Continual Learning in Semantic Segmentation

Maciej Kowalski

MInf Project (Part 2) Report
Master of Informatics
School of Informatics
University of Edinburgh

2022

Abstract

Class-Incremental Semantic Segmentation (CISS) is an emerging trend of Continual Learning (CL) in Computer Vision. On top of the catastrophic forgetting issue known in CL, CISS suffers from the semantic drift of the background class, further increasing forgetting. Existing attempts aim to solve this using pseudo-labelling, knowledge distillation or model freezing. We attempt to improve upon these methods by focusing predominately on the offline architecture, claiming that without suitable adjustments, the offline training of the architecture removes valuable information for future steps, harming overall performance.

We claim two main contributions. (1) Combining multiple class-specific 3×3 convolutions into one, large, shared module to improve efficient scaling for new, continual tasks (2) Introducing Dropout into the DeepLabV3 architecture to improve regularisation and decrease the COMPRESSION of information, a concept introduced in the Information Bottleneck principle that we adapt for CL.

To demonstrate the effectiveness of our approach, we have created our Dropout Continual Semantic Segmentation model (DCSS) and compared it experimentally with existing work. DCSS achieves state-of-the-art results on various CISS scenarios on the PASCAL VOC dataset. We improve the IoU on the 15-1 and 10-1 scenarios by over 2% and 3% respectively while maintaining a smaller memory and MAdds footprint. Last, we propose a new benchmark setting that lies closer to the nature of lifelong learning in the hope for more realistic and valuable architectures in the future.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Maciej Kowalski*)

Acknowledgements

First of all, I would like to thank Professor Amos Storkey for the two years of invaluable support I have received. It has been an honour to be your student and receive feedback from you; I would not be here if you had not agreed to my project, for which I will be forever grateful.

Special thanks to Tom Lee. I am very glad that we could have our meetings together, which were always interesting. Perhaps because I always took up some of your meeting's time, sorry! I wish you many great achievements on your PhD journey. I am sure you will achieve a lot.

Finally, I would like to thank the people who gave me their feedback and have always supported me, even when things did not go so well: Marcin, Michał, Karolina and Wojtek.

Table of Contents

Chapter 1

Introduction

1.1 Motivation

Tremendous progress has been made in recent years in Computer Vision. Many benchmarks from only a couple of years ago have been saturated and are no longer challenging. The focus on the improvement of supervised models and the chase for outperforming state-of-the-art results, even in the context of complex high-dimensional problems that we encounter in vision, has contributed towards stagnation in a local minimum of 2D image recognition[1]. To move on from this issue, we should tackle other problems to advance the perception algorithms and learn from visual data.

A current limitation of the dominant approach to Computer Vision is using static datasets, where all of the training data is available outright and does not change in time. The model is also trained with a specific, closed set of tasks or classes. This approach is far from real-world, dynamic scenarios where data distribution can shift over time [47], business requirements can be updated, necessitating a change in prediction power, and other important factors can come into play. In other words, the approach of supervised training on a static, labelled dataset is becoming obsolete for more and more practical applications.

The area of Continual Learning aims to tackle this issue by assuming that the data or the tasks might not be static and can change over time. This problem relaxation leads to surprisingly complex challenges, with the most common issue being *catastrophic forgetting* [42], which describes the model's inability to maintain the learned representation of the past. Forgetting is caused when past experiences are overwritten with a new training signal. A similar issue can be noticed in *Multi-task Reinforcement Learning*. Even though the tasks are interleaved throughout training and learned in parallel, standard artificial neural networks suffer from significant task interference when learning from distinct episodes corresponding to a subset of tasks [29]. Therefore, to account for this, our models need to be rigid enough to remember the essential details of the past and flexible enough to adapt to future requirements.

[1]This has been inspired by a tweet from Andrej Karpathy: `https://twitter.com/karpathy/status/1491452689825165314`

Recent work on Continual Learning focuses on the numerous ways to combat forgetting using replay methods, continually changing the models' architecture to adapt to new knowledge or distillation methods. Most problems, however, tend to have an offline phase where the model is initialised and learns the initial data and tasks. Mirzadeh et al. [38] show that wide convolutional models outperform standard ResNet-based encoders [20] in Continual Learning, proving that the choice of the initial architecture is fundamental to lessening the effect of catastrophic forgetting. Therefore, we claim that we should put just as much effort into the training of the *offline* phase as we put into the *online* phase; this issue tends to be overlooked by many publications in this area.

1.2 Context of current research with previous work

This project is a continuation of the work done for MInf Project Part 1 from the previous year titled "*Efficient compression of semantic segmentation neural networks*". Part 1, described further in Section 2.1, focused on the various compression techniques applied to semantic segmentation. Segmentation will continue to be the base scenario in this project, with the focus shifting on improvements to the Continual Learning of the classes, further referred to as Class-Incremental Semantic Segmentation (CISS).

Continual Semantic Segmentation is still a relatively new topic, with little research attempting to tackle the problem, which we describe further in Section 2.5. Our work attempts to improve upon the current state-of-the-art performance of CISS models, with the Information Bottleneck principle [48] being a foundation for the proposed improvements.

Despite some of its claims being questioned [43], Information Bottleneck theory by Tishby et al. [48] forms a basis for a framework that fits very well in the realms of Continual Learning. Therefore, this project studies the effect of rigid feature space caused by COMPRESSION in the context of Continual Semantic Segmentation, which to our best knowledge, has not been studied before. We propose several new contributions, forming a new Dropout Continual Semantic Segmentation (DCSS) model that achieves state-of-the-art results on a range of scenarios on the PASCAL VOC dataset [14]. The knowledge gained last year plays a crucial role in the success of DCSS, particularly the use of depthwise-separable convolution as an efficient operation with fewer parameters. Last, we have proposed a new evaluation protocol that better examines the performance of continual segmentation models when trained for a single epoch, limiting access to online data.

1.3 Main contributions

- Information Bottleneck theory has been introduced and discussed in the context of Continual Learning.
- Dropout Continual Semantic Segmentation (DCSS) model has been proposed that surpasses current state-of-the-art models in the majority of Class-Incremental Semantic Segmentation tasks on the PASCAL VOC dataset.

- DCSS outperforms recent models while having a smaller memory footprint and computational complexity in the case of multiple consecutive tasks. With only a handful of parameters added at each step, DCSS is easier to train and can work well with data-restricted scenarios.
- Inspired by the Information Bottleneck theory, a DCSS has introduced a new Dropout layer to the DeepLabV3 architecture to produce sparser feature representation.
- Extensive ablation study has confirmed the usefulness of DCSS contributions.
- New evaluation protocol, Restricted Class-Incremental Semantic Segmentation, has been proposed where the continual steps are trained for only one epoch, simulating restricted access to the stream of online data. DCSS also outperforms previous works in this protocol.
- Finally, a new repository for Continual Semantic Segmentation has been created, based on code from [4]. It has been reworked and extended to support multi-GPU training, training visualisations using Tensorboard and the `tqdm` library. It has been open-sourced and can be found at `https://github.com/mazo20/dcss`.

1.4 Structure of the report

This section has outlined the context of this work. In Chapter 2, we will further describe work done in Part 1 of this project, followed by a short reminder on semantic segmentation and a detailed description of Continual Learning, Information Bottleneck and recent approaches to Continual Semantic Segmentation. In Chapter 3, we will propose and describe our contribution to the topic. Chapter 4 includes dataset details, problem evaluation metrics and implementation details. This is followed by experimental settings and results thereof in Chapter 5, closed by a final discussion in Chapter 6 and conclusion in Chapter 7.

Chapter 2

Background research

2.1 Previous work

This research is the second part to the *MInf Project Part 1 Report* delivered in April 2021[1]. The last project explored the popular compression techniques for Convolutional Neural Networks, focusing on semantic segmentation. The DeepLabV3 model [6] has been used as the baseline for the experiments on the PASCAL VOC dataset [14].

2.1.1 Motivation

In image classification, arguably the most popular and researched Computer Vision task, it is possible to use smaller architectures like MobileNetV2 or EfficientNet, known for their excellent scaling and efficiency on low-powered devices that utilise CPU for inference. The general direction in achieving a small and high-throughput model is using efficient convolution blocks that have a reduced number of parameters and Multiply-Adds without compromising on performance [10]. However, semantic segmentation models can vary substantially from models designed for other tasks, especially in the upper parts. The following question was studied: will the typical compression techniques applied to image classification models work on the pixel-level tasks?

2.1.2 Cheap convolution

The project discussed the use of *cheap convolution* blocks [10] like bottleneck or depthwise-separable convolution in place of standard convolution. This technique provides similar performance for a fraction of the parameters and Multiply-Add (MAdd) operations and produces an activation map of the exact resolution as standard convolution. Hence, it is possible to swap the convolution blocks without changing the overall architecture. We also tested the Kernel-Sharing Atrous Convolution [26] that uses just one set of parameters for multi-scale atrous convolution in the Atrous Spatial Pyramid Pooling (ASPP) [6] module, improving their learning by having more gradient back-propagated through it.

[1]The project can be found here: `https://bit.ly/3ixUCTj`

2.1.3 Distillation

We used Knowledge Distillation [22] and Attention Transfer [54] for compression in semantic segmentation, which were successfully used in other Computer Vision tasks like classification. A typically larger, higher quality *teacher* model was used to transfer the knowledge to a smaller, more efficient *student* model. Distillation promises that we can use the teacher's signal to produce a higher-quality student model, which we could not do by simply training the model only on the original data. Thus, we effectively hoped to capture only the rich and essential features of an otherwise cumbersome and inefficient model.

Nonetheless, the popularity and success of distillation do not seem to be transferable to some tasks like image segmentation. In the first project, we showed that DeepLabV3 did not benefit from the extra distillation loss. This finding has been attributed to the inherent difference between image and pixel-level classification tasks like segmentation.

2.1.4 Compressed Atrous Spatial Pyramid Pooling

The use of ASPP was proposed by the DeepLabV3 [6] model to capture long-range feature dependencies without reducing the feature resolution, required for final mask prediction, using atrous (dilated) convolution. However, operating on large feature resolutions increased the computation complexity, with up to half of the parameters and MAdds in DeepLabV3 belonging to the ASPP module, depending on the backbone used.

Thus, our work suggested using multiple sequential bottleneck structures in place of a single, standard atrous convolution. Our Compressed-ASPP (CASPP) module slightly improved the IoU on the PASCAL VOC dataset [14] (74.25 vs 73.99) while having fewer parameters (26.08M vs 39.76M) and MAdds (8.89G vs 14.94G).

2.2 Semantic segmentation

Image segmentation is a pixel-level classification task, where each pixel in the image is classified into one of the available classes, with a background class covering the unclassified pixels. Segmentation is typically split into two distinct tasks: *semantic segmentation* and *instance segmentation*. According to the definition by Arnab et al. [1], instance segmentation treats each object of the same class as a distinct instance, while semantic segmentation treats them as the same entity. In this project, we will focus on semantic segmentation as a representative of the broad class of pixel-level classification problems.

Due to the importance of maintaining high feature resolution and their location in the image, Fully Convolutional Networks (FCN) excel at semantic segmentation, achieving impressive results on several benchmarks. Although the output resolution is the same as the input, we are typically computationally constrained and cannot perform computation on the original resolution whilst still benefiting from a local and global feature aggregation done by convolutional operation and pooling. Thus, an Encoder-Decoder architecture has been typically used, with an encoder extracting higher-level features from the data, usually utilising the architecture from the mainstream image classification models. That allows us to use and share the pre-trained encoder modules as feature extractors across tasks. This efficient initialisation vastly improves the performance and eventual convergence to optimal predictions.

Decoder modules utilise additional multi-scale feature extractors to obtain short- and long-range dependencies for improved pixel-level prediction. PSPNet [56] introduced multi-scale pooling in the form of a pyramid pooling decoder module. DeepLabV3 [6] uses a multi-scale atrous convolution to obtain a wider field of view. Most recently, StripPooling [24] added a long-range pooling to PSPNet, further increasing the segmentation performance. These modules effectively decode the intermediate representation and extract features in various resolutions, helping to predict the tight boundaries around the high-frequency object masks and the wide field of view required for better semantic recognition. The final prediction is then upscaled to the original size, as required by the task.

2.3 Continual Learning

Continual Learning aims to address the shortcoming of standard supervised learning that requires large quantities of labelled data. It pursues a more natural, human-like ability to quickly and continuously learn from small exposures to training data. The learner experiences a stream of tasks whose relatedness is not known beforehand [5] and has the potential to quickly learn a new task by re-using past experiences. Clearly, we have to put constraints on the amount of memory used to store the past to prevent the model from naively storing all experiences. This constraint enforces the need to efficiently compress the knowledge to a modest size and limit the computation used to recreate representation for a task at hand.

More precisely, Continual Learning in Machine Learning is the ability of a model to smoothly update the prediction model to take into account different tasks and data distributions using a stream of incoming data. This can be thought of as a learning procedure where not all of the data, tasks, or distributions are initially known. This environment is different from the typical approach to Machine Learning, where all requirements are known from the start, and no changes will be made in the future. This slightly more realistic scenario of models dynamically learning from the data imposes multiple new limitations to consider when developing our solutions.

2.3.1 Continual Learning scenarios

With the relaxation of constraints on the training procedures, Continual Learning necessitates many new approaches and proposed evaluation scenarios. Unfortunately, despite being actively researched, there is still no consensus on the proper settings and metrics for comparing the proposed CL techniques.

We follow the definition of van de Ven et al. and Hsu et al. [49, 25] who use three distinct scenarios for Continual Learning in the hope of standardising the evaluation procedure: Task-Incremental, Domain-Incremental and Class-Incremental learning.

Task-Incremental In the first scenario, models incrementally learn new tasks. We can utilise task-specific components since we know the task's identity ahead of time. Typically this implies the use of *multi-headed* output layer where each task has a separate output unit, but the rest of the network is shared. Only the head of task t will be activated at test time to make predictions. This prior task knowledge makes it the most straightforward continual learning scenario, although still vulnerable to forgetting.

Domain-Incremental In Domain-Incremental learning, task identity is not available at test time. However, the typical example of this scenario is a continual shift of input distribution. A real-world example is an agent who learns to survive in different environments without the need to explicitly identify the environment at hand [49].

Class-Incremental Lastly, Class-Incremental learning considers a scenario where the model must be able to solve each task *and* infer what task it is. The dataset is typically split into some steps, with each step containing an exclusive set of classes in a dataset. Thus, the model learns each set of classes incrementally while trying not to forget the previous steps. This project will focus on Class-Incremental learning, where we split the set of semantic segmentation classes into multiple steps and learn them sequentially.

2.3.2 Catastrophic forgetting

Catastrophic forgetting [42, 33] is a phenomenon observed primarily in Continual Learning where the earlier learned concepts are forgotten while incorporating the more recent samples. Similarly, in Multi-task Reinforcement Learning, the system experiences training episodes with different tasks, where each task might require different parameter weights [29]. Since each episode can contain information from disjoint domains, but we train a single model, we can overwrite previous knowledge, causing interference.

Forgetting appears when previously-learned representations are degraded by more recent exposures, a typical case in all SGD-based algorithms. The weight changes necessary to reduce the error for one task will differ from the changes required for another task. Since catastrophic forgetting is a crucial problem of Continual Learning, multiple solutions have been proposed to address this issue. They can be grouped into the following categories:

Rehearsal learning In rehearsal learning, we utilise the fact that, although we have lost access to the original training data from the past, we are usually allowed to maintain the data in a different form. [4] store exemplar images in a small memory buffer that can be replayed to simulate previous data distribution. We can also store the past experiences in a more compressed form. Compressed features in the form of embeddings [19, 28] can be used as an efficient form of memory that can also have advantages in terms of privacy. Finally, we can use generative strategies [44] that can efficiently store and reconstruct past experiences to enable the replay of raw images. [30] uses a brain-inspired dual-memory system where the new memories are consolidated from recent memory to a long-term memory using a generative model, similar to mechanisms that occur during sleep.

Adaptive architecture Other approaches focus on the adaptability of the model architectures. To integrate new classes and tasks, we can extend the architecture using feature extractors with domain-specific trainable layers [33, 52]. Model freezing can help with performance degradation during fine-tuning of the new models for new tasks. It is also possible to adapt the networks without explicitly adding new modules. [17, 16] dynamically re-arranges existing sub-networks, each specialised in one specific task, to account for new knowledge. Moreover, continually changing data distributions can be accounted for by explicitly correcting the classifier drift [2, 51].

Knowledge distillation Lastly, knowledge distillation methods consider the use of a *teacher-student* approach. A knowledgeable teacher model trained on past inaccessible experiences can inform the current model of the past. Therefore, distillation aims at constraining the model to prevent forgetting as it changes to adopt new data. There are several ways to constrain the model, with the most salient methods being applied to the weights [31], gradients [5, 36] or output probabilities [33, 40].

Although these are the distinct categories that we can split our approaches into, in reality, a mixture of them is used for Continual Learning, with each method having its clear advantages and disadvantages. Feature extractors tend to underperform on the new tasks because the shared representation fails to represent crucial discriminative information for the task, and fine-tuning leads to forgetting. Duplicating and fine-tuning help with forgetting at the cost of a linear increase of memory footprint and computation at test time. Distillation can be costly and can prevent the model from adapting to new data.

Moreover, it is worth noting that not every approach is comparable since specific methods are aimed at different task definitions. Thus, the resulting performance can vary substantially depending on the given constraints. For example, a set of raw images containing examples of previous classes can be used for rehearsal learning and usually allows us to recover and adapt to the underlying data distribution, thus limiting the severity of catastrophic forgetting. However, it can be impossible to store past raw data for privacy or storage reasons.

2.3.3 Continual Learning datasets

Continual learning of high-dimensional data streams is a challenging but also crucial problem. Existing datasets can usually be adapted for CL purposes, helping us achieve proof-of-concept solutions that, hopefully, can help us eventually converge to the results of offline learning. However, none of them were explicitly designed for lifelong learning, lacking the necessary details like spatial or temporal cohesion of training objects that cause the model's failure outside the artificial configuration.

Therefore, new datasets are emerging, with the foremost aim of developing methods for combating catastrophic forgetting while being manageable in size and complexity to speed up the research cycle. PermutedMNIST [55] randomly permutes all MNIST [32] pixels, differently for each task. Random permutation ensures that the model cannot learn a simple operation like rotation or translation. Each task is equally difficult as the original MNIST, although it requires a new solution.

CORe50 [35] contains a collection of 50 objects, each represented with a 15 seconds video delivering 300 RGB-D frames. The presence of temporal data (objects in the videos gently move across the frame) simulates visual data streaming. Since each video is experienced as a sequence of images of a specific class, it can quickly deteriorate the performance of past objects when using standard SGD optimisation.

Multi-modal data like CORe50 can help us transition better from 2D to 3D images and video understanding. However, many Computer Vision tasks do not yet have a corresponding Continual Learning dataset, including semantic segmentation. Therefore, there is still the need to transform existing, static datasets to account for data streaming. These adapted datasets should be considered as a temporary solution and a reference base for future improved datasets.

Tiered-ImageNet and mini-ImageNet [41] are popular classification datasets generated from the original ImageNet, with a categorical split among training, validation, and testing subsets. PASCAL VOC [14] and ADE20k [57] are also regularly adapted for Continual Learning in semantic segmentation and object recognition [3]. In this project, we will focus on the former, described further in Section 4.1.1.

2.4 Continual Learning in Semantic Segmentation

Despite vast progress in image segmentation, most algorithms are still trained offline, restricting the number of possible applications. On the other hand, continuous learning advances are mainly being researched in the context of image classification, with no reference to segmentation or considering other visual tasks as an afterthought. Only in the last couple of years has there been some progress in Continual Semantic Segmentation, formulating the problems and setting up initial benchmarks and their corresponding baselines.

2.4.1 Problem definition and notation

We consider the setting initially formulated by Cermelli et al. [3] and further extended by Cha et al. [4]. In Class-Incremental learning, the training happens with $t = 1, \dots, T$ incremental tasks, with $t = 1$ being the offline step. For each incremental state t, we observe a training dataset D_t that consists of pairs $(\mathbf{x}_t , \mathbf{y}_t)$, where $\mathbf{x}_t \in \mathcal{X}^N$ denotes an input image of N pixels, and $\mathbf{y}_t \in \mathcal{Y}_t^N$ denotes the corresponding ground-truth pixel labels.

Equation 2.1 describes the set of labels $\mathcal{Y}_t$ used *only* at t, consisting of the current classes and the background class c_b. In $\mathcal{C}^t$ we don't consider the past or the future classes. Thus, on top of the the true background pixels c_b^t, the c_b label is also assigned to the pixels of potential objects that belong to past classes $\mathcal{C}^{1:t-1}$ and the future classes $\mathcal{C}^{t+1:T}$ (Equation 2.2).

$$\mathcal{Y}_t = \mathcal{C}^t \cup c_b^t \tag{2.1}$$

$$c_b^t = c_b \cup \underbrace{\mathcal{C}^{1:t-1}}_{past} \cup \underbrace{\mathcal{C}^{t+1:T}}_{future} \tag{2.2}$$

This counter-intuitive behaviour of the background class' labels causes the problem of *background shift* [3], on top of the catastrophic forgetting found in CL. The background class modification introduces unwanted noise into the model because the same object can be labelled differently, depending on the step t. Background shift is a key problem in Continual Semantic Segmentation and requires unique approaches, described in the next section.

The architecture of f_θ^t is typically a fully-convolutional network, which consists of a convolutional feature extractor, followed by the final 1 × 1 classifier filters $\{\phi_c^t\}_{c \in \mathcal{Y}^{1:t}}$, one for each output class in $\mathcal{Y}^{1:t}$. The learning of f_θ^t can be done in conjunction with the previous model f_θ^{t-1}, depending on the chosen Continual Learning constraints, to prevent forgetting when incrementally updating the model. Determining the shape of the classifier, the activation and loss functions as well as how to transfer the knowledge of f_θ^{t-1} to f_θ^t are design choices, with each work having its own approach.

The semantic segmentation model f_θ^t is required to predict whether a pixel belongs to all the classes learned so far, $\mathcal{C}^{1:t-1} \cup \mathcal{C}^t$, or the true background. Once the learning of task t by the model f_θ^t is done, the prediction for pixel i of an input image $\mathbf{x}$ at test time is obtained by

$$\tilde{y}_i^t = arg\,max_{c \in \mathcal{Y}^{1:t}} f_{\theta,ic}^t(\mathbf{x}_t) \tag{2.3}$$

for all the classes learned so far in $\mathcal{Y}^{1:t}$. The performance is measured by the Intersection-over-Union (IoU) metric for the classes in $\mathcal{Y}^{1:t}$, described further in Section 4.2.

2.5 Related work

Continual Learning in semantic segmentation is a relatively new topic that has not been adequately researched yet. There have been some recent initial attempts at improving the catastrophic forgetting with pixel-level classification in mind and working around the background shift problem, which we will cover in this section.

2.5.1 Modelling the background

Cermelli et al. in MiB [3] defined the issue of background shift and proposed to adapt the loss function such that it takes into consideration potential previous classes $\mathcal{C}^{1:t-1}$ that could be included in c_b^t. They adapt the output probabilities to match the available labels or distillation probabilities. In particular, they sum the output probabilities of $\mathcal{C}^{1:t}$ and c_b to match the available label c_b^t that contains them when computing the cross-entropy loss. Conversely, they sum the probabilities of $\mathcal{C}^t$ and c_b^{t-1} for knowledge distillation to match the output of f_θ^{t-1}.

The summation of class probabilities to modify the cross-entropy and distillation losses is a novel approach in semantic segmentation, but it makes it hard to learn the underlying probability distribution for the classes at each pixel. Nonetheless, this eases the background shift of previous classes and substantially improves the baseline scores produced by standard CL algorithms like EWC [31], ILT [37] or simple SGD.

2.5.2 Pseudo labels with Localised Pooled Distillation

PLOP [12] extends the knowledge distillation on output probabilities with an additional attention transfer mechanism called Local POD or Localised Pooled Distillation [12], applied at each convolution layer across multiple scales for each filter. Using Local POD, the new model preserves the activation signal of previous classes while accommodating the new set of classes. Local POD helps with catastrophic forgetting and acts as a distilled supervision signal, helping to preserve existing features in the model.

Moreover, the target labels $\mathbf{y}^t$ are augmented with the predicted classes by a model from a previous step f_θ^{t-1}, commonly called *pseudo-labels* [27]. For each pixel i we find the predicted class using a model from a previous step $t-1$ as in Equation 2.4.

The confidence score (Equation 2.5) describes the certainty of the prediction by model f_θ^{t-1}, where $\sigma(.)$ is the sigmoid function. In case of a noisy, low-confidence prediction, we are not interested in using this label in the next steps. Therefore, we limit the pseudo-labels propagation with a threshold τ. PLOP uses an entropy-based threshold to account for the noisiness of the predictions. Pseudo-labels with the ratio of entropy and maximum pixel entropy in the image below τ are not used in $\tilde{y}_i^t$.

Equation 2.6 shows the final label generation for each pixel i. We use labels of $\mathbf{y}^t$ and update them with pseudo-labels $\hat{\mathbf{y}}^t$ only if they pixel currently belongs to the background class, was predicted by a previous model as member of $\mathcal{C}^{1:t-1}$ and has a confidence score μ_i greater than τ.

$$\hat{y}_i^t = arg\,max_{c \in \mathcal{Y}^{t-1}} f_{\theta,ic}^{t-1}(\mathbf{x}_t) \tag{2.4}$$

$$\mu_i = max_{c \in \mathcal{Y}^{t-1}} \sigma(f_{\theta,ic}^{t-1}(\mathbf{x}^t)) \tag{2.5}$$

$$\tilde{y}_i^t = \begin{cases} \hat{y}_i^t & y_{t,i} = c_b \wedge \hat{y}_i^t \in \mathcal{C}^{1:t-1} \wedge \mu_i > \tau \\ y_i^t & otherwise \end{cases} \tag{2.6}$$

In other words, for each training image the labels $\hat{\mathbf{y}}^{t-1}$ with classes $\mathcal{C}^{1:t-1}$ are predicted by the model f_θ^{t-1} and are copied to the target labels $\mathbf{y}^t$ if they contain important information. This yields the target semantic map $\tilde{\mathbf{y}}^t$ containing background class b, pseudo labels $\hat{\mathbf{y}}^{t-1}$ and current labels $\mathbf{y}^t$.

Pseudo-labelling is an essential addition to the background modelling approach from [3], recovering the signal for the past classes from the background. However, it does not deal with the potential future classes appearing in the unclassified pixels. Local POD

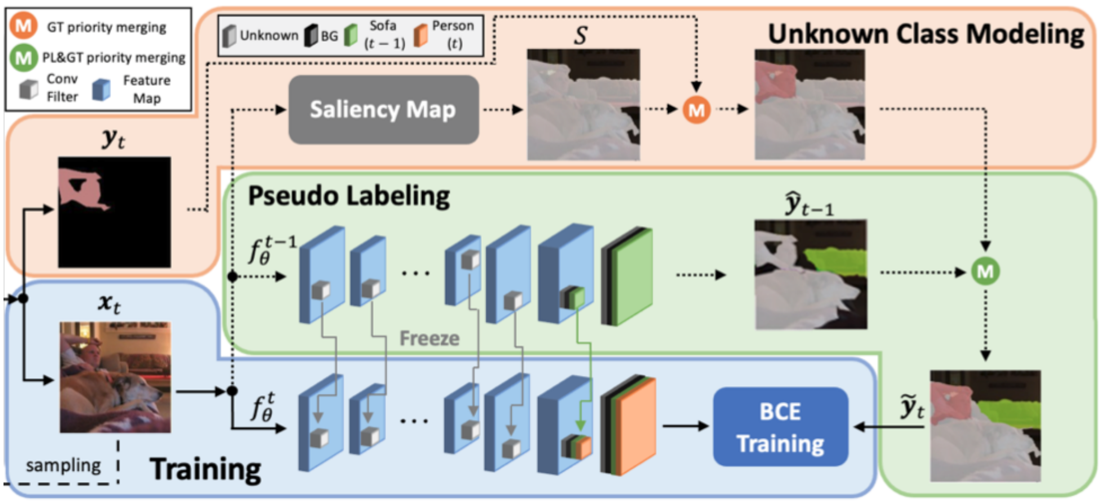


Figure 2.1: *Architecture of SSUL [4]. Labels* $\mathbf{y}^t$ *are extended with the saliency map and pseudo-labels. Image taken from [4].*

helps to prevent catastrophic forgetting, but at the same time, it heavily constrains the model from learning new tasks.

2.5.3 Model freezing with unknown label

SSUL [4] is the current state-of-the-art solution for this problem. Cha et al. approach the issue of catastrophic forgetting by freezing the model to ensure the previous knowledge is unaffected by new tasks. Moreover, they use *saliency* maps as the *unknown* class to distinguish potential future classes from the background pixels, further reducing the effect of background shift.

Model freezing SSUL changes the approach to combating catastrophic forgetting in Continual Semantic Segmentation. Instead of fine-tuning the whole model at step t, all parameters from f_θ^{t-1} are being frozen in f_θ^t, preventing any changes to the prediction of classes $C^{1:t}$. Freezing means that no gradient is being propagated to these parameters, and they are not being updated anymore. The only trainable parameters per step are in a single 3×3 convolution layer and the final 1×1 vector producing the output value for each pixel. These belong to the newly added classifier for the current set of classes, predicting final values from the output of the decoder.

The benefits of freezing also include reduced training time and resource utilisation for the remaining parameters, as we do not have to track and propagate the gradients for frozen convolutions. Moreover, less trainable parameters reduce the required training time and data requirements at the cost of rigidity of the representation.

Foreground prediction SSUL uses saliency maps generated by a separate, off-the-shelf salient object detector with short connections [23] to predict a region of interest from the background. Semi-supervised foreground prediction helps to differentiate the difference between true background and background that may contain a past or future class. The saliency maps are predicted with a separate model before training and added to the image target labels in place of a part of the background class. As can be seen at the top of Figure 2.1, similarly to pseudo-labels, the saliency map is added to the label $\mathbf{y}_t$ and predicted by the model during training. The unknown class c_u and the background class c_b predictions are combined at test time, representing the unclassified pixels.

The authors claim the addition of a *foreground* class is beneficial to the model as it can learn to differentiate the potential future classes from the true background. This assumes that the future classes are more likely to be objects like a `bike` or a `dog`, rather than boundary-less classes like `sky` or `sea`. Moreover, assuming that potential future classes are contained in the *unknown* class, SSUL performs weight transfer from the unknown class to the new classifier at the beginning of each continual step (Equation 2.7), providing a better initialisation that can reduce reliance on the quantity of data.

$$\phi_{c_u}^{t-1} \to \phi_c^t \tag{2.7}$$

Binary cross-entropy loss SSUL relies heavily on the frozen model maintaining its predictions for past classes and preventing catastrophic forgetting. However, the background classifier cannot become fully frozen as it needs to adapt to the new tasks that might be a subset of the background. The prediction can be further deteriorated by the noise introduced by the pseudo labels. All this can change the relative outputs produced by the softmax function, even with frozen parameters. Therefore, SSUL proposes the use of a sigmoid activation function with binary cross-entropy loss to accommodate the labels' noisiness instead of the typical softmax and cross-entropy loss.

Cha et al. claim that the sigmoid function with binary cross-entropy increases the model's stability and better handles the cases of mislabelled pixels by the pseudo-labelling process that can cause trainable classifiers $\phi_{c_u}^t$ and $\phi_{c_b}^t$ for unknown and background class respectively to overpredict.

Limitations of the frozen model While extending existing offline models for Continual Learning is an important feature, there are times when we can anticipate the future learning adjustment. Thus, we argue that it is just as important to train a task-aware offline model as it is to improve CL algorithms. However, except for the foreground prediction, SSUL trains the offline model at step $t = 1$ as if it was not meant to be trained in a continual learning manner. Therefore, the feature representation available for the new classifier might be too rigid to learn new classes. Indeed, being aware of the continual nature of the training is generally challenging to account for since we do not have access to an oracle to predict future tasks. However, we can limit the COMPRESSION with a handful of techniques, including better regularisation, use of a self-supervised backbone that will contain more features or, perhaps, have skip-connections to the final layer so that the classifier can have access to a richer feature representation.

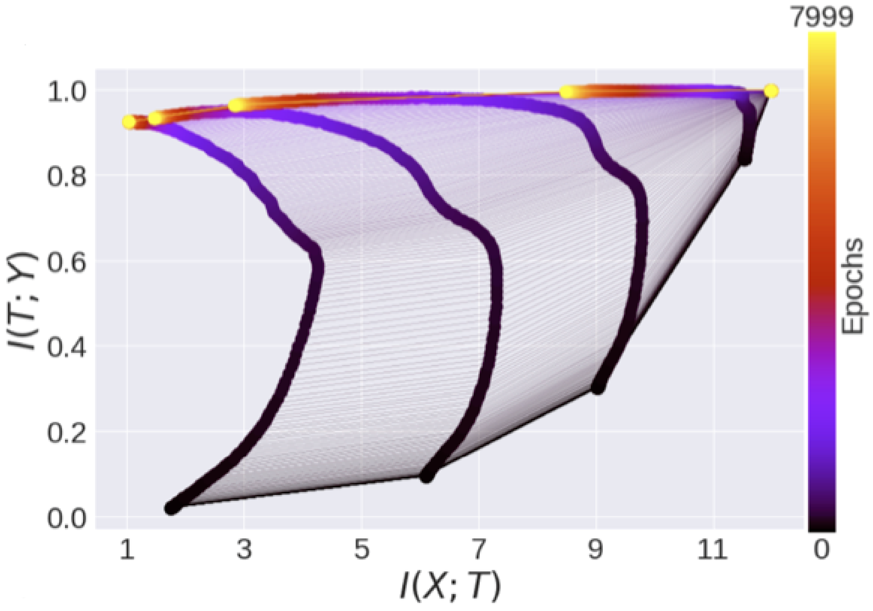


Figure 2.2: *Information Bottleneck visualisation. The x-axis plots information between each layer and the input, while the y-axis plots information between each layer and the output. The colour scale indicates training time in epochs. Mutual information about the input tends to decrease as the training goes by, signalling the compression phase. Image taken from [43].*

2.6 Information Bottleneck theory

In supervised learning, it is common to use pre-trained models for good weights initialisation that promises faster convergence when training. Pre-trained models provide us with good initial feature representation even if not trained on a similar domain as the target task. They tend to be trained on extensive datasets like ImageNet [11] or COCO [34].

Their generalisation performance is crucial in domain adaptation since adding convolutional filters is very resource and time-intensive, but it is relatively easy to disregard them. Thus, since we tend to have *more* features than necessary, during fine-tuning, we can simple *remove* unnecessary features and boost the signal of favourable ones.

Therefore, it is possible that by performing supervised training at step $t = 1$, our model is likely to remove and prevent the propagation of unused features from the pre-trained encoder model. This behaviour is desirable in offline training when our static dataset already contains all classes and is a good representation of the true data distribution. The model is looking for a quick and cheap way to train the model as fast as possible, so undesired features should be removed. However, excessive compression means that we might remove features that are crucial for learning a prediction of one of the future classes that we might not be aware of at this time.

The Information Bottleneck Principle (IB) [48] can partially explain and support the above difficulty of training Continual Learning models with gradient-based optimisation. Tishby et al. claim that modern Deep Neural Networks undergo two phases of learning:

GENERALISATION and COMPRESSION (Figure 2.2). During GENERALISATION phase, the model learns an internal feature representation to extract high-level information, used for final prediction. In COMPRESSION, unused and noisy features that prevent robust adaptation to the data are stripped away and removed from the feature space, ultimately improving useful features' signal quality.

The IB theory is based on the idea of mutual information. In probability theory, mutual information represents the amount of information obtained about one variable A by observing another random variable B (Equation 2.8). The higher the value, the more dependency there is between the variables.

$$I(A;B) = H(A) + H(B) - H(A;B) \tag{2.8}$$

Let X and Y be input and output layers, while T is any hidden layer of a Deep Neural Network. Tishby et al. measured the mutual information $I(X;T)$ and $I(T;Y)$, which quantify the hidden layer's information about the input and the output, respectively.

In Figure 2.2 we see that as the training is happening, the amount of information about the output $I(T;Y)$ steadily increases until a high number of epochs is experienced, and the model starts overfitting. The information about the input $I(X;T)$ raises initially but quickly starts decreasing. This adverse change can be considered as the COMPRESSION phase because we remove information from the input that does not contribute much to the output. COMPRESSION accelerates rapidly when the number of training epochs is high. Although the importance of the Information Bottleneck theory has been questioned by some papers [43], it serves as an interesting point of view on the learning phases of neural networks and can be noticed empirically.

2.6.1 Effect on Continual Learning

COMPRESSION phase, although natural and expected in static supervised environments, is not particularly useful in Continual Learning. The uncertainty about the future requires us to maintain as many high-quality features as possible since they might be sought in other tasks. Thus, it is in our best interest to promote GENERALISATION whilst limiting the effect of the COMPRESSION phase in Continual Learning. We know that wider models better manage catastrophic forgetting [38], proving that passing information through the bottleneck, which inherently implies COMPRESSION, can have adverse effects on future tasks.

Being aware of the COMPRESSION, we can think of ways to alleviate some of its unwanted effects on lifelong learning. Therefore, this work aims to determine the potential impact of enforced feature sparsity in a typical encoder-decoder model for Semantic Segmentation and how well it behaves under parameter freezing conditions. Having a reliable and accurate frozen representation with a broad range of features can be a key to short-term advances in Continual Learning.

Chapter 3

DCSS model design

SSUL [4], the current state-of-the-art model, achieved a significant improvement over the previous papers tackling Class-Incremental Semantic Segmentation, in particular MiB and PLOP (Table 5.1). SSUL prevents catastrophic forgetting as we know it in typical Stochastic Gradient Descent approaches, even in a long sequence of continual tasks. We attribute this improvement to the model freezing combined with the sigmoid activation function and binary cross-entropy loss function. Counter-intuitively, model freezing also helps with the learning of the new classes, compared to previous distillation-heavy approaches of MiB and PLOP. This demonstrates that, in the short-term, Continual Learning methods benefit disproportionately from increased learning stability, even at the cost of more rigid architecture that limits their flexibility to learn new feature representations.

Notwithstanding, the results of SSUL are still far from the practical upper bound achievable with offline, JOINT training. Despite significant improvements in learning of the continual tasks, the difference is growing the more steps there are. We notice that SSUL trains the offline model at step $t = 1$ as if it was not meant to be trained further in a Continual Learning manner. The COMPRESSION phase of Deep Learning can severely impact future performance in CL and has to be accounted for, while SSUL's approach to introducing new classifiers into the model is not scalable and introduces new issues. Our goal was to improve upon the architecture of SSUL by introducing superior and more efficient feature space produced by the frozen model, inspired by the Information Bottleneck theory.

Therefore, in this section we introduce our **Dropout Continual Semantic Segmentation** model (DCSS) for Class-Incremental Semantic Segmentation. The main contribution of this work is determining whether the SSUL [4] model for Continual Learning can be improved with regularisation methods. Regularisation of the offline model should alleviate the unwanted effect of the COMPRESSION phase and maintain more features for the online phase. We suggest new approaches to mitigate this issue that offer other benefits like computational efficiency and implementation simplicity. We describe our contributions that provide quantifiable advantages over SSUL and set an additional context for their origins. To prove the advantages of DCSS, we perform a range of experiments in Section 5. Additionally, we propose the Restricted Class-Incremental

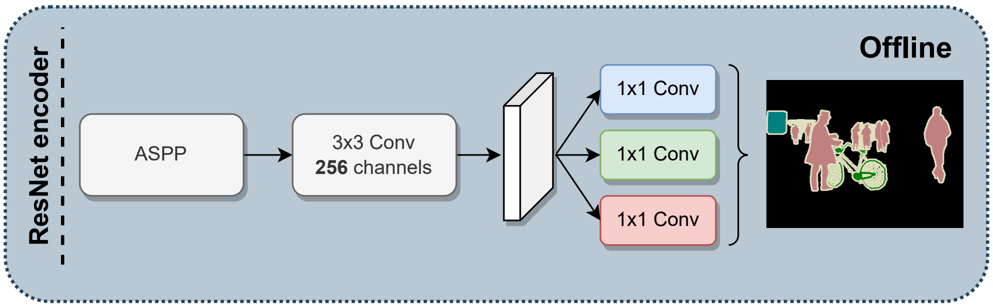


Figure 3.1: *High-level architecture of DeepLabV3 used for offline training. The ASPP module is followed with a single* 3×3 *convolution and* 1×1 *classifiers of 256 channels to produce output value for each class.*

Semantic Segmentation (RCISS) task, a new Continual Learning evaluation protocol that uses only one training epoch at each step $t > 1$.

3.1 Classifiers and shared convolution

In Class-Incremental Continual Learning we add a set of classes $\mathcal{C}^t$ at each step t. Since we do not know at step $t = 1$ how many classes will be added, we have to add additional classifiers $\{\phi_c^t\}_{c \in \mathcal{C}^t}$ to the model to predict the prediction score for each new class. Usually, this means a 1×1 convolution that sums over all channels in semantic segmentation.

In DeepLabV3, the final 1×1 convolution is preceded by the ASPP module and one 3×3 convolution (Figure 3.1). The last 3×3 convolution is important to aggregate and mix multi-scale features produced by the ASPP module. We will refer to this 3×3 convolution layer as HEAD for readability reasons. Recent papers tackling Class-Incremental Semantic Segmentation, including MiB, PLOP and SSUL, replicate the HEAD module for each step. Therefore, each set of classes $\mathcal{C}^t$ has its own HEAD and final classifier, as seen in Figure 3.2a

The addition of a new HEAD at each step facilitates extracting additional features from the frozen model, which, in theory, should vastly improve the prediction capability and performance of the model. However, in DeepLabV3, the HEAD layer is a costly operation that applies 256 filters, each having 256 3×3 kernels, totalling over 0.5M parameters. Therefore, each step of SSUL adds a substantial amount of parameters which accumulate as we add more and more classes. Despite this significant linear model growth being a substantial modification to the architecture, we did not manage to find any reference to that specific implementation detail in the above papers.

3.1.1 Shared head in DCSS

Therefore, we compare the existing SSUL implementation with a separate HEAD (SEPARATE) per task with our DCSS model containing a single shared HEAD (SHARED) trained offline and only 1×1 convolutions added at online steps $t > 1$.

Using the original HEAD from DeepLabV3 for the shared module, we are left with a single, 256-dimensional 1×1 vector of trainable parameters for each new class. We argue that, although this is sufficient for offline learning and is a typical size found in DeepLabV3, it might not be enough for Continual Learning properties. A large number of output channels from a frozen model is crucial, as this will determine the maximal amount of information that can be propagated for further use by the classifiers.

Thus, to recuperate some of the performance lost with a SHARED model, we increase the size of HEAD output channels to 2048. Hence, a more detailed feature space is available for the classifiers to exploit while removing the need to learn the HEAD module at each step. Our experiments in Section 5.1.1 show a significant reduction in the computational requirements in the long run while producing comparable or better results.

Moreover, to balance the required larger number of channels with the expensive convolution, we use depthwise-separable convolution in the shared HEAD. Depthwise-separable convolution produces the same activation map using a cheaper operation with fewer parameters and MAdds. In our work, we have used an intermediate width multiplier of 4 and 4 convolution groups (Figure 3.3). Standard shared HEAD has almost twice the parameters of the depthwise-separable implementation (4.71M vs 2.68M).

Therefore, despite initially having more parameters than SSUL, DCSS will be smaller after only two continual steps. The impact on the model size will be further studied in Section 5.1.1. The high-level architecture comparison can be found in Figure 3.2.

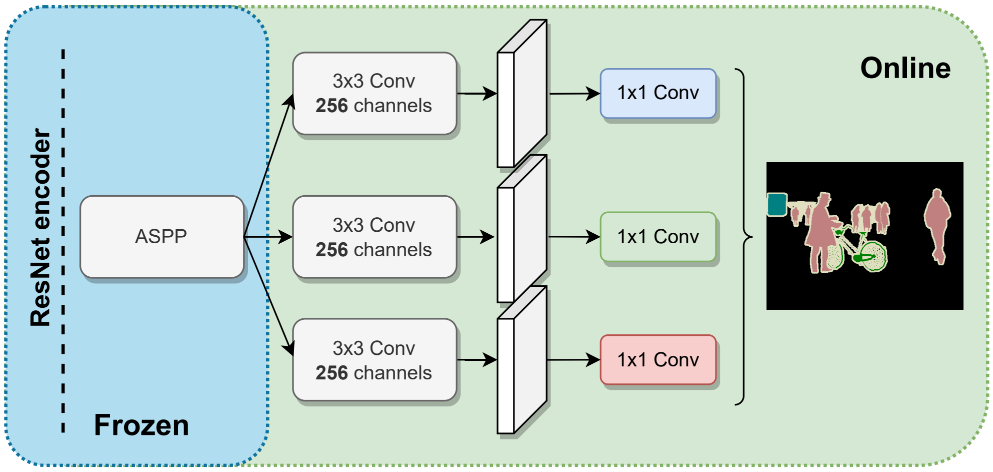
3.2 Regularisation

COMPRESSION phase tends to remove information that could be used in the future. It can be helpful since it increases the stability of the trained model and helps with overfitting. In Continual Learning, this is a less desirable property. We argue that in the majority of the cases, it is better to sacrifice a small chunk of performance at step $t = 1$ to maintain more features for steps $t > 1$ and achieve a better model overall. Thus, we test the addition of regularisation techniques in the form of DROPOUT and WEIGHT DECAY.

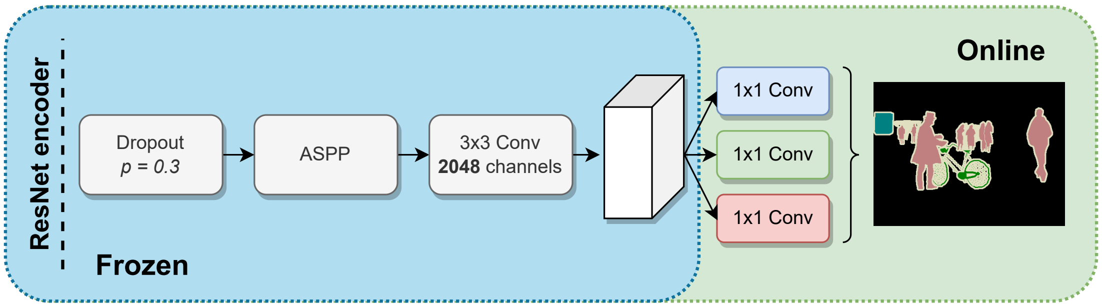
3.2.1 Weight decay

A popular technique to regularise the model's weights and prevent it from overfitting and over-compression is weight decay. Large weights cause large changes in output even for small changes in the inputs, causing sharp transitions in the functions. By replacing the original objective of minimizing the prediction loss on the training labels with the new objective of minimizing the sum of the prediction loss and the penalty term, we ensure that the weights are prevented from growing too large.

$$\mathcal{L}_{all} = \underbrace{\mathcal{L}_{BCE}}_{prediction} + \underbrace{\frac{\lambda}{2}||\mathbf{w}||^2}_{penalty} \tag{3.1}$$



(a) *High-level architecture of* ***SSUL*** *[4]. Each step has its own* 3×3 *convolution block that is trained in the online phase.*



(b) *High-level architecture of our* ***DCSS*** *model. One large* 3×3 *convolution is trained in the offline step and frozen. With 2048 channels, the* 1×1 *convolution has enough representation to outperform SSUL containing multiple modules.*

Figure 3.2: *Comparison of the SSUL and DCSS high-level architectures. The Blue area represents frozen parameters after the offline step. The Green area represents example modules trained in the online phase. Only one* 1×1 *convolution is trained at each step.*

Weight decay is a standard technique widely used in Deep Learning, with the ℓ_2 loss (Equation 3.1) being the most popular in Convolutional Neural Nets. However, we would like to test whether the increased value of the hyper-parameter λ can positively impact the sparsity of features, reducing the effect of COMPRESSION. Increasing the penalty can induce a more holistic approach to feature propagation, removing the reliance on salient features and increasing the importance of less prominent ones.

3.2.2 Dropout

Initially proposed by [46], Dropout randomly sets select activations of a layer to 0 with probability p, effectively disabling their contribution to the calculation of the output (Figure 3.4). As described by the authors, this can be thought of as having many ensembles of smaller models, with each iteration effectively creating a new model. Dropout introduces uncertainty that the model must account for by learning a more sparse feature representation since a particular feature can disappear with probability

```
from torch import nn

def separateHead():
  #256x256x3x3 parameters (0.59M) per step
  return nn.Conv2d(256, 256, 3, padding=1)

def standardSharedHead():
  #256x2048x3x3 parameters (4.71M) for all steps,
  return nn.Conv2d(256, 2048, 3, padding=1)

def separableSharedHead():
  #1024x(256/4)x3x3 + 1024x2048x1x1 parameters (2.68M) for all steps
  return nn.Sequential(
    # Separable Conv
    nn.Conv2d(256, 1024, 3, padding=1, groups=4),
    # PointWise Conv
    nn.Conv2d(1024, 2048, 1, padding=0),
  )
```

Figure 3.3: *Example PyTorch implementation of the different* HEAD *types. SSUL uses the first one, while DCSS uses the last one. We notice a significant difference in the depthwise separable implementation parameter count.*

p. Therefore, the uncertainty prevents the model from removing too much available information from the encoder, learning more robust and feature-rich embeddings. The improved representation should, in turn, allow the new classes to achieve better online performance.

The placement of the Dropout is also essential. Following [45] we conclude that the best place to put the Dropout layer in the context of semantic segmentation is between the encoder and decoder modules. DeepLabV3 uses a pre-trained backbone model like ResNet50 to compute high-level features, later decoded with the Atrous Spatial Pyramid Pooling (ASPP) [6] module. Since we train the ASPP from scratch, we can improve its sparsity by limiting the number of passed features from the encoder module with Dropout.

We test two popular Dropout approaches in CNNs: standard Dropout as proposed by [46] and channel Dropout, where the whole convolutional channel can be dropped with probability p. Moreover, [45] shows that using ScheduledDropout yields significant improvements to the training phase. ScheduledDropout linearly increases the dropout probability from 0 to p during training. During the exploration phase (GENERALISATION), we use smaller values of p to increase feature accumulation. Heavier regularisation is more useful during the exploitation phase (COMPRESSION), thus requiring higher values of p. In the end, we should obtain similar benefits while improving learning convergence.

Last, we suggest the removal of the Dropout after offline training. We cannot add new features in the online phase with model freezing. Therefore, limiting the access to

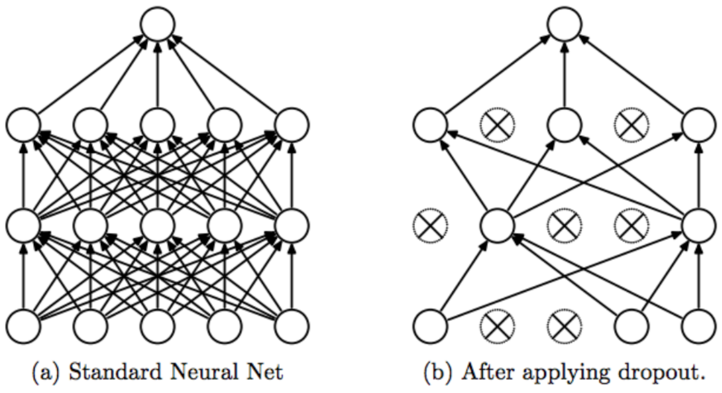


Figure 3.4: *Dropout illustration on a fully-connected neural network. Dropout reduces model's over-reliance on particular features. Image taken from [46].*

features will only restrain the model from using them while offering few benefits in generalisation since we are less likely to overfit with only a single trainable vector per class.

3.3 Restricted access to continual data

The initial problem formulation introduced by Cermelli et al. [3] is the first proper introduction of Continual Learning in semantic segmentation. SSUL and previous works propose important contributions that bring us closer to achieving usable, continually learning models. Nonetheless, the allowed training configuration might not encourage the efficient use of data. We are only constrained to using data with labels $\mathcal{Y}^t$ belonging to the current step t, but we can use them in any way we desire.

PLOP and MiB train each step for 30 epochs, while SSUL trains the model for 50 epochs. That means storing data for step t in memory and a longer training time to extract more information. Given that we already restrain ourselves from using any memory buffers to store previous class training examples, we believe this is an unnatural setting that promotes irrelevant models. Continual Learning models should aim to efficiently use the available data and aim for an approach similar to few-shot learning, where only a few examples for each class are available.

Therefore, we propose Restricted Class-Incremental Semantic Segmentation (RCISS), a new benchmark for Continual Semantic Segmentation, where each of the continual steps can be trained for only one epoch. This restriction effectively allows our model to see each online training example just once, simulating the limited chance of being able to store the online data, even for training during the same step. This scenario lies closer to the *lifelong learning* setting by Chaudhry et al. in their A-GEM work [5]. We evaluate SSUL and DCSS on RCISS in Section 5.4 to obtain initial results.

Chapter 4

Methodology

4.1 Dataset

With Continual Learning becoming a popular research topic, new datasets are being released regularly. There are not that many datasets available for Continual Semantic Segmentation, with the majority of the papers still using the popular PASCAL VOC or ADE20k. The most common approach to adapt them to the continual setup is to divide the dataset into the main part, trained in an offline fashion, that is later extended with additional training episodes consisting of the remaining classes.

It is worth noting that the approach taken to split the dataset has a crucial impact on the final performance, given that it changes the difficulty of training and the data distribution at each step. Unfortunately, there is no concrete guideline for the dataset splitting procedure common across visual tasks. Therefore, in this work, we will follow the assumptions made by [3, 12, 4].

In the case of Class-Incremental learning, that means splitting the dataset into multiple parts where the initial classes are learned offline, usually representing more than half of the available classes. In subsequent steps, new and only new classes are being shown, in batches or one by one. The new tasks test the model's ability to generalise new data while not forgetting the previous experiences.

4.1.1 PASCAL VOC 2012

The experiments are evaluated on the PASCAL VOC 2012 [14] semantic segmentation dataset, which contains $1,464$ train, $1,449$ validation and $1,456$ test images. The images have a 513×513 resolution. Each image is labelled on a pixel level, with 20 foreground object classes and one common background class. The dataset is augmented with the extra contour annotations [18] of the PASCAL VOC 2011 dataset, commonly referred to as PASCAL VOC 2012aug. This yields a total of $10,582$ training images. Following [12], we split the training set and used 80% for training and 20% for validation, with the testing set left untouched for model evaluation.

Following the existing evaluation techniques applied to our Class-Incremental problem

[3, 4, 12], we perform several different experiments concerning the different ways of splitting the dataset for offline and online episodes.

[3] introduces two different settings for Class-Incremental Semantic Segmentation: *Disjoint* and *Overlapped*. Both cases consider training examples where only the background class c_b and classes in $\mathcal{C}^t$ are labelled. In *Disjoint*, only the old and present classes can appear, while in *Overlapped* pixels can belong to any old, current and future classes. Thus, following the previous works, we also focus only on the Overlapped setting as it is more challenging and also more realistic, given that in the real world, we do not have access to any oracle method that can exclude future classes from the background.

The difficulty of the Class-Incremental challenge also depends on the number of steps and the number of new classes. Thus, our model is evaluated on five different scenarios generated from the PASCAL VOC, each with a varying level of difficulty:

- **10-1** (11 tasks)
- **15-1** (6 tasks)
- **5-3** (6 tasks)
- **19-1** (2 tasks)
- **15-5** (2 tasks)

The numbers in each scenario define the number of classes introduced at each step. For example, **5-3** (6 tasks) means learning 5 base classes at the offline step $t = 1$, followed by 5 incremental steps $t \in \{2,\dots,6\}$ introducing 3 new classes at each step, yielding 6 training steps covering all 20 classes for PASCAL VOC.

Since the order in which particular classes are being introduced to the model can have a significant impact on the final score, and the number of permutations for 20 classes in PASCAL VOC is larger than 10^{18}, we follow the simplification used by other works [3, 12, 4] and consider only the alphabetical ordering of classes.

4.1.2 Saliency maps for an *unknown* class

To help distinguish the background containing potential future classes from the true background, SSUL [4] proposed the use of an off-the-shelf salient object detector with short connections [23] to predict a region of interest. Thus, we follow their implementation and extend the labels of the PASCAL VOC training set to include the additional foreground class.

4.2 Evaluation metrics

The standard accuracy metric used to measure the performance of a segmentation model is the mean Intersection over Union, averaged across all classes or **mIoU** (Equation 4.1), where $0 \leq \text{mIoU} \leq 1$. For visibility reasons we will report our results in **IoU**, where $\text{IoU} = 100 * \text{mIoU}$ and $0 \leq \text{IoU} \leq 100$.

VOC 15-1 (6 tasks)		
0-15	16-20	all
77.31	36.59	67.61
77.66	**42.69**	**69.33**

Table 4.1: *Example evaluation results. Numerical values in bold represent the best IoU score in a corresponding column.*

$$mIoU = \frac{\sum_{i=1}^{|C|} \frac{A_i \cap B_i}{A_i \cup B_i}}{|C|} \tag{4.1}$$

For each class i, the intersection $A_i \cap B_i$ indicates the number of shared pixels found both in the prediction mask and ground truth mask. The union $A_i \cup B_i$ indicates the number of pixels found in either of the masks. The intersection and union ratio represents how well the predicted mask overlaps the target mask - a too small prediction mask will lead to the intersection being small, a too-large prediction will lead to the union being large, making the IoU smaller in both cases.

In an offline setting, the calculated overlap for each class is averaged out over the number of classes C, yielding the IoU. In our scenario with a mixture of offline learning (initial step) and the following online learning, the use of a single metric does not give the whole perspective on the final performance. Therefore, we will report the performance of our models with the IoU over the old classes, and the new classes learned online separately to distinguish the performance of learning from forgetting, as reported in [4, 12, 3]. Table 4.1 shows example results. The first row signifies the CISS scenario, in this case **15-1**. Values in columns show the IoU in old classes (0-15), new classes learned continually (16-20) and overall average (all). Values in bold show the highest score in the corresponding column.

Last, we compare the performance to jointly-trained models on all classes in set $\mathcal{C}$ in an offline manner. This method's performance can be considered as a reasonable upper bound on the IoU that the model can achieve [33]. The bound will be later referred to as JOINT.

4.3 Implementation details

The code for this project has been based on the last year's project and the implementations of PLOP [12] papers and SSUL [4], which is further based on the PyTorch implementation of DeepLabV3 and DeepLabV3+ by [15].

The codebase contains the implementation of the popular semantic segmentation models DeepLabV3 and DeepLabV3+, with an option to use MobileNetV2, ResNet50 or ResNet101 for the backbone. The standard PASCAL VOC image loader has been adopted to allow for splitting the dataset to perform Class-Incremental training.

Backbone The structure of DeepLabV3 [6] utilises encoder modules, known as backbones, to supply high-level features to the upper layers. The quality of the features depends on the size of the encoder. Therefore, most papers [3, 12, 4] use a large ResNet101 model as the backbone due to its extensive and accurate features at the cost of large compute requirements. To compare the results of DCSS, we also use ResNet101 while keeping in mind that a smaller encoder would have a better performance-to-cost ratio.

Segmentation head The ability to reuse pre-trained models for other tasks significantly boosts a range of tasks, including segmentation. However, the architecture of models designed to perform image classification requires slight modifications to preserve more spatial resolution of the image features for tasks like pixel-level classification. Typical ResNet model designed for classification compresses the features to $\frac{1}{256}$ of the input resolution, also described as having an *output stride* of 256. DeepLabV3 is typically used with an output stride of 16, maintaining the higher resolution for segmentation map prediction. To achieve that output stride, it uses atrous convolutions in place of the strided convolution in the ResNet backbone to maintain a large receptive field, similarly to striding, without the unwanted effect of reducing the resolution.

In DCSS, we use a modified version of the DeepLabV3 semantic segmentation model, as described in Section 3.1. Compared to SSUL and previous works, we use one SHARED decoder instead of the SEPARATE one with multiple HEAD modules for each step on top of the background and unknown classes. Therefore, we have one sizeable HEAD instead of $t+2$ medium-sized ones.

Data augmentation The setup used for data preprocessing consists of applying the same data augmentation as in SSUL [4], originally used for DeepLabV3 [6]. Each image is normalised channel-wise with a mean μ of $[0.485, 0.456, 0.406]$ and standard deviation σ of $[0.229, 0.224, 0.225]$. Additionally, a random scaling of the input images by a factor in the range $[0.5, 2]$ has been applied to each image in the training set, followed by a random horizontal flip and a random image crop of 513×513.

Learning rate policy A standard learning rate of 0.01 has been used, with the value decreased to 0.001 for the backbone when using pre-trained weights [6]. The lower learning rate for the encoder part ensures the model relies on the visual features extracted from the image and prevents it from overfitting, thus promoting healthier learning of the classifier. We use *poly*[1] learning rate scheduler, as proposed by the DeepLabV3 authors [6].

Training protocol Larger batch size values can have regularisation effects on the model. On the other hand, smaller batches can induce faster convergence to optimal values. Thus, we have used a batch size of 24 for all experiments compared to the size of 32 used by SSUL, as we have found more success with smaller values, especially for the continual steps. We use binary cross-entropy loss with sigmoid activation function, as in

[1]The scheduler has been called “poly” by the authors of DeepLabV3, even though the learning rate is not a polynomial [50].

SSUL. Pseudo-labels are added to labels in the continual steps, with an entropy-based threshold of $\tau = 0.9$. SSUL uses a smaller value of $\tau = 0.7$, but in our experiments, this diminishes the model's confidence by a large margin and reduces the performance of new class predictions.

The training phase of continual models consists of several distinct steps. Hence, the weights from a previous step are loaded in from memory, and the training details are reset to simulate the nature of learning in distinct episodes. Lastly, to reduce the impact of random seeds influencing the results [39] we have conducted each experiment at least three times on random seeds. Thus, each score reported in this work is a mean of the experiment results.

Multi-GPU setup Thorough hypothesis evaluation is a crucial step in each project. Unfortunately, the size of the semantic segmentation models like DeepLabV3 is a major obstacle in performing an extensive set of experiments. With the batch size recommended by the authors, the model requires 16GB of GPU memory to hold model weights and parameter gradients for the backpropagation. The largest GPU available for undergraduate students is RTX 2080Ti, with 11GB of memory. Therefore, for last year's project, we were forced to use smaller models (ResNet50 vs ResNet101) and evaluate the models with a smaller image crop size (256 vs 513), reducing the number of activations at the cost of decreased accuracy.

In this project, we have utilised the `torch.distributed` library to run each experiment on multiple GPUs in parallel. This extension has allowed us to replicate solutions from published papers and produce state-of-the-art results at the cost of increased compute utilisation and prolonged training.

With 4 GPUs running in parallel, each experiment takes approximately 2-4 hours, depending on the scenario. To accommodate for the large number of GPUs required and long training time, we have opted to use Informatics GPGPU cluster[2] as well as BayesWatch GPU servers[3], each containing tens of GPUS that we used to perform hyper-parameter tuning in parallel.

[2]`https://computing.help.inf.ed.ac.uk/teaching-cluster`

[3]`https://www.bayeswatch.com/gpu_resources/`

Chapter 5

Experiments

In this section, we compare the performance of our Dropout Continual Semantic Segmentation model (DCSS) with current state-of-the-art approaches on a selection of Class-Incremental tasks from Section 4.1.1. We analyse the impact of our shared HEAD module on performance, complexity and memory footprint. We verify the usefulness of the addition of the Dropout layer and perform a thorough hyper-parameter search, proving the importance of DCSS contributions with an extensive ablation study. Lastly, we suggest a new evaluation protocol for Class-Incremental Semantic Segmentation that lies closer to the lifelong learning philosophy.

5.1 Experimental results of DCSS

In Table 5.1 we observe that DCSS consistently outperforms other models while having a simpler and more extendable architecture that is also easier to train (Figure 3.2b). DCSS offers a significant improvement in learning of the continual steps while also preventing the *catastrophic forgetting* of old classes, even with multiple continual steps. In the most popular **15-1** scenario, DCSS achieves a 1.72% improvement over SSUL, a current state-of-the-art model. The biggest overall gain of over 3% can be found in **10-1** scenario that has a larger number of continual steps.

Most importantly, in all scenarios we have improved the mean score of the new, continual classes, up to 6% in the case of **15-1**. We believe that, although there is still a large IoU gap between classes learned offline and online, DCSS significantly improves the performance of online learning and brings it closer to the performance of offline learning. This gain can be a surprising result, considering that we are only training a single 2048-dimensional vector at each step.

Since model freezing prevents any new features from being added, we only learn a linear mapping of *existing* features during the continual phase. Therefore, we conclude that adding a new HEAD module at each step, like in SSUL, does not bring any noticeable improvements. On the other hand, it is cumbersome to train, requiring more training data, which the PASCAL VOC dataset does not have. These factors cause DCSS to outperform SSUL, with the regularisation further improving the scores (Table 5.2).

	VOC 10-1 (11 tasks)			VOC 15-1 (6 tasks)			VOC 5-3 (6 tasks)			VOC 19-1 (2 tasks)			VOC 15-5 (2 tasks)		
Method	0-10	11-20	all	0-15	16-20	all	0-5	6-20	all	0-19	20	all	0-15	15-20	all
ILT [37]	7.15	3.67	5.50	8.75	7.99	8.56	22.51	31.66	29.04	67.75	10.88	65.05	67.08	39.23	60.45
MiB [3]	12.25	13.09	12.65	34.22	13.50	29.29	57.10	42.56	46.71	71.43	23.59	69.15	76.37	49.97	70.08
PLOP [12]	44.03	15.51	30.45	65.12	21.11	54.64	17.48	19.16	18.68	75.35	37.35	73.54	75.73	51.71	70.09
SSUL [4]	71.31	45.98	59.25	77.31	36.59	67.61	**71.17**	45.38	52.75	**77.73**	29.68	**75.44**	77.82	50.10	71.22
DCSS (ours)	**73.34**	**50.20**	**62.32**	**77.66**	**42.69**	**69.33**	68.10	**48.83**	**54.34**	77.22	**36.85**	75.30	**77.49**	**51.49**	**71.30**
Joint (V3)	78.41	76.35	77.43	79.77	72.35	77.43	76.91	77.63	77.43	77.51	77.04	77.43	79.77	72.35	77.43
Joint (DCSS)	77.80	76.58	77.22	78.77	72.25	77.22	76.29	77.61	77.22	77.30	75.60	77.22	78.77	72.25	77.22

Table 5.1: *Main results for our DCSS model. We substantially outperform previous works in new classes (middle columns) in all scenarios and achieve improvements in combined scores in the more difficult scenarios with multiple tasks.*

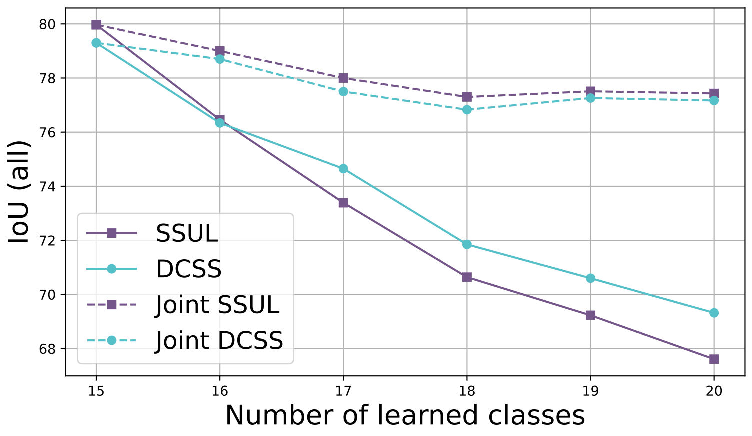
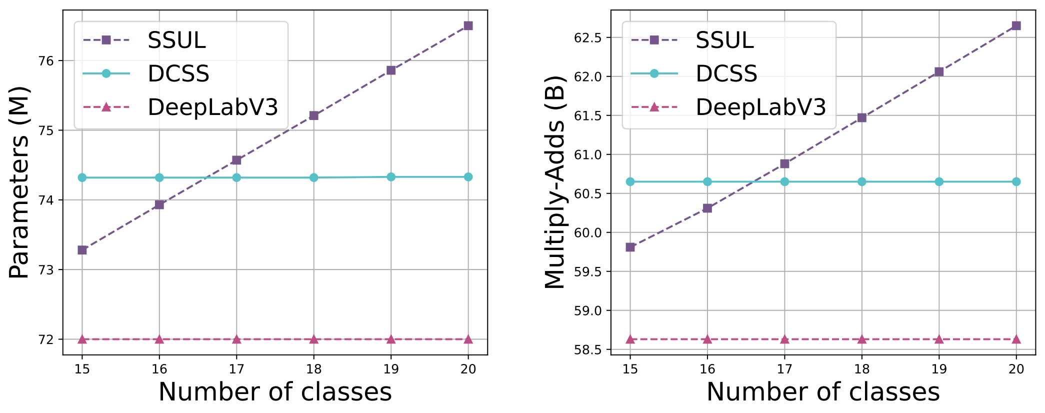


Figure 5.1: *IoU comparison of DCSS and SSUL with their respective Joint upper-bounds.*

Moreover, adding large modules goes against the nature of Continual Learning in which it is vital to be able to quickly learn from a small amount of data that might not be accessible for long. Therefore, relying more heavily on the existing feature representation gathered during the offline phase is likely beneficial when using model freezing instead of using large quantities of data to learn large convolutional layers for new tasks. Things can change in the very extreme case of a **2-2** scenario (10 tasks) and **2-1** (19 tasks) (Table A.1), where SSUL slightly outperforms DCSS. We attribute this to the small initial set of classes leading to insufficient amount of features. This finding is described further in the Appendix.

JOINT results show the practical upper-bound that we can achieve by training on all classes offline (Table 5.1, Figure 5.1). DCSS offers worse performance in joint training than SSUL, directly based on a DeepLabV3 model. This can be mainly attributed to adding a Dropout layer to DCSS and changing the classifier's size. Most changes applied to the models designed for optimal offline performance, like DeepLabV3, will be detrimental to the accuracy. To learn a more balanced embedding space, we have to sacrifice some of the performance during the offline phase. Indeed, in our case, we include a Dropout layer with a high probability p that decreases the model's



(a) *Model size comparison.* **(b)** *Model complexity comparison*

Figure 5.2: *Resource comparison in the* ***15-1*** *scenario. SSUL adds significantly more trainable parameters at each step, surpassing DCSS size and complexity in just 2 steps.*

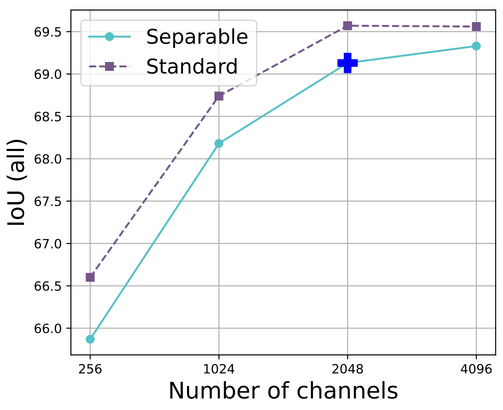
performance. Despite worse initial IoU, our experiments confirm that this trade-off is favourable for DCSS, which maintains better performance in the long run and shows a general direction for all Continual Learning models that rely on parameter freezing.

5.1.1 Size and complexity

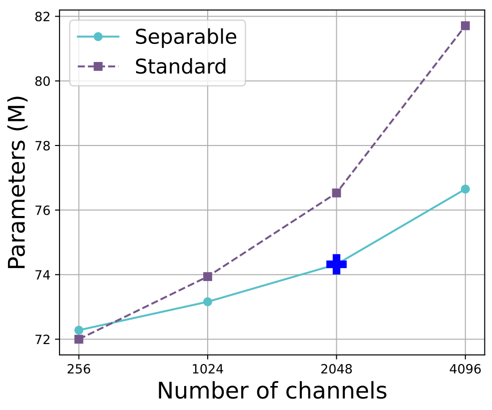
To compare our results to the ones produced by previous works, we have to constrain ourselves to a model of a similar learning capacity. Ideally, we are interested in comparable results for the offline step, showing that we are not increasing the capacity. A visible improvement in all online steps would suggest that our addition helps with continual learning and not just the segmentation capability. Figure 5.1 proves that we increase Continual Learning performance despite the decreased overall performance of DCSS in offline learning, further signifying the importance of our contributions to online learning.

What is most important, all of this is achieved with a smaller and easier to train model. In **15-1** DCSS has over 2M parameters less than SSUL (Figure 5.2), while in the extreme case of **10-1** the model is actually smaller by over 5M parameters and the difference grows linearly with each additional step.

As can be noted in Figure 5.2, SSUL implementation increases linearly in size and complexity with each additional continual step. The reason why at step $t = 1$ the DCSS is larger than standard DeepLabV3, despite both being trained in an offline fashion, is the requirement of having a separate classifier for the background class c_b and the unknown class c_u, which means that we already have three HEAD modules at $t = 1$, including the one for classes in $\mathcal{C}^{t=1}$. In contrast, increasing the size of the HEAD to produce 2048 channels with separable convolution as in DCSS, compared to 256 channels in SSUL, yields a significantly smaller model in the long run while achieving better results than SSUL.



(a) *IoU comparison*



(b) *Memory footprint comparison*

Figure 5.3: *Comparison of the DCSS model with a different number of channels and standard or separable convolution. (a) shows the achieved IoU while (b) compares models' size. The Blue cross shows the chosen setting for DCSS.*

5.2 Ablation study

5.2.1 Head size

We tested a range of configurations to choose an optimal size and configuration for the HEAD. In particular, DCSS with a HEAD with 256, 1024, 2048 or 4096 channels, with or without the depthwise separable convolution. Figure 5.3a shows a plotted graph of the results. The larger number of channels increases the performance, with the gains diminishing with more than 2048 channels. HEAD with a separable convolution achieves worse results across all sizes, conforming that we lose some performance when compressing convolution blocks.

The model parameter count in Figure 5.3b tells the other side of the story. Increasing the number of channels increases the size exponentially, undermining the use of a very large HEAD. However, we can recuperate some of the performance with the separable convolution. DCSS with 2048 channels and separable convolution, marked with a blue cross in the figures, achieves the best ratio of performance and efficiency and thus was used for all remaining experiments.

5.2.2 Model regularisation

The increase in HEAD size helps SHARED model be both more accurate and more effortless to train than SSUL, as seen in Figures 5.1 and 5.2 respectively. Notwithstanding, it relies heavily on a good embedding space learned by the model before freezing since we have only one trainable linear layer to predict the pixel's class probability. Inspired by the Information Bottleneck theory, we have attempted to reduce the impact of the COMPRESSION phase, assuming that this should maintain a richer and more balanced high-level feature representation, even if contained only in the final 2048 channels. Therefore, we have experimented with stronger regularisation in the decoder modules.

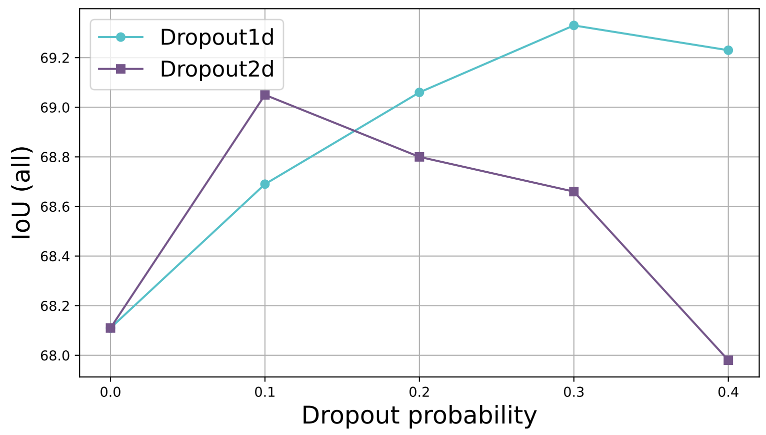


Figure 5.4: *IoU comparison of the Standard and Channel Dropout in **15-1** scenario.* DROPOUT2D *is more effective with small probabilities but gets outperformed by* DROPOUT1D *with $p = 0.3$, which we use for further experiments.*

		VOC 15-1 (6 tasks)		
Dropout	p	0-15	16-20	all
None	-	77.09	39.39	68.11
Standard	0.1	77.35	40.70	68.69
Standard	0.2	77.30	41.88	69.06
Standard	0.3	77.66	**42.69**	**69.33**
Standard	0.4	**77.72**	42.02	69.23
Channel	0.1	77.71	41.34	69.05
Channel	0.2	77.28	41.65	68.80
Channel	0.3	77.22	41.43	68.66
Channel	0.4	76.98	39.29	67.98

Table 5.2: *IoU comparison of the Standard and Channel Dropout in **15-1** scenario.* DROPOUT1D *with $p = 0.3$ achieves the best IoU scores in new tasks, proving the effect of regularisation of frozen models used for Continual Learning.*

Dropout Table 5.2 and Figure 5.4 present the scores of DCSS in the **15-1** scenario with a range of Dropout settings. We have tested standard Dropout (DROPOUT1D) as proposed in [46] as well as channel Dropout (DROPOUT2D) with a range of dropout probabilities p.

Dropout improves the model results in both new and old tasks. In the **15-1** scenario, the DCSS model was able to improve the overall IoU by more than 1% and the average IoU of the new classes by more than 2.5%. DCSS learned a better feature representation before freezing despite having an initially worse performance at $t = 1$. This is an influential discovery since Dropout does not change the complexity of the model while being very easy to add to existing architectures, signifying a natural enhancement to Continual Learning models.

In our experiments, we found that adding Dropout right after the ResNet encoder did help, confirming our predictions. Interestingly, DeepLabV3 already has one Dropout

Dropout p at step t		VOC 15-1 (6 tasks)		
$t = 1$	$t > 1$	0-15	16-20	all
0.3	0.3	76.90	40.92	68.31
0.3	0.1	77.14	41.45	68.65
0.3	0.0	**77.66**	**42.69**	**69.33**

(a) *Reduce Dropout at step t*

	VOC 15-1 (6 tasks)		
ScheduledDropout	0-15	16-20	all
✓	**77.66**	**42.69**	**69.33**
✗	77.54	39.60	68.50
-	-	-	-

(b) *ScheduledDropout*

Table 5.3: *Comparison of the Dropout strategies. (a) Removing Dropout for continual steps improves the performance of new tasks. (b) ScheduledDropout helps to learn better features while eventually still offering the benefits of Dropout.*

layer in the ASPP module with probability $p = 0.1$ suggesting that the authors found it helpful even in the offline training. We have tested an increased Dropout likelihood in the existing ASPP layers and additional Dropout layers closer to the final 1×1 convolution. Surprisingly, these additions did not seem to help and yielded worse results. We conclude that the location of the Dropout layer plays an important role, with the addition between a pre-trained encoder and an untrained decoder being a good starting point [45].

Figure 5.4 shows that DROPOUT1D outperforms DROPOUT2D, despite many papers claiming that DROPOUT2D should work better for convolutional networks [21, 45]. Interestingly, we have noticed increased sensitivity of the DROPOUT2D, which works best with a small probability of $p = 0.1$, with larger probabilities decreasing the overall performance. Standard DROPOUT1D works best with a probability $p = 0.3$, outperforming DROPOUT2D by a sizable margin. Thus, this is the setting that we have decided to use for DCSS.

The results follow our intuition, which suggests that dropping the whole channels removes a feature across the whole image, whereas DROPOUT1D can work inter-channel, which is vital in pixel-level tasks. Removing whole channels can prevent the propagation of features, while removing specific neurons can decrease the reliance on more salient parts of the image and promote more holistic attention.

We have also experimented with the DROPOUT1D probability during the continual steps. Since regularisation helps us learn better and more reliable feature representation, its importance is lessened with frozen parameters. Results in Table 5.3a prove that we can achieve better results with no Dropout at steps $t > 1$. By training just one final classifier layer at t, we are essentially learning the mapping of features to final probabilities, which in our understanding, does not benefit from Dropout. Therefore, all results reported for DCSS use DROPOUT1D with $p = 0.3$ at step $t = 1$ and probability $p = 0$ in continual steps $t > 1$.

Scheduled Dropout Although Dropout maintains more features for online learning, the increased entropy of the output features can cause problems with proper learning. Following IB theory, we are primarily interested in regularising the COMPRESSION phase of learning while leaving the GENERALISATION unconstrained. In other words, adding Dropout is beneficial in the exploitation phase as long as it does not interfere exceedingly with the exploration phase. Therefore, we have tested the use of Sched-

		VOC 15-1 (6 tasks)		
Decay	λ	0-15	16-20	all
ℓ_2	0.0001	**77.66**	**42.69**	**69.33**
ℓ_2	0.0005	77.43	38.74	68.21
ℓ_2	0.001	75.72	33.66	65.70

(a) *Weight decay*

	VOC 15-1 (6 tasks)		
Weight transfer	0-15	16-20	all
$Random \rightarrow \phi_c^t$	**77.66**	**42.69**	**69.33**
$\phi_{c_u}^{t-1} \rightarrow \phi_c^t$	77.58	40.89	68.84
-	-	-	-

(b) *Weight transfer*

Table 5.4: *Ablation study for the weight decay and weight transfer. (a) Increasing the weight decay decreases performance for new classes. (b) Impact of the weight transfer for the new classifier's parameters at step t.*

uledDropout [45], where the probability increases linearly from 0 to p during training (Section 3.2.2). Table 5.3b proves that ScheduledDropout improves the IoU by almost 1%, despite using the simplest, linear scheduling of the Dropout. Therefore we conclude that the optimal introduction of the regularisation can have a decisive effect on its success.

Weight decay DCSS uses a standard SGD optimiser with a momentum of 0.9 and ℓ_2 weight decay of 0.0001, as in DeepLabV3 [6]. Inspired by Dropout's promising regularisation results, we tested an increase in the weight decay λ parameter. Results in Table 5.4a show that any increase from the original value of 0.0001 used in most semantic segmentation models decreases performance, especially in the ability to learn new classes, while old classes learned in the offline step were mainly unaffected. Therefore, we must wisely choose the regularisation technique to achieve a sparse and feature-rich representation.

5.2.3 Weight transfer

SSUL relies heavily on the weight transfer from the unknown class ${c_b}^{t-1}$ to each of current classes $\mathcal{C}^t$. In the original paper [4], it is shown that for scenario **15-1** weight transfer makes a considerable difference, especially for the new classes (36.59 IoU vs 23.99 IoU). This step is probably required due to the large number of parameters trained at each step (over 0.5M) with insufficient data. Weight initialisation to the foreground predictor's weights assumes that the classifier is more or less ready from the get-go to recognise new classes.

In the context of the reduced set of trainable parameters in DCSS, we found that weight transfer is unnecessary and might prevent the model from finding an optimal solution. Table 5.4b shows that random initialisation outperforms weight transfer from both the background class and the unknown class.

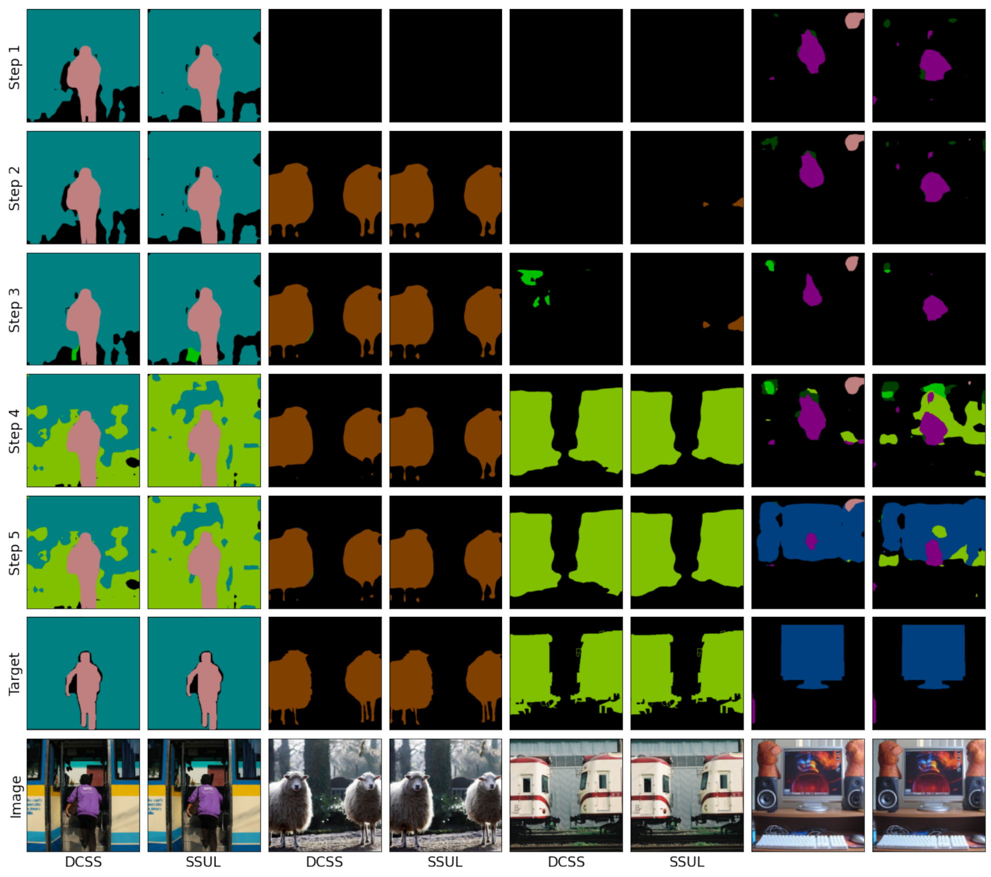


Figure 5.5: *Example predictions of DCSS and SSUL in the* ***15-1*** *scenario.*

5.3 Qualitative results

We examine the predicted masks learned continually to compare the results of SSUL and DCSS qualitatively. We see that both models can learn certain classes like `sheep` with ease. We notice DCSS behaving more stable under challenging conditions where the predictions are starting to become noisy. Class boundaries of similar classes like `bus` and `train` can be fuzzy in both cases, leading to mispredicted patches and severe degradation. The fuzziness is the drawback of using frozen models combined with binary cross-entropy: it is challenging to learn the actual distribution of the classes, causing under or over-prediction of classes with similar frequencies. However, DCSS tends to be less noisy than SSUL, which we attribute to having less trainable parameters, thus less overfitting while having higher quality features due to the Dropout layer.

5.4 New Protocols and Evaluation

We have seen the performance of DCSS on the tasks proposed initially by [3]. DCSS offers a better IoU score while being a generally cheaper model to train in the long run. We argue, however, that the evaluation protocol has an unrealistic assumption about the ability to store the continual training data and use it multiple times over the step t.

		VOC 10-1 (11 tasks)			**VOC 15-1 (6 tasks)**			**VOC 5-3 (6 tasks)**			**VOC 15-5 (2 tasks)**		
Method	Epochs	0-10	11-20	all	0-15	16-20	all	0-5	6-20	all	0-15	15-20	all
SSUL [4]	50	71.31	45.98	59.25	77.31	36.59	67.61	71.17	45.38	52.75	77.82	50.10	71.22
DCSS	50	73.34	50.20	62.32	77.66	42.69	69.33	68.10	48.83	54.34	77.49	51.49	71.30
SSUL [4]	1	73.23	38.10	56.50	**77.34**	32.44	66.65	69.90	27.52	39.63	76.68	**47.64**	69.76
DCSS	1	**73.78**	**43.26**	**59.24**	77.16	**37.46**	**67.71**	**68.44**	**38.84**	**47.30**	**77.46**	45.60	**69.88**

Table 5.5: *Experimental results on the Restricted Class-Incremental Semantic Segmentation protocol. Both SSUL and DCSS decrease their performance in new tasks, although the scores still lie in similar ranges. This proves that more training cannot extract missing features from the frozen model.*

Therefore, we also evaluate our DCSS model on the proposed Restricted Class-Incremental Semantic Segmentation protocol, where we train each continual step for only one epoch. The setting for the offline step $t = 1$ remains relaxed, with no constraints on its training protocol. Additionally, to account for the changes in the training protocol, we have removed learning rate scheduling, maintaining the learning rate at 0.01 throughout the training. Moreover, we have reduced the batch size in the continual phase from 24 to 4, effectively increasing the number of backpropagation steps in the hope that, while being noisy, it will better utilise the limited exposure to data.

Table 5.5 shows the result of our experiments on four scenarios in RCISS, similar to previous scenarios in standard CISS. Again, DCSS outperforms SSUL while having a simpler and more extendable architecture. We notice that both models struggle more with learning multiple classes at once (**5-3**), although DCSS manages to outperform SSUL in the continual classes by a large margin. Things change around in **15-5** where SSUL achieves better results in the new classes, the only case that this happened in our experiments. This difference can be explained by the different weight initialisation (Table 5.4b). Our experiments found that weight transfer from the unknown class is worse than random initialisation, but we see that it might be necessary for low-epoch scenarios. Moreover, a more regularised training signal containing multiple classes in a batch can usually benefit from having more training epochs, further increasing the difficulty of our protocol.

Notwithstanding, although we see a decrease in performance, we notice that the results are mainly in the same range as the models trained on the entire 50 epochs. We believe that this shows both the strengths of the frozen model architectures and the weaknesses of these implementations since no amount of training can extract missing features from the pre-computed feature representation.

Chapter 6

Discussion

This chapter provides a final analysis of the problem and the outcomes of our work. The conducted experiments proved that changes introduced by the DCSS model improve its performance versus the SSUL model that it was based on. We will discuss the impact of the results and highlight crucial aspects that we think will be important in the near and long-term future.

6.1 Frozen model with few trainable parameters

A shared HEAD with the same number of output channels initially decreases the model's performance, with fewer features available for the model to learn from (Figure 5.3a). This decline is expected and suggests why all previous works have used a separate HEAD. However, it is worth noting that none of the authors mentioned the adverse effect this approach has on the model's complexity. Adding more than 0.5M parameters at each step is not a sustainable policy assuming that the number of steps can become considerable.

However, the above information indicates also the strengths of DCSS. Using shared HEAD we can essentially freeze it after offline training and only focus on the classifiers ϕ_c^t. A single vector at step t is a lot simpler to train, and since we cannot realistically introduce new features without unfreezing the encoder, we achieve the same goal of mapping frozen representation to output probabilities.

Thankfully, the performance of the SHARED approach can be effortlessly recuperated with a larger number of channels while still making the model smaller after t steps compared to the SEPARATE implementation of SSUL. One larger module gives us also more predictability and flexibility when it comes to the expected final size of the model. Moreover, we can use depthwise separable convolution to decrease some of the parameter and MAdd costs while maintaining the crucial, large number of channels. Although separable convolution entails adverse effects on the predictive power, the experiments have proved that this is a worthwhile trade-off. We believe that separable convolution should be further researched and exploited in Continual Learning models with frozen parameters.

A frozen, shared HEAD means that we train only a single 1×1 classification vector per online class, causing new problems. A single trainable layer limits the learning capacity of the model by a large margin, usually causing underfitting. Nonetheless, we argue that this aspect can be beneficial in the context of architectures with frozen feature extraction modules, preventing sudden prediction shifts from appearing in similar classes like `cat` and `dog`.

6.2 Information Bottleneck theory and Dropout

Following the Information Bottleneck theory, we see that the COMPRESSION phase can impact the number of features available for future learning. Information Bottleneck theory suggests that Deep Learning models keep adding new features in the initial exploration phase but start removing unused or noisy ones in the exploitation phase. This behaviour is beneficially in the context of static, supervised learning but is counter-productive in Continual Learning. Adding new features tends to be expensive and can require multiple layers to extract complex relations. In contrast, removal of a feature connected with the COMPRESSION can happen rapidly and even in a single layer. Thus, we should aim to maintain a diversified representation as long as possible, reducing the COMPRESSION phase to its minimum.

COMPRESSION can also be connected to the overfitting, with overly compressed networks removing features not valuable for the training, causing over-reliance on the training set. Therefore, we can look into the numerous ways to combat overfitting, including early-stopping, weight decay and Dropout. Heavy regularisation can reduce the score of offline training, causing underfitting. We argue that with the requirement to maintain features for future use being of a higher priority, we should settle for a lower offline score with the hope of a better generalisation.

Weight decay did not offer the results that we were looking for. This misprediction can be partially attributed to the negative impact on the pre-trained weights of the ResNet encoder. Fine-tuning a pre-trained model with a more significant weight penalty term can cause undesirable incentives for the model to remove some of its existing features, clearly contradicting our initial plans. Despite this, we believe that there still is a place for a different approach to weight decay for Continual Learning with frozen models. However, we did not find a successful configuration that would yield promising results.

On the other hand, Dropout does not cause detrimental effects on the encoder module, as long as we place it in a strategic place. We can use the signal uncertainty produced by turning off random features to regularise the segmentation module without interfering with the encoder's feature generation. Moreover, the intelligent introduction of the Dropout using the ScheduledDropout strategy helps us achieve both uninterrupted initial learning and regularised final model.

We saw that ScheduledDropout combined with the larger, shared HEAD could make DCSS perform on-par with standard SSUL or DeepLabV3 model, albeit at the cost of space. Despite that, we have shown that it is a trade-off worth taking as DCSS outperforms any existing model in the ability to learn new tasks while still preventing overfitting.

6.3 Restricted access to data in Continual Learning

DCSS has an unquestionable advantage of being easy to adapt to new tasks. The large output of the frozen module allows for as little as a single layer to be trained, lessening the requirements for training and data. The clear advantage of being less reliant on the amount of continual data can also be a good candidate for few-shot learning.

Therefore, we proposed the Restricted Class-Incremental Semantic Segmentation benchmark to simulate more restricted environments where the data is seen just once as if it was streamed. RCISS follows the structure of the training protocol introduced by [3], which was also used in this work, but limits the number of epochs to just one, similar to lifelong learning.

Analysis of the results shows that both SSUL and DCSS deteriorate in their ability to learn new classes in RCISS. While DCSS still outperforms SSUL, the performance suggests that the restricted access to data has a destructive effect on already troublesome Continual Semantic Segmentation. Methods like A-GEM [5] could improve our models and should be tested in future works. Ultimately, we believe that this benchmark should be preferred to the one from [3] as it better relates to the nature of lifelong learning and focuses on the efficient use of data, an essential and often overlooked characteristic of Deep Neural Networks.

6.4 DCSS limitations

In Section 5 we have shown that DCSS improves upon the previous work of SSUL. Nevertheless, there is still plenty of room for improvement. The counter-intuitive use of a single trainable layer per continual step due to the use of shared HEAD, although successful, imposes strong limitations that will prevent the model from reaching optimal performance in the long term. Lacking any non-linearities, we essentially perform a simple linear mapping of the existing features learned in the offline phase to new tasks (Section 3.1.1). Therefore, Continual Learning is strictly limited to features obtained in the past. In scenarios where limited offline training is allowed (Table A.1), this leads to worse performance than SSUL. Moreover, model freezing prevents the *effects* of the informational collapse, but it does not *prevent* catastrophic forgetting. Thus, it should be considered a temporary approach, not a long-term solution.

In Section 3.2.2 we claim that introduction of Dropout reduces the effect of COMPRESSION of the features from a pre-trained encoder. However, we might not be using pre-trained models, thus limiting the effectiveness of our approach. Ultimately, the biggest gain could be achieved not with the reduction of COMPRESSION, but with a different approach to learning the representation that will increase the GENERALISATION. Recent trends tend to remove the reliance on supervised training for a more contextualised signal using self-supervised learning in the form of contrastive models or multi-modal data, where a more holistic type of learning must appear to capture the whole context. Altogether, these approaches should allow for encoders that are less task-specific, providing the required flexibility in Continual Learning.

Chapter 7

Conclusion

This project proposed a new DCSS method for Class-Incremental Semantic Segmentation (CISS) that has achieved state-of-the-art results on the PASCAL VOC dataset. To improve the model freezing approach suggested by SSUL, we have made two main contributions. The first one is scaling down the models' linear size growth with each additional continual step to a minimum. We have introduced a wider, shared HEAD module with depthwise separable convolution instead of the separate, smaller HEADS used by previous works. The importance of this change has been proven experimentally and showed performance gain while decreasing the overall model footprint. Second, inspired by the Information Bottleneck theory, we have studied the impact of heavier model regularisation during offline training to maintain more features for future learning. The addition of a Dropout layer has indeed increased the performance of DCSS, significantly improving the ability to learn new tasks at the cost of slight underfitting of the offline model.

Despite improving upon the results of SSUL and achieving current state-of-the-art results, we acknowledge certain limitations of our model. Although successfully used in our work, model freezing imposes unnatural constraints. The discrepancy between DCSS and the reasonable upper bound that we achieve with offline training is still significant and leaves plenty of room for improvement. Moreover, the current framework for CISS can promote unreasonable solutions that overfit to the specific task definition [3]. Therefore, we have introduced a new Restricted Class-Incremental Semantic Segmentation (RCISS) protocol, where the models are allowed to be trained only for epoch during the continual training, simulating a stream of data that cannot be stored in memory. While DCSS continues to outperform SSUL in that setting, both models achieve similar results to the original CISS setting despite limited training possibilities. RCISS shows the limitation of model freezing - no amount of training can recuperate the performance when missing crucial features.

Last, future work on this topic should be extended beyond the PASCAL VOC dataset, as any work on datasets not explicitly designed for Continual Semantic Segmentation should be treated as a temporary solution. New, dedicated datasets need to emphasize the issue of the background shift problem and introduce temporal and spatial locality of the data, which should spur new interest in this topic.

Bibliography

[1] Anurag Arnab, Shuai Zheng, Sadeep Jayasumana, Bernardino Romera-Paredes, Mans Larsson, Alexander Kirillov, Bogdan Savchynskyy, Carsten Rother, Fredrik Kahl, and Philip Torr. Conditional random fields meet deep neural networks for semantic segmentation: Combining probabilistic graphical models with deep learning for structured prediction. 2018.

[2] Eden Belouadah and Adrian Popescu. Scail: Classifier weights scaling for class incremental learning. pages 1255–1264, 03 2020.

[3] Fabio Cermelli, Massimiliano Mancini, Samuel Rota Bulò, Elisa Ricci, and Barbara Caputo. Modeling the background for incremental learning in semantic segmentation, 2020.

[4] Sungmin Cha, Beomyoung Kim, Youngjoon Yoo, and Taesup Moon. Ssul: Semantic segmentation with unknown label for exemplar-based class-incremental learning, 2021.

[5] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem, 2018.

[6] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation, 2017.

[7] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation, 2018.

[8] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. *arXiv preprint arXiv:2002.05709*, 2020.

[9] Xinlei Chen, Haoqi Fan, Ross Girshick, and Kaiming He. Improved baselines with momentum contrastive learning. *arXiv preprint arXiv:2003.04297*, 2020.

[10] Elliot J. Crowley, Gavin Gray, and Amos Storkey. Moonshine: Distilling with cheap convolutions, 2017.

[11] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[12] Arthur Douillard, Yifu Chen, Arnaud Dapogny, and Matthieu Cord. Plop: Learning without forgetting for continual semantic segmentation, 2021.

[13] Utku Evci, Vincent Dumoulin, Hugo Larochelle, and Michael C. Mozer. Head2toe: Utilizing intermediate representations for better transfer learning, 2022.

[14] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, January 2015.

[15] Gongfan Fang. Deeplabv3plus-pytorch. `https://github.com/VainF/DeepLabV3Plus-Pytorch`, 2019.

[16] Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A. Rusu, Alexander Pritzel, and Daan Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. 2017.

[17] Siavash Golkar, Michael Kagan, and Kyunghyun Cho. Continual learning via neural pruning. 03 2019.

[18] B. Hariharan, P. Arbeláez, L. Bourdev, S. Maji, and J. Malik. Semantic contours from inverse detectors. In *2011 International Conference on Computer Vision*, 2011.

[19] Tyler L Hayes, Kushal Kafle, Robik Shrestha, Manoj Acharya, and Christopher Kanan. Remind your neural network to prevent catastrophic forgetting. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[21] Mingjie He, Jie Zhang, Shiguang Shan, Xiao Liu, Zhongqin Wu, and Xilin Chen. Locality-aware channel-wise dropout for occluded face recognition. *IEEE Transactions on Image Processing*, 31:788–798, 2022.

[22] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015.

[23] Qibin Hou, Ming-Ming Cheng, Xiaowei Hu, Ali Borji, Zhuowen Tu, and Philip H. S. Torr. Deeply supervised salient object detection with short connections. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(4):815–828, apr 2019.

[24] Qibin Hou, Li Zhang, Ming-Ming Cheng, and Jiashi Feng. Strip Pooling: Rethinking spatial pooling for scene parsing. In *CVPR*, 2020.

[25] Yen-Chang Hsu, Yen-Cheng Liu, and Zsolt Kira. Re-evaluating continual learning scenarios: A categorization and case for strong baselines, 10 2018.

[26] Ye Huang, Qingqing Wang, Wenjing Jia, and Xiangjian He. See more than once – kernel-sharing atrous convolution for semantic segmentation, 2019.

[27] Dong hyun Lee. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks.

[28] Ahmet Iscen, Jeffrey Zhang, Svetlana Lazebnik, and Cordelia Schmid. Memory-efficient incremental learning through feature adaptation. 04 2020.

[29] Abhiram Iyer, Karan Grewal, Akash Velu, Lucas Oliveira Souza, Jeremy Forest, and Subutai Ahmad. Avoiding catastrophe: Active dendrites enable multi-task learning in dynamic environments, 2022.

[30] Ronald Kemker and Christopher Kanan. Fearnet: Brain-inspired model for incremental learning. In *International Conference on Learning Representations*, 2018.

[31] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks, 2016.

[32] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[33] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(12):2935–2947, 2018.

[34] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2014.

[35] Vincenzo Lomonaco and Davide Maltoni. Core50: a new dataset and benchmark for continuous object recognition. In *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78, pages 17–26, 2017.

[36] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning, 2017.

[37] Umberto Michieli and Pietro Zanuttigh. Incremental learning techniques for semantic segmentation. In *International Conference on Computer Vision (ICCV), Workshop on Transferring and Adapting Source Knowledge in Computer Vision (TASK-CV)*, 2019.

[38] Seyed Iman Mirzadeh, Arslan Chaudhry, Dong Yin, Timothy Nguyen, Razvan Pascanu, Dilan Gorur, and Mehrdad Farajtabar. Architecture matters in continual learning, 2022.

[39] David Picard. Torch.manual_seed(3407) is all you need: On the influence of random seeds in deep learning architectures for computer vision, 2021.

[40] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H. Lampert. icarl: Incremental classifier and representation learning, 2016.

[41] Mengye Ren, Renjie Liao, Ethan Fetaya, and Richard S. Zemel. Incremental few-shot learning with attention attractor networks, 2018.

[42] Anthony V. Robins. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connect. Sci.*, 7:123–146, 1995.

[43] Andrew Michael Saxe, Yamini Bansal, Joel Dapello, Madhu Advani, Artemy Kolchinsky, Brendan Daniel Tracey, and David Daniel Cox. On the information bottleneck theory of deep learning. In *International Conference on Learning Representations*, 2018.

[44] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017.

[45] Thomas Spilsbury and Paavo Camps. Don't ignore dropout in fully convolutional networks, 2019.

[46] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[47] A.J. Storkey. When training and test sets are different: Characterising learning transfer. In Candela Sugiyama Schwaighofer Lawrence, editor, *Dataset Shift in Machine Learning*, chapter 1, pages 3–28. MIT Press, 2009.

[48] Naftali Tishby and Noga Zaslavsky. Deep learning and the information bottleneck principle, 2015.

[49] Gido M. van de Ven and Andreas Savas Tolias. Three scenarios for continual learning. *ArXiv*, abs/1904.07734, 2019.

[50] Yanzhao Wu, Ling Liu, Juhyun Bae, Ka-Ho Chow, Arun Iyengar, Calton Pu, Wenqi Wei, Lei Yu, and Qi Zhang. Demystifying learning rate policies for high accuracy training of deep neural networks, 2019.

[51] Yue Wu, Yinpeng Chen, Lijuan Wang, Yuancheng Ye, Zicheng Liu, Yandong Guo, and Yun Fu. Large scale incremental learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 374–382, 2019.

[52] Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks. In *International Conference on Learning Representations*, 2018.

[53] Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society Series B*, 68:49–67, 02 2006.

[54] Sergey Zagoruyko and Nikos Komodakis. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer, 2016.

[55] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, page 3987–3995, 2017.

[56] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6230–6239, 2017.

[57] Bolei Zhou, Hang Zhao, Xavier Puig, Tete Xiao, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Semantic understanding of scenes through the ade20k dataset, 2016.

Appendix A

Supplementary materials

A.1 Additional experimental results

In Section 5 we have presented the results on 5 main tasks, in particular **10-1**, **15-1**, **5-3**, **15-5** and **19-1**. We notice in Table 5.1 that DCSS surpasses the results of other works. We have also carried additional experiments as introduced by Cha et al. [4], shown in Table A.1. DCSS performs slightly worse in **2-1** and **2-2** scenario. The shared HEAD module of DCSS means that we only have a single 1×1 trainable vector per class. In these extreme scenarios, the frozen encoder does not contain enough information and can benefit from the additional trainable layer of SSUL. We conclude that the number of initial classes used for offline training plays a crucial role in further offline training.

A.1.1 Per-class performance

Table A.2 shows the summarized results of DCSS model on the PASCAL VOC dataset by each class.

A.1.2 Confusion matrix of DCSS

Figure A.1 shows the confusion matrix of the DCSS model in the **15-1** scenario. We can notice that the model correctly predicts most of the pixels, with a large part of the mistakes caused by over-prediction of the background class.

	VOC 10-1 (11 tasks)			**VOC 5-1 (16 tasks)**			**VOC 2-1 (19 tasks)**			**VOC 2-2 (10 tasks)**		
Method	0-10	11-20	all	0-5	6-20	all	0-2	3-20	all	0-2	3-20	all
PLOP [12]	44.03	15.51	30.45	0.12	9.00	6.46	0.01	5.22	4.47	24.05	11.92	13.66
SSUL [4]	71.31	45.98	59.25	69.32	40.38	48.65	**62.35**	**34.32**	**38.32**	**62.38**	**42.46**	**45.31**
DCSS (ours)	**73.34**	**50.20**	**62.32**	**70.22**	**40.59**	**49.05**	61.06	32.60	36.67	56.94	41.36	43.59

Table A.1: *IoU results on the more extreme scenarios with low initial number of classes. DCSS struggles if the frozen model was trained only on a few classes.*

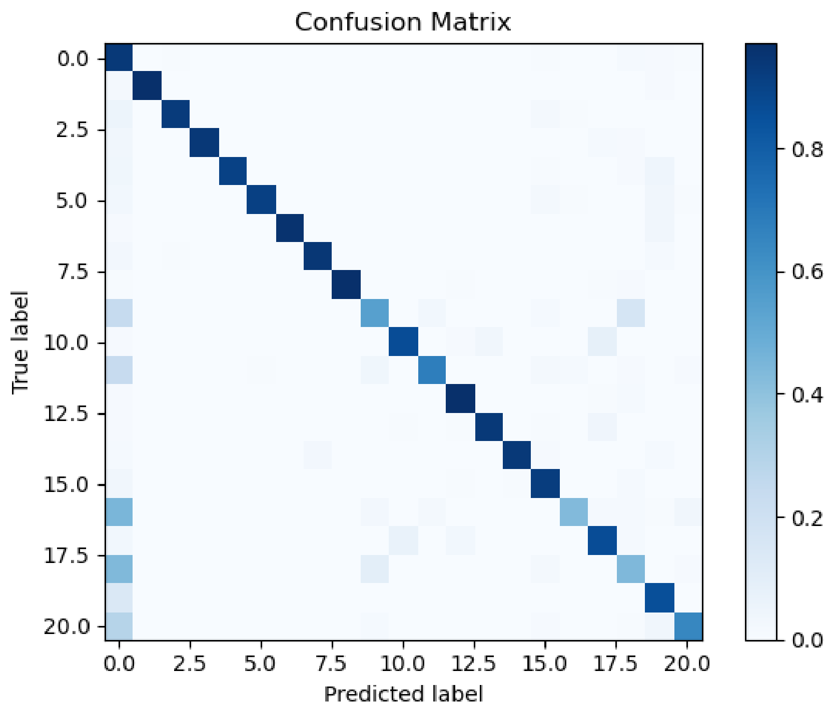


Figure A.1: *Confusion matrix of the accuracy of DCSS model in the **15-1** scenario.*

	bg	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	all
10-1 (11 tasks)	88.76	83.91	38.36	88.48	65.01	79.16	87.63	88.08	85.94	33.34	68.05	29.83	70.31	46.79	73.45	79.43	27.85	53.63	24.66	47.80	48.22	62.32
15-1 (6 tasks)	90.54	88.97	37.15	89.05	70.35	81.02	86.78	88.38	94.17	35.71	80.14	56.00	89.86	84.27	84.66	85.41	30.98	58.78	25.15	55.97	42.59	69.33
5-3 (6 tasks)	87.75	76.87	32.98	83.85	54.33	72.79	53.00	71.94	73.18	10.94	49.95	24.15	65.78	47.90	67.00	77.42	24.84	54.23	21.15	48.31	42.76	54.34
19-1 (2 tasks)	92.63	89.36	39.67	89.05	73.74	80.75	92.23	87.17	92.01	40.42	84.27	57.49	90.49	83.64	85.65	84.82	58.85	83.04	51.11	87.99	36.85	75.30
15-5 (2 tasks)	91.01	86.45	39.14	88.22	68.61	79.07	93.03	86.97	92.31	34.85	79.66	57.84	89.49	83.01	85.46	84.81	35.14	64.54	30.73	74.29	52.74	71.30

Table A.2: *Details of PASCAL VOC IoU performance per task per class.*

A.2 Additional discussions

DCSS proposes distinct changes that yield a more reliable model (Table 5.1). Notwithstanding, these changes are only a fraction of the applicable approaches, even when considering only the approaches with a frozen model.

A.2.1 Residual connection

Initially, we have tested the addition of skip-connections from low-level layers to the final decoder module, providing low-level features to the classifier. This is a similar approach to the one of DeepLabV3+ [7], although there, the low-level features are used to inform the offline model of more precise mask boundaries.

The idea behind residual connections for Continual Learning stems from the same Information Bottleneck theory as discussed before. Feature-rich representation in the output layer of a frozen model is a crucial requirement for the successful learning of new classes. Therefore, we have experimented with increasing the capacity of the final layer by doing a bit more than simply extending the number of channels as in DCSS.

Additionally, residual connections benefit from being separated from the main model trunk. This separation can be used by having a separate skip-connection for each continual step, thus enabling the model to look further into the feature representation and potentially reduce the effect of COMPRESSION. Unfortunately, we did not have any success with this naive addition, potentially because the residual connections increased the GENERALISATION by learning more features but did not weaken COMPRESSION.

A possible option suggested in Head2Toe [13] is the use of group lasso [53] to select specific parameters across the layers for model fine-tuning instead of relying only on the output features. Dynamic selection of features from the frozen model could prevent information bottlenecks in the models. We conjecture that this approach could extend and improve the naive residual connections and should be researched in the feature.

A.2.2 Self-supervised learning

In self-supervised learning, the model is trained using a proxy task, like in contrasting learning. The idea behind the proxy task is that we can generate a similar task that is easier to train or can be trained on a large quantity of unlabelled data that we utilise to produce a proxy supervision signal. Therefore, to obtain results on par with supervised learning, we must learn a better representation using more varied data that will do just as well or better on the target dataset. Self-supervised learning implies a very sparse representation containing many potentially useful features.

We can use this characteristic of self-supervised learning to our advantage in Continual Learning, where such training of the offline step can yield a more robust model with a large quantity of data. Although this type of training is expensive and data-hungry, it could most likely lead to state-of-the-art results using the frozen model approach. Temporarily, even using a self-supervised pre-trained model like MoCoV2 [9] or SimCLR [8] could be a viable option for inducing a sparser feature representation during fine-tuning.

A.2.3 Adaptable frozen model

Despite having frozen weights to maintain the learned prediction ability for past tasks, there are ways to overcome this excess rigidity for new classes. Using adaptive modules like Squeeze-and-Excitation (SE) in the backbone can offer us increased plasticity while maintaining previous representation. The squeeze element of SE captures the details of the channels with global average pooling and passes the signal through non-linearities and fully-connected layers. The vector produced by the Squeeze procedure describes the channels' importance, which is then applied in the Excitation phase. In essence, we can extract the value of each filter and scale the detected features accordingly.

Although SE would not allow us to add new features to the model, it could extend the flexibility of representing less salient features that could be crucial for new tasks. Thus, each task could learn its independent Squeeze vectors. During the forward pass, we would be required to dynamically select the scaling vectors not to repeat the calculation for each task. In theory, the dynamic selection should maintain efficiency while offering additional predictive power for Continual Learning.

Moreover, learning new features usually occurs across several layers and might require new channels to prevent catastrophic forgetting. Some works focus on enabling the dynamic channel activation depending on the class or introduction of new, task-specific modules across the depth of the model. However, in the short term, we are probably left with the freezing approach in which DCSS' approach achieves visibly better results. Therefore, we claim that we should aim to provide better features with the frozen model that can be quickly and efficiently utilised for continual learning rather than trying to add new features on top of the offline representation.