Université de Montréal

IIRC: Incremental Implicitly-Refined Classification

par

Mohamed Abdelsalam

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en Informatique

May 04, 2021

Résumé

Nous introduisons la configuration de la "Classification Incrémentale Implicitement Raffinée / Incremental Implicitly-Refined Classification (IIRC)", une extension de la configuration de l'apprentissage incrémental des classes où les lots de classes entrants possèdent deux niveaux de granularité, c'est-à-dire que chaque échantillon peut avoir une étiquette (label) de haut niveau (brute), comme "ours", et une étiquette de bas niveau (plus fine), comme "ours polaire". Une seule étiquette (label) est fournie à la fois, et le modèle doit trouver l'autre étiquette s'il l'a déjà apprise. Cette configuration est plus conforme aux scénarios de la vie réelle, où un apprenant aura tendance à interagir avec la même famille d'entités plusieurs fois, découvrant ainsi encore plus de granularité à leur sujet, tout en essayant de ne pas oublier les connaissances acquises précédemment. De plus, cette configuration permet d'évaluer les modèles pour certains défis importants liés à l'apprentissage tout au long de la vie (lifelong learning) qui ne peuvent pas être facilement abordés dans les configurations existantes. Ces défis peuvent être motivés par l'exemple suivant: "si un modèle a été entraîné sur la classe ours dans une tâche et sur ours polaire dans une autre tâche; oubliera-t-il le concept d'ours, déduira-t-il à juste titre qu'un ours polaire est également un ours ? et associera-t-il à tort l'étiquette d'ours polaire à d'autres races d'ours ?" Nous développons un benchmark qui permet d'évaluer les modèles sur la configuration de l'IIRC. Nous évaluons plusieurs algorithmes d'apprentissage "tout au long de la vie" (lifelong learning) de l'état de l'art. Par exemple, les méthodes basées sur la distillation sont relativement performantes mais ont tendance à prédire de manière incorrecte un trop grand nombre d'étiquettes par image. Nous espérons que la configuration proposée, ainsi que le benchmark, fourniront un cadre de problème significatif aux praticiens.

Mots Clés: Apprentissage Automatique, Apprentissage Profond, Apprentissage Tout Au Long De La Vie, Apprentissage Continu, Apprentissage Progressif, Vision Par Ordinateur.

Abstract

We introduce the "Incremental Implicitly-Refined Classification (IIRC)" setup, an extension to the class incremental learning setup where the incoming batches of classes have two granularity levels. i.e., each sample could have a high-level (coarse) label like "bear" and a low-level (fine) label like "polar bear". Only one label is provided at a time, and the model has to figure out the other label if it has already learned it. This setup is more aligned with real-life scenarios, where a learner usually interacts with the same family of entities multiple times, discovers more granularity about them, while still trying not to forget previous knowledge. Moreover, this setup enables evaluating models for some important lifelong learning challenges that cannot be easily addressed under the existing setups. These challenges can be motivated by the example "if a model was trained on the class *bear* in one task and on *polar bear* in another task, will it forget the concept of *bear*, will it rightfully infer that a *polar bear* is still a *bear*? and will it wrongfully associate the label of *polar bear* to other breeds of *bear*?". We develop a standardized benchmark that enables evaluating models on the IIRC setup. We evaluate several state-of-the-art lifelong learning algorithms and highlight their strengths and limitations. For example, distillation-based methods perform relatively well but are prone to incorrectly predicting too many labels per image. We hope that the proposed setup, along with the benchmark, would provide a meaningful problem setting to the practitioners.

Keywords: Machine Learning, Deep Learning, Lifelong Learning, Continual Learning, Incremental Learning, Computer Vision.

Contents

List of tables

List of figures

List of Abbreviations

AGEM	Average Gradient Episodic Memory
ANN	Artificial Neural Networks
BCE	Binary Cross Entropy
BiC	Large Scale Incremental Learning / Bias Correction
CIFAR	Canadian Institute for Advanced Research
CIL	Class Incremental Learning
CNN	Convolutional Neural Network
DL	Deep Learning
ER	Experience Replay
GEM	Gradient Episodic Memory
iCaRL	Incremental Classification and Representation Learning

iid	Independent and Identically Distributed
IIRC	Incremental Implicitly Refined Classification
JS	Jaccard Similarity
LUCIR	Learning a Unified Classifier via Rebalancing
ML	Machine Learning
OOD	Out Of Distribution
pw-JS	precision weighted Jaccard Similarity
SGD	Stochastic Gradient Descent
TIL	Task Incremental Learning

Acknowledgements

First, I would like to thank my supervisor, Sarath Chandar, for his continuous guidance and support throughout my degree. Sarath has been a great mentor, he was always available for insightful discussion and feedback, from which I have learned a lot. I am grateful to have had this opportunity of working with him.

Second, I would like to thank my main collaborators throughout my degree, Mojtaba Faramarzi and Shagun Sodhani, and also Prasanna Parthasarathi, with whom I had the opportunity to work on other interesting side projects. I would like also to acknowledge all the lab members, with whom we have shared lots of discussions, insights, and fun.

Finally, I would like to thank my family, and especially my mother, who have provided me with unconditional support and care, and without their support I would have never reached this stage.

Chapter 1

Introduction

Machine learning is a broad field of research that focuses on building systems which learn how to map a set of inputs to a set of outputs in order to solve a specific task. Since most useful tasks are too complex to directly craft such mappings, and with the increase in the size of data available for such tasks, there is a growing trend in data-driven machine learning, where a system learns the mapping function directly from the data. An example for such tasks is the task of insurance premium estimation, where the machine needs to estimate the insurance premium that needs to be paid for a specific client given the available data about that client. Another example is the task of automatically detecting brain tumours given the magnetic resonance imaging (MRI) and computed tomography (CT) scans. A third example is the task of machine translation, where the machine needs to read a piece of text in some language, and output the text in some other language. These examples show how these tasks can very in nature and in complexity.

During the twentieth century, extensive research was done in two direction in machine learning, the first one is now dubbed classical machine learning, and the other one is now dubbed deep learning . Classical machine learning assumes that each problem is unique and requires a unique representation that can only be guaranteed by specialized algorithms, and that includes a lot of manual feature engineering to provide the algorithms with the most useful features instead of the raw data. Classical machine learning includes powerful algorithms such as SVMs [Hearst, 1998] and Random Forests [Breiman, 2001]. On the other hand, deep learning assumes that the artificial neural networks (ANNs) are powerful enough to learn the required representations. ANNs apply a cascade of projection operations on the input, which makes them able to learn highly complex mappings and reduce the need for manual feature engineering, given that enough data is available. For the most part, classical machine learning algorithms used to be the more practical choice for most tasks, since they are less computationally expensive, their decisions are more interpretable, and they worked

better on smaller datasets. In brief, they were the more rational choice because, simply, they worked.

The effect of classical methods remained limited in scope due to their reliance on feature engineering to perform properly. This kind of feature engineering can be prohibitive when it comes to some forms of data with extremely high dimensional nature (images, videos, text, etc).

As deep learning algorithms do not need this high degree of feature engineering, and with the advances in the available computational power, and the exponential increase in the amount of data available, deep learning started to gain more ground during the last twenty years. Deep learning algorithms have led to transformational breakthroughs in computer vision [Deng et al., 2009, He et al., 2016], natural language processing [Hochreiter and Schmidhuber, 1997, Vaswani et al., 2017], speech processing [Amodei et al., 2016, Baevski et al., 2020], reinforcement learning [Mnih et al., 2015, Silver et al., 2017], robotics [Akkaya et al., 2019, Hafner et al., 2019], recommendation systems [Cheng et al., 2016, He et al., 2017] etc. On several tasks, deep learning models have either matched or surpassed human performance. However, such *super-human* performance is still limited to some very narrow and well-defined setups. These setups roughly follow the following progression:

(1) A large amount of training data is collected for whatever task we want the model to solve, this training data is assumed to be drawn in an independently and identically distributed (iid) manner from the same distribution of the unseen data.

(2) The model is trained on this data using an iterative optimization algorithm such as stochastic gradient descent (SGD). In doing so, the model has to perform several passes (epochs) over the training data till it reaches some satisfasctory low value on the training loss.

(3) The model is deployed to solve the target task, and it is expected to perform well if the training and testing distributions proved to be similar (as per the initial assumption).

If there is not enough data available for a specific task, then the power of deep learning as a way of learning meaningful representations allows for training of the model on a similar task, for which we have access to sufficient data. Then this model can be finetuned on the small data of the original task [Tan et al., 2018].

Tom M. Mitchell Definition for Machine Learning [Mitchell, 1997]

«A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E. »

Tom M. Mitchell provides a definition for machine learning in Box 1. This definition is particularly relevant to deep learning, since more data usually translates to better performance. However, if the model is trained incrementally on this additional data in the absence of the older data, the model might overfit to this additional data. The problem is even more severe if this additional data does not follow the same distribution as the original data, or if it corresponds to newer classes, as in this case the model usually forgets what it has learned in the original data, a phenomenon which is known as catastrophic forgetting.

Let's imagine a specific realistic scenario to explain this problem more concretely. We have a model that we would like deploy in order to classify cars based on the car model, and we have sufficient data to train such model. However, new car models always keep coming out, which necessitates that we keep augmenting the knowledge of our model with these new car models, so as to preserve its capacity to differentiate between the different car models.

A seemingly easy solution for this scenario is to use the initial training data corresponding to the older car models, in addition to the new data corresponding to the newer car models, to finetune or retrain our model. However, it is easy to spot the problems associated with this approach. The first problem is what if the initial data was not available anymore. This can be due to one of many reasons, most importantly of which would be for memory constraints or due to privacy restrictions that do not allow for keeping all the data. The second problem would be the time constraint, where the time required for this finetuning/retraining would keep increasing as long as the car models keep increasing.

Another alternative solution would be to finetune the model only on the new data corresponding to the new car models. However, *catastrophic forgetting* happens in this case, as the new information starts to interfere with the previously learned information, which usually leads to a drop in the performance of the model on the previously introduced classes/tasks.

These challenges are broadly studied under the domain of Lifelong Learning [Thrun and Mitchel, 1995], also called Incremental Learning [Schlimmer and Granger, 1986], Continual Learning [Thrun and Mitchel, 1997], and Never Ending Learning [Mitchell et al., 2018]. In the general lifelong learning setup, the model experiences new knowledge, in terms of new tasks, from the same domain or different domains. The model is expected to learn and solve new tasks while retaining useful knowledge from previous tasks. Humans can continually learn and accumulate knowledge over their lifetime, while the current learning algorithms are known to suffer from several challenges when training over a sequence of tasks [Chaudhry et al., 2018, Goodfellow et al., 2013, McCloskey and Cohen, 1989, Sodhani et al., 2020].

There are two popular paradigms in lifelong learning [van de Ven and Tolias, 2018]: **i)** *task incremental* learning, where the model has access to a task delimiter (say a *task id*),

which distinguish between tasks. Models for this setup are generally multi-headed, where there exists a separate classification layer for each task. **ii)** *class incremental learning*, where the model does not have access to a task delimiter, so it needs to discriminate between all classes from all tasks at inference time. Therefore, models developed for this paradigm are generally single-headed. The class incremental setup is more closely aligned with the real-life scenarios and is more challenging than the task incremental scenario.

1.1. Contributions

In this thesis, we try to extend the class incremental learning setup to a more challenging setup (which we call *IIRC*, or incremental implicitly refined classification). *IIRC* setup allows for exploring a breadth of challenges which could be faced by any model that is required to continue learning after deployment, and which cannot be examined using the current lifelong learning setups.

The main contributions of the thesis are as follows:

(1) We propose the *Incremental Implicitly-Refined Classification (IIRC)* setup, where the model starts training with some coarse, high-level classes and observes new, fine-grained classes as it trains over new tasks. Only the labels that belong to the current task are provided during training, and hence the model needs to use what it acquired during previous tasks to recognize if the samples also belong to a higher-level class in addition to the fine-grained class. During the lifetime of the model, it may encounter a new sample or an old sample with another label (the fine-grained label).

(2) We provide a standardized benchmark to evaluate lifelong learning algorithms in the *IIRC* setup. We adapt the commonly used ImageNet and CIFAR datasets, and provide the benchmark setup compatible with several major deep learning frameworks (PyTorch and Tensorflow)[1].

(3) We evaluate well-known lifelong learning algorithms on the benchmark, and highlight their strengths and limitations, while ensuring that the models are compared in a fair and standardized setup.

1.2. Thesis Outline

The chapters will proceed as follows: Chapter 2 will explain what is lifelong learning, what are the current state of the art methods, and how do they try to overcome the challenges of the current lifelong learning setups. Chapter 3 explains our proposed setup and benchmark, the

[1] `https://chandar-lab.github.io/IIRC/`

datasets and how they were modified, our introduced metric for evaluating the performance, and the baselines compared and how the different methods were adapted for use with this setup. Chapter 4 presents the experiments and their results, and compares between the performance of the different methods and what makes each method performs the way it does. Finally, chapter 5 presents the conclusion and discusses what are the promising future directions.

Chapter 2

Background

In this chapter, we shall start by the basic definitions for supervised learning, deep learning, and lifelong learning. We shall see what are the current available setups for lifelong learning, and why deep learning fails in the context of lifelong learning. Finally, we shall explore how the different approaches try to deal with this problem.

2.1. Supervised Learning

Machine Learning is synonymous with learning from data, where there exists some data from the past, and this data is to be used to recognize patterns and predict the future. Machine Learning is used as a tool that helps in making the process of decision making more informed and more autonomous. Supervised learning is a machine learning paradigm which is characterized by the availability of labeled data for whatever task we want to create a machine learning model for, and this is the branch that we are interested in for the rest of this thesis. Building a supervised learning model usually proceeds in the following steps:

(1) Collecting labeled data in the form of input-output pairs, where the input corresponds to what the model is expected to encounter, and the output corresponds to what the decision that the model is expected to take. The input can be in its raw format, or it can be represented by a set of hand-crafted features.

(2) This data is divided into a training, validation and test splits. The training split is used for training the model, the validation split is used for evaluating the model during the development stage, and the test split is used for the final evaluation of the model before deployment.

(3) A algorithm is picked to be used for training. Many algorithms can be tried, and the final algorithm can be chosen using the validation split.

(4) The optimal model given a set of data can be obtained analytically using a closed form solution for some of these algorithms (ex. Linear Regression). However, finding

a closed form solution is usually intractable, and hence iterative optimization techniques are employed (ex. gradient descent). Some of these algorithms are convex, and hence a global minimum is guaranteed (ex. SVM), and others are non-convex but it is usually sufficient to find a good local minimum (ex. neural networks).

In supervised learning, we would like to learn a function $f : \mathcal{X} \to \mathcal{Y}$ that is able to predict a target vector $y \in \mathcal{Y}$, when given an input sample $x \in \mathcal{X}$ (where x can be in raw form, or in the form of a curated set of features for the raw input). To do so, we have access to some training data $D = \{(x_i, y_i)_{i=1}^n\}$, which consists of n pairs of input samples $x_i \in \mathcal{X}$ and their corresponding target vectors $y_i \in \mathcal{Y}$, and that we assume are drawn independently and identically distributed (iid) from a fixed distribution $P(x,y)$.

In order to train this function f, we use some loss function L that captures how much the function prediction $\hat{y} = f(x)$ is different from the ground truth y given a sample x. The risk associated with this function becomes:

$$R(f) = \mathbb{E}_{x,y\sim P}[L(f(x), y)]\,, \tag{2.1}$$

and hence the optimal function f^* is the function that minimize this risk:

$$f^* = \arg\min_f R(f)\,, \tag{2.2}$$

However, since the distribution P is unknown, the risk R cannot be computed. As an alternative, the Empirical Risk Minimization (ERM) principle [Vapnik, 1991] is usually used, which seeks to obtain the optimal function $\hat{f}$ that minimizes the empirical risk $\hat{R}$

$$\hat{R}(f) = \frac{1}{n}\sum_{i=1}^{n} L(f(x_i), y_i)\,, \tag{2.3}$$

$$\hat{f} = \arg\min_f \hat{R}(f)\,, \tag{2.4}$$

2.2. Deep Learning

Although any algorithm can be used with *IIRC*, we shall be limited in scope to neural networks (deep learning). Neural networks (Figure 2.1) are networks that consist of a cascade of n layers, where each layer $\mathbf{h}_l$ is composed of nodes $\mathbf{h}_{l,:}$, and each node $\mathbf{h}_{l,j}$ (the j-th node in the l-th layer) is a non-linear transformation of the previous layer $\mathbf{h}_{l-1}$

$$\mathbf{h}_l = g_l(\mathbf{h}_{l-1}; \theta_l) = \sigma(\theta_l \mathbf{h}_{l-1})\,, \tag{2.5}$$

$$f(x;\theta) = g_n \circ g_{n-1} \ldots g_2 \circ g_1(x)\,, \tag{2.6}$$

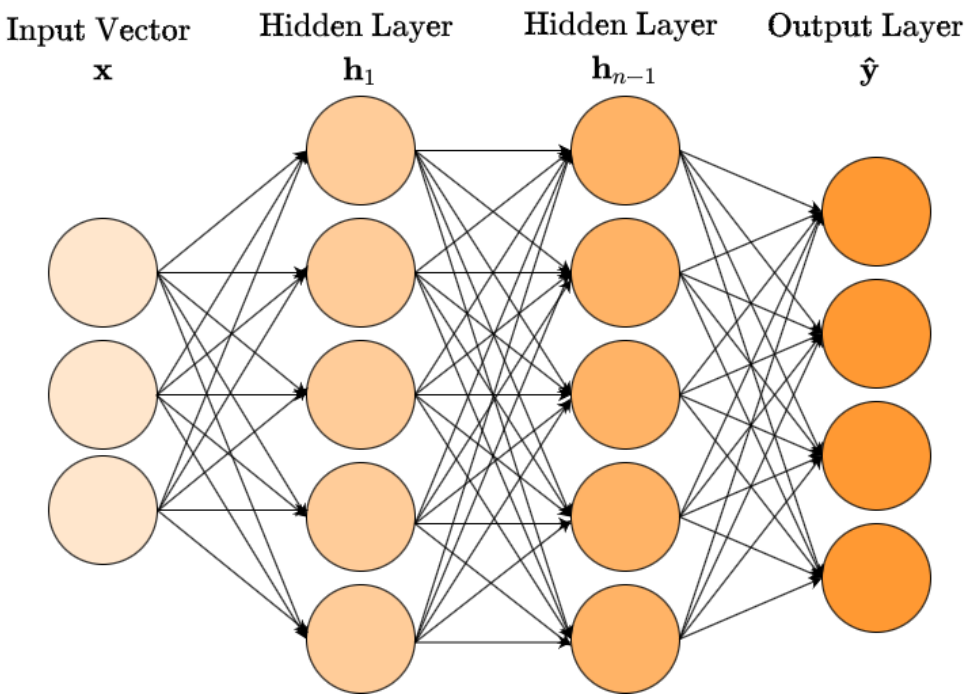


Fig. 2.1. Illustration of a fully connected neural network. In this network, each node in a layer l is connected to all the nodes of the previous layer $l-1$ by a linear transformation followed by a nonlinear activation function. This cascade of nonlinear layers is what gives neural networks their ability to learn complex representations

where σ is an element-wise non-linear function (sigmoid, RelU, etc.), θ_l is the matrix of weights for layer l (where each row represents a node), and the bias term was omitted for brevity.

This form of neural networks is known as fully connected network (Figure 2.1). Another form of neural networks more suitable for dealing with images is known as Convolutional Neural Networks (CNN) (Figure 2.2), where each layer is composed of channels, and each of these channels are constructed by convolving its filter/kernel over the channels of the previous layer.

Although a single layer (Equation 2.5) is a simple function, the fact that it is a non-linear function, and that a neural network model f is a cascade of such functions, makes f a highly complex nonlinear function that is capable of learning extremely complex patterns. Moreover, the setup of neural networks, where layers are built upon layers, helps the network learn more meaningful representation/features, where the earlier layers learn more finegrained details of the input (for example, the edges present in an image), and the later layers capture more coarse concepts (for example, the object present in an image), see Figure 2.3. This

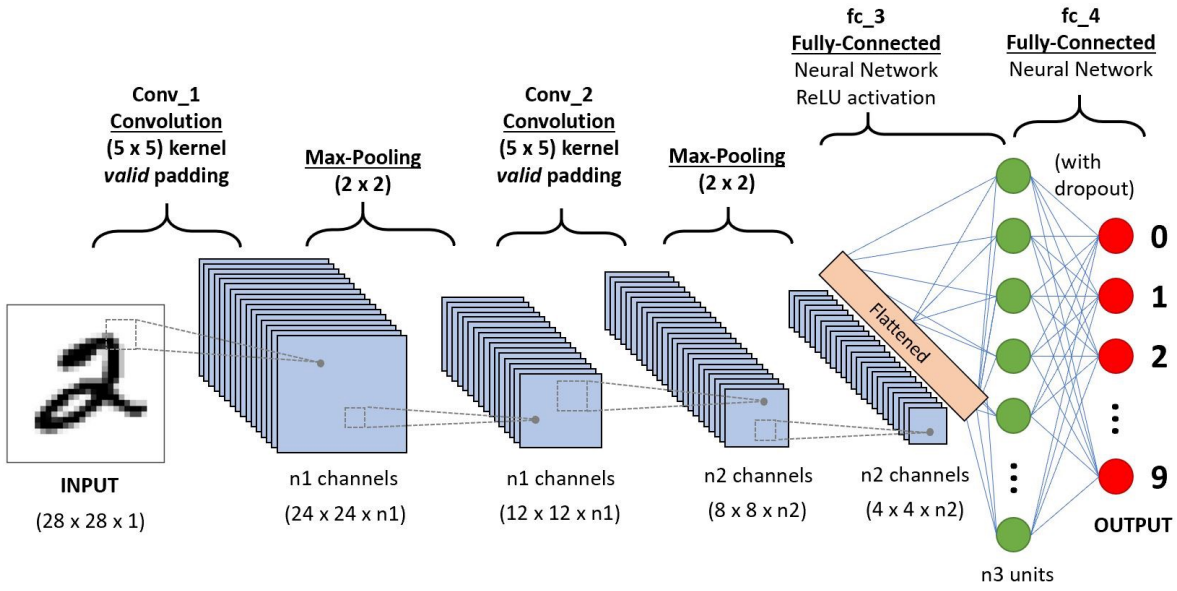


Fig. 2.2. Illustration of how a typical CNN performs an image classification task. A CNN is more suitable for dealing with images, since it takes into account the spatial information, and allowes weight sharing which decreases the complexity. Source[1]

limits the need for the manual handpicking of useful features, especially when dealing with structured inputs such as images, where the raw pixels themselves are not useful but rather how the pixels are organized together.

The high complexity of a neural network model gives it a high capacity, but it also makes it more difficult to deal with analytically. Fortunately, simple optimization methods, which only have convergence guarantees in the case of convex problems, still perform decently in the case of neural networks. Gradient descent and its variants are considered the most popular choice for optimizing neural networks. Gradient descent is an iterative optimization algorithm, where the variables to be optimized take a step after each iteration in the direction that minimizes the objective function (in this case, the empirical risk $\hat{R}$).

$$\theta_{new} = \theta_{old} - \epsilon \nabla_\theta \hat{R}\,, \tag{2.7}$$

where ϵ represents the step size (learning rate). This update rule will cease to take any more steps if $\nabla_\theta \hat{R}$ is equal to zero across all the dimensions, which only happens in the case of stationary points. It is also guaranteed to converge in the case of a convex functions (if

[1]https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

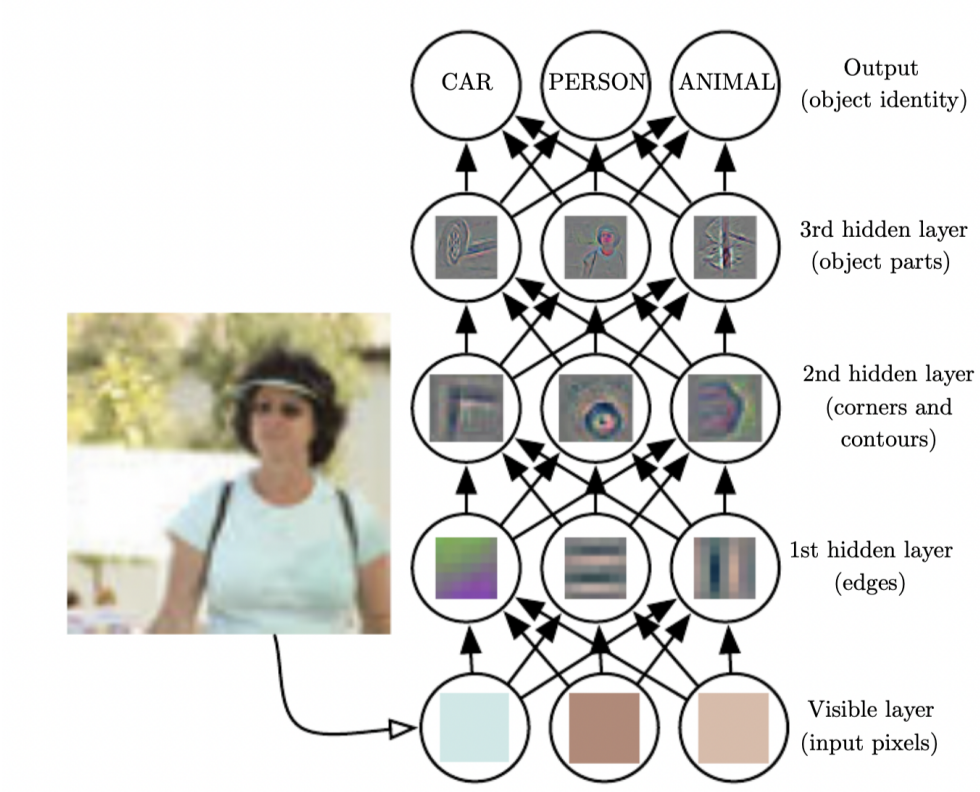


Fig. 2.3. Shallow layers learn more fine-grained information about the input sample (like edges). As each layer builds upon the output of the previous layer, deeper layers are able to combine this finegrained information into more sophisticated information, and consequently deeper layers are able to learn more abstract concepts (like object parts). Source:[Goodfellow et al., 2016]

$\hat{R}$ is convex in terms of θ) given that a suitable step size is used. As neural networks are highly non convex with respect to weights, it can be extremely difficult to reach a global minimum, or even a local minimum. However, it is not usually required to reach a minimum for the neural networks to perform well in most realistic settings, as achieving a relatively low value for the loss is usually sufficient. However, for this to be the case, the training set $\mathcal{D}$ over which $\hat{R}$ is calculated needs to be drawn iid from the test distribution $P(x,y)$, and the gradient descent algorithm would need to do several passes/epochs over this training set. Since this is not the case in lifelong learning, due to its incremental nature as we would see in the next section, it should be evident why an optimization technique like gradient descent would lead to catastrophic forgetting if left unregularized.

2.3. Lifelong Learning

Lifelong Learning is a broad, multi-disciplinary, and expansive research domain with several synonyms: Incremental Learning [Schlimmer and Granger, 1986], Continual Learning [Thrun and Mitchel, 1997], and Never Ending Learning [Mitchell et al., 2018]. In lifelong learning, the data is not available at once, but rather the model has to train sequentially on a continuous stream of tasks/concepts and their associated data. In an optimal scenario, the model should learn the new concepts without the need to retrain on the previous concepts and without the need to have access to the previous data. Moreover, as the model is trained on more concepts, it has more understanding of the world, and hence it should be able to learn new concepts faster and more efficiently, and it should be able as well to curate its understanding of the previous concepts and improve its performance on them. Finally, the model's capacity should not be limited so as to limit its ability to learn new concepts. In the lifelong learning literature, these ideas are referred to by the following terms:

(1) Catastrophic Forgetting: The deterioration of the model's performance on older tasks as it learns new tasks.
(2) Forward Transfer: The ability to learn newer tasks faster and more efficiently.
(3) Backward Transfer: The increase in the model's performance on older tasks as it learns newer tasks.
(4) Capacity Saturation: The inability of the model to learn new tasks.

Although these four concerns are all important for a machine learning model to be considered intelligent, neural networks are especially prone to catastrophic forgetting (see Figure 2.4 for an illustration), due to what is known as the stability-plasticity dilemma [Abraham and Robins, 2005]. Plasticity is the ability of the model to unreservedly change its weights to be able to acquire new tasks, but then large changes in weights will make these weights irrelevant to older tasks, and hence catastrophic forgetting happens. On the other hand, stability is to keep the weights of the network stable and prevent them from making large changes, but then this will limit the model's ability to learn new tasks (which can be seen as capacity saturation).

Different lifelong methods try to tackle these challenges from different perspectives, but let's first explain the most common lifelong learning setups, as some of these methods are more tailored to a specific setup.

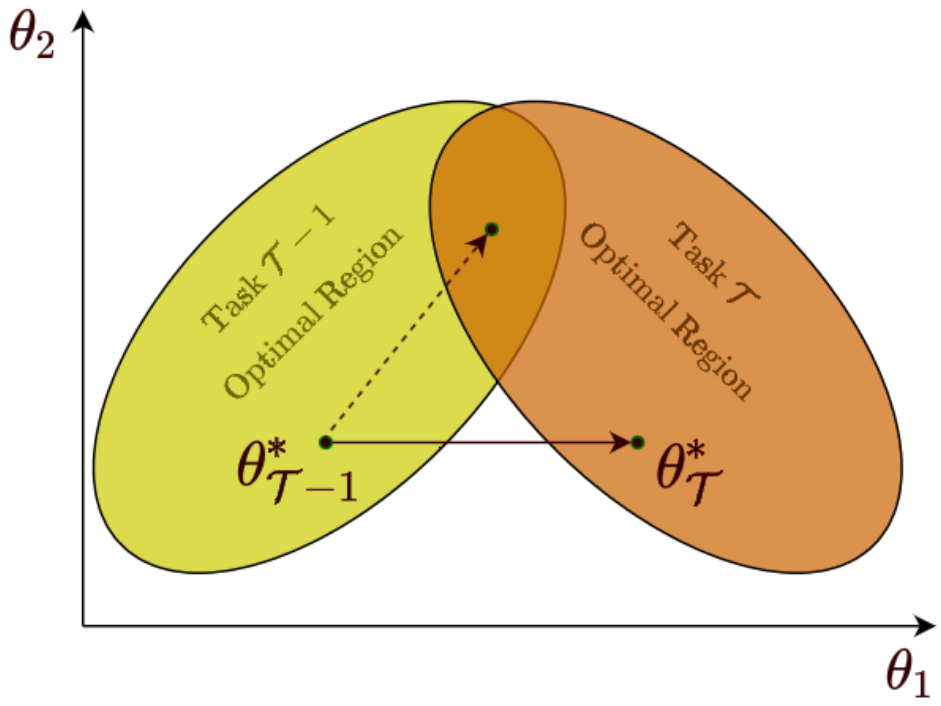


Fig. 2.4. During training on task $\mathcal{T}-1$, the optimal model parameters $\theta^*_{\mathcal{T}-1}$ reach an optimal region where the loss is minimal for task $\mathcal{T}-1$. During training on task $\mathcal{T}$, the optimal model parameters $\theta^*_{\mathcal{T}}$ reach an optimal region where the loss is minimal only for task $\mathcal{T}$. Although, for sufficiently large models, there exists regions of overlap between the optimal region for task $\mathcal{T}-1$ and the optimal region for task $\mathcal{T}$, the model typically does not reach there if not trained to do so (for example, by joint training). This is known as *catastrophic forgetting*.

2.3.1. Lifelong Learning Setups

Lifelong Learning has three main setups (following the definitions in [van de Ven and Tolias, 2018], which are widely adapted in the literature):

(1) Task-incremental Learning (TIL): In this setup, the model is always given the task id during training and during evaluation, which means that the model is only required to differentiate between the classes within its task. Hence, a multi-headed architecture is typically used within this setup, where the tasks share most of the network, and each task has its own output layer/head. For example, if the first task is $\{0, 1\}$ and the second task is $\{2, 3\}$, then when the model encounters a new sample that belongs

to 0, it is told that it belongs to the first task and hence it has only to identify whether it is 0 or 1.

(2) Domain-incremental Learning (DIL): In this setup, the model is not given the task id neither during training nor during evaluation, but it is not required to know it either. Therefore, a single head architecture is typically used. So if we apply the previous example to this setup, then the model does not need to know whether an input sample belongs to the first or the second task, as it only needs to predict whether a new sample belongs to the first output node (0 or 2), or to the second output node (1 or 3).

(3) Class-incremental Learning (CIL): In this setup, the set of classes keeps expanding and the model has to identify which class among all the classes an input sample belongs to. To say in a way more aligned with the previous definitions, the task id is only given during training, and the model has to identify during evaluation both the task id and the class within this task. An expanding single-head is usually used with this setup. So if we apply the previous example to this setup, then if the first task is $\{0, 1\}$ and the second task is $\{2, 3\}$, and the test input belongs to 0, then the model has to predict whether it belongs to 0 or 1 or 2 or 3.

IIRC is more related to CIL than to the other setups, hence let us define how the CIL setup differs from the supervised learning setup in more concrete terms.

In CIL setup, there exists a sequence of tasks, or episodes, where each task t represents a set of unique classes $\mathcal{C}^{(t)}$, where $\mathcal{C}^{(t)} \subseteq \mathcal{Y}$ (the set of all possible classes), and $\mathcal{C}^{(a)} \cap \mathcal{C}^{(b)} = \Phi \quad \forall a, b \in W$ if $a \neq b$. Each task t comes as well with its set of data $D^{(t)} = \{(x_i, y_i)_{i=s}^{s+n_t}\}$, where $x_i \in \mathcal{X}$ and $y_i \subseteq \mathcal{C}^{(t)}$. In incremental class learning, the output space $\mathcal{Y}^{(\mathcal{T})}$ keeps expanding whenever a new task $\mathcal{T}$ is introduced $\mathcal{Y}^{(\mathcal{T})} = \bigcup_{t=1}^{\mathcal{T}} \mathcal{C}^{(t)}$, the goal is still to learn the function that maps the input to output space across all the previous tasks $f_{\mathcal{T}} : \mathcal{X} \to \mathcal{Y}^{(t)}$. Applying the ERM principle as is would lead us to the following equation:

$$\hat{R}_{\mathcal{T}}(f) = \frac{1}{\mathcal{T}} \sum_{t=1}^{\mathcal{T}} \frac{1}{|D^{(t)}|} \sum_{(x_i, y_i) \in D^{(t)}} L(f(x_i), y_i)\,, \tag{2.8}$$

$$\hat{f}_{\mathcal{T}} = \arg\min_{f} \hat{R}_{\mathcal{T}}(f)\,, \tag{2.9}$$

However, as the data from older tasks $t < \mathcal{T}$ is not available anymore, calculating the risk this way becomes infeasible. On the other hand, minimizing the risk on only the data available will lead to good performance on the current task only, and catastrophic forgetting

of the previous tasks. We shall explain in the next sections how the different currently state-of-the-art methods try to deal with this issue.

2.3.2. Lifelong Learning Methods

Lifelong learning methods take different approaches when dealing with the challenges presented in Section 2.3, they can be divided into these three broad categories:

(1) replay based methods

(2) regularization based methods

(3) parameter isolation based methods

Parameter isolation based methods are a family of methods which use different sub-networks for different tasks, either by using a mask that decides which parameters to be used for each task, or by adding a new sub-network for each new task with connections to the older sub-networks. Due to the way parameter isolation methods work, they need access to a task identifier, making them a good fit for the TIL setup. For this reason, we will not use any parameter isolation methods in this work. Prominent methods that follow this approach include *Progressive Neural Networks* [Rusu et al., 2016], *Piggyback* [Mallya et al., 2018], *PackNet* [Mallya and Lazebnik, 2018], *HAT* [Serra et al., 2018], *TFM* [Masana et al., 2020b], *DAN* [Rosenfeld and Tsotsos, 2018], *PathNet* [Fernando et al., 2017].

Regularization based methods can be further categorized into two categories: importance based regularization and distillation based regularization. The way importance based regularization works is by adding a regularization term in the objective function that constrains the parameters of the network during training on a new task from deviating too much from the optimal parameters obtained for the previous task. This constraint is selective in the sense that changes in more important parameters are more penalized than changes in less important parameters. Each method defines its way of measuring that importance, but simplifications of the Fischer Information Matrix are usually used as a proxy for the importance of the parameters (For example, Kirkpatrick et al. [2017] uses the diagonal terms of the Fischer Information Matrix). Among the most prominent works in this direction are *Elastic Weight Consolidation (EWC)* [Kirkpatrick et al., 2017], *Synaptic Intelligence (SI)* [Zenke et al., 2017], *Memory Aware Synapses (MAS)* [Aljundi et al., 2018], and *Riemannian Walk* [Chaudhry et al., 2018]. Although importance based regularization methods are not limited by definition to TIL setup the way parameter isolation based methods are, they still tend to perform very poorly in the CIL setup compared to replay based methods [Masana et al., 2020a].

Distillation based regularization was introduced by *Learning Without Forgetting (LWF)* [Li and Hoiem, 2017], and it is considered as an extension to the idea of knowledge distillation introduced by Hinton et al. [2015]. The idea is basically to keep a snapshot of the model trained on the last task ($f_{\mathcal{T}-1}$), and whenever a new task is introduced, the output of this frozen snapshot ($f_{\mathcal{T}-1}$) is used as soft targets during the training of ($f_{\mathcal{T}}$) on the new task (this happens for the output on the base classes shared between the two models, not the new classes introduced in the new task). The assumption is that the samples of the newer task might provide a poor sampling for the older tasks, and hence teaching the model to give the same output it used to give for older tasks might limit their catastrophic forgetting. *Learning Without Memorization (LWM)* [Dhar et al., 2019] extends *LWF* [Li and Hoiem, 2017] by adding an attention distillation loss so that the new model ($f_{\mathcal{T}}$) produces similar attention maps as the old model ($f_{\mathcal{T}-1}$) for the classes that belong to old tasks. Many replay based methods, to be discussed, are built upon the concepts of *LWF* [Li and Hoiem, 2017].

As for the replay based approaches, *iCaRL* [Rebuffi et al., 2017] extends *LWF* by making use of a replay buffer, and by separating the training and evaluation phases. *iCaRL* selects exemplars for the replay buffer using the herding strategy, and alleviates catastrophic forgetting by using distillation loss during training, and using a nearest-mean-of-exemplars classifier during inference. *EEIL* [Castro et al., 2018] modifies *iCaRL* by learning the feature extractor and the classifier jointly in an end to end manner. *LUCIR* [Hou et al., 2019] applies the distillation loss on the normalized latent space rather than the output space, proposes to replace the standard softmax layer with a cosine normalization layer, and uses a margin ranking loss to ensure a large margin between the old and new classes. *BIC* [Wu et al., 2019] tries to overcome the bias of the network towards newer classes by adding a bias correction layer that is trained on a balanced subset of the data in a separate stage after each task. Other works include *LGM* [Ramapuram et al., 2017], *IL2M* [Belouadah and Popescu, 2019], and *ER* [Rolnick et al., 2019]. *GEM* [Lopez-Paz and Ranzato, 2017] is another replay-based method, which solves a constrained optimization problem. It uses the replay buffer to constrain the gradients on the current task by projecting them in a direction so that the loss on the previous tasks does not increase. Although *GEM* is still a replay based method, it can be also regarded as a regularization based method, and it has only been shown to perform well in the incremental task setup. *AGEM* [Chaudhry et al., 2019a] improves over *GEM* by relaxing some of the constraints, and hence increasing the efficiency, while retaining the performance. Finally, Chaudhry et al. [2019b] shows that vanilla experience replay, where

the model simply trains on the replay buffer along with the new task data, is by itself a very strong baseline.

In this thesis, we include adapted versions of *iCaRL*, *LUCIR*, *BiC*, *AGEM*, and vanilla experience replay as baselines. Hence we shall explain them in more details in the next section.

2.4. Lifelong Learning Baselines

2.4.1. Experience Replay (ER)

The most naive approach for training a model in a continual fashion is to keep finetuning the model whenever a new task is introduced, but as mentioned earlier, this would cause catastrophic forgetting, since the loss is being minimized only on the current task data $D^{(\mathcal{T})}$, and there is no way for the model to know what it should preserve from before and what it should not. Naive finetuning would be equivalent to finding the optimal model $f_{\mathcal{T}}$ that minimizes the objective with respect to the current task only:

$$\theta^*_{\mathcal{T}} = \arg\min_{\theta} \frac{1}{|D^{(\mathcal{T})}|} \sum_{(x_i,y_i)\in D^{(\mathcal{T})}} L(f(x_i;\theta), y_i)\,, \tag{2.10}$$

Since training a neural network model would usually be done using an iterative optimization algorithm like gradient descent, the only effect that the parameters trained on task $\mathcal{T}-1$ ($\theta^*_{\mathcal{T}-1}$) would have on this optimization process is that they will act as the starting point for the training trajectory of the parameters ($\theta_{\mathcal{T}}$).

The most basic solution to this problem that can help alleviate catastrophic forgetting, is to keep some samples $M^{(c)}$ for each class c learnt so far in a replay buffer $\mathcal{M}$, and replay them along with the current task data $D^{(\mathcal{T})}$. Hence equation 2.10 becomes:

$$\theta^*_{\mathcal{T}} = \arg\min_{\theta} \frac{1}{|\mathcal{D}|} \sum_{(x_i,y_i)\in\mathcal{D}} L(f(x_i;\theta), y_i), \text{ where } \mathcal{D} = D^{(\mathcal{T})} \cup \mathcal{M}\,, \tag{2.11}$$

[Chaudhry et al., 2019b] shows that, as simple as this baseline seems, it performs exceedingly well compared to other more sophisticated algorithms.

Replay buffers can be either of a total fixed size (m), and hence after any task t, the samples kept per any class c ($M^{(c)}$) would be of size $|M^{(c)}| = \frac{m}{|\mathcal{Y}^{(t)}|}$. This type of buffers make sense if the only constraints that prevents keeping all the previous data are memory and time constraints. The other type of replay buffer is the buffer with a fixed number of samples per class $|M^{(c)}| = \bar{m}$. Hence, the total size of the replay buffer after any task t is basically $\bar{m} \times |\mathcal{Y}^{(t)}|$. This buffer makes more sense if the constraints that prevents keeping all

the previous data are also privacy related constraints. For the rest of the thesis, the buffer used is the buffer with a fixed number of samples per class, unless otherwise stated.

The samples in the buffer are usually picked either randomly, or by an approach called herding, which shall be explained in the next subsection.

2.4.2. Incremental Classifier and Representation Learning (iCaRL)

iCaRL [Rebuffi et al., 2017] was among the first deep learning methods to use exemplar rehearsal in order to alleviate the catastrophic forgetting in the class incremental learning setup. Most of the other methods developed for class incremental learning are one way or another related to *iCaRL*, which makes it an important baseline to consider.

iCaRL is based upon three concepts, namely:

(1) Learning representations using knowledge distillation and experience replay

(2) Choosing exemplars based on the herding approach

(3) Classification at test time using the nearest-mean-of-exemplars

So the first part is the same as in experience replay, keeping samples in the buffer for all the observed classes, and replaying them along with the data that belongs to incoming classes. However, the different aspect here is using knowledge distillation [Hinton et al., 2015] when replaying these samples. This idea is building upon the contribution of Li and Hoiem [2017] (*LWF*), and it goes as follows: whenever a new set of classes are introduced, the target for the older classes is not the ground truth label, but rather the output of the older model, which means that the classification loss is only applied to the new classes, while the distillation loss is applied to the older classes to make sure that the output of the current model is close to the output of the previous model. *iCaRL* uses the binary cross entropy loss for classification, hence the resulting loss becomes:

$$L(f(x;\theta),y;\theta^*_{\mathcal{T}-1}) = -\sum_{c\in\mathcal{C}^{(\mathcal{T})}} [\delta_{c=y}\log f_c(x;\theta) + \delta_{c\neq y}\log\left(1-f_c(x;\theta)\right) - \sum_{c\in\mathcal{Y}^{(\mathcal{T}-1)}} f_c(x;\theta^*_{\mathcal{T}-1})\log f_c(x;\theta) + (1-f_c(x;\theta^*_{\mathcal{T}-1}))\log\left(1-f_c(x;\theta)\right)], \quad (2.12)$$

The second element in *iCaRL* is how to choose the exemplars. As mentioned before, *iCaRL* keeps a specific number of exemplars (k) per new class in its replay buffer. However, instead of choosing the exemplars randomly, they are chosen so that they their mean approximates the class mean in the latent space, which is called the herding approach. The intuition here is that if they approximate the class mean in the latent space, then they are a better representation of the whole class than a randomly sampled set of exemplars. So let's

divide a neural network model $f(x;\theta)$ into a feature function $g(x;\theta_{l-1})$ and a classification layer $h(x;\theta_l)$, such that $f(x) = h(g(x))$ (we shall omit the dependence on θ except when needed, for brevity). Then the latent space of a sample x becomes its representation $g(x)$, and the class mean for class c becomes:

$$\mu_c = \frac{1}{|D^{(c)}|} \sum_{(x_i,y_i)\in D^{(c)}} g(x_i) \tag{2.13}$$

where if class c belongs to task t, then $D^{(c)}$ is the subset of $D^{(t)}$ with $y_i = c$. Hence in herding, the exemplars are added sequentially in a greedy manner to M^c to best approximate μ_c:

$$M_j^c = \underset{\substack{x \\ (x,y)\in D^{(c)} \\ x\notin M^c}}{\arg\min} \left|\left|\mu_c - \frac{1}{j}[g(x) + \sum_{i=1}^{j-1} g(M_i^c)]\right|\right|, \tag{2.14}$$

It has to be noted that μ and g are normalized before being used in this equation.

The final part in *iCaRL* is that, although they use a classification layer h during training, they use a nearest-mean-of-exemplars approach during evaluation, where the class mean is approximated by the mean of the class exemplars in the latent space. Therefore, for any test sample x observed after task $\mathcal{T}$, the model prediction $\hat{y}$ is obtained as follows:

$$\hat{y} = \underset{y\in\mathcal{Y}^{(\mathcal{T})}}{\arg\min} \left|\left|g(x) - \frac{1}{|M^{(y)}|} \sum_{x_i\in M^{(y)}} g(x_i)\right|\right| \tag{2.15}$$

2.4.3. Learning a Unified Classifier Incrementally via Rebalancing (LUCIR)

LUCIR [Hou et al., 2019] is another replay based method that has proved to be an effective method in the context of class incremental learning. It tries to deal with the imbalance between the number of samples for older classes (the samples in the replay buffer) and newer classes. It claims that this imbalance has the following effects (see Figure 2.5):

(1) Imbalanced Magnitudes: The class embeddings (weight vectors) of the newer classes are significantly larger than those of the older classes.

(2) Deviation: The relationship between the class embeddings of the older classes and the feature vectors of the samples that belong to these classes keeps deviating when learning newer tasks.

(3) Ambiguities: the class embedding of newer classes are close to those of older classes, leading to ambiguities.

In order to mitigate these effects, *LUCIR* proposes applying the following three techniques for each of the three effects respectively:

(1) Cosine Normalization: This ensures that the magnitudes of the class embeddings are balanced across both the older and newer classes.
(2) Less-Forget Constraint: This helps in preserving the geometric configuration of the older classes.
(3) Inter-Class Separation: This ensures the existence of a margin that separates older and newer classes, reducing the ambiguities.

So the first technique is cosine normalization. The result of a node i that belongs to the last layer l is given by $h_i(x) = \theta_{l,i}^T g(x) + b_{l,i}$, which is followed by a softmax or a sigmoidal unit to give the class probability. This tends to favor newer classes as they have higher magnitudes

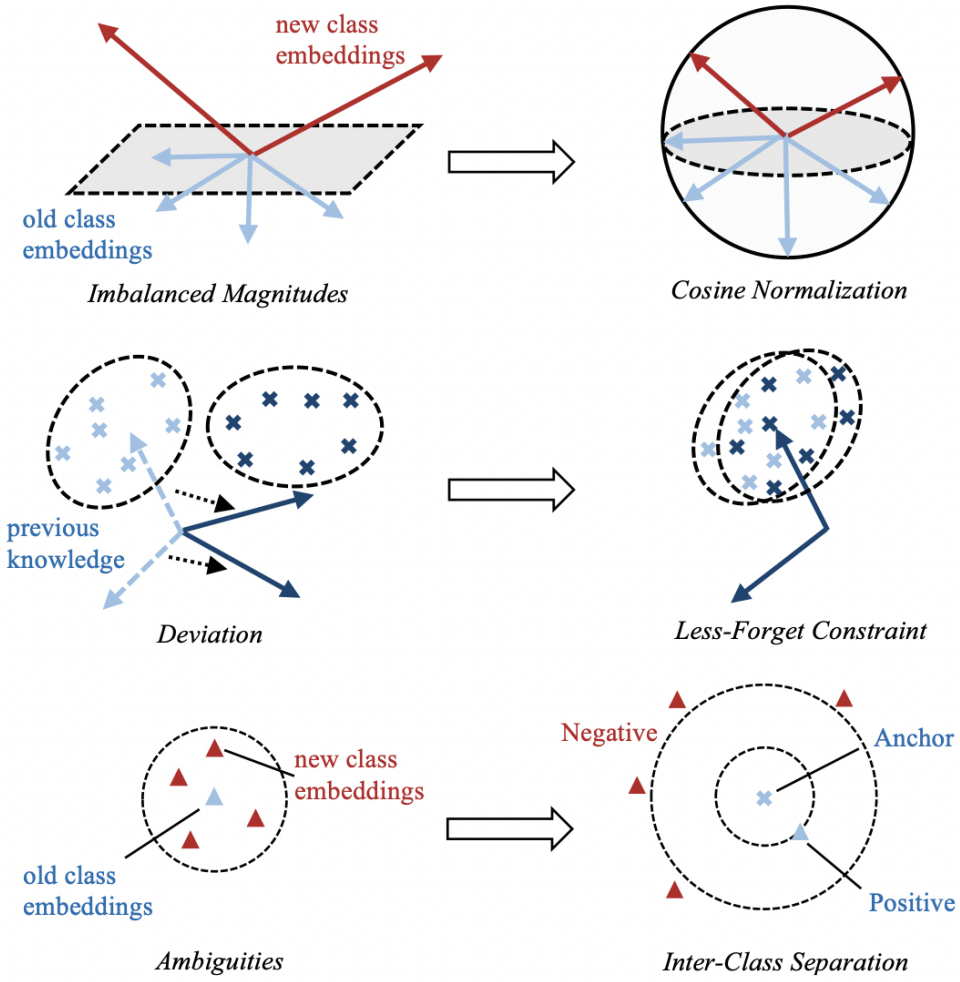


Fig. 2.5. Illustration of how the imbalance between the older and newer classes affect the learned embeddings in the incremental learning setting, and how *LUCIR* tries to tackle these effects. Source:[Hou et al., 2019]

for their weights and biases. To address this issue, *LUCIR* uses a cosine normalization in the last layer

$$h_i(x) = \eta \langle \bar{\theta}_{l,i}, \bar{g}(x) \rangle \,, \tag{2.16}$$

where $\bar{v} = \frac{v}{||v||^2}$, $\langle v_1, v_2 \rangle = ||v_1||||v_2|| \cos\alpha$, and α is the angle between v_1 and v_2, hence, $\langle \bar{v}_1, \bar{v}_2 \rangle = \cos\alpha$. Since $\cos\alpha$ is constrained in the range [−1,1], η is introduces as a learnable multiplier which is shared across all the output nodes. This way, the magnitudes of the classification nodes of both old and new class are within the same range.

The second technique is the less-forget constraint, which is a stronger distillation loss than what was presented in *iCaRL*. In *iCaRL*, the distillation is performed on the level of the model output $f(x)$, but in *LUCIR*, they propose fixing the weights of the last layer corresponding to the older classes, and applying the distillation on the level of the latent space features $g(x)$:

$$L_{dis}(x) = 1 - \langle \bar{g}(x; \theta_{:l-1}), \bar{g}(x; \theta^*_{\mathcal{T}-1:l-1}) \rangle \,, \tag{2.17}$$

Although this is a stronger regularization as is, it becomes even stronger since *LUCIR* is fixing the weights of the older classes nodes $\theta_{:l,:m} = \theta^*_{\mathcal{T}-1:l-1,:m}$, where m is the number of older classes $m = |\mathcal{Y}^{(\mathcal{T}-1)}|$. It has to be noted that L_d is constrained in the range [0,2]. L_{dis} is weighted by a multiplier λ, which increases its weight as the number of observed classes to new classes increases:

$$\lambda = \lambda_{base} \sqrt{\frac{|C^{\mathcal{T}}|}{|\mathcal{Y}^{(\mathcal{T}-1)}|}} \,, \tag{2.18}$$

The third and final technique is adding a margin ranking loss, which increases the inter-class separation between the older classes and newer classes, so that it becomes insufficient to just classify the fewer samples of the older classes correctly, but to also keep a distance from the newer classes. So for a given sample x, where $x \in M^{(c)}$, we have

$$L_{mr}(x, c) = \sum_{k \in K} \max(0, m - \langle \bar{\theta}_{l,c}, \bar{g}(x) \rangle + \langle \bar{\theta}_{l,k}, \bar{g}(x) \rangle) \,, \tag{2.19}$$

Where $K \subseteq \mathcal{C}^{(\mathcal{T})}$ represent the top K newer classes which have the highest output scores h for the sample x, and m is the margin distance.

Hence, the final loss for *LUCIR* becomes:

$$L = \frac{1}{|\mathcal{D}|} \sum_{(x_i, y_i) \in \mathcal{D}} L_{classification}(f(x_i), y_i) + \lambda L_{dis}(x_i) + \frac{1}{|\mathcal{M}|} \sum_{(x_j, y_j) \in \mathcal{M}} L_{mr}(x_j, y_j) \,, \tag{2.20}$$

where $\mathcal{D} = D^{\mathcal{T}} \cup \mathcal{M}$, and $\mathcal{T}$ is the current task. *LUCIR* uses the herding approach for filling the replay buffer.

2.4.4. Large Scale Incremental Learning (BiC)

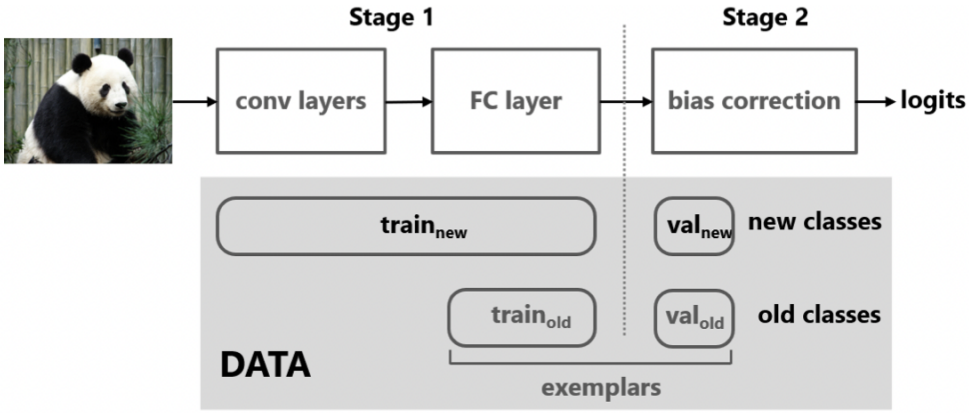


Fig. 2.6. In *BiC*, there exists two training stages per task. The first stage is the stage done by all the replay based methods, where the model is trained on the new data along with the existing replay buffer (using distillation from previous model). The second stage is a bias correction stage, where two balanced validation sets (not used in the first stage) from the replay buffer and the new training data are used to tune a bias correction layer. Source:[Wu et al., 2019]

Large Scale Incremental Learning (BiC) [Wu et al., 2019] is an extension to *LWF* [Li and Hoiem, 2017] and *iCaRL* [Rebuffi et al., 2017]. It is borrowing from these methods the concept of knowledge distillation between the teacher model $f_{\mathcal{T}-1}$ and the student model $f_{\mathcal{T}}$, and keeping a replay buffer $\mathcal{M}$ which is filled using the herding approach. The main differentiator for this method is its emphasis on the concept of bias correction (hence the name *BiC*). So *BiC* reasons that the main challenge that prevents lifelong learning methods from performing well on large sequences of classes comes to two coupled factors, namely:

- The imbalance in the training data between the number of samples available for the older classes, and the number of samples available for the newer classes.
- As the model observes more classes, the probability of the presence of visually similar classes becomes higher, which means many visually similar classes would have smaller margins around the decision boundary, and these margins would be especially susceptible to the problem of data imbalance.

To deal with this problem, *BiC* proposes the introduction of a bias correction layer, which is a layer of two parameters. The tuning of this layer takes place in a separate stage after the training on the task is done (see Figure 2.6).

After each task is introduced, *BiC* is first trained using a baseline very common with *iCaRL*, where there is a classification loss and a distillation loss. The classification loss is a

softmax cross entropy:

$$L_{classification}(f(x;\theta),y) = -\sum_{c\in\mathcal{Y}^{(\mathcal{T})}} \delta_{c=y} \log f_c(x;\theta)\,, \tag{2.21}$$

while the distillation loss is formulated as:

$$L_{dis}(f(x;\theta),y;\theta^*_{\mathcal{T}-1}) = -\sum_{c\in\mathcal{Y}^{(\mathcal{T})}} f_c(x;\theta^*_{\mathcal{T}-1}) \log f_c(x;\theta)\,, \tag{2.22}$$

with $f_c(x;\theta) = \frac{e^{q_c(x;\theta)}}{\sum_{\hat{c}\in\mathcal{Y}^{(\mathcal{T})}} e^{q_{\hat{c}}(x;\theta)}}$.

The two losses are combined as follows:

$$L = \frac{1}{|\mathcal{D}|} \sum_{(x_i,y_i)\in\mathcal{D}} \lambda L_{dis}(x_i) + (1-\lambda) L_{classification}(f(x_i),y_i)\,, \tag{2.23}$$

where $\mathcal{D} = D^{\mathcal{T}} \cup \mathcal{M}$, and $\mathcal{T}$ is the current task. λ is a scalar that is calculated according to the following formula $\lambda = \frac{|\mathcal{Y}^{(\mathcal{T}-1)}|}{\mathcal{Y}^{(\mathcal{T})}}$, hence it gives more importance for the distillation loss as the number of observed classes increases.

After this training on the task data is done, a second phase of training (bias correction phase) takes place on a special validation set, this validation set is created by merging a subset from the training set with another subset from the buffer, and it is not used during the first training phase. This validation set is balanced as it has an equal number of samples per class for all the classes in $\mathcal{Y}^{(\mathcal{T})}$. This bias correction takes place by applying a linear model to the logits $o_c(x;\theta)$ of the new classes $c \in \mathcal{C}^{(\mathcal{T})}$.

$$q_c(x;\theta^*_{\mathcal{T}}) = \begin{cases} o_c(x;\theta^*_{\mathcal{T}}) & c \in \mathcal{Y}^{(\mathcal{T}-1)} \\ \alpha o_c(x;\theta^*_{\mathcal{T}}) + \beta & c \in \mathcal{C}^{(\mathcal{T})} \end{cases} \tag{2.24}$$

The parameters α and β are shared among by all the new classes $c \in \mathcal{C}^{(\mathcal{T})}$, and they are trained using the softmax cross entropy loss.

2.4.5. Averaged Gradient Episodic Memory (*AGEM*)

Averaged Gradient Episodic Memory (*AGEM*) [Chaudhry et al., 2019a] lies at the intersection of replay based methods, where they keep a replay buffer of samples from previous tasks, and regularization based methods, where they do not minimize the loss on the replay buffer directly, but they use it to regularize the direction of the gradient on the current task $\mathcal{T}$. *AGEM* (and *GEM* [Lopez-Paz and Ranzato, 2017]) were mainly introduced in the context of incremental task learning, where *AGEM* was proposed as a computationally more efficient version of *GEM*. *GEM* is a constrained optimization method, which uses a replay

buffer $\mathcal{M} = \cup_{t<\mathcal{T}} M^{(t)}$ to constrain the gradients of the loss on the current task $D^{(\mathcal{T})}$, so as to update the model parameters $\theta_{\mathcal{T}}$ in a direction that would not interfere with the performance on the previous tasks.

$$\mathcal{L}_D(\theta) = \frac{1}{|D|} \sum_{(x_i,y_i)\in D} L(f(x_i;\theta), y_i)\,, \tag{2.25}$$

$$\theta^*_{\mathcal{T}} = \arg\min_{\theta} \mathcal{L}_{D^{(\mathcal{T})}}(\theta) \quad \text{s.t.} \quad \mathcal{L}_{M^{(t)}}(\theta) \leq \mathcal{L}_{M^{(t)}}(\theta^*_{\mathcal{T}-1}) \quad \forall t < \mathcal{T}\,, \tag{2.26}$$

GEM is a very computationally expensive approach that is not applicable to large-scale setups, since it requires that the loss on each of the previous tasks $t < \mathcal{T}$ (represented by $M^{(t)}$) does not increase, which requires computing the gradient using the whole replay buffer $\mathcal{M}$, as well as solving a quadratic programming problem. *AGEM* provides a more efficient version of GEM, by relaxing the constraints as it only requires that the overall loss on the previous tasks does not increase, which reduce the constraints from $\mathcal{T}-1$ constraints to a single constraint:

$$\theta^*_{\mathcal{T}} = \arg\min_{\theta} \mathcal{L}_{D^{(\mathcal{T})}}(\theta) \quad \text{s.t.} \quad \mathcal{L}_{\mathcal{M}}(\theta) \leq \mathcal{L}_{\mathcal{M}}(\theta^*_{\mathcal{T}-1})\,, \tag{2.27}$$

This can be solved using just an inner product between the gradients of $\mathcal{L}_{D^{(\mathcal{T})}}$ and $\mathcal{L}_{\mathcal{M}}$ instead of a quadratic program. It also uses a random batch from the replay buffer rather than the whole replay buffer.

Chapter 3

Incremental Implicitly Refined Classification

This thesis is mainly based on the following publication:

Mohamed Abdelsalam, Mojtaba Faramarzi, Shagun Sodhani, and Sarath Chandar. IIRC: Incremental implicitly-refined classification. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2021

My contribution starts with suggesting the idea based on a discussion among the four authors. The idea was then refined into a full standardized setup with its set of metrics based on discussions among the four authors. I was responsible for much of the coding, with some help from Mojtaba Faramarzi and Shagun Sodhani, and where Mojtaba Faramarzi was responsible for reviewing that the code is up to standard and without any faults. The experiments were mostly run by me and the Shagun Sodhani, and the results were discussed among the four authors. The writing of the paper was mainly done by me, with the help of the Mojtaba Faramarzi and Shagun Sodhani, and with constant feedback from Sarath Chandar. My supervisor, Sarath Chandar, was constantly advising and giving periodic feedback. I would also like to thank Mojtaba Faramarzi for his artistic contribution, as he is responsible for drawing Figure 3.1 and Figure 3.2, mostly by hand.

3.1. Introduction

Several useful benchmarks have been proposed for evaluating models in the lifelong learning setting [Antoniou et al., 2020, Lomonaco and Maltoni, 2017]. While useful for measuring high-level aggregate quantities, these benchmarks take a narrow and limited view on the broad problem of lifelong learning. One common assumption that many class incremental setups make is "information about a given sample (say label) can not change across tasks".

For example, an image of a bear is always labeled as "bear", no matter how much knowledge the model has acquired.

While this assumption appears to be "obviously correct" in the context of the supervised learning paradigm (where each sample generally has a fixed label), the assumption is not always satisfied in real-life scenarios. Humans often interact with the same entities multiple times and discover new information about them. Instead of invalidating the previous knowledge or outright rejecting the new information, they refine their previous knowledge using the new information. Figure 3.1 illustrates an example where a child may recognize all bears as "bear" (and hence label them as "bear"). However, while growing up, they may hear different kinds of bear being called by different names, and so they update their knowledge as: "Some bears are brown bears, some bears are polar bears, and other bears are just bears. Brown bears and polar bears are both still bears, but they are distinct". This does not mean that their previous knowledge was wrong (or that previous label "bear" was "incorrect"), but they have discovered new information about an entity and have coherently updated their knowledge. This is the general scheme of learning in humans.

A concrete instantiation of this learning problem is that two similar, or even identical, input samples can have two different labels across two different tasks. We would want the model to learn the new label, associate it with the old label without forgetting the old label. Evaluating lifelong learning models for these capabilities is generally outside the scope of existing benchmarks. We propose the *Incremental Implicitly-Refined Classification (IIRC)* setup to fill this gap. We adapt the publicly available CIFAR100 and ImageNet datasets to create a benchmark instance for the *IIRC* setup and evaluate several well-known algorithms on the benchmark. The goal is not to develop a new state-of-the-art model, but rather to surface the challenges posed by the *IIRC* setup, and how the different lifelong learning algorithms react to this setup, highlighting their strengths and limitations.

While the class incremental learning setup is a challenging and close to real-life formulation of the lifelong learning problem, most existing benchmarks do not explore the full breadth of the complexity of the problem. They tend to over-focus on catastrophic forgetting (which is indeed an essential aspect) at the expense of several other unique challenges to the lifelong learning paradigm. In this thesis, we highlight those challenges and propose the *Incremental Implicitly-Refined Classification (IIRC)* setting, an extension of the class incremental learning setting, that enables the studying of these under-explored challenges, along with the other well-known challenges like catastrophic forgetting. We provide an instantiation of the setup, in the form of a benchmark, and evaluate several well-known lifelong learning algorithms on it.

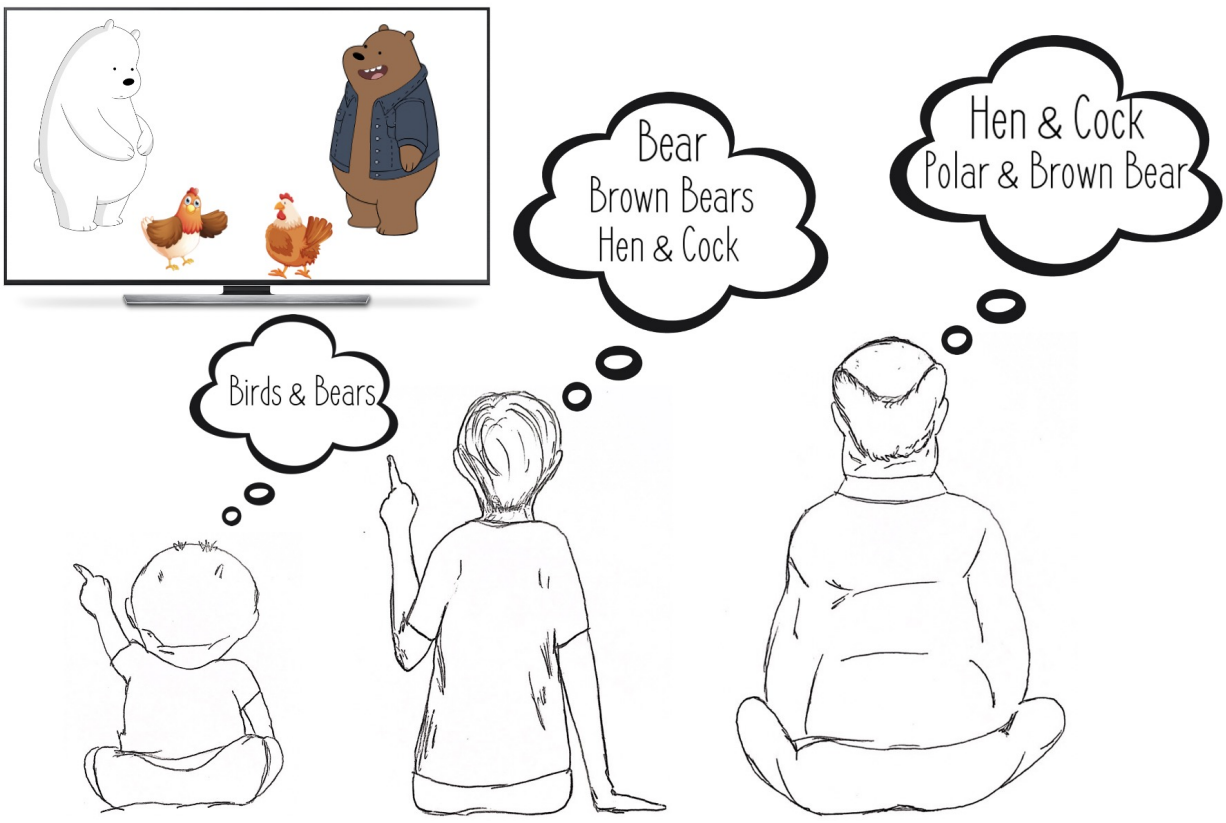


Fig. 3.1. Humans incrementally accumulate knowledge over time. They encounter new entities and discover new information about existing entities. In this process, they associate new *labels* with entities and refine or update their existing *labels*, while ensuring the accumulated knowledge is coherent.

3.2. Under-explored challenges in class incremental learning setting

In class incremental learning, the model encounters new classes as it trains over new tasks. The nature of the class distributions and the relationship between classes (across tasks) can lead to several interesting challenges for the learning model: If the model is trained on a *high-level* label (say "bear") in the initial task, and then trained on a *low-level* label, which is a refined category of the previous label (say "polar bear"), what kind of associations will the model learn and what associations will it forget? Will the model generalize and label the images of polar bear as both "bear" and "polar bear"? Will the model catastrophically forget the concept of "bear"? Will the model infer the spurious correlation: "all bears are polar bears"? What happens if the model sees different labels (at different levels of

granularity) for the *same sample* (across different tasks)? Does the model remember the latest label or the oldest label or does it remember all the labels? These challenges can not to be trivially overcome by removing restrictions on memory or replay buffer capacity (as we show in Section 4.2).

3.3. Terminology

We describe the terminology used in the paper with the help of an example. As shown in Figure 3.2, at the start, the model trains on data corresponding to classes "bear", "bus" and "dog". Training the model on data corresponding to these three classes is the first **task**. After some time, a new set of classes ("polar bear", "lamp" and "whippet") is encountered, forming the second task. Since "whippet" is a type of "dog", it is referred as the **subclass**, while "dog" is refereed as the **superclass**. The "dog-whippet" pair is referred to as the superclass-subclass pair. Note that not all classes have a superclass (example "lamp"). We refer to these classes as **orphan subclasses**. When training the model on an example of a "whippet", we may provide only "whippet" as the supervised learning label. This setup is referred to as the **incomplete information** setup, where if a task sample has two labels, only the label that belongs to the current task is provided. Alternatively, we may provide both "whippet" and "dog" as the supervised learning labels. This setup is referred as the **complete information** setup, where if a task sample has two labels, labels that belong to the current and previous tasks are provided. Note that majority of our experiments are performed in the incomplete information setup as that is closer to the real life setup, requiring the model to recall the previous knowledge when it encounters some new information about a known entity. We want to emphasize that the use of the word **task** in our setup refers to the arrival of a new batch of classes for the model to train on in a single-head setting, and so it is different from its use to indicate a distinct classification head in the task incremental learning setup.

As the model is usually trained in an *incomplete information* setup, it would need access to a validation set to monitor the progress in training that is still an *incomplete information* set, otherwise there would be some sort of labels leakage. On the other hand, after training on a specific task, the model has to be be evaluated on a *complete information* set, hence a *complete information* validation set is needed to be used during the process of model development and tweaking, so as to not overfit on the test set. We provide both in the benchmark, where we call the first one the **in-task validation set**, while the latter one the **post-task validation set**.

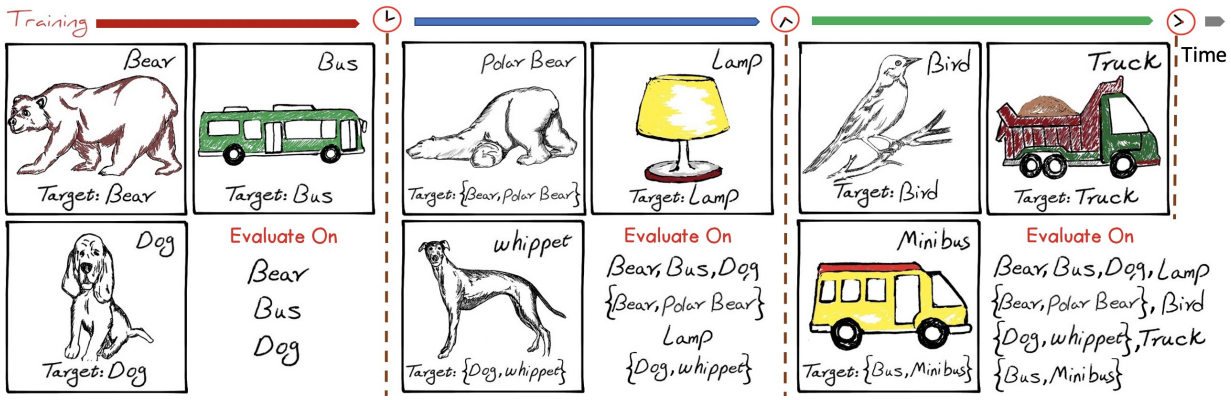


Fig. 3.2. *IIRC* setup showing how the model expands its knowledge and associates and re-associates labels over time. The top right label shows the label model sees during training, and the bottom label (annotated as "Target") is the one that model should predict during evaluation. The right bottom panel for each task shows the set classes that model is evaluated on and the dashed line shows different tasks.

3.4. Setup

We describe the high-level design of the *IIRC* setup (for a visual illustration, see Figure 3.2). We have access to a series of N tasks denoted as $T_1, \cdots, T_N$. Each task comprises of three collections of datasets for training, validation, and testing. The ground truth of each sample can have one or two labels associated with it. In the case of samples with two labels, one label is a subclass and the other label is a superclass. For any superclass-subclass pair, the superclass is always introduced in an earlier task, with the intuition that a high-level label should be relatively easier to learn. Moreover, the number of samples for a superclass is always more than the number of samples for each of its subclasses (the number of samples for a superclass increases linearly with the number of its subclasses, up to a limit). During training, we always follow the incomplete information setup, as it allows the studying of the challenges mentioned earlier. During the first task, only a subset of superclasses (and no subclasses) are used to train the model. The first task has more classes (and hence, samples) compared to the other tasks, as it can be seen as a kind of pretraining task. The subsequent tasks have a mix of superclasses, subclasses, and orphan subclasses. During the training phase, the model is evaluated on the in-task validation set (which follows the incomplete information setup), and during the evaluation phase, the model is evaluated on the post-task validation set and the test set (both follow the complete information setup).

In more formal terms, extending the formulation of the *CIL* setup in Section 2.3, in *IIRC*, there still exists a sequence of tasks, where each task has its unique set of classes $\mathcal{C}^{(t)} =$

$\{\ldots, p_m, \ldots, b_n, \ldots, c_l, \ldots\}$ and its associated data $D^{(t)} = \{(x_i, \tilde{y}_i)_{i=s}^{s+n_t}\}$, with $x_i \in \mathcal{X}$ and $\tilde{y}_i \in \mathcal{C}^{(t)}$. $p, b,$ and c represent superclasses, subclasses, and orphan subclasses respectively. However, $\tilde{y}_i^{(t)}$ does not represent the ground truth target for x_i anymore. There exists a ground truth $q_i \in \mathcal{Y}$ which is a superclass-subclass pair $q_i = \{p_a, b_b\}$, or a single orphan subclass $q_i = \{c_a\}$. $\tilde{y}_i = q_i \cap \mathcal{C}^{(t)}$ represents a piece of the ground truth which belongs to the current task. After a new task $\mathcal{T}$ is observed, the output space become $\mathcal{Y}^{(\mathcal{T})} = \bigcup_{t=1}^{\mathcal{T}} \mathcal{C}^{(t)}$, and the objective is to learn a function that predicts the part of the ground truth which belongs to the observed tasks so far $y_i^{\mathcal{T}} = q_i \cap \mathcal{Y}^{\mathcal{T}}$.

3.5. Benchmark

3.5.1. Dataset

We use two popular computer vision datasets in our benchmark - ImageNet [Deng et al., 2009] and CIFAR100 [Krizhevsky, 2009]. For both datasets, we create a two-level hierarchy of class labels, where each label starts as a leaf-node and visually similar labels are assigned a common parent. The leaf-nodes are the *subclasses* and the parent-nodes are the *superclasses*. Some of the subclasses do not have a corresponding superclass, so as to enrich the setup and make it more realistic. While the datasets come with a pre-defined hierarchy (e.g. ImageNet follow the WordNet hierarchy), we develop a new hierarchy as the existing hierarchy focuses more on the semantics of the labels and less on the visual similarity, for example, "sliding door" and "fence" are both grouped under "barriers" in WordNet hierarchy although they are not visually similar. We refer to these adapted datasets as *IIRC-ImageNet* and *IIRC-CIFAR*.

	With Duplicates		Without Duplicates			
Dataset	Train	In-task Validation	Train	In-task vVlidation	Post-task Validation	Test
IIRC-CIFAR	46,160	5,770	40,000	5,000	5,000	10,000
IIRC-Imagenet-full	1,215,123	51,873	1,131,966	48,802	51,095	49,900

Table 3.1. The number of samples for each split of the training set. "With Duplicates" represents the number of samples including the duplicates between some superclasses and their subclasses (the samples that the model see two times with two different labels). This does not happen for the post-task validation set and the test set, as they are in the complete information setup

In *IIRC-CIFAR*, each superclass has a similar number of subclasses (four to eight). On the other hand, the subclasses distribution in *IIRC-ImageNet* is much more skewed (see

Figure 3.3) as the number of sublcasses per superclass varies from 3 to 118. We explicitly decided not to *fix* this imbalance to ensure that visually similar classes are grouped together. Moreover, in real life, not all classes are observed at the same frequency, making our setup more realistic. More statistics about the *IIRC-CIFAR* and *IIRC-ImageNet* datasets are provided in Table 3.1, Table 3.2, and Table 3.3. Moreover, the full class hierarchies for both *IIRC-ImageNet* and *IIRC-CIFAR* are provided in AppendixB.

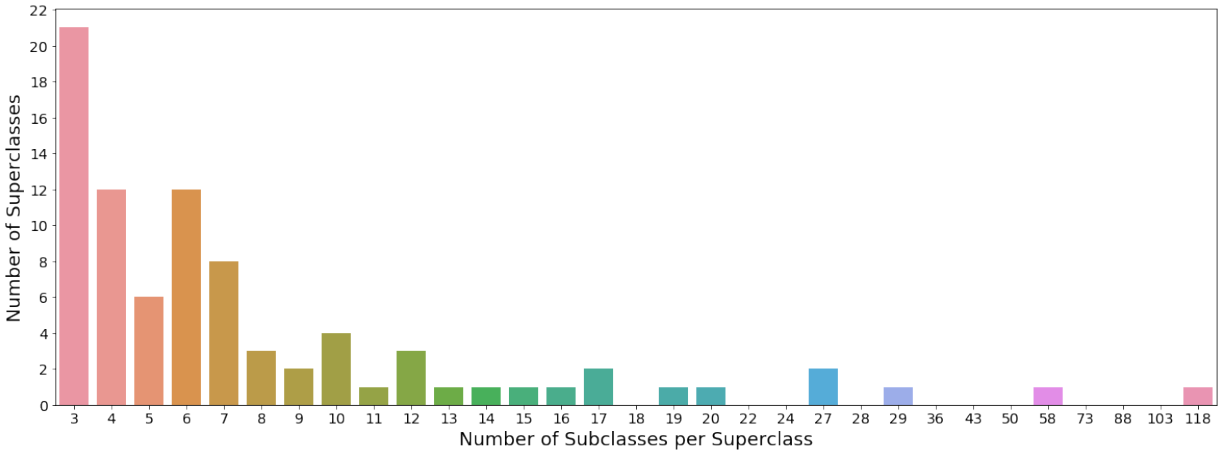


Fig. 3.3. The distribution of the number of subclasses per superclass in *IIRC-ImageNet*. Some superclasses have a large number of subclasses (like "dogs" which has 118 subclasses), while most superclasses have $3-8$ subclasses.

As mentioned in Section 3.4, we use two validation sets - one with incomplete information (for model selection and monitoring the model performance during training) and one with complete information (for the model evaluation after each task). Each validation dataset comprises 10% of the training data for CIFAR, and 4% of the training data for ImageNet, and is fixed through all the runs. Some aggregate information about the splits is provided in Table 3.1.

Dataset	Superclasses	Subclasses	Orphan Subclasses	Total Number of Classes
IIRC-CIFAR	15	77	23	115
IIRC-Imagenet-full	85	788	210	1083

Table 3.2. For each dataset, these are the number of superclasses, the number of subclasses that belong to these superclasses, the number of orphan subclasses that don't have a superclass, as well as the total number classes

Since we are creating the class hierarchy, superclasses do not have any samples assigned to them. For the training set and the in-task validation set, we assign 40% of the samples that

belong to each subclass to its superclass, while retain 80% of the samples for the subclass. This means that superclass-subclass pairs share about 20% of the samples. In other words, in 20% of the cases, the model observes the same sample with different labels (across different tasks). Since some superclasses have an extremely large number of subclasses, we limit the total number of samples in a superclass. A superclass with more than eight subclasses, uses $\frac{8}{\text{number of subclasses}} \times 40\%$ of the samples that belong to its subclasses. We provide the pseudo code for the dataloader in AppendixA.

Dataset	Superclass	Num of Subclasses	Superclass Size	Subclass	Subclass Size
IIRC-CIFAR	vehicles	8	1,280	bus	320
IIRC-CIFAR	small mammals	5	800	squirrel	320
IIRC-CIFAR	-	-	-	mushroom	400
IIRC-ImageNet	bird	58	3,762	ostrich	956
IIRC-ImageNet	big cat	6	2,868	leopard	956
IIRC-ImageNet	keyboard instrument	4	1,912	grand piano	956
IIRC-ImageNet	-	-	-	wooden spoon	1,196

Table 3.3. Several examples for different classes and the number of samples they have in the training set. The subclass on the right is a subclass that belongs to the superclass (if any) on the left. The left side is blank for orphan subclasses.

Now that we have a dataset with superclasses and subclasses, and with samples that belong to both kinds of classes, the tasks are created as follows: The first task is always the largest task with 63 superclasses for *IIRC-ImageNet*, and 10 superclasses for *IIRC-CIFAR*. In the subsequent tasks, each new task introduces 30 classes for *IIRC-ImageNet* and 5 classes for *IIRC-CIFAR*. Recall that each task introduces a mix of superclasses, subclasses, and orphan subclasses. *IIRC-ImageNet* has a total of 35 tasks, while *IIRC-CIFAR* has a total of 22 tasks. Since the order of classes can have a bearing on the models' evaluation, we create 5 preset class orders (called task configurations) for IIRC-ImageNet and 10 task configurations for *IIRC-CIFAR*, and the average (and standard deviation) of the performance on these configurations is reported in the experiments in Chapter 4.

Finally, we acknowledge that while IIRC-ImageNet provides interesting challenges in terms of data diversity, training on the dataset could be difficult and time consuming. Hence, we provide a shorter, lighter version which has ten tasks (with five tasks configurations). We shall call the original version *IIRC-ImageNet-full*, and the lighter version *IIRC-ImageNet-lite*, while referring to both collectively as IIRC-ImageNet. This lighter version is not supposed to replace the full version for benchmarking the model performance, it is just to make it easier

for others to perform quicker debugging experiments and to not overfit on *IIRC-ImageNet-full*. All the metrics on *IIRC-ImageNet-lite*are reported as well.

3.5.2. Metrics

Most lifelong learning benchmarks operate in the single-label classification setup, making accuracy the appropriate metric. In *IIRC* setup, the model should be able to predict multiple labels for each sample, even if those labels are seen across different tasks. We considered using the *Exact-Match Ratio* (MR) metric [Sorower, 2010], a multi-label extension of the accuracy metric. MR is defined as:

$$MR = \frac{1}{n}\sum_{i=1}^{n} I(Y_i == \hat{Y}_i)\,, \tag{3.1}$$

where I is the indicator function, $\hat{Y}_i$ are the set of (model) predictions for the i_{th} sample, Y_i are the ground truth labels, and n is the total number of samples. One limitation for the MR metric is that it does not differentiate between the partially incorrect predictions and the completely incorrect predictions.

Another popular metric for multi-label classification is the *Jaccard similarity*(JS), also called "intersection over union"[Sorower, 2010]. JS is defined as:

$$JS = \frac{1}{n}\sum_{i=1}^{n} \frac{|Y_i \cap \hat{Y}_i|}{|Y_i \cup \hat{Y}_i|}\,, \tag{3.2}$$

which, as opposed to MR, gives partial credit to the partially correct predictions. To further penalize the imprecise models, we *weight* the Jaccard similarity by the per sample precision (i.e., the ratio of true positives over the sum of true positives and false positives per sample). The motive behind this is that one of the main challenges in *IIRC* are whether the model can predict the correct associations between labels, and hence we wanted the performance metric to capture this aspect by further penalizing the imprecision of a model. We refer to this variant as the *precision-weighted Jaccard similarity (pw-JS)* metric. Section 4.3.1 shows a comparison between both metrics.

We measure the performance of a model on task k after training on task j using the precision-weighted Jaccard similarity, denoted R_{jk}, as follow:

$$R_{jk} = \frac{1}{n_k}\sum_{i=1}^{n_k} \frac{|Y_{ki} \cap \hat{Y}_{ki}|}{|Y_{ki} \cup \hat{Y}_{ki}|} \times \frac{|Y_{ki} \cap \hat{Y}_{ki}|}{|\hat{Y}_{ki}|}\,, \tag{3.3}$$

where $(j \geq k)$, $\hat{Y}_{ki}$ is the set of (model) predictions for the i_{th} sample in the k_{th} task, Y_{ki} are the ground truth labels, and n_k is number of samples in the task. R_{jk} can be used as a

proxy for the model's performance on the k^{th} task as it trains on more tasks (i.e. as the j increases).

We evaluate the overall performance of the model after training till task j, as the average *precision-weighted Jaccard similarity* over all the classes that the model has encountered so far. Note that during this evaluation, the model has to predict all the correct labels for a given sample, even if the labels were seen across different tasks (i.e. the evaluation is performed in the complete information setup). We denote this metric as R_j and computed it as follow:

$$R_j = \frac{1}{n} \sum_{i=1}^{n} \frac{|Y_i \cap \hat{Y}_i|}{|Y_i \cup \hat{Y}_i|} \times \frac{|Y_i \cap \hat{Y}_i|}{|\hat{Y}_i|}, \tag{3.4}$$

where n is the total number of evaluation samples for all the tasks seen so far.

3.6. Baselines

We evaluate several well-known lifelong learning baselines. We also consider two training setups where the model has access to all the labels for a given sample (complete information setup): **i)** *joint* where the model is jointly trained on all the classes/tasks at once and **ii)** *incremental joint* where as the model trains across tasks, it has access to all the data from the previous tasks in a *complete information* setup. In the **Finetune** baseline, the model continues training on new batches of classes without using any replay buffer. Vanilla **Experience Replay (ER)** method finetunes the model on new classes, while keeping some older samples in the replay buffer and rehearsing on them. **Experience Replay with infinite buffer (ER-infinite)** is similar to *incremental joint*, but in *incomplete information* setup as in ER. This means that if a new label is introduced that applies to an old sample, the target for that sample will be updated with that new label in the incremental joint baseline but not in the ER-infinte baseline . We also have **A-GEM** [Chaudhry et al., 2019a] that is a constrained optimization method in the replay-based methods category. It provides an efficient version of GEM [Lopez-Paz and Ranzato, 2017] by minimizing the average memory loss over the previous tasks at every training step. Another baseline is **iCaRL** [Rebuffi et al., 2017] that proposed using the exemplar rehearsal along with a distillation loss. **LUCIR** [Hou et al., 2019] is a replay-based class incremental method that alleviates the catastrophic forgetting and the negative effect of the imbalance between the older and newer classes. **BiC** is also a replay-based class incremental method that introduces a bias correction phase after each task to aleviate the bias towards newer classes. More details about the baselines can be found in Section-2.4.

3.6.1. Model Adaptations

The earlier-stated baselines were proposed for the single label *CIL* setup, while *IIRC* setup requires the model to be able to make multi-label predictions. Therefore, some changes have to be applied to the different models to make them applicable in the *IIRC* setup. To this end, we use a sigmoid unit followed by the binary cross-entropy loss (BCE) as the classification loss in all the baselines. This was already used in the *iCaRL* paper [Rebuffi et al., 2017], however, *LUCIR* [Hou et al., 2019], *BiC* [Wu et al., 2019], and *AGEM* [Chaudhry et al., 2019a] originally used a softmax unit followed by a cross-entropy loss. The BCE loss is averaged by the number of observed classes so that its range does not increase as the number of classes increases during training. During prediction, the output of the sigmoid unit is used and classes with values above a threshold (0.5 in our experiments) are considered as the model's predicted labels.

3.6.1.1. iCaRL. Although *iCaRL* [Li and Hoiem, 2017] uses a classification layer during training, it uses a nearest-mean-classifier during evaluation (see Section 2.4.2 for more details). However, using the nearest-mean-classifier strategy for classifying samples is not feasible for our setting, as the model should be able to predict a variable number of labels. To overcome this issue, we use the output of the classification layer, which was used during training, and call this variant as *iCaRL-CNN*. We further consider a variant of *iCaRL-CNN*, called *iCaRL-norm*, which uses a cosine normalization in the last layer. [Hou et al., 2019] suggests that using this normalization improves the performance in the context of incremental learning. Hence the classification score is calculated as:

$$p_i(x) = \sigma(\eta \langle \bar{\theta}_{l\,i}\,, \bar{g}(x) \rangle)\,, \tag{3.5}$$

where σ is the sigmoid function, $\bar{\theta}_{l\,i}$ are the normalized weights of the last layer that correspond to label i, and $\bar{g}(x)$ is the output of the last hidden layer for sample x. η is a learnable scalar that controls the peakiness of the sigmoid. It is important to have η since $\langle \bar{\theta}_{l\,i}\,, \bar{g}(x) \rangle$ is restricted to the range $[-1,1]$. We can either fix the η or consider it as a learnable parameter. We observed that learning η works better in practice.

3.6.1.2. BiC. In Equation 2.24 in the original *BiC* method [Wu et al., 2019], the bias correction parameters are only applied on the logits of the newer classes (see Section 2.4.4 for more details). These parameters are responsible for changing the magnitudes of these logits so as to balance the older and newer classes, and since a softmax is followed, these changes reflect on the output of all the classes (increasing the logits of newer classes increases the output of these classes and simultaneously decreases the output of older classes, and vice

versa). However, since we are using a sigmoid instead of a softmax (due to the presence of multiple labels), changes applied to the logits of newer classes would not reflect on the outputs of older classes. Hence, we replace Equation 2.24 by the following equation:

$$q_c(x;\theta^*_{\mathcal{T}}) = \begin{cases} \frac{o_c(x;\theta^*_{\mathcal{T}})}{\alpha} - \beta & c \in \mathcal{Y}^{(\mathcal{T}-1)} \\ \alpha o_c(x;\theta^*_{\mathcal{T}}) + \beta & c \in \mathcal{C}^{(\mathcal{T})} \end{cases} \tag{3.6}$$

This way, increasing the logits of newer classes still reflect on those of the older classes and vice versa. Another change from the original *BiC* is that *BiC* uses a separate balanced validation set for the bias correction phase. In our experiments, the buffer (after being updated with the new classes samples) is itself used for the bias correction phase to make it more aligned with the other baselines.

3.7. Summary

In this chapter, we introduced the *IIRC* setup along with its motivations, challenges, terminology, and metrics. We introduced as well the benchmark and how the CIFAR and ImageNet datasets were adapted to be used with *IIRC*, along with details about the hierarchy we proposed for them. We introduced the baselines that will be compared in the experiments in the next chapter, and how these baselines were modified to be used with the *IIRC* setup. In the next chapter, we shall discuss the experiments and their results, along with the different aspects associated with these experiments.

Chapter 4

Experiments and Discussion

In this chapter, the results of the different methods and baselines on *IIRC-CIFAR*, *IIRC-ImageNet-lite*, and *IIRC-ImageNet-full* are presented. We design our experimental setup to surface the challenges that lifelong learning algorithms face when operating in the *IIRC* setup. Our goal is neither to develop a new state-of-the-art model nor to rank the existing models. The aim is to highlight the strengths and weakness of the dominant lifelong learning algorithms, with the hope that this analysis will spur new research directions in the field.

All the plots in this chapter are based on the average of several runs corresponding to pre-specified class orders, as explained in Section 3.5.1, with 10 class orders in the case of *IIRC-CIFAR*, and 5 class orders for each of *IIRC-ImageNet-full* and *IIRC-ImageNet-lite*. The same figures are included in Appendix D with the standard deviation included in the plots. The reason for providing the two versions is that the plots with only the mean are less cluttered and easier to comprehend. The raw data used to plot the figures is provided as well in Appendix C for easier future comparisons.

4.1. Experimental Setup

All the models used are built upon the ResNet architecture [He et al., 2016], with ResNet-50 being used for *IIRC-ImageNet*, and the reduced ResNet-32 is used for *IIRC-CIFAR*.

The optimizer used is stochastic gradient descent (SGD) with momentum, as it performs better in the lifelong learning setups compared with the optimizers which use adaptive learning rates [Mirzadeh et al., 2020]. In all the experiments, the value of the momentum is set to 0.9, and a scheduler is used to decrease the learning rate by a factor of 10 upon plateau of the performance of on in-task validation subset that belongs to the current task. For the *IIRC-CIFAR* experiments, the starting learning rate is set to 1.0 for all the methods. For the *IIRC-ImageNet* experiments, the starting learning rate is set to 0.5 in the case of *iCaRL-CNN*, *iCaRL-norm*, and *BiC*, and 0.1 in the case of *finetune*, *ER* and, *LUCIR*. The number of training epochs per task is 140 for *IIRC-CIFAR*, and 100 for *IIRC-ImageNet*,

with the first task always trained for double the number of epochs due to its larger size. The minibatch size is set to 128, and the weight decay parameter is set to $1e-5$. Moreover, the *AGEM* memory batch size, which is used to calculate the reference gradient, is set to 128. For *LUCIR*, the margin distance m is set to 0.5 (Equation 2.19), and λ_{base} is set to 5 (Equation 2.18). All the hyperparameters were tuned based on the validation performance in experiments that include a short sequence that includes only the first four tasks. The intuition behind that is that in real-life, we never know in advance how many tasks the model will encounter, hence it does not make sense to do the hyperparameter tuning based on the whole sequence of tasks.

During training, data augmentations are applied as follows:

- for the two *IIRC-ImageNet* variants, a random crop of size (224×224) is sampled from the image, then a random horizontal flip is applied, and finally the pixels are normalized by a precalculated per-channel mean and standard deviation.
- for *IIRC-CIFAR*, a padding of size 4 is added to each edge, then a random crop of size (32×32) is sampled, followed by a random horizontal flip, then the pixels are normalized.

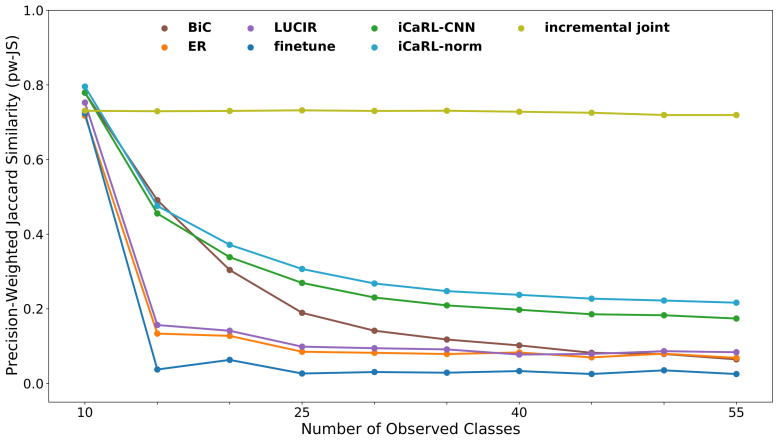
Replay buffers with a fixed number of samples per class were used (20 samples per class, except otherwise indicated). Hence, the capacity increases linearly as the model learns more classes. These samples are chosen randomly in the case of *ER* and *AGEM*, and using the herding approach in the case of *iCaRL-CNN*, *iCaRL-norm*, *LUCIR*, and *BiC* (refer to Chapter 2 for more details).

4.2. Results and Discussion

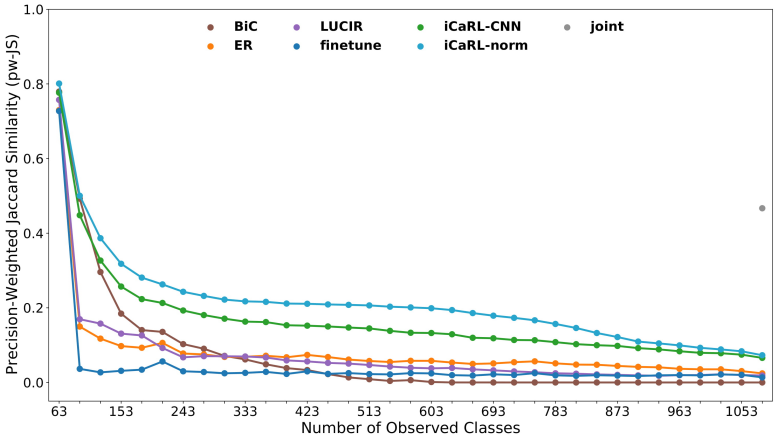
4.2.1. Overall Performance

Let us start by analyzing how well does the model perform over all the observed classes as it encounters new classes. Specifically, as the model finishes training on the j^{th} task, we report the average performance R_j, as measured by the *pw-JS* metric using Equation 3.4, over the test set of all the tasks that the model has seen so far (Figures 4.1 and 4.2). Recall that when computing R_j, the model has to predict all the correct labels for a given sample, even if the labels were seen across different tasks. This makes R_j a challenging metric, as the model cannot achieve a good performance by just memorizing the older labels, but it has to learn the relationship between older and newer labels.

Firstly, in Figure 4.2, we notice that there exists a big discrepancy between the performance of the *ER*-infinite baseline and the incremental joint baseline. Recall from Section 3.6



(a) IIRC-ImageNet-lite



(b) IIRC-ImageNet-full

Fig. 4.1. Average performance on *IIRC-ImageNet-lite* and *IIRC-ImageNet-full* as measured by the precision-weighted Jaccard Similarity (Equation 3.4). (see Figure D.1 for the standard deviation)

that, although both baselines do not discard samples from previous tasks, incremental joint is using the *complete information* setup, and hence it updates the older samples with the newly learned labels if applicable, while *ER*-infinite is using the *incomplete information* setup. This result tells us that dealing with the memory constraint is not sufficient by itself for a model to be able to perform well in the IIRC setup, as only dealing with the catastrophic forgetting challenge (the way *ER*-infinite does) is not sufficient to obtain decent performance.

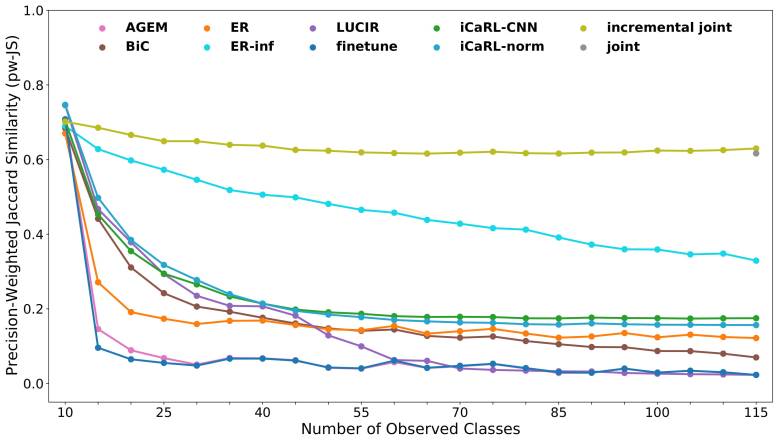
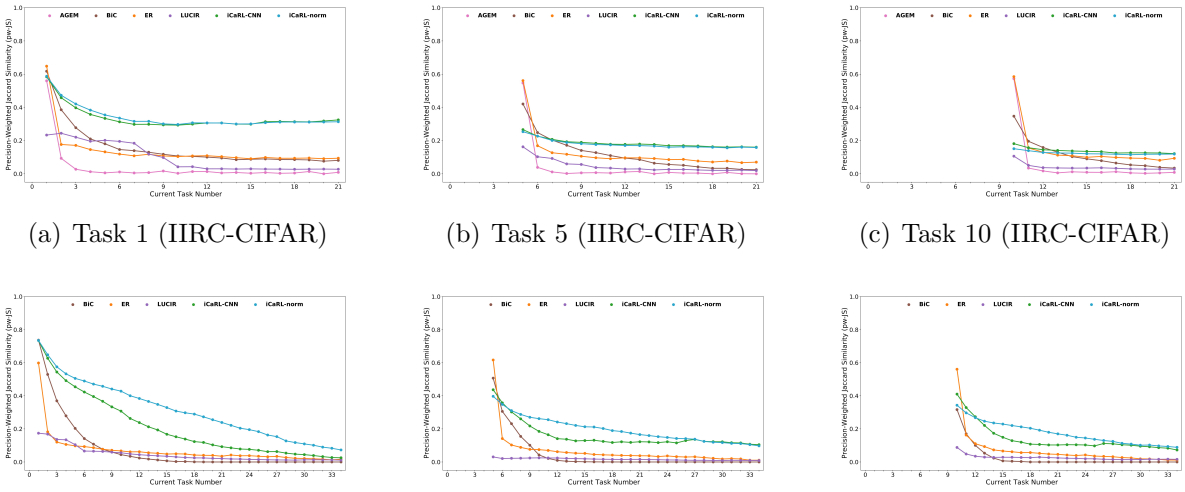


Fig. 4.2. Average performance on *IIRC-CIFAR* as measured by the precision-weighted Jaccard Similarity (Equation 3.4). (see Figure D.2 for the standard deviation)

In Figures 4.1 and 4.2, we observe that the *iCaRL-CNN* and *iCaRL-norm* models perform relatively better than the other methods, with *iCaRL-norm* having the edge in the case of *IIRC-ImageNet*. The reason for this outperformance is that *iCaRL* uses knowledge distillation from the previous model to the newer model (as explained in Section 2.4.2), and hence the previous model acts as a provider for pseudo-labels, which is a potential way for incorporating the model's previous knowledge into the guidance it is receiving. However, this also has a side effect, as the *iCaRL* family of models are usually predicting more labels (some of which are incorrect), a problem we shall explore in more details in Section 4.2.3. We can notice that *BiC* and *LUCIR* perform worse than *ER* after some point, which indicates that they do not make the best use of the replay buffer in their current form in the *IIRC* setup (as opposed to the *CIL* setup, where they generally perform better than *iCaRL*). We also note that the *AGEM* model performs poorly in the case of *IIRC-CIFAR*, even when compared to vanilla *ER*, and hence we didn't run *AGEM* on *IIRC-ImageNet*.

4.2.2. Knowledge Retention and Model Capacity

In lifelong learning setups, the model should retain the previous knowledge as it learns new tasks. Our setup is even more challenging because the model should not only retain previous knowledge, but it should incorporate the new labels as well in this previous knowledge. In Figure 4.3, we track how well the model performs on a specific task, as it is trained on subsequent tasks. Unlike the standard class incremental setup, the model should be able to continually re-associate labels across different tasks to keep performing well on a previous



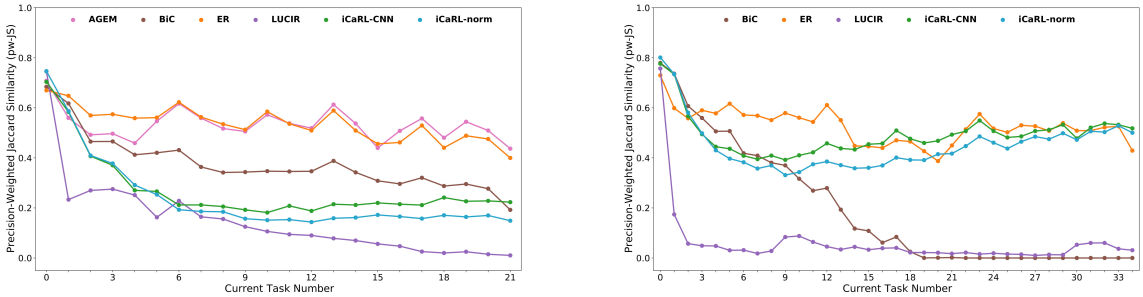
(a) Task 1 (IIRC-CIFAR) (b) Task 5 (IIRC-CIFAR) (c) Task 10 (IIRC-CIFAR)

(d) Task 1 (IIRC-ImageNet-full) (e) Task 5 (IIRC-ImageNet-full) (f) Task 10 (IIRC-ImageNet-full)

Fig. 4.3. The Performance of three middle tasks throughout the whole training process, to measure their catastrophic forgetting and backward transfer. Note that a degradation in performance is not necessarily caused by catastrophic forgetting, as a new subclass of a previously observed superclass might be introduced and the model would be penalized for not applying that label retroactively. (see Figure D.3 for the standard deviation)

task. The key takeaway is that, while the baselines are generally expected to reasonably alleviate catastrophic forgetting, their performance degrades rapidly as the model trains on more tasks. *ER*'s poor performance may be accounted for by two hypothesis: **i)** The model is trained on a higher fraction of samples per class for classes that belong to the current task, than those of previous tasks, causing bias towards newer classes. **ii)** The model sometimes gets conflicting supervising signal, as the model might observe samples that belong to the same subclass (ex. "polar bear"), once with the superclass label from the buffer ("bear"), and another with the subclass label from the current task data ("polar bear'), and it doesn't connect these two pieces of information together. In the case of *LUCIR*, we hypothesize that the model's performance deteriorates also because of the conflicting signal problem, which is exacerbated by *LUCIR*'s use of the margin ranking loss (see Section 4.3.3 for more details on the effect of the margin ranking loss).

We can see as well in Figure 4.4 the performance of each model on the current task j, after training on that task (R_{jj} using Equation 3.3). The general trend is that the less the model is regularized, the higher it can perform on the current task, which is intuitive. Since *LUCIR* is highly regularized, and the *IIRC* setup requires the flexibility of the model to consolidate between its previous beliefs and the new information, we can see that *LUCIR*

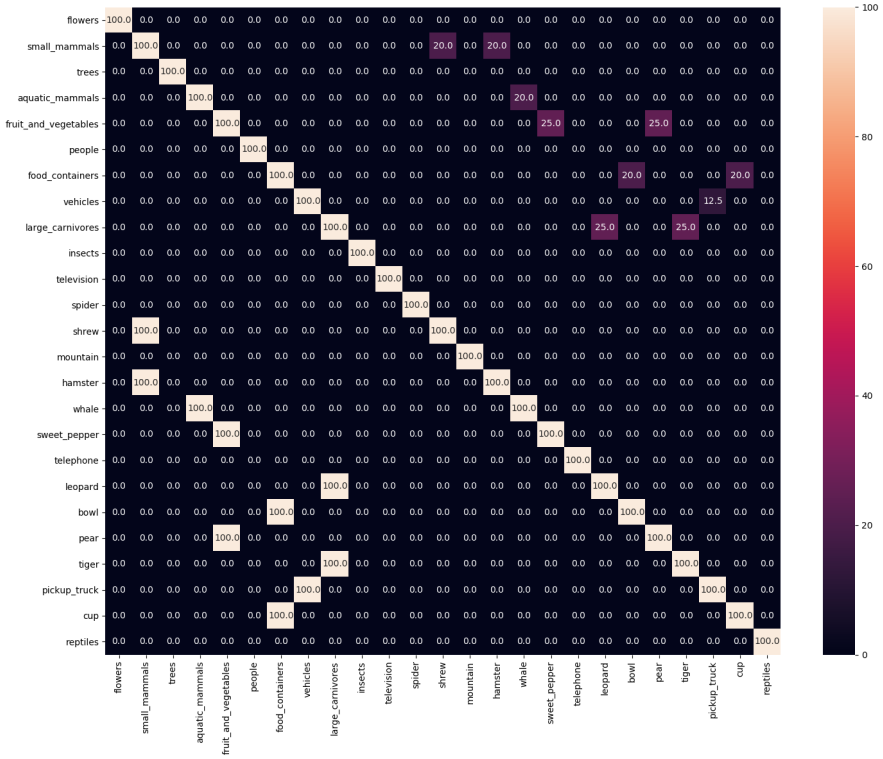


(a) IIRC-CIFAR

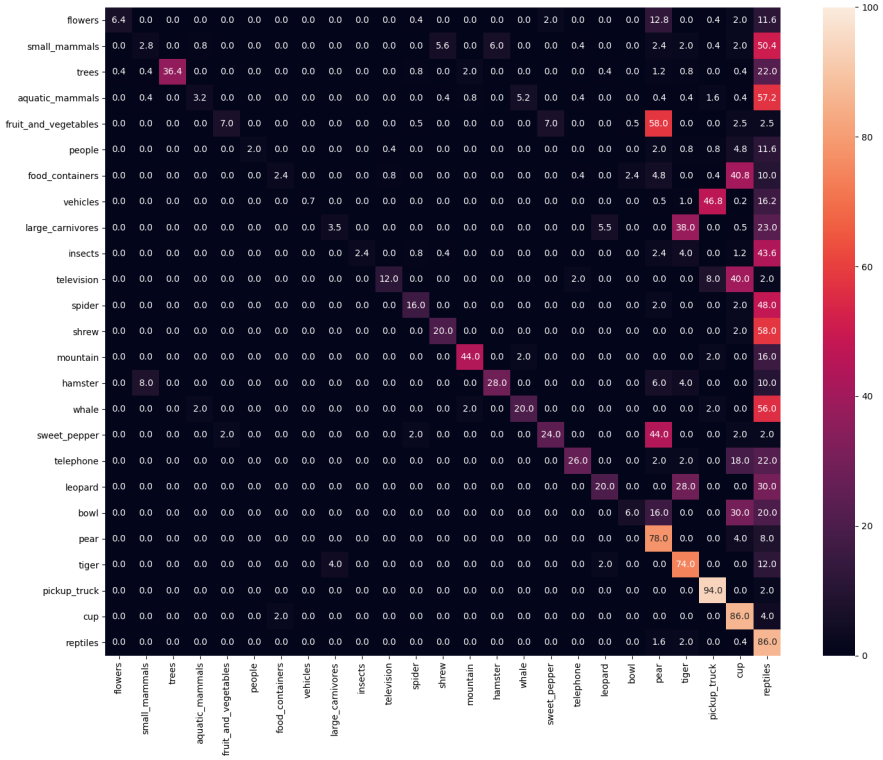
(b) IIRC-ImageNet-full

Fig. 4.4. Current task performance; Per task performance over the test samples of a specific task j, after training on that task (R_{jj} using Equation 3.3). see Figure D.4 for the standard deviation)

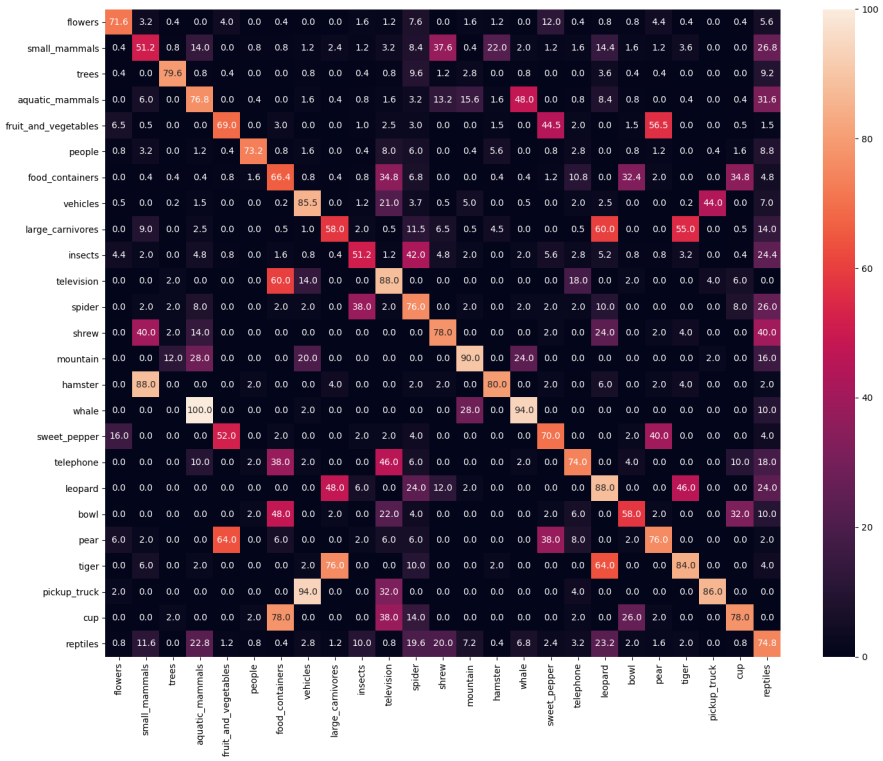
nearly fails to learn new classes. We can see as well that *BiC* forgets quickly on one hand (Figure 4.3), and its ability to learn new classes deteriorates with the time (Figure 4.4), both although *BiC* is similar to *iCaRL-CNN* but with the addition of a bias correction phase. The reason is that although *BiC* uses distillation during training, it does not use distillation during the bias correction phase, which reduces its ability for the retention of knowledge (see Section 4.3.4 for the effect of using distillation during the bias correction phase). Moreover, the reason for the collapse of its ability to learn new classes in the longer sequence of *IIRC-ImageNet-full* is that as the number of observed classes increase, the λ becomes much more skewed towards the distillation loss (see Equation 2.23), which hinders its ability to learn new classes, especially that the relationship between classes is not as simple as in the *CIL setup*.



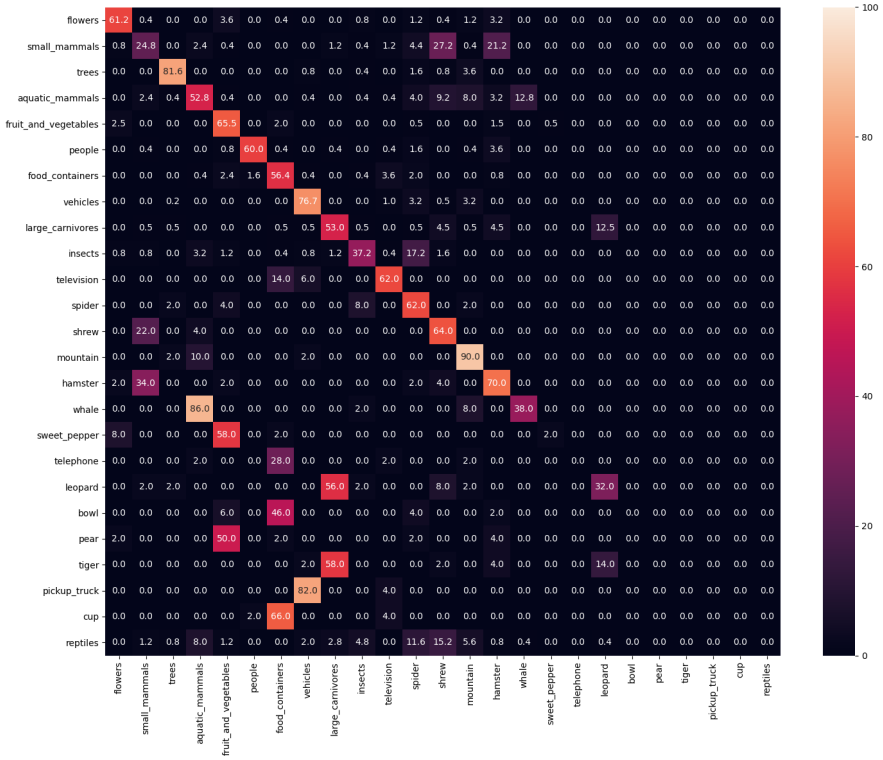
(a) Ground Truth



(b) ER

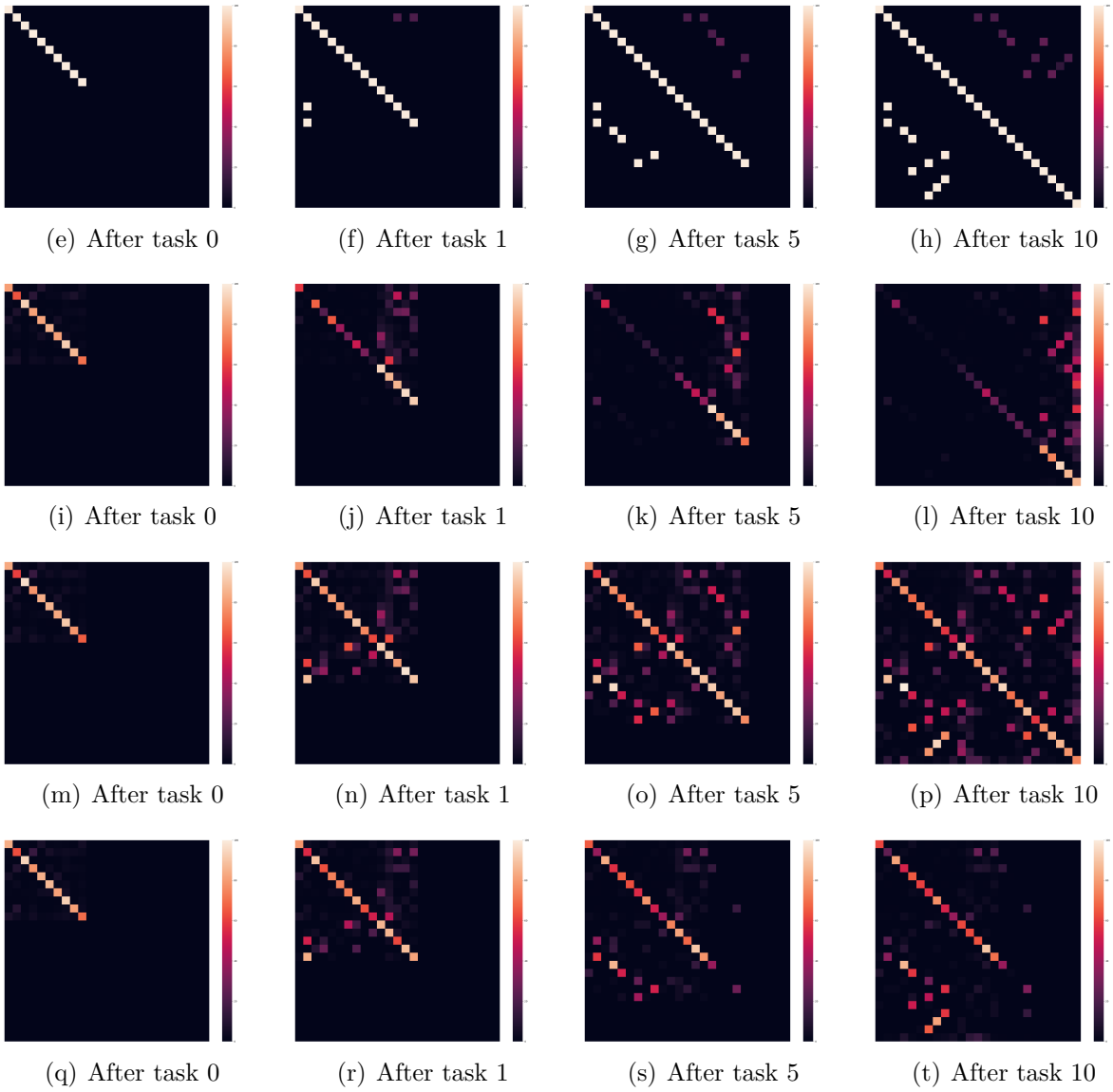


(c) iCaRL-norm



(d) LUCIR

Fig. 4.4. Confusion matrix after observing task 10 of *IIRC-CIFAR*. The y-axis is the correct label (or one of the correct labels). The x-axis is the predicted labels. Labels are arranged by their order of introduction. Only 25 labels across the tasks are shown for better visibility.



(e) After task 0 (f) After task 1 (g) After task 5 (h) After task 10

(i) After task 0 (j) After task 1 (k) After task 5 (l) After task 10

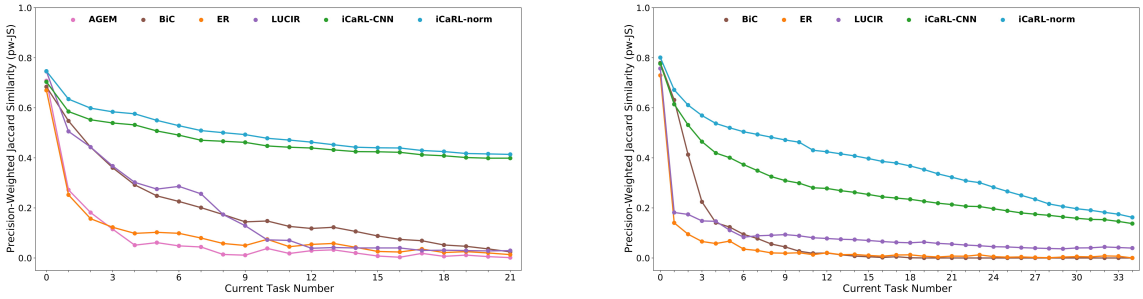
(m) After task 0 (n) After task 1 (o) After task 5 (p) After task 10

(q) After task 0 (r) After task 1 (s) After task 5 (t) After task 10

Fig. 4.5. Confusion matrix after introducing tasks 0, 1, 5, 10 of IIRC-CIFAR respectively. The y-axis is the correct label (or one of the correct labels). The x-axis is the model predicted labels. Labels are arranged by their order of introduction. Only 25 labels are shown for better visibility. (1st row) Ground Truth, (2nd row) *ER*, (3rd row) *iCaRL-norm*, (4th row) *LUCIR*.

4.2.3. Confusion Between Related Classes

Some other important questions are whether the model correctly associates the newly learned subclass labels to their previously learned superclass, and whether it incorrectly associates/confuses the newly learned subclass label with other previously learned subclasses



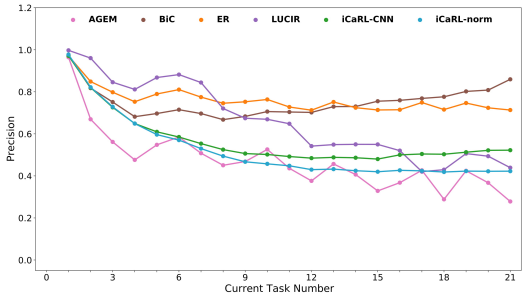
(a) IIRC-CIFAR

(b) IIRC-ImageNet-full

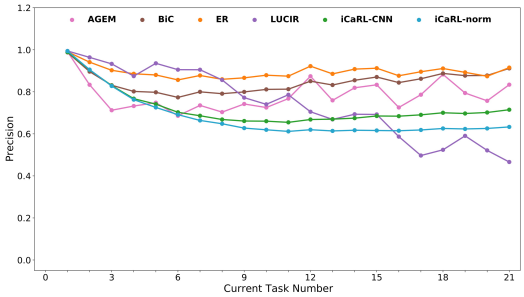
Fig. 4.6. The average performance of *IIRC-ImageNet-full* and *IIRC-CIFAR* if only the superclasses are taken into account for calculating this performance. (see Figure D.5 for the standard deviation)

(that have the same superclass). We dig deeper into the confusion matrix (Figure 4.4) for the predictions of the different models after training on ten tasks of *IIRC-CIFAR*. Note that in Figure 4.4, the lower triangular matrix shows the percentage the model predicts older labels for the newly introduced classes, while the upper triangular matrix represents the percentage the model predict newer labels to older classes, with the ground truth being Figure 5(a). We can see that the *ER* method predictions are highly skewed towards the newly learned labels (last five classes), as shown in Figure 5(b)). The *iCaRL-norm* model, as shown in Figure 5(c), performs relatively well in terms of associating the (newly learned) subclasses to their (previously learned) superclasses. For example, whales are always correctly labeled as aquatic mammals, and pickup trucks are correctly labeled as vehicles 94% of the time. However, these models learn some spurious associations as well. For instance, "television" is often mislabeled as "food containers". Similarly, the model in general correctly associates older superclasses with their newer subclasses, but many times it incorrectly create these associations (eg associating " aquatic mammals" with "whales" 48% of the time and "vehicles" with "pickup trucks" 44% of the time, while by looking at figure 5(a), we see that they only represent 20% and 12.5% of their superclasses respectively) The *LUCIR* model provides accurate superclass labels to the subclasses. This is shown in Figure 5(d) where *LUCIR* follows the trends of the ground truth more closely than *iCaRL-norm* in the lower triangular part of the confusion matrix. However, it fails to learn new associations. We provide more instances of such plots in Figure4.5, which shows that the observed trends are quite general.

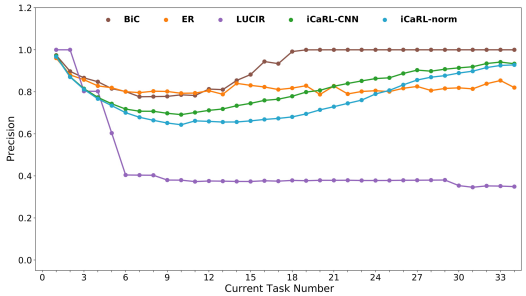
In order to provide these insights in a more quantitative way, we provide in Figure 4.6 how much each model is able to retain the superclasses it has learned, and apply them to the



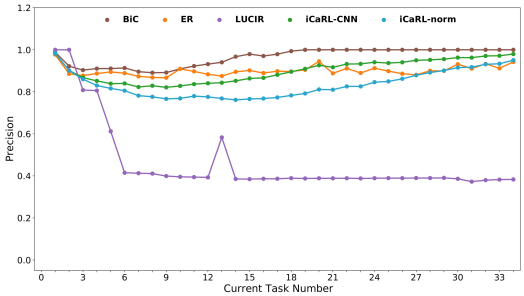
(a) Subclasses Precision (IIRC-CIFAR)



(b) Orphan Subclasses Precision (IIRC-CIFAR)



(c) Subclasses Precision (IIRC-ImageNet)



(d) Orphan Subclasses Precision (IIRC-ImageNet)

Fig. 4.7. The average precision of *IIRC-CIFAR* and *IIRC-ImageNet-full* over each type of subclasses, excluding other types of classes, to measure how much do the models confuse the subclasses as they encounter more related subclasses. (see Figure D.6 for the standard deviation)

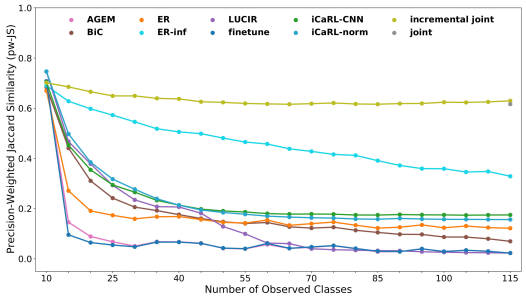
newly introduced samples that belong to their subclasses. This is measured by calculating the pw-JS solely over the superclasses (only the model predictions that belong to superclasses are taken into account).

Moreover, Figure 4.7 shows how much each model is confused between the subclasses that belong to superclasses, which is measured by calculating the precision over only these subclasses (only the model predictions that belong to these subclasses are taken into account). This is contrasted against the precision over orphan subclasses, so as to give an idea of how much the loss in precision can be attributed to the visual similarity between the subclasses. Intuitively, we can see that the models collectively tend to be less precise with subclasses vs orphan subclasses. It should be noted that if the model predicts no labels (zero true positives and zero false positives), the convention used is to set the precision to one, since the goal of this figure is to capture how much the model confuses the outputs together, rather than if the model is learning or not. This explains why the precision of *BiC* significantly

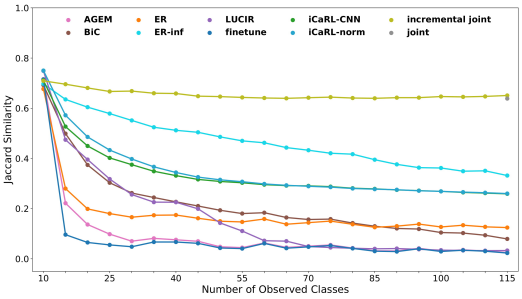
increases when the model collapses. We can also observe that the precision of both *iCaRL* variants increases in the second half of the tasks sequence in the case of *IIRC-ImageNet-full*, which can be explained by the model's increased forgetting of the previous labels, and hence decreased confusion.

4.3. Ablations

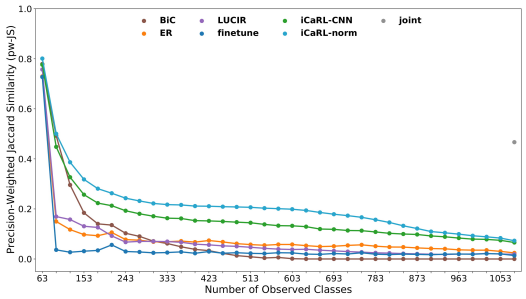
4.3.1. JS vs pw-JS:



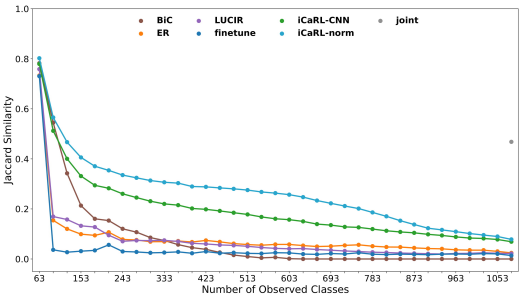
(a) precision-weighted Jaccard Similarity (IIRC-CIFAR)



(b) Jaccard Similarity (IIRC-CIFAR)



(c) precision-weighted Jaccard Similarity (IIRC-ImageNet-full)



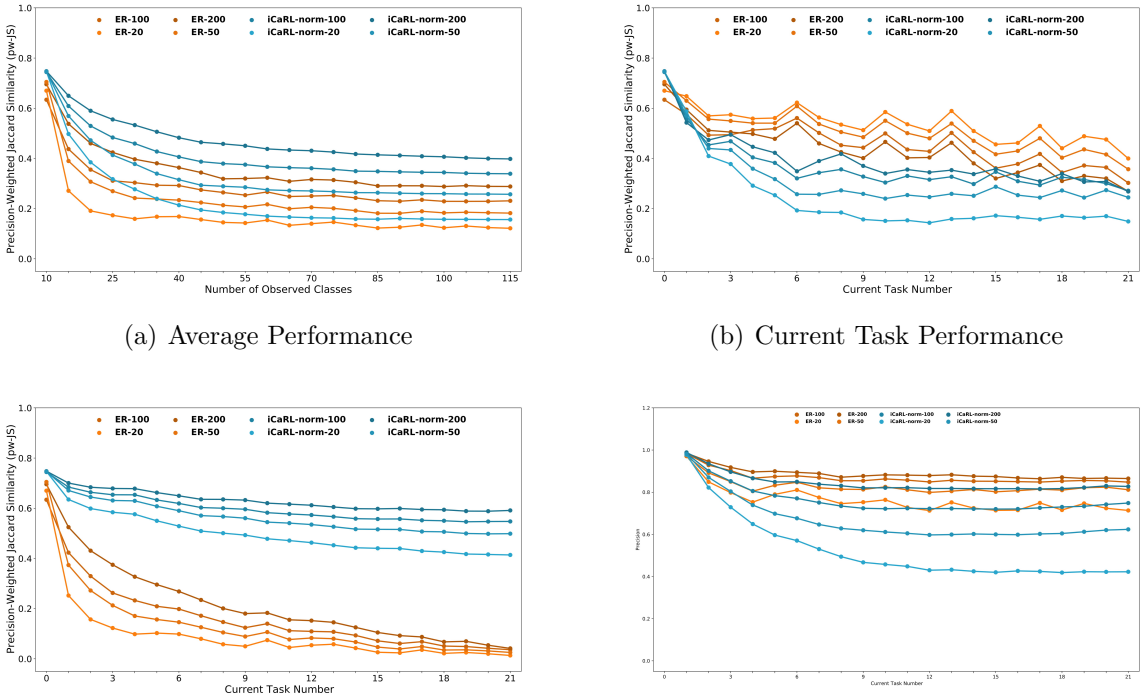
(d) Jaccard Similarity (IIRC-Imagenet-full)

Fig. 4.8. Average performance on *IIRC-CIFAR* and *IIRC-ImageNet-full*, as measured by the precision-weighted Jaccard Similarity compared to the Jaccard Similarity. (see Figure D.7 for the standard deviation)

As mentioned in Section 3.5.2, the reasoning behind introducing the *precision-weighted Jaccard similarity* is to give a special weight to whether the model is predicting the correct associations between the classes. In Figure 4.8, we compare between the overall performance of the different methods when measured using the *JS* versus when measured using our

proposed *pw-JS*. As evident from the figure, both metrics mostly follow the same trend. However, the discrepancy appears in the case of the distillation based methods (*iCaRL-CNN* and *iCaRL-norm*), as they tend to be imprecise due to the nature of their objective, as elaborated earlier. This discrepancy is basically what we wanted to capture by using the *pw-JS* as the main metric, as it gives a better idea of which methods perform better in the knowledge retention aspect, and simultaneously are more able to modify their previous knowledge and associate the labels together in a precise way.

4.3.2. Buffer Size Effect



(a) Average Performance

(b) Current Task Performance

(c) Superclasses Performance

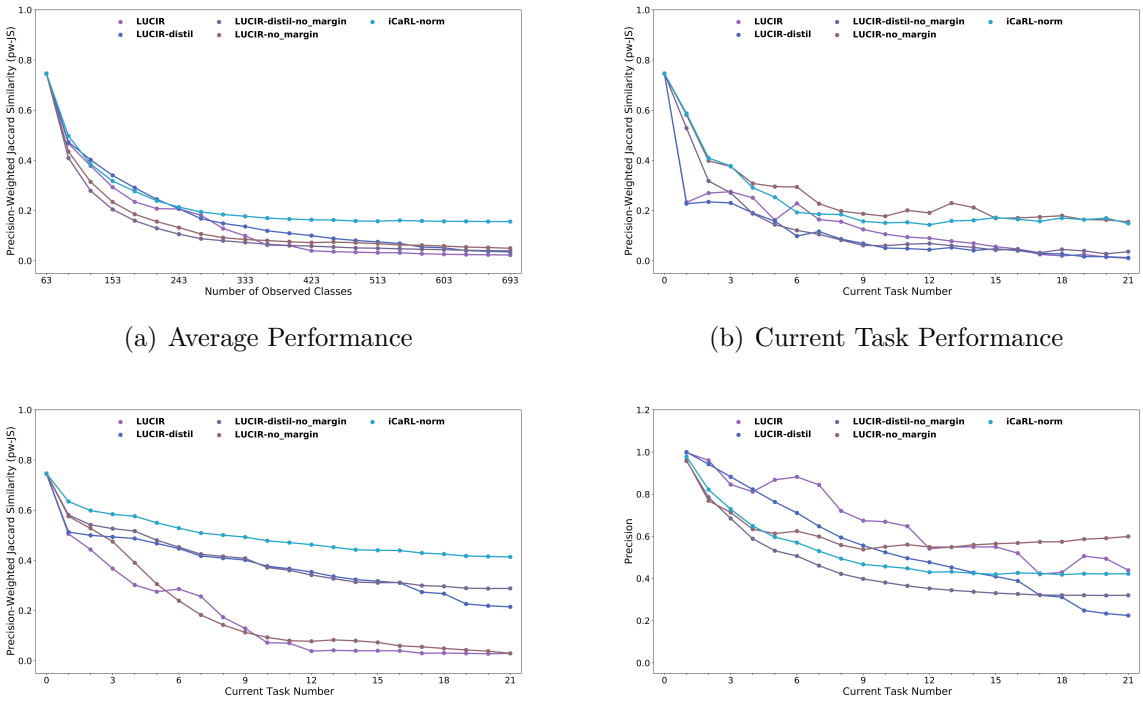
(d) Subclasses Precision

Fig. 4.9. The effect of increasing the size of the buffer on the performance of *ER* and *iCaRL-norm* in *IIRC-CIFAR*. For each baseline, darker lines correspond to larger buffer size. (See Figure D.8 for the standard deviation)

In all the previous experiments, the replay buffer size was set to 20 samples per class. In this set of experiments, we try the *ER* baseline and the *iCaRL-norm* baseline using different sizes for the replay buffer (20, 50, 100, and 200 samples per class) to see how their performance reacts to increasing the replay buffer size. Intuitively, increasing the buffer size

increases the performance for both baselines (Figure 9(a)). However, The large buffer size does not prevent *ER* from forgetting the superclasses, which are mostly introduced during the early tasks (Figure 9(c)). An interesting observation in Figure 9(d) is that although using pseudo-labels makes *iCaRL-norm* prone to lower precision, especially when the classes are visually similar, increasing the buffer size significantly increases the subclasses precision.

4.3.3. Effect of Margin Ranking Loss and Distillation in LUCIR



(a) Average Performance

(b) Current Task Performance

(c) Superclasses Performance

(d) Subclasses Precision

Fig. 4.10. The performance of *LUCIR* after removing the margin ranking loss *LUCIR-no_margin*, using a distillation objective *LUCIR-distil*, and both *LUCIR-distil-no_margin*. (See Figure D.9 for the standard deviation)

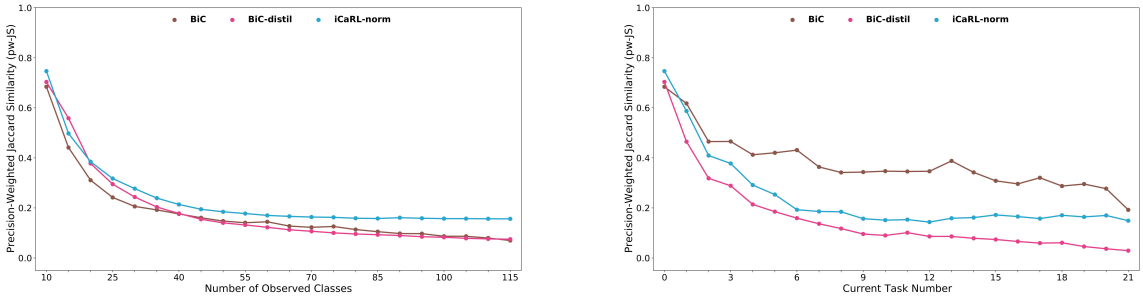
If we have a look at Section 2.4.3, we would find that there are two problems with the *LUCIR* approach that need to be tackled in order to improve its performance in the *IIRC* setup. The first problem is that, as in the case of *ER*, *LUCIR* does not use the pseudo-labels provided by the older model as training targets. We have seen in Section 4.2.2 that this does not work in the context of *IIRC*, since the model would be receiving conflicting signals with no way of consolidating them. We try a variation, which we call *LUCIR-distil* that

uses the pseudo-labels during training in order to see their effect. The other problem is that one of the three techniques that *LUCIR* employs to alleviate the problem of catastrophic forgetting, is the technique of applying the *margin ranking loss* on the samples of the replay buffer (Equation 2.19). This loss helps in increasing the interclass separation between the older classes and newer classes. However, it can be seen how this technique can hurt the performance on *IIRC*, since it can try to increase the separation between the superclass and its newly introduced subclass, which would make it more difficult to learn such subclass. We try a variant, which we call *LUCIR-no_margin*, where this loss is removed. Finally, we try a third variant *LUCIR-distil-no_margin* which combines the two variants together.

In Figure 10(b), we can see that removing the margin ranking loss does moderately increase the capacity of the model to learn new tasks in both *LUCIR-no_margin*, and to a smaller extent *LUCIR-distil-no_margin*. On the other hand, We can see that using the distillation targets improves the knowledge retention of the previously learnt superclasses for both *LUCIR-distil* and *LUCIR-distil-no_margin* (Figure 10(c)), but it also decreases the subclasses precision in Figure 10(d). However, when it comes to the overall performance, these variants only marginally outperform *LUCIR*(Figure 10(a)). This might indicate that *LUCIR* and its variants are held back by their strong regularization, which decrease their flexibility at breaking associations between labels and creating new ones.

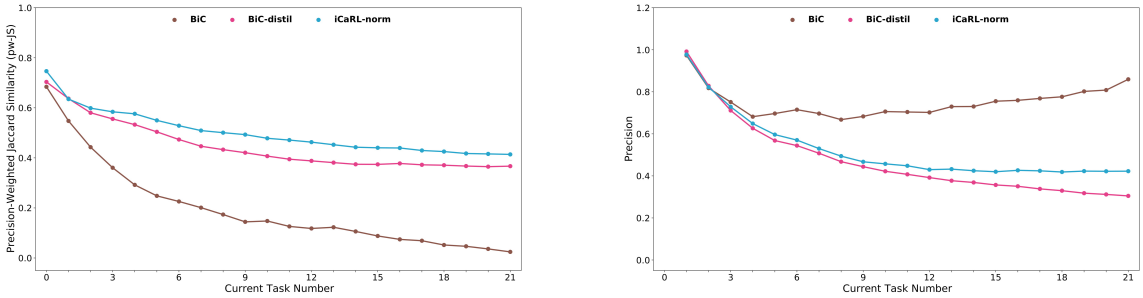
4.3.4. Bias Correction in BiC using pseudo-labels

BiC method tries to overcome the bias towards newer classes by having a separate bias correction phase after each task during which the parameters of a bias correction layer are tuned. This layer scales the logits of the classes in order to balance them (see Section 2.4.4 for more details). Although *BiC* uses knowledge distillation as is the case in *iCaRL-norm*, it does not apply these pseudo-labels as targets during the bias correction phase. This might make sense in the context of the class incremental setup, since the dataset used during this bias correction phase is balanced, however it might hurt the performance in the case of *IIRC* since the model is not able to incorporate its previous knowledge during the bias correction process. We experiment with a *BiC* variant (*BiC-distil*), where we try to see how would the model perform if the pseudo-labels provided by the previous model were used as targets during the bias correction phase as well. In Figure 11(c), we can see that *BiC-distil* significantly improves the ability of *BiC* in knowledge retention of the superclasses and in applying these previously learned superclasses to the samples that belong to them in subsequent tasks, while hurting its precision when it comes to distinguishing between subclasses (Figure 11(d)), which is similar to what happened when distillation was used



(a) Average Performance

(b) Current Task Performance



(c) Superclasses Performance

(d) Subclasses Precision

Fig. 4.11. The performance of *BiC* after using a distillation objective during the bias correction phase *BiC-distil*. (See Figure D.10 for the standard deviation)

in *LUCIR-distil*. However, The overall performance of *BiC-distil* almost does not improve upon the performance of *BiC*, as *BiC-distil* has a significantly lower ability to learn new tasks (Figure 11(b)). This can be attributed to the higher weight that *BiC-distil* gives to the distillation loss as the number of observed classes increase (Equation 2.23), as opposed to *iCaRL* which does not have such weighting (Equation 2.12). Moreover, the pseudo-labels are provided for the older labels, but they are not provided for the newer labels. This means that the new samples help in retaining the old classes, but the old samples from the buffer try to suppress the newer classes, and since the dataset is balanced across the old classes and new classes during the bias correction phase, this effect becomes more pronounced.

Chapter 5

Conclusion and Future Directions

In the previous chapters, we have seen what is lifelong learning, and how it derives its importance from the very first principles of machine learning. Lifelong learning is a difficult problem, as it includes the challenges that we as humans face during our lifetimes. However, the current lifelong learning setups are still limited in the breadth of the challenges that they explore, which necessitates the introduction of new setups that allow the exploration of such challenges.

To that end, the *IIRC* setup and benchmark was introduced. In the *IIRC* setup, the model starts training on the high-level superclasses, and then observes the subsequent tasks which are mainly composed of more fine-grained classes. The training happens in an incomplete information setting, where the labels that belong to the current task are the only labels that are provided during training, and hence the model needs to use what it has acquired during the previous tasks to recognize if the samples belong as well to some higher level superclass it has already learned. *IIRC-CIFAR* and *IIRC-ImageNet* were introduced to serve as benchmarks for this setup, and precision-weighted Jaccard similarity was introduced to serve as a metric of performance that captures the different aspects related to the setup. The benchmark was introduced in a standardized way with a preset tasks configurations and two validation sets to allow for fair and easy comparison.

Although the *IIRC* setup follows a particular scenario that might seem too specific, the strength of *IIRC* lies in that the challenges it tries to address are generalizable to a much broader set of scenarios that exist in real-life. The idea that when we learn a concept, we might not be told the whole story at each time, but we might be given some part of the truth and we are required to use our previous knowledge in order to complete the whole picture. This is especially true in a lifelong learning scenario, since it can be expensive to provide extensive data with all the labels, especially that it should be expected that the number of observed classes are to increase dramatically in such scenario. Removing that constraint, by incorporating what the model already knows, makes the process smarter and

easier. Moreover, real-life is rarely static, we learn new things everyday that may require us to alter our previous understanding, and hence the quality of being flexible enough to not only keep the previous knowledge, but also alter it when new facts are revealed, is essential for a model to be deployed in a realistic lifelong learning environment. Finally, learning in a gradually refined way has its roots in how humans learn, and this also reflects on real life since it is always easier to collect data for the less refined classes and augment the model with the refined concepts once they are available.

We also tested the most important lifelong learning methods after adapting them to *IIRC* setup. Distillation-based methods proved to be strong baselines, as they allow the model to incorporate its previous knowledge and to gradually change the associations they build between the different concepts. However, they are weak when it comes to detecting out-of-distribution (OOD) samples, as they might apply some pseudo-labels to these OOD samples, as in applying a superclass pseudo-label to a newly introduced orphan subclass that it has not seen before. Another weakness for distillation based methods in a partially multi-label setting like *IIRC* is that it exacerbates the model's ability to distinguish between visually similar classes, since in a multi-label setting the model is allowed to give the samples of two similar classes "A" and "B" both the labels "A" and "B". On the other hand, *LUCIR* have a strong regularization which helps in the case of *CIL* setup. However, since *IIRC* requires more flexibility from the model's side, this strong regularization hurts in the context of *IIRC*. Some ablations were done to make *LUCIR* less regularized and to use the distillation strategy, however it then suffers from the same problems as the other distillation based baselines.

5.1. Future Directions

One future direction to strengthen the performance of the distillation-based models can be by employing an explicit complementary OOD detection algorithm [Lee et al., 2018, Vyas et al., 2018] in detecting labels which have no associations with previous labels. However, this algorithm would need to be adapted so that it does not suffer itself from catastrophic forgetting, and hence confusing previously seen classes as OOD. Moreover, unlabeled data can be used as well for this OOD algorithm [Yu and Aizawa, 2019], as unlabeled data is easily available even in the context of lifelong learning. Another advantage for using unlabeled data is that this data by itself can be used in a self-supervised manner to enrich the representation learnt by the lifelong learning model and regularize it in meaningful manner [Zhang et al., 2020].

On the other hand, further research is needed in the direction of extending the setups and benchmarks in the area of lifelong learning so as to make the lifelong learning models more suitable for deployment in real life. Typically, a realistic setup would include all the aspects from *class incremental learning* (where the classes are added sequentially), to *task incremental learning* (where the classes can sometimes belong to different tasks with different output heads), to *domain incremental learning* (where the domain shifts for some of the classes), to *IIRC* (where the classes are related and only some of the labels are present at a time). Incorporating all these challenges in a single setup would result in models which are more prepared for real-life. However, at the current state of the lifelong learning research, this setup might prove too difficult and hence would be limited in utility, since it contains a lot of components which would make analyzing where do the models fail very challenging. Hence, developing models for these more specialized setups is beneficial, as long as they take into account the existence of other challenges that would need to be tackled at some point, and the longer term goal should be to start this kind of universal setups once the models start to show decent performance in each of these more specialized setups.

References

Mohamed Abdelsalam, Mojtaba Faramarzi, Shagun Sodhani, and Sarath Chandar. IIRC: Incremental implicitly-refined classification. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2021.

W. Abraham and A. Robins. Memory retention – the synaptic stability versus plasticity dilemma. Trends in Neurosciences, 28:73–78, 2005.

Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. arXiv preprint arXiv:1910.07113, 2019.

Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In Proceedings of the European Conference on Computer Vision (ECCV), pages 139–154, 2018.

Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In Proceedings of the International conference on Machine Learning, pages 173–182, 2016.

Antreas Antoniou, Massimiliano Patacchiola, Mateusz Ochal, and Amos Storkey. Defining benchmarks for continual few-shot learning, 2020.

Alexei Baevski, Henry Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. arXiv preprint arXiv:2006.11477, 2020.

Eden Belouadah and Adrian Popescu. Il2m: Class incremental learning with dual memory. In Proceedings of the IEEE International Conference on Computer Vision, pages 583–592, 2019.

Leo Breiman. Random forests. Mach. Learn., 45(1):5–32, October 2001. ISSN 0885-6125. doi: 10.1023/A:1010933404324. URL `https://doi.org/10.1023/A:1010933404324`.

Francisco M Castro, Manuel J Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari. End-to-end incremental learning. In Proceedings of the European conference on computer vision (ECCV), pages 233–248, 2018.

Arslan Chaudhry, Puneet K Dokania, Thalaiyasingam Ajanthan, and Philip HS Torr. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In Proceedings of the European Conference on Computer Vision (ECCV), pages 532–547, 2018.

Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. In International Conference on Learning Representations, 2019a.

Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K. Dokania, Philip H. S. Torr, and Marc'Aurelio Ranzato. On tiny episodic memories in continual learning, 2019b.

Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. Wide & deep learning for recommender systems. In Proceedings of the 1st workshop on deep learning for recommender systems, pages 7–10, 2016.

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

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

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

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. MIT Press, 2016. `http://www.deeplearningbook.org`.

Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. arXiv preprint arXiv:1312.6211, 2013.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In Proceedings of the International Conference on Machine Learning, pages 2555–2565. PMLR, 2019.

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.

Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In Proceedings of the 26th international conference on world wide web, pages 173–182, 2017.

Marti A. Hearst. Support vector machines. IEEE Intelligent Systems, 13(4):18–28, July 1998. ISSN 1541-1672. doi: 10.1109/5254.708428. URL https://doi.org/10.1109/5254.708428.

Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. In NIPS Deep Learning and Representation Learning Workshop, 2015. URL http://arxiv.org/abs/1503.02531.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. Neural computation, 9(8):1735–1780, 1997.

Saihui Hou, Xinyu Pan, Chen Change Loy, Zilei Wang, and Dahua Lin. Learning a unified classifier incrementally via rebalancing. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 831–839, 2019.

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

Alex Krizhevsky. Learning multiple layers of features from tiny images. pages 32–33, 2009. URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

Kimin Lee, Kibok Lee, Honglak Lee, and Jinwoo Shin. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. In Proceedings of the 32nd International Conference on Neural Information Processing Systems, NIPS'18, page 7167–7177, Red Hook, NY, USA, 2018. Curran Associates Inc.

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

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

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

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.

Arun Mallya, Dillon Davis, and Svetlana Lazebnik. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In Proceedings of the European Conference on Computer Vision (ECCV), pages 67–82, 2018.

Marc Masana, Xialei Liu, Bartlomiej Twardowski, Mikel Menta, Andrew D Bagdanov, and Joost van de Weijer. Class-incremental learning: survey and performance evaluation. arXiv preprint arXiv:2010.15277, 2020a.

Marc Masana, Tinne Tuytelaars, and Joost van de Weijer. Ternary feature masks: continual learning without any forgetting. arXiv preprint arXiv:2001.08714, 2020b.

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.

Seyed Iman Mirzadeh, Mehrdad Farajtabar, Razvan Pascanu, and Hassan Ghasemzadeh. Understanding the role of training regimes in continual learning, 2020.

Thomas M. Mitchell. Machine Learning. McGraw-Hill, Inc., USA, 1 edition, 1997. ISBN 0070428077.

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

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. nature, 518(7540):529–533, 2015.

Jason Ramapuram, Magda Gregorova, and Alexandros Kalousis. Lifelong generative modeling. arXiv preprint arXiv:1705.09847, 2017.

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.

David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. In Advances in Neural Information Processing Systems, pages 350–360, 2019.

Amir Rosenfeld and John K Tsotsos. Incremental learning through deep adaptation. IEEE transactions on pattern analysis and machine intelligence, 2018.

Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks, 2016.

Jeffrey C Schlimmer and Richard H Granger. Incremental learning from noisy data. Machine learning, 1(3):317–354, 1986.

Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. arXiv preprint arXiv:1801.01423, 2018.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. nature, 550(7676):354–359, 2017.

Shagun Sodhani, Sarath Chandar, and Yoshua Bengio. Toward training recurrent neural networks for lifelong learning. Neural computation, 32(1):1–35, 2020.

Mohammad Sorower. A literature survey on algorithms for multi-label learning, 2010.

Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A Survey on Deep Transfer Learning: 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4–7, 2018, Proceedings, Part III, pages 270–279. 10 2018. ISBN 978-3-030-01423-0. doi: 10.1007/978-3-030-01424-7_27.

Sebastian Thrun and Tom M. Mitchel. Lifelong robot learning. Robotics and Autonomous Systems, 1995.

Sebastian Thrun and Tom M. Mitchel. Child: A first step towards continual learning. Machine Learning, 28:77–104, 1997.

Gido M. van de Ven and Andreas S. Tolias. Three scenarios for continual learning. In Neurips 2018, 2018. URL `https://arxiv.org/pdf/1904.07734.pdf`.

V. Vapnik. Principles of risk minimization for learning theory. In Proceedings of the 4th International Conference on Neural Information Processing Systems, NIPS'91, page 831–838, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc. ISBN 1558602224.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Advances in neural information processing systems, pages 5998–6008, 2017.

Apoorv Vyas, Nataraj Jammalamadaka, Xia Zhu, Dipankar Das, Bharat Kaul, and Theodore L. Willke. Out-of-distribution detection using an ensemble of self supervised leave-out classifiers. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, Computer Vision – ECCV 2018, pages 560–574, Cham, 2018. Springer International Publishing. ISBN 978-3-030-01237-3.

Yue Wu, Yinpeng Chen, Lijuan Wang, Yuancheng Ye, Zicheng Liu, Yandong Guo, and Yun Fu. Large scale incremental learning. In Proceedings of the IEEE Conference on Computer

Vision and Pattern Recognition, pages 374–382, 2019.

Qing Yu and Kiyoharu Aizawa. Unsupervised out-of-distribution detection by maximum classifier discrepancy. In 2019 IEEE/CVF International Conference on Computer Vision (ICCV), pages 9517–9525, 2019. doi: 10.1109/ICCV.2019.00961.

Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In Proceedings of the International conference on Machine Learning, volume 70, pages 3987–3995. PMLR, 2017.

Song Zhang, Gehui Shen, Jinsong Huang, and Zhi-Hong Deng. Self-supervised learning aided class-incremental lifelong learning, 2020.

Appendix A

Pseudo Codes

Algorithm 1: IncrementalTrain

Require: *tasks* // A list of the classes to-be-introduced at each task

1 $trainSet$, $validSet_{inTask}$, $validSet_{postTask}$, $testSet$ ← LoadDatasets()
2 $model$ ← CreateModel()
`/* create an empty buffer */`
3 $buffer$ ← CreateBuffer()
4 **for** $task$ in $tasks$ **do**
5 $model$ ← TrainOnTask($model$, $buffer$, $trainingSet$, $validSet_{inTask}$)
`/* add randomly selected samples to buffer */`
6 $buffer$ ← AddToBuffer ($buffer$, $trainingSet$)
7 PostTaskEvaluate($model$, $validSet_{postTask}$, $testSet$)
8 **end**

Algorithm 2: LoadDatasets

input : $rawData_{train}$ // The default single-label full dataset (train)
input : $rawData_{test}$ // The default single-label full dataset (test)
input : $classHierarchy$ // A dictionary that maps each superclass to its constituent subclasses

1 $multilabeledData_{train} \leftarrow$ AddSuperclassLabels($rawData_{train}$, $classHierarchy$)
2 $multilabeledData_{test} \leftarrow$ AddSuperclassLabels($rawData_{test}$, $classHierarchy$)
3 $multilabeledData_{train}$, $multilabeledData_{valid_{inTask}}$, $multilabeledData_{valid_{postTask}} \leftarrow$ SplitData($multilabeledData_{train}$)
4 $trainSet \leftarrow$ IncompleteInfoIncrementalDataset($multilabeledData_{train}$)
5 $validSet_{inTask} \leftarrow$ IncompleteInfoIncrementalDataset($multilabeledData_{valid_{inTask}}$)
6 $validSet_{postTask} \leftarrow$ CompleteInfoIncrementalTestDataset($multilabeledData_{valid_{postTask}}$)
7 $testSet \leftarrow$ CompleteInfoIncrementalTestDataset($multilabeledData_{test}$)

output : $trainSet$ // The incomplete information incremental learning training set
output : $validSet_{inTask}$ // The incomplete information incremental learning validation set (for in-task performance)
output : $validSet_{postTask}$ // The complete information incremental learning validation set (for post-task performance)
output : $testSet$ // The complete information incremental learning test set

Algorithm 3: IncompleteInfoIncrementalDataset

input : $multilabelData$ // A list of samples with each sample in the form of ($image$, ($superclassLabel$, $subclassLabel$)) or ($image$, ($subclassLabel$)))
input : $superclassToSubclass$ // a mapping that maps superclasses to their constituent subclasses
input : $tasks$ // The classes to-be-introduced at each task
Require: $subclasses$ // All refined subclasses (those who have a superclass as well as those who don't)
output : a dataset object with the data changing along the tasks

Initialization:
$classToDataIndices \leftarrow$ EmptyDictionary
$currentTaskId \leftarrow 0$
$dataIndices_{task} \leftarrow []$
for $subclass$ in $subclasses$ **do**
    `/* get the indices of the samples which correspond to this subclass */`
    $dataIndices_{subclass} \leftarrow$ GetSamplesIndices($mtultilabelData$, $subclass$)
    if $subclass$ has superclass **then**
        $dataSubsetLength_{superclass} \leftarrow 0.4$ * Length($dataIndices_{subclass}$)
        $dataSubsetLength_{subclass} \leftarrow 0.8$ * Length($dataIndices_{subclass}$)
        $dataIndices_{subclass} \leftarrow$ Shuffle($dataIndices_{subclass}$)
        $dataSubsetIndices_{subclass} \leftarrow dataIndices_{subclass}[:dataSubsetLength_{subclass}]$
        $dataSubsetIndices_{superclass} \leftarrow dataIndices_{subclass}[\text{-}dataSubsetLength_{superclass}:]$
        $classToDataIndices[subclass] \leftarrow dataIndices_{subclass}$
        $classToDataIndices[superclass] \leftarrow classToDataIndices[superclass] \cup dataSubsetIndices_{superclass}$
    end
    else if $subclass$ has no superclass **then**
        $classToDataIndices$[subclass] $\leftarrow dataIndices_{subclass}$
    end
end

IncrementTask:
$currentTaskId \leftarrow currentTaskId + 1$
$dataIndices_{task} \leftarrow []$
for $class$ in $tasks[currentTaskId]$ **do**
    $dataIndices_{task} \leftarrow dataIndices_{task} \cup classToDataIndices[class]$
end

GetItem:
Require: $classes_{task}$ // The classes present in the current task
input : $index$ // an index in the range of length of $dataIndices_{task}$
$image, labels \leftarrow multilabelData[dataIndices_{task}[index]]$
$label \leftarrow labels \cap classes_{task}$
output : $image$ // The sample image
output : $label$ // The label corresponding to this image that exists in the current task

Algorithm 4: CompleteInfoIncrementalTestDataset

input : $multilabelData$ // A list of samples with each sample in the form of ($image$, ($superclassLabel$, $subclassLabel$)) or ($image$, ($subclassLabel$)))
input : $superclassToSubclass$ // a mapping that maps superclasses to their constituent subclasses
input : $tasks$ // The classes available at each task
Require: $subclasses$ // All refined subclasses (those who have a superclass as well as those who don't)
output : a test dataset object which keeps collecting data along the tasks

Initialization:
$classToDataIndices \leftarrow$ empty_dictionary
$classes_{observed} \leftarrow []$
$dataIndices_{accessible} \leftarrow []$
for $subclass$ in $subclasses$ **do**
    /* get the indices of the samples which correspond to this subclass */
    $dataIndices_{subclass} \leftarrow$ GetSamplesIndices($multilabelData$, $subclass$)
    $classToDataIndices[subclass] \leftarrow dataIndices_{subclass}$
    if subclass has superclass **then**
        $classToDataIndices[superclass] \leftarrow classToDataIndices[superclass] \cup classToDataIndices[subclass]$
    end
end

LoadTask
input : $taskId$ // The index of the task to load
$dataIndices_{accessible} \leftarrow []$
$classes_{observed} \leftarrow classes_{observed} \cup tasks[taskId]$
for $class$ in $tasks[taskId]$ **do**
    $dataIndices_{accessible} \leftarrow dataIndices_{accessible} \cup classToDataIndices[class]$
end

LoadAllObservedData
Require: $classes_{observed}$ // All classes observed till now in all previous tasks
$dataIndices_{accessible} \leftarrow []$
for $class$ in $classes_{observed}$ **do**
    $dataIndices_{accessible} \leftarrow dataIndices_{accessible} \cup classToDataIndices[class]$
end
$dataIndices_{accessible} \leftarrow$ RemoveDuplicates($dataIndices_{accessible}$)

GetItem
Require: $classes_{observed}$ // All classes observed till now in all previous tasks
input : $index$ // an index in the range of task_data_indices
$image, labels \leftarrow multilabelData[dataIndices_{accessible}[index]]$
$labels \leftarrow labels \cap classes_{observed}$
output : $image$ // The sample image
output : $labels$ // The labels corresponding to this image that exist in the $classes_{observed}$

Appendix B

IIRC Datasets Hierarchies

B.1. IIRC-CIFAR Hierarchy

superclass	subclasses
aquatic mammals	beaver, dolphin, otter, seal, whale
fish	aquarium fish, flatfish, ray, shark, trout
flowers	orchid, poppy, rose, sunflower, tulip
food containers	bottle, bowl, can, cup, plate
fruit and vegetables	apple, orange, pear, sweet pepper
household furniture	bed, chair, couch, table, wardrobe
insects	bee, beetle, butterfly, caterpillar, cockroach
large carnivores	leopard, lion, tiger, wolf
large omnivores and herbivores	bear, camel, cattle, chimpanzee, elephant, kangaroo
medium sized mammals	fox, porcupine, possum, raccoon, skunk
people	baby, boy, girl, man, woman
reptiles	crocodile, dinosaur, lizard, snake, turtle
small mammals	hamster, mouse, rabbit, shrew, squirrel
trees	maple tree, oak tree, palm tree, pine tree, willow tree
vehicles	bicycle, bus, motorcycle, pickup truck, train, streetcar, tank, tractor
-	mushroom, clock, keyboard, lamp, telephone, television, bridge, castle, house, road, skyscraper, cloud, forest, mountain, plain, sea, crab, lobster, snail, spider, worm, lawn mower, rocket

B.2. IIRC-ImageNet Hierarchy

superclass	subclasses
dog	dalmatian, basenji, pug, Leonberg, Newfoundland, Great Pyrenees, Mexican hairless, Brabancon griffon, Pembroke, Cardigan, Chihuahua, Japanese spaniel, Maltese dog, Pekinese, Shih-Tzu, toy terrier, papillon, Blenheim spaniel, Rhodesian ridgeback, boxer, bull mastiff, Great Dane, Saint Bernard, Eskimo dog, Tibetan mastiff, French bulldog, malamute, Siberian husky, Samoyed, Pomeranian, chow, keeshond, toy poodle, miniature poodle, standard poodle, Afghan hound, basset, beagle, bloodhound, bluetick, redbone, Ibizan hound, Norwegian elkhound, otterhound, Saluki, Scottish deerhound, Weimaraner, black-and-tan coonhound, Walker hound, English foxhound, borzoi, Irish wolfhound, Italian greyhound, whippet, Bedlington terrier, Border terrier, Kerry blue terrier, Irish terrier, Norfolk terrier, Norwich terrier, Yorkshire terrier, Airedale, cairn, Australian terrier, Dandie Dinmont, Boston bull, Scotch terrier, Tibetan terrier, silky terrier, soft-coated wheaten terrier, West Highland white terrier, Lhasa, Staffordshire bullterrier, American Staffordshire terrier, wire-haired fox terrier, Lakeland terrier, Sealyham terrier, German short-haired pointer, vizsla, kuvasz, schipperke, Doberman, miniature pinscher, affenpinscher, Brittany spaniel, clumber, cocker spaniel, Sussex spaniel, English springer, Welsh springer spaniel, Irish water spaniel, English setter, Irish setter, Gordon setter, flat-coated retriever, curly-coated retriever, golden retriever, Labrador retriever, Chesapeake Bay retriever, miniature schnauzer, giant schnauzer, standard schnauzer, Greater Swiss Mountain dog, Bernese mountain dog, Appenzeller, EntleBucher, briard, kelpie, komondor, Old English sheepdog, Shetland sheepdog, collie, Border collie, Bouvier des Flandres, Rottweiler, German shepherd, groenendael, malinois

bird	cock, hen, ostrich, bee eater, hornbill, hummingbird, jacamar, toucan, coucal, quail, partridge, peacock, black grouse, ptarmigan, ruffed grouse, prairie chicken, water ouzel, robin, bulbul, jay, magpie, chickadee, brambling, goldfinch, house finch, junco, indigo bunting, black swan, European gallinule, goose, drake, red-breasted merganser, pelican, albatross, king penguin, spoonbill, flamingo, limpkin, bustard, white stork, black stork, American coot, oystercatcher, red-backed sandpiper, redshank, dowitcher, ruddy turnstone, little blue heron, bittern, American egret, African grey, macaw, sulphur-crested cockatoo, lorikeet, vulture, kite, bald eagle, great grey owl
garment	suit, abaya, kimono, cardigan, feather boa, stole, jersey, sweatshirt, poncho, brassiere, jean, gown, military uniform, pajama, apron, academic gown, vestment, bow tie, Windsor tie, fur coat, lab coat, trench coat, hoopskirt, miniskirt, overskirt, sarong, cloak
beverage	espresso, red wine, cup, eggnog
aircraft	airship, balloon, airliner, warplane, wing, space shuttle
bear	brown bear, American black bear, ice bear, sloth bear
fox	red fox, kit fox, Arctic fox, grey fox
wolf	timber wolf, white wolf, red wolf, coyote
bag	backpack, mailbag, plastic bag, purse, sleeping bag
footwear	clog, cowboy boot, Loafer, running shoe, sandal
toiletry	hair spray, lotion, perfume, face powder, sunscreen, lipstick
box	carton, chest, crate, mailbox, pencil box, safe
rodent	hamster, porcupine, marmot, beaver, guinea pig, fox squirrel
bottle	beer bottle, pill bottle, pop bottle, water bottle, wine bottle, water jug, whiskey jug
fabric	velvet, wool, bib, dishrag, handkerchief, bath towel, paper towel
cup	beer glass, goblet, cocktail shaker, measuring cup, pitcher, beaker, coffee mug
fungus	coral fungus, gyromitra, stinkhorn, earthstar, hen-of-the-woods, bolete, agaric
musteline	weasel, mink, polecat, black-footed ferret, otter, skunk, badger

truck	fire engine, garbage truck, pickup, tow truck, trailer truck, moving van, police van, recreational vehicle, forklift, harvester, snowplow, tractor
headdress	crash helmet, football helmet, bearskin, bonnet, cowboy hat, sombrero, bathing cap, mortarboard, shower cap, pickelhaube
ball	baseball, basketball, croquet ball, golf ball, ping-pong ball, punching bag, rugby ball, soccer ball, tennis ball, volleyball
car	ambulance, beach wagon, cab, convertible, jeep, limousine, Model T, racer, sports car, minivan, grille, golfcart
measuring instrument	barometer, scale, odometer, rule, sundial, digital watch, hourglass, parking meter, stopwatch, analog clock, digital clock, wall clock
tool	hammer, plunger, screwdriver, shovel, cleaver, letter opener, can opener, corkscrew, hatchet, chain saw, plane, scabbard, power drill, carpenter's kit
watercraft	schooner, catamaran, trimaran, fireboat, gondola, canoe, yawl, lifeboat, speedboat, pirate, wreck, container ship, liner, aircraft carrier, submarine, amphibian, paddle
dish	Petri dish, mixing bowl, soup bowl, tray
bus	minibus, school bus, trolleybus
cart	horse cart, jinrikisha, oxcart
tracked vehicle	snowmobile, half track, tank
lamp	candle, spotlight, jack-o'-lantern, lampshade, table lamp
optical instrument	binoculars, projector, sunglasses, lens cap, loupe, Polaroid camera, reflex camera
gymnastic apparatus	balance beam, horizontal bar, parallel bars
swine	hog, wild boar, warthog
rabbits	hare, wood rabbit, Angora
echinoderm	starfish, sea urchin, sea cucumber
wild dog	dingo, dhole, African hunting dog
pouched mammal	wombat, wallaby, koala
aquatic mammal	dugong, grey whale, killer whale, sea lion
person	ballplayer, scuba diver, groom
mollusk	chiton, chambered nautilus, conch, snail, slug, sea slug

weapon	bow, projectile, cannon, missile, rifle, revolver, assault rifle, holster
bovid	bison, water buffalo, ram, ox, bighorn, ibex, hartebeest, impala, gazelle
salamander	European fire salamander, common newt, eft, spotted salamander, axolotl
frog	tree frog, tailed frog, bullfrog
big cat	leopard, snow leopard, jaguar, lion, tiger, cheetah
domestic cat	tabby, tiger cat, Persian cat, Siamese cat, Egyptian cat
cooking utensil	spatula, frying pan, wok, Crock Pot, Dutch oven, caldron, coffeepot, teapot
primate	Madagascar cat, indri, gibbon, siamang, orangutan, gorilla, chimpanzee, marmoset, capuchin, howler monkey, titi, spider monkey, squirrel monkey, guenon, patas, baboon, macaque, langur, colobus, proboscis monkey
fish	barracouta, electric ray, stingray, hammerhead, great white shark, tiger shark, sturgeon, gar, puffer, rock beauty, anemone fish, lionfish, eel, tench, goldfish, coho
lizard	banded gecko, common iguana, American chameleon, whiptail, agama, frilled lizard, alligator lizard, Gila monster, green lizard, African chameleon, Komodo dragon
turtle	mud turtle, terrapin, box turtle, loggerhead, leatherback turtle
spider	black and gold garden spider, barn spider, garden spider, black widow, tarantula, wolf spider, spider web
insect	ringlet, sulphur butterfly, lycaenid, cabbage butterfly, monarch, admiral, dragonfly, damselfly, lacewing, cicada, leafhopper, cockroach, mantis, walking stick, grasshopper, cricket, bee, ant, fly, tiger beetle, ladybug, ground beetle, long-horned beetle, leaf beetle, weevil, dung beetle, rhinoceros beetle
green groceries	acorn, hip, ear, fig, pineapple, banana, jackfruit, custard apple, pomegranate, strawberry, orange, lemon, Granny Smith, buckeye, rapeseed, corn, cucumber, artichoke, cardoon, mushroom, bell pepper, mashed potato, zucchini, spaghetti squash, acorn squash, butternut squash, broccoli, cauliflower, head cabbage
keyboard instrument	accordion, organ, grand piano, upright

percussion instrument	chime, drum, gong, maraca, marimba, steel drum
stringed instrument	banjo, acoustic guitar, electric guitar, cello, violin, harp
wind instrument	ocarina, harmonica, flute, panpipe, bassoon, oboe, sax, cornet, French horn, trombone
crustacean	isopod, crayfish, hermit crab, spiny lobster, American lobster, Dungeness crab, rock crab, fiddler crab, king crab
pen	ballpoint, fountain pen, quill
display	desktop computer, laptop, notebook, screen, television, monitor
electronic equipement	cassette player, CD player, modem, oscilloscope, tape player, iPod, printer, joystick, dial telephone, pay-phone, cellular telephone, mouse, hand-held computer
snake	sea snake, horned viper, boa constrictor, rock python, Indian cobra, green mamba, diamondback, sidewinder, thunder snake, ringneck snake, hognose snake, green snake, king snake, garter snake, water snake, vine snake, night snake
geological formation	cliff, geyser, lakeside, seashore, valley, promontory, alp, volcano, coral reef, sandbar
food	dough, guacamole, chocolate sauce, carbonara, French loaf, bagel, pretzel, plate, trifle, ice cream, ice lolly, pizza, potpie, burrito, consomme, hot pot, hotdog, cheeseburger, meat loaf
white home appliances	dishwasher, refrigerator, washer, stove
kitchen appliances	microwave, toaster, waffle iron, espresso maker
wheel	car wheel, paddlewheel, pinwheel, potter's wheel, reel, disk brake
seat	toilet seat, studio couch, park bench, barber chair, folding chair, rocking chair, throne
baby bed	bassinet, cradle, crib
cabinet	medicine chest, wardrobe, china cabinet, bookcase, chiffonier, file, entertainment center, plate rack
table	desk, pool table, dining table
bridges	steel arch bridge, suspension bridge, viaduct
fence	chainlink fence, picket fence, stone wall, worm fence
long structures	beacon, obelisk, totem pole
movable homes	mountain tent, mobile home, yurt

building	planetarium, barn, cinema, boathouse, palace, monastery, castle, dome, church, mosque, stupa, bell cote, thatch, tile roof, triumphal arch
body armor	chain mail, cuirass, bulletproof vest, breastplate
mask	mask, oxygen mask, gasmask, ski mask
curtain-screen	window shade, shower curtain, theater curtain
bike	moped, bicycle-built-for-two, tricycle, unicycle, mountain bike, motor scooter
train	passenger car, freight car, electric locomotive, bullet train, streetcar, steam locomotive
swimsuit	bikini, maillot, swimming trunks
socks mittens	Christmas stocking, mitten, sock
keyboard	computer keyboard, space bar, typewriter keyboard

-	African crocodile, American alligator, triceratops, trilobite, harvestman, scorpion, tick, centipede, tusker, echidna, platypus, jellyfish, sea anemone, brain coral, flatworm, nematode, crane, hyena, cougar, lynx, mongoose, meerkat, sorrel, zebra, hippopotamus, Arabian camel, llama, armadillo, three-toed sloth, Indian elephant, African elephant, lesser panda, giant panda, abacus, altar, apiary, ashcan, bakery, Band Aid, bannister, barbell, barbershop, barrel, barrow, bathtub, binder, birdhouse, bobsled, bolo tie, bookshop, bottlecap, brass, breakwater, broom, bucket, buckle, butcher shop, car mirror, carousel, cash machine, cassette, chain, cliff dwelling, coil, combination lock, confectionery, crutch, dam, diaper, dock, dogsled, doormat, drilling platform, drumstick, dumbbell, electric fan, envelope, fire screen, flagpole, fountain, four-poster, gas pump, go-kart, greenhouse, grocery store, guillotine, hair slide, hamper, hand blower, hard disc, home theater, honeycomb, hook, iron, jigsaw puzzle, knee pad, knot, ladle, lawn mower, library, lighter, loudspeaker, lumbermill, magnetic compass, manhole cover, matchstick, maypole, maze, megalith, microphone, milk can, mortar, mosquito net, mousetrap, muzzle, nail, neck brace, necklace, nipple, oil filter, packet, padlock, paintbrush, parachute, patio, pedestal, pencil sharpener, photocopier, pick, pier, piggy bank, pillow, plow, pole, pot, prayer rug, prison, puck, quilt, racket, radiator, radio, radio telescope, rain barrel, remote control, restaurant, rotisserie, rubber eraser, safety pin, saltshaker, scoreboard, screw, seat belt, sewing machine, shield, shoe shop, shoji, shopping basket, shopping cart, ski, slide rule, sliding door, slot, snorkel, soap dispenser, solar dish, space heater, spindle, stage, stethoscope, strainer, stretcher, sunglass, swab, swing, switch, syringe, teddy, thimble, thresher, tobacco shop, torch, toyshop, tripod, tub, turnstile, umbrella, vacuum, vase, vault, vending machine, wallet, washbasin, water tower, whistle, wig, window screen, wooden spoon, web site, comic book, crossword puzzle, street sign, traffic light, book jacket, menu, hay, bubble, daisy, yellow lady's slipper, toilet tissue

Appendix C

Results Raw Data

task	model						
	ER	iCaRL-CNN	iCaRL-norm	LUCIR	AGEM	incremental joint	ER-infinite
0	0.72 (0.039)	0.71 (0.042)	0.75 (0.037)	0.74 (0.036)	0.72 (0.032)	0.71 (0.032)	0.72 (0.033)
1	0.31 (0.046)	0.47 (0.032)	0.5 (0.035)	0.5 (0.101)	0.15 (0.024)	0.7 (0.035)	0.64 (0.067)
2	0.23 (0.054)	0.36 (0.017)	0.39 (0.024)	0.35 (0.178)	0.1 (0.023)	0.67 (0.029)	0.63 (0.029)
3	0.2 (0.027)	0.3 (0.022)	0.32 (0.026)	0.3 (0.162)	0.07 (0.028)	0.66 (0.021)	0.58 (0.054)
4	0.17 (0.024)	0.26 (0.022)	0.27 (0.024)	0.27 (0.148)	0.05 (0.01)	0.66 (0.02)	0.55 (0.04)
5	0.18 (0.024)	0.23 (0.017)	0.24 (0.027)	0.25 (0.13)	0.07 (0.034)	0.65 (0.019)	0.55 (0.027)
6	0.19 (0.03)	0.21 (0.022)	0.21 (0.028)	0.23 (0.133)	0.07 (0.031)	0.64 (0.02)	0.52 (0.02)
7	0.17 (0.035)	0.19 (0.021)	0.19 (0.027)	0.19 (0.141)	0.06 (0.045)	0.63 (0.029)	0.51 (0.019)
8	0.16 (0.02)	0.18 (0.02)	0.18 (0.026)	0.18 (0.135)	0.04 (0.014)	0.63 (0.022)	0.49 (0.026)
9	0.15 (0.021)	0.17 (0.018)	0.18 (0.022)	0.15 (0.134)	0.04 (0.018)	0.63 (0.021)	0.47 (0.033)
10	0.17 (0.035)	0.16 (0.017)	0.17 (0.017)	0.14 (0.118)	0.06 (0.039)	0.62 (0.019)	0.45 (0.036)
11	0.15 (0.018)	0.16 (0.017)	0.16 (0.018)	0.13 (0.117)	0.04 (0.019)	0.62 (0.023)	0.44 (0.043)
12	0.15 (0.03)	0.16 (0.017)	0.16 (0.015)	0.12 (0.109)	0.05 (0.033)	0.62 (0.022)	0.43 (0.035)
13	0.15 (0.025)	0.15 (0.016)	0.16 (0.016)	0.13 (0.094)	0.05 (0.025)	0.62 (0.016)	0.43 (0.02)
14	0.14 (0.017)	0.15 (0.011)	0.15 (0.014)	0.11 (0.096)	0.04 (0.022)	0.62 (0.014)	0.42 (0.028)
15	0.13 (0.012)	0.15 (0.01)	0.15 (0.014)	0.11 (0.094)	0.03 (0.012)	0.62 (0.012)	0.4 (0.02)
16	0.14 (0.011)	0.15 (0.013)	0.15 (0.015)	0.09 (0.094)	0.03 (0.005)	0.63 (0.013)	0.39 (0.037)
17	0.14 (0.019)	0.15 (0.013)	0.15 (0.013)	0.07 (0.084)	0.04 (0.021)	0.63 (0.017)	0.39 (0.011)
18	0.14 (0.012)	0.15 (0.012)	0.15 (0.012)	0.06 (0.081)	0.03 (0.014)	0.63 (0.013)	0.37 (0.01)
19	0.14 (0.019)	0.15 (0.009)	0.14 (0.012)	0.06 (0.075)	0.03 (0.012)	0.63 (0.007)	0.36 (0.009)
20	0.13 (0.011)	0.15 (0.01)	0.14 (0.011)	0.06 (0.072)	0.03 (0.015)	0.63 (0.007)	0.35 (0.011)
21	0.13 (0.005)	0.15 (0.01)	0.15 (0.01)	0.06 (0.071)	0.02 (0.003)	0.63 (0.008)	0.35 (0.012)

Table C.1. The average performance on IIRC-CIFAR after each task using the precision-weighted Jaccard Similarity. This table represents the same results as in Figure 4.2 with the standard deviation between brackets

task	model				
	ER	iCaRL-CNN	iCaRL-norm	LUCIR	incremental joint
0	0.7 (0.027)	0.78 (0.018)	0.8 (0.019)	0.76 (0.025)	0.73 (0.02)
1	0.13 (0.022)	0.46 (0.039)	0.49 (0.034)	0.17 (0.044)	0.73 (0.026)
2	0.12 (0.071)	0.34 (0.047)	0.38 (0.041)	0.15 (0.048)	0.73 (0.019)
3	0.08 (0.01)	0.27 (0.035)	0.31 (0.024)	0.14 (0.045)	0.73 (0.012)
4	0.08 (0.01)	0.23 (0.022)	0.27 (0.015)	0.1 (0.068)	0.73 (0.015)
5	0.07 (0.012)	0.2 (0.018)	0.25 (0.014)	0.1 (0.063)	0.73 (0.01)
6	0.07 (0.017)	0.18 (0.018)	0.23 (0.017)	0.06 (0.062)	0.73 (0.01)
7	0.06 (0.004)	0.17 (0.013)	0.22 (0.013)	0.04 (0.057)	0.73 (0.013)
8	0.07 (0.013)	0.16 (0.01)	0.21 (0.01)	0.03 (0.056)	0.72 (0.015)
9	0.06 (0.002)	0.16 (0.011)	0.2 (0.01)	0.03 (0.053)	0.72 (0.017)

Table C.2. The average performance on IIRC-ImageNet-lite after each task using the precision-weighted Jaccard Similarity. This table represents the same results as in Figure 1(a) with the standard deviation between brackets

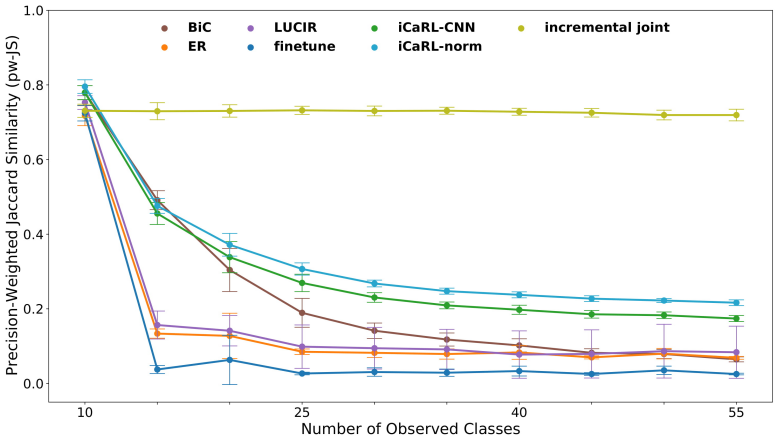
task	model				
	ER	iCaRL-CNN	iCaRL-norm	LUCIR	joint
0	0.7 (0.024)	0.78 (0.009)	0.8 (0.009)	0.75 (0.017)	-
1	0.13 (0.023)	0.47 (0.016)	0.51 (0.014)	0.15 (0.044)	-
2	0.1 (0.024)	0.34 (0.026)	0.39 (0.02)	0.14 (0.046)	-
3	0.08 (0.008)	0.27 (0.02)	0.32 (0.014)	0.13 (0.044)	-
4	0.08 (0.021)	0.23 (0.014)	0.28 (0.008)	0.12 (0.061)	-
5	0.1 (0.064)	0.21 (0.025)	0.26 (0.011)	0.11 (0.06)	-
6	0.07 (0.008)	0.19 (0.015)	0.23 (0.011)	0.1 (0.056)	-
7	0.06 (0.007)	0.17 (0.017)	0.22 (0.013)	0.11 (0.051)	-
8	0.06 (0.006)	0.16 (0.016)	0.21 (0.011)	0.1 (0.045)	-
9	0.06 (0.005)	0.15 (0.015)	0.2 (0.009)	0.09 (0.06)	-
10	0.06 (0.015)	0.15 (0.019)	0.19 (0.014)	0.08 (0.054)	-
11	0.06 (0.005)	0.14 (0.017)	0.19 (0.012)	0.08 (0.05)	-
12	0.06 (0.011)	0.14 (0.012)	0.19 (0.008)	0.08 (0.047)	-
13	0.06 (0.006)	0.13 (0.013)	0.19 (0.005)	0.07 (0.045)	-
14	0.06 (0.007)	0.13 (0.011)	0.18 (0.006)	0.07 (0.042)	-
15	0.05 (0.001)	0.13 (0.015)	0.18 (0.005)	0.07 (0.04)	-
16	0.05 (0.003)	0.12 (0.018)	0.18 (0.007)	0.06 (0.036)	-
17	0.05 (0.008)	0.12 (0.017)	0.17 (0.005)	0.06 (0.034)	-
18	0.05 (0.01)	0.12 (0.019)	0.17 (0.008)	0.05 (0.031)	-
19	0.05 (0.003)	0.11 (0.021)	0.17 (0.009)	0.05 (0.03)	-
20	0.05 (0.004)	0.11 (0.022)	0.16 (0.009)	0.05 (0.028)	-
21	0.04 (0.004)	0.11 (0.024)	0.16 (0.01)	0.04 (0.025)	-
22	0.04 (0.004)	0.1 (0.023)	0.16 (0.009)	0.04 (0.024)	-
23	0.04 (0.008)	0.1 (0.019)	0.15 (0.008)	0.04 (0.022)	-
24	0.04 (0.001)	0.1 (0.018)	0.15 (0.007)	0.03 (0.023)	-
25	0.03 (0.003)	0.1 (0.017)	0.14 (0.007)	0.02 (0.022)	-
26	0.04 (0.004)	0.09 (0.018)	0.13 (0.007)	0.02 (0.021)	-
27	0.03 (0.002)	0.09 (0.018)	0.12 (0.004)	0.02 (0.019)	-
28	0.03 (0.002)	0.08 (0.016)	0.12 (0.003)	0.02 (0.018)	-

29	0.03 (0.002)	0.08 (0.016)	0.11 (0.003)	0.02 (0.017)	-
30	0.03 (0.005)	0.08 (0.014)	0.11 (0.003)	0.02 (0.016)	-
31	0.02 (0.006)	0.08 (0.015)	0.1 (0.01)	0.02 (0.015)	-
32	0.02 (0.01)	0.08 (0.016)	0.09 (0.007)	0.02 (0.014)	-
33	0.02 (0.005)	0.08 (0.017)	0.09 (0.011)	0.02 (0.012)	-
34	0.01 (0.001)	0.07 (0.016)	0.08 (0.01)	0.01 (0.015)	0.42 (0.018)

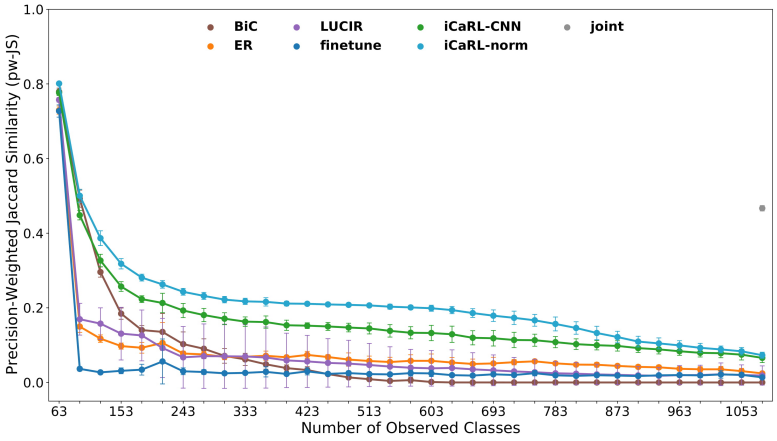
Table C.3. The average performance on IIRC-ImageNet-full after each task using the precision-weighted Jaccard Similarity. This table represents the same results as in Figure 1(b) with the standard deviation between brackets

Appendix D

Graphs with Standard Deviation



(a) IIRC-ImageNet-lite



(b) IIRC-ImageNet-full

Fig. D.1. Average performance on *IIRC-ImageNet-lite* and *IIRC-ImageNet-full* as measured by the precision-weighted Jaccard Similarity (Equation 3.4). (see Figure D.1 for the original figure)

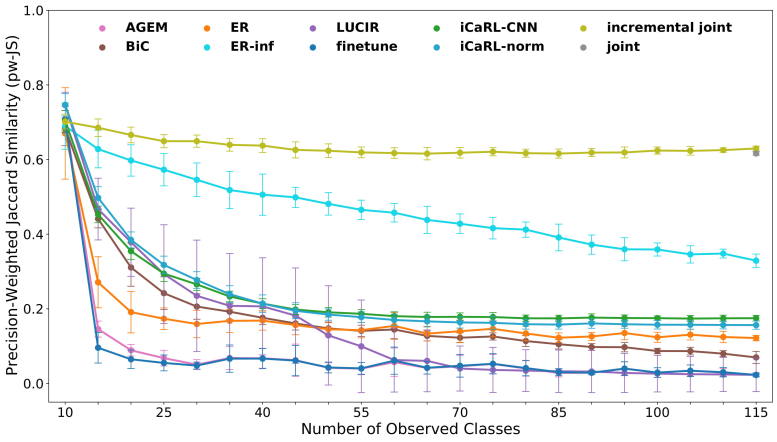
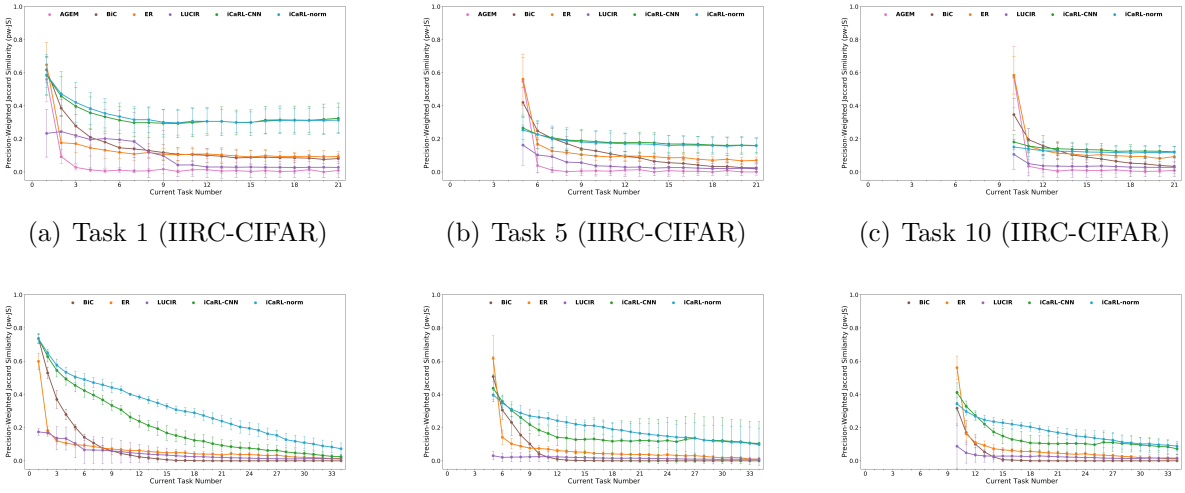


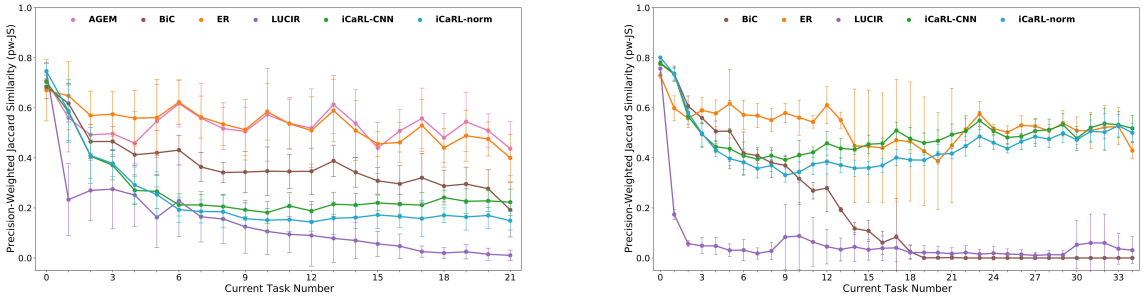
Fig. D.2. Average performance on *IIRC-CIFAR* as measured by the precision-weighted Jaccard Similarity (Equation 3.4). (see Figure D.2 for the original figure)



(a) Task 1 (IIRC-CIFAR) (b) Task 5 (IIRC-CIFAR) (c) Task 10 (IIRC-CIFAR)

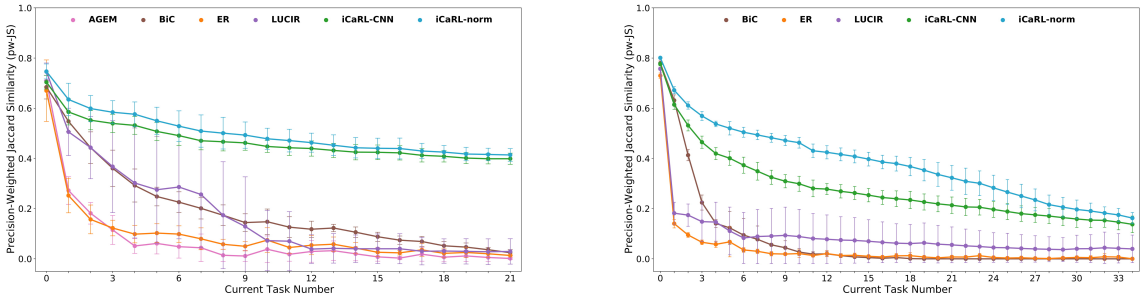
(d) Task 1 (IIRC-ImageNet-full) (e) Task 5 (IIRC-ImageNet-full) (f) Task 10 (IIRC-ImageNet-full)

Fig. D.3. The Performance of three middle tasks throughout the whole training process, to measure their catastrophic forgetting and backward transfer. Note that a degradation in performance is not necessarily caused by catastrophic forgetting, as a new subclass of a previously observed superclass might be introduced and the model would be penalized for not applying that label retroactively. (see Figure 4.3 for the original figure)



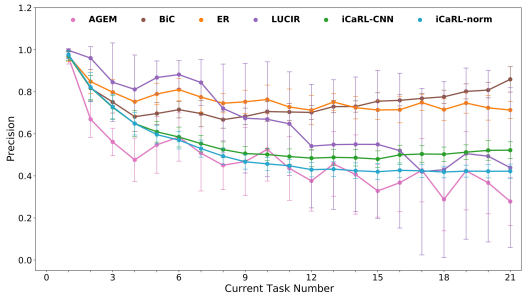
(a) IIRC-CIFAR (b) IIRC-ImageNet-full

Fig. D.4. Current task performance; Per task performance over the test samples of a specific task j, after training on that task (R_{jj} using Equation 3.3). see Figure 4.4 for the original figure)

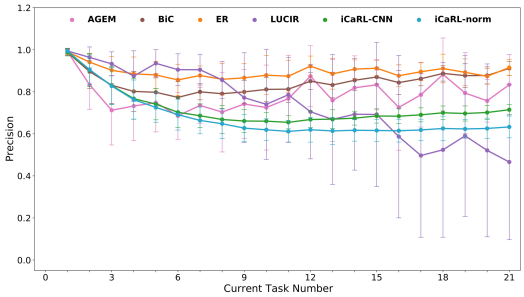


(a) IIRC-CIFAR (b) IIRC-ImageNet-full

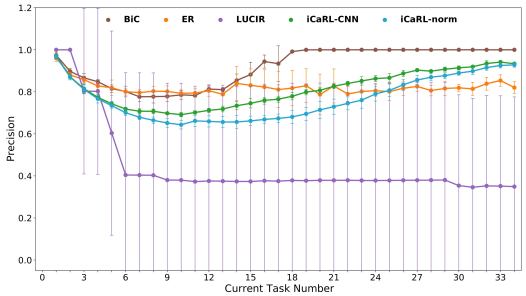
Fig. D.5. The average performance of *IIRC-ImageNet-full* and *IIRC-CIFAR* if only the superclasses are taken into account for calculating this performance. (see Figure 4.6 for the original figure)



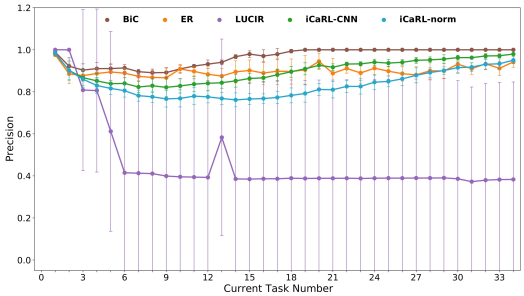
(a) Subclasses Precision (IIRC-CIFAR)



(b) Orphan Subclasses Precision (IIRC-CIFAR)

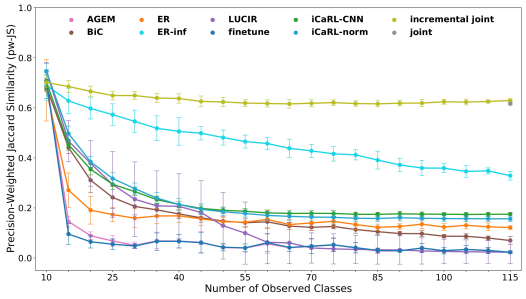


(c) Subclasses Precision (IIRC-ImageNet)

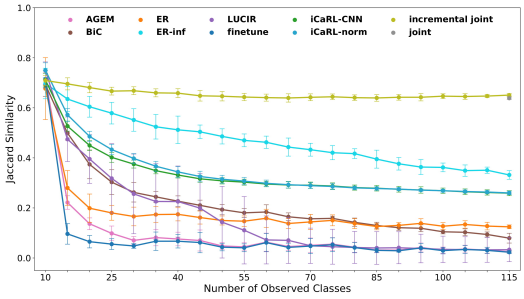


(d) Orphan Subclasses Precision (IIRC-ImageNet)

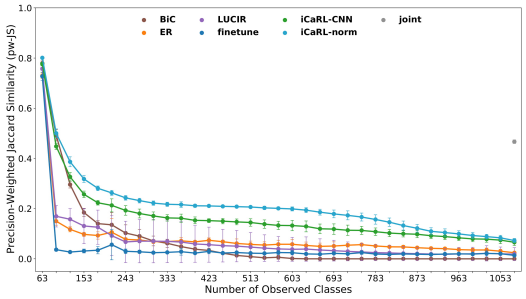
Fig. D.6. The average precision of *IIRC-CIFAR* and *IIRC-ImageNet-full* over each type of subclasses, excluding other types of classes, to measure how much do the models confuse the subclasses as they encounter more related subclasses. (see Figure 4.7 for the original figure)



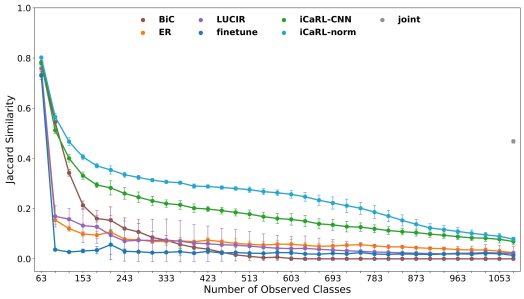
(a) precision-weighted Jaccard Similarity (IIRC-CIFAR)



(b) Jaccard Similarity (IIRC-CIFAR)

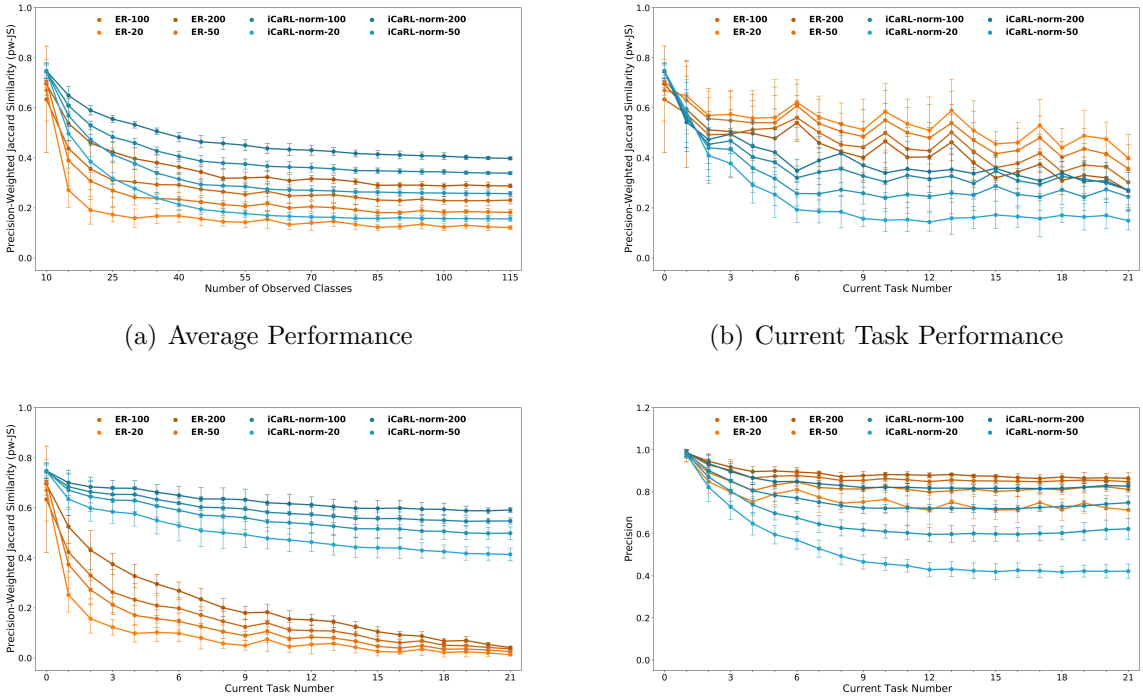


(c) precision-weighted Jaccard Similarity (IIRC-ImageNet-full)



(d) Jaccard Similarity (IIRC-Imagenet-full)

Fig. D.7. Average performance on *IIRC-CIFAR* and *IIRC-ImageNet-full*, as measured by the precision-weighted Jaccard Similarity compared to the Jaccard Similarity. (see Figure 4.8 for the original figure)



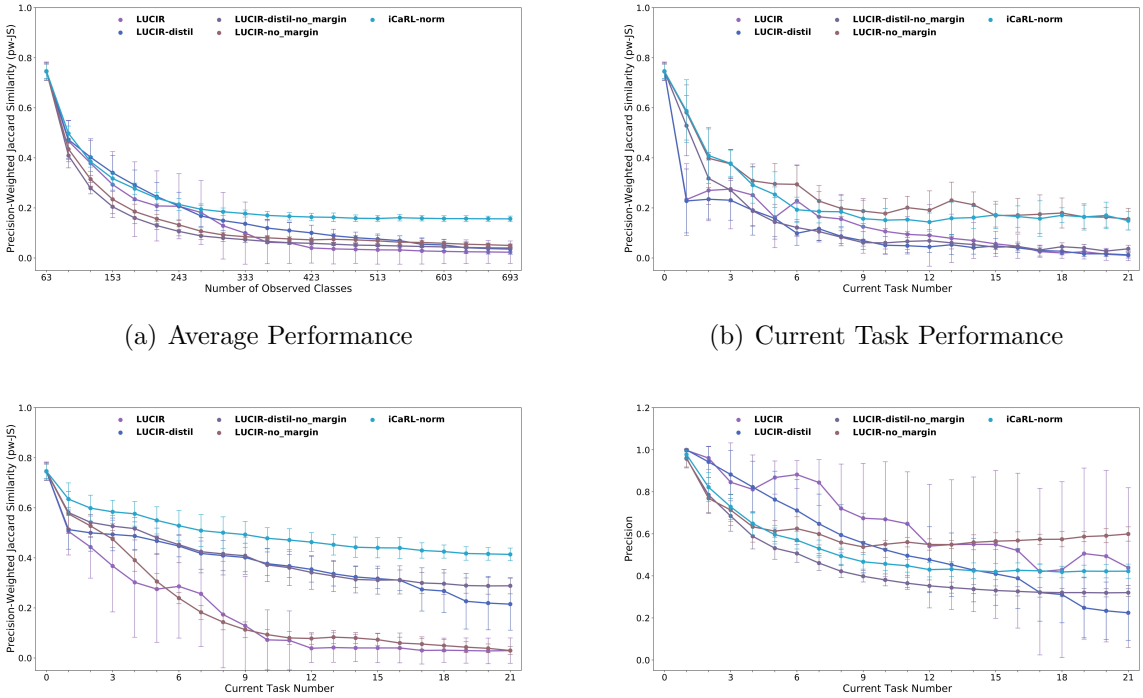
(a) Average Performance

(b) Current Task Performance

(c) Superclasses Performance

(d) Subclasses Precision

Fig. D.8. The effect of increasing the size of the buffer on the performance of *ER* and *iCaRL-norm* in *IIRC-CIFAR*. For each baseline, darker lines correspond to larger buffer size. (See Figure 4.9 for the original figure)



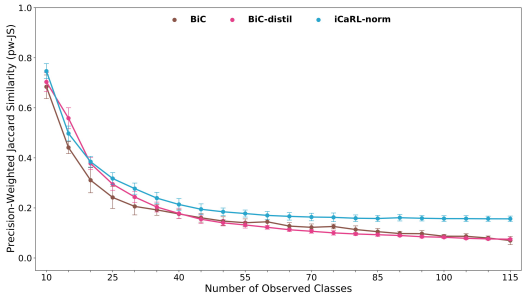
(a) Average Performance

(b) Current Task Performance

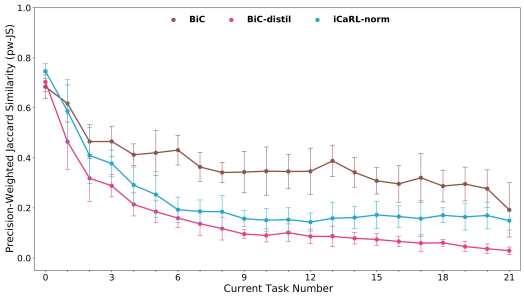
(c) Superclasses Performance

(d) Subclasses Precision

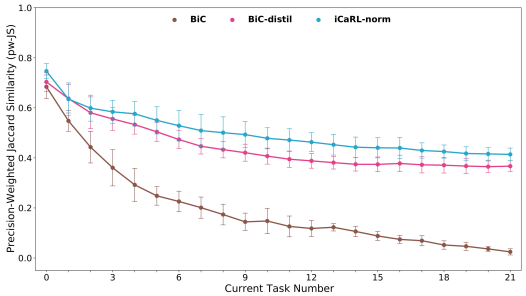
Fig. D.9. The performance of *LUCIR* after removing the margin ranking loss *LUCIR-no_margin*, using a distillation objective *LUCIR-distil*, and both *LUCIR-distil-no_margin*. (See Figure 4.10 for the original figure)



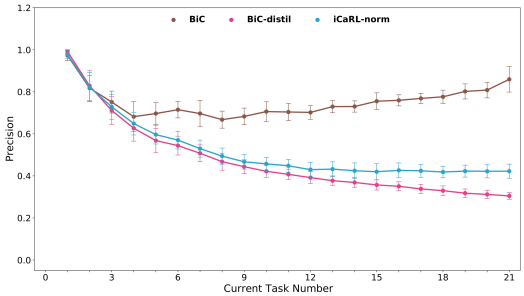
(a) Average Performance



(b) Current Task Performance



(c) Superclasses Performance



(d) Subclasses Precision

Fig. D.10. The performance of *BiC* after using a distillation objective during the bias correction phase *BiC-distil*. (See Figure 4.11 for the original figure)