Class-incremental Learning using a Sequence of Partial Implicitly Regularized Classifiers

Sobirdzhon Bobiev and **Albina Khusainova** and **Adil Khan**
Innopolis University
Innopolis, Russia

S.M. Ahsan Kazmi
University of the West of England
Bristol, United Kingdom

Abstract

In class-incremental learning, the objective is to learn a number of classes sequentially without having access to the whole training data. However, due to a problem known as *catastrophic forgetting*, neural networks suffer substantial performance drop in such settings. The problem is often approached by *experience replay*, a method that stores a limited number of samples to be replayed in future steps to reduce forgetting of the learned classes. When using a pretrained network as a feature extractor, we show that instead of training a single classifier incrementally, it is better to train a number of specialized classifiers which do not interfere with each other yet can cooperatively predict a single class. Our experiments on CIFAR100 dataset show that the proposed method improves the performance over SOTA by a large margin.

Introduction

Artificial neural networks (ANNs) have been at the top of the machine learning landscape for a while. While inspired by the biological brain, still there are some shortcomings that make them different. Specifically, they are not designed to learn in an incremental way, as humans do. Humans keep learning new knowledge throughout their lives. However, experiments show that ANNs almost completely forget their previous knowledge when they are trained on a new task (Kemker et al. 2017). This is termed as *catastrophic forgetting* (McCloskey and Cohen 1989).

The common way to train ANNs is to provide them the whole dataset at once and let them iterate through it multiple times. However, there are scenarios when this is not a feasible option. There can be memory limitations in the learning device, making it impossible to store all data. Also, there may be security concerns restricting the permanent storage of sensitive data. Besides that, re-learning a large model from scratch any time new data comes may be too computationally costly. Catastrophic forgetting in ANNs was first addressed by McCloskey (McCloskey and Cohen 1989) in 1989 but is still an unsolved problem hindering the progress towards building AI agents that can learn continuously.

The focus of this paper is class-incremental learning, where at each training session multiple new classes are to be learned while also maintaining the knowledge of previous classes. The closely related problem is called task-incremental learning, where the model learns a sequence of tasks sequentially, and during inference it is provided with the task identity. In contrast, for class-incremental learning, no such information is provided at inference time, and the model is required to predict both the correct class and the task.

Despite many proposed solutions for continual learning in the literature, the baseline methods like Experience Replay (ER) (Buzzega et al. 2020) and GDumb (Prabhu, Torr, and Dokania 2020) are still shown to be as effective as the state of the art. Given that each training session contains examples of only a few classes, the deep networks are prone to overfitting. Therefore, it is plausible to use a frozen feature extractor and smaller classifiers on top. ER trains only a single network, extending its output units for the new classes. In each training session, the network is trained on the examples of new classes as well as a small number of examples from previous classes which have been stored in the limited memory buffer. While retraining on these few stored examples helps retain the knowledge of past classes, there is still no guarantee that the new knowledge will not interfere with the past. To solve this problem, we propose to train a separate classifier for each group of new classes. These classifiers do not share any weights with each other, implying that the newly acquired knowledge will be stored separately without overriding previous classifiers.

Related Work

In this section, we overview the literature on continual learning methods. We divide these methods into three groups based on the main idea: replay, regularization, and architectural techniques.

Replay Methods

A straightforward solution to prevent catastrophic forgetting is to revisit the previous tasks. Rehearsal methods accomplish this by storing a small portion of the seen examples for later retraining. When the model is faced with new training data, it augments it with memory samples to reduce the forgetting of previous knowledge.

iCaRL (Rebuffi et al. 2017) uses the nearest-mean-of-exemplars classification approach, classifying items to the

class with the nearest center. It uses a heuristic approach for updating the memory buffer, prioritizing items based on their proximity to their class means. Together with network distillation, it has been able to learn in a class-incremental learning scenario. Gradient Episodic Memory (GEM) (Lopez-Paz and Ranzato 2017) is designed for task-incremental learning from streaming data. In this setting, the model receives a series of tuples (x, t, y) consisting of input, task label, and target, respectively. GEM uses the memory samples not for replay but to serve the inequality constraints that prevent the loss for the past tasks to decrease. Replay using Memory Indexing (REMIND) (Hayes et al. 2020) applies quantization to the extracted feature maps from deeper layers of a CNN and stores their indices in memory. This results in a compressed representation and allows for a much bigger number of stored examples. REMIND's architecture consists of a frozen feature extractor followed by a trainable classifier. REMIND was shown to outperform existing approaches in a streaming setup on the ImageNet (Deng et al. 2009) and other datasets.

Rethinking Experience Replay (Buzzega et al. 2020): In this work, the authors emphasize that *simple experience replay* (or simple rehearsal) (Ratcliff 1990) is still as effective as the state of the art if certain issues are resolved. As stated by the authors, the three important issues with this approach are: overfitting to stored examples, biased prediction and accuracy towards the later classes, and non-i.i.d stored data in cases where the buffer is small. Their proposals include an additional bias correction layer (BiC) (Wu et al. 2019), exponential decay of learning rate, and balanced sampling for the memory buffer. In their reported results ER has outperformed SOTA sophisticated replay methods such as iCaRL (Rebuffi et al. 2017), GEM (Lopez-Paz and Ranzato 2017), A-GEM (Chaudhry et al. 2018), and HAL (Chaudhry et al. 2020), sometimes by a very large margin.

GDumb (Prabhu, Torr, and Dokania 2020): This paper proposes a simple and most generic baseline for continual learning. It basically maintains a balanced memory buffer and only trains on the samples contained in it. The reported results have shown that it outperforms many SOTA methods in their respective settings. GDumb serves as a strong baseline for all continual settings including class-incremental learning.

Regularization Methods

These methods impose a type of regularization that helps to retain the learned knowledge of past tasks. This is achieved by adding special loss terms to the loss function.

Learning without forgetting (LwF) (Li and Hoiem 2017) presents a modification to the standard fine-tuning. A new network is initialized as a copy of the old one with an extension of the output layer (called *multihead* approach) for the new task. A distillation loss is added between old and new task heads so the new network is reminded of the old tasks indirectly. In the backward pass, only the new network is updated.

Elastic Weight Consolidation (EWC) (Kirkpatrick et al. 2017) and Synaptic Intelligence (SI) (Zenke, Poole, and Ganguli 2017) both try to approximate the importance of each of the parameters for previous tasks and selectively apply regularization to limit their change. Regularization methods alone, as several papers demonstrate, are not sufficient for proper class-incremental learning (Kemker et al. 2017; Lesort, Stoian, and Filliat 2020).

Architectural Methods

Architectural methods try to manipulate weights, neurons, layers, or architecture of the network to protect the learned knowledge while acquiring the new one. They either use fixed or dynamic architectures. These methods have the advantage of completely eliminating interference between tasks while allowing knowledge transfer between them. However, practically, they are coupled with scalability issues as the number of tasks grows.

Progressive Neural Networks (PNN) (Rusu et al. 2016) inherently target the task-incremental learning. Their architecture grows laterally, each time adding a new neural network, called "column", for a new task. The new columns have connections to other layers of previous networks and thus, highly benefit from knowledge transfer. Once a column is trained, it will be kept frozen making the PNNs completely immune to catastrophic forgetting. A big issue with PNNs is that they keep growing too large, limiting their use in practice to only a small number of tasks. Compacting, Picking and Growing (CPG) (Hung et al. 2019) tries to overcome this issue by dynamically controlling the architecture of the network. The weights of the network are grouped into "compact" and "free", where the newly added weights are considered free until they are trained and then compacted. For each new task, a learnable binary mask is created and applied to the compact weights to select a subset of them. Then this task is learned using this subset of compact weights and the free weights. The network is provided more free weights during training if the performance does not reach a satisfactory level. The training is followed by pruning to compact the newly learned weights. This method is useful in scenarios with fewer but larger tasks.

Similarity with other works Here we compare our approach with similar works in the literature. Our main difference is that we train a set of specialized classifiers dedicated to each class group while also making them able to detect if a sample is from previous classes.

Aljundi et al. (Aljundi, Chakravarty, and Tuytelaars 2019) train a specialized model for each new task and propose a method for choosing the relevant one at inference time. Specifically, they train separate autoencoders describing each task's data distribution, and during inference they choose the model for which the corresponding autoencoder has the least reconstruction error. In our work, the classifiers are not separate models, but parts of a larger model, which allows for knowledge sharing.

AR1 (Maltoni and Lomonaco 2019) trains a linear classifier on top of a feature extractor. In each phase, a new linear classifier is trained for the current group of classes. Then a mean-normalization is applied to the weights of this classifier to eliminate the prediction bias. During testing, the pre-

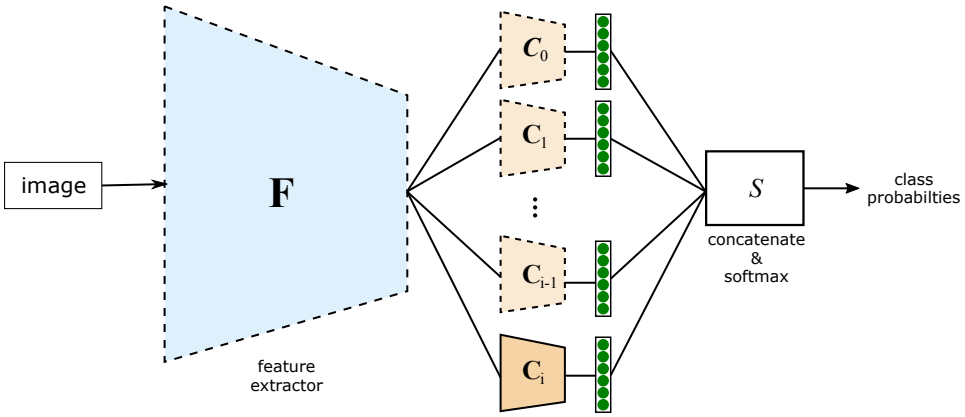

Figure 1: The proposed model architecture: a feature extractor $\mathbf{F}$ (pale-blue trapezoid) and a growing number of classifiers (orange trapezoids), followed by the final layer S which concatenates all classifier outputs together and applies the softmax function. At each training session i, only the newly added classifier $\mathbf{C}_i$ is trained while the previous classifiers and the feature extractor are kept frozen. The frozen modules are represented by dashed borders.

diction of the model is the softmax over all classifiers' outputs. In contrast to their approach, we use deeper classifiers and memory buffer.

Methods

Our goal is to sequentially train on a number of disjoint datasets containing different classes. Relying on a pretrained deep feature extractor, we only consider training small networks on top of it. The traditional way is to construct and train a single head with expanding output units to facilitate the prediction of new classes. However, despite the replay mechanism, the single head is still not sufficiently equipped against catastrophic forgetting. We improve the situation by instead training much smaller classifiers, one at a time, that are specific only to new classes in the training session. For a descriptive diagram of our method, please see Fig. 1. In the following paragraph, we further formalize the training setting and details of our method.

In class-incremental learning we aim to train a network on a sequence of datasets $\mathcal{D}_i = (x_i^j, y_i^j)_{j=1}^{n_i}$ with inputs x_i^j and labels $y_j^i \in \mathcal{Y}_i$, where $\mathcal{Y}_i \cap \mathcal{Y}_k = \emptyset$ for any $i \neq k$. Note that in the training session i only the dataset $\mathcal{D}_i$ is available. Suppose we are allowed to store a small number of examples from the current training session in a limited memory buffer. We denote the samples in the memory buffer by $\mathcal{M}_i$ and its capacity by B, thus $\mathcal{M}_i \leq B$. The memory can be updated with new examples at each training session. Having access to past examples through the memory buffer, the training data for the i-th session will then consist of $\mathcal{D}_i \cup \mathcal{M}_i$.

Let $\mathbf{F}(\cdot)$ be the feature extraction network. At i-th training session we create a new classifier network $\mathbf{C}_i$ with $|\mathcal{Y}_i|$ output units that classifies between the new classes. The final classification is done by the final layer $S(\cdot) = \text{Softmax}(\oplus(\cdot))$ which simply concatenates the outputs of all classifiers and then applies the softmax. The objective is to minimize the cross-entropy loss between the outputs of the final layer and the true targets over all training samples:

$$\mathcal{L}_{\theta_i}(x, y) = -\sum_{j}^{C} t_j \log(S(c_0, \ldots, c_i)_j)$$

where

$$\begin{aligned} c_k &= \mathbf{C}_k(\mathbf{F}(x)) \quad \text{for} \quad k = 0, \ldots, j \\ \theta_i &- \text{ parameters of the last classifier, } \mathbf{C}_i \\ t &- \text{the one-hot encoded vector of target } y \\ C &= \sum_{k=1}^{i} |\mathcal{Y}_k| \quad \text{(i.e. the number of classes seen so far)} \end{aligned}$$

Note that only the last classifier is trained, while the loss is a function of the outputs of all classifiers. In this way, the last classifier is adjusting itself to respect the prediction of previous classifiers. In other words, it is learning to produce higher output values for the samples belonging to its feature space (the new classes in the current training session) while suppressing itself for the samples belonging to previous classifiers. Although all of the classifiers are in a sense "partial", i.e. they can only predict the classes belonging to them, they also see previous classes during the training as "negative" examples acting as a regularizer which makes them even stronger and more robust. We also emphasize that since the previous classifiers are frozen, it eliminates the problem of "forgetting" for them.

Memory buffer

As the memory should be kept updated to include new samples, a sampling strategy has to decide which examples should be selected or removed. We adopt the greedy sampling approach described in (Prabhu, Torr, and Dokania 2020). It randomly replaces some of the old examples in the

memory with the new ones, while trying to satisfy the balancing constraint, that is, to maintain an equal number of examples for each class.

Training

Training consists of multiple sessions. During each session, we train a new classifier which has the output size equal to the number of classes in the training data. We train it by mini-batch gradient descent where each mini-batch contains an equal number of samples from the current dataset and the memory buffer. At the end of each training session, the memory buffer is updated with new samples.

Early Stopping

As the number of examples per class in the memory keeps decreasing over time, the model becomes prone to overfitting. Therefore, an early stopping mechanism is essential. Apart from splitting the incoming data into train and validation parts, we also dedicate a part of the memory buffer for validation samples.

Bias correction layer

Since the mini-batches contain a nonequal number of samples from the old and new classes, it poses a class imbalance problem. This problem has been first addressed in (Wu et al. 2019) and the authors proposed a simple yet effective solution, called Bias Correction (BiC). It is a linear transformation layer with only two parameters which is applied to the output logits belonging to new classes $\mathcal{Y}_i$ as follows:

$$q_k = \begin{cases} \alpha o_k + \beta & k \in \mathcal{Y}_i \\ o_k & otherwise \end{cases}$$

where α and β are the bias parameters and o_k is the output logits of the final layer. The BiC layer is trained separately at the end of each training session with a small amount of data as it contains only two parameters.

Experimental Results

We consider the class-incremental learning scenario on the CIFAR100 dataset (Krizhevsky 2009) by splitting it into 5, 10, and 20 disjoint parts, each part containing 20, 10, and 5 classes respectively. In all of our experiments, we use the same class ordering which is obtained by random shuffling. We also run experiments with different memory sizes. We compare our method against two state-of-the-art baselines: ER equipped with BiC, and GDumb.

Implementation details

For feature extraction, we use a EfficientNet-B0 network (Tan and Le 2019) that is pretrained on ImageNet. We remove from it the final convolution layer, the output layer, and the final MBConv block. Since this feature extractor accepts inputs of size 224×224, we resize the CIFAR100 images which are 32×32 to match the input size. The feature extractor is kept frozen in all experiments of our method and the baselines. For classifiers, we use a single 1×1 convolution layer followed by global average pooling and a dense layer. In all experiments, the classifiers have the same architecture except the number of filters in the convolution layer and the number of units in the final dense layer. Overall, we have made sure that our method does not use more trainable parameters in total than the baselines (see Table 1).

Table 1: Number of trainable parameters in each of the experiment settings, when training on CIFAR100 split into 5, 10, and 20 parts.

Method	# trainable parameters		
	5 splits	10 splits	20 splits
GDumb	118K	118K	118K
ER	118K	118K	118K
Ours	17K $\rightarrow$ 86K	8K $\rightarrow$ 82K	4K $\rightarrow$ 80K

Table 2: Accuracies at the end of the training. CIFAR100 split into 5, 10, and 20 parts.

Splits	5	10			20	
Memory	2000	500	1000	2000	1000	2000
GDumb	48.24	23.79	39.89	48.74	39.57	47.83
ER w/ BiC	57.89	42.46	48.97	54.68	48.78	55.23
Ours	**67.83**	**56.45**	**63.48**	**65.79**	**61.60**	**63.25**

In all experiments, we stop a training session when the validation loss does not improve for 10 consecutive epochs. We have dedicated 10% of the data for validation. For all methods, we start with a learning rate of 0.01. For our method, we decrease the learning rate when the validation loss does not improve for 3 consecutive epochs. For GDumb and ER we apply exponential learning rate scheduling. At the end of each training session, just before testing, we train the BiC layer to remove the prediction bias towards later classes. Since the whole purpose of BiC is to remove prediction bias, we train it only on the validation part of the memory buffer (after it has been updated to include all seen classes) which the model itself has never been trained on.

Baselines

GDumb and ER use the same architecture, and the difference is in training. Gdumb only trains on memory samples. It updates the memory buffer at the beginning of each training session and then trains only on the memory buffer discarding the rest of the data. We also train a BiC layer at the end of each training session of ER. BiC layer is not necessary for GDumb since it trains on the balanced dataset, i.e., the memory buffer.

Analysis

We test the models at the end of each training session over all classes that have been encountered so far. In Table 2 we report the accuracy at the end of the training in three different settings with varying memory buffer sizes (500, 1000, and 2000) and dataset splitting into 5 (Fig. 2), 10 (Fig. 3), and 20 (Fig. 4) parts. In all settings, we see a large gap in accuracy between our method and the second best method,

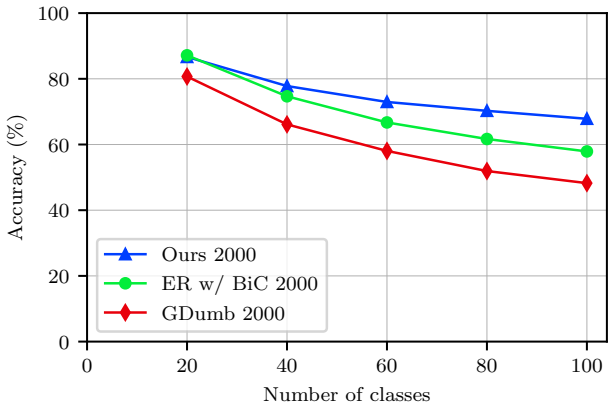

Figure 2: Accuracies at the end of each training session. CIFAR100 split into 5 parts.

ER. ER always outperforms GDumbs, as expected, because it trains on all available data, and the early stopping mechanism that we introduced here prevents it from unnecessary training which causes more forgetting of past data.

We observe that the gap in accuracy tends to get larger when we shrink the memory buffer size. Our method still reaches a remarkable performance of 56.45% when memory size is 500 leaving a gap of 14% relative to the next method, ER. On the other hand, we see a large performance drop of GDumb. This is mainly because it is highly dependent on the memory size as it trains only on the memory samples. However, GDumb shows the least difference in performance when training with different dataset splittings, meaning that it might have an advantage in settings closer to online learning.

Ablation studies

We conduct the experiments to see if all components of our method are indeed important. The first component is freezing the classifiers' weights after training to prevent the problem of "forgetting". Therefore we have conducted an experiment where we let our method keep updating all classifiers. The second component is the additional BiC layer. Looking at the confusion matrices, we have observed a prediction bias towards the last group of classes which was a signal to incorporate BiC layer. We also present the results here where our method does not use a BiC layer. These experiments are conducted in the case of CIFAR100 split into 10 classes, and a memory buffer size of 2000. The reported results are in Table 3. We can see the benefit of using BiC layer, which confirms that our method would have some prediction bias without it. On the other hand, we can see a large drop in accuracy (65.79% $\rightarrow$ 57.04%) when we allow updating our classifiers. Nevertheless, still, the performance stays above the other two methods (looking at Table 2, 54.68% and 48.74% of ER and GDumb, respectively).

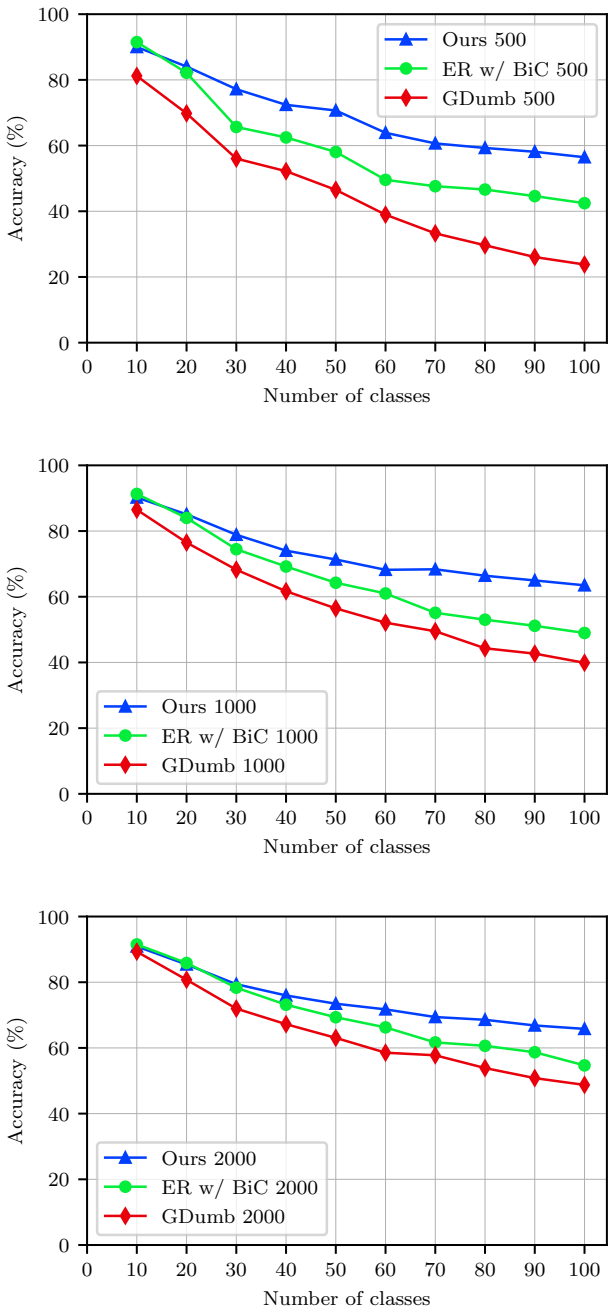

Figure 3: Accuracies at the end of each training session. CIFAR100 split into 10 parts.

Table 3: Ablation studies testing our method in two alternative forms: (1) without freezing the classifiers, and (2) without using BiC layer.

Method	Accuracy %
Ours w/out freezing	57.04
Ours w/out BiC	62.86
Ours	65.79

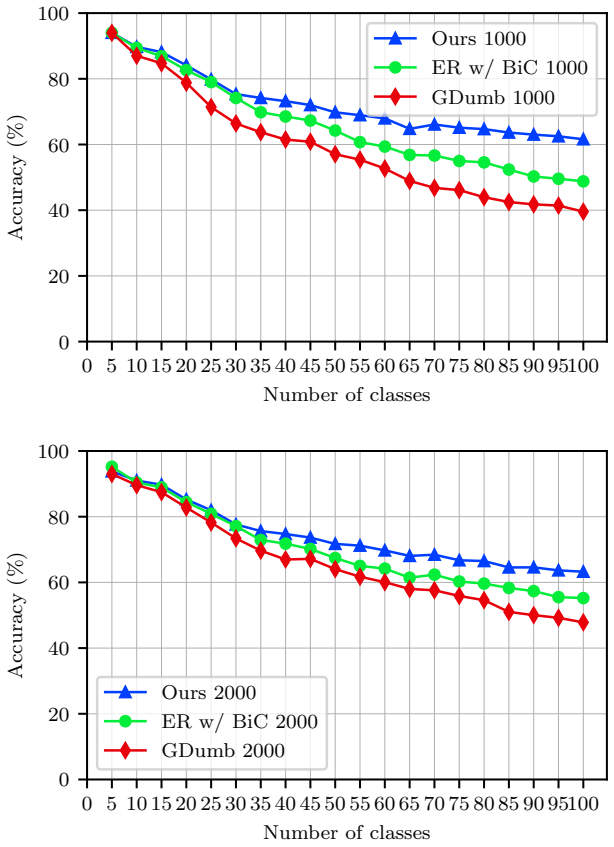

Figure 4: Accuracies at the end of each training session. CIFAR100 split into 20 parts.

Conclusion

In this paper, we proposed a new approach for class-incremental learning. We tackled the problem by training a separate classifier for each new group of classes. By freezing these classifiers after they have been trained and correcting bias towards later classes, we have limited the problem of "forgetting" and achieved big improvements over strong SOTA baselines. Our method consistently achieved better performance in different learning scenarios for the CIFAR100 dataset.

References

Aljundi, R.; Chakravarty, P.; and Tuytelaars, T. 2017. Expert gate: Lifelong learning with a network of experts.

Buzzega, P.; Boschini, M.; Porrello, A.; and Calderara, S. 2020. Rethinking experience replay: a bag of tricks for continual learning. *arXiv preprint arXiv:2010.05595*.

Chaudhry, A.; Ranzato, M.; Rohrbach, M.; and Elhoseiny, M. 2018. Efficient lifelong learning with a-gem. *arXiv preprint arXiv:1812.00420*.

Chaudhry, A.; Gordo, A.; Dokania, P. K.; Torr, P.; and Lopez-Paz, D. 2020. Using hindsight to anchor past knowledge in continual learning. *arXiv preprint arXiv:2002.08165*.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, 248–255. Ieee.

Hayes, T. L.; Kafle, K.; Shrestha, R.; Acharya, M.; and Kanan, C. 2020. Remind your neural network to prevent catastrophic forgetting. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

Hung, C.-Y.; Tu, C.-H.; Wu, C.-E.; Chen, C.-H.; Chan, Y.-M.; and Chen, C.-S. 2019. Compacting, picking and growing for unforgetting continual learning. In *Advances in Neural Information Processing Systems*, 13669–13679.

Kemker, R.; McClure, M.; Abitino, A.; Hayes, T.; and Kanan, C. 2017. Measuring catastrophic forgetting in neural networks. *arXiv preprint arXiv:1708.02072*.

Kirkpatrick, J.; Pascanu, R.; Rabinowitz, N.; Veness, J.; Desjardins, G.; Rusu, A. A.; Milan, K.; Quan, J.; Ramalho, T.; Grabska-Barwinska, A.; et al. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences* 114(13):3521–3526.

Krizhevsky, A. 2009. Learning multiple layers of features from tiny images. 32–33.

Lesort, T.; Stoian, A.; and Filliat, D. 2020. Regularization shortcomings for continual learning.

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

Lopez-Paz, D., and Ranzato, M. 2017. Gradient episodic memory for continual learning. In *Advances in neural information processing systems*, 6467–6476.

Maltoni, D., and Lomonaco, V. 2019. Continuous learning in single-incremental-task scenarios. *Neural Networks*.

McCloskey, M., and Cohen, N. J. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of learning and motivation* 24:109–165.

Prabhu, A.; Torr, P.; and Dokania, P. 2020. Gdumb: A simple approach that questions our progress in continual learning. In *The European Conference on Computer Vision (ECCV)*.

Ratcliff, R. 1990. Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. *Psychological review* 97(2):285.

Rebuffi, S.; Kolesnikov, A.; Sperl, G.; and Lampert, C. H. 2017. icarl: Incremental classifier and representation learning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 5533–5542.

Rusu, A. A.; Rabinowitz, N. C.; Desjardins, G.; Soyer, H.; Kirkpatrick, J.; Kavukcuoglu, K.; Pascanu, R.; and Hadsell, R. 2016. Progressive neural networks. *arXiv preprint arXiv:1606.04671*.

Tan, M., and Le, Q. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, 6105–6114. PMLR.

Wu, Y.; Chen, Y.; Wang, L.; Ye, Y.; Liu, Z.; Guo, Y.; and Fu, Y. 2019. Large scale incremental learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 374–382.

Zenke, F.; Poole, B.; and Ganguli, S. 2017. Continual learning through synaptic intelligence. *Proceedings of machine learning research* 70:3987.