Institut für Robotik und Mechatronik	BJ.: 2020 IB.Nr.: DLR-IB-RM-OP-2020-115

MASTERARBEIT

A PERSISTENT INCREMENTAL LEARNING APPROACH FOR OBJECT CLASSIFICATION OF UNSEEN CATEGORIES USING CONVOLUTIONAL NEURAL NETWORKS ON MOBILE ROBOTS

Freigabe:	Der Bearbeiter:	Unterschriften
	Markus Knauer	
	Betreuer:	
	Maximillian Denninger	
	Der Institutsdirektor	
	Prof. Alin Albu-Schäffer	

Dieser Bericht enthält 120 Seiten, 50 Abbildungen und 33 Tabellen

Ort: Oberpfaffenhofen	Datum: 08.09.2020	Bearbeiter:	Zeichen:

Masterarbeit

Studiengang Informatik
Master of Science

A Persistent Incremental Learning Approach for Object Classification of Unseen Categories using Convolutional Neural Networks on Mobile Robots

Markus Knauer

Matrikelnummer
305721

Aufgabensteller/Prüfer	Prof. Dr. Jürgen Brauer
Zweitkorrektor	Prof. Dr. Jochen Staudacher
Arbeit vorgelegt am	28.08.2020
durchgeführt in der	Fakultät Informatik
durchgeführt bei	Deutsches Zentrum für Luft- und Raumfahrt e. V. (DLR), Oberfaffenhofen, Institut für Robotik und Mechatronik, Perzeption und Kognition (RM/PEK-OP)
Betreuer	Maximilian Denninger, RM/PEK-OP PD Dr. Rudolph Triebel, RM/PEK-OP Leitung
Anschrift des Verfassers	Romatsrieder Straße 27, 87654 Friesenried markus.w.knauer@stud.hs-kempten.de

Abstract - German

Die Verwendung von neuronalen Netzen, insbesondere von Convolutional Neural Networks (CNN), hat in vielen Bereichen, wie etwa der Objekt-Klassifikation, zu guten Ergebnissen geführt. Diese Netze sind jedoch dadurch limitiert, dass sie nur einmal trainiert und anschließend für die Auswertung von ähnlichen Daten eingesetzt werden. Es können also nur diejenigen Aufgaben durchgeführt werden, welche zu Beginn gelernt wurden. Kommen nun neue Aufgaben hinzu, muss das Netzwerk nochmals mit allen Daten neu trainiert werden. Diese Netze sind daher normalerweise nicht in der Lage, kontinuierlich hinzulernen zu können. In dieser Arbeit wird ein neuer Ansatz vorgestellt, welcher in der Lage ist fortlaufend neue, bisher unbekannte Objekt-Kategorien auf Bildern hinzulernen zu können. Das Ziel hierbei ist eine Anwendung im Bereich der mobilen Robotik. Zunächst werden hierzu verschiedene Strategien vorgestellt, welche die Architektur des Netzwerks dynamisch erweitern, sobald neue Kategorien hinzugelernt werden sollen. Im ersten Ansatz wird die letzte Neuronenschicht des Netzwerkes dynamisch an die Anzahl der Kategorien angepasst. Der zweite Ansatz erweitert diese Technik, indem für jede Sequenz von neuen Kategorien zusätzliche Neuronenschichten angelegt werden. Um zu verhindern, dass das Netzwerk bisher Erlerntes wieder vergisst, werden zusätzlich verschiedene Regularisierungsstrategien vorgestellt. Unter anderem auch eine neue Methode, bei welchem der Classification-Loss durch einen Regression-Loss ersetzt wird. Für das Problem einer Ungleichverteilung der Output-Werte, wird ein neuartiger Ansatz vorgestellt, bei dem die Output-Werte durch die Varianz der jeweiligen Kategorie geteilt, und somit ausgeglichen werden. Zu dieser Arbeit gehört auch ein neuer Datensatz für Continual Learning, welcher speziell für die Service-Robotik entwickelt wurde (HOWS-CL-25). Dieser besteht aus 150.795 synthetischen Farbbildern von 25 verschiedenen Haushaltsobjektkategorien. Der hier vorgestellte Ansatz wird in den Bereich des Online Learnings eingeordnet. Dies ist eine spezielle Art des Incremental Learnings, bei welchem das Netzwerk lediglich Zugriff auf die Trainingsbilder der aktuellen Sequenz hat und vorherige Trainingsbilder nicht gespeichert werden dürfen. Diese Methode wird auch als Rehearsal-free bezeichnet. Online Learning ist ein noch ungelöstes Problem und komplexer, als andere inkrementelle Ansätze, welche Zugriff auf alte Trainingsbilder haben. Der Ansatz dieser Arbeit wurde auf verschiedenen Datensätzen evaluiert und mit anderen Ansätzen aus der Literatur verglichen. Dies beinhaltet auch eine Evaluation auf dem CLVISION Workshop der CVPR 2020.

Abstract

Neural Networks and especially Convolutional Neural Networks (CNN) show remarkable results in many fields, among others in object classification and recognition. But these networks are limited by the tasks they are trained on, as they are designed to learn all tasks they will need during their lifetime in the beginning and hence are frozen. If now new tasks arrive, the network has to be trained completely new. These networks are therefore usually not able to learn in a continual manner, like humans are capable of. In this work, a novel approach is presented, where a deep neural network is used to continually learn new unseen object categories on images, which can be used in different fields, like mobile robots. First, different architectural strategies are proposed to dynamically adapt the network according to the categories it learns over time. This includes one strategy, where the last layer of our network is adapted and another one where multiple fully-connected layers are created for each new sequence. In order to prevent forgetting, different regularization strategies are shown, including a novel loss function where the classification is replaced by a regression. So, it is ensured that already learned categories are not forgotten by simultaneously enabling the network to learn new categories. Furthermore, the emerging problem of a discrepancy in the output distribution is recognized and different solutions are proposed. This includes a novel regularization strategy, where the outputs are divided by the variance per category. Finally, a novel dataset for continual learning is presented, which is especially suited for object recognition in our mobile robot environment (HOWS-CL-25). It consists of 150,795 synthetic images of 25 different household object categories in a randomly changing environment. Our approach can be classified as online learning, a special variant of incremental learning, where one is limited by the data the network can observe in a specific time step, without the access to previous training examples - also called rehearsal-free. This is a challenging and unsolved problem in comparison to other incremental learning approaches, which also use previous training examples, but as this thesis is focusing on an approach for mobile robots, online learning is more relevant. Our approach is tested on different datasets and compared with other solutions from literature. Additionally, our method was evaluated in the CLVISION workshop at CVPR 2020.

Acknowledgements

I would like to thank my supervisor Maximilian Denninger, as well as the rest of my team at DLR, who helped me through my whole master thesis. Additionally, I would like to thank Dr. Rudolph Triebel, who made it possible for me to research in such an exciting field at an outstanding research institute. I had the chance to deepen my knowledge within a high class research environment, at workhops and hackatons, especially in the field of Computer Vision, Deep Learning and Continual Learning, which I will benefit greatly from. I would also like to thank my professor Prof. Dr. Jürgen Brauer and his PhD. Student Bonifaz Stuhr for all the good advises in an open and friendly environment. Thanks to all of them, I had a great environment to write my masters thesis, where I also had a lot of fun, researching on Continual Learning.

Contents

List of figures

List of tables

Table of Abbreviations

ADAM	Adaptive moment estimation - optimizer [1]
AlexNet	CNN architecture [2]
AR1	Continal learning strategy, which combines architectural and regularization techniques [3]
BlenderProc	A procedural Blender pipeline for photorealistic training image generation [4, 5]
CIFAR-100	Canadian Institute For Advanced Research dataset [6]
CL	Continual Learning
CLVISION	Continual Learning on computer vision, workshop at CVPR 2020 [7]
CNN	Convolutional Neural Network
COIL-100	Columbia Object Image Library [8]
CORe50	A dataset and benchmark for Continual Learning, Object Recognition, Detection and Segmentation [9]
CRELU	Concatenated Rectified Linear Unit - activation function [10]
CVPR	IEEE Conference on Computer Vision and Pattern Recognition
CWR	Copy weights with re-init [9]
CWR+	extended, improved CWR [3]
DLR	Deutsches Zentrum für Luft- und Raumfahrt e.V.
EWC	Elastic Weight Consolidation [11]
ELU	Exponential Linear Unit - activation function [12]
GAN	Generative Adversarial Network [13]
GEM	Gradient Episodic Memory [14]
GLO	Generative Latent Optimization [15]
GP	Gaussian Processes [16]

GoogLeNet	CNN architecture [17]
HOWS-CL-25	Household Objects Within Simulation dataset for Continual Learning (ours)
iCaRL	Incremental Classifier and Representation Learning [18]
iCIFAR-100	Modified CIFAR-100 dataset to be incremental [18]
ImageNet	One of the biggest public image databases [19]
Inception-v3	CNN architecture [20]
InceptionResNet	CNN architecture [21]
KNN	K-nearest neighbor
LWF	Learning Without Forgetting [22]
MNIST	database of handwritten digits [23]
NC	Multi-task new classes scenario of CORe50 dataset
NI	New Instances scenario of CORe50 dataset
OP	Company location Oberpfaffenhofen
PAL	Persistent Anytime Learning [24]
PEK	Department for Perception und Cognition
PNN	Progressive Neural Network [25]
RELU	Rectified Linear Unit - activation function [26]
ResNet50	CNN architecture [27]
ResNet50V2	Second version of ResNet50 - CNN architecture [28]
RM	Institut of Robotics und Mechatronics
SELU	Scaled Exponential Linear Units - activation function [29]
SGD	Stochastic gradient descent - optimizer [30]
SI	Synaptic Intelligence [31]

SIREN	Implicit Neural Representations with Periodic - activation functions [32]
TFRecord	Data-format from tensorflow, which is optimized for streaming data over a network
VGG-16	CNN architecture [33]

1 Motivation

In the past decade, the demand for social caregivers has risen drastically, especially in countries like Germany or Japan, where an ageing society is already at an advanced stage. This has lead to a reinforced research pull in the field of service robotics. Robots are already in use to assist the human work in industry or in places where it is too dangerous for humans, like in outer space. Therefore, robots showed to be predestined for assisting humans. In social care, they could help elderly care takers to grasp objects, open doors, getting out of bed, alarm others in a case of danger or accident and much more. These robots need an elaborate understanding of their environment including the capabilities to learn new things on the fly. As humans have this skill since birth, it is hard for them to comprehend the limitations of robots, which usually can only learn once and are henceforth unable to learn new things.

For a better understanding figure (1.1) is provided, where Justin, a service robot from the DLR is shown. There he finds three new objects, he has never seen before. As a service robot, he is operating in a constantly changing environment. So Justin should be

Figure 1.1: Continual learning example on our robot Justin: As a service robot, Justin is working in a constantly changing environment. On this picture he found three new objects he has never seen before. This thesis is motivated to solve the problem allowing Justin to learn new object categories on the fly without forgetting already learned ones.

able to learn these objects, even if he was not trained on them before. This thesis tries to tackle this problem of continual learning, where new things are introduced during the lifetime of our robotic system.

In the field of incremental learning, the thesis introduces a novel online learning approach, to be able to learn new object categories, when the need arises. In order to get closer to a future, where mobile robots can be deployed in everyday homes help and serve our caregivers.

The ability to continually learn new tasks, in our case object categories, therefore is highly rewarding. Incremental learning can be used in other fields, too. Robots in industrial production are used for welding, grapping, lifting, etc. Those robots are able to only fulfill one task, they are programmed for. As soon as one of the component characteristics, like the material or the size changes, the robot is not able to carry out the work step anymore, without being completely reprogrammed. Our approach could be used there to learn those new categories, so that the robot knows what it's dealing with. A robot which is able to dynamically adapt to new conditions, by only learning new tasks while still be able to conduct the old tasks, would save a lot of money by reducing machine downtime.

Our approach can be used outside of the field of robotic as well. For example, at online retailers like Amazon, for continually identification and classification of uploaded product pictures.

2 Introduction

"The transfer of knowledge within the lifetime of an individual has been found to be one of the dominating factors of natural learning and intelligence. If computers ever are to exhibit rapid learning capabilities similar to that of humans, they will most likely have to follow the same principles." Sebastian Thrun (1996 [34] page 193).

Humans are remarkably good in absorbing new knowledge about unknown things very fast from only a few single examples, continually throughout their lifetime. Thus, life-long learning presents a crucial capability in our daily life. Deep neural networks have shown excelled results on a wide range of problems, from recognition, reconstruction and localization tasks in computer vision, language processing, etc. [27, 35, 36, 37]. Typically, these algorithms apply batch-wise training to large datasets e.g. ImageNet [19] and need many iterations of the whole dataset to obtain satisfactory performance. In contrast to humans, neural networks rely on a training with the whole, static dataset, containing everything they have to know. This dataset is then repetitively presented to the network to obtain a good generalization and accuracy. Usually, this presenting has to be done several thousand times. That means, everything the network should learn has to be available before the training starts. Continually learning new tasks through the lifetime of a neural network is therefore not possible. If new things arrive, which the network is not able to compensate by generalization, the network has to be trained completely again, as generalization is limited to known categories. In this thesis a novel algorithm, to solve this problem, is presented and different solution strategies are evaluated. With our approach, the robot is enabled to learn new things on the fly, without the need of retraining the whole network. In contrast to most of other solutions in literature, this approach does not use additional memory to store previous training data.

Our aim is to solve the problem of continually learning new object categories, in a setting with a limited memory capacity, where it is not possible to save data from previously learned categories, like images. This so called online learning strategy is more difficult, but also more practically relevant for service robots, as those are limited on disc space and sometimes, it is not possible to save images of each category, the robot knows so far. Thus solving this problem without the need of previous training data would be the faster, saver and less storage intensive solution. Therefore, the focus of this thesis is on online learning. Additionally, a possible integration of the so called rehearsal strategy is shown at the end of the work, where it is allowed to reuse previous training data.

But, there occur some fundamental problems, when adapting a network to learn in a

continual manner. Those problems are described in the next section.

2.1 Problem description

Taken an theoretically infinite stream of data, a continual learning algorithm has to learn from a sequence of data or data-batches, without access to the whole dataset. Commonly used neural networks for object classification, like ResNet50 or Inception-v3, are not designed to be used in a continual learning process, as their main focus is on learning only once. This is due to the fact that these more researched, non-continual learning algorithms have access to all the data in the beginning. If those are used to train in a continual manner, the network suffers from forgetting [38]. In the following, this problem is described in more detail and the aim of this thesis is defined.

2.1.1 Catastrophic forgetting

Forgetting is one of the biggest problems continual learning algorithms face nowadays. It describes the problem, when a network forgets previously gathered knowledge by learning new tasks. In literature, this is known as catastrophic forgetting or catastrophic inference [38, 39]. This problem applies in particular for networks, which use a softmax and cross-entropy as classification loss. There, the network experiences a rapid overwriting of the model parameters, when learning in a continual manner [40]. These methods are state-of-the-art in the field of classification in computer vision, but are not suited for continual learning.

The reason of catastrophic forgetting, in the context of continual learning, is the lack of comparable features between the categories of the different sequences. A good example is the way a human would learn different categories. For example: Assuming, there is a human subject, who doesn't know the categories "dog" and "cat" (see figure 2.1). There are now two possible ways to teach this. First, by showing a lot of different dog pictures. Thereby, our subject might discover that a dog has four paws, two eyes and a snout. But after the person also observes some cat pictures, the emerging problem of this procedure is shown, as the subject will not be able to see a difference between dogs and cats, as they are having a lot of features in common. Another, more effective way, is by showing the person pictures of dogs and cats simultaneously. By finding the difference between both categories, our subject will be able to recognize dogs and cats after the training. The same problem occurs for neural networks. As the training examples of previous categories are not longer available, the network can not find the differences to the current shown categories. Therefore, it is not able to learn new categories, without forgetting the old ones.

Figure 2.1: Example for comparable features: A dog and cat are shown to a human, who does not know either. If the subject first observes the dog- and later the cat images it is much more difficult to learn how to distinguish between those two categories, then by showing images of dogs and cats simultaneously. The pictures are from [41] and [42].

An example of catastrophic forgetting is shown in table 7.2 of our result section. The challenge now is to find a way to prevent this catastrophic forgetting.

Data distribution shift Another reason for forgetting can be found in literature, which is named data distribution shift [43, 44]. Gepperth et al. and Lesort et al. found, that taking temporal structure of data samples into account, one can observe changes in the data distribution that occur over time. This also applies to continual learning, as there a form of temporal structure is used. This refers to the fact, that the different tasks are split into several sequences, which are learned chronologically by the network (see 2.2). Those changes in data statistics are referred to as *concept drift* or *concept shift* [43]. Especially online learning suffers from this problem, as it does not keep data of previous examples, to compare the new data with. If there is no external information about a data distribution shift, the continual learning algorithm has to detect it. Otherwise an undetected shift would lead to forgetting [44].

According to Gepperth et al. [43], concept drifts can be distinguished in two different types:

- **Virtual concept drift** or **covariate shift** occur by changes in the input distribution, which can easily appear, e.g., due to adding of dissimilar object categories to a classification problem causing an imbalance distribution.
- **Real concept drift** is caused by novelty on data or new categories, e.g., when the model has to be re-adapted on visually similar, but new categories, which is at the core of continual learning.

These concept drifts can happen gradually or abruptly and may also appear when changing the task of the network.

Knowledge transfer In order to deal with catastrophic forgetting, one has to find a strategy to handle the knowledge within the network. In continual learning, there are a lot of different ways to store information about already learned tasks:

- Raw data from image examples
- Any kind of representation of the training examples, e. g. latent space
- Model weights
- Regularization matrices, e.g. importance matrix
- Reconstruction values, etc.

An efficient strategy should be able to only save important data and transfer this knowledge through all sequences, to mitigate forgetting. A combination of these strategies is common as described in chapter 3.2. Further information of the different techniques are described in our related work section 3.

2.1.2 Aim of this thesis

The aim of this thesis is to develop an approach, which is able to learn unkown object categories on continually appearing images, in a robotic environment, considering the problems, described before. The solution should be applied in the field of computer vision, by using a convolutional neural network. Furthermore, approaches are preferred, where additional memory for old training examples is not necessary (online learning).

2.2 Incremental Learning procedure

In order to simulate a continuous learning process, the training and validation procedure has to be adapted accordingly. Usually, a neural network learns all training examples at once. But in the case of incremental learning, the network starts with a basic knowledge and then continually learns new tasks over time. As shown in figure 2.2 the incremental learning procedure is conducted differently to the standard way, a neural network is trained. Here, the training images arrive in batches, distributed over sequences and each category is only available in one sequence. For example, the first sequence contains categories $[0-9]$, the second sequence categories $[10-14]$, etc. Thus, category "apple" is only available in sequence zero and nowhere else. This comes close to a practical situation,

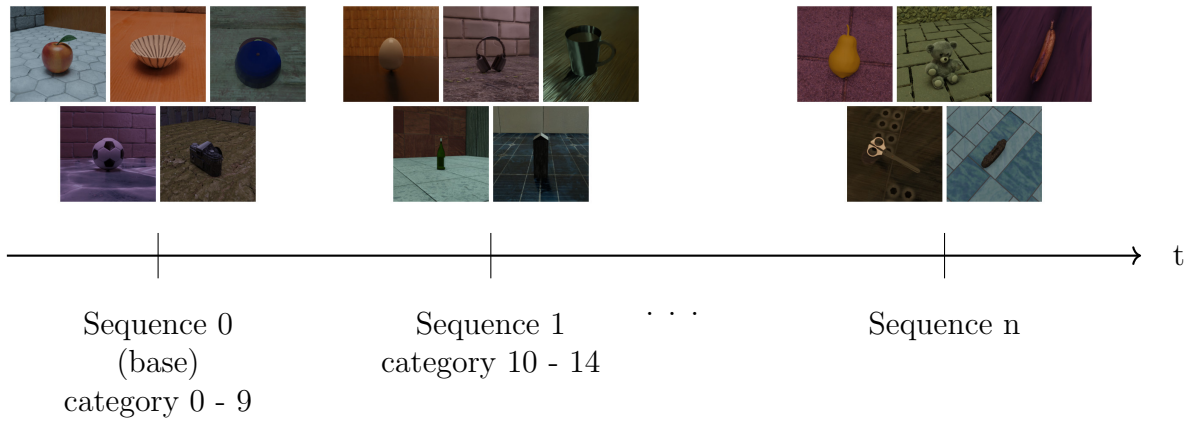


Figure 2.2: The incremental learning procedure: The different object categories are divided into sequences, which are shown to the network over time. Each category is only available in one of those sequences and the network therefore only has access to the images of the certain categories during this sequence. In the next sequences, the task is to learn new categories, while not forgetting the old ones. This graphic uses pictures from the HOWS-CL-25 dataset.

when a robot observes new, unseen objects, as shown in figure 1.1. Continually learning new objects it observes over time, is much more challenging, than training with the images of all categories at once, since the network must always keep the knowledge from previous sequences.

2.3 Thesis structure

After the introduction into the topic of this thesis and a problem description, an evaluation of solutions from literature and their difference to our approach are discussed in the related work chapter. After that, important methods of our work are described more detail in the chapter "General". Then, our approach is presented in the following chapter. The used datasets, as well as details to the learning procedure and implementation are shown in an experimental setup, followed by the achieved results. In our ablation studies, the impact of different hyperparameter choices are presented. In the end of this thesis, future steps are discussed, followed by a conclusion. Additional information on the used hyperparameters are attached in the appendix.

3 Related work

In this section, the definition of continual learning and its solutions from literature are discussed. This includes their advantages and disadvantages and a highlighting of their differences to our approach. After an overview, where the different methods are subdivide into three strategies, the approaches are discussed in more detail.

3.1 Definition and differentiation

In literature, there are several definitions of the process, of continually learning new things, like categories, instances or tasks. Gepperth and Hammer [43] or Rebuffi et al. [18] call it *Incremental Learning*, Chen and Liu [45] or Thrun and Mitchell [46] *Lifelong Learning* and Carlson et al. [47] and Mitchell et al. [48] call it *Never Ending Learning*. Like Lesort et al. [44], this thesis refers to all continuous, incremental and lifelong learning synonyms as continual learning (CL). In our definition, incremental learning is a sub-area of continual learning, where new tasks are learned chronological over time. And online learning, on the other hand, is defined as a sub-area of incremental learning, where the network only has access to the current task and explicitly not to previous tasks (see figure 3.1). The difference between those three areas is shown in table 3.1. Online learning is the most challenging part of incremental learning and the major focus of this thesis.

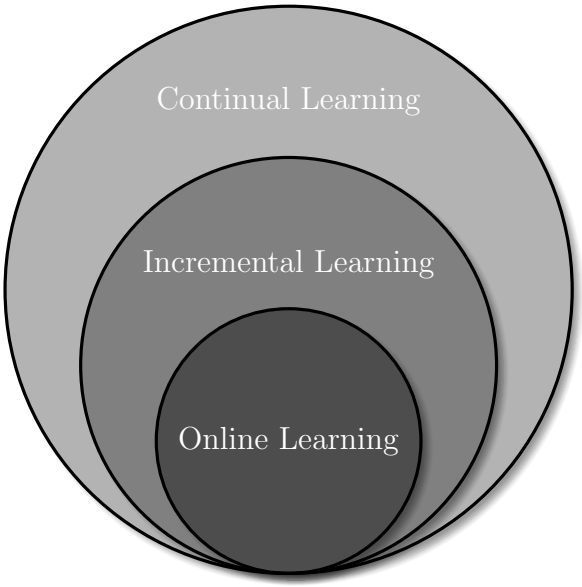


Figure 3.1: Definition of online-, incremental- and continual learning: In general, the procedure of continually learning new tasks is known in literature as continual learning. Incremental learning is defined as a part of continual learning, which deals with sequentially appearing data, thus new tasks are learned chronological over time. Online learning is a special kind of incremental learning, where the network only has access to the current task.

Table 3.1: Online and incremental learning

Research area	Description
Continual Learning	All different techniques to continually learn new tasks.
Incremental Learning	Sequential continual learning, where data arrives in chronological batches.
Online Learning	Special case of incremental learning, where data of previous sequences are not available. Also called rehearsal-free incremental learning.

The research field of continual learning is interlocked with many other areas and works, which are often used in combination, in order to improve or to build a base for another technique.

A lot of continual learning algorithms are based on **transfer learning** [49, 50, 51, 52], where knowledge is gathered from previously learned tasks. A common way is e.g. to pretrain the network on ImageNet [19] and then fine tune it on example images of the categories, the network actually should learn. Compared to continual learning, the network does not have to be able to solve previously learned tasks. Therefore, this approach cannot be adapted on a one-to-one basis. In computer vision, this is also referred as *domain adaption* [53].

A similar approach is **meta learning** [54, 55], a sub-field of machine learning, where metadata about previously gathered knowledge is used as a hyper parameter. The goal is, to use this metadata to understand how different algorithms perform on a given tasks and therefore improve the performance of an existing learning algorithm, or to learn the algorithm itself. It is also known as "learning to learn". As well as transfer learning, also a modified meta learning approach could be used for continual learning.

Especially in the field of robotics **few-shot learning** [56] becomes interesting. It describes the ability to learn tasks from only a few examples, as taking training examples is very time-consuming. If the robot only needs to take a few images, the whole process would greatly benefit. This method has nothing to do with continual learning directly, but could be used as a further improvement step for an already created approach.

As neural networks tend to be overconfident, the research field of **uncertainty estimation** becomes important for continual learning [57]. It contains methods to measure the certainty of the network about its own prediction, which becomes crucial, especially for service robots, as wrong classifications would hurt the trust of the user.

The same is true for **active learning** [58], a semi-supervised machine learning method

where, for example, the system interactively asks the user for the label of a new, unknown object category, which it detects itself. Here, it is important to know, what the robot does not know, to trigger a further handling on those examples.

3.2 Continual Learning strategies

The field of continual learning (CL) can be divided into three strategies, as shown in figure 3.2 [3]. After this short overview, each method will be described more deeply.

- **Architectural strategies:** These are strategies, which try to moderate forgetting by changing the architecture of the neural network. For instance, by using special architectures, layer changing or different freezing strategies. Here, these changes are often done dynamically, for example at every sequence. This also includes techniques, which try to imitate the hippocampus-cortex duality, for example by dual-memory-models. A dynamically growing network makes it possible to learn in a continual manner but also carries the risk of memory overflow as also the storage is dynamically growing. Rusu et al. introduced PNN [25] (Progressive Neural Network), where they combined parameter freezing and network expansion for reinforcement learning on Atari Games, where they showed effectiveness on short series of simple tasks. Lomonaco et al. presented CWR [9] (Copy weights with re-init), which is an easier version of PNN, but with a fixed number of shared parameters, resulting in better performance on longer task sequences and less flexibility.

- **Regularization strategies:** Regularization strategies mitigate forgetting by techniques, like changing the loss function, selective consolidation of important weights, dropout or early stopping. This field is influenced by neural studies on how the brain is solving this problem. For instance, the approach SI [31] (Synaptic Intelligence) by Zenke et al., where they tackle the problem of forgetting by using an importance matrix on the weights of the network. Or, EWC [11] (Elastic Weights Consolidation), proposed by Kirkpatrick et al., where they use a loss to prevent a change of important weights. Another approach in this field is LWF (Learning Without Forgetting) [22] from Li et al., where they reduce forgetting by using knowledge distillation on old tasks, which is proposed by Hinton et al. [59].

- **Rehearsal strategies:** Rehearsal strategies prevent forgetting by saving past information and replay them to the model, in order to strengthen memories of categories, it has already learned. A simple approach is the storing of previous seen training images of each categories and interleaving them with pictures of the current sequence,

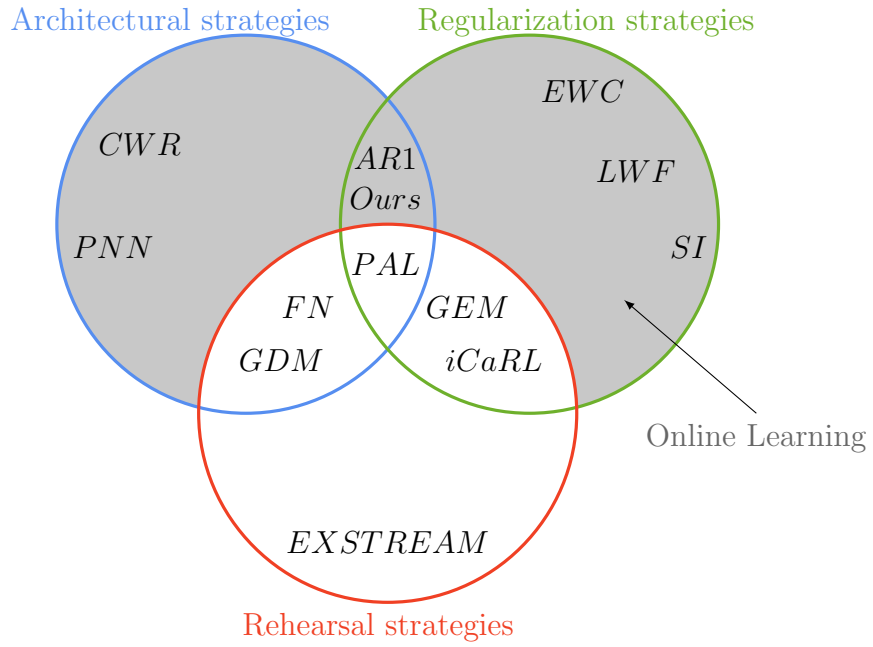


Figure 3.2: Venn diagramm of incremental learning strategies, where online learning is represented in gray. Our approach, as well as AR1, are using architectural and regularization strategies.

during training. The EXSTREAM [60] approach from Hayes et al., for example, stores all past data for a stream clustering.

As shown in (3.2), there are also approaches, which use more than one strategy. iCaRL [18] (Incremental Classifier and Representation Learning), by Rebuffi et al., stores a subset of old task data and additionally uses distillation. GEM [14], by Lopez-Paz et al., uses a fixed memory for old patterns in addition to a loss regularization. FN [61] (FearNet), by Kemker et al. and GDM [62] (Growing Dual-Memory), by Parisi et al., on the other hand, combine architectural and rehearsal strategies in their double-memory system for short- and long-term memory. AR1 [3], proposed by Maltoni et al., as well as our approach are a combination between regularization and architectural strategies. AR1 uses an extended CWR from [9], called CWR+ in combination with the regularization approach SI from [31]. Last, but not least, there is also an approach, which uses all tree strategies, proposed by Denninger and Triebel, called PAL [24] (Persistent Anytime Learning of Objects from Unseen Classes), which is based on random forests. After this quick overview, next the different approaches are analysed more precisely.

3.3 Progressive Neural Networks (PNN)

Progressive Neural Networks (PNN), by Rusu et al. [25], is an early published method in the field of continual learning, where they solve the problem by an architectural strategy.

In their reinforcement learning approach, they propose an efficient way to learn short series of simple tasks, shown on a classic Atari 2600 task set.

They propose to dynamically expand the model's architecture, by allocating novel sub-networks with a fixed capacity. When a new sequence arrives, one of those sub-networks is initialized. Next, it is trained on the data of the novel sequence, where it also learns the lateral connections to the existing network. After training, this network-branch gets frozen to not change anymore.

In comparison to our work, they also let their network dynamically grow, according to the number of sequences. However, in our experiments, a complete freezing of the sub-networks resulted in a catastrophic forgetting scenario, as those network parts still have to learn how to distinguish between the different categories across the sequences. Thus, in our work, all sub-network parts are trained together, to enable the network to learn those differences.

3.4 Copy Weight with Reinit (CWR)

Copy Weight with Reinit is proposed by Lomonaco et al. [9], as a baseline for continual learning, using architectural strategies. This approach is similar to PNN and our architectural step. They also freeze the feature extraction network (using Mid-CaffeNet and VGG) and only train the last fully connected layer of the network. The only exception is the first sequence, where the whole network is trained. For each sequence of data, they randomly initialize the last layer of the network and train it on the current images. After training, the weights of the last layer are copied into a separate head for validation and test. When a new task arrives, they reinitialize the last layer again, learn it on the image examples and then concatenate it with the saved weights from previous sequences. As the network doesn't take previously learned weights into account, and only trains the newly initialized last layer, the comparison between the previous and current categories is missing in the training process. The lack of features, which are used to distinguish between different categories, leads to catastrophic forgetting (described in chapter 2.1.1). Furthermore, the concatenation of weights, which are trained in different sequences, is problematic. As they use a classification loss with softmax, they concatenate different probability distributions. The integral over the resulting distribution is not longer a probability distribution, since the integral over all categories is not one. A probability distribution over all categories is therefore only possible, if they are trained together.

This approach is similar to PNN, which also freezes old neurons, but also takes the old weights into account, while training the new ones. Furthermore, their proposal, to train the whole network in the first sequence, performs worse in our experiments. This

algorithm was further improved in the AR1 approach, described in section 3.10.

3.5 Learning without forgetting (LWF)

In literature, learning without forgetting (LWF) is often used for comparison [3, 18, 31, 14]. They propose a regularization strategy to stabilize the model accuracy on old tasks by using knowledge distillation, proposed by Hinton et al. [59]. Similar to our approach, the logits of the previous sequence network and the current network are encouraged to be similar, when applied to data from the new sequence. Therefore, they also compute the prediction of the network for each new image of the current sequence, using network weights from the previous sequence. These network predictions are saved in order to prevent forgetting, when training the new categories. The difference to our approach is, that they propose a form of knowledge distillation, where they record a set of label probabilities for each training image on the previous network weights. Whereas in our work the exact network outputs without softmax are recorded. On top of that, a regression loss is used instead of their used classification loss, as the probability distribution changes, when new categories are added to the last layer and therefore the recorded label probabilities suffer under a distribution shift. In LWF, they try to compensate that problem by first freezing the old neurons and only train the new neurons in a "warm-up step".

$$(1 - \lambda) \cdot E_{old}(\hat{y}, p) + \lambda * E_{new}(y, out) \tag{3.1}$$

In equation 3.1 the loss function of LWF is shown. This two-part-loss combines the loss on old neurons E_{old}, using a modified cross entropy loss on the recorded $\hat{y}$ and current probabilities p, and a multinomial logistic loss on the new weights E_{new}, which the one-hot ground truth labels y and the softmax network output out. The value λ is a loss balance weight. Experiments showed, that it is also possible to use a cross-entropy loss for both loss parts, resulting in similar results [3].

In our approach another loss function is used, as our experiments showed the problem of forgetting due to a discrepancy in logits distribution, triggered by the usage of a cross-entropy loss with softmax. Furthermore, they use Stochastic Gradient Descent instead of the Adam optimizer. Another difference is the used feature extraction network, where they use a freezed AlexNet and VGG-16. A direct comparison to our approach is not possible, as they use datasets, which do not meet the requirements for this thesis approach. The reason for this is that some of their used datasets are based on ImageNet [19], which is the dataset our feature extraction network is pre-trained on. Or, their datasets contain images of buildings and places, which are not considered in our approach.

3.6 Elastic Weight Consolidation (EWC)

Elastic Weight Consolidation [11] allows continual learning in a reinforcement learning context, by using a regularization method, which is based on neural science. There they mitigate forgetting by selectively slowing down the learning of some parts of the network weights, which are important for the previous learned tasks. They propose to use a fisher information matrix to find those important weights and use a quadratic penalty on the difference between the weights of the new and the old sequences, in order to prevent them from changing. When a model trains on a task and reaches a minimum loss value, it is possible to estimate the sensitivity of each of the models weights θ_k by observing the curvature of the loss surface along the direction, determined by a change of the specific weight θ_k. A high curvature can be interpreted as a slight change of a specific weight θ_k, leading to a sharp increase of the loss. The diagonal of the fisher information matrix F is equivalent to the second derivative (curvature) of the loss near a minimum. Furthermore, it can be computed from first-order derivatives, which makes it easy to calculate even for large models and additionally, it is guaranteed to be positive semi-definite.

Therefore, the k^{th} diagonal element of the fisher matrix indicate the importance of the specific weight θ_k. Those important weights are prevented from changing, while the model is trained on a new task. After each sequence training, the fisher information matrix must be computed and the set of optimal weights $\hat{\theta}_k$ has to be stored for the next sequences. The loss function of EWC takes the fisher importance matrix as a regularization term:

$$L = E(y, out) + \frac{\lambda}{2} \cdot \sum_k F(\theta_k - \hat{\theta}_k)^2 \tag{3.2}$$

Equation 3.2 shows the loss function used in EWC. The cross-entropy loss E with label y and network output out is supplemented by the sum of the squared error between the current network weight θ_k and the optimal network weight from the previous sequence $\hat{\theta}_k$. Value λ indicates the importance of the old tasks in comparison to the new one. Therefore, the model is able to take the change of important weights into account, while learning new categories.

They test their approach on Atari 2600 tasks and MNIST, where they show that it is possible to mitigate forgetting by using a weight importance matrix. Compared to our approach, instead of changing all weights of the last layer, they propose to only avoid a changing of important weights. But, as the computation of the diagonal of the fisher matrix requires summing over all possible output labels, the complexity is linear to the number of outputs and limits the network to low-dimensional output spaces.

An usage of this approach was out of scope for this thesis. The next method is simi-

lar, but presents a more advanced technique, by preventing the network from changing important weights, called synaptic intelligence.

3.7 Synaptic Intelligence (SI)

Synaptic Intelligence [31], by Zenke et al. is a variant of EWC, where they, instead of using a fisher important matrix, propose to calculate the weight importance on the fly, using Stochastic Gradient Decent, as it is less computational intensive. This approach is also rooted in neural science, as they argue that a biological synapse accumulates task relevant information over time and stores new memories without forgetting old ones. A behavior they try to simulate.

$$\Delta E_k = \Delta\theta_k \cdot \frac{\partial E}{\partial \theta_k} \tag{3.3}$$

In equation 3.3 the loss change ΔE_k by a single weight update step is given, using SGD, where $\Delta\theta_k$ indicates the weight update amount $\hat{\theta}_k - \theta_k$, of the optimal weight $\hat{\theta}_k$ and the current weight distribution θ_k, and $\frac{\partial E}{\partial \theta_k}$ indicates the gradient. The total loss change, triggered by changing a specific weight θ_k can be calculated by the running sum of the product of the gradient with the parameter update (sum over the weight trajectory). The importance matrix of a specific parameter I_k^s at the current sequence s can therefore be calculated as shown in equation 3.4, where Δ_k indicates the total weight change of parameter θ_k from initialization $\Delta_k^s \equiv \theta_k^s - \theta_k^{s=0}$. ξ is a small value to prevent dividing by zero.

$$I_k^s = \sum_{\hat{s}<s} \frac{\Delta E_k^{\hat{s}}}{(\Delta_k^{\hat{s}})^2 + \xi} \tag{3.4}$$

The difference to a fisher matrix is, that the whole data, which is needed to calculate the importance matrix, is available during SGD. So, it is less computational intensive.

3.8 Incremental Classifier and Representation Learning (iCaRL)

Incremental Classifier and Representation Learning, by Rebuffi et al. [18] proposes a class-incremental algorithm, which uses a regularization strategy in form of a nearest-mean-of-exemplar classification and a rehearsal strategy by saving feature representations over time. The training of a classifier and representation is decoupled. In order, to create an exemplar set for each learned categories, iCaRL uses representation learning. Therefore, a set of feature representations is updated for each new category. First, the training images of the current sequence and all so far saved feature representations are

combined for training. Thereby, each network ouput for all training examples are saved (similar to LWF). Like LWF, iCaRL uses a combination of knowledge distillation and classification loss to train the network. After that, a reduction of the exemplar set is performed to improve storage usage. Simultaneously, the new categories are predicted by learning a classifier. This classifier is customized to predict a label y by computing a prototype vector for each category observed so far.

$$y^* = \underset{y=1,\ldots,d}{\operatorname{argmin}} ||out - \eta|| \tag{3.5}$$

Equation 3.5 shows the nearest-mean-of-exemplar classifier, where η is the average feature vector over all exemplars d of a certain category and *out* indicates the feature vector of the current example image. The label y^* with the most similar prototype will be assigned. This classifier is designed to be robust against changes of the feature representation. Furthermore, they introduced a new continual learning dataset, called incremental CIFAR100. A variation of the CIFAR object recognition dataset with 100 categories. This dataset splits the categories equally into different task sets (sequences).

Compared to our approach, iCaRL uses a rehearsal strategy, and is therefore not classified as online learning, as they rely on a subset of the original training data to keep the performance on old categories. Saving previous training examples leads to an increase of the networks accuracy in cost of all the challenges coming along with rehearsal strategies, described in section 3.1. Although, iCaRL has access to previous training data, it is outperformed by our approach on CORe50 dataset, shown in section 7.1.3.

3.9 Gradient Episodic Memory (GEM)

Gradient Episodic Memory (GEM) from Lopez-Patz and Ranzato [14], uses, as well as iCaRL, a combination of rehearsal and regularization strategies. The differences to iCaRL are:

- While iCaRL is designed to fill the total memory after every batch, GEM uses a fixed storage amount for each batch. The memory limit is only reached at the end of the last batch.
- Instead of keeping the predictions of past sequences invariant, by using distillation, GEM uses the losses as inequality constraint. This avoids their increase, but allows their decrease, in order to make positive backward transfer possible.

Their main feature is an episodic memory, which stores a subset of trained examples for each category, similar to iCaRL. Besides accuracy, they propose to take backward- and

forward transfer into account, while training the network. In general, a backward transfer (BWT) indicates the influence, that learning a new task has on the performance of a previous tasks. A positive backward transfer is defined by an increase of the performance of previously learned tasks, while learning a new tasks. Therefore, negative backward transfer is known as catastrophic forgetting. Forward transfer (FWT) indicates the influence, that learning a new task has on the performance of a future task. GEM focus on minimizing negative backward transfer by using episodic memory and allowing positive backward transfer. On the right hand side of figure 3.3, GEM outperforms iCaRL and EWC in test accuracy over 20 sequences. On the left, it can be found that GEMs strategy, taking backward transfer into account, seems to work, as it always has the lowest negative, and on one test, even a positive backward transfer, while the other methods perform worse. They show in their paper, that GEM outperforms EWC on MNIST and CIFAR-100 and iCaRL on the CIFAR-100 dataset, as well in accuracy and computational costs (shown in figure 3.3).

Like Rebuffi et al. [18], our approach focuses on keeping the predictions of past sequences invariant and does not consider the possibility of a positive backward transfer, proposed in GEM. Additionally, GEM main focus is on using an episodic memory and is therefore not classified as online learning.

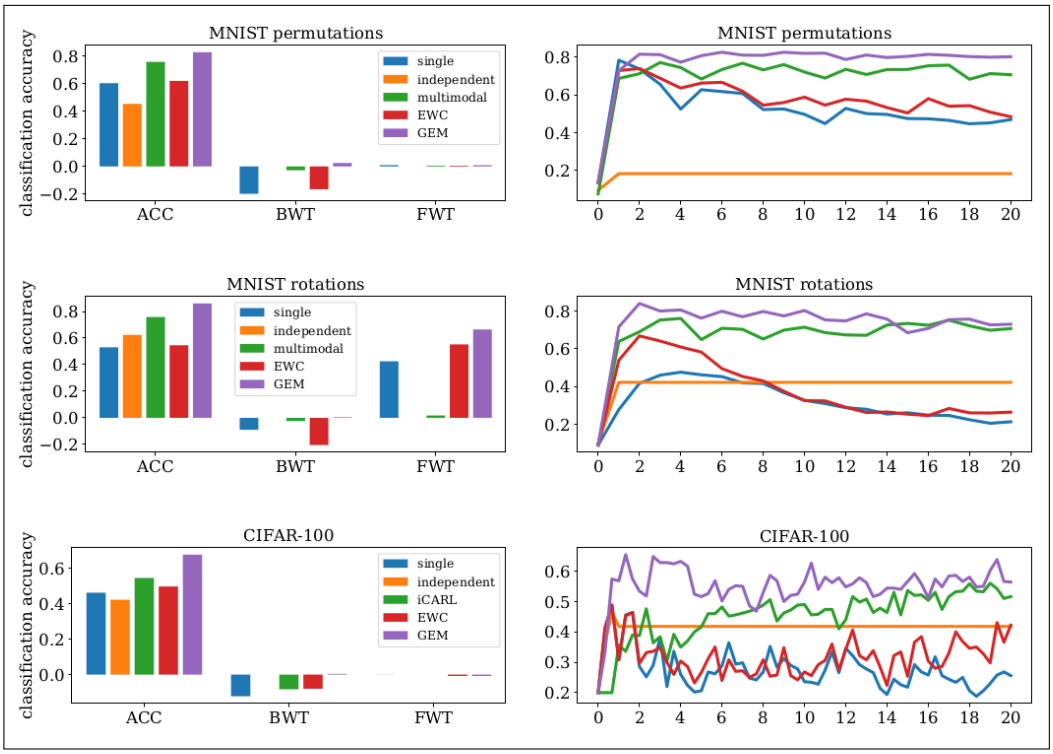


Figure 3.3: Comparison of EWC, iCaRL and GEM from [14]: The different approaches are compared on MNIST and CIFAR-100 datasets, whereas ACC indicates the accuracy, BWT the backward transfer and FWT the forward transfer.

3.10 Continuous Learning in Single-Incremental-Task Scenarios (AR1)

AR1, proposed by Maltoni and Lomonaco, [3] is a combination of an improved CWR (see section 3.4), called CWR+ and Synaptic Intelligence (see section 3.7) approach. Compared to the previous described CWR approach, CWR+ has two modifications:

- Mean-shift: They propose to divide the network weights by their own mean. This normalization step might also solve the problem of data distribution shifts, described in section 2.1.1. In our approach a division with the variance per category is proposed, to solve this problem. A comparison of both strategies is out of scope of this thesis.

- Zero-init: They propose to initialize the weights of the last fully-connected layer, by zero, instead of using a typical Gaussian or Xavier random initialization. In our experiments, it was also found that the network performance strongly depends on the initialization. By initializing all neurons with the same value, they are fixing this problem. They also highlight that, using zero as a start value, is not nullifying the back-propagation effects, as they prove that back-propagation still works in the last layer.

AR1 also uses architectural and regularization strategies, like our approach. Compared to them our approach is able to learn in a continuous manner, as their approach only supports a maximum of 50 different categories. In this way, their approach is optimized for CORe50 dataset, whereas our approach is more general usable. Furthermore, they propose to use an important matrix from the SI approach, which was beyond the scope of this work. This idea, of only protecting important weights from changing, instead of all parameters, seems intuitive, but as they use this importance matrix in combination with Stochastic Gradient Descent (SGD), it can be argued, that it is similar to an ADAM optimizer step. The idea of mean shifting in the modified CWR+ method, where they divide the weights of a layer by the average over all weights, could be useful to prevent output discrepancy in the different sequence layers (explained in section 5.2.3). This problem is also experienced in our approach, where a division of the difference of the output and label, by the variance per category, is proposed (descried in more detail in section 5.2.3).

The initialization of new layers with zero values was beyond the scope of this thesis. But, our test results also show, that a similar initialization value improves the stability of the network. Another difference is, that our approach uses several layers and heads and is therefore capable of more complex category separations, whereas AR1 uses one layer

and one head. Lastly, even though they found that ResNet50 was performing better, they used GoogLeNet as feature extraction network, as it is more light-weighted. In our experiments, also several feature extraction networks are tested, including the newest versions of GoogLeNet (Inception-v3 and Inception-ResNet). But, also ResNet50 is found to perform best on the CORe50 dataset (see 8).

3.11 Persistent Anytime Learning of Objects from Unseen Classes (PAL)

Up to now, only approaches are evaluated, which use convolutional neural networks and fully-connected layers for continual learning. But there are also papers, which propose different machine learning techniques. For example "Persistent Anytime Learning of Objects from Unseen Classes"(PAL) [24] by Denninger and Triebel. They also use CNNs for feature extraction, but instead of fully connected layers, they use a random forest classifier. When a new batch of data arrives, they first evaluate how their current random forest performs on the new observations. After that, they create a training subset by randomly sub-sampling from the new training set (bagging) and train a new random tree based on this subset. The new resulting tree is then evaluated on a validation set and if it performs better than the worst performing tree, it gets accepted and the worst performing tree gets replaced.

Like our approach, their work focuses on an object classification task, which is particularly suited for robotic applications. Furthermore, they do not assume that the number of categories have to be given beforehand, thus their model also dynamically changes over time, like our approach. The difference is, that they, instead of increasing the number of neurons in the fully-connected layer, start with a fixed amount of binary decision trees, which they dynamically replace in their random forest. Furthermore, they propose to limit the number of trees in the forest to not increase the prediction time, whereas our approach has no limitation so far, which might become a problem for a high amount of categories. But, for only a few thousands of different categories, this problem can be neglected, as these are more than enough for a service robot. Since our approach uses a classification layer, each neuron in the last layer can be assumed to be responsible for one category, whereas it can not be assumed that each tree only is responsible for one category, as they use more trees, than categories in their forest.

The key difference is, that PAL, in addition to the architectural and regularization strategy, also uses a rehearsal strategy, where they save a subset of the training data for all learned categories so far. Therefore, a comparison to our approach is not possible. As shown in figure 3.4, their method (left) is able to efficiently learn new categories, without

the problem of forgetting, as they use containers to store feature vectors from previous sequences. They also show the difficulty of an online learning scenario (right), where they found that the removal of trees leads to catastrophic forgetting and furthermore the performance on learning new categories is decreasing over time.

Even if the aim of their and our work is similar, and both focus on incremental learning in robotics, the strategies and focuses are different. They showed a robust incremental learning approach, which almost performs like offline training, whereas the strength of our work is in online learning, where a dedicated memory for storing previous training examples is not needed at all, which makes it easier to use for life long learning scenarios. Their idea of using static containers for efficiently saving features of previous categories, is used by our additionally proposed rehearsal strategy, described in section 9.1.

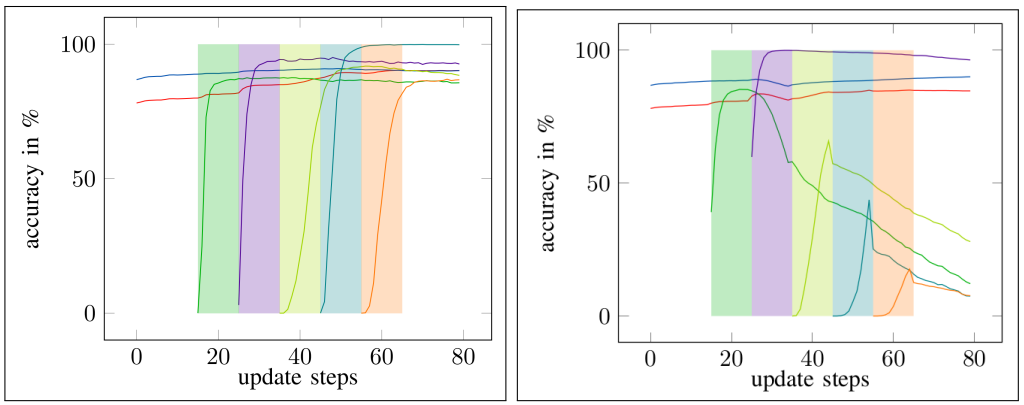


Figure 3.4: Results of the persistent anytime learning approach from [24]: On the left hand side, the result of their incremental learning approach is shown and on the right hand side the result of an online learning scenario. Each color indicates a set of new instances. Therefore, five different sets of instance categories are learned over time. It can be observed that their approach is able to not forget anything at all. Furthermore, they show how hard it is to solve an online learning scenario in comparison, where the first set of instances (green) is learned. However, after that, new instances are not learned and on top of that are quickly forgotten.

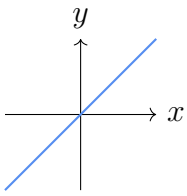
4 General

In order to support a better understanding of our approach, first some general topics are discussed. This includes activation functions, different feature extraction networks in form of convolutional neural networks, and the baseline approach of this thesis.

4.1 Activation functions

Activation functions can be divided into linear and non-linear functions, which define the behavior of the used neurons. Non-linear activation functions, like RELU [26] enable a neural networks to become deep and be able to solve non-linear tasks, as several fully-connected layers with a linear activation function would work like only one layer. In our approach, several activation functions are used for test purposes, which are described more detailed next.

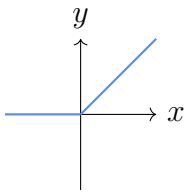
- Identity function (linear): The output of this function is equivalent to its input (see figure 4.1).



$$y = x$$

Figure 4.1: Identity activation function

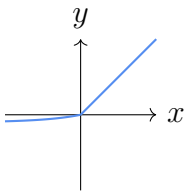
- RELU[26]: Rectified Linear Unit is a popular, non-linear, activation function, which leads to comparable good results in many works [63, 2, 27, 35] (see figure 4.2).



$$y = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{else} \end{cases}$$

Figure 4.2: RELU activation function

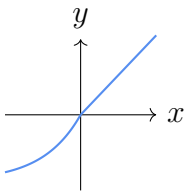
- ELU[12]: Exponential Linear Unit is similar to RELU, except of the handling with negative inputs. There, in contrast, it also gives a negative output. Furthermore, ELU becomes slowly smooth, whereas RELU smooths sharply (see figure 4.3).



$$y = \begin{cases} 0.2(e^x - 1) & \text{if } x < 0 \\ x & \text{else} \end{cases}$$

Figure 4.3: ELU activation function

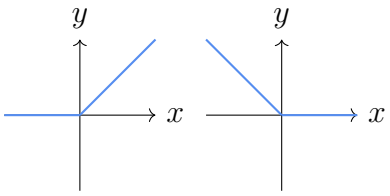
- SELU[29]: Scaled Exponential Linear Unit is a modified RELU function, where it additionally uses a self-normalization and fixes the problem of vanishing gradients (see figure 4.4).



$$y = 1.05 \begin{cases} 1.67e^x - 1.67 & \text{if } x < 0 \\ x & \text{else} \end{cases}$$

Figure 4.4: SELU activation function

- CRELU[10]: Using a Concatenated Rectified Linear Unit, each value is calculated by two RELU functions, preserves both positive and negative phase information, which doubles the depth of activations and leads to better recognition performance in some tests. It is computed by concatenating the layer output *out* as [RELU(out), RELU($-out$)] (see figure 4.5).



$$y = [RELU(x), RELU(-x)]$$

Figure 4.5: CRELU activation function

- SIREN[32]: SIREN is a new approach, where a sinus function is used for activation. This also comes with a different initialization strategy, where the weights are initialized in an uniform distribution between $-\sqrt{6/\text{ nr of neurons}}$ and $\sqrt{6/\text{ nr of neurons}}$. There are two new variables introduced for initialization, called "first-omega" and "hidden-omega" in their open-source code. Those variables are responsible for the number of periods, the sine function spans over $[-1, 1]$. Variable "first-omega" is used to control this behavior in the first fully-connected layer. The other variable on the other hand, is used to control the sine-like initialization in the other fully-connected layers. They propose to choose 30 as value for both variables. SIREN

is optimized for reconstruction tasks, where it reached state-of-the-art results in several tasks, like image-, shape- and audio reconstruction (see figure 4.6).

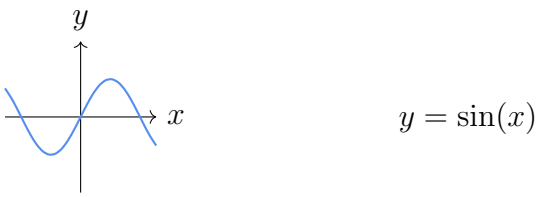


Figure 4.6: SIREN activation function

4.2 Convolutional Neural Networks

In this thesis, the focus is on convolutional neural networks for feature extraction, as they reached state of the art performance in the field of image classification and recognition [27, 64, 2, 33]. Convolutional neural networks were first introduced in the 1980s by Yann LeCun [65] based on the work of Kunihiko Fukushima named neocognitron [66], a basic image recognition neural network. The first CNN, called LeNet, was able to recognize handwritten digits. At that time, CNNs had the problem to not scale, as they needed a lot of data and computing resources to work efficiently. Also, these networks were only usable to images with low resolution, due to the limited computational power. Futhermore, there was not enough labeled data available. Due to the technical progress in 2012 these problems had been fixed and AlexNet [2] showed the enormous potential of neural networks.

According to the universal approximation theorem [67], a feedforward network with a single hidden layer is sufficient to represent any function if it has infinite capacity. The common trend in research is instead to go deeper. After AlexNet, the next state of the art network for image classification was VGG-16 [33] and GoogleNet [17] (ILSVRC-2014) followed by different ResNet architectures [27]. In addition to the increasing of depth, the success of CNNs can be attributed to several important discoveries like convolutions, pooling, dropout, RELU, etc. Typical architectures in the field of continual learning are VGG [22], ResNet or GoogLeNet [3].

A smart combination of the different techniques is crucial for the network to perform well, without the problems of vanishing gradiens or overfitting. In this chapter, some of the most successful architectures are evaluated, which are relevant to our approach.

4.2.1 ResNet

Residual Network (ResNet) [27] was published 2016 by He et al. and is still one of the most used CNNs in the field of image recognition, especially in combination with ImageNet [19], proposed by Deng et al. [63, 64, 68, 69]. There are several different versions of ResNet, like ResNet34, ResNet50, ResNet101, etc. Here, the number indicates the amount of layers of the network. In the following, the ResNet50 architecture is explained, which is also used as a feature extractor in this work.

Each ResNet architecture consists of several stages of ResNet blocks. As shown in figure 4.7, each block inside of one stage has the same output size. In ResNet50 there are four stages in total. Instead of using pooling, the first block in every stage uses a stride of 2×2 to halve the size of the output of the previous stage. Furthermore, the number of filters are doubled from the first to the last stage of ResNet blocks. In our approach, the the last fully connected layer is replaced with several dense layers.

ResNet block As shown in figure 4.8 the ResNet blocks inside of a stage consist of two convolutional layers with filter size 3×3, for ResNet18 and ResNet34 (left side) and three convolutional layers with different configurations for ResNet50 and ongoing. Those are called ResNet bottleneck blocks (right side). There, the second layer again has a filter size of 3×3, but the first and the last layer only have a filter size of 1×1. They decrease the number of channels before applying the 3×3 filter and increase it afterwards. In this way, the filter amount can be increased while keeping the number of parameters almost the same. The difference between ResNet and other architectures is the usage of shortcut connections between different ResNet-blocks. This shortcut enables the network to skip certain blocks, if these are not necessary for special features. This solves the problem of vanishing gradients and additionally the network learns the optimal path for certain features through the network.

He et al. [27] proposed those skip connections, or shortcuts, to also solve the problem, that each network reaches a limit of layers, after adding further ones, its performance gets worse. This is due to the fact, that the training gets more complicated, when the network's capacity is increased. By using skip connections, the network reaches lower error rates and it is therefore possible to add more layers, which is important as it increases the network capacity and therefore enables it to distinguish between more features. Li et al. [70] also showed how drastically skip connections improve the loss landscape (figure 4.9) and therefore leads to improved results. While, the optimizer in deeper networks without skip connection is more likely to get stuck in a local optima, good global minima can be found much easier using skip connections.

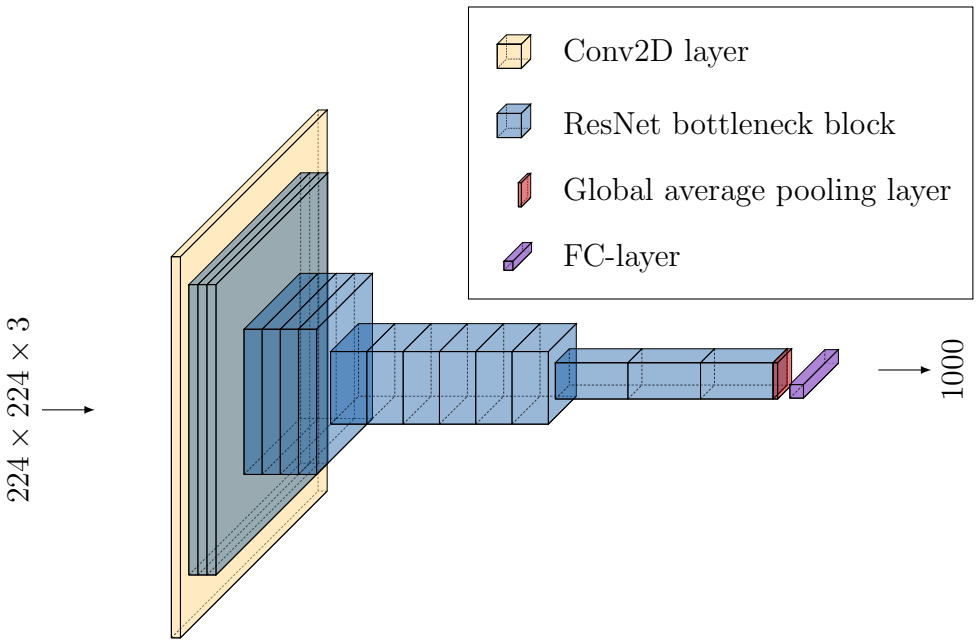


Figure 4.7: Visualization of the ResNet50 architecture: First features from $224 \times 224 \times 3$ images are extracted in a 2D convolution layer, reducing size with stride 2×2 and filter size of 64, followed by a max-pooling layer, the output size now is 56×56. After that, the data goes through 16 ResNet bottleneck blocks, organized in four stages, which each increase the filter amount and halves the input value. Thus, the number of input channels are doubled from 64, in the first, to 512 in the last stage. The number of output channels are doubled from 256, in the first, to 2048 in the last stage and the output value drops from 56×56, in the first, to 7×7 in the last stage. So, the input value develops from $56 \times 56 \times 64$ to $14 \times 14 \times 512$ and the output values from $56 \times 56 \times 256$ to $7 \times 7 \times 2048$ over the different stages. Each ResNet block consists of three convolutional layers, shown in figure 4.9. These ResNet blocks are followed by a global average pooling layer, calculating a feature output of 2048×1 and one fully connected layer with 1000 neurons for classification of the 1000 ImageNet classes.

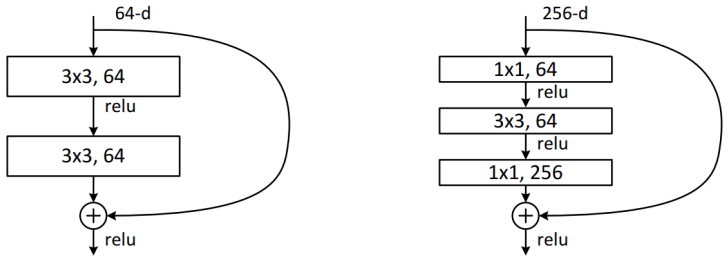
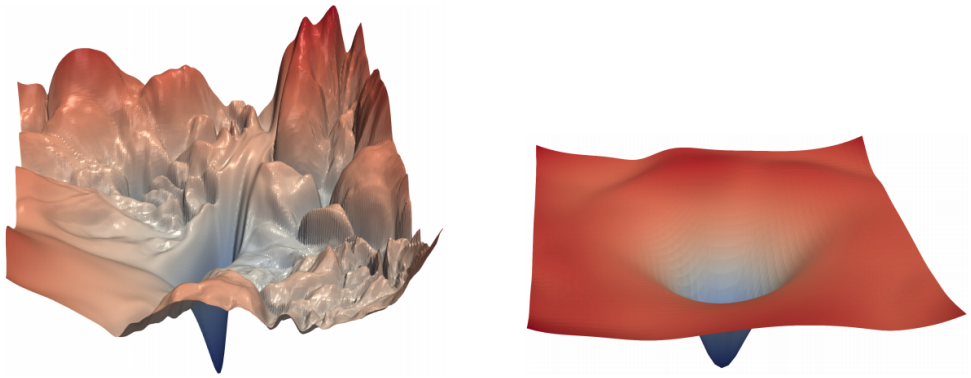


Figure 4.8: Comparison of the two different ResNet blocks and the shortcut connections between them. On the left hand side, the standard ResNet block with two convolutional layer is shown, as it is used in ResNet18 and ResNet34 and on the right hand side the ResNet bottleneck block with three convolutional layers and the bottleneck is shown, how it is used in ResNet50, ResNet101 etc. Each rectangle indicates one convolutional layer with the corresponding filter size and number of channels. This graphic is from the ResNet paper by He et al. [27].



(a) without skip connections (b) with skip connections

Figure 4.9: Comparison of skip connections in deep neural networks. Here the different loss landscapes of a ResNet architecture with and without skip connection is shown. On this graphic the significant difference becomes clear: Using skip connection, the surface becomes smoother and finding a good minima becomes therefore more likely. This graphic is from Li et al. [70].

Figure 4.10 gives an overview of the architecture of different ResNet versions. Each version consists of five stages. In the first stage a convolution and max pooling layer are applied to the input image. Afterwards, in the case of ResNet18 und ResNet34 four stages with normal ResNet blocks are used (see left hand side of figure 4.9) and in the case of ResNet50, ResNet101 and ResNet152 four stages with ResNet bottleneck blocks (see right hand side of figure 4.9) are applied. Finally, each feature goes through an average pooling layer and softmax. Inside of the convolution layers of the ResNet block, a Conv2D layer is applied, then the batch normalization and after that the activation function (RELU). In the second version of ResNet, they changed the sequence in order to remove the non-linearity. This is achieved by using identity mapping on the short-cut connections of ResNet, which leads in their experiments to a smoother information propagation. This also allows to directly propagate the forward and backward signals [28]. In our approach "ResNet" refers to the first version.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 4.10: Architecture of the different ResNet versions: The five stages (shown on the left) consist of different structures and amount of convolution layers, for each version (shown on the top). There, 50-layer, for example, stands for ResNet50. This figure is adapted from the ResNet paper [27].

4.2.2 Inception-v3

Inception-v3 [20], by Szegedy et al., is the third generation of GoogLeNet [17], with improvements like batch normalization and factorization, compared to the previous versions. The core technique behind all versions is the inception module, shown in figures 4.11 and 4.12. The key idea is to deploy multiple convolutions with multiple filters and pooling layers simultaneously, within the same module (inception layer). For instance, the architecture, shown in figure 4.11, employs convolutions with 1×1, 3×3 and 5×5 filters and a max pooling layer in parallel. Therefore, a individual best-process way for each feature can be learned from the network, as the path with the lowest error can be chosen. Additionally, they propose to reduce the number of channels in the network. Figure 4.12 demonstrates, how the usage of 1×1 filters achieves a dimensional reduction and thus speeds up the training process. The intention is, to let the network learn the best way to treat the different features and automatically select the most useful ones by additionally reducing the number of dimensions.

In the third version, they introduce factorizing convolutions, in order to reduce the number of parameters, without decreasing the networks efficiency. As they claim in their paper that convolutions with larger filters (e. g. 5×5), in terms of computation, tend to be disproportional expensive, they propose to replace them by two smaller filter. Replacing one 5×5 with two 3×3 filter for instance, reduces the number of parameters from $5 \times 5 = 25$ to $3 \times 3 \times 2 = 18$, so by 28%.

In figure 4.13 the architecture of Inception-v3 is shown. In order to first extract general features and reduce the input size, they propose to use five convolutions and two max pooling layers, before putting the input through a variety of inception modules. Furthermore, they propose an efficient grid size reduction, to reduce the size without the

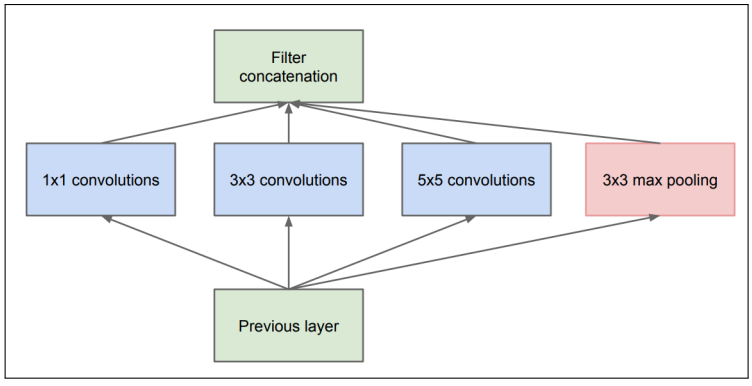


Figure 4.11: Inception module naïve version from the original paper [17]: Convolution layers with different filter size are deployed in parallel to achieve dynamical best feature treatment.

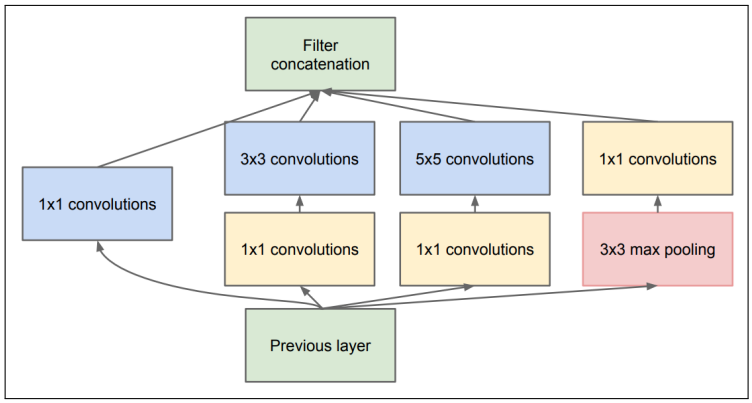


Figure 4.12: Inception module with dimensional reduction from the original paper [17]: In addition to the naïve approach the paper also propose to use 1×1 convolution layers to reduce the number of channels.

disadvantages of using max pooling, where it is either too greedy, when max pooling is followed by a convolution layer, or too expensive, when a convolution layer is followed by max pooling.

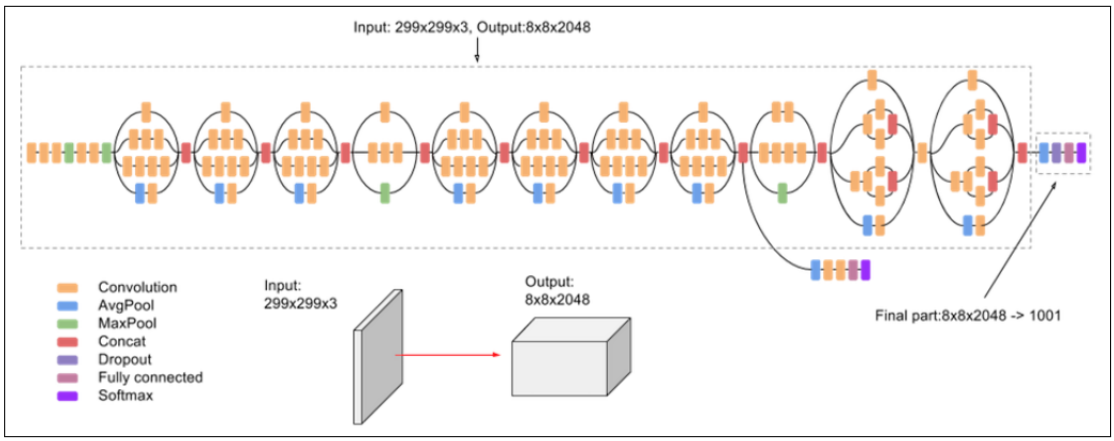


Figure 4.13: Inception-v3 architecture from [71]: The input image of size $299 \times 299 \times 3$ first goes through five convolutions and two max pooling layers. Followed by nine inception modules of three different types and two grid size reductions. The output of the feature extraction is $8 \times 8 \times 2048$. In the final part, these values go through a global average pooling, dropout, fully-connected and softmax layer. The final output is 1001 for all 1000 categories of ImageNet and one category for the background. Additionally, to the output of the final part, they propose an auxiliary classifier for regularization, shown at the bottom.

4.2.3 InceptionResNet

The fourth generation of GoogLeNet is a combination of the inception architecture and residual connections of ResNet. The resulting network is called InceptionResNet [21]. Szegedy et al. presented different versions of InceptionResNet. In this work, only InceptionResNetV2 is considered, as this version is shown to significantly improve recognition performance. They demonstrate in their paper that residual connections accelerate the training of inception networks significantly and also improve their recognition performance on the ILSVRC 2012 classification task. Figure 4.14 shows an example of the usage of residual connection in a 35×35 grid inception module.

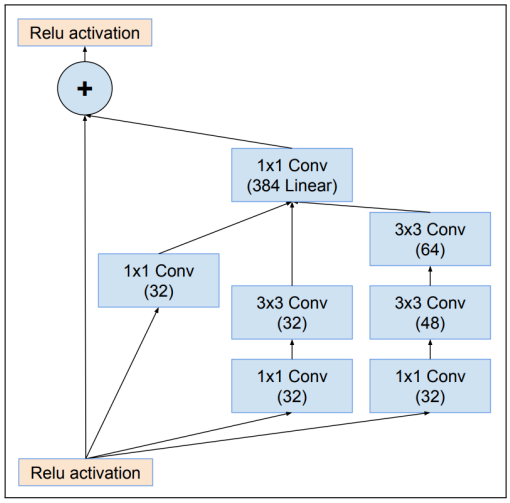


Figure 4.14: Example part of the InceptionResNet from the original paper [21]: The inception layer is shown by the five convolution layers on the right and the residual connection is implemented by the 1×1 convolution layer on the left and the combination in the last 1×1 convolution layer on the top.

4.3 Our baseline approach

For our baseline, a ResNet50, pre-trained on ImageNet [19], is used as feature extractor. The last layer of ResNet50 (see figure 4.7) is replaced by three fully-connected layers and a classification layer, which already contains the amount of all categories of the given dataset (see figure 4.15). First, the images are pre-processed. As ResNet50 uses mode 'caffe', the images are first converted from RGB to BGR, and then each color channel is zero-centered, with respect to the ImageNet dataset. After that, the images are resized, to fit the ResNet50 input size of $224 \times 224 \times 3$. In order to improve the performance, a resize-layer is added at the beginning of the ResNet50-input-layer, as a part of the feature

extraction, which is computed directly on the GPU. The baseline uses a completely frozen ResNet50, as our tests show that this freezing strategy outperforms each other strategy, where parts, or the whole feature-extraction network, is trained (see section 7 for a detailed analysis). This baseline is a first naïve approach, where the features of ResNet50 are used for transfer learning, similar to the naïve approach CWR (see 3.4). All neurons of the last layer are trained at every sequence with a cross-entropy loss, ADAM optimizer and RELU activation function. One of the problems of this baseline is, that first, the network is limited by its last layer. The amount of neurons in the last layer is fixed and corresponds to the number of categories, it is able to learn. Furthermore, this architecture suffers from catastrophic forgetting, shown in the result section 7. This is because of the used classification loss with softmax, which strength is the learning from a static dataset. If it is used in a continual manner, the network always focuses on learning the categories of the current sequence, without caring about the previously learned categories.

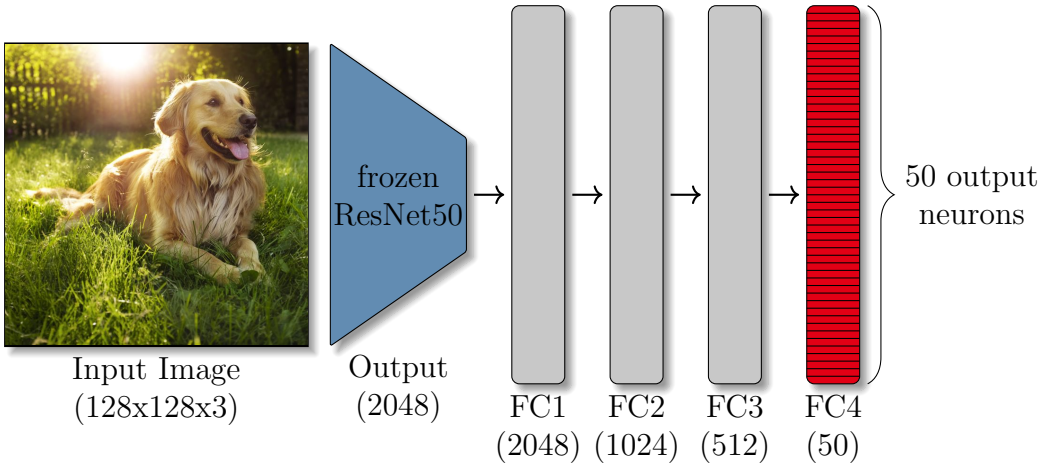


Figure 4.15: Our baseline-network architecture for a total of 50 categories. First, the feature map of each image is calculated by a completely frozen ResNet50 network. The input image size differs between the datasets. In this case, the image size is $128 \times 128 \times 3$. As ResNet50 by default only accepts a size of $224 \times 224 \times 3$, the images are first resized. The application of a ResNet50 results in a 2048×1 feature space. The features are forwarded into three fully-connected layer with 2048, 1024 and 512 neurons and finally a classification-layer with 50 neurons.

5 Our Approach

For our approach, the baseline in 4.3 is extended from a network for category classification, to a continuously learning one. First, an architectural strategy is used to transform our baseline to be able to continually learn new categories. After that, several regression strategies are introduced, to prevent catastrophic forgetting, while learning new categories. Giving the network data of previous training examples in a rehearsal fashion is less practical relevant in robotics. But, at the end of this work, a rehearsal strategy is shown, which also highlights the difference to our online learning approach.

5.1 Architectural strategies

The first step, to make it even possible for our baseline to learn in a continual manner, is by letting the network grow dynamically. First, a simple approach is shown, by only editing the last layer of the network according to the categories it knows. As this approach does not seem to be enough as shown in our experiments, it gets improved by changing not only the last layer, but the whole head of the network, when new categories arrive. As proposed in our baseline, a convolutional neural network (CNN) is used to extract the features from an image. Therefore, several state-of-the-art CNNs are evaluated (see chapter 8). Each of them is pre-trained on ImageNet [19] and the last layer is removed.

5.1.1 Expanding Network

Due to the procedure, showing new categories in sequences, in this thesis it is proposed to let the network grow with respect to the number of categories and sequences. This step is important to enable the network to continually learn new categories, without having a limitation at a certain amount of categories. Therefore, a first proposal is to expand the last layer of the network. This is similar to the PNN, CWR or AR1 approaches, shown in the related work section, but compared with them, our approach is not limited in the number of categories it can learn.

The process starts with a given number of categories, in the so called base training (= sequence zero). The number of categories in each sequences depends on the used dataset. Most of the time, there are between five and ten categories per sequence. It is also common, that the number of categories in the base sequence is higher than in the following sequences, as this one counts as basic knowledge. The base learning step works exactly like our baseline approach. Furthermore, ResNet50 is used for feature extraction, pre-trained on ImageNet [19]. In this step, all fully-connected layers and the classification layer are initialized according to the number of categories in this sequence (see figure 5.1).

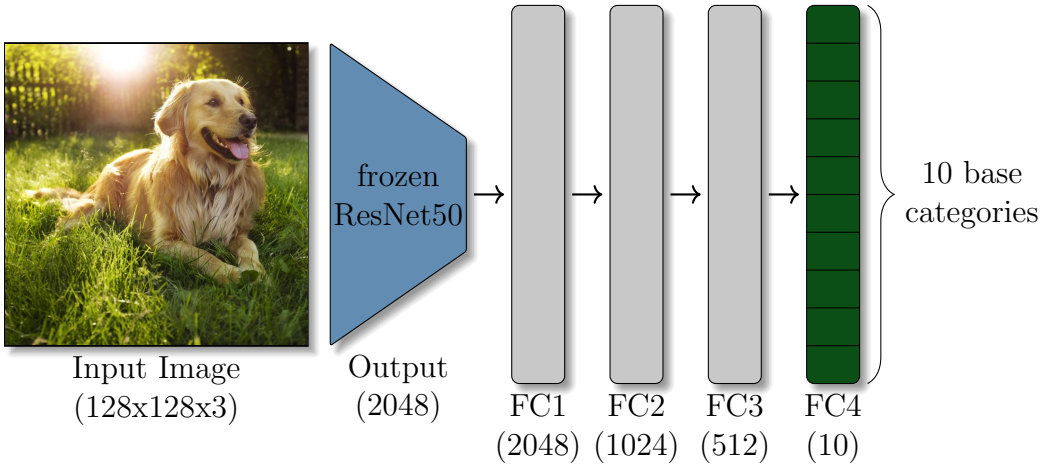


Figure 5.1: Base Training (Sequence zero): This is the same procedure, as shown in our baseline approach (see figure 4.15). Each picture of the first sequence is processed by a frozen feature extraction network, in this case ResNet50, and afterwards the feature output of 2048×1 goes trough several fully connected layers and a final classification layer, where the amount of neurons is exactly the number of new categories in the current sequence. In this example case it is ten. The green color indicates an initialized, not yet learned, classification layer.

After that, the network is trained using a cross-entropy loss function with softmax.

So far, this is how a common neural network learns categories. But, as soon as a new sequence arrives, the behavior changes. Now, the whole network from the previous sequence stays the same. Only the classification layer expands the number of neurons according to the amount of new categories in this new sequence (see figure 5.2). After expanding, all fully-connected layers are trained on the new data. The ResNet50 stays frozen. This procedure is repeated for every following sequence (see figure 5.3).

It has to be highlighted, that this step only enables the network to be able to learn continually. However, the problem of forgetting is not solved, yet. In order to solve this problem, different regularization strategies are proposed. Therefore, starting from the first sequence, the loss function of the baseline is not used anymore. But, this is described in section 5.2. Next, an improvement of the expanding network approach is described.

5.1.2 Different Heads

In the last section, it is proposed to only adapt the last layer of the network, in order to enable it to grow continually. To the best knowledge of the authors, every other architectural approach in the literature changes the last layer. But, there are also important connections within the other fully connected layers, which are getting hurt at every new sequence. If a network trains all categories at once, the connections within the

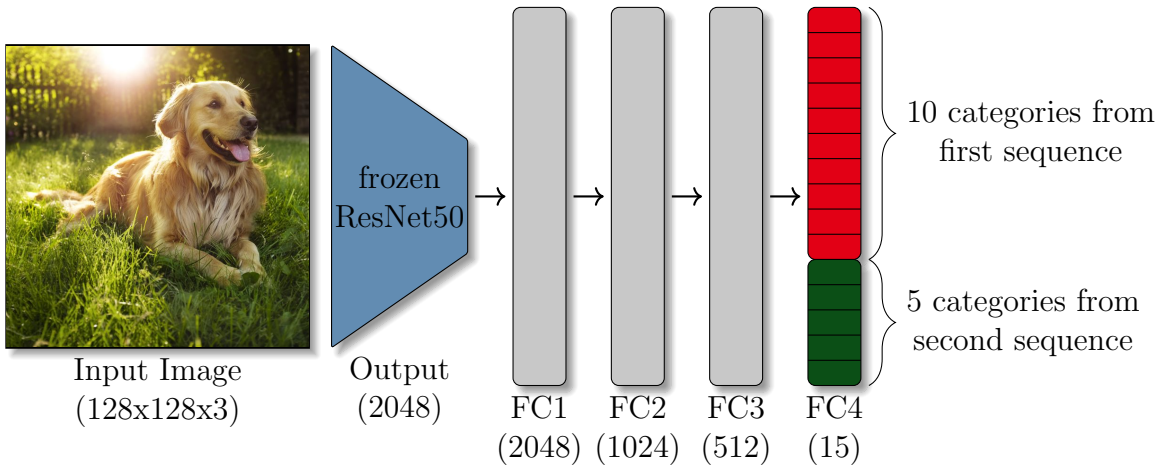


Figure 5.2: Expanding network at the second sequence: Compared to the network in the first sequence (see figure 5.1), only the last layer of this network changes from ten to 15 neurons. The red colored part of the classification layer, are those neurons, which are already trained from the previous sequence and the green part indicates the new initialized (untrained) neurons for the new categories of the current sequence.

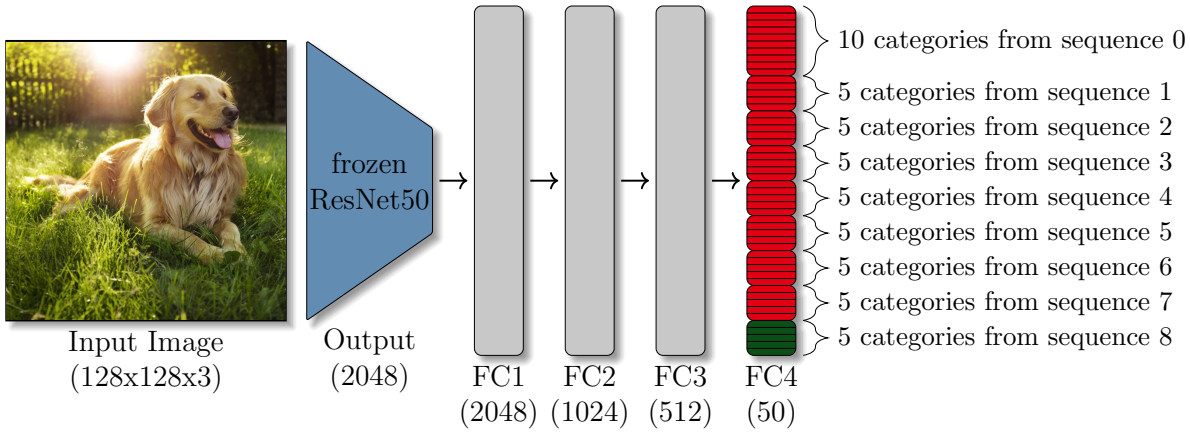


Figure 5.3: Expanding network in sequence eight: This figure shows the network structure in the eight sequence with now 50 neurons in the last layer. Again, the red part of the classification layer are the already trained neurons and the green part are the new initialized neurons of the current sequence.

fully-connected layers are optimized for all of them, but as the categories are learned in separated sequences, this is not true for incremental learning. Instead, these weights are always trained on the current sequence, where previous object categories are not present anymore and therefore connections, which have been important at previous sequences, are changed to improve the current sequence performance, with leads to forgetting.

In order to solve this problem, instead of only changing the last layer of the network, when a new category arrives, now a new head for each sequence is created, including all fully connected layers (see figure 5.4). Thus, everything until the end of the feature extraction network stays the same, and for each sequence, their own fully-connected layers are created, which all get the same feature input of 2048×1, but now the network is able to better address the features treatment regarding to the categories of certain sequences, instead of taking all categories into account. Therefore, the network is able to better adapt to the different categories. This assumption is proven by our result section, as the performance of our approach was benefiting from the usage of different heads instead of only changing the last layer.

Dynamically expanding the network is essential to be able to learn in a continual manner. But, that also leads to a linear growing memory consumption and computation time, which is analyzed in the following.

Each weight consists of three float values (ADAM optimizer), summing up to 12 bytes and one float value for the bias. Taking a previous layer with 512 neurons, as shown in figure 5.1, when adapting the last layer, each new category therefore needs about $(512 + 1) \cdot 12$ byte $= 6156$ byte. Taking this into account, the model is able to learn thousands of categories with a few megabyte, thus the memory consumption is negligible for this case.

This is different for the second case, where different heads are used for each sequence, as this needs more neurons per category. The memory consumption of one category, when using different heads, depends on the number of categories per sequence and on the number of fully connected layers and their size. If it is calculated with four fully connected layers, like shown in figure 5.1, and ten categories per sequence, the memory needed for one category is about $(2048 \cdot 1024 + 1024 \cdot 512 + 512 \cdot 10$ weight values $+4$ bias $) \cdot 3 = 7,879,692$ float values $\cdot$ 4 byte$)/10$ categories ≈ 3.15 megabyte per category. With that approach it takes about three gigabyte to learn one thousand categories, but in the use-case of learning different household object categories, which is the aim of this thesis, this is enough. Also, for most of other possible tasks, a few hundred different categories should be adequate, especially considering that one of the largest image datasets contains 1000 categories. Furthermore, our best performing configuration uses two fully-connected layer,

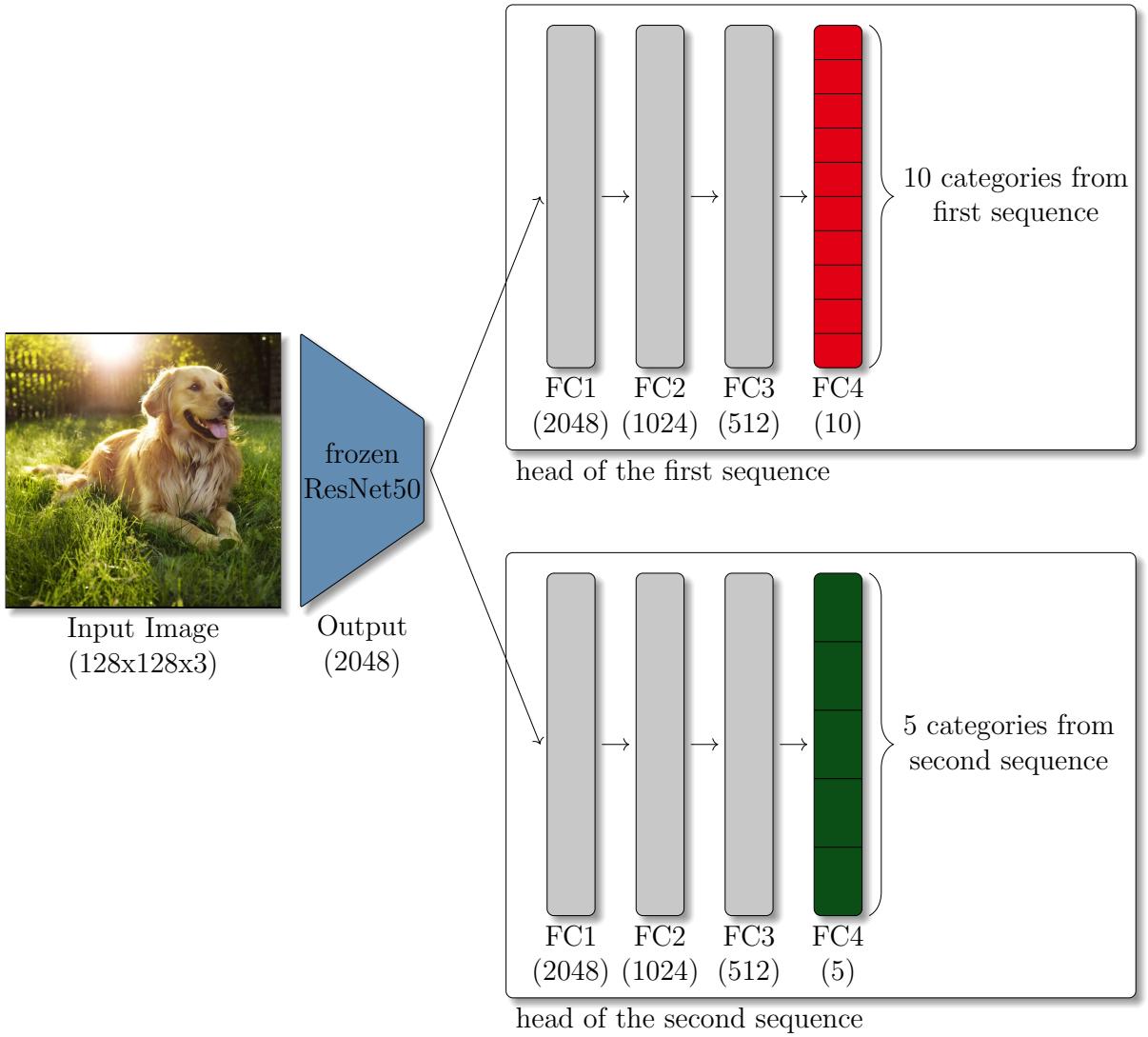


Figure 5.4: The network expands by adding a head for each sequence, which includes several fully-connected layers. All of these dense layers are exclusively responsibly for the categories of one sequence.

which results in about 2.5 megabyte per category. Therefore, the problem of memory consumption only applies for large numbers of different categories.

The same is true for the prediction time, as the time needed also grows nearly linearly with the number of sequences. For our approach, the computational time is also negligible, as the training and prediction step for thousands of images and up to 50 categories takes less than one second, due to the optimized pipeline (described in section 6.4).

5.2 Regularization Strategies

As our network is enabled to continually learn new categories, the next step is to tackle the problem of catastrophic forgetting. In our approach, several regularization strategies are used to mitigate forgetting, while still enabling the network to learn new categories.

The key idea is to prevent already trained neurons of previous sequences from changing too much. Some approaches in the literature, like CWR [9] completely freezing the old neurons, whereas our proposal is to always use all neurons for training, as they have to learn how to distinguish between the different categories, referring here to the dog-and-cat example in section 2.1.1.

In order to accomplish that, another label for already learned categories is created (called reconstruction label), additionally, a loss function is introduced, where instead of a classification loss, a regression loss is used. Furthermore, a novel loss function is presented, which is called reconstruction loss. In order to build up our online learning approach, which continually learns new object categories, with strongly reduced forgetting of previous learned ones. Finally, problems are shown, which occur, using our approach, as well as our proposals to deal with them.

5.2.1 Reconstruction label

In our baseline, image-label pairs (x, y) are used for training, where the label represents the one-hot encoded category number of an object. Those image-label pairs are necessary, if one uses a classification loss and softmax. In the case of continual learning, this leads to catastrophic forgetting, as the labels of previous shown categories are all set to zero, as those are not represented in the training data anymore. Therefore, in the backpropagation step, all neurons, which are responsible for already learned categories are not considered anymore and are pushed towards zero, while neurons, responsible for the current categories gets strengthened. To prevent this, the usage of another label for already learned neurons is proposed. As a label represents a value, which the networks output should approximate, a usage of the original neuron output is proposed. This value is called reconstruction label.

With respect to the training procedure that means, that at the training of the first sequence categories, the network gets the usual image-label pairs (x_0, y_0). But, as soon as a new sequence arrives, these old labels y_0 are not longer present. Only the new category labels y_1 are present now. Therefore, the representations of the previous categories $y_1[0 : C_{old}]$ would all be zero. But instead of doing so, a replacement by a reconstruction label r is proposed.

In the following, our algorithms of the procedures are presented. Algorithm 1 gives an overview of the training and validation process, where for each sequence, first the training, and then the validation of the network is applied. The training procedure is shown in algorithm 2. There it can be observed, that the training of the first sequence is different to all others, as there is no reconstruction label calculated. The rest is similar. The network gets expanded (according to the description in section 5.1.1) and afterwards, the new sequence gets trained. The calculation of the reconstruction label is depicted in algorithm 3 and additionally in figure 5.5.

Given the model n from our approach, sequences $S \in \mathbb{N}_0$ and categories $C \in \mathbb{N}_0$, where each sequence $s \in S$ has its own categories $C_s \subseteq C$, where $\forall c \in C_s, \dot{c} \in C \setminus C_s : c \neq \dot{c}$ and its network weights θ_s:

- First, the base training on the model n, which its current network weights θ_s, is executed, using the image-class pairs (x_0, y_0) of sequence $s = 0$ (see algorithm 2, line four).

- For each new sequence $s \in S$, first the logits $out_{\hat{s}}$ (network output without softmax) are calculated for each image of the current sequence s, using the weights $\theta_{\hat{s}}$ of the previous sequence $\hat{s} = s - 1$ (see algorithm 3). This value is stored as reconstruction label r_s. As there is a label for each category, $\forall c \in C_{\hat{s}} : r_s[c] = out_{(\hat{s})}[c]$ is applied. So, the reconstruction label does not correspond to the category, but to the reaction of the network for a given input image (see algorithm 2, line six).

- The new network weights θ_s are created, by expanding the model $n_{\hat{s}}$ with the previous network weights $\theta_{\hat{s}}$ according to the amount of categories in the current sequence $|C_s|$ (see section 5.1.1).

- Now, the current sequence is ready to be trained. There, the expanded one-hot encoded labels y_s are used as a target for the new category weights and the newly created reconstruction labels r_s are the target for the old category weights.

If these reconstruction labels are used, the neurons, which are responsible for previous learned categories should stay the same. So, that the network learns to reconstruct the

previous network output, and simultaneously learns the new categories. To make that possible, the loss function has to be adapted accordingly.

Algorithm 1 our training procedure

procedure PROCESS_ALL_SEQUENCES(Network n, Sequences S, Categories C, TrainingsImages x, ValidationImages x^* , TrainingsLabels y, ValidationLabels y^*)
    Accuracy *acc*
    for s in S **do**
        TRAIN_NETWORK(n, s, C, x_s, y_s) ▷ See algorithm 2
        acc.append(VALIDATE_NETWORK(n, s, x_s^*, y_s^*))
    return *acc*

Algorithm 2 Network training procedure

procedure TRAIN_NETWORK(Network n, Sequence s, Categories C, TrainingsImages x_s, Labels y_s)
    if $s == 0$ **then**
        EXPAND_NETWORK$(n, |C_s|)$
        n.train$(X = x_s, Y = y_s)$
    else
        r_s = CALC_REC_LABEL(n, s, x_s) ▷ See algorithm 3
        EXPAND_NETWORK$(n, |(C_s)|)$
        n.train$(X = x_s, Y = \text{concatenate}(r_s, y_s))$
    return n

Algorithm 3 Reconstruction label calculation

procedure CALC_REC_LABEL(Network n, Sequence s, TrainingsImages x)
    $r = \text{List}()$ ▷ Reconstruction label
    if $s > 0$ **then**
        for i in x **do**
            r.append(n.predictWithoutSoftmax$(X = i)$)
    return r

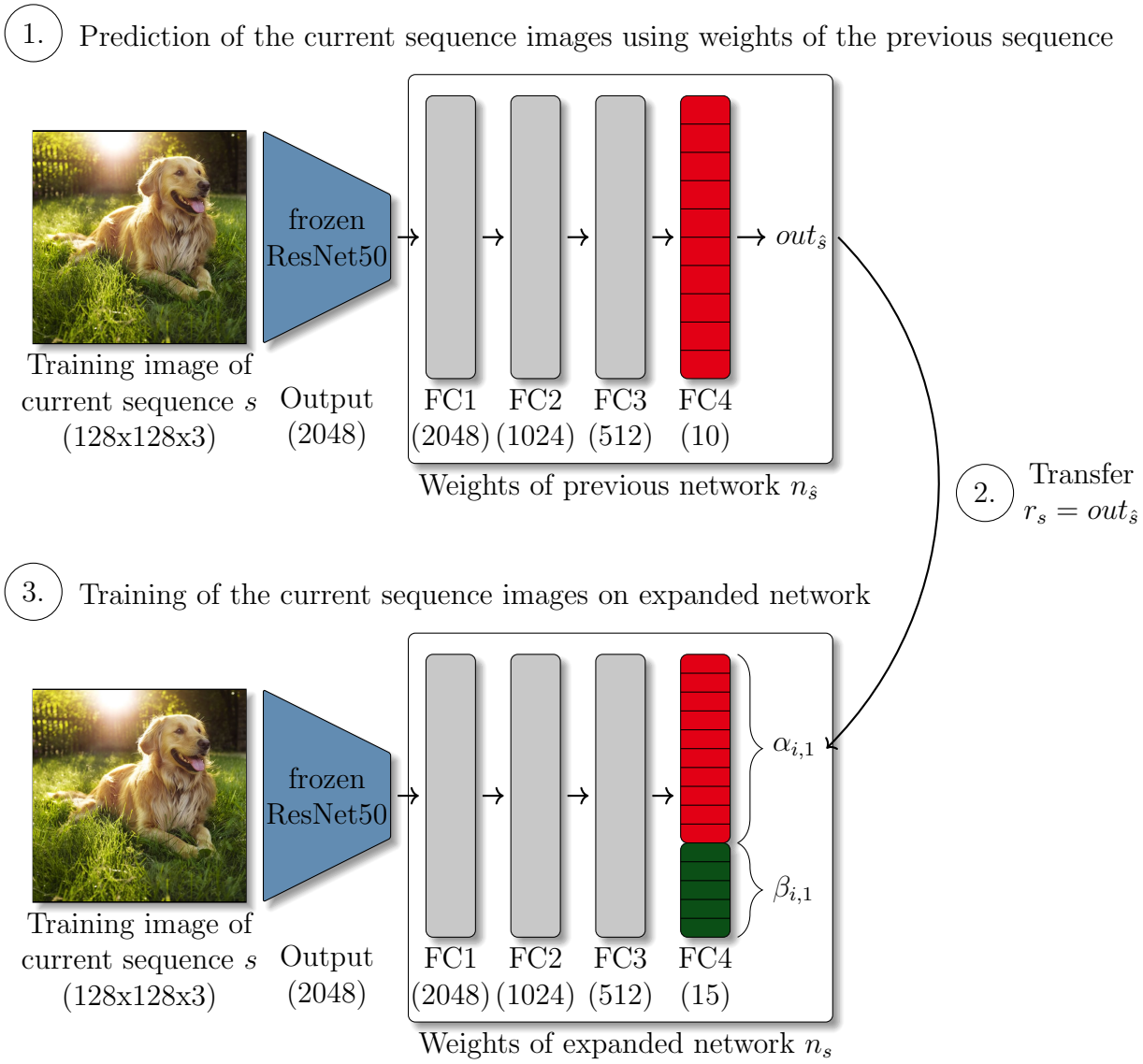


Figure 5.5: Calculation of the reconstruction label: In step one the network of the previous sequence is shown, which contains ten neurons in the last layer, corresponding to the ten different categories. The red color indicates, that these weights are already trained. Now, this network is taken, to calculate the output value *out* (without softmax) for each training example of the current sequence s. These values are stored, as they are used to reconstruct the network output, when the network observes those training images again. Therefore, these values are called "reconstruction labels" in the second step $r_s = out_{\hat{s}}$. After the reaction of the network to the new images is recorded, the network gets expanded according to the number of categories in the current sequence (see 5.1.1) in step three. In the training, the reconstruction labels are now used as the target for the old neurons, which are responsible to predict categories of previous sequences in order to prevent forgetting. For the new neurons, which are responsible to learn the current categories, the classification-labels y_s are used.

5.2.2 Loss function

The standard loss function for classification, which is also used in our baseline, is the cross entropy loss with softmax. This function is used in the first sequence training, but as reconstruction labels are introduced in each of the following sequences, the loss function has to be adapted. Therefore, the usage of a combination of regression and classification loss is proposed, which is discussed next.

Our loss function contains three parts, which are responsible for different things:

1. **Loss on the known category weights α:** The aim is to force the weight space, which is responsible for the known categories of previous sequences, to only slightly adapt to the new categories of the current sequence, but to mainly stay the same, as those categories are not represented in the training anymore. This is achieved by using the reconstruction labels. This part prevents forgetting.

$$R_{i,c} = (out_i[c] - r_i[c])^2 \tag{5.1}$$

Equation (5.1) shows the regression loss, which is used instead of a classification loss, where the squared error between the reconstruction label r and the network output out for a given training example i and a given category c is calculated. This is applied in order to reconstruct the previous network output of the certain neurons and therefore to prevent forgetting. Taking this into account, the loss on known categories for one training example in the current sequence $\alpha_{i,s}$ is defined in equation 5.2.

$$\alpha_{i,s} = \frac{1}{|C_{known,s}|} \sum_{c=0}^{|C_{known,s}|} R_{i,c} \tag{5.2}$$

In equation (5.2) the regression loss from the previous equation 5.1 is used in order to calculate the squared error for each training example i and each known category (neuron) $c \in C_{known,s}$ of the current sequence $s \in S$, where

$$C_{known,s} = \begin{cases} \emptyset & \text{if } s = 0 \\ \bigcup_{s'}^{[0,\ldots,\hat{s}]} C_{s'} & \text{else} \end{cases}$$

With this loss function a distribution shift can be prevented, which is at the core of mitigate forgetting, as each change leads to an increase in the error. But it is still necessary to assure, that the network also learns new categories. Therefore, the second part of the loss function is needed, which is described next.

2. **Loss on the new category weights β:** In order to learn the new categories of the current sequence, the standard cross entropy loss with softmax is used on the new neurons, which is possible, as for this part still classification-labels are used in an one-hot encoding style.

$$E_{i,c} = j_i[c] \log(p_i[c]) \tag{5.3}$$

Equation (5.3) shows the used classification loss - a cross entropy loss with softmax, for one category, where the prediction probability is indicated by $p_i[c] = softmax(out_i[c])$ and $j_i[c]$ defines a binary indicator, which is zero $j_i[c] = 0$, if the predicted label of category c at trainings example i is correct ($out_i[c] = y_i[c]$) or one $j_i[c] = 1$, if the predicted label is wrong ($out_i[c] \neq y_i[c]$). If the prediction is wrong the logarithmic function penalizes bigger errors disproportionately stronger than smaller ones to stake big steps to the right direction and than fine-tune the network. Taking this into account, the loss on new categories for one training example in the current sequence β, can be described in equation 5.4.

$$\beta_{i,s} = \tfrac{1}{|C_s|} \sum_{c \in C_s} E_{i,c} \tag{5.4}$$

In (5.4) the natural logarithm over the probability distribution of the network outputs is calculated over all new neurons of the current sequence C_s. For the output of our network's softmax, the term "probability" is used, which does not represent the certainty of the network about its prediction. With this loss, our network is able to learn new categories.

To balance those two loss parts, a third loss function is defined next.

3. **Loss on all categories γ:** This loss part takes the different loss-ranges of the old and new categories into account to support a good interaction within the network. There, also the standard cross entropy loss with softmax from equation 5.3 is used, but this time on all neurons (all previous and new categories) $C'_s = \bigcup_{s'}^{[0,\ldots,s]} C_{s'}$, where s is the current sequence, which is described in equation 5.5.

$$\gamma_{i,s} = \tfrac{1}{|C'_s|} \sum_{c=0}^{|C'_s|} E_{i,c} \tag{5.5}$$

This third part (equation 5.5) is optional. One has to take into account, that this loss part again uses classification-labels and a classification loss function. Therefore, this part again tries to weaken the old categories, as they are represented with a zero value in the training process. But, in combination with the other loss parts,

the network seems to compensate this problem, as the overall performance gets improved in our experiments.

Combining the three formula parts, our loss function on all training examples of the current sequences s is defined in equation 5.6. There, λ is a hyperparameter to control the importance of not forgetting the old categories in relation to learning new ones. This is valid $\forall s \in S | s \neq 0$. The loss in the first sequence (base) only uses equation 5.5 of the loss function as there are no previous categories given.

$$L_s = \begin{cases} -\frac{1}{|d|} \sum_{i=1}^{|d|} [\gamma_{i,s}] & \text{if } s = 0 \\ -\frac{1}{|d|} \sum_{i=1}^{|d|} [\lambda \cdot \alpha_{i,s} + \beta_{i,s} + \gamma_{i,s}] & \text{else} \end{cases} \tag{5.6}$$

In figure 5.6 these loss functions are shown graphically.

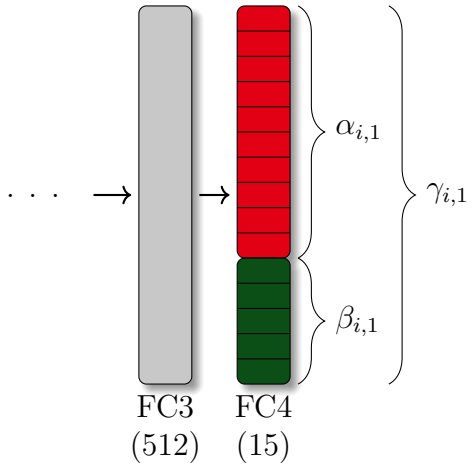


Figure 5.6: Three-part loss function shown at the second sequence $s = 1$ (excerpt from figure 5.2): As shown, the loss is split into three parts, where each has different responsibilities. For the already trained neurons of the classification layer (represented in red) a regression loss function $\alpha_{i,1}$ is used with reconstruction labels r to prevent them from changing and therefore mitigate forgetting. The new, untrained neurons (indicated in green) are trained with a classification loss $\beta i_{i,1}$ and classification-labels y to enable the network to also learn the new categories of the current sequence. To support a balance between not forgetting and learning, the third loss γ applies a classification loss over all neurons with classification-labels y.

5.2.3 Discrepancy in output distribution

Training the network in a sequential manner causes the problem, that the network outputs *out* for each sequence are differently distributed. For example, in one of our tests the output of the base neurons are between $[-30, 30]$ and the output of the first sequence neurons are between $[-90, 90]$ (see figure 5.7). Therefore, the different categories are not weighted equally during training, which leads to forgetting. This phenomenon is caused by the usage of the softmax and cross-entropy loss function for new categories, as those methods strengthen the new categories during training, which is also described in 5.2.1. This problem concerns both, already learned neurons (see figure 5.7) and new neurons (see figure 5.9).

This is caused by a previous usage of the softmax function in the second sequence, where the network forced these outputs to be stronger, compared to the previous sequence, in order to learn the new categories. Now, at the third sequence these neurons are part of the known categories. Because of the distribution difference, these neurons will be considered unequally in the following learning and validation steps. Caused through this unequally consideration, the older sequences categories will be increasingly forgotten over time, as this problem increases from one sequence to another.

To solve those distribution discrepancy problems, two strategies are proposed. In the case of already learned neurons, a division of the difference between the network output

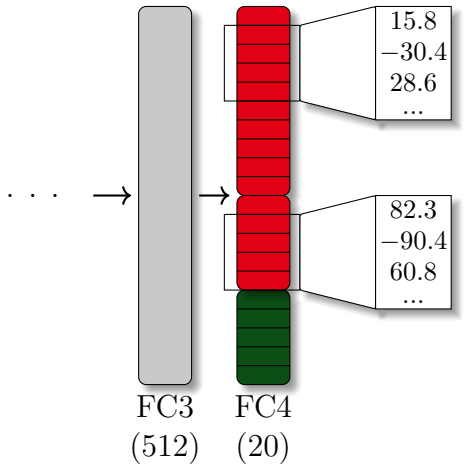


Figure 5.7: Distribution discrepancy within learned neurons: In this figure, a network in its third sequence is shown. The red neurons are the already learned categories of previous sequences and green indicates the new, un-trained neurons of the current sequence. Even by using our three-part loss function, it can be observed that the output values within the red neurons are not equally distributed. Instead, the neurons of the second sequence are stronger weighted than the ones of the first sequence.

and the reconstruction label, by the variance of the weights, is proposed. For the case of a distribution discrepancy in new neurons, the usage of a normalization technique is proposed (see paragraph 5.2.3).

Variance per category To solve the problem of distribution discrepancy within already learned neurons (figure 5.7), a division of the difference, between the network output and the reconstruction label, by the variance per category, is proposed.

$$\sigma[c] = \mathbb{E}[(out[c] - \mu[c])^2], \forall c \in C_{known,s} \tag{5.7}$$

Equation 5.7 shows the calculation of the variance per category $\sigma[c]$, where μ is the mean over the network-outputs out over all known categories $c \in C_{known,s}$ during one training step. This variance is used to adapt the output values of each neuron in a way that they are all treated equally. So, the reconstruction-loss function from equation (5.1) is changed, resulting in a modified reconstruction loss, shown in equation 5.8. This equation shows, that the difference of the network output $out_i[c]$ and the reconstruction label $r_i[c]$ of each neuron is divided by the variance of the respective category $\sigma[c]$. This reduces the discrepancy in the output of the neurons. Therefore, an error of a neuron, with a comparable small output results in a same value, like the same error on a neuron with a comparable big output value. It is important to treat them equally within the training procedure to prevent forgetting.

$$R_{i,c} = ((out_i[c] - r_i[c])/\sigma[c]))^2. \tag{5.8}$$

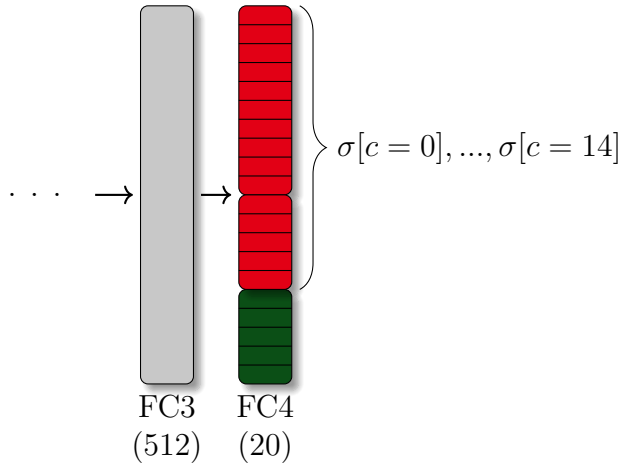


Figure 5.8: The variance per category σ is calculated for each category. In this graphic, the training of the third sequence is shown. It is important to note, that these variances are only calculated for the previous sequence categories (red), using our reconstruction labels.

This technique can only be used for reconstruction tasks, so another technique for classification is proposed next.

Normalization The distribution discrepancy problem on new categories is the source of the distribution problem in older categories, which was discussed before. Figure 5.9 shows how the output distributions develops during training. The output of the new neurons become bigger, in comparison to the output of the already learned neurons. This can be prevented by using a normalization technique. In this approach, a batch normalization is used (see figure 5.10). As outlined in the figure, for each sequence a separated batch normalization is applied, as just one batch norm over all neurons results in the same problem, described before, as the outputs are not equally distributed. Having them separated, gets each neuron normalized in respect to the other neurons of a certain sequence. By this technique, the mean and variance values of the last layer's output are normalized, in order to consider the old and neurons equally during training and validation.

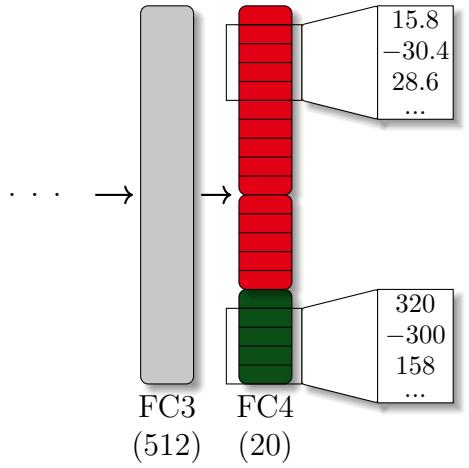


Figure 5.9: Distribution discrepancy at the new neurons become even wider from sequence to sequence as the softmax function always tries to increase the output value of the new categories (green), in order to learn them. This effect is already reduced by our loss function, but still takes place. The consequence is shown in increasingly forgetting over time.

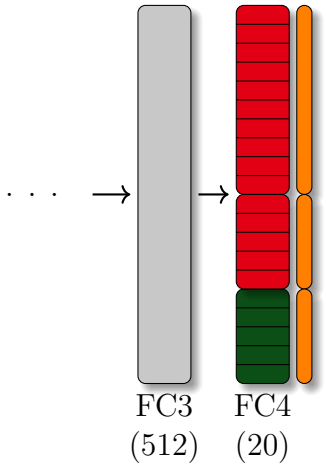


Figure 5.10: Normalization solution for distribution discrepancy: The usage of a batch normalization layer (shown in orange) after the activation function of the last layer is proposed. There it is important that a normalization always only considers the categories of one sequence, as those neurons are trained together. A separation is necessary, as otherwise the neurons of the different sequences are not treated equally.

5.2.4 Reconstruction loss

The source of almost each problem, which causes forgetting, is the usage of a classification loss function in combination with a softmax. A softmax function puts the network outputs in relation to each other. As the targets of different sequences are not distributed in the same range, putting them into relations often causes problems, as some categories are under- and others are over represented. Although, different techniques are already shown to compensate this effect, their negative influence can still be observed (discrepancy in weight distribution).

Therefore, another solution is proposed, going back to the root of the problem and completely replacing the softmax function and classification loss. A regression loss is already used for old neurons, where the labels are reconstructed directly, instead of classification (cross-entropy). In this section, this regression loss is proposed to be used for all neurons, where the softmax function is replaced by clipping the network outputs between zero and one. This new loss function is called reconstruction-loss.

The first change, compared to the loss function, proposed in section 5.2.2, is the usage of clipping, instead of the softmax. Clipping is shown in equation 5.9. Here, the network output out of the neuron c is clipped to zero, if it is smaller than zero, it stays the same, if it is between zero and one, and is again clipped to one if the network output is bigger than one. It is important to note, that those clipped network outputs $\hat{out}$ do not represent a probability distribution, in constract to a softmax function. They are just in a range

between zero and one. A clipping of these values is still necessary as they slow down the training, if the network output gets too high.

$$\hat{out} = \begin{cases} 0 & out_i[c] < 0 \\ out_i[c] & 0 < out_i[c] < 1 \\ 1 & out_i[c] > 1 \end{cases} \tag{5.9}$$

The second step is the change of the loss function. Therefore, each classification loss (defined in equation 5.3) is replaced by a regression loss (defined in equation 5.10). This is true for the loss on new categories β (see equation 5.4) and the loss on all categories γ (see equation 5.5). The new regression loss is defined in the following equation 5.10, where the squared error between the clipped network output $\hat{out}$ and classification-label y is calculated over each training example i and neuron c.

$$\hat{E}_{i,c} = (\hat{out}_i[c] - y_i[c])^2 \tag{5.10}$$

Applied to our loss function, defined in section 5.2.2, the new three part loss function is defined as follow:

1. **Loss on the known category weights $\boldsymbol{\alpha}$:** It stays the same, as the regression loss is already in use on this part (see equation 5.2).

2. **Loss on the new category weights $\boldsymbol{\beta}$**: For this loss function, a classification loss is used (see equation 5.4), which is replaced with the regression loss 5.10. So, our new loss function $\hat{\beta}$ is defined in equation 5.11.

$$\hat{\beta}_{i,s} = \tfrac{1}{|C_s|} \sum_{c \in C_s} \hat{E}_{i,c} \tag{5.11}$$

3. **Loss on all categories $\boldsymbol{\gamma}$**: Here, the original loss function also uses a classification loss with softmax. The new loss on all categories, using a regression loss, is defined as $\hat{\gamma}$ in equation 5.12.

$$\hat{\gamma}_{i,s} = \tfrac{1}{|C'|} \sum_{c=0}^{|C'|} \hat{E}_{i,c} \tag{5.12}$$

Combining all three loss function parts, our new loss function $\hat{L}$ is defined in equation 5.13.

$$\hat{L} = \begin{cases} -\frac{1}{|d|} \sum_{i=1}^{|d|} [\hat{\gamma}] & s = 0 \\ -\frac{1}{|d|} \sum_{i=1}^{|d|} [\lambda \cdot \alpha + \hat{\beta} + \hat{\gamma}] & \forall s \in S | s \neq 0 \end{cases} \tag{5.13}$$

The change in contrast to eq. 5.6 is the updated loss functions for new categories $\hat{\beta}$ and all categories $\hat{\gamma}$.

With this new loss function, the network is able to learn in a continual manner, without the problems described in the previous section 5.2.3. Therefore, the techniques, proposed to prevent a discrepancy in weight distributions, are not necessary anymore, as those shifts do no longer take place. Nevertheless, the reconstruction loss is also combined with other techniques, like batch normalization, to further improve our approach. In our experiement section, different combinations of loss functions with other techniques are tested. An in depth ablation study is shown in chapter 8.

5.2.5 Further techniques

With the combination of our loss functions with the reconstruction label, the baseline is already improved to now be able to learn new categories in a continual manner with less forgetting. However, there are other techniques to further improve the performance of the network.

Learning rate The learning rate has a big influence on the results in our approach. If it is too high, the optimizer changes the old neurons too strong, which again leads to forgetting. If it is chosen to low, the network is not able to learn new categories. The challenge is to find a good balance between those two extreme scenarios. As ADAM is used as an optimizer, it is not guaranteed that the learning rate is decreasing, the longer a neuron is trained. In our approach, it is important that newer neurons are adapted more than older ones, as a change of old neurons always carries the risk of damaging its ability to recognize the category, it is responsible for. It is proposed to support this process, by adapting the learning rate, so that it is possible to choose a higher learning rate for new categories, and decreasing the learning rate for already known neurons, according to their time of existence. Therefore, the old neurons are adapted, by multiplication of the individual learning rate per neuron $\omega_{\hat{s}}$ with a hyperparameter $0 < q <= 1$ in each sequence, resulting in a new learning rate ω_s (see equation 5.14) for the current sequence. By doing so, the learning rate of the oldest sequence is reduced the most and the closer it get to the current sequence, the more the neurons are allowed to adapt.

$$\omega_s = q \cdot \omega_{\hat{s}} \tag{5.14}$$

Technique combinations Different combinations of techniques and also of different hyperparameter are tested. For instance, different batch normalization strategies, activation functions, feature extraction networks, fully-connected layer amounts, etc. Those

are commonly known methods to improve neural networks, although their effect on online learning has hardly been researched so far. Therefore, after the following experiment chapter, where the test environment of our apporach is shown, and the results on them are discussed, the application of these different techniques are shown on our approach in chapter 8.

6 Experimental setup

6.1 Environment

At the department for Perception and Cognition in the Institute of Robotics and Mechatronics of the DLR, a high end computing cluster with 46 GPUs is at free disposal for everyone within the department. The management and access of the cluster within a distributed linux environment is done using slurm. The whole implementation of our approach is done using Python 2.7 and Tensorflow 1.13.

6.2 Datasets

Finding a dataset which fits the purposes of our approach is challenging. Popular datasets such as ImageNet [19] are designed to be used in an offline fashion, thus the entire dataset is split in two parts: one training- and one test set. For continual learning the training and test set is need to be split into a number of sequences. Datasets, which meet this requirement are not common and some are just created recently. Therefore, researcher in the field of incremental/online/continual learning use a variety of different datasets. Table 6.1 lists some of those datasets and the methods, which use those.

As can be seen in table 6.1, there seems to be no consensus on which dataset to use in the field of continual learning. The ImageNet dataset is not suited for our methods, as all the feature extraction networks considered for our approach are pretrained on ImageNet, as this is one of the biggest public image databases [19]. The same is true for MNIST,

Table 6.1: Variety of datasets used for continual learning

Dataset	**Approaches**
Atari game task [72]	[11, 25]
BigBrother [73]	[74]
Caltech-101 [75]	[76]
CIFAR-10 [6]	[77]
CORe50 [9]	[3, 9]
CUB200-2011 [78]	[76, 79]
Datasets based on ImageNet [19]	[80, 81, 51, 77, 22, 82, 79]
iCIFAR-100 [18]	[18, 3, 76]
iCubWorld28 [83]	[74]
MNIST [23]	[22, 82, 11]

which only consists of hand written digits. Based on that, several requirements are made, which the dataset should comply with:

- **Suited for continual learning:** The structure of the dataset should provide the possibility to only show a unique subset of the data in a chronological order.
- **Household objects:** As the target of this work is to deploy our approach on mobile service robots, the dataset should consist out of household objects.
- **Category learning:** The dataset should provide support for category learning. Other scenarios, like instance learning are not in the scope of this thesis. As, there is no need for a service robot to be able to distinguish between several different kinds of cubs. It would be much more preferable that the robot can distinguish between several different categories like scissors, cubs, knife, etc.
- **Non-handhold:** Finally, the objects in the pictures should be placed on a surface and not be hold in a hand of a robot or a human. A dataset without "handhold" is preferred, since the network could not learn the features of an object, but the placement of the fingers, which varies between the different objects. Furthermore, the background of the images could be similar, which hurts the generalization. Additionally, it is of less practical use for a robot to see images of objects, with are hold by a human hand.

After evaluating the datasets regarding these requirements, three of them remain:

- **COIL-100**: A dataset, which is a close match to the requirements, as it consists of non-handhold household objects for category learning. However, it is not designed to be used for continual learning. Nevertheless, a transformation is possible (see section 6.2.1).
- **iCIFAR-100**: This dataset is from the iCARL paper [18]. There they take images from CIFAR-100 [6], re-structure them in a continually manner and call it incremental CIFAR-100 (iCIFAR-100). But, this structure is not publicly available, which prevents a comparison to our approach.
- **CORe50**: This dataset was exclusively designed for continual learning, where they also provide different scenarios, from class to instance learning. Therefore, CORe50 was used for the first continual learning workshop at CVPR2020. The problems with this dataset is that it does not meet all of our requirements as it consists of handhold objects and as it only contains ten different categories and 50 instances, so it is better suited for instance than category learning.

Being in a robotic environment, our approach focuses on household-objects. Therefore, first the COIL-100 dataset is used. Also, the CORe50 dataset is chosen, as it is part of the CLVISION challenge at CVPR 2020 [84]. But as CORe50 does not meet all requirements, it was decided to create our own dataset. So in total, three datasets are used to show the strengths of our approach. In the following, those datasets are analysed and discussed. For continual learning the objects must be organized in sequences and it also has to be guaranteed that a category is only used in one sequence exclusively.

6.2.1 COIL-100

COIL-100 (Columbia Object Image Library) is a dataset of household objects from the Columbia University [8]. It contains 7200 images of 100 different categories. The pictures of each of the objects are recorded using a motorized turntable, which takes pictures from all sides in five degree steps. As all of those pictures have a black background and the same lightning conditions (see figure 6.1), this dataset is not optimal, as the network generalizes better for a diverse dataset. This dataset is also not designed for continual learning but it is possible to split it up into learnable sequences.

The new continual dataset is structured in six sequences, according to our use-case, where a robot is simulated, which observes different categories over time. The first sequence consists of 50 different categories, used as "base knowledge", which the robot should not forget. In each of the following five sequences there are ten additional categories the robot should learn over time. 80 percent of the images are used for training and 20 percent for validation. Our resulting incremental dataset is called iCOIL-100.

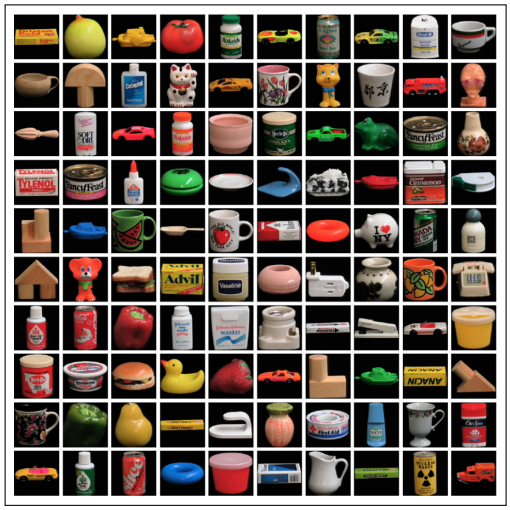


Figure 6.1: Some pictures of all categories of the COIL-100 dataset [8]: Each of these 100 household objects is placed on a turntable and while rotating they take pictures at every five degrees.

During the tests of our approach on this dataset, several problems occur. First of all, there is always only one instance of each object category in the training-, as well as in the validation set. The difference between the images is the different angle, at which those are taken. In order to have a meaningful validation, there should be pictures of other objects of the same category as well. Combined with the lack in diversity, as each image has the same lightning conditions and the same backgrounds, a good generalization of the network is difficult, although a variety of augmentation methods are used. Due to these disadvantages, another dataset is chosen for a further usage of our approach, which is described next.

6.2.2 CORe50

The Continual Learning and Object Recognition, Detection, Segmentation dataset (CORe50) [9] by Lomonaco and Maltoni, is explicitly designed as a benchmark for continuous learning. It consists of 50 domestic, handhold objects, where each belongs to one of the ten categories: plug adapter, mobile phones, scissors, light bulbs, cans, glasses, balls, markers, cups and remote controls (see figure 6.2). The images are taken in eleven sessions (indoor and outdoor), covering different backgrounds and lightning conditions. In total, the dataset includes 164,886 RGB images with a resolution of 128x128, taken from a 15 seconds video for each object, which was recorded with a Kinect 2.0 sensor. Three of those eleven sessions are selected for validation and the remaining eight sessions for training.

In their paper, they already provide different continual learning scenarios:

Figure 6.2: Example pictures of all the 50 instances of CORe50 [9]: There are ten different object categories, one per column, and five instances each, depicted in the rows. The categories are from left to right: plug adapters, mobile phones, scissors, light bulbs, cans, glasses, balls, markers, cups and remote controls.

- New Classes (NC): The dataset is split into nine sequences, which include two categories in the first, and one category in each following one. As one category has five instances, the first sequence consists of ten and the following ones of five object instances. Thus, the network learns one new category at each new sequence. In relation to figure 6.2, this corresponds to a column-by-column processing. It could be criticized, that this is more an instance learning tasks, as the network learns to distinguish between several instances in each sequence, instead of categories. So, the term "class" is misleading here.

- New Instances (NI): The training dataset is divided into eight sequences, according to the eight recorded sessions for each object. Each of these sequences contains the same 50 instances, but in different environmental conditions. At the first sequence, all ten categories are learned and at every following sequence, images of the different category instances are shown. With respect to figure 6.2, this corresponds to a row-by-row processing.

- New Instances and Classes (NIC): This scenario combines both, new categories and new instances split in 79 sequences.

As this thesis focuses on category learning, none of those scenarios is suiting. However, as scenario NC comes the closest to our requirements, it is used in order to compare our approach with others.

CLVISION Challenge at CVPR2020 One of our main reasons, to use CORe50, is, that it was used at the CLVISION Challenge at CVPR 2020 [7, 84]. Just after this thesis started, this first workshop on continual learning was announced. As it is a good opportunity to compare our approach with others, a participation was desirable. The challenge is based on the CORe50 dataset, with three different scenarios and five metrics. The three different challenge tracks New Instances (NI), Multi-Task New Classes (NC) and New Instances and Classes (NIC) are oriented on the continual learning scenarios of the dataset. For NIC, they changed the number of batches from 79 to 391, every other scenario is adapted as described in the paper. The challenge is designed to allow a participation in one or more tracks. As our approach's focus is on learning new categories, a participation in Multi-Task-NC (NC) has been chosen.

The challenge is divided in two different phases:

- Pre-selection phase: Everyone was able to hand in the test- and metric results into the codalab website [84].

- Final evaluation: The top 10 teams of the pre-selection phase had to submit their code for a remote evaluation.

The solutions are evaluated across the following metrics:

- Final accuracy on the test set: This test is applied after the last sequence. In the case of NC, after the training and validation of sequence eight. The network again gets tested on a test set, which contains images of each category instance, learned so far. Those pictures are not contained in one of the validation sets.
- Average accuracy over time on the validation sets: After each sequence, a validation of the current and all previous sequences is applied to the network. For those, a mean over the validation accuracies is calculated.
- Total run time for all trainings, validations and tests.
- Maximal RAM usage.
- Maximal disk usage.

The metric for the disk usage seemed to be important to our task, as this should indicate if a solution uses rehearsal strategies or online learning. Therefore, a comparison of our approach and other, rehearsal-free solutions, might have been possible. But most submissions do not contain the memory, they need for their rehearsal strategies, into this metric. After the challenge, the other approaches papers revealed, that as far it is known by the authors, every solution, which works better than ours, uses a rehearsal strategy. As they store all the data in the RAM, the disk usage metric is not used. So, one of our main critics on this challenge is, that they haven't offered any possibility to distinguish between online learning and other continual learning strategies. But those two methodologies are different from each other and therefore can not be compared in only one table, like done here. Online learning is a more difficult task compared to saving previous training examples.

The CORe50 dataset gathers more and more attention, as it was chosen to be used on the first continual learning workshop at CVPR 2020. This continual learning dataset also meets most of our requirements, but there are still some critics. First of all, the NC scenario definition is an instance learning problem, where for example, different plug adapter, are treated as different classes. An example is shown in figure 6.3, where, according to their definition, two different classes are shown. So, CORe50 only consists of ten categories, but 50 instances. Even the scenario "new classes" (NC) does not focus on learning new categories but on instances, as in each sequence only one category, but

Figure 6.3: CORe50 class examples: On the left, an example of class 3 and on the right an example of class 4 are shown. Both objects are instances of the category "plug adapter" [9].

five instances are shown. As our approach treats each new class as a new category with its own neuron at the last fully-connected layer, it tries to not only distinguish between different categories, but also between different instances. However, our approach is not designed for dealing with instance classification, but still reaches remarkable results on it. Furthermore, knowing several different types of one object category e.g. plug adapter is not practically relevant for a service robot. It is much more relevant to know different household categories.

In addition, the CORe50 dataset does not meet our requirement in respect to non-handhold images. Therefore, it is necessary to create our own dataset, which meets all the requirements. This dataset is described in the following section 6.2.3.

6.2.3 Creation of a new dataset

As there was no dataset found, which fits our requirements perfectly, this thesis presents a novel dataset for continual learning. For the creation, the tool BlenderProc, was used, which is described in the next section. After that, the details of our own dataset are shown.

6.2.3.1 BlenderProc The images of the dataset are created with Blenderproc [4, 5], a modular procedural pipeline based on Blender for creating realistic looking synthetic images for convolutional neural networks. Using this open-source tool one only needs a 3D model from the object, the network should learn. There are already plenty of datasets integrated in BlenderProc e.g. ShapeNet [85], T-Less [86]. Now, these objects can be placed in different rooms, which are also already ready to use, like Replica [87] or SUNCG [88], or in our case a cube with a randomly changing background. There is also the possibility to change the objects, for instance by using displacement, different

materials and/or textures. Furthermore, one can randomly change the lighting conditions and place the camera in different positions to get a big variety of different images, which is proven to increase the generalization ability of the network [89]. Besides color images, it is also possible to additionally create normal, depth and segmentation images.

6.2.3.2 HOWS-CL-25 In this section, the Household Objects Within Simulation dataset for Continual Learning (HOWS-CL-25), a novel synthetic dataset for object classification and recognition for continuous learning is shown. The first version of our dataset already contains 150,795 unique synthetic images using 25 different household objects and 925 instances. In figure 6.4, an example image of each category in the dataset is shown. Although, the images are from synthetic 3D models, the different lightning and shadowing techniques of Blender make them look almost realistic, which is due to the usage of ray tracing. For this dataset, only objects are chosen, with are available in the most households. Those objects are placed on a surface, which makes them look like they are placed on a desk or on the ground - a position they would also be in, in the real world. Most of our 3D Models are taken from ShapeNet dataset [85], others are from different internet sources.

Structure of the dataset The 25 household categories are split into training and validation sets, distributed over several sequences in order to learn in a continual manner. Instances, which are contained in the training set are not contained in the validation set and vice versa.

The images are organized in five sequences $[0, ..., 4]$, which contain five categories each. About ten percent of the images are used for validation and the other 90 percent for training. For a better overview about the categories and their assignment to the different sequences, tables 6.2, 6.3, 6.3, 6.4, 6.5 and 6.6 are provided. Each category of the dataset has at least four different 3D models/instances and 6000 pictures, which are taken in a randomly changing environment. Additionally, the material and displacement of the 3D models has been changed randomly, which results in more than 925 instances in total.

To accomplish that, first a cube has been created, with randomly placed ground and walls (containing different types of asphalt, bark, bricks, carpet, concrete, cork, corrugated steel, gravel, ground, ice, marble, metal, paint, stone, plank, road, sheet metal, snow, solar panel, terrazzo, tiles and wood), and a light source with randomly chosen lightning conditions (different light color and brightness levels). After one of the 3D objects is placed in the resulting room, this object gets customized. This includes a random replacement of the object's material or a changing of the objects textures. For example, the surface of the object "mobile phone" is randomly replaced with plastic and metal materials and

Figure 6.4: Overview of the HOWS-CL-25 dataset: From the top left to the bottom right examples of all 25 categories of our dataset are presented: apple, ball, bowl, camera, cap, egg, glass bottle, headset, milk, mug, pear, scissors, teddy, bag, banana, bread, can, computer keyboard, fork, glasses, knife, mobile phone, pan, pen, spoon.

Table 6.2: Categories of sequence 0

class name	class id
apple	0
ball	1
bowl	2
camera	3
cap	4

Table 6.3: Categories of sequence 1

class name	class id
egg	5
glass bottle	6
headset	7
milk	8
mug	9

Table 6.4: Categories of sequence 2

class name	class id
pear	10
scissors	11
teddy	12
bag	13
banana	14

Table 6.5: Categories of sequence 3

class name	class id
bread	15
can	16
computer keyboard	17
fork	18
glasses	19

Table 6.6: Categories of sequence 4

class name	class id
knife	20
mobile phone	21
pan	22
pen	23
spoon	24

if the material is not replaced, it gets slightly deformed by using a strengthened bump map. So that, for example, dumps on apples become more or less visible. After that, the object is ready and three pictures of this setting are taken, in randomly placed camera positions.

In this way, a diverse dataset is created, containing three pictures per instance and in total 6000 color pictures per category. Additionally, a corresponding normal image, segmentation map and depth image have been created for each of those color images (see figure 6.5). In our approach, only the color images are used.

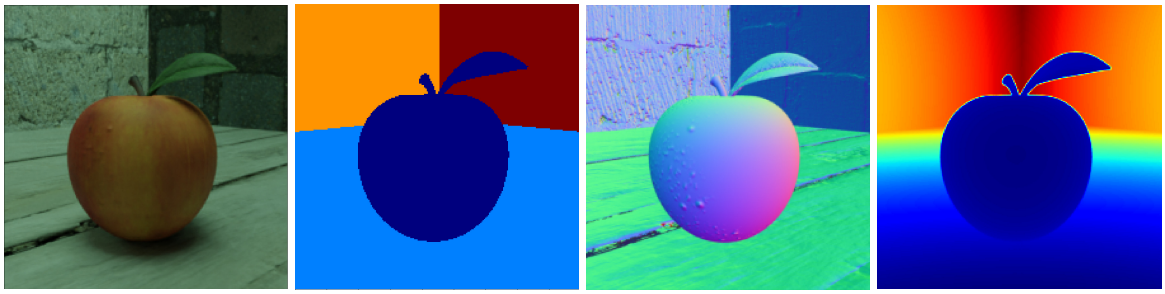


Figure 6.5: Example for the different image types of the HOWS-CL-25 dataset: For each RGB-image of an object (left), a segmentation map (second from left), a normal image (third from left) and a depth image (right) is created.

Dataset comparison Comparing our dataset with CORe50 in table 6.7, it is found, that the first version of our dataset already has almost as many images as CORe50. It contains 2.5 times more categories and more than 18.5 times more instances (only the 925 different 3D objects are counted, without considering the different appearance, resulting of random materials, textures, etc.). Furthermore, the HOWS-CL-25 dataset provides a variety of randomly sampled backgrounds, displaced textures, lightning conditions and camera positions. Thus our dataset is more diverse, compared to CORe50. The diversity is important for the training of a neural network, in order to improve its generalization.

Compared to CORe50 and COIL-100, our dataset satisfies all the requirements (see table 6.8). First of all, COIL-100 is not designed to be used in a continual manner, whereas the CORe50, as well as the HOWS-CL-25 dataset structures the data in different sets, distributed over several sequences and are therefore suited for continual learning. The second requirement is fulfilled from all datasets, since each of them only uses household objects, almost everyone would have at home, too. Next, COIL-100 only has one instance per category, which is not optimal for category learning, since the network can hardly generalize on one example. The HOWS-CL-25 dataset contains more instances, and even more important, also more categories than CORe50. The last requirement is the usage of objects, which are placed on surfaces and are not hold in a human or robot hand. COIL-100 as well as HOWS-CL-25 dataset fulfill this requirement.

As HOWS-CL-25 is the only dataset, which fullfills all our requirements, it is suited best for our online learning approach on mobile robots. Furthermore, it is only a question of computation time, to create an even bigger dataset with BlenderProc, whereas to expand

Table 6.7: Online and incremental learning

CORe50 Dataset	**Our Dataset**	**Factor**
164.866 pictures	150.795 pictures	0,91x
10 categories	25 categories	2,5x
50 instances	> 925 instances	> 18,5x

Table 6.8: Requirements fulfillment of HOWS-CL-25

Requirements	**COIL-100**	**CORe50**	**HOWS-CL-25**
Suited for Continual Learning	✗	✓	✓
Household objects	✓	✓	✓
Optimized for category learning	–	✗	✓
Non-handhold	✓	✗	✓

the number of categories or instances in CORe50 is much more time intensive, as these pictures are manually recorded. A second version, of our dataset is already planed, where the objects are going to be placed in different, virtual designed, rooms, to get an even more realistic and diverse dataset. It is also planned, to make the dataset publicly accessible.

One might be concerned that using synthetic data will not generalize for real world images. But, as shown from Hodaň et al. [89] and Denninger et al. [5], indeed a generalization from synthetic- to real world images, is possible. For the experiments on our approach the HOWS-CL-25 dataset and, in context of the CLVISION challenge, also the CORe50 dataset are chosen.

6.3 Training- and validation procedure

As already shown in the introduction on figure 2.2 the general procedure, incremental learning is conducted, differs from the standard way a neural network learns. The training, as well as the validation images arrive in batches, distributed over sequences, where each category is only available during one sequence. In this section, the special training- and evaluation procedure are described, which is used in our approach, in literature and in the CLVISION workshop.

6.3.1 Training procedure

In the trainings procedure, each sequence has its unique training set, which all contain the same amount of new categories and a similar amount of pictures. What is special, that the data of a specific sequence is available exclusively and only once in the lifetime of a network. While the network has access to the images of a sequence, there are no restrictions in terms of processing these. Therefore, augmentation and shuffling is allowed. In this thesis, the guidelines for "Multi-Task New Classes (NC)" of CLVISION workshop are followed for CORe50 (see section 6.2.2). This procedure is also adapted to our own dataset. So for CORe50 the training is applied in nine sequences, 1-2 categories each, and in case of HOWS-CL-25 in five sequences of five categories each.

During the training, the loss curve and accuracy of the current sequence (new categories), the previous sequences (old categories), as well as the time and the maximum RAM and disc storage consumption is recorded.

6.3.2 Validation and testing procedure

Also the validation of the network is different, as usually a network is trained once, and tested afterwards. Compared to the standard procedure, in continual learning, the

network is validated after each sequence with validation images from the current sequence and, to measure forgetting, the model is also validated with images from all previous sequences as well. Thus validation of the third sequence also includes images from the first and second sequences, separated in individual batches. There are also other techniques, which always validate the data on every sequence, even for future once, like it is proposed by Vincenco Lomonaco [9], but as the network cannot possible know those categories, presented in a future step, our approach validates only on the current and all previous sequences, which the network has seen so far. In the special case of CORe50 dataset, the network is additionally tested after the last sequence. The test set includes never seen images of each instance of CORe50, according to the specifications of the CLVISION workshop challenge. The labels for these test images are not publicly available. Therefore, a file with image name and prediction, is created and send to the workshop for validation. This was only possible until the end of the workshop in May 2020.

6.4 Implementation details

Improving Data First, the generalization of the network is improved, by using different augmentations on the data. This includes hue, saturation, brightness, contrast, cropping, flipping and rotating the images. In this way, the data amount is quadrupled (1× original, 3× augmented data) in the case of CORe50 and doubled (1× original, 1× augmented data) for our own dataset. As the HOWS-CL-25 dataset is already diverse, compared to CORe50, it needs less augmentation to improve the generalization of the network. In order to speed up the training process those data sets are stored in TFRecords, a data-format from TensorFlow, to store a sequence of binary records. This is optimized for streaming data to a network, as the data is serialized and stored in a set of files (100-200MB each), and therefore can be read linearly. Furthermore, as it is found that the best results are achieved, when the feature extraction network is frozen, TFRecords for the feature-output of each training- and validation set are created, as otherwise the network calculates the same features over and over again. With this methods, the training- and validation process speeds up by a good margin, by simultaneously improving the generalization ability, due to the usage of augmentations.

In our own dataset, also normal-, depth- and segmentation images for each color image are created, which can also be considered in the training process. In our approach, only the color images are used. The other image information could be used, for example, for a use-case, where the robot is also equipped with a depth camera.

Another important practice is the usage of shuffling, since the pictures are organized according to their category. For instance, if the first 1100 images are from the category

"apple" and the batch-size of the network is 1024, the network only observes pictures of one category. As already described in our dog-cat example in section 2.1.1, it is more important for a network to learn the difference between several categories, than the features of a category itself, thus it will not learn properly. To prevent this, a shuffling of our training data sets is used. The choice of the shuffle-size and batch-size depends on the available memory. As shuffle-size indicates how much data is loaded into the RAM of the computer, and batch-size are the datapoints loaded in the graphic card's memory. The oldest node, used in the cluster, has 30 GByte RAM and 11 GByte GPU memory. In regards to this, for our approach the following shuffle- and batch-sizes is chosen (see table 6.9).

Table 6.9: Our shuffle- and batch-size settings

Data	**batch-size**	**shuffle-size**
Images	32	8192
Pre-calculated features	8192	32768

Evaluation Additionally to the accuracy at every sequence, different runs are compared, according to the mean over all accuracy values (all sequences), the mean accuracy of the last sequence especially and the median over all accuracy values. Furthermore, the results are split into base (first sequence) and sequences, as the development of the results of the first sequence are a good indicator to measure catastrophic forgetting. The mean accuracy over the validations of the last sequence is the most important value, as it indicates how much the network knows or forgets at the end. For a better comparison, a script is developed, which automatically evaluates the different runs according to their hyperparameters and accuracy results. The result of the evaluation is plot in customizable graphs. An example is shown in figure 6.6, where the results of the different runs are shown depending on the used learning rate and loss function.

Additional to the evaluation of the different runs, TensorBoard is also used, to evaluate the behavior of the loss function of promising runs. TensorBoard is a visualization toolkit from TensorFlow, with various evaluation tools. In our work, it is used to evaluate the change in the loss values and also to see how the error of the different parts of our loss function behave. This tool also allows to better understand how different techniques improve or deteriorates the results. An example of this is shown in figure 6.7.

To further analyse which categories the network is having problems with, another script is provided to create a confusion matrix. There, it is possible to compare the original

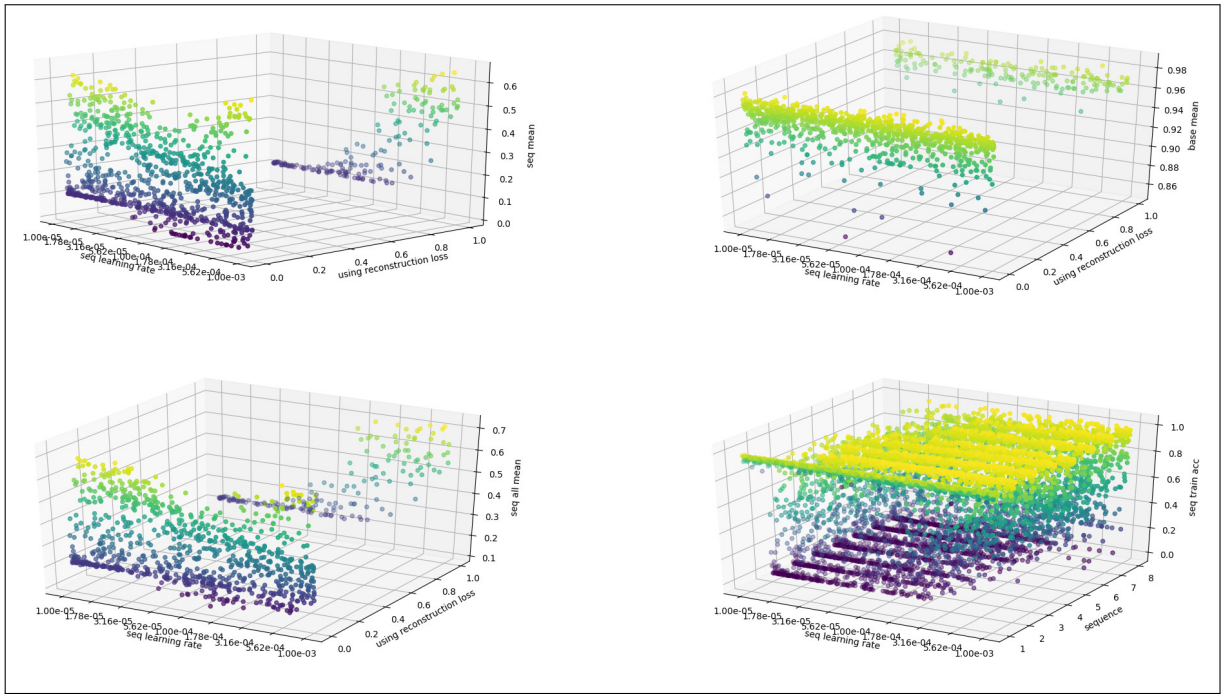


Figure 6.6: An example of our scatter plot evaluation: The first plot on the top left shows the mean over the accuracy over all sequences, except of the first one. The first (base) sequence is evaluated on the top right plot and a combination of both is shown in the bottom left plot. Those three plots have the same structure. They show the resulting average accuracy (where yellow is the best, and purple the worst result) on the y-axis in dependence on the chosen learning-rate (x-axis) and loss-function (z-axis). First of all it is shown, that the training of the first sequence works good and the development in the following sequences highly depend on the chosen learning-rate, whereas each loss function works best with different learning-rates. In the plot on the bottom right, the development of the average accuracy (y-axis) over the sequences (z-axis) in dependence of the chosen learning-rate (x-axis) are shown. This scatter plot contains 1012 test runs, which took about seven hours on six GPUs.

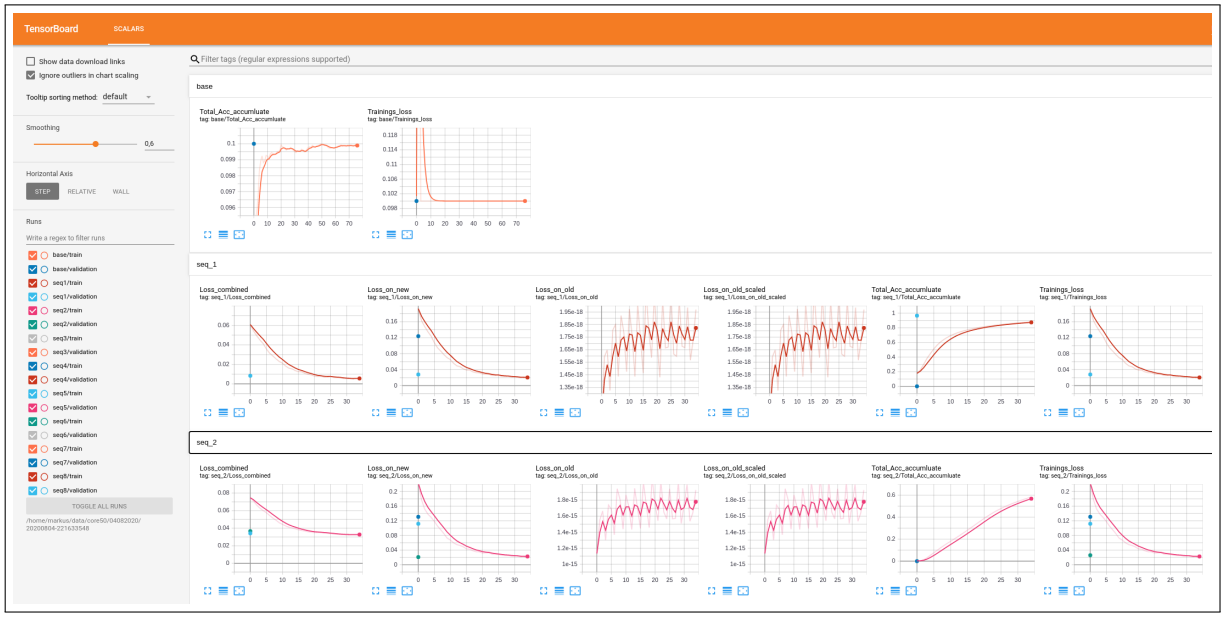


Figure 6.7: A TensorBoard example run: There are different graphs, ordered by sequence. From the second sequence, the different loss parts are shown. In this example it can be observed that the overall performance is quite good, as the total training loss is falling and the accuracy rising. Especially the loss on all categories and the loss for new categories improves over time, but the loss on old categories, which indicates forgetting is not dropping but increasing over time. This increase is low but indicates that it might be a good idea for this example to strengthen this part of the loss function in a following test.

label with the prediction of the network, for each category, in each sequence, or over all sequences. An example for this confusion matrix is shown in the CORe50 results section 7.1.1.

6.4.1 Feature extraction networks

As the used feature extraction networks of this approach are frozen, the pre-processing of the input images has to be adapted according to the one, these CNNs are trained with. And the network also has to be adapted according to the respective feature output. In this section, the different characteristics, which are important for implementation, are shown.

There are two different pre-processing modes used:

- Mode 'caffe': Converts the images from RGB to BGR, then each color channel gets zero-centered, with respect to the ImageNet dataset [19]. This mode does not scale the images.
- Mode 'tf': Scales the pixels sample-wise between -1 and 1.

In table 6.10, the different implementation characteristics of the different feature extraction networks are shown.

Table 6.10: Implementation characteristics of the different feature extraction networks

CNN	**Pre-processing mode**	**input size**	**feature output size**
ResNet50	caffe	$224 \times 224 \times 3$	2048
ResNet50V2	tf	$224 \times 224 \times 3$	2048
Inception-v3	tf	$299 \times 299 \times 3$	2048
InceptionResNet	tf	$299 \times 299 \times 3$	1536

7 Results

In this chapter, the results of our approach are presented in comparison to our baseline and solutions from the literature. Those results are arranged according to the tested datasets. First, the results on the CORe50 dataset are shown, which also includes the performance in the CLVISION workshop. After that, the results on the HOWS-CL-25 dataset are presented.

7.1 CORe50

In the following, the results of our approach and our baseline on the CORe50 dataset are shown. In order to assess the difficulty of the dataset, the first table 7.1 shows the maximal possible accuracy value for each of the nine sequences. This is found by training each sequence offline, using an usual CNN architecture (in our case a ResNet50 with two fully-connected layers). The results show, that the difficulty of each sequence is balanced and that the network is able to learn all sequences fairly accurate, except of sequence four with seems to be more difficult than the others.

Table 7.2 shows the results of our baseline approach on CORe50. Compared to table 7.1, it can be seen that the network is able to learn the respective sequences. But, it instantly forgets everything else it has learned from the previous sequences. A possible solution for that problem is a decreasing of the learning rate. The vanilla baseline approach uses the default ADAM learning rate of $1e{-}05$. In order to show, how the baseline performs with a modified learning rate of $7e{-}06$, table 7.3 is provided, where the baseline is modified in order to mitigate forgetting. As can be seen in the table, the modified baseline forgets more slowly compared to the vanilla version, but at the last sequence it forgot everything from sequence zero to sequence five and most of sequence six. Furthermore, the learning performance is worse than the one of the vanilla baseline, especially in sequences one, two and six.

Table 7.1: Offline results of the different sequences of CORe50

Sequences	Seq. 0	Seq. 1	Seq. 2	Seq. 3	Seq. 4	Seq. 5	Seq. 6	Seq. 7	Seq. 8
Results	97.33	98.67	98.67	97.77	93.77	97.33	99.11	97.77	98.22

Table 7.2: Our baseline approach on CORe50 displays catastrophic forgetting.

Training \ Validation	Seq. 0	Seq. 1	Seq. 2	Seq. 3	Seq. 4	Seq. 5	Seq. 6	Seq. 7	Seq. 8	Avg. Acc.
Sequence 0	96.66	-	-	-	-	-	-	-	-	96.66
Sequence 1	0.00	98.66	-	-	-	-	-	-	-	49.33
Sequence 2	0.00	0.00	97.77	-	-	-	-	-	-	32.59
Sequence 3	0.00	0.00	0.00	98.66	-	-	-	-	-	24.66
Sequence 4	0.00	0.00	0.00	0.00	95.11	-	-	-	-	19.02
Sequence 5	0.00	0.00	0.00	0.00	0.00	95.11	-	-	-	15.85
Sequence 6	0.00	0.00	0.00	0.00	0.00	0.00	98.22	-	-	14.03
Sequence 7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	95.99	-	12.00
Sequence 8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	99.11	11.01

Table 7.3: Our modified baseline approach on CORe50 with an adapted learning rate

Training \ Validation	Seq. 0	Seq. 1	Seq. 2	Seq. 3	Seq. 4	Seq. 5	Seq. 6	Seq. 7	Seq. 8	Avg. Acc.
Sequence 0	76.89	-	-	-	-	-	-	-	-	76.89
Sequence 1	12.00	48.44	-	-	-	-	-	-	-	30.22
Sequence 2	3.56	20.00	42.67	-	-	-	-	-	-	22.07
Sequence 3	0.22	4.89	39.11	64.44	-	-	-	-	-	27.17
Sequence 4	0.00	0.00	0.00	56.44	70.02	-	-	-	-	25.33
Sequence 5	0.00	0.00	0.00	30.67	40.89	68.89	-	-	-	23.41
Sequence 6	0.00	0.00	0.00	18.22	4.00	57.33	39.11	-	-	16.95
Sequence 7	0.00	0.00	0.00	0.00	0.00	8.44	22.67	58.67	-	11.22
Sequence 8	0.00	0.00	0.00	0.00	0.00	0.00	5.78	51.11	50.67	12.00

For a comparison, the results of our approach on the NC scenario of the CORe50 dataset are shown in table 7.4. With respect to the offline learning accuracy in table 7.1, the performance of our approach is decreasing from sequence to sequence, which is also a weaker performance regarding to the vanilla baseline approach in table 7.2. The same behavior can be observed in the modified baseline approach in table 7.3. But, compared to the baseline, the problem of forgetting is improved massively by our approach. In some sequences (e.g. 1, 5, 6) it seems like the network even stopped to forget at all. Therefore, the overall performance is better, compared to the baseline approaches. In the last sequence the performance of the baseline is improved by a factor of about five.

Table 7.4: Results of our approach on CORe50

Training \ Validation	Seq. 0	Seq. 1	Seq. 2	Seq. 3	Seq. 4	Seq. 5	Seq. 6	Seq. 7	Seq. 8	Avg. Acc.
Sequence 0	97.33	-	-	-	-	-	-	-	-	97,33
Sequence 1	88.66	92.88	-	-	-	-	-	-	-	90.77
Sequence 2	82.66	91.55	80.88	-	-	-	-	-	-	85.03
Sequence 3	80.22	81.33	80.00	84.88	-	-	-	-	-	81.61
Sequence 4	79.77	75.99	75.55	83.99	72.00	-	-	-	-	77.46
Sequence 5	77.55	75.55	69.77	83.55	68.44	68.44	-	-	-	73.88
Sequence 6	77.55	75.55	68.44	83.55	65.77	67.55	65.33	-	-	71.97
Sequence 7	77.33	75.55	68.00	80.88	65.33	67.55	64.44	79.55	-	72.33
Sequence 8	76.22	75.55	68.00	79.55	64.88	67.55	64.44	79.11	72.88	72.02

Additionally, the comparison of our approach to the vanilla baseline is presented in figure 7.1. There, it is shown that the baseline forgets faster, compared to our approach. Furthermore, it is depicted that the baseline is better than an random guessing approach.

In order to get a deeper look into the performance of our approach on the different sequence levels, figure 7.2 is provided. There it is shown, that forgetting especially takes place at the first and second sequences. In later sequences, for example five and six, it seems like the network even stopped forgetting at all.

The tests on the CORe50 dataset show, that our online learning approach improved our baseline and is therefore able to learn new categories with less forgetting. Furthermore, it can also be used for instance learning, as the scenario NC is a instance classification task. Next, an exemplary confusion matrix is shown, to further analyse the performance

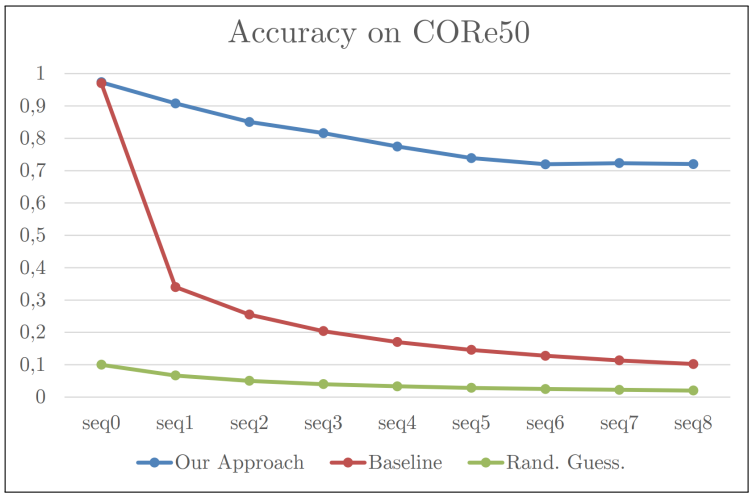


Figure 7.1: Accuracy on the CORe50 dataset: Comparison of our approach (blue) with the baseline (red) and an approach with random guessing (green). Due to our different techniques, the performance of the network is improved, with a high margin. The network seems still to forget, but this is slowed down enormously and even seems to stop from sequence 6.

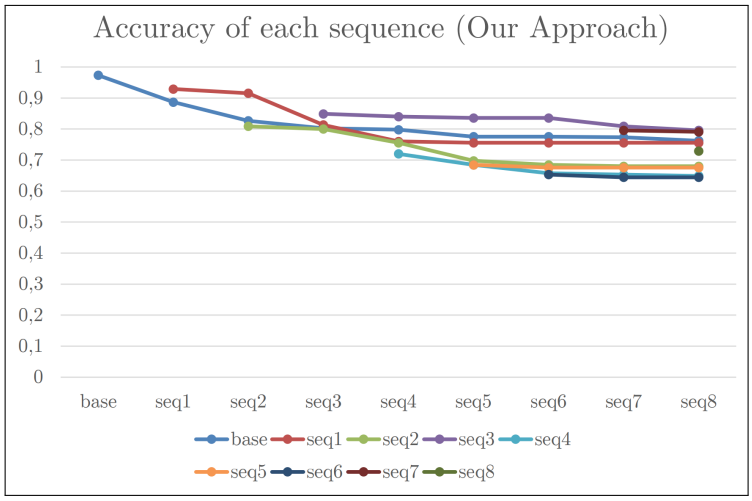


Figure 7.2: The performance of our approach over the sequences on CORe50: The level of forgetting is different from sequence to sequence. Some seem to forget comparably strong in the beginning (base, sequence 1) and others seem to not forget at all (sequence 5,6,7).

of our approach.

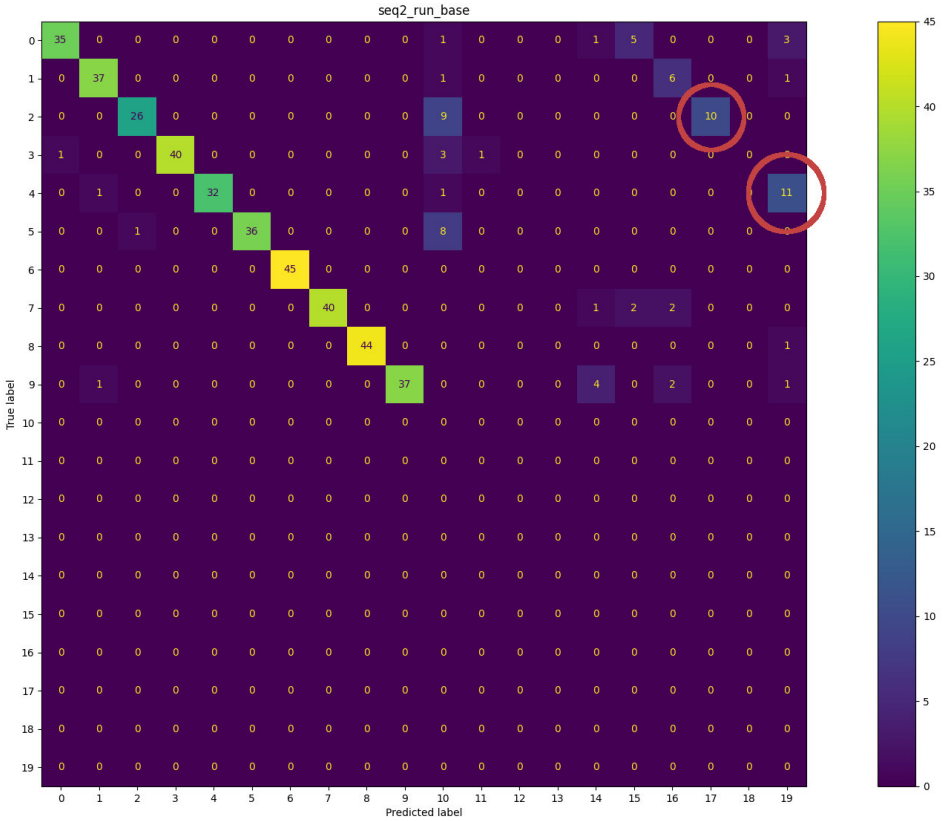
7.1.1 Confusion matrix on CORe50

A confusion matrix shows, which categories the network struggles with. In an exemplar confusion matrix, depict in figure 7.3 (a) it is shown, that the network confuses some instances of category "plug adapter" with some instances of category "light bulbs". The reason could be, that the network learns the position of the hand instead of the objects, which can be seen in figure 7.3 (b). Or, as the network learns to distinguish between different instances of one category, it less focuses on distinguishing between different categories.

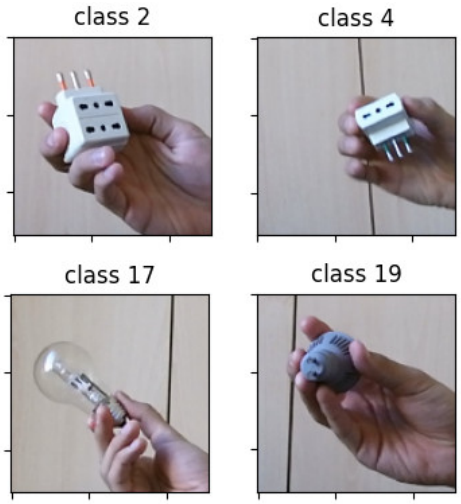
Next, the results of the CVPR workshop are shown.

7.1.2 CLVISION Challenge at CVPR2020

At the CLVISION Challenge, our approach is evaluated in the NC and NI scenario. In table 7.5, the result of our approach, compared to the winner of NC is shown. The metrics "accuracy", "RAM usage" and "time" are extracted from the challenge. As "RAM usage" is not representing the memory, which is used for saving previous training data, additional columns are added. The table shows, that each approach which performs better than ours uses a rehearsal strategy and is therefore not classified as online learning. The winner of Multi-Task-NC for example uses a memory of up to two gigabyte to store previous data. It is possible to store the whole CORe50 dataset in approximately two gigabyte, when



(a)



(b)

Figure 7.3: Confusion matrix on the CORe50 dataset. This shows the validation results using the categories of the first sequence after the training of the third sequence. There are 45 images of each of the ten classes, resulting in 450 validation images. On the top, the predicted- and true labels are shown in correlation to each other. Yellow indicates a high agreement of the values, whereas purple indicates a low one. The classes with the worst result are shown below. There it is shown, that the network often confuses classes two with 17, and four with 19. Classes two and four are of category "plug adapter", whereas 17 and 19 are of category "light bulbs.

the pictures are cropped to a size of 64×64. If one takes that into account, reaching 97 percent accuracy is not surprising and not comparable to an online learning approach with no previous data. More surprising is how close our online learning approach comes to the others. The total ram usage and the run-time are highly depending on the used hardware and how the training data is being prepared.In our case, the data is quadrupled by augmentation and stored in TFRecords. Compared to the winner of Multi-Task-NC, our total run-time and memory consumption is lower.

Table 7.5: Extraction of CLVISION challenge results for scenario NC

Approach	**Accuracy on CORe50**	**Total RAM usage**	**Memory usage for old training examples**	**Time**	**Strategy**
Winner (NC)	97	16081 MByte	2 GByte	40.6 minutes	Rehearsal
...					Rehearsal
Ours	65	2726 MByte	0	4.76 minutes	Rehearsal-free

Table 7.6 shows the results of our approach for the NI scenario, compared to the winner. Again, a comparison is not possible, as the other approach uses a rehearsal strategy. But, it can be seen, that our approach comes close to the accuracy of the winner, although it is not designed for instance classification.

Table 7.6: Extraction of CLVISION challenge results for scenario NI

Approach	**Accuracy on CORe50**	**Strategy**
Winner (NI)	95	Rehearsal
Ours	80	Rehearsal-free

After the challenge, the results of our approach on Multi-Task-NC are improved to an accuracy of over 72 percent on the validation of the last sequence. Although, our approach does not use old training data, our accuracy on Multi-Task-NC and NI are competitive to the incremental learning techniques, which use rehearsal strategies.

7.1.3 CORe50 leader-board

As shown before, a comparison with the results of the challenge is difficult. Therefore, our approach is additionally compared with other incremental learning methods from the official leader-board of the CORe50 dataset [90]. This also contains approaches, which are described in our related work section 3. Except of AR1 and our approach, the results from

table 7.7 are adapted from the official leader-board. It should be noted, that this leader-board does not contain newer methods. Therefore, AR1 is chosen to compare with. It is shown, that our approach outperforms online learning strategies like CWR, LWF and EWC, as well as iCaRL, which uses rehearsal strategies, by a high margin. The AR1 approach reaches similar results.

Table 7.7: Our strategy compared to the official leader-board on CORe50 [90] (23. August 2020). *excerpt from a graph.

Strategy	**Accuracy on CORe50**
Ours	72.02
AR1 [3]	$\approx 70^*$
iCaRL [18]	43.62
CWR [9]	42.32
LWF [22]	27.60
EWC [11]	26.22
Naive [9]	10.75

7.2 Our Dataset

In this section, the results of our approach on the HOWS-CL-25 are presented and compared to our baseline. Similar to the evaluation of CORe50, first the offline learning results of the different sequences are depicted in table 7.8. It is shown, that the network is able to learn each sequence, where sequence four seems to be more difficult than the others. Therefore, it is expected, that the performance of the different approaches decrease in the last sequence.

The results of our baseline approach on the HOWS-CL-25 is shown in table 7.9. The behavior seems to be similar to the results on CORe50, as the respective sequences are learned, but are also instantly forgotten. So, the baseline gets modified, in order to mitigate forgetting. The results for that are shown in table 7.10. Compared to the vanilla baseline, the reduction of forgetting is achieved at the expense of the training performance. That confirms our observations of the baseline performance on the CORe50 dataset.

Table 7.8: Offline results of the different sequences of the HOWS-CL-25

Sequences	Seq. 0	Seq. 1	Seq. 2	Seq. 3	Seq. 4
Results	96.65	95.67	95.06	91.37	76.12

Table 7.9: Results of our baseline approach on HOWS-CL-25

Training \ Validation	Seq. 0	Seq. 1	Seq. 2	Seq. 3	Seq. 4	Avg. Acc.
Sequence 0	95.10	-	-	-	-	95.10
Sequence 1	0.00	96.70	-	-	-	48.35
Sequence 2	0.00	0.00	92.40	-	-	30.80
Sequence 3	0.00	0.00	0.00	89.40	-	22.35
Sequence 4	0.00	0.00	0.00	0.00	70.20	14.04

Table 7.10: Results of our modified baseline approach on HOWS-CL-25

Training \ Validation	Seq. 0	Seq. 1	Seq. 2	Seq. 3	Seq. 4	Avg. Acc.
Sequence 0	85.35	-	-	-	-	85.35
Sequence 1	42.94	55.91	-	-	-	49.43
Sequence 2	8.77	18.11	63.81	-	-	30.23
Sequence 3	2.39	4.19	14.93	45.93	-	16.86
Sequence 4	0.46	0.59	2.72	27.52	35.02	13.26

The results of our approach over the different sequences is shown in table 7.11. These results are similar to the results on CORe50 in table 7.4. The network learns each sequence and especially in the later sequences hardly forgets. Sequence four is learned comparably bad, which is caused by a different degree of difficulty (see table 7.8).

A further analysis of sequence four shows, that the difficulty might be caused by objects, like spoon and knife, which are sometimes hard to distinguish, as they are small on most of the images and also look the same from certain angles (see figure 7.4).

Table 7.11: Results of our approach on HOWS-CL-25

Training \ Validation	Seq. 0	Seq. 1	Seq. 2	Seq. 3	Seq. 4	Avg. Acc.
Sequence 0	94.8	-	-	-	-	94.80
Sequence 1	91.1	90.8	-	-	-	90.95
Sequence 2	86.8	88.0	81.8	-	-	85.53
Sequence 3	80.3	84.6	77.9	75.3	-	79.52
Sequence 4	77.4	76.7	77.5	74.3	39.8	69.14

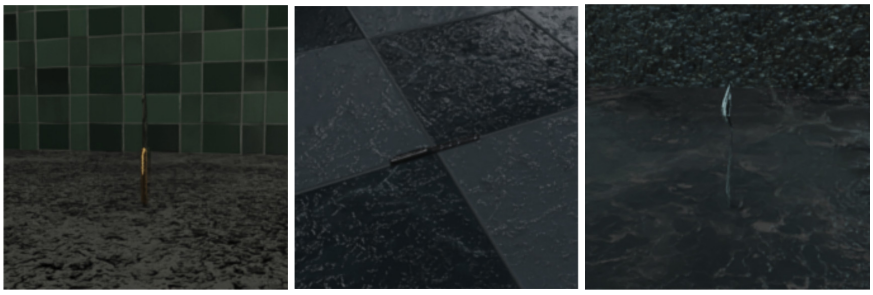


Figure 7.4: Image examples of the fourth sequence of the HOWS-CL-25 dataset. On the left hand side category "knife" is shown, followed by categories "pencil" and "spoon".

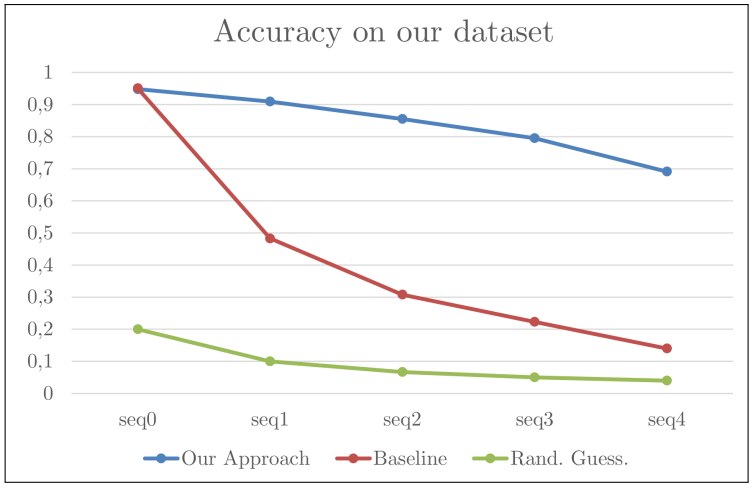


Figure 7.5: Accuracy on HOWS-CL-25: Our approach (blue) is compared with the baseline (red) and a randomly guessing approach (green). Due to our different techniques, the performance of our approach compared to the baseline has been improved with a high margin. The network seems still to forget, especially in the last sequence. In figure 7.6, the reason for this is shown. The network does not forget but, the last sequence is comparably hard to learn.

Compared to the baseline and random guessing, our approach also performs best on the HOWS-CL-25 dataset (see figure 7.5). The network performs comparably well over all sequences, even if it seems that it forgets at the last sequence, which is caused by the higher difficulty on this one. The problem on this is that the dataset has not enough sequences, to better evaluate, if the network is forgetting or not. A suggestion for a future work is to split the dataset into more sequences. For example, its possible to divide it in twelve sequences, with two categories each, except of the first sequence with three categories.

In figure 7.6 a deeper look into the performance of our approach on the different sequences levels is taken. Here again, the reason for the bad performance in the last sequence is noticed, as sequence four is comparably hard to learn for the network. However, it is

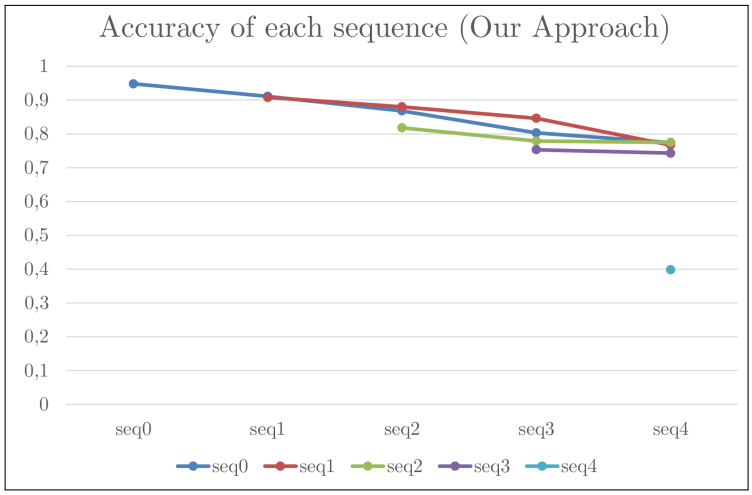


Figure 7.6: The performance of our approach over the sequences of the HOWS-CL-25: It can be observed the same, as in the previous analysis in figure 7.2. The level of forgetting is different from sequence to sequence. Forgetting seems to be more smooth and at sequence two and three, the network even stopped forgetting. The result on sequence four shows the difficulty of its categories.

shown that forgetting is slowed down and in the later sequences it even seems to stop at all.

The results of our approach show, that our method performs well on different datasets. It is also shown, that our approach is able to handle category- and instance classification tasks. Lastly, a comparison of our approach on the different datasets is shown in table 7.12. This comparison highlights the impact of the proposed architectural- and regularization proposals of this thesis. The introduction of the three-part-loss and the reconstruction labels already improve the accuracy on both datasets by a factor of about five in comparison to the baseline approach. By increasing the amount of weights when using several heads, instead of only changing the last layer, also results in a better accuracy. The reason for this might be that important connections within the fully-connected layers are not damaged anymore, as each sequence has its own layers. A further improvement is reached by dividing the output of the last layer by the variance per category. This step reduces the discrepancy in the output distribution and reaches the best result on the HOWS-CL-25 dataset with an accuracy of 69.19 percent. The introduction of the new reconstruction loss in combination with several heads also works fairly well on both datasets. On the CORe50 dataset, it even outperformed the other methods by a accuracy of 72.02 percent.

Table 7.12: Impact of our proposed techniques

Technique	**Accuracy on CORe50**	**Accuracy on HOWS-CL-25**
Baseline	10.02	14.04
Classification loss & reconstruction label	60.51	66.97
Previous + using several heads	63.13	68.95
Previous + variance per category	69.65	**69.19**
Reconstruction loss	62.39	57.41
Previous + using several heads	**72.02**	66.07

8 Ablation studies

In this chapter, the used hyperparameter are presented and the impact of them on our approach is evaluated. Due to our efficient training pipeline, where the features of each image are saved in a TFRecord file, one complete run (training, validation, test over all sequences) only takes around two to three minutes. Therefore, it is possible to run a huge amount of different tests. At the end, 165,352 runs with random hyperparameter are performed. In the appendix of this work, all 17 hyperparamters with their tested values are listed (see 11.3).

In order to evaluate various methods and hyperparameter combinations, different techniques are used. With our evaluation tools it is possible, to put those combinations in dependency to find the best ones. For instance, a plot of a combination of different learning rates and loss functions and their resulting accuracy over the last sequence is shown in figure 6.6. Each point represents one run over all sequences. There, for example it can be seen, that both loss functions perform worse on a learning rate smaller than 3.16e-05.

As there are 17 hyperparameter with several different values, testing all combinations is beyond the scope of this thesis. Therefore, the value ranges of these parameter are limited to a spectrum, which resulted in the best accuracies. These optimized ranges, as well as the most appropriate sampling function, is also listed in table 11.3. It is also found, that some hyperparameter have a much bigger influence on the accuracy than others. Some important parameters are for instance the learning rate, loss function and activation function. Less relevant are the lambda value and the learning rate scale.

In the following, further hyperparameters are analyzed, which have a strong influence on the results.

8.1 Batch normalization

Batch normalization is known to speed up the training process, as the outputs are normal distributed around zero. In this thesis it is tried to place the batch norm before and after the activation function and at different locations of the network. The results on our approach show, that there is no difference between placing the batch norm before or after the activation function.

Therefore, now the question is where the batch norm should be used. In table 8.1, the different possible placements are tested. Enumeration "only in core", means that the batch normalization is used in each fully connected layer, but not on the last one. In figure 8.1 these are FC1 - FC3. On the contrary, "only in last layer" means that the normalization is only applied on FC4.

Table 8.1: Results on CORe50 and HOWS-CL-25 for different placed batch norms

Strategy	Accuracy on CORe50	Accuracy on HOWS-CL-25
No batch norm	**72.02**	**69.19**
Only in core	57.01	52.06
Only in last layer	63.70	50.04
Everywhere	67.11	56.57

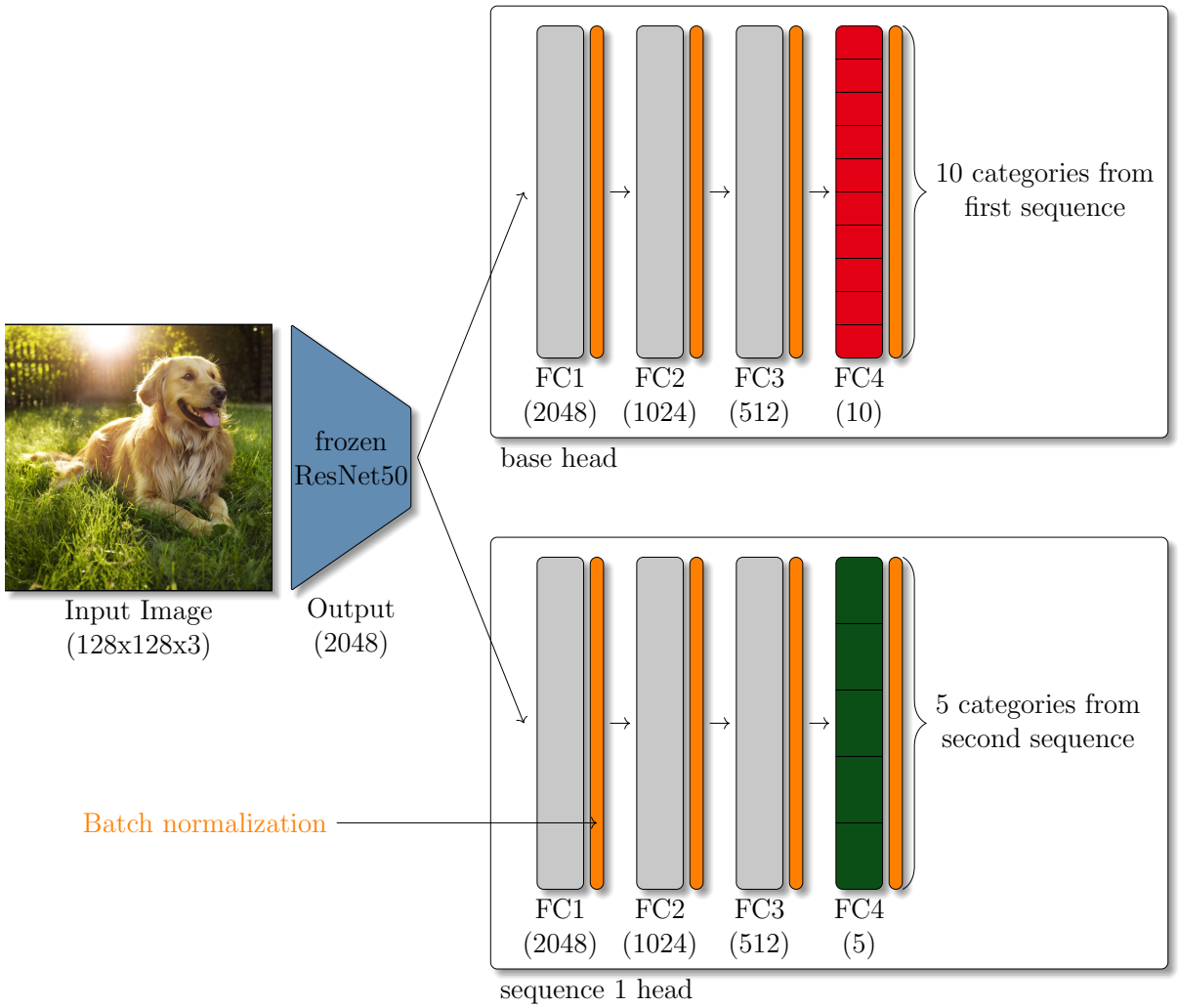


Figure 8.1: Network using batch normalization of type "everywhere", shown in orange.

It is important to assure, that the batch normalization layers are not initialized again in each sequence as it contains learned variables. A lose of this information results in a worsened performance. Additionally, batch normalization does not always lead to better results. It depends on the chosen loss and activation function. Batch Normalization works best in combination with a linear activation function (identity) and worse with RELU in our experiments.

8.2 Activation Function

In our approach, different activation functions in the head of the neural network are tested (see table 8.2). There, SIREN resulted in the best accuracy for the CORe50 dataset with a value of 72.02 percent and RELU in the best accuracy for the HOWS-CL-25 dataset with a value of 69.16 percent. Furthermore, the activation functions Identity, RELU and SIREN performed better on both datasets than the other evaluated activation functions.

Table 8.2: Results on the CORe50 and the HOWS-CL-25 for different activation functions

Strategy	**Accuracy on CORe50**	**Accuracy on HOWS-CL-25**
CRELU	47.75	53.26
ELU	48.94	53.66
Identity	68.64	56.57
RELU	61.87	**69.19**
SELU	51.60	49.01
SIREN	**72.02**	67.66

Some further findings are listed below:

- Identity function: Worked well in our experiments. As this is linear, it can be assumed, that the feature values, resulting from the feature extraction network are linearly separable, which might be the case, as the backbone already uses RELU to solve non-linear approximations. Furthermore, the results in our experiments, when using the identity function, are improved by applying a batch normalization.
- RELU [26]: a non-linear activation function, which leads to one of the best results in our work, except if it is combined with batch normalization.

- SIREN [32]: This activation function gives the best results, but as written in their paper, it highly depends on the initialization values, which can be controlled by the first and hidden omega values (see description in section 4.1).

8.3 Initialization

The initialization is found to be crucial in continuous learning. As well as shown in literature [3] and in our experiments, the best results are achieved, when the initialization of each sequence is similar to each other. This is also confirmed in table 8.3, where different initialization strategies are compared. When the initialization of each sequence is done based on the first sequence, the results are better, than if each sequence is initialized differently (e.g. from the previous sequence). But random is still the best strategy.

Table 8.3: Results on the CORe50 and the HOWS-CL-25 dataset for different initialization strategies

Strategy	**Accuracy on CORe50**	**Accuracy on HOWS-CL-25**
Random	**72.02**	**69.19**
From first sequence	70.02	52.58
From previous sequence	56.39	50.38

8.4 Feature extraction networks

In table 8.4 it can be observed, that all tested feature extraction networks work good on CORe50 and the HOWS-CL-25 dataset. As CORe50 is more focused on instance-classification and the HOWS-CL-25 on category classification, it is assumed that newer CNNs are improved to better distinguish between different object categories than instances. It can also be seen that ResNet50 works better than the second version ResNet50V2. On the CORe50 dataset ResNet50 outperforms each other network. These results have been partly confirmed by other papers like [3], where they also found that ResNet50 outperformed GoogLeNet on the CORe50 dataset.

8.5 Freezing or fine tuning

In the field of transfer learning there are several papers dealing with the problem, which part of the network should be learned and which one should stay freezed. Using CNNs, the first layers of the network are most likely creating general feature detectors like Gabor filters [91] or color blobs. These features are not dataset related and therefore should stay

Table 8.4: Results of different feature extraction networks

Feature-Extractor	Accuracy on CORe50	Accuracy on HOWS-CL-25
ResNet50	**72.02**	59.94
ResNet50V2	64.04	59.51
Inceptionv3	61.87	67.66
InceptionResNetV2	51.55	**69.19**

the same. The features computed by the last layer of the network depend greatly on the chosen task or dataset. Usually, only the last part of a network should be fine tuned [51].

For our approach, different freezing strategies for the feature extraction network are tested. The results show, that when more parts of the networks are used for fine tuning, the more the accuracy decreases. Only training the last part of the network leads to good results and the best results are achieved by freezing the whole feature extraction network (see table 8.5). In this table, the different levels of ResNet blocks are called "stage".

Table 8.5: Results of different freezing strategies on CORe50

Strategy	Accuracy on CORe50
Freeze_ResNet_Stage 1-3	27.06
Freeze_ResNet_Stage 1-4	44.18
Freeze_ResNet_Complete	**72.02**

The reason of this might be that the fine tuning would work pretty good on a single sequence, but as the training process is applied in a continual manner, the sensible connections of the network are torn up every time, when the next sequence arrives. Therefore, the complete feature extraction network is frozen, especially as these are already generating good feature representations for each category. Another benefit is the improvement of the training time as the network has less parameters to adapt on.

8.6 Number of layers

Different amounts of fully-connected layers are evaluated in order to figure out, if more layer lead to a better accuracy. In table 8.6 the resulting accuracy on both datasets is depicted, recording to the number of used layers. It is shown, that the best results for both datasets are achieved with two fully-connected layers. Furthermore, using just one layer also results in comparable outcomes, which leads to the assumption, that the features are mostly linearly separable. According to those results, more layers do not lead to a better

Table 8.6: Results of different amounts of fully-connected layers

Layer amount	Accuracy on CORe50	Accuracy on HOWS-CL-25
1	57.01	53.26
2	**72.02**	**69.19**
3	63.70	52.59
4	59.87	52.78
5	59.28	52.29
6	62.24	53.82
7	53.68	49.57

performance on our approach.

8.7 K-nearest neighbor

It is assumed, that images, which contain the same object category, should result in similar feature outputs, if applied on a feature extraction network. This theory can be evaluated by applying a simple K-nearest-neighbor function (KNN) to the output-features of the CNN. This offline learning function uses clustering of the feature points according to their distance to classify their category. An application on CORe50 dataset results in 60 percent accuracy in an offline mode. Keeping in mind that CORe50 is optimized for instance recognition, this result proves our assumption. It also confirms that the features, resulting from the completely frozen feature extraction network are good enough to distinguish between the different categories. This finding is also the base for other ideas in the future works section 9.5. The reason to use a neural network instead of K-nearest neighbor is, that it is about 30 times faster and, instead of KNN, able to be applied for online learning.

In this chapter the different hyperparameter and their influence on our approach are evaluated. Additionally, tables for the best parameter combination for each datasets are provided in our appendix (see table 11.1 for the parameter of the best run of the CORe50 dataset, and table 11.2 for the best run of the HOWS-CL-25 dataset). Next, methods are shown to further improve our approach in a future work.

9 Future steps

Our approach forms the base for future developments on continual learning of mobile robots. There are already a lot of ideas, on how to continue this work, but as it is beyond the scope of this thesis, those ideas are outlined in this chapter.

9.1 Rehearsal Strategy

Adding a rehearsal strategy, like memory replay, is proven to improve the accuracy of a network by several papers [24, 60, 18]. But in this case, our approach is not longer classified as online learning, what leads to several disadvantages:

- Having a memory of each seen category leads to a continual increasing of the amount of memory usage of the system.
- The training and validation process is becoming longer and more computational intensive compared to online learning.

To improve that, our suggestion is, instead of saving pictures, to save the feature space of each category. As the feature spaces of different instances of a certain category are quite similar, what is proven by using KNN (see section 8.7), it is also enough to save a few feature space samples for each category. For this purpose, a ringpuffer can be used to prevent too high memory consumption. Denninger and Triebel [24] solved this, by using feature containers. Something similar is also proposed by Pellegrini et al. [92].

In figure 9.1, a first draft of that idea is shown. The storage needed for these feature vectors is comparably small as $2048 \cdot 4$ byte (float) $= 8192$ byte are needed per image. Thus the whole CORe50 dataset needs less than $8192 \cdot 164866 \approx 1.35$ Gbyte, as the validation and test images are not subtracted in this calculation.

9.2 Dreaming

As a robot is usually deactivated for some time during the day, it would be reasonable to use this standby-time to replay what the robot learned during its day, like humans do. That means, a robot has to collect experiences e.g., example images or feature vectors of them, to replay those experiences at night. This is also known as "dreaming" in robotics [93]. The combination of offline and online learning should massively mitigate the forgetting problem. It further depends on the learning strategy, if the robot should always keep a few examples for each category it knows, or only the one it learns on the respective day, which would also include deleting these examples after dreaming.

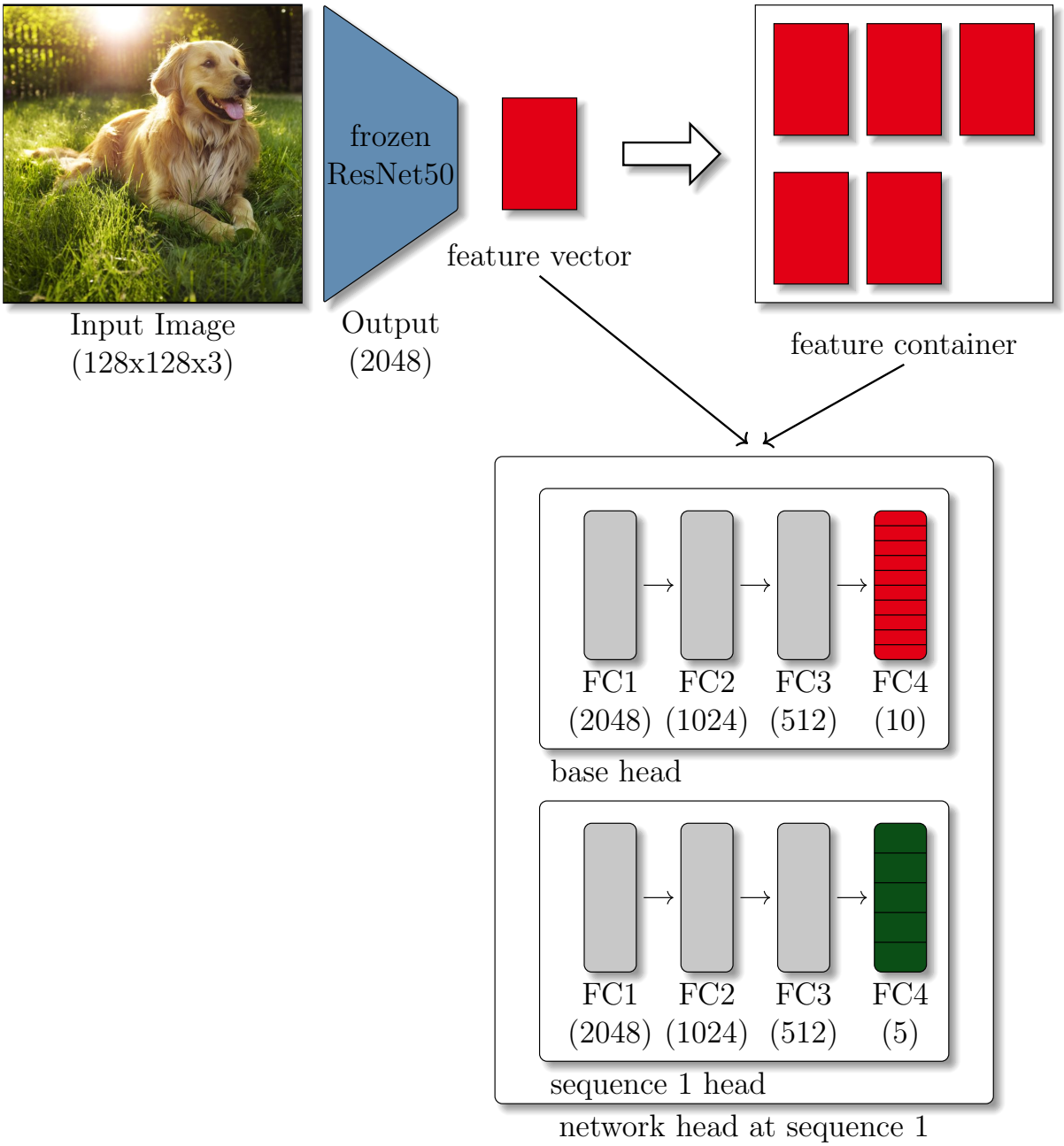


Figure 9.1: In our incremental learning approach, the feature vector of each input image is saved in a feature container and both, the current feature vector and the present feature container are used in a memory replay manner, for training. The network shown below is currently at the second sequence, including ten categories in the first, and five categories in the second sequence head. All fully connected layers of the present network are called "network head". Therefore, the network has at least one example of each category it knows so far, which mitigate forgetting compared to our online learning approach.

9.3 Uncertainty Estimation

Considering that CNNs tend to be overconfident [94], uncertainty estimation becomes important. Especially in the field of continual learning on mobile robots, the safety of the network decision should be guaranteed, as assisting a human with the right materials is one of the core tasks for service robots. There are several ways to use uncertainty estimation in our approach. For instance, via dropout or adding noise on the features [95]. A first approach is shown in figure 9.2, where the network is fivefolded, and random noise vectors are added to the feature input of each head. Then the heads have to vote, which category they observe on the input image. Usually, uncertainty estimation needs several runs of the network, but our approach is able to implement this method in a parallel way, thus it is competitively fast and only needs one run to predict the category and measures the uncertainty.

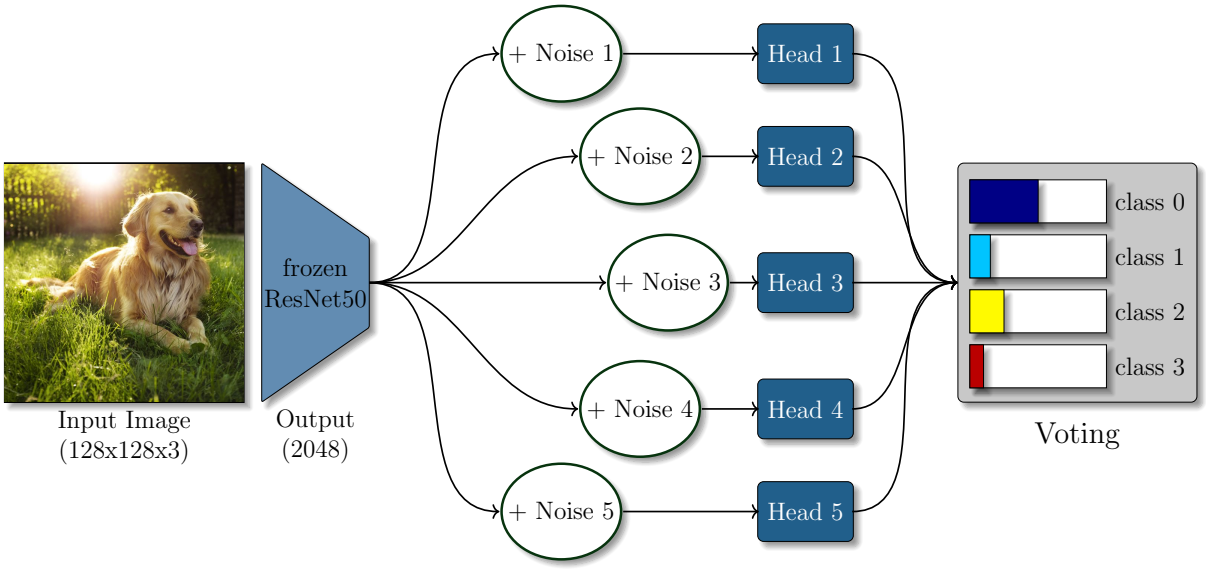


Figure 9.2: Uncertainty estimation approach: One possible uncertainty estimation network architecture, where all fully connected layers are combined as one network head. There the network is fivefolded, where each head has to vote for a category, but each of them gets a different input value, as there are five dissimilar noises added to the feature output of the CNN.

9.4 Active Learning

Active Learning [58] is a special case of machine learning, where the aim is to detect unknown tasks, to let them be labeled by a human or another source afterwards. After integration of uncertainty estimation, the next logical step would be the usage of active learning, as our network is now capable of knowing, which categories it does not know. This is important in continual learning on mobile robots, as the robot should be able to recognize new, unknown object categories, when it observes them. Exactly then, the robot is able to trigger further methods, to actively learn these new categories, e.g. by asking a human supervisor.

9.5 Inverted GLOs

Using a novel method, which is called inverted GLOs, it could be theoretical possible to further decrease forgetting in incremental learning (see figure 9.4). This idea is inspired by Generative Latent Optimization (GLO) [15], which is a method to train deep convolution generators. It works similar to an autoencoder but without the encoder part, or as they described it in their paper "GLO can be vied as [..] a 'discrimiator-less' GAN". Generative Adversarial Networks (GANs) [13] are an unsupervised learning method, which use a generator and a discriminator to generate, for example, realistic natural images.

Our approach uses the idea of latent optimization. But, instead of generating a latent space from an image, the latent space is calculated based on the logits from our feature

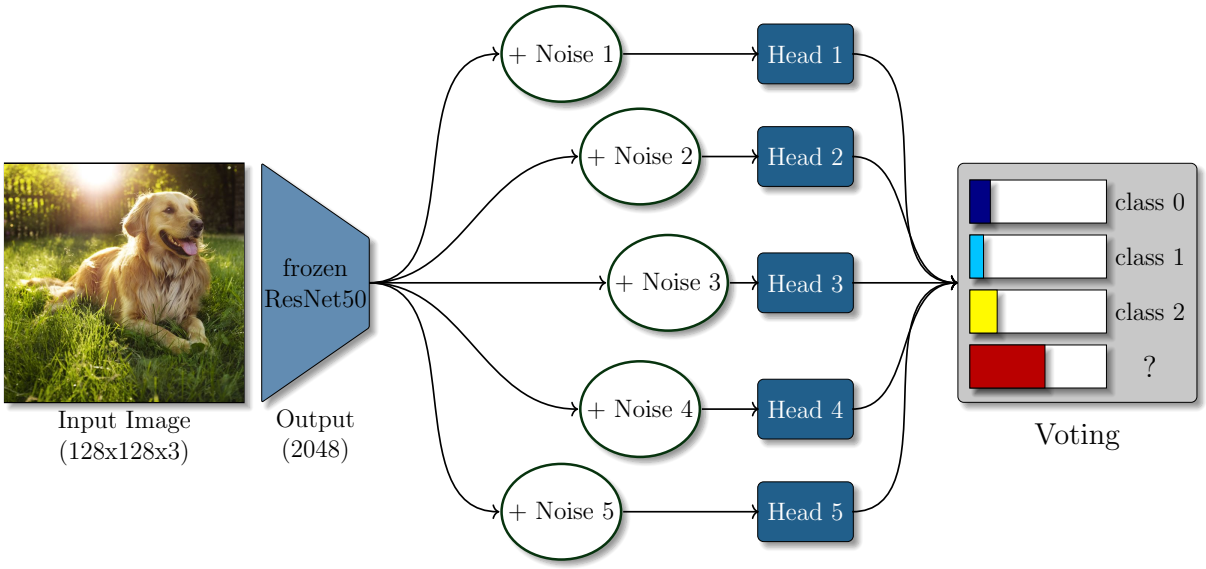


Figure 9.3: Active learning approach: Our architecture idea on uncertainty estimation in figure 9.2 can be extended to an active learning architecture, where the network is also able to detect categories it does not know so far to trigger a further handling on them.

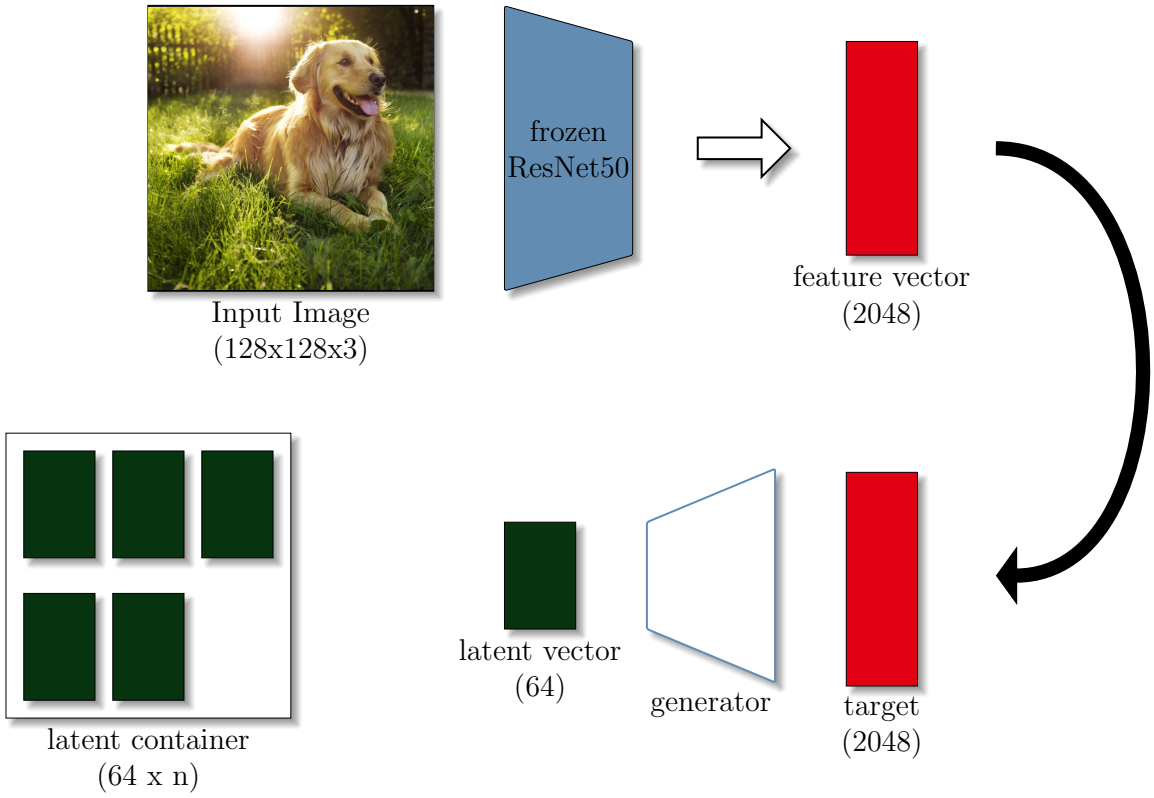


Figure 9.4: Continually learning procedure of inverted GLOs. At the top the feature vector of a given image is calculated. After that this vector is becoming the target to calculate the latent vector. Therefore, a latent vector is initialized for each training example. Now, this latent vector gets optimized and then, the generator is trained to reconstruct the feature vector by the given latent vector. In the end, these latent vectors also can be saved in a latent container.

extractor. The space in which this output vector is defined is called feature space in this work. This inversion gives it its name: "inverted GLO".

The idea is, to create a latent value for each training example. This value should be similar within the pictures of a category. Thus, it would be possible to save one representation per category and therefore continual learning could be improved to not forget anymore.

This procedure is described below:

- First, the feature space of a given training image is produced by a feature extraction network, like ResNet50.
- Secondly, the calculated feature vector is being freezed and becomes a target in the following process.
- After that, the latent vector is initialized. This can be done randomly, but it is also possible to initialize it according to the category ID. Therefore, it would make sense

doing a binary initialization. A fix latent space of 64 dimensions would result in $2^{64} = 1.84e + 19$ (18 trillion) possible categories.

- At the first run, the generator is also initialized randomly. It is responsible to reconstruct the feature value from a given latent vector. The generator is initialized only once.
- Now, the first step is to optimize the latent vector with a gradient decent approach through the generator, based on the feature vector.
- After that, the generator is being trained with this latent value as input and feature vector as target.
- When the next training images arrive, this procedure is done again, except of initializing the generator. This part stays the same, as it has to learn to regenerate the feature values from the given latent values.
- After some iterations, it might be reasonable to start saving the latent values in a latent container (similar to the feature container in [24]) or as only saving one representation per category, by using a mean over the latent values of a certain category.
- In the validation process, the network searches for a latent vector with comes close to the one, given from the training image.

As shown by Sundermeyer et al. [96], an evaluation step over the latent vectors in the latent container can be done in real time. Thus, the prediction process, can be done on the fly, with comparably less memory and computational effort.

There is a potential in this approach, as it could not only improve continual learning but also lead to a lot of other different approaches as this brakes down a complex problem to an easier one, on which it is possible to use different machine learning methods on. For example, the implementation of an uncertainty estimation approach, by using Gaussian Processes (GPs) [16] on the latent container. Therefore, also active learning is possible, where the network checks if a given latent vector is close enough to any other in the latent container, and much more.

10 Conclusion

This thesis shows, that the difficult task of online learning can be efficiently solved by a combination of several architectural and regularization techniques, under the usage of convolutional neural networks.

In literature, pure online learning is rare and most of the approaches show a comparable weak performance to our approach. This impression was confirmed by the first workshop for continual learning at CVPR 2020, where almost all methods used a dedicated memory to save previous training examples. Even if that workshop was exclusively for continual learning, they did not distinguish between rehearsal and rehearsal-free methods, which is a big difference in our view. But, especially in the field of mobile robots, online learning has a lot of advantages, compared to incremental learning methods with rehearsal strategies, as it is faster, less memory intensive and does not rely on previous data. That's why this work focuses especially on the task of online learning.

In this work, different incremental learning methods from literature are shown and their differences to our approach of online learning are discussed. Furthermore, some novel methods are proposed in order to mitigate catastrophic forgetting, one of the biggest problems in continual learning. First of all, two architectural strategies are presented to enable the network to learn in a continual manner, as it is dynamically adapting according to the amount of categories it knows. This includes a straight forward proposal, where the last layer of the network is adapted, and an improved one, where instead a new head with several fully-connected layers is created for each new sequence.

With those techniques, the network is able to learn in a continually manner. In order to prevent forgetting, a combination of reconstruction labels with a three part loss function is proposed. Using this method, the problem of a discrepancy in the output distribution occurs, which again leads to forgetting. To solve this problem, some regularization strategies are presented including a novel normalization technique, where the outputs are divided by the variance per category. On top of that, other regularization strategies are shown to support the loss function and mitigate forgetting, by simultaneously enabling the network to learn new categories on the fly. This also includes a novel loss function, called reconstruction loss, where the classification is replaced by a regression. Our approach is compared with other online learning techniques on the CORe50 dataset, where most of them are outperformed, by reaching an accuracy of 72.02 percent in the last sequence. This is an improvement of about factor five compared to our baseline approach. This result is achieved in the NC scenario. Furthermore, our approach performed comparably well in the NI scenario, with an accuracy of 80 percent on the last sequence. Additionally, in an extensive experiment setting with $165,352$ test, it is shown how different parameter

influence the performance of our approach.

Compared to the classification of a static dataset, a well researched field, the training procedure of incremental learning is quite different, since the dataset is split into several chronological sequences. Therefore, the amount of datasets in this area is limited, most of them are non-public and therefore not comparable. Even though, a good dataset (CORe50) is found to compare with, it did not meet all of our requirements for mobile robots. Hence, also a novel dataset for continual learning is presented, which is especially suited for object recognition in our mobile robot environment, called HOWS-CL-25. It is created with BlenderProc and consists of 150,795 synthetic images of 25 different household object categories in a randomly changing environment. This dataset meets all of our requirements and additionally contains more categories than CORe50.

An evaluation on several datasets shows, that our approach is able to continually learn new object categories from images, with less forgetting, which can be used for robotic applications. Additionally, some promising future steps of this work are shown, to further improve online learning, the robustness of its decisions and the ability to detect new categories, to get closer to a future, where mobile robots can be deployed in everyday homes, to help and serve our caregivers.

References

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

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[3] Davide Maltoni and Vincenzo Lomonaco. Continuous learning in single-incremental-task scenarios. *CoRR*, abs/1806.08568, 2018.

[4] Maximilian Denninger, Martin Sundermeyer, Dominik Winkelbauer, Youssef Zidan, Dmitry Olefir, Mohamad Elbadrawy, Ahsan Lodhi, and Harinandan Katam. Blenderproc. *arXiv preprint arXiv:1911.01911*, 2019.

[5] Maximilian Denninger, Martin Sundermeyer, Dominik Winkelbauer, Dmitry Olefir, Tomáš Hodaň, Youssef Zidan, Mohamad Elbadrawy, Markus Knauer, Harinandan Katam, and Ahsan Lodhi. Blenderproc: Reducing the reality gap with photorealisitc rendering. *arXiv preprint arXiv:1911.01911*, 2020.

[6] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[7] Pau Rodriguez, German I. Parisi, David Vazquez, Vincenzo Lomonaco, Nikhil Churamani, Zhiyuan Chen, and Marc Pickett. Clvision challenge on "continual learning for computer vision". https://sites.google.com/view/clvision2020/challenge.

[8] Sameer A Nene, Shree K Nayar, Hiroshi Murase, et al. Columbia object image library (coil-100). 1996.

[9] Vincenzo Lomonaco and Davide Maltoni. Core50: a new dataset and benchmark for continuous object recognition. *arXiv preprint arXiv:1705.03550*, 2017.

[10] Wenling Shang, Kihyuk Sohn, Diogo Almeida, and Honglak Lee. Understanding and improving convolutional neural networks via concatenated rectified linear units. In *international conference on machine learning*, pages 2217–2225, 2016.

[11] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[12] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[13] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[14] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *Advances in neural information processing systems*, pages 6467–6476, 2017.

[15] Piotr Bojanowski, Armand Joulin, David Lopez-Paz, and Arthur Szlam. Optimizing the latent space of generative networks. *arXiv preprint arXiv:1707.05776*, 2017.

[16] Radford M Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

[17] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[18] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H. Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[19] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[20] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[21] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*, 2017.

[22] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

[23] Yann LeCun. The mnist database of handwritten digits. *http://yann.lecun.com/exdb/mnist/*, 1998.

[24] Maximilian Denninger and Rudolph Triebel. Persistent anytime learning of objects from unseen classes. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4075–4082. IEEE, 2018.

[25] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[26] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[28] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[29] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Advances in neural information processing systems*, pages 971–980, 2017.

[30] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

[31] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. *Proceedings of machine learning research*, 70:3987, 2017.

[32] Vincent Sitzmann, Julien NP Martel, Alexander W Bergman, David B Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *arXiv preprint arXiv:2006.09661*, 2020.

[33] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[34] Sebastian Thrun. Explanation-based neural network learning: A lifelong learning approach, 1996.

[35] Maximilian Denninger and Rudolph Triebel. 3d scene reconstruction from a single viewport. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020.

[36] Eric Brachmann and Carsten Rother. Learning less is more–6d camera localization via 3d surface regression–supplementary materials–.

[37] Ludwig Kürzinger, Dominik Winkelbauer, Lujun Li, Tobias Watzel, and Gerhard Rigoll. Ctc-segmentation of large corpora for german end-to-end speech recognition. *arXiv preprint arXiv:2007.09127*, 2020.

[38] Robert M French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

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

[40] Vincenzo Lomonaco. *Continual Learning with Deep Architectures*. PhD thesis, alma, 2019.

[41] susannp4. Image of a cat. https://pixabay.com/de/photos/katze-jungtier-neugierig-wildkatze-2083492/.

[42] Jonathan Steele. https://500px.com/photo/71612981/Golden-Retriever-by-Jonathan-Steele/.

[43] Alexander Gepperth and Barbara Hammer. Incremental learning algorithms and applications. 2016.

[44] Timothée Lesort, Vincenzo Lomonaco, Andrei Stoian, Davide Maltoni, David Filliat, and Natalia Díaz-Rodríguez. Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges. *Information Fusion*, 58:52–68, 2020.

[45] Zhiyuan Chen and Bing Liu. Lifelong machine learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 12(3):1–207, 2018.

[46] Sebastian Thrun and Tom M Mitchell. Lifelong robot learning. *Robotics and autonomous systems*, 15(1-2):25–46, 1995.

[47] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R Hruschka, and Tom M Mitchell. Toward an architecture for never-ending language learning. In *Twenty-Fourth AAAI conference on artificial intelligence*, 2010.

[48] Tom Mitchell, William Cohen, Estevam Hruschka, Partha Talukdar, Bishan Yang, Justin Betteridge, Andrew Carlson, Bhavana Dalvi, Matt Gardner, Bryan Kisiel, et al. Never-ending learning. *Communications of the ACM*, 61(5):103–115, 2018.

[49] Lorien Y Pratt. Discriminability-based transfer between neural networks. In *Advances in neural information processing systems*, pages 204–211, 1993.

[50] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[51] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[52] Hossein Azizpour, Ali Sharif Razavian, Josephine Sullivan, Atsuto Maki, and Stefan Carlsson. From generic to specific deep representations for visual recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 36–45, 2015.

[53] Vishal M Patel, Raghuraman Gopalan, Ruonan Li, and Rama Chellappa. Visual domain adaptation: A survey of recent advances. *IEEE signal processing magazine*, 32(3):53–69, 2015.

[54] Pavel Brazdil, Christophe Giraud Carrier, Carlos Soares, and Ricardo Vilalta. *Metalearning: Applications to data mining*. Springer Science & Business Media, 2008.

[55] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017.

[56] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Advances in neural information processing systems*, pages 4077–4087, 2017.

[57] Xiaocong Du, Zheng Li, Jae-sun Seo, Frank Liu, and Yu Cao. Noise-based selection of robust inherited model for accurate continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 244–245, 2020.

[58] Dennis Mund, Rudolph Triebel, and Daniel Cremers. Active online confidence boosting for efficient object classification. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1367–1373. IEEE, 2015.

[59] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[60] Tyler L Hayes, Nathan D Cahill, and Christopher Kanan. Memory efficient experience replay for streaming learning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 9769–9776. IEEE, 2019.

[61] Ronald Kemker and Christopher Kanan. Fearnet: Brain-inspired model for incremental learning. *arXiv preprint arXiv:1711.10563*, 2017.

[62] German I Parisi, Jun Tani, Cornelius Weber, and Stefan Wermter. Lifelong learning of spatiotemporal representations with dual-memory recurrent self-organization. *Frontiers in neurorobotics*, 12:78, 2018.

[63] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[64] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[65] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[66] Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982.

[67] Balázs Csanád Csáji et al. Approximation with artificial neural networks. *Faculty of Sciences, Etvs Lornd University, Hungary*, 24(48):7, 2001.

[68] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4681–4690, 2017.

[69] Eldar Insafutdinov, Leonid Pishchulin, Bjoern Andres, Mykhaylo Andriluka, and Bernt Schiele. Deepercut: A deeper, stronger, and faster multi-person pose estimation model. In *European Conference on Computer Vision*, pages 34–50. Springer, 2016.

[70] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In *Advances in Neural Information Processing Systems*, pages 6389–6399, 2018.

[71] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Advanced guide to inception v3 on cloud tpu. https://cloud.google.com/tpu/docs/inception-v3-advanced.

[72] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[73] Annalisa Franco, Dario Maio, and Davide Maltoni. The big brother database: evaluating face recognition in smart home environments. In *International Conference on Biometrics*, pages 142–150. Springer, 2009.

[74] Vincenzo Lomonaco and Davide Maltoni. Comparing incremental learning strategies for convolutional neural networks. In *IAPR Workshop on Artificial Neural Networks in Pattern Recognition*, pages 175–184. Springer, 2016.

[75] Li Fei-Fei, Rob Fergus, and Pietro Perona. One-shot learning of object categories. *IEEE transactions on pattern analysis and machine intelligence*, 28(4):594–611, 2006.

[76] Prithviraj Dhar, Rajat Vikram Singh, Kuan-Chuan Peng, Ziyan Wu, and Rama Chellappa. Learning without memorizing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5138–5146, 2019.

[77] Ghouthi Boukli Hacene, Vincent Gripon, Nicolas Farrugia, Matthieu Arzel, and Michel Jezequel. Budget restricted incremental learning with pre-trained convolutional neural networks and binary associative memories. *Journal of Signal Processing Systems*, 91(9):1063–1073, 2019.

[78] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The caltech-ucsd birds-200-2011 dataset. 2011.

[79] Sang-Woo Lee, Jin-Hwa Kim, Jaehyun Jun, Jung-Woo Ha, and Byoung-Tak Zhang. Overcoming catastrophic forgetting by incremental moment matching. In *Advances in neural information processing systems*, pages 4652–4662, 2017.

[80] Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer. *arXiv preprint arXiv:1511.05641*, 2015.

[81] Tianjun Xiao, Jiaxing Zhang, Kuiyuan Yang, Yuxin Peng, and Zheng Zhang. Error-driven incremental learning in deep convolutional neural network for large-scale image classification. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 177–186, 2014.

[82] Xiaoliang Dai, Hongxu Yin, and Niraj K Jha. Incremental learning using a grow-and-prune paradigm with efficient neural networks. *arXiv preprint arXiv:1905.10952*, 2019.

[83] Giulia Pasquale, Carlo Ciliberto, Francesca Odone, Lorenzo Rosasco, and Lorenzo Natale. Real-world object recognition with off-the-shelf deep conv nets: how many objects can icub learn? *arXiv preprint arXiv:1504.03154*, 2015.

[84] Pau Rodriguez. Codalab for clvision workshop at cvpr2020. https://competitions.codalab.org/competitions/23317.

[85] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

[86] Tomáš Hodaň, Pavel Haluza, Štepán Obdržálek, Jiri Matas, Manolis Lourakis, and Xenophon Zabulis. T-less: An rgb-d dataset for 6d pose estimation of texture-less objects. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 880–888. IEEE, 2017.

[87] Julian Straub, Thomas Whelan, Lingni Ma, Yufan Chen, Erik Wijmans, Simon Green, Jakob J Engel, Raul Mur-Artal, Carl Ren, Shobhit Verma, et al. The replica dataset: A digital replica of indoor spaces. *arXiv preprint arXiv:1906.05797*, 2019.

[88] Shuran Song, Fisher Yu, Andy Zeng, Angel X Chang, Manolis Savva, and Thomas Funkhouser. Semantic scene completion from a single depth image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1746–1754, 2017.

[89] Tomáš Hodaň, Vibhav Vineet, Ran Gal, Emanuel Shalev, Jon Hanzelka, Treb Connell, Pedro Urbina, Sudipta N Sinha, and Brian Guenter. Photorealistic image synthesis for object instance detection. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 66–70. IEEE, 2019.

[90] Vincenco Lomonaco. Official core50 leaderboard. https://vlomonaco.github.io/core50/leaderboard.

[91] Dennis Gabor. Theory of communication. part 1: The analysis of information. *Journal of the Institution of Electrical Engineers-Part III: Radio and Communication Engineering*, 93(26):429–441, 1946.

[92] Lorenzo Pellegrini, Gabrile Graffieti, Vincenzo Lomonaco, and Davide Maltoni. Latent replay for real-time continual learning. *arXiv preprint arXiv:1912.01100*, 2019.

[93] Sebastian G Brunner, Peter Lehner, Martin J Schuster, Sebastian Riedel, Rico Belder, Daniel Leidner, Armin Wedler, Michael Beetz, and Freek Stulp. Design, execution, and postmortem analysis of prolonged autonomous robot operations. *IEEE Robotics and Automation Letters*, 3(2):1056–1063, 2018.

[94] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in neural information processing systems*, pages 6402–6413, 2017.

[95] Janis Postels, Francesco Ferroni, Huseyin Coskun, Nassir Navab, and Federico Tombari. Sampling-free epistemic uncertainty estimation using approximated variance propagation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2931–2940, 2019.

[96] Martin Sundermeyer, Zoltan-Csaba Marton, Maximilian Durner, Manuel Brucker, and Rudolph Triebel. Implicit 3d orientation learning for 6d object detection from rgb images. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 699–715, 2018.

11 Appendix

11.1 Appendix A: Parameter of the best result on the CORe50 dataset

Table 11.1: Parameter of the best CORe50 result of our approach

Hyperparameter	**Value**
Activation function	SIREN
Activation before batchnorm	False
Batch Normalization Type	no batchnorm
Different Heads	True
Feature extraction network	ResNet50
Freezing strategy	freeze all
Initializing strategy	random
Lambda	0.04
Layer number	2
Learning rate	$9.35e-4$
Learning rate scale	0.856
Loss function	reconstruction loss
SIREN first omega	2.13
SIREN hidden omega	0.017
Training steps base	5
Train steps sequence	6
Use variance per category	False

11.2 Appendix B: Parameter of the best result on the HOWS-CL-25 dataset

Table 11.2: Parameter of the best HOWS-CL-25 result of our approach

Hyperparameter	**Value**
Activation function	RELU
Activation before batchnorm	True
Batch Normalization Type	no batchnorm
Different Heads	True
Feature extraction network	InceptionResNetV2
Freezing strategy	freeze all
Initializing strategy	random
Lambda	0.0009
Layer number	2
Learning rate	$2.19e-4$
Learning rate scale	0.25
Loss function	classification loss
Training steps base	3
Train steps sequence	4
Use variance per category	True

11.3 Appendix C: Hyperparameter overview

Table 11.3: Hyperparameter part 1

Hyperparameter	**Value range**	**Random sampling strategy**	**Proposed value range**	**Explanation**
Activation function	[Identity, RELU, SIREN, ELU, SELU, CRELU]	random choice	[Identity, RELU, SIREN]	Choice of activation function
Activation before batchnorm	[True,False]	boolean	[True,False]	If True, the activation function is applied before the Batch Normalization, if False vice versa.
Batch Normalization Type	[no batchnorm, batchnorm only in core, batchnorm only in last layer, batchnorm everywhere]	random choice	[no batchnorm, batchnorm everywhere]	States whether a Batch Normalization is used and if yes, where it is used.
Different Heads	[True, False]	boolean	[True]	If true, instead of only adapting the last layer, each sequence gets its own head of fully-connected layers.
Feature extraction network	[Resnet50, Resnet50V2, InceptionV3, InceptionResnetV2]	random choice	[Resnet50, InceptionResnetV2]	The used CNN in the backend.
Freezing strategy	[train all, freeze first stage, freeze second stage, ..., freeze all]	random choice	[freeze all]	It indicates the amount of stages, which are not trained.
Initializing strategy	[random, from first sequence, from previous sequence]	random choice	[random, from first sequence]	The way the fully connected layers are initialized.
Lambda	$[0, \ldots, 1]$	logarithmic	$[1e{-}04, \ldots, 0.11]$	Lambda value of the loss function to balance between not forgetting and learning new categories
Layer number	$[0, \ldots, 6]$	integer value	$[1, \ldots, 6]$	Amount of fully connected layers (without the classification layer).

Table 11.4: Hyperparameter part 2

Hyper-parameter	**Value range**	**Random sampling strategy**	**Proposed value range**	**Explanation**
Learning rate	$[1e{-}8, \ldots, 1e{-}2]$	logarithmic	Depends highly on other parameter	The learning rate of the ADAM optimizer.
Learning rate scale	$[0, \ldots, 1]$	uniform	$[0.2, \ldots, 0.9]$	This value indicates the adaption of the learning rate over time. If set to one it does not change anything, the smaller this number, the stronger the learning rate gets decreased over time.
Loss function	[classification loss, reconstruction loss]	random choice	[Classification loss, Reconstruction loss]	Either our proposed three-part classification loss or tree-part reconstruction loss.
SIREN first omega	$[1e{-}3, \ldots, 10000]$	logarithmic	$[1e{-}3, \ldots, 10000]$	If SIREN activation function is used, the first omega is used for initialization of the first fully-connected layer.
SIREN hidden omega	$[1e{-}4, \ldots, 100]$	logarithmic	$[1e{-}4, \ldots, 3.57]$	If SIREN activation function is used, the hidden omega is used for initialization of the hidden fully-connected layer.
Training steps base	$[1, \ldots, 30]$	integer value	$[1, \ldots, 7]$	The amount of epochs the network trains a training set in the first sequence.
Train steps sequence	$[1, \ldots, 30]$	integer value	$[2, \ldots, 10]$	The amount of epochs the network trains a training set in each sequence expect of the first one. It works best if it is bigger than the training steps at base.
Use variance division	[True, False]	boolean	[True, False]	If true, the loss function is combined with the division of the variance per category.

Erklärung

Ich versichere hiermit, dass ich diese Masterarbeit selbständig angefertigt, nicht anderweitig für Prüfungszwecke vorgelegt, alle benutzten Quellen und Hilfsmittel angegeben, sowie wörtliche und sinngemäße Zitate gekennzeichnet habe.

Kempten, den ....
Unterschrift

Ermächtigung

Hiermit ermächtige ich die Hochschule Kempten zur Veröffentlichung der Kurzzusammenfassung (Abstract) meiner Arbeit, z. Bsp. auf gedruckten Medien oder auf einer Internetseite.

Kempten, den ....
Unterschrift