2-1-2023 1:00 PM

Look-Ahead Selective Plasticity for Continual Learning

Rouzbeh Meshkinnejad, *The University of Western Ontario*

Supervisor: Mohsenzadeh, Yalda, *The University of Western Ontario*
: Daniel, Lizotte, *The University of Western Ontario*
A thesis submitted in partial fulfillment of the requirements for the Master of Science degree in Computer Science

Abstract

Recent progress in contrastive representation learning has shown to yield robust representations that can avoid catastrophic forgetting in continual learning tasks. Most of these methods avoid forgetting by limiting changes in components of the deep neural network (DNN) that hold significant information about previously seen tasks. While these previous methods have been successful in preserving aspects of learned parameters believed to be most relevant for distinguishing previous classes, the retained parameters may be overfitted to seen data, leading to poor generalization even though "forgetting" is avoided. Inspired by modulation of early sensory neurons by top-down feedback projections of cortical neurons in perception and visual processing, we propose a class-incremental continual learning algorithm that identifies and attempts to preserve weights that contribute to the model performing well on new unseen classes by assessing their generalizability on a small predictive batch of the next episode of data. With experiments on popular image classification datasets, we demonstrate the effectiveness of the proposed approach and explain how using the model's first encounter with new data to simulate a feedback signal for modulating plasticity of weights provides more information for training compared to using the loss value alone, and how it can guide the model's learning through new experiences.

Keywords: Continual Learning, Representation Learning, Neuromodulation

Summary for Lay Audience

Continual learning is the field of training a neural network on a sequence of tasks defined by their corresponding datasets. A major issue that this field attempts to solve is catastrophic forgetting, when a neural network's performance on previous learned tasks rapidly decreases, in contrast to how humans learn. Previous work has made significant progress in providing neural networks that output representations (a vector) for each data sample (an image) that are robust to forgetting. Most of these methods avoid forgetting by limiting changes in components of the neural network that hold significant information about previously seen tasks. While these previous methods have been successful in preserving aspects of learned parameters believed to be most relevant for distinguishing previous classes of data, the retained parameters may be working well on the data they were trained on but perform poorly on similar data that they have not seen and lacking generalizability, even though "forgetting" is avoided. Inspired by modulation of early sensory neurons (near eyes) by top-down feedback of higher level neurons in the brain when processing visual stimuli, this thesis proposes a continual learning algorithm that identifies and attempts to preserve neurons and connections that contribute to the model performing well on new unseen classes by assessing their performance on a small subset of the next episode of data. With experiments on popular image classification datasets, the effectiveness of the proposed approach is demonstrated. It is also explained that how using the model's first encounter with new data to simulate a feedback signal for modulating the allowance of change in neurons (plasticity) provides more information for training compared to using the loss value (used for training of the network and is indicator of performance) alone, and how it can guide the model's learning through new experiences.

Acknowledgements

I want to thank my supervisors, Drs. Yalda Mohsenzadeh and Dan Lizotte for their guidance, support, and much needed patience throughout the completion of this thesis.

I would especially like to thank Dr. Jie Mei for her guidance and insight throughout this thesis and other projects we did together during my studies. Dr. Mei inspired me and showed me a way out whenever I found my self stuck in solving the research challenges. Working with you was the highlight of my studies at Western. Thank you.

I would like to thank the wonderful group of people I met at Western and especially Mohammad, Vahid Reza, and Anthony for their help and advice. We shared a lot of good, funny, and enlightening moments together that I will remember for the years to come.

This thesis was funded in part by a generous scholarship from Vector Institute as well as the Western Graduate Research Scholarship. I would like to thank both Western and Vector Institute for supporting my research.

Finally, I want to express my deepest gratitude to my parents and my sister for providing me with unconditional support and continuous encouragement to trust my self and believe in what I am doing and my research. This thesis would not have been possible without your support. Thank you for everything.

To the loving memory of my grandmother

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context

Deep Neural Networks (DNN) are known to be powerful function approximators. While showing prominent success in tasks like Image Classification, Image Segmentation, and Natural Language Understanding, they require a large and diverse dataset of training examples to effectively learn generalizable patterns and perform well on held-out datasets. Moreover, their usecase typically consists of very specific tasks with a large dataset, while the most useful problems are known to be more general (for example a self-improving assembly line that detects anomalies, assesses quality, receives feedback, and makes decisions regarding the production line to make the end-product better) with relatively few training examples available. Meanwhile, biological organisms learn from experience and adapt continually over their lifetime. Learning occurs using much less training data, and the biological system can generalize the acquired knowledge to unseen data [45]. Researchers have therefore questioned whether a DNN can learn and adapt continually to different tasks; however, early experiments showed that when trained on new tasks, DNNs performance on previous tasks rapidly deteriorates, prompting scientists to term this phenomenon *catastrophic forgetting* [61].

The field of Continual Learning (also known as Sequential, Incremental, or Lifelong Learning) studies how neural networks can keep learning new tasks sequentially, while previously acquired knowledge is not forgotten. Forgetting in this context is defined as a decrease in performance metrics on a previously learned task as the network is trained on subsequent tasks. Continual learning is studied in various problem settings with varying assumptions about task characteristics (with or without task identification numbers, classes in tasks, same classes but new instances/domain) and varying assumptions about the available training data at different points in the process (each data sample seen only once, vs. the option to train on a task's data multiple times). An example of a Continual Learning problem can be illustrated by defining a sequence of tasks. Imagine a dataset of handwritten digit images including only the digits 0 and 1. A neural network can be trained on this dataset to distinguish images of digit 0 from 1 with high accuracy. Now imagine that a second dataset of handwritten digit images of digits 2 and 3 becomes available. The goal of Continual Learning is to train the neural network on this dataset to not only discriminate images of 2 and 3, but also images of digits it has previously seen, namely 0 and 1. While trying to reach this goal, a limitation should be considered: The

previous dataset, one that had digits 0 and 1, may not be available or the access to it may become very limited. The field of Continual Learning studies algorithms that can train a model to learn the new dataset while not forgetting previously learned dataset(s) even though the access to previous dataset(s) is limited. Furthermore, subsequent tasks can also become available. After training the network on images of 2 and 3, a new dataset of images of digits 4 and 5 can become available while access to previous datasets becomes limited.

1.2 Motivation

Virtually all of the previous techniques introduced in Continual Learning fall into one or more of the following categories:

- Model growing
- Rehearsal of samples stored in a memory
- Discrimination and conservation of learned knowledge via methods like
  - Regularization of parameters deemed important,
  - Parameter isolation, or
  - knowledge distillation
- Meta-Learning
- Incremental Representation Learning.

Except for Meta-Learning approaches, the rest of the techniques use some form of regularization to preserve knowledge from previous tasks in order to prevent forgetting. In order to do so, these methods use either data samples from the previous tasks which are stored in memory, or the model's parameters, which are derived from previous tasks' data. While successful to some extent, these methods have a significant and inherent drawback: *Focusing only on preserving previously acquired knowledge neglects generalizability on new tasks that share some underlying structure with previously learned tasks*. In this thesis we aim to introduce a Continual Learning setting and framework that attempts to preserve knowledge not only based on the acquired knowledge of previous tasks (and consequently the parameters that may hold this knowledge), but also what parts of this knowledge and which parameters are likely to generalize to upcoming tasks. In doing so, we are going to define a metric that quantitatively measures how important each network parameter is for a good performance on past tasks and the next upcoming task.

In biological brains, a mechanism called *Neuromodulation* enables neurons to communicate with each other via neurotransmitters in different spatial and temporal scales. Neuromodulation is believed to play a significant role in mitigating catastrophic forgetting [45, 6, 22, 93, 63, 19, 59] by selectively modifying the connectivity of neurons, attention, and plasticity [41, 21].

Inspired from Neuromodulation and its role in behavioral adaptation, a main aim of this thesis is to explore various methods to leverage Neuromodulation for continually learning visual tasks.

1.3 Contributions

The main contributions of this thesis are as follows:

1. A new approach/setting for continual learning that uses a predictive batch of new upcoming data as well as data from past tasks to identify *salient* parameters.
2. Two new *saliency* methods that combine to find salient parameters.
3. Three new methods that leverage saliency information with inspiration from neuromodulation to reduce change in parameters that are performing well on past and a predictive batch of new upcoming data.

Each novel setting and method has been implemented and evaluated on popular vision datasets in order to analyze the benefits of Neuromodulation-Inspired mechanisms on adaptation to new data and mitigating catastrophic forgetting.

1.4 Scope

This thesis focuses on continual learning in visual tasks (supervised image classification). While continual learning is also studied in other areas such Reinforcement Learning, the methods review and analyses here are mostly focused on computer vision, and the contributions are also in the field of computer vision and whether the proposed methods are applicable to other areas was not investigated.

1.5 Organization

This thesis is organized as follows: First, in Chapter 2, preliminary terms and definitions required to understand the subsequent chapters are explained. Next, in the Literature Survey chapter (Chapter 3), previous work in Continual Learning and Neuromodulation-Inspired methods is explored and described. In the Methods section (Chapter 4), novel techniques for identifying salient parameters and preserving them in the new setting are explained. Then, the Results are presented in Chapter 5 followed by Discussion and Conclusion (Chapter 6).

Chapter 2

Background

In this chapter, terminology and preliminary definitions needed to understand the remaining part of this thesis are laid out. First the general concepts of Supervised Machine Learning and the training process of Neural Networks is explained. Then the focus is brought to a specific type of neural networks called Convolutional Neural Networks (CNNs) that are used in Computer Vision tasks, and a specific type of CNNs called Residual Networks which are employed as one of the main architectures in this thesis. Moreover, in order to properly introduce Continual Learning, first Transfer Learning, Multi-Task Learning and Meta-Learning paradigms are described. A definition of Continual Learning is then provided, followed by assumptions used to simplify and more easily formulate the Continual Learning problem. Next, Representation Learning and a special form of it called Contrastive Learning are explained. Finally, concepts of Neuromodulation, Neuroplasticity, and Top-Down visual feedback are introduced as the main inspirations for the methods used in this thesis.

2.1 Supervised Machine Learning

Supervised machine learning methods are designed to solve a problem of predicting accurate labels given a set of input features. Specifically, a labelled dataset is given, where each data point is composed of a set of input features and a label. The goal is to train a model using the given dataset that takes a set of input features (possibly from outside of the training dataset) and predicts the label accurately.

2.2 Neural Network Learning

This thesis focuses on supervised learning of artificial neural networks. An artificial neural network is essentially a set of interconnected artificial neurons. Each so called neuron computes a non-linear function of its inputs. Since neurons are usually stacked into layers, the inputs to a neuron are usually the output (called activations) of previous layer neurons or the input to the network itself. Specifically, the ith neuron in layer l computes the following function:

$$a_i^{(l)} = f(\sum_j w_{i,j}^{(l)} a_j^{(l-1)} + b) \tag{2.1}$$

where f is a non-linear function and b is a constant named bias. $w^l_{i,j}$ is the weight of the connection between the ith neuron in layer l and the jth neuron in the previous layer. $a^{(l)}_i$ is the output (or activation) of the described neuron. When the number of stacked layers becomes large, such a network will be called a *deep* neural network. Deep Neural Networks have shown to be very powerful function approximators, presenting impressive results and performance on a variety of tasks in Computer Vision and Natural Language Processing, among others.

Artificial neural networks are trained to find an optimal set of weights w and biases b, such that a loss function is minimized. In order to asses a network's performance on a dataset, a loss function is defined. A function of network weights, this loss function will have a large value if the network is making poor predictions, whereas its value will be low if the network is making accurate predictions. The optimization of weights w can be done in different ways, such as evolution algorithms or finding a closed-form solution analytically. A method that is used more than the others, however, is called *gradient descent*, where the partial derivatives of the loss function with respect to each weight $w^l_{i,j}$ are computed. Next, each weight is modified in the opposite direction of the gradient, hence the term gradient descent. Each update of the weights can generally be written as:

$$w^l_{i,j} = w^l_{i,j} - \gamma \frac{\partial \ell(w, ...)}{\partial w^l_{i,j}} \tag{2.2}$$

where $\ell(w, ...)$ is the loss function and γ is a "hyperparameter" known as the learning rate. Hyperparamters are those that control certain aspects of the training algorithm and the optimization process, but are not among the network weights.

The process in which partial derivatives are calculated layer by layer, top to bottom, is called backpropagation [77]. There can be other update rules to the weights with different *optimizers*. Improving optimizers and finding ones better suited for specific classes of problems is still an active area of research. Overall, given a training dataset, a loss function, and an optimizer (e.g. gradient descent), a network can be trained via the backpropagation algorithm.

2.3 Convolutional Neural Networks

Convolutional Neural Networks (CNN) [49] are a class of neural networks that are mostly used for visual inputs. In the context of visual inputs, CNNs reorganize weights of a neural network into *filters* or *channels*, and apply the same weight across different positions of the input, much like the convolution operation. One or more of such filters will make up a *convolutional layer*, and stacking one or more convolutional layers will result in a CNN. After training, each filter or channel will learn to recognize a pattern, with the lower layers of CNN responding to simple patterns like edges and higher layers reacting to more complex patterns (like certain features in objects). As a result, each convolutional layer in a CNN is capable of combining the inputs and recognizing more complex patterns.

2.4 Residual Networks

In the first few years of developing and training CNNs, two observations were made:

1. Deeper (more layers) CNNs perform better when trained on large datasets [84, 89].

2. Deep CNNs are hard to train, and the optimization process may diverge [29].

While tricks such as normalized initialization and intermediate normalizing layers added to deep CNNs [51, 38] were successful in stabilizing training and addressing problems such as vanishing gradients, a degradation problem still remained [35]: Increasing network depth would cause the accuracy to get saturated and then rapidly degrade. The source of this problem was verified not to be over-fitting [35, 86, 34] as the training error also increased with the addition of layers to a deep network, but rather an inherent issue with how the optimization is performed. If a network is made deeper by adding some layers, one of the solutions to the optimization problem would be the solution of the shallower network by simply setting the new layers to be identity. As a result, a deeper network should demonstrate a performance that is at least as good as its shallow version. Residual networks were designed to address the degradation problem by replacing the optimization problem from approximating the function $H(x)$ to approximating a different function $F(x) = H(x) - x$. This is implemented as residual connections in each layer, adding the input of a sub-network to its output and hence, making up $F(x) + x$. Residual networks are known to be easier to train, can be designed to be deeper, and perform better than ordinary deep neural networks. As will be seen later, Residual architectures have been used both in previous Continual Learning work and this thesis.

2.5 Transfer Learning

To better introduce the concept of Continual Learning, it is good to describe how Continual Learning is different to areas of Transfer Learning, Multi-Task Learning, and Meta-Learning. First, Transfer Learning is a popular area of research in Machine Learning and involves two or more domains, a *source domain* and a set of *target domains*. There is usually a large amount of data and labels for the source domain, while data available for target domains is limited. It is also possible that there are multiple source domains, a case which is termed *multi-source transfer learning*. The goal is to train the network on the large data available in the source domain to help learning from the small amount of data in the target domains [102]. In contrast to Continual Learning, Transfer Learning focuses only on the performance on the target domains and does not make any attempt to preserve the performance of the model on the source domain(s). Continual Learning, however, aims to train a model on two or more datasets such that the performance is good both on the current dataset and the datasets the model was previously trained on.

Transfer Learning approaches in the context of DNNs can be categorized into four groups [68]: **Instance-based** approaches reuse parts of the source data by either computing the loss function with a weight assigned to each training instance or sampling training examples according to a probability. Instance weighting and importance sampling are two main methods used in these approaches. **Feature-representation-transfer** approaches try to learn a ”good” representation using data from the source domain that results in good predictive performance when used as the representation for the target domain. **Parameter-transfer approaches** suggest a portion of model parameters trained on the source domain(s) can also be shared and used for the target domain(s). It essentially assumes that the learned knowledge of the source

domain is captured in the shared parameters or the prior distribution of hyperparameters (e.g. covariance matrix of a Gaussian Process) and it identifies/learns priors that transfer learned knowledge from the source task(s) to target tasks. **Relational-knowledge-transfer** approaches work towards transfer of knowledge in relational domains. Built upon the assumption in these domains, that the relationship between the data has similarities in the source and target domains (e.g. professor to student in an academic context could be similar to director to actor in the Internet Movie Database (IMDb) [64]), it aims to learn and transfer the relationship among the data in the source domain(s) to the target domain(s).

2.6 Multi-Task Learning

Multi-Task Learning is the problem of learning multiple tasks (defined by their respective datasets) simultaneously, to essentially leverage the shared underlying structure between tasks to learn generic feature representations and improve the generalization of the learned model on all of the tasks [100].

While similar to Transfer Learning, Multi-Task Learning has a significant difference in the way it prioritizes tasks. In Transfer Learning, the focus is on the performance on the target domain(s), while Multi-Task Learning aims to improve performance on all tasks and treats them equally. Compared to Continual Learning, Multi-Task Learning has access to all of the tasks' dataset at the same time for the duration of training. In Continual Learning, however, when training the model on a task, access to previous tasks' datasets is very limited or not present at all.

2.7 Few-Shot Learning

A challenging problem within the machine learning community, few-shot learning is the problem of training a neural network using only a few training examples. While humans can learn from few examples in a robust and efficient manner, deep neural networks usually require a large dataset to learn and accurately approximate a function. A prominent area withing few-shot learning is the K-shot N-way classification problem, in which using a large dataset of examples from different classes, the goal is to train a classifier that can learn to distinguish between N new classes, using K samples from each class. There have been a large body of literature dedicated to solve the few-shot learning problem, a survey of which can be found in [95].

2.8 Meta-Learning

Meta-Learning, also known as *learning to learn*, aims to learn better training strategies by monitoring how different training approaches work to select aspects that are likely to lead to better performance [92].

When humans learn new skills, they are likely to reuse learning approaches and strategies that have worked well in past experiences, rather than starting from scratch [47]. Learning each skill helps learning new skills faster and easier, using fewer samples and resorting to less trial

and error. Humans not only learn new skills, but they *learn how to learn* them. *Meta-Learning* refers to any learning that is based on prior learning experiences.

The above-mentioned definition of Meta-Learning is fundamentally different to Continual Learning. However, as will be seen in some of the previous work, Meta-Learning can be used to train networks that show a good performance when learning tasks continually/sequentially. There will not be any Meta-Learning in the methods used in this thesis but it is good to note why such decision was made. A specific form of Meta-Learning will be explained here as a concept and some related previous work will be reviewed later to explain the decision not to use Meta-Learning.

While there are various approaches in the domain of Meta-Learning, such as meta-learning the update function or learning rule of a learner [80, 7, 3, 70], in the context of this thesis, only the *Model-Agnostic* Meta-Learning (MAML) [24] approach is considered. MAML approach meta-learns initial network parameters of a neural network, the learner, such that the learner can make accurate predictions subject to a few optimization steps. The learner's prediction would normally be evaluated using a loss function, with gradients backpropagated over the optimization steps and back to the generated initial-weights, so that in the next iteration, the initial weights work better subject to the optimization steps.

In the context of MAML, some Meta-Learning approaches divide the parameters of the learner into two groups, namely *slow weights* and *fast weights*. There will be two types of optimization loops corresponding to these two groups of weights, namely *outer loop* and *inner loop*. Inner loop simply trains both the fast and slow weights of the learner on the dataset at hand, based on a given learner-specific loss function. At the end of the inner loop, a "meta loss" function is computed, evaluating the performance of the trained learner given the fact that it was supplied with the generated slow weights. In the outer loop, the so called "meta loss" is backpropagated across the optimization steps taken in the inner loop, back to the slow weights (and possibly the network generating them) and an optimization step is taken to update these slow weights. As a result, slow weights get updated after a whole running of the inner loop, while the fast weights are updated in each iteration of the inner loop, hence the terms slow and fast.

2.9 Continual Learning

Continual Learning (also referred to as Lifelong, Sequential, and Incremental Learning), is a learning paradigm that learns continuously by accumulating knowledge from previous tasks and using it to help future learning [17]. This paradigm is in contrast to the dominant learning process in which a model is trained on a given dataset in isolation. In the *isolated learning paradigm*, there is no attempt to preserve the learned knowledge and use it for future learning.

2.9.1 Tasks

In continual learning, the learner is sequentially trained on a sequence of N **Tasks**, $T_1, T_2, ..., T_N$. Each task is defined by a corresponding dataset. At each point in time, the task T_i on which the model is being trained on is called the **new** or **current task**, while the task(s) $T_1, T_2, ..., T_{i-1}$

previously used to train the model are referred to as **previous tasks**. Similarly, the tasks $T_{i+1}, T_{i+2}, ..., T_N$ that the model is yet to see are called **future tasks**.

2.9.2 Forgetting

When being trained on the current task, it has been observed that the model's performance on previous tasks can degrade significantly [61]. The decrease in a performance metric on a previous task is referred to as **forgetting**. The performance metric of choice for defining forgetting is task accuracy, however, if a task only contains one class, then recall is usually chosen. In the remaining part of this thesis, whenever forgetting is mentioned without specifying the metric, the intended metric should be assumed to be recall.

2.9.3 Experiences

The training process on a task is called an **Experience**. Experiences describe the training process by storing and providing attributes such as the number of training epochs and the strategy used.

2.10 Evaluation Metrics

In machine learning, precision, recall, and F_1-score are among the most popular metrics to evaluate a model's predictions. In order to define these metrics, we first need to define the terms True Positive, True Negative, False Positive, and False Negative.

- **True Positives (TP)** are samples in which the condition is correctly predicted to be present.
- **False Positives (FP)** are samples where the condition is falsely predicted as present.
- **True Negatives (TN)** are samples where the condition is correctly predicted to be absent.
- **False Negatives (FN)** are samples in which the prediction wrongly indicates that the condition is absent.

These are defined in the context of binary classification. In multi-class classification, one of the classes must be identified as the "positive" class and the rest as the "negative" class, for the purposes of calculating these quantities. Once we choose a class to act as the positive class, the model's **recall** is the fraction of instances of that particular class that were correctly predicted to be in that class, while **precision** is a fraction of instances predicted to belong to a particular class that truly belonged to the predicted class. F1-score is the harmonic mean of precision and recall. Given a dataset of examples to which the model is applied,

$$\text{Precision} = \frac{\#\text{TP}}{\#\text{TP} + \#\text{FP}}$$

$$\text{Recall} = \frac{\#\text{TP}}{\#\text{TP} + \#\text{FN}}$$

$$\text{F1-Score} = 2 \times \frac{\text{Precision.Recall}}{\text{Precision} + \text{Recall}}$$

Another widely used metric is **accuracy**, which is formally defined as:

$$\text{accuracy} = \frac{\#\text{TP} + \#\text{TN}}{\text{Number of Samples}}$$

In multiclass settings, the performance of a model is often reported as an average over the performance achieved when each class is considered to be the positive class. This may be weighted by the prevalence of instances of each class (micro-averaged) or performance on each class may be given equal weight (macro-averaged.)

Alternatively, performance on all classes may be presented simultaneously using a *confusion matrix*. Given a dataset for each task $D_1, D_2, ..., D_n$ a **Confusion Matrix (CM)** can be defined by setting $\text{CM}_{i,j}$ to be the number of samples belonging to class i that were predicted to be in class j. A confusion matrix can be informative on which classes/tasks are being predicted correctly, and which classes are being misunderstood for each other.

In continual learning literature, a term commonly encountered is **Task Accuracy**, which is essentially the accuracy of the model on a task's data. However, when the task only contains one class, a better term to use would be recall.

While the above-mentioned metrics can help in measuring performance across tasks, it is also desirable to measure the transfer of knowledge between tasks. The following metrics have been defined to evaluate different kinds of transfer of knowledge [58]:

- **Backward Transfer (BWT)** measures how training on the current task influences performance on previous tasks. Backward transfer is positive when training on the current task results in performance improvement on previous tasks, while a negative backward transfer happens when training on the current task degrades the model's performance on previous tasks (forgetting). Intuitively, a large negative backward transfer indicates catastrophic forgetting.

- **Forward Transfer (FWT)** measures how training on the current task would influence performance on future tasks. If training on the current task results in better performance on future tasks, forward transfer is going to be positive. Conversely, if training on the current task negatively impacts the performance on previous tasks, then forward transfer is going to be negative.

In order to formally define performance measures commonplace in continual learning literature, we first need to construct an **accuracy matrix** [53][58]:
Assuming a sequence of tasks $T_1, T_2, ..., T_N$ and access to a test set for each of these tasks, after training on each task T_i, the model can be evaluated on all of the tasks (previous, current, and future tasks). The accuracy matrix $R^{N \times N}$ is then constructed by setting $R_{i,j}$ to be the model's test classification accuracy on task T_j after being trained on task T_i.
Letting b denote the vector of task accuracy for a randomly-initialized model, the following metrics can then be defined based on R and b:

Average Accuracy:

$$\text{ACC} = \frac{1}{N}\sum_{i=1}^{N} R_{N,i} \qquad [58] \quad or$$

$$\text{ACC} = \frac{1}{N}\sum_{i=1}^{N} R_{i,i} \qquad [53]$$

Backward Transfer:

$$\text{BWT} = \frac{1}{N-1}\sum_{i=1}^{N-1} R_{N,i} - R_{i,i} \qquad [58] \quad or$$

$$\text{The average of lower triangular entries of } R \qquad [53]$$

.

Forward Transfer:

$$\text{FWT} = \frac{1}{N-1}\sum_{i=2}^{N} R_{i-1,i} - b_i \qquad [58] \quad or$$

$$\text{The average of upper triangular entries of } R \qquad [53]$$

There is no widely accepted performance measure for all continual learning scenarios. In this thesis, precision, recall, and f1-score after training on each task will be employed as the main indicators of performance to compare different methods.

2.11 Simplifying Assumptions Used in Continual Learning Research

A general definition of continual learning would assume a stream of data in the form of (x_t, y_t) with t denoting the timestep of sample index. Letting $Y_t = \cup_{i=1}^{t} y_i$ the set of samples seen until time t, the goal is to provide a mapping $f_{\theta(x)_t} \rightarrow y$ at any given time t that can accurately predict the label $y \in Y_t \cup \tilde{y}$ from the input x. $\tilde{y}$ is an indicator for when the input sample does not belong to any of the seen labels.

This definition does not impose any constraints on the size of the label set (it can grow arbitrarily), the structure and the order of seen inputs and labels, or the resources available. As solving this general problem has proved to be extremely difficult, most of the work in the continual learning space uses a simplified formulation by making some assumptions. In what comes next, popular assumptions, along with their drawbacks, are explained.

2.11.1 Disjoint task formulation

This formulation assumes that the data stream can be split into certain intervals of time, during each the data stream will bring samples belonging only to certain class(es), in a predefined order of classes. In the general formulation, samples could come in any order and the distribution

of classes at any duration of time could be unknown. Here, however, the order of classes, and the intervals that each class data is seen is predefined as a disjoint set of tasks. For example, the popular MNIST dataset can be split into five disjoint sets, each containing two consecutive digits $\{0, 1\}, \{2, 3\}, ..., \{8, 9\}$.
This assumption simplifies the general formulation, because the unknown growing nature of possible labels is now known and constrained. The computation and space can also be more easily allocated since each task and its dataset can be identified. Following the majority of previous work, the methods in this thesis make this assumption.

2.11.2 Task vs. Class vs. Domain Incremental

The Task-Incremental formulation of continual learning builds upon the disjoint task assumption and presumes that with each data sample, task information or id is also provided. As a result, instead of only (x, y), a three-tuple of (x, y, tid) is given, with tid denoting the task information or id. This assumption greatly simplifies the general formulation as the label space for each task becomes only a portion of the whole set of labels. If applied to the previous MNIST example, for the first task each sample could only belong to class 0 or 1, resulting in a null accuracy (i.e. accuracy when always predicting the class with most samples) of at least 50 percent, while the total number of classes is 10 and the null accuracy would generally be 10 percent (number of samples belonging to each class is approximately equal). The Class-Incremental formulation makes no assumption on the availability of the task information and proceeds assuming it is not provided. More concretely, in the Class-Incremental setting, the learner makes predictions based on the input (x, y) only. The disjoint task assumption, however, is usually considered and test scenarios consist of disjoint task datasets appearing in the data stream sequentially.

An example can better illustrate the difference between task and class incremental settings. The MNIST dataset can be split into 5 (sub)datasets, each holding two classes. The first dataset will include images of digits 0 and 1, the second dataset will include images of digits 2 and 3, and so on. In the task incremental setting, during training and inference, the network will be given the task id. For example, when training on the second dataset, the model also takes tid $= 2$ as input. This will allow the model to process the inputs (x, y) differently according to the tid, for example by using a task-specific set of parameters for making predictions. At inference time, the same tid will be provided to the model, allowing to use the task-specific set of parameters. Moreover, these task-specific sets of parameters can take the form of a classification head, with a node for each label in the task. For example, for the first task, the only possible labels are 0 and 1 and the classification head can include only these two labels. Thus, when predicting inputs (x, y) that the model knows belong to tid $= 1$, the network has less uncertainty and a higher null accuracy. In class incremental setting, the tid is not available in either of training or inference times. Consequently, the algorithm can not process the inputs differently based on which tasks they belong to. Inputs coming from the second dataset will be used by model the same way the first task inputs were processed. In this thesis, the more difficult class incremental setting is assumed.

It is also useful to know about a practical setting called Domain-Incremental. While the distribution of each task's dataset is assumed to stay the same as the training progresses in both Task-Incremental and Class-Incremental scenarios, the Domain-Incremental setting makes no

such assumption. The set of labels in the Domain-Incremental formulation is usually predefined and does not change, while data distribution for each class changes with time and new tasks.

Table 2.11.2 shows a summary of different datasets used in Continual Learning and the settings they can be used in.

Dataset	Task/Class-Incremental	Domain-Incremental	Used in	Construction Process
Permuted MNIST		✓	[43, 58, 98]	Each task is a random permutation of the original dataset, with all the classes
Rotated MNIST		✓	[58, 13]	Each task is a random rotation of the original dataset, with all the classes
SplitMNIST	✓		[98, 13]	Each task is a subset of classes from the original dataset
SplitCIFAR10/100	✓		[58, 72, 98, 13]	Each task is a subset of classes from the original dataset
SplitTinyImageNet	✓		[13]	Each task is a subset of classes from the original dataset
SplitOmniglot	✓		[39, 6]	Each task is a subset of classes from the original dataset
CORe50 [56]	✓	✓	[56]	Each task involves both new classes and different perspectives of previous classes
CLEAR [53]		✓		Each task involves a time-varied version of the classes e.g. bus from 1990 and 2000s.

2.11.3 Online vs. Offline CL

In the general CL formulation, the learning is assumed to not have sufficient storage for all of the data; however, the learner can store some samples based on its space budget. The learner can then revisit samples and update its parameters. Nevertheless, there can be two different settings when it comes to the time allowed to process the streaming data. In the *online* setting, the learner is not allowed to use a data sample for parameter update twice, unless it is stored in memory. In the *offline* setting, however, the learner has unconstrained access to the entire current task dataset, as well as samples stored in memory. The learner can revisit samples (current task and memory) as many times as it chooses and perform arbitrary number of parameter updates. The online setting is considered to be closer to how biological organisms function as real-life experiences are seldom repeated and learning occurs using only a few examples. On the other hand, the offline setting is also practical in applications where the learner can be updated in the background (by traversing data in more than one epoch), while predictions are being made in real-time.

For technical reasons such as inability to train a large network using only a single pass on the data, this thesis assumes training to be offline and traverses the current task dataset for a dataset-specific number of epochs.

2.11.4 Memory Availability

The general CL formulation assumes access only to the current task data. With this assumption, the class-incremental setting becomes very difficult as the learner should differentiate current task samples from previous task samples, while having access only to the current task's data. Without access to previous task's data, the learner usually catastrophically forgets about previous tasks. In contrast, a learner in the task-incremental setting observes very little forgetting as it can use task ids to differentiate labels among tasks. A more practical and biological plausible approach [31, 75, 76, 55] is to add a memory module to store a subset of previous tasks' data and revisit these samples while training on the current task. This memory module can simply be an allocated space in memory, or it can be a Generative Adversarial Network (GAN) [30] trained to produce samples similar to those of previously seen classes.

In this thesis, it is assumed that a small memory is available to store a subset of previous task samples.

2.12 Representation Learning

Given the recent success of Contrastive Learning, the methods in this thesis also use Contrastive Learning. To properly introduce Contrastive Learning, first the concept of Representation Learning is explained, and then its specific form, Contrastive Learning, will be described.

Representation learning aims to learn useful representations, i.e. mappings from inputs x to feature vectors, that can be transferred to downstream tasks. While the usefulness depends on the domain and specification of the downstream task, it is commonly assumed that representations should create latent features that represent various aspects of the input data on a high level of abstraction. Deep neural networks combine linear and non-linear operations to produce and learn suitable representations for the tasks they are trained to do. In this way, any deep neural network used for a classification task can be thought as an encoder and a classifier. The encoder transforms an input instance to its corresponding representation, while the classifier differentiates between representations that belong to different classes. In the context of representation learning, however, the training process is different. In representation learning, the encoder does not output class probabilities, but it rather maps input data to representations that will be learned directly.

Representation Learning, similar to other fields in deep learning, can be achieved via supervised, unsupervised (self-supervised), and semi-supervised learning. In supervised representation learning, input data are annotated with labels, while in self-supervised representation learning, the labels are not available, and other approaches, often related to compression/reconstruction, are formulated to help the model capture the most out of the available data. Moreover, in semi-supervised learning scenarios, the learner has access to a small set of labelled samples, and a much larger set of unlabelled data. In semi-supervised learning, approaches in supervised and unsupervised learning are concurrently used to exploit the labelled

and unlabelled datasets.

Recently, in Computer Vision, contrastive learning has achieved state-of-the-art performance for image classification tasks. In the next section, a high-level description of contrastive learning is provided.

2.12.1 Contrastive Learning

Contrastive Learning is an approach that attempts to learn representations by (1) augmenting each data sample and generating multiple *views*, (2) computing the representations corresponding to these views, and (3) comparing and contrasting the representations via a contrastive loss. Data augmentation for image data usually involves transformation of the image (e.g flipping, rotations, and random cropping) such that the content/class does not change. The contrastive loss encourages the representations of samples from the same class to be similar, while simultaneously pushing away the representations of samples from differing classes. This contrastive loss can be self-supervised [16] or supervised [42]. There has been many variants and applications of contrastive learning in Computer Vision, as well as domains like audio, video, language, and graphs, among others [48, 46].

2.13 Neuromodulation

In this and the following two sections, main inspirations for the methods presented in this thesis are explored. It is good to note that the methods are by no means a replica of the processes seen in the brain, but rather been loosely inspired by them.

Biological neurons are much more complex than the simple artificial neuron implemented in artificial neural networks [28, 8], to the extent that each biological neuron requires a neural network of 5-8 layers to be well approximated [8]. This does not come as a surprise, since neuromodulatory neurons not only receive, perform computation, and output signals based on the inputs, but they communicate on various spatio-temporal scales with other neurons using neurotransmitters. Acting as chemical messengers, these neuromodulators play various roles in behavioral adaptation. For example, in hippocampus, a brain area that plays a significant role in learning and memory, a neuromodulator called acetylcholine (ACh) works with another neuromodulator called serotonin (5-HT) in coordination to make way for important cognitive and consciousness-related functioning [65, 78, 62]. Noradrenaline (NA) is another neuromodulator that has been attributed to arousal and attention, as well as novelty and surprise [20]. Moreover, another neuromodulator called dopamine (DA) is believed to encode a reward prediction error signal [82]. Overall, with the biological organisms performing well in adaptation and continual learning, simulating the activity and functions of neuromodulators not inherently accessible to artificial neural networks can guide research in mitigating catastrophic forgetting and learning generalizable representations.

2.14 Neuroplasticity

Biological neuronal networks have the ability to rewire, reshape, and grow with new experiences. Known as **Neuroplasticity**, this ability enables biological brains to learn new functions by shaping new neuronal pathways [18]. In this thesis, this ability will be referred to as the *plasticity* of biological neurons and will be viewed as one of the main biological functions that may play a significant role in the continual learning of new skills.

2.15 Top-Down Visual Feedback

In everyday tasks, we look around and use our visual sensory inputs. These visual inputs play a significant role in our behaviour and how we respond to different environments. There has been a long standing debate on how we select the objects we attend to [90]. One view suggests that this selection is based on how salient the objects are, in an automatic bottom-up fashion starting from neurons encoding simple patterns and moving on to those responding to more complex and abstract features ([11] for example), while a contrasting view claims that the attention to select objects occurs voluntarily according to our behavioral goals, in a top-down fashion [25]. There is evidence that a neuromodulatory top-down feedback based on dopamine (DA) [87, 81, 85] and acetylcholine (ACh) [5, 79, 10, 103] exists, and a change in activity of lower areas in the human visual system based on the top-down feedback has been observed [23]. The so called top-down attention or selection signal can prove to be useful in modulating an artificial neural network when it attempts to continually learn visual tasks. An attention signal can help the learner identify which aspects of itself should be modulated to be more plastic and which parts should be preserved. As a result, this neuromodulatory feedback mechanism can help guide the learner in adapting to new experiences. A few novel approaches for simulating this feedback mechanism are provided in the methods section of this thesis and evaluated on popular Computer Vision datasets.

Chapter 3

Literature Survey

This chapter reviews the literature on continual learning in the context of visual tasks, identifying established and state-of-the-art results relevant to this thesis. Section 3.3 reviews architectures used in continual learning, and Section 3.2 reviews established datasets used to develop and evaluate continual learning methods.

3.1 Continual Learning Desiderata

In the previous chapter, the general formulation and various settings for continual learning were described. While the mentioned settings are meant to represent real-life continual learning scenarios visited in practice, it is useful to always remember what the practical desiderata of continual learning are, and what abilities a continual learning agent should showcase when deployed in real-life scenarios. For example, Hadsell [31] imagines a hypothetical robot that is tasked with performing any household chore in any home. The robot can not be simply pre-programmed in the factory, since the house chores may change over time. Moreover, it should be able to learn new skills (e.g. washing the dishes, tidying, doing laundry) and learn the variations within each task. For example, in order to do the laundry task, it should first separate different clothes, then wash according to each cloth type, then dry using appropriate settings for each cloth type, and finally sort and maybe iron some of the clothes. In order to do all of these tasks properly, the robot needs to adapt quickly to learn new tasks and their underlying variations, while remembering previous tasks. Forgetting may be allowed only if it is limited and recovery is fast. Moreover, the robot should perform better on future tasks after learning each task (*positive forward transfer*) and also transfer the knowledge acquired by learning the current task to previous tasks and perform better on them (*positive backward transfer*). It is also practical to assume that the robot is limited in storing and accessing previous tasks' memories, increasing the model size, and time-available to perform each task (i.e. processing time).

If we put the expected features and requirements of a real-life continual learner together, we can define the following desiderata [31]:

- **Limited access to previous tasks' data**. The available storage is limited and only a portion of previous experiences can be stored. As a result, the agent can not have unlimited interaction (learning) with previous tasks.

- **Limited increase in model size and compute**. The available computation resources are limited in a realistic agent. As a result, the continual learning approach should be scalable: The computational resource demands can not increase linearly with the number of tasks or dataset size.
- **Minimal catastrophic forgetting and interference**. Learning new tasks should not significantly degrade performance on previous tasks.
- **Fast adaptation and knowledge recovery**. The model should be able to quickly adapt to novel tasks and/or recover prior knowledge from previous tasks.
- **Sustained learning ability and plasticity**. The model's ability to learn should be sustained after training on some tasks. It should be capable of adapting to a new task at any time.
- **Maximized forward and backward transfer**. The knowledge acquired by learning the current task should transfer to both past and future tasks, to improve performance and learning efficiency.
- **Task-agnostic learning**. There is no oracle helping with task labels in the real-world, and the model should not rely on task labels to perform well.

As can be seen, a continual learning agent needs to make compromises between various objectives as some objectives go against each other. For example, it is impossible for a fixed-capacity model to learn an arbitrarily large number of tasks without forgetting. Making compromises when needed can take different forms. For example, one continual learner may choose to forget previous tasks to some extent, adapt to the current task and when asked to perform previous tasks again, demonstrate quick recovery and show good performance. Another continual learner may choose to adapt less to new tasks (decrease model capacity by fixing some parameters which may result in worse performance in the current tasks) in an attempt to keep performing well on previous tasks. Before describing each continual learning approach and their compromises, the datasets commonly used to evaluate continual learning approaches are described.

3.2 Continual Learning Datasets

3.2.1 Split-MNIST

MNIST [50] is a dataset of handwritten English digits, namely 0 to 9. It consists of a training set of 60000 samples and a test set of 10000 samples. Each sample is a 28×28 gray-scale image showcasing a handwritten digit. While this dataset has been used as a whole to benchmark various image classification algorithms [71], for evaluation of continual learning approaches this dataset gets broken into multiple chunks, with each one representing a task. The continual learning approach is then trained on each task sequentially and the performance (using the performance metric and protocol of choice) is evaluated. The splitting for task-incremental and class-incremental continual learning scenarios is usually done by classes, i.e. each task

will be discrimination of a subset of digits in $\{0, 1, ..., 9\}$. For example, the 10 digits present in the dataset can be split into 5 tasks or chunks, with the first task to discriminate between 0 and 1, the second task to discriminate between 2 and 3, and so on. For the domain-incremental setting where the set of possible class labels remains the same across tasks while the distribution of data changes, the split takes place across samples, i.e. the set of samples for each digit is split into multiple chunks, and the first chunk of all digits is considered to be the first task, while the second chunk of all digits is going to be the second task and so on.

3.2.2 Split-CIFAR10/100

CIFAR-10 and CIFAR-100 (also mentioned interchangeably as CIFAR10 and CIFAR100, or CIFAR10/100 to refer to both) are labelled datasets of objects, scenes, animals, and people. These datasets are subsets of the 80 million tiny images dataset (that was taken down). CIFAR10 has 10 classes namely airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. It consists of a training set of 50000 samples and a test set of 10000 samples, with 5000 and 1000 samples for each class in the training and test sets respectively. Each sample is a colored (RGB) 32×32 image belonging to one of the mentioned classes.

Similarly, CIFAR100 is a dataset of 100 classes and consists of training and tests sets that comprise 50000 and 10000 samples respectively. The dataset is also balanced as there are 500 training and 100 test samples assigned to each class. Each sample is a 32×32 RGB image belonging to one of the 100 classes. It is also useful to note the 100 classes in this dataset can be grouped into 20 super-classes.

CIFAR10/100 has been used to benchmark a wide range computer vision tasks [71]. For continual learning, however, it needs to be split into multiple tasks first to evaluate how a continual learner performs when trained on a number of tasks sequentially. Similar to Split-MNIST, the splitting process can take place across classes for task-incremental and class-incremental scenarios and across samples for domain-incremental settings. For example, in a class-incremental setting, the first task can be learning to discriminate the classes airplane and frog, while the second task is learning to distinguish automobile and bird from each other and from classes seen previously, with the rest of the tasks defined in a similar way. If the considered setting was task-incremental, at each task the model needed only to distinguish between the given classes (e.g. automobile and bird) and it did not need to discriminate between the classes in the task and classes seen in previous tasks. The domain-incremental setting can also be applied to this dataset by splitting each class's data into multiple chunks and assigning each chunk of all classes to each task.

3.2.3 Split-TinyImageNet

TinyImageNet is a dataset used for image classification. It was first provided as part of the cs231n course at Stanford. It consists of 200 classes, with 500 training, 50 validation, and 50 test images assigned to each class, summing up to 600 images per class, and 100000 train, 10000 validation, and 10000 test samples in total. Each sample is a 64×64 colored image.

The TinyImageNet dataset has been on of the popular computer vision benchmarks [27]. Similar to MNIST and CIFAR10/100, it can be used to evaluate continual learning approaches by splitting the dataset into multiple tasks. The splitting can take place across classes (for

task-incremental and class-incremental settings) and across samples (for domain-incremental settings).

3.3 Architectures

Since datasets used in continual learning are usually small (relatively low number of samples and resolution), the architectures used to train on these datasets are also relatively small. Specifically, for the MNIST dataset, a simple Convolutional Neural Network (CNN) [49] or a feed-forward neural network with 2 or 3 layers is used [58, 98, 74]. For CIFAR10/100 and TinyImageNet datasets, however, a deeper CNN such as the ResNet-18 [35] or a reduced version of it [58] is used.

3.4 Attribution

In order to preserve knowledge of past tasks that is both essential to maintain performance on past tasks and likely to perform well on upcoming tasks, we need a means to attribute the performance of a neural network to its parameters. A similar body of work to this problem falls under the name of Attribution. In this section, the connection between the Attribution problem and what comes later in methods (chapter 4) for identifying salient parameters is explained.

As DNNs were being employed more and more into operational workflows, it was important to know why these networks make the predictions that they do, regardless of being correct or wrong. The attribution problem attempts to attribute the output of a model to its input features. For example, given an image and a classification prediction of "dog", the attribution problem wants to know which pixels were most decisive in the particular prediction of "dog". Attribution has useful applications, such as

- Debugging incorrect network predictions.
- Providing an explanation of why the network made that prediction.
- Analysis of how robust a network is.
- Deciding prediction confidence.

among others [88].

There has been a large body of literature addressing the attribution problem (see [1] for a review). The purpose of using attribution in this thesis, however, is different. Here, we would like to attribute the *performance* of a network to its parameters instead. While this problem is different in essence to the attribution problem, the methods and techniques employed in the original attribution problem can still work. For example, a naïve approach to the attribution problem is to drop an input feature (set it to zero) and assess the change in the predictions. This same naïve approach can similarly be applied to the performance attribution problem: Drop a parameter and measure the change in performance. It is useful to note that assessing performance can be done simply by using any performance measure of choice, like accuracy, recall, or f1-score.

Among the popular attribution methods, **Excitation Backpropagation**[99] was chosen to be used in this thesis for its biological plausibility and easier adaptation to the performance attribution problem.

3.4.1 Excitation Backpropagation

Excitation backpropagation aims to attribute predictions to and measure contribution of all neurons (not just the input) via a top-down neural attention process. In this method, a forward bottom-up pass is first performed to compute unit activations. Then, using an attention vector on the output layer as input, a top-down attention signal is computed layer-by-layer, iteratively.

Formally, a generic feedforward neural network model and a prior distribution $P(A_0)$ on the output units is considered. Assuming N to be the set of all neurons, and A_i to denote the selected neurons on ith layer from the top, the distribution over the activation of the neurons in each layer is computed recursively based a conditional *winning* probability $P(A_t|A_{t-1})$ where $A_t, A_{t-1} \in N$. Formulating attribution of a neuron as the probability of being selected as a winning neuron, a Marginal Winning Probability (MWP) $P(a_j)$ needs to be computed for each neuron $a_j \in N$. The MWP $P(a_j)$ can be written as:

$$P(a_j) = \sum_{a_i \in \mathcal{P}_j} P(a_j|a_i)P(a_i) \tag{3.1}$$

with $\mathcal{P}_j$ denoting the parent neuron set of a_j in top-down order (layer closer to the output layer). As a result, given MWPs for each layer, the probabilities for being selected as a winning neuron can be computed iteratively.

Considering an input of a_i, a weight of w_{ji}, a bias of b_j, and a non-linear activation function of ϕ, the computation performed by a neuron can be written as $\hat{a}_i = \phi(\sum_j w_{ji}\hat{a}_j + b_j)$. The excitation backprop framework makes the following two assumptions:

1. The activation value of a neuron is non-negative.

2. A neuron responds to certain visual patterns, and its activation is positively correlated with the confidence of detecting that pattern.

The first assumption is usually satisfied as neural networks mostly use the Rectified Linear Unit (ReLU) activation function. The second assumption has also been verified empirically in various computer vision works and literature ([101] for example beside many others). Given these two assumptions, it follows that the negative weights in will only decrease activation values:

$$\begin{aligned} \hat{a}_i &= \phi\left(\sum_{j,w_{ji}\geq 0} w_{ji}\hat{a}_j + \sum_{j,w_{ji}<0} w_{ji}\hat{a}_j + b_j\right) \\ &\leq \phi\left(\sum_{j,w_{ji}\geq 0} w_{ji}\hat{a}_j + b_j\right) \quad \text{if} \quad \hat{a}_j \geq 0 \end{aligned} \tag{3.2}$$

The connections in the network can then be labelled as *excitatory* and *inhibitory*. Those connections with positive weights can increase a neurons activation value and will be *excitatory*,

while those with negative weights will only decrease the activation values and thus will be *inhibitory*. The excitation backprop works by sending a top-down signal only through the excitatory connections. Setting C_i to be the child node set of a_i in the top-down order (layer closer to the input layer), for each $a_j \in C_i$ the MWP $P(a_j|a_i)$ is defined as:

$$P(a_j|a_i) = \begin{cases} Z_i \hat{a}_j w_{ji} & \text{if } w_{ji} \geq 0, \\ 0 & \text{otherwise.} \end{cases} \tag{3.3}$$

Z_i is just a normalization factor to make sure $\sum_{a_j \in C_i} P(a_j|a_i) = 1$, so

$$Z_i = \frac{1}{\sum_{j, w_{ji} \geq 0} \hat{a}_j w_{ji}} \tag{3.4}$$

In the case of denominator in 3.4 being zero, Z_i will be set to zero. With the MWP of each layer defined in 3.3, the top-down attention or excitation signal can be propagated iteratively by simply computing MWPs and using 3.1 to compute the selection or winning probability of each neuron. As a result, given an input sample and a prior distribution over the output nodes, this framework can attribute the prior distribution to the networks neurons. Note that there is no assumption on what the prior distribution over the output layer neurons should be. It can be used to denote a prediction's confidence as seen usually in the computer vision literature, or, as will be seen later in the methods section, the performance of the network.

Furthermore, a contrastive variant of excitation backpropagation (c-EB) is defined as well. Normally, a single pass of excitation backprop can generate attention maps over the network layers. The contrastive variant adds another pass, with the weights of the last layer of the network negated this time. This second pass will identify the neurons contributing most for inhibiting a certain output. For example, if the target label is "elephant", negating the last layer weights and producing attention maps with excitation backprop will identify the neurons most influential for predicting "not elephant". Subtracting the attention maps for "not elephant" from attention maps for "elephant" will then better identify neurons used to predict "elephant", as the neurons contributing to both "elephant" and "not elephant" will be cancelled out and those that are most discriminative will be selected more prominently.

3.5 Approaches to Continual Learning

In this section, a representative subset of previous work that related to this thesis is described and reviewed.

3.5.1 Knowledge Distillation

One of the main approaches to preserve knowledge from past tasks is called Knowledge Distillation. In this method, a neural network's output is regularized to be similar to a second neural network's output, enabling the network parameters to change as long as they are producing the same output. In the context of Continual Learning, this first neural network is simply the neural network that is being trained on the current task, while the second neural network is a snapshot of the neural network at the end of training on previous task.

Learning Without Forgetting (LwF)

Learning without Forgetting (LwF) [52] attempts to solve the task-incremental continual learning setting by adding a new classification *head* for each task. Each classification *head* is essentially a one layer neural network that projects to an output layer with number of nodes equal to the number of target labels. In other words, LwF operates on a multi-head architecture. An important assumption of LwF is that when training on a new task, there is no access to previous tasks' data.

LwF divides the network into 3 portions:

1. Feature extractor with parameters θ_s that are *shared* for all tasks.
2. Previous tasks' classifications heads, with parameters θ_o.
3. New added classification head for the training of the current task, with parameters θ_n.

The output of each classification head is going to be a probability distribution over the potential labels for each task.

Before starting training on a new task, LwF records the responses of all previous task heads using all of the training data, resulting in a set of probability distributions for each previous task for each training sample. Next, when training on a new task (x, y), LwF first trains the network by freezing θ_s and θ_o as a warm-up stage. Then, the network is jointly trained (all parameters) using a loss for the new tasks, and a distillation loss for previous tasks. The new task loss is simply a multinomial logistic loss:

$$\ell_{\text{new}}(\mathbf{y}_n, \hat{\mathbf{y}}_n) = -\mathbf{y}_n.\log(\hat{\mathbf{y}}_n) \tag{3.5}$$

where $\hat{\mathbf{y}}_n$ is the network prediction using the new task classification head, and $\mathbf{y}_n$ is the one-hot ground truth label vector. The distillation loss is a modified cross-entropy with increased weights for smaller probabilities:

$$\begin{aligned}\ell_{\text{old}}(\mathbf{y}_o, \hat{\mathbf{y}}_o) &= -H(\mathbf{y}'_o, \hat{\mathbf{y}}'_o) \\ &= -\sum_{i=1}^{l} y_o'^{(i)} \log(\hat{y}_o'^{(i)})\end{aligned} \tag{3.6}$$

with l denoting the number of labels and $y_o'^{(i)}$ and $\hat{y}_o'^{(i)}$ representing the modified versions of recorded and newly computed probabilities respectively:

$$y_o'^{(i)} = \frac{(y_o^{(i)})^{1/T}}{\sum_j (y_o^{(j)})^{1/T}}, \quad \hat{y_o}'^{(i)} = \frac{(\hat{y}_o^{(i)})^{1/T}}{\sum_j (\hat{y}_o^{(j)})^{1/T}} \tag{3.7}$$

The parameter T controls how much more weight is put towards smaller probabilities [37], with $T > 1$ increasing the weight of smaller logits and encouraging the network to better predict class similarities. The loss is then summed over all previous tasks and all labels.

LwF allows forward and backward transfer, does not store previous task examples in a memory, and shows competitive performance when compared to other methods. There are, however, some limitations:

1. LwF works on a task-incremental setting, and needs task descriptors at evaluation time.
2. LwF assumes access to all of a tasks' data in the beginning of training on that task.
3. The performance of LwF generally depends on how similar the tasks are. If tasks are dissimilar, the performance decreases.

An unexplained and inherent assumption with the usual knowledge distillation approach is that all of the neurons in the output of the neural network are of equal importance. However, this is generally not true, as some of the neurons may be more discriminative than others. Moreover, it is assumed that all of the knowledge captured in the snapshot neural network and its output is worth preserving, while parts of the output may be features that are overfitted to previous tasks and should not be used for regularization. In this thesis, we will use knowledge distillation as one of the knowledge preservation methods while trying to address the mentioned limitations.

3.5.2 Parameter Isolation and Regularization

While knowledge distillation attempts to preserve knowledge by regularizing the output of a neural network and does not impose any restrictions on individual network parameters explicitly, a group of methods in Continual Learning identify parameters important to previous tasks and either freeze them or regularize them. A representative part of the literature using these methods will be explained in this section.

Elastic Weight Consolidation (EWC)

The Elastic Weight Consolidation (EWC) method [43] cites recent evidence that shows when a mouse learns a new skill, some of the excitatory synapses are strengthened as the volume of dendritic spines of these neurons increases. These dendritic spines remain enlarged even after subsequent learning, and account for retained performance several months later [97]. Following this evidence in biological neurons, EWC tries to identify and retain parameters important for each task when training on subsequent tasks.

Assuming θ represents a network's parameters, due to what EWC identifies as a common overparameterization of neural networks, it finds it likely that there exists $\theta^\star_B$, parameters optimized for task B, that are close to $\theta^\star_A$, parameters optimized for task A, when training on task B after task A. EWC attempts to keep the network parameters close to $\theta^\star_A$ while training on task B via quadratic regularization. The regularization weights for different parameters are, however, not the same. The parameters that are identified to be more important for task A will be regularized more with larger weights.

From a Bayesian view, optimizing network parameters θ is equivalent to maximizing their probability given the input dataset $\mathcal{D}$. Using the Bayes rule, this probability can be computed from the prior probability of parameters and the probability of the dataset given the network parameters:

$$\log p(\theta|\mathcal{D}) = \log p(\mathcal{D}|\theta) + \log p(\theta) - \log p(\mathcal{D}) \tag{3.8}$$

The log probability of the dataset given network parameters ($\log p(\mathcal{D}|\theta)$) is essentially the negative of the loss function regarding the current task $-\ell(\theta)$. Considering a second task B and

naming the first task A, the dataset $\mathcal{D}$ can be split into two portions: Task A's dataset $\mathcal{D}_A$ and task B's dataset $\mathcal{D}_B$. Rewriting equation 3.8 then gives:

$$\log p(\theta|\mathcal{D}) = \log p(\mathcal{D}_B|\theta) + \log p(\theta|\mathcal{D}_A) - \log p(\mathcal{D}_B) \tag{3.9}$$

where the probability of the dataset given the parameters ($\log p(\mathcal{D}|\theta)$) is now turned into $\log p(\mathcal{D}_B|\theta)$ since only the new task's data is available and the prior probability of parameters ($p(\theta)$) has now become $p(\theta|\mathcal{D}_A)$ as the parameters now have been trained on task A. Assuming there is no access to previous tasks' data, EWC notes that the posterior probability $p(\theta|\mathcal{D}_A)$ should now contain all the information about task A and which parameters are more important. Since this probability is intractable, EWC approximates this posterior as a Gaussian distribution with mean $\theta^\star_A$ and a diagonal precision given by the Fisher information matrix F. With this approximation, the new loss function becomes:

$$\ell(\theta) = \ell_B(\theta) + \sum_i \frac{\lambda}{2} F_i(\theta_i - \theta^\star_{A,i})^2 \tag{3.10}$$

where ℓ_B is the loss for task B, λ controls the importance of the previous task compared to the new one, and i traverses the network parameters. The loss in 3.10 applies only when training on a second task. When there are more tasks, EWC stores optimized parameters after training on each task (for example $\theta^\star_C$ for task C) and adds a similar quadratic penalty to the loss.

EWC has been one of the main benchmarks for task-incremental continual learning since introduction. It works well to prevent forgetting, enables forward transfer, and it does not need to store previous task examples. Nevertheless, there are a few drawbacks to how it works:

1. It is hard to scale to many tasks.

2. The required storage grows with the number of tasks as it stores the optimized parameters and a Fisher information matrix for each task.

3. While backward transfer can happen, it is not directly encouraged, as the only direct incentive is too keep parameters important for previous tasks from changing too much and shaping new, better, and more generalizable features is not considered.

While there has been a prior work termed *Online EWC* [83] that addresses the second drawback and computational scalability by keeping only the latest parameters and one Fisher information matrix, the other two drawbacks, mainly the absence of backward transfer and scalability to a large number of tasks, remain unsolved. It is also good to note that there is no attempt of measuring the importance of parameters using an alias of how likely they are to transfer to future tasks. Given the difficulty this method has to scale to many tasks, it shows that probably a significant portion of parameters that end up being regularized are those that are overfitted to previous tasks and do not generalize to next tasks.

Synaptic Intelligence (SI)

Similar to EWC, Synaptic Intelligence (SI) observes that biological synapses are involved in complex molecular interactions that can modulate plasticity at different spatio-temporal scales

[73]. In an effort to increase the complexity of neurons and identify important parameters to regularize, SI adds a local measure of importance to each neuron. Assuming θ represents the network parameters, SI introduces an importance measure ω_k^μ that tracks the importance of parameter θ_k for task μ based on its contributions to decrease the task loss ℓ_μ.

SI characterises the training process of neural network as a trajectory $\boldsymbol{\theta}(t)$ in the parameter space. The change in loss when taking an optimization step $\boldsymbol{\delta}(t)$ at time t can be written as:

$$\ell(\boldsymbol{\theta}(t) + \boldsymbol{\delta}(t)) - \ell(\boldsymbol{\theta}(t)) \approx \sum_k g_k(t)\sigma_k(t) \tag{3.11}$$

which shows that a parameter change of $\delta_k(t) = \theta_k'(t)$ contributes $g_k(t)\delta_k(t)$ to the change in the loss. A path integral would then compute the total contributions to the loss over the training trajectory from the start time t_0 to the final time t_1:

$$\int_C \boldsymbol{g}(\boldsymbol{\theta}(t))d\boldsymbol{\theta} = \int_{t_0}^{t_1} \boldsymbol{g}(\boldsymbol{\theta}(t)).\boldsymbol{\theta}'(t)dt \tag{3.12}$$

Note that gradient is a conservative field, and the value of integral is going to be equal to the change in loss from the start point to the end point: $\ell(\boldsymbol{\theta}(t_1)) - \ell(\boldsymbol{\theta}(t_0))$. Equation 3.12 can be rewritten in terms of individual parameters, where the importance measure is going to be defined:

$$\begin{aligned}\int_{t^{\mu-1}}^{t^\mu} \boldsymbol{g}(\boldsymbol{\theta}(t)).\boldsymbol{\theta}'(t)dt &= \sum_k \int_{t^{\mu-1}}^{t^\mu} g_k(\theta(t))\theta_k'(t)dt \\ &\equiv \sum_k \omega_k^\mu \end{aligned} \tag{3.13}$$

The importance measure ω_k^μ can in practice be approximated by computing the running sum of the product of the gradient $g_k(t) = \frac{\partial \ell}{\partial \theta_k}$ and the change in parameter $\theta_k'(t) = \frac{\partial \theta_k}{\partial t}$.

With the importance measure for each parameter defined, SI introduces a regularization term to the loss that penalises parameters that strain too far away from their initial value when training on a task began. The new loss, augmented with regularization, is as follows:

$$\tilde{\ell}_\mu = \ell_\mu + c\sum_k \Omega_k^\mu \left(\tilde{\theta}_k - \theta_k\right) \tag{3.14}$$

where c is a strength parameter that controls how much regularization is introduced, $\tilde{\theta}_k = \theta_k(t^{\mu-1})$ is the value of a parameter at the end of the previous task's training, and Ω_k^μ is the parameter-specific regularization weight:

$$\Omega_k^\mu = \sum_{\nu<\mu} \frac{\omega_k^\nu}{(\Delta_k^\nu)^2 + \xi} \tag{3.15}$$

The parameter $\Delta_k^\nu = \theta_k(t^\nu) - \theta_k(t^{\nu-1})$ measures the amount of total change in a parameter over the course of training of a task, and the parameter ξ is for stability as Δ_k^ν may get very small. Consequently, the regularization weight depends on:

1. The contribution of a parameter to the decrease in loss, and

2. The amount of change seen in that parameter.

Similar to EWC, SI works by regularizing parameters that it deems to be important for the performance of the network on previous tasks. While providing performance that slightly outperforms EWC, SI suffers from the same drawbacks of EWC:

1. It is hard to scale to a large number of tasks,
2. Requires the storage of importance parameters per task, which makes the storage requirement to grow linearly with the number of tasks.
3. While backward transfer can happen, it is not directly encouraged.

PackNet

PackNet [60] is a parameter isolation method that draws from network pruning techniques [33, 32] to propose a continual learning algorithm. The PackNet algorithm uses weight-based pruning techniques to free up weights that are less important in the network for future learning, while keeping the remaining weights frozen.

The pruning technique removes (set to zero) a fixed portion of weights in each convolutional or fully-connected layer. The weights in a layer are sorted by their value, and the weights sitting at the bottom with the smallest values are removed. Pruning is performed specific to a task, meaning only those weights that were used for a task are considered, and weights frozen for previous tasks are ignored.

Specifically, considering three tasks, PackNet works as follows:

1. Uses a pre-trained network or simply trains the network on task 1.
2. Runs the pruning process and removes a fixed portion of weights in each layer. The pruned weights $\theta_1^{(p)}$ are frozen for now.
3. Trains the remaining weights $\theta_1^{(r)}$ on task 1 for fewer epochs to get a network with relatively sparse weights. The trained weights $\theta_1^{(r)}$ are frozen hereafter and remembered for use at inference time.
4. The pruned weights of previous task $\theta_2 = \theta_1^{(p)}$ are unfrozen and trained on task 2.
5. Runs a pruning process on θ_2 and freezes the pruned weights $\theta_2^{(p)}$.
6. The remaining weights $\theta_2^{(r)}$ are retrained on task 2 for fewer epochs and then frozen hereafter. These weights are also remembered for inference.
7. The pruned weights of previous task $\theta_3 = \theta_2^{(p)}$ are unfrozen and trained on task 3.
8. Runs a pruning process on θ_3 and freezes the pruned weights $\theta_3^{(p)}$.
9. The remaining weights $\theta_3^{(r)}$ are retrained on task 3, frozen hereafter, and remembered for inference.

The process then continues similarly for more tasks, until tasks end or there is no parameter left to train the task on. At inference time, using remembered weights for each task (task descriptor is needed to know which weights to use), predictions are computed. Each weight is going to be used for the task it was trained for (for example task K) and subsequent tasks. As a result, for each task only a binary mask needs to be stored, and the storage overhead is small.

PackNet was able to outperform methods like LwF [52], while enabling forward transfer, avoiding catastrophic forgetting, and adding a small storage overhead. There are, however, some drawbacks:

1. Can only be used for task-incremental continual learning settings (needs task id at inference time).
2. The number of tasks that can be trained on can be very limited depending on the network capacity.
3. Prioritizes tasks that come first over tasks seen later.
4. Needs to keep binary masks for inference time.

This method can be thought of an extreme case of the regularization-based Continual Learning approaches. Given the fact that parameters to isolate are identified based on their performance on past tasks, the problem with regularization approaches still remains: The parameters deemed to be important may be overfitted to previous tasks and are not likely to generalize to future tasks.

Attention-based Selective Plasticity

Attention-based selective plasticity [44] is inspired by neuromodulatory mechanisms in the brain, where bottom-up stimulus-driven and top-down goal-driven attention helps to put the focus more on task-specific stimuli and less on distractions [4, 67]. Overall, it uses the contrastive Excitation Backpropagation (c-EB) framework to selectively modulate the plasticity of network weights and regularize parameters it deems to be important for each task. Similar to [43, 98], synaptic importance parameters are computed for each network weight and a quadratic regularization term is added to the loss to avoid important parameters from straying too far from their initial values when training on a task started.

Assume $f_i^{(l)}$ denotes the ith neuron in the layer l of a neural network, $P_c(f_i^(l))$ represents parameter contributions measured by the c-EB framework, and $\gamma_{ji}^{(l)}$ denotes the synaptic importance parameter for the weight between $f_j^{(l-1)}$ and $f_i^{(l)}$. While an optimization step for a neural network would typically involve a forward pass to make predictions and a backward pass to compute gradients, in this work the forward pass is followed by a backward pass that computes parameter contributions using the c-EB framework. After the c-EB pass, synaptic importance parameters are updated using Hebbian learning [36] and Oja's rule [66]:

$$\gamma_{ji}^{(l)} = \gamma_{ji}^{(l)} + \epsilon \left(P_c(f_j^{l-1}) P_c(f_i^{(l)}) - P_c(f_i^{(l)})^2 \gamma_{ji}^{(l)} \right) \tag{3.16}$$

where ϵ is the rate of Oja's learning rule. Note that synaptic importance parameters are task-specific, and a new $\gamma_{ji}^{(l)}$ is computed and stored for each task. The synaptic importance parameters are initialized to zero at the beginning of training on each task. Next, using the computed

synaptic importance parameters, the loss is updated to incorporate at a regularization term for parameters that are important for previous tasks. The loss is the same as the one used EWC [43] 3.10, with the regularization weights now set to the computed task-specific synaptic importance parameters. Specifically, in the case of two tasks, the loss when training on task B can be written as:

$$\tilde{\ell}(\theta) = \ell(\theta) + \lambda \sum_{k} \gamma_{A,k}(\theta_k - \theta^{\star}_{A,k})^2 \tag{3.17}$$

where θ represents the network parameters, k iterates over individual network parameters, $\gamma_{A,k}$ is synaptic importance for the kth parameter specific to task A, and $\theta^{\star}_{A,k}$ denotes the network parameters after being optimized on task A. For subsequent tasks, a similar regularization term can be added to the loss. After the modification to the loss, a backward pass for computation of gradients is performed and an optimization step is taken.

The proposed neuromodulation-inspired approach performs on par with EWC [43] and Synaptic Intelligence (SI) [98] without extensive hyperparameter tuning. It is similar to SI as synaptic importance parameters are computed in an online manner, in contrast to EWC where they are computed after the training on each task. Nevertheless, it suffers from a few drawbacks:

1. Difficult to scale to many tasks.

2. During training, it needs to store synaptic importance parameters for each task.

3. While backward transfer can happen, it is not directly encouraged.

While this method can be thought to be a regularization-based Continual Learning approach similar to EWC and SI having the same limitations, it proves a notable point: Identifying important parameters using Excitation Backprop is possible and demonstrates competitive performance. As a result, this method fits nicely with our goal of identifying important parameters using Attribution methods and encourages us to follow a similar approach.

3.5.3 Meta-Learning

Here we will briefly review a few Meta-Learning approaches used to solve the Continual Learning problem and explain why we chose not to adopt this approach.

Online Meta Learning (OML)

The Online Meta Learning (OML) [39] algorithm suggests a meta-learning approach to continual learning that trains two neural networks, namely a feature extractor that takes in the input and produces representations, and a predictor that takes in the representations and makes the final predictions. Leveraging meta-learning, OML meta-learns the feature extractor to produce representations for a predictor that gets trained on each task, while the feature extractor is frozen. The goal is for the feature extractor to produce representations that are robust such that using them, the predictor can be trained on any task quickly (in a few optimization steps) and show good performance in the end.

OML considers each task to be defined by a stream of samples $\mathcal{T} = (x_1, y_1), (x_2, y_2), ...(x_t, y_t), ...$ with x being the input and y the labels, sampled from sets $\mathcal{X}$ and $\mathcal{Y}$. Assuming a marginal distribution $\mu : \mathcal{X} \rightarrow [0, \infty]$ that describes the observation frequency of each input, individual x_ts can potentially depend on previous samples x_{t-1} and x_{t-2}. Labels y_t, on the other hand, depend only on their respective x_t. OML defines *trajectory* S_k of length k to be a random sequence of samples $(x_{j+1}, y_{j+1}), (x_{j+2}, y_{j+2}), ..., (x_{j+k}, y_{j+k})$ from task $\mathcal{T}$, with $p(S_k|\mathcal{T})$ representing the distribution of all length-k trajectories the can be sampled for task $\mathcal{T}$. OML aims to learn a function $f_{W,\theta}$ predicting y_t from x_t that optimizes the following objective for each task $\mathcal{T}$:

$$\ell_{\mathcal{T}}(W, \theta) = \mathbb{E}\left(\ell(f_{W,\theta}(\mathcal{X}), \mathcal{Y})\right) = \int \left[\int \ell(f_{W,\theta}(x), y) p(y|x) dy \right] \mu(x) dx \tag{3.18}$$

where W and θ represent the network weights, and ℓ is a chosen loss function (e.g. Cross Entropy). Each training session involves only a trajectory S_k sampled from $p(S_k|\mathcal{T})$, and the network should be able to optimize 3.18 from this trajectory alone.

OML proposes two networks, a feature extractor with slow weights θ, and a predictor network with fast weights W. The slow weights θ are trained using the meta-objective of 3.18 and fixed during inference time, whereas the fast weights W are trained for each task using a single trajectory S with the chosen loss function. Assuming a distribution over tasks of $p(\mathcal{T})$, the overall OML objective can then be written as:

$$\min_{W,\theta} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \text{OML}(W, \theta) = \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \sum_{S_k^j \sim p(S_k|\mathcal{T}_i)} \ell_{\mathcal{T}_i}\left(U(W, \theta, S_k^j)\right) \tag{3.19}$$

where S_k^j is a trajectory of length k sampled from the jth task's dataset, and $U(W_t, \theta, S_k^j) = (W_{t+k}, \theta)$ is the inner-loop optimization function with W_{t+k} representing the weights after k optimization steps. Specifically, the jth optimization step updates weights W_{t+j-1} using (x_{t+j}, y_{t+j}) of the corresponding task to get W_{t+j}. The OML algorithm is described in algorithm 1. Since it will be computationally expensive to evaluate the learned weights on all tasks' data from $p(\mathcal{T})$, OML approximates the objective in 3.19 by evaluating the network on a trajectory of the same task it was trained on, as well as a random sample of the available tasks. Essentially, this encourages the network to learn a new task rapidly, while still performing well on previously learned tasks. Also note that in the algorithm 1, it is impractical to randomly reinitialize *all* of the weights W, as the predictor will lose the knowledge acquired from previous tasks, and will not be able to perform well on previous tasks when trained only on the current tasks. Only the weights leading to nodes essential for prediction of the current task (the classes being trained on, or regression output nodes) are reinitialized with random values, and the remaining weights are left untouched.

In evaluation time, the slow weights θ are frozen and the fast weights W are trained and fine-tuned on the each incoming task. OML not only provides a feature extractor with representations that could be used for any task from $p(\mathcal{T})$, it also provides an algorithm that can adapt quickly to a given task, with few available training examples. The training procedure explained acts as a pre-training stage to OML, building a good feature extractor applicable to an arbitrary (in-distribution) task. OML does not provide forward or backward transfer (as the slow weights remain unchanged in meta test/inference time), but it makes up for it by being able to quickly learn a predictor network. It does not need a memory to revisit old samples and

Algorithm 1 OML Meta-Learning Algorithm

Require: $p(\mathcal{T})$, the distribution of task datasets
Require: α and β, step size hyperparameters
  Initialize the slow weights θ randomly
  while not done **do**
    Initialize the fast weights W randomly
    Sample a task $\mathcal{T}_i \sim p(\mathcal{T})$
    Sample a trajectory S_i from $p(S_k|\mathcal{T}_i)$
    Set $W_0 \leftarrow W$
    for $j = 1, 2, \ldots, k$ **do**
      $(x_j, y_j) \leftarrow S_i[j]$
      $W_j \leftarrow W_{j-1} - \alpha \nabla_{W_{j-1}} \ell_i \left(f_{\theta, W_{j-1}}(x_j), y_j \right)$
    end for
    Sample a test trajectory $S_{i,\text{test}} \sim p(S_k, \mathcal{T}_i)$
    $\theta \leftarrow \theta - \beta \nabla_\theta \ell_i \left(f_{\theta, W_k}(S^x_{i,\text{test}}), S^y_{i,\text{test}} \right)$
  end while

the computational requirements do not grow. While OML offers interesting features, it has a few drawbacks:

1. Does not provide forward transfer.

2. Does not provide backward transfer.

3. Needs a dataset of tasks for pre-training.

4. Computationally expensive to train on large datasets.

It is also good to note that the objective of OML (to quickly adapt to a new task with robust representations), while useful and effective, differs from the objective of continual learning. While fast adaptations is one of the key aspects of biological brains, the ability to accumulate knowledge to improve performance on past and future tasks is also essential. Since the slow weights θ are frozen at inference time, if a task that is slightly out of the distribution of $p(\mathcal{T})$ is encountered, the trained network will not be able to adapt, as it relies on representations given by the fixed feature extractor. A continual learner agent should, however, be able to find similarities between the tasks it was trained on and the new (slightly) out of the distribution task, to learn generic features that are effective for both tasks, as if the network was trained on them jointly.

A Neuromodulated Meta-Learning Algorithm (ANML)

A Neuromodulated Meta-Learning Algorithm (ANML) introduced in the paper "learning to continually learn" [6], leverages the objective introduced in OML [39] with a novel neuromodulation-inspired architecture. While OML uses a feature extractor with slow meta-learned weights followed by predictor network with fast weights, ANML utilizes a neuromodulatory network

with slow meta-learned weights that gates the last layer activations of a predictor neural network (with fast weights). The two networks have the same architecture, except for the last layer of the predictor, which includes a linear map for classification/regression.

The last layer activations of the predictor network are multiplied (scalar) by the output of the neuromodulatory network. With this operation, the neuromodulatory network can control what kind of information is passed through in the last layer, and as a result, gradients are modulated as well. Consequently, the neuromodulatory network can in theory guide the activations and (indirectly) gradients to avoid catastrophic forgetting. This assembly of two networks is optimized using the OML objective 3.19 with the same algorithm as in 1.

At inference time, all the network parameters (neuromodulatory and predictor network) are frozen, with the exception of the last layer of the predictor network. The last layer is fine-tuned on each task encountered.

The architecture change in ANML improves the performance of OML, as it can learn more than 600 tasks in sequence, using a few samples for each task, without catastrophic forgetting. It is not clear, however, that which aspects of the new architecture can the improved performance be attributed to. Compared to OML [39], ANML uses CNNs for both the neuromodulatory and predictor networks, whereas OML uses 6 convolutional layers for the feature extractor and 2 fully-connected layers for the predictor. While the number of parameters used is comparable, whether this architecture difference is the source of improved performance is not ruled out.

All in all, ANML illustrates how simple neurmodulation-inspired design can achieve high performance in continual learning. The drawbacks in meta-learning approaches like OML, however, remain:

1. Does not provide forward transfer.
2. Does not provide backward transfer.
3. Needs a dataset of tasks for pre-training.
4. Computationally expensive to train on large datasets.
5. Problem with learning and accumulating knowledge from out of distribution tasks.

3.5.4 Representation Learning

In this section, a class of Continual Learning approaches that learn and update representations as the training on new tasks progresses is explained. As will be seen, Representation Learning allows us to build upon high performing previous work. Moreover, the recently popular contrastive loss can be employed when working in the context of Representation Learning.

Incremental Classifier and Representation Learning (iCaRL)

Incremental Classifier and Representation Learning or iCaRL [72], is one of the relatively early works in continual learning. iCarL is based on *rehearsal* of past samples, representation learning, and regularization using a distillation loss. It leverages representation learning and a nearest class mean classifier (NCMC) to learn tasks in the class-incremental setting. iCaRL

has three components: The feature extractor network ϕ with parameters θ, the NCMC, and an exemplar management system.

Training of the network is essentially fine-tuning on the current task with two additions. First, each batch is augmented with a number of past task exemplars to help prevent forgetting. Second, a distillation loss is added to ensure that the produced representations do not stray to far away from their initial values.

The exemplar management system, stores new samples from the current task and discards samples from previous tasks to ensure the number of exemplars stored does not exceed a predefined limit. As a result, two routines are provided to control the storing and deletion of exemplars. The storing routine iterative selects and adds an exemplar to the memory until the limit for a class's memory is reached. At each step, a sample of the current class training set that makes the average representation of samples in the memory best approximate the average of all class samples' representation is added to the memory. The storing routine enables the second routine, namely the deletion of memory samples, to become relatively easy. In order to remove k samples for a class's memory, it simply removes the last k samples that were added, as the addition of samples to the memory has been done in a sorted manner, with the most representative samples added first. The algorithms for storing and discarding samples are illustrated in 2 and 3 respectively.

Algorithm 2 iCaRL: ConstructExemplarSet

Require: Exemplar set $X = \{x_1, x_2, \ldots, x_n\}$ belonging to class y
Require: m the limit for number of samples of class y in memory
Require: ϕ the feature extractor function
  $\mu \leftarrow \frac{1}{n} \sum_{x \in X} \phi(x)$ ▷ True class mean of all samples
  for $k = 1, 2, \ldots, m$ **do**
    $\text{proto}_k \leftarrow \underset{x \in X}{\text{argmin}} \left\| \mu - \frac{1}{k} [\phi(x) + \sum_{j=1}^{k-1} \phi(\text{proto}_j)] \right\|$
  end for
  $P \leftarrow (\text{proto}_1, \text{proto}_2, \ldots, \text{proto}_m)$
  return P

Algorithm 3 iCaRL: ReduceExemplarSet

Require: Current exemplar memory $P = (\text{proto}_1, \text{proto}_2, \ldots, \text{proto}_{|P|})$
Require: m the limit for number of samples
  return $P = (\text{proto}_1, \text{proto}_2, \ldots, \text{proto}_m)$

The NCMC works by computing class means from samples in the memory. Specifically, samples for each class are fed into the feature extractor to get representations, which then are averaged to become class prototypes. For evaluation and classification of a sample x, the class label with the most similar prototype is returned:

$$y^\star = \underset{y=1,\ldots,t}{\text{argmin}} \|\phi(x) - \mu_y\| \tag{3.20}$$

where t denotes the number of classes seen so far. The network output (for training and defining

the loss) are produced by a function g and a set of vectors $w_1, w_2, \ldots, w_t$:

$$g_y(x) = \frac{1}{1 + \exp(-a_y(x))} \quad \text{where} \quad a_y x = w_y^T \phi(x) \tag{3.21}$$

In order to train a task and update the feature extractor, first a dataset D is constructed by augmenting current task data (of classes s to t, $X^s, \ldots, X^t$) with exemplar sets $P = (P_1, P_2, ..., P_{s-1})$ from memory:

$$\mathcal{D} = \bigcup_{y=s,\ldots,t} \{(x, y) : x \in X^y\} \cup \bigcup_{y=1,\ldots,s-1} \{(x, y) : x \in P^y\} \tag{3.22}$$

For distillation, before the start of training on the current task, the network outputs are stored. Specifically, the network output of the ith sample from class y is recorded in q_i^y. The loss can then be defined as:

$$\begin{aligned} \ell(\theta) = -\sum_{(x_i,y_i)\in\mathcal{D}} \Bigg[& \sum_{y=s}^{t} \delta_{y=y_i} \log(g_y(x_i)) + \delta_{y\neq y_i} \log(1 - g_y(x_i)) \\ & + \sum_{y=1}^{s-1} q_i^y \log(g_y(x_i)) + (1 - q_i^y) \log(1 - g_y(x_i)) \Bigg] \end{aligned} \tag{3.23}$$

After updating the feature extractor using the loss function of 3.23, the memory is updated by first discarding samples to make room for new class samples according to algorithm 3, such that the memory is divided equally between classes. New samples are then added to memory according the storing procedure in algorithm 2.

iCaRL, while working in the class-incremental setting, was able to outperform methods such as Learning without Forgetting (LwF) [52] that were designed to solve the easier problem of task-incremental continual learning. Leveraging representation learning, iCaRL is able to update class representations as it encounters new classes and tasks, allows forward and backward transfer, and does not put any limits on the number of classes it can learn. Nevertheless, iCaRL has a few drawbacks:

1. It needs to store samples for each class, and the performance relies on the size of the memory.

2. The performance on large scale datasets can be better.

Due to flexibility and good performance of iCaRL, we will adopt a similar Representation Learning approach towards solving the Continual Learning problem. However, the loss function will be set similar to Contrastive Learning as described in subsequent sections.

Contrastive Learning of Visual Representations (SimCLR)

SimCLR [16] introduces a framework for self-supervised learning of visual representations, by simplifying the previously proposed contrastive self-supervised algorithms. SimCLR explores various design choices systematically, and provides settings where self-supervising representation would lead to then state-of-the-art performance on large scale image classification datasets. Specifically, SimCLR shows

1. Combining image data augmentation variants is essential to defining contrastive prediction tasks, with stronger augmentations needed compared to supervised methods.

2. Adding a projection network for a non-linear transformation of representations to embeddings in a space where the loss is defined considerably improves performance.

3. Contrastive loss defined on normalized embeddings (the output of the projection network) enhances representation learning.

4. Larger batch sizes and longer training times help contrastive learning more than supervised learning, while deeper and wider (with more channels in each layer) networks similarly benefit both contrastive and supervised learning algorithms.

For a complete analysis of what role each of the described components play in the performance of SimCLR, see [16]. Here we only describe how the components in the SimCLR framework work.

The are four modules at the core of SimCLR. A *data augmentation* module takes each sample x in the mini-batch and generates two augmented samples $\tilde{x}_i$ and $\tilde{x}_j$ using a stochastic image augmentation operation. Each of the augmented images are called to be *views* of the original image. The augmentation operations typically involve

- A random crop of the image that is then resized to the original image size.
- A random color jitter that distorts colors of the original image.
- Adding Gaussian noise to the image.
- Turning an image to grayscale.
- Flipping the image horizontally or vertically, if suitable.
- Rotation by a random degree, if suitable.

The second module is a neural network termed *base encoder* and denoted by $f(.)$. The base encoder takes in the input images and generates representation vectors. The encoder used in SimCLR is simply a ResNet [35]. The representation of augmented images can then be computed by taking the output of the average pooling layer: $h_i = f(\tilde{x}_i) = \text{ResNet}(\tilde{x}_i)$.

The third module is called the *projection head*, a neural network denoted by $g(.)$ that transforms the representations h to embeddings z in a space where the contrastive loss is applied. In SimCLR, a simple Multi-Layer Perceptron (MLP) is used as the projection head: $z_i = g(h_i) = W^{(2)}\sigma(W^{(1)}h_i)$ where σ is an activation function (ReLU was used).

The fourth and final module is the contrastive loss. For each mini-batch of size N, each image is augmented twice to get two views and each view is fed to the encoder and projection networks. This will result in $2N$ embeddings, two for each image. For each image, the two views generated from it are considered positive samples and the remaining $2(N-1)$ images are defined to be negative samples. The similarity of two samples can be defined as the dot product of their ℓ_2 normalized embeddings u and v:

$$\text{sim}(u, v) = \frac{u^T v}{||u||||v||} \tag{3.24}$$

The contrastive loss for a positive pair (i, j) can then be defined as:

$$\ell_{i,j} = -\log \frac{\exp(\text{sim}(z_i, z_j)/\tau)}{\sum_{k=1}^{2N} \mathbf{1}_{[k \neq i]} \exp(\text{sim}(z_i, z_k)/\tau)} \tag{3.25}$$

where $\mathbf{1}_{[k \neq i]} \in \{0, 1\}$ is equal to 1 iff $k = i$, and 0 otherwise. The τ parameter is known as the *temperature*, controlling the sharpness of $\exp(\text{sim}(i, j))$. The overall contrastive loss is computed by summing up $\ell_{i,j}$ for all positive pairs (i, j) (as well as (j, i)) in the mini-batch. The training then proceeds with computing and backpropagating the contrastive loss and taking an optimization step.

Intuitively, the loss in 3.25 encourages representations of different views of the same image to be as similar as possible (with a chosen definition of similarity, here the dot product of their embeddings), while pushing away embeddings extracted from views of different images. Overall, it will end up training the encoder and projection networks to assign representations that will be similar for similar images, effectively learning and encoding visual representations.

Learning representations in a self-supervised manner, SimCLR has demonstrated state-of-the-art performance in self-supervised image classification tasks. Its impressive performance raised the question of whether the SimCLR framework can be applied to supervised learning settings, and whether it was possible to leverage available labels as well.

Supervised Contrastive Learning (SupCon)

In a work called "Supervised Contrastive Learning" [42], inspired by the appealing performance gains of SimCLR, a similar contrastive loss for supervised learning scenarios was introduced. The proposed loss is essentially an extension of the contrastive loss used in SimCLR. The authors examined two versions of the supervised contrastive loss (SupCon) and reported back the version that performed better. Here, the proposed loss is reviewed without in-depth analysis, and the reader is referred to [42] for a more complete analysis of the SupCon loss.

Assuming the same modules of SimCLR [16], namely the data augmentation module, the base encoder $f(.)$, and the projection network $g(.)$, the only different component in Supervised Contrastive Learning is the loss. Augmenting each image in a mini-batch twice to get corresponding views, a set of size $2N$ can be constructed. Let $I = \{1, 2, ..., 2N\}$ denote the index of this multi-viewed mini-batch. Also let $A(i)$ represent I minus the ith sample, namely the view $\tilde{x}_i$. The set $P(i) = \{p \in A(i) : y_p = y_i\}$ is used to denote the set of *positive* samples, i.e. samples other than i sharing the same label y_i. With $z_i = g(f(\tilde{x}_i))$ representing the embedding resulted from passing the view $\tilde{x}_i$ to the encoder and projection networks, the SupCon loss is defined as:

$$\ell_{\text{SupCon}} = -\sum_{i \in I} \frac{1}{\|P(i)\|} \sum_{p \in P(i)} \log \frac{\exp(\text{sim}(z_i, z_p)/\tau)}{\sum_{a \in A(i)} \exp(\text{sim}(z_i, z_a)/\tau)} \tag{3.26}$$

where τ is the temperature parameter similar to SimCLR [16].

Intuitively, the loss in 3.26 encourages positive samples, those in the same class or sharing the same label, to have similar representations, while simultaneously pushing away *negative* samples, those with a different class or label. The difference to SimCLR loss in 3.25 is on how the positive and negative view are defined. In SimCLR, positive views were those originating from the same image, while in SupCon, positive samples are defined as samples with the same label.

SupCon has the following features:

1. It results in representations that are more robust to noise than then cross-entropy loss,
2. Encourages learning from hard positive and negative samples,
3. Is less sensitive to the choice of hyperparameters and optimizers, compared to the cross-entropy loss.

Networks trained using SupCon were able to achieve state-of-the-art performance on a variety of supervised image classification datasets. In practice, however, training a network using SupCon can be tricky as the training process is dependent on the batch size, and in case of using a large batch size (batch size $\geq$ 512), a warmup procedure (i.e. starting out with a small learning rate and gradually increasing it, for the first few epochs of training) should be used.

Contrastive Continual Learning (Co^{2}L)

Inspired from recent advances in contrastive representation learning, Co^{2}L [13] asks whether contrastively learned representation would also help learning tasks sequentially. In the paper "Contrastive Continual Learning" authors note that forgetting does not only come from restricted access to past examples, but it rises from limited access to future tasks as well, since learning features and representations that transfer to future tasks is as important as preserving knowledge acquired from previous tasks. In order to learn representations that transfer to future tasks better, Co^{2}L uses the Supervised Contrastive Learning (SupCon) framework [42] and makes an important observation: Representations learned by the SupCon framework suffer considerably less forgetting compared to those learned using cross-entropy loss.

Co^{2}L proposes a modified version of the SupCon loss and a new distillation loss to continually learn representations and preserve them. It uses a decoupled encoder-classifier architecture: An encoder $f(.)$ and a projection network $g(.)$ are used for training, while at evaluation time, a linear classifier $w(.)$ is trained, taking in representations produced by the encoder $f(.)$ and predicting class probabilities.

Co^{2}L uses rehearsal via a memory buffer to help mitigate forgetting, as every mini-batch of the current task samples is augmented with some samples from a memory. Co^{2}L calls its proposed loss "Asynchronous SupCon". While in SupCon the loss is summed across all samples in the mini-batch, in the proposed Asynch SupCon the loss is only summed up for current task samples. Using the same terminology as described for SupCon, let $S \subset \{1, 2, \ldots, 2N\}$ denote the index of samples for the current task (excluding memory samples). The modified loss can then be written as:

$$\ell_{\text{Asynch}}^{\text{SupCon}} = -\sum_{i \in S} \frac{1}{\|P(i)\|} \sum_{p \in P(i)} \log \frac{\exp(\text{sim}(z_i, z_p)/\tau)}{\sum_{a \in A(i)} \exp(\text{sim}(z_i, z_a)/\tau)} \tag{3.27}$$

The purpose of this modification is to prevent the network from overfitting to a sample subset of previous task samples from memory. In the experiments, it is empirically shown that this actually is the case, and the modified loss improves performance.

For distillation and preserving learned representations, Co^{2}L introduces the Instance-wise Relation Distillation (IRD) loss. Commonly used for distillation, before training of the current task, the weights of the encoder network at the end of the training of the previous task are stored ($\theta^\star$). At training, for each multi-viewed mini-batch, two sets of representations are computed, one with the stored weights $\theta^\star$, denoted as z_i^{past} and the other with current weights θ, denoted as z_i. For each view x_i in the mini-batch of representation, a vector p, termed the instance-wise similarity vector is computed:

$$p(x_i) = [p_{i,1}, p_{i,2}, ..., p_{i,i-1}, p_{i,i+1}, ..., p_{i,2N}] \tag{3.28}$$

where $p_{i,j}$ is the normalized instance-wise similarity:

$$p_{i,j} = \frac{\exp(\text{sim}(z_i, z_j)/\kappa)}{\sum_{k\neq i}^{2N} \exp(\text{sim}(z_i, z_k)/\kappa)} \tag{3.29}$$

κ is yet another temperature parameter. The IRD loss can then be defined as:

$$\ell_{\text{IRD}} = \sum_{i=1}^{2N} -p(z^{\text{past}}).\log(p(z)) \tag{3.30}$$

The overall training loss can then be written as:

$$\ell = \ell_{\text{Asynch}}^{\text{SupCon}} + \lambda\, \ell_{\text{IRD}} \tag{3.31}$$

with λ controlling the weight of distillation.

For managing the samples in the memory, whenever the memory becomes full and space needs to be freed, a sample is randomly pushed out to make way for new class samples. This will keep the memory divided (approximately) equally between classes.

Co^{2}L demonstrates state-of-the-art performance for various computer vision datasets, and in all of the task-incremental, class-incremental, and domain-incremental settings. It offers forward and backward transfer as the encoder can learn new features by contrasting new task data against previous tasks, is successful in mitigating forgetting, and uses fixed computational resources and memory. The only drawback to this method is that the adaptation power of the network decreases over time. While the network can achieve near perfect accuracy on the first task, by the time it gets trained on the fifth task, there seems to be a limit on the accuracy on the new task that the network is unable to go beyond. While the contrastive loss (SupCon) can partially be responsible for this effect, it is more likely that it arises from the distillation loss, as it introduces a regularization to the entirety of the representation vectors, while parts of the representation vectors may not be contributing to preserving the previously acquired knowledge. While Co^{2}L enjoys performance gains resulting from Contrastive Learning, it still suffers from drawbacks seen in previous Continual Learning approaches such as EWC [43], SI [98], and LwF [52]. Regularization of representations is done via distillation loss and according to previous tasks only, with no measure of how likely these representation are to transfer to next upcoming tasks. Although representations learned via Contrastive Learning may be more transferable, contribution from different parts of the representations towards generalizability performance may not be equal. The distillation loss in Co^{2}L implicitly assumes equal contribution and regularizes the whole representations, even parts that may be overfitted.

3.6 Neural Similarity Learning

Finally, in this section, a work that acts as building block in this thesis towards identifying important parts of representations in the Representation Learning context is reviewed.

In Convolutional Neural Networks (CNN), each convolution operation takes in inputs $\tilde{X}$ and multiplies them by a set of kernels with weights $\tilde{W}$, where each kernel is multiplied to the input in different positions, similar to a sliding window. Assuming a kernel of size $C \times H \times V$ with C denoting channels, H height, and V the width, and assuming a portion of the inputs with same size as the kernel (getting multiplied by it), one can flatten the inputs to get X and the kernel to get W. As a result, X and W will be 1-D vectors holding inputs and the kernel weights respectively. The convolution operation can then be thought as a dot product between these two 1-D vectors ($W^T X$), similar to what a simple neuron computes in feed-forward neural networks. In a work named "Neural Similarity Learning" [54], authors propose to generalize the dot product formulation by adding a *bilinear similarity matrix*:

$$f_M(W, X) = W^T M X \tag{3.32}$$

where f is the convolution operation parameterized by M, the bilinear similarity matrix. While it is not necessary to constrain M, in order to be more parameter efficient and avoid imposing a large computational overhead, the authors suggest to let M be a block-diagonal matrix with C shared blocks:

$$f_M(W, X) = W^T \begin{bmatrix} M_s & & \\ & \ddots & \\ & & M_s \end{bmatrix} X \tag{3.33}$$

where $M = \text{diag}(M_s, \ldots, M_s)$ and M_s is of size $HV \times HV$. M_s can be constrained to be a diagonal matrix (diagonal neural similarity or DNS), effectively computing a weighted dot product, or left unconstrained to be more flexible at the cost of more parameters to train (unconstrained neural similarity or UNS).

There can be two variants to neural similarity learning, namely static and dynamic. In static neural similarity learning, the matrix M_s is learned jointly with the rest of the network parameters using backpropagation in an end-to-end fashion. At inference time, the bilinear similarity matrix can be incorporated into weights W and there is no need to store additional parameters. Static neural similarity learning is quite similar to matrix factorization, as the weights in dot product $W^T X$ can now be seen as a factorized $M^T W$.

While the static variant of neural similarity learning can generalize the convolution operation, the bilinear similarity matrix will be learned during training time and maintain its effect regardless of the inputs. Dynamic neural similarity learning, on the other hand, attempts to learn a small network that *generates* M, rather than learning it directly. Specifically, the matrix M will be constructed by blocks M_s that are generated with a neural network $M_\theta(.)$, parameterized by θ. Similar to static neural similarity learning, both the network parameters W and θ can be learned jointly using backpropagation.

Moreover, the bilinear similarity matrix M can be regularized, either via imposing a certain structure (e.g. block-diagonal with shared blocks), or by encouraging it to be sparse (with ℓ_1 norm penalty).

Neural similarity networks (NSN) can be constructed by changing each convolutional layer with one equipped with neural similarity learning (bilinear similarity matrix). In order to reduce the number of additional parameters, authors suggest all convolutional kernels in the same convolution layer to share the bilinear similarity matrix. If all convolution operations are changed with the static (dynamic) variant of neural similarity learning, the resulting network will be a static (dynamic) NSN.

Results in this work showed that the dynamic variant generally outperforms the static variants of NSNs. Additionally, both variants outperform a baseline CNN with the same number of parameters and normal unchanged convolutions. Using unconstrained neural similarity (UNS) matrices, however, performed on par with the diagonal neural similarity (DNS) matrices, showing that DNS matrices are flexible enough for performance gains in image recognition tasks. Overall, neural similarity learning can improve a network's performance by making the convolution (or feed-forward) operations more flexible and enabling adaptation of kernels to inputs.

Chapter 4

Methods

As seen in the Literature Survey chapter, a common approach to continual learning is to find a way to identify parameters that are playing a crucial role in the performance of the DNN on previous tasks. There is, however, a very limiting assumption in both task-incremental and class-incremental settings about the transition to a new task. For task-incremental learning, it is assumed when training on a new task starts, there is no access to previous tasks data, unless it is a few samples stored in the memory. Similarly, in the class-incremental setting, it is assumed that when new classes of data are coming in as a new task, the access to all but a few samples of previous tasks' data in the memory are lost. The same limiting assumption applies to the next tasks' data as well. When training on the current task, it is assumed that no data from the next task is available. This strict assumption limits a learner's ability to contrast new data against previous tasks' data. While moving on from a task A to learn task B, there is limited access either to task A, or B, since at any time, either A is considered a previous task, or B is considered a next task. A more appropriate and practical setting considers a stream of incoming samples $\{(x_1, y_1), ..., (x_t, y_t), ...\}$. Assuming a limited (hardware) memory to store samples from this data stream, the data available to the learner can be seen as a sliding window on this stream. For example, at time t, the learner has access to samples $(x_t, y_t), ..., (x_{t+w}, y_{t+w})$ where w is the length of the window. In this setting and with the disjoint task assumption, when task A's data ends and is succeeded by task B's data, the window will see not only task A's data, but some of task B's data samples as well.

Figure 4.1: The new proposed setting: Data comes in as a stream and access is defined by a sliding window.

Inspired by this observation and the role of neuromodulatory processes in the brain for learning, here we propose a new setting for continual learning that more closely mimics the data access patterns we have in practice, along with three novel methods for knowledge preservation in this new setting: *Selective Distillation*, *Gradient Modulation*, and *Half-Network*. As

stepping stones to create the three new methods, we also introduce two methods to compute and use saliency information to improve continual learning performance. These are *Salient Representative Selection*, and *Salient Excitation Backprop*.

The new setting has a more practical view towards access to next task's upcoming data. Normally, it is assumed that when training on a task begins, the data comes in as batches, rather than being available in its entirety. Given the sliding window formulation, this assumption is too limiting. At a point in time when training on a task begins (named current task), there is very limited access to previous tasks' data, as well as current task data, which is not usual in practice. In this work, we take a small step towards this new setting. It is now assumed when training on a task begins, we have access to a number of samples from the current task (a *predictive batch*). These samples are a *subset* of the first batch of new task data, so *there is no need to have additional access to current task data*, although the new setting allows it. Using the network that has been trained on previous tasks and has not seen the current task data yet, this predictive batch allows us to define the importance of network parameters based on how generalizable and transferable they are, as well as how much they contribute to previous tasks. By evaluating a network on a small sample of current task's data before training on it begins, it is possible to attribute performance on past data *and* the predictive batch of current task data to each individual network parameter, and try to retain the transferable knowledge acquired by the network. To achieve this goal, we first define and compute on the last layer neurons a measure of generalizability using previous tasks' data in memory and the predictive batch of the new upcoming data. Further, we extend the Excitation Backprop (EB) framework [99] to work with our choice of representation learning and to attribute the computed performance measure, rather than a specific output neuron's activation, to every network parameter. We call our approach "Look-Ahead Selective Plasticity", as it leverages a predictive batch of new upcoming data to selectively modulate the plasticity of network weights.

This work can be seen as a motivation towards adopting the proposed setting. Here, we will show that using a small subset of new task's data as the predictive batch can improve continual learning performance, which will in turn suggest that using a larger subset of current task's data can improve performance further. Access to a larger number of samples from a task before training on it begins, however, is not granted in the typical continual learning setting, emphasizing on the plausibility of the proposed setting.

4.1 Saliency Methods

Here we introduce two new methods for computing and using saliency.

4.1.1 Salient Representative Selection

While the previous work used a form of regularization on the entirety of the output of the network to preserve knowledge of past tasks, here we aim to find certain parts of the output that are essential to performing well on previous tasks *and* are likely to perform well on future tasks. After this process, one can either regularize the parts of the output that are found to be important, or find a way to attribute performance of those parts to individual network parameters. In subsequent sections, we will propose methods for both.

Similar to an interesting previous work called Neural Similarity Learning [54], taking in the feature extractor f_θ, we learn a minimal selection mask that identifies a portion of the produced representation (e.g. first half: representation[:middle]) that works equally well or better than the complete representations. Assessing "working well" here is done by calculating accuracy.

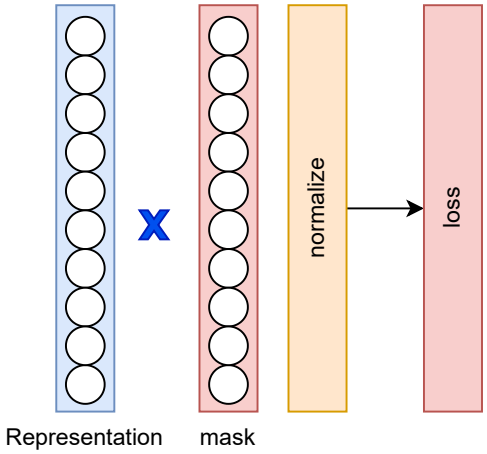

Figure 4.2: Salient Representative Selection: Identifying parts of the representations that transfer well.

First a dataset $\mathcal{D}$ is formed by combining past samples in the memory and a predictive batch of new upcoming data. The number of samples taken from memory is 100, unless the memory size is less than 100, in which case all of the memory samples will be added to this dataset. The number of samples in the predictive batch is also 100. Assuming $\mathbf{r}^{[d]} \in \mathbb{R}^d$ denotes a d-dimensional representation produced by the feature extractor, we aim to learn a d-dimensional mask $\mathbf{m}^{[d]} \in \{0, 1\}^d$ that selects a minimal part of $\mathbf{r}^{[d]}$ that classifies representations well. The selection process takes out a portion of the original representations and thus, the selected representations are going to have lower dimensions. Both the original and new representations are divided by their ℓ_2 norm to lie on a unit hypersphere. Letting $\mathbf{s}^{[k]}$ to denote the selected representation where k is the number of new dimensions, the selected representation can formally be computed as:

$$\mathbf{s}^{[k]} = \frac{\mathbf{r}^{[d]}[m = 1]}{\|\mathbf{r}^{[d]}[m = 1]\|_2} \tag{4.1}$$

In order to define a loss, class means need to be computed first. The mean of class c can be written as:

$$\mathbf{a}_c = \frac{\sum_{i:y_i=c} \mathbf{s}_i^{[k]}}{|\{\mathbf{s}_i^{[k]} : y_i = c\}|} \tag{4.2}$$

where y_i is the ith sample's label, a_c is the mean of class c selected representations, and $\mathbf{s}_i^{[k]}$ is the k-dimensional selected representation of sample i (produced by $f_\theta(x_i)$). The similarity of a selected representation $\mathbf{s}_i^{[k]}$ to a class mean $\mathbf{a}_c$ is defined as their normalized dot product (cosine distance):

$$\text{sim}(\mathbf{s}_i^{[k]}, \mathbf{a}_c) = \frac{\mathbf{s}_i^{[k]} \cdot \mathbf{a}_c}{\|\mathbf{s}_i^{[k]}\| \, \|\mathbf{a}_c\|} \tag{4.3}$$

For a sample (x_i, y_i), the cross entropy loss can be defined as:

$$\ell_i = -\log \frac{\exp(\text{sim}(\mathbf{s}_i^{[k]}, \mathbf{a}_{y_i}))}{\sum_{c=1}^{C} \text{sim}(\mathbf{s}_i^{[k]}, \mathbf{a}_c))} \tag{4.4}$$

where C is total number of classes. The total cross entropy loss can either be the sum or the mean of the cross entropy loss for each sample. A simplified nearest class mean (NCM) loss can also be defined:

$$\ell_i = -\text{sim}(\mathbf{s}_i^{[k]}, \mathbf{a}_{y_i}) \tag{4.5}$$

which tries to maximize the similarity of each sample with its corresponding class mean. The total NCM loss is then calculated as the sum of the loss for each sample.

While keeping the computed representations $R = \{\mathbf{r}_i^{[d]} = f(x_i) \, \forall x_i \in \mathcal{D}\}$ frozen and treating the mask $\mathbf{m}^{[d]}$ as the only trainable vector, one of the above mentioned losses can be optimized to find a mask that selects a portion of the representation that classifies the samples best. Moreover, in order to encourage the selecting mask $\mathbf{m}^{[d]}$ to be sparse and minimal (i.e. minimize k in $\mathbf{s}_i^{[k]}$), an ℓ_1 norm loss is added. The total loss will be:

$$\ell_{\text{total}} = \ell_{\text{classification}} + \lambda \|\mathbf{m}^{[d]}\|_1 \tag{4.6}$$

where λ controls the trade-off between classification performance and the sparsity of the mask.

Optimizing for $\mathbf{m}^{[d]}$, a portion of the original representation will be selected. We will call the selected output neurons to be the *salient* ones. However, there needs to be a quantitative measure of salience. If the original representations were not classifying the samples well in the first place, there is little chance that this selected portion outperforms the original representations by a large margin. To define the salience more quantitatively, the salience of selected neurons will be set to be their nearest class mean accuracy on dataset $\mathcal{D}$. As a result, the salience of the ith output neuron can be defined as:

$$\hat{\mathbf{s}}_i = \begin{cases} 0 & \mathbf{m}[i] = 0 \\ \text{NCM accuracy} & \mathbf{m}[i] = 1 \end{cases} \tag{4.7}$$

$\hat{\mathbf{s}}$ will be the output of this *Salient Representative Selection* process, where salient output nodes that are representative of the original representations are selected. $\hat{\mathbf{s}}$ will be used as an input to the extended EB framework to compute the salience of all network weights.

4.1.2 Salient Excitation Backprop

Given a prior distribution on the output neurons of a neural network, the EB framework is able to attribute that prior distribution to all of the neurons in the network, but not the weights between them (recall that in the EB a neuron is defined as $\hat{a}_i = \phi(\sum_j w_{ji}\hat{a}_j + b_j)$, and the prior distribution is attributed to a_is). Moreover, the EB framework backpropagates the top-down attention signal only through the positive weights [99]. In the feature extractor, the EB framework will attribute the given prior distribution to neurons with the ReLU activation function properly, as expected. However, for the last fully-connected layer in a ResNet [35] this does not hold, since representation learning via a contrastive loss objective is being performed,

and neurons in the second to last layer can play a significant role even if they are connected to the output layer with a negative weight, as long as the output node is salient. Formally, assuming a_j to be a neuron in the second to last layer, a_i to be a salient neuron in the last (output) layer (according to the Salient Representative Selection), and w_{ji} the weight connecting these two neurons, we propose to modify the Marginal Wining Probability (MWP) calculation in the EB framework only for the last year as follows:

$$P(a_j|a_i) = Z_i \, \hat{a}_j \, \text{abs}(w_{ji}) \tag{4.8}$$

where Z_i is just a normalization factor $Z_i = \frac{1}{\sum_j \hat{a}_j \text{abs}(w_{ji})}$. The MWP calculation will remain the same for all other network layers, where a_j is in the child node set C_i of a_i in top-down order:

$$P(a_j|a_i) = \begin{cases} Z_i \hat{a}_j w_{ji} & \text{if} w_{ji} \geq 0, \\ 0 & \text{otherwise.} \end{cases} \tag{4.9}$$

As a result, a neuron in the second to last layer can be salient even if it is connected to a salient node with a negative weight.

Furthermore, to attribute salience to network weights, similar to Oja's rule [66], we set the salience of a weight (i.e. connection) to be the geometric mean of the salience of neurons on its two ends. Formally, considering w_{ji} as the weight between neuron a_i and a_j, and denoting the salience of these two neurons with $P(a_i)$ and $P(a_j)$, the salience of the weight w_{ji} can be defined as:

$$P(w_{ji}) = \sqrt{P(a_i)P(a_j)} \tag{4.10}$$

With this extension of the EB framework, given the salient neurons $\hat{\mathbf{s}}$ in the output layer, the importance (salience) of each weight can be computed using an additional backward pass. Using these computed salience values, three novel techniques to retain a network's (transferable) knowledge of past tasks are introduced.

4.2 Knowledge Preservation Methods

Based on the two saliency methods described above, we introduce three different methods for knowledge preservation which we will evaluate and compare.

4.2.1 Selective Distillation

This method does not require the extended EB framework to compute weight salience. It only needs the salience map on the output layer of the feature extractor (representations). While works such as Co^2L [13] and LWF [52] use knowledge distillation on the entirety of the output of the network, here we propose to distill only the part of the representation that is identified to be salient. This method aims to allow the network more freedom in learning new tasks while ensuring at minimal yet salient part of the representation is regularized to work well on previous tasks as well. Compared to previous work, there are two main contributions:

- The distillation loss is defined on a portion of the produced representations, in contrast to all of it, allowing more learning freedom.

- The criteria for choosing this portion is the portion's performance on previous tasks, as well as a predictive batch of the upcoming new task, encouraging the network to preserve representations that transfer.

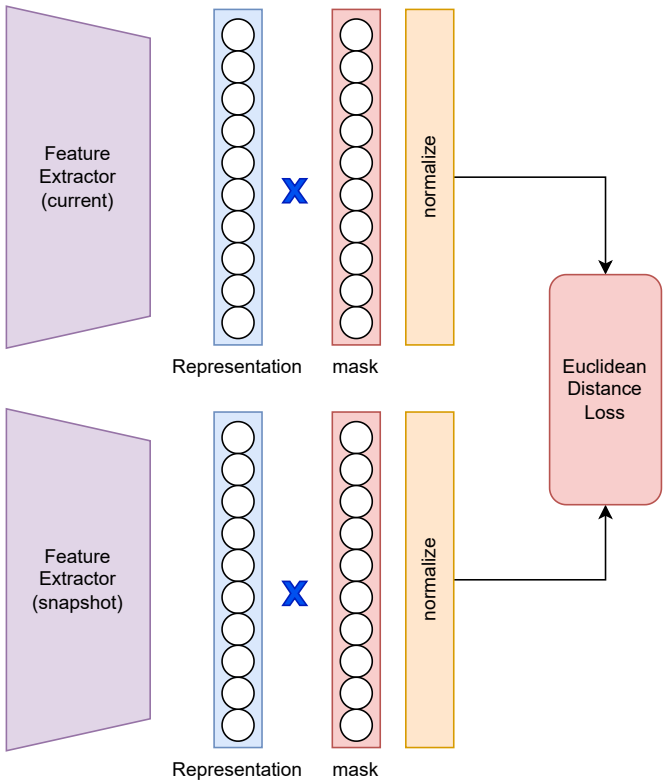

Figure 4.3: Selective Distillation: Knowledge Distillation only on parts found to be salient.

To add this distillation loss, before the start of training on a new task, feature extractor weights $\theta^\star$ are saved. Next, at training time, assuming the network weights to be denoted by θ, the following distillation loss is added to the SupCon loss:

$$\ell_{\text{distill}} = \left(\sum_{x_i \in \text{minibatch}} \left(\frac{f_\theta(x_i) \odot \hat{\mathbf{s}}}{\|f_\theta(x_i) \odot \hat{\mathbf{s}}\|_2} - \frac{f_{\theta^\star}(x_i) \odot \hat{\mathbf{s}}}{\|f_{\theta^\star}(x_i) \odot \hat{\mathbf{s}}\|_2} \right)^2 \right) / |\text{minibatch}| \tag{4.11}$$

where $\odot$ denotes element-wise product.

This loss ensures that the selected salient parts of the representations are preserved when new representations are being learned. The total loss will then become:

$$\ell_{\text{total}} = \ell_{\text{SupCon}} + \lambda \, \ell_{distill} \tag{4.12}$$

4.2.2 Gradient Modulation

For this method, before the start of training on a task, using the Salient Representative Selection process, the salient output nodes performing well on the previous tasks and the predictive batch

of new upcoming data are identified. Next, EEB is employed to compute the salience of each feature extractor parameter/weight. This method is then applied in each iteration of the current task's training.

Specifically, after backpropagating the loss back to the feature extractor weights θ, the gradients are modified according to each weight's salience. Assuming $g_{w_{ji}}$ to be the gradient with respect to weight w_{ji} and $P(w_{ji})$ to be the weight's importance, the gradient is modulated as follows:

$$g_{w_{ji}} = g_{w_{ji}} \times (1 - \min(1, P(w_{ji}))) \tag{4.13}$$

As a result, the more salient a weight is, the smaller the gradients become. In the extreme case that a weight's importance is $P(w_{ji}) \geq 1$, the weight will not change for the duration of training on a task. In the other extreme, if a weight's importance is near zero, it will be trained by the loss with no limitation/regularization. Similar to how neuromodulator processes in the brain can modulate the plasticity of a neuron, this process attempts to retain the important network weights by limiting change (plasticity) while allowing the less important weights to be learned via the loss and the normal training process.

4.2.3 Half-Network

In addition to previous methods, a half-network approach is introduced. While the gradient modulation method can retain important weights, there is concern for the case that the gradient modulation method deems a large portion of network weights to be important and limits the network too much. In order to examine this scenario, the half-network approach regularizes half of the network weights in each layer, leaving the other half to be trained normally. Specifically, in each convolutional layer, weights are averaged in each channel to get an overall channel importance. Assuming $P(c_i^{(l)})$ to denote the importance of the ith channel (kernel) in the lth layer, for each layer the channels are sorted according to their salience. The weights residing in the top-half channels with the largest importance are then regularized based on their initial value before training on the current task started.

Specifically, before the training on a task begins, a snapshot $\theta^\star$ of the feature extractor weights are taken. A set I of the index of the important weights is then formed by identifying the weights residing in the important channels. By definition, the size of I will be at most half of the network parameters ($|I| \leq |\theta|/2$). At each training iteration, a regularization loss is added to the SupCon loss:

$$\ell_{\text{hn-reg}} = \frac{\sum_{i \in I}(w_i - w_i^\star)^2}{|I|} \tag{4.14}$$

where $w_i^\star$ is the weight stored in $\theta^\star$ corresponding to the current weight $w_i \in \theta$. The total loss then becomes:

$$\ell_{total} = \ell_{\text{SupCon}} + \lambda\, \ell_{\text{hn-reg}} \tag{4.15}$$

where λ controls the degree of regularization.

4.2.4 Baselines

To evaluate how each method performs, experiments with a baseline setting were also conducted. The baseline setting simply uses the SupCon loss and a memory module without any

other additions (no supplementary method). As a result, this setting will enable the comparison and evaluation of each method with the baseline setting where it was not present. Additionally, a second baseline named *simultaneous training* is added to show the result when training was performed on the entire dataset (MNIST, CIFAR10, or TinyImageNet) on all classes simultaneously. The simultaneous training baseline usually shows an upperbound on the performance of the continual learner, as it had access to the entirety of each the dataset during training, and was trained on all classes at the same time.

4.3 Training Procedure

We have chosen to work on the class-incremental setting, as it is more challenging and more useful in practice, and the task ids/descriptions are not usually available. To learn and evaluate in this setting, we divide each dataset (MNIST, CIFAR10, and TinyImageNet) into 5 tasks, with equal number of classes in each task. For example, the dataset for the first task of MNIST involves digits 0 and 1, while for SplitTinyImageNet, the first task's dataset includes the first forty classes.

Most of the training process is based on Co^2L [13]. The architecture, similar to Co^2L is a decoupled encoder-classifier. The term *feature extractor* will also be used to refer to the encoder. For larger datasets of SplitCIFAR10 and SplitTinyImageNet, a reduced version of ResNet-18 as in [58] is used for the feature extractor to produce representations (extracted from the output of the fully-connected layer), followed by a single hidden-layer MLP to produce embeddings. For MNIST, a simple 3-layer CNN (16 channels in each convolutional layer, 32 neurons in the fully-connected layer, and a representation size of 32) is used without any projection head, hence the representations and embeddings are in the same space. The Supervised Contrastive loss (SupCon) [42] is then defined on embeddings, and gradients are computed and backpropagated for the training to proceed.

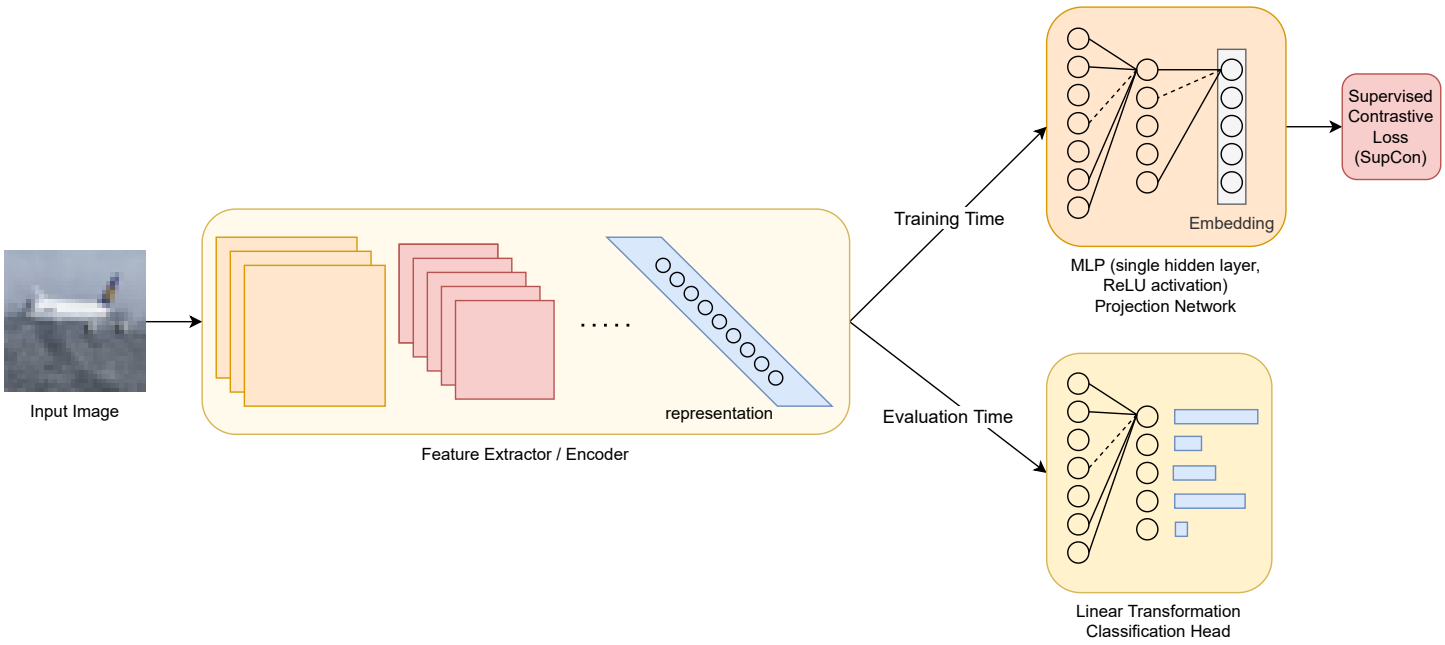

Figure 4.4: Architecture in training and evaluation time

A memory is used to retain a small sample of previous tasks. In each training iteration, the current task mini-batch is augmented with the same number of samples from memory. The memory management is identical to iCaRL [72], with modules for adding and removing examples from the memory, such that the number of samples stored from each class is nearly equal.

The performance measures of precision, recall, and f1-score are computed on each task and observed class after training on each task. While focus will be on f1-score for SplitMNIST as it is slightly unbalanced, for SplitCIFAR10 and SplitTinyImageNet, average accuracy will be used to compared methods with each other and previous work. In order to get predictions, similar to Co^2L [13], after training is finished on a task, a separate classification head with output units the same as the number of classes is jointly trained on the task's dataset and the samples in the memory. During training of the classification head, the projection head is not present and the feature extractor weights are frozen. This classification head is trained with cross entropy loss. After training of the classification head, it can be used to transform representations to class probabilities and compute the mentioned metrics.

The described methods (selective distillation, gradient modulation, and half-network) will be applied after training on the current task finishes and before moving on to the training of the next task.

For easier comparison and accumulation of results, each performance metric (measured and reported after training on each task was finished) was computed and averaged over seen classes (for precision, recall, and f1-score) and seen tasks (for average accuracy). It is good to note that NaN values were discarded for computing the average. Specifically, precision for a class can be undefined/NaN if there were no samples that were predicted to be in that class. Moreover, if precision and recall on a class are zero, then the f1-score will be undefined. The average operator (as well as standard deviation computation) did not consider NaN values, and

reported the results using only the valid measurements.

4.4 Experimental Settings

Most of the hyperparameters are set according to Co^2L [13]. There are, however, some differences. First and foremost, the loss used here is the normal supervised contrastive SupCon loss, not the asynchronous version used in Co^2L [13]. Moreover, a smaller batch size is used for the training of the network on MNIST and CIFAR10, while a large batch size is employed for SplitTinyImageNet. For implementation of the mentioned approaches, the PyTorch (1.12.1) [69] and Avalanche (0.2.1) libraries [57] were employed. All architectures are implemented and trained from scratch. The extension of the excitation backprop was implemented based on the TorchRay library (2.0.01) [26]. Visualizations were made using the seaborn [96] and matplotlib libraries.

Each experiment on SplitMNIST and SplitCIFAR10 was repeated independently 10 times with random seeds, while each experiment on SplitTinyImageNet was repeated 5 times, due to time constraints. Each experiment (one repetition) took 5 minutes on SplitMNIST, 5 hours on SplitCIFAR10, and 20 hours on SplitTinyImageNet using an Nvidia RTX 3090 GPU. Over the course of this thesis, more than 170 experiments were conducted, verified and clear results of which will be presented in the next chapter.

A list of the hyperparameters are provided in table 4.1:

SplitTinyImageNet	Learning Rate	0.5
	Batch Size	512
	Memory size	200
	Projection head embedding size	128
	SupCon loss temperature	0.07
	λ	1
	Start epochs (first task)	300
	epochs (except first task)	250
	Augmentations	TorchVision transforms: RandomResizedCrop(scale=0.2), RandomHorizontalFlip(p=0.5), RandomColorJitter(0.4, 0.4, 0.1) with p=0.8, RandomGrayScale(p=0.2)
SplitCIFAR10	Learning Rate	0.01
	Batch Size	32
	Memory size	200
	Projection head embedding size	128
	SupCon loss temperature	0.07
	λ	1
	Start epochs (first task)	40
	epochs (except first task)	30
	Augmentations	TorchVision transforms: RandomResizedCrop(scale=0.2), RandomHorizontalFlip(p=0.5), RandomColorJitter(0.4, 0.4, 0.1) with p=0.8, RandomGrayScale(p=0.2)
SplitMNIST	Learning Rate	0.5
	Batch Size	16
	Memory size	50
	Projection head embedding size	N/A
	SupCon loss temperature	0.07
	λ	1
	Start epochs (first task)	1
	epochs (except first task)	1
	Augmentations	TorchVision transforms: RandomResizedCrop(scale=0.7) RandomPerspective(distort_scale=0.3, p=0.2)

Table 4.1: Look-Ahead Selective Plasticity Hyperparameters

4.5 Ablation Studies

To better illustrate the effectiveness of proposed ideas and approaches, the following ablations studies were conducted:

4.5.1 Effect of the Using the Predictive Batch

Here the effect of the predictive batch is studied. In order to see whether the network's performance benefits from a predictive batch of upcoming new data, additional experiments with the methods of selective distillation, half-network, and gradient modulation were performed. In these experiments, the salient parts of the representation and salient parameters are identified based on 200 samples (or memory size, whichever is less) from memory alone, instead of the original setting were 100 samples from memory and 100 samples from upcoming data (the predictive batch) were used. By comparing the network's performance for each method with its corresponding variant where predictive batch was not realized, the improvements that are a direct result of the predictive batch can be measured. The results of this ablation study are provided in the results chapter 5.

4.5.2 Effect of Memory Size

Continual Learning methods that leverage a rehearsal memory show varying performance when the size of the memory changes ([13] for example). To examine the effect of memory size, additional experiments with a larger memory were conducted. Specifically, for each method of selective distillation, half-network, and gradient modulation, a corresponding variant with a larger memory of size 100 was trained on SplitMNIST, as well as a corresponding variant with a larger memory of size 1000 for SplitTinyImageNet. All other hyperparameters and settings were kept the same. Comparing performance metrics resulting from these additional variants, it is possible to examine how much a larger memory can benefit a network's performance, and how this improvement varies in different methods. The results of these additional variants are also provided in the results chapter 5.

Chapter 5

Results

In this chapter, the results of experiments using each introduced method (selective distillation, gradient modulation, half-network, and the baseline) are presented. As described in Chapter4, the baseline is essentially sequential training of a network (that is chosen according to the dataset) using rehearsal of memory samples and supervised contrastive loss, without any additional components. The simultaneous training baseline is the training of the network on the entirety of a dataset (no splitting) and all of the classes simultaneously. Selective distillation adds a distillation loss on a portion of representations that are computed to be most important. The half-network method regularizes the top half of the most important parameters in each layer. Finally, gradient modulation multiplies the gradient of each weight by an importance measure to ensure important network parameters see less change.

The architectures used in experiments are a reduced version of ResNet-18 and a simple 3-layer CNN. The ResNet-18 network is used for SplitCIFAR10 and SplitTinyImageNet datasets, while the simple CNN network is used for the SplitMNIST dataset. Each original dataset (MNIST, CIFAR10, TinyImageNet) is divided across classes to 5 equal groups to make up SplitMNIST, SplitCIFAR10, and SplitTinyImageNet respectively. The training procedure involves training the corresponding network on each task of the dataset sequentially: The network is trained on the first task, then the performance metrics on seen classes are evaluated, then it is trained on the second task and performance metrics on all seen classes are computed, and so on.

Moreover, the results for the two ablation studies mentioned in Chapter 4 are described here. These two ablation studies investigate the effect of rehearsal memory size and the predictive batch.

In order to obtain the results, each of the introduced methods is tested on each dataset using the architecture corresponding to that dataset. For each of the mentioned approaches, the metrics precision, recall, and f1-score of the network are evaluated after the training on each task finishes. For brevity and clearer assessment of an approach's performance, the reported measures are averaged across classes. For example, the reported recall metric after training on the first task of SplitMNIST that includes classes 0 and 1 is essentially the average of recalls of classes 0 and 1. Moreover, these metrics are accompanied by confusion matrices computed after the training of each task. These confusion matrices *grow* in the sense that a confusion matrix for a task seen after another contains more classes (rows and columns) as the network has seen and been trained on more classes. It is good to note the metrics reported

at the end of training on all tasks hold more value, as they assess the end result of a model trained sequentially on a set of tasks. For comparison to previous work, the metric of average accuracy will be reported to compared different methods when testing on SplitCIFAR10 and SplitTinyImageNet.

Furthermore, for a better comparison of different methods against each other, each of the metrics of precision, recall, and f1-score are plotted across tasks. These plots will capture the change in metric values across tasks better, and compare this change across proposed methods. This will also illustrate different forgetting trends for different methods and tasks/classes.

5.1 Experimental Results on SplitMNIST

Here the performance metrics of a three layer CNN trained on a splitted MNIST dataset are presented. The performance metrics are reported on a held out test set. There are a total of 5 tasks for SplitMNIST, with the first task including classes 0 and 1, the second task including classes 2 and 3, and so on. Since SplitMNIST is slightly unbalanced, F1-score will be used to compared methods.

5.1.1 Measured F1-Scores

The mean and std of f1-scores after training the network on each task can be seen in table 5.1. For the first task, there is no difference between the methods since the regularization and distillation methods are only introduced from the second task onwards and during the training of the first task there is no prior knowledge to preserve. As expected, performance metrics are similar for different methods. Task 2 is trained immediately after task 1, and there has not been much time for the network to forget task 1. Moreover, all of the samples in the memory are from task 1 when network is being trained on task 2, leading to revisiting of past samples with a higher diversity (more samples). Nevertheless, the two methods that work better on this dataset, namely gradient modulation and selective distillation show slight improvement compared to the baseline. The half-network method, however, has slightly decreased the performance of the baseline method. For the third task, all of the proposed methods, gradient modulation, selective distillation, and half-network improve the performance over the baseline on the f1-score metric. Gradient modulation and selective distillation, however, provide a more significant increase compared to the half-network approach. In the fourth task, the network has had reasonable time to forgo changes that result in forgetting about previous tasks. But even the baseline still performs well. This can be attributed to the use of a memory module (although small), and the SupCon [42] loss's robustness against forgetting in small networks. Similar to previous task results, all the proposed methods outperform the baseline, where selective distillation demonstrates the best performance, the gradient-modulation method a close second, and the half-network approach providing the least amount of improvement among these three methods. The performance metrics for the fifth and final task are shown in last column of table 5.1. The selective distillation method shows to be the most effective in preventing forgetting and has the best overall performance in terms of the f1-score. The gradient modulation method also benefits the network with a better performance but the gain in performance metrics is less than

selective distillation. Lastly, the half-network method fails to improve upon the baseline and degrades performance.

Table 5.1: SplitMNIST: Mean (std) of class f1-score on the test set after each task

method	Task 1 f1-score	Task 2 f1-score	Task 3 f1-score	Task 4 f1-score	Task 5 f1-score
simultaneous training	0.9908 (0.0051)	0.9869 (0.0092)	0.9845 (0.0145)	0.9843 (0.0135)	0.9823 (0.0161)
baseline	0.9988 (0.0011)	0.983 (0.0102)	0.9596 (0.0219)	0.9338 (0.0232)	0.8837 (0.0198)
gradient-modulation	0.999 (0.0007)	**0.9879 (0.0039)**	**0.9621 (0.0097)**	0.9404 (0.0147)	0.8905 (0.021)
half-network	0.9993 (0.0007)	0.9739 (0.02)	0.9604 (0.0163)	0.9383 (0.022)	0.8799 (0.0269)
selective distillation	0.9994 (0.0005)	0.9829 (0.0133)	0.962 (0.0156)	**0.9497 (0.0038)**	**0.9073 (0.0155)**

5.1.2 Confusion Matrices

In what follows (figures 5.1 to 5.5), mean and std of confusion matrices for each run is depicted. These figures generally help to identify which classes are being miss classified with each other and provide insight for further enhancing the proposed methods. For SplitMNIST, these matrices generally are diagonal which is near ideal.

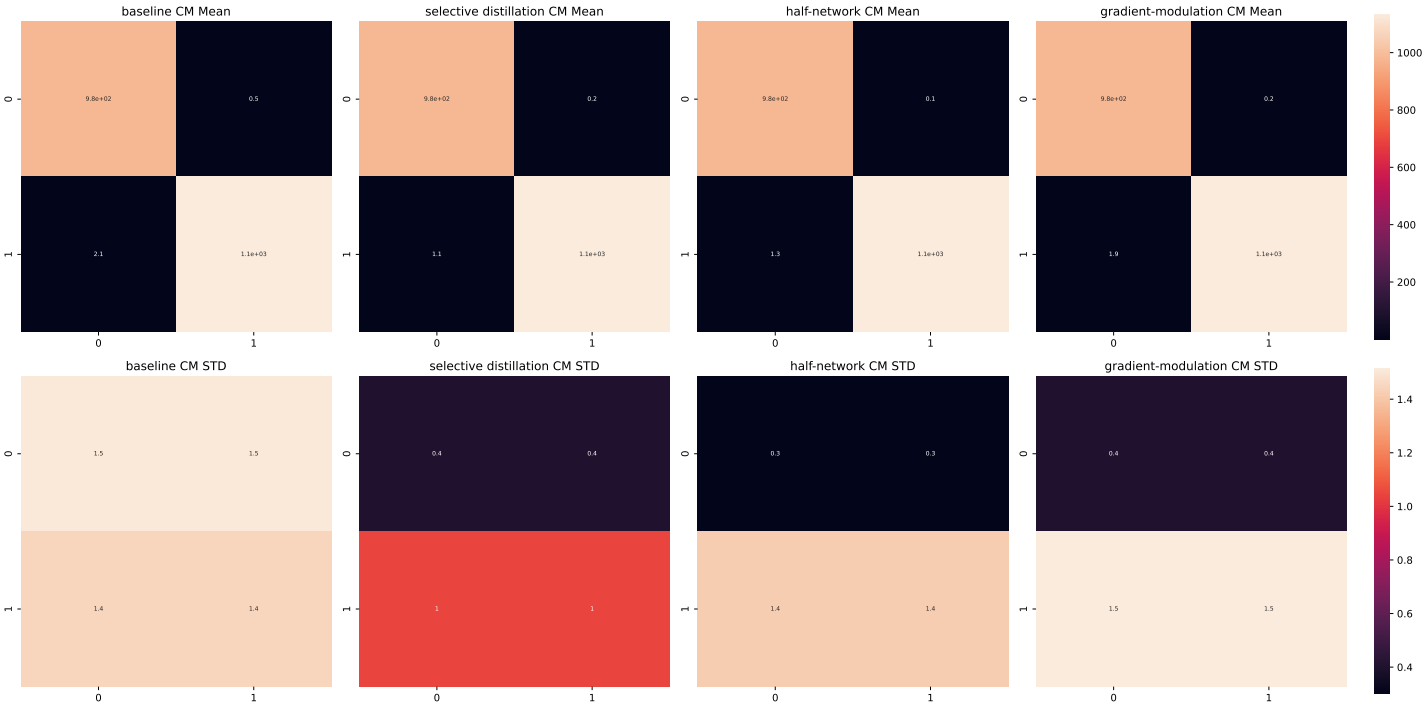

Figure 5.1: SplitMNIST: Task 1 Confusion Matrix

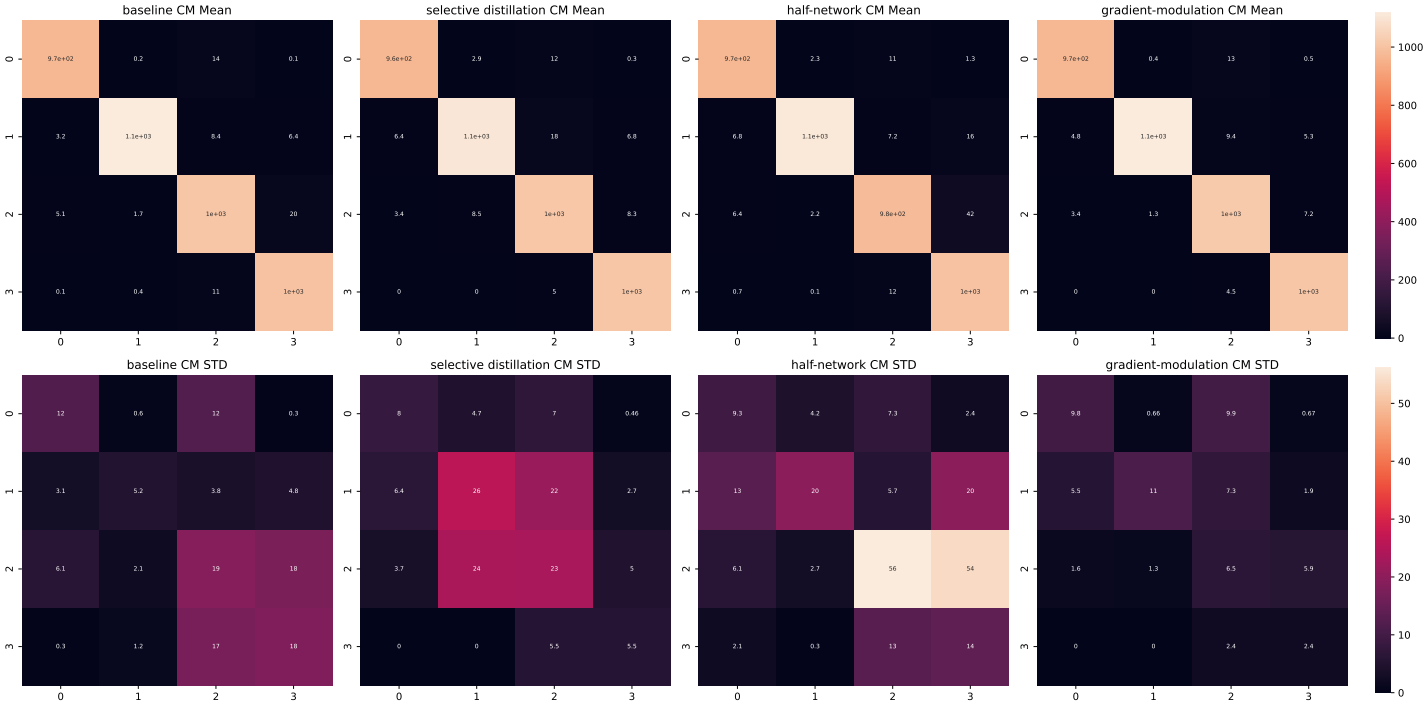

Figure 5.2: SplitMNIST: Task 2 Confusion Matrix

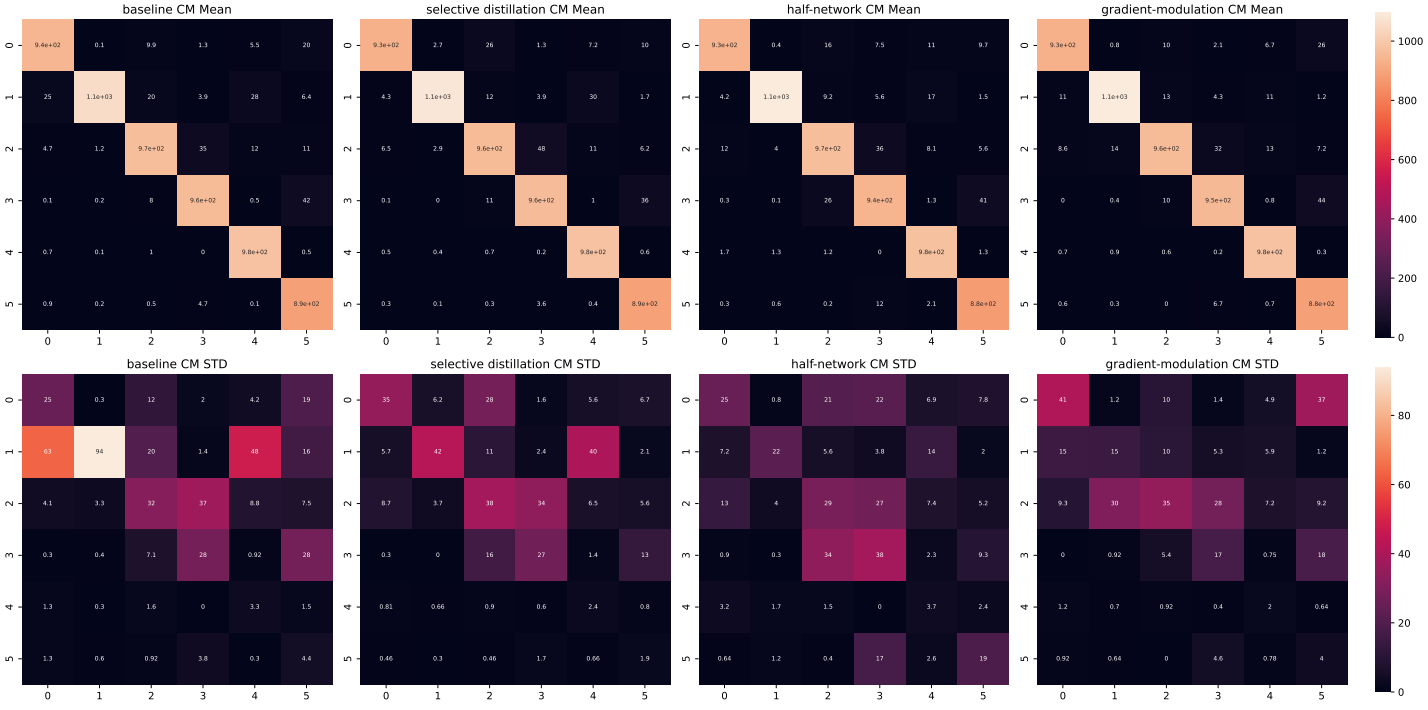

Figure 5.3: SplitMNIST: Task 3 Confusion Matrix

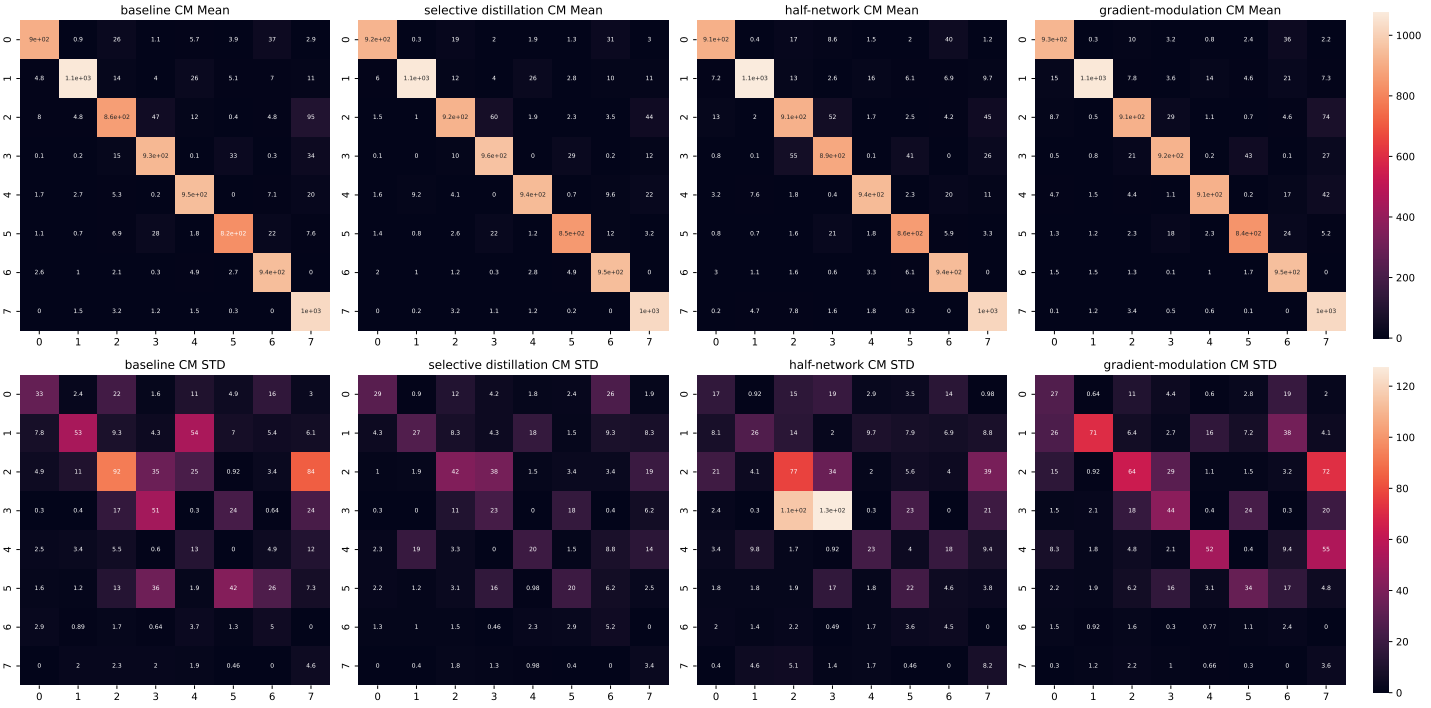

Figure 5.4: SplitMNIST: Task 4 Confusion Matrix

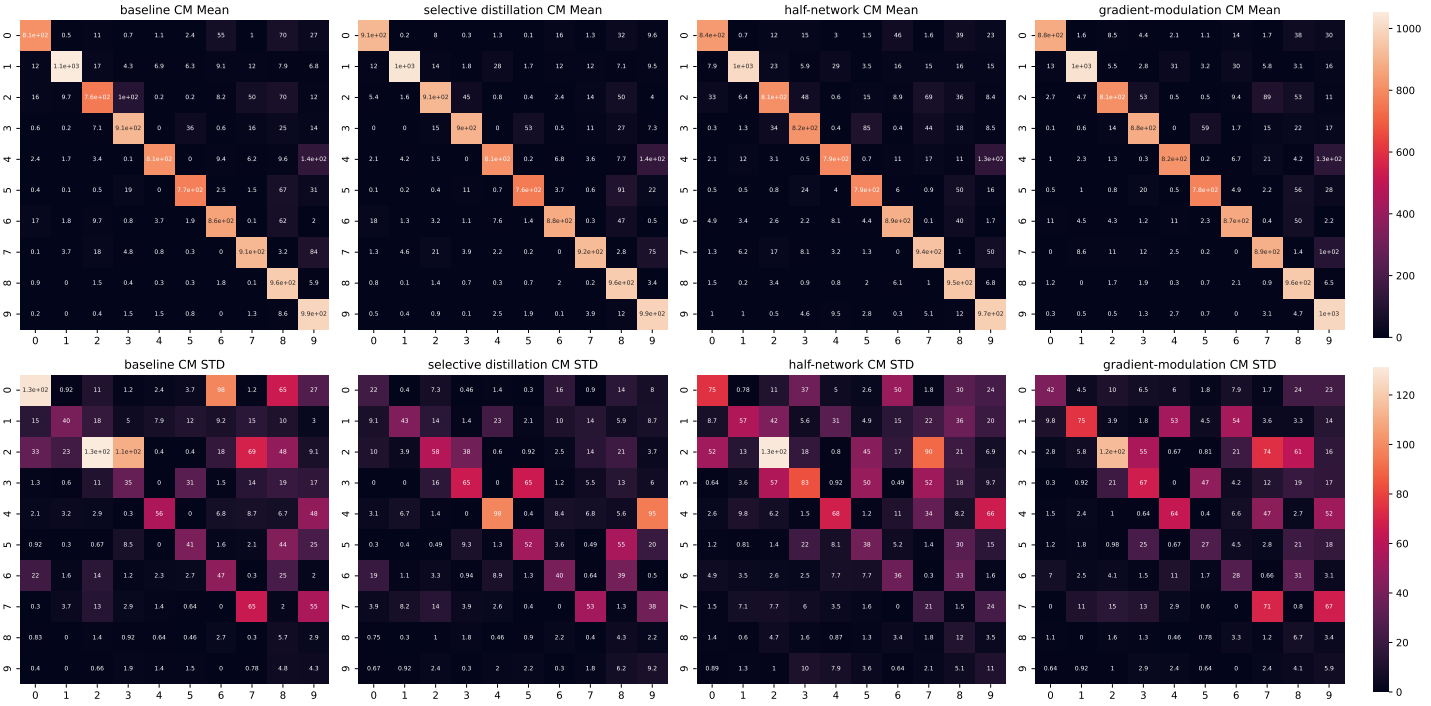

Figure 5.5: SplitMNIST: Task 5 Confusion Matrix

5.1.3 Metric Plots

In this section, for each class the metrics of precision, recall, and f1-score are plotted against the tasks (experiences). These figures show more specifically how each method performed in comparison to other methods. Note that the selective distillation method usually has higher or equal performance measures compared to other methods.

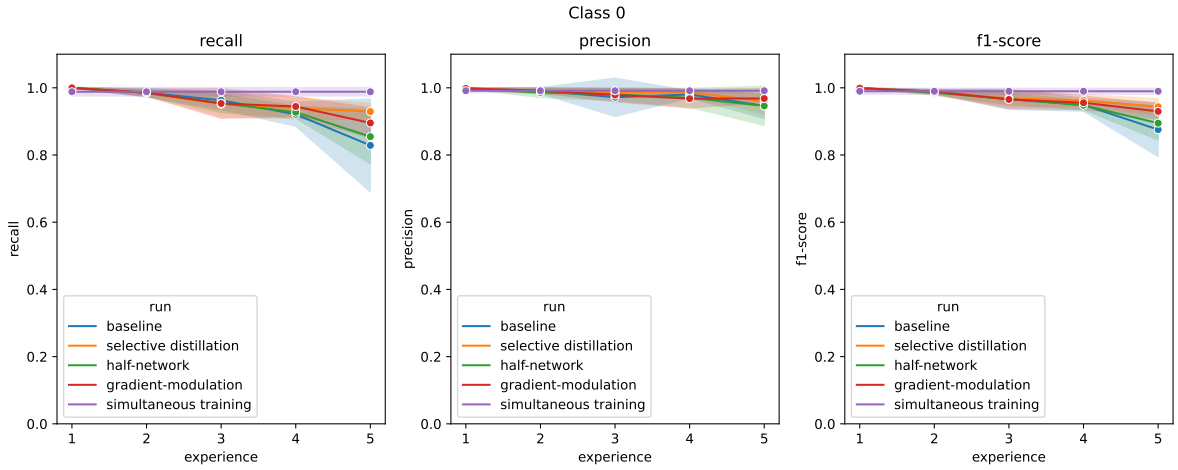

Figure 5.6: SplitMNIST: Precision, recall, and f1-score of class 0 across tasks for different implemented methods.

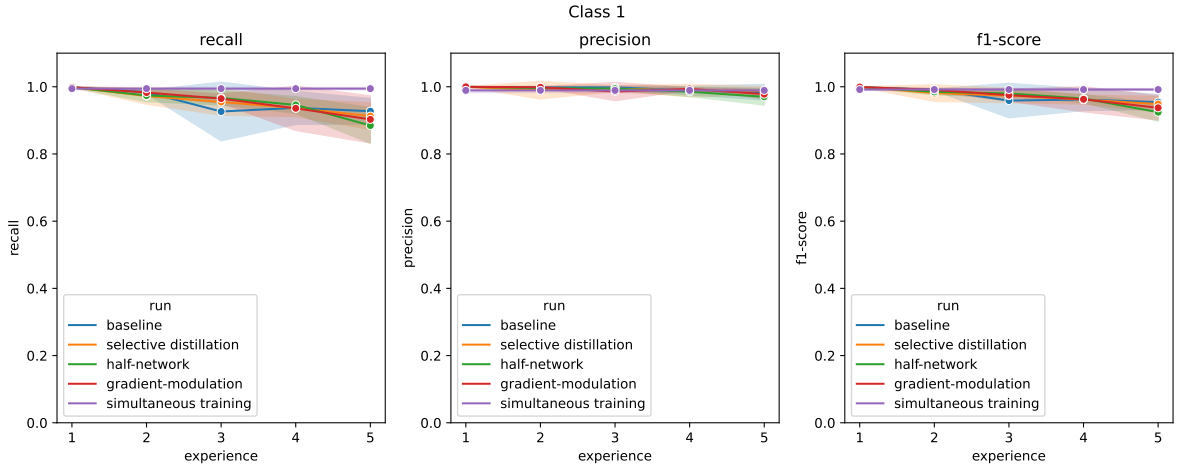

Figure 5.7: SplitMNIST: Precision, recall, and f1-score of class 1 across tasks for different implemented methods.

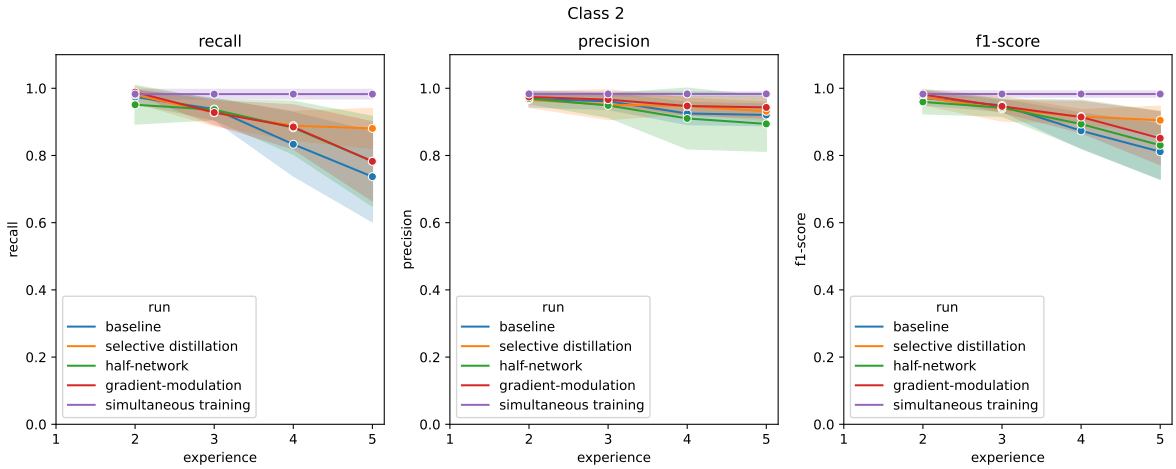

Figure 5.8: SplitMNIST: Precision, recall, and f1-score of class 2 across tasks for different implemented methods.

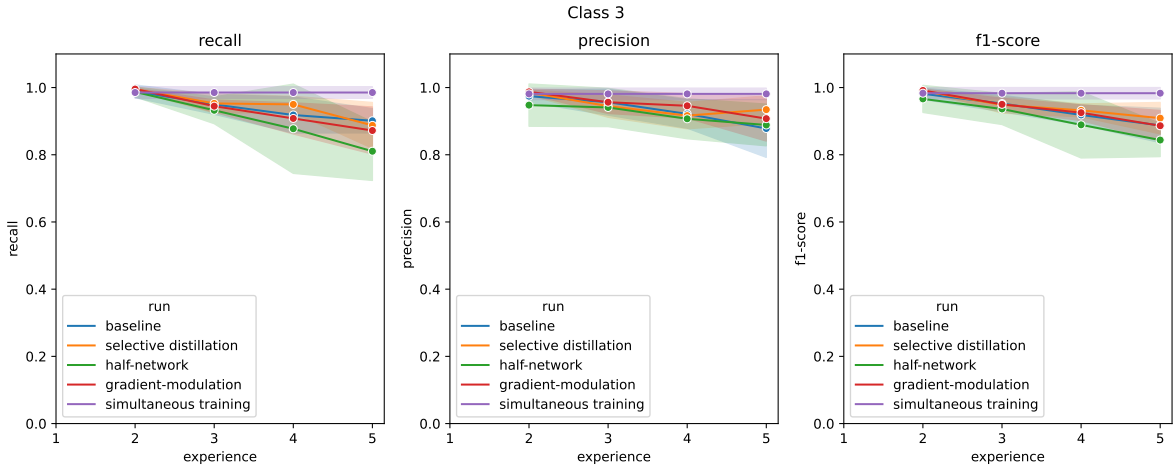

Figure 5.9: SplitMNIST: Precision, recall, and f1-score of class 3 across tasks for different implemented methods.

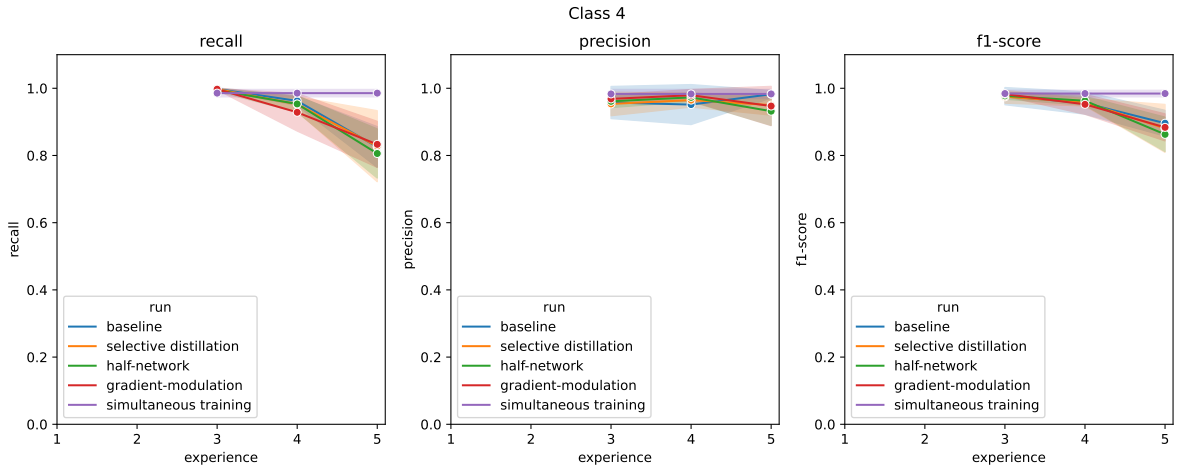

Figure 5.10: SplitMNIST: Precision, recall, and f1-score of class 4 across tasks for different implemented methods.

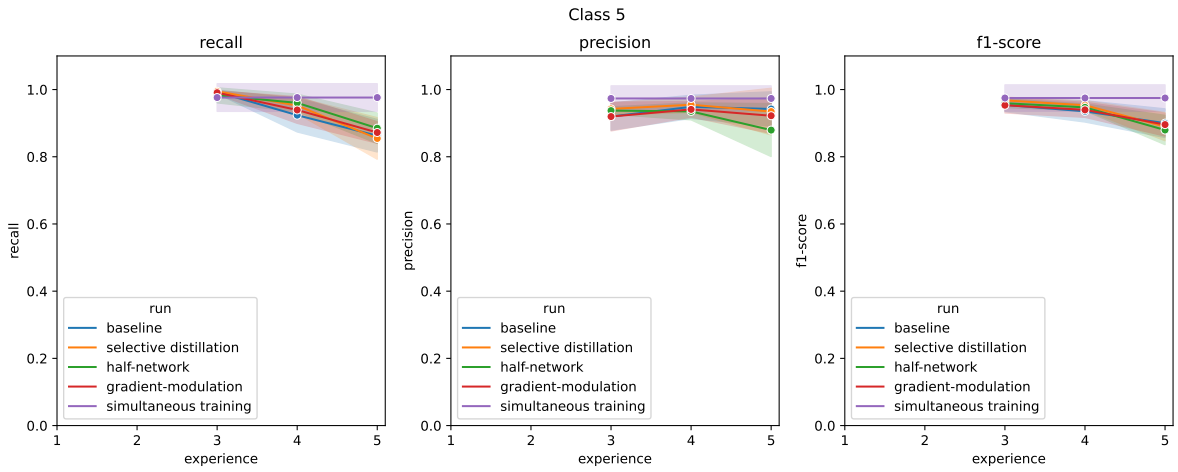

Figure 5.11: SplitMNIST: Precision, recall, and f1-score of class 5 across tasks for different implemented methods.

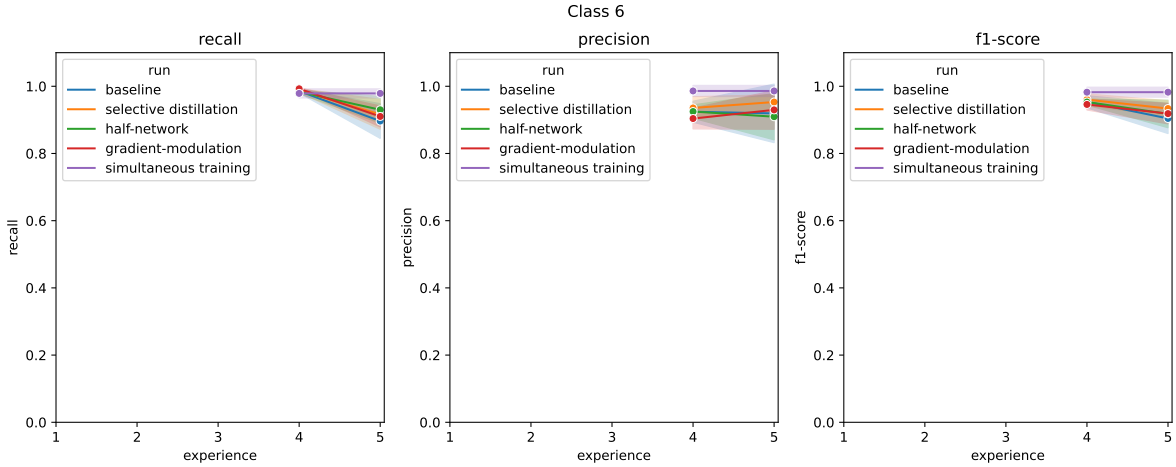

Figure 5.12: SplitMNIST: Precision, recall, and f1-score of class 6 across tasks for different implemented methods.

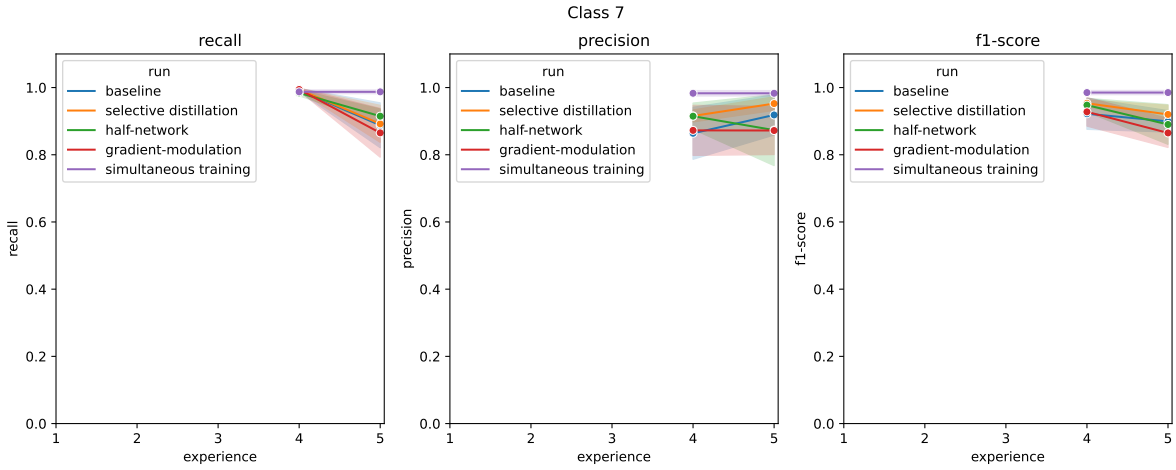

Figure 5.13: SplitMNIST: Precision, recall, and f1-score of class 7 across tasks for different implemented methods.

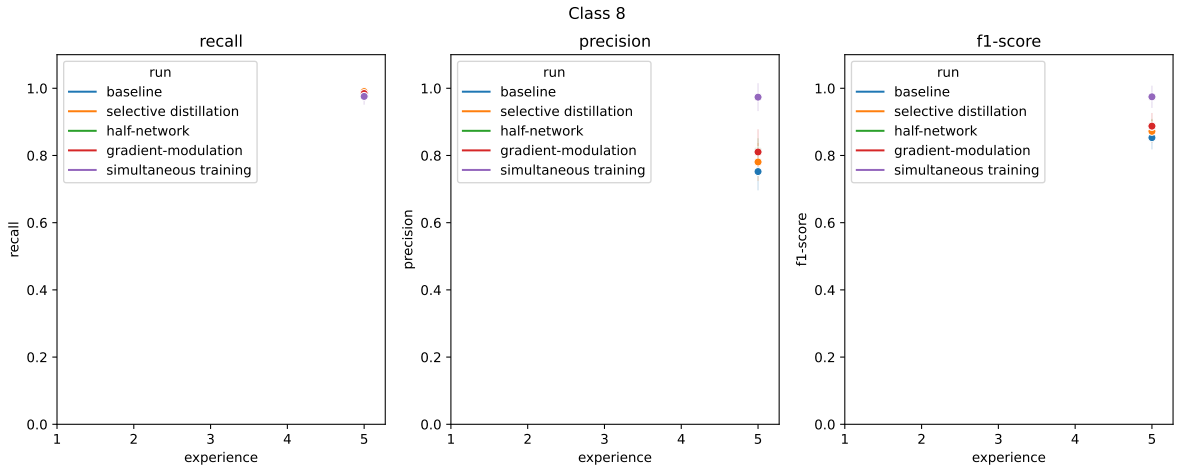

Figure 5.14: SplitMNIST: Precision, recall, and f1-score of class 8 across tasks for different implemented methods.

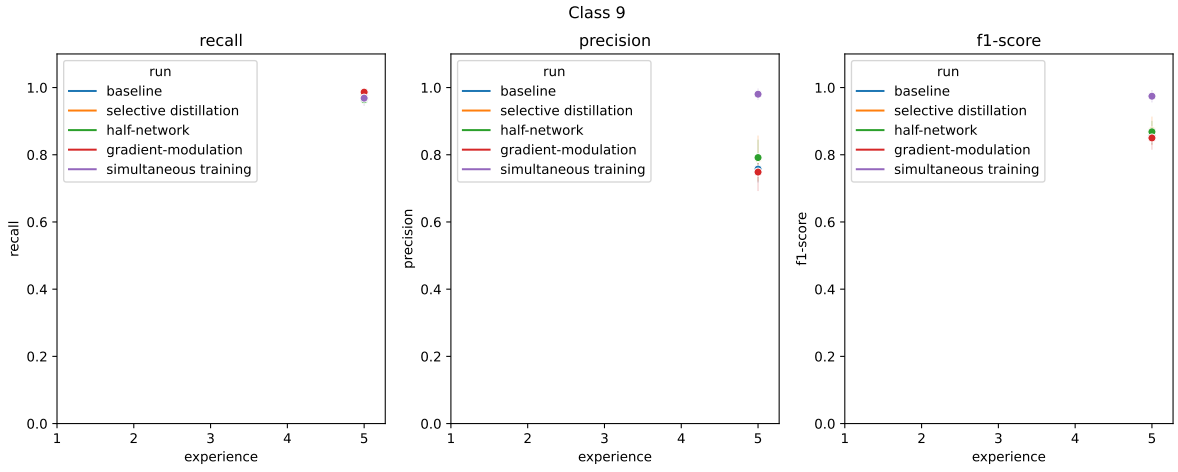

Figure 5.15: SplitMNIST: Precision, recall, and f1-score of class 9 across tasks for different implemented methods.

5.2 Experimental Results on SplitCIFAR10

The results for experiments on SplitCIFAR10 are presented in this section. SplitCIFAR10 is formed by splitting the dataset by classes into 5 equal groups, with the first group comprising the first two classes, the second group comprising the second two classes, and so on. A reduced ResNet-18 was used as the feature extractor, with a single hidden layer MLP as the projector. The network is then trained on each task sequentially, and performance metrics are evaluated as training on each task finishes.

5.2.1 Average Accuracy Evaluated After Each Task

Average accuracy over seen classes after training on the first task of the SplitCIFAR10 dataset is captured the first column of table 5.2 for each method. There is no difference among methods during the training of the first task, and the resulting recorded accuracy are similar, as expected. Note that the simultaneous training method achieves lower performance metrics for classes in the first task. The reason is that in simultanous training the network is trained on all classes to solve more difficult problem of discriminating between 10 classes, while other methods solve an easier problem of discriminating between two classes. Average accuracy after training on task 2 are provided in the second column of table 5.2. There is a considerable amount of forgetting observed across various methods. This decrease in performance, however, is not equal. While of the proposed methods outperform the baseline, the gradient modulation suffers less from forgetting and shows the highest accuracy.Moving on to performance metrics measured after task 3, more forgetting is observed. The third column of table 5.2 shows the performance measures after task 3. Gradient modulation and half-network methods outperform the baseline, while selective distillation achieves similar performance. The gradient modulation method achieves highest average accuracy compared to other methods. Column 4 of table 5.2 shows average accuracy of seen classes after training on task 4. While all methods suffer more from forgetting, the gradient modulation and half-network methods remain to be outperforming the baseline, while selective-distillation has a lower average accuracy than the baseline. Similar to previous tasks, gradient modulation method achieves highest average accuracy compared to other methods. Column 5 of table 5.2 shows the measured average accuracy of different methods after training on the fifth and final task. None of the methods manage to outperform the baseline in terms of the average accuracy. The gradient modulation and half-network methods, however, achieve a very close average accuracy while selective distillation shows the lowest performance among the methods.

Overall, the gradient modulation method shows the best performance over the course of training and competitive accuracy after training in the last task. Compared to the half-network method, gradient modulation shows better performance after all tasks, indicating that the original hypothesis where gradient-modulation could be regularizing too many parameters can be rejected.

Table 5.2: SplitCIFAR10: Task mean (std) accuracy on the test set

method	Task 1 Avg. Accuracy	Task 2 Avg. Accuracy	Task 3 Avg. Accuracy	Task 4 Avg. Accuracy	Task 5 Avg. Accuracy
simultaneous training	0.9174 (0.0103)	0.7964 (0.0149)	0.7867 (0.0105)	0.8194 (0.0074)	0.8394 (0.0059)
baseline	0.9802 (0.0015)	0.773 (0.0116)	0.5657 (0.0176)	0.4757 (0.0096)	**0.4735 (0.0133)**
gradient-modulation	0.9814 (0.0029)	**0.7953 (0.008)**	**0.591 (0.0096)**	**0.4904 (0.01)**	0.4639 (0.0127)
half-network	0.9804 (0.002)	0.7869 (0.0075)	0.5865 (0.011)	0.4795 (0.01)	0.4638 (0.0124)
selective distillation	0.9811 (0.002)	0.78 (0.0113)	0.5654 (0.0119)	0.4503 (0.0102)	0.4335 (0.012)

5.2.2 Confusion Matrices

The confusion matrices for the classes seen up until each task are shown in what follows (figures 5.16-5.20). As can be seen, classes are usually misclassified for classes that were present in the most recent task, resulting in higher values in rightmost columns compared to the leftmost ones. This forgetting is observed across all methods, but it is difficult to see which classes observe more forgetting for each method. The plots in the next section show precision, recall, and f1-score of each class over the course of training and allow the comparison of the amount of forgetting in each method for each class.

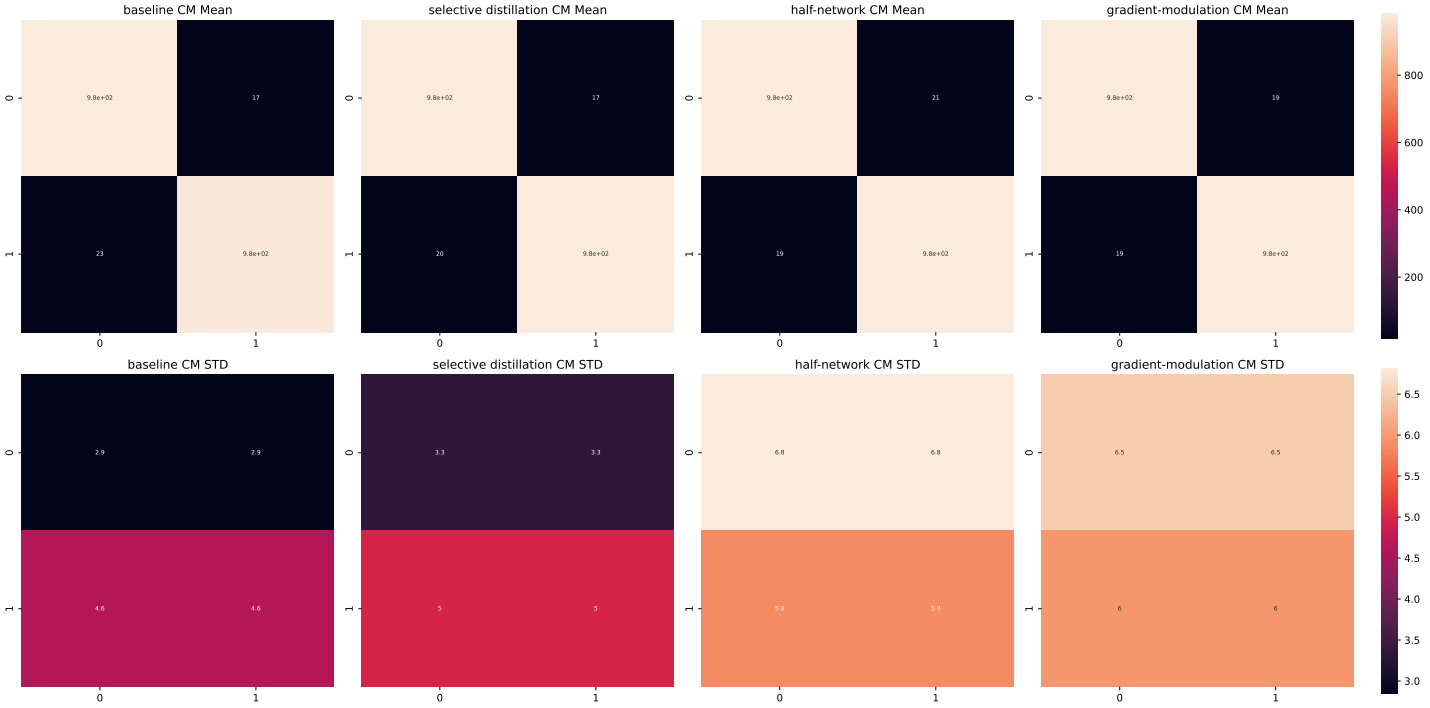

Figure 5.16: SplitCIFAR10: Task 1 Confusion Matrix

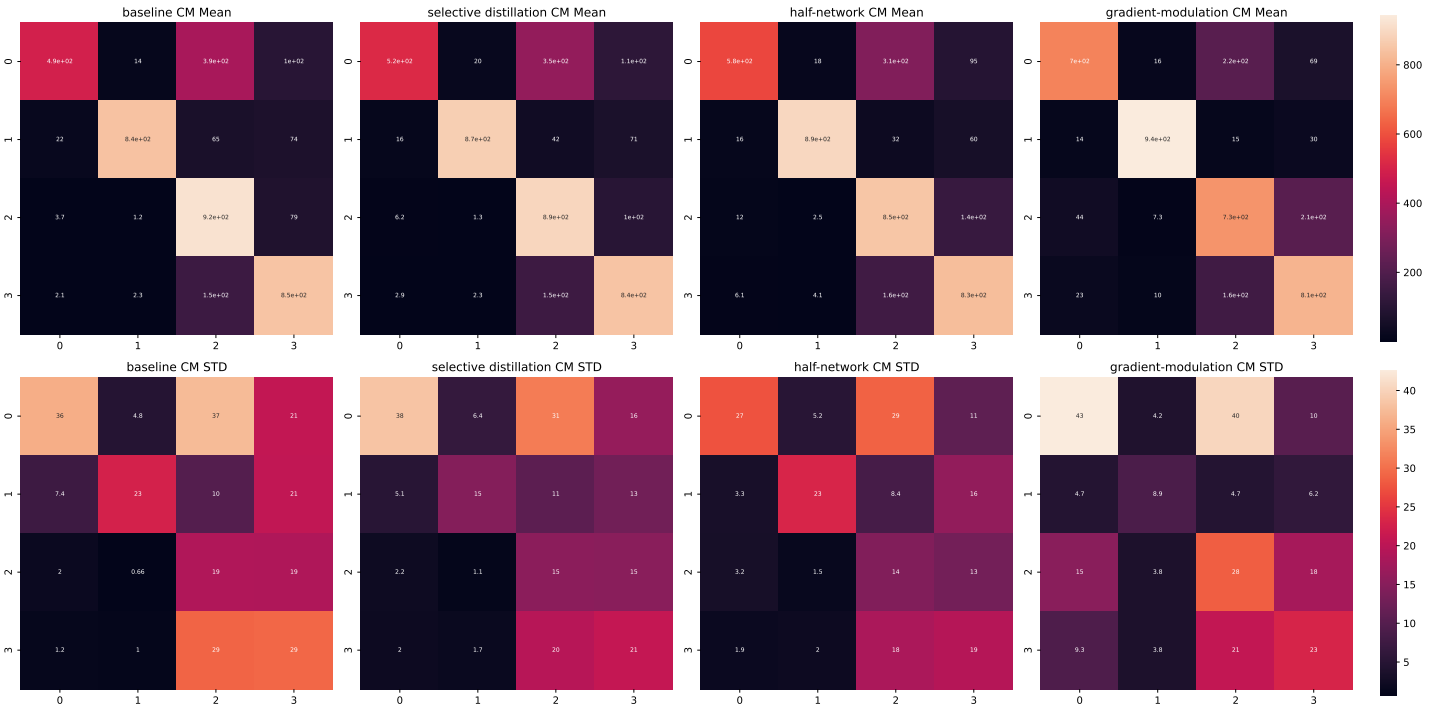

Figure 5.17: SplitCIFAR10: Task 2 Confusion Matrix

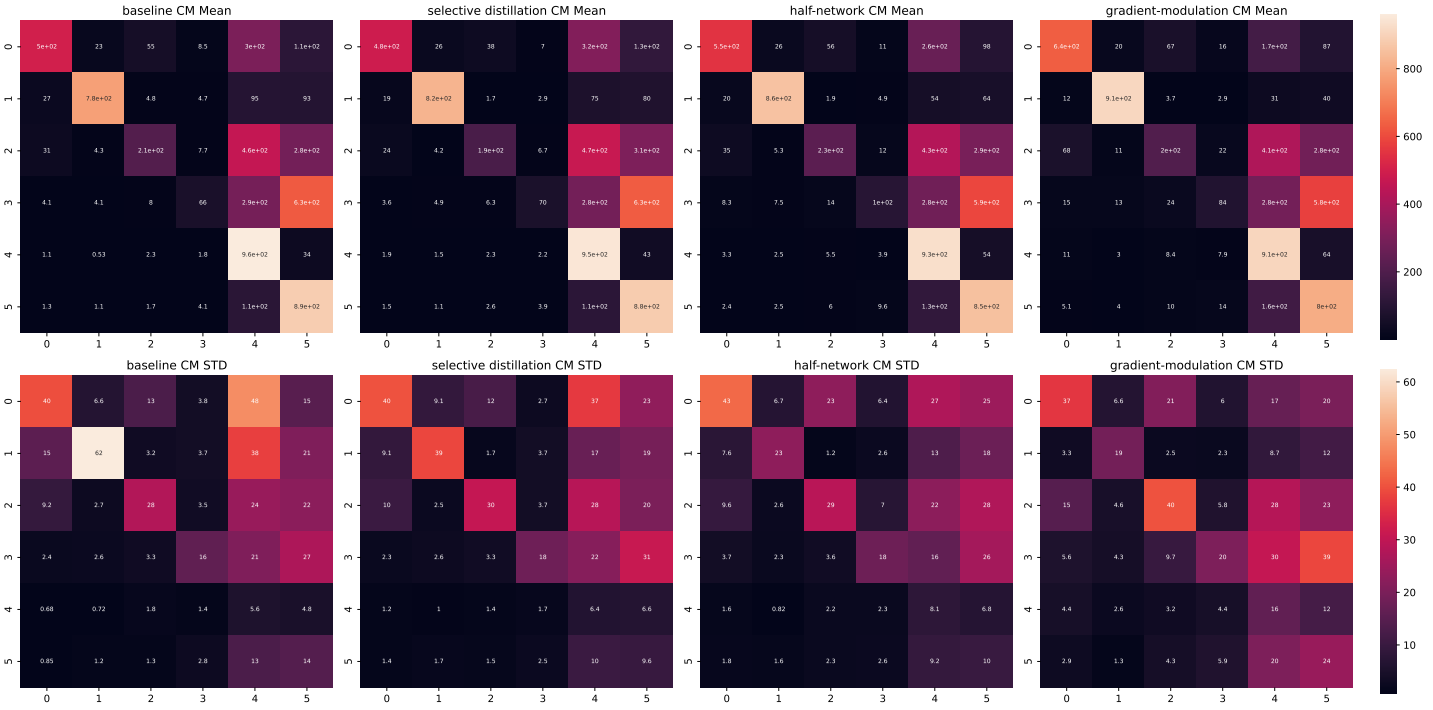

Figure 5.18: SplitCIFAR10: Task 3 Confusion Matrix

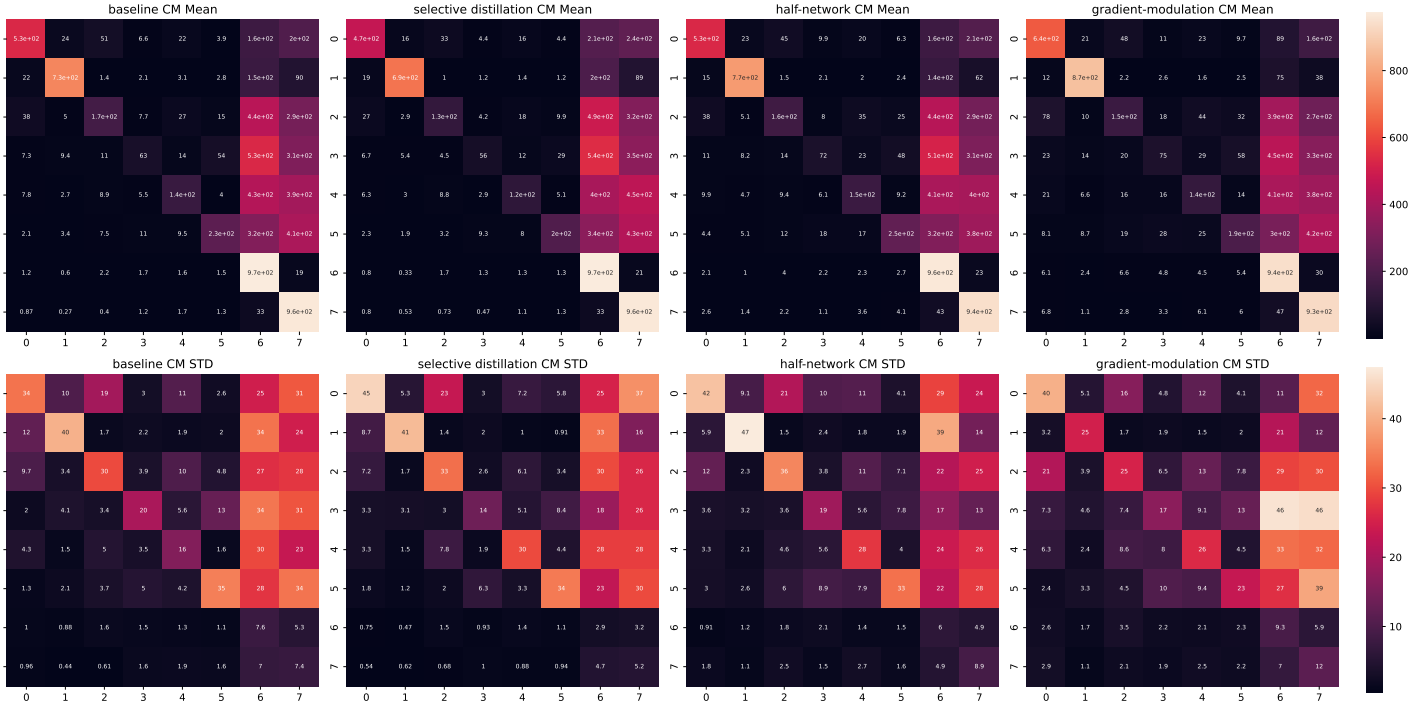

Figure 5.19: SplitCIFAR10: Task 4 Confusion Matrix

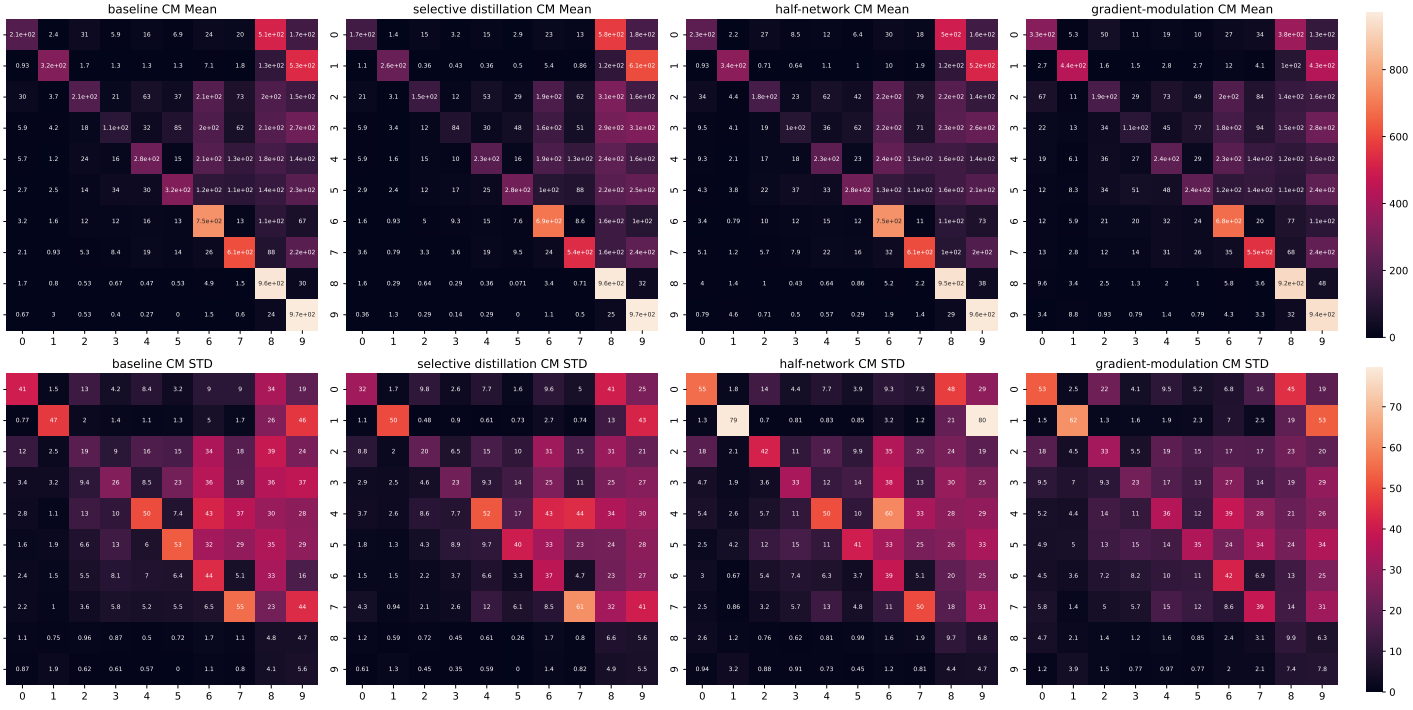

Figure 5.20: SplitCIFAR10: Task 5 Confusion Matrix

5.2.3 Metric Plots

For each class in CIFAR10, each metric of precision, recall, and f1-score was evaluated across tasks. The following plots (figures 5.21 - 5.30) show how each method performs compared to the baseline, simultaneous training, and other methods, grouped by each class. For the first two classes, the gradient modulation method clearly outperforms other approaches in terms of the f1-score. For the remaining classes, however, the baseline method seems to be generally superior in terms of the f1-score, while the gradient-modulation method follows closely. Moreover, the forgetting measure for any metric of choice varies for different classes. While the decrease in performance is more gradual for the first two classes (present in the first task), there is more sudden and abrupt loss of performance measure for the remaining classes. It is also good to note some recovery of performance takes place after training on the fifth task. The f1-score of classes 2 to 9 generally increases from task task 4 to task 5, while the decrease of f1-score continues in the final task for the first two classes. This recovery can be attributed to the use of supervised contrastive loss and the memory module. While most of the samples given to the network are from the current task, the rehearsal of previous task data allows the supervised contrastive loss to separate previous class representations from the current ones, even when the number of training samples is very limited.

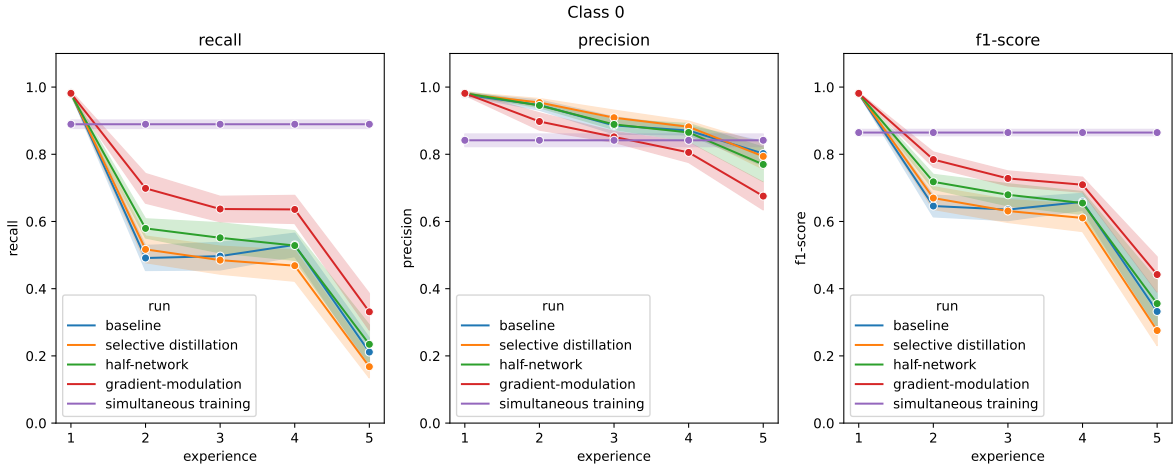

Figure 5.21: SplitCIFAR10: Precision, recall, and f1-score of class 0 across tasks for different implemented methods.

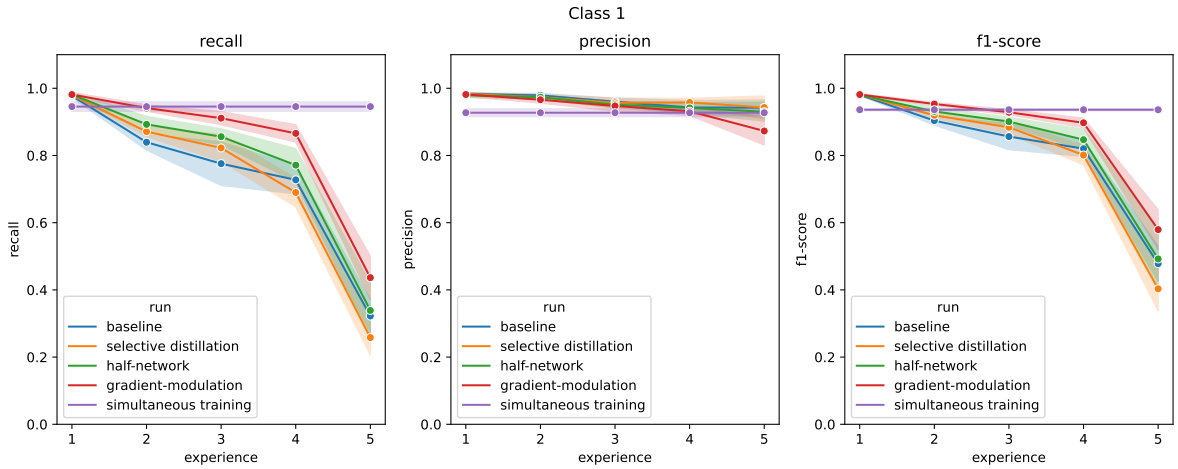

Figure 5.22: SplitCIFAR10: Precision, recall, and f1-score of class 1 across tasks for different implemented methods.

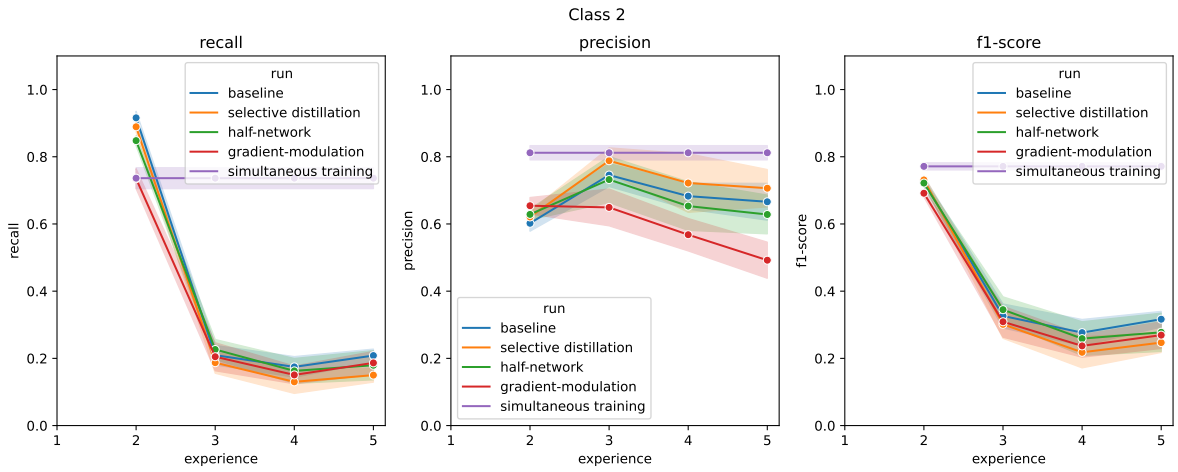

Figure 5.23: SplitCIFAR10: Precision, recall, and f1-score of class 2 across tasks for different implemented methods.

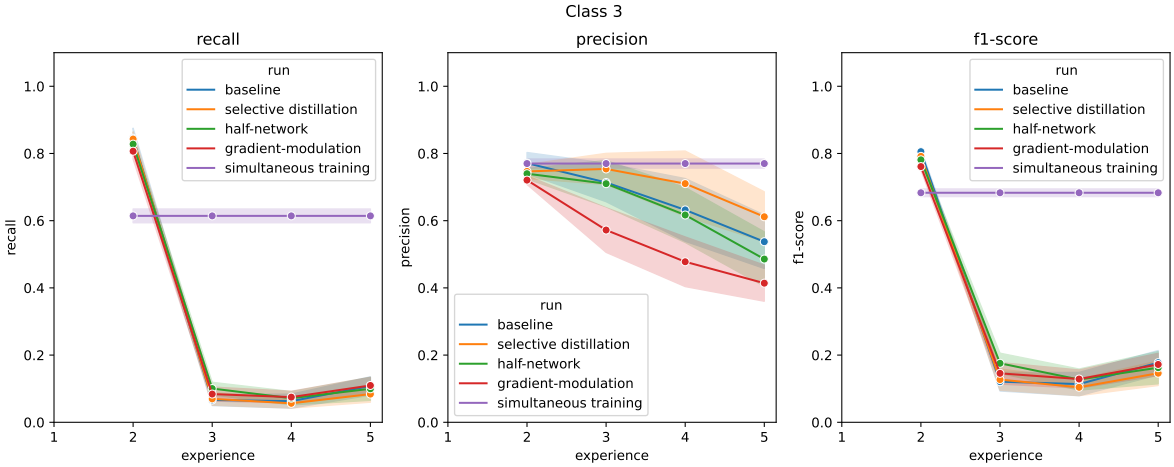

Figure 5.24: SplitCIFAR10: Precision, recall, and f1-score of class 3 across tasks for different implemented methods.

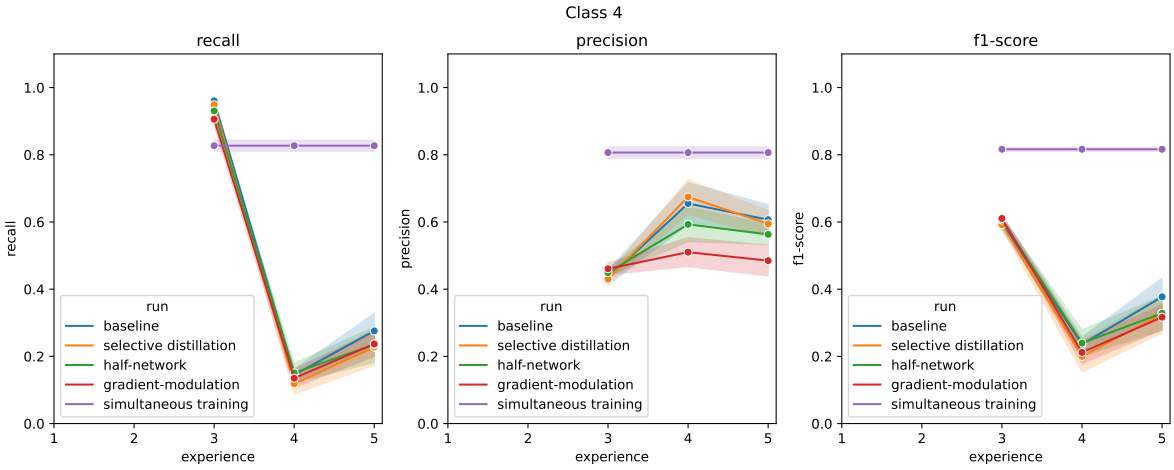

Figure 5.25: SplitCIFAR10: Precision, recall, and f1-score of class 4 across tasks for different implemented methods.

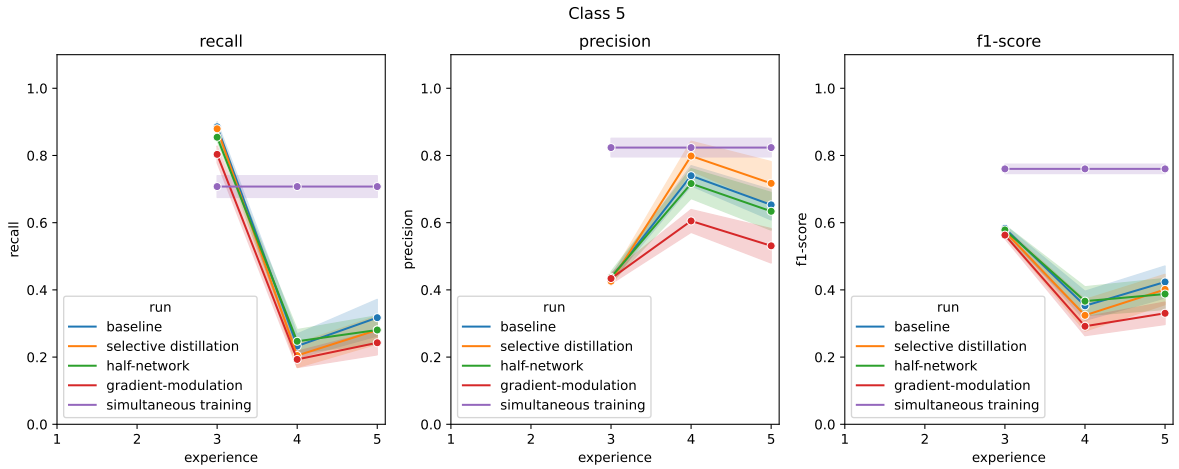

Figure 5.26: SplitCIFAR10: Precision, recall, and f1-score of class 5 across tasks for different implemented methods.

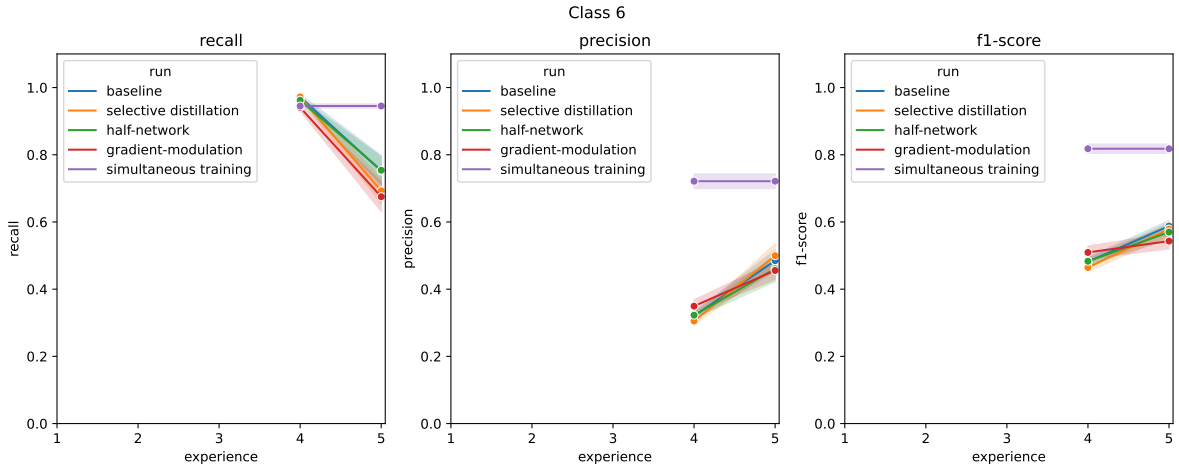

Figure 5.27: SplitCIFAR10: Precision, recall, and f1-score of class 6 across tasks for different implemented methods.

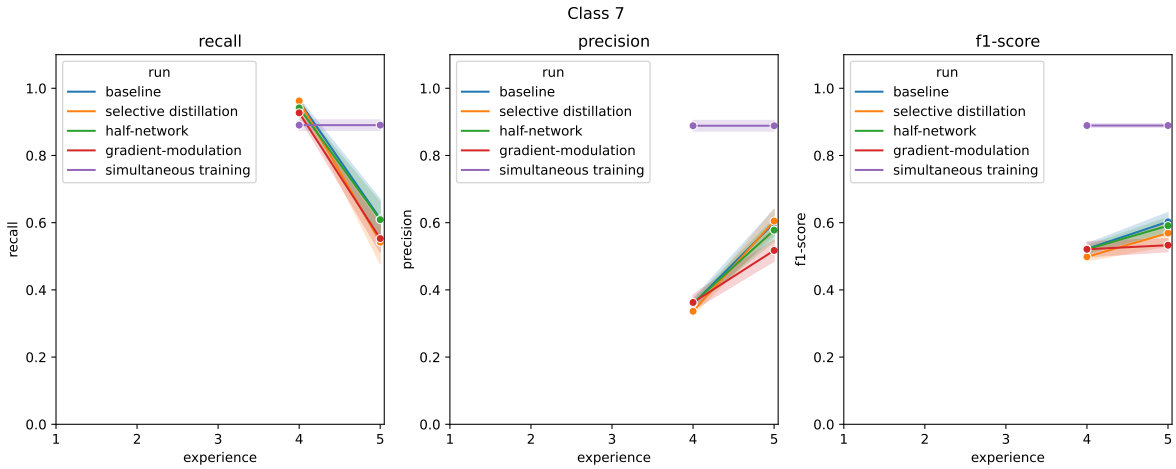

Figure 5.28: SplitCIFAR10: Precision, recall, and f1-score of class 7 across tasks for different implemented methods.

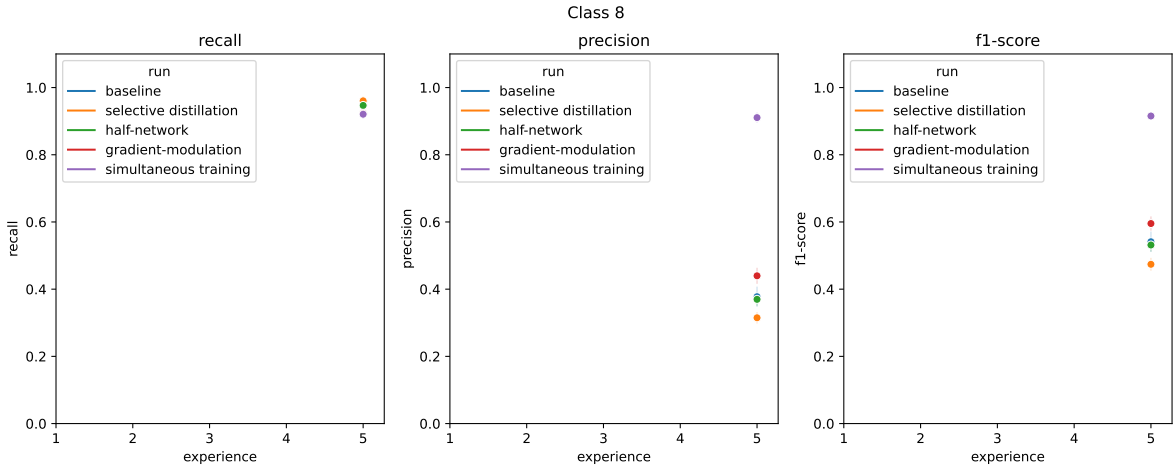

Figure 5.29: SplitCIFAR10: Precision, recall, and f1-score of class 8 across tasks for different implemented methods.

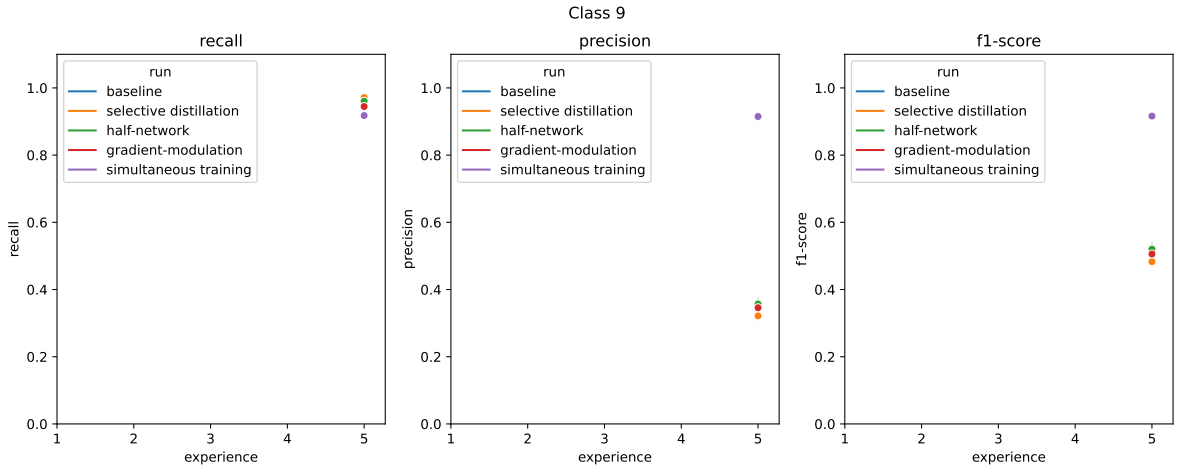

Figure 5.30: SplitCIFAR10: Precision, recall, and f1-score of class 9 across tasks for different implemented methods.

5.3 Experimental Results on SplitTinyImageNet

The TinyImageNet dataset was splitted into 5 groups, each including 40 classes (out of 200). The first task will be discriminating between the first 40 classes, the second task will be discriminating the second 40 classes and so on. Similar to experiments on the SplitCIFAR10 dataset, the feature extractor was a reduced ResNet-18, while the projector network was a single hidden layer MLP. Each performance metric of precision, recall, and f1-score was computed after training of each task finished. Each performance metric was averaged across seen classes, and provided in the following section. The main metric for comparing methods, however, is average accuracy. For the results in this section, each method was trained in 5 independent runs, compared to SplitCIFAR10 and SplitMNIST's 10.

5.3.1 Average Accuracy Evaluated After Each Task

The average accuracy measured after training on the first task are shown in the first column of table 5.3. None of the methods of gradient modulation, half-network, or selective distillation have taken effect in the first task, and results should be similar (note the high std of gradient modulation method). Training using a larger batch size on SplitTinyImageNet generally led to high variance for results. Also note the relatively low accuracy of the simultaneous training on the first task, showing the difficulty of this dataset. Average accuracy on the first two tasks after training of the second task are recorded in the second column of table 5.3. While the gradient modulation method resulted in lowest average accuracy among different methods and the baseline after task 1, it achieved the highest average accuracy after task 2, showing how promising the use of gradient modulation can be in order to prevent forgetting and preserve knowledge. The methods of half-network and selective-distillation, however, did not surpass the baseline's average accuracy. Overall, significant forgetting is observed for all methods when compared to the first task's accuracy. The average accuracy evaluated after task 3 are provided in the third column of table 5.3. None of the proposed methods manage to outperform the baseline in terms of average accuracy after this task. The forgetting trend continues and around 10 percent loss of accuracy is seen across the methods. Among the proposed methods, gradient modulation shows competitive performance compared to the baseline while significantly outperforming half-network and selective distillation methods. After training on Task 4, average accuracy was recorded and provided in the fourth column of table 5.3. Similar to the third task, none of the proposed methods manage to outperform the baseline in terms of average accuracy. Gradient modulation method follows the baseline closely while half-network and selective-distillation remain to be show lower accuracy. Significant forgetting is observed in general and the recorded average accuracy are very low. Finally, average accuracy after training on the fifth task are provided in last column of table 5.3. The performance measure after this task is very informative as all 200 classes in the dataset have now been observed and trained on. The gradient modulation method outperforms the baseline and other methods while achieving state of the art average accuracy (see [13]). While the results on the SplitTinyImageNet were promising, the variation was large, and there is still a large gap between the proposed methods and the simultaneous training benchmark. Interestingly, while after task 5 there are more classes to predict and the expectation for the average accuracy is to continue the trend and decrease, the network's average accuracy sees an increase for all methods. Since this effect is

seen in all of the methods, it probably results from the use of supervised contrastive loss and a memory module to rehearse past task samples. Moreover, there is a chance the fourth task included difficult classes to learn from, and the network needed more time and training epochs to learn them.

Comparison with state-of-the-art method in continual learning will be provided later in this chapter, after the ablation studies are described.

Table 5.3: SplitTinyImageNet: Task mean (std) accuracy on the test set

method	Task 1 Avg. Accuracy	Task 2 Avg. Accuracy	Task 3 Avg. Accuracy	Task 4 Avg. Accuracy	Task 5 Avg. Accuracy
simultaneous training	0.3832 (0.038)	0.4114 (0.0429)	0.3944 (0.0395)	0.3852 (0.0398)	0.3802 (0.0377)
baseline	0.6234 (0.0051)	0.369 (0.0036)	**0.2691 (0.0066)**	**0.0955 (0.0031)**	0.1423 (0.0081)
gradient-modulation	0.5865 (0.086)	**0.3697 (0.0237)**	0.2655 (0.019)	0.0922 (0.013)	**0.1427 (0.0082)**
half-network	0.6171 (0.0135)	0.2941 (0.0045)	0.2089 (0.0064)	0.0833 (0.0074)	0.1163 (0.0051)
selective distillation	0.6101 (0.0377)	0.3392 (0.0087)	0.2385 (0.0164)	0.086 (0.0095)	0.1356 (0.0027)

5.3.2 Confusion Matrices

The confusion matrices for the classes seen up until each task are shown in what follows (figures 5.31 - 5.35). Similar to experiments on SplitCIFAR10, previous task classes get misclassified mostly with classes present in the most recent task. Notably, the confusion matrices are diagonal near the last 40 classes (last task the model was trained on) while being scrambled and less diagonal for classes of the past tasks. This shows that forgetting is not specific to *some* of the classes in previous tasks, but affects all of them significantly.

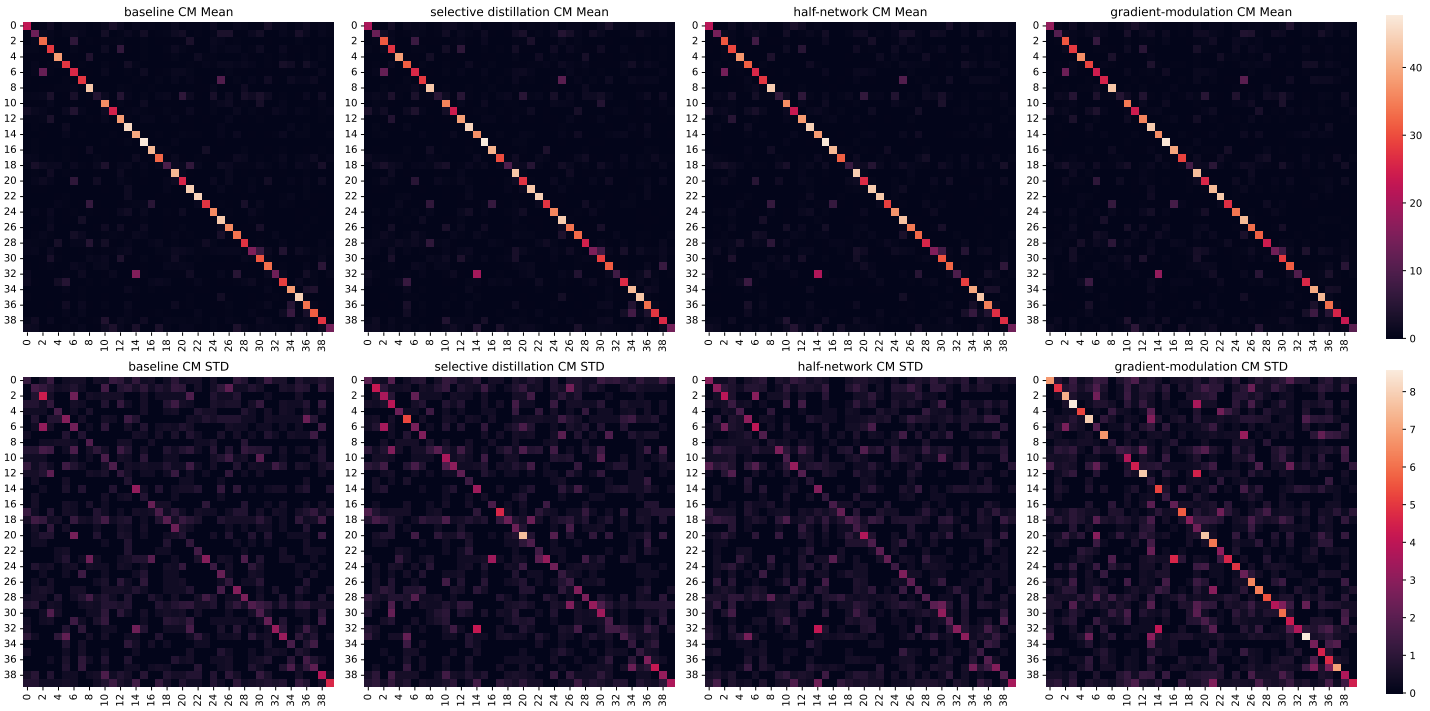

Figure 5.31: SplitTinyImageNet: Task 1 Confusion Matrix

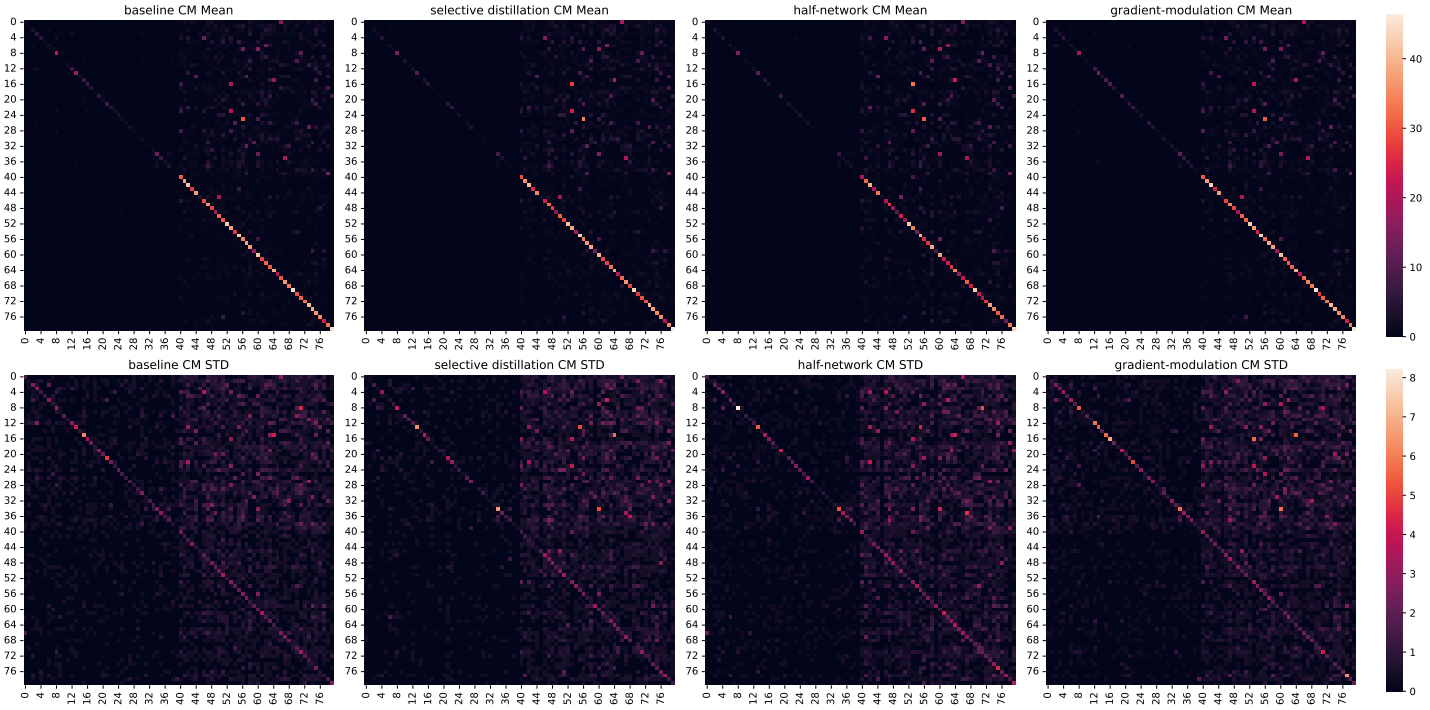

Figure 5.32: SplitTinyImageNet: Task 2 Confusion Matrix

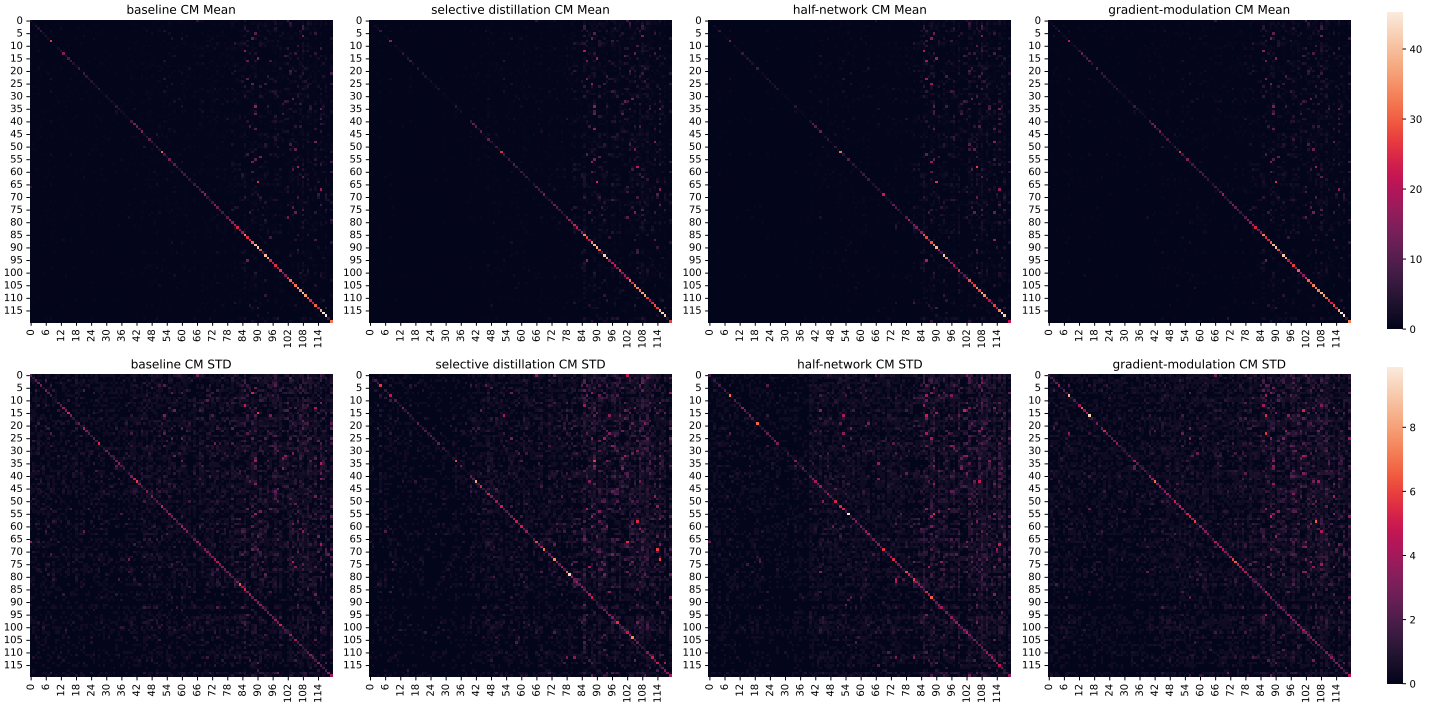

Figure 5.33: SplitTinyImageNet: Task 3 Confusion Matrix

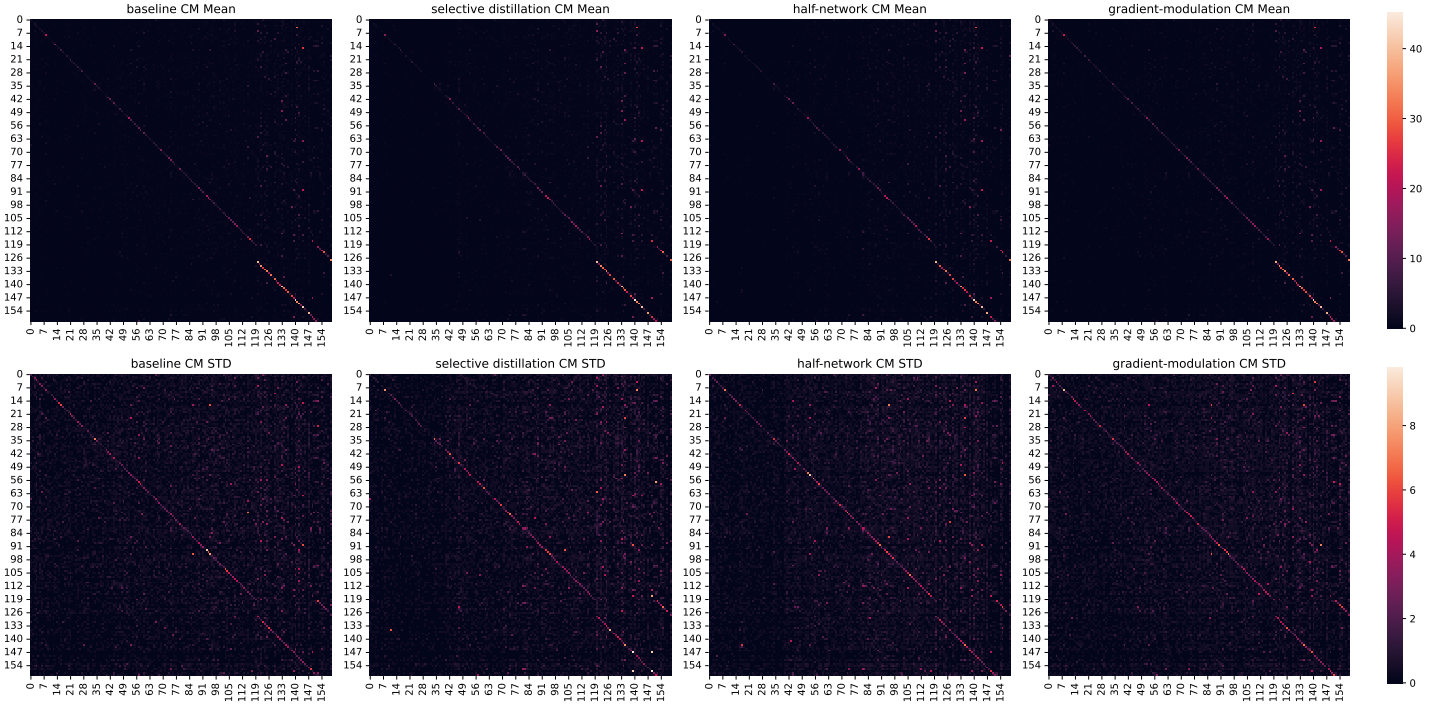

Figure 5.34: SplitTinyImageNet: Task 4 Confusion Matrix

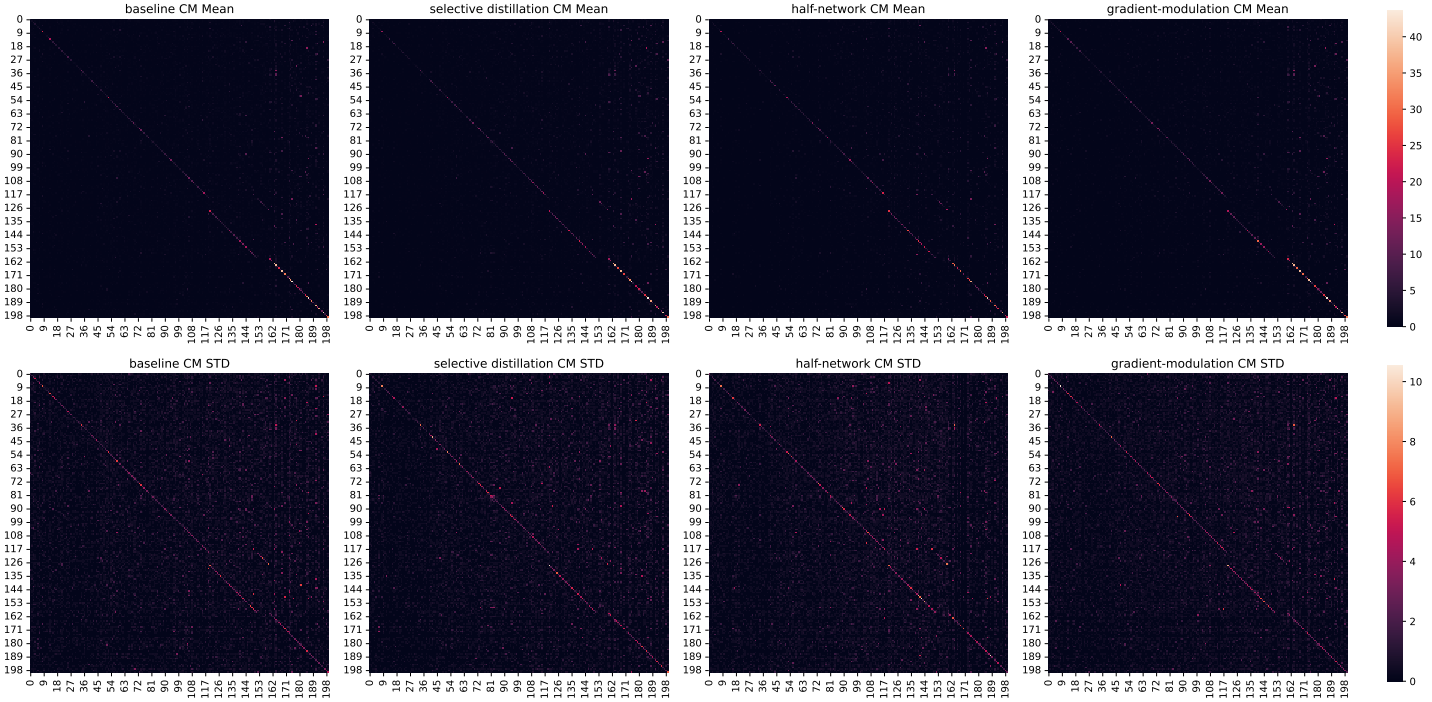

Figure 5.35: SplitTinyImageNet: Task 5 Confusion Matrix

Furthermore, the confusion matrix resulting from training on the entire dataset (all classes at once, simultaneously) is shown in figure 5.36. As can be seen, when training on all classes the confusion matrix is near diagonal, in contrast to continually learning tasks that results in earlier classes being misclassified as classes in the last seen task. This confusion matrix, however, is not perfectly diagonal because of the difficulty of this classification of this dataset.

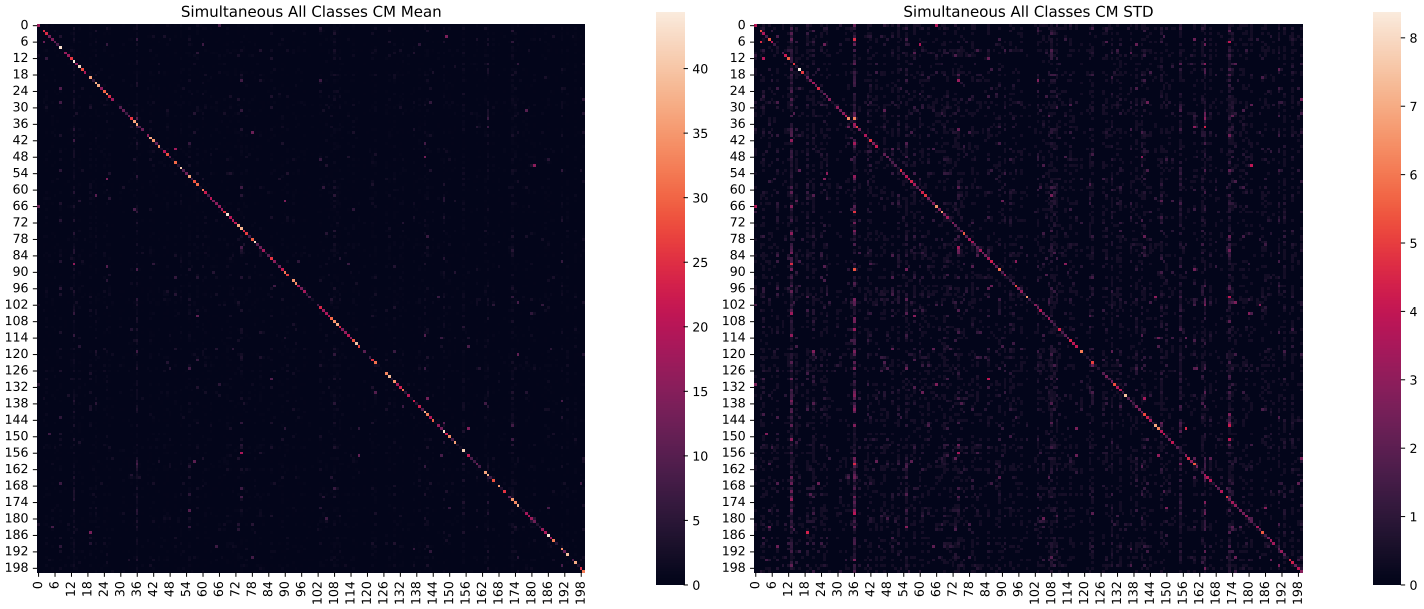

Figure 5.36: SplitTinyImageNet: Simultaneous Training Confusion Matrix

5.3.3 Metric Plots

Each of the following figures (5.37 - 5.41) plots each of performance metrics computed on a task across tasks that model was trained after. Interestingly, the gradient modulation method generally (and marginally) performs better than the baseline in multiple metrics, and in multiple tasks. One can identify the reason why overall average accuracy reported in 5.3 increased from task 4 to task 5. Apparently, the network found it difficult to learn classes in the fourth task while showing better performance on the classes in task 5, and there is no performance recovery similar to those seen in SplitCIFAR10 results. Compared to performance on SplitCIFAR10, however, the decrease in performance metrics is more gradual.

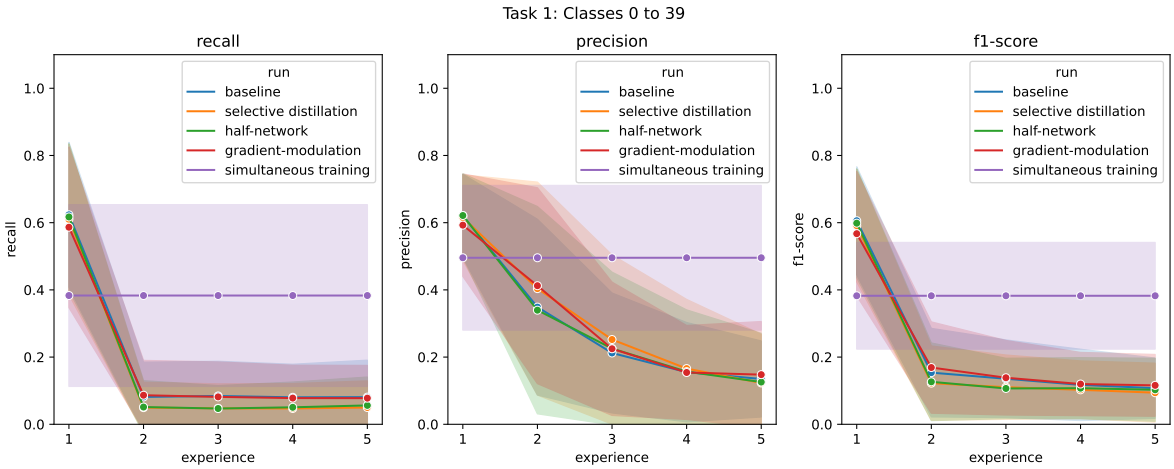

Figure 5.37: SplitTinyImageNet: Precision, recall, and f1-score of task 1 across tasks for different implemented methods.

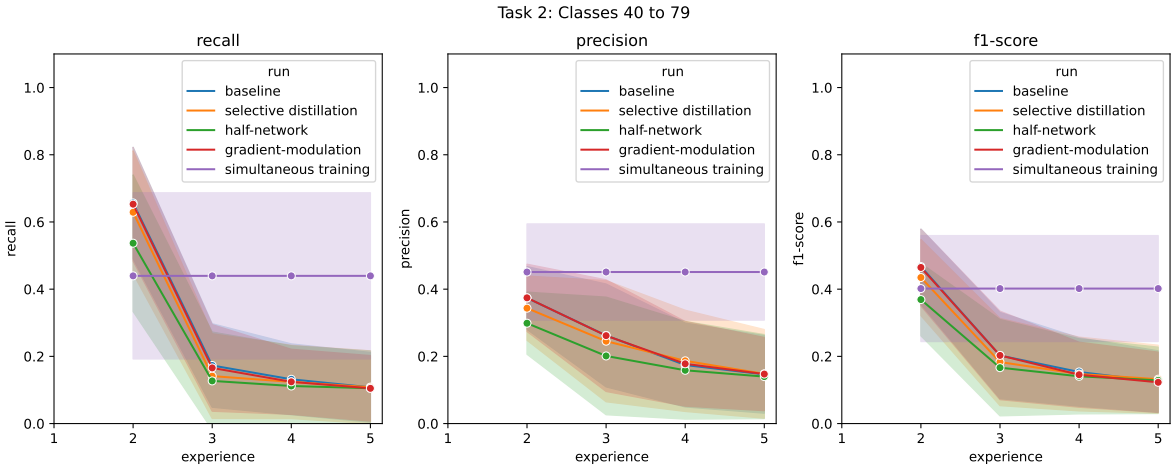

Figure 5.38: SplitTinyImageNet: Precision, recall, and f1-score of task 2 across tasks for different implemented methods.

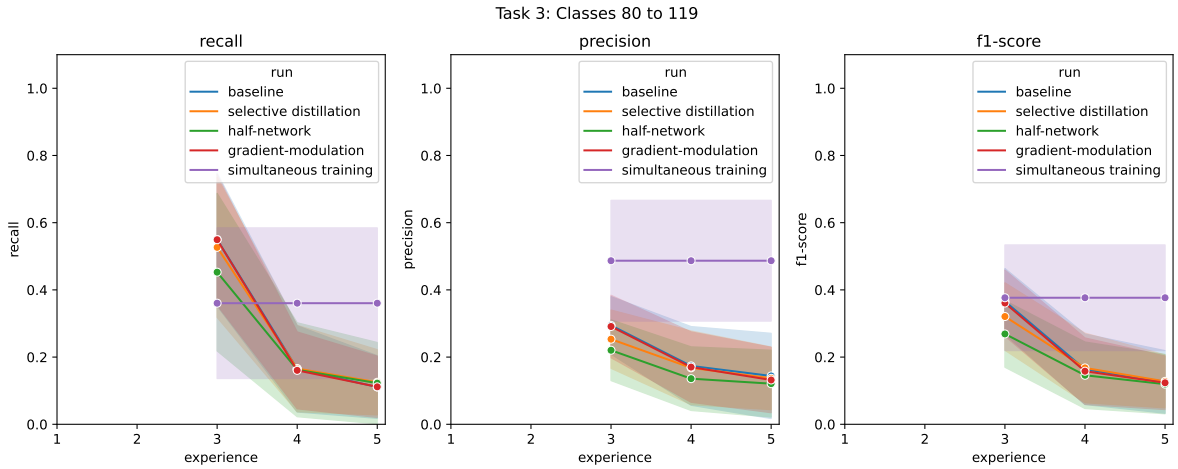

Figure 5.39: SplitTinyImageNet: Precision, recall, and f1-score of task 3 across tasks for different implemented methods.

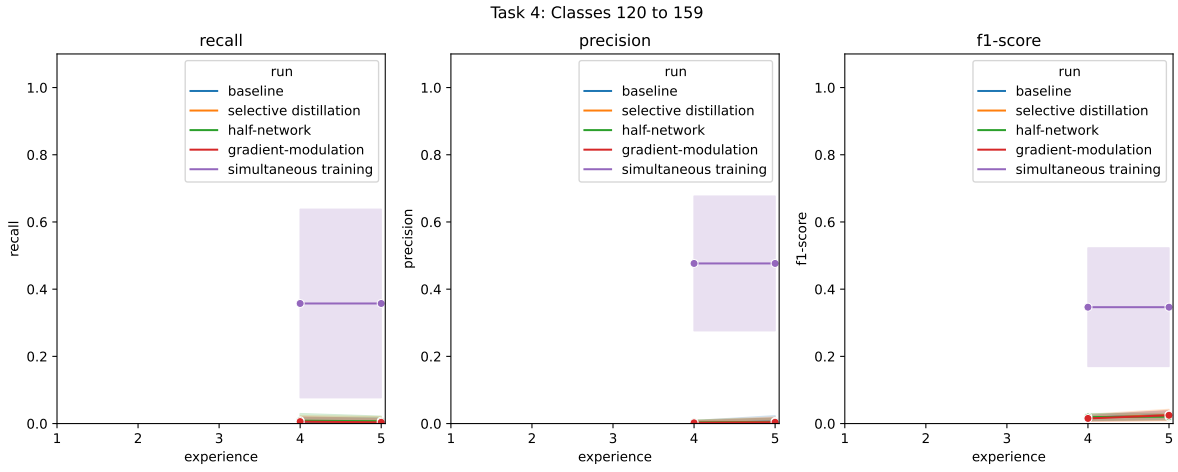

Figure 5.40: SplitTinyImageNet: Precision, recall, and f1-score of task 4 across tasks for different implemented methods.

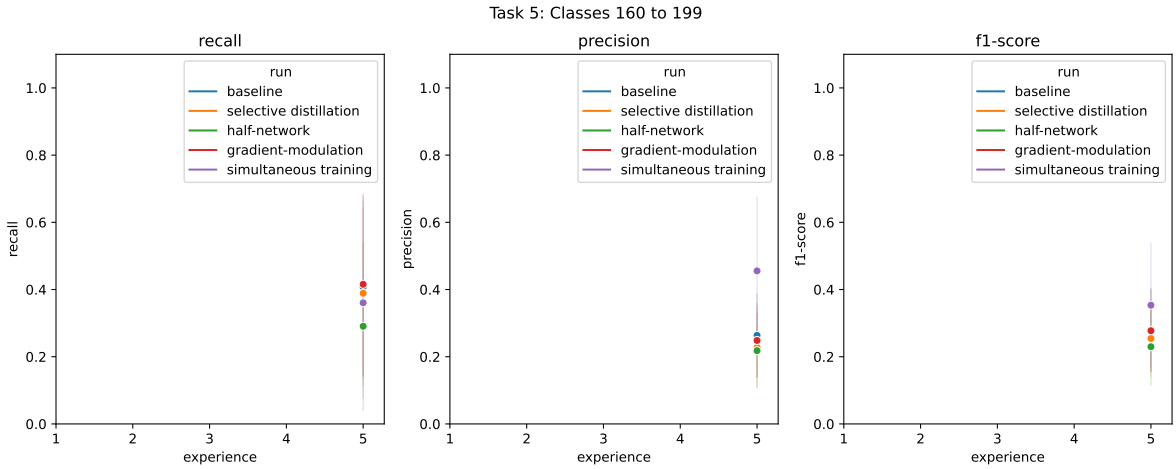

Figure 5.41: SplitTinyImageNet: Precision, recall, and f1-score of task 5 across tasks for different implemented methods.

5.4 Ablation Studies

In order to study the effect of the predictive batch and investigate how the size of rehearsal memory affects performance, additional experiments with SplitMNIST (representative of a small and relatively easy to learn dataset) and SplitTinyImageNet (representative of a larger and harder to learn dataset) datasets were conducted. In these experiments, the size of memory and the use of the predictive batch were manipulated and the resulting performance metrics were calculated. The results are described in the following sections.

5.4.1 Effect of the Predictive Batch

In an effort to examine whether identifying salient network parameters using the predictive batch provides improvements in performance, for each method (selective distillation, half-network, and gradient modulation) and the SplitMNIST and the SplitTinyImageNet datasets, an additional experiment where the salient parameters and parts of representations were chosen only based on the performance on a memory of previous tasks was conducted. The f1-score of each variant is shown in table 5.4 for SplitMNIST. For SplitTinyImageNet, the average accuracy of each variant is shown in table 5.5.

Experimental results on SplitMNIST

Table 5.4 summarizes each method's performance compared to its corresponding variant where the predictive batch was not used. There is, however, only one baseline (no salience computation in general, so not using the predictive batch is irrelevant for the baseline).

Comparing each method to its no-predictive-batch variant, gradient modulation achieves higher accuracy without using the predictive batch, while the half-network and selective distillation methods generally perform better when the predictive batch is leveraged for computing the salience of parameters/representations.

Comparing methods with each other, gradient modulation outperforms without using the predictive batch other methods after training on task 2 and 3, while selective distillation shows the highest f1-score after training on task 4 and 5 when using the predictive batch.

Table 5.4: SplitMNIST: Mean (std) of class f1-score on the test set after each task for variants using and not using the predictive batch. Variants that do not use the predictive batch are marked with *.

method	Task 1 f1-score	Task 2 f1-score	Task 3 f1-score	Task 4 f1-score	Task 5 f1-score
baseline	0.9988 (0.0011)	0.983 (0.0102)	0.9596 (0.0219)	0.9338 (0.0232)	0.8837 (0.0198)
gradient-modulation	0.999 (0.0007)	0.9879 (0.0039)	0.9621 (0.0097)	0.9404 (0.0147)	0.8905 (0.021)
gradient-modulation*	0.999 (0.001)	**0.9893 (0.0036)**	**0.9637 (0.0123)**	0.945 (0.0111)	0.8945 (0.0193)
half-network	0.9993 (0.0007)	0.9739 (0.02)	0.9604 (0.0163)	0.9383 (0.022)	0.8799 (0.0269)
half-network*	0.9996 (0.0004)	0.9717 (0.0115)	0.9556 (0.0203)	0.9335 (0.0208)	0.8817 (0.0287)
selective distillation	0.9994 (0.0005)	0.9829 (0.0133)	0.962 (0.0156)	**0.9497 (0.0038)**	**0.9073 (0.0155)**
selective distillation*	0.9993 (0.0005)	0.9826 (0.0059)	0.9587 (0.0184)	0.9436 (0.0182)	0.8751 (0.0336)
simultaneous training	0.9908 (0.0051)	0.9869 (0.0092)	0.9845 (0.0145)	0.9843 (0.0135)	0.9823 (0.0161)

Experimental results on SplitTinyImageNet

Table 5.5 summarizes each method's performance compared to its corresponding variant where the predictive batch was not used. There is, however, only one baseline (no variant where the predictive batch is not used), as it does not originally compute salience for any parameters/representations and using/not using the predictive batch is irrelevant.

The variant of each method that uses the predictive batch generally outperforms the variant that not uses the predictive batch in all tasks (with the exception of gradient modulation method in the final task). This, alongside results from SplitMNIST, shows that the using the predictive batch to compute salience of parameters/representations generally provides gains for knowledge preservation, both in small experiments (small architecture and datasets) and larger-scale experiments (with deeper architectures and larger datasets).

Comparing each method and variant with each other, both variants of the gradient modulation method outperform the selective distillation and half-network methods. The gradient modulation method also outperforms the baseline after task 2 and task 5, while no method manages to achieve a higher accuracy than the baseline after tasks 3 and 4.

Table 5.5: SplitTinyImageNet: Task mean (std) accuracy on the test set after each task for variants using and not using the predictive batch. Variants that do not use the predictive batch are marked with *.

method	Task 1 Avg. Accuracy	Task 2 Avg. Accuracy	Task 3 Avg. Accuracy	Task 4 Avg. Accuracy	Task 5 Avg. Accuracy
baseline	0.6234 (0.0051)	0.369 (0.0036)	**0.2691 (0.0066)**	**0.0955 (0.0031)**	0.1423 (0.0081)
gradient-modulation	0.5865 (0.086)	**0.3697 (0.0237)**	0.2655 (0.019)	0.0922 (0.013)	0.1427 (0.0082)
gradient-modulation*	0.5962 (0.055)	0.3638 (0.0132)	0.2622 (0.0125)	0.0918 (0.0128)	**0.1452 (0.0058)**
half-network	0.6171 (0.0135)	0.2941 (0.0045)	0.2089 (0.0064)	0.0833 (0.0074)	0.1163 (0.0051)
half-network*	0.5769 (0.0996)	0.2546 (0.0846)	0.1827 (0.0555)	0.0716 (0.0331)	0.1043 (0.0302)
selective distillation	0.6101 (0.0377)	0.3392 (0.0087)	0.2385 (0.0164)	0.086 (0.0095)	0.1356 (0.0027)
selective distillation*	0.5618 (0.1187)	0.3311 (0.0099)	0.234 (0.0141)	0.0784 (0.015)	0.1321 (0.0069)
simultaneous training	0.3832 (0.038)	0.4114 (0.0429)	0.3944 (0.0395)	0.3852 (0.0398)	0.3802 (0.0377)

5.4.2 Effect of Memory Size

In this ablation study, the effect of memory size is investigated. To this end, for each introduced method (selective distillation, half-network, and gradient modulation), an additional experiment was conducted with a larger memory to see the effect on performance. Specifically, while the memory size for training on the SplitMNIST dataset was originally 50, in the new experiments the memory size was set to 100. Similarly, for training on SplitTinyImageNet, memory size was set to 1000, in contrast to the original memory size of 200. The network architecture is also chosen according to the dataset, with a simple three layer CNN used for SplitMNIST while a reduced ResNet-18 was used for SplitTinyImageNet. In what follows the experimental results with a larger memory are compared to results with a small memory:

Experimental Results on SplitMNIST

Same as before, f1-score of seen classes after training on each task concludes are averaged and reported, for both the small and larger memory. These results are provided in the table 5.6. Generally, a larger memory leads to better performance (higher f1-score) for each method. The difference is going to be in how methods with a larger memory compare to each other.

When using a small memory of size 50, the gradient modulation and selective distillation methods outperform the baseline and manage to get the highest f1-score at different points in time. Using a larger memory of size 100, however, none of the methods achieve a higher f1-score than the baseline in the last two tasks. The gradient modulation and selective distillation methods outperform the baseline after task 2 with the former showing the highest f1-score. Measuring f1-score after task 3, only selective distillation achieves a higher f1-score compared to the baseline.

Table 5.6: SplitMNIST: Mean (std) of class f1-score on the test set after each task

Mem Size	method	Task 1 f1-score	Task 2 f1-score	Task 3 f1-score	Task 4 f1-score	Task 5 f1-score
	simultaneous training	0.9908 (0.0051)	0.9869 (0.0092)	0.9845 (0.0145)	0.9843 (0.0135)	0.9823 (0.0161)
50	**baseline**	0.9988 (0.0011)	0.983 (0.0102)	0.9596 (0.0219)	0.9338 (0.0232)	0.8837 (0.0198)
	gradient-modulation	0.999 (0.0007)	**0.9879 (0.0039)**	**0.9621 (0.0097)**	0.9404 (0.0147)	0.8905 (0.021)
	half-network	0.9993 (0.0007)	0.9739 (0.02)	0.9604 (0.0163)	0.9383 (0.022)	0.8799 (0.0269)
	selective distillation	0.9994 (0.0005)	0.9829 (0.0133)	0.962 (0.0156)	**0.9497 (0.0038)**	**0.9073 (0.0155)**
100	**baseline**	0.999 (0.0014)	0.9859 (0.0128)	0.9768 (0.011)	**0.9683 (0.0077)**	**0.9421 (0.0073)**
	gradient-modulation	0.9995 (0.0004)	**0.9904 (0.003)**	0.9764 (0.0091)	0.9671 (0.0068)	0.9385 (0.0066)
	half-network	0.9993 (0.0015)	0.9146 (0.1062)	0.937 (0.0404)	0.9302 (0.0453)	0.9075 (0.0358)
	selective distillation	0.9993 (0.0005)	0.9889 (0.0031)	0.9805 (0.0039)	0.9648 (0.0071)	0.9361 (0.0134)

Assessing performance for two different memory sizes provides an important insight: The proposed methods' effects are more significant when access to previous task samples is very limited (smaller memory). Overall, supervised contrastive learning with a larger memory appears to very robust and difficult to outperform.

The following figures (5.42-5.51) describe how each task's performance metrics change over time as the network gets trained on new tasks and allow comparison of each introduced method with other methods provided. While the baseline manages to provide better performance (precision, recall, and f1-score), it can be seen in the plots that the gradient modulation and selective distillation methods follow very closely.

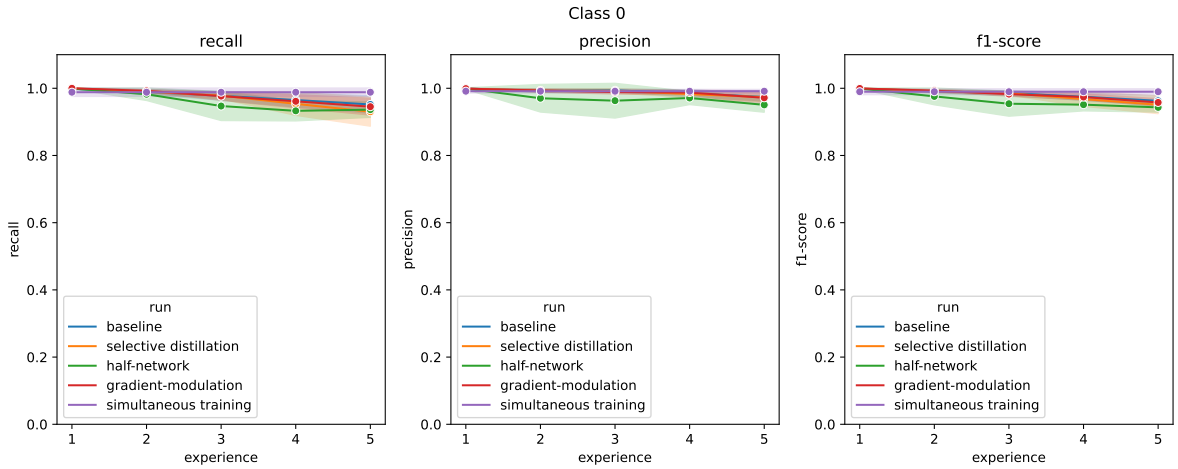

Figure 5.42: SplitMNIST: Precision, recall, and f1-score of class 0 across tasks for different implemented methods with a larger memory of size 100.

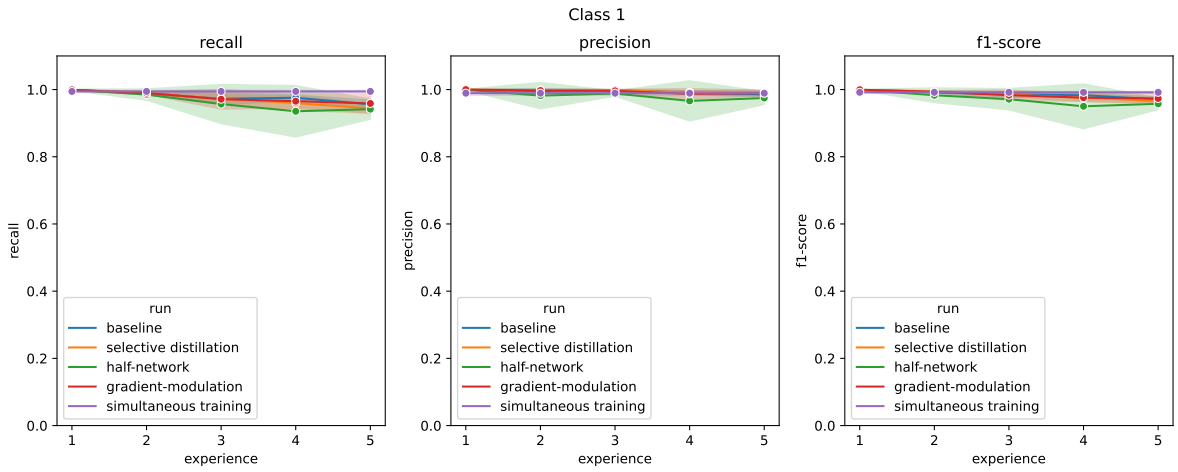

Figure 5.43: SplitMNIST: Precision, recall, and f1-score of class 1 across tasks for different implemented methods with a larger memory of size 100.

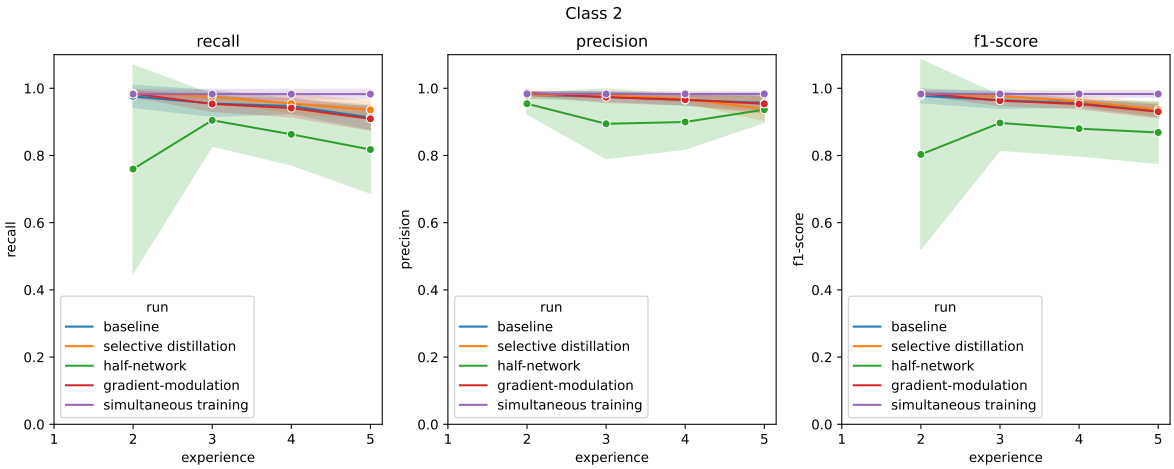

Figure 5.44: SplitMNIST: Precision, recall, and f1-score of class 2 across tasks for different implemented methods with a larger memory of size 100.

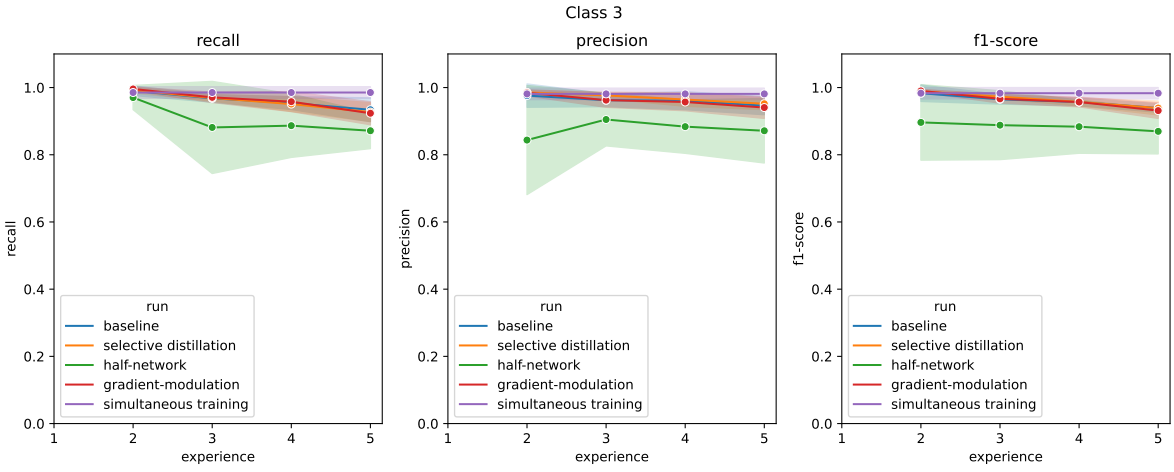

Figure 5.45: SplitMNIST: Precision, recall, and f1-score of class 3 across tasks for different implemented methods with a larger memory of size 100.

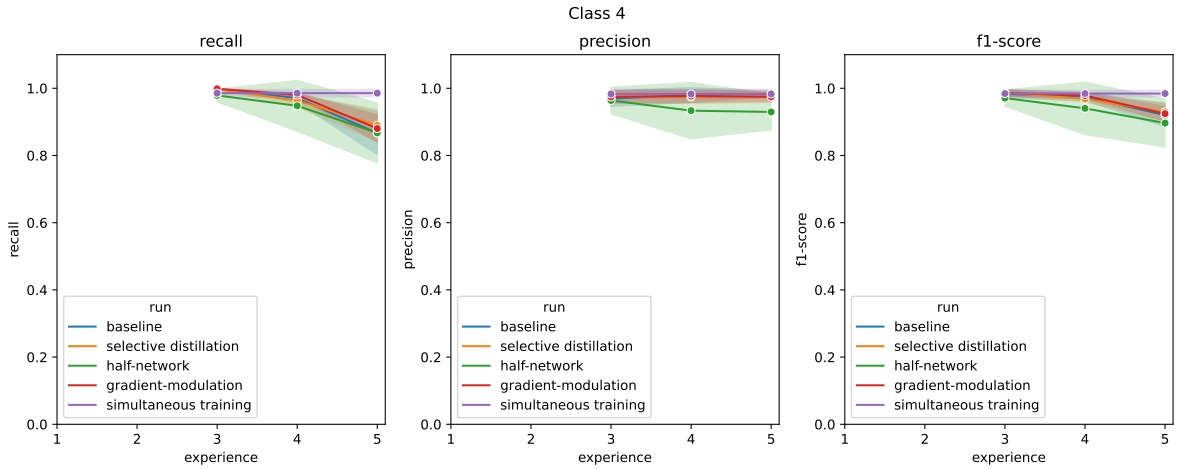

Figure 5.46: SplitMNIST: Precision, recall, and f1-score of class 5 across tasks for different implemented methods with a larger memory of size 100.

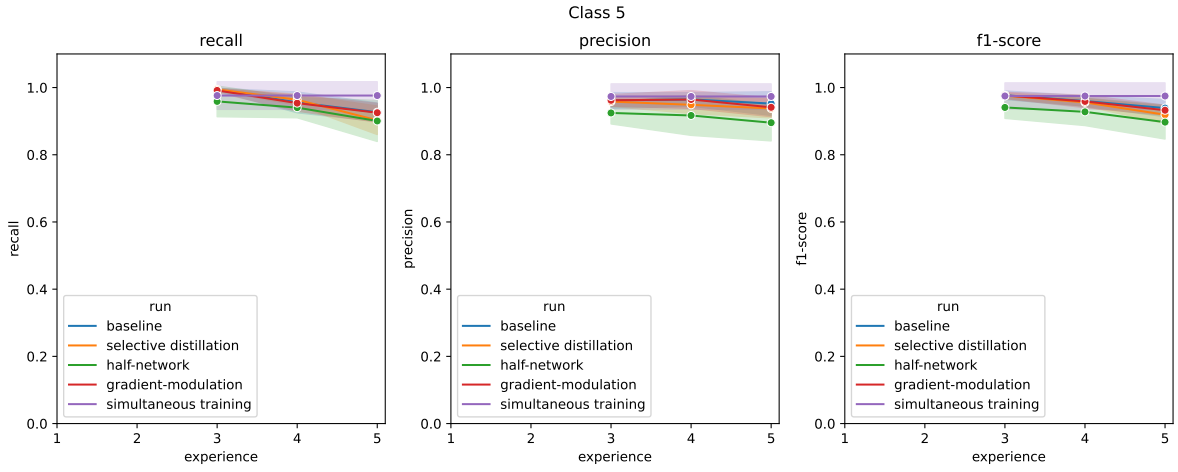

Figure 5.47: SplitMNIST: Precision, recall, and f1-score of class 5 across tasks for different implemented methods with a larger memory of size 100.

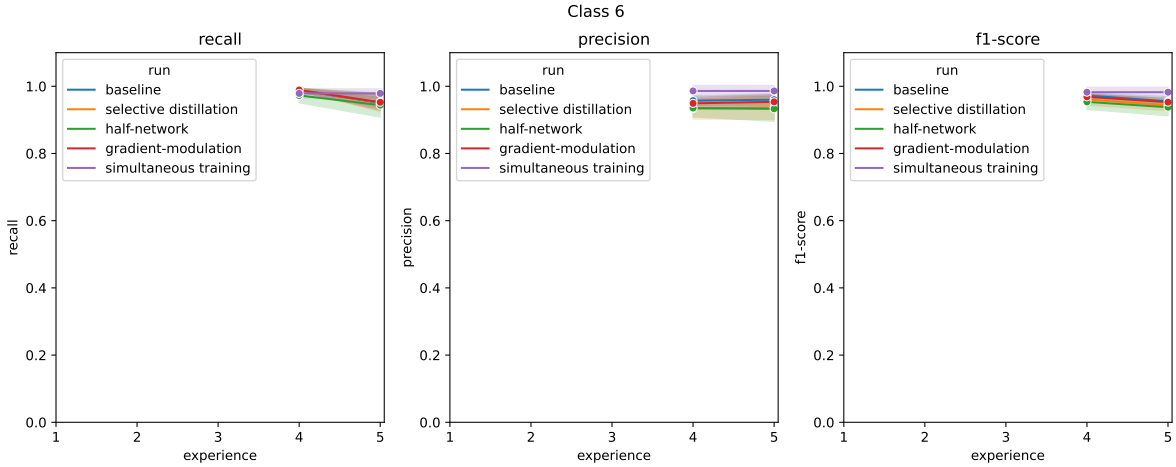

Figure 5.48: SplitMNIST: Precision, recall, and f1-score of class 6 across classes for different implemented methods with a larger memory of size 100.

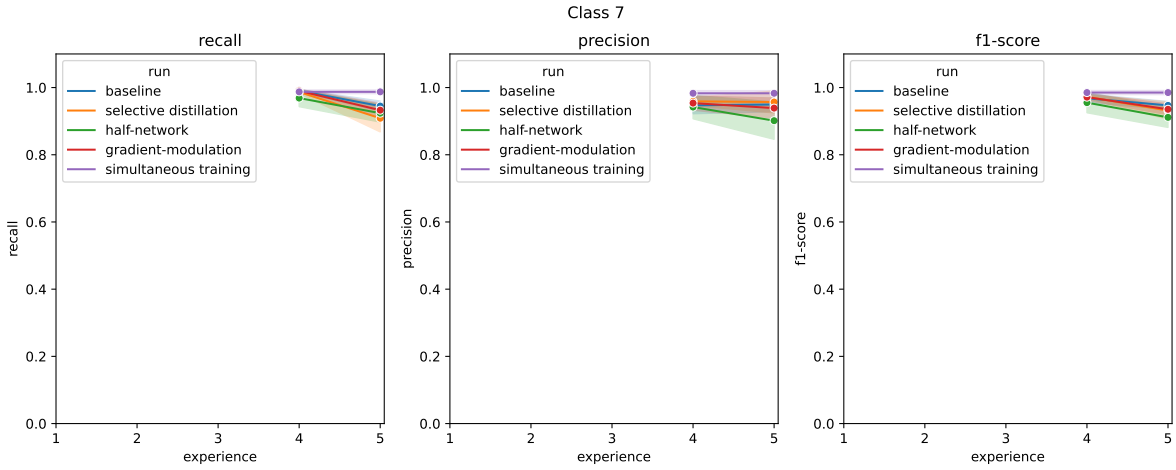

Figure 5.49: SplitMNIST: Precision, recall, and f1-score of class 7 across tasks for different implemented methods with a larger memory of size 100.

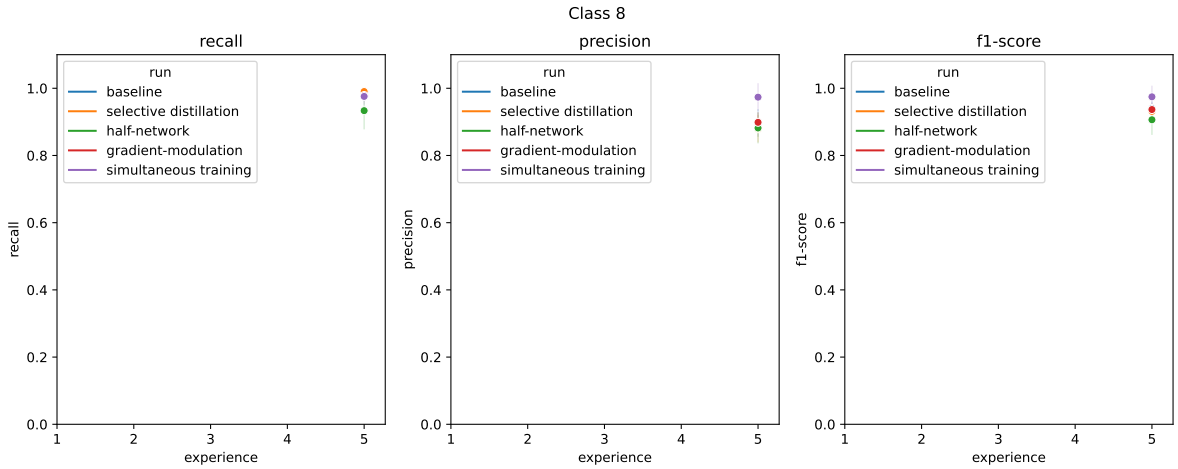

Figure 5.50: SplitMNIST: Precision, recall, and f1-score of class 8 across tasks for different implemented methods with a larger memory of size 100.

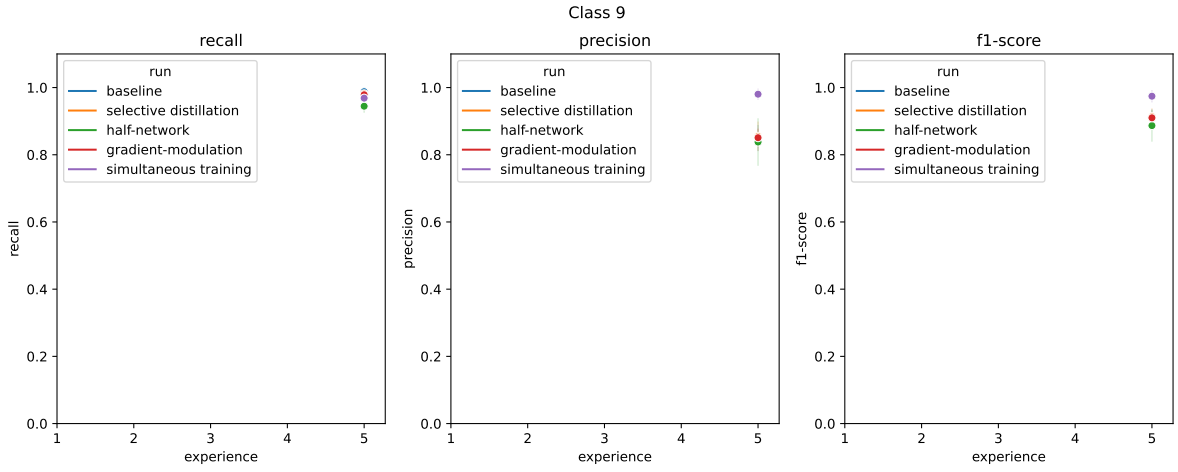

Figure 5.51: SplitMNIST: Precision, recall, and f1-score of class 9 across tasks for different implemented methods with a larger memory of size 100.

Experimental Results on SplitTinyImageNet

The accuracy on seen tasks is averaged after training on each task concludes. Table 5.7 describes the results evaluated after each task. The larger memory variant of each method naturally observes higher accuracy, in all of the tasks. It is good to note that the selective distillation method sees less improvement than the baseline, gradient modulation, and half-network methods, when comparing each method to its large memory variant. Moreover, comparing methods with each other when a larger memory is employed describes how performance of methods changes when the memory size is altered. While using a small memory, the baseline and gradient modulation methods generally outperformed the half-network and selective-distillation methods, with the gradient modulation method achieving a higher average accuracy after tasks 2 and 5. Similar to when a smaller memory was being used, using a larger memory, the baseline and gradient modulation methods show close performance metrics while outperforming selective distillation and half-network methods. However, when a larger memory is employed, the baseline achieves highest average accuracy after tasks 3, 4, and 5 while gradient modulation outperforms the baseline only after the second task. This follows the results on SplitMNIST: the effects of proposed methods are more significant when access to previous task samples is very limited and the size of the memory is small.

Table 5.7: SplitTinyImageNet: Task mean (std) accuracy on the test set for different memory sizes

Mem Size	method	Task 1 Avg. Accuracy	Task 2 Avg. Accuracy	Task 3 Avg. Accuracy	Task 4 Avg. Accuracy	Task 5 Avg. Accuracy
	simultaneous Training	0.3832 (0.038)	0.4114 (0.0429)	0.3944 (0.0395)	0.3852 (0.0398)	0.3802 (0.0377)
200	**baseline**	0.6234 (0.0051)	0.369 (0.0036)	**0.2691 (0.0066)**	**0.0955 (0.0031)**	0.1423 (0.0081)
	gradient-modulation	0.5865 (0.086)	**0.3697 (0.0237)**	0.2655 (0.019)	0.0922 (0.013)	**0.1427 (0.0082)**
	half-network	0.6171 (0.0135)	0.2941 (0.0045)	0.2089 (0.0064)	0.0833 (0.0074)	0.1163 (0.0051)
	selective distillation	0.6101 (0.0377)	0.3392 (0.0087)	0.2385 (0.0164)	0.086 (0.0095)	0.1356 (0.0027)
1000	**baseline**	0.6157 (0.0239)	0.4239 (0.0077)	**0.3193 (0.0052)**	**0.1478 (0.0049)**	**0.1626 (0.0054)**
	gradient-modulation	0.6222 (0.011)	**0.428 (0.0047)**	0.314 (0.0095)	0.1448 (0.0073)	0.16 (0.0062)
	half-network	0.6163 (0.027)	0.3545 (0.0182)	0.262 (0.0157)	0.1399 (0.0073)	0.1458 (0.0095)
	selective distillation	0.5249 (0.1717)	0.3441 (0.071)	0.2613 (0.0392)	0.1188 (0.0289)	0.1398 (0.0195)

Lastly, in what follows plots of performance metrics on a task across current and next tasks are illustrated (figures 5.52-5.56). Similar to before, these plots show how each performance metric on a task changes over time as the network gets trained on new tasks. These plots show how closely the gradient modulation method follows the baseline in different metrics.

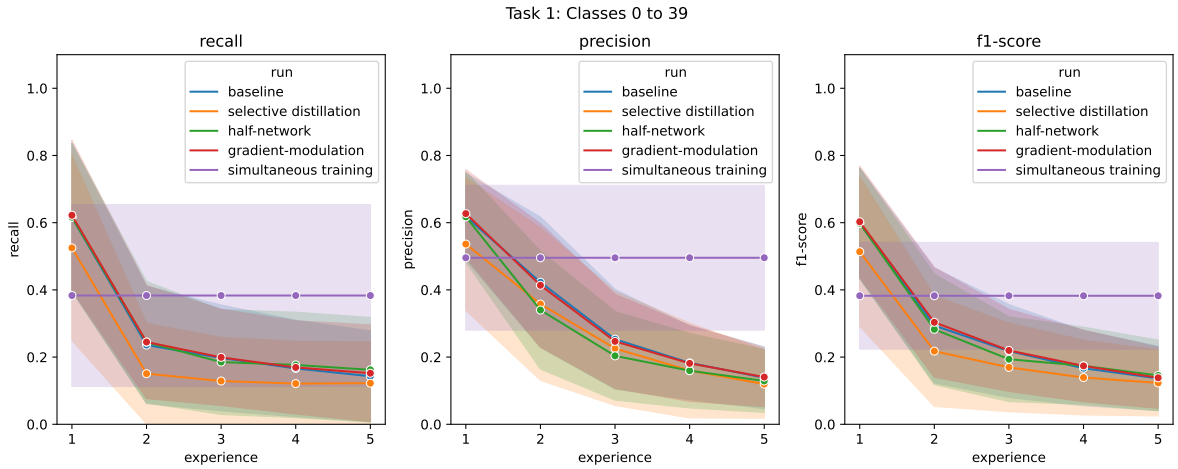

Figure 5.52: SplitTinyImageNet: Precision, recall, and f1-score of task 1 across tasks for different implemented methods with a larger memory of 1000.

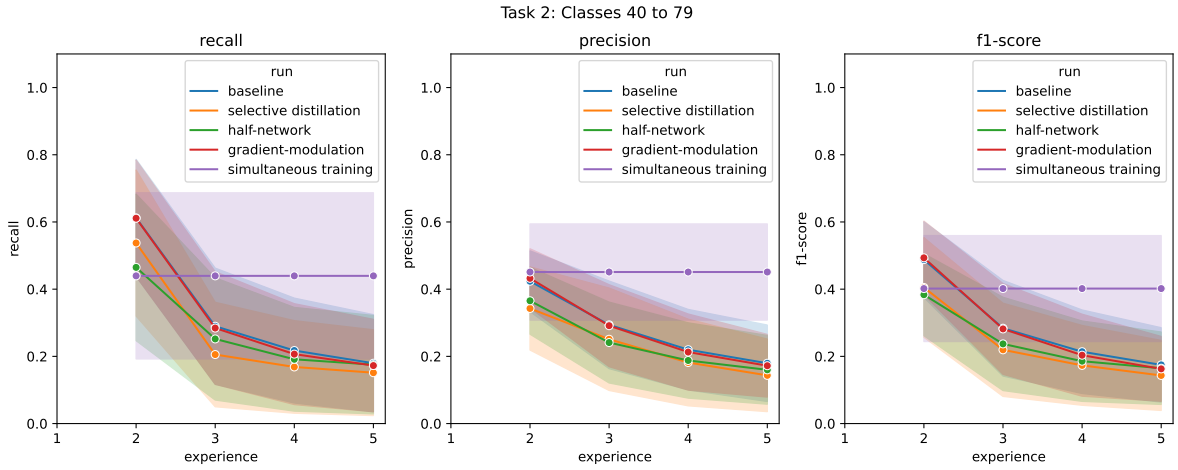

Figure 5.53: SplitTinyImageNet: Precision, recall, and f1-score of task 2 across tasks for different implemented methods with a larger memory of 1000.

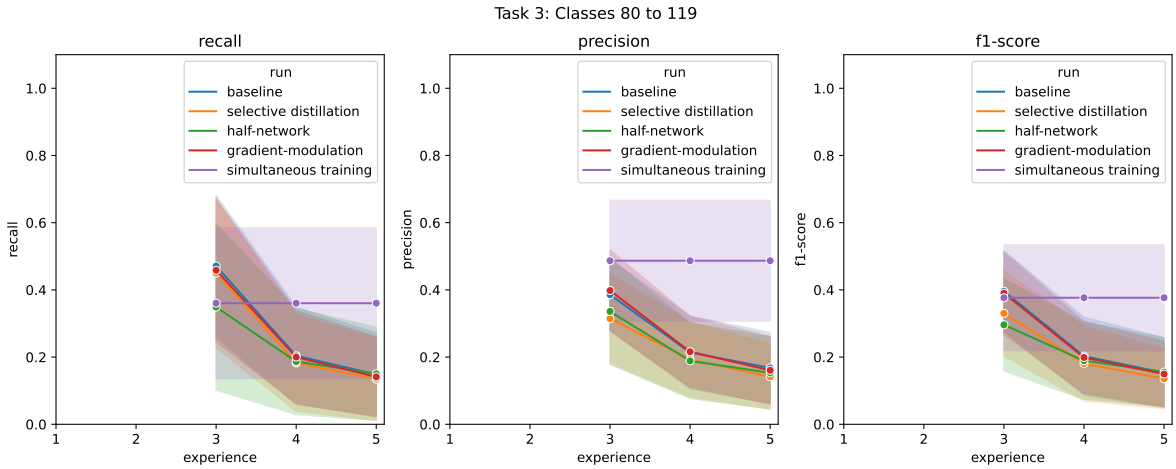

Figure 5.54: SplitTinyImageNet: Precision, recall, and f1-score of task 3 across tasks for different implemented methods with a larger memory of 1000.

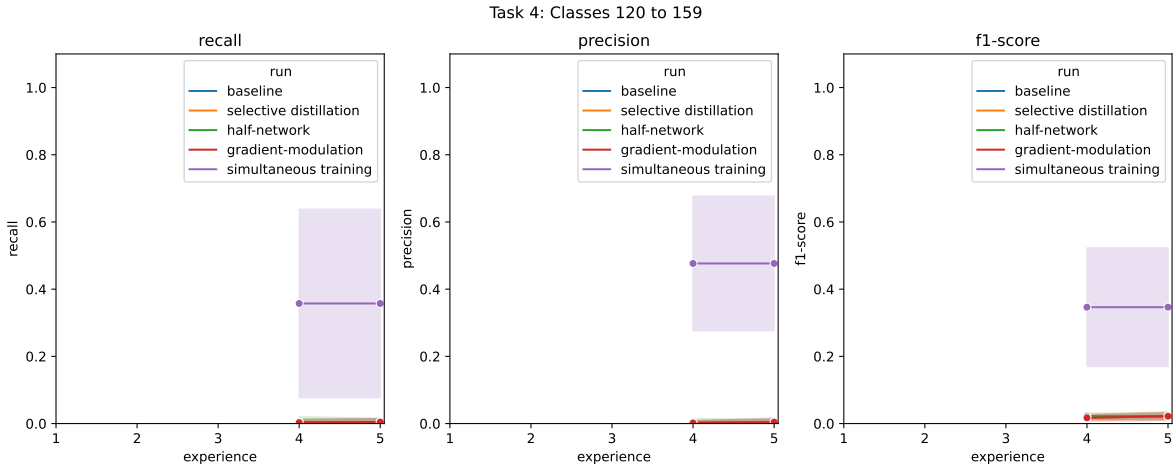

Figure 5.55: SplitTinyImageNet: Precision, recall, and f1-score of task 4 across tasks for different implemented methods with a larger memory of 1000.

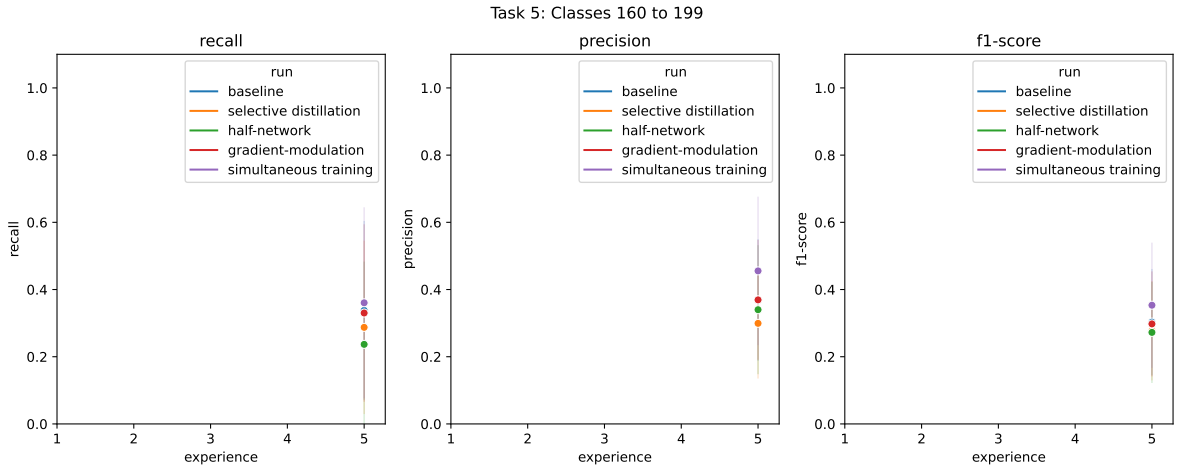

Figure 5.56: SplitTinyImageNet: Precision, recall, and f1-score of task 5 across tasks for different implemented methods with a larger memory of 1000.

5.5 Comparison of Proposed Methods to State-of-the-Art

While experiments on SplitMNIST were to show preliminary results on small scale datasets, it is good to compared the performance of proposed methods with previous state-of-the-art. However, doing so requires experimenting in identical or very similar settings. Here, we chose to experiment with SplitCIFAR10 and SplitTinyImageNet datasets in the class-incremental setting only. Moreover, only a memory size of 200 was tested, while previous work also experimented with a larger memory size of 500. It should also be noted that the network architecture being used for the proposed methods is a reduced version of ResNet-18, while previous work employed the original full-size ResNet-18. Table 5.8 shows the average accuracy of proposed methods accompanied by results of previous state-of-the-art continual learning approaches. The results of previous state-of-the-art are extracted from [13] and [12].

Table 5.8: Comparison of proposed methods with previous state-of-the-art continual learning approaches: average (std) accuracy over all tasks are reported. State-of-the-art results are measured over 10 independent trials. The best performance are marked with bold. '-' denotes that results were not recorded because of incompatibility issues or intractable training time.

dataset **method**	SplitCIFAR10	SplitTinyImageNet
ER [74]	44.79 (1.86)	8.49 (0.16)
GEM [58]	25.54 (0.76)	-
A-GEM [15]	20.04 (0.34)	8.07 (0.08)
iCaRL [72]	49.02 (3.20)	7.53 (0.79)
FDR [9]	30.91 (2.74)	8.70 (0.19)
GSS [2]	39.07 (5.59)	-
HAL [14]	32.36 (2.70)	-
DER [12]	61.93 (1.79)	11.87 (0.78)
DER++ [12]	64.88 (1.17)	10.96 (1.17)
Co^{2}L[13]	**65.57 (1.37)**	13.88 (0.40)
LASP: gradient-modulation (ours)	46.39 (1.27)	**14.27 (0.82)**
LASP: half-network (ours)	46.38 (1.24)	11.63 (0.51)
LASP: selective distillation (ours)	43.35 (1.20)	13.56 (0.27)
baseline (ours)	47.35 (1.33)	14.23 (0.81)
simultaneous training (upper bound)	83.94 (0.59)	38.02 (3.77)

The proposed gradient modulation method achieves state-of-the-art average accuracy on the SplitTinyImageNet dataset (class-incremental, memory size of 200) while using a smaller architecture and fewer number of parameters. For SplitCIFAR10, however, none of the proposed methods achieve a higher accuracy than previous state-of-the-art. This can be in part attributed to unstable training when using larger batch sizes like Co^{2}L for SplitCIFAR10 and inability to reproduce results close to Co^{2}L with our implementation.

Chapter 6

Discussion and Conclusion

In this thesis, the general continual learning problem and its different settings was described. A representative set of previous work in the field was also explained and their limitations were identified. Marking the limitations by previous work, a new continual learning setting was proposed. In the new setting, it is possible to access a part of the upcoming dataset at once, allowing the network to be evaluated on new unseen data. Moreover, inspired by the neuromodulatory processes in the brain, two main and novel approaches were presented that assess which parts of a representation generated by a feature extractor are more salient and backpropagate the computed salience to each network parameter. Consequently, network parameters that hold the knowledge of past tasks and are likely to transfer to future tasks were identified. Three novel methods were also introduced to preserve the knowledge acquired by the network using the computed salience values for network parameters and representations.

The results presented in chapter 5 show that each of the proposed methods can be helpful in different circumstances. For example, selective distillation was able to achieve highest f1-score among others for SplitMNIST, while not being able to outperform the baseline for SplitCIFAR10. While the results explain in which conditions (memory size, with/without using the predictive batch, dataset) each proposed method appears to work best, additional experiments are required to understand why some of the proposed methods are unable to outperform the baseline.

For SplitCIFAR10, the decrease in recall over the course of training on tasks is large. After the steep decrease to around 0.2, however, there is no more decrease, and in some of the classes (2, 3, 4, and 5) there is an increase in recall after training on the fifth and final task. This shows that the network can recover some of the lost performance and knowledge by revisiting a small sample of each task's data and training the subsequent tasks. While this backward transfer effect is not very large, it is promising as in the longer sequence of tasks, this may provide a solution to the catastrophic forgetting problem by slowly recovering performance.

The training process on the SplitTinyImageNet and SplitCIFAR10 datasets was unstable when a large batch size was being used: The network had issues converging in some runs. There was no reported instability or convergence problems in [42]. However, a *collapse* (and a *dimensional collapse*) problem has been identified as one of the main issues with contrastive loss [40], where the network outputs a constant embedding for all inputs, or outputs embeddings lying in a lower dimensional space compared to the actual dimensions of the embedding vector. Training with the right learning rate scheduling and warm up strategies could mitigate

this problem (see [91]). Making the supervised contrastive loss more stable and finding better training strategies (including for this thesis) remains an open problem and a valid future direction of work.

The effect of memory and the predictive batch was also studied. The proposed methods generally perform better (according to precision, recall, f1-score, and average accuracy) when equipped with a larger memory. However, methods perform differently compared to each other given the size of the memory: While the proposed methods demonstrate a better performance compared to the baseline when a small memory is used, they are unable to outperform the baseline when a larger memory is incorporated. The employed memory strategy can be playing a significant part in these behaviour changes. While in this thesis iCaRL's [72] herding strategy was used for managing the memory, other memory strategies like reservoir sampling [74, 94] could lead to better performance when a large memory is being used. The resulting behaviours of the proposed methods could also get affected by this choice of memory management strategy. The use of the predictive batch also proved to be useful in many instances, showing how measuring a network's performance on new unseen data and attributing it to the network parameters can help evaluate whether they would generally transfer or not. An specific and detailed answer to whether the network would generate representations that transfer to downstream tasks, however, depends on the specific properties of downstream tasks and requires further examination.

The implementation used for this study is different from the one used for Co^2L [13], although extra care was taken to replicate all of its details. While Co^2l achieves much better accuracy for CIFAR10, the implementation here was not able to replicate the results of Co^2L with the same hyperparameters and model details. Further experiments using the available implementation for Co^2L is required to test the methods proposed in this thesis.

While the methods introduced in this study have the following advantages,

1. They allow and encourage forward and backward transfer, via selecting and preserving neurons that perform well on past, current, and (a predictive batch of) future task.

2. They require limited computational resources.

3. They work in the class-incremental continual learning setting, and there is no theoretical limit to the number of classes they can learn.

they have a few drawbacks:

1. The performance on preventing catastrophic forgetting could be better.

2. For larger datasets like SplitCIFAR10 and SplitTinyImageNet, they require multiple passes through the data to perform well. On the SplitMNIST, however, they can be trained with only two passes (nearly online).

Given the promising results presented in this thesis, there is room for significant improvement with a more stable training process and a more detailed identification (and possibly guidance) of salient network parameters in the future.

Bibliography

[1] Kumar Abhishek and Deeksha Kamath. Attribution-based xai methods in computer vision: A review. *arXiv preprint arXiv:2211.14736*, 2022.

[2] Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning. *Advances in neural information processing systems*, 32, 2019.

[3] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. *Advances in neural information processing systems*, 29, 2016.

[4] Michael C Avery, Nikil Dutt, and Jeffrey L Krichmar. Mechanisms underlying the basal forebrain enhancement of top-down and bottom-up attention. *European Journal of Neuroscience*, 39(5):852–865, 2014.

[5] Markus Bauer, Christian Kluge, Dominik Bach, David Bradbury, Hans Jochen Heinze, Raymond J Dolan, and Jon Driver. Cholinergic enhancement of visual attention and neural oscillations in the human brain. *Current Biology*, 22(5):397–402, 2012.

[6] Shawn Beaulieu, Lapo Frati, Thomas Miconi, Joel Lehman, Kenneth O Stanley, Jeff Clune, and Nick Cheney. Learning to continually learn. *arXiv preprint arXiv:2002.09571*, 2020.

[7] Samy Bengio, Yoshua Bengio, Jocelyn Cloutier, and Jan Gescei. On the optimization of a synaptic learning rule. In *Optimality in Biological and Artificial Networks?*, pages 281–303. Routledge, 2013.

[8] David Beniaguev, Idan Segev, and Michael London. Single cortical neurons as deep artificial neural networks. *Neuron*, 109(17):2727–2739, 2021.

[9] Ari S Benjamin, David Rolnick, and Konrad Kording. Measuring and regularizing networks in function space. *arXiv preprint arXiv:1805.08289*, 2018.

[10] Paul Bentley, Jon Driver, and Raymond J Dolan. Cholinergic modulation of cognition: insights from human pharmacological functional neuroimaging. *Progress in neurobiology*, 94(4):360–388, 2011.

[11] Bryan R Burnham. Displaywide visual features associated with a search display's appearance can mediate attentional capture. *Psychonomic Bulletin & Review*, 14(3):392–422, 2007.

[12] Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and Simone Calderara. Dark experience for general continual learning: a strong, simple baseline. *Advances in neural information processing systems*, 33:15920–15930, 2020.

[13] Hyuntak Cha, Jaeho Lee, and Jinwoo Shin. Co2l: Contrastive continual learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9516–9525, 2021.

[14] Arslan Chaudhry, Albert Gordo, Puneet Dokania, Philip Torr, and David Lopez-Paz. Using hindsight to anchor past knowledge in continual learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 6993–7001, 2021.

[15] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. *arXiv preprint arXiv:1812.00420*, 2018.

[16] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[17] Z Chen and B Liu. Lifelong machine learning: Synthesis lectures on artificial intelligence and machine learning. *San Rafael, CA, USA: Morgan and Claypool Publishers*, pages 1–127, 2016.

[18] Moheb Costandi. *Neuroplasticity*. MIt Press, 2016.

[19] Anurag Daram, Angel Yanguas-Gil, and Dhireesha Kudithipudi. Exploring neuromodulation for dynamic learning. *Frontiers in Neuroscience*, 14:928, 2020.

[20] Peter Dayan and Angela J Yu. Phasic norepinephrine: a neural interrupt signal for unexpected events. *Network: Computation in Neural Systems*, 17(4):335–350, 2006.

[21] Kenji Doya. Metalearning and neuromodulation. *Neural networks*, 15(4-6):495–506, 2002.

[22] Kai Olav Ellefsen, Jean-Baptiste Mouret, and Jeff Clune. Neural modularity helps organisms evolve to learn new skills without forgetting old skills. *PLoS computational biology*, 11(4):e1004128, 2015.

[23] Xiaoxu Fan, Fan Wang, Hanyu Shao, Peng Zhang, and Sheng He. The bottom-up and top-down processing of faces in the human occipitotemporal cortex. *Elife*, 9:e48764, 2020.

[24] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

[25] Charles L Folk, Roger W Remington, and James C Johnston. Involuntary covert orienting is contingent on attentional control settings. *Journal of Experimental Psychology: Human perception and performance*, 18(4):1030, 1992.

[26] Ruth Fong, Mandela Patrick, and Andrea Vedaldi. Understanding deep networks via extremal perturbations and smooth masks. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 2950–2958, 2019.

[27] Chuanxing Geng, Sheng-jun Huang, and Songcan Chen. Recent advances in open set recognition: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 43(10):3614–3631, 2020.

[28] Albert Gidon, Timothy Adam Zolnik, Pawel Fidzinski, Felix Bolduan, Athanasia Papoutsi, Panayiota Poirazi, Martin Holtkamp, Imre Vida, and Matthew Evan Larkum. Dendritic action potentials and computation in human layer 2/3 cortical neurons. *Science*, 367(6473):83–87, 2020.

[29] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[30] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[31] Raia Hadsell, Dushyant Rao, Andrei A Rusu, and Razvan Pascanu. Embracing change: Continual learning in deep neural networks. *Trends in cognitive sciences*, 24(12):1028–1040, 2020.

[32] Song Han, Jeff Pool, Sharan Narang, Huizi Mao, Enhao Gong, Shijian Tang, Erich Elsen, Peter Vajda, Manohar Paluri, John Tran, et al. Dsd: Dense-sparse-dense training for deep neural networks. *arXiv preprint arXiv:1607.04381*, 2016.

[33] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

[34] Kaiming He and Jian Sun. Convolutional neural networks at constrained time cost. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5353–5360, 2015.

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

[36] Donald Olding Hebb. *The organization of behavior: A neuropsychological theory*. Psychology Press, 2005.

[37] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7), 2015.

[38] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[39] Khurram Javed and Martha White. Meta-learning representations for continual learning. *Advances in Neural Information Processing Systems*, 32, 2019.

[40] Li Jing, Pascal Vincent, Yann LeCun, and Yuandong Tian. Understanding dimensional collapse in contrastive self-supervised learning. *arXiv preprint arXiv:2110.09348*, 2021.

[41] Paul Katz and DH Edwards. *Beyond neurotransmission*. Oxford University Press New York, 1999.

[42] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. *Advances in Neural Information Processing Systems*, 33:18661–18673, 2020.

[43] J Kirkpatrick, R Pascanu, N Rabinowitz, J Veness, G Desjardins, AA Rusu, K Milan, J Quan, T Ramalho, A Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences of the United States of America*, 114(13):3521–3526, 2017.

[44] Soheil Kolouri, Nicholas Ketz, Xinyun Zou, Jeffrey Krichmar, and Praveen Pilly. Attention-based structural-plasticity. *arXiv preprint arXiv:1903.06070*, 2019.

[45] Dhireesha Kudithipudi, Mario Aguilar-Simon, Jonathan Babb, Maxim Bazhenov, Douglas Blackiston, Josh Bongard, Andrew P Brna, Suraj Chakravarthi Raja, Nick Cheney, Jeff Clune, et al. Biological underpinnings for lifelong learning machines. *Nature Machine Intelligence*, 4(3):196–210, 2022.

[46] Pranjal Kumar, Piyush Rawat, and Siddhartha Chauhan. Contrastive self-supervised learning: review, progress, challenges and future research directions. *International Journal of Multimedia Information Retrieval*, pages 1–28, 2022.

[47] Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum, and Samuel J Gershman. Building machines that learn and think like people. *Behavioral and brain sciences*, 40, 2017.

[48] Phuc H Le-Khac, Graham Healy, and Alan F Smeaton. Contrastive representation learning: A framework and review. *IEEE Access*, 8:193907–193934, 2020.

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

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

[51] Yann A. LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. *Efficient BackProp*, pages 9–48. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[52] Zhizhong Li and Derek Hoiem. Learning without forgetting. In *European Conference on Computer Vision*, pages 614–629. Springer, 2016.

[53] Zhiqiu Lin, Jia Shi, Deepak Pathak, and Deva Ramanan. The clear benchmark: Continual learning on real-world imagery. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

[54] Weiyang Liu, Zhen Liu, James M Rehg, and Le Song. Neural similarity learning. *Advances in Neural Information Processing Systems*, 32, 2019.

[55] Yunzhe Liu, Raymond J Dolan, Zeb Kurth-Nelson, and Timothy EJ Behrens. Human replay spontaneously reorganizes experience. *Cell*, 178(3):640–652, 2019.

[56] Vincenzo Lomonaco and Davide Maltoni. Core50: a new dataset and benchmark for continuous object recognition. In *Conference on Robot Learning*, pages 17–26. PMLR, 2017.

[57] Vincenzo Lomonaco, Lorenzo Pellegrini, Andrea Cossu, Antonio Carta, Gabriele Graffieti, Tyler L Hayes, Matthias De Lange, Marc Masana, Jary Pomponi, Gido M Van de Ven, et al. Avalanche: an end-to-end library for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3600–3610, 2021.

[58] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30, 2017.

[59] Sandeep Madireddy, Angel Yanguas-Gil, and Prasanna Balaprakash. Neuromodulated neural architectures with local error signals for memory-constrained online continual learning. *arXiv preprint arXiv:2007.08159*, 2020.

[60] Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 7765–7773, 2018.

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

[62] Jie Mei, Eilif Muller, and Srikanth Ramaswamy. Informing deep neural networks by multiscale principles of neuromodulatory systems. *Trends in Neurosciences*, 2022.

[63] Thomas Miconi, Aditya Rawal, Jeff Clune, and Kenneth O Stanley. Backpropamine: training self-modifying neural networks with differentiable neuromodulated plasticity. *arXiv preprint arXiv:2002.10585*, 2020.

[64] Lilyana Mihalkova, Tuyen Huynh, and Raymond J Mooney. Mapping and revising markov logic networks for transfer learning. In *Aaai*, volume 7, pages 608–614, 2007.

[65] Victor Minces, Lucas Pinto, Yang Dan, and Andrea A Chiba. Cholinergic shaping of neural correlations. *Proceedings of the National Academy of Sciences*, 114(22):5725–5730, 2017.

[66] Erkki Oja. Simplified neuron model as a principal component analyzer. *Journal of mathematical biology*, 15(3):267–273, 1982.

[67] Nicolas Oros, Andrea A Chiba, Douglas A Nitz, and Jeffrey L Krichmar. Learning to ignore: a modeling study of a decremental cholinergic pathway and its influence on attention and learning. *Learning & Memory*, 21(2):105–118, 2014.

[68] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[69] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[70] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. 2016.

[71] Waseem Rawat and Zenghui Wang. Deep convolutional neural networks for image classification: A comprehensive review. *Neural computation*, 29(9):2352–2449, 2017.

[72] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 2001–2010, 2017.

[73] Roger L Redondo and Richard GM Morris. Making memories last: the synaptic tagging and capture hypothesis. *Nature Reviews Neuroscience*, 12(1):17–30, 2011.

[74] Matthew Riemer, Ignacio Cases, Robert Ajemian, Miao Liu, Irina Rish, Yuhai Tu, and Gerald Tesauro. Learning to learn without forgetting by maximizing transfer and minimizing interference. *arXiv preprint arXiv:1810.11910*, 2018.

[75] Anthony Robins. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science*, 7(2):123–146, 1995.

[76] David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. *Advances in Neural Information Processing Systems*, 32, 2019.

[77] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[78] Martin Sarter, Cindy Lustig, Anne S Berry, Howard Gritton, William M Howe, and Vinay Parikh. What do phasic cholinergic signals do? *Neurobiology of Learning and Memory*, 130:135–141, 2016.

[79] H Sato, Y Hata, H Masui, and T Tsumoto. A functional role of cholinergic innervation to neurons in the cat visual cortex. *Journal of neurophysiology*, 58(4):765–780, 1987.

[80] Jürgen Schmidhuber. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. PhD thesis, Technische Universität München, 1987.

[81] Wolfram Schultz, Paul Apicella, and Tomas Ljungberg. Responses of monkey dopamine neurons to reward and conditioned stimuli during successive steps of learning a delayed response task. *Journal of neuroscience*, 13(3):900–913, 1993.

[82] Wolfram Schultz, Peter Dayan, and P Read Montague. A neural substrate of prediction and reward. *Science*, 275(5306):1593–1599, 1997.

[83] Jonathan Schwarz, Wojciech Czarnecki, Jelena Luketina, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning. In *International Conference on Machine Learning*, pages 4528–4537. PMLR, 2018.

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

[85] Luisa Speranza, Umberto di Porzio, Davide Viggiano, Antonio de Donato, and Floriana Volpicelli. Dopamine: The neuromodulator of long-term synaptic plasticity, reward and movement control. *Cells*, 10(4):735, 2021.

[86] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

[87] Elizabeth E Steinberg, Ronald Keiflin, Josiah R Boivin, Ilana B Witten, Karl Deisseroth, and Patricia H Janak. A causal link between prediction errors, dopamine neurons and learning. *Nature neuroscience*, 16(7):966–973, 2013.

[88] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *International conference on machine learning*, pages 3319–3328. PMLR, 2017.

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

[90] Jan Theeuwes. Top–down and bottom–up control of visual selection. *Acta psychologica*, 135(2):77–99, 2010.

[91] Yonglong Tian. Github repository issues · hobbitlong/supcontrast.

[92] Joaquin Vanschoren. Meta-learning: A survey. *arXiv preprint arXiv:1810.03548*, 2018.

[93] Roby Velez and Jeff Clune. Diffusion-based neuromodulation can eliminate catastrophic forgetting in simple neural networks. *PloS one*, 12(11):e0187736, 2017.

[94] Jeffrey S Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57, 1985.

[95] Yaqing Wang, Quanming Yao, James T Kwok, and Lionel M Ni. Generalizing from a few examples: A survey on few-shot learning. *ACM computing surveys (csur)*, 53(3):1–34, 2020.

[96] Michael L Waskom. Seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021, 2021.

[97] Guang Yang, Feng Pan, and Wen-Biao Gan. Stably maintained dendritic spines are associated with lifelong memories. *Nature*, 462(7275):920–924, 2009.

[98] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, pages 3987–3995. PMLR, 2017.

[99] Jianming Zhang, Sarah Adel Bargal, Zhe Lin, Jonathan Brandt, Xiaohui Shen, and Stan Sclaroff. Top-down neural attention by excitation backprop. *International Journal of Computer Vision*, 126(10):1084–1102, 2018.

[100] Yu Zhang and Qiang Yang. A survey on multi-task learning. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[101] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Object detectors emerge in deep scene cnns. *arXiv preprint arXiv:1412.6856*, 2014.

[102] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76, 2020.

[103] W Zinke, MJ Roberts, K Guo, JS McDonald, R Robertson, and A Thiele. Cholinergic modulation of response properties and orientation tuning of neurons in primary visual cortex of anaesthetized marmoset monkeys. *European Journal of Neuroscience*, 24(1):314–328, 2006.

Curriculum Vitae

Name:	Rouzbeh Meshkinnejad
Post-Secondary	2013 - 2017 Allameh Helli High School Tehran, Iran
Education and	2017 - 2021 B.Sc. Computer Engineering, Sharif University of Technology Tehran, Iran
Degrees:	2021 - current M.Sc. Computer Science, University of Western Ontario London, ON
Honours and	2021 Vector Scholarship Recipient
Awards:	Gold Division in USA Computing Olympiad 2017 - current Member of National Iranian Elites Foundation
Related Work	Teaching Assistant C Programming (1 term) AI (3 terms) Probability and Statistics (1 term)
Experience:	2022 - current Data Scientist at ALS Goldspot Discoveries 2021 - current Graduate Fellow at University of Western Ontario

Publications:

Mei, J., **Meshkinnejad, R.**, & Mohsenzadeh, Y. *Effects of Neuromodulation-Inspired Mechanisms on the Performance of Deep Neural Networks in a Spatial Learning Task*.	iScience	2023
Shamsipour, P., Kourkounakis, T., Aghaee, A., **Meshkinnejad, R.**, Zaveri, M., & Hood, S. (2022). *Beyond Stationary Simulation; Modern Approaches to Stochastic Modelling*. arXiv preprint arXiv:2208.02875.	Under Review	2022
Meshkinnejad, R., Karimi, A., & Soleymani Baghshah, M., *DoubleA: Flexible and Controllable Embedding Augmentation for Large PreTrained Language Models*	Unpublished	2020