OvA-INN: Continual Learning with Invertible Neural Networks

Anonymous authors
Paper under double-blind review

Abstract

In the field of Continual Learning, the objective is to learn several tasks one after the other without access to the data from previous tasks. Several solutions have been proposed to tackle this problem but they usually assume that the user knows which of the tasks to perform at test time on a particular sample, or rely on small samples from previous data and most of them suffer of a substantial drop in accuracy when updated with batches of only one class at a time. In this article, we propose a new method, OvA-INN, which is able to learn one class at a time and without storing any of the previous data. To achieve this, for each class, we train a specific Invertible Neural Network to output the zero vector for its class. At test time, we can predict the class of a sample by identifying which network outputs the vector with the smallest norm. With this method, we show that we can take advantage of pretrained models by stacking an invertible network on top of a features extractor. This way, we are able to outperform state-of-the-art approaches that rely on features learning for the Continual Learning of MNIST and CIFAR-100 datasets. In our experiments, we are reaching 72% accuracy on CIFAR-100 after training our model one class at a time.

1 Introduction

A typical Deep Learning workflow consists in gathering data, training a model on this data and finally deploying the model in the real world (Goodfellow et al., 2016). If one would need to update the model with new data, it is required to merge the old and new data and process a training from scratch on this new dataset. Nevertheless, there are circumstances where this method may not apply. For example, it may not be possible to store the old data because of privacy issues (health records, sensible data) or memory limitations (embedded systems, very large datasets). In order to address those limitations, recent works propose a variety of approaches in a setting called *Continual Learning* (Parisi et al., 2018).

In Continual Learning, we aim to learn the parameters w of a model on a sequence of datasets $\mathcal{D}_i = \{(x_i^j, y_i^j)\}_{j=1}^{n_i}$ with the inputs $x_i^j \in \mathcal{X}$ and the labels $y_i^j \in \mathcal{Y}^i$, to predict $p(y^*|w, x^*)$ for an unseen pair (x^*, y^*). The training has to be done on each dataset, one after the other, without the possibility to reuse previous datasets. The performance of a Continual Learning algorithm can then be measured with two protocols : *multi-head* or *single-head*. In the multi-head scenario, the task identifier i is known at test time. For evaluating performances on task i, the set of all possible labels $\mathcal{Y}$ is then $\mathcal{Y}^i$. Whilst in the single-head scenario, the task identifier is unknown, in that case we have $\mathcal{Y} = \cup_{i=1}^{N} \mathcal{Y}^i$ with N the number of tasks learned so far. For example, let us say that the goal is to learn MNIST sequentially with two batches: using only the data from the first five classes and then only the data from the remaining five other classes. In multi-head learning, one asks at test time to be able to recognize samples of 0-4 among the classes 0-4 and samples of 5-9 among classes 5-9. On the other hand, in single-head learning, one can not assume from which batch a sample is coming from, hence the need to be able to recognize any samples of 0-9 among classes 0-9. Although the former one has received the most attention from researchers, the last one fits better to the desiderata of a Continual Learning system as expressed in Farquhar & Gal (2018) and (van de Ven & Tolias, 2019). The single-head scenario is also notoriously harder than its multi-head counterpart (Chaudhry et al., 2018) and is the focus of the present work.

Updating the parameters with data from a new dataset exposes the model to drastically deteriorate its performance on previous data, a phenomenon known as *catastrophic forgetting* (McCloskey & Cohen, 1989). To alleviate this problem, researchers have proposed a variety of approaches such as storing a few samples from previous datasets (Rebuffi et al., 2017), adding *distillation* regularization (Li & Hoiem, 2018), updating the parameters according to their usefulness on previous datasets (Kirkpatrick et al., 2017), using a generative model to produce samples from previous datasets (Kemker & Kanan, 2017). Despite those efforts toward a more realistic setting of Continual Learning, one can notice that, most of the time, results are proposed in the case of a sequence of batches of multiple classes. This scenario often ends up with better accuracy (because the learning procedure highly benefits of the diversity of classes to find the best tuning of parameters) but it does not illustrate the behavior of those methods in the worst case scenario. In fact, Continual Learning algorithms should be robust in the size of the batch of classes.

In this work, we propose to implement a method that is especially designed to handle the case where each task consists of only one class. It will therefore be evaluated in the single-head scenario. Our approach, named One-versus-All Invertible Neural Networks (OvA-INN), is based on a recent network architecture, the Invertible Network (Gomez et al., 2017). We use it in a One-versus-All strategy: each network is trained to make a prediction of a class and the most confident one on a sample is used to identify the class of the sample. Our idea is to train for each subsequent class a specific Invertible Network to output a vector with a small norm on samples from that class. We rely on the properties of Invertible Networks to ensure that it will be less likely to produce a vector of small norm on samples from the class it has not been trained on. By feeding a non-invertible feedforward network with data from only one class, the network would trivially learn to output a zero vector to get the smallest output norm. But, as Invertible Networks are designed to compute a bijection between their input and their output, this behavior is excluded and the network learns to minimize the output norm on the data it is given only. This procedure allows each network to learn the specific features of each class for which it has been trained on. The inference consists then in running each network on a sample and identify the one that produced the vector with the smallest norm. In contrast to most other methods, the training phase of each class can be independently executed from one another.

The contributions of our work are : *(i)* a new approach for Continual Learning with one class per batch; *(ii)* a neural architecture based on Invertible Networks that does not require to store any of the previous data; *(iii)* state-of-the-art results on several tasks of Continual Learning for Computer Vision (CIFAR-100, MNIST) in this setting.

We start by reviewing the closest methods to our approach in Section 2, then explain our method in Section 3 and analyse its performances in Section 4.

2 Related work

Generative models Inspired by biological mechanisms such as the hippocampal system that rapidly encodes recent experiences and the memory of the neocortex that is consolidated during sleep phases, a natural approach is to produce samples of previous data that can be added to the new data to learn a new task. FearNet (Kemker & Kanan, 2017) relies on an architecture based on an autoencoder, whereas Deep Generative Replay (Shin et al., 2017) and Parameter Generation and Model Adaptation (Hu et al., 2018) propose to use a generative adversarial network. Those methods present good results but require complex models to be able to generate reliable data. Furthermore, it is difficult to assess the relevance of the generated data to conduct subsequent training iterations.

Coreset-based models These approaches alleviate the constraint on the availability of data by allowing the storage of a few samples from previous data (which are called *coreset*). iCaRL (Rebuffi et al., 2017) and End-to-end IL (Castro et al., 2018) store 2000 samples from previous batches and rely on respectively a distillation loss and a mixture of cross-entropy and distillation loss to alleviate forgetting. The authors of SupportNet (Li et al., 2018) have also proposed a strategy to select relevant samples for the coreset. Gradient Episodic Memory (Lopez-Paz et al., 2017) ensures that gradients computed on new tasks do not interfere with the loss of previous tasks. Those approaches give the best results for single-head learning. But, similarly to generated data, it is not clear which data may

be useful to conduct further training iterations. In this paper, we are challenging the need of the coreset for single-head learning.

Distance-based models These methods propose to embed the data in a space which can be used to identify the class of a sample by computing a distance between the embedding of the sample and a reference for each class. Among the most popular, we can cite Matching Networks (Vinyals et al., 2016) and Prototypical Networks (Snell et al., 2017), but these methods have been mostly applied to few-shot learning scenarios rather than continual.

Regularization-based approaches These approaches present an attempt to mitigate the effect of catastrophic forgetting by imposing some constraints on the loss function when training subsequent classes. Elastic Weight Consolidation (Kirkpatrick et al., 2017), Synaptic Intelligence (Zenke et al., 2017) and Memory Aware Synapses (Aljundi et al., 2018) all seek to prevent the update of weights that were the most useful to discriminate between previous classes. Hence, it is possible to constrain the learning of a new task in such a way that the most relevant weights for the previous tasks are less susceptible to be updated. Learning without forgetting (Li & Hoiem, 2018) proposes to use knowledge distillation to preserve previous performances. The network is divided in two parts : the shared weights and the dedicated weights for each task. When learning a new task A, the data of A get assigned "soft" labels by computing the output by the network with the dedicated weight for each previous task. Then the network is trained with the loss of task A and is also constrained to reproduce the recorded output for each other tasks. In Rannen et al. (2017), the authors propose to use an autoencoder to reconstruct the extracted features for each task. When learning a new task, the features extractor is adapted but has to make sure that the autoencoder of the other tasks are still able to reconstruct the extracted features from the current samples. While these methods obtain good results for learning one new task, they become limited when it comes to learn several new tasks, especially in the one class per batch setting.

Expandable models In the case of the multi-head setting, it has been proposed to use the previously learned layers and complete them with new layers trained on a new task. This strategy is presented in Progressive Networks (Rusu et al., 2016). In order to reduce the growth in memory caused by the new layers, the authors of Dynamically Expandable Networks (Yoon et al., 2018) proposed an hybrid method which retrains some of the previous weights and add new ones when necessary. Although these approaches work very well in the case of multi-head learning, they can not be adapted to single-head and are therefore not included in benchmarks with OvA-INN.

3 Class-by-Class Continual Learning with Invertible Networks

3.1 Motivations and Challenge

We investigate the problem of training several datasets in a sequential fashion with batches of only one class at a time. Most approaches of the state-of-the-art rely on updating a features extractor when data from a new class is available. But this strategy is unreliable in the special case we are interested in, namely batches of data from only one class. With few or no sample of negative data, it is very inefficient to update the weights of a network because the setting of deep learning normally involves vast amounts of data to be able to learn to extract valuable features. Without enough negative samples, the training is prone to overfit the new class. Recent works have proposed to rely on generative models to overcome this lack of data by generating samples of old classes. Nevertheless, updating a network with sampled data is not as efficient as with real data and, on the long run, the generative quality of early classes suffer from the multiple updates.

On the other hand, we already have pretrained networks at our disposal and it is possible to transfer their weights as features extractors when learning with data from new classes. Furthermore, there is currently promising results in training features extractors those first layers generalise very well on computer vision tasks (Asano et al., 2019; Kolesnikov et al., 2019). Those observations encourage us to rely on pretrained features extractors that won't be updated during the sequential training. This design choice will greatly simplify the learning process and make it more stable when adding new classes.

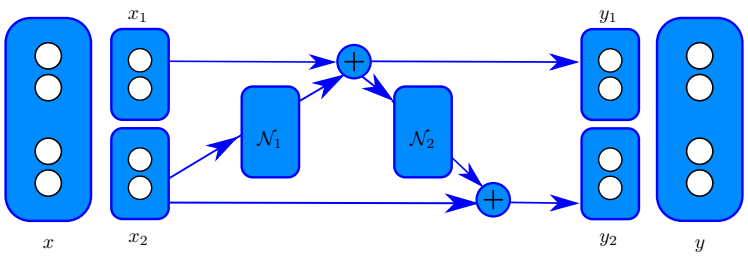

Figure 1: Forward pass in an invertible block. x is splitted in x_1 and x_2 of equal size. $\mathcal{N}_1$ and $\mathcal{N}_2$ can be any type of Neural Networks as long as the dimension of their output dimension is the same as their input dimension. In our experiments, we stack two of these blocks one after the other and use fully-connected feedforward layers for $\mathcal{N}_1$ and $\mathcal{N}_2$

3.2 PRINCIPLE

The simplest way to classify in a continual fashion with a fixed features extractor is to compute a *prototype* for each class. A prototype corresponds to the mean vector of the features for one class. At test time, one can predict the class of a sample by identifying the class of the most similar (the closest under euclidian distance) prototype to that sample. Nevertheless, this approach assumes that the vectors of extracted features for a class are clustered in a ball around their prototype and that no balls are interfering with one another. But this is generally not the case because networks are not trained that way, unless they are specifically constrained to (Koch et al., 2015). To overcome this issue, our idea is to apply a transformation on the vector of extracted features that would make vectors from a given class close to one another and keep vectors from other classes apart. This idea is closely related to Siamese Networks (Koch et al., 2015), except that in our case we do not have access to negative samples.

Our solution relies on Invertible Neural Networks trained to output a vector with a small norm when given samples from a particular class. These networks are constrained to represent a bijection between their input and their output. This property also holds when several invertible blocks are stacked on each other. With such a constraint, this architecture is conditioned to predict one part of an input features given another part of the input in a feedforward fashion (See subsection 3.3). This leads to a specific representation for each class. Although this approach requires to store an additional network for each new class, one could take advantage of model superposition to make the memory consumption of OvA-INN linear in m instead of m^2 for each new layer with a weight matrix $W \in \mathbb{R}^{m \times m}$ (Cheung et al., 2019).

3.3 INVERTIBLE NEURAL NETWORKS

The architecture of Invertible Neural Networks allows them to recover exactly the activation of any layer given the activation of the next layer (Gomez et al., 2017). Hence, it is possible to compute the input of an Invertible Neural Network given its output layer. These types of networks have been successfully applied on a GAN for image-to-image translation (van der Ouderaa & Worrall, 2019), a ResNet for image classification (Jacobsen et al., 2018) and a RNN to increase memory efficiency during training (MacKay et al., 2018).

An invertible block (see Figure 1) consists in splitting the input x into two subvectors x_1 and x_2 of equal size; then successively apply two (non necessarily invertible) networks $\mathcal{N}_1$ and $\mathcal{N}_2$ following the equation :

$$\begin{cases} y_1 = \mathcal{N}_1(x_2) + x_1 \\ y_2 = \mathcal{N}_2(y_1) + x_2, \end{cases} \tag{1}$$

and finally, concatenate y_1 and y_2. The inverse operation can be computed with :

$$\begin{cases} x_2 = y_2 - \mathcal{N}_2(y_1) \\ x_1 = y_1 - \mathcal{N}_1(x_2). \end{cases} \tag{2}$$

With these invertible equations, we can see that Invertible Networks operate a bijection between their input and their output. This way, when applied on data from classes it has never seen, the network won't be able have an output similar to the outputs on data from its training set. In particular, when constrained to output a vector of small norm (namely, a vector as close as possible to the zero vector), we can see from equation 1 that each layer will learn to predict the features of one part of the input given the features of the other part. This behavior will lead to the specialization of each network to the features of the class it has been trained on.

3.4 Continual Learning setting

We propose to specialize each Invertible Network to a specific class by training them to output a vector with small norm when presented with data samples from their class. Given a dataset $\mathcal{X}_i$ of class i and an Invertible Network $\mathcal{F}_i$, our objective is to minimize the loss $\mathcal{L}$:

$$\mathcal{L}(\mathcal{X}_i) = \frac{1}{|\mathcal{X}_i|} \sum_{x \in \mathcal{X}_i} \|\mathcal{F}(x)\|_2^2 \tag{3}$$

Once the training has converged, the weights of this network won't be updated when new classes will be added. At inference time, after learning t classes, the predicted class y^* for a sample x is obtained by running each network and identifying the one with the smallest output :

$$y^* = \underset{y=1,\dots t}{\arg\min} \|\mathcal{F}_y(x)\|_2^2 \tag{4}$$

As it is common practice in image processing, one can also use a preprocessing step by applying a fixed pretrained features extractor beforehand.

4 Experimental Results

We compare our method against several state-of-the-art baselines for single-head learning on MNIST and CIFAR-100 datasets.

4.1 Implementation details

Topology of OvA-INN Due to the bijective nature of Invertible Networks, their output size is the same as their input size, hence the only way to change their size is by changing the depth or by compressing the parameters of the intermediate networks $\mathcal{N}_1$ and $\mathcal{N}_2$. In our experiments, these networks are fully connected layers. To reduce memory footprint, we replace the square matrix of parameters W of size $n \times n$ by a product of matrices AB of sizes $n \times m$ and $m \times n$ (with a compressing factor for the first and second block $m = 16$ for MNIST and $m = 32$ for CIFAR-100). More details on the memory cost can be found in Appendix A.

Regularization When performing learning one class at a time, the amount of training data can be highly reduced: only 500 training samples per class for CIFAR-100. To avoid overfitting the training set, we found that adding a weight decay regularization could increase the validation accuracy. More details on the hyperparameters choices can be found in Appendix B.

Rescaling As ResNet has been trained on images of size 224×224, we rescale CIFAR-100 images to match the size of images from Imagenet.

4.2 Evaluation on MNIST

We start by considering the MNIST dataset (LeCun et al., 1998), as it is a common benchmark that remains challenging in the case of single-head Continual Learning.

Table 1: Comparison of accuracy and memory cost in number of parameters (and memory usage for storing samples if relevant) of different approaches on MNIST at the end of the Continual Learning.

Model	Accuracy (%)	Memory cost	Learning type
PGMA (Hu et al., 2018)	81.7	6,000k	2 by 2
SupportNet (Li et al., 2018)	89.9	940k	2 by 2
DGR (Shin et al., 2017)	95.8	12,700k	2 by 2
iCaRL (Rebuffi et al., 2017)	96.0	940k	2 by 2
OvA-INN (this work)	**96.4**	**520k**	**1 by 1**

Baselines

Generative models:
- Parameter Generation and Model Adaptation (PGMA) (Hu et al., 2018)
- Deep Generative Replay (DGR) (Shin et al., 2017)

Coreset-based models:
- iCaRL (Rebuffi et al., 2017)
- SupportNet (Li et al., 2018)

For Parameter Generation and Model Adaptation (PGMA) (Hu et al., 2018) and Deep Generative Replay (DGR) (Shin et al., 2017), we report the results from the original papers; whereas we use the provided code of SupportNet to compute the results for iCaRL and SupportNet with the conventional architecture of two layers of convolutions with poolings and a fully connected last layer. We have also set the coreset size to $s = 800$ samples.

Analysis

We report the average accuracy over all the classes after the networks have been trained on all batches (See Table 1). Our architecture does not use any pretrained features extractor common to every classes (contrarily to our CIFAR-100 experiment) : each sample is processed through an Invertible Network, composed of two stacked invertible blocks.

Our approach presents better results than all the other reference methods while having a smaller cost in memory (see Appendix A) and being trained by batches of only one class. Also, our architecture relies on simple fully-connected layers (as parts of invertible layers) whilst the other baselines implement convolutional layers.

4.3 Evaluation on CIFAR-100

We now consider a more complex image dataset with a greater number of classes. This allows us to make comparisons in the case of a long sequence of data batches and to illustrate the value of using a pretrained features extractor for Continual Learning.

Baselines

Distance-based model:
- Nearest prototype : our implementation of the method consisting in computing the mean vector (prototype) of the output of a pretrained ResNet32 for each class at train time. Inference is done by finding the closest prototype to the ResNet output of a given sample.

Generative model:
- FearNet (Kemker & Kanan, 2017) : uses a pretrained ResNet48 features extractor. FearNet is trained with a warm-up phase. Namely, the network is first trained with the all the first 50 classes of CIFAR-100, and subsequently learns the next 50 classes one by one in a continual fashion.

Coreset-based models:
- iCaRL (Rebuffi et al., 2017) : retrains a ResNet32 architecture on new data with a distillation loss.
- End-to-end IL (Castro et al., 2018) : retrains a ResNet32 architecture on new data with a cross-entropy together with distillation loss.

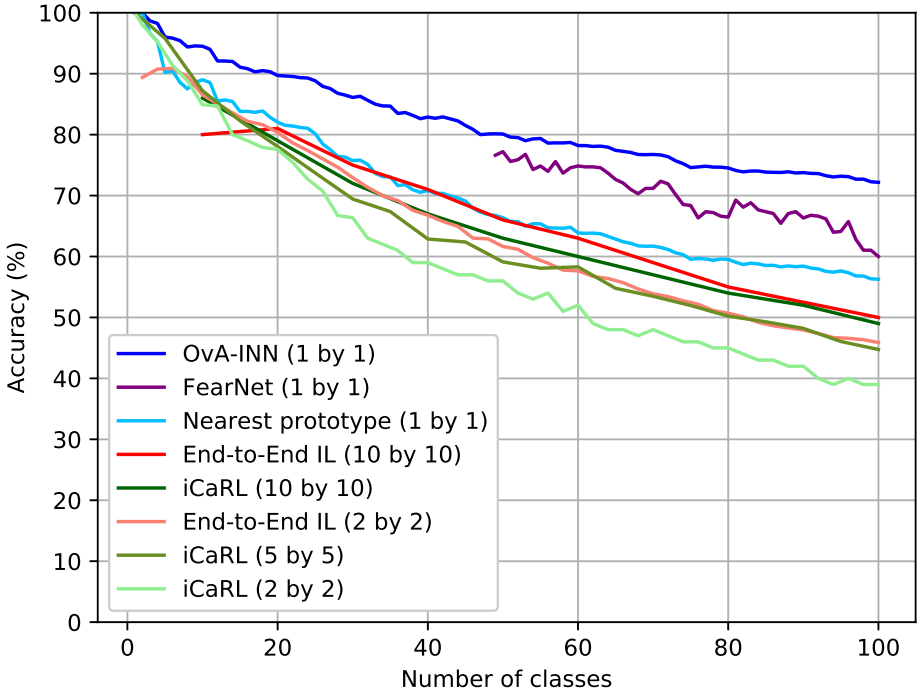

Figure 2: Comparison of the accuracy of several Continual Learning methods on CIFAR-100 with various batches of classes. FearNet's curve has no point before 50 classes because the first 50 classes are learned in a non-continous fashion.

Analysis

The data is provided by batch of classes. When the training on a batch ($\mathcal{D}_i$) is completed, the accuracy of the classifier is evaluated on the test data of classes from all previous batches ($\mathcal{D}_1, ..., \mathcal{D}_i$). We report the results from the literature with various size of batch when they are available.

OvA-INN uses the weights of a ResNet32 pretrained on ImageNet and never update them. FearNet also uses pretrained weights from a ResNet. iCaRL and End-to-End IL use this architecture but retrain it from scratch at the beginning and fine-tune it with each new batch.

The performance of the Nearest prototype baseline proves that there is high benefit in using pretrained features extractor on this kind of dataset. FearNet shows better performance by taking advantage of a warm-up phase with 50 classes. Still, we can see that OvA-INN is able to clearly outperform all the other approaches, reaching 72% accuracy after training on 100 classes. We can see that the performances of methods retraining ResNet from scratch (iCaRL and End-to-End IL) quickly deteriorate compared to those using pretrained parameters. Even with larger batches of classes, the gap is still present.

It can be surpising that at the end of its warm-up phase, FearNet still has an accuracy bellow OvA-INN, even though it has been trained on all the data available at this point. FearNet is indeed training an autoencoder and uses its encoding part as a features extractor (stacked on the ResNet) before classifying a sample. This can diminish the discriminative power of the network since it is also constrained to reproduce its input (only a single autoencoder is used for all classes).

To further understand the effect of an Invertible Network on the feature space of a sample, we propose to project the different features spaces in 2D using t-SNE (Maaten & Hinton, 2008). We project the features of the five first classes of CIFAR-100 test set (see Figure 3). Classes that are already well represented in a cluster with ResNet features (like violet class) are clearly separated from the clusters of Invertible Networks. Classes represented with ambiguity with ResNet features (like light green and red) are better clustered in the Invertible Network space.

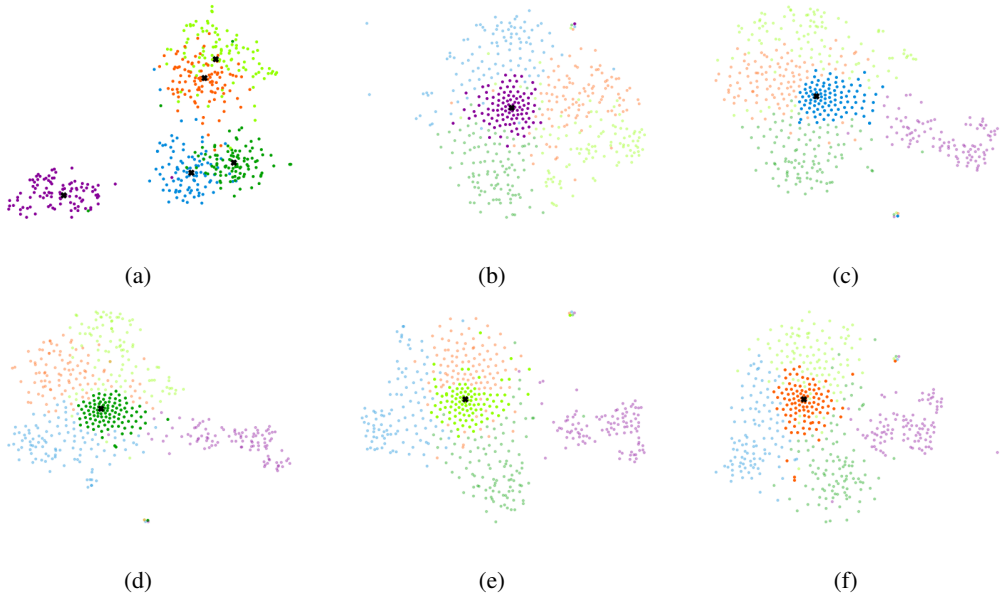

Figure 3: t-SNE projections of features spaces for five classes from CIFAR-100 test set (colors are given by the ground truth). *(a)*: features space before applying Invertible Networks (black crosses are the clusters centers). *(b),(c),(d),(e),(f)*: each features space after the Invertible Network of each class. The samples of a class represented by a network are clustered around the zero vector (black cross) whilst the samples from other classes appear further away from the cluster. Another visualization highlighting the differences between OvA-INN and Nearest Prototype is presented in Annexe C.

5 Conclusion

In this paper, we proposed a new approach for the challenging problem of single-head Continual Learning without storing any of the previous data. For each class, we trained an Invertible Network to output a vector with small norm when presented samples from that class. Because these networks are bijective, they get less effective at producing such output for other classes. This way, we show that we can identify the class of a sample by running each network and finding the one with the smallest norm. This setting allows us to take full benefit of pretrained models, which results in very good performances on the class-by-class training of CIFAR-100 compared prior work.

A limiting factor in our approach is the necessity to add a new network each time one wants to learn a new class. This makes the memory consumption of OvA-INN linear with the number of classes. Recent works in networks merging could alleviate this issue by sharing weights (Chou et al., 2018) or relying on weights superposition (Cheung et al., 2019). This being said, we showed that Ova-INN was able to achieve superior accuracy on CIFAR-100 class-by-class training than approaches reported in the literature, while using less parameters.

Another constraint of using Invertible Networks is to keep the size of the output equal to the size of the input. When one wants to apply a features extractor with a high number of output channel, it can have a very negative impact on the memory consumption of the invertible layers. Feature Selection or Feature Aggregation techniques may help to alleviate this issue (Tang et al., 2014).

Finally, we can notice that our approach is highly dependent on the quality of the pretrained features extractor. In our CIFAR-100, we had to rescale the input to make it compatible with ResNet. Nonetheless, recent research works show promising results in training features extractors in very efficient ways (Asano et al., 2019). Because it does not require to retrain its features extractor, we can foresee better performance in class-by-class learning with OvA-INN as new and more performant features extractors are discovered.

REFERENCES

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

Yuki M Asano, Christian Rupprecht, and Andrea Vedaldi. Surprising effectiveness of few-image unsupervised feature learning. *arXiv preprint arXiv:1904.13132*, 2019.

Francisco M Castro, Manuel J Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari. End-to-end incremental learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 233–248, 2018.

Arslan Chaudhry, Puneet K Dokania, Thalaiyasingam Ajanthan, and Philip HS Torr. Riemannian walk for incremental learning: Understanding forgetting and intransigence. *arXiv preprint arXiv:1801.10112*, 2018.

Brian Cheung, Alex Terekhov, Yubei Chen, Pulkit Agrawal, and Bruno A. Olshausen. Superposition of many models into one. *CoRR*, abs/1902.05522, 2019. URL `http://arxiv.org/abs/1902.05522`.

Yi-Min Chou, Yi-Ming Chan, Jia-Hong Lee, Chih-Yi Chiu, and Chu-Song Chen. Unifying and merging well-trained deep neural networks for inference stage. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pp. 2049–2056. International Joint Conferences on Artificial Intelligence Organization, 7 2018. doi: 10.24963/ijcai.2018/283. URL `https://doi.org/10.24963/ijcai.2018/283`.

Sebastian Farquhar and Yarin Gal. Towards robust evaluations of continual learning. *arXiv preprint arXiv:1805.09733*, 2018.

Aidan N Gomez, Mengye Ren, Raquel Urtasun, and Roger B Grosse. The reversible residual network: Backpropagation without storing activations. pp. 2214–2224, 2017.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. 2016.

Wenpeng Hu, Zhou Lin, Bing Liu, Chongyang Tao, Zhengwei Tao, Jinwen Ma, Dongyan Zhao, and Rui Yan. Overcoming catastrophic forgetting via model adaptation. 2018.

Jorn-Henrik Jacobsen, Arnold Smeulders, and Edouard Oyallon. i-RevNet: Deep invertible networks. *arXiv preprint arXiv:1802.07088*, 2018.

Ronald Kemker and Christopher Kanan. FearNet: Brain-inspired model for incremental learning. *arXiv preprint arXiv:1711.10563*, 2017.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, pp. 201611835, 2017.

Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, volume 2, 2015.

Alexander Kolesnikov, Xiaohua Zhai, and Lucas Beyer. Revisiting self-supervised visual representation learning. *arXiv preprint arXiv:1901.09005*, 2019.

Yann LeCun, Leon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Yu Li, Zhongxiao Li, Lizhong Ding, Peng Yang, Yuhui Hu, Wei Chen, and Xin Gao. SupportNet: solving catastrophic forgetting in class incremental learning with support data. *arXiv preprint arXiv:1806.02942*, 2018.

Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(12):2935–2947, 2018.

David Lopez-Paz et al. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems*, pp. 6467–6476, 2017.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

Matthew MacKay, Paul Vicol, Jimmy Ba, and Roger B Grosse. Reversible recurrent neural networks. In *Advances in Neural Information Processing Systems*, pp. 9043–9054, 2018.

Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. 24:109–165, 1989.

German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *arXiv preprint arXiv:1802.07569*, 2018.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

Amal Rannen, Rahaf Aljundi, Matthew B Blaschko, and Tinne Tuytelaars. Encoder based lifelong learning. pp. 1320–1328, 2017.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. iCaRL: Incremental classifier and representation learning. pp. 5533–5542, 2017.

Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. pp. 2990–2999, 2017.

Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. pp. 4077–4087, 2017.

Jiliang Tang, Salem Alelyani, and Huan Liu. Feature selection for classification: A review. *Data classification: Algorithms and applications*, pp. 37, 2014.

Gido M. van de Ven and Andreas S. Tolias. Three scenarios for continual learning. *CoRR*, abs/1904.07734, 2019. URL `http://arxiv.org/abs/1904.07734`.

Tycho FA van der Ouderaa and Daniel E Worrall. Reversible gans for memory-efficient image-to-image translation. *arXiv preprint arXiv:1902.02729*, 2019.

Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. pp. 3630–3638, 2016.

Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks. 2018.

Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. pp. 3987–3995, 2017.

A MEMORY USAGE

A.1 MNIST

OvA-INN uses 2 blocks with 2 layers ($\mathcal{N}_1$ and $\mathcal{N}_2$) for 10 classes. The weight matrix of each layer W is a product of two matrices A and B of size 392×16 and 16×392. The memory required for OvA-INN is :

$\mathcal{S}_{OvA-INN,MNIST} = (392 \times 16 \times 2 + 392) \times 2 \times 2 \times 10 = 517440$

We set the coreset size of iCaRL and SupportNet to $s = 800$ with each image of size 28×28, the convolutional network is composed of a layer of 64 channels with 5×5 kernel, a layer of 32 channels with 5×5 kernel, a fully-connected layer with 100 channels applied on an input of size 7×7 and a final layer of 10 channels :

$\mathcal{S}_{iCaRL,CIFAR} = 28 \times 28 \times 800 + (5 \times 5 + 1) \times 32 + (5 \times 5 + 1) \times 64 + (7 \times 7 \times 64 + 1) \times 100 + (100 + 1) \times 10 = 944406$

A.2 CIFAR-100

Since every method rely on a ResNet32 (around 20M parameters) to compute their features (except FearNet which uses ResNet48). We do not count the features extractor in the memory consumption.

OvA-INN uses 2 blocks with 2 layers ($\mathcal{N}_1$ and $\mathcal{N}_2$) for 100 classes. The weight matrix of each layer W is a product of two matrices A and B of size 256×32 and 32×256. The memory required is :

$\mathcal{S}_{OvA-INN,CIFAR} = (256 \times 32 \times 2 + 256) \times 2 \times 2 \times 100 = 6656000$

We use the default coreset size $s = 2000$ of iCaRL and End-to-End IL with each image of size 32×32 :

$\mathcal{S}_{iCaRL,CIFAR} = 32 \times 32 \times 3 \times 2000 = 6144000$

B HYPERPARAMETERS SETTINGS

Our implementation is done with Pytorch (Paszke et al., 2017), using the Adam optimizer (Kingma & Ba, 2014) and a scheduler that reduces the learning rate by a factor of 0.5 when the loss stops improving. We use the resize transformation from torchvision with the default bilinear interpolation.

Table 2: MNIST Hyperparameters

Hyperparameter	**Value**
Learning Rate	0.002
Number of epochs	200
Weight decay	0.0
Patience	20

Table 3: CIFAR-100 Hyperparameters

Hyperparameter	**Value**
Learning Rate	0.002
Number of epochs	1000
Weight decay	0.0002
Patience	30

C VISUALIZATION

We highlight the differences between OvA-INN and Nearest Prototype when classifying 20 classes of CIFAR-100 in Figure 4.

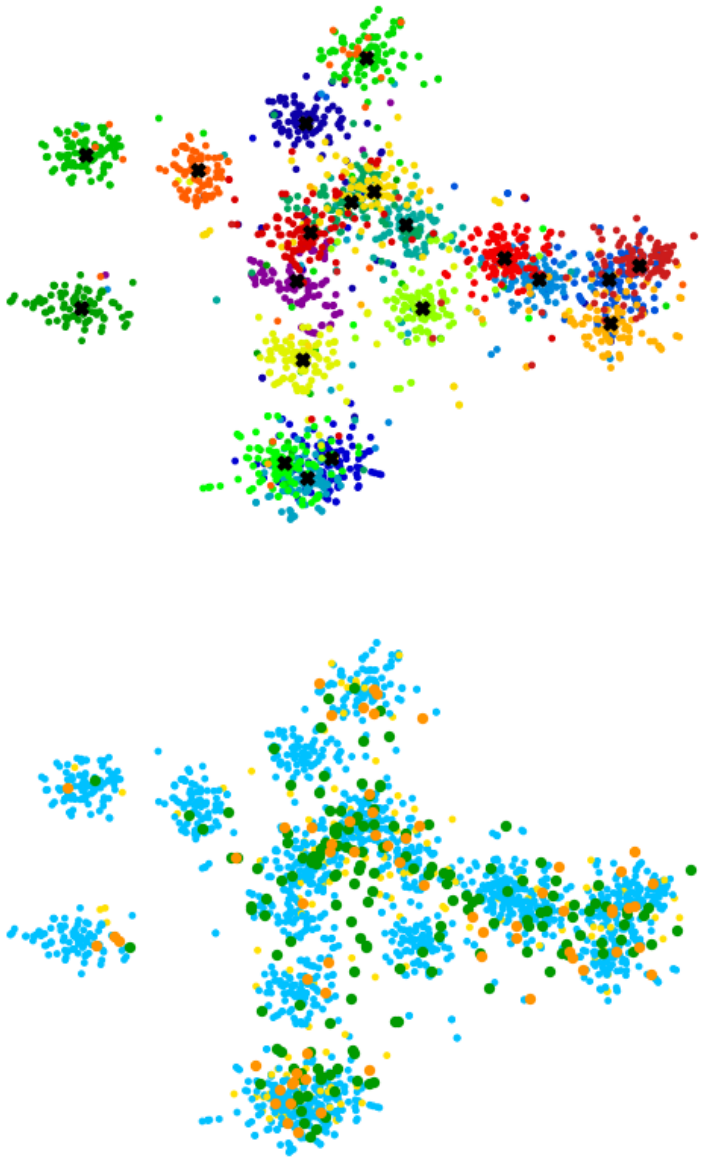

Figure 4: *top*: t-SNE projection of the features space before applying Invertible Networks (black crosses are the clusters centers) for 20 classes from CIFAR-100 test set (colors are given by the ground truth). *bottom*: in blue and yellow are the samples correctly and wrongly classified by both Nearest Prototype and OvA-INN, in green the samples better classified by OvA-INN than Nearest Prototype and orange the samples better classified by Nearest Prototype than OvA-INN.