university of groningen

faculty of science and engineering

Exploring Conceptor Regularization for Incremental Learning of Segmentation Tasks

Graduation Project
(Computational Intelligence)

H.B.R. van de Langemheen (s2985489)

October 5, 2022

Internal Supervisor: Prof. Dr. Herbert Jaeger (Artificial Intelligence, University of Groningen)
External Supervisor: Dr. Hamidreza Kasaei (Artificial Intelligence, University of Groningen)

Artificial Intelligence / Computational Intelligence
University of Groningen, The Netherlands

Acknowledgment

My deepest thanks to Herbert Jaeger for without this project would not have been possible. More than that, thank you for making, and for helping me to make, this project such an engaging one to work on. You have inspired in me an appreciation for the more theoretical side of machine learning which I will carry with me going forward. I could not have wished for a better supervisor. I would additionally like to thank Hamidreza Kasaei for your insight during the project. Given your familiarity with continual learning and of course your previous supervision of Paul Hofman's project, I immediately thought you to be the right external supervisor. Finally, my thanks to Paul Hofman for taking the time to introduce me to your previous work and for allowing me to continue where you signed off. Thanks as well for sharing your code with me, which made getting started on the project a lot smoother.

Abstract

Semantic segmentation is one of many tasks in computer vision where convolutional neural networks (CNNs) are state of the art. However, catastrophic interference prevents feedforward networks from learning a new task in an incremental manner without performance loss on previously learnt tasks. To overcome this, a regularization technique named conceptor-based pseudo-rehearsal (CPR) has recently been adapted for CNNs. Incremental training with CPR has been shown to yield results close to traditional training with fully connected and partially convolutional networks, but has yet to reach similar performance with fully convolutional networks. Segmentation, as opposed to classification, is often solved with fully convolutional networks. To allow for CPR to be used in segmentation, it needs to be adapted to fully convolutional networks. We apply CPR to an incremental adaptation of the Cityscapes dataset, and find that the regularization term is initially too large and becomes numerically unstable. A novel hierarchical conceptor, intended to regulate entire convolutional filters rather than individual weights alone, is introduced as an extension to CPR. These hierarchical conceptors allow for CPR to be used on networks with many weights and filters per layer, but CPR remains insufficient for solving the segmentation task incrementally. We show that, to combat the observed numerical instability, changes to the loss and optimization scheme are required.

Contents

1 introduction

A significant portion of human cognition revolves around visual perception. Distinguishing and recognizing objects is done reflexively, rarely requiring conscious effort from the beholder. Yet this fundamental task which seems so simple to us remains one of the core challenges in computer vision. Parsing digital images in order to attain region description and identification of objects was once thought to be a suitable summer project [1], but in actuality remains a field of active research both theoretical and applied to this day. It is one of the primary obstacles preventing automation of self-driving cars, diagnosis from medical images, farm and crop monitoring, theft detection, and much more.

Figure 1: Image of a cat (left) and a possible semantic segmentation (right).
Legend: sky tree cat grass

1.1 Segmentation

The granularity of the information required from each application of computer vision differs from task to task. In the simplest case, all that is needed is to detect the presence of an object in the image (scene), also known as classification. From there we might like to know where the detected object is located, segmenting the object from the rest of the scene. A next step would be to distinguish multiple instances of the same type of object within the scene, a computer vision task named instance segmentation. Sometimes what is required instead is to distinguish more than one type of object at

the same time within one scene. Each pixel is allocated a single label corresponding to what object type (class) the pixel belongs to. This is referred to as a semantic segmentation of the scene. An example of semantic segmentation is given in Fig. 1. In this work we will primarily focus on the semantic segmentation task, although it should apply to other tasks for which similar models are used.

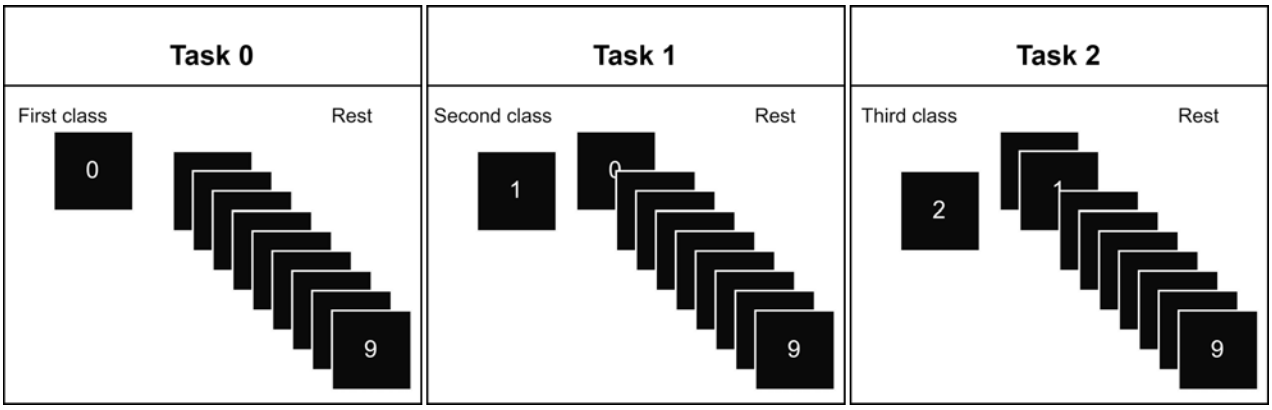


Figure 2: One-vs-all incremental learning scheme. For each task, only the labels of a single associated class are present, the other classes (rest) are unlabelled.

Semantic segmentation traditionally relies on a wide variety of techniques, including clustering approaches, Markov processes, and random forests [2]. However, due to advancements in artificial neural networks many of these traditional techniques have been outperformed by feedforward networks on many of the well-known datasets [3]. These networks typically rely on various deep network architectures and variants of convolutional neural networks (CNNs) [4].

1.2 Incremental Learning

While deep neural networks are powerful function approximators that can be trained to perform many tasks, the classical way of optimizing these networks prevents the model from learning a new task without performance loss on previously learnt tasks. This phenomena, known as catastrophic interference (or catastrophic forgetting), was already documented in 1989 [5], but remains an area of active research. Catastrophic interference arises from network weights optimized for the previously learnt tasks being adjusted by training on the new task, hence interfering with the learnt tasks. In order to optimize the network for both the new task and the previously learnt tasks, the network would have to be retrained from scratch including data for all tasks. However this raises a few issues: retraining may be computationally prohibitive, the data associated with the previously learnt tasks may not be able to be stored (legally) for long enough, and is conceptually far from human learning. After all, humans do not forget learnt tasks just by practicing a new one.

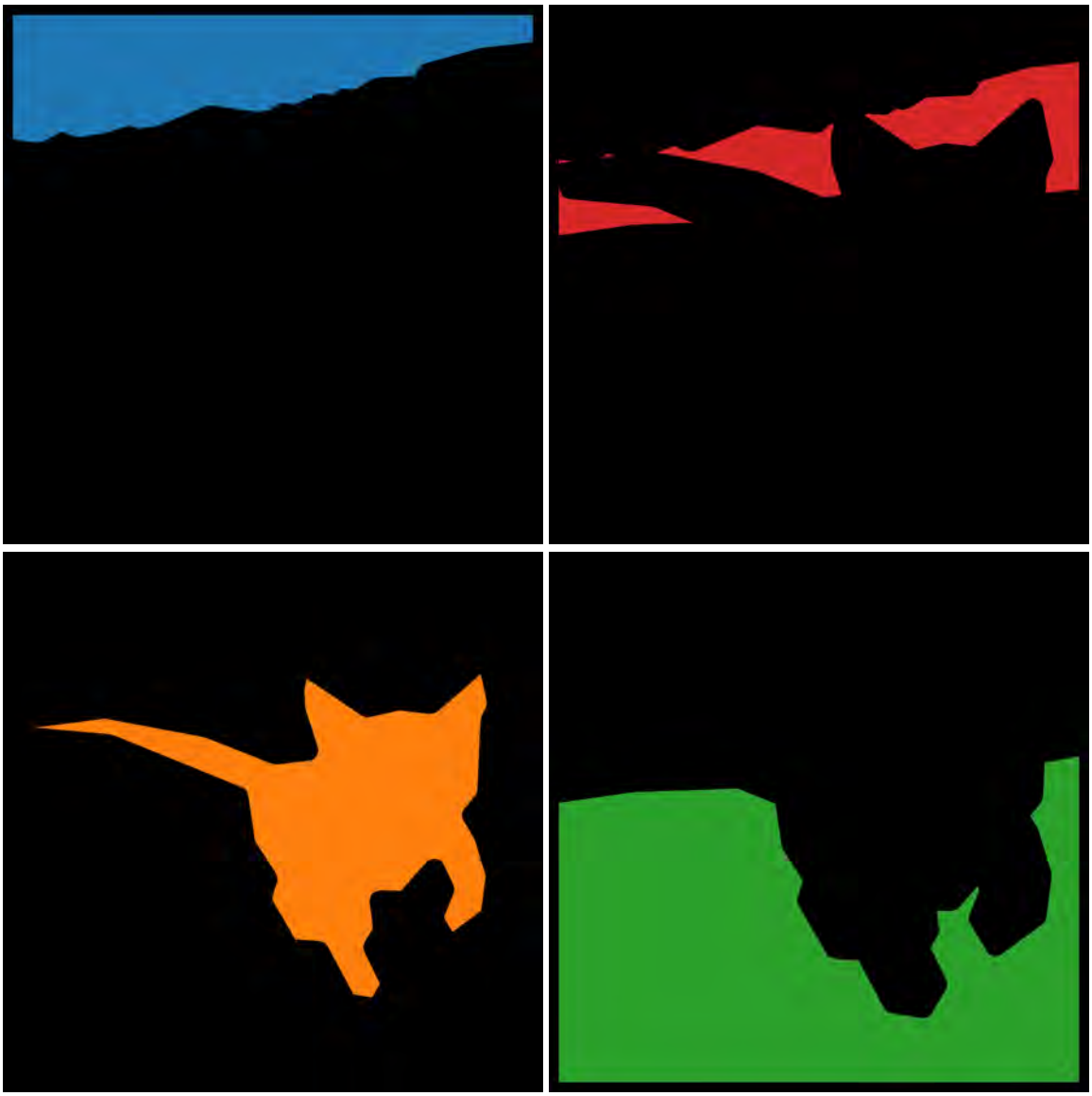


Figure 3: Illustration of how a single image can appear in four different One-vs-all semantic segmentation tasks.
Legend: sky tree cat grass

Training the model on every task simultaneously, henceforth referred to as joint training, is also not possible for all settings. In some cases the data is coming in from a (changing) environment, or the task requirements shift. To deal with this, the model would have to be able to learn a sequence of tasks rather than all tasks at once, without losing performance on all tasks learnt so far. Such

a training scheme is commonly referred to as continual learning or incremental learning, and this training scheme will henceforth be referred to as incremental training.

Though there is only one way to perform joint training, there are many variants on incremental training. These variants arise from different requirements, like whether the model needs to infer which of the learnt tasks it is performing or not [6]. Other alternatives stem from what data the model has access to: are only the task related classes present, or are all classes present but only the task related class is distinguished? While it may be possible to keep only the task related classes present in classification tasks, in semantic segmentation where multiple classes can be present in a single input this is not possible. Our incremental training task, as seen in Fig. 2 differs from those described in [6] in that all training data is available for each task, but only the task-related class is labelled in the straightforward One-vs-all fashion [7]. To adapt a semantic segmentation for One-vs-all incremental learning, for each task only the pixels of the corresponding class are labelled as 1. The other pixels are set to 0. In Fig. 3 the previously shown semantic segmentation is adapted for four different One-vs-all incremental tasks.

As in One-vs-all incremental learning the input remains the same from task to task, and only the expected output (labels) change(s) between tasks, each task can be trained on the same network using a different, dedicated final layer (head) for each task. The multi-headed approach is standard for one-hot encoded output, where adding a new category implies adding a new head [8]. Furthermore, it has seen use in transfer learning and knowledge / network distillation [9, 8].

1.3 Outline

We first go over the background of directly relevant work, and introduce the incremental learning approach on which we use in Related Work. Our newly proposed extensions for one-vs-all incremental learning of segmentation is described in detail in the Proposed Solution section. The section Experimental Setup contains information pertaining to dataset and network architecture used, and the design choices made. It also contains the structure of the experiments performed in order to test our proposed solution, the results of which are detailed in section Results. The results are discussed and thesis limitations are mentioned in section Discussion. The Discussion section also contains a discussion on the newly proposed hierarchical conceptors, and offers directions for future work. Lastly, we conclude with a summary in section Conclusion.

2 Related Work

There are many ways by which the field of study dedicated to incremental learning can be subdivided. Each division comes with its own approaches and applications. In the case of incremental segmentation and image classification, most of the research relies on convolutional neural networks to one degree or another. In this section we first distinguish the different approaches to incremental learning with CNNs. Next, we make a further distinction between the types of incremental learning tasks. These distinctions help narrow down and compare the existing literature. We summarize some of the most notable approaches for incremental learning in relation to segmentation, and provide a non-exhaustive list of relevant literature. Finally, we dedicate a significant part of this section to conceptors and the works that are the primary sources of this thesis work.

2.1 Incremental Learning Approaches for Convolutional Neural Networks

Catastrophic interference is a problem now decades old. In convolutional neural networks this problem has been widely studied with regards to image classification. Michieli and Zanuttigh provide a useful categorization for incremental learning techniques for CNNs [10]. These categories are given as follows:

- Replay-based: This approach relies on replaying data associated with previously learnt tasks while learning the new task. In the simplest case, a curated subset of training data from previous tasks can be stored and reused to prevent catastrophic interference while learning the next task. In cases where data cannot be stored, learning a model that generates training data can be used instead.

- Dynamic architectures: This approach changes the architecture by adding and/or pruning neurons, filters or layers. Typically, neurons trained on previous task(s) are frozen, and new neurons are added to allow for further learning.

- Parameter isolation: This approach generally aims to train tasks only on a subset of the network parameters. Subsequent tasks are then trained on parameters that are so far unused, thus preventing interference with previously learnt tasks.

- Regularization: This approach is the broadest and most popular of the categories listed. In essence, regularization techniques put an additional loss or constraint on the network or latent space. The two primary variants on regularization are knowledge distillation and penalty computing, of which the former is the more prevalent approach.

  - Knowledge distillation: This variant aims to remember or teach knowledge of previous tasks, often in the form of an old model, to the model that is being trained on the new task.

  - Penalty computing: This variant identifies (combinations of) weights that are important to previously learnt tasks, and penalizes changes in them. It shields important (combinations of) weights from being changed, preventing interference with previously learnt tasks and only learning the new task on unimportant weights.

It should be noted that this is not the only way of classifying incremental learning approaches for CNNs, and many solutions incorporate aspects of several categories. Regardless, this categorization allows similar approaches to be grouped, and provides the context for this work. Our approach falls squarely in the penalty computing variant of regularization approaches.

2.2 Distinguishing Incremental Learning Tasks

Besides differentiating between approaches, another important distinction can be made by means of the type of tasks the model has to solve. This can be defined in terms of input and expected output. For example, in some conditions, the model may have to deal with changes in shape and meaning of input data. Other contexts require the model to be able to change its output shape or channels depending on the input. In this and related work, the input and output shape stays the same.

A common way of distinguishing between the different types of incremental learning tasks stems from van de Ven and Tobias [6]. They distinguish between the following three categories:

- Task-incremental: The model knows which task it is performing, and only needs to distinguish between the classes within said task.
- Domain-incremental: The model does not have to distinguish between tasks or classes, and instead only has to identify the index of the correct class.
- Class-incremental: The model has to identify the correct class without knowing which task it stems from.

When it comes to classification and semantic segmentation, the primary objective is to identify the exact label associated with the input image or pixel. As such, our work and the relevant work summarized in next subsection are all instances of class-incremental learning. Yet even within this classification further distinction can and should be made. The amount of classes present in each task (and the order of which) strongly influences the complexity of the task at hand, with learning multiple classes at once rather than sequentially generally being easier than the inverse. Having multiple classes present within one task (rather than one class per task) provides additional information about the relationship between classes, and the level of detail necessary to generalize within a single class while still distinguishing between other classes.

In segmentation, having access to multiple classes during training also provides more information on the background. This alleviates the problem of *background shift* to some degree. Background

shift is, as introduced in [11], the problem that all segmentation classes not present in a task are labelled as background. Intuitively, the problem is at its worst when only one segmentation class is present in each task. Some of the background may in actuality belong to classes seen in previous tasks, or to classes that are yet to be seen in future tasks. This introduces a semantic shift in what the background means for each task.

Given the relevance of the amount of classes present in each task to task complexity, class-incremental learning can additionally be understood by the distribution of classes over the tasks. Some examples are:

- Uniform: Each task has the same amount of classes, typically half of the classes in the first task, and the other half in the second.
- Step-wise: The first task has the largest amount of classes. The remaining classes are distributed uniformly over the subsequent tasks. For example, the first task features 15 classes, the second, third and last task each feature 5 classes. The two special cases are when there are only two tasks, or when each subsequent task beyond the first has only one class.
- One-vs-all: Each task has exactly one class. This is a special case of the uniform class distribution, where the amount of tasks is equal to the amount of classes.

A final distinction is made in terms of presence of previous input images and previously seen classes:

- Sequential: Each task has a unique (task related) set of inputs, with the labels of both previous and newly introduced classes being present for each task. For each task, there is no input which features pixels belonging to classes that will be learnt in future tasks.
- Disjoint: Here the inputs are similarly unique as in ”Sequential”, but only the classes associated with the current task are present.
- Overlapped: The input remains the same for each task, where only the current classes are annotated and the other classes are labelled as background. This is different from the previous two categories as images can now contain pixels of classes that will be learnt in future tasks.

While it may seem excessive to subdivide the class-incremental learning task to such a degree, the performance of relevant work provided in the next subsection will show why these subcategorizations are important. Crucially, solutions to catastrophic interference may respond differently to variations in task definition. Task definitions can vary along the aforementioned axes, but broader design choices should be considered as well: explicitly distinguishing the background from unlabelled classes, changing the order of tasks or classes, and limiting training time or network architecture all influence performance. When an article uses a specific task definition, or makes design choices specific to their solutions, their findings may not be replicated in subsequent studies with a dissimilar task definition.

2.3 Summary of Relevant Work

In this subsection we will introduce some of the prominent solutions to incremental learning applied to CNNs. Specifically we chose approaches that have been or could be applied to semantic segmentation. Important to note is that each of the articles described here have their own model architecture and task design choices. As such, the performance cannot be directly compared, and instead the only comparison in performance should be between the incremental training and joint training performance of the same model. A table containing an overview of the works listed here is provided at the end of this subsection.

2.3.1 Selfless Sequential Learning

As an improvement to introducing network sparsity, Selfless Sequential Learning (SSL) introduces a method for the parameter isolation approach that instead disentangles representation. Local neurons are discouraged from activating together for a given task, unless weights from previous tasks are reused. This introduces a sparsity in the network that is less likely to result in catastrophic interference when training a new task [12].

2.3.2 Deep Generative Replay

Rather than having a model that learns a new task, and another model that teaches previously learnt tasks for knowledge distillation, Deep Generative Replay (DGR) combines both into a *scholar* model. In DGR, a scholar is trained for each task which learns from the previous scholar and the current task [13].

2.3.3 Incremental Classifier and Representation Learning

While most classification and segmentation methods in this list use CNNs for the classification / segmentation, in Incremental Classifier and Representation Learning (iCaRL) the CNN is a trainable feature extracter. The features from the CNN are paired with exemplar images collected from each task for incremental classification [14].

2.3.4 Elastic Weight Consolidation

By identifying which individual weights are important to a learnt task, Elastic Weight Consolidation (EWC) learns new tasks without interfering with weights important to previously learnt tasks. The weights important to a learnt task are identified by means of the Fisher information matrix. [15]

2.3.5 Conceptor-based Pseudo-Rehearsal

This thesis work is directly based off of the Conceptor-based Pseudo-Rehearsal (CPR) work by He [16], which was later adapted to CNNs by Hofman [17]. CPR uses regularized identity maps computed from network activation called conceptors, as defined by Jaeger [18], to prevent interference with previously learnt tasks. Conceptors and CPR will be given a more extensive introduction further on in this section.

2.3.6 Learning Without Forgetting

Rather than using data from previous tasks to prevent catastrophic interference, Learning Without Forgetting (LWF) uses models from previous tasks. After the first task, the new model is trained on the data from the new task and on the previous model's response to the new data (the distillation loss). The target is to obtain a new model that is able to learn the new task while behaving similar to the old model on data from the previous task(s) [8].

2.3.7 Learning Without Forgetting Multi-Class

The same paper that introduced iCaRL introduced a multi-class variant of LWF (LWF-MC). This variant is derived from iCaRL by similarly using a distillation loss to prevent catastrophic interference in the CNN, yet unlike iCaRL the predictions are made from the CNN itself, rather than on the basis of the exemplars introduced in [14].

2.3.8 Incremental Learning Techniques

One of the few approaches developed initially for incremental segmentation rather than incremental classification, the Incremental Learning Techniques (ILT) paper proposes several extensions to knowledge distillation. The first extension improves knowledge distillation in the output layer, the second proposes a frozen encoder part of the model, and the final extension explores distilling the latent space rather than the model output [19].

2.3.9 Modelling the Background

Introducing the problem of *background shift*, Modelling the Background (MIB) proposes a solution by predicting the background with previously learnt classes. Unlabelled pixels can either be from previously learnt classes or not (true background, or classes that will be learnt in future tasks). If the model can label an unlabelled pixel as a previously learnt class then this provides additional information to the current task, and alleviates some of the problems with background shift [11].

2.3.10 Class-Incremental Learning

Another approach that predicts the background is named Class-Incremental Learning for Semantic Segmentation Re-Using Neither Old Data Nor Old Labels (CIL). This approach limits the task loss to only the pixels that are labelled in the current task, and adds a distillation loss to the background pixels. This distillation loss is weighted to emphasize uncertainty of the teacher model [20].

2.3.11 Sparse and Disentangled Representations

Combining various approaches, Sparse and Disentangled Representations (SDR) use regularization, knowledge distillation and parameter isolation to prevent catastrophic interference. SDR introduces a combination of prototype matching, contrastive learning, and feature sparsity in the latent space. Furthermore, SDR adds a distillation loss at the output level as in MIB [21].

2.3.12 Incremental Learning for Semantic Segmentation

In the case of remote sensing, data may be available for one city but not for another. Incremental Learning for Semantic Segmentation of Large-Scale Remote Sensing Data (ILSS) exploits this by replaying on a curated selection of training data for previously encountered cities, and distilling previous models for previously learnt classes when training new tasks with satellite images from new cities [22].

2.3.13 Twin-Auxiliary

Rather than only using the previous model for the distillation loss, Twin-Auxiliary (TA) modules additionally train a segmentation and a binary classifier for the new model as well as for its twin: the frozen, previous model. Li et al. present an extensive pipeline for recombining the outputs from the twin modules that reportedly outperforms joint training [23].

2.3.14 Overview

Table 1 provides an overview of the aforementioned methods with a mean IoU (mIoU) score where applicable. The source column indicates from which original work the referenced incremental and joint training scores are. The reported incremental mIoU scores are all from experiments with step-wise class distributions, where all but the first tasks feature one class per task. This distribution is the closest to the one-vs-all distribution used in [16, 17] and in this work.

Different source materials sometimes compare the same methods. In the case of MIB and SDR, a comparison is also done on the same (VOC2012) dataset. Yet even when the methods and dataset are the same, and the class distribution is step-wise, discrepancies in performance can still be observed. Specifically, LWF outperformed LWF-MC in [21], while the opposite is true for [11].

Table 1: Summary of Incremental Learning Approaches in relation to Segmentation

Source	Method	Approach	Segmentation?	One-vs-all?	Dataset	mIoU	
						Incremental	*Joint*
SSL [12]	SSL	Parameter isolation	No	No	MNIST, CIFAR-100 & Tiny ImageNet	-	-
DGR [13]	DGR	Replay-based	No	No	MNIST & SVHN	-	-
iCaRL [14]	iCaRL	Replay-based	No	No	iCIFAR-100 & iILSVRC	-	-
EWC [15]	EWC	Penalty computing	No	No	MNIST & Atari	-	-
CPR [16]	CPR	Penalty computing	Yes	Yes	Cityscapes	0.570	0.840
MIB [11]	LWF	Knowledge distillation	Yes	No	Cityscapes	0.259	0.731
	LWF-MC	Knowledge distillation	Yes	No	Cityscapes	0.396	0.731
	ILT	Knowledge distillation	Yes	No	Cityscapes	0.238	0.731
	MIB	Knowledge distillation	Yes	No	Cityscapes	0.607	0.731
	LWF	Knowledge distillation	Yes	No	VOC2012	0.018	0.774
	LWF-MC	Knowledge distillation	Yes	No	VOC2012	0.069	0.774
	ILT	Knowledge distillation	Yes	No	VOC2012	0.057	0.774
	MIB	Knowledge distillation	Yes	No	VOC2012	0.297	0.774
SDR [21]	LWF	Knowledge distillation	Yes	No	VOC2012	0.219	0.754
	LWF-MC	Knowledge distillation	Yes	No	VOC2012	0.058	0.754
	ILT	Knowledge distillation	Yes	No	VOC2012	0.046	0.754
	CIL	Knowledge distillation	Yes	No	VOC2012	0.059	0.754
	MIB	Knowledge distillation	Yes	No	VOC2012	0.367	0.754
	SDR	Regularization & knowledge distillation	Yes	No	VOC2012	0.392	0.754
TA [23]	EWC	Penalty computing	Yes	No	Vaihingen	0.211	0.747
	ILT	Knowledge distillation	Yes	No	Vaihingen	0.736	0.747
	ILSS	Replay-based	Yes	No	Vaihingen	0.553	0.747
	MIB	Knowledge distillation	Yes	No	Vaihingen	0.740	0.747
	TA	Knowledge distillation	Yes	No	Vaihingen	0.757	0.747

2.4 Conceptor Regularization

A novel approach that allows for one-vs-all incremental learning with feedforward networks utilizes regularized identity maps called conceptors. These conceptors form the basis of the conceptor regularization approach and is core to this work. As such, we will first provide an introduction to conceptors as they relate to this work. Next, we describe how conceptors were used to overcome catastrophic forgetting in one-vs-all incremental learning. Finally we introduce the thesis work of which this work is a direct continuation. This previous thesis work adapts the conceptor regularization to convolutional neural networks, allowing for one-vs-all incremental segmentation, which is the focus of our work.

2.4.1 Conceptors

Jaeger introduces conceptors for a variety of neuro-computational functionalities [18, 24], providing a generic computational mechanism for neural networks. While the scope of our work is limited to fully convolutional networks, conceptors are not limited to just feedforward artificial neural networks. Instead they are architecture agnostic, and can be used to describe patterns in any neural network or dynamical system. Yet conceptors are even more broadly applicable; they can be seen as a new approach towards explaining how high-level logical reasoning can arise from low-level dynamic processes as is happening in the human brain.

An overview of some of the applications for conceptors given by Jaeger [18] follows. Conceptors lend themselves to:

- Perceptual focusing
- Noise filtering
- Memory management
- Incremental learning
- Smooth transitioning between modes of (motor) control
- Symbolic representations of dynamical patterns
- Hierarchical classification and Boolean logic on said representation

All of this speaks to a potential integration of otherwise disparate cognitive processes through the conceptor as unifying factor.

There is no one ”best” way to instantiate a conceptor. For example, conceptors can be implemented in individual neurons, stored explicitly as a matrix, or learnt through a separate sub-network. An adaptation exists where the conceptor no longer needs to be calculated and stored away from the network by an outside process, and the conceptor is instead formed over time by the network dynamics themselves. Efforts have even been put towards identifying a basis for conceptors that is biologically (not im)plausible.

Unfortunately, this thesis work does not have the scope to adequately address conceptors in its entirety. Instead we will introduce the core insight behind conceptors, and summarize useful properties as they relate to the usage of conceptors in incremental learning with feedforward networks. Before conceptors can be more formally explained, we must first sketch the context in which the core idea underlying conceptors arose. The key insight behind conceptors stems from the behaviour of dynamic, non-feedforward recurrent neural networks (RNNs). These networks feature complex dynamical behaviour in which time is an integral part of the model. Activation in the form of the model’s internal state changes over time as dictated by model dynamics and input patterns.

Let $\mathcal{N}$ be an RNN of any type with N neurons. From these neurons the internal state set $\{x\}$ (the network activations) can be obtained. In RNNs, $\{x\}$ is a function of not only some input sequence but also of previous internal state(s). This way, when a sequence (pattern) p is fed into $\mathcal{N}$ some information from earlier on in the sequence is retained in the network. Formally, $\mathcal{N}$ *is being driven by dynamical input pattern* p. The network may be driven by different patterns $p^1, p^2, ..., p^j$, which will yield different internal state sets $\{x^1\}, \{x^2\}, ..., \{x^j\}$. When driving the network with a pattern p^j, the resulting internal state set $\{x^j\}$ lies in an N-dimensional state cloud. The geometry of this state cloud is a property of the dynamics in $\mathcal{N}$ and pattern p^j. This geometry can be described in formal terms by an ellipsoid C^j, which is *the conceptor* C^j *associated with pattern* p^j *in network* $\mathcal{N}$ [24]. The conceptors associated with some patterns p^1, p^2 and p^3 are shown in Figure 4. Note how the *conceptor space* is three dimensional in this example, which means activation was collected from three neurons.

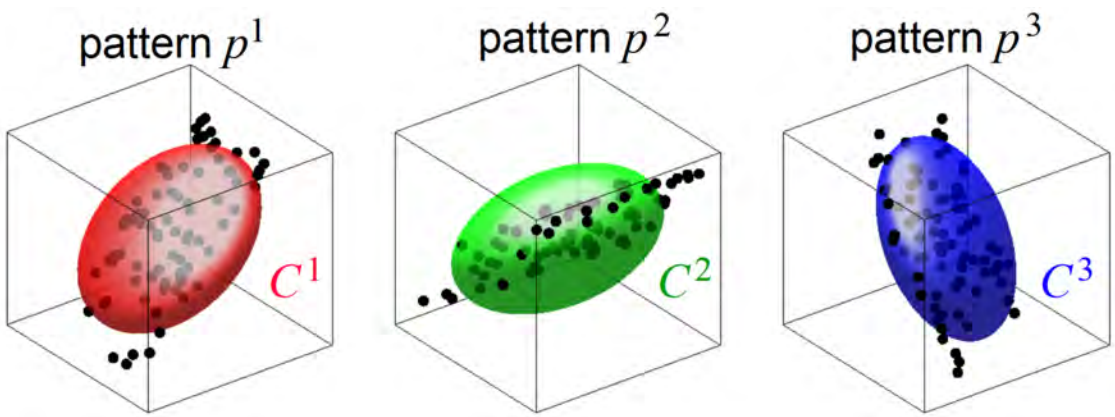


Figure 4: Neural activation in response to three different patterns projected onto *conceptor space*, and the associated ellipsoidal conceptor representation. Figure from [24]

As mentioned previously, there are multiple ways of obtaining and representing a conceptor C^j. This process can generally be described as learning a regularized identity map. The most explicit method to learn and represent C^j is by means of a *conceptor matrix*. This conceptor matrix can be constructed by sampling from $\{x^j\}$. Going forward we will simply refer to the conceptor matrix of C^j as *the conceptor* C^j, given that in this work we only interact with conceptors by means of their matrix representation. To calculate a conceptor C^j, drive the network $\mathcal{N}$ with pattern p^j. Sample the network state set $x^j(n)$ at each sampling point $n \in \{0, 1, ..., V-1\}$ from $\mathcal{N}$ as vectors of length N, where N is the number of neurons in $\mathcal{N}$. Collect each state activation $x^j(n)$ as rows in a L by N state matrix X^j. From the state matrix, a state correlation matrix is computed: $R^j = X^j(X^j)^\top/V$. Here V is the amount of network state samples, or the sampling time.

While the shape and dimensions of C^j are defined by the ellipsoidal state correlation matrix R^j,

not all directions are equally relevant. A parameter called *aperture* α is introduced to regulate how many of the leading dimensions of R^j contribute to C^j. The state correlation matrix together with aperture defines a conceptor. This relationship can be made explicit by writing C^j as $C^j(R^j,\alpha)$. The conceptor $C^j(R^j,\alpha)$ can be learnt by minimizing the cost function $\mathcal{L}(C^j|R^j,\alpha)$. Minimizing this function means minimizing two parts, one associated with R^j and one associated with α.

Because conceptors are defined as regularized identity maps, we want C^j to reconstruct activation x^j that is aligned with the principal components ("thick" dimensions) of the ellipsoid represented by C^j: $C^j x^j \approx x^j$. This requirement yields the first component of the cost function: $\mathbb{E}_n[||x^j(n) - C^j x^j(n)||^2]$. This component is the L_2 norm of the difference between a state activation x^j and its conceptor projection $C^j x^j$, averaged over each time step n. The second component of the cost function is simply the α scaled Frobenius norm of the conceptor: $\alpha^{-2}||C^j||^2_{fro}$. A conceptor $C = C(R,\alpha)$ associated with R and α is then obtained through

$$C(R,\alpha) = argmin_C \mathbb{E}[||x - Cx||] + \alpha^{-2}||C||^2_{fro} \tag{1}$$

Which has a closed-form solution given as

$$C(R,\alpha) = R(R + \alpha^{-2}I)^{-1} \tag{2}$$

Here I is the N by N identity matrix. Eq. (2) allows us to construct C from the singular value decomposition (SVD) of R. Let the SVD of R be $R = U\Sigma U^\top$. The SVD of $C(R,\alpha)$ is then given as $C = USU^\top$. The singular values s_i of C are acquired from the singular values σ_i of R with $s_i = \sigma_i/(\sigma_i + \alpha^{-2})$. The proof of this is given in [18].

A conceptor C obtained as described above has certain useful properties. The mean value of S intuitively measures the fraction of the N dimensions of $\mathcal{N}$ occupied by C. This is referred to as the *quota* q of C, $q(C) = N^{-1}\sum_{i=0}^{N-1} s_i$. If C^j is obtained from a network $\mathcal{N}$ that has learnt pattern p^j, then $q(C^j)$ is an indication of how much of the N-dimensional *memory space* of $\mathcal{N}$ is occupied by learning p^j.

It is easy to see how such properties of conceptors are useful for memory management. When learning a second pattern, e.g. p^m after having already learnt p^j, a conceptor $C^{\{j,m\}}$ spanning both learnt patterns can be computed from a merged state activation matrix $X^{\{j,m\}}$. This merged state activation matrix is obtained by merging X^j with X^m. However, if we already have C^j we do not need to merge at the dataset level to obtain $C^{\{j,m\}}$. Because conceptors lend themselves to Boolean operations, merging C^j with C^m (the conceptor computed from X^m) also yields $C^{\{j,m\}}$. This merging operation is the logical OR operation. The Boolean operation $\neg, \vee, \wedge$ on conceptors C^j and C^m are shown visually in Figure 5 and are defined as follows:

$$\neg C := I - C \tag{3}$$

$$C^j \vee C^m := \neg(\neg C^j \wedge \neg C^m) \tag{4}$$

$$C^j \wedge C^m := ((C^j)^{-1} + (C^m)^{-1} - I)^{-1} \tag{5}$$

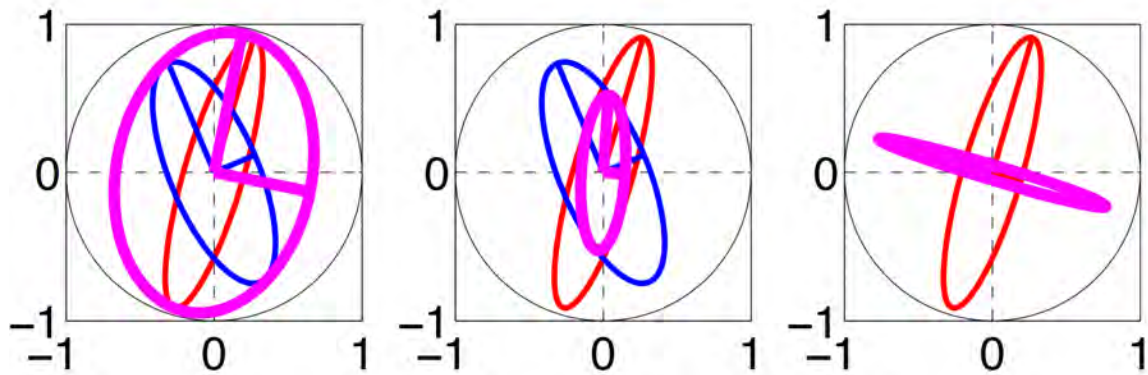


Figure 5: The three Boolean operations $\vee$ (left), $\wedge$ (middle), and $\neg$ (right) on conceptors C^j (red) and C^m (blue). Figure from [18].

When learning patterns in a sequential manner, the currently learnt patterns and how much memory space they occupy can be tracked by keeping an *updated* conceptor which represents all learnt patterns after learning a new pattern. For example, the updated conceptor $C^m = C^{\{j,m\}}$ is obtained with $C^m \mapsto C^m \vee C^j$ using Eq. (4). Unused, available memory space on which new patterns can be learnt is revealed with the NOT operator from Eq. (3): $\neg C^m$.

2.4.2 Conceptors in Feedforward Networks

In Section 2.4.1 we summarized some of Jaeger's work on conceptors, and how they can be applied for incremental learning in recurrent neural networks [18]. Even though conceptors are inherently network agnostic, to use conceptors for incremental learning in feedforward networks they have to be adapted to work with backpropagation. This is a non-trivial task: feedforward neural networks learn through small changes in weights guided by a loss function. The loss has to be differentiable to obtain the gradients, and these have to be backpropagated to change the network weights in gradually.

The weight changes during backpropagation are also the cause of the catastrophic interference introduced previously. Any attempt at using conceptors for incremental learning will have to do so while preventing catastrophic forgetting. Recent work by He and Jaeger introduces an extension to backpropagation in which interference with previously learnt tasks is prevented by shielding gradients with conceptors [25]. This extension, called *conceptor aided backpropagation* (CAB), is shown experimentally to outperform two other recently proposed solutions to the catastrophic interference problem.

Some of the terminology described in Section 2.4.1 will have to be adapted in order to relate the use of conceptors to feedforward networks. Network $\mathcal{N}$ is no longer *driven by pattern p*, but instead is *trained on task t*. Furthermore, a deep feedforward network $\mathcal{N}$ consists of L layers of N_l neurons per layer l. Conceptors C_l are still calculated from a state correlation matrix R_l, but now per layer l rather than over the entire network. Layer activation x_l is sampled during the forward pass by providing the network with V samples of training data. The conceptor obtained from layer l after training on task t is then denoted as C_l^t.

After training normally on the first task a conceptor is computed for each layer l. CAB then aims to train the network on each subsequent task j such that the changes W_l^{inc} made to the previously learnt weights W_l^{j-1} do not interfere with any previously learnt tasks. Intuitively, this is done by only allowing changes in weights in unused memory space. Unused memory space is characterized by the conceptor F_l^{j-1}. Here, F_l^{j-1} is the result of using the NOT operation from Eq. (3) on the *updated* conceptor A_l^{j-1}. The updated conceptor characterizes the memory space of all $0, ..., j-1$ previously learnt tasks thus far, such that $A_l^{j-1} = C_l^0 \vee ... \vee C_l^{j-1}$ and $F_l^{j-1} = \neg A_l^{j-1}$. This is the case for each layer l in the network, except for the last one which is the layer head. After training on each subsequent task with CAB, the conceptor in each layer is updated to include the memory space of the newly learnt task such that $A_l^j = A_l^{j-1} \vee C_l$, $F_l^j = \neg A_l^j$.

With F^{j-1} we can find a change in weights W_{inc}^j for learning the new task j exclusively outside the linear subspace occupied by previous tasks. This is done by requiring the Frobenius norm of the conceptor projected gradient to approach zero: $||W_l^{inc} A^{j-1}$. This effectively means that only changes in weights in the "thick" directions (directions with large singular values) of F^{j-1} are allowed through.

In CAB, the activation state matrix for the conceptors is collected by sampling the neural activation x_{l-1} in the layer $l-1$ *before* a given set of weights W_l. In He and Jaeger [25] this is called the *pre-synaptic* activation. Similarly, activation in neurons x_l *after* a given set of weights W_l is called the *post-synaptic* activation. CAB uses the pre-synaptic activation rather than the post-synaptic activation because the conceptors have to apply in a backwards fashion: to the backpropagated gradients.

Computing conceptors from pre-synaptic activation x_{l-1} to prevent catastrophic forgetting in the subsequent weights W_l has its limits, however. When different tasks have partial or even completely overlapping input space, the input distribution in the pre-synaptic neurons for the new task will be similar or identical to previously learnt tasks. This causes there to be no input components available for further adaptation, preventing significant learning from occurring in all layers except for the model head. Many computer vision tasks, especially with the one-vs-all training scheme frequently seen in incremental learning, have identical input spaces. This limits the applicability of CAB.

He, having identified this drawback in CAB, proposes an alternative that uses post-synaptic activation [16]. This new solution, later called *conceptor-based pseudo-rehearsal* (CPR), is experimentally shown to yield close to joint training performance on a one-vs-all incremental learning task, overcoming the hurdle of shared input space. CPR introduces a novel *conceptor regularization* term Ω_C, which shields weights based on conceptors calculated from post-synaptic activation.

Where CAB shields weights through the backwards projection of the gradients into conceptor space, CPR shields weights by projecting *pseudo-activation* forwards into conceptor space.

CPR shields weights in tasks beyond the first by providing *Gaussian pseudo-data* ξ (normal distributed noise) as input to the pre-synaptic neurons in layer $l-1$, and collecting the pseudo-activation in the post-synaptic neurons in layer l with activation function g. This is done for the weights as they were after previously learnt task $t-1$: $g(W_l^{t-1}\xi)$, and for the new weights that are currently being trained on the new task t: $g(W_l^t\xi)$.

Having post-synaptic pseudo-activation for both the old and the new weights allows us to gain a measure of difference d_l between the two, done here by subtracting the former from the latter such that $d_l = g(W_l^t\xi) - g(W_l^{t-1}\xi)$. If C_l^{t-1} is the *updated* conceptor characterizing the used memory space of all previously learnt tasks up to and including task $t-1$ in layer l, then projecting d_l onto this memory space with $C_l^{t-}d_l$ indicates how much the new weights changed from the old weights along directions occupied by previously learnt tasks. As changes in directions occupied by previously learnt tasks would interfere with said tasks, we want $C_l^{t-1}d_l$ to approach 0. The L_2 norm is a measure for this. Averaging over each of the $v \in \{0,...,V-1\}$ pseudo-data samples yields the loss term $\mathbb{E}_v[||C_l^{t-1}d_l||_2^2] \approx 0$ for layer l.

The full conceptor regularization term Ω_C for layer l, previous weights W_l^{t-1}, current weights W_l^t, and the *updated* conceptor C_l^{t-1} is then expressed by $\Omega_C(W_l^{t-1}, W_l^t, C_l^{t-1}) := \mathbb{E}_v[||C_l^{t-1}(g(W_l^t\xi) - g(W_l^{t-1}\xi))||_2^2] \approx 0$. Note that an *updated* conceptor C_l^{t-1} is the conceptor characterizing the memory space of all tasks learnt up to and including $t-1$ for layer l. This is similar to how A_l^{t-1} in CAB is obtained, through the $\vee$ operation on each individual task conceptor.

In CPR as in CAB, the first task is learnt normally with the task-specific loss $\mathcal{L}_{task}(\hat{y}, y^t)$, where $\hat{y}$ is the prediction of network $\mathcal{N}$ on input x, or $\mathcal{N}(x) = \hat{y}$ and y^t is the true label associated with x. Additionally, a regularization term is placed on the change in weights to prevent the network from using the complete memory space for a single task when this is unnecessary for task performance. This regularization term is controlled by λ, with higher values being more restrictive. We then write this regularization term as $\lambda \sum_{l=0}^{L-1} ||W_l^{inc}||_{fro}^2$. Putting this together with the task-specific loss, the objective for the first task $t=1$ becomes

$$\mathcal{J}^{t=1} = \mathcal{L}_{task}(\hat{y}, y^t) + \lambda \sum_{l=0}^{L-1} ||W_l^{inc}||_{fro}^2 \tag{6}$$

After a (first) task $t-1$ is learnt, but before starting training on the next task t, a conceptor C_l^{t-1} is computed for each layer l. If previous conceptors exist (after the second task onward), the newly computed conceptors are updated with the previous ones. With these conceptors, the tasks after the first can be learnt incrementally with the addition of the conceptor loss Ω_C. The conceptor loss is summed over each layer except the model head. Due to this summation the minimum value is still 0, but the values can become very large and depend on both the amount of layers as well as the change in weights from one task to the next. The summed conceptor loss is then scaled by γ. Higher values

of γ means more of the memory space occupied by previously learnt tasks is preserved. Adding this to Eq. (6), the objective for incremental learning in CPR beyond the first task becomes

$$\mathcal{J}^{t\forall} = \mathcal{L}_{task}(\hat{y}, y^t) + \lambda \sum_{l=0}^{L-1} ||W_l^{inc}||_{fro}^2 + \gamma \sum_{l=0}^{L-2} \Omega_C(W_l^{t-1}, W_l^t, C_l^{t-1}) \tag{7}$$

He shows both formally and empirically how training fully connected feedforward networks incrementally with Eq. (7) overcomes the shortcoming of CAB, and achieves state-of-the-art performance in the Disjoint MNIST [26] task [16].

2.4.3 Conceptors in Convolutional Neural Networks

Where He achieved state-of-the-art with fully-connected feedforward networks, state-of-the-art in many other computer vision tasks is achieved with convolutional neural networks (CNNs). To utilize CPR with CNNs, Hofman shows that conceptors can be computed and used as in [16] by unfolding the convolutional filters and input image/feature map [17]. By vectorizing the convolutional operation, CPR can be used in CNNs without any modification to the objectives shown in Eq (6) and Eq (7). Because the conceptor operations in Hofman's work are identical to those in [18, 16], we will only explore the unfolding operation for CNNS in detail. Furthermore, as this thesis work can be understood as a direct continuation of [17], we will summarize Hofman's experiments and results in greater detail than that of previous works discussed in this section.

The name *convolutional* neural networks stem from the idea that, in a convolutional layer, one or multiple filters *convolve* over the input image or feature map. This convolving operation can be thought of as sliding the filter over the input. The convolution operation is, with this framing, prohibitively costly. If you have f_l and f_{l-1} filters on layers l and $l-1$ respectively, each with a square kernel of k by k weights, and the output of layer l is a feature map of h_{out} by w_{out} pixels, then the naive convolution has time complexity $O(k^2 * f_l * f_{l-1} * h_{out} * w_{out})$.

By unfolding both the convolutional filters and the input, the convoluting operation can be implemented as a single matrix multiplication. The filter dimensions are given as $[k,k,c]$, With F k by k filters each having c channels. The input dimensions are given as $[h_{in}, w_{in}, c]$, or h_{in} pixels high by w_{in} pixels wide, having c channels. The key insight here is that a single output from the convolution is the result of multiplying the flattened filter(s) by a flattened input patch the size of the filters (in this case, k by k). Repeating this flattening operation for all F filters and each of the P input patches, the input and filters are reshaped into $[k*k*c, P]$ and $[F, k*k*c]$ respectively. This also makes explicit the weight sharing in convolutional networks: the same weights are used P times.

After training on a task, the activation to compute conceptors from can be sampled as in [16] by taking the flattened feature map as a single row in state activation matrix X. From here, the rest of the CPR operations apply. Note that the pseudo-data used in Ω_C has to be unfolded similar to other input to a convolutional layer, and the pseudo-activation has to be flattened to be projected into conceptor space.

Hofman shows that a small convolutional neural network with only two convolutional layers and a fully connected one can outperform the state-of-the-art set by He [16] for the incremental one-vs-all MNIST experiment. Given the effective application of CNNs with CPR on incremental learning of classification tasks, Hofman proceeded with an incremental one-vs-all segmentation experiment. This experiment was performed on an one-vs-all incremental variant of the Cityscapes [27] dataset. In the experiment the order of the segmentation tasks are as follows: First roads, secondly cars and thirdly buildings.

Joint training on only the first two tasks yielded an Intersection over Union (IoU) [28] score of 0.84. However, training incrementally already showed a decrease in performance, only reaching an IoU of 0.78. This dropped to 0.57 after learning the third task, while the performance for joint training reportedly stayed at 0.84. While in the one-vs-all classification experiment, incremental training with CNNs and CPR yielded results approaching joint training, this is not the case for one-vs-all segmentation.

Theoretically, the network used by Hofman should still have capacity to learn the new tasks. Even after training incrementally on all three tasks, the quota q of the conceptors are still significantly below 0.5 in each layer. However, q may not be the limiting factor. Hofman poses that there may not be a change in weights W^{inc} that minimizes both the conceptor regularization term and the task loss. In other words, learning the new task may require a W^{inc} that initially increases Ω_C.

CPR is still a novel approach to incremental learning, and Hofman was the first to apply CPR with CNNs. What causes the decrease in performance in segmentation specifically, or how CPR interacts with the weight sharing and CNNs in general, are still not well understood. Our thesis work is a continuation of Hofman's, with the aim to gain a better understanding of CPR for incremental one-vs-all segmentation. To this end we propose two extensions to CPR which we believe may improve performance with convolutional filters.

3 Proposed Solution

While we have just discussed several solutions towards incremental learning of segmentation tasks, there is an intersection between penalty computing approaches and the specific one-vs-all incremental segmentation task described previously for which little to no literature exists. Most penalty computing approaches regulate some encoded representation stemming from one or more fully connected layers, making it incompatible with fully convolutional networks. Previously discussed work also usually puts several classes together in each task, as opposed to the one class per task seen in one-vs-all incremental learning. This allows the network to learn generalizations on the classes presented together within a class, making it an easier task to solve than the one-vs-all variant.

Using unfolded convolutional layers, Hofman was able to apply CPR to convolutional networks. This yielded good results for classification, in which the network contained fully connected layers, but performed poorly in segmentation with a fully convolutional network. To explore what may be causing this poor performance we repeat the experiment described in [17] with a different training scheme, and perform additional diagnostics. As a step towards a solution, we further propose two extensions to CPR. This section defines the newly proposed extensions to CPR. The repeated experiment, as well as comparable experiments for the two extensions and diagnostic experiments, are detailed in section Experimental Setup. The Experimental Setup section also provides full details with regards to the specific architecture used, implementation specific settings, and other design choices.

3.1 Filter Conceptors

The key insight for the first proposed extension is that conceptors computed from an entire convolutional layer contain redundant or even misleading information. For a given convolutional layer l its conceptor C_l is computed from autocorrelation matrix R_l. R_l contains the n by n autocorrelation of activation samples collected from each of the n post-synaptic neurons in l. If l is a convolutional layer with more than one filter, the weights in a given filter only affect the feature map corresponding to said filter. As such, only the interaction with other weights within a filter are relevant to the output. Computing C_l from R_l will contain correlations of weights between different filters, and the singular vectors of C_l may be based on linear combinations of weights across more than one filter.

Small singular values might incorrectly suggest that the model still has capacity for learning when these singular values correspond to singular vectors that are constituted by weights across more than one filter. Learning a new task in this seemingly free memory space can inadvertently still negatively influence previously learnt tasks, as changes in weights within a filter influences the output of said filter differently than fully connected layers due to weight sharing. Changing weights in this seemingly free memory space also does not ensure that learning occurs in these (sub)dimensions, since weights in different filters do not interact with each other. Concretely, C_l incorrectly characterizes the available memory space in l.

A straightforward solution for fully convolutional networks would be to then compute F conceptors for all layers L instead, excluding the model head $L-1$. This yields one conceptor for each of the F filters $\forall f \in \{0,...,F-1\}$ in each layer but the last $\forall l \in \{0,...,L-2\}$. We then introduce an additional index to the notation to specify a filter: R_l^f becomes the k^2 by k^2 autocorrelation matrix obtained from activation samples of the k^2 neurons in filter f in layer l with kernel shape [k, k]. Similarly, C_l^f is the (flattened) filter-sized conceptor computed from R_l^f.

Filter-sized conceptors, or filter conceptors for short, can otherwise be used identically to how conceptors were used by Hofman in [17]. These conceptors can be used for CPR in incremental learning, by computing and updating the conceptors after each task [16]. We add one more index for each task t in the total set of T tasks $\forall t \in \{0,...,T-1\}$. For example after training on two tasks, the autocorrelation matrix of filter f in layer l on the current (the second) task t is noted as $R_l^{t,f}$, and the conceptor computed from $R_l^{t,f}$ becomes $C_l^{t,f}$. The conceptor that was computed from previous (the first) task would then be $C_l^{t-1,f}$.

Continuing the example, after training on the second task (and each subsequent task beyond the first) every conceptor $\{C_l^{t,f} | f \in \{0,...,F-1\}, l \in \{0,...,L-2\}\}$ must be updated with its counterpart from the previous task to characterize not just the currently learnt task but also (every) previous task. This is done with the $\vee$ operator, as in [17, 16]. Each conceptor is then updated to characterize all tasks learnt thus far: $\{C_l^{t,f} \mapsto C_l^{t,f} \vee C_l^{t-1,f} | f \in \{0,...,F-1\}, l \in \{0,...,L-2\}\}$. Now the conceptors are ready for the next task to be learnt incrementally with CPR.

Even though the conceptors are ready to be used with CPR, the conceptor regularization loss Ω_C from previous section needs to be adapted to use the multiple filter conceptors rather than a single conceptor per layer. In the original regularization term the loss of each conceptor per layer is summed up. To adapt Ω_C to filter conceptors, the only change that needs to be made is to also sum over each filter conceptor, rather than each conceptor per layer. The resulting loss is the filter conceptor regularization loss Ω_F, which becomes $\Omega_F(W_l^{t-1,f}, W_l^{t,f}, C_l^{t-1,f}) = \mathbb{E}[||C_l^{t-1,f}(g_l(W_l^{t,f}\xi_{l-1}^f) - g_l(W_l^{t-1,f}\xi_{l-1}^f))||_2^2]$. Here, the weights of a filter f for layer l on previous task $t-1$ and current task t are $W_l^{t-1,f}$ and $W_l^{t,f}$ respectively. The (updated) conceptor computed from the activation of the neurons in filter f of layer l which characterizes all previously learnt tasks is $C_l^{t-1,f}$. The conceptor regularization loss is thus the expected value of the L_2 norm of the conceptor-projected (difference in) pseudo rehearsal. As in CPR, Gaussian pseudo-data is passed through the new and old weights with activation function g_l, yielding new and old pseudo feature maps. In this case, the activation function g is ReLU for each but the last layer $\forall l \in 0,...,L-2$. The pseudo-data is shaped like the input to said weights, which is equal to the output of previous layer $l-1$. Hence the Gaussian pseudo-data input to a filter f is ξ_{l-1}^f.

Taking the difference of the pseudo rehearsal by subtracting the old pseudo feature map from the new pseudo feature map as done with $(g_l(W_l^{t,f}\xi_{l-1}^f) - g_l(W_l^{t-1,f}\xi_{l-1}^f))$ offers a measure of change in the new weights $W_l^{t,f}$ from the old weights $W_l^{t-1,f}$. By multiplying this measure of change with the

associated conceptor that characterizes previously learnt tasks $C_l^{t-1,f}$ it is projected into conceptor space. This is similar to how this operation was described previously from Hofman's work, except that the operations are done on a per filter/feature map basis, rather than concatenating the feature maps to perform the operation on a per layer basis. Once again, projecting this difference onto conceptor space means differences in directions where the singular values of the conceptor are small tend to zero, while differences in directions with large singular values are magnified. In other words, changes in the new filter weights which project onto previously learnt tasks (characterized by large singular values) that would interfere with previously learnt tasks are magnified, and changes which would not interfere are orthogonal to previously learnt tasks (characterized by small singular values). The L_2 norm of this projection then discourages changes that interfere with previously learnt tasks, while tending to zero for changes that do not interfere with previously learnt tasks. The use of filter conceptors prevents unused memory space from appearing where none exists (e.g. in linear combinations of neurons across different filters).

Replacing Ω_C with Ω_F, Eq. 7 can be rewritten as

$$\mathcal{J}^{t\neq 1} = \mathcal{L}_{task}(a_{L-1}, y^t) + \lambda \sum_{l=0}^{L-1} ||W_l^{inc}||_{fro}^2 + \gamma \sum_{l=0}^{L-2} \sum_{f=0}^{F-1} \Omega_F(W_l^{t-1,f}, W_l^{t,f}, C_l^{t-1,f}) \tag{8}$$

Note the additional summation over each filter in the filter conceptor loss. Aside from this additional summation and Ω_F replacing Ω_C the objective $\mathcal{J}^{t\neq 1}$ is identical to previous section and is described in more detail there. This small change allows filter conceptors to be used with CPR, shorthand fCPR. CPR can become computationally prohibitive as the inverse of an n by n autocorrelation matrix has to be taken to construct the conceptor for each layer. n in the case of CPR is equal to the number of neurons in a layer, so layers with more than approximately 10000 will not yield an inverse of the autocorrelation matrix within reasonable time. With fCPR the dimensions of the filter are equal to the length of the flattened filters, so if a filter has a k by k kernel only the inverse of a k^2 by k^2 needs to be computed. As k is typically below 100, fCPR can be used with any sized convolutional network. fCPR is our first extension of CPR, and is also the basis for our second extension.

3.2 Hierarchical Conceptors

Where computing a single conceptor per convolutional layer may suggest unused memory space where there is none, there are still reasons why having information across filters can be beneficial. For example, two or more filters strongly correlating may hint at a redundancy in learnt filters, where only some (linear combinations of) filters need to be kept and the others can be dedicated towards new tasks. Additionally, identifying unused (linear combinations of) filters would allow us to learn new tasks only on unused combinations of filters, preventing changes in combinations of filters important to previously learnt tasks. But because filter conceptors pertain only to single filters, these potential benefits are unattainable with fCPR.

As filters form a natural hierarchy with convolutional layers in a convolutional network, with each layer consisting of multiple filters and each filter consisting of multiple neurons, one way of conceptualizing filters is to abstractly view filters to a layer as neurons are to a filter. In this analogy, the multiple filters make the layer as the multiple neurons make the filter. Just as we want to identify unused combinations of weights in a filter to train the next task on, we similarly want to identify unused combinations of filters in a layer to train the next task on. With filter conceptors we characterized this unused memory space at the level of individual neurons in a filter. Using tFhe analogy we might be able to once more adapt CPR to characterize unused memory space, this time at the level of individual filters in a layer.

The primary obstacle in making this adaptation is that there is no good analogy to neural activation for filters. While neural activation is a single number per neuron, each filter produces a many valued feature map. Collecting activation to compute the autocorrelation matrix, which is the basis for the singular values of the conceptor, is not straightforward. Concatenating feature maps results in a conceptor identical to those used in [17]. Instead we would like a single valued analogy to neuron activation for filters, which still contains information of how the filter output relates to current and previously learnt tasks.

To compute a filter conceptor, we take the autocorrelation of the activation matrix. This activation matrix of $S*V$ rows by k^2 columns is constructed from the activation of the k^2 individual neurons in the filter in response to S samples with V input patches of training data. The conceptor C that can then be obtained through this activation matrix is k^2 by k^2 shaped. Note how C is constructed from output patches p before they are combined into the full feature map output of the convolutional layer. We can then flatten p, which becomes the flattened vector of the output patch (activation) $\vec{p}$. $\vec{p}$ can be projected onto the conceptor space, yielding the vector $\vec{p}_c = C*\vec{p}$. Here $\vec{p}$ is explicitly denoted as a vector since it is the *flattened* output patch. This projection is an indication of how much the activation falls into used (dimensions represented by large singular values) or unused (smaller singular values) memory space, as characterized by C. The dot product between $\vec{p}_c$ and the transpose of the original vector $\vec{p}^\top$ can be taken as an estimate of how much the original vector differs from its conceptor projection, as in $u = \vec{p}^\top * \vec{p}_c$. This can be shortened to $u = \vec{p}^\top C\vec{p}$.

If $\vec{p}$ resulted from a filter with weights trained for a given task t, and C is the filter conceptor characterizing the filter response to t, then u can be seen as a characteristic of the filter response $\vec{p}$ in relation to C. This way, u is now the single valued characteristic of the many valued filter response $\vec{p}$ in relation to C that we were looking for. We call this the *conceptor activation* going forward. The conceptor activation bridges the analogical gap between neuron activation and filter response. Similar to how we collected neuron activation to input sample patches, we can now also compute filter conceptor activation from the filter response to input sample patches. For all $S*V$ input patches and all F filters, we denote the flattened filter response $\vec{p}$ of any filter f and associated filter conceptor C_f to any such input patch as $\{u_f^v = \vec{p}_v^\top C_f \vec{p}_v | \forall v \in \{0,...,S*V-1\}, \forall f \in \{0,...,F-1\}\}$. This gives us the *conceptor activation matrix* X_c with $S*V$ rows and F columns filled as follows,

$$X_c = \begin{pmatrix} u_0^0 & u_1^0 & \cdots & u_f^0 \\ u_0^1 & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ u_0^v & \cdots & \cdots & u_f^v \end{pmatrix} \tag{9}$$

where u_f^v is the conceptor activation of conceptor C_f to flattened filter response $\vec{p}_v$. Notice how the conceptor activation matrix of Eq. (9) is similarly shaped to the activation matrix seen in CPR and fCPR. The rows still correspond to individual samples of activation. The columns now correspond to the amount of neurons/filters over which we want to compute a conceptor.

Having constructed an activation matrix X_c, we can use this to once again compute an autocorrelation matrix R_c as before: $R_c = X_c X_c^\top / (S * V)$. From R_c we can compute another conceptor, but note how this conceptor was computed not from neural activation but from an "activation" involving F other conceptors: from the conceptor activation. To avoid confusion, conceptors computed from conceptor activation in this manner shall be called hierarchical conceptors going forward and denoted H as opposed to C. The hierarchical conceptor computed from the conceptor activation of the F filter conceptors $\{C_l^{t,f} | \forall f \in \{0, ..., F-1\}\}$ in layer l after training on task t is then denoted as H_l^t.

This hierarchical conceptor can be computed, updated, and used alongside the filter conceptors with CPR, although the order in which filter and hierarchical conceptors should be updated will be discussed in a dedicated subsection hereafter. As in fCPR, the conceptor regularization term Ω_C must be adapted to allow for this change in the conceptors. Similar to how the conceptor activation was collected for the autocorrelation, the same operation can be applied to the Gaussian pseudo data for the conceptor-based pseudo rehearsal with hierarchical conceptors. The operation in this case projects the flattened filter response to the pseudo data onto the filter conceptor space, of which the dot product is taken with the transposed flattened filter response.

It is the same $u = \vec{p}^\top C \vec{p}$ operation, where now during pseudo rehearsal $\vec{p}$ is the difference in flattened filter response d. During pseudo rehearsal d then becomes the difference in flattened filter response between the new and old filter to the Gaussian pseudo data, rather than input originating from actual training samples. In this case, it is the filter response of the filter associated with the filter conceptor C. This operation then yields a pseudo conceptor activation.

In other words, to allow for hierarchical conceptors in pseudo rehearsal the Gaussian pseudo data is passed through the filter and activation function using the new weights and the old weights, similar to fCPR. Their difference d is taken just as in fCPR. Unlike fCPR, the L_2 norm is not yet taken. Instead, the pseudo conceptor activation is computed for each (differenced) filter response patch in d.

The pseudo conceptor activation can then be projected onto the hierarchical conceptor space. This projection tends to zero when the changes in filter response to the pseudo data are in directions associated with small singular values of the hierarchical conceptor. Conversely, the projection be-

comes larger when changes in filter response to the pseudo data are in directions associated with large singular values of the hierarchical conceptor. By taking the L_2 norm of this projection we encourage changes in filters in directions associated with small singular values, while discouraging the converse. This way, hierarchical conceptors are used to shield combinations of filters in a layer similar to how filter conceptors are used to shield combinations of neurons in a filter. This adaptation to CPR will be referred to as hierarchical conceptor-based pseudo rehearsal, or hCPR for short.

Having illustrated the basic principles behind hCPR, we can formally change Ω_C into Ω_H to accept hierarchical conceptors. As Ω_H relies on both the filter conceptors and the hierarchical conceptor, this then results in $\Omega_H(W_l^{t-1,z}, W_l^{t,z}, H_l^{t-1}, C_l^{t-1,z}) = \mathbb{E}[||H_l^{t-1}\Phi_l||_2^2]$. In the left hand part of the formula, $z = \{0, ..., F-1\}$ such that z enumerates all F filters. $W_l^{t-1,z}$ and $W_l^{t,z}$ are the old $(t-1)$ and new (t) weights as before for layer l. Note how the weights of all filters in a layer are used, rather than only the weights of a single filter as in Ω_F. Next is H_l^{t-1}, which is the (updated) hierarchical conceptor computed from conceptor activation of filter conceptors in layer l after training on previous task $t-1$. Following the hierarchical conceptor is the set of all F updated filter conceptors (as enumerated by z) in layer l: $C_l^{t-1,z}$, also obtained after training on previous task $t-1$.

In the right hand side of the formula, a major part is substituted by Φ_l for readability. Here Φ_l is the differenced pseudo conceptor activation as exemplified previously. This differenced activation is projected onto hierarchical conceptor space where, as previously mentioned, differences along important dimensions to previously learnt tasks (directions with large singular values) are increased and those along unimportant dimensions (directions with small singular values) tend to zero. Taking the L_2 norm of this projection as the loss encourages changes in unimportant combinations of filters while discouraging the opposite.

For pseudo rehearsal with hierarchical conceptors, a (differenced) pseudo conceptor activation Φ_l has to be collected from Gaussian pseudo data. We use the same $u = \vec{p}^\top C\vec{p}$ operation described previously to obtain the conceptor activation. We write $\Phi_l = \{\vec{p}^\top C_l^{t-1,f}\vec{p} \mid \forall\vec{p} \in P, \forall f \in z\}$ where $P = \{g_l(W_l^{t,f}\xi_{l-1}^f) - g_l(W_l^{t-1,f}\xi_{l-1}^f) \mid \forall f \in z\}$. Here P is similar to the pseudo rehearsal seen in fCPR. To reiterate: it contains $S*V$ patches $\vec{p}$ which are the flattened and differenced filter response to the Gaussian pseudo data. Here S is the amount of samples of Gaussian pseudo data, and V the amount of input (and thus output) patches in a single sample. These flattened, differenced output patches $\vec{p}$ are the filter responses of each filter $\forall f \in z$ with the newly trained weights $W_l^{t,f}$ to Gaussian pseudo data input ξ_{l-1}^f passed through activation function g_l, subtracted by the same but with the weights from previous task $W_l^{t-1,f}$.

With this definition for hierarchical conceptor-based pseudo rehearsal, or hCPR, we can replace Ω_C with Ω_H in Eq. (7). This gives us

$$\mathcal{J}^{t\neq 1} = \mathcal{L}_{task}(a_{L-1}, y^t) + \lambda \sum_{l=0}^{L-1} ||W_l^{inc}||_{fro}^2 + \gamma \sum_{l=0}^{L-2} \Omega_H(W_l^{t-1,z}, W_l^{t,z}, H_l^{t-1}, C_l^{t-1,z}) \tag{10}$$

While most of the formula remains identical to Eq. (7), Ω_H requires an additional hierarchical con-

ceptor, and the weights and conceptor to be split up by filter into a set of F old weights, new weights and (filter) conceptors as enumerated by z. hCPR retains the benefits of fCPR, as it relies on the smaller filter conceptors rather than one large conceptor per layer. The addition of an F by F hierarchical conceptor computed with all F filter conceptors on a layer adds another generally small conceptor which is intended to overcome the limitations of fCPR. The hierarchical conceptor characterizes how (much) combinations of individual filters are used towards the learnt tasks, and can give a measure of available filter dimensions on which new tasks can be learnt across filters.

3.2.1 Updating Hierarchical Conceptors

Just as the conceptors computed per layer or per filter are updated with the previously learnt tasks to characterize not just the newly learnt task but all of the learnt tasks thus far, the hierarchical conceptor has to be updated as well. However, because a hierarchical conceptor H is computed from conceptor activation, which in turn relies on other conceptors C^z (where z indexes all conceptors used in the conceptor activation), the order in which H and C^z are updated matters. We identify three procedures by which the hierarchical and filter conceptors can be updated, though which of the three is the best procedure is not yet explored.

Let the F filter conceptors for layer l, indexed by $z = \{0, ..., F-1\}$, which represent the previously learnt task(s) $t-1$ be $C_l^{t-1,z}$. The hierarchical conceptor obtained with $C_l^{t-1,z}$ is then H_l^{t-1}. Let the filter conceptors obtained after training on the new task t be $C_l^{t,z}$. The goal is to update this new set of filter conceptors with the old set of filter conceptors using the $\vee$ operator such that $C_l^{t,z}$ characterizes both the new task and the previously learnt task(s). Furthermore, we want to obtain a hierarchical conceptor with similar characterization for filters.

The three procedures listed here can be summarized as follows: Filter first, which first updates the filter conceptors, computes a new hierarchical conceptor from the updated filter conceptors, and then updates previous hierarchical conceptor with the new one. Hierarchical replacement, which similarly updates the filter conceptors and computes a new hierarchical conceptor, but replaces the old hierarchical conceptor with the new one rather than updating it. And finally, parallel updates, which computes a new hierarchical conceptor from the new filter conceptors before updating the filter conceptors, and then updates both the old filter conceptors and the old hierarchical conceptor with the new ones. Each procedure is listed in more detail below, with additional remarks.

- Filter first:
  - Compute the filter conceptors for the newly learnt task: $C_l^{t,z}$
  - Update the new filter conceptors with those characterizing previously learnt tasks for neurons in a filter: $C_l^{t,z} \mapsto \{C_l^{t,f} \vee C_l^{t-1,f} \mid \forall f \in z\}$
  - Compute the new hierarchical conceptor H_l^t from the updated filter conceptors $C_l^{t,z}$ for the newly learnt task

  - Update the new hierarchical conceptor with the hierarchical conceptor characterizing previously learnt tasks for filters in a layer: $H_l^t \mapsto H_l^t \vee H_l^{t-1}$
    * Note how this adds characterizations of the previously learnt task(s) into the updated hierarchical conceptor twice: the old hierarchical conceptor H_l^{t-1} characterizes the previously learnt task(s) as it is computed from $C_l^{t-1,z}$, but the new hierarchical conceptor H_l^t through this procedure is computed from updated conceptor filters $C_l^{t,z}$ which by being updated with $C_l^{t-1,z}$ also characterize the previously learnt task(s)

- Hierarchical replacement:
  - Compute the filter conceptors for the newly learnt task: $C_l^{t,z}$
  - Update the new filter conceptors with those characterizing previously learnt tasks for neurons in a filter: $C_l^{t,z} \mapsto \{C_l^{t,f} \vee C_l^{t-1,f} \mid \forall f \in z\}$
  - Compute the new hierarchical conceptor H_l^t from the updated filter conceptors $C_l^{t,z}$ for the newly learnt task
  - With the new hierarchical conceptor now containing information on both previously and newly learnt tasks from the updated filter conceptors $C_l^{t,z}$, the old hierarchical conceptor H_l^{t-1} can be disregarded
    * How much information from previous tasks is retained in this way in higher level conceptors is currently unexplored.

- Parallel updates:
  - Compute the filter conceptors for the newly learnt task: $C_l^{t,z}$
  - Compute the new hierarchical conceptor H_l^t from the not yet updated filter conceptors $C_l^{t,z}$ for the newly learnt task
  - Update the new conceptors with the old conceptors in parallel: $C_l^{t,z} \mapsto \{C_l^{t,f} \vee C_l^{t-1,f} \mid \forall f \in z\}$ and $H_l^t \mapsto H_l^t \vee H_l^{t-1}$
    * This order for computing and adding new conceptors to those from previous task(s) is the one chosen for the rest of this thesis.

Of course, this is done for each (convolutional) layer $l \in \{0, ..., L-2\}$, excepting the last layer $L-1$ which is the model head and for which no conceptors are computed. After updating the hierarchical and filter conceptors they are ready to be used for training with hCPR on the next task. Updating conceptors is only done after training on the second task and beyond, as before that there are no previous conceptors to update with.

3.3 Constraints

The idea that linear combinations of weights in a filter can represent used or unused memory space in said filter is questionable in the first place. Due to weight sharing, the same weights are used multiple times to produce the resulting feature map. This repetition influences how changes in (linear combinations of) weights affect the output neural activation, distinguishing them from neural activation in fully connected networks where the weights of each neuron can be changed independent of other neural activation in the same layer. In a convolutional filter, a change in a single weight can affect the entirety of the output feature map, whereas in a fully connected layer a single weight will only affect a single output.

Regulating entire filters in a layer with hierarchical conceptors is, analogies aside, distinct from regulating weights of neurons in a fully connected layer. Dependant on deep learning architecture and design choices, one or more feature maps are summed or averaged as input to the next (convolutional) layer. The architecture used in this work simply averages all feature maps as the input to the next layer by default. The same is seen in Hofman's work [17], of which this work is a continuation. This may also affect to what degree unused filters can be identified and used towards new tasks.

Hierarchical conceptors and hCPR introduces an extension to CPR that adds Ω_H to the loss in the forward pass. In this work, backpropagation is handled by the Autograd tool included in PyTorch [29]. However, for a more complete understanding of hCPR and how to backpropagate the new loss, Ω_H should be differentiated. This is as of yet unexplored.

Finally, the idea of hierarchical conceptors stems from an analogy between neurons and filters. It should be noted however that it is as of yet unsure whether this analogy holds, and whether the so-called conceptor activation can be treated identically to neural activation. But even in the case where the analogy does not hold, hCPR introduces a novel characteristic for conceptors in the conceptor activation and lays the groundwork to hierarchies of a greater depth for feedforward networks.

4 Experimental Setup

In previous section we proposed computing conceptors from a characteristic of filter response rather than from the neural activation in an entire layer. Furthermore, we introduced a novel hierarchical conceptor to overcome loss of information from splitting the activation collection procedure. To test the efficacy of these two approaches, we perform experiments similar to the segmentation task in [17], using the same network architecture. As our implementation, optimization scheme and evaluation criteria might feature minute differences from previous work by Hofman, we also perform baseline experiments to test model capacity for learning given tasks jointly and incrementally with conceptor activation collected as in [17, 16, 25].

We first introduce the dataset used for all experiments. Secondly, we provide the structure for the experiments we perform, split up into core and supplementary experiments. The core experiments pertain to the performance of the model across three tasks with the following variations:

1. *Baseline* joint training
2. *Incremental* training without conceptor regularization
3. *Conceptor* incremental training with regularization as in [17]
4. *Filter* conceptors incremental training with fCPR, using Ω_F instead of Ω_C
5. *Hierarchical* conceptor incremental training with hCPR, changing Ω_C to Ω_H

Furthermore, to estimate learning capacity of the model and to get an upper bound in performance for any given task under ideal circumstances, some supplementary experiments are provided:

1. *Individual* class training: training and testing the model on only a single class, for each of the three classes used in the previous experiments
2. *Growing* of classes: training and testing the model on the first class, then the same for the first two classes, then for the first three, all the way up to training and testing the model on the full set of 27 classes

After this, we define the optimization scheme and further settings used for the experiments. Finally, we provide various criteria by which performance of the different experiments are measured.

4.1 Dataset

Figure 6: Example image from the Cityscapes dataset (top) and associated segmentation mask (bottom).

The dataset used for the experiments is the Cityscapes coarse semantic segmentation dataset [27].

- Contains 17045 coarsely annotated images.

- Over 30 distinct segmented categories.
- Overlapping and transparent regions are labelled as the foreground category.
- Images of German cityscapes of cities like Aachen, Bremen, Cologne, and more.
- Months in spring, summer and fall for consistent lighting.
- Exclusively taken during day time.
- Only features images with clear or overcast weather.

An example image with associated segmentation mask is shown in Fig. 6. Note how the segmented areas are distinct from each other and annotated areas rarely touch. Not all categories occur in each image, and some categories occur more frequently or consistently than others. The hood and logo of the car are segmented out as background.

The tasks performed in the experiments are, unless stated otherwise: roads, cars, buildings. The order in which tasks are presented can influence performance, for consistency and comparability between models the abovementioned order is adhered to unless mentioned differently. As the expected output for each task the pixels of its associated class are set to 1, and 0 everywhere else. To reduce computational costs, images and associated output are scaled to be 128 pixels wide by 64 pixels high. More example input and output images are provided in Appendix A.

4.2 Network Architecture

The network architecture used here is a simple fully convolutional neural network (FCN) with 8 layers, identical to the one used in [17]. Each convolutional operation keeps the dimensions of the input as-is, owing to a 5 by 5 filter kernel and a padding of 2. Instead, reducing and expanding the dimensions of the feature maps depends on max-pooling and bilinear upsampling respectively. This FCN follows the traditional U-Net architecture [30], without the skip-weight connections. Skip-weight connections can help prevent vanishing gradients in large networks, but this network is not deep enough for skip-weight connections to improve performance [17]. Further network details, such as the amount of filters per layer and where feature map dimensions are reduced or expanded, can be gleaned from the diagram in Fig. 7. A complete overview of settings used is provided in Table 2.

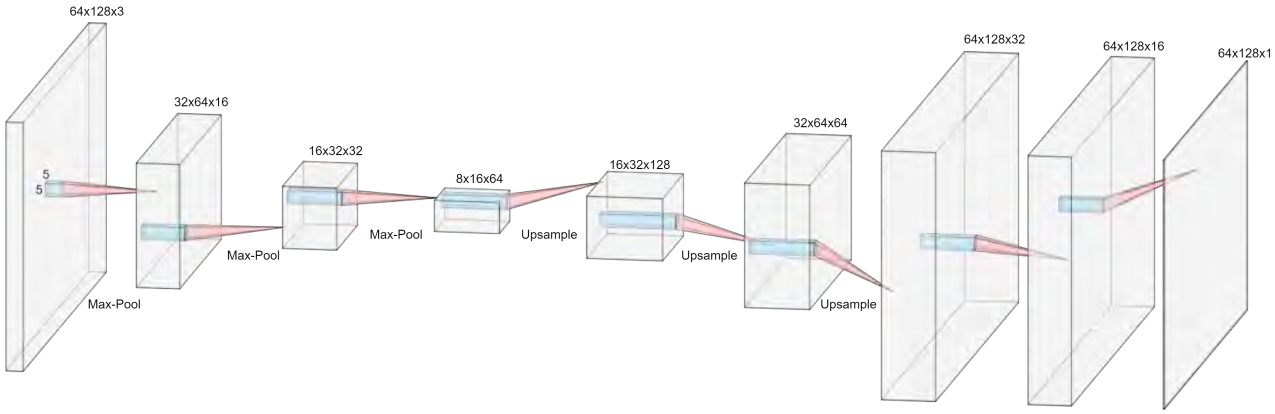


Figure 7: Network architecture with 8 convolutional layers, each with filters having a 5x5 kernel. Input is a scaled Cityscapes image, outputs a semantic segmentation of the input.

4.3 Experiment Design

The experiments are split up in core experiments and supplemental experiments. For each core experiment the settings and used software are listed in Table 2. The supplemental experiments are meant as a means of estimating what optimal model performance in the incremental parts of the core experiments would be, and to estimate model capacity outside of incremental training. The results are, unless stated otherwise, averaged over ten trials splitting the data into a 90%/10% train/test split. Each trial is treated as a fold in cross-validation, such that each test fold contains unique data.

4.3.1 Core

4.3.1.1 Baseline A model is trained on the three tasks/classes in a joint training fashion. This experiment serves to establish baseline performance of the model in joint training over three tasks. The experiment here is identical to the 3rd step of the *Growing* supplementary experiment.

4.3.1.2 Incremental A model is trained on the three tasks in an incremental fashion without any form of conceptor regularization. This experiment serves to establish the degree of catastrophic interference occurring in the model with the given (order of) tasks.

4.3.1.3 Conceptor A model is trained on the three tasks in an incremental fashion, with conceptor regularization implemented as described by Hofman [17]. However, different from [17] the network is trained with a different optimizer (RAdam [31] instead of Adam [32]) and for more epochs, as described later on in this section.

4.3.1.4 Filter Model is trained on the three tasks in an incremental fashion, with conceptor regularization on conceptors computed from activation in individual filters, rather than over all filters in a layer. This experiment serves to establish if the information lost by splitting the activation collection to individual filters and computing smaller conceptors is detrimental to model performance and task retention.

4.3.1.5 Hierarchical A model is trained on the three tasks in an incremental fashion, with conceptor regularization on the hierarchical conceptor computed per layer over the individual filter conceptors in each respective layer. Given limitations imposed by the implementation, we compute the hierarchical conceptor using only 10,000 random input patches from the total collected activation (see Table 2). Similarly, the batch size for the pseudo-data in the loss is set to 1, as opposed to the batch size used otherwise. Though this should be sufficient as the hierarchical conceptor is far smaller than conceptors computed from entire layers of activation, it may lead to some more random variation in the results. This experiment serves to establish the validity and efficacy of hierarchical conceptors, and to assert the degree to which it remedies potential drops in performance in previous experiment if any are present.

4.3.2 Supplementary

4.3.2.1 Individual Three models are trained and evaluated on only a single class, for each of the three classes used in all aforementioned experiments. This experiment establishes what the optimal performance of the model on a given task would be with the settings and optimization scheme used. Attaining this level of performance is not necessarily realistic even in joint training with more than one class, but serves as an upper bound on performance.

4.3.2.2 Growing Models are trained and evaluated on the first t classes, $\forall_t \in \{1, ..., T\}$. This yields $T = 27$ results, and an upper bound of performance of the model under joint training of 1 to T tasks. It also shows at which point, if at all, the capacity of the model to learn more tasks is reached.

4.4 Optimization Scheme

While the *Baseline* and *Conceptor* experiment share model architecture and experiment design with [17], results will still vary due to a difference in optimization scheme. We have opted for the Rectified Adam (RAdam) optimizer, which may adapt to the extreme conceptor loss at the start of a new task better than Adam. Furthermore, Hofman trains his model for 5 epochs, but we found the model to still improve beyond that. Instead we train the model for 25 epochs starting with a moderate learning rate, which is generally enough for both training loss and task performance to plateau.

In tasks beyond the first, when training with conceptor regularization, we observed the training loss going up within the alotted 25 epochs. This points at a numerical instability. Common procedure is to lower the learning rate, allowing the optimizer to make more precise changes to the gradient. If the training loss in any given epoch is higher than that in the previous, the epoch is restarted and the learning rate halved. However, doing this five times slows down training to the point where no significant change is made within reasonable time. Thus we stop the training procedure after restarting five times.

We are also using a different task training loss $\mathcal{L}_{task}$ from [17], namely the IoU loss or Jaccard Index [28]. It can be made differentiable as shown in [33]. Using this loss makes sense given the Intersection-over-Union (IoU) evaluation criterion (more on that in the next subsection). It would also allow the network to predict multiple labels for overlapping classes in the image, as argued in [34]. Lastly, it discourages only outputting 0 for sparse classes compared to pixel accuracy, and it discourages making large mistakes compared to a loss based on the Dice Coefficient/F-score.

4.5 Evaluation Criteria

As mentioned previously, the primary evaluation criterion of model performance is given as IoU (for a single task), or average IoU over tasks (mIoU). To identify what is causing performance to drop after learning only three segmentation tasks as in [17], more involved evaluation methods are in order.

For a first indication as to what the model is or is not learning, we can look directly at the (changes in) filters the incremental model has learnt after each epoch or task. Comparing this qualitatively to previously learnt filters, or to those learnt in the jointly trained model, provides an idea of what or what degree of change is occurring in the filters with conceptor regularization.

Another evaluation method revolves around the singular values of the conceptors. Plotting these as a singular value spectrum gives an overview of the memory space occupied by each conceptor. More singular values close to 1 means that more linear combinations of weights contribute towards the task(s) represented by said conceptor. In other words more dimensions of memory space are occupied. Conversely, singular values close to 0 indicate capacity for the network to learn new tasks on the corresponding linear combinations of weights. This metric can be quantified as the quota of a conceptor $q(C) = \sum_{i=0}^{N} s_i/N$ for the N singular values from which C was constructed, which tends to 1 when all dimensions are completely used towards characterizing the learnt task(s) and no more unused memory space is available.

Table 2: Settings and Experiment Design Choices

Implementation specifics	**Value**
Layers	8
Kernel size	5x5
Stride	1
Padding	2
Dilation	1
Grouping	1
Optimizer	RAdam [31]
Learning rate	0.0001
Weight decay (λ)	$1e^{-7}$
Conceptor loss decay (γ)	1.0
Conceptor aperture (α)	4.0
Task loss	IoU loss [28]
Task order	Roads, Cars, Buildings
Batch size	15
Activation collection size	30*
Cross validation folds	10
Python version [35]	3.9.7
Pytorch version [29]	1.11.0

*: Number of batches used towards activation collection for computing R in conceptors. The final collection features *batch size * activation collection size* images

5 Results

As described in the Experimental Setup section, the experiments are broken down into two categories: core and supplementary. The core results are compacted into a single table. We also provide some example graphs of how training loss may change over epochs. After presenting the results from all experiments, we look at some of the singular value spectrograms. We also compare the conceptor quota over tasks, which shows how much extra memory space is occupied by each subsequent task. Finally, we provide some model output for visual inspection of model performance.

5.1 Core

Table 3: Results Comparison Different Models

Experiment	**Scheme**	**Task IoU**			**Average (mIoU)**
		1	***2***	***3***	
Baseline*	Joint	0.913±0.001	0.827±0.001	0.754±0.001	0.831
Incremental	Incremental	0.300±0.001	0.423±0.001	0.748±0.001	0.490
Conceptor	Incremental	0.885±0.011	0.619±0.013	0.509±0.004	0.671
Filter	Incremental	0.876±0.021	0.556±0.047	0.510±0.003	0.647
Hierarchical	Incremental	0.827±0.019	0.711±0.009	0.525±0.013	0.687

*: Uses BCE for task loss and a learning rate of 0.001

The results of our core experiments are shown in Table 3. In this table, the performance of the model in each experiment after training on all three tasks (either incrementally or jointly) are shown per individual task and as an averaged mean IoU over the three tasks.

Unsurprisingly, the best performance is by the jointly trained model. These results show that with the current architecture and settings, a mIoU of 0.826 is attainable when training on the three tasks. This would be the ideal result for incremental training. More notable is the IoU of the jointly trained model per individual task. Of the three tasks learnt, the first task (roads) appear to be the easiest to learn while the buildings are the hardest to learn. The implication here is that task performance on task three is likely to always be lower than that on task 2, which is in turn lower than that on task 1.

Another unsurprising result comes from the Incremental experiment. This is a clear case of catastrophic interference, where task performance on previously learnt tasks get worse as new tasks are learnt. Somewhat interestingly, the performance on the third and final tasks appears to be lower in this experiment than in the jointly trained experiment, even though it had to learn only one task rather than three. In the Incremental experiment the model started learning task three with weights from having learnt task 2, which may be detrimental to task performance on task 3.

As for the three experiments with conceptor regularization, none of the experiments have matched performance with joint training. The Conceptor experiment, which follows the activation collec-

tion scheme seen in previous work [17, 16, 25], does outperform the similar experiment performed in [17]. This confirms that the model stood to benefit from additional training epochs. Fig. 8 shows one full training run (25 epochs) on task 3 from the Hierarchical experiment.

Conceptor regularization using hierarchical conceptors yields the highest average IoU of the experiments with conceptor regularization. While the average is very close to that of the Conceptor experiment, the way the average breaks down is different. The Hierarchical experiment has worse performance on the first task, but better performance on subsequent tasks. This seems to suggest that using hierarchical conceptors yields a better balance between learning new tasks and remembering old tasks than other conceptors, even with the same aperture and other parameters.

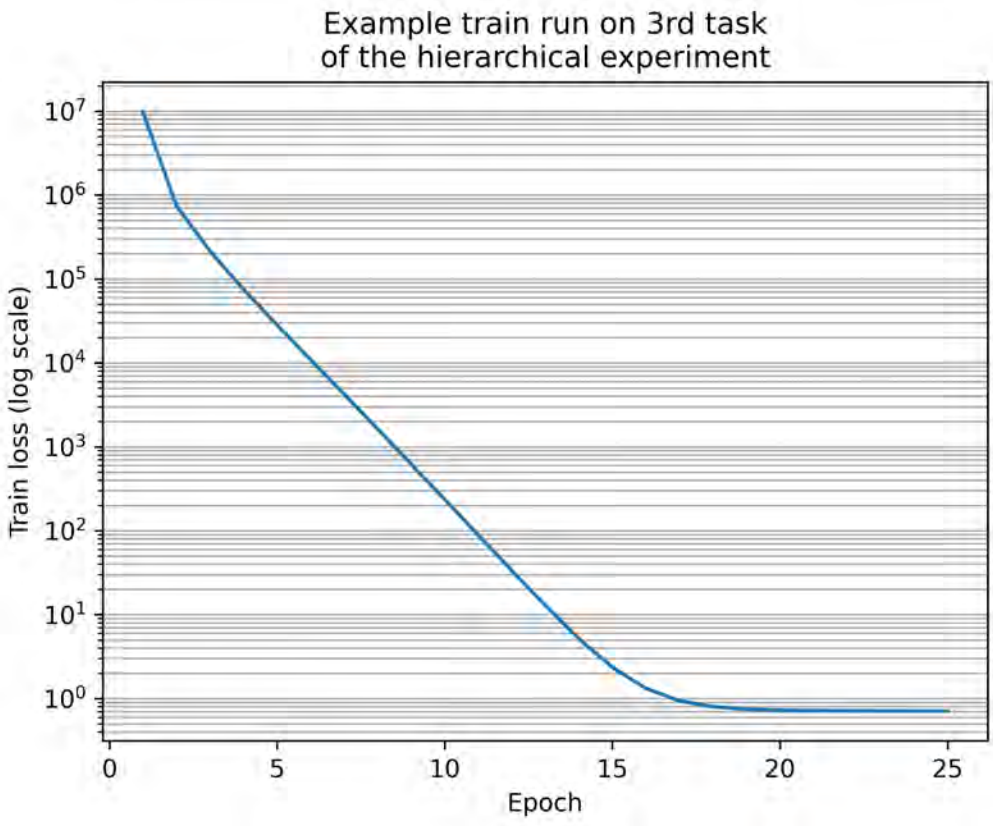


Figure 8: An example training run with the training loss shown on a logarithmic scale for each epoch. This particular run was from the first fold in the hierarchical experiment, training on the third task.

The results shown in Table 3 only show part of the results of each model. For a more complete overview we would like to refer the reader to Appendix B: Additional Results. This includes maxi-

mum IoU performance over all folds, average amount of epochs trained, and the performance after training on the first and second task.

5.2 Supplementary

5.2.1 Individual

Table 4 contains the results for training the model on one class at a time, for the three classes used in the Core experiments. What can be seen is that the IoU is generally a bit higher when training on a single class, except for the Roads class.

Table 4: Results for the Individual supplementary experiment

Experiment	**Scheme**	**Class IoU**		
		Roads	***Cars***	***Buildings***
Individual	Joint	0.915±0.001	0.842±0.001	0.757±0.001

5.2.2 Growing

In this experiment 27 models were trained, each model being trained jointly on one more class than its predecessor. Note that these are not instances of incremental learning, but instances of joint training where the same experiment is repeated multiple times with an increasing number of classes. The full results are shown in Figure 9. Shown are the IoUs for each class, for each model. The model is indicated by the amount of classes it had to learn on the x-axis. We see that performance is not limited by model capacity at any point. Had the model been limited, some or all of the performances on previously encountered classes would drop as the model has to learn more classes. Instead the performance per class is a straight line, in other words the performance on a given class is stable whether the model only has to learn a few select classes or all 27 classes.

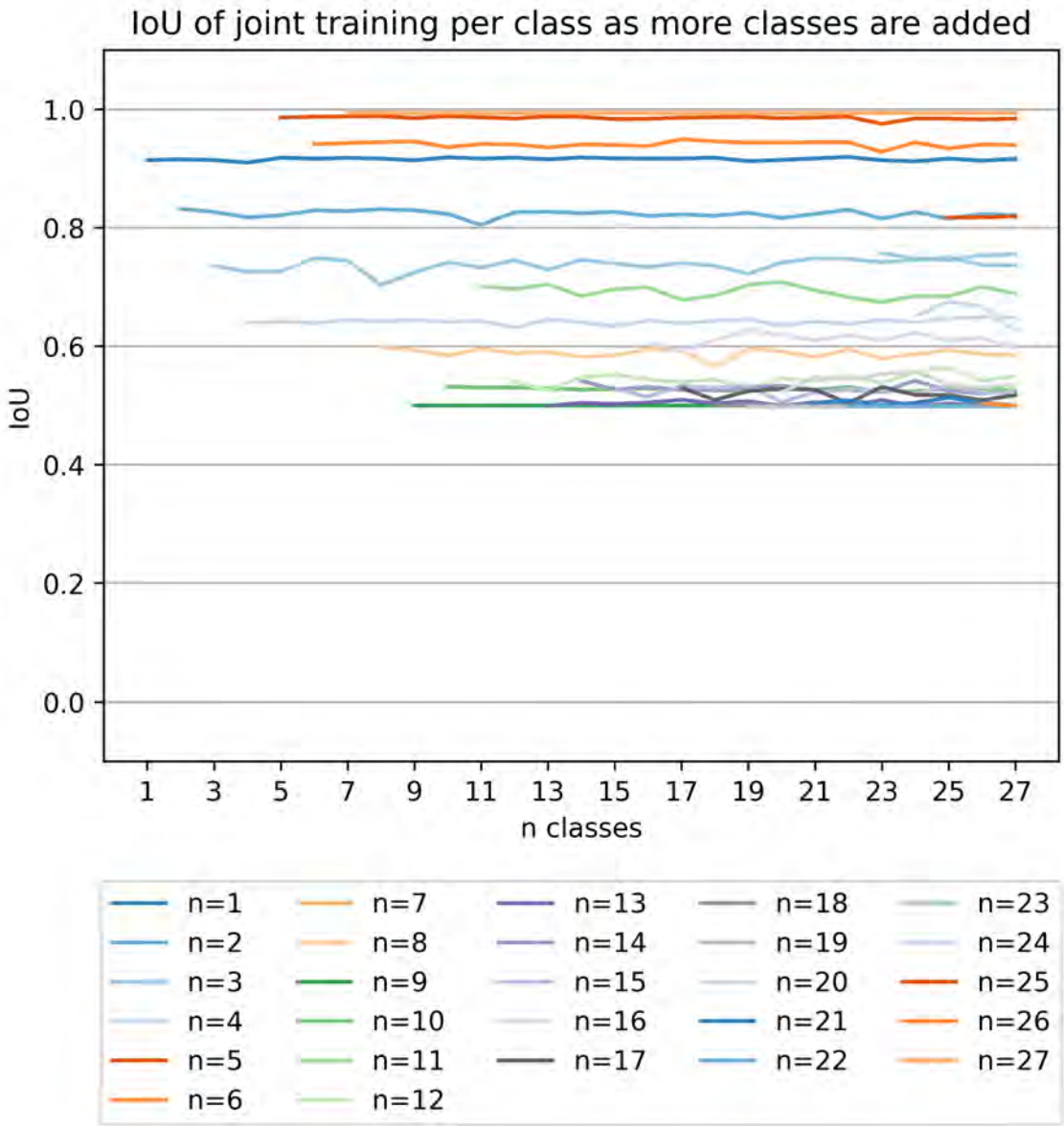


Figure 9: An example training run with the training loss shown on a logarithmic scale for each epoch. This particular run was from the first fold in the hierarchical experiment, training on the third task.

5.3 Singular Value Spectra

Looking at the singular values of the conceptors can tell us how much memory capacity of the network is used towards the learnt tasks. The less visual but more exact quota q is a measure of the same. Only some of the spectrograms are shown for the sake of efficiency. For each spectrogram that is shown, the spectrogram as it was at the end of training for each task is overlaid in different colors. For most tasks the spectrogram changes so little as to not be noticeably different from task to task. Two spectrograms are shown for the Conceptor and Hierarchical experiment. The Filter experiment featured too many small conceptors for visualization, so for that experiment we provide a bar chart of the quota instead.

Layer conceptors

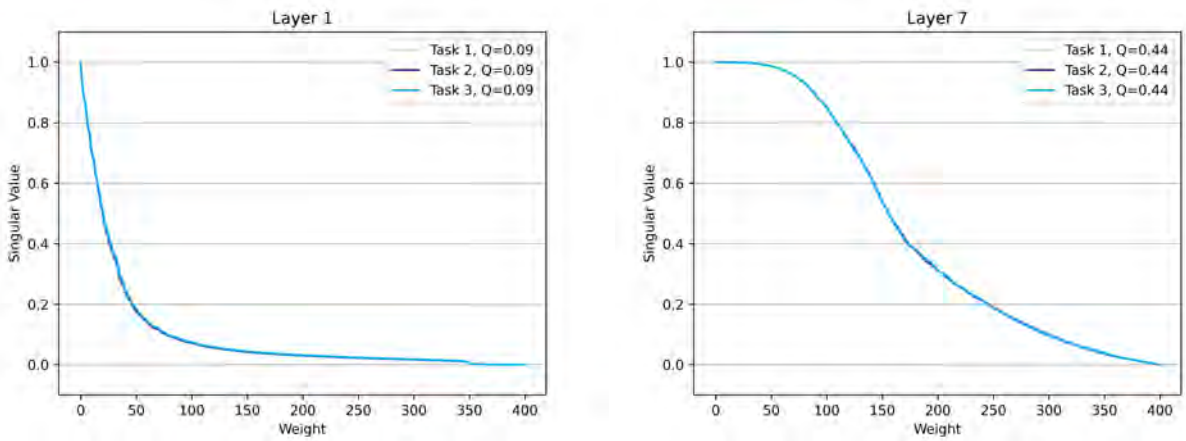


Figure 10: Singular value spectrogram of the first and last layer conceptors from the first fold of the Conceptor experiment, after training incrementally on all three tasks.

The spectrograms for the first and seventh (the last layer on which conceptors are computed) layers for the Conceptor experiment are shown in Fig. 10. These follow closely with similar spectrograms seen in previous work [17]. The difference over tasks is completely unapparent, and the difference in q values is insignificant. With also no observed differences in the other layers, all of the learning for each new task is likely occurring at the task head.

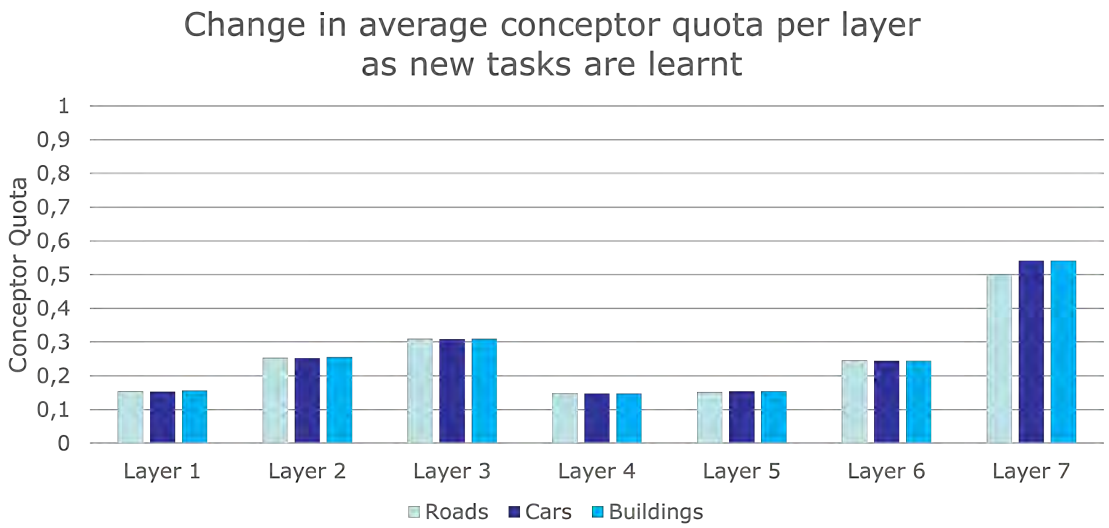


Figure 11: Average conceptor quota across all filter conceptors in a given layer at the end of the listed task.

While the many small spectrograms associated with filter conceptors do not lend themselves well to visual inspection, we can still look at the average quota. The bars shown in Fig. 11 correspond with the average quota for each layer at the end of training on each of the given tasks. Again there is little observable difference. However, this time the seventh layer appears to change somewhat from task 1 (roads) to task 2. Notice also how the quota is a little higher than that from the Conceptor experiment.

Hierarchical conceptors

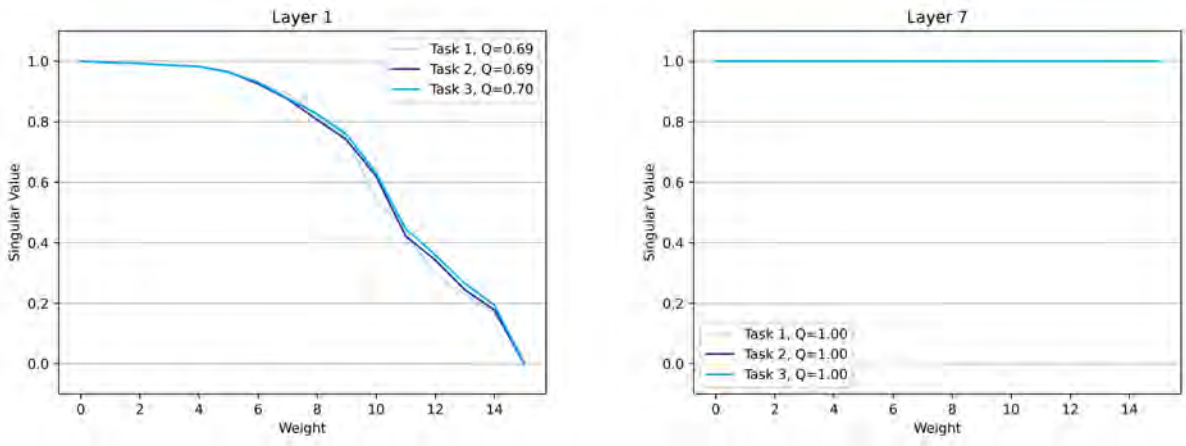


Figure 12: Singular value spectrogram of the first and last hierarchical conceptors from the first fold of the Hierarchical experiment, after training incrementally on all three tasks.

The spectrograms for the first (the input layer) and seventh (the last layer on which conceptors are computed) layers for the Hierarchical experiment are shown in 12. These are radically different from the ones shown before. This difference may stem from the fact that these are the singular values for hierarchical conceptors, constructed from conceptor activation and not network activation. The singular values in the seventh layer are all 1, but what this means for network capacity is as of yet unknown. It appears to indicate that each filter conceptor in the seventh layer contributes fully to the conceptor activation from which the hierarchical conceptor is constructed.

The entire set of spectrograms for both the Layer experiment and Hierarchical experiment are given in Appendix C: Singular Value Spectra. The graphs are obtained after training on all three tasks incrementally, on the first fold of the training data.

5.4 Model Output

Figure 13: Expected output (left) and model output (right) overlaid on the corresponding input image. The segmented areas for roads, cars, and buildings are given in red, blue and yellow respectively.

An example of expected and predicted output is given in Fig. 13. More examples are listed under Appendix D: Model Output Examples, for the models from the Joint, Layer, Filter and Hierarchical experiments. Shown here is a good combination of a road, some cars, and the peculiarities of the building class. While at first it may seem as though there should be buildings labelled in the ground truth on the right of the road, upon closer inspection the observant reader will spot some trees in front of the building. Looking at the model output, it is clear that the model struggles with this and generally segments the upper half of the screen as buildings. Most of the road and the cars are still segmented out to some degree, even after training on three tasks.

6 Discussion

In this thesis work we set out to continue earlier thesis work by Hofman [17], with the goal of identifying why network performance drops after learning only three segmentation tasks in a One-vs-all incremental manner. To aid in this investigation we explore a variation on how the activation for computing the conceptors used in conceptor regularization is taken, and introduce a novel hierarchical conceptor. We additionally perform experiments to gain a better understanding of what optimal performance would be under ideal circumstances.

Comparing the experiments with the joint training scheme to those with incremental training it is obvious that there is still a significant gap in performance. Incremental learning of segmentation with any version CPR explored in this work performs far below joint training on the same classes. Interestingly, the per-class performance in joint training does not depend on the amount of classes learnt. It appears that the model has a limit as to how well it can learn each class, which varies from class to class. This is likely a product of several things: of the architecture used, of the imbalance between classes and sparsity in classes, and of the quality of the annotations.

Following our findings, we identify some potential explanations as to what is causing the aforementioned drop in performance over tasks. Where possible, we propose a solution as to how each obstacle may be overcome. Furthermore, we discuss the results of our two proposed adaptations to conceptor regularization: Firstly, that of collecting activation from individual filters rather than entire convolutional layers; Secondly, that of using hierarchical conceptors. Finally, we disclaim the major limitations to our work, and propose future work that we believe holds potential in light of what is discussed.

6.1 Constraints of Conceptor Regularization

6.1.1 Bad Initialization

The operations of CPR are such that the initial conceptor regularization term is exceedingly high. For a network of the size used in our experiments, and with the pseudo-data used, we have observed a value of over half a billion in the first steps of the second task. The conceptor regularization term is normalized with regards to the amount of pseudo-data samples used to compute the regularization term. However, it still scales with the size of the network and the (conceptor projected) differences between the old and new weights. As this conceptor regularization term invariably dominates the objective during the first few epochs, this moves the new weights in a direction independent of the new task to be learnt. The result is a potentially poor initial starting position for the weights once the CPR term does become small enough for the task loss to become significant. Intuitively this is less of a problem for smaller networks with smaller input sizes, as the initial CPR term will be much lower. This could explain why this problem was not observed in the non-segmentation experiments performed on MNIST in [17] and [16]. Another cause for this discrepancy could be

that convolutional filter weights are influenced more heavily than fully connected weights due to weight sharing.

A trivial way for the weights to be changed to minimize the conceptor loss is to set W^{inc} to zero. If the new weights do not change from the old weights, the conceptor loss will be zero. In Figure 10 we show that W^{inc} beyond the first tasks are indeed likely set to zero: the singular value spectrogram does not change beyond the first task. The conceptors do not change from task to task, which means that the network stays the same at least for those layers regulated by the conceptors. It is likely that the high conceptor loss pushes W^{inc} to zero as it is not being informed by the task loss until several epochs into training. This would also explain the decrease in train loss seen in Figure 8: it is simply trying to set W^{inc} to zero as fast as the learning rate (and the optimizer RAdam) allows.

If the high initial conceptor loss is indeed what is preventing the model in our experiments from learning beyond the first few tasks, there are a couple of avenues that may alleviate this problem. Most straightforward, a way to scale or average the CPR term such that it is no longer unbounded might provide a simple solution. Pre-training each task with just the task loss before adding conceptor regularization might put the weights in a better position for the new task, although we are unsure how this would impact performance on previously learnt tasks. Finally, if the conceptor regularization still affects the weights too much and prevents the gradient from being informed by the task loss, alternating between one or more optimization steps with the task loss and the conceptor regularization term may allow both to perform gradient updates on the network.

6.1.2 Filter changes

As anyone who has owned an aquarium will be able to tell you, regular filter changes are absolutely necessary. So too with convolutional networks. However, once a (set of) filter(s) have been learnt for a specific task, there is no straightforward way to revisit the learnt filters. A new task may adapt or use the learnt filter, but the optimizer may not find a transition from the old filter to a more general filter that benefits both learnt and new tasks. While this more general solution may exist, conceptor regularization might constrain movement in that direction due to an increase in the conceptor regularization term before the solution is found. This could explain why we see few to no changes in many of the filters, as shown in the results.

6.1.3 Illusory Overcapacity

Hofman suggested the idea that the remaining memory space of the model after learning two tasks may not contain a satisfying solution for the third task. However, the reported quota of the conceptors appear to suggest an overcapacity if anything. Not a single layer is at half of its capacity, with the average quota being below 30%. Furthermore, every conceptor, excepting maybe that of the first layer, have singular values of which some if not most are close to 0. These are directions in the memory space of the layer where changes should not influence the previously learnt task(s) at all! How then can it be that no solution exists within these directions of unused memory space?

The first layer has few directions in memory space for which the singular values are approximately 0 (weights which are completely unused). It could be, however unlikely, that the first layer contains filters that all prevent the network from learning the new task without changes that interfere with previously learnt tasks. Given the nature of weight sharing in convolutional filters, a single weight change influences the entire feature map associated with the filter in question. And as established, there are few weights available in the first layer which are completely unused. Each dimension in memory space contributes to the previously learnt tasks in at least some small way, excepting a few individual weights.

Alternatively it may be that the unused capacity of the network is overestimated. After all, the quota is calculated based on the singular values of the autocorrelation matrix R of activation samples X in an entire layer. But what does it mean for the activation of neurons in different filters to strongly or weakly correlate? Each filter outputs its own feature map, with no interaction with weights in other filters of the same layer. This interaction only becomes meaningful in subsequent layers which take combined inputs from multiple feature maps. Activation collection is only done per layer however, so this information is not captured in conceptors computed per layer. To get a closer estimate of the remaining capacity of an FCN we should thus look at the autocorrelation on a per filter basis, rather than a per weight basis.

6.2 Filter Response

One core finding from our experiment on computing conceptors for filters rather than layers is that performance did not suffer from a loss in information. Conceptors computed per filter rather than per layer do not contain information on correlations between neurons in different filters, but this did not affect performance. As mentioned previously, this makes sense as each filter outputs its own feature map with no information shared between weights in different filters. As a similar performance to using a single conceptor per layer (layer conceptors) can be attained, the smaller and more computationally and memory efficient conceptors per filter (filter conceptors) should be preferred for convolutional network layers.

Looking at the singular value spectrum for each filter conceptor tells a different story from that told by the singular values of the layer conceptors. Excepting filters with only weights close to 0 (unused filters), the singular values confirm that all directions in memory space contribute to a significant degree to the previously learnt task(s). This suggests that the capacity for the network to learn new tasks may be far more limited than would inferred from the singular values of the layer conceptors.

Alternatively it might be that aperture is too high and that the network is regulated too strictly. All of the CPR variants perform very well in preventing catastrophic interference with the first segmentation task. Table 3 shows that even after learning three tasks, the IoU on the first task is still far above that of the other two. Reducing the aperture may improve performance in later tasks. However, this would likely come at the cost of performance in the first task. Hofman explored

various settings for aperture [17], and found that apertures below 100 already lead to a decrease in performance. This makes it unlikely that the aperture is too high; if anything, it is too low.

6.3 Hierarchical Conceptors

Having constructed filter conceptors, a hierarchical conceptor can be defined for that layer. This provides a much smaller, hierarchical conceptor-based characterization of the activation patterns in an entire layer. This allows for conceptors to be used where otherwise constructing a single large conceptor may be too computationally expensive (when using more than 10,000 weights). We find that the best performance using hierarchical conceptors in CPR matches that of non-hierarchical conceptors, but also observe a larger variation in performance as indicated by the higher standard error.

Looking at the singular values again, those of the hierarchical conceptors paint a different picture from those seen in the other experiments. Far more singular values are close to 1, and most of those that are not are close to 0. The latter can be explained by unused filters whose weights are close to 0. That so few singular values are somewhere between 0 and 1, as opposed to those of non-hierarchical conceptors, may have several explanations. It may be that the aperture is too high, as it is applied once in its constituent conceptors and once in constructing the higher level conceptor. Alternatively, it may be that this is a consequence of how the conceptor activation is computed: it would mean that each constituent conceptor with singular values above 0 contributes fully to the conceptor activation.

It should be noted that, while the hierarchical conceptors do appear to perform well, this may simply be because Ω_H partially contains Ω_F. What exactly the conceptor activation used in computing the hierarchical conceptor and Ω_H represents is still largely unexplored. The intuition is that it characterizes some information between its constituent lower level conceptors. Whether this means that a hierarchical conceptor H resulting from filter conceptors C can regularize individual filters as the lowest level conceptor can regularize individual weights is not as of yet known.

Finally, the model regulated with hierarchical conceptors took more epochs before having to restart. This may seem to indicate that it is a more stable loss, yet likely it owes to the fact that Ω_H is a harder loss to optimize than Ω with non-hierarchical conceptors. The training loss simply took longer to reach the same point at which the other, non-hierarchical conceptor regulated experiments had to restart.

6.4 Limitations

On top of the previously mentioned constraints on conceptor regularization, perhaps the biggest problem of all is that the loss appears to be numerically unstable. As W^{inc}, and thus the conceptor regularization term, approaches zero (typically after 15+ epochs), eventually the conceptor loss will increase again. This means that the learning rate had to be lowered to the point where the network no longer learns in feasible time. Even if the network would (given enough time) actually be able to learn further tasks perfectly, we will not be able to verify this with the optimization techniques used.

This puts a large disclaimer on our findings for filter conceptors and hierarchical conceptors, as the results might actually differ from our findings would the model be allowed to converge. Perhaps optimization techniques which compute second derivatives, such as L-BFGS [36] would be able to overcome the observed numerical instability.

Additionally, as is often the case with thesis work, the time constraints meant we were unable to explore the many hyperparameters involved and predominantly used those listed in Table 2. As Hofman suggested it may very well be that a network with more capacity in additional filters and/or layers could be a simple solution for learning more tasks. Our findings with filter conceptors may indicate that the network capacity is indeed the bottleneck. Perhaps incremental learning is simply more memory inefficient with regards to network capacity, as the network does perform well in joint training.

6.5 Future Work

As shown in Section 2: Related Work, it is not always possible to compare methods across different works simply due to differences in design choices and experimental setup. As such it is difficult to compare our work to the other approaches we previously outlined, and to decide on what counts as the state-of-the-art. For a better comparison, performing experiments similar to those seen in previously seen work such as the step-wise incremental tasks would help compare our approach to previous methods. It also would be worthwhile to see how the other approaches handle the incremental one-vs-all segmentation tasks.

One addition to CPR that may be worth pursuing for incremental segmentation is to model the background as done in MIB and SDR [11, 21]. This may help alleviate the problem of background shift, which is especially an obstacle in the incremental one-vs-all segmentation experiments performed in this work.

On top of the proposed solutions mentioned earlier in this section, we would hope to encourage further research into hierarchical conceptors. Should their function align with our intentions for implementing them, hierarchical conceptors could help break down current computational costs associated with large conceptors. The hierarchical conceptors may characterize activation patterns over multiple filters, layers, and even an entire network. This would typically be impossible to do with non-hierarchical conceptors.

7 Conclusion

Convolutional neural networks have some of the best performance in computer vision tasks and beyond. Unfortunately, CNNs being feedforward deep neural networks suffer from catastrophic interference. This prevents CNNs from being used more broadly and in a more versatile manner. Specifically, it prevents the model from learning new tasks without re-training on the previously learnt tasks. However, many use-cases of computer vision are set in an ever changing environment, creating a demand for methods that can learn in an incremental manner.

As a now decades old problem, there is a wide variety of proposed solutions to catastrophic interference. However, for CNNs most research is done on knowledge distillation. The majority of this body of research is also predominantly applied to classification. Methods that employ penalty computing for incremental semantic segmentation are seen relatively infrequently. Of those, works that perform the challenging one-vs-all class/task distribution are rarer still. As such it is difficult to find a direct comparison for this work, where we attempt to solve the incremental one-vs-all semantic segmentation task with a penalty computing approach.

This thesis is an extension to earlier thesis work by Hofman which looked at overcoming catastrophic interference in convolutional networks using a conceptor-based regularization technique. Conceptors were initially introduced by Jaeger as a neuro-computational device for dynamical (non-feedforward) recurrent neural networks. The conceptor, being a regularized identity map, has since been adapted for incremental learning in fully-connected feedforward neural networks by He.

Previous work by Hofman showed promise of using conceptors with convolutional layers in incremental image classification. However the proposed method was unable to learn the semantic segmentation task without performance loss after only learning three tasks in a one-vs-all incremental fashion with fully convolutional networks. We set out to understand why the segmentation performance suffers even after regularization with conceptor-based pseudo rehearsal. To this extent, we proposed to collect activation for the construction of the conceptors on a per filter basis, rather than on a per layer basis. We furthermore introduced a novel hierarchical conceptor computed over multiple filter-sized conceptors. Empirical results show that using the hierarchical conceptors outperform using layer conceptors. This may indicate that convolutional neural networks are better (additionally) regulated on a per-filter basis.

Further investigation showed that there were few changes in the individual filter weights. This might stem from an extremely high initial conceptor loss, which pushes the weights at the start of learning a new task in a direction uninformed by the new task. The trivial solution to decreasing the conceptor regularization term is by putting any changes in the network to zero. This may contribute towards the observed drop in performance in subsequent tasks with incremental training, as only the non-regulated model head is able to contribute towards learning the new task.

Additionaly, we found the conceptor loss to be numerically unstable. As the change in weights between tasks tend to zero, eventually the conceptor regularization term starts to significantly increase again. As a consequence, we are unsure whether the obtained performance in the experi-

ments is the performance that the models would have converged to, or is an artifact of the high initial conceptor loss and subsequent numerical instability.

We believe that conceptor regularization can still become a useful tool for overcoming catastrophic interference. Before conceptor regularization can reach that potential for semantic segmentation however, the aforementioned problems will need to be overcome. With the introduction of hierarchical and filter conceptors, conceptor regularization can now also be used on much larger networks, increasing the versatility of conceptor-based pseudo rehearsal. We hope to see the hierarchical conceptors be explored further, as it is currently unclear what exactly the conceptor activation from which the conceptors are computed entails. The applications for and research into conceptors is of course broader still; we look forward to learning what more conceptors have to offer.

References

[1] S. Papert, "The summer vision project," 1966. Available online at: https://dspace.mit.edu/handle/1721.1/6125, last accessed on 2022.08.26.

[2] M. Thoma, "A survey of semantic segmentation," *arXiv preprint arXiv:1602.06541*, 2016.

[3] Y. Guo, Y. Liu, T. Georgiou, and M. S. Lew, "A review of semantic segmentation using deep neural networks," *International Journal of Multimedia Information Retrieval*, vol. 7, no. 2, pp. 87–93, 2018.

[4] A. Garcia-Garcia, S. Orts-Escolano, S. Oprea, V. Villena-Martinez, P. Martinez-Gonzalez, and J. Garcia-Rodriguez, "A survey on deep learning techniques for image and video semantic segmentation," *Applied Soft Computing*, vol. 70, pp. 41–65, 2018.

[5] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," in *Psychology of Learning and Motivation*, vol. 24, pp. 109–165, Elsevier, 1989.

[6] G. M. Van de Ven and A. S. Tolias, "Generative replay with feedback connections as a general strategy for continual learning," *arXiv preprint arXiv:1809.10635*, 2018.

[7] R. Rifkin and A. Klautau, "In defense of one-vs-all classification," *The Journal of Machine Learning Research*, vol. 5, pp. 101–141, 2004.

[8] Z. Li and D. Hoiem, "Learning without forgetting," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 12, pp. 2935–2947, 2017.

[9] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive neural networks," *arXiv preprint arXiv:1606.04671*, 2016.

[10] U. Michieli and P. Zanuttigh, "Continual semantic segmentation via repulsion-attraction of sparse and disentangled latent representations," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1114–1124, 2021.

[11] F. Cermelli, M. Mancini, S. Rota Bulo, E. Ricci, and B. Caputo, "Modeling the background for incremental and weakly-supervised semantic segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2021.

[12] R. Aljundi, M. Rohrbach, and T. Tuytelaars, "Selfless sequential learning," *arXiv preprint arXiv:1806.05421*, 2018.

[13] H. Shin, J. K. Lee, J. Kim, and J. Kim, "Continual learning with deep generative replay," *Advances in neural information processing systems*, vol. 30, 2017.

[14] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, "iCaRL: Incremental classifier and representation learning," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 2001–2010, 2017.

[15] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the national academy of sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[16] X. He, "Continual learning by conceptor regularization," 2018. Available online at: https://marcpickett.com/cl2018/CL-2018_paper_55.pdf, last accessed on 2022.08.26.

[17] P. Hofman, "Using conceptors to overcome catastrophic forgetting in convolutional neural networks," Master's thesis, Rijksuniversiteit Groningen, Groningen, Sept. 2021.

[18] H. Jaeger, "Controlling recurrent neural networks by conceptors," *CoRR*, vol. abs/1403.3369, 2014.

[19] U. Michieli and P. Zanuttigh, "Incremental learning techniques for semantic segmentation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pp. 0–0, 2019.

[20] M. Klingner, A. Bär, P. Donn, and T. Fingscheidt, "Class-incremental learning for semantic segmentation re-using neither old data nor old labels," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1–8, 2020.

[21] U. Michieli and P. Zanuttigh, "Continual semantic segmentation via repulsion-Attraction of sparse and disentangled latent representations," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1114–1124, 2021.

[22] O. Tasar, Y. Tarabalka, and P. Alliez, "Incremental learning for semantic segmentation of large-scale remote sensing data," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 12, no. 9, pp. 3524–3537, 2019.

[23] J. Li, X. Sun, W. Diao, P. Wang, Y. Feng, X. Lu, and G. Xu, "Class-incremental learning network for small objects enhancing of semantic segmentation in aerial imagery," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, pp. 1–20, 2021.

[24] H. Jaeger, "Conceptors: an easy introduction," *CoRR*, vol. abs/1406.2671, 2014.

[25] X. He and H. Jaeger, "Overcoming catastrophic interference using conceptor-aided backpropagation," in *International Conference on Learning Representations*, 2018.

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

[27] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," *CoRR*, vol. abs/1604.01685, 2016.

[28] P. Jaccard, "The distribution of the flora in the alpine zone," *New Phytologist*, vol. 11, no. 2, pp. 37–50, 1912.

[29] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.

[30] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015* (N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, eds.), (Cham), pp. 234–241, Springer International Publishing, 2015.

[31] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and J. Han, "On the variance of the adaptive learning rate and beyond," *CoRR*, vol. abs/1908.03265, 2019.

[32] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2015.

[33] F. van Beers, "Using intersection over union loss to improve binary image segmentation." Bachelor's Project, Rijksuniversiteit Groningen, Groningen, 2018.

[34] M. Klingner, A. Bär, P. Donn, and T. Fingscheidt, "Class-incremental learning for semantic segmentation re-using neither old data nor old labels," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1–8, IEEE, 2020.

[35] G. Van Rossum and F. L. Drake Jr, *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[36] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Mathematical Programming*, vol. 45, no. 1, pp. 503–528, 1989.

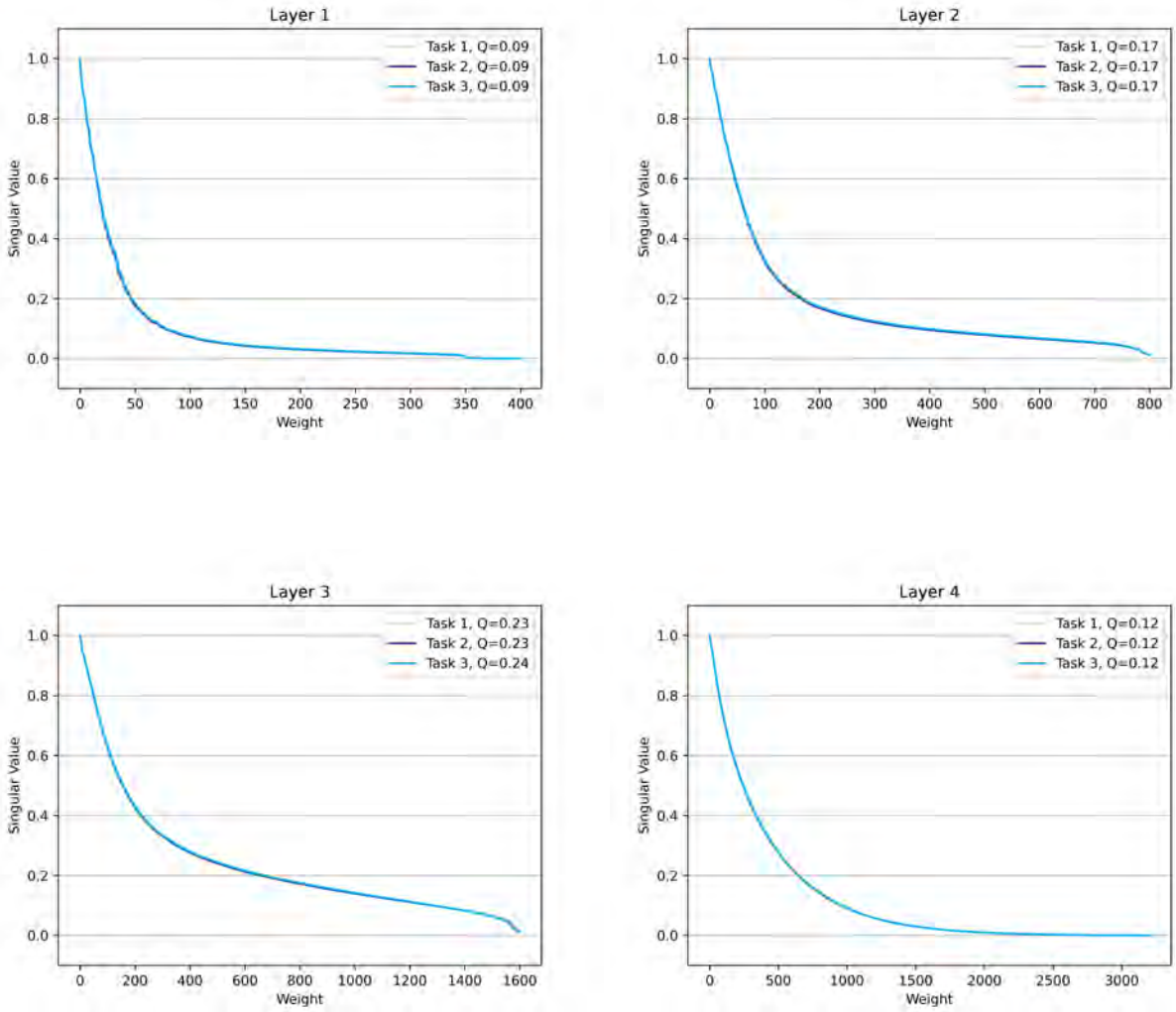
A Dataset

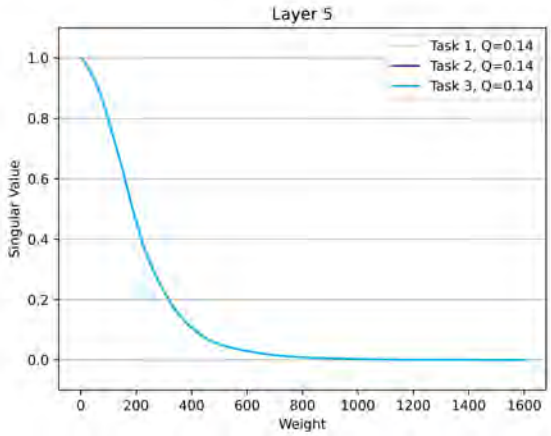
B Additional Results

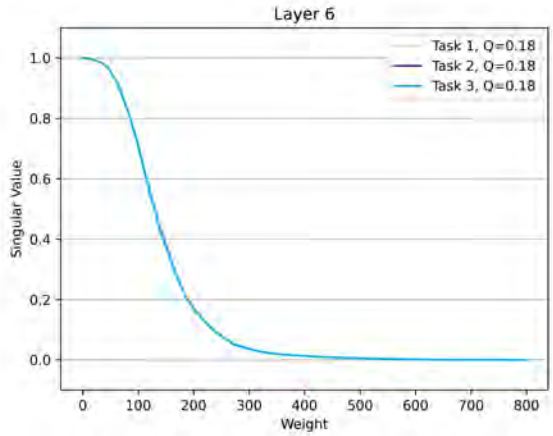
		Epochs			**Task 1**			**Task 2**			**Task 3**			**mIoU**
	Task	***Max***	***Mean***	***Std. Err.***	***Max***	***Mean***	***Std. Err.***	***Max***	***Mean***	***Std. Err.***	***Max***	***Mean***	***Std. Err.***	
Conceptor	***1***	24	24.000	0.000	0.901	0.897	0.001							0.897
	2	24	20.400	1.275	0.901	0.886	0.010	0.672	0.617	0.014				0.752
	3	24	19.500	1.241	0.901	0.885	0.011	0.673	0.619	0.013	0.519	0.509	0.004	0.671
Filter	***1***	24	24.000	0.000	0.900	0.897	0.001							0.897
	2	15	13.700	0.335	0.900	0.874	0.022	0.721	0.574	0.037				0.724
	3	17	13.400	0.653	0.900	0.876	0.021	0.722	0.556	0.047	0.531	0.510	0.003	0.647
Hierarchical	***1***	24	24.000	0.000	0.901	0.897	0.001							0.897
	2	24	24.000	0.000	0.898	0.875	0.007	0.748	0.711	0.009				0.793
	3	24	24.000	0.000	0.893	0.827	0.019	0.748	0.711	0.009	0.570	0.525	0.011	0.688

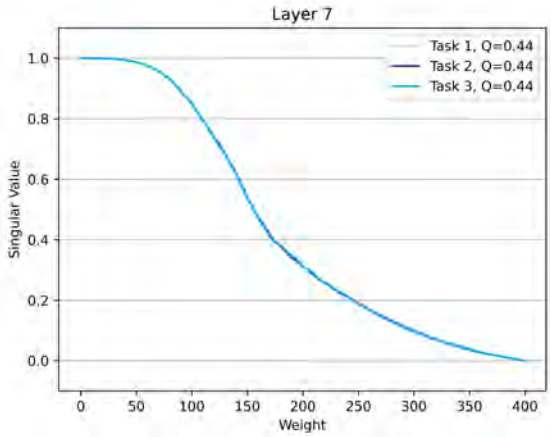
C Singular Value Spectra

C.1 Layer conceptors

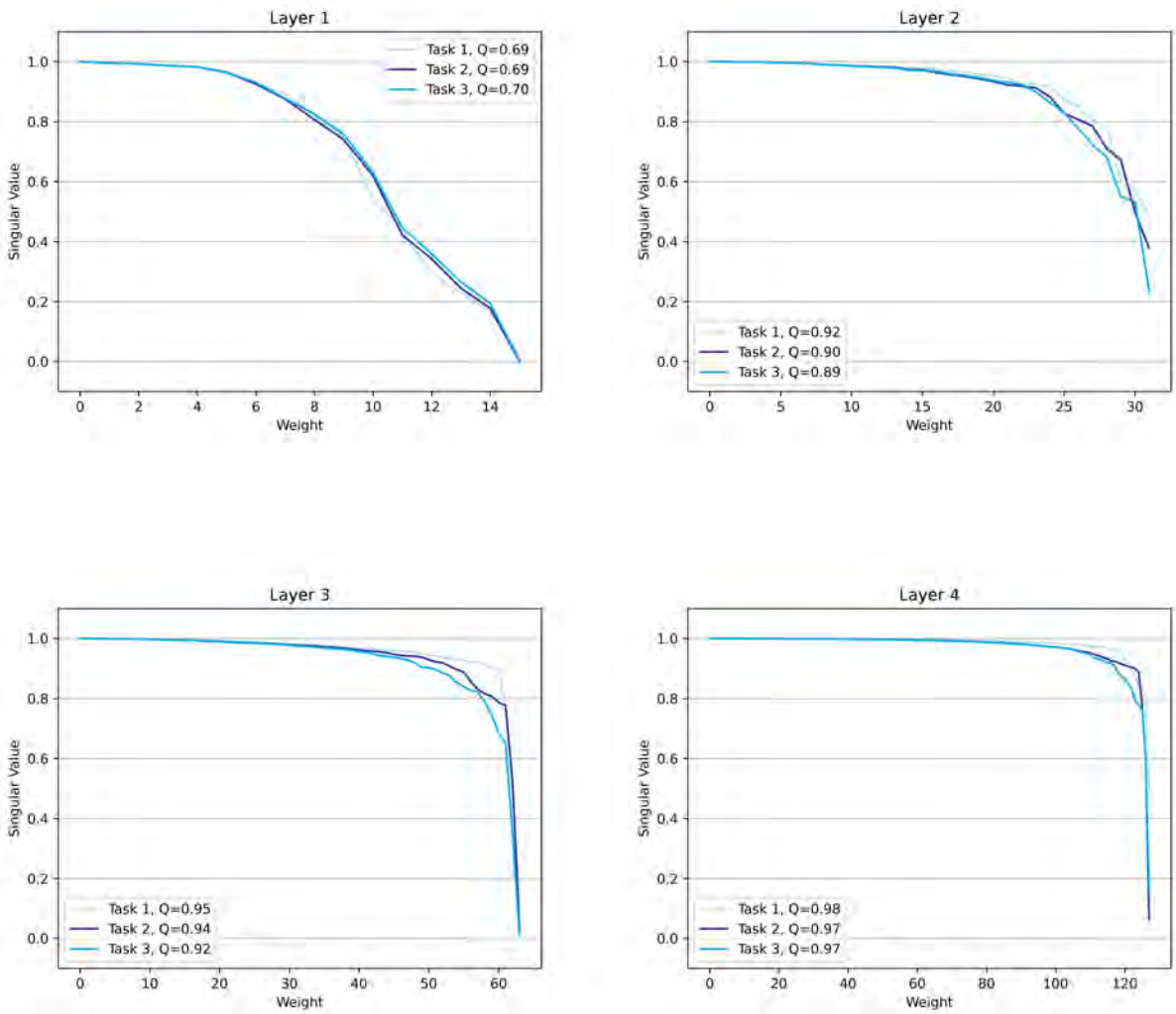


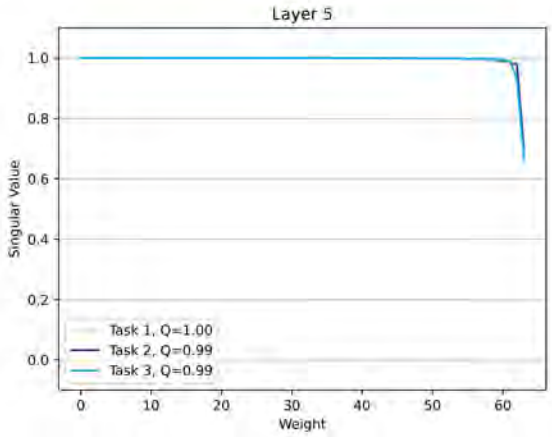

Layer 5
Task 1, Q=0.14
Task 2, Q=0.14
Task 3, Q=0.14
Singular Value
Weight

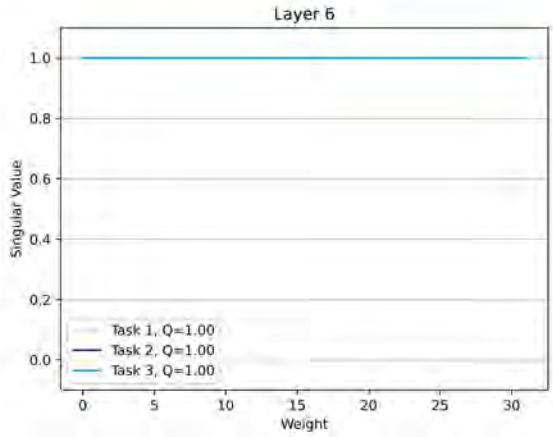

Layer 6
Task 1, Q=0.18
Task 2, Q=0.18
Task 3, Q=0.18
Singular Value
Weight

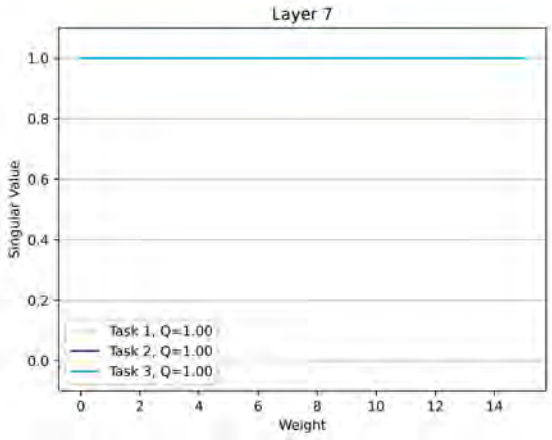

Layer 7
Task 1, Q=0.44
Task 2, Q=0.44
Task 3, Q=0.44
Singular Value
Weight

C.2 Hierarchical conceptors




Layer 5
Singular Value
Weight
Task 1, Q=1.00
Task 2, Q=0.99
Task 3, Q=0.99


Layer 6
Singular Value
Weight
Task 1, Q=1.00
Task 2, Q=1.00
Task 3, Q=1.00


Layer 7
Singular Value
Weight
Task 1, Q=1.00
Task 2, Q=1.00
Task 3, Q=1.00

D Model Output Examples

D.1 Joint model

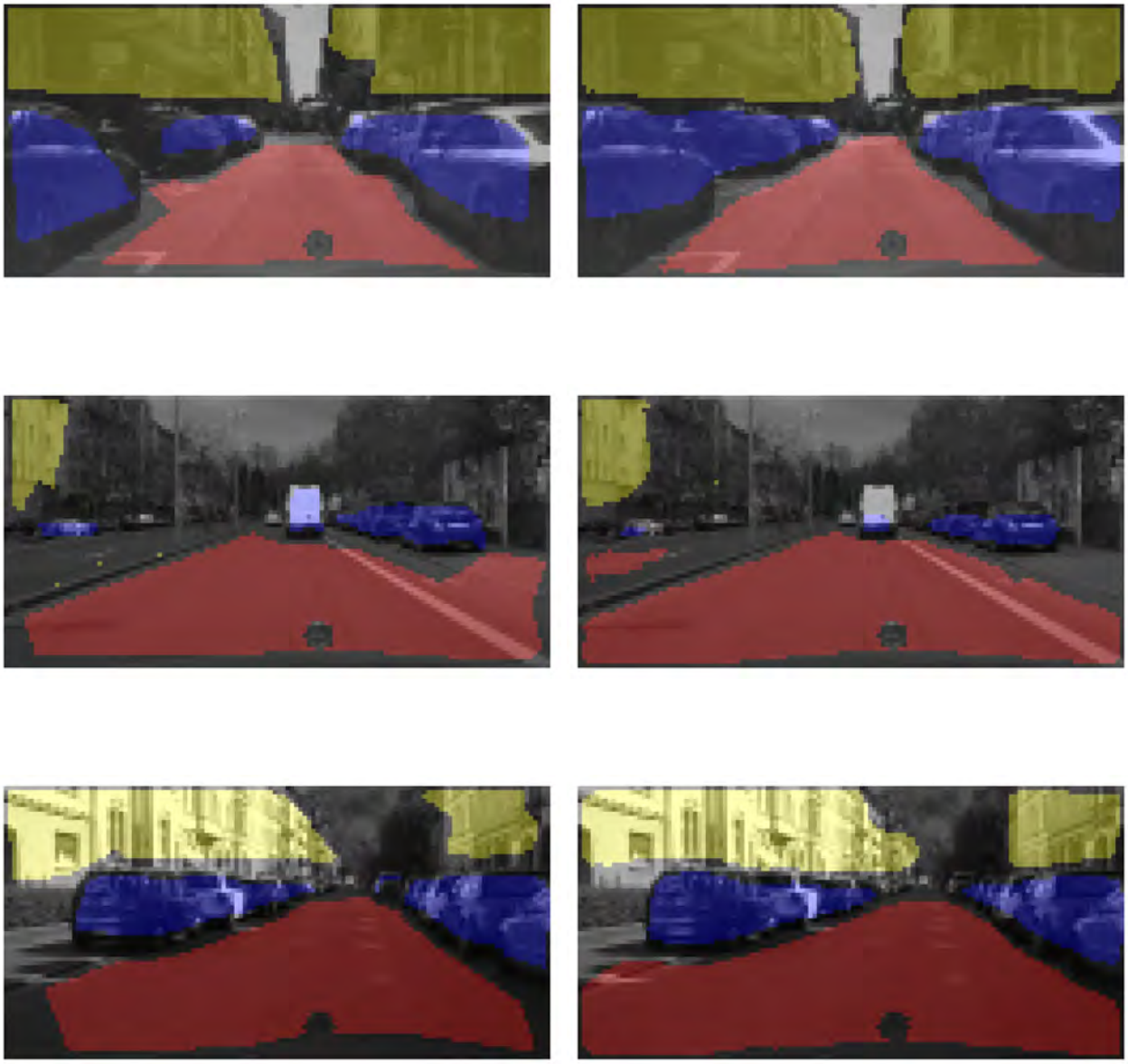


Figure 14: Ground truth (left) and Joint model prediction (right)

D.2 Layer model

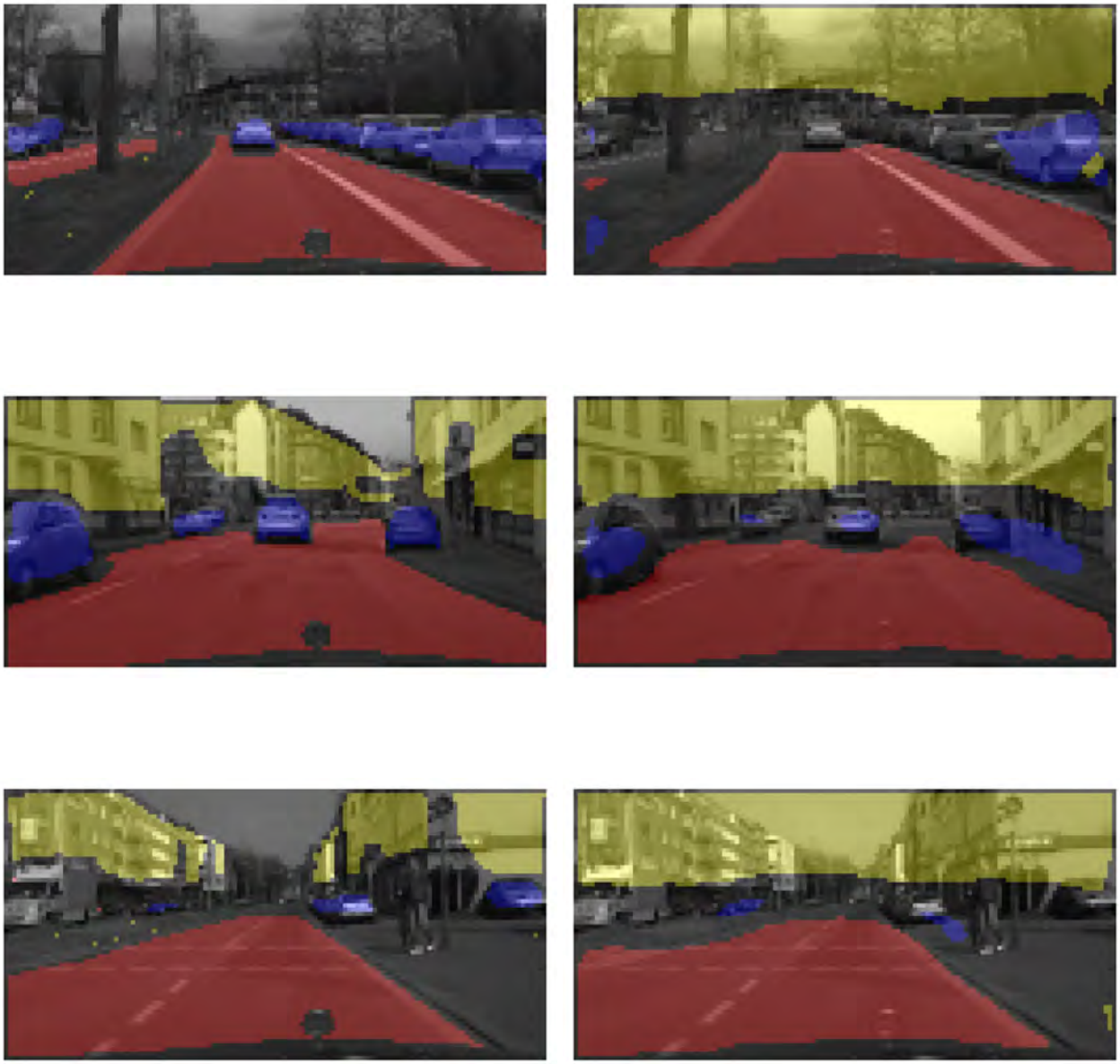


Figure 15: Ground truth (left) and Layer model prediction (right)

D.3 Filter model

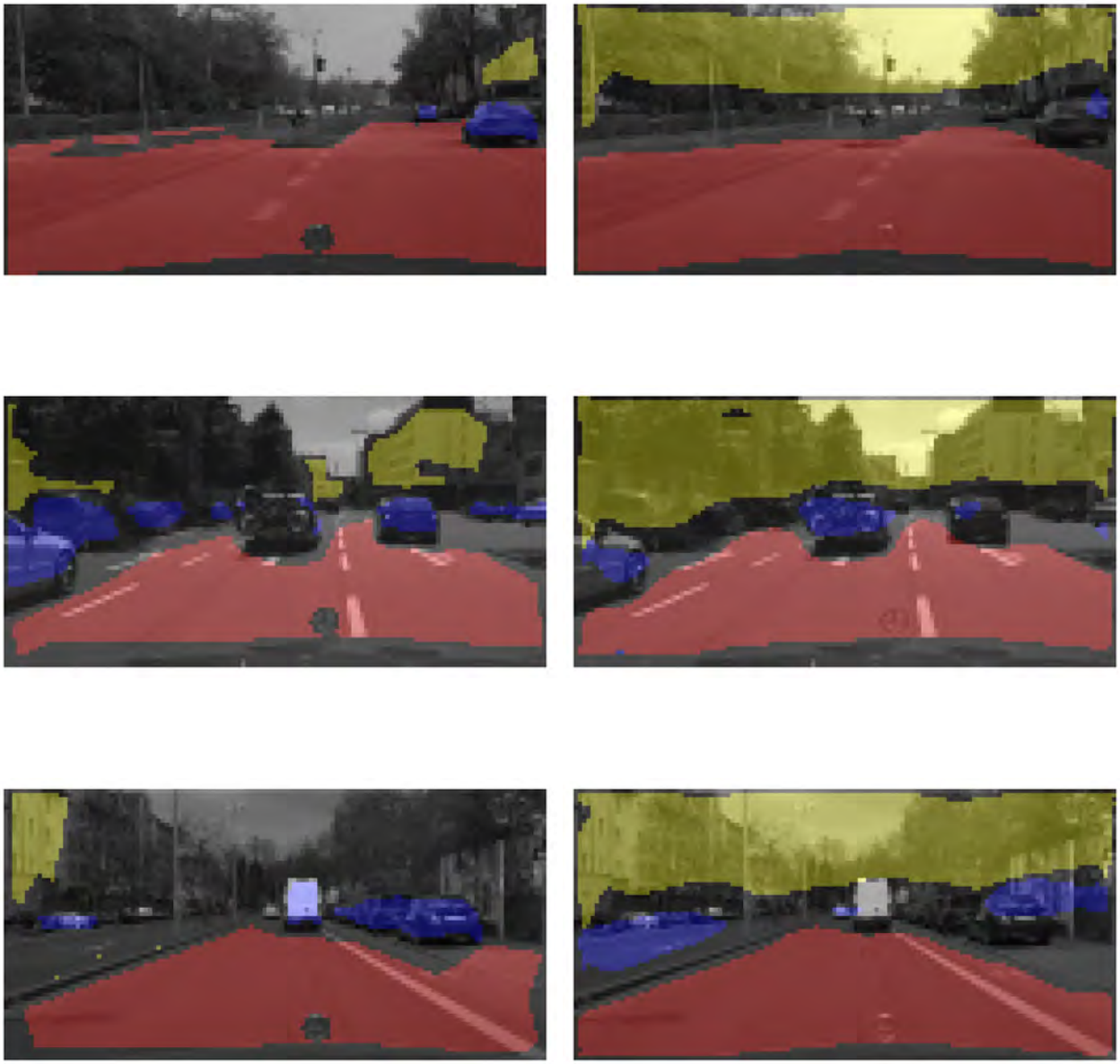


Figure 16: Ground truth (left) and Filter model prediction (right)

D.4 Hierarchical model

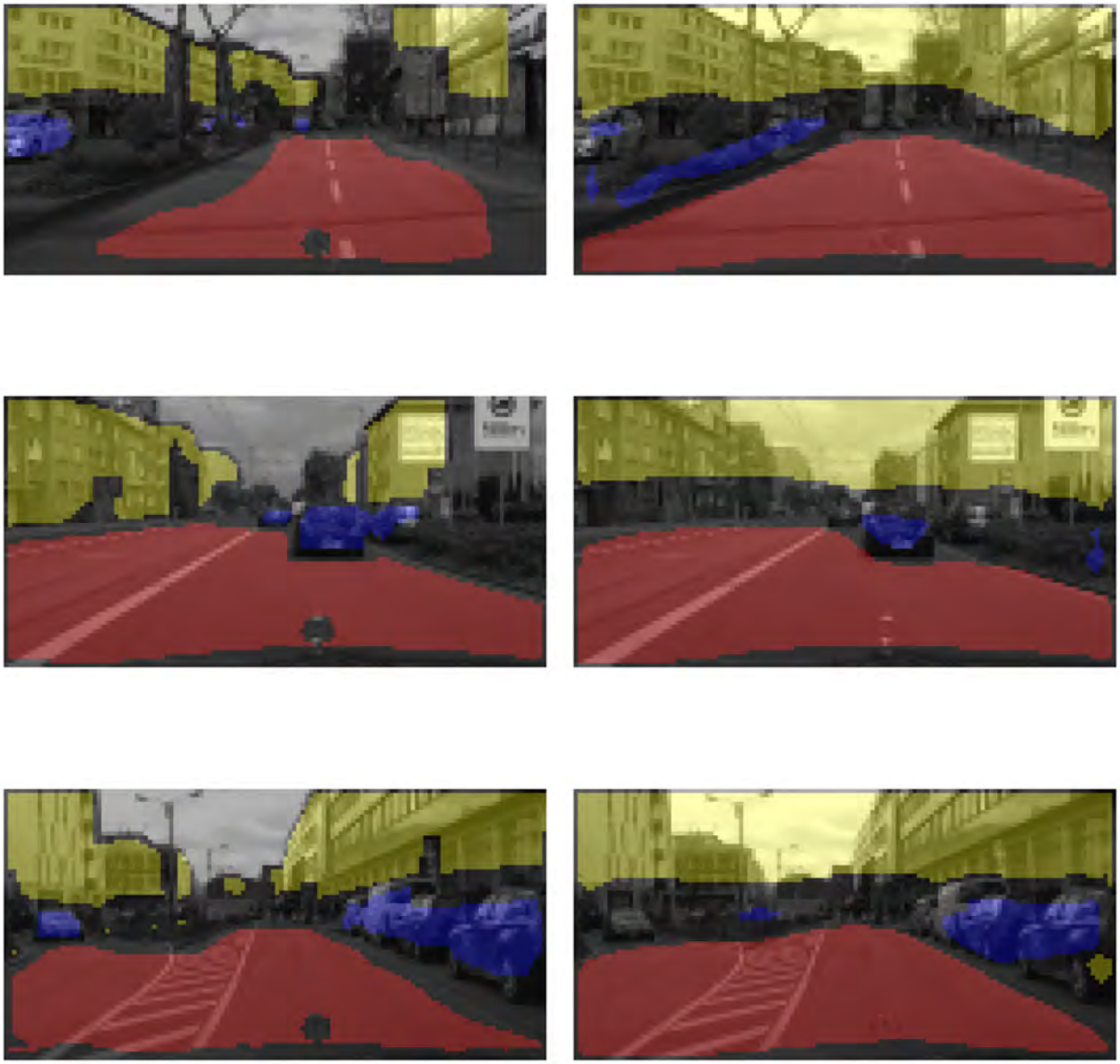


Figure 17: Ground truth (left) and Hierarchical model prediction (right)