After being trained on the second task, the model should now be able to classify samples from the distribution of the first and second tasks. This process repeats until the model has been trained on all tasks. At the end of the process, the model should be able to classify samples following the distribution of all the tasks it was trained on. This workflow is illustrated in figure 4.1.

The main difference between the continual learning setting and classic machine learning is that in the classic machine learning setting, the model is not retrained after deployment. In the continual learning setting, however, the model is retrained whenever a new task arrives.

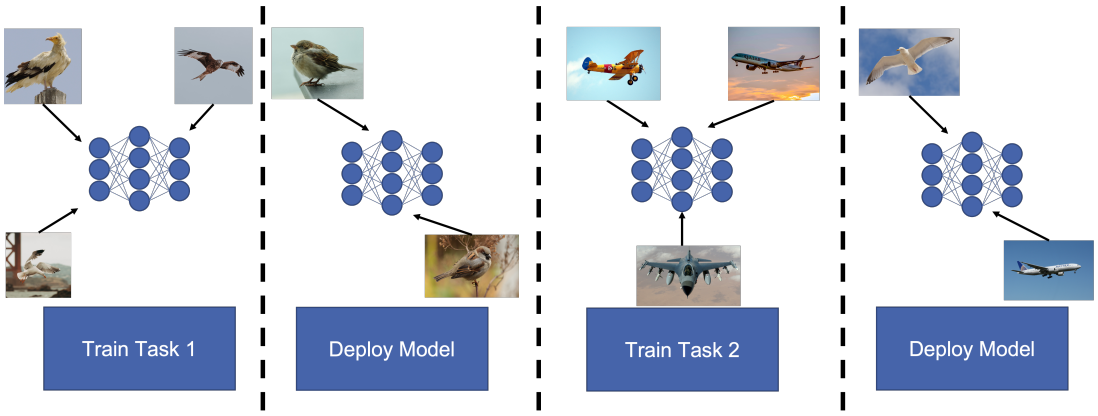

Figure 4.1: Example for the classic continual learning workflow. In this example, the model is first trained on different species of birds. It is then deployed to differentiate these species. Next, the model is trained on planes. After being deployed again, the model should now differentiate the planes as well as the birds.

4.1.2 Pool-based Active Learning Setting

In the pool-based active learning setting, the model is trained sequentially on the current labeled pool. At first, the labeled pool is empty. Next, it is initialized by randomly selecting samples from the unlabeled pool to label. On the contrary, the unlabeled pool contains the complete dataset at first and is emptied in the process. After initializing the labeled pool, the model is trained on it. Next, b samples from the labeled pool are selected until the total budget is exhausted. The selected samples are the ones determined to be the most informative by the current active learning strategy. The oracle labels them, and then they added to the labeled pool. Next, the model is trained on the current labeled pool, and the process repeats. We provide a more detailed description of pool-based active learning in section 2.1.2.

Active learning contrasts with continual learning because all data stems from the same task. Another difference between pool-based active learning and continual learning is that a model trained by active learning is trained on all previously selected batches. Consequently, the model is trained on some data multiple times, which might cause parts of the training to be redundant.

4.1.3 Combining Continual and Active Learning

The problem with classic active learning is that it is very resource intensive. When training a model using pool-based active learning on a dataset of size n, with batch size b, the model will be trained $\frac{n}{b}$ times on the current labeled pool, equating to $\frac{n(n+b)}{2b}$ data points overall (we provide a short derivation of this formula in A.1). The problem with the number of data points used for training is that it is dependent on the batch size b. The smaller the batch size, the larger the overall number of data points that the model is trained on. In the extreme case of $b = 1$, the model is trained on $\frac{n(n+1)}{2}$ data points. While [74] note that the batch size has a negligible effect on the model performance, a typical batch size is less than ten percent of the complete training set. Even in this more realistic case, the model is trained on $5.5n$ data points. When comparing this to the classic continual learning setting, where the model is trained once using

n data points, it is clear that active learning comes with considerable overhead. The overhead of active learning is even more pronounced when considering the execution time of the active learning algorithms. For more details, we refer to chapter 6 and 7.

On the other hand, a massive problem with classic continual learning is that task order significantly impacts the model performance [75]. Hacohen et al. studied the problem of task ordering previously and showed that training samples in decreasing order of difficulty results in faster learning and improved generalization error [76]. This motivates us to use active learning to perform task ordering. We should note here that we assume the free choice of the next task. This assumption is not always realistic, especially when tasks arrive sequentially, but studying the effect of task-ordering benefits the study of these scenarios, too. Furthermore, insights into task-ordering help towards a more rigorous evaluation of continual learning in research because they allow us to assess the influence of task-ordering on experiment results when using classic benchmark datasets.

Our approach aims to overcome the overhead of active learning and the issue of task ordering by combining both learning paradigms. We modify the active learning process by training only on the currently selected batch instead of the entire labeled pool. This way, the model is trained on $\sum_{i=1}^{\frac{n}{b}} b = n$ data points, which fully eliminates the overhead of active learning from a data-centric perspective. Nevertheless, the query time of the active learning algorithm remains an overhead. From a continual learning perspective, we join all tasks to a single dataset and use this dataset as the unlabeled pool for active learning. In each iteration of the active learning process, we select a batch B from the unlabeled pool using the given active learning strategy. Next, the oracle labels this batch. We then treat the current batch B as a new task and train our model using only the data points of B with the continual learning strategy of choice. This process repeats until the unlabeled pool is empty. The full algorithm is described in algorithm 5, where we highlight the difference between classic pool-based active learning and our new continual active learning approach in bold.

Algorithm 5 Pool-based continual active learning

Input: Unlabeled data U, Labeled data $L = \emptyset$, Oracle O, Model M, budget B

1: Select k data points from U at random, obtain labels by querying O and set $L = \{x_1, \ldots, x_1\}$ and $U = U \setminus \{x_1, \ldots, x_1\}$ ▷ Initialization
2: Train M on initial labeled set L
3: **while** Label budget B not exhausted **do**
4: Select l data points from U predicted to be the most informative by the active learning strategy
5: Obtain labels $y_i, \ldots, y_l$ by querying O for $x_i, \ldots, x_l$
6: **Train M on current labeled batch** $\{(\mathbf{x_i}, \mathbf{y_i}), \ldots, (\mathbf{x_l}, \mathbf{y_l})\}$
7: Set $L = L \cup \{x_i, \ldots, x_l\}$ and $U = U \setminus \{x_i, \ldots, x_l\}$
8: **end while**

4.1.4 Replay strategy

In this subsection, we describe a continual learning strategy named Replay. Applying Replay to overcome catastrophic forgetting is not an entirely new idea. On the contrary, Robins et al. first

proposed using Replay in the 1990s [77]. The approach we describe modifies the classic Replay by compressing the replay buffer. To understand the modification, we must first understand the classic Replay strategy.

Replay is a continual learning strategy that stores a subset (or all) data points from each task in a so-called replay buffer. When training on a new task T_N, the model is trained on the data from the current task plus a sample from the replay buffer. After training on task T_N, the replay buffer is updated with the data points from the current task.

A significant drawback of the classic Replay strategy is that the replay buffer can grow to extreme sizes over time. This is especially true when the replay buffer stores all data points from each task. It is more desirable to have a continual learning strategy with a fixed memory footprint because this reflects real-world applications of continual learning where memory is limited.

Our proposed modification of the Replay strategy is to compress the replay buffer after each task. Assume we have a replay buffer of size n. After training on task T_N, the replay buffer is combined with the data points from the current task to form a new set of data points P. We then select n data points from P to create the new replay buffer. These points are selected by performing one iteration of the active learning strategy CoreSet [22] with P as the unlabeled pool and n as the batch size. The n data points selected by CoreSet then represents the new replay buffer. When training on task T_{N+1}, the model is trained on the data from the current task plus the full replay buffer.

4.2 Continual Active Learning for Model Stealing

In this section, we describe how we apply the continual active learning approach to model stealing. Using the continual active learning approach in the model stealing domain allows us to determine if continual active learning achieves similar performance in the model stealing domain compared to the standard setup where the oracle returns truthful labels. We motivate transferring our approach to the model stealing domain by highlighting the differences between the setup mentioned in 4.1.3 and continual active learning for model stealing.

The first difference is that the labels returned by the oracle in the model stealing domain have a limited semantic meaning. This is because the oracle is another machine learning model. So even if the data point whose label is queried stems from the same distribution as the target model dataset, the label returned by the oracle is not necessarily correct since machine learning models do not generalize perfectly. Since the target model dataset is unknown to the attacker, he cannot knowingly construct a thief dataset that overlaps with the target model dataset. Therefore, the data points the attacker queries the target model with will most likely be from a different distribution than the target model dataset and the labels returned by the target model will be incorrect. The example in figure 4.2 highlights the meaning of the associated label. In the given example, the target model is trained to classify planes, which is why it associates the label “Fighter Jet” with a sparrow.

The second difference between the standard continual active learning approach and continual active learning for model stealing lies again in the labels returned by the oracle. In the classic continual active learning approach, the labels of the oracle are not only truthful, but the label is just a single value, i.e., the respective class. In the model stealing domain, the label returned

by the oracle is either a single value or the per-class probabilities of the output layer of the target model. The latter is the more interesting case for continual active learning because it reveals more information about the function learned by the target model.

Next, we describe how we apply continual active learning for model stealing. The first step is to use the thief dataset as the initial unlabeled pool for active learning. The active learning strategy uses the thief dataset in each iteration, just as in classic continual active learning. After selecting the most informative samples to label, the target model is queried for the labels of the selected samples. Next, the label returned, which is either a single value or the per-class probabilities of the output layer of the target model, is used as the label of the respective data point throughout the complete training process. This means that the gradient updates during training are based on the loss computed with a softmax label, allowing for a more fine-grained optimization of the weights to approximate the target model function. This process is repeated until the unlabeled pool is empty. A visual example of continual active learning for model stealing is shown in figure 4.2.

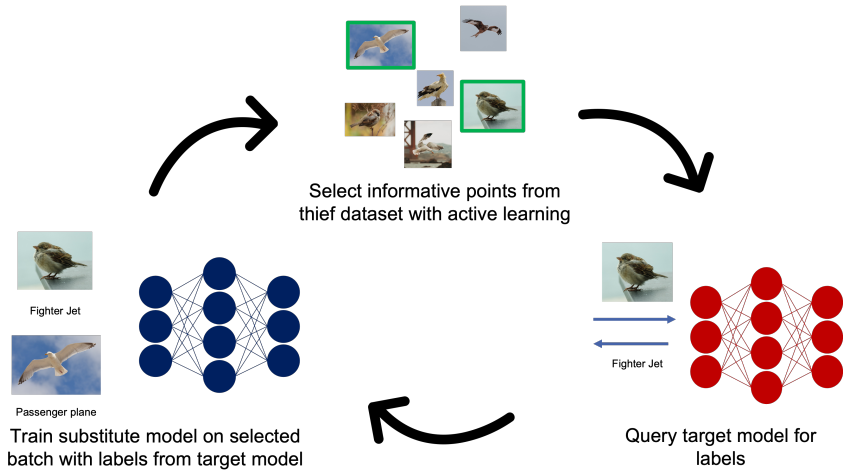

Figure 4.2: Example of continual active learning for model stealing. In this example, the thief dataset consists of birds, while the target model was trained to classify planes. In each iteration, a batch of informative samples is selected by the active learning strategy. Next, the target model is queried for the labels of the selected samples. Since our thief dataset is composed of NNPD data, the associated labels have an incorrect semantic meaning. The thief model is then trained on the selected samples from the current batch. This process is repeated iteratively.

5 Experiment Setup

The findings of this thesis rely heavily on thorough experimentation. To reproduce our experiments, we specify the conditions under which we conducted the experiments. This includes training hyperparameters, datasets, neural network architectures, and information about the code and libraries used. In general, we divide our experiments into two categories: Experiments that explore the classic continual active learning setting as in section 4.1.3 and experiments that explore continual active learning for model stealing as in section 4.2. We perform this differentiation because many training hyperparameters, datasets, and neural network architectures depend on whether we evaluate classic continual active learning or continual active learning for model stealing. First, we mention the general setup, including hardware, software, datasets, and the training hyperparameters shared between the two categories of experiments. Next, we list the specific configuration for continual active learning and continual active learning for model stealing. Finally, we mention the evaluation metrics used in our experiments.

5.1 General Experiment Setup

In this section, we describe the general experiment setup. The parameters described in this section are shared across all experiments, unless explicitly stated otherwise.

5.1.1 Hardware

All experiments are conducted on the high performance computing (HPC) cluster bwUniCluster 2.0 [78]. bwUniCluster 2.0 is an HPC cluster funded by the Ministry of Science, Research and the Arts Baden-Württemberg and the Universities of the State of Baden-Württemberg. It currently consists of more than 840 compute nodes with each node falling one of the following categories: “Thin”, “HPC”, “IceLake”, “Fat”, “GPUx4”, “GPUx8”, “GPUx4 A100”, “GPUx4 H100” and “Login”. In our experiments, we use the nodes GPUx4, GPUx8 and GPUx4 A100. We list their hardware specifications in appendix A.2.2. Furthermore, we conduct experiments where we measure runtime on the GPUx8 nodes.

5.1.2 Software and Libraries

All the code used in this thesis is written in version 3.9.4 of the programming language Python [79]. Furthermore, we use the deep learning library PyTorch [80] (version 1.13.1) to implement both active and continual learning algorithms. All further libraries and their respective versions can be found in appendix A.2.2.

5.1.3 Continual Learning Strategies

In our experiments, we use the continual learning strategies EWC, MAS, IMM, ALASSO, A-GEM and our custom Replay strategy proposed in section 4.1.4. For details on these continual learning strategies, we refer to section 3.2. As a baseline, we used the naive strategy of performing classic gradient descent without regularization. In the following, we will refer to this strategy as Naive.

Following the paper introducing MAS [16], we set the regularization parameter λ to 1.0 for MAS. The choice of this parameter is crucial because it enables a sound evaluation. If we trivially chose to set λ to 0, there would be no difference between MAS and the naive strategy. On the other hand, fine-tuning is neither possible (we use more than 25 combinations of continual learning strategies and active learning strategies on three datasets) nor fair (to evaluate the quality and effect of all continual learning strategies, they should be tested under the same conditions). We, therefore, set λ in EWC and IMM to 1.0 accordingly. The regularization parameter c in ALASSO, which is equivalent to λ in EWC, MAS, and IMM is set to 0.5 for all experiments. We tried setting c to 1.0, however, we noticed that this led to divergence during gradient descent despite employing heavy gradient clipping. Since employing stronger gradient clipping was not a viable solution, because it impedes model convergence, we decided to relax the regularization parameter c to 0.5.

As mentioned, we employ gradient clipping to ensure convergence of gradient descent. Not only is gradient clipping a remedy against the exploding gradient problem, but Zhang et al. have demonstrated, that it can accelerate the training process [81]. In our implementation, we clip gradients by their l_2 norm. Across all continual learning procedures, including Naive, we clip the gradients to a maximum l_2 norm of 20.0 to accelerate the training process. When conducting experiments with MAS and ALASSO we encountered exploding gradient problems, which we further investigated for MAS and ALASSO separately. While we aimed to eliminate the exploding gradient problem, which is mitigated more by clipping smaller gradients, we did not want to restrict the model too much to hinder model convergence. After carefully exploring different values to clip at, we found setting the threshold for the l_2-Norm to 2.0 to be effective, mitigating the exploding gradient problem while simultaneously enabling model convergence.

Apart from the parameter weighting of the regularization term, the continual learning strategies do not share any hyperparameters. EWC does not have any further hyperparameters, and neither does MAS. On the other hand, for IMM, we have the choice between mean-IMM and mode-IMM. Furthermore, we can choose to apply weight-transfer, l_2-transfer, and dropout-transfer, and we can choose values for the parameter α. In our setup, we use mean-IMM, weight-transfer and l_2-transfer and set the α parameter to [0.45,0.55] (i.e. $\alpha_1 = 0.45, \alpha_2 = 0.55$), as suggested by the authors. Both EWC and IMM use Fisher information to determine parameter importance. To compute the FIM, we use five percent of the training set of the current task. Regarding ALASSO, we set the parameter a, which controls the overestimation on the unobserved side, to 3.0. Following the authors' recommendation, we perform parameter decoupling for the Ω updates. Therefore, we set a' to 1.5 and c' to 0.25. For A-GEM, we set S, the number of samples drawn from the episodic memory to compute the reference gradients, to 2,000. The second hyperparameter, P, which controls the number of patterns from a task saved to the memory, is also set to 2,000.

We provide a detailed summary of the hyperparameters used in appendix A.2.3.

5.1.4 Active Learning Strategies

In our experiments, we use the active learning strategies BADGE, LC, CoreSet, BALD, and VAAL. For details on these active learning strategies, we refer to section 3.1. For BADGE, we use the Euclidean distance in the k-means++ algorithm. For CoreSet, we use the Euclidean distance between the activations of the penultimate layer of the neural network as a distance metric for k-Center algorithm. For BALD, we use the Monte Carlo dropout with $T = 25$ samples. We omit Monte Carlo dropout in cases where the model does not contain dropout layers, as the prediction of such a model is deterministic. LC does not contain hyperparameters. For VAAL, we train VAE and discriminator for 20 epochs per iteration. We use the neural network architecture and training hyperparameters from the GitHub repository published by the authors along with the paper [82].

Across all active learning strategies, we use the same initial budget, i.e., the number of points we sample from the training set before the first iteration of the active learning algorithm. In our experiments, we set the initial budget to be the same as the batch size.

5.1.5 Datasets

In our experiments, we use the datasets MNIST [37], CIFAR-10 [83], CIFAR-100 [83], Tiny ImageNet [84], and a subset of the ILSVRC2012-14 dataset [85]. The subset of the ILSVRC2012-14 dataset we use is the first of ten training batches downloaded from [86]. From now on, we will refer to this subset of the ILSVRC2012-14 dataset as Small ImageNet. For all datasets, we use the standard train test split, as proposed by PyTorch. We rescale all images to 32x32 pixels, because for the success of a model stealing attack, the image shape of the thief dataset has to match the image shape accepted by the target model. We will cover this in more detail in section 5.3. Moreover, we normalize the train and test split of all datasets by using the mean and standard deviation of the training set. On all datasets, apart from MNIST, we apply data augmentation. We use random horizontal flips with a probability of 0.5 followed by random cropping. We provide a detailed list of the datasets used in appendix A.2.4.

5.1.6 Neural Network Architectures

We use the neural network architectures ResNet18 [2] and CNN architectures from the ActiveThief paper [8] in our experiments. In the following, we will refer to this CNN architecture as “ActiveThiefConv”. Since there are multiple variations of the architecture, more specifically ones with two, three, or four convolutional blocks, we will refer to those as ActiveThiefConv2, ActiveThiefConv3, and ActiveThiefConv4, respectively. We present the layout of the three ActiveThiefConv architectures in appendix A.2.5.

5.1.7 Training Hyperparameters

We use PyTorch’s stochastic gradient descent (SGD) optimizer with a learning rate of 0.1 across all experiments. Furthermore, we schedule the learning rate by a factor of 0.1. The number of epochs after which we schedule the learning rate differs between experiments. Furthermore, we re-instantiate the SGD optimizer after each active learning iteration. We will go into detail

on this in the description of the respective experiments. Apart from scheduling the learning rate, we use momentum [87] of 0.9 and l_2 regularization of 0.0005. For all experiments, we use a batch size of 128 and shuffle the entire training set before each epoch.

5.2 Special Setup for Continual Active Learning

We experiment with continual active learning using ResNet18 as our neural network architecture and CIFAR-10 as our dataset. We perform experiments with a batch size of 1,000, 2,000 and 4,000, respectively. Regardless of the batch size, we conduct active learning until the unlabeled pool is exhausted. This means that we perform active learning for 49, 24, and 12 iterations, respectively. For batch sizes 1,000 and 2,000, each query consists of 1,000 and 2,000 points, respectively, while the last query for batch size 4,000 consists of 2,000 points. For batch sizes 2,000 and 4,000, we train for 150 epochs per iteration, decaying the learning rate by ten after 80 and 120 epochs, respectively. For batch size 1,000, we train for 80 epochs and decay the learning rate by ten after 60 epochs. To compare the performance of continual active learning with active learning, we computed the results for active learning with identical batch sizes. In this experiment setup, we use warm start and train for 200 epochs in each iteration. Here, we decay the learning rate by ten after 100 and 150 epochs, respectively.

5.3 Special Setup for Continual Active Learning for Model Stealing

When transferring continual active learning to the model stealing domain, we change our setup compared to continual active learning previously. Instead of using ResNet18 as our model architecture, we use the CNN architecture from the ActiveThief paper [8], mainly because we want to compare our continual active learning approach to the framework proposed by the authors of ActiveThief. For most of our experiments, we use ActiveThiefConv3 apart from experiments investigating the effect of the model architecture on model stealing attacks. We train the target models on the datasets CIFAR-10, CIFAR-100, and MNIST. We train the target models on CIFAR-10 and CIFAR-100 for 150 epochs using a momentum of 0.9 and l_2 regularization of 0.0005 without learning rate decay. When training the target models on MNIST, we change the number of epochs to 50. We omit to retrain the target model for all model stealing attacks. Instead, we train one target model per dataset, save them to the disk and load them for all model stealing attacks. This way, we reduce the uncertainty in our experiments introduced by different weight initializations of the target model. We use Small ImageNet as our thief dataset for all model stealing attacks.

Similar to the classic continual learning setting, we compute a baseline with active learning for model stealing. For this baseline, we use active learning with a batch size of 1,000 and a total budget of 20,000 points. The models are trained using cold start for 200 epochs per iteration. We decay the learning rate by ten after 100 and 150 epochs.

For continual active learning for model stealing, we use a batch size of 2,000 and a total budget of 20,000. We decided to increase the batch size for continual active learning compared to pure active learning because we noticed a clear correlation between batch size and model performance when training using continual active learning. We train the substitute model for

150 epochs per iteration and decay the learning rate by ten after 80 and 120 epochs with a momentum of and l_2 regularization of 0.0005.

5.4 Evaluation Metrics

We describe and motivate the metrics for the evaluation of continual active learning in this section. Since we use continual active learning in two distinct domains, we will discuss the metrics for each domain separately. We will start with the metrics for the classic continual learning setting and then move on to the metrics for the model stealing domain.

5.4.1 Metrics for Continual Active Learning

When choosing a metric to evaluate continual active learning, we should remember why we proposed the approach. The aim of continual active learning is to mitigate the overhead of active learning compared to the classic training loop. Therefore, it is desirable to incorporate a metric that measures this goal. In our case, this is the overall execution time. However, execution time is not the only metric we want to use to evaluate the approach. Since the second goal of the continual learning approach is to improve model performance, we will also evaluate the experiments based on this. What makes a machine learning model performant has been discussed vividly in the literature [88]. In our case, we will use the validation accuracy as a metric to evaluate model performance. When we evaluate the experiments, we need to consider *both* runtime and validation accuracy. A model with low runtime and validation accuracy is not desirable, but neither is a model with high runtime and validation accuracy. Since we are trying to improve active learning by introducing continual learning into the pool-based active learning process, we aim to outperform the classic pool-based active learning approach. More precisely, we want to reduce the runtime and increase the validation accuracy. We will evaluate the success of our continual active learning approach against the classic pool-based active learning approach based on the runtime improvement and the accuracy improvement.

5.4.2 Metrics for Continual Active Learning for Model Stealing

As presented in section 2.3.2, model stealing attacks can be performed for multiple reasons. As we build our framework on ActiveThief, we adopt the metrics used in the original paper. The ActiveThief framework aims purely at stealing the model functionality, i.e., approximating the model's relationship between input and output best, and so does ours. Therefore, we will use the same evaluation metric, the agreement between the substitute and target model, computed on the validation split of the target model dataset. For an explanation of the model agreement metric, we refer the reader to section 2.3.1. The baseline which we compare our results against is the ActiveThief framework. We will therefore measure the success of using continual active learning for model stealing by comparing the model agreement of our approach to the model agreement of the ActiveThief framework.

6 Evaluation

In this chapter, we present the results of our experiments. Our experiments are split into two parts. The first part is the evaluation of our novel continual active learning approach. In the second part, we evaluate the performance of our continual active learning approach when applied to model stealing. In each of these respective subsections, we will first describe the experiments schedule for each part and then present the results of the experiments.

6.1 Continual Active Learning

In this section, we evaluate the results of our experiments using continual active learning in its classic setting. First, we experiment with regularization-based continual learning strategies. Next, we analyze the performance of our custom replay strategy from section 4.1.4. Finally, we test the combination of exemplar rehearsal continual learning and representation-based active learning.

6.1.1 Regularization-based Continual Learning

We start our experiments by running each combination of the active learning strategies BALD, BADGE, CoreSet, LC and Random with the continual learning strategies Naive, EWC, MAS, ALASSO and IMM. We use the dataset CIFAR-10, the neural network architecture ResNet18 and a batch size of 4,000 for these experiments. For each experiment, we present the validation accuracy with the increasing size of the labeled pool and the overall execution time. The results by validation accuracy and execution time are shown in figure 6.1 and table 6.1, respectively.

First, we evaluate the results for random sampling. In terms of execution time, there is a large gap between the baseline and the continual learning strategies. IMM is the fastest method, followed by MAS, EWC and Naive who perform on the same level. ALASSO is the slowest continual learning strategy, albeit being approximately six times as fast as the baseline. In terms of validation accuracy, the baseline outperforms all continual learning strategies by at least ten percentage points. Naive, EWC and IMM demonstrate a similar accuracy progression, with EWC and Naive marginally outperforming IMM. MAS has a higher validation accuracy than the three aforementioned strategies at first but fails to keep up with their continuous increase in validation accuracy. ALASSO starts with a higher validation accuracy than the other continual learning strategies but falls behind at around 8,000 samples due to exploding gradients.

Next, we re-run the previous experiment using LC. In terms of execution time, the results are similar to the experiment with random sampling. ALASSO is again the slowest strategy, followed by MAS, EWC, IMM and Naive. All continual learning strategies are significantly faster than the baseline, with ALASSO being about six times as fast, and Naive about ten

times as fast. The gap in validation accuracy between the baseline and the continual learning strategies remains significant and increases during the last 25,000 samples. IMM, EWC, and Naive perform almost identically across the experiment, with MAS following closely and outperforming the remaining strategies within the last 5,000 samples. The four aforementioned methods experience a decrease in validation accuracy caused by the inferior representativeness of the final batches. ALASSO starts competitively but suffers from a heavy decrease in validation accuracy at around 8,000 samples, which is due to exploding gradients.

The third experiment is run with the active learning strategy BALD. The ranking of the execution time of the continual learning strategies and the baseline is similar to the previous experiments with Random and LC. In terms of accuracy, the baseline outperforms the continual learning strategies significantly. The gap between validation accuracy is most prominent at roughly 15,000 samples and in the final batch. Naive, IMM and EWC perform similarly, showing an s-shaped validation accuracy curve. The performance of MAS follows a similar curve. However, MAS performs better in the first half of the experiment, and worse in the second half compared to the three former strategies. Since we use ResNet18, which does not contain dropout layers, we are unable to run Monte Carlo dropout to accurately estimate $\mathbb{E}_{\theta \sim p(\theta|L)}[H[y \mid x, \theta]]$. This leads to the inferior performance of BALD compared to the other active learning strategies. ALASSO starts as the best continual learning strategy, but its validation accuracy decreases steadily throughout the experiment, which is again caused by exploding gradients.

We now run the experiment with the identical setup using CoreSet. The gap in execution time between the baseline and the continual learning strategies remains significant. IMM is the fastest continual learning strategy, being about six times as fast as the baseline. On the other hand, ALASSO is the slowest continual learning strategy, boasting around one-fourth of the execution time of the baseline. In terms of validation accuracy, the accuracy progression closely resembles the experiment with LC. Naive is the best-performing method, lacking ten percentage points to the baseline during the first 25,000 samples. IMM, EWC and MAS follow in that order. ALASSO starts with a similar validation accuracy as the other continual learning strategies but falls behind after 8,000 samples. All continual learning approaches have a declining validation accuracy in the final 10,000 samples because these are less representative than the previous samples.

Our final experiment in this setting is run with BADGE. Out of all experiments in this series, this one shows the smallest relative gap in execution time between the baseline and the continual learning strategies. This is because BADGE exhibits a high query time, which is a bottleneck from a runtime perspective. The ranking between the continual learning methods remains similar to previous experiments, with ALASSO being the slowest and IMM being the fastest. The validation accuracy progression of the baseline and the continual learning strategies is similar to the experiments with LC and CoreSet. ALASSO is an exception to this, however. While ALASSO experiences a decline in validation accuracy after 8,000 samples, it stalls at around 60% validation accuracy until the penultimate batch. We assume this is because BADGE selects more representative samples than the other active learning strategies, partially alleviating ALASSO's exploding gradients problem.

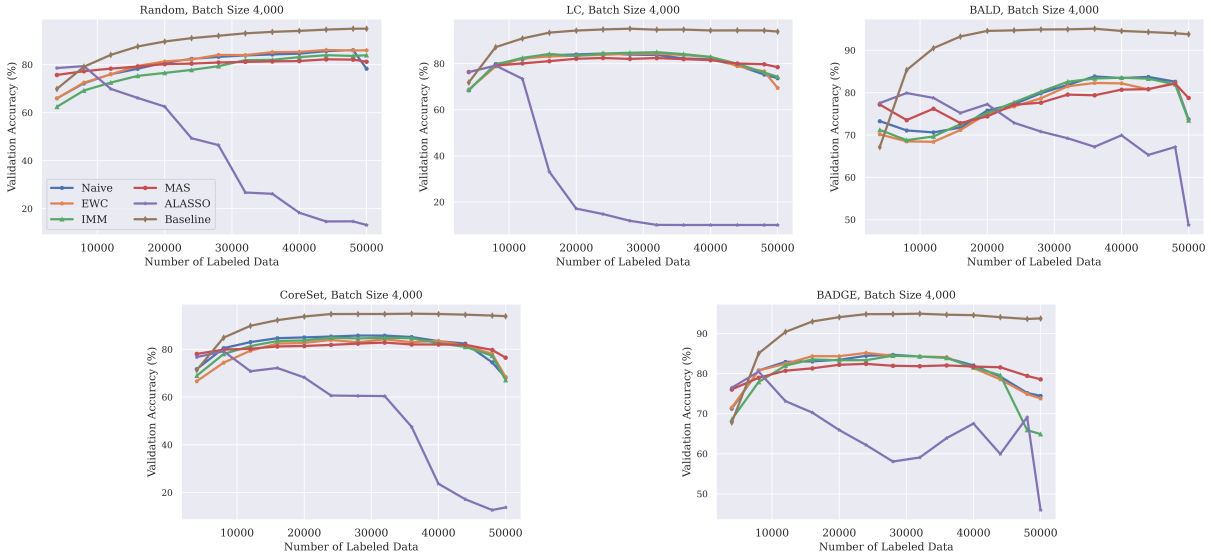

Figure 6.1: Comparison of validation accuracy of regularization-based continual learning strategies with batch size 4,000.

	Random	LC	BALD	CoreSet	BADGE
Naive	82	77	78	160	497
EWC	82	83	82	145	493
IMM	76	78	76	129	487
MAS	81	88	81	145	501
ALASSO	118	127	120	189	534
Baseline	717	712	721	782	1,311

Table 6.1: Comparison of execution time (in minutes) of regularization-based continual learning strategies with batch size 4,000.

Influence of Batch Size

Next, we study how the batch size of the selected active learning algorithm influences the results. To do so, we re-run each experiment from figure 6.1 with batch sizes 1,000 and 2,000. In this section, we only show the results for BADGE since the results for the other active learning strategies are similar. For completeness, we present the results of the remaining active learning strategies in appendix A.3.

In figure 6.2, we present the validation accuracy using BADGE with batch sizes 1,000, 2,000, and 4,000. The execution times of these experiments are displayed in table 6.2, respectively. We notice a positive correlation between batch size and validation accuracy. This correlation is caused by neural networks' tendency to overfit their training data. Since continual active learning methods only use the current batch as their training samples in a given iteration, the neural network is more likely to overfit the training data. A commonly known remedy for this problem is increasing the amount of training data [89]. Therefore, using a larger batch size yields better validation accuracy. Active learning algorithms, on the other hand, are not affected by overfitting because they use the entire labeled pool for training. We can see that

overfitting is an issue for small batch sizes not only because of the lower validation accuracy but also because the validation accuracy curve is more erratic.

In terms of execution time, we observe minor differences between the batch sizes for continual active learning and substantial differences for pure active learning. This is because the continual active learning strategies will always use the same number of training data across a complete experiment, regardless of the batch size. Active learning methods, however, use the entire labeled pool as their training data. In section 4.1.3, we mentioned that a model trained using active learning with a total budget of n and a batch size of b is trained on $\frac{n(n+b)}{2b}$ samples. To demonstrate the influence of the batch size in this setting, we calculate the total number of samples trained. The results are given in table 6.2. We can see that the number of training samples behaves anti-proportionally to the batch size. Therefore, the total execution time is higher for smaller batch sizes. Note, however, that this behavior is not anti-proportional, because the total execution time is broken down into training and query time. The training time is anti-proportional to the batch size. The query time, however, remains constant because the batch size does not influence the total amount of queried points.

Figure 6.2: Comparison of validation accuracy of Continual Learning strategies used with the active learning strategy BADGE.

Batch Size	Baseline	Naive	EWC	IMM	MAS	ALASSO
4,000	1,311	497	493	487	501	534
2,000	1,935	523	513	500	515	547
1,000	3,171	493	486	501	505	509

Table 6.2: Comparison of execution time of regularization-based continual learning strategies combined with BADGE.

b	Total number of training points
1,000	1,275,000
2,000	650,000
4,000	362,000

Table 6.3: Total number of training points for varying batch sizes on the CIFAR-10 dataset

Delaying the Start of Continual Learning

After running the experiments in figure 6.1, we notice that all of them demonstrate a significant discrepancy in validation accuracy between active learning and continual active learning. To further investigate this gap, we evaluate a hybrid approach where we run active learning for the first i iterations before switching to continual active learning. With these experiments, we hope to decrease the gap in validation accuracy to active learning. We vary i between 0 and 6, and perform two sets of experiments, one using the continual learning strategy MAS and the other using EWC. In both experiments, we use the active learning strategy BADGE. The results of the two sets of experiments can be found in figure 6.3. For MAS and EWC, the validation accuracy drops immediately after switching from active learning to continual active learning. However, in the long run MAS retains validation accuracy better than EWC. We also notice that while the validation accuracy does drop after switching to continual active learning, the accuracy progression after the switch outperforms continual active learning. We attribute these two observations to the regularization effect of MAS and EWC. The reason why MAS performs better than EWC in this setting is because it employs heavier regularization than EWC, which we observed in previous experiments.

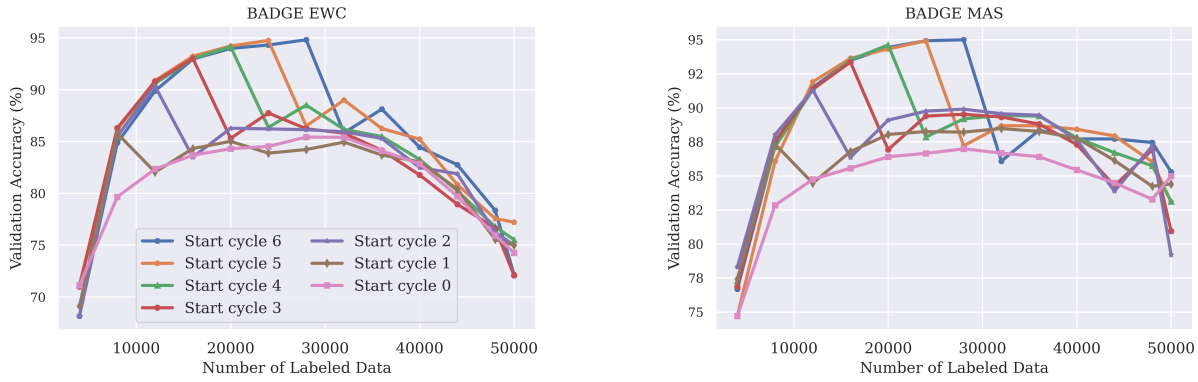

Figure 6.3: Comparison of validation accuracy for a delayed start of continual learning.

Varying the initialization of the labeled pool

Motivated by the findings from the previous section, we wonder if the initialization of the labeled pool impacts the validation accuracy of the continual learning strategies. Beck et al. [74] show that using a facility location selection [90] yields better validation accuracy when training on the initial labeled pool. We, therefore, test the effect of an initialization using facility location selection compared to our random initialization. As our active learning strategy, we use BADGE with a batch size of 4,000 and MAS as our continual learning method. The results of the experiment can be found in figure 6.4. While the facility location approach performs better than random initialization, the difference is marginal. Interestingly, the validation accuracy of the facility location approach is lower than random initialization in the first iteration. The most significant drawback of the facility location initialization is its resource intensity. Initialization with facility location takes about 24 hours to complete and requires more than 100GB of memory

(compared to less than 10 seconds and less than 2GB) for CIFAR-10 using the implementation provided by Beck et al. [74].

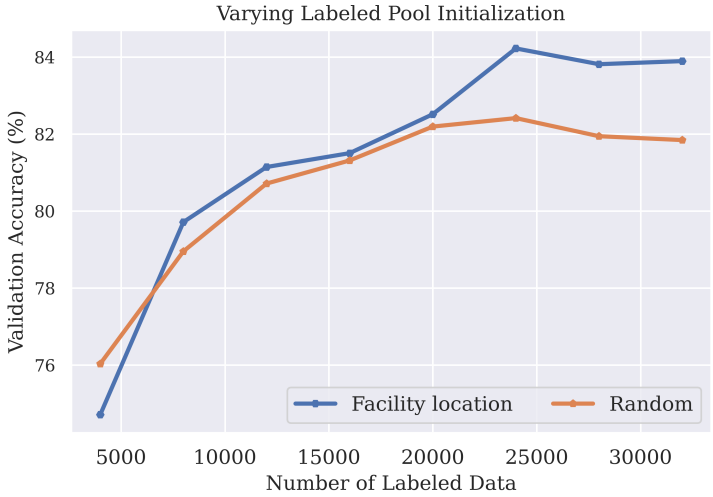

Figure 6.4: Comparison of validation accuracy for facility location initialization and random initialization.

6.1.2 Replay Continual Learning

In section 4.1.4, we introduced a custom replay strategy for continual active learning. Using our replay strategy is motivated by the main finding of section 6.1.1: the validation accuracy increases with increased batch size. In this set of experiments, we use the active learning strategy CoreSet and vary the batch size as well as the size of the replay buffer. Moreover, we investigate the effect of CoreSet selection in the replay buffer compared to random selection. We present our results in figure 6.5. When varying the batch size and buffer size, we notice that increasing either or both of them yields a higher validation accuracy. However, our replay strategy does not outperform the naive approach when using the same amount of training data (i.e., batch size plus replay buffer size for the replay strategy and batch size for the naive approach). This leads us to believe that the most decisive factor for the performance of continuous active learning is the amount of training data. For further reasoning, we refer to section 6.1.1.

To evaluate the importance of CoreSet selection in the buffer compression process, we compare the validation accuracy of our replay strategy when using CoreSet selection and random selection. We notice that the validation accuracy remains mostly the same for both buffer compression methods, apart from the last 15,000 samples, where CoreSet selection outperforms random selection. This is because the first 25,000 samples queried are informative enough to increase or maintain validation accuracy. Therefore, using random or CoreSet selection does not influence the validation accuracy at the beginning of the experiment. In the end, however, CoreSet selection ensures that the buffer contains enough informative samples while random selection does not.

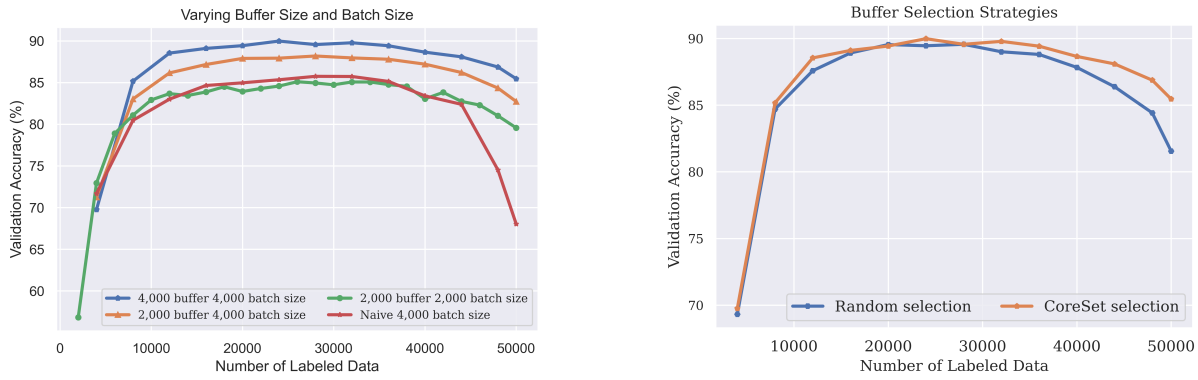

Figure 6.5: Experiments with our Replay strategy

The execution time of our Replay strategy is given in table 6.4. We notice that the buffer size is the main driver of execution time. This is because we replay the entire buffer in each iteration, meaning that we train on $\frac{n}{b} \cdot$ buffer size additional samples, where n is the number of iterations and b is the batch size. The batch size indirectly influences execution time, because an increased batch size reduces the number of active learning iterations, decreasing the total number of training points when the buffer size is fixed.

Batch size	Buffer size	Execution time
4,000	4,000	121
4,000	2,000	107
2,000	2,000	130

Table 6.4: Comparison of execution time using the Replay strategy in combination with CoreSet.

6.1.3 Exemplar Rehearsal Continual Learning and Representation-based Active Learning

Unsatisfied with the results from previous experiments, we decide to implement additional continual and active learning strategies. We perform an extensive literature search, investigating the suitability of representation-based active learning strategies and continual learning strategies from the exemplar rehearsal category in the summary paper by Mundt et al. [34]. The active learning strategy which we implement is VAAL [24], and the continual learning strategy is A-GEM [23]. We decide to implement VAAL because it is the only representation-based active learning strategy that consistently performs better than random sampling (apart from CoreSet, which we have already implemented). We choose to implement A-GEM as our continual learning strategy as it is one of the few exemplar rehearsal strategies applicable to (semi-) supervised learning (many other strategies focus on reinforcement learning). Furthermore, A-GEM is computationally efficient and demonstrates strong performance in the experiments by Chaudhry et al. [23].

First, we analyze the performance of VAAL as an active learning strategy. We run VAAL with a batch size of 4,000 and compare it to Random and CoreSet. We choose Random because it is

the baseline for active learning strategies and CoreSet because it is among the best performing active learning methods used before. The results are depicted in the left plot in figure 6.6. While VAAL outperforms random sampling by a large margin, it is outperformed by about the same margin by CoreSet. To ensure that the results are not due to a suboptimal hyperparameter choice, we run VAAL training VAE and discriminator for 20 epochs in one run and 100 epochs in another. Because we are interested in the performance of VAAL in our continual active learning setting, we use continual active learning with a batch size of 4,000 and the continual learning strategy Naive. We present the results in the right plot of figure 6.6. Surprisingly, the training time of the discriminator and VAE marginally impacts validation accuracy. If anything, the validation accuracy is higher when training for 20 epochs. We attribute the performance gap between CoreSet and VAAL to the fact that the samples from the labeled and unlabeled pool are too similar. This also explains why training discriminator and VAE for more epochs does not increase validation accuracy.

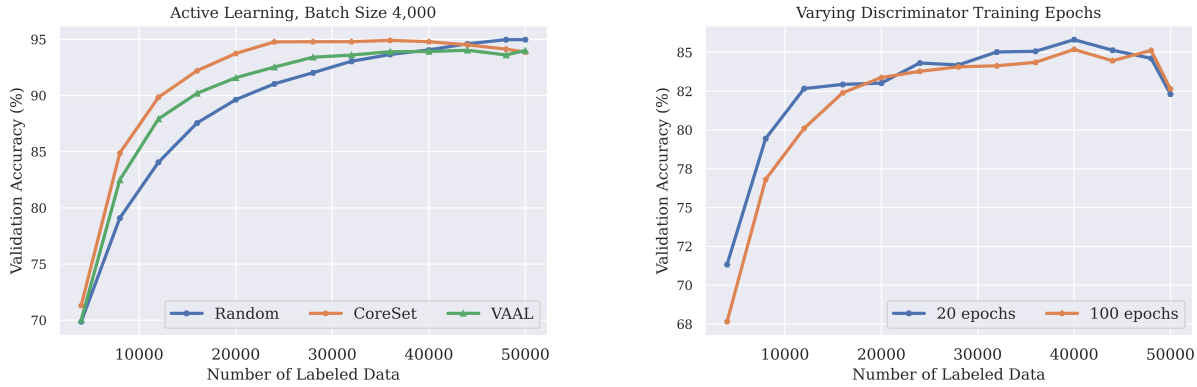

Figure 6.6: Left: Comparing VAAL to Random and CoreSet. Right: Comparison of validation accuracy when varying the training epochs for VAE and discriminator.

Next, we experiment with A-GEM. A-GEM has two hyperparameters: S, the number of samples randomly drawn from the memory to compute the reference gradients, and P, the number of data points stored in the memory from each task. We run A-GEM with the active learning strategy LC and a batch size of 2,000. To assess the performance of A-GEM, we compare its validation accuracy to the naive approach. We vary S and P and present our results in the right plot of figure 6.7. The validation accuracy increases with increased S and P until about 12,000 samples, after which the validation accuracy drops for all values of S and P. A-GEM outperforms the naive approach for the first 15,000 samples but is outperformed by Naive for the remainder of the experiment.

After investigating the influence of A-GEM's hyperparameters, we compare the combination of VAAL and A-GEM to VAAL and Naive, using a batch size of 4,000. The results are shown in the left plot of figure 6.7. Although A-GEM performs worse than the Naive approach when using LC, it outperforms the naive approach when using VAAL as the active learning strategy. We believe that A-GEM profits from the representation-based sampling of VAAL and the increased batch size, which explains the improved performance in this setting.

In table 6.5 we present the execution time of VAAL with active learning (baseline), VAAL and Naive as well as VAAL and A-GEM. We notice that VAAL with A-GEM is significantly

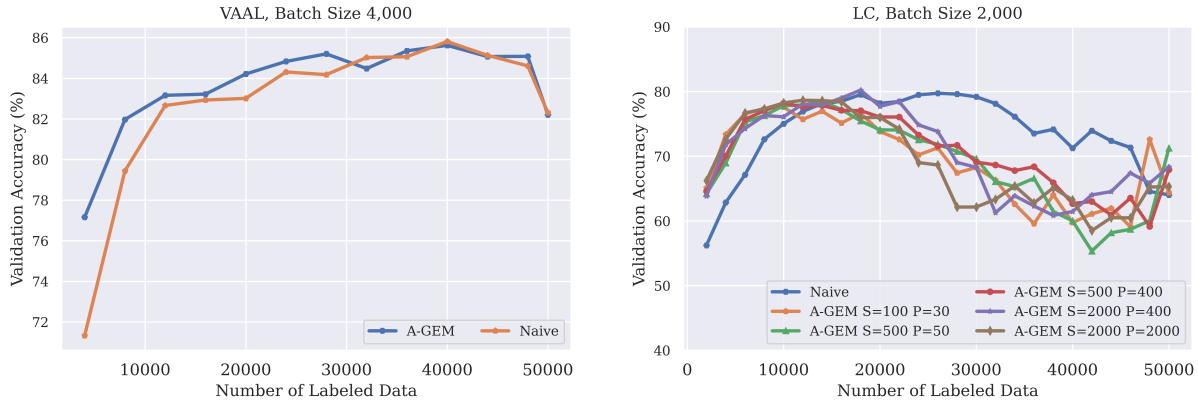

Figure 6.7: Left: Comparison of validation accuracy of A-GEM to Naive with VAAL as the active learning strategy. Right: Validation accuracy of A-GEM for different values of S and P.

slower than the naive approach because of the costly computation of the reference gradients. Overall, VAAL is among the slower active learning strategies due to the extensive training of the VAE and the discriminator on the entire dataset.

	Execution time
Baseline	961
Naive	341
A-GEM	862

Table 6.5: Comparison of execution time using VAAL

6.2 Continual Active Learning for Model Stealing

After extensively studying continual active learning in its natural setting, we shift the focus of our experiments to model stealing. Since we build our work on the ActiveThief framework, we first evaluate the performance of the ActiveThief framework, including the influence of model architecture and thief dataset. For these experiments, we refer to section A.4.1 of the appendix. After experimenting with the ActiveThief framework, we evaluate the performance of continual active learning for model stealing attacks. We perform experiments on the datasets MNIST, CIFAR-10, and CIFAR-100. Initially, we planned to compute the same experiments using ImageNet as our target model dataset. We decided against running the experiments because our thief dataset is a subset of ImageNet, an unrealistic setting in the real world. Furthermore, the validation accuracy of the target model would most likely be lower than for CIFAR-100, making it even more challenging to evaluate the different continual active learning strategies because their performance would not differ significantly. First, we analyze continual active learning using regularization-based continual learning strategies. Next, we move on to evaluate the combination of exemplar rehearsal continual learning and representation-based active learning.

6.2.1 Regularization-based Continual Learning

In this section, we evaluate the success of model stealing attacks using continual active learning with regularization-based continual learning strategies. We perform two sets of experiments: in the first set, the target model returns the predicted class label, and in the second set, the target model returns the softmax probabilities for each class. The remaining setting for both sets of experiments is as follows: We use the target model datasets MNIST, CIFAR-10, and CIFAR-100, the active learning strategies Random, LC, BALD, BADGE and CoreSet and the continual learning approaches Naive, EWC, IMM, MAS and ALASSO. We use the ActiveThiefConv3 model as the target and substitute model and perform one model stealing attack for each combination of active learning, continual learning strategy, and target model output. For the baseline runs, we use a batch size of 1,000 with a total query budget of 20,000. The query budget remains identical for the continual active learning runs, but we increase the batch size to 2,000.

The numbers reported in the tables represent the model agreement at the end of each experiment, i.e., after using up the query budget of 20,000. Readers interested in the progression of the model agreement across the experiments can find the respective plots in appendix A.4.2. As a baseline, we extract the models using active learning with the strategies mentioned before. To evaluate the active learning strategies and the continual learning strategies, we compute the average over the model agreement of all continual learning strategies combined with a fixed active learning strategy and the average of all active learning strategies combined with a fixed continual learning strategy. These numbers are given at the end of each column and row, respectively. The active learning strategy and the continual learning strategy with the highest average model agreement are highlighted in bold.

Training with Predicted Labels

In the first set of experiments, we perform model stealing attacks using the predicted class labels of the target model. We start with the target model dataset MNIST. The results of the experiments with MNIST are shown in table 6.6. First, we observe a large margin between the model agreement of the active learning strategies and the continual active learning strategies. However, the combination of BALD and Naive comes close to its baseline. It demonstrates a model agreement of 69.18%, about seven percentage points lower than the respective baseline. Overall, the naive approach performs best with an average model agreement of 51.29% across all active learning strategies. MAS, EWC and IMM follow in that order, whereas ALASSO suffers from exploding gradient problems and performs worst. The best active learning strategy is BALD with an average model agreement of 55.73%. BALD performs well in this setting because the substitute model, ActiveThiefConv3, contains dropout layers, which allow using Monte Carlo dropout for an accurate estimation of $\mathbb{E}_{\theta \sim p(\theta|L)}[H[y \mid x, \theta]]$.

For the next set of experiments, we use the target model dataset CIFAR-10 and present our results in table 6.7. Across the board, the model agreement is lower than in the previous experiment because the CIFAR-10 dataset is more complex than MNIST. Interestingly, the ranking of the continual learning strategies has changed significantly. EWC is now the best performing strategy, followed by IMM and Naive. MAS and ALASSO perform worst. The main reason for the large gap between the naive approach and EWC as well as IMM is the poor performance of BALD and Naive. Surprised by the poor performance of this combination, we

	Random	LC	BALD	CoreSet	BADGE	∅
Baseline	67.56	80.36	76.29	81.62	76.43	76.45
Naive	43.44	58.84	69.18	44.88	40.12	**51.29**
EWC	46.76	47.3	52.36	36.84	44.73	45.6
IMM	47.07	10.43	58.71	51.0	47.0	42.84
MAS	50.52	46.79	49.51	44.96	49.47	48.25
ALASSO	10.44	46.89	48.89	16.27	10.43	26.68
∅	39.65	50.97	**55.73**	38.79	38.35	-

Table 6.6: Model agreement (in %) of continual active learning strategies on MNIST using the predicted class label.

re-run the experiment and received similar results. We assume that the poor model agreement, initially caused by the complexity of the dataset, making it difficult for BALD to find informative samples. The best active learning strategy is CoreSet, with an average model agreement of 43.57%. Random follows next, outperforming BADGE, LC, and BALD.

	Random	LC	BALD	CoreSet	BADGE	∅
Baseline	60.23	49.73	61.28	63.78	62.26	59.46
Naive	49.84	30.03	10.18	48.44	45.25	36.75
EWC	50.37	47.12	50.22	47.94	49.11	**48.95**
IMM	49.84	44.25	43.55	49.64	48.22	47.1
MAS	45.76	33.27	36.87	39.98	40.24	32.02
ALASSO	19.05	38.16	30.91	31.86	25.04	29.0
∅	42.97	38.57	34.35	**43.57**	41.57	-

Table 6.7: Model agreement (in %) of continual active learning strategies on CIFAR-10 using the predicted class label.

Finally, we compute the model agreement using CIFAR-100 as our target model dataset. When starting experiments with BADGE, we observed that they took significantly longer than the other experiments. We investigated this further and found the estimated runtime to be more than 40 days using our hardware. This is why we could not deliver results for this combination. BADGE has a high runtime because its query time scales linearly with the number of classes. Since CIFAR-100 has 100 classes, the query time is ten times longer than the query time on CIFAR-10, which took four days. We present our results in table 6.8. Across the board, the model agreement has decreased compared to the previous experiment, which is again due to increased dataset complexity. While the absolute gap between the continual learning methods and the baseline is smaller than in the previous experiments, the relative gap has increased significantly. The baseline strategies boast a model agreement of 20.42% on average, whereas the best continual learning strategy, which is IMM, demonstrates a model agreement of 8.07% on average. IMM significantly outperforms both the naive approach and EWC, which exhibit almost identical performance. We were surprised by the poor performance of EWC and BALD, which is why we conducted this experiment one more time, however, we achieved similar results. It seems that the model is not accurate enough for BALD to select informative

queries in this setting, impeding further progress in model agreement. The remaining continual learning strategies, MAS and ALASSO, perform worse than the naive approach. Remarkably, ALASSO outperforms MAS, making this the only set of experiments in which ALASSO is not the worst-performing continual learning strategy. Another surprise is the performance of the random sampling, which outperforms the remaining active learning strategies. CoreSet, which demonstrated strong performance in previous experiments, falls behind Random and LC, outperforming only BALD by about one percentage point. CoreSet fails to perform in this setting because the uncertainty of its query selection increases with the number of classes [22].

	Random	LC	BALD	CoreSet	∅
Baseline	21.65	19.5	19.64	20.9	20.42
Naive	7.93	7.63	4.98	4.82	6.34
EWC	8.79	6.55	2.07	7.77	6.3
IMM	8.59	8.44	7.18	8.05	**8.07**
MAS	5.37	5.44	5.3	4.52	5.16
ALASSO	5.28	6.11	5.72	5.51	5.66
∅	**7.19**	6.83	5.05	6.13	-

Table 6.8: Model agreement (in %) of continual active learning strategies on CIFAR-100 using the predicted class label.

Training with Softmax Probabilities

We move on to the second set of experiments, in which we train the substitute model using the softmax probabilities of the target model. First, we compute model agreement on the MNIST dataset. The results of this experiment can be found in table 6.9. In terms of the model agreement, all continual active learning attacks perform significantly worse than the baseline. The best-performing attack is BALD with MAS, achieving a model agreement of 67.84%. The most performant active learning strategy is BALD with an average model agreement of 52.52%, whereas the best continual learning strategy is MAS. MAS achieves an average model agreement of 57.73%. While the continual learning strategies struggled to outperform the naive approach in previous experiments, most of them outperform the naive approach in the model stealing setting. The only exception to this is ALASSO which experiences exploding gradient problems. Overall, the model agreement is about ten percentage points higher than training with the predicted label. This is because the softmax probabilities reveal more about the target model's learned prediction function than the predicted label.

We move on to the dataset CIFAR-10. Our results are shown in table 6.10. Similar to our findings from previous experiments, the baseline outperforms the continual active learning approaches. However, the difference in performance between them is marginal at a maximum of 12 percentage points. ALASSO is an exception to this, performing significantly worse than the other continual learning strategies due to exploding gradient problems. The best continual active learning strategy is EWC with an average model agreement of 60.14%. At the same time, EWC is the only continual learning strategy to outperform the naive approach, albeit the margin between the two is only 0.48 percentage points. Overall, CoreSet is the most performant active learning strategy, achieving an average model agreement of 54.12%.

	Random	LC	BALD	CoreSet	BADGE	∅
Baseline	87.91	82.39	83.64	91.22	79.68	84.97
Naive	47.74	43.02	59.61	58.36	48.89	51.52
EWC	59.26	53.67	59.18	56.13	50.28	55.7
IMM	48.18	62.95	49.71	63.43	58.76	56.61
MAS	64.42	50.27	67.84	55.58	50.54	**57.73**
ALASSO	28.04	15.5	26.28	10.39	12.8	20.6
∅	49.57	45.08	**52.52**	48.78	44.24	-

Table 6.9: Model agreement (in %) of continual active learning strategies on MNIST using softmax output.

	Random	LC	BALD	CoreSet	BADGE	∅
Baseline	71.58	70.04	71.96	71.45	71.44	71.29
Naive	60.8	56.62	61.84	61.61	57.42	59.66
EWC	58.67	56.8	61.57	61.79	61.88	**60.14**
IMM	60.78	52.24	60.32	61.39	56.98	58.34
MAS	50.32	51.45	52.09	52.23	55.68	52.35
ALASSO	17.26	24.87	28.24	33.59	32.93	27.38
∅	49.57	48.4	52.81	**54.12**	52.98	-

Table 6.10: Model agreement (in %) of continual active learning strategies on CIFAR-10 using softmax output.

Finally, we experiment with the CIFAR-100 dataset. As in section 6.2.1, we omit the results for BADGE due to their high runtime. The results for the remaining combinations can be found in table 6.11. Across the board, the model agreement is significantly lower than for MNIST and CIFAR-10, caused by the complexity of this dataset. In this setting, EWC outperforms the remaining continual learning strategies, and it is one of two continual learning strategies, which outperform the naive approach. The other strategy is IMM, which puts on a performance in between EWC and the naive approach. MAS follows behind the naive approach and ALASSO is left behind. The best active learning strategy is CoreSet, which outperforms BALD by 0.57 percentage points.

	Random	LC	BALD	CoreSet	∅
Baseline	25.43	26.92	28.01	27.48	26.96
Naive	18.46	18.48	16.8	17.69	17.86
EWC	19.45	17.46	20.67	19.98	**19.39**
IMM	18.16	17.9	20.39	18.75	18.8
MAS	15.75	14.85	14.72	15.45	15.19
ALASSO	5.2	4.95	6.7	10.24	6.77
∅	15.4	14.73	15.85	**16.42**	-

Table 6.11: Model agreement (in %) of continual active learning strategies on CIFAR-100 using softmax output.

6.2.2 Exemplar Rehearsal Continual Learning and Representation-based Active Learning

After conducting our experiments with regularization-based continual learning strategies, we test the combination of the active learning strategy VAAL with the continual learning strategy A-GEM. We motivate this experiment by the performance of VAAL and A-GEM in the classic continual learning setup, given in figure 6.7. We conduct the experiments with the same hyperparameters as in section 6.2.1. To evaluate the performance of VAAL and A-GEM, we compare the results to the best performing active learning strategy from the experiments in section 6.2.1, CoreSet, and the combination of the best performing continual learning strategy and the most performant active learning strategy, which are EWC and CoreSet, respectively. The results are given in Table 6.12. We present the model agreement progression with VAAL and A-GEM in appendix A.4.2. While VAAL and A-GEM cannot keep up with the performance of the baseline, it outperforms CoreSet and EWC in three out of six experiments while being outperformed in the remaining three.

Attack strategy	MNIST		CIFAR-10		CIFAR-100	
	Softmax	Label	Softmax	Label	Softmax	Label
VAAL A-GEM	52.54	53.8	61.48	50.5	18.29	8.77
CoreSet EWC	56.13	36.84	61.79	47.94	19.98	7.77
Coreset Baseline	90.65	77.58	71.61	61.68	27.52	20.96

Table 6.12: Comparison of model agreement (in %) using VAAL and A-GEM.

7 Discussion

In this chapter, we discuss our findings from the experiments in chapter 6. We divide this chapter into two parts: One for continual active learning and one for model stealing. The first section will discuss the findings from the first batch of experiments, where we tested continual active learning as an alternative to pure active learning. Next, we discuss the findings from our experiments with model stealing.

7.1 Continual Active Learning

The aim of this section is to discuss the findings from our experiments using continual active learning as an alternative to active learning. When developing the continual active learning approach proposed in this thesis, our initial aim was to improve model performance, measured by validation accuracy and the execution time compared to active learning. By studying the results, it is evident that we did not outperform active learning in any of the experiments. On the contrary, active learning, even naive active learning using random sampling, significantly outperforms continual active learning by validation accuracy. However, we achieved the second goal of improving the training time. Before we analyze the reasons for the poor performance of continual active learning by validation accuracy, we will first discuss the performance in terms of execution time.

7.1.1 Execution time

In terms of execution time, we observe that continual active learning outperforms active learning in all experiments. To aid the explanation of the results, we first break down the total execution time of the experiments into the time spent on training the model, referred to as training time, and the time spent on determining the informative samples, referred to as query time. We note that continual active learning does not influence the query time since the amount of data queried is the same for active learning and continual active learning. Therefore, the only way to reduce the execution time is by reducing the training time.

The main reason for the reduced training time is that continual active learning trains the model on significantly fewer data points than active learning. In section 4.1.3, we derived that an active learning algorithm with a total budget of n and a batch size of b is trained on $\frac{n(n+b)}{2b}$ data points. We investigated this using the dataset CIFAR-10 in section 6.1.1. Furthermore, we found that the total number of training points behaves anti-proportional to the batch size. Therefore, in order to decrease execution time as much as possible, we should set the batch size as large as possible.

For active learning strategies such as Random and LC, the query time is marginal compared to the training time. Therefore, continual active learning strategies using Random and LC

achieve the highest speed-up. BALD behaves similarly when training a model where Monte Carlo dropout is not applicable. If Monte Carlo dropout is applicable, the query time of BALD is determined by the number of dropout iterations, and how much of a bottleneck it is depends on this number. The query time of VAAL also depends on a hyperparameter. VAAL uses a VAE and a discriminator to determine informative samples. These models need to be trained for a number of iterations, and training them is computationally expensive, especially because they are trained on the total dataset in each iteration. Therefore, it is important not to train them for too many iterations. The query time of BADGE and CoreSet largely depends on the model, the size of the dataset, and, in the case of BADGE, the number of classes in the dataset. In most cases, at least one of the previously mentioned parameters is so large that the query time becomes a bottleneck. Therefore, the speed-up of continual active learning using BADGE and CoreSet is not as significant as the speed-up of other continual learning strategies.

Next, we investigate the contribution of the continual learning strategies in terms of runtime. Overall, we observe that the overhead introduced by the continual learning strategies is significantly smaller compared to the overhead introduced by the active learning strategies, except for A-GEM and Replay. A-GEM introduces a significant overhead because it computes reference gradients in each training step. The parameter S, which determines how many samples are drawn to compute the reference gradients, determines the size of this overhead. Replay introduces overhead by training on more samples. The larger the replay buffer, the greater the overhead because Replay uses the current batch and the full replay buffer to train the model.

Out of the regularization-based continual learning strategies, the greatest increase in runtime is established by ALASSO. In our experiments, ALASSO increased the execution time by 30 to 60 Minutes. The reason for the increase in runtime is the backpropagation step. ALASSO needs two backward passes through the model to compute the complete gradients: The first is used to compute the unregularized gradients which are needed for the computation of Ω in equation 3.21 and the second is used to compute the gradients for the surrogate loss. MAS, IMM, and EWC only need one backward pass to compute the gradients, just as the naive approach, because they do not require saving the unregularized gradients.

Apart from the overhead introduced by backpropagation, the continual learning strategies introduce overhead by estimating weights for the parameters of the neural network at the end of each task. ALASSO computes its weights by using equation 3.21, MAS uses the gradients of the validation step whereas EWC and IMM compute the diagonals of the FIM. This overhead is marginal, however, as can be seen from the execution times presented in section 6.1.1. Surprisingly, IMM and EWC demonstrate minimally lower runtime than the Naive approach in a few experiments. While this is not possible in theory, we believe that the superior performance of IMM and EWC can be explained by implementation details. Additionally, factors beyond our control, such as operating system overhead or disk access speed, influence execution time. At this stage, we would like to point out that we do not intend to give a perfect ranking of the continual learning strategies in terms of execution time. Instead, we aim to give an estimation of the speed-up introduced by using continual active learning over active learning.

7.1.2 Validation Accuracy

After analyzing the results by execution time, we will focus on the validation accuracy. In terms of validation accuracy, the active learning strategies significantly outperform the continual active learning strategies. Before we analyze the difference in performance between active learning and continual active learning, we will first investigate them individually.

When analyzing the validation accuracy curves of the active learning methods, we see that, apart from random sampling, they follow the same pattern: First, the validation accuracy increases rapidly until the size of the labeled pool has reached about 20% of the complete labeled set. When the labeled pool comprises 20% to 50 % of the total labeled set, the validation accuracy increases moderately until it reaches its peak at 50% of the labeled set. Between 50% and 100% of the labeled set, the validation accuracy decreases marginally until it reaches the level it achieved with 20% of the complete training set. Before we analyze why this behavior occurs, we point out that the batch size has a negligible effect on the validation accuracy progression in our experiments, which is in line with the findings of Beck et al. [74]. We explain the validation accuracy curve of active learning as follows: First, we train the model on the most informative samples, leading to a rapid increase in validation accuracy. However, the larger the size of the labeled pool, the less informative samples exist, and the less informative are even the most informative samples of the unlabeled pool. This is the reason for the moderate increase in validation accuracy between 20% and 50% of the labeled pool. As soon as the labeled pool comprises 50% of the complete labeled set, the model has seen all the informative samples. At this point, the validation accuracy begins to decline because the newly labeled samples are uninformative. Nevertheless, the validation accuracy does not experience a sharp decline because the labeled pool still contains the informative samples labeled in the beginning.

The validation accuracy curves of the continual learning strategies are more diverse than the ones of the active learning strategies. When analyzing these validation accuracy curves, one should remember that a single continual active learning strategy consists of an active learning strategy and a continual learning strategy. Therefore, we need to analyze the contribution of both the active learning strategy and the continual learning strategy to the validation accuracy. Since we used more than 25 combinations of continual and active learning strategies in our experiments, we will not analyze each combination in detail. However, the contribution of active learning and continual learning to the validation accuracy can be isolated in most cases, making an analysis of each combination redundant.

First, we will analyze how the continual learning strategies contribute to the validation accuracy of continual active learning. We start with ALASSO, which exhibits the most inferior performance of all continual learning strategies. One problem of ALASSO is its erratic behavior. While ALASSO does not perform significantly worse than the other Continual Learning strategies at the beginning of each experiment, it has by far the highest variance in its validation accuracy. For smaller batch sizes, ALASSO performs similarly to other continual learning strategies, whereas it becomes practically unusable after about 20% of all samples for larger batch sizes. We believe that overestimating the loss on a single side of the loss function creates an uneven loss function, causing ALASSO's substandard performance. The problem is that overestimation is done for each parameter individually, meaning that the loss could be overestimated for one parameter and correctly estimated for another. The exploding gradient problem we observe in our experiments supports the thesis that parameter optimization using gradient

descent is challenging when the loss is estimated as is done with ALASSO. While using gradient clipping did mitigate the exploding gradient problem, it did not solve it completely. When observing the loss progression for ALASSO, we notice that the loss dramatically increases for the first epoch of each active learning iteration. Subsequently, the loss decreases but does not recover from the gradient update performed in the first epoch. We tried clipping the gradient at even smaller values of the l_2 norm to eradicate the loss increase in the first epoch. However, this impeded model convergence.

Next, we analyze IMM and EWC. Overall, IMM and EWC demonstrate a similar validation accuracy to the naive approach. We believe this is because they employ regularization less stringently than ALASSO and MAS. The experiments where the validation accuracy for Naive, IMM and EWC drops off similarly while MAS manages to maintain its validation accuracy support this hypothesis. Our experiment with the delayed start of continual active learning in figure 6.3 provides further evidence for this hypothesis.

Third, we investigate the behavior of MAS. MAS performs significantly better than ALASSO but is outperformed by IMM, EWC and Naive. We believe that this is because MAS restricts parameter updates more than EWC and IMM. When observing the validation accuracy progression of MAS, we note that MAS does not exhibit the steady increase in validation accuracy EWC, IMM, and Naive do. On the other hand, it also does not experience a sharp decline in validation accuracy within the final 10,000 samples. While it is desirable that MAS effectively manages to preserve previously learned knowledge, we argue that it cannot adapt to new tasks.

Finally, we analyze the naive approach, which enables us to bridge between the contribution of continual learning and active learning to validation accuracy. The naive approach performs significantly better than ALASSO and MAS in the first 80% of the experiments because it does not restrict parameter updates. Since the accuracy of the naive approach increases for the first 50% of all samples, we assume that these samples are the ones that are informative enough to make the model further learn the task it is trained for. After about 40-60% of all samples, the accuracy of the naive approach starts to decline, which is caused by the fact that the samples in this period are less informative than before. Nevertheless, these samples are still informative enough to impede forgetting. However, the final ten percent of all samples cause a significant reduction in validation accuracy. This is interesting because it demonstrates that a task can be unlearned to some extent when using data that belongs to the same task.

We move on to the contribution of active learning to the validation accuracy. First, we analyze the performance of LC. Continual active learning strategies using LC generally follow the pattern described above. To add to the previous section, we believe that for LC, the drop in validation accuracy within the final ten percent of all samples is caused by an uneven class distribution in the final queries. We observed the class distribution for LC over multiple iterations and noticed that it became increasingly inhomogeneous over time.

Next, we investigate the contribution of BALD to the validation accuracy of continual active learning. BALD uses Monte Carlo dropout to measure a model's uncertainty about given samples within the labeled pool. Therefore, it is crucial that the model used contains dropout layers and that the number of dropout iterations is sufficiently large (we recommend using at least 25) to ensure an accurate estimation of model uncertainty. We see the consequences of not using Monte Carlo dropout in figure 6.1 where BALD performs significantly worse than the remaining active learning strategies. By contrast, Monte Carlo dropout is used for the experiments in table 6.6, where BALD outperforms all remaining active learning strategies.

Regarding BADGE and CoreSet, we notice that they show matching performance, similar to EWC and IMM in the continual learning setting. BADGE and CoreSet follow the typical continual active learning curve, described above, the most. This is because they incorporate diversity into their sampling strategies which smooths the validation accuracy curve except for the final ten percent of samples and experiments with small batch sizes. We believe the final ten percent of samples generally do not represent the task the model is trained for well. Therefore, even a representative subset of these samples cannot improve the validation accuracy. For the setting with small batch sizes, i.e., small b, the issue is rather that b points are insufficient to represent the unlabeled pool well.

Finally, we briefly discuss the performance of VAAL and A-GEM. VAAL by itself is outperformed by the remaining active learning strategies which demonstrates its inability to select the most informative samples. However, this is an advantage when using VAAL in conjunction with A-GEM. In our experiment with LC and A-GEM, we noticed that A-GEM outperformed LC in terms of validation accuracy within the first 30% of samples but suffered from a heavy drop in validation accuracy thereafter. It seems that A-GEM struggles to retain the knowledge of previously learned samples when their informativeness changes rapidly. Ironically, the fact that VAAL is not as performant as the other active learning strategies helps A-GEM increase its validation accuracy when combined with VAAL.

We close this section by aiming to answer the most decisive question: **Why does continual active learning fail to outperform active learning?** We believe this is mainly **due to the number of training points used in each iteration**. While active learning strategies use the complete labeled pool in each iteration ($i \cdot b$ data points where i is the number of iterations), continual active learning strategies only use the data points labeled in the current task, i.e., b points. It is worth pointing out that the correlation between data points trained on, and validation accuracy achieved is not perfect. However, especially if the number of data points trained on is less than 20% of the total size of the training set, just one or two percentage points can make a big difference in validation accuracy. The observation that the validation accuracy increases with increasing batch size in section 6.1.1 supports this. Our Replay strategy further demonstrates this behavior. We observe identical performance for various sizes of the replay buffer and the batch size as long as the sum of replay buffer size and batch size is identical. Analysis of the second experiment with our Replay strategy shows that for the first 50% of samples, it is irrelevant how we select the data we replay, which further supports the thesis.

A second component to the answer to this question is that the continual learning strategies we used, or any continual learning strategy, are not designed for a setting in which all the training data belong to the same task. The continual learning strategies have been developed for and evaluated on task-incremental and class-incremental learning settings. Since all the data points stem from the same distribution in our setting, weights that are important for one task are also important for the other tasks. However, **regularization-based continual learning approaches complicate the update of important weights, impeding further improvements in validation accuracy**. This results in the poor performance of regularization-based continual learning strategies, not only compared to the baseline, but also compared to the naive approach.

7.2 Continual Active Learning for Model Stealing

In this section, we discuss the suitability of our continual active learning strategy for model stealing attacks. First, we observe that using the target model's prediction probabilities yields better model agreement than using the predicted class label. This finding is consistent with attacks presented by Orekondy et al. [9] and Pal et al. [8]. The superior performance of learning with softmax probabilities is due to their increased informativeness compared to the predicted class label. When using the predicted class label, the only information conveyed is which label the target model would most likely assign to the given sample. However, when using the softmax probabilities, the substitute model can profit from the target model's prediction certainty (i.e., how high is the top 1 probability?) and a comprehensive ranking of the most likely labels.

Overall, we observe that continual active learning attacks are outperformed by active learning attacks in terms of the model agreement. However, we note that the difference in model agreement heavily depends on the target model dataset. For less complex datasets, such as MNIST, the performance gap is negligible. While we do not observe this behavior in our experiments in section 6.2, we found that renouncing data augmentation during the model stealing attack significantly increases model agreement for MNIST. Results of this experiment are presented in appendix A.4.3. We assume that the reason for this behavior is that data augmentation obscures the target model's function and makes it harder for the substitute model to learn it. With increasingly complex datasets, such as CIFAR-10 and CIFAR-100, the performance gap between active learning and continual active learning increases. In these cases, the function learned by the target model becomes too complex to learn by training with just a fraction of the dataset. Since real-world datasets are usually at least as complex as CIFAR-100, **we conclude that continual active learning is not a suitable attack strategy for model stealing attacks**.

Next, we analyze a unique phenomenon we only observe when using continual active learning for model stealing. In many of our experiments, we observe that model agreement stalls at $\frac{1}{c}$ where c is the number of classes, meaning that the model has not learned the task at all, and its predictions are equal to random guessing. In some experiments, the model agreement recovers from this state and increases after a few iterations. There are experiments, however, in which the model agreement remains at $\frac{1}{c}$ for the entire duration of the attack. Furthermore, this phenomenon is more likely to occur for more complex datasets and when using the predicted class label instead of the target model's prediction probabilities. We assume the following vicious cycle causes this phenomenon: First, the model is initialized with random weights and trained using a random subset of the thief dataset. Depending on the initialization and the informativeness of the selected samples, the model might have low model agreement from this first iteration. If the model agreement is as low as $\frac{1}{c}$, the model will effectively guess predictions, complicating query selection for the active learning strategy. Unless the queries are informative by chance, the training process will never exit this vicious cycle. This behavior is more prevalent for attacks on more complex datasets and when using the predicted class label because, in these cases, the target model function is more complex to learn with the small number of training points used in continual active learning.

Finally, we point out that model stealing is a domain that contains many hyperparameters, such as the target model, the substitute model, training hyperparameters for each of these, and

multiple more. **Optimizing the performance of model stealing attacks is therefore tricky and requires extensive experimentation.** Furthermore, the experiments from section 6.2 and appendix A.4.3 demonstrate that model stealing is a domain where randomness plays a huge role and where a single hyperparameter has a tremendous influence on the outcome of an experiment.

8 Conclusion

In this thesis, we proposed a novel approach named continual active learning, which combines active learning and continual learning. We implement multiple active learning strategies, namely BALD, LC, BADGE as well as CoreSet, and combine them with the regularization-based continual learning strategies EWC, IMM, ALASSO and MAS, to create continual active learning strategies. Next, we evaluate them on the dataset CIFAR-10 and the neural network architecture ResNet18. Our results demonstrate that continual active learning strategies have significantly lower runtime than their active learning counterparts, but fail to achieve their accuracy. Therefore, we decide to implement the active learning strategy VAAL along with the continual learning strategy A-GEM to see if they can outperform the other continual active learning strategies but receive similar results. Our analysis of the results suggests continual active learning achieves sub-par validation accuracy because the number of training points used in each active learning iteration is too small. Further analysis of the results from our experiments with a custom Replay strategy support this theory.

Before applying continual active learning to the model stealing domain, we rigorously analyze the setting of the ActiveThief paper [8]. While we agree with most findings of the paper, we find that the architecture of the target and the substitute model has a greater influence on the model agreement achieved than the authors of ActiveThief suggest. Furthermore, we find that the choice of the thief dataset significantly influences model agreement, and so does using data augmentation on the thief dataset.

Finally, we perform model stealing attacks with continual active learning, using MNIST, CIFAR-10, and CIFAR-100 as target model datasets. The results from these experiments closely resemble those from the classic continual active learning setting, i.e., we find that continual active learning achieves lower model agreement than model extraction using active learning.

In general, we find that our approach is (resource-) *efficient* but not *effective* because it is significantly outperformed by the baseline, both in the classic continual active learning and in the model stealing setting, in terms of accuracy and agreement, respectively. However, we point out that the validity of our results is limited. First, the list of continual and active Learning strategies we evaluated is not exhaustive, although we made every effort to include the most popular strategies. Second, we only evaluated our approach with one neural network architecture and one dataset in the classic continual active learning setting and one neural network architecture plus three datasets in the model stealing setting.

8.1 Future Work

The approach we proposed in this thesis is not limited to the settings we evaluated it in. While the most straightforward extension would be to evaluate the method on new datasets and neural network architectures, we believe that, from a research perspective, it would be more

interesting to apply continual active learning to unrelated settings. In our methodology, we had an active-learning-centric view on continual active learning, i.e., our main goal was to improve the accuracy of active learning while reducing its runtime. Future work might be interested in approaching continual active learning from a continual learning perspective, for example, by using active learning to perform task ordering in a task-incremental setting. Another idea could be to leverage the information of active learning for continual learning and vice-versa. In our current framework, active learning and continual learning work independently, and we see potential for improvement by introducing interdependence between them.

Broadly speaking, our approach can be used in any setting where the data distribution of the task changes and labeling data is expensive. In the computer security domain, our approach may be applied to malware analysis. Previous work has demonstrated the effectiveness of using machine learning for malware analysis [91, 92]. Malware analysis is well suited for our approach because the features that classify malware change over time, and determining if a given piece of software contains malware is time-consuming.

Bibliography

[1] Ian Goodfellow et al. “Generative adversarial networks”. In: *Communications of the ACM* 63.11 (2020), pp. 139–144.

[2] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[3] Yann LeCun et al. “Backpropagation applied to handwritten zip code recognition”. In: *Neural computation* 1.4 (1989), pp. 541–551.

[4] Reza Shokri et al. “Membership inference attacks against machine learning models”. In: *2017 IEEE symposium on security and privacy (SP)*. IEEE. 2017, pp. 3–18.

[5] Seong Joon Oh, Bernt Schiele, and Mario Fritz. “Towards reverse-engineering black-box neural networks”. In: *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning* (2019), pp. 121–144.

[6] Nicolas Papernot et al. “Practical black-box attacks against machine learning”. In: *Proceedings of the 2017 ACM on Asia conference on computer and communications security*. 2017, pp. 506–519.

[7] Florian Tramèr et al. “Stealing Machine Learning Models via Prediction APIs.” In: *USENIX security symposium*. Vol. 16. 2016, pp. 601–618.

[8] Soham Pal et al. “Activethief: Model extraction using active learning and unannotated public data”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 01. 2020, pp. 865–872.

[9] Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. “Knockoff nets: Stealing functionality of black-box models”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 4954–4963.

[10] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[11] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[12] Robert Nikolai Reith, Thomas Schneider, and Oleksandr Tkachenko. “Efficiently stealing your machine learning models”. In: *Proceedings of the 18th ACM workshop on privacy in the electronic society*. 2019, pp. 198–210.

[13] Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. “Prediction poisoning: Towards defenses against DNN model stealing attacks”. In: *International Conference on Learning Representations* (2020).

[14] Mika Juuti et al. “PRADA: protecting against DNN model stealing attacks”. In: *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE. 2019, pp. 512–527.

[15] James Kirkpatrick et al. "Overcoming catastrophic forgetting in neural networks". In: *Proceedings of the national academy of sciences* 114.13 (2017), pp. 3521–3526.

[16] Rahaf Aljundi et al. "Memory aware synapses: Learning what (not) to forget". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 139–154.

[17] Sang-Woo Lee et al. "Overcoming catastrophic forgetting by incremental moment matching". In: *Advances in neural information processing systems* 30 (2017).

[18] Dongmin Park et al. "Continual learning by asymmetric loss approximation with single-side overestimation". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 3335–3344.

[19] David D Lewis. "A sequential algorithm for training text classifiers: Corrigendum and additional data". In: *Acm Sigir Forum*. Vol. 29. 2. ACM New York, NY, USA. 1995, pp. 13–19.

[20] Jordan T Ash et al. "Deep batch active learning by diverse, uncertain gradient lower bounds". In: *International Conference on Learning Representations* (2020).

[21] Neil Houlsby et al. "Bayesian active learning for classification and preference learning". In: *CoRR* (2011).

[22] Ozan Sener and Silvio Savarese. "Active learning for convolutional neural networks: A core-set approach". In: *International Conference on Learning Representations* (2017).

[23] Arslan Chaudhry et al. "Efficient lifelong learning with a-gem". In: *International Conference on Learning Representations* (2019).

[24] Samarth Sinha, Sayna Ebrahimi, and Trevor Darrell. "Variational adversarial active learning". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 5972–5981.

[25] Burr Settles. "Active learning literature survey". In: *Computer Sciences Technical Report 1648* (2009).

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

[27] Jia-Jie Zhu and José Bento. "Generative Adversarial Active Learning". In: *NIPS Workshop on Teaching Machines, Robots, and Humans* (2017).

[28] David A Cohn, Zoubin Ghahramani, and Michael I Jordan. "Active learning with statistical models". In: *Journal of artificial intelligence research* 4 (1996), pp. 129–145.

[29] Eric B Baum and Kenneth Lang. "Query learning can work poorly when a human oracle is used". In: *International joint conference on neural networks*. Vol. 8. Beijing China. 1992, p. 8.

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

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

[32] Michael McCloskey and Neal J Cohen. "Catastrophic interference in connectionist networks: The sequential learning problem". In: *Psychology of learning and motivation*. Vol. 24. Elsevier, 1989, pp. 109–165.

[33] Gail A. Carpenter and Stephen Grossberg. "The ART of adaptive pattern recognition by a self-organizing neural network". In: *Computer* 21.3 (1988), pp. 77–88.

[34] Martin Mundt et al. "A wholistic view of continual learning with deep neural networks: Forgotten lessons and the bridge to active and open world learning". In: *Neural Networks* (2020).

[35] Tinne Tuytelaars & Andreas S. Tolias Gido van de Ven. "Three Types of incremental learning". In: *Nature Machine Intelligence* 4 (2022), pp. 1185–1197.

[36] Gido M. van de Ven and Andreas S. Tolias. *Generative replay with feedback connections as a general strategy for continual learning*. 2019. arXiv: `1809.10635 [cs.LG]`.

[37] Yann LeCun, Corinna Cortes, and Christopher Burges. *The MNIST Database of Handwritten Digists*. Available online: `http://yann.lecun.com/exdb/mnist/` (accessed on 25 April 2023). 2012.

[38] German I Parisi et al. "Continual lifelong learning with neural networks: A review". In: *Neural networks* 113 (2019), pp. 54–71.

[39] Friedemann Zenke, Ben Poole, and Surya Ganguli. "Continual learning through synaptic intelligence". In: *International conference on machine learning*. PMLR. 2017, pp. 3987–3995.

[40] Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. "Distilling the knowledge in a neural network". In: *NIPS Deep Learning and Representation Learning Workshop* (2015).

[41] Zhizhong Li and Derek Hoiem. "Learning without forgetting". In: *IEEE transactions on pattern analysis and machine intelligence* 40.12 (2017), pp. 2935–2947.

[42] Amal Rannen et al. "Encoder based lifelong learning". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 1320–1328.

[43] Hanul Shin et al. "Continual learning with deep generative replay". In: *Advances in neural information processing systems* 30 (2017).

[44] Timur Ash. "Dynamic node creation in backpropagation networks". In: *Connection science* 1.4 (1989), pp. 365–375.

[45] Daryna Oliynyk, Rudolf Mayer, and Andreas Rauber. "I Know What You Trained Last Summer: A Survey on Stealing Machine Learning Models and Defences". In: *ACM Computing Surveys* (2023).

[46] Binghui Wang and Neil Zhenqiang Gong. "Stealing hyperparameters in machine learning". In: *2018 IEEE symposium on security and privacy (SP)*. IEEE. 2018, pp. 36–52.

[47] Mengjia Yan, Christopher Fletcher, and Josep Torrellas. "Cache telepathy: Leveraging shared resource attacks to learn DNN architectures". In: *USENIX Security Symposium*. 2020.

[48] Daniel Lowd and Christopher Meek. "Adversarial learning". In: *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. 2005, pp. 641–647.

[49] Jialong Zhang et al. "Protecting intellectual property of deep neural networks with watermarking". In: *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*. 2018, pp. 159–172.

[50] Sebastian Szyller et al. "Dawn: Dynamic adversarial watermarking of neural networks". In: *Proceedings of the 29th ACM International Conference on Multimedia*. 2021, pp. 4417–4425.

[51] Manish Kesarwani et al. "Model extraction warning in mlaas paradigm". In: *Proceedings of the 34th Annual Computer Security Applications Conference*. 2018, pp. 371–380.

[52] Ibrahim M Alabdulmohsin, Xin Gao, and Xiangliang Zhang. "Adding robustness to support vector machines against adversarial reverse engineering". In: *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*. 2014, pp. 231–240.

[53] Buse Gul Atli et al. "Extraction of complex dnn models: Real threat or boogeyman?" In: *Engineering Dependable and Secure Machine Learning Systems: Third International Workshop, EDSMLS 2020, New York City, NY, USA, February 7, 2020, Revised Selected Papers 3*. Springer. 2020, pp. 42–57.

[54] Linda Guiga and AW Roscoe. "Neural Network Security: Hiding CNN Parameters with Guided Grad-CAM." In: *ICISSP*. 2020, pp. 611–618.

[55] Yi Shi and Yalin E Sagduyu. "Evasion and causative attacks with adversarial deep learning". In: *MILCOM 2017-2017 IEEE Military Communications Conference (MILCOM)*. IEEE. 2017, pp. 243–248.

[56] Huadi Zheng et al. "Bdpl: A boundary differentially private layer against machine learning model extraction attacks". In: *Computer Security–ESORICS 2019: 24th European Symposium on Research in Computer Security, Luxembourg, September 23–27, 2019, Proceedings, Part I 24*. Springer. 2019, pp. 66–83.

[57] Hervé Chabanne, Vincent Despiegel, and Linda Guiga. "A protection against the extraction of neural network models". In: *7th International Conference on Information Systems Security and Privacy* (2021).

[58] Hui Xu et al. "Deepobfuscation: Securing the structure of convolutional neural networks via knowledge distillation". In: *arXiv preprint arXiv:1806.10313* (2018).

[59] Kálmán Szentannai, Jalal Al-Afandi, and András Horváth. "Preventing Neural Network Weight Stealing via Network Obfuscation". In: *Intelligent Computing: Proceedings of the 2020 Computing Conference, Volume 3*. Springer. 2020, pp. 1–11.

[60] Gert W Wolf. "Facility location: concepts, models, algorithms and case studies." In: *Contributions to Management Science, International Journal of Geographical Information Science* (2011).

[61] Thomas M Cover and Joy A Thomas. "Information theory and statistics". In: *Elements of information theory* 1.1 (1991), pp. 279–335.

[62] Yarin Gal and Zoubin Ghahramani. "Dropout as a bayesian approximation: Representing model uncertainty in deep learning". In: *international conference on machine learning*. PMLR. 2016, pp. 1050–1059.

[63] Ye Zhang, Matthew Lease, and Byron Wallace. "Active discriminative text representation learning". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 31. 1. 2017.

[64] Burr Settles, Mark Craven, and Soumya Ray. "Multiple-instance active learning". In: *Advances in neural information processing systems* 20 (2007).

[65] David Arthur and Sergei Vassilvitskii. "K-means++ the advantages of careful seeding". In: *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. 2007, pp. 1027–1035.

[66] Irina Higgins et al. "beta-vae: Learning basic visual concepts with a constrained variational framework". In: *International conference on learning representations*. 2017.

[67] Jacob Goldberger and Sam Roweis. "Hierarchical clustering of a mixture model". In: *Advances in neural information processing systems* 17 (2004).

[68] David Lopez-Paz and Marc'Aurelio Ranzato. "Gradient episodic memory for continual learning". In: *Advances in neural information processing systems* 30 (2017).

[69] David JC MacKay and David JC Mac Kay. *Information theory, inference and learning algorithms, chapter 27: Laplace's method*. Cambridge university press, 2003.

[70] Abhishek Aich. "Elastic Weight Consolidation (EWC): Nuts and Bolts". In: *CoRR* abs/2105.04093 (2021). arXiv: 2105.04093. URL: https://arxiv.org/abs/2105.04093.

[71] Varun Chandrasekaran et al. "Exploring connections between active learning and model extraction". In: *29th USENIX Security Symposium (USENIX Security 20)*. 2020, pp. 1309–1326.

[72] Yi Shi et al. "Active deep learning attacks under strict rate limitations for online API calls". In: *2018 IEEE International Symposium on Technologies for Homeland Security (HST)*. IEEE. 2018, pp. 1–6.

[73] Melanie Ducoffe and Frederic Precioso. "Adversarial active learning for deep networks: a margin based approach". In: *arXiv preprint arXiv:1802.09841* (2018).

[74] Nathan Beck et al. "Effective evaluation of deep active learning on image classification tasks". In: *arXiv preprint arXiv:2106.15324* (2021).

[75] Samuel J Bell and Neil D. Lawrence. "The Effect of Task Ordering in Continual Learning". In: *arXiv preprint arXiv:2205.13323* (2022).

[76] Guy Hacohen and Daphna Weinshall. "On the power of curriculum learning in training deep networks". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 2535–2544.

[77] Anthony Robins. "Catastrophic forgetting, rehearsal and pseudorehearsal". In: *Connection Science* 7.2 (1995), pp. 123–146.

[78] Research Ministry of Science and the Arts Baden-Württemberg. *Information on BwUnicluster 2.0*. Accessed April 24, 2023. https://wiki.bwhpc.de/e/Category:BwUniCluster_2.0. 2022.

[79] Guido van Rossum. *Python tutorial*. Tech. rep. CS-R9526. Amsterdam: Centrum voor Wiskunde en Informatica (CWI), May 1995.

[80] Adam Paszke et al. "Pytorch: An imperative style, high-performance deep learning library". In: *Advances in neural information processing systems* 32 (2019).

[81] Jingzhao Zhang et al. “Why gradient clipping accelerates training: A theoretical justification for adaptivity”. In: *International Conference on Learning Representations* (2020).

[82] Samarth Sinha, Sayna Ebrahimi, and Trevor Darrell. *VAAL GitHub Repository*. Accessed May 2, 2023. `https://github.com/sinhasam/vaal`. 2020.

[83] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. *CIFAR (canadian institute for advanced research)*. Accessed May 4th, 2023. `https://www.cs.toronto.edu/~kriz/cifar.html`.

[84] Ya Le and Xuan Yang. “Tiny imagenet visual recognition challenge”. In: *CS 231N* 7.7 (2015), p. 3.

[85] Jia Deng et al. “Imagenet: A large-scale hierarchical image database”. In: *CVPR* (2009).

[86] Jia Deng et al. *ILSVRC2012-14 dataset*. Accessed May 9, 2023. `http://www.image-net.org/data/downsample/Imagenet32_32.zip`. 2014.

[87] Ashok Cutkosky and Harsh Mehta. “Momentum improves normalized sgd”. In: *International conference on machine learning*. PMLR. 2020, pp. 2260–2268.

[88] Peter Flach. “Performance evaluation in machine learning: the good, the bad, the ugly, and the way forward”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 33. 01. 2019, pp. 9808–9814.

[89] Xue Ying. “An overview of overfitting and its solutions”. In: *Journal of physics: Conference series*. Vol. 1168. IOP Publishing. 2019, p. 022022.

[90] Rishabh Iyer et al. “Submodular combinatorial information measures with applications in machine learning”. In: *Algorithmic Learning Theory*. PMLR. 2021, pp. 722–754.

[91] Hiran V Nath and Babu M Mehtre. “Static malware analysis using machine learning methods”. In: *Recent Trends in Computer Networks and Distributed Systems Security: Second International Conference, SNDS 2014, Trivandrum, India, March 13-14, 2014, Proceedings 2*. Springer. 2014, pp. 440–450.

[92] Muhammad Ijaz, Muhammad Hanif Durad, and Maliha Ismail. “Static and dynamic malware analysis using machine learning”. In: *2019 16th International bhurban conference on applied sciences and technology (IBCAST)*. IEEE. 2019, pp. 687–691.

[93] Research Ministry of Science and the Arts Baden-Württemberg. *Hardware Configuration of BwUnicluster 2.0*. Accessed April 24, 2023. `https://wiki.bwhpc.de/e/BwUniCluster2.0/Hardware_and_Architecture`. 2023.

A Appendix

A.1 Proof of Data points Trained Using Active Learning

Theorem 1. *Let X be a set of data points of size n, and let $b < n, n \mod b = 0$ be the batch size. Then a machine learning model trained using pool-based active learning is trained $\frac{n}{b}$ times on the current labeled pool of data points. Overall, the model is trained on $\frac{n(n+b)}{2b}$ data points.*

Proof. The first part is trivial, given that the model is trained until the labeled pool is exhausted and in each iteration b points are queried for their label. To determine the total number of points used, we need to sum up the number of points used in each iteration. These are

$$\sum_{i=1}^{\frac{n}{b}} i \cdot b. \tag{A.1}$$

Using the formula for the sum of the first n natural numbers, we get

$$\sum_{i=1}^{\frac{n}{b}} i \cdot b = b \cdot \sum_{i=1}^{\frac{n}{b}} i = b \cdot \frac{\frac{n}{b}(\frac{n}{b}+1)}{2} = \frac{n(\frac{n}{b}+1)}{2} = \frac{n(n+b)}{2b} \tag{A.2}$$

□

A.2 Experiment Setup

In this section, we will list tables and figures containing the specifications of the hardware and software used in our experiments.

A.2.1 Hardware

Since we run our experiments on BwUnicluster2.0, we provide a detailed list of the hardware specifications of the nodes we used. Please note that this table only contains the nodes we used in our experiments. For a more detailed list of all nodes as well as all further information, see [93].

	GPU x4	GPU x8	GPU x4 A100
Processors	Intel Xeon Gold 6230	Intel Xeon Gold 6248	Intel Xeon Platinum 8358
Number of sockets	2	2	2
Processor frequency (GHz)	2.1	2.6	2.5
Total number of cores	40	40	64
Main memory	384 GB	768 GB	512 GB
Local SSD	3.2 TB NVMe	15TB NVMe	6.4 TB NVMe
GPUs	4x NVIDIA Tesla V100	8x NVIDIA Tesla V100	4x NVIDIA A100
GPU Memory	32 GB	32 GB	80 GB
Interconnect	IB HDR	IB HDR	IB HDR200

Table A.2: Hardware configuration for the three nodes used on BWUniCluster2.0.

A.2.2 Software

In this section, we provide a comprehensive list of the libraries we used in our experiments as well as their version and a link to their documentation.

Library Name	Version	Link
numpy	1.23.0	`https://numpy.org/`
tqdm	4.64.1	`https://tqdm.github.io/`
torchvision	0.14.1	`https://pytorch.org/vision/stable/index.html`
torch	1.13.1	`https://pytorch.org/`
PyYAML	6.0	`https://pyyaml.org/`
scipy	1.10.1	`https://scipy.org/`
scikit-learn	1.2.1	`https://scikit-learn.org/stable/index.html`
wget	3.2	`https://pypi.org/project/wget/`

Table A.3: Software libraries used for the experiments.

A.2.3 Continual Learning strategies

In this section, we extend the parameter description of the continual learning strategies given in section 5.1.3 by presenting tables with the exact hyperparameter configurations used in the experiments.

Hyperparameter	Description	Value
λ	Balances between old and new tasks. A higher value indicates more focus on preserving knowledge from the old task.	1.0
Sample size	Relative size of the sample compared to the full training set that is used to compute the FIM.	0.05
Gradient clip	Maximum value of the l_2 norm of the gradient. If the current norm is larger, the gradient is clipped to that value.	20.0

Table A.5: Hyperparameter configuration for EWC.

Hyperparameter	Description	Value
λ	Balances between old and new tasks. A higher value indicates more focus on preserving knowledge from the old task.	1.0
Gradient clip	Maximum value of the l_2 norm of the gradient. If the current norm is larger, the gradient is clipped to that value.	2.0

Table A.7: Hyperparameter configuration for MAS.

Hyperparameter	Description	Value
λ	Balances between old and new tasks. A higher value indicates more focus on preserving knowledge from the old task.	1.0
Sample size	Relative size of the sample compared to the full training set that is used to compute the FIM.	0.05
Gradient clip	Maximum value of the l_2 norm of the gradient. If the current norm is larger, the gradient is clipped to that value.	20.0
α	Weights the importance of the previous tasks. The sum of all α values must be 1.0.	0.45,0.55

Table A.9: Hyperparameter configuration for IMM.

Hyperparameter	Description	Value
c	Balances between old and new tasks. A higher value indicates more focus on preserving knowledge from the old task.	0.5
c'	Used to update parameter importances in equation 3.23.	0.25
Gradient clip	Maximum Value of the l_2-Norm of the gradient. If the current norm is larger, the gradient is clipped to that value.	2.0
a	Determines the overestimation of the loss on the unobserved side.	3.0
a'	Used to update parameter importances in equation 3.23.	1.5

Table A.11: Hyperparameter configuration for ALASSO.

A.2.4 Datasets

In our experiments, we use the datasets MNIST, CIFAR-10, CIFAR-100, Tiny ImageNet and Small ImageNet. In the following, we give a detailed overview of their properties.

	MNIST	CIFAR-10	CIFAR-100	Tiny ImageNet	Small ImageNet
Size of training set	60,000	60,000	50,000	100,000	128,116
Size of validation set	10,000	10,000	10,000	10,000	50,000
Number of classes	10	10	100	200	1,000
Image Dimension	28x28	32x32	32x32	64x64	32x32
Number of channels	1	3	3	3	3
Source	torchvision	torchvision	torchvision	`http://cs231n.stanford.edu/tiny-imagenet-200.zip`	`http://www.image-net.org/data/downsample/Imagenet32_32.zip`

Table A.13: Information on the datasets used for the experiments.

A.2.5 Neural Network Architectures

In our experiments with model stealing, we use the same architectures as proposed by Pal et al. [8]. They introduce a proprietary CNN architecture family which consists of l convolution blocks followed by a single fully connected layer. Each convolution block consists of

1. a convolutional layer with k filters of size 3×3,
2. a ReLU activation layer,
3. a batch normalization layer,
4. a convolutional layer with k filters of size 3×3,
5. a ReLU activation layer,
6. a batch normalization layer,
7. a max pooling layer with kernel size 2×2,

where $k = 32 \cdot 2^{i-1}$ is the number of filters for block number $i \in \{1, \ldots, l\}$.

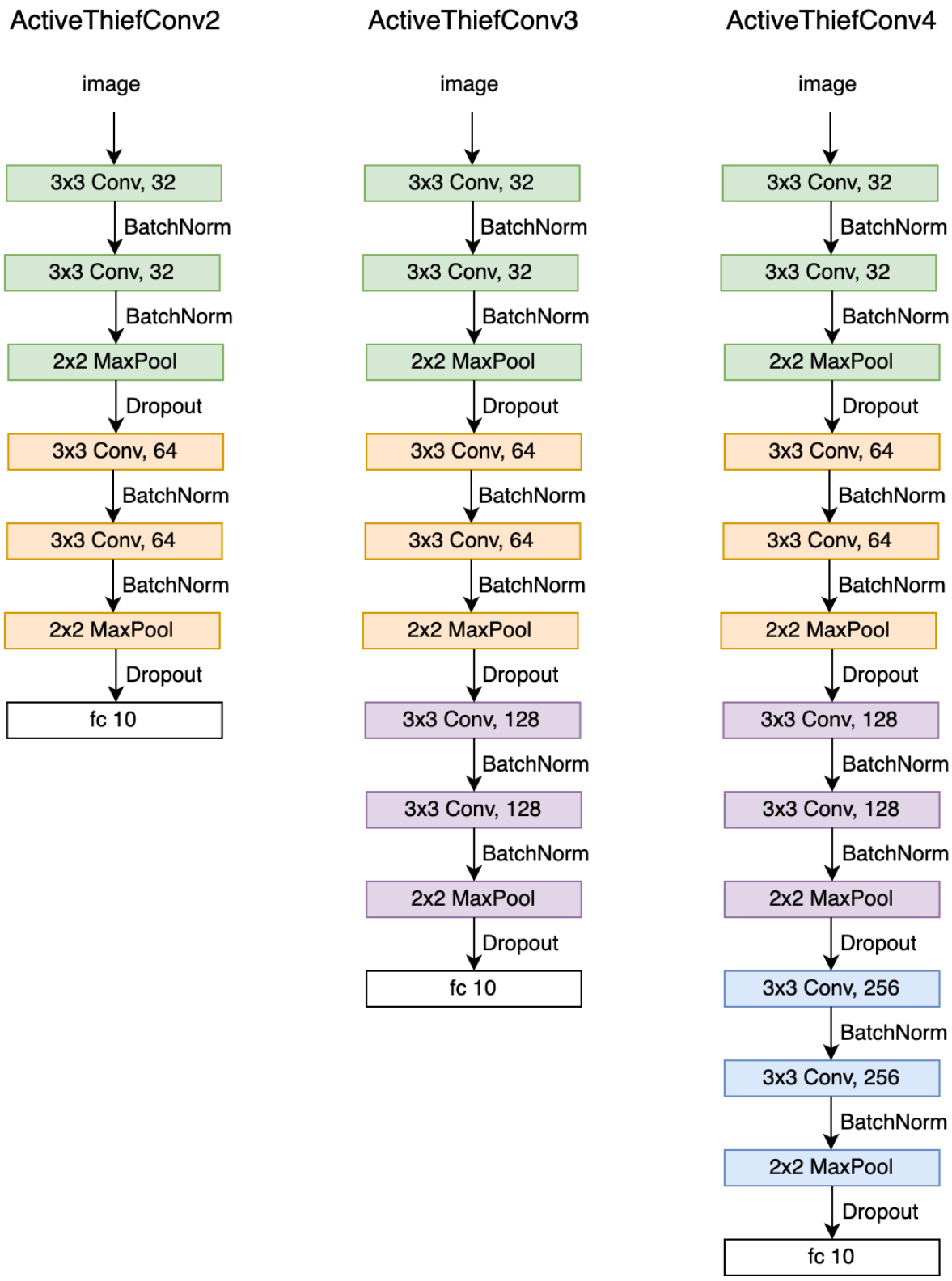

Figure A.1: Example network architectures for CIFAR-10.

A.3 Continual Active Learning

In section 6.1.1, we evaluate the performance of regularization-based continual learning strategies combined with the active learning strategies Random, LC, BALD, CoreSet and BADGE. For space reasons, we omitted the results of our experiments for batch sizes 1,000 and 2,000 in the main paper. Therefore, we list the results for these batch sizes in the following. The results by validation accuracy are shown in figure A.2 and figure A.3, respectively. The execution time of these experiments is presented in table A.14 and table A.15, respectively.

Overall, we observe similar patterns to the results presented in the main paper, regarding both the relative performance of the active learning strategies and the continual learning strategies. However, the results for batch size 1,000 exhibit high variance, which is caused by overfitting on the small training set.

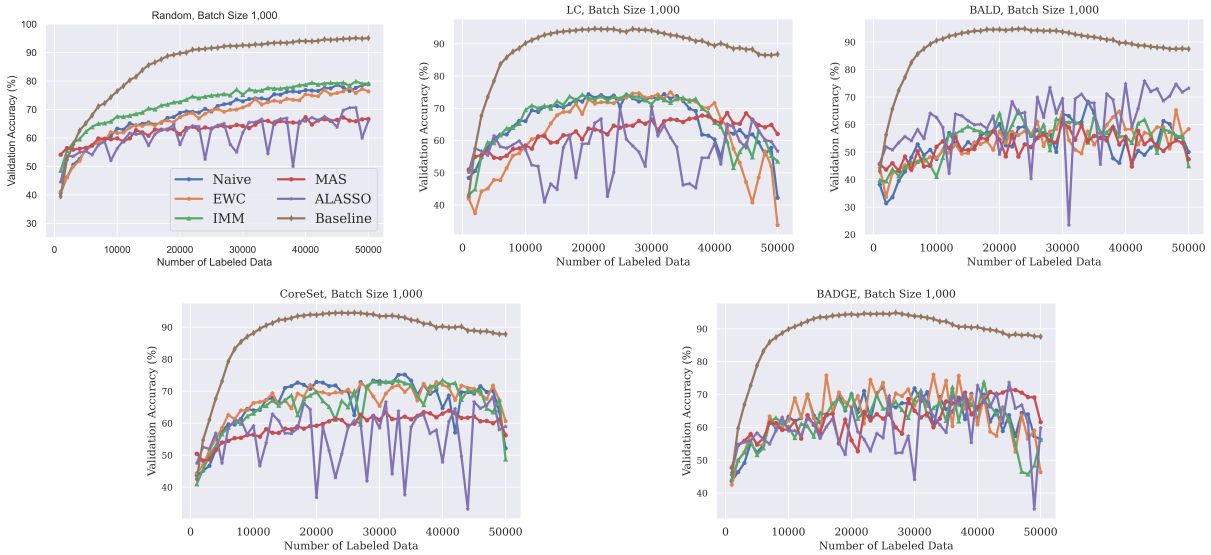

Figure A.2: Comparison of validation accuracy of continual active earning strategies with batch size 1,000.

	Random	LC	BALD	CoreSet	BADGE
Baseline	2,522	2,536	2,549	2,650	3,171
Naive	44	50	50	144	493
EWC	44	53	51	140	486
IMM	42	49	48	137	501
MAS	48	55	53	147	505
ALASSO	68	75	75	168	509

Table A.14: Comparison of execution time of regularization-based continual learning strategies with batch size 1,000.

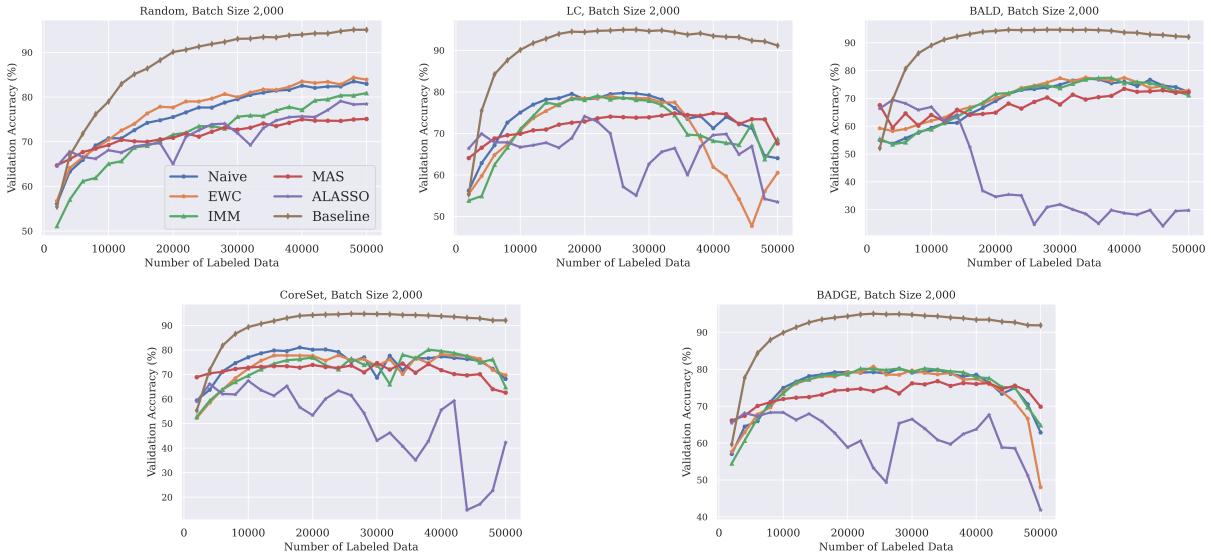

Figure A.3: Comparison of validation accuracy of continual active learning strategies with batch size 2,000.

	Random	LC	BALD	CoreSet	BADGE
Baseline	1,290	1,303	1,315	1,342	1,935
Naive	76	80	78	162	523
EWC	83	90	86	152	513
IMM	77	76	78	145	500
MAS	84	84	85	161	515
ALASSO	119	131	122	188	547

Table A.15: Comparison of execution time of regularization-based continual learning strategies with batch size 2,000.

A.4 Model Stealing

A.4.1 Evaluation of ActiveThief

Because we build our continual active learning approach upon the ActiveThief framework, we believe it is important to rigorously evaluate it before we apply continual active learning to it. In the ActiveThief paper [8], Pal et al. evaluate active learning for model stealing for computer vision and natural language processing. Since this thesis focuses on computer vision tasks, we only investigate the part of the ActiveThief framework that deals with computer vision. For the ActiveThief framework, Pal et al. introduce a proprietary thief dataset, which we dubbed Small ImageNet, and three different target model architectures, which we call ActiveThiefConv2, ActiveThiefConv3 and ActiveThiefConv4. In the following, we evaluate the influence of the thief dataset and the target model architecture on the success of the model stealing attack.

We start off by computing the validation accuracies of the ActiveThiefConv model family, which the ActiveThief framework and ours use as target and substitute models. These numbers have not been given by the authors of ActiveThief in their paper. However, we believe they are crucial to understanding the performance of the ActiveThief framework and our continual active learning approach. Each model of the ActiveThiefConv family is trained on MNIST and CIFAR-10. Additionally, we train ActiveThiefConv3 on CIFAR-100, because we will conduct experiments with CIFAR-100 in section 6.2. We omit the results for ActiveThiefConv2 and ActiveThiefConv4 on CIFAR-100 because they are not relevant to this thesis. The results are depicted in table A.16. They show that ActiveThiefConv4, the most complex model, achieves the highest validation accuracy on both MNIST and CIFAR-10. While the validation accuracy for MNIST is above 98 % for all models, there is a difference in validation accuracy of almost 20 percentage points between ActiveThiefConv2 and ActiveThiefConv4 on CIFAR-10. This shows that the complexity of the target model architecture has a significant influence on validation accuracy.

	ActiveThiefConv2	ActiveThiefConv3	ActiveThiefConv4
MNIST	98.42	98.91	99.01
CIFAR-10	66.61	80.67	84.47
CIFAR-100	-	42.90	-

Table A.16: Validation accuracies (in %) of our target model architectures on MNIST, CIFAR-10 and CIFAR-100.

Next, we evaluate the influence of the target model architecture and substitute model architecture on the success of the model stealing attack. We use the ActiveThiefConv model family as target and substitute models and perform one model stealing attack for each combination of target and substitute model. We use the active learning strategy CoreSet with a batch size of 1,000 and a total budget of 20,000 to perform the attacks. The results are depicted in figure A.5. The plots in the figure represent the progression of agreement between the target and substitute model on the validation set of the CIFAR-10 dataset at the end of each experiment. Overall, we observe that the agreement between the target and substitute model is highest when we use a target model of low complexity (i.e., ActiveThiefConv2) and a substitute model of moderate to high complexity (i.e., ActiveThiefConv3 and ActiveThiefConv4). This is in

line with the results of the ActiveThief paper [8]. However, we report the accuracy after a budget of 20,000, whereas Pal et al. presumably report the accuracy after training on the full thief dataset. To further study the behavior of model stealing attacks using different model architectures, we conduct the same experiment using MNIST as the target model dataset. The results of this experiment can be found in figure A.4. Overall, the results are similar to those on CIFAR-10. However, it is evident that the discrepancies between the target and substitute model combinations are larger.

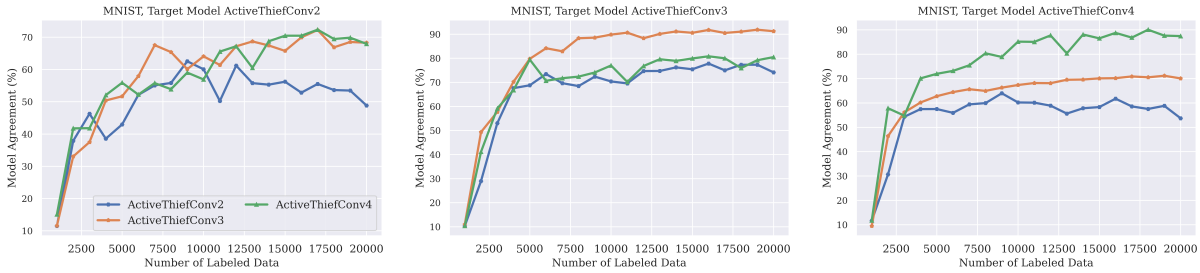

Figure A.4: Agreement progression with varying model architectures on MNIST.

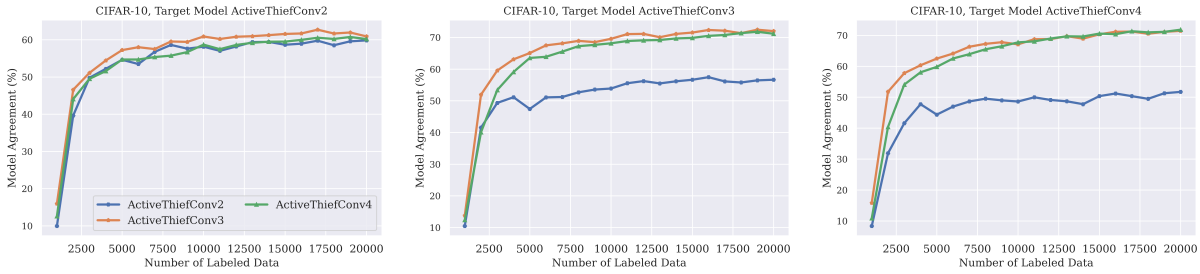

Figure A.5: Agreement progression with varying model architectures on CIFAR-10.

After evaluating the influence of the target and substitute model architecture on the success of the model stealing attack, we evaluate how the thief dataset effects model stealing attacks. Our motivation for this experiment is that Pal et al. mention having tested CIFAR-10 as a thief dataset without success. We perform a model stealing attack using ActiveThiefConv3 as the target and substitute model and CoreSet with batch size 2,000 as our active learning strategy. The total query budget is 20,000. We use MNIST as the target model dataset and change between Small ImageNet, Tiny ImageNet, and CIFAR-10 as our thief datasets. The results of this experiment can be seen in figure A.6. While model agreement increases throughout the experiment when using Small ImageNet, the model agreement for Tiny ImageNet plateaus after just three iterations at 20%. Model stealing with CIFAR-10 yields the best results, however. At the end of the experiment, the setup with CIFAR-10 as the thief dataset achieves a model agreement five percentage points higher than the setup with Small Imagenet.

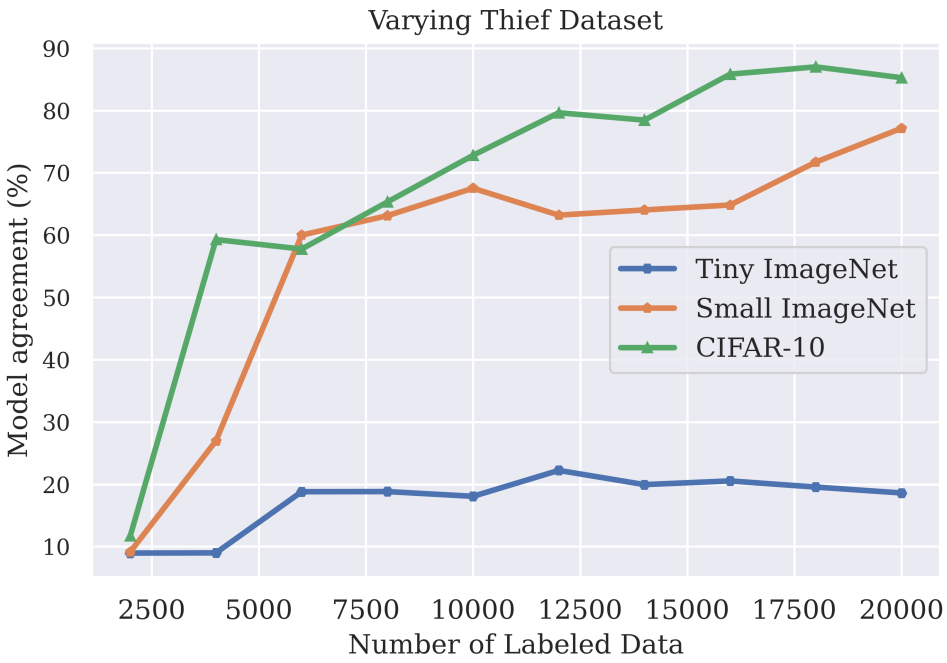

Figure A.6: Comparison of model agreement for various thief datasets.

A.4.2 Continual Active Learning for Model Stealing

This section contains plots of the complete runs of all experiments in section 6.2.

MNIST

In this section, we present the complete runs of all experiments which involve continual active learning using MNIST as a target model dataset.

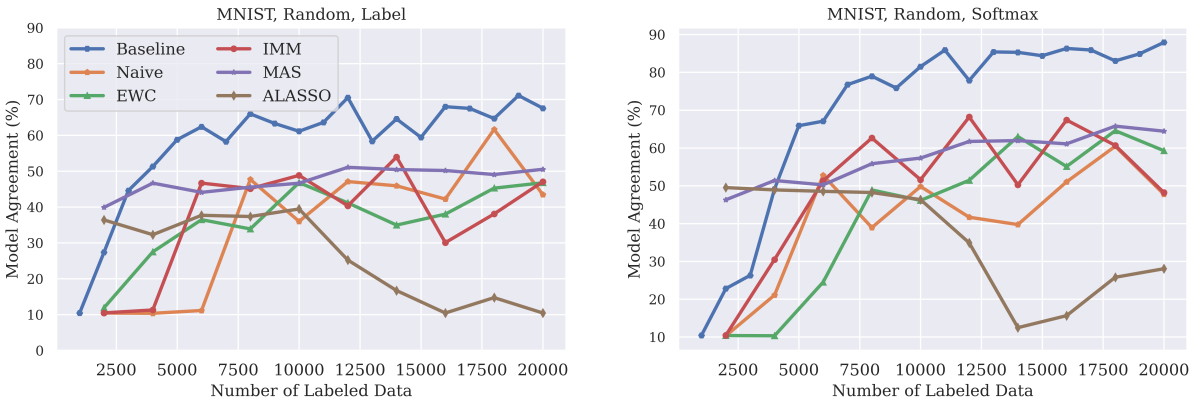

Figure A.7: Agreement comparison for model stealing on MNIST using the active learning strategy Random.

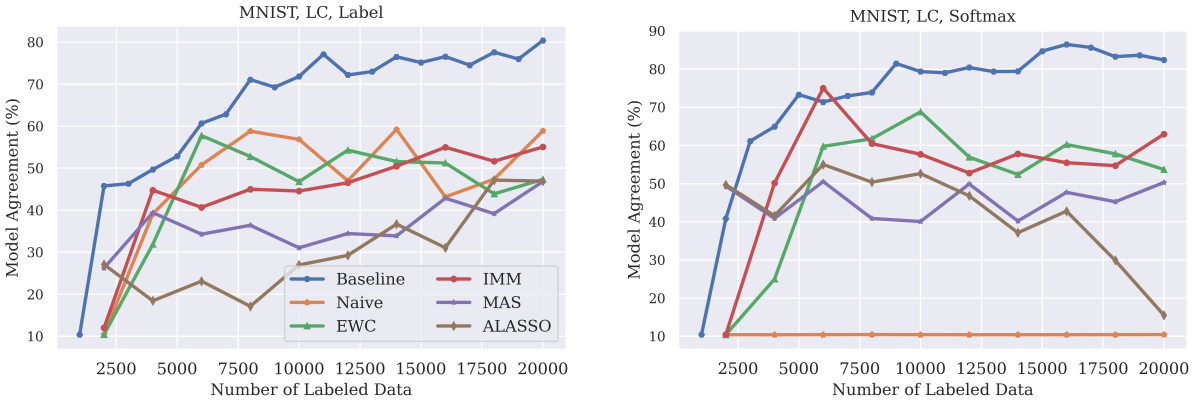

Figure A.8: Agreement comparison for model stealing on MNIST using the active learning strategy LC.

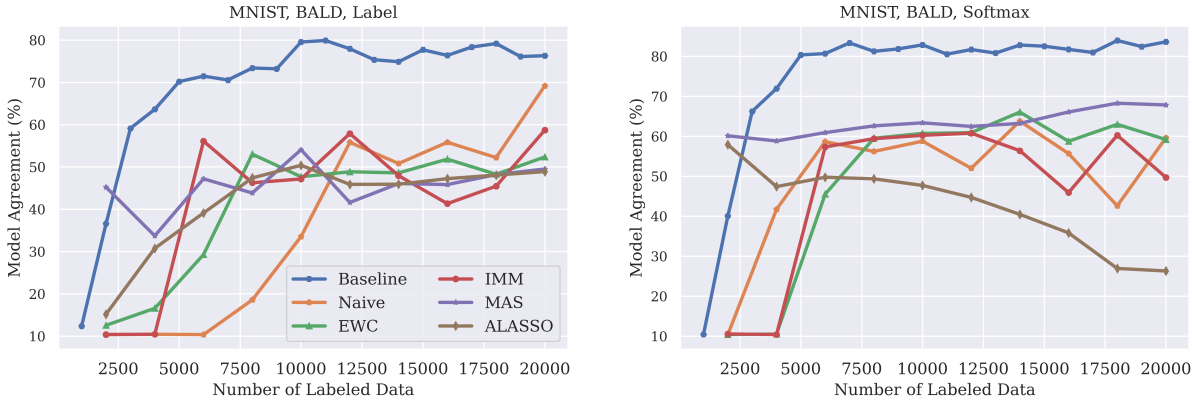

Figure A.9: Agreement comparison for model stealing on MNIST using the active learning strategy BALD.

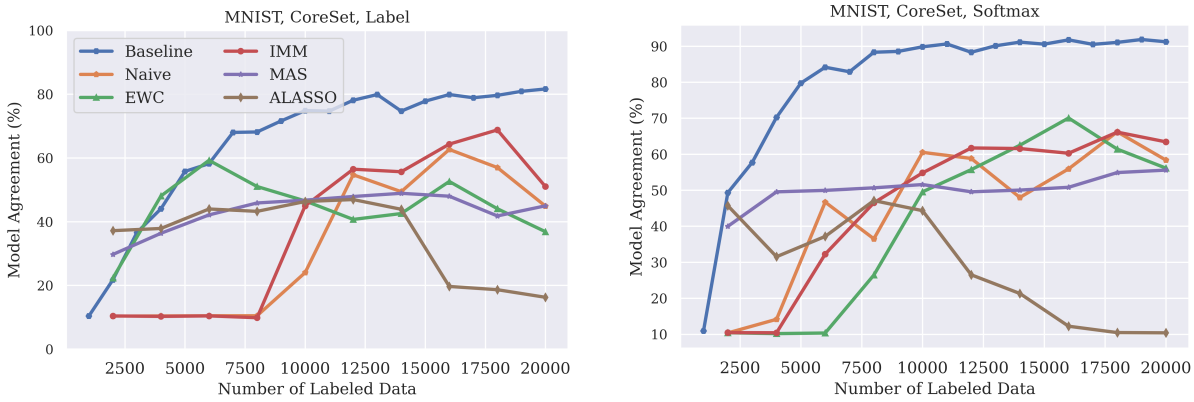

Figure A.10: Agreement comparison for model stealing on MNIST using the active learning strategy CoreSet.

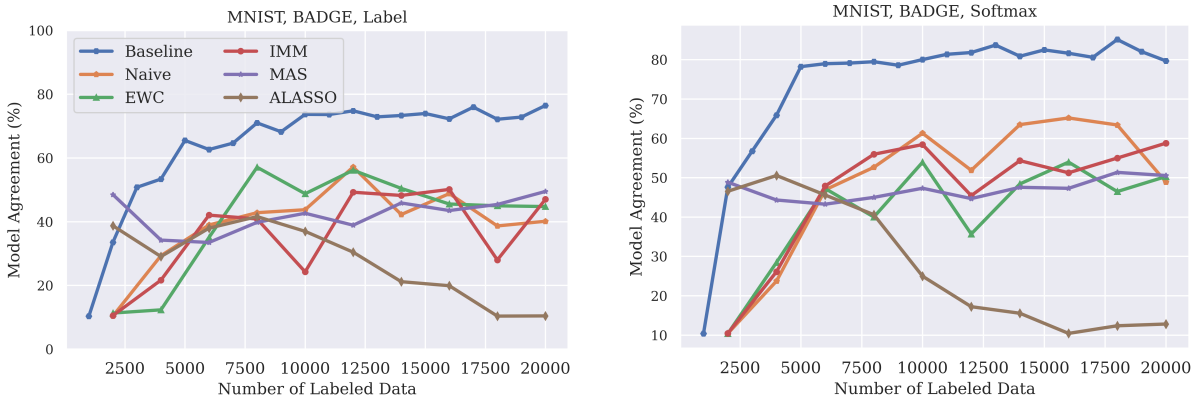

Figure A.11: Agreement comparison for model stealing on MNIST using the active learning strategy BADGE.

CIFAR-10

In this section, we present the complete runs of all experiments which involve continual active learning using CIFAR-10 as a target model dataset.

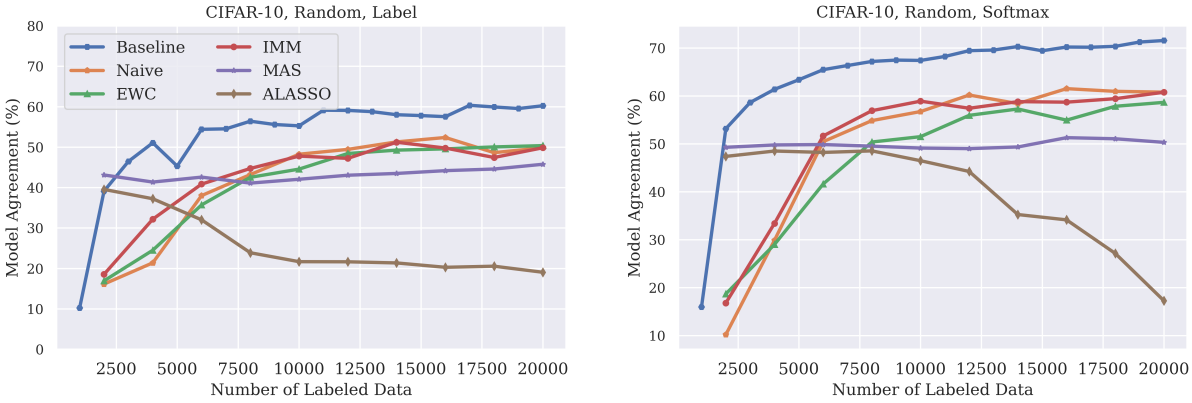

Figure A.12: Agreement comparison for model stealing on CIFAR-10 using the active learning strategy Random.

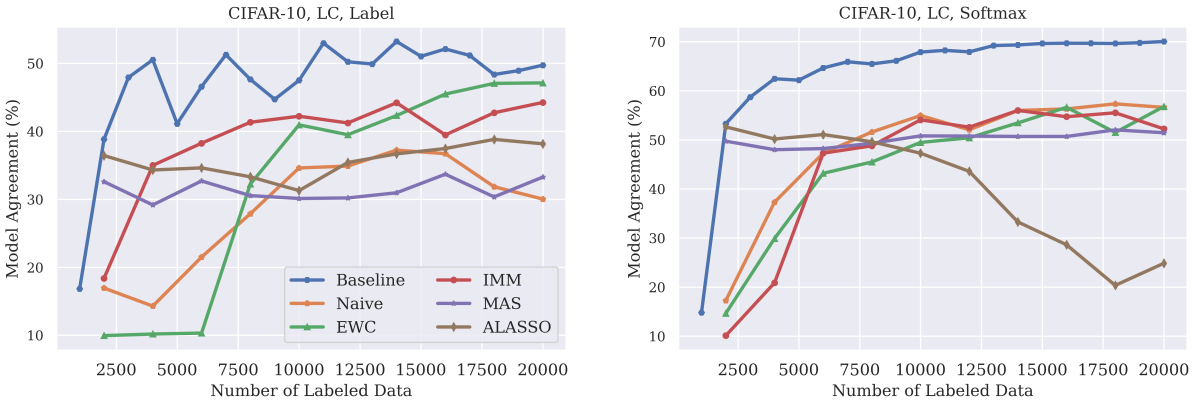

Figure A.13: Agreement comparison for model stealing on CIFAR-10 using the active learning strategy LC.

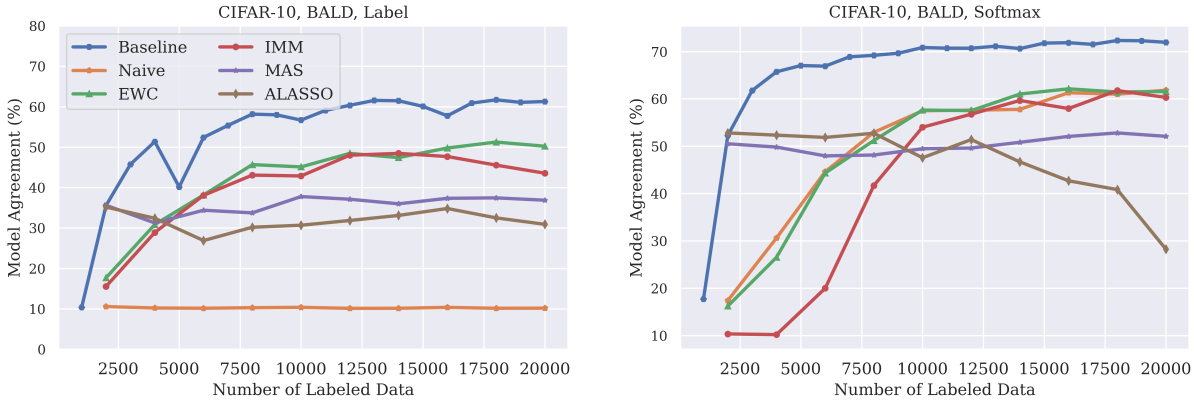

Figure A.14: Agreement comparison for model stealing on CIFAR-10 using the active learning strategy BALD.

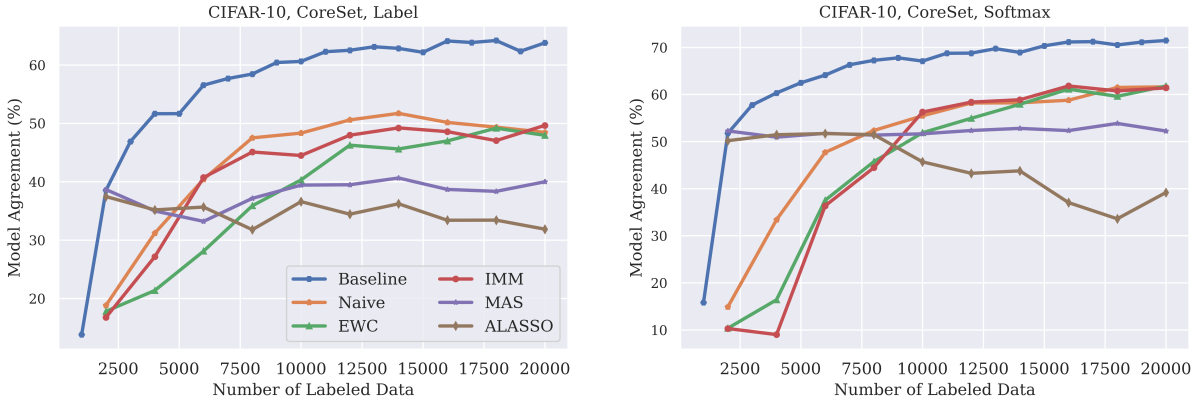

Figure A.15: Agreement comparison for model stealing on CIFAR-10 using the active learning strategy CoreSet.

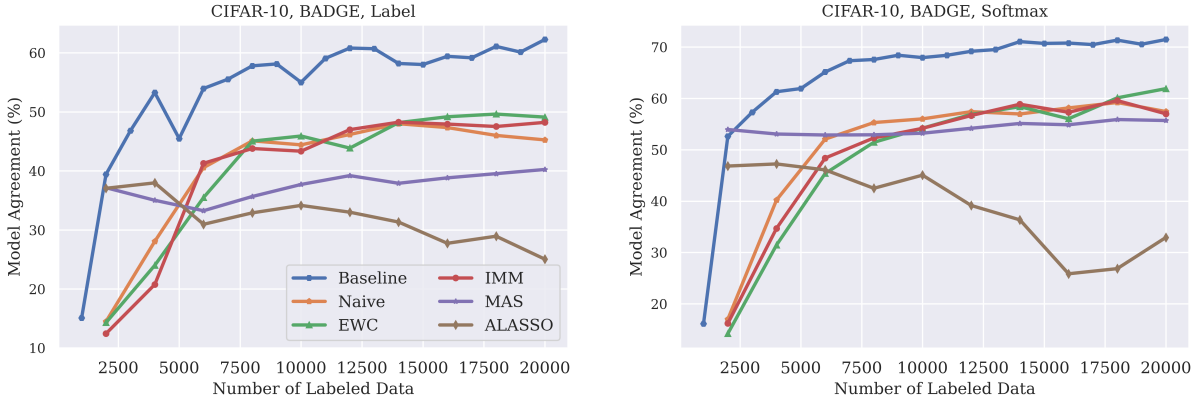

Figure A.16: Agreement comparison for model stealing on CIFAR-10 using the active learning strategy BADGE.

CIFAR-100

In this section, we present the complete runs of all experiments which involve continual active learning using CIFAR-100 as a target model dataset.

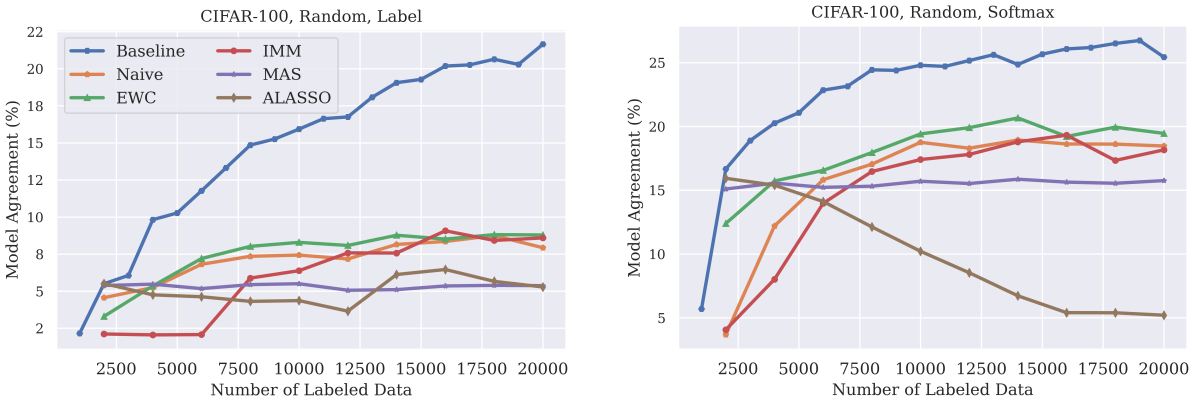

Figure A.17: Agreement comparison for model stealing on CIFAR-100 using the active learning strategy Random.

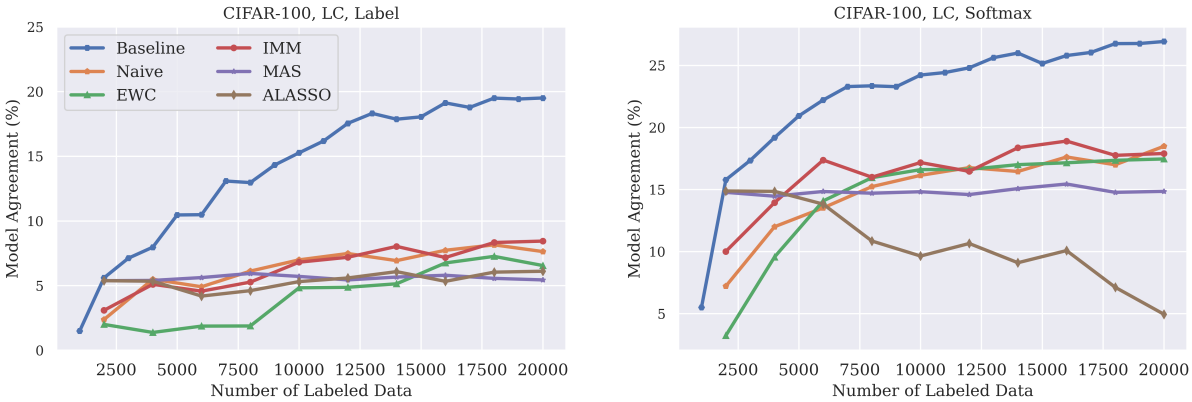

Figure A.18: Agreement comparison for model stealing on CIFAR-100 using the active learning strategy LC.

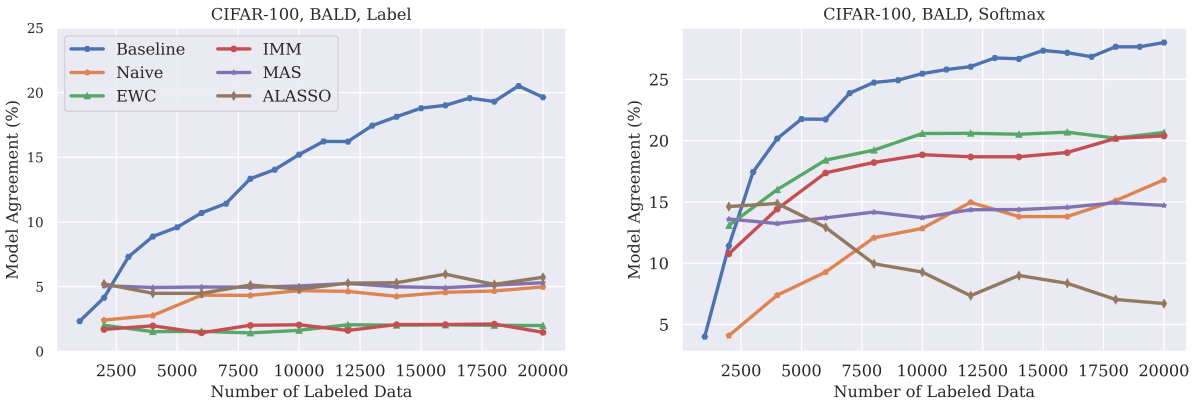

Figure A.19: Agreement comparison for model stealing on CIFAR-100 using the active learning strategy BALD.

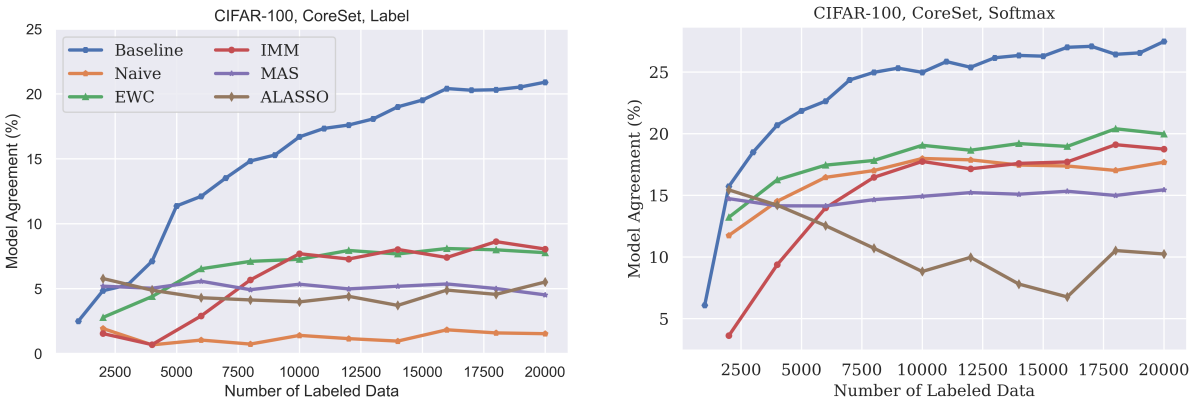

Figure A.20: Agreement comparison for model stealing on CIFAR-100 using the active learning strategy CoreSet.

VAAL and A-GEM

In the following we list the complete results for the experiments using VAAL and A-GEM on the datasets MNIST, CIFAR-10 and CIFAR-100.

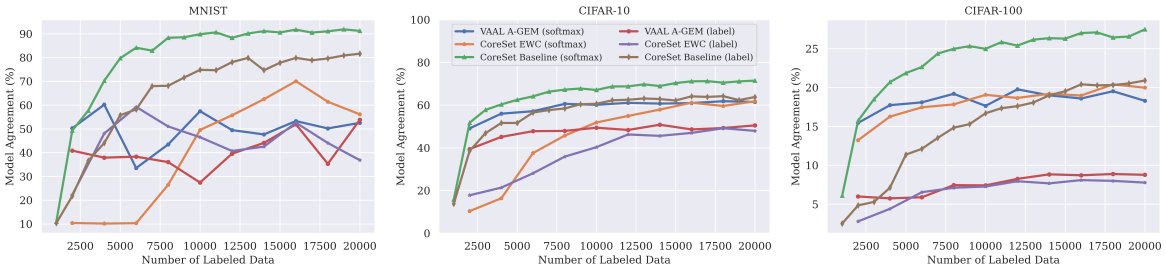

Figure A.21: Agreement comparison for model stealing using VAAL and A-GEM.

A.4.3 Effect of Data Augmentation on Model Stealing Attacks

While we managed to reproduce the findings of Pal et al. for CIFAR-10, we failed to do so for MNIST. Since multiple training hyperparameters of the experiments presented in ActiveThief were not disclosed by the authors, we experiment with different combinations of hyperparameters, including varying the optimizer, the learning rate, the number of training epochs, the batch size, and switching between cold start and warm start for active learning. Sadly, we were still not able to reproduce the results of ActiveThief on MNIST. After conducting the experiments in section 6.2, we decided to investigate another hyperparameter: training with or without data augmentation. For this set of experiments, we leave all hyperparameters equal to the previous experiments, apart from data augmentation. We underline that using or not using data augmentation for model stealing refers to the thief dataset, not the target model dataset. Training the target model is always done using data augmentation. We perform the same model stealing attacks as before, using MNIST and CIFAR-10 as the target model dataset and present the results in figure A.22. When performing model stealing attacks without data augmentation on CIFAR-10, we notice that model agreement, both for active learning and continual active learning, is significantly lower than when using data augmentation. In this scenario, continual active learning with data augmentation performs on par with active learning without data augmentation. However, when using MNIST as the target model dataset, the results are reversed. Here, model agreement is significantly lower when using data augmentation. More importantly, continual active learning without data augmentation shows performance comparable to active learning with data augmentation.

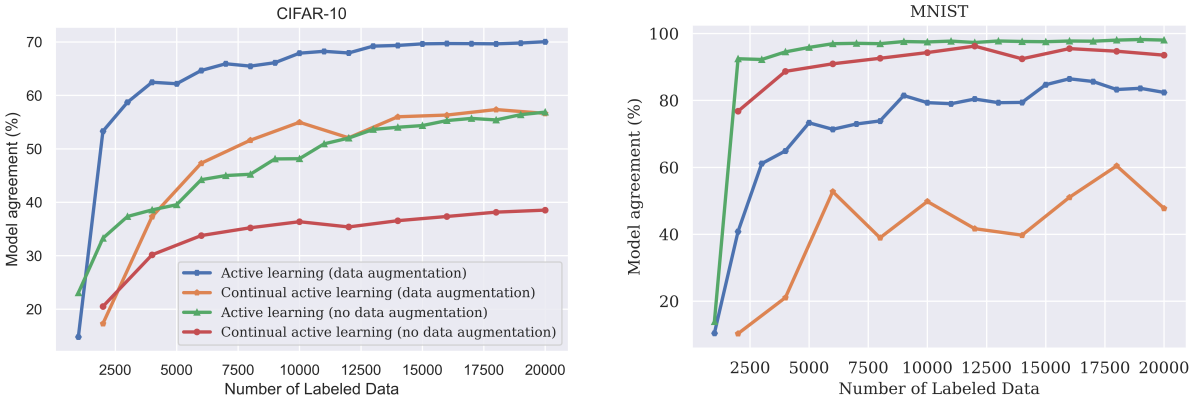

Figure A.22: Comparison of model agreement with and without data augmentation.