Fast Incremental Learning by Transfer Learning and Hierarchical Sequencing

Laura Llopis-Ibor[a,*], Cesar Beltran-Royo[a], Alfredo Cuesta-Infante[a], Juan J. Pantrigo[a]

[a] *Computer Science and Statistics Department, King Juan Carlos University, Calle Tulipán s/n, Móstoles 28933, Madrid, Spain*

Abstract

In this paper we address the Class Incremental Learning (CIL) problem, characterized by sequences of data batches in which examples of different classes occur at different times. From a theoretical point of view, we propose a new approach that we call *hierarchical sequencing* and prove that any CIL task can be sequenced into simple incremental classification tasks by means of the hierarchical sequencing. From a practical point of view, we propose the HILAND method for image classification, which combines the hierarchical sequencing with transfer learning. In our experiments, the HILAND method has obtained state-of-the-art results for the CIL problem, but with far less training effort through transfer learning.

Keywords: class incremental learning, transfer learning, image classification

1. Introduction

Image classification is the most basic supervised task in computer vision, and is fundamental to build smarter models that are capable of performing higher-level tasks, such as detection, segmentation or generation. In detection networks such as YOLO (Redmon et al., 2016) or SSD (Liu et al., 2016), the

[*] Corresponding author

Email addresses: `laura.llopis@urjc.es` (Laura Llopis-Ibor), `cesar.beltran@urjc.es` (Cesar Beltran-Royo), `alfredo.cuesta@urjc.es` (Alfredo Cuesta-Infante), `juanjose.pantrigo@urjc.es` (Juan J. Pantrigo)

image is divided in small regions and each region predicts the bounding box of the entities in the picture as well as its class. Segmentation is another typical computer vision task aimed to classify every single pixel in the image. Generative Adversarial Networks architecture (Goodfellow et al., 2014) include a discriminator that is trained to distinguish between true and fake images. Image classification is also the bread and butter of many automated processes, such as facial recognition at security checkpoints, image organization in photographic applications, or item separation on conveyor belts, just to mention a few.

Convolutional Neural Networks (CNN) have boosted the performance of image classification up to human levels in several academic challenges. However, real applications differ from these challenges in several points. First, in classification challenges both the number of samples and the number of different classes is fixed. Second, the whole data set is available from the beginning. Third, the data set consists of a number of examples large enough to get the models to generalize. And fourth, the classifier is meant to be executed once, not to be deployed in a broader system.

On the other hand, if the data set is large, labeling is time consuming and expensive, otherwise the performance is far from the expected. Moreover, training CNNs usually demands time and computing power, and for inference the classifier has to be integrated into the final system. Finally, it frequently happens that data arrives one by one or in small packages, and must be used and discarded, as it happens in streams that cannot be stored due to its velocity and volume. Besides, real applications may want to increase the number of classes due to new requirements, which is usually a task that needs batches of new data rather than streams.

Each one of the aforementioned issues has been tackled with a variety of techniques. The lack of labeled data can be overcome by transfer learning. In fact, currently the most popular neural network libraries, such as PyTorch (Paszke et al., 2019) and Tensorflow (Abadi et al., 2015), include ready-to-use CNNs that have been pretrained on large image data sets. Transfer learning also allows faster training for new specific classes if the CNN is frozen, for it can be loaded as a compiled module so that the training process is limited to the weights of a new top. Additionally, this compiled module also facilitates the integration into broader systems. Regarding the incoming data flow, two different approaches have been followed in parallel, according to how this data is fed into the system during learning: streams and batches of data. If it arrives in streams, it is dealt with both supervised and unsupervised methods for massive online analysis (Bifet et al., 2018). Our work focuses on data arriving in batches in order to learn whole new classes sequentially in a supervised fashion.

Arguably, being able to increase the number of classes to discriminate after training the classifiers is the goal that has attracted more interest and research effort. It is well known that the training process depends on the data set, so changes in it bring further changes into the weights of the network. Since adding new classes means adding new labeled samples to learn from, the system must be trained with the data set extended with these new samples in order to keep its previous capabilities. Otherwise *catastrophic forgetting* will happen.

In recent years, an important stream of research has been devoted to overcome catastrophic forgetting, under different denominations: continual learning (Chen & Liu, 2018), lifelong learning (Parisi et al., 2019), sequential learning (McCloskey & Cohen, 1989) or incremental learning (Rebuffi et al., 2017). The objective is to learn from data, preserving and extending acquired knowledge in the different learning steps.

Specifically, the term *Class Incremental Learning (CIL)* is introduced in (Rebuffi et al., 2017) together with the required properties of an algorithm to qualify as class-incremental:

1) It should provide a multiclass classifier trainable from a sequence of data batches in which different classes occur at different batches.

2) At any time, the provided multiclass classifier should be competitive for the classes observed so far.

3) Its computational requirements and memory footprint should remain bounded,

or at least grow very slowly, with respect to the number of classes seen so far.

In other words, CIL algorithms face two main problems:

1) *Data availability.* The samples from all the classes that can or will be ultimately learned are not available simultaneously at the beginning, so the algorithms ignore all about the missing classes. This challenge is addressed through two strategies: By parameter fixing (parameter isolation methods) or by parameter updating (replay methods and regularization methods) (Delange et al., 2021).

2) *Storage constraints.* To store the large amount of data used by CIL algorithms is not possible in most of cases. To cope with this difficulty, two strategies are used: By replaying previous data information (replay methods) or by only using the current data information (regularization-based methods and parameter isolation methods) (Delange et al., 2021).

CIL methods can be classified into three groups: Replay methods, regularization-based methods and parameter isolation methods. Replay methods need to store samples in raw format of already learned classes or generate pseudo-samples with a generative model. To alleviate forgetting, these samples are replayed while learning new classes. Some methods of this group are Incremental Classifier and Representation Learning (iCaRL) (Rebuffi et al., 2017), Generative-Rehearsal (GR) (Lee et al., 2021) and Gradient Episodic Memory (GEM) (Lopez-Paz & Ranzato, 2017), among others. According to the comparison reported in the survey (Delange et al., 2021), the leading method in this class is iCaRL, which stores a subset of exemplars per class, selected to best approximate class means in the learned feature space (this strategy is known as *hearding*). Therefore, iCaRL uses a nearest-mean-of-exemplars classification strategy. Another leading replay approach is the Bias Correction (BiC) method (Wu et al., 2019). This method employs a bias correction strategy to balance the importance between the already learned classes and the new ones, thus, addressing the class-imbalance problem.

In regularization-based methods an extra regularization term is used in the loss function in order to consolidate already learned classes while learning new classes. The importance of all neural network parameters is estimated and, during learning later classes, changes to important parameters are penalized. This family of methods avoids storing samples of already learned classes. Some methods of this class are: Memory Aware Synapses (MAS) (Aljundi et al., 2018), Learning without Forgetting (LwF) (Li & Hoiem, 2018) and Elastic Weight Consolidation (EWC) (Kirkpatrick et al., 2017).

Parameter isolation methods do not require to store samples of already learned classes. Instead, these methods isolate and fix parameters for already learned classes. In this way these methods can guarantee maximal stability of the learned task (zero forgetting). Some methods in this class are: Packing multiple tasks to a single Network (PackNet) (Mallya & Lazebnik, 2018) and overcoming catastrophic forgetting with Hard Attention to the Task (HAT) (Serra et al., 2018).

Our contribution is twofold:

1) *Theoretical point of view:* We prove that any CIL task can be sequenced into more simple incremental classification tasks by means of a new approach that we call hierarchical task sequencing (*hierarchical sequencing*, for short). The hierarchical sequencing is based on the so-called *perfect* incremental classifiers, which attain 100% accuracy. These classifiers can be trained sequentially and independently, which is well suited for the CIL problem.

2) *Practical point of view:* Based on the hierarchical sequencing, we propose the HILAND method for class incremental learning, where HILAND stands for *Hierarchical Incremental Learning Appending Naive Discriminators.* This new method obtains state-of-the-art results for the CIL problem, but with less training effort by using transfer learning (Chollet, 2017). More specifically, HILAND uses a pretrained convolutional base

combined with a densely connected linear classifier. The training effort is very low for two reasons: (i) Only the linear classifier is trained. (ii) Having a frozen convolutional base allows to extract and store the feature vectors by running the convolutional base *only once* for every input image (the convolutional base is the most expensive part of the whole neural network). Then, these feature vectors are used repeatedly to train the linear classifier. This approach is much faster than training with the raw images, which requires running the convolutional base for every input image *at each* training iteration.

The rest of the paper is organized as follows. Section 2 introduces the concept of perfect decomposition classifier which can be used to *decompose* any multiclass classification problem into simpler tasks. Section 3 introduces the hierarchical sequencing which can be used to sequence any CIL task into simpler tasks. In Section 4 we derive the HILAND method, basically by combining the hierarchical sequencing with transfer learning (based on pretrained CNNs). In Section 5 we compare our incremental learning approach to state of the art CIL algorithms and test the impact that different pretrained CNNs may have in the HILAND performance. A conclusion of our research is given in section 6 together with directions for future research.

2. Class static learning by task decomposition

In this paper we consider the multiclass classification task and distinguish between class static learning and class incremental learning. In the former all the training data is available simultaneously (only one stage); whereas in the later, the training data is available incrementally (one group of classes at each stage) as we will see in Section 3. In this section, we show that any multiclass classification task can be decomposed into a set of decomposition classification tasks in the context of class static learning.

It is well known that task decomposition can be used to solve large-scale multiclass classification tasks by parallel computing (Lu & Ito, 1999). However, in this paper we will use task decomposition not for parallel computing but for designing an effective method for class incremental learning. That is, we will be inspired by the simple results on task decomposition of this section to develop the more elaborated results on task sequencing of Section 3.

Definition 1 (*Multiclass classification task)***.** *Let us consider a set of C classes indexed by $c \in \mathcal{C} = \{1, \ldots, C\}$, a set of non-intersecting class data domains $\{\mathcal{X}_c\}_{c \in \mathcal{C}}$, and a partition of the data domain into classes, i.e., $\mathcal{X} = \mathcal{X}_1 \cup \ldots \cup \mathcal{X}_C$. The* perfect multiclass classification task *$T(\mathcal{X}, \mathcal{C})$ aims to correctly decide the class of any data vector $x \in \mathcal{X} \subset \mathbb{R}^n$.*

Definition 2 (*Perfect multiclass classifier)***.** *A perfect multiclass classifier for task $T(\mathcal{X}, \mathcal{C})$, corresponds to a function $f^*(x) : \mathcal{X} \to \mathcal{C}$ such that:*

$$f^*(x) = c \quad \textit{if} \quad x \in \mathcal{X}_c. \tag{1}$$

That is, a perfect multiclass classifier f^* gives the right answer for the question 'Which is the class of x?' for any $x \in \mathcal{X}$. Therefore a perfect multiclass classifiers attains an accuracy of 100%. This requires non-intersecting class data domains in order to have a univoque answer of the perfect multiclass classifier. That is, to classify any x there will be no ambiguity since it is in a single class. Of course, this is an ideal situation since in practice we have imperfect classifiers and multiclass classification tasks with data domains that may intersect in some ambiguity regions. Nonetheless, these ideal assumptions will help us to develop useful algorithms for realistic incremental learning in Section 4.

Notice that task $T(\mathcal{X}, \mathcal{C})$ can be carried out by different perfect multiclass classifiers and we denote by $\mathcal{F}^*(\mathcal{X}, \mathcal{C})$ the set such classifiers. In this context, it is useful to define the set $\mathcal{Y}_c = f^*(\mathcal{X}_c)$ for all $c \in \mathcal{C}$. Thus, for any $x_0 \in \mathcal{X}_c$ there is $y_0 \in \mathcal{Y}_0$ such that $y_0 = f^*(x_0) = c$. In this paper, we use the notation $[a : b]$ to denote the set of integers from a to b. We also consider a set of K groups of classes indexed by $k \in \mathcal{K} = \{1, \ldots, K\}$. For simplicity of notation and without loss of generality, we assume that all the groups contain the same number of classes, say L. That is, the k-th group corresponds to the classes indexed by $\mathcal{C}_k =$

$[(k-1)L+1 : kL]$ and the corresponding group data domain can be partitioned into non-intersecting data domains as follows: $\mathcal{X}_{\mathcal{C}_k} = \mathcal{X}_{(k-1)L+1} \cup \ldots \cup \mathcal{X}_{kL}$. This implicitly assumes that $C = LK$.

Definition 3 (*Decomposition classification task*)**.** *Let us consider a set of non-intersecting group data domains $\{\mathcal{X}_{\mathcal{C}_k}\}$, for $k \in \mathcal{K}$, and the corresponding partition of the data domain $\mathcal{X} = \mathcal{X}_{\mathcal{C}_1} \cup \ldots \cup \mathcal{X}_{\mathcal{C}_K}$. The objective of the decomposition classification task $T_{\mathcal{C}_k}(\mathcal{X}, \mathcal{C})$ is to correctly decide whether any data vector $x \in \mathcal{X}$ is either in group k or in its complement. In the first case, this task also aims to give the correct class of x.*

Definition 4 (*Perfect decomposition classifier*)**.** *A perfect decomposition classifier for task $T_{\mathcal{C}_k}(\mathcal{X}, \mathcal{C})$, corresponds to a function $f^*_{\mathcal{C}_k}(x) : \mathcal{X} \to \{0\} \cup \mathcal{C}_k$ such that:*

$$f^*_{\mathcal{C}_k}(x) = \begin{cases} c & \text{if} \quad x \in \mathcal{X}_c \subset \mathcal{X}_{\mathcal{C}_k} \\ 0 & \text{if} \quad x \in \mathcal{X} \setminus \mathcal{X}_{\mathcal{C}_k}. \end{cases}$$

That is, a perfect decomposition classifier $f^*_{\mathcal{C}_k}$ gives the right answer for the question 'Is x in a class of group k?' for any $x \in \mathcal{X}$. If the answer is affirmative, it also determines the class of x. We denote by $\mathcal{F}^*_{\mathcal{C}_k}(\mathcal{X}, \mathcal{C})$ the set of all the perfect decomposition classifier that can perform the task $T_{\mathcal{C}_k}(\mathcal{X}, \mathcal{C})$.

In the following proposition, we show that any multiclass classification task can be decomposed into a set of decomposition classification tasks.

Proposition 1 (*Task decomposition*)**.** *Let us consider a set of perfect decomposition classifiers $\{f^*_{\mathcal{C}_k}\}_{k \in \mathcal{K}}$ such that $f^*_{\mathcal{C}_k} \in \mathcal{F}^*_{\mathcal{C}_k}(\mathcal{X}, \mathcal{C})$. If we define*

$$f^*(x) = max_{k \in \mathcal{K}}\{f^*_{\mathcal{C}_k}(x)\} \quad \text{for all } x \in \mathcal{X},$$

then $f^* \in \mathcal{F}^*(\mathcal{X}, \mathcal{C})$. Therefore, the multiclass classification task $T(\mathcal{X}, \mathcal{C})$ can be decomposed into the set of decomposition classification tasks $\{T_{\mathcal{C}_k}(\mathcal{X}, \mathcal{C})\}_{k \in \mathcal{K}}$.

Proof: Let us prove that f^* is a perfect multiclass classifier associated to task $T(\mathcal{X}, \mathcal{C})$, that is, let us see that if $x \in \mathcal{X}_{c_0} \subset \mathcal{X}_{\mathcal{C}_{k_0}}$ then $f^*(x) = c_0$. Given the hypothesis, $f^*_{\mathcal{C}_{k_0}}(x) = c_0$, and $f^*_{\mathcal{C}_k}(x) = 0$ for any $k \neq k_0$. Thus

$$\max_{k\in\mathcal{K}}\{f^*_{\mathcal{C}_k}(x)\} = c_0.$$

■

3. Class incremental learning by hierarchical sequencing

As we already pointed out, in the context of class incremental learning we assume that all the data is available incrementally (one group of classes at each stage). This is in sharp contrast with the previous section, where all the data was available simultaneously. The objective of this section is to show that any multiclass classification task can be sequenced into a set of incremental classification tasks. Specifically, we adapt the task decomposition of Proposition 1 to the hierarchical sequencing of Proposition 2.

3.1. Hierarchical sequencing

As in the previous section, we consider a set of K groups of classes indexed by $k \in \mathcal{K}$. We also assume that all the groups contain the same number of classes, say L. Let us define the incremental data domain up to group k as $\mathcal{X}_{[1:k]} = \mathcal{X}_{\mathcal{C}_1} \cup \ldots \cup \mathcal{X}_{\mathcal{C}_k}$, where $\mathcal{X}_{\mathcal{C}_k} = \mathcal{X}_{(k-1)L+1} \cup \ldots \cup \mathcal{X}_{kL}$, for any $k \in \mathcal{K}$. For simplicity of notation we will write $\mathcal{X}_{[k]}$ instead of $\mathcal{X}_{[1:k]}$. We begin defining the incremental classification task as follows.

Definition 5 (*Incremental classification task*)**.** *Let us consider a set of non-intersecting group data domains $\{\mathcal{X}_{\mathcal{C}_k}\}_{k\in\mathcal{K}}$ and the corresponding partition of the data domain $\mathcal{X} = \mathcal{X}_{\mathcal{C}_1} \cup \ldots \cup \mathcal{X}_{\mathcal{C}_K}$. The objective of the incremental classification task $T_{[k]}(\mathcal{X}, \mathcal{C})$ is to correctly decide whether any data vector $x \in \mathcal{X}$ is either in group k or in the union of groups prior to k, if any. In the first case, this task also aims to give the correct class of x.*

Next, let us define perfect incremental classifier. Roughly speaking, a set of perfect incremental classifiers will correspond to a set of classifiers that are trained with data available incrementally, and that become more trustable as new data become available (see Figure 1).

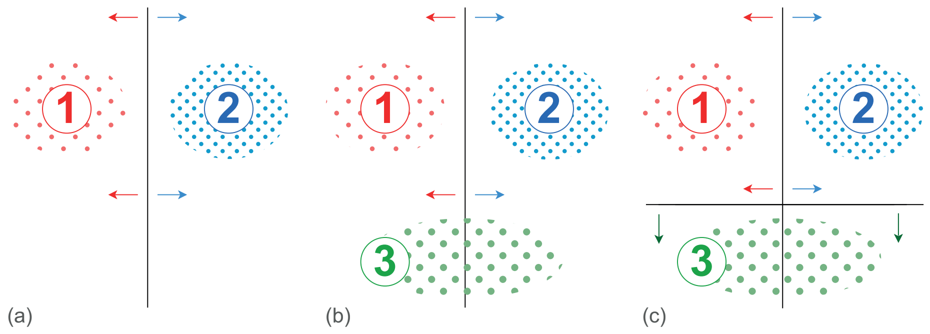

Figure 1: Perfect incremental classifier. In this figure, data is available incrementally with two groups of classes: classes 1 and 2 (group 1), available at the first stage, and class 3 (group 2) available at the second stage. (a) At the first stage, the first *perfect incremental classifier* $f^*_{[1]}$, the vertical line, classifies data from classes 1 and 2 with 100% accuracy. (b) At the second stage, data form class 3 becomes available but the classifier $f^*_{[1]}$ is not trustable for this new class. (c) At the second stage, the second *perfect incremental classifier* $f^*_{[2]}$, the horizontal line, shows 100% accuracy in order to classify data as 'class 3' or 'no class 3'. In summary, $f^*_{[2]}$ is more trustable that $f^*_{[1]}$ since it has been trained with old and new data (see Definition 6). This fact suggests a hierarchy, where recent classifiers should have preference over their predecessors in deciding the class of a data point.

Definition 6 (*Perfect incremental classifier*)**.** *Given any $k \in \mathcal{K}$, a perfect incremental classifier for task $T_{[k]}(\mathcal{X}, \mathcal{C})$, corresponds to a function $f^*_{[k]}(x) : \mathcal{X} \to \overline{\mathcal{C}}_k := \{0\} \cup \mathcal{C}_k$ such that:*

$$f^*_{[k]}(x) = \begin{cases} 0 & \text{if } \ x \in \mathcal{X}_{[k-1]}, \\ & \quad (\text{only for cases with } k > 1) \\ c & \text{if } \ x \in \mathcal{X}_c \subset \mathcal{X}_{\mathcal{C}_k} \\ Y_{k,k+1} & \text{if } \ x \in \mathcal{X}_{\mathcal{C}_{k+1}} \\ \vdots & \\ Y_{k,K} & \text{if } \ x \in \mathcal{X}_{\mathcal{C}_K}, \\ & \quad (\text{only for cases with } k < K), \end{cases}$$

where $Y_{k,i}$ is a categorical random variable with categories $\overline{\mathcal{C}}_k$ and with unknown *vector of parameters $p_{k,i}$ for any $i > k$, if any.*

More specifically, $Y_{k,i} = c$ if an element x of group i is (wrongly) classified as class c of group k and $Y_{k,i} = 0$, otherwise. Furthermore, vector $p_{k,i} = (p_{k,i,c})_{c \in \overline{\mathcal{C}}_k}$ such that $p_{k,i,c} = P(Y_{k,i} = c)$. That is, a perfect incremental classifier $f^*_{[k]}$ gives the right answer for the question 'Is x in a class of group k?' for any $x \in \mathcal{X}_{[k]}$. If the answer is affirmative, it also determines the class of x. However, for any $x \in \mathcal{X} \setminus \mathcal{X}_{[k]}$ the answer may be wrong and it is given by a categorical random variable. This is because the classifier $f^*_{[k]}$ has no information regarding the groups of classes posterior to the k-th group, if any. Therefore, a perfect incremental classifier becomes more *trustable* as k increases. We denote by $\mathcal{F}^*_{[k]}(\mathcal{X}, \mathcal{C})$ the set of all the perfect incremental classifiers that can perform the task $T_{[k]}(\mathcal{X}, \mathcal{C})$.

In the following proposition, we show that any multiclass classification task can be *sequenced* into a set of incremental classification tasks, which will be useful in the case of data only available incrementally, one group of classes at each stage. Equation (2) establishes a hierarchy, where recent perfect incremental classifiers, being more trustable, have preference over their predecessors in deciding the class of a data point. Therefore, the following proposition establishes a hierarchical sequencing (see Figure 2).

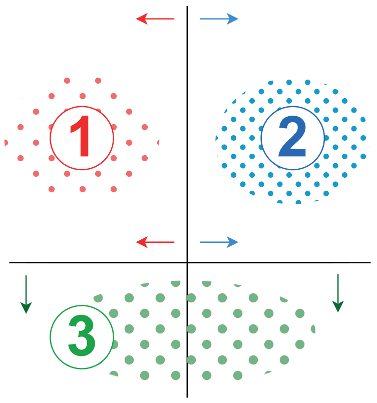

Figure 2: Hierarchical sequencing. In this figure, the 3-class classification task can be sequenced by the perfect incremental classifiers $f^*_{[1]}$ and $f^*_{[2]}$, trained at stages 1 and 2, respectively (see Figure 1). Classifier $f^*_{[1]}$ gives the output 1 or 2, to declare a data point as class 1 or class 2, respectively (even for points of class 3). Classifier $f^*_{[2]}$ gives the output 3 or 0, to declare a data point as class 3 or 'no class 3', respectively with 100% accuracy. In the hierarchical sequencing (Proposition 2), the set of classifiers $\{f^*_{[1]},\ f^*_{[2]}\}$ is combined into a single classifier $f^*(x) = \max\{f^*_{[1]}(x), f^*_{[2]}(x)\}$, which works as follows. Given a data point, say x_0, we have two cases. Case 1): If $f^*_{[2]}(x_0) = 3$ then x_0 can be classified as class 3 with 100% accuracy. It does not matter the class declared by $f^*_{[1]}(x_0)$ (class 1 or class 2). If, for example, $f^*_{[1]}(x_0) = 2$ we would have $f^*(x_0) = \max\{2, 3\} = 3$, the right class. Case 2): If $f^*_{[2]}(x_0) = 0$, then x_0 can be classified as 'no class 3' with 100% accuracy. In this case, $f^*_{[1]}(x_0)$ will give the class of x_0 with 100% accuracy (class 1 or class 2). If, for example, $f^*_{[1]}(x_0) = 2$ we would have $f^*(x_0) = \max\{2, 0\} = 2$, the right class. In summary, the class of x_0 can be determined by $\max\{f^*_{[1]}(x_0), f^*_{[2]}(x_0)\}$ with 100% accuracy. This formula establishes a hierarchy, where the more recent classifier $f^*_{[2]}$ has preference over its predecessor $f^*_{[1]}$ in deciding the class of a data point (see Proposition 2).

Proposition 2 (*Hierarchical sequencing*)**.** *Let us consider a set of perfect incremental classifiers* $\{f^*_{[k]}\}_{k\in\mathcal{K}}$ *such that* $f^*_{[k]} \in \mathcal{F}^*_{[k]}(\mathcal{X},\mathcal{C})$. *If we define*

$$f^*(x) = max_{k\in\mathcal{K}} \left\{ f^*_{[k]}(x) \right\} \qquad \textit{for all } x \in \mathcal{X}, \tag{2}$$

then, $f^* \in \mathcal{F}^*(\mathcal{X},\mathcal{C})$. *Therefore, the multiclass classification task* $T(\mathcal{X},\mathcal{C})$ *can be sequenced into the set of incremental classification tasks* $\{T_{[k]}(\mathcal{X},\mathcal{C})\}_{k\in\mathcal{K}}$.

Proof: Let us prove that f^* is a perfect multiclass classifier associated to task $T(\mathcal{X},\mathcal{C})$. Two steps: Step 1) Let us see that if $x \in \mathcal{X}_{c_0}$, such that $c_0 \in \mathcal{C}_{k_0}$, then $f^*(x) = c_0$. Given the hypothesis,

$$\begin{aligned} f^*_{[k]}(x) &\in \overline{\mathcal{C}}_k && \text{for all } k < k_0 \\ f^*_{[k_0]}(x) &= c_0 && \\ f^*_{[k]}(x) &= 0 && \text{for all } k > k_0. \end{aligned}$$

Thus

$$\begin{aligned} f^*(x) &= \max_{k\in\mathcal{K}} \left\{ f^*_{[k]}(x) \right\} \\ &= \max\Big\{ \ \max_k\{f^*_{[k]}(x) \mid k < k_0\}, \quad c_0, \quad 0 \Big\} \\ &= c_0, \end{aligned}$$

where we have used that

$$\max_k\{f^*_{[k]}(x) \mid k < k_0\} \ < \ c_0. \tag{3}$$

We prove (3) in two steps. Step 1.a) Given that

$$\{f^*_{[k]}(x) \mid k < k_0\} \subset \bigcup_{k<k_0} \overline{\mathcal{C}}_k = [0 : (k_0 - 1)L],$$

it is clear that

$$\max_k\{f^*_{[k]}(x) \mid k < k_0\} \leq (k_0 - 1)L.$$

Step 1.b) Given that

$$c_0 \in \ \mathcal{C}_{k_0} = [(k_0 - 1)L + 1 \ : \ k_0 L].$$

it is also clear that $(k_0 - 1)L + 1 \leq c_0$, which proves (3).

Step 2) For any $x \in \mathcal{X}$, let us see that if $f^*(x) = c_0 \in \mathcal{C}_{k_0}$ then $x \in \mathcal{X}_{c_0}$. By hypothesis,

$$f^*(x) = \max_{k \in \mathcal{K}} \left\{ f^*_{[k]}(x) \right\} = c_0,$$

which implies:

$$\begin{aligned} f^*_{[k]}(x) &\in \overline{\mathcal{C}}_k && \text{for all } k < k_0 \\ f^*_{[k_0]}(x) &= c_0 && \qquad (4) \\ f^*_{[k]}(x) &= 0 && \text{for all } k > k_0. \qquad (5) \end{aligned}$$

Now, let us prove by contradiction that $x \in \mathcal{X}_{[k_0]}$. If we suppose that $x \notin \mathcal{X}_{[k_0]}$, then there would exist $k_1 \neq k_0$ such that $x \in \mathcal{X}_{[k_1]}$. We distinguish two subcases: Subcase 2.a) If $k_1 > k_0$ this would imply that $f^*_{[k_1]}(x) > 0$ which contradicts (5). Subcase 2.b) If $k_1 < k_0$ this would imply that $f^*_{[k_0]}(x) = 0$ which contradicts (4). Thus, $x \in \mathcal{X}_{[k_0]}$. More specifically, $x \in \mathcal{X}_{c_0} \subset \mathcal{X}_{[k_0]}$ since $f^*_{[k_0]}(x) = c_0$. ■

4. Hierarchical sequencing with incremental CNNs

The theoretical results of Section 3 are based on perfect incremental classifiers (100% accuracy). However, in real life we have good, but imperfect, classifiers as is the case of convolutional neural networks (CNN). The objective of this section is to design an effective algorithm for class incremental learning. This algorithm will use CNNs and will be based on the task sequencing summarized in Proposition 2.

Let us define the incremental data set up to group k as $\mathcal{D}_{[k]} = \mathcal{D}_{\mathcal{C}_1} \cup \ldots \cup \mathcal{D}_{\mathcal{C}_k}$, where $\mathcal{D}_{\mathcal{C}_k} = \mathcal{D}_{(k-1)L+1} \cup \ldots \cup \mathcal{D}_{kL}$, for any $k \in \mathcal{K}$. For any given $c \in \mathcal{C}$, data set $\mathcal{D}_c \subset \mathcal{X}_c \times \mathcal{Y}_c$ contains data pairs of class c, i.e., $\mathcal{D}_c = \{(x^{(i)}, y^{(i)}\}_{i \in \mathcal{I}}$, where $x^{(i)}$ is the i-th example and $y^{(i)}$ is the corresponding class and $\mathcal{I} = \{1, \ldots, I\}$ (we are assuming that the number data pairs is the same in all the classes to lightening the notation).

Definition 7 (*Incremental CNN*)**.** *Given any $k \in \mathcal{K}$, an incremental CNN for task $T_{[k]}(\mathcal{X}, \mathcal{C})$, corresponds to a CNN $f_{[k]}(x; \theta_k) : \mathcal{X} \to [0,1]^{L+1}$ such that:*

$$f_{[k]}(x; \theta) = g_k(\Phi_k(x, \theta_{k1}), \theta_{k2}),$$

where Φ_k is the convolutional base, $\Phi_k(x, \theta_{k1}) : \mathbb{R}^n \to \mathbb{R}^m$, function g_k is a multiclass classifier, $g_k(y, \theta_{k2}) : \mathbb{R}^m \to [0,1]^{L+1}$ and vector $\theta = (\theta_{k1}, \theta_{k2})$ accounts for all the trainable parameters.

Algorithm 1 *(Hierarchical sequencing with incremental CNNs).*

- *Objective:* To perform hierarchical sequencing by using incremental CNNs.
- *Input:*
  1) A set of groups of classes indexed by $k \in \mathcal{K}$, a set of incremental data sets $\{\mathcal{D}_{[k]}\}_{k \in \mathcal{K}}$. The number of classes per group is L and the total number of classes is $C = KL$. Select a constant threshold vector $\gamma = (\gamma_1, \ldots, \gamma_C) \in (0.5, 1)^C$.
  2) A set of incremental CNNs $\{f_{[k]}(x; \theta_k)\}_{k \in \mathcal{K}}$, where θ_k is the initial vector of weights (notice that data set $\mathcal{D}_{[k]}$ is the training set for $f_{[k]}$). The output of each incremental CNN is a vector of logits in $\mathbb{R}^{L+1}$. Exception: The output of the first incremental CNN has dimension L (in this case there is not local class 0 for the previous classes).
- *Output:* A multiclass classifier $f_\gamma : \mathcal{X} \to \mathcal{C}$, based on the set of optimal incremental CNNs, $\left\{f_{[k]}(x; \theta_k^*)\right\}_{k \in \mathcal{K}}$.
- *Steps:*
  1) Train the CNNs. For the incremental steps $k = 1, \ldots, K$:
     a) Relabel the data of $\mathcal{D}_{[k]}$ by using the local labels $[0 : L]$:
        i) Case $k = 1$. Use local labels $[1 : L]$ for classes in group $\mathcal{C}_1$ (no local label 0 is used in this case).
        ii) Case $k > 1$. Use local labels $[1 : L]$ for classes in group $\mathcal{C}_k$ and local label 0 for classes in $\mathcal{C}_1 \cup \ldots \cup \mathcal{C}_{k-1}$.

b) Train the kth incremental CNN to obtain θ_k^*, by using the cross-entropy loss and $\mathcal{D}_{[k]}$ as training data, with balanced batches (the same number of images per local label: $0, 1, .., L$).

2) Define the multiclass classifier f_γ as follows:

a) Compute the vector of estimated probabilities $\widehat{y}(x)$ as follows. Given any $x \in \mathcal{X}$:

i) For each group $k \in \mathcal{K}$, set the local vector of logits $z_{[k]}(x) \in \mathbb{R}^L$:

$$z_{[k]}(x) = \big([f_{[k]}(x; \theta_k^*)]_j \big)_{j=2}^{L+1}.$$

Notice that the first component ($j = 1$) is not used here since it corresponds to the local class 0.

ii) Compute the vectors of local probabilities

$$\widehat{y}_{[k]}(x) = \text{softmax}\big(z_{[k]}(x) \big) \qquad \text{for all } k \in \mathcal{K}. \tag{6}$$

Notice that each component $[\widehat{y}_{[k]}(x)]_c$ is an estimation of $P\big(x \in \mathcal{X}_c \big)$, for any class $c \in \mathcal{C}_k$.

iii) Set the vector of estimated probabilities $\widehat{y}(x) = \big(\widehat{y}_{[k]}(x) \big)_{k \in \mathcal{K}} \in [0, 1]^C$.

b) Define the set of candidate classes for x as $\mathcal{C}_\gamma(x) = \{c \mid \widehat{y}_c(x) \geq \gamma_c\}$, for a given threshold vector γ and a data pair $(x, y) \in \mathcal{D}$.

c) Define the multiclass classifier:

$$f_\gamma(x) = \begin{cases} \max \mathcal{C}_\gamma(x) & \text{if } \mathcal{C}_\gamma(x) \neq \emptyset \\ \text{argmax}_{c \in \mathcal{C}} \{\widehat{y}_c(x)\} & \text{if } \mathcal{C}_\gamma(x) = \emptyset. \end{cases} \tag{7}$$

■

In the the previous algorithm notice that:

- In Definition 6 we have seen that the output of a perfect incremental classifier is 0 or $c \in \mathcal{C}$, the predicted global label. In contrast, in Algorithm

1 the predicted label given by the CNN is formatted as a one-hot vector. Furthermore, this label is local and is translated into the corresponding global label by using vector $\widehat{y}(x)$, set at Step 2.a.ii).

- The components of vector $\widehat{y}(x)$ are in the interval $[0, 1]$ in contrast with the pure binary outputs of perfect incremental classifiers. For this reason, in Step 2.b) we use the vector of thresholds γ to select the components of $\widehat{y}(x)$ which are close enough to 1.
- Equation (7) adapts the hierarchical sequencing given by equation (2) and intended for a perfect incremental classifier, for the case of an imperfect incremental classifier.

4.1. Fast incremental learning by transfer learning

The objective of this section is to show how Algorithm 1 can be accelerated by means of transfer learning (TL), in order to gain efficiency. Let us call it Algorithm 1-TL. Transfer learning is the improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned (Zhuang et al., 2021). A popular transfer learning approach in artificial vision corresponds to the use of a pretrained convolutional networks (pre-CNN) to set a new convolutional networks (new-CNN). In any CNN two parts are considered: the convolutional base, a series of convolutional and pooling layers, followed by a densely connected classifier. The simplest transfer learning approach, the so called feature extraction, consist of using the convolutional base of a pre-CNN as the convolutional base of the new-CNN. Then, in the new-CNN the parameters of the convolution base remain frozen and only the parameters of the densely connected classifier are trained (Chollet, 2017). Specifically, in Algorithm 1-TL we use the following incremental CNNs:

$$f_{[k]}(x; \theta_{k2}) = g_k(\Phi(x), \theta_{k2}) \qquad\qquad \text{for all } k \in \mathcal{K},$$

where $\Phi(x)$ is the convolutional base imported from a pre-CNN.

Algorithm 1-TL has some advantages compared to Algorithm 1:

- It allows for a simpler and faster training of the incremental CNNs, since only the densely connected classifier is trained.

- Additionally, one can run the pretrained convolutional base over the full training data set, record the feature vectors (one per image) and then use them as the input data to train the densely connected classifier. This approach is fast since it only requires to run the convolutional base once for every input image and the convolutional base is the most expensive part of the CNN (Chollet, 2017).

- The catastrophic forgetting is thus avoided, since each densely connected classifier is trained only once and later used with the same convolutional base used at training time. However, even without catastrophic forgetting, we observe a drop of accuracy in our approach compared to the classical non incremental approach. As we will see in our experiments, this is mainly due to the limited number of exemplars of already learned classes used in the learning process of new classes (storage constraints).

- The overall performance of the multiclass classifier f_γ depends on the performance of the pretrained CNN as we will see in the experiments. Therefore, it is likely that this approach will benefit from future improvements on pretrained CNNs.

- The output of Algorithm 1 is a multiclass classifier f_γ, based on a set of incremental CNNs $f_{[k]}$ (as many as incremental steps). In contrast, in Algorithm 1-TL, this set can be merged into a single CNN by using the common convolutional base in all the incremental CNNs. In the output of this single CNN, say $\widehat{y}(x)$, the common convolutional base is followed by the densely connected classifiers stacked in a vector, that is, $\widehat{y}(x) = \big(q_k(x)\big)_{k\in\mathcal{K}}$, where

$$q_k(x) = \big([g_k(\Phi(x), \theta^*_{k2}]_j\big)_{j=2}^{L+1}.$$

In summary, the output of Algorithm 1-TL is a multiclass classifier f_γ based on a single CNN.

The disadvantage of freezing the convolutional base is some potential accuracy loss. It is possible that by also training the pretrained convolutional base with the incremental data, one would obtain a feature extractor better adapted to the user classes, which in turn could improve the overall accuracy. Nonetheless, in this paper we will use a frozen convolutional base, knowing that there is room for future research in this question.

So far we have gained training speed by freezing the convolutional base. In the architecture of the fully connected classifier there is also room to accelerate the training. For example, the use of linear classifiers allows for a simpler an faster training compared to nonlinear classifiers. In this paper we will concentrate on linear classifiers since they are fast to train and can give good results as we will see in the experiments.

4.2. Threshold optimization

The accuracy of the multiclass classifier f_γ computed by Algorithm 1 or Algorithm 1-TL depends on the threshold vector γ, which is set by the user heuristically. A more convenient way would be to look for an optimal γ^*, which would minimize the classification error that can be attained by the family of classifiers $\{f_\gamma\}$ for all suitable γ. This can be done by the following algorithm.

Algorithm 2 *(Threshold optimization).*

- *Objective:* To obtain an optimal multiclass classifier $f_{\gamma^*}(x)$.
- *Input:*
    1) A set of classes indexed by $c \in \mathcal{C}$, the last incremental data set $\mathcal{D}_{[K]}$, which accounts for all the training data available at the last step K of Algorith 1.
    2) A box of suitable γ, say, $\mathcal{B} = [0.5, 1]^C$, that bounds the threshold vector $\gamma = (\gamma_1, \ldots, \gamma_C)$.
    3) A set of optimal incremental CNNs, that is, $\left\{f_{[k]}(x; \theta_k^*)\right\}_{k \in \mathcal{K}}$ (obtained by Algorithm 1).

- *Output:* An optimal multiclass classifier $f_{\gamma^*} : \mathcal{X} \to \mathcal{C}$, where γ^* is the optimal threshold vector.

- *Steps:*

  1) Define the multiclass classifier f_γ (see Step 2 of Algorithm 1).

  2) Define the error function $E(\gamma) : \mathcal{B} \to [0, 1]$ as follows:

$$E(\gamma) = 1 - \frac{1}{N} \sum_{(x,y)\in\mathcal{D}} \Big[f_\gamma(x) = y \Big], \tag{8}$$

  where $[\cdot]$ is the Iverson bracket ($[P] = 1$ if P is true and $[P] = 0$, otherwise).

  3) Solve the following threshold optimization problem:

$$\gamma^* = \operatorname{argmin}_{\gamma\in\mathcal{B}} \quad E(\gamma),$$

  which defines an optimal multiclass classifier $f_{\gamma^*}(x)$. ■

Notice that:

- This threshold optimization has to be performed after training the set of incremental CNNs of Algorithm 1 or 1-TL and by using the training data.

- The error function $E(\gamma)$ in (8) is highly complex since it is based on the Iverson bracket and $f_\gamma(x)$. Thus, in order to solve the threshold optimization problem one can use Powell's method (Powell, 1964) which does not require to use derivatives and can consider box constraints.

- Powell's method gives a local optimizer. For this reason one can try and compare different initial points, as for example $\gamma_0 = 0.75(1, 1, \ldots, 1)$, the center of the box $\mathcal{B}$ or, also, a random γ_0 from the box.

4.3. The proposed method: HILAND

In this section we summarize the proposed method, HILAND (Hierarchical Incremental Learning Appending Naive Discriminators), which corresponds to run Algorithm 1-TL followed by Algorithm 2, such that, at each incremental step of Algorithm 1-TL, only a reduced number of exemplars of already learned classes is stored. Our source code is publicly available online[1].

The HILAND method qualifies as a class incremental learning (CIL) algorithm since:

1) It provides the multiclass classifier f_{γ^*} that is trainable from a sequence of data batches in which different classes occur at different batches.

2) The multiclass classifier f_{γ^*} is competitive for the classes observed so far at any time.

3) The computational requirements and memory footprint of the HILAND method grow very slowly with respect to the number of classes seen so far, as we explain below.

HILAND is a CIL method of type *replay,* since it stores a reduced number of exemplars in raw format of already learned classes. To alleviate forgetting, these exemplars are replayed while learning new classes by means of the incremental data sets $\mathcal{D}_{[k]}$ (see Algorithm 1). The HILAND method stores the same number of exemplars per class. Although the default selection method for these exemplars is *random sampling,* other selection method can be used as for example *herding* (Rebuffi et al., 2017). In summary, the HILAND method corresponds to run Algorithm 1-TL followed by Algorithm 2, where, at each incremental step, only a reduced number of exemplars of already learned classes is stored.

5. Experiments

We will test two versions of the HILAND method: The first version, HILAND-I, corresponds to the pure HILAND method as described in Section 4.3. The second version, HILAND-II, corresponds to the HILAND method with the modification described in Algorithm 1.b (see Section 5.2 for details).

[1]https://github.com/capo-urjc/HILAND

5.1. Testing HILAND-I

In this section we describe the experiments performed to evaluate HILAND-I and analyze their results. Our code has been implemented in PyTorch (Paszke et al., 2019) 1.7.1 and we have used the following CNNs from the PyTorch library: VGG-16, Densenet-161, Inception-v3 and ResNet-34 (all of them pretrained on ImageNet). The experiments have been conducted on a Intel Xeon E5-2698v4 CPU and a NVIDIA Tesla V100 GPU with 32 GB of RAM. We have run all the experiments ten times with different class orders and have reported average values, according to the experimental design presented in (Rebuffi et al., 2017). In order to reduce the required computational time, in the context of transfer learning, we have used the pretrained convolutional base, out-of-the-box as provided by the PyTorch library. That is, we have used a saved network that was previously trained on a large data set (ImageNet in this case). Then, we have run the pretrained convolutional base over the full training data set, recorded the feature vectors (one per image) and used them as the input data to train the densely connected classifier. Notice that in all of the following experiments we use the transfer learning/pretraining described in this paragraph.

5.1.1. Experiment 1: Testing Algorithm 1-TL

In this experiment, we perform two small tests to illustrate the use of Algorithm 1-TL based on the FashionMNIST data set (Xiao et al., 2017). This data set consists of 28x28 pixel gray scale images of clothing items grouped into 10 classes. Each class consists of 6000 training images and 1000 test images.

In Test 1 the FashionMNIST classification problem is solved in a non incremental setting by means of a multiclass CNN with a pretrained convolutional base combined with a multiclass linear classifier (softmax output). That is, we use a pretrained VGG-16 network (Simonyan & Zisserman, 2015) as the feature extractor and a fully connected network (FCN) to solve the classification problem. The input and output dimensions of the FCN are 25088 and 10, respectively. In the context of transfer learning, only the FCN is trained (250,890 training parameters). We use the stochastic gradient descent (SGD) with the following training parameters: Batch size, 200 images; learning rate, 0.01; momentum parameter, 0.9. After 8550 SGD iterations the multiclass CNN attains a test accuracy of 92.29% (training time 104 seconds).

In Test 2 the same classification problem is solved in an incremental setting by Algorithm 1-TL, adding one single class at each incremental step. That is, in Algorithm 1-TL, we consider 10 groups of classes with one class each ($L = 1, K = C = 10$). In each incremental CNN, the pretrained convolutional base (VGG-16) is frozen and combined with a binary linear classifier (sigmoid output). The linear classifier corresponds to a FCN whose input and output dimensions are 25088 and 1, respectively. In the context of transfer learning, only the FCN is trained and we use the SGD method with the same tuning parameters as in Test 1 (the total number of training parameters is the same as in Test 1). However, we consider an increasing number of iterations for each incremental CNN $f_{[k]}$ since the size of each incremental data set $\mathcal{D}_{[k]}$ also increases with k. Specifically, we use $200+150k$ SGD iterations to train $f_{[k]}$ ($k = 1, 2, ...$), which in total amounts to 8550 SGD iterations to learn the ten classes incrementally (training time 155 seconds). All the thresholds γ_c are set to 0.5. The resulting multiclass classifier f_γ attains a test accuracy of 90.41%. Notice that in this example the incremental CNNs are binary classifiers. Thus, in the first incremental step of Algorithm 1-TL, in order to have a binary classification problem, we have to consider simultaneously the global classes 1 and 2 (with local labels 0 and 1). In the second incremental step we have to consider the global class 3 (with local label 1) and the global classes 1 and 2 (with local label 0), and so on.

5.1.2. Experiment 2: Testing Algorithm 1-TL under data reduction

The objective of this experiment is to show the impact of data reduction in Algorithm 1-TL. In Experiment 1 we have seen that Algorithm 1-TL obtains a test accuracy of 90.41% when solving the FashionMNIST classification problem, by using the full training set (60,000 exemplars). This accuracy drops to 81.77% after limiting the total number of stored exemplars to $N = 2000$. More specifi-

Table 1: Results on FashionMNIST.

Experiment	**Algorithm**	**Accuracy (%)**	**Training time (s)**
1	Multiclass CNN-TL	92.29	104
1	Alg. 1-TL	90.41	155
2	Alg. 1-TL + Data reduction	81.77	44
3	Alg. 1-TL + Data reduction + Alg. 2 (HILAND-I method)	84.67	44

cally, in Algorithm 1-TL, just before learning the kth class, $N/(k-1)$ exemplars of each already learned class are *randomly* selected and used for training. The rest of examples are dismissed. On the other hand, all the available exemplars of the current class, the kth one, are used for training (6000 exemplars). In this experiment, we use the SGD method with the same tuning parameters as in Test 2. However, we consider a constant number of iterations to train each incremental CNN $f_{[k]}$ since the size of each incremental data set $\mathcal{D}_{[k]}$ is also constant (due to data reduction). Specifically, we use 350 SGD iterations to train each incremental CNN which in total amounts to 3150 SGD iterations (training time 44 seconds).

5.1.3. Experiment 3: Testing the HILAND-I method

In this experiment, we test the HILAND-I method. As already pointed out, the HILAND-I method corresponds to run Algorithm 1-TL followed by Algorithm 2. Furthermore, at each incremental step of Algorithm 1-TL, only a reduced number of exemplars of already learned classes is stored (data reduction). In Experiment 2 we have seen that Algorithm 1-TL obtains a test accuracy of 81.77% when solving the FashionMNIST classification problem, by using a reduced training set (2000 exemplars). This accuracy rises up to 84.67% if one optimizes the vector of thresholds γ by Algorithm 2 (after training the neural network by Algorithm 1-TL). In Algorithm 2 we have used: the bounding box $\mathcal{B} = [0.5, 1]^C$, a random initial vector γ_0 from the bounding box and the

Powell's optimization method [2]. For the SGD method, we have used the same tuning parameters as in Experiment 2.

Table 1 summarizes the results of Experiments 1 to 3. In Experiment 1, we observe that both approaches, non incremental (multiclass CNN-TL) and incremental (Alg. 1-TL) achieve close accuracies. Experiment 2 shows that data reduction has the largest impact to worsen the accuracy. Experiment 3 shows that optimizing the threshold vector (Alg. 2) has a relevant impact to improve the accuracy. In summary, for the FashionMNIST data set, the HILAND-I method attains an overall accuracy of 84.67% compared to 92.29%, the upper bound given by the non incremental counterpart.

5.2. Testing HILAND-II

In this section we introduce the HILAND-II method which, in our preliminary tests, has obtained better accuracy results than the HILAND-I method for large multiclass classification problems (CIFAR-100 and CUB-200). HILAND-II is nothing but HILAND-I based on the following Algorithm 1.b, a second version of Algorithm 1. Notice that in the following algorithm we only describe the step that has has been modified in Algorithm 1 (Step 2.a).

Algorithm 1.b

- *Steps:*

    2) Define the multiclass classifier f_γ as follows:

        a) Compute the vector of estimated probabilities $\widehat{y}(x)$ as follows. Given any $x \in \mathcal{X}$:

            i) For each group $k \in \mathcal{K}$, set the local vector of logits $z_{[k]}(x) \in \mathbb{R}^L$:

$$z_{[k]}(x) = \left([f_{[k]}(x; \theta_k^*)]_j \right)_{j=2}^{L+1}.$$

[2]Implemented in the function `scipy.optimize.minimize` from the Scipy library, version 1.6.1 (Virtanen et al., 2020).

ii) Set the global vector of logits $z(x) = \big(\, z_{[k]}(x) \,\big)_{k \in \mathcal{K}} \in \mathbb{R}^C$.

iii) Compute the vector of global probabilities

$$\widehat{y}(x) = \mathrm{softmax}\big(\, z(x) \,\big), \tag{9}$$

where each component $\widehat{y}_c(x)$ is an estimation of $P\big(\, x \in \mathcal{X}_c \,\big)$, for any class $c \in \mathcal{C}$. ■

Notice that the main difference between Algorithms 1 and 1.b is on the use of the softmax function in the definition of the multiclass classifier. Specifically, Algorithm 1 applies the softmax function to each local vector of logits $z_{[k]}(x)$ (equation (6)). In contrast, Algorithm 1.b applies the softmax function to the global vector of logits $z(x)$ (equation (9)).

5.2.1. Experiment 4: Comparing HILAND-II with state-of-the-art methods on CIFAR-100

The objective of this experiment is to compare the HILAND-II method with two state-of-the-art methods, namely iCaRL (Rebuffi et al., 2017) and BiC (Wu et al., 2019), which have been introduced in Section 1. The data set employed is CIFAR-100 (Krizhevsky, 2009). This data set contains 100 classes with 600 RGB images each, 500 for training and 100 for testing. Each image has a size of 32×32 pixels. iCaRL, and BiC use a ResNet-32 (see (He et al., 2016)), which is trained from scratch in these methods. However, in HILAND-II, which is based on transfer learning, we have used a pretrained ResNet-34 (there is not a pretrained ResNet-32 in the official PyTorch library).

These three methods are used to solve the CIFAR-100 classification problem in an incremental setting, adding a group of ten classes at each incremental step ($L = K = 10, C = 100$). Following (Rebuffi et al., 2017), we run all the experiments ten times with different class orders and report average values. We also follow the *benchmark protocol* proposed in (Rebuffi et al., 2017): For a given multiclass data set, the classes are arranged in a fixed random order. Each method is then trained in a class-incremental way on the available training data. After each group of classes, the resulting classifier is evaluated on the test part data

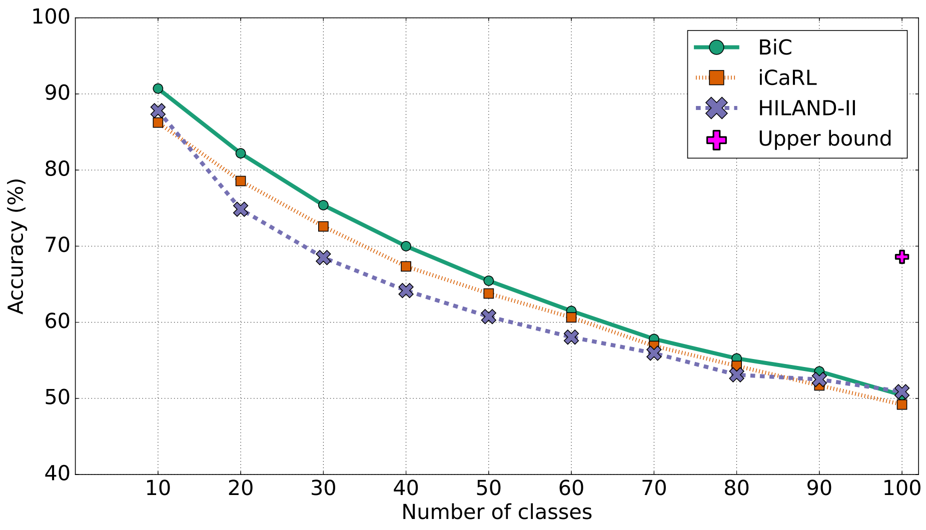

Figure 3: Experiment 4 - Comparison between HILAND-I and state-of-the-art methods on CIFAR-100. The 'Upper bound' corresponds to the accuracy obtained by the non incremental multiclass CNN.

of the data set, considering only those classes that have already been trained. The result of the evaluation are depicted as curves of the classification accuracy values after each group of classes (see Figure 3). We also report their average values, called *average incremental accuracy (AIA)* (Rebuffi et al., 2017) (see Table 2). As an upper bound to the final accuracy after incrementally learning the 100 classes, the CIFAR-100 classification problem was also solved in (Rebuffi et al., 2017) by means of a multiclass CNN with convolutional base (ResNet-32) combined with a multiclass linear classifier (non incremental setting).

In Figure 3 and Table 2 we have the results of the benchmark protocol for iCaRL and BiC as reported in (Rebuffi et al., 2017) and (Wu et al., 2019), respectively. The HILAND-II results have been obtained by following the same benchmark protocol. However, in contrast with the other two methods, only the linear FCN is trained in the context of transfer learning. The SGD method is used with the following training parameters: Batch size, 200 images; learning rate, 0.01; momentum parameter, 0.9; SGD iterations, 350 to train each incre-

Table 2: Experiment 4 - Comparison between HILAND-II and state-of-the-art methods on CIFAR-100. 'Accuracy' reports the final accuracy after learning the 100 classes. 'AIA' stands for Average Incremental Accuracy. 'Epochs' reports the number of epochs per training step. The 'Upper bound' corresponds to the accuracy obtained by the non incremental multiclass CNN.

Approach	**Accuracy (%)**	**AIA (%)**	**Epochs**
Upper bound	68.60	-	-
HILAND-II	50.88	62.65	10
BiC	50.43	66.20	250
iCaRL	49.19	64.10	70

mental CNN which in total amounts to 3500 SGD iterations. In Algorithm 2 we have used the bounding box $\mathcal{B} = [0.5, 1]^C$ and a random initial vector γ_0 from the bounding box. The constant number of stored exemplars is $N = 2000$ for iCaRL, BiC and HILAND-II.

Table 2 reports the accuracy results and the number of epochs at each training step. One epoch corresponds to the 7000 images used at each training step: 2000 stored exemplars plus 10 classes with 500 images each. Compared with the other methods, we have obtained similar accuracy results but with less training epochs. Furthermore, the computing time per epoch is far less expensive in HILAND-II since, in the context of transfer learning, every input image goes through the convolutional base only once (we train the linear classifiers with feature vectors).

5.2.2. Experiment 5: Performance of the HILAND-II method with different CNNs on CIFAR-100

In this experiment we analyze the influence of using different CNN architectures on HILAND-II and CIFAR-100. We compare the results obtained by HILAND-II based on the following pretrained CNNs: VGG-16, ResNet-34, DenseNet-161 (Huang et al., 2017) and Inception-v3 (Szegedy et al., 2016). To train them, we use the same SGD method and training parameters as in

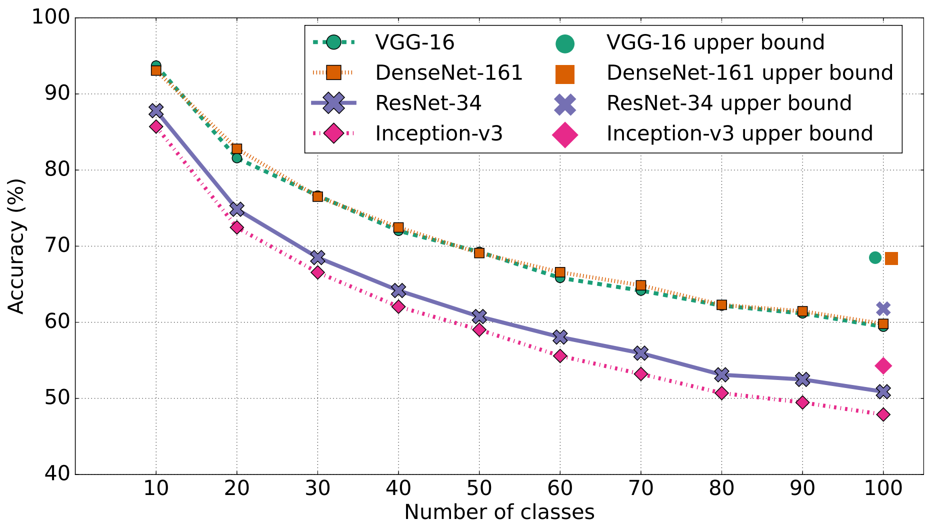

Figure 4: Experiment 5 - Performance of HILAND-II with different CNNs on CIFAR-100. The 'Upper bound' corresponds to the accuracy obtained by the corresponding non incremental multiclass CNN.

580 Experiment 4.

In Figure 4 and Table 3 we observe that the performance of HILAND-II is very similar for the pair (DenseNet-161, VGG-16) and for the pair (ResNet-34, Inception-v3). If we compare Figures 3 and 4, we observe that the accuracy obtained by HILAND-II based on a pretrained DenseNet-161 or VGG-16 is among the best results for the CIFAR-100 CIL problem. Finally, Figure 4 shows that the choice of the CNN has a significant influence on the performance of the HILAND-II method. However, as illustrated in Figure 5 all the pretrained CNNs exhibit a similar accuracy degradation between two consecutive incremental steps.

5.2.3. *Experiment 6: Comparing HILAND-II with state-of-the-art methods on CUB-200*

The objective of this experiment is to compare the HILAND-II method with three state-of-the-art methods, namely iCaRL (Rebuffi et al., 2017), FearNet

Table 3: Experiment 5 - Performance of HILAND-II with different CNNs on CIFAR-100. 'Accuracy' reports the final accuracy after learning the 100 classes. 'AIA' stands for Average Incremental Accuracy. 'Upper bound' reports the accuracy obtained by the corresponding non incremental multiclass CNN.

Pretrained CNN	**Accuracy (%)**	**AIA (%)**	**Upper bound (%)**	**Training time (s)**
DenseNet-161	59.79	70.88	68.36	339
VGG-16	59.41	70.59	68.49	295
ResNet-34	50.88	62.65	61.78	213
Inception-v3	47.87	60.26	54.28	324

(Kemker & Kanan, 2018) and Xiang *et al.* (Xiang et al., 2019). The data set employed is CUB-200 (Wah et al., 2011). This data set contains 11,788 images belonging to 200 classes of birds, 5,994 images for training and 5,794 for testing.

We use the same evaluation scheme proposed by Xiang *et al.* Finetuning is performed on the backend ResNet-50 with the first 100 classes of the data set. Then, the convolutional base of ResNet-50 is frozen to be used in the 10 incremental learning stages performed on the remaining 100 classes (i.e., only the linear classifiers are trained). We store 20 exemplars per class as in Experiment 4 ($N = 4000$).

The accuracy and AIA are calculated without taking into account the first 100 classes since Xiang *et al.* do not consider it an incremental stage. We also run all the experiments ten times with different random class orders and report average values. Training, evaluation and hyperparameters follow the same scheme described in the Section 5.2.1, but we train for 147 epochs. We have also applied a learning rate decay with a factor of 0.1 after 55 SGD iterations. In Figure 6 and Table 4 we have the accuracy results for iCaRL, FearNet and Xiang's method as reported in (Xiang et al., 2019). We observe that HILAND-II and Xiang's method obtain similar results and clearly outperform iCarL and FearNet.

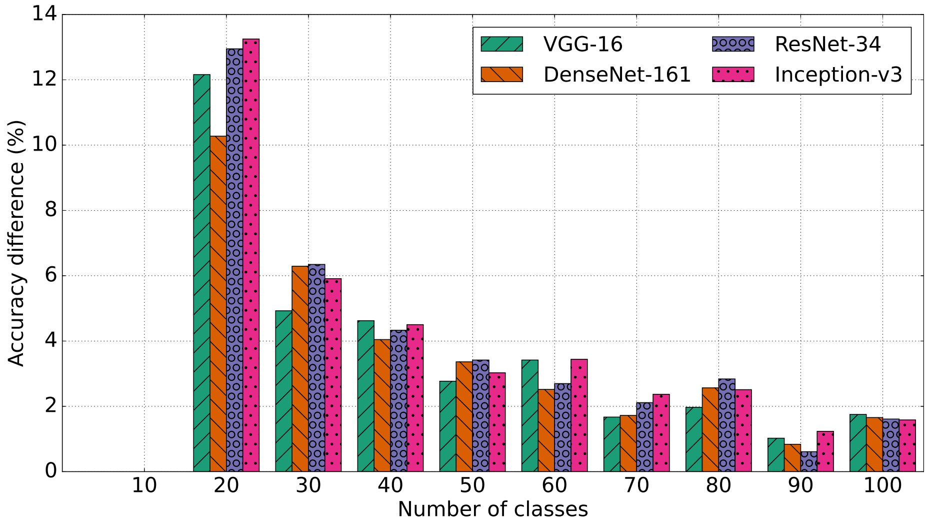

Figure 5: Experiment 5 - Incremental accuracy difference for each pair of tasks for all backends with HILAND-II.

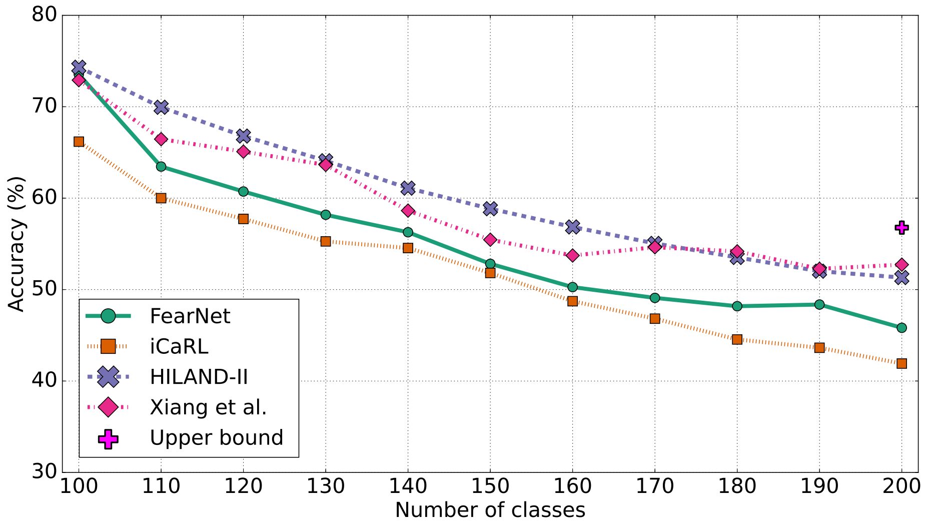

Figure 6: Experiment 6 - Comparison between HILAND-II and state-of-the-art methods on CUB-200. The 'Upper bound' corresponds to the accuracy obtained by the non incremental multiclass CNN.

Table 4: Experiment 6 - Comparison between HILAND-II and state-of-the-art methods on CUB-200. 'Accuracy' reports the final accuracy after learning the 200 classes. 'AIA' stands for Average Incremental Accuracy. The 'Upper bound' corresponds to the accuracy obtained by the non incremental multiclass CNN.

Approach	**Accuracy (%)**	**AIA (%)**
Upper bound	56.77	-
iCaRL	41.90	50.50
FearNet	45.81	53.31
Xiang *et al.*	52.72	57.68
HILAND-II	51.31	58.95

Table 5: Experiment 7 - Performance of HILAND-II with different CNNs on CUB-200. 'Accuracy' reports the final accuracy after learning the 200 classes. 'AIA' stands for Average Incremental Accuracy. 'Upper bound' reports the accuracy obtained by the corresponding non incremental multiclass CNN

Pretrained CNN	**Accuracy (%)**	**AIA (%)**	**Upper bound (%)**	**Training time (s)**
DenseNet-161	59.14	66.22	69.64	2137
VGG-16	39.71	47.56	52.12	1552
ResNet-50	51.31	58.95	56.77	1454
Inception-v3	46.74	54.79	51.19	1662

5.2.4. Experiment 7 - Performance of HILAND-II with different CNNs on CUB-200

In this experiment we analyze the influence of using different CNN architectures on HILAND-II and CUB-200 following the same training, testing and hyperparameters criteria used in Experiment 6. We compare the results obtained by HILAND-II based on the following pretrained CNNs: VGG-16, ResNet-50, DenseNet-161 and Inception-v3. In Figure 7 and Table 5 we observe that DenseNet-161 obtains the best performance as observed in Experiment 5.

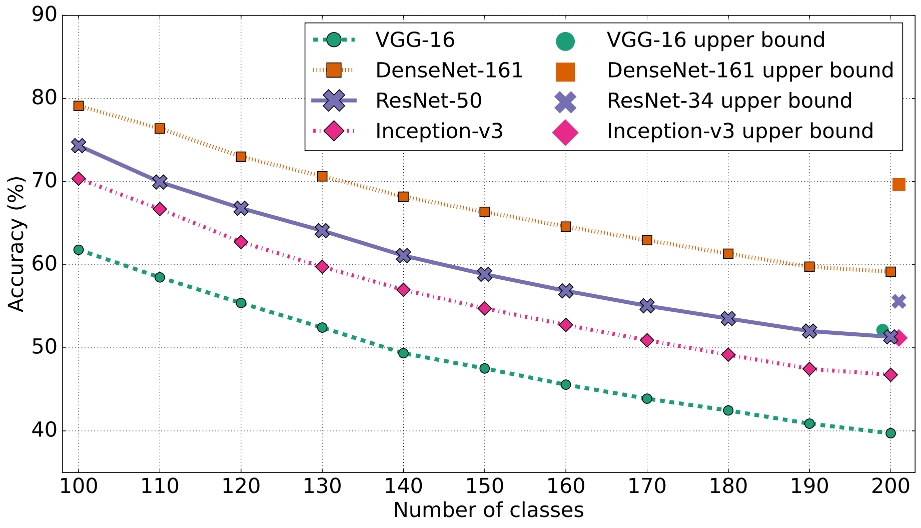

Figure 7: Experiment 7 - Performance of HILAND-II with different CNNs on CUB-200. The 'Upper bound' corresponds to the accuracy obtained by the corresponding non incremental multiclass CNN.

6. Conclusion

In this paper we have addressed the Class Incremental Learning (CIL) problem. From a theoretical point of view, we have proved that any CIL task can be sequenced into more simple incremental classification tasks by means of the hierarchical sequencing, based on perfect incremental classifiers. This kind of classifiers can be trained sequentially and independently, which is well suited for the CIL problem. From a practical point of view, we have adapted the previous theoretical results to design an effective CIL algorithm based on CNNs. More specifically, we have proposed the HILAND method for image classification, which combines the hierarchical sequencing with transfer learning (based on pretrained CNNs).

In our experiments, based on the FashionMNIST, CIFAR-100 and CUB-200 data sets, we have compared the HILAND method to state-of-the-art CIL algorithms. In terms of accuracy we obtained similar results but with far less training effort. We attribute this speedup to two factors. 1) *Transfer learning:* Under this approach, we do not retrain the pretrained convolutional base in the whole learning process. Furthermore, the computing time per epoch is far less expensive in HILAND by exploiting the so called feature extraction. That is, prior to train the incremental linear classifiers, the whole data set goes through the pretrained convolutional base only once to obtain the corresponding vectors of features, which are used repeatedly to train the incremental linear classifiers. 2) *Hierarchical sequencing:* By using this methodology, each incremental linear classifier is trained only once, at the corresponding incremental stage. On the other hand, in our experiments, we have also observed that the HILAND performance is highly dependent on the pretrained CNN used as feature extractor. Furthermore, the accuracy values obtained by HILAND combined with some of the state-of-the-art pretrained CNNs are among the best results for the CIFAR-100 and CUB-200 CIL problems. Regarding future research, we will focus on developing the HILAND method without storing old exemplars (the current version of HILAND is a replay method).

Acknowledgments: We thank the editor and reviewers for their constructive and stimulating suggestions. This research has been supported by the Spanish Government research funding RTI2018-098743-B-I00 and RTI2018-094269-B-I00 (MICINN/FEDER) and the Comunidad de Madrid research funding grant Y2018/EMT-5062.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z. et al. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. URL: `https://www.tensorflow.org/`.

Aljundi, R., Babiloni, F., Elhoseiny, M., Rohrbach, M., & Tuytelaars, T. (2018). Memory aware synapses: Learning what (not) to forget. In V. Ferrari, M. Hebert, C. Sminchisescu, & Y. Weiss (Eds.), *Computer Vision*

– *ECCV 2018* (pp. 144–161). Cham: Springer International Publishing. doi:10.1007/978-3-030-01219-9_9.

Bifet, A., Gavaldà, R., Holmes, G., & Pfahringer, B. (2018). *Machine Learning for Data Streams: with Practical Examples in MOA*. The MIT Press. doi:10.7551/mitpress/10654.001.0001.

Chen, Z., & Liu, B. (2018). Lifelong machine learning, second edition. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, *12*, 1–207. doi:10.2200/S00832ED1V01Y201802AIM037.

Chollet, F. (2017). *Deep Learning with Python*. Manning.

Delange, M., Aljundi, R., Masana, M., Parisot, S., Jia, X. et al. (2021). A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (pp. 1–1). doi:10.1109/TPAMI.2021.3057446.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2* NIPS'14 (p. 2672–2680). MIT Press. doi:10.5555/2969033.2969125.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 770–778). doi:10.1109/CVPR.2016.90.

Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 2261–2269). doi:10.1109/CVPR.2017.243.

Kemker, R., & Kanan, C. (2018). Fearnet: Brain-inspired model for incremental learning. In *International Conference on Learning Representations*.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., & Hadsell, R. (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, *114*, 3521–3526. doi:10.1073/pnas.1611835114.

Krizhevsky, A. (2009). Learning multiple layers of features from tiny images. URL: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

Lee, S., Chang, K., & Baek, J.-G. (2021). Incremental learning using generative-rehearsal strategy for fault detection and classification. *Expert Systems with Applications*, *184*, 115477. doi:https://doi.org/10.1016/j.eswa.2021.115477.

Li, Z., & Hoiem, D. (2018). Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *40*, 2935–2947. doi:10.1109/TPAMI.2017.2773081.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S. et al. (2016). Ssd: Single shot multibox detector. In B. Leibe, J. Matas, N. Sebe, & M. Welling (Eds.), *Computer Vision – ECCV 2016* (pp. 21–37). Cham: Springer International Publishing. doi:10.1007/978-3-319-46448-0_2.

Lopez-Paz, D., & Ranzato, M. (2017). Gradient episodic memory for continual learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* NIPS'17 (p. 6470–6479). Red Hook, NY, USA: Curran Associates Inc. doi:10.5555/3295222.3295393.

Lu, B.-L., & Ito, M. (1999). Task decomposition and module combination based on class relations: a modular neural network for pattern classification. *IEEE Transactions on Neural Networks*, *10*, 1244–1256. doi:10.1109/72.788664.

Mallya, A., & Lazebnik, S. (2018). Packnet: Adding multiple tasks to a single network by iterative pruning. In *2018 IEEE/CVF Conference on Computer*

Vision and Pattern Recognition (pp. 7765–7773). doi:10.1109/CVPR.2018.00810.

McCloskey, M., & Cohen, N. J. (1989). Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of Learning and Motivation - Advances in Research and Theory*, *24*, 109–165. doi:10.1016/S0079-7421(08)60536-8.

Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., & Wermter, S. (2019). Continual lifelong learning with neural networks: A review. *Neural Networks*, *113*, 54–71. doi:10.1016/j.neunet.2019.01.012.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J. et al. (2019). Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems 32* (pp. 8024–8035). Curran Associates, Inc. volume 32. URL: https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf.

Powell, M. J. D. (1964). An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The Computer Journal*, *7*, 155–162. doi:10.1093/comjnl/7.2.155.

Rebuffi, S.-A., Kolesnikov, A., Sperl, G., & Lampert, C. H. (2017). iCaRL: Incremental classifier and representation learning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 5533–5542). doi:10.1109/CVPR.2017.587.

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 779–788). doi:10.1109/CVPR.2016.91.

Serra, J., Suris, D., Miron, M., & Karatzoglou, A. (2018). Overcoming catastrophic forgetting with hard attention to the task. In *International Conference on Machine Learning* (pp. 4548–4557). PMLR.

Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. `arXiv:1409.1556`.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 2818–2826). doi:`10.1109/CVPR.2016.308`.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T. et al. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, *17*, 261–272. doi:`10.1038/s41592-019-0686-2`.

Wah, C., Branson, S., Welinder, P., Perona, P., & Belongie, S. (2011). Technical Report CNS-TR-2011-001 California Institute of Technology.

Wu, Y., Chen, Y., Wang, L., Ye, Y., Liu, Z. et al. (2019). Large scale incremental learning. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 374–382). doi:`10.1109/CVPR.2019.00046`.

Xiang, Y., Fu, Y., Ji, P., & Huang, H. (2019). Incremental learning using conditional adversarial networks. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (pp. 6618–6627). doi:`10.1109/ICCV.2019.00672`.

Xiao, H., Rasul, K., & Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. `arXiv:cs.LG/1708.07747`.

Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H., & He, Q. (2021). A comprehensive survey on transfer learning. *Proceedings of the IEEE*, *109*, 43–76. doi:`10.1109/JPROC.2020.3004555`.