Incremental Learning with Pre-Trained Convolutional Neural Networks and Binary Associative Memories

Ghouthi Hacene, Vincent Gripon, Nicolas Farrugia, Matthieu Arzel, Michel Jezequel

Incremental Learning with Pre-Trained Convolutional Neural Networks and Binary Associative Memories

Ghouthi Boukli Hacene, Vincent Gripon, Nicolas Farrugia, Matthieu Arzel and Michel Jezequel
name.surname@imt-atlantique.fr

IMT Atlantique, 655 avenue du technopole, Plouzan 29280, France

Abstract

Thanks to their ability to absorb large amounts of data, Convolutional Neural Networks (CNNs) have become state-of-the-art in numerous vision challenges, sometimes even on par with biological vision. They rely on optimisation routines that typically require intensive computational power, thus the question of embedded architectures is a very active field of research. Of particular interest is the problem of incremental learning, where the device adapts to new observations or classes. To tackle this challenging problem, we propose to combine pre-trained CNNs with binary associative memories, using product random sampling as an intermediate between the two methods. The obtained architecture requires significantly less computational power and memory usage than existing counterparts. Moreover, using various challenging vision datasets we show that the proposed architecture is able to perform one-shot learning – and even use only a small portion of the dataset – while keeping very good accuracy.

1. Introduction

For the past few years, Deep Neural Networks (DNNs), and in particular Convolutional Neural Networks (CNNs), have achieved state-of-the-art performance (Hong et al., 2015; Pan & Yang, 2010; Krizhevsky et al., 2012) in several domains of supervised learning, sometimes even being on par with the visual cortex (Cadieu et al., 2014). DNNs rely on hundreds of millions of parameters that are trained to deal with large amounts of data. In this context a major drawback of the method is the need for intensive computation and memory usage during the learning phase. This limitation is critical for embedded systems such as smartphones or sensor networks.

A lot of effort has been driven towards optimized hardware implementations of DNNs (Qiu et al., 2016). For example in (Iandola et al., 2016), the authors propose an architecture with state-of-the-art performance on the ImageNet challenge (Deng et al., 2012) using less than one megabyte, and in (Lin et al., 2016) a fixed point quantization of CNNs was introduced to reduce the network size. However, learning its parameters still require the processing of the whole dataset, involving many gradient computations. Moreover, the whole training dataset has to be stored in memory for learning.

Incremental methods provide solutions to process the learning data sequentially, using subsets of the training dataset. An incremental technique is defined as such (Polikar et al., 2000; 2001): a) it is able to learn additional information from new data (example-incremental), b) it does not require access to the original data used to train the existing classifiers (in order to limit memory usage), c) it preserves previously acquired knowledge (avoid catastrophic forgetting) and d) it is able to accommodate new classes that may be introduced with new data (class-incremental). Although models have been proposed and studied extensively during the last decades, finding a good compromise between accuracy and required resources remains challenging. Indeed, most of existing works retrain the model when receiving new data (Syed et al., 1999; Poggio & Cauwenberghs, 2001), and reuse some prior data for the retraining process (Polikar et al., 2001; Sun et al., 2016).

In this paper we propose an incremental learning model with the following claims:

- It is possible to adapt the model to new data without retraining it,
- It uses much less computational power than existing counterparts,
- It approaches state-of-art accuracy on challenging vi-

Proceedings of the 33^{rd} International Conference on Machine Learning, New York, NY, USA, 2016. JMLR: W&CP volume 48.

sion datasets (CIFAR10, ImageNet),

- It dramatically decreases the memory usage (by several orders of magnitude compared to nearest neighbour search),
- It only requires a few learning examples.

We point out that these claims are of particular interest when targeting embedded applications.

We rely on an increasingly popular method to benefit from the accuracy of DNNs without the need to train them on the targeted dataset, termed "transfer learning" (Girshick et al., 2014; Pan & Yang, 2010). The idea is to use pre-trained CNNs on large datasets as feature extractors, and retrain the final layer of DNNs.

In this work, we propose to combine transfer learning with binary associative memories to achieve fully incremental learning. Binary associative memories are devices that are able to perform one-shot learning with very limited resources. The output of the DNN is quantized in a specific manner to combine it with the binary associative memories. This solution allows processing data sequentially one subset at a time, without forgetting initially processed data, and using only a sample of the database for learning. An overview of the proposed method is depicted in Figure 1. We evaluate the proposed method on challenging vision datasets (ImageNet and CIFAR10), and compare both accuracy and resources with alternative methods.

The outline of the paper is as follows. In Section 2 we introduce related work. We present an overview of the proposed method in Section 3 and a detailed description in Section 4. The experimental results are outlined in Section 5. Finally, Section 6 is a conclusion.

2. Related Work

The term *incremental* usually refers to the ability of a learning process to learn sequentially, thus being able to handle new data and new classes without the need to retrain the whole system (Polikar et al., 2001). As an example, the "learn++" algorithm introduced in (Polikar et al., 2001) accommodates new classes using weak one-vs-all classifiers. This approach conveniently manages the insertion, deletion and recurrence of classes over learning data (Sun et al., 2016). However, this method requires to continuously train new classifiers in order to accommodate for new data, resulting in a potentially large computational intensiveness and memory usage.

Another approach was proposed to deal with a large amount of data (Syed et al., 1999; Poggio & Cauwenberghs, 2001; Lomonaco & Maltoni, 2016). The idea is to replace batches in classical learning methods with a process based on Support Vector Machines (SVM), in which learning is performed using only one subset at a time, independently of the others. As a consequence, it is possible to limit memory usage. However, training the SVMs can be computationally expensive.

In (Zhou & Chen, 2002) incremental learning refers to three distinct problems: example-incremental learning (Syed et al., 1999; Poggio & Cauwenberghs, 2001; Lomonaco & Maltoni, 2016), class-incremental learning (Polikar et al., 2001; Sun et al., 2016), and attribute-incremental learning. In (Zheng et al., 2013), the authors propose a SVM inspired method to handle both the first two concepts defined above. However, it requires the training of novel SVMs using new examples and old SVMs. In addition, SVMs suffer from *catastrophic forgetting*, which is the loss of previously learned information (Kasabov, 2013; French, 1999). To address this problem a combination between SVMs and learn++ method called "SVM-learn++" (Erdem et al., 2005) was proposed, showing a promising improvement on biological datasets (Molina et al., 2014). However, this method still needs to retrain a new SVM each time new data is processed, and some knowledge is forgotten while new information is being learned.

The method we propose in this paper is quite different as it combines incremental aspects with one-shot learning using binary associative memories. Consequently, there is no need to retrain the system with old data, nor to perform computationally intensive processing with a new one. In addition, learning new data does not damage previously learned information, and only a few examples are required for learning, resulting in substantial savings in memory during the learning process.

3. Overview of the Proposed Method

The proposed method is built using three main techniques: 1) a pre-trained deep CNN to perform feature extraction, 2) product quantizing techniques to embed data in a finite alphabet and 3) binary associative memories to store and classify data as a proxy to a nearest neighbour search. In the next paragraphs, we introduce each of these techniques.

The first key idea is to use the internal layers of a pre-trained deep CNN (Krizhevsky et al., 2012) as a generic feature extractor, on which subsequent learning is performed. This process has become increasingly popular in the past few years and is often referred to as "Transfer Learning" (Oquab et al., 2014). The aim is to transfer acquired knowledge on a dataset to another related problem (Pan & Yang, 2010).

The next step we perform consists in embedding a feature vector in a finite alphabet. This step is crucial as it allows to

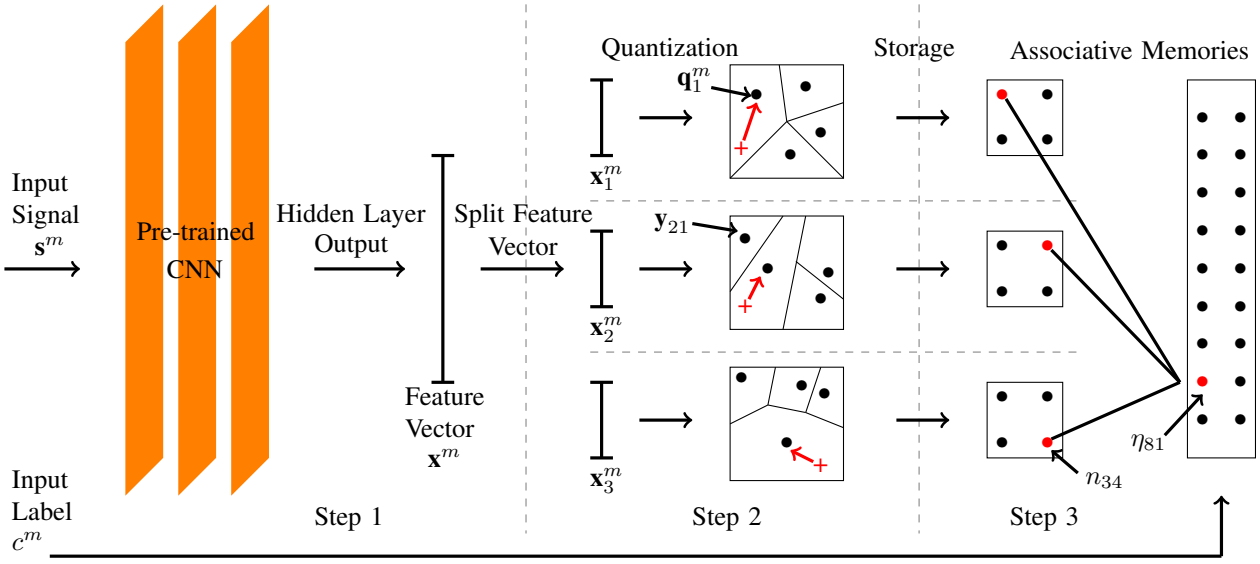

Figure 1. Overview of the proposed method, comprising three main steps. Given a set of samples, we first use a pre-trained CNN for feature extraction (Section 4.1). Subsequently, we use a PQ technique to quantize the feature vectors (Section 4.2). Finaly, we use a binary associative memory to store and classify the quantized data (Section 4.3)

map outputs of Step 1 to the inputs of Step 3. There is a lot of literature dedicated to this problem, including methods relying on Product Quantization (PQ) (Jegou et al., 2011). PQ is of particular interest since it performs quantization in a product of sub-spaces, which is particularly adapted to the type of associative memories that we use in Step 3. Because we aim at providing computationally light solutions, we rather use product random sampling in this work. Basically, we split features into multiple subvectors, which are quantized independently from each others using random selection of anchor subvectors in the dataset.

After Step 2, the feature vectors are transformed into words of fixed length over a finite alphabet of anchor subvectors. These subvectors are associated with a corresponding output (typically an indicator vector of the class, c.f. next section for more details) through a binary sparse associative memory (Gripon & Berrou, 2011). The key idea here is that feature vectors are mapped onto cliques in bipartite graphs. More precisely, a collection of feature vectors is associated with a union of cliques. This process is known to allow the storage of a large number of data points while providing very good accuracy in retrieving one of them from noisy probes.

Our method is a combination of a deep pre-trained CNN that does not change during the training process, and associative memories that are modified after each newly observed example or class. This combination allows to handle both example and class incremental approaches, with no other prior about the learning dataset, using only few learning examples and without having to retrain the model or damage the previously obtained knowledge (Goodfellow et al., 2013).

4. Detailed Description of the Proposed Method

In this section we describe the proposed method in details. First, we introduce how to obtain the feature vectors using pre-trained CNNs. Second we discuss the quantization part using product random sampling. Then we present how to map example signals to their corresponding class using binary associative memories. In a fourth subsection, we explain how to process unlabelled signals to perform classification.

4.1. Pre-Trained Convolutional Neural Networks

To obtain features of an input signal, our method relies on using CNNs that are pre-trained on a large number of examples. The idea is to use a broad dataset (e.g. ImageNet) distinct from the actual one we want to handle. Consequently, using the pre-trained inner layers the CNN acts as a generic feature extractor (Oquab et al., 2014; Hong et al., 2015; Pan & Yang, 2010). In this paper, our experiments focus on the use of Inception V3 (Szegedy et al., 2015) and SqueezeNet (Iandola et al., 2016), that are trained on 1000 classes from ImageNet[1]. Inner layers of a deep CNN offer a good generic description of an input image, even when it does not belong to the learning domain (Oquab et al.,

[1]ImageNet challenge url: http://image-net.org/challenges/LSVRC/2012/index

2014).

Using "Transfer Learning" ideas, we are not interested in this work in the network's architecture details, as we simply use the appropriate layers to extract features from a given input. Here, we take outputs of the last pooling layer[2].

In the remaining of this paper, we denote by $\mathbf{s}^m$ the m-th input training signal and by $\mathbf{x}^m$ its corresponding feature vector, where $1 \leq m \leq M$ and M is the total number of training signals.

4.2. Product Random Sampling

Once we obtain the feature vector set $X = \{\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^M\}$ associated with the input sample set $S = \{\mathbf{s}^1, \mathbf{s}^2, \ldots, \mathbf{s}^M\}$, the next step consists in quantizing them. For this purpose, we use product random sampling, as our experiments showed it provides very good performance together with limited computational intensiveness.

In details, we split all feature vectors $\mathbf{x}^m$ into P disjoint subparts of equal size denoted $\left(\mathbf{x}_p^m\right)_{1 \leq p \leq P}$, and quantize each induced subspace independently from each other. In each of the P subspaces, quantized states are obtained by randomly sampling K subvectors. Consider the p-th subspace, we denote $Y_p = [\mathbf{y}_{p1}, ..., \mathbf{y}_{pK}]$ the obtained quantized states. Hence, each $\mathbf{y}_{pk}$ is such that $\exists \mathbf{x}^m \in X, \mathbf{x}_p^m = \mathbf{y}_{pk}$. In the remaining of this work, we refer to $\left(\mathbf{y}_{pk}\right)_{1 \leq p \leq P, 1 \leq k \leq K}$ as anchor subvectors.

Then, each subvector $\mathbf{x}_p^m$ is quantized by choosing the closest anchor subvector in its corresponding subspace, as described in Equation (1). We use the Euclidean distance in our experiments. Note that each quantization is independent from each other, such that the process can be performed concurrently.

$$\begin{cases} k^\star(m,p) &= \arg\min_k \|\mathbf{x}_p^m - \mathbf{y}_{pk}\|_2 \\ \mathbf{q}_p^m &= y_{pk^\star(m,p)} \end{cases} \quad (1)$$

Using the same methodology, it is possible to process unlabelled input signals and quantize them accordingly (c.f. Figure 1, Step 2).

After this step, the feature vector set X is transformed into the set $Q = \{\mathbf{q}^1, \mathbf{q}^2, \ldots, \mathbf{q}^M\}$, where $\mathbf{q}^m = \left(\mathbf{q}_p^m\right)_{1 \leq p \leq P}$. This process can be performed independently for each example or class. For more details, see section 5.1.

[2] https://github.com/Hvass-Labs/TensorFlow-Tutorials/blob/master/08_Transfer_Learning.ipynb

4.3. Binary Associative Memories

The last part of the proposed method consists in associating quantized feature vectors with their corresponding classes. For this purpose, we use binary associative memories mainly for three reasons: a) their ability to efficiently store large amounts of data, b) their ability to perform approximate nearest neighbour search with limited computational complexity and memory usage, and c) their ability to handle new examples or classes in one-shot and without need for retraining.

The outputs of product random sampling are words of fixed length P over a finite alphabet (containing K distinct symbols). The idea here is to use a neural network comprising two layers, the input one that is organized in P clusters containing K neurons each, and the output one containing RC neurons, where R is the number of neurons for each class and C is the number of classes in our dataset. Consider the neurons in the input layer to be indexed by two variables $p, 1 \leq p \leq P$ and $k, 1 \leq k \leq K$, where p denotes the index of the cluster and k the index of the neuron inside the cluster, and the neurons in the output layer to be indexed by two variables $c, 1 \leq c \leq C$ and $r, 1 \leq r \leq R$. We denote neurons in the input layer n_{pk} and neurons in the output layer η_{cr}.

When processing a training input signal $\mathbf{s}^m$, a number of neurons are activated in the network. Namely, we activate the neurons in the input layer whose indexes p, k are corresponding to the indexes of the activated anchor subvectors $\mathbf{q}_p^m$ obtained in Step 2. We activate in the output layer a neuron whose first index c is the index of the class c^m the training vector is part of, the second index r being drawn uniformly at random. Then we add connections (since the network is binary, there is no connection weight but only presence or absence of connections) between η_{cr} and all n_{pk}, printing a bipartite clique into the network (note that if a connection already existed, it is left unchanged) (c.f. Algorithm 1 and Figure 1, Step 3).

Algorithm 1 Store a learning input signal using associative memories

input Subvectors $\mathbf{q}_p^m$ obtained in Step 2, $1 \leq p \leq P$, the class c^m of the input learning signal.

Choose r uniformly at random in $\{1, 2, \ldots, R\}$
for $p = 1$ **to** P **do**
  Connect $n_{pk^\star(m,p)}$ with η_{cr}
end for

To handle a new input sample set S, containing new examples or classes, we add new neurons to each cluster of the input layer corresponding to the newly obtained anchor subvectors in step 2 (note that the number of the new neurons depends of the number of new examples, and it is

defined in section 5), and R neurons in the cluster of the output layer corresponding to the new class. Then each input signal $\mathbf{s}^m$ of the new input sample set S is processed through the three steps. This allows to handle both example and class incremental approaches.

In this approach, we use associative memories to easily add new neurons to the different clusters for new learning examples or classes and do not use previous data or modify existing connections, thus the incremental learning is performed without retraining the model or damaging the previously obtained knowledge.

4.4. Classify Unlabelled Data

To classify an unlabelled input signal $\tilde{\mathbf{s}}^m$, we follow sequentially three steps. First, we use the pre-trained CNN to obtain the corresponding feature vector $\tilde{\mathbf{x}}^m$. We then split $\tilde{\mathbf{x}}^m$ into P parts and obtain the P associated subvectors $\tilde{\mathbf{x}}_p^m$, $1 \leq p \leq P$. Finally, we activate neurons in the input layer of the binary associative memory corresponding to the indexes of the obtained $\mathbf{q}_p^m$. The decision of the overall process is obtained by looking at the first index of the neurons in the output layer that are connected to the most numerous number of activated neurons in the input layer (c.f. Algorithm 2).

Algorithm 2 Classify an unlabelled input signal $\tilde{\mathbf{s}}^m$

input Input signal $\tilde{\mathbf{s}}^m$
Require: Set of subvectors $\mathbf{y}_{pk}$, $1 \leq p \leq P, 1 \leq k \leq K$, the connections in the binary associative memory
output An estimate of the class associated with the input signal $\tilde{\mathbf{s}}^m$

Use the pre-trained CNN to obtain the feature vector $\tilde{\mathbf{x}}^m$
Quantize $\tilde{\mathbf{x}}^m$ into P anchor subvectors $\mathbf{q}_p^m$
Activate neurons in the input layer of the binary associative memory corresponding to the retained indexes
for each neuron η_{cr} in the output layer **do**
  Count how many activated neurons in the input layer it is connected to
end for
Look at the neurons in the output layer that maximise this count
Choose one of them uniformly at random
Its first index is the estimated class

5. Experiments

In this section we first describe the implementation details and strategies followed to quantize feature vectors $\mathbf{X}$. The accuracy of each strategy is then presented and discussed. We also investigate the behaviour of the accuracy, when changing some parameters, as the number of parts P to split feature vectors, as the number of neurons $\tilde{k}$ for each class in the input layer, and as the number of neurons R for each class in the output layer.

5.1. Implementation Details and Strategies Followed

For our method, we adopt the Inception V3 CNN model. It takes as input an image which is resized to $299 \cdot 299$ pixels and outputs a 2048 dimensional vector from the layer before the first fully-connected one (Szegedy et al., 2015). The output vector represents the feature vector of the input image. To get the subvectors $\mathbf{y}_{pk}$ introduced in the previous section, we split the feature vectors $\mathbf{X}$ into P parts and we choose randomly $\tilde{k}$ subvectors from each class for each part (*i.e.* $K = C\tilde{k}$, C is the number of classes in our dataset and K is the total number of neurons in each cluster of the input layer).

We compare three strategies to emphasize the interest of the proposed method. They are described in the following paragraphs.

5.1.1. The "Independent Incremental" Approach (I-I)

In this approach, training is not necessary: from each class we sample a portion of the example vectors and directly associate them to the corresponding output neurons using the associative memory. New data does not impact previously acquired knowledge, avoiding catastrophic forgetting.

In the case where $R = 1$, note that the associative memory is equivalent to counting how many quantized subvectors belong to each class and selecting the maximal one.

5.1.2. The "Non-Independent Incremental" Approach (N-I)

In this approach, learning new elements can affect previously learned data. More precisely, each new input vector is quantized using all the already acquired anchor subvectors, independently of the class of the example that added them. The learning procedure is therefore computationally more complex than for the I-I method.

5.1.3. The "Non-Independent Offline" Approach (N-O)

In this approach, the selection of anchor vectors is performed prior to any storing in the associative memory, so that the latter becomes independent of the order on which examples and classes are presented to the network.

5.2. Evaluation

We evaluate the proposed methods using three distinct datasets. The two first datasets (called in this paper Im-

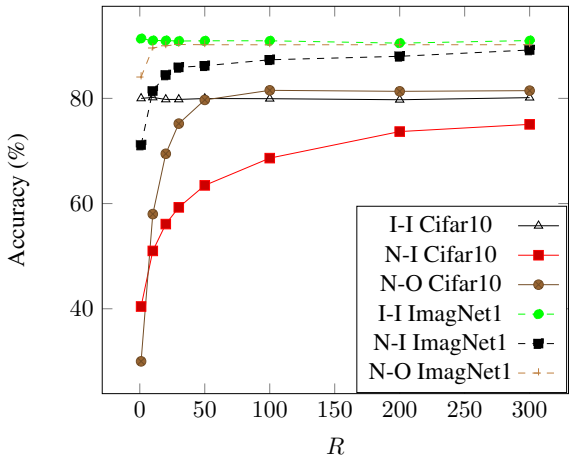

Figure 2. Evolution of the accuracy of the I-I approch ($P = 16$ and $\tilde{k} = 20$) as a function of the number of neurons R in the output layer for each class (ImageNet1 and Cifar10).

ageNet1 and ImageNet2) use 10 different classes of imageNet which were not used to train the CNN model. We use Cifar10 as the third dataset. Throughout the experimental part, the given accuracy is the average one over 10 realisations of each experiment.

5.2.1. Comparing the approches

Our first experiments consist of stressing the effect of the number of neurons per class R in the output layer of the associative memory on the accuracy of the three proposed approaches, for fixed values of the quantization parameters P and $\tilde{k}$. Figure 2 depicts the evolution of the accuracy of the methods as a function of the number of R. Expectedly, we observe that performance increases as a function of the number of neurons in the output layer. More interesting is the behaviour of the I-I method, which seems almost independent on R, while staying very close to the N-O method even when the latter is using a large value of R.

Additionally, the I-I approach shows better accuracy than the N-I one even when varying the parameters P and $\tilde{k}$, as shown in Table 1. With this in mind, we focus on the I-I approach with $R = 1$ in the following experiments.

Next, we consider two incremental protocols: class-incremental and example-incremental.

5.2.2. Class-Incremental Protocol

We first evaluate the effect of adding a new class on the accuracy. To do so, we start with an empty quantizer and an empty associative memory, and we add classes one by one. In order to avoid arbitrary decisions in the order in which classes are presented to the method, we perform experiments with random shuffles (200 times) and plot the

	I-I accuracy (%)	N-I accuracy (%)
$P = 16, \tilde{k} = 20$	**80.14**	75.06
$P = 16, \tilde{k} = 10$	**77.72**	65.52
$P = 64, \tilde{k} = 15$	**80.09**	75.61
$P = 32, \tilde{k} = 5$	**76.36**	59.38

Table 1. Comparing the accuracy of I-I approach with N-I approach with $R = 300$ and various cluster parameters (P and $\tilde{k}$) (Cifar10).

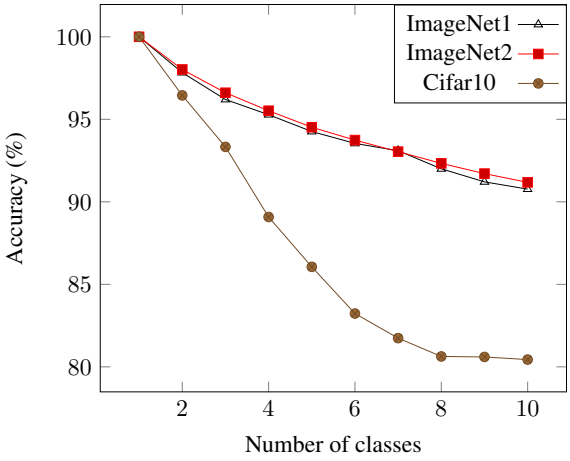

Figure 3. Evolution of the accuracy of the proposed method as a function of number of classes for $P = 16$, $\tilde{k} = 20$ and $R = 1$ (ImageNet1, ImageNet2 and Cifar10).

average. We consider the following parameters: $P = 16$, $\tilde{k} = 20$ and $R = 1$. The accuracy a_c when introducing a novel class C_c is computed from scratch by adding new test examples to old one, according to Equation (2), where z_c represents the number of well classified test examples of all classes (for C_1 to C_c), m_c is the number of test examples of the class C_c and M_c is the total number of all test examples from classes C_1 to C_c.

$$\begin{cases} M_c = M_{c-1} + m_c \\ a_c = \frac{z_c}{M_c} \\ \text{with } M_0 = 0 \end{cases} \tag{2}$$

Each time a novel class is to be learned, we randomly sample $\tilde{k}$ subvectors for each of the P subspaces. We jointly add $\tilde{k}$ corresponding neurons in each cluster of the input layer. Finally, we add a new neuron corresponding to the newly added class in the output layer.

The obtained results are depicted in Figure 3. Of course the effect of adding new classes is more significant for a few number of classes, as it considerably strengthens the

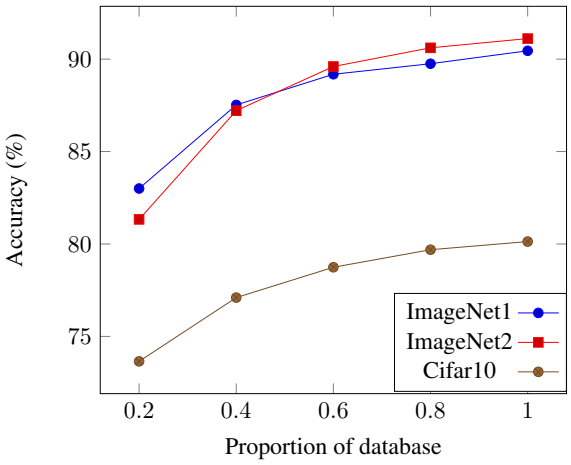

Figure 4. Evolution of the accuracy as a function of the number of learning examples ($P = 16$ and $\tilde{k} = 20$) (ImageNet1, ImageNet2 and Cifar10).

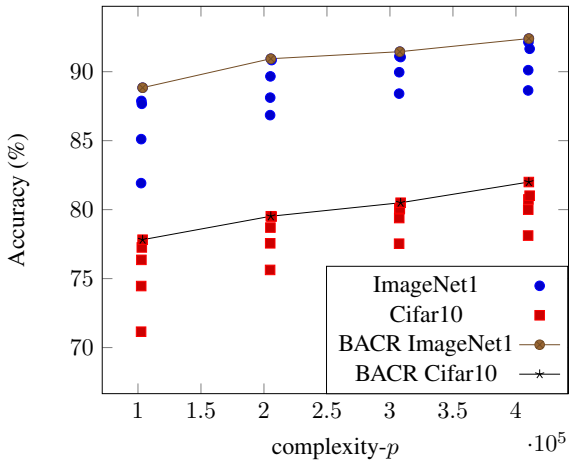

Figure 5. Evolution of the accuracy of I-I approach as a function of complexity-p (ImageNet1 and Cifar10) when varying K and P ($T = 2048$, $R = 1$ and $C = 10$).

problem. The accuracy obtained after training all 10 classes approaches the one corresponding to the N-O approach for each dataset.

5.2.3. Example-Incremental Protocol

Next we evaluate the effect of adding new learning examples, without introducing new classes, on the accuracy of the I-I approach. To do so, we split the learning database into 5 parts and we learn one part at a time. The same testing dataset is used to measure accuracy at each step. For each part to be learned, we proportionally sample subvectors for each of the P subspaces and add the corresponding neurons in each cluster of the input layer. In Figure 4, for the three datasets, our method handles the incremental learning improving its accuracy and reaching the same final results as in the previous tests.

5.3. Complexity and memory usage

A key factor in proposing interesting solutions when targeting embedded architectures are complexity and memory usage. We refer to complexity as the number of arithmetical operations needed to learn the database (complexity-ℓ) or to classify an unlabelled input (complexity-p). The complexity-ℓ of the proposed method is negligible because we do not have to train the model with the whole dataset (c.f. subsection 5.4), while complexity-p is defined as $TK + PRC$ where T is the feature vector size ($T = 2048$ in our case due to the use of inception v3).

Figure 5 represents the accuracy as a function of the complexity-p when varying K and P (the other parameters are fixed to $T = 2048$, $R = 1$ and $C = 10$) and shows the best accuracy-complexity ratio (BACR). For a given K, BACR is obtained as the maximum in accuracy for similar values of complexity-p. We use the set of parameters reaching the BACR to compare our method with a nearest neighbour search.

Memory usage is defined by the size of clusters on input or output layers, and of the binary matrix which stores the connections between neurons. Memory usage sums up to $KTf + KPRC$ where f is the number of bits used per vector coordinates (we use $f = 32$ bits). Note that the size of the pre-trained CNN is not considered and the memory usage for learning and classifying are considered similar. We estimate accuracy, complexity and memory usage to compare the proposed method with a λ nearest neighbour (λ-NN) approach. The complexity-ℓ of the λ-NN search is also negligible, the complexity-p is defined by MT and memory usage is MTf. Results are shown in Table 2.

	Proposed Method	Other techniques	
		1-NN	5-NN
Accuracy(%)	82	85	**87**
complexity-ℓ	**negligible**	**negligible**	**negligible**
complexity-p	$\mathbf{4.1 \cdot 10^5}$	10^8	10^8
Memory usage-ℓ	$\mathbf{1.3 \cdot 10^7}$	$3.3 \cdot 10^9$	$3.3 \cdot 10^9$
Memory usage-p	$\mathbf{1.3 \cdot 10^7}$	$3.3 \cdot 10^9$	$3.3 \cdot 10^9$

Table 2. Accuracy, complexity and memory usage of I-I approach ($P = 64$, $K = 200$ and $R = 1$) compared to λ-NN search for Cifar10.

We observe a loss in accuracy using our method, from 87% for nearest neighbour to 82%. On the other hand we obtain

important gains in complexity and memory usage. One of the reason is that nearest neighbour search requires storing all training examples, which does not meet the criteria that define incremental learning algorithms (Polikar et al., 2001).

Instead of using λ-NN search, we can accelerate it using PQ (Product Quantification). Namely, we split all feature vectors $\mathbf{x}^m$ into P' of equal size denoted $\left(\mathbf{x}_p^m\right)_{1\leq p\leq P'}$, and for each subspace, we perform K-means on the feature vector set $X_p = \{\mathbf{x}_p^1, ..., \mathbf{x}_p^M\}$ to extract K' centroids. When using K-means, we lowerbound the complexity-ℓ by taking into account only the MTK' operations needed to quantize the learning dataset before storing it, with no consideration for the price of performing K-means. We motivate this choice as one could instead use product random sampling as described in our method. The complexity-p is $TK' + MP'$ and the memory usage is $TfK' + MP'log_2(K')$. Table 3 shows the obtained results.

	Proposed method	Other techniques	
		1-NN	5-NN
Accuracy(%)	82	82.6(82)	**86.07**(83)
complexity-ℓ	**negligible**	$\geq 2 \cdot 10^{10}$	$\geq 2 \cdot 10^{10}$
complexity-p	$\mathbf{4.1 \cdot 10^5}$	$3.2 \cdot 10^6$	$3.2 \cdot 10^6$
Memory usage-ℓ	$\mathbf{1.3 \cdot 10^7}$	$3.7 \cdot 10^7$	$3.7 \cdot 10^7$
Memory usage-p	$\mathbf{1.3 \cdot 10^7}$	$3.7 \cdot 10^7$	$3.7 \cdot 10^7$

Table 3. Accuracy, complexity and memory usage ratio of I-I approach ($P = 64, K = 200$ and $R = 1$) compared to λ-NN search using PQ ($K' = 200$, $P' = 64$) for Cifar10. Numbers between brackets accounts for product random sampling instead of PQ.

NN search using PQ not only gives a good accuracy (86.07% compared with 82% of the I-I approach), but also reduces the complexity-p and memory usage by a factor of 100. However it requires a large computational power for learning process and stores a quantized version of the whole dataset, again not complying with the incremental learning algorithms criteria (Polikar et al., 2001). In addition both complexity and memory usage depends of number of learning examples M and could quickly become problematic.

Note that the proposed method uses product random sampling because it offers almost the same accuracy as using K-means instead. Moreover to use K-means it is required to store the whole database and perform expensive operations.

Finally, to assess the robustness of the proposed method with regards to the chosen CNN feature extractor, we perform similar experiments using the SqueezeNet (Iandola et al., 2016) architecture. This network makes even more sense with regards to embedded platforms given its very small memory usage. Table 4 shows the obtained results that comfort the ones obtained using Inception V3.

	Proposed method	Other techniques	
		1-NN	5-NN
Accuracy(%)	84	88	**89**
complexity-ℓ	**negligible**	**negligible**	**negligible**
complexity-p	$\mathbf{2 \cdot 10^5}$	$8.4 \cdot 10^5$	$8.4 \cdot 10^5$
Memory usage-ℓ	$\mathbf{6.5 \cdot 10^6}$	$3.2 \cdot 10^8$	$3.2 \cdot 10^8$
Memory usage-p	$\mathbf{6.5 \cdot 10^6}$	$3.2 \cdot 10^8$	$3.2 \cdot 10^8$

Table 4. Accuracy, complexity and memory usage ratio of I-I approach ($P = 64, K = 200$ and $R = 1$) using SqueezeNet compared to λ-NN search for ImageNet2.

5.4. Discussion

The proposed method achieves incremental learning (Figures 3 and 4) with substantial reduction of computational complexity and memory usage, compared to nearest neighbour search, without compromising classification accuracy (Tables 2 and 3). Note that we preferred the I-I approach in our tests, since it obtained better accuracy than N-I. Since we also use product random sampling to feed the associative memories and require only one neuron per class in the output layer, we obtain that the neuron n_{pk} corresponding to y_{pk} is only connected to the neuron η_c, where $\exists \mathbf{x}^m \in X, \mathbf{x}_p^m = \mathbf{y}_{pk}$ and x^m belongs to the class C_c. As a consequence, knowing the connections of neurons obtained from random sampling with the neurons of the output layer, we need only few examples to train our model. Thus, the proposed method needs only a portion of the learning dataset to train, resulting in even lighter computational intensiveness and memory usage.

6. Conclusion

We introduced a novel incremental algorithm based on pre-trained CNNs and associative memories to classify images, the first ones using connection weights to process images, the second one using existence of connections to store them efficiently. This combination of methods allows to learn and process data using very few examples, memory usage and computational intensiveness. The obtained accuracy is close to other state-of-the-art methods based on transfer learning. As a consequence, we believe this method is promising for embedded devices and consider proposing thrifty hardware implementations of it as future work.

References

Cadieu, Charles F, Hong, Ha, Yamins, Daniel LK, Pinto, Nicolas, Ardila, Diego, Solomon, Ethan A, Majaj, Najib J, and DiCarlo, James J. Deep neural networks rival the representation of primate it cortex for core visual object recognition. *PLoS Comput Biol*, 10(12):e1003963, 2014.

Deng, Jia, Berg, Alex, Satheesh, Sanjeev, Su, Hao, Khosla, Aditya, and Li, Fei-Fei. Large scale visual recognition challenge. *www. image-net. org/challenges/LSVRC/2012*, 1, 2012.

Erdem, Zeki, Polikar, Robi, Gurgen, Fikret, and Yumusak, Nejat. Ensemble of svms for incremental learning. In *International Workshop on Multiple Classifier Systems*, pp. 246–256. Springer, 2005.

French, Robert M. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

Girshick, Ross, Donahue, Jeff, Darrell, Trevor, and Malik, Jitendra. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587, 2014.

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

Gripon, Vincent and Berrou, Claude. Sparse neural networks with large learning diversity. *IEEE transactions on neural networks*, 22(7):1087–1096, 2011.

Hong, Seunghoon, You, Tackgeun, Kwak, Suha, and Han, Bohyung. Online tracking by learning discriminative saliency map with convolutional neural network. *CoRR*, abs/1502.06796, 2015. URL http://arxiv.org/abs/1502.06796.

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

Jegou, Herve, Douze, Matthijs, and Schmid, Cordelia. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2011.

Kasabov, Nikola. *Evolving connectionist systems: Methods and applications in bioinformatics, brain study and intelligent machines*. Springer Science & Business Media, 2013.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Lin, Darryl D, Talathi, Sachin S, and Annapureddy, V Sreekanth. Fixed point quantization of deep convolutional networks. In *International Conference on Machine Learning (ICML)*, June 2016.

Lomonaco, Vincenzo and Maltoni, Davide. Comparing incremental learning strategies for convolutional neural networks. In *IAPR Workshop on Artificial Neural Networks in Pattern Recognition*, pp. 175–184. Springer, 2016.

Molina, José Fernando García, Zheng, Lei, Sertdemir, Metin, Dinter, Dietmar J, Schönberg, Stefan, and Rädle, Matthias. Incremental learning with svm for multimodal classification of prostatic adenocarcinoma. *PloS one*, 9 (4):e93600, 2014.

Oquab, Maxime, Bottou, Leon, Laptev, Ivan, and Sivic, Josef. Learning and transferring mid-level image representations using convolutional neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

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

Poggio, Tomaso and Cauwenberghs, Gert. Incremental and decremental support vector machine learning. *Advances in neural information processing systems*, 13:409, 2001.

Polikar, Robi, Udpa, Lalita, Udpa, Satish S, and Honavar, Vasant. Learn++: an incremental learning algorithm for multilayer perceptron networks. In *Acoustics, Speech, and Signal Processing. ICASSP'00. Proceedings.IEEE International Conference on*, volume 6, pp. 3414–3417. IEEE, 2000.

Polikar, Robi, Upda, Lalita, Upda, Satish S, and Honavar, Vasant. Learn++: An incremental learning algorithm for supervised neural networks. *IEEE transactions on systems, man, and cybernetics, part C (applications and reviews)*, 31(4):497–508, 2001.

Qiu, Jiantao, Wang, Jie, Yao, Song, Guo, Kaiyuan, Li, Boxun, Zhou, Erjin, Yu, Jincheng, Tang, Tianqi, Xu, Ningyi, Song, Sen, et al. Going deeper with embedded fpga platform for convolutional neural network. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 26–35. ACM, 2016.

Sun, Yu, Tang, Ke, Minku, Leandro L, Wang, Shuo, and Yao, Xin. Online ensemble learning of data streams with gradually evolved classes. *IEEE Transactions on Knowledge and Data Engineering*, 28(6):1532–1545, 2016.

Syed, Nadeem Ahmed, Huan, Syed, Kah, Liu, and Sung, Kay. Incremental learning with support vector machines. 1999.

Szegedy, Christian, Vanhoucke, Vincent, Ioffe, Sergey, Shlens, Jonathon, and Wojna, Zbigniew. Rethinking the inception architecture for computer vision. *arXiv preprint arXiv:1512.00567*, 2015.

Zheng, Jun, Shen, Furao, Fan, Hongjun, and Zhao, Jinxi. An online incremental learning support vector machine for large-scale data. *Neural Computing and Applications*, 22(5):1023–1035, 2013.

Zhou, Zhi-Hua and Chen, Zhao-Qian. Hybrid decision tree. *Knowledge-based systems*, 15(8):515–528, 2002.