AdvisIL - A Class-Incremental Learning Advisor

Eva Feillet[1], Grégoire Petit[1,2] Adrian Popescu[1], Marina Reyboz[3], Céline Hudelot[4]
[1]Université Paris-Saclay, CEA, LIST, F-91120, Palaiseau, France
[2]LIGM, Ecole des Ponts, Univ Gustave Eiffel, CNRS, Marne-la-Vallée, France
[3]Université Grenoble Alpes, CEA, LIST, F-38000 Grenoble, France
[4]Université Paris-Saclay, CentraleSupélec, MICS, France
{eva.feillet, gregoire.petit, adrian.popescu, marina.reyboz}@cea.fr,
celine.hudelot@centralesupelec.fr

Abstract

Recent class-incremental learning methods combine deep neural architectures and learning algorithms to handle streaming data under memory and computational constraints. The performance of existing methods varies depending on the characteristics of the incremental process. To date, there is no other approach than to test all pairs of learning algorithms and neural architectures on the training data available at the start of the learning process to select a suited algorithm-architecture combination. To tackle this problem, in this article, we introduce AdvisIL, a method which takes as input the main characteristics of the incremental process (memory budget for the deep model, initial number of classes, size of incremental steps) and recommends an adapted pair of learning algorithm and neural architecture. The recommendation is based on a similarity between the user-provided settings and a large set of precomputed experiments. AdvisIL makes class-incremental learning easier, since users do not need to run cumbersome experiments to design their system. We evaluate our method on four datasets under six incremental settings and three deep model sizes. We compare six algorithms and three deep neural architectures. Results show that AdvisIL has better overall performance than any of the individual combinations of a learning algorithm and a neural architecture. AdvisIL's code is available at `https://github.com/EvaJF/AdvisIL`.

1. Introduction

Practical applications of artificial intelligence often require handling data streams under computing power and memory constraints. This situation is typically encountered in embedded systems and real-time applications of computer vision [10, 16, 24, 38], which must handle new data over time with limited resources. Continual learning addresses this challenge as it aims to build models that can integrate new knowledge over time while preserving previously acquired knowledge. In the context of class-incremental learning (CIL), training a classification model is a sequential process where each step consists in integrating a set of new classes to the model. This process is particularly challenging in the exemplar-free setting (EFCIL), in which the model is updated without having access to examples of past classes. Among the works addressing CIL, many [19, 42, 45, 52, 55, 56, 57] adapt knowledge distillation [18] to this task, while others are based either on vanilla fine-tuning[4] or on a transfer learning scheme [2, 15].

A major challenge of CIL is that it is subject to catastrophic forgetting [23, 29], namely the tendency of learning algorithms to abruptly forget previously acquired information when confronted with new information. In order to learn reliable neural representations both for past and new classes, CIL algorithms must balance between information retention, i.e. stability, and information acquisition, i.e. plasticity. However, existing comparative studies [5, 28, 40] showed that when CIL algorithms are tested in different incremental scenarios, no method outperforms all others. In addition to the CIL algorithm itself, the main factors influencing the classification performance are the architecture of the backbone neural network and the characteristics of the CIL scenario i.e., the memory budget, the number of incremental steps, the number of classes in the initial step and the size of the subsequent incremental steps.

The performance variability for two scenarios is illustrated in Figure 1. It shows that the same combination of CIL algorithm and backbone network is not ranked the same from one EFCIL scenario to another. This variability is further highlighted in our experiments (Section 4). Thus, unlike other learning processes (i.e. classical, few-shot...) for which the study of the state of the art on evaluation benchmarks gives good indications for the selection of the model, EFCIL requires a more acute consideration of the learning scenario. These observations raise the following questions:

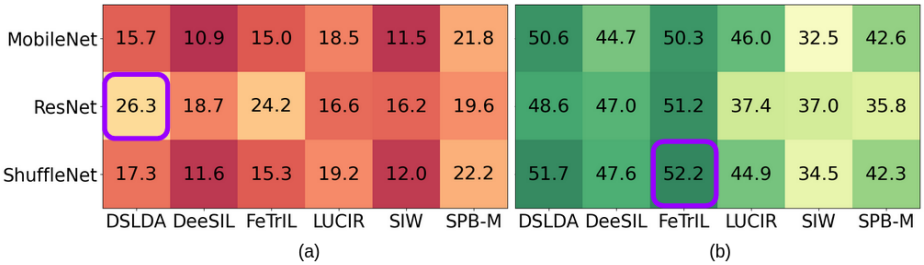

Figure 1: Classification performance in percent for various combinations of CIL algorithm and backbone network, averaged over five reference datasets containing 100 classes each in total, see Subsection 4.1. Scenario (a) has a memory budget of 1.5M parameters and consists of 20 steps with 5 classes each. Scenario (b) has a memory budget of 3.0M parameters, and consists of 4 steps with 25 classes each. Here, as highlighted in purple, scenario (a) is best handled by the combination of DSLDA and ResNet, while the combination of FeTrIL and ShuffleNet is a better match for scenario (b). As the same combinations of CIL algorithm and backbone network are not ranked the same from one scenario to another, it highlights the need for a recommendation method to select the best combination of CIL algorithm and backbone network depending on the scenario. Note that the scaling heuristic presented in Subsection 3.2 is used to fit neural networks to each memory budget.

1. *Since choosing the best CIL algorithm first requires characterizing the CIL scenario, what knowledge of this scenario may be realistically provided by the user?*

2. *Given a user's CIL scenario, how to select a suitable combination of learning algorithm and backbone network, without benchmarking each possible configuration?*

We argue that, regarding (1), the users have little knowledge about their data and can only approximately provide their CIL scenario's characteristics. Based on this assumption, we propose to tackle the selection issue (2) as a recommendation problem. We develop a user-centric method, called AdvisIL, that recommends a combination of an exemplar-free CIL algorithm and a backbone network scaled to the user's needs (Figure 2). Based on a set of pre-computed EFCIL experiments, AdvisIL provides a recommendation as follows:

1. The users specify their incremental learning settings (memory budget, number of steps, number of initial classes and size of the incremental update),

2. The pre-computed experiments with settings closest to the user's settings are selected,

3. The result is the combination of EFCIL algorithm and backbone network that ranks highest in terms of classification performance with respect to the selected experiments.

As a result, our recommendation method facilitates the use of CIL approaches since the user only provides the essential information about the incremental process. It prevents the users from benchmarking each CIL algorithm and backbone network, hence saving them time and computation efforts. AdvisIL is assessed by an evaluation protocol which uses four test datasets and eighteen scenarios, allowing us to highlight the relevance of our recommendations in a variety of experimental settings. To allow the use and the enrichment of AdvisIL by the community, and thus the quality of its recommendations, we will share the code and the pre-computed experiments on which the method is based.

2. Related work

2.1. CIL algorithms

Various approaches have been proposed to learn a classification problem incrementally while avoiding catastrophic forgetting (see the recent surveys of [5, 9, 28]).

The different approaches can be divided into two families, depending on whether they use knowledge distillation or rely on other strategies to reduce forgetting. In our study, algorithms of both types are used, *always without keeping past examples in memory*.

Approaches based on knowledge distillation. Knowledge distillation, as introduced in the seminal paper of Hinton [18], is a principle that consists in transferring the knowledge from a large "teacher" model into a smaller "student" model. In the case of incremental learning, it has been first used in the approach named "Learning without Forgetting (LwF)" [25] in which, at each learning step, the outputs of the neural network are constrained through regularization to remain close to the outputs obtained in the previous step. Inspired by LwF, iCaRL [42] relies on a strategy called *rehearsal* or *replay*, that consists in keeping in memory a limited number of training examples from past classes. In iCaRL, the neural network is used as a feature extractor and the authors substitute the classical fully-connected output layer by a nearest class mean classifier [31]. Most of the approaches that emerged after iCaRL improve classification performance by modifying the loss function associated to the neural network. The LwM approach [12] introduces an attention mechanism. The BiC approach [54] tries to rebalance the classification in favour of past classes by adding a new layer to correct the bias which favours more recent classes. Knowledge distillation was applied to the feature space in LUCIR [19], or to neural model weights in PodNet [13]. LUCIR combines a cosine classifier framework [41] and an inter-class separability component [44]. It inspired the authors of SPB [52], who combine an Euclidean-

based component to the loss function and reciprocal adaptive weights to ensure a good balance between the stability and plasticity of the incremental process.

Other approaches. The authors of [3, 4, 15, 39] are inspired by transfer learning [46] and freeze the feature extractor after the initial step. DSLDA [15] adapts linear discriminant analysis [36] for training incrementally the output layer of a deep neural network. DeeSIL [3] trains SVM classifiers for each new class against the other classes observed in the same learning step. SIW [4] discards the distillation component, and uses vanilla fine-tuning to simplify the CIL process. It stores the initial classifiers learned for each new class and reuses a normalized version of them in subsequent incremental steps. Recent method FeTrIL [39] combines a fixed feature extractor and a pseudo-feature generator based on a geometric translation for past classes to improve inter-class separation.

2.2. Relevance of architecture choice in CIL

The authors of [28] perform experiments with different backbone networks to test CIL algorithms. They observe that, for a given algorithm, the choice of the backbone network has an influence on classification accuracy. This highlights the need to take both aspects into consideration when training CIL models.

In this line of research, recent experiments of [32, 33] explore the influence of network architecture on the performance of incremental learning models. The authors of [32] showed that increasing the width of neural networks mitigates forgetting and increases model accuracy. Furthermore, they observed that increasing depth has no or negative effect on the performance of the model. These observations were confirmed by [33] where the influence of classical architectural components such as batch normalization is studied. Note that the results reported in these two papers are mostly obtained with overparameterized models and without the use of a CIL algorithm.

For classic supervised learning, methods for finding an architecture ensuring good performances and respecting a memory budget have been developed e.g. Neural Architecture Search (NAS) [14, 47], pruning [34], or compound scaling as proposed for EfficientNets [48]. However, these three approaches are not adapted to CIL. NAS for CIL methods [21, 20] are generally time-consuming and computationally expensive, as they explore a large space of possible network architectures and require training of candidate models to evaluate an architecture. They are not suitable for the practical case of a model adapting quickly to a dynamic environment, which is encountered in CIL. As in the case of NAS, the problem of data availability arises for pruning. Finally, compound scaling requires the calibration of hyperparameters, a costly and approximate process in the case where the user has access to only a subset of the classes. This motivates us to propose a recommendation method that requires only few pieces of information and no hyperparameter calibration by the user.

3. Proposed method

In this section, we first describe a CIL process and define the settings of a CIL experiment as a reference configuration. Then, we introduce a heuristic to scale an existing neural architecture so that it fits a given memory budget. In this article, any CIL experiment involves a backbone network scaled with this heuristic. Finally, we introduce our recommendation method AdvisIL, which takes as input the incremental learning settings provided by users and provides them with a combination of EFCIL algorithm and backbone adapted to their needs. This recommendation is based on pre-computed experiments corresponding to a set of reference configurations.

3.1. Defining a CIL process

The objective of CIL is to train a model that integrates all classes of a dataset that arrives in a stream. Without loss of generality, we assume that examples of past classes cannot be stored (exemplar-free setting), and that the model relies on a convolutional backbone network b.

We consider a sequential learning process composed of k non-overlapping steps $s_1, s_2, \ldots, s_k$. A step s_i consists in learning from the examples contained in a dataset D_i. To each dataset D_i is associated a set P_i of classes, such that each learning example in D_i uses a class belonging to P_i. The datasets composing the complete dataset $\mathcal{D} = D_1 \cup D_2 \cup \cdots \cup D_k$ satisfy the following constraint: for $i, j \in \{1, 2, \ldots, k\}$ with $i \neq j$, $P_i \cap P_j = \emptyset$. The number of classes of P_1, which corresponds to the initial step, is denoted α. Following a common assumption in CIL [5, 8], the sets $P_2, P_3, \ldots, P_k$, which correspond to the incremental steps $s_2, s_3, \ldots, s_k$, have the same number of classes, which we denote β. For our experiments, we express a memory budget m as the maximum number of parameters allowed for the backbone network at the final step s_k. Thus, this memory constraint is considered at inference time rather than at training time.

Definition 1 *A CIL scenario is a quadruplet (m, k, α, β) composed of a memory budget m, a number of steps k, a number of initial classes α, and an incremental update size β. It describes the main settings that users may be able to provide to design their incremental learning system.*

In this article, the training of an incremental classification model on $\mathcal{D}$, follows a scenario (m, k, α, β), uses an EFCIL algorithm a and relies on a backbone network b, such that the number of parameters of b is less or equal than m. All these parameters form a *configuration*.

Definition 2 *A configuration is defined as an 7-uplet* $(a, b, \mathcal{D}, m, k, \alpha, \beta)$ *containing: the name of an algorithm* a*, the name of a backbone network* b*, a complete dataset* $\mathcal{D}$*, the values corresponding to a memory budget* m*, a number of steps* k*, a number of initial classes* α *and an incremental update size* β*.*

The training of this model is performed as follows. At the first step s_1, the model $\mathcal{M}_1$ is trained on the dataset D_1 that involves α classes. For each of the next steps, the same procedure is applied. For $i = 2, 3, \ldots, k$, at the step s_i, the model $\mathcal{M}_i$ first recovers the weights from the model $\mathcal{M}_{i-1}$ that we obtained in the previous step. It is then trained using the examples of the dataset D_i involving β new classes.

The overall classification performance r on $\mathcal{D}$ of the final model $\mathcal{M}_k$ is called the average incremental accuracy and is computed by:

$$r = \frac{1}{k-1} \sum_{i=2}^{k} q_i, \tag{1}$$

where q_i is the accuracy of the model $\mathcal{M}_i$ on $D_1 \cup D_2 \cup \cdots \cup D_i$, after performing the learning step s_i. Let us remark that each dataset used in this article is subdivided in a training subset and a test subset, and that reported results are always computed on its test subset.

3.2. Scaling backbone networks for CIL

In any experiment involving a memory budget m, we want to obtain a configuration (Definition 2), where the number of parameters of the backbone network used is less or equal than m. To address this requirement, we introduce a scaling heuristic to fit existing convolutional architectures to a given memory budget m.

We adopt the point of view of users who has a memory budget m and aims at learning incrementally a classification model. They want to avoid any preliminary calculation, so they select an existing backbone network, which however does not necessarily fit their memory budget. Our proposed scaling heuristic is inspired by the works of [32, 48], whose authors raised the following questions:

1. Is it better for the deep architecture to be deeper or wider?

2. Is it better to scale only by width (resp. depth) or by both at once?

We ran preliminary experiments, described in the supplementary material, with neural networks smaller than those presented in [32, 33], and obtain similar results. For a given memory budget, scaling with the unique objective of maximizing the total number of parameters is not optimal in terms of accuracy. We observed that when we reduce unidimensionally the network, the accuracy is better preserved when decreasing the depth rather than the width. Thus, our scaling heuristic consists in reducing the network depth to maximize the network width. Let L be the architecture of a convolutional backbone network b where its memory footprint m_L corresponds here to its number of parameters. The scaling heuristic method is:

$$\psi : (L, m_L) \rightarrow (L', w, d) \tag{2}$$

where $d \in]0, 1]$ (resp. $w \in \mathbb{R}^{*+}$) is a depth (resp. width) coefficient applied to L that allows to obtain the backbone architecture L' respecting the memory constraint. The backbone network L' with $m_{L'}$ parameters is obtained by uniformly multiplying the number of layers of L by d, and the number of convolution filters by w.

The depth coefficient d is chosen first in order to make the network as shallow as possible while preserving the structure of the original network. The width coefficient w is then chosen to maximize the number of convolution filters while respecting the memory budget constraint, i.e., $m_{L'} \leq m$.
In our experiments, we use this heuristic to fit existing architectures to various memory budgets, regardless of whether the initial architecture is already within budget or over budget. We performed numerous experiments with respect to a large set of reference configurations and using the heuristic to scale existing convolutional architectures (Section 4.1).

3.3. Recommending algorithms-backbone combinations

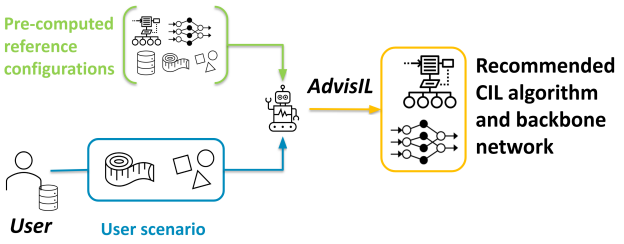

Figure 2: AdvisIL takes as inputs a set of pre-computed reference configurations and a user's scenario and returns a suited CIL algorithm - backbone network combination.

The working process of AdvisIL is illustrated in Figure 2 and explained below. Let $\mathcal{A}$ be a set of EFCIL algorithms, $\mathcal{B}$ a set of backbone networks, $\mathcal{S}$ a set that contains CIL scenarios, where each one is represented as a quadruplet (m, k, α, β) (Definition 1), and $\mathcal{J} = \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_l\}$ a set that contains l complete datasets. We generate a set of reference configurations (Definition 2) as follows:

$$C = \{(a, b, \mathcal{D}_i, m, k, \alpha, \beta) \,|\, a \in \mathcal{A},\ b \in \mathcal{B},\ \mathcal{D}_i \in \mathcal{J},\ (m, k, \alpha, \beta) \in S\}.$$

The user provides a scenario $u = (m^*, k^*, \alpha^*, \beta^*)$. We aim at recommending to the user a pair $(a, b) \in \mathcal{A} \times \mathcal{B}$ composed by an algorithm a and a backbone network b, using our set of reference configurations. The process consists of

two steps. First, a reference scenario which is closest to the user-defined one is selected. Second, the best algorithm-backbone pair which has the best performance on the selected reference scenario is recommended.

Step 1. To compare the user's scenario u to the scenarios in S, which were used for generating the set C, we normalize on $[0, 1]$ any value of memory budget, number of steps, number of initial classes and incremental step size. The procedure is the same for all the parameters. We give it in the case of a memory budget m. We obtain its normalized value by the application $m \rightarrow \frac{m_{\max}-m}{m_{\max}-m_{\min}}$, where $m_{\max} = \max(\max_{1\leq i\leq n} m_i, m^*)$ and $m_{\min} = \min(\min_{1\leq i\leq n} m_i, m^*)$. The normalized value of m is denoted by $\widetilde{m}$. Similarly, we obtain $\widetilde{k}, \widetilde{\alpha}$ and $\widetilde{\beta}$, which are the normalized values corresponding to a number of step k, a number of initial classes α and an incremental step size β, respectively.
For any scenario $v = (m, k, \alpha, \beta) \in \mathcal{S}$, we measure the distance between it and the user's scenario u by:

$$d(u,v) = \sqrt{(\widetilde{m} - \widetilde{m}^*)^2 + (\widetilde{k} - \widetilde{k}^*)^2 + (\widetilde{\alpha} - \widetilde{\alpha}^*)^2 + (\widetilde{\beta} - \widetilde{\beta}^*)^2} \quad (3)$$

We denote by $\hat{v} = (\hat{m}, \hat{k}, \hat{\alpha}, \hat{\beta})$ the scenario in $\mathcal{J}$ whose distance is the smallest to the user's scenario u. This scenario $\hat{v}$ will be used in the next step to obtain a suited pair of algorithm and backbone network.

Step 2. In this step, we rely on the reference configurations in C whose scenario is $\hat{v}$. For each dataset $\mathcal{D}_i \in \mathcal{J}$, we construct the ranking of all pairs $(a, b) \in \mathcal{A} \times \mathcal{B}$ according to their classification performance on that dataset with the scenario $\hat{v}$. For $i = 1, 2, \ldots, l$, the ranking is defined by a partially-ordered set $Rank_i = (\mathcal{A} \times \mathcal{B}, \preceq)$ with the following order relation. For $a, a' \in \mathcal{A}$ and $b, b' \in \mathcal{B}$, we have:

$$(a, b) \preceq (a', b') \text{ iff } r \geq r', \quad (4)$$

where r and r' are the classification performances of the experiments which use the reference configurations $(a, b, \mathcal{D}_i, \hat{m}, \hat{k}, \hat{\alpha}, \hat{\beta})$ and $(a', b', \mathcal{D}_i, \hat{m}, \hat{k}, \hat{\alpha}, \hat{\beta})$ of C, respectively, and are computed using Equation 1. To the ranking $Rank_i$ of the dataset D_i, we associate a function that gives the position of a pair (a, b) in it. For $i = 1, 2, \ldots, l$, this function is:

$$g_i : \mathcal{A} \times \mathcal{B} \mapsto \{1, 2, \ldots, \text{card}(\mathcal{A} \times \mathcal{B})\}.$$

Finally, for a pair (a, b), we can aggregate all its positions in the rankings associated to the l datasets by:

$$\begin{aligned} Agg : \mathcal{A} \times \mathcal{B} &\mapsto \{l, l+1, \ldots, l \cdot \text{card}(\mathcal{A} \times \mathcal{B})\} \\ : (a, b) &\rightarrow \sum_{i=1}^{l} g_i(a, b). \end{aligned} \quad (5)$$

The recommendation provided to the user is the pair(s) of algorithm and backbone network with the lowest overall position computed by the function Agg.

4. Experiments

4.1. Generating the reference configurations

We describe the CIL algorithms, backbone networks, datasets and scenarios used to generate the set of reference configurations.

EFCIL algorithms. In our experiments, we use a representative panel of the algorithms presented in Section 2: LUCIR [19], SPB [52], SIW [4], DeeSIL [2], DSLDA [15] and FeTrIL. We remind that LUCIR, SPB and SIW update the backbone network at each incremental step, and thus focus on the plasticity of representations. In contrast, DeeSIL, DSLDA and FeTrIL use a representation which is fixed after the initial step, and thus favour stability. We implement all algorithms using PyTorch [37] (see the supplementary material for implementation details).

Backbone networks. In our experiments, we use: ResNet18 [17], MobileNetv2 [43] and ShuffleNetv2 [27]. ResNet18 is widely used in the CIL literature. MobileNetv2 and ShuffleNetv2 are designed for high accuracy while considering computational efficiency for embedded applications. These backbone networks are scaled to fit various memory budgets using Equation 2.

Datasets for reference configurations. We consider five datasets which are sampled from ImageNet [11] and denoted by INFood, INFauna, INFlora, INRand_0 and INRand_1. The first three datasets are thematic, and were obtained by sampling leaf ImageNet classes which belong to the "food", "fauna", and "flora" sub-hierarchies, respectively. The two other datasets were obtained by randomly sampling classes from ImageNet. Each dataset contains 100 classes, with 340 images per class for training, and 60 images for testing. Each sampled class is only used in one dataset. Since we have performed many experiments to test the method, the datasets are designed in such a way that the total execution time remains tractable.

CIL scenarios. The memory budget m is defined as the number of parameters of the final model (that contains 100 classes here) and is taken in $\{1.5\text{M}, 3\text{M}, 6\text{M}\}$. The chosen budgets reflect the computational constraints of embedded devices, for which CIL is particularly useful [16]. The other parameters of the scenarios are summarized in the first row of Table 1. Following [42], half of the scenarios have an even split of classes among all states. The other half starts with a higher number of classes in the initial step and evenly splits the remaining classes among subsequent steps, as proposed in [19]. In this way, we cover the two types of scenarios most frequently used in literature.

These settings allow us to form a set 1620 reference configurations, where each one is a combination of an algorithm, a backbone network, a dataset and a scenario presented here. For each reference configuration, we build the corresponding neural network and train it. Finally, we measure the classification performance of the resulting model.

Settings (k, α, β)	**Even splits** $(\alpha = \beta)$			**Larger initial step** $(\alpha > \beta)$		
Reference	(4, 25, 25)	(10, 10, 10)	(20, 5, 5)	(5, 40, 15)	(7, 40, 10)	(11, 40, 6)
Test	(5, 20, 20)	(25, 4, 4)	(50, 2, 2)	(6, 50, 10)	(11, 50, 5)	(13, 40, 5)

Table 1: Settings for evaluating AdvisIL (k: number of steps, α: number of initial classes, β: incremental update size).

4.2. Evaluation settings

To evaluate our recommendation method, we apply AdvisIL to four test datasets and eighteen scenarios.

Test datasets We ran experiments on four test datasets denoted by INRand$_2$, FOOD100, INAT100 and LAND100. INRand$_2$ is obtained by randomly sampling 100-classes of ImageNet [11]. The three other test datasets FOOD100, INAT100 and LAND100 contain 100 classes sampled from FOOD101 [6], iNaturalist [49] and Google Landmarks v1 [35], respectively. These three datasets are thematic and fine-grained since they were designed for the classification of images of food, natural concepts and tourist points of interest, respectively.

Test scenarios. The memory budgets are the same three as those used for generating reference configurations. We consider six new distributions of classes across steps, whose settings are presented in the "Test" row of Table 1.

The same algorithms and backbone networks as in Subsection 4.1 are used. We run experiments for each combination of algorithm, backbone network, test dataset and test scenario. This corresponds to 1296 test configurations.

4.3. Evaluation protocol

As in the case of reference configurations, in our evaluation, model performance is measured in terms of average incremental accuracy (Equation 1). We assess the recommendations provided by AdvisIL as follows. Given a test dataset and a test scenario, we compare the average incremental accuracy of the models trained according to:

i) AdvisIL's recommended pair (a,b). The model built with this pair is called the *recommended model*.

ii) *Oracle* pair: an upper-bound for AdvisIL which selects by brute force the best-performing pair of algorithm and backbone network for each test configuration. It is not usable in practice since it assumes the access to whole stream of data from the beginning. In addition, the extensive computations it requires are costly in terms of time and computational power.

iii) *Baseline pairs*, which are three fixed combinations: b_1: (FeTrIL, ResNet), b_2: (DSLDA, ShuffleNet) and b_3: (SPB, MobileNet) whose corresponding models are called *baseline models*. These pairs were selected according to their aggregated rank on reference datasets (Equation 5), i.e. that they have the highest rank according to their accuracy on all reference experiments. In practice, we present in Table 2 the performance gap between these baseline models and the recommended model. These gaps are denoted Δ_{b_1}, Δ_{b_2}, and Δ_{b_3}.

4.4. Main results

Settings		Incr. acc.	Incr. acc. difference			
		AdvisIL	$\Delta_{\mathcal{O}}$	Δ_{b1}	Δ_{b2}	Δ_{b3}
CIL setting (k, α, β)	(50, 2, 2)	24.42	-1.49	1.01	0.81	9.72
	(25, 4, 4)	35.07	-0.63	0.30	4.35	13.44
	(5, 20, 20)	55.74	-0.65	0.97	3.16	7.46
	(13, 40, 5)	62.56	-0.73	2.33	0.06	12.17
	(11, 50, 5)	64.40	-1.26	2.02	0.22	9.98
	(6, 50, 10)	64.12	-1.56	1.12	0.55	6.64
Budget m	1.5M	45.94	-1.46	0.37	2.76	6.96
	3.0M	51.85	-0.79	2.32	0.52	10.27
	6.0M	54.86	-0.92	1.18	1.30	12.47
Test dataset	INRand$_2$	52.02	-0.73	1.20	1.71	12.51
	INAT100	50.18	-1.22	1.57	1.66	10.03
	FOOD100	28.03	-1.39	1.99	0.34	5.22
	LAND100	74.52	-0.89	0.40	2.89	11.85
Avg		50.88	-1.04	1.29	1.52	9.90

Table 2: Classification performance of models built with AdvisIL's recommendations on four test datasets and six test scenarios. Results are grouped either by user-defined (k, α, β), a memory budget m, or by test dataset. The last row corresponds to the average on all test configurations. The difference between the classification performance of the oracle pair and AdvisIL's recommendation is showed (Δ_{Oracle}). Similarly, the performance gaps between the three baseline recommendations ($\Delta_{b1}, \Delta_{b2}, \Delta_{b3}$) and AdvisIL's recommendation are showed.

AdvisIL performances on test scenarios. Our results are presented in Table 2 and Table 3. In Table 2, we compare the performance of models built according to AdvisIL's recommendation to the performance of models built according to the oracle pair and to the three baseline pairs. In Table 3 the performance of the models built according to AdvisIL and according to the baseline recommendations are compared for all test scenarios.

On average, across all four test datasets and eighteen test scenarios, the recommended model outperforms the best fixed model by 1.29% (Table 2). The accuracy of the recommended models is below the oracle, with an average gap of 1.04%. The gap between the average classification performance of the recommended model and of that of the oracle is stable across scenarios, regardless of the number of steps and class distribution among the steps, and regardless of the memory budget. It is also stable across test datasets. Therefore, the recommendations are relevant whatever the scenario and dataset. Results per scenario are presented in Table 3, where AdvisIL recommends the best available (a, b) combination for 15 out of the 18 test scenarios. The perfor-

Combination (a, b)	Mem. budget m	CIL setting (k, α, β)						avg
		(50, 2, 2)	(25, 4, 4)	(5, 20, 20)	(13, 40, 5)	(11, 50, 5)	(6, 50, 10)	
(SPB, MobileNet)	1.5M	12.46	19.87	44.60	48.80	52.46	55.71	38.98
(DSLDA, ShuffleNet)		**12.77**	18.12	45.43	**59.72**	**61.84**	61.24	43.19
(FeTrIL, ResNet)		10.62	27.75	**53.00**	58.20	61.60	**62.30**	45.58
AdvisIL's pair (a, b)		11.34 (DS, Res)	**28.52** (DS, Res)	**53.00** (FT, Res)	**59.72** (DS, Shu)	**61.84** (DS, Shu)	61.24 (DS, Shu)	**45.94**
(SPB, MobileNet)	3.0M	13.78	22.09	48.57	51.47	55.10	58.45	41.58
(DSLDA, ShuffleNet)		22.81	**35.18**	55.93	**63.71**	**65.47**	64.88	51.33
(FeTrIL, ResNet)		22.08	34.58	55.42	60.70	61.85	62.50	49.52
AdvisIL's pair (a, b)		**24.37** (DS, Res)	34.69 (DS, Res)	**57.05** (DS,Shu)	**63.71** (DS, Shu)	**65.47** (DS, Shu)	**65.81** (DS, Mob)	**51.85**
(SPB, MobileNet)	6.0M	14.86	22.94	51.7	50.89	55.68	58.25	42.39
(DSLDA, ShuffleNet)		32.24	38.86	56.38	64.07	65.21	64.59	53.56
(FeTrIL, ResNet)		**34.55**	**42.00**	55.88	61.80	63.68	64.18	53.68
AdvisIL's pair (a, b)		**34.55** (FT, Res)	**42.00** (FT, Res)	**57.18** (FT, Mob)	**64.25** (DS, Mob)	**65.89** (DS, Mob)	**65.29** (DS, Mob)	**54.86**

Table 3: Classification performance for the three baseline models and for the recommended model. Performance is averaged over the four test datasets. For each scenario (m, k, α, β), the best result is in bold and the combination of algorithm (either DSLDA (DS) or FeTrIL (FT)) and backbone (MobileNet (Mob) , ResNet (Res) or Shufflenet (Shu)) recommended by AdvisIL is provided.

mance improvement obtained by AdvisIL is explained by the fact that its recommendations follow the trends observed in the experiments corresponding to the reference configurations, whereas no fixed combination of algorithm and backbone network outperforms all others in all scenarios. This was pointed out in Figure 1 where it can be seen that the same combinations of CIL algorithm and backbone network is not ranked the same from one scenario to another.

In practice, the recommended model is either built with FeTrIL or DSLDA (Table 3), two algorithms that do not use distillation. These algorithms are recommended because they are often the best-performing algorithms in the experiments corresponding to our set of reference configurations (see appendix for more details). Our results, obtained in the case of several neural architectures and for small memory budgets, confirm previous findings regarding the competitiveness of transfer-learning-based algorithms in EFCIL, which are reported in the comparative studies of [5, 28]. Regarding the recommended backbone networks, Table 3 shows that it is relevant to use a different backbone network depending on the memory budget. ResNet and ShuffleNet appear as the best choices for 1.5M parameters, ShuffleNet and MobileNet for 3.0M parameters, and MobileNet for 6.0M parameters. This highlights the relevance of considering different backbone networks in AdvisIL.

4.5. Ablation study

AdvisIL recommends an algorithm-backbone combination based on pre-computed experiments on several reference datasets. In what follows, we study the individual contribution of the different components of the method: choice of the backbone network, choice of the algorithm and choice of the number of available reference datasets.

Incr. acc.	(i) Fixed backbone net			(ii) Fixed algorithm		
AdvisIL	Δ_{Mob}	Δ_{Shu}	Δ_{Res}	Δ_{FT}	Δ_{DS}	Δ_{SPB}
50.88	-3.93	-1.19	-0.97	-0.45	-0.42	-9.74

Table 4: Classification performance of AdvisIL and of its variants which use (i) a single possible backbone network taken among ResNet18 (Res), MobileNetv2 (Mob), and ShuffleNetv2 (Shu), and (ii) a single possible algorithm, either FeTrIL (FT), DSDLA (DS) or SPB. Results are averaged over all test scenarios and test datasets.

Using a single backbone network. We present results with a fixed backbone network and different algorithms in the left part of Table 4. Each column corresponds to the average classification performance of models trained according to AdvisIL recommendations when the set of reference configurations is reduced to consider a single type of backbone network. The results indicate that AdvisIL's recommendations are better when we use multiple backbone networks rather than just one.

Using a single CIL algorithm. Similarly, we present in the right part of Table 4 results obtained when only a single algorithm is available. Each column corresponds to the average classification performance obtained with AdvisIL recommendations when the set of reference configurations is reduced to consider a single algorithm and multiple backbone networks. When we reduce the set of configurations to DSLDA or FeTrIL, the performance drop is rather low, because these two algorithms have similar performance, and tend to perform much better than the others tested, so that they are often recommended by AdvisIL. On the contrary, when using only SPB, the third best-performing algorithm, performance drops significantly.

Using fewer reference datasets. AdvisIL's recommen-

dation are computed by ranking the classification performance of candidate algorithm-backbone pairs on the reference datasets. Here, we measure how using less reference datasets impacts the relevance of the recommendations. Results obtained by using only one ($\Delta_{\#1}$), two ($\Delta_{\#2}$), three ($\Delta_{\#3}$) and four ($\Delta_{\#4}$) out of five reference datasets are presented in Table 5. They are averaged over all possible combinations of reference datasets. Using less reference datasets is damageable to recommendation relevance, with up to 1.88% loss in the averaged classification performance when using only one reference dataset. Interestingly, when using only two reference datasets, AdvisIL already surpasses the baselines presented in Table 2. The impact of the ablation fades when the number of reference datasets grows, and becomes negligible when four of them are used.

Incr. acc.	Number of reference datasets			
AdvisIL	$\Delta_{\#1}$	$\Delta_{\#2}$	$\Delta_{\#3}$	$\Delta_{\#4}$
50.88	-1.88	-0.71	-0.39	-0.09

Table 5: Classification performance of AdvisIL and of its variants which use a smaller number of reference datasets. Results are averaged over all possible combinations of reference datasets, and over all tested scenarios.

Overall, the results of Table 4 show that both the choice of an algorithm and the choice of a backbone network contribute to the relevance of AdvisIL's recommendations. The results from Table 5 show that AdvisIL needs few reference datasets to provide relevant recommendations. More ablation results are provided in the supplementary material.

5. Discussion and Conclusion

In this article, we introduced a recommendation method named AdvisIL, which facilitates the choice of a suited pair of CIL algorithm and backbone network for a user-defined incremental learning scenario. AdvisIL requires little information from the user and provides a recommendation by leveraging trends observed on pre-computed experiments from a set of reference configurations. Our evaluation indicates that AdvisIL is effective, as it often provides a relevant recommendation. Recommendations are relevant because they are made by selecting reference experiments whose scenarios are close to the user's scenario. On average, the resulting models outperform by at least 1.29% the models that do not tailor the recommendation to the user's scenario. Beyond performance gain, AdvisIL avoids the users to perform cumbersome preliminary computations to design their incremental learning application.

In what follows, we discuss how AdvisIL may be improved and give some perspectives. The set of reference configurations could be enriched with recent algorithms [50, 55, 56, 55], other backbone networks, such as SqueezeNet [22], MobileVit [30] or EfficientNet [48], and more scenarios could further increase the relevance of AdvisIL in real-life applications [16]. We designed AdvisIL in order to facilitate the integration of new components and contributions. The full code, the raw data used, and the set of reference configurations will be published to facilitate take-up in the community. A binding to Avalanche [26] will also be developed to easily integrate contributions of other researchers to its improvement.

In this article, we have done many experiments corresponding to a set of reference configurations in order to exhaustively test all possible combinations of CIL algorithm and backbone network. This entailed a consequent computation effort. We may generate the reference configurations more efficiently, taking inspiration from hyperparameter search approaches, e.g. [53]. Regarding the recommendation process, we could take into account the similarity between the user's dataset and the datasets used to build the set of reference configurations [1]. From a practical point of view, we note that this similarity can only be computed using the first batch of the user's dataset as it is the only one we may have access to.

We focused on memory constraints and expressed them as the number of parameters of the neural networks. It may be possible to take other constraints into account, such as the training complexity (the maximum number of computations for each incremental step) and inference time. Regarding the training with CIL algorithms, we note that fine-tuning-based methods [4, 19, 52] have significantly higher computational requirements compared to some transfer-based methods [2, 15]. Fine-tuning-based methods retrain the full model in each incremental state, while the transfer-based methods retrain only the classification layer. Among the fine-tuning-based methods, some of them [19, 52] rely on knowledge distillation, which requires keeping in memory two models, namely the current version of the model which is being trained, and the previous version of the model which is used for constraining the training at this step. Regarding the inference time constraint, given a model size, the inference time varies depending on the structure of the deep architecture [7]. If we extend our recommendation method by taking into account training and inference times, the recommended pair of CIL algorithm and backbone network may be selected according to the modified variant of the NetScore [51] introduced in [16]. This score mixes accuracy, total number of parameters and the number of seconds required to run the experiment.

Acknowledgements. This work was supported by the European Commission under European Horizon 2020 Programme, grant number 951911 - AI4Media. It was made possible by the use of the FactoryIA supercomputer, financially supported by the Ile-de-France Regional Council.

References

[1] Alessandro Achille, Michael Lam, Rahul Tewari, Avinash Ravichandran, Subhransu Maji, Charless C Fowlkes, Stefano Soatto, and Pietro Perona. Task2vec: Task embedding for meta-learning. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6430–6439, 2019.

[2] Eden Belouadah and Adrian Popescu. Deesil: Deep-shallow incremental learning. *TaskCV Workshop @ ECCV 2018.*, 2018.

[3] Eden Belouadah and Adrian Popescu. Deesil: Deep-shallow incremental learning. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, pages 0–0, 2018.

[4] Eden Belouadah, Adrian Popescu, and Ioannis Kanellos. Initial classifier weights replay for memoryless class incremental learning. In *British Machine Vision Conference (BMVC)*, 2020.

[5] Eden Belouadah, Adrian Popescu, and Ioannis Kanellos. A comprehensive study of class incremental learning algorithms for visual tasks. *Neural Networks*, 135:38–54, 2021.

[6] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. Food-101 – mining discriminative components with random forests. In *European Conference on Computer Vision*, 2014.

[7] Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*, 2016.

[8] Francisco M. Castro, Manuel J. Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari. End-to-end incremental learning. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XII*, pages 241–257, 2018.

[9] Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Aleš Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE transactions on pattern analysis and machine intelligence*, 44(7):3366–3385, 2021.

[10] Giorgos Demosthenous and Vassilis Vassiliades. Continual learning on the edge with tensorflow lite. *arXiv preprint arXiv:2105.01946*, 2021.

[11] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, pages 248–255, 2009.

[12] Prithviraj Dhar, Rajat Vikram Singh, Kuan-Chuan Peng, Ziyan Wu, and Rama Chellappa. Learning without memorizing. *CoRR*, abs/1811.08051, 2018.

[13] Arthur Douillard, Matthieu Cord, Charles Ollion, Thomas Robert, and Eduardo Valle. Podnet: Pooled outputs distillation for small-tasks incremental learning. In *Computer vision-ECCV 2020-16th European conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part XX*, volume 12365, pages 86–102. Springer, 2020.

[14] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019.

[15] Tyler L Hayes and Christopher Kanan. Lifelong machine learning with deep streaming linear discriminant analysis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 220–221, 2020.

[16] Tyler L. Hayes and Christopher Kanan. Online continual learning for embedded devices. *ArXiv*, 2203.10681v2, 2022.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition*, CVPR, 2016.

[18] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015.

[19] Saihui Hou, Xinyu Pan, Chen Change Loy, Zilei Wang, and Dahua Lin. Learning a unified classifier incrementally via rebalancing. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 831–839, 2019.

[20] Shenyang Huang, Vincent Francois-Lavet, and Guillaume Rabusseau. Understanding capacity saturation in incremental learning. *Proceedings of the Canadian Conference on Artificial Intelligence*, 6 2021.

[21] Shenyang Huang, Vincent François-Lavet, and Guillaume Rabusseau. Neural architecture search for class-incremental learning. *ArXiv*, abs/1909.06686, 2019.

[22] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[23] Ronald Kemker, Marc McClure, Angelina Abitino, Tyler Hayes, and Christopher Kanan. Measuring catastrophic forgetting in neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[24] Young D Kwon, Jagmohan Chauhan, Abhishek Kumar, Pan Hui HKUST, and Cecilia Mascolo. Exploring system performance of continual learning for mobile and embedded sensing applications. In *2021 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 319–332. IEEE, 2021.

[25] Zhizhong Li and Derek Hoiem. Learning without forgetting. In *European Conference on Computer Vision*, ECCV, 2016.

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

[27] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision (ECCV)*, pages 116–131, 2018.

[28] Marc Masana, Xialei Liu, Bartlomiej Twardowski, Mikel Menta, Andrew D. Bagdanov, and Joost van de Weijer. Class-incremental learning: survey and performance evaluation on image classification, 2021.

[29] Michael Mccloskey and Neil J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. *The Psychology of Learning and Motivation*, 24:104–169, 1989.

[30] Sachin Mehta and Mohammad Rastegari. Mobilevit: light-weight, general-purpose, and mobile-friendly vision transformer. *arXiv preprint arXiv:2110.02178*, 2021.

[31] Thomas Mensink, Jakob Verbeek, Florent Perronnin, and Gabriela Csurka. Distance-based image classification: Generalizing to new classes at near-zero cost. *IEEE transactions on pattern analysis and machine intelligence*, 35(11):2624–2637, 2013.

[32] Seyed-Iman Mirzadeh, Arslan Chaudhry, Huiyi Hu, Razvan Pascanu, Dilan Görür, and Mehrdad Farajtabar. Wide neural networks forget less catastrophically. *CoRR*, abs/2110.11526, 2021.

[33] Seyed Iman Mirzadeh, Arslan Chaudhry, Dong Yin, Timothy Nguyen, Razvan Pascanu, Dilan Gorur, and Mehrdad Farajtabar. Architecture matters in continual learning. *arXiv preprint arXiv:2202.00275*, 2022.

[34] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[35] Hyeonwoo Noh, Andre Araujo, Jack Sim, Tobias Weyand, and Bohyung Han. Large-scale image retrieval with attentive deep local features. In *ICCV*, pages 3476–3485. IEEE Computer Society, 2017.

[36] Shaoning Pang, Seiichi Ozawa, and Nikola Kasabov. Incremental linear discriminant analysis for classification of data streams. *IEEE*

transactions on Systems, Man, and Cybernetics, part B (Cybernetics), 35(5):905–914, 2005.

[37] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[38] Lorenzo Pellegrini, Vincenzo Lomonaco, Gabriele Graffieti, and Davide Maltoni. Continual learning at the edge: Real-time training on smartphone devices. *ArXiv*, abs/2105.13127, 2021.

[39] Grégoire Petit, Adrian Popescu, Hugo Schindler, David Picard, and Bertrand Delezoide. Fetril: Feature translation for exemplar-free class-incremental learning. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2023.

[40] Ameya Prabhu, Philip HS Torr, and Puneet K Dokania. Gdumb: A simple approach that questions our progress in continual learning. In *European Conference on Computer Vision*, pages 524–540. Springer, 2020.

[41] Hang Qi, Matthew Brown, and David G Lowe. Low-shot learning with imprinted weights. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5822–5830, 2018.

[42] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H. Lampert. icarl: Incremental classifier and representation learning. In *Conference on Computer Vision and Pattern Recognition*, CVPR, 2017.

[43] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[44] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[45] James Smith, Yen-Chang Hsu, Jonathan Balloch, Yilin Shen, Hongxia Jin, and Zsolt Kira. Always be dreaming: A new approach for data-free class-incremental learning. *arXiv preprint arXiv:2106.09701*, 2021.

[46] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A survey on deep transfer learning. In *International conference on artificial neural networks*, pages 270–279. Springer, 2018.

[47] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. MnasNet: Platform-Aware Neural Architecture Search for Mobile. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2815–2823, Long Beach, CA, USA, June 2019. IEEE.

[48] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114. PMLR, 2019.

[49] Grant Van Horn, Oisin Mac Aodha, Yang Song, Yin Cui, Chen Sun, Alex Shepard, Hartwig Adam, Pietro Perona, and Serge Belongie. The inaturalist species classification and detection dataset. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8769–8778, 2018.

[50] Vinay Kumar Verma, Kevin J. Liang, Nikhil Mehta, Piyush Rai, and Lawrence Carin. Efficient feature transformations for discriminative and generative continual learning. *CoRR*, abs/2103.13558, 2021.

[51] Alexander Wong. Netscore: towards universal metrics for large-scale performance analysis of deep neural networks for practical on-device edge usage. In *International Conference on Image Analysis and Recognition*, pages 15–26. Springer, 2019.

[52] Guile Wu, Shaogang Gong, and Pan Li. Striking a balance between stability and plasticity for class-incremental learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1124–1133, 2021.

[53] Jia Wu, Xiu-Yun Chen, Hao Zhang, Li-Dong Xiong, Hang Lei, and Si-Hao Deng. Hyperparameter optimization for machine learning models based on bayesian optimization. *Journal of Electronic Science and Technology*, 17(1):26–40, 2019.

[54] Yue Wu, Yinpeng Chen, Lijuan Wang, Yuancheng Ye, Zicheng Liu, Yandong Guo, and Yun Fu. Large scale incremental learning. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 374–382, 2019.

[55] Fei Zhu, Zhen Cheng, Xu-yao Zhang, and Cheng-lin Liu. Class-incremental learning via dual augmentation. *Advances in Neural Information Processing Systems*, 34, 2021.

[56] Fei Zhu, Xu-Yao Zhang, Chuang Wang, Fei Yin, and Cheng-Lin Liu. Prototype augmentation and self-supervision for incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5871–5880, 2021.

[57] Kai Zhu, Wei Zhai, Yang Cao, Jiebo Luo, and Zheng-Jun Zha. Self-sustaining representation expansion for non-exemplar class-incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9296–9305, 2022.