Understanding Capacity Saturation in Incremental Learning

Shenyang Huang^{†,*}, Vincent Francois-Lavet[‡], Guillaume Rabusseau[◇]

[†] Mila, McGill University, Montreal, Canada

[‡] VU Amsterdam, Amsterdam, Netherlands

[◇] DIRO - Mila, Université de Montréal, Montreal, Canada, Canada CIFAR AI chair

Abstract

In class-incremental learning, a model continuously learns from a sequential data stream in which new classes are introduced. There are two main challenges in class-incremental learning: catastrophic forgetting and capacity saturation. In this work, we focus on capacity saturation where a learner is unable to achieve good generalization due to its limited capacity. To understand how to increase model capacity, we present the continual architecture design problem where at any given step, a continual learner needs to adapt its architecture to achieve a good balance between performance, computational cost and memory limitations. To address this problem, we propose Continual Neural Architecture Search (CNAS) which takes advantage of the sequential nature of class-incremental learning to efficiently identify strong architectures. CNAS consists of a task network for image classification and a reinforcement learning agent as the meta-controller for architecture adaptation. We also accelerate learning by transferring weights from the previous learning step thus saving a large amount of computational resources. We evaluate CNAS on the CIFAR-100 dataset in several incremental learning scenarios with limited computational power (1 GPU). We empirically demonstrate that CNAS can mitigate capacity saturation and achieve performances comparable with full architecture search while being at least one order of magnitude more efficient.

Keywords: Continual learning and Neural Architecture Search and Deep Learning

1. Introduction

Many modern machine learning tasks are incremental in nature. For example, a product classification system might continuously encounter novel product categories over time. However, traditional classification models are designed such that after the training phase, no new classes can be added. In this work, we examine the key question of how to adapt the capabilities of existing models to learn novel concepts.

Continual learning, or lifelong learning [1] is the ability to acquire new knowledge while retaining previously learned experiences. Learning algorithms in that setting are referred to as continual learners. There are two main challenges for any continual learner: *catastrophic forgetting* and *capacity saturation*. First, *catastrophic forgetting* is where learning new information interferes with previously acquired knowledge [2], that is, the learner forgets how to perform old tasks when new ones are learned. In the continual learning literature, a limitation on the storage of past data is often enforced [3–8], and preventing catastrophic forgetting while having limited access to past data is the main focus of existing approaches.

Second, continual learners are also susceptible to the phenomenon of *capacity saturation*, where a fixed-capacity neural network is used to learn an indefinitely long sequence of tasks thus limiting its ability to generalize to newer tasks [9, 10]. There is limited literature discussing the problem of capacity saturation. To decouple the effect of capacity saturation from catastrophic forgetting, we examine a scenario where all past examples are used for future training and the impact of forgetting is minimized. This scenario is motivated by many modern applications where the memory required for preserving data samples is a minor

*corresponding shenyang.huang@mail.mcgill.ca

concern when compared to performance and computational efficiency. We also experiment with settings where the storage capacity is limited to various degrees.

In a practical setting, a continual learning must consider a constant three-way trade-off as seen in Figure 1 (right). As a continual learner is expected to be tested on a growing number of tasks, it is important to be efficient in (i) computational cost, (ii) size of the neural network (as well as memory or storage of past samples) and (iii) performance. To address the capacity saturation problem with the above trade-off in mind, we introduce the *continual architecture design* problem where at any given step, the learner must optimize its performance on all seen classes by selecting the most competitive neural architecture while being computationally efficient and respecting any given memory constraint. To address this problem, we propose Continual Neural Architecture Search (CNAS). CNAS consists of three parts: a task network for solving the classification task, a deep reinforcement learning based meta-controller for adaptively exploring the architecture search space and a heuristic function for deciding when to expand the continual learner. Each time new data arrives, the meta-controller generates candidate architectures that can be obtained using Net2Net [11] transformations of the current task network. The decision of whether to expand the current architecture is based on the performance of all the candidate architectures on a held-out dataset. This process allows the network structure to adaptively evolve in reaction to arrival of new classes or to other changes in the data distribution. By leveraging the Net2Net transfer learning technique, CNAS achieves significant computational savings as well as shortening the training time. In addition, the use of the reinforcement learning controller can dynamically adjust the architecture search space and restrict unnecessary expansions.

The autonomous nature of CNAS makes it an autoML approach [12, 13], offering an efficient and off-the-shelf learning system that avoids the tedious and costly task of manually selecting the correct neural architecture at each time step. As the observed dataset becomes more complex or includes examples from multiple training distributions, manually designing the architecture for a continual learner is not only time-consuming but also increasingly difficult. Therefore, reducing human intervention is a natural progression to develop robust and self-sufficient continual learners.

Summary of our contributions:

- As one of the first works to study capacity saturation, we empirically show that capacity saturation can have a strong detrimental impact on model performance when data from new classes are introduced.
- We introduce the continual architecture design problem, where at any given step, a continual learner needs to adapt its architecture to achieve a good trade-off between performance, computational efficiency and memory constraints.
- We propose a novel method, CNAS, which efficiently and adaptively increases the capacity of the continual learner as more tasks are introduced. CNAS takes advantage of transfer learning and reinforcement learning techniques to reuse trained weights and constrain the search space to achieve high computational savings.
- Our experiments on the CIFAR-100 dataset [14] show that CNAS constitutes a sound and promising approach to various class-incremental learning scenarios. In particular, CNAS achieves competitive performances when compared to a full architecture search while being orders of magnitude more efficient.

Reproducibility: The code is available at <https://github.com/shenyangHuang/CapacitySaturation>.

2. Related Work

Lopez-Paz and Ranzato [15] defined the goal of continual learning as learning a predictor $f : \mathcal{X} \times \mathcal{T} \rightarrow \mathcal{Y}$ where $\mathcal{T}$ refers to a set of task descriptors. This setting is known as

Figure 1. (left) the Venn diagram of continual learning with canonical references, (right) Continual learning methods balance a trade-off between performance, computational efficiency and memory limitation.

Table 1. Overview of how existing methods address capacity saturation

Method	Property	Minimize Forgetting	Width Growth	Depth Growth	Task-free
CNAS (ours)		✓	✓	✓	✓
iCaRL [4]		✓			✓
ER [17]		✓			
A-GEM [3]		✓			
APD [7]		✓			
RCL [6]		✓	✓		
DEN [5]		✓	✓		
CPG [18]		✓	✓		

task-incremental learning and the classical experimental setting uses an image classification dataset such as CIFAR100 [14] or MNIST [16] that is separated into N tasks (each containing k categories) [3, 6, 15]. In this work, we study class-incremental learning where no task descriptor is provided. Different continual learning settings are illustrated in Figure 1(left).

Chaudhry et al. [17] reported that a simple method which jointly trains on both examples from the current task as well as examples stored in a small episodic memory outperforms many continual learning methods. To study how CNAS performs under limited storage of past samples, we adapt the above experience replay (ER) method into the CNAS framework. In addition, CNAS can be easily adapted for many existing method designed specifically for solving catastrophic forgetting such as iCaRL [4].

Table 1 summarizes many recent continual learning approaches and how they address capacity saturation. In particular, approaches based on a fixed neural network structure [3, 4, 17] are prone to capacity saturation. Other approaches such as [5, 6, 18] only grows the architecture in width thus not leveraging the representational power that comes with increased network depth. In comparison to other methods in Table 1, CNAS is able to adapt the network structure in both width and depth, making it possible to efficiently address the capacity saturation problem. In addition, [5, 6, 18] only addressed capacity saturation in the task-incremental learning setting while CNAS is designed for the more challenging class-incremental learning setting.

Techniques for automatically designing deep neural networks using reinforcement learning (RL) agents have shown promising results. Methods such as Neural Architecture Search (NAS) [19] and Efficient Architecture Search (EAS) [20] use a policy gradient approach called REINFORCE [21], allowing for high flexibility in the policy network design. EAS further proposes to use Net2Net [11] transformations to initialize sampled architectures, thus achieving huge computational savings. In contrast with EAS, CNAS aims to solve the *continual architecture design* problem where the neural architecture should evolve naturally as the classification task increases in difficulty.

3. Preliminaries: Continual Learning

3.1. Class-incremental Learning

In the class-incremental learning setting, a model learns continuously from a sequential data stream in which new classes occur [4]. At any time step, the learner is required to perform multi-class classification for all classes observed so far. Formally, the goal of class-incremental learning is to learn, at each time step T , a classifier $f: \mathcal{X} \rightarrow \mathcal{Y}$ given the aggregation of the datasets seen up to now $(D_1, D_2, \dots, D_T)$, where each dataset $D_t = \{(x_1, y_1), \dots, (x_{n_t}, y_{n_t})\} \subset \mathcal{X} \times \mathcal{Y}$. Here, $\mathcal{X}$ is the input space and $\mathcal{Y} \subseteq \mathbb{N}$ is the set of categories. At each time step t , new classes can be introduced into the training data. Denoting by $\mathcal{Y}_t = \{y_i \mid (x_i, y_i) \in D_t\} \subset \mathcal{Y}$ the set of classes present in D_t , we assume that each dataset D_t is identically and independently drawn from the distribution $\mathcal{D}_{|Y \in \mathcal{Y}_t}$ where $\mathcal{D}$ is an unknown distribution over $\mathcal{X} \times \mathcal{Y}$ and $\mathcal{D}_{|Y \in \mathcal{Y}_t}$ denotes that $\mathcal{D}$ is conditioned on labels belonging to $\mathcal{Y}_t$. In this setting, the learning objective at time t corresponds to identifying an hypothesis f_t that minimizes the risk over the classes seen so far:

$$f_t = \arg \min_f \mathbb{E}_{(X,Y) \sim \mathcal{D}_{|Y \in \mathcal{Y}_1 \cup \dots \cup \mathcal{Y}_t}} \mathcal{L}(f(X), Y), \quad (3.1)$$

where $\mathcal{L}$ is a loss function penalizing prediction errors over the random variables $(f(X), Y)$. The common scenario for class-incremental learning is the one where k new classes are introduced at each time step which we refer to as *k-class incremental learning*. Similar to [4], after receiving a batch of data from k new classes, the model calls an update routine to update its internal knowledge.

In accordance with the learning objective defined in Eq. (3.1), a natural metric to evaluate the performance of a model at test time is the *average incremental accuracy* introduced in [4]. The average incremental accuracy at time step t is the test accuracy of the model f_t on the part of the test data consisting only of the classes seen up to time t :

$$\text{Average Incremental Accuracy} = \frac{1}{C} \sum_{i=1}^C R_i, \quad (3.2)$$

where $C = |\mathcal{Y}_1 \cup \dots \cup \mathcal{Y}_t|$ is the total number of classes seen until time t and R_i is the test accuracy of the model f_t on category i discriminating from C classes.

3.2. Continual Architecture Design

In this work, we define *continual architecture search* as the setting where, at each time step t , the continual learner must select the best neural architecture for classifying all classes seen so far. We impose a constraint with practical settings in mind: *the initial architecture at $t = 1$ is selected based on the initial dataset D_1 only*. Continual architecture search is concerned with hyperparameter optimization on a growing dataset while architecture search is traditionally conducted on a fixed training distribution. This difference implies that the architecture search space is continually growing thus making exhaustive search methods (such as grid search) intractable from a computational standpoint. More concretely, one should consider the trade-off outlined in Figure 1(right). As a continual learner is always acquiring new knowledge, the previously used architecture and learned weights should be transferable to the next time step, otherwise the computational cost and memory requirement could be unbounded. In this work, CNAS takes advantage of the sequential nature of class-incremental learning by (i) limiting the architecture search space by considering the structure of the task network from the previous step as a starting point and (ii) using Net2Net techniques to rapidly transfer weights from the previous step.

Algorithm 1: CNAS IncrementalLearn

Input: Past data $D_{|Y \in \mathcal{Y}_1 \cup \dots \cup \mathcal{Y}_{t-1}}$, new data $D_{|Y \in \mathcal{Y}_t}$, task network β_{t-1} .

Output: New task network β_t

Concatenate $D_{|Y \in \mathcal{Y}_1 \cup \dots \cup \mathcal{Y}_{t-1}}$ with $D_{|Y \in \mathcal{Y}_t}$;

Expand output dimension for β_{t-1} ;

Train network β_{t-1} on $D_{|Y \in \mathcal{Y}_1 \cup \dots \cup \mathcal{Y}_t}$ and let v_{t-1} be its validation accuracy ;

$(\beta_*, V_{sampled}) \leftarrow \text{ArchSearch}(\beta_{t-1}, D_{|Y \in \mathcal{Y}_1 \cup \dots \cup \mathcal{Y}_t})$ (See Algorithm 2);

$(\beta_t) \leftarrow \text{HeuristicFunc}(\beta_{t-1}, \beta_*, v_{t-1}, V_{sampled})$ (See Algorithm 3);

Train β_t with $D_{|Y \in \mathcal{Y}_1 \cup \dots \cup \mathcal{Y}_t}$;

Return β_t ;

4. Continual Neural Architecture Search (CNAS)

In this section, we present our proposed method: Continual Neural Architecture Search (CNAS). At any given time step t , CNAS provides a deep neural network with trained weights that is able to classify all observed categories so far. There are three components: a *task network*, a *meta-controller* and a *heuristic function*.

The task network performs classification for all observed classes and is implemented as a standard deep neural network with convolution (CNN) [22], maxpooling, dropout [23] and fully-connected layers. At each time step, the number of neurons in the last layer of the task network is equal to the number of observed classes C , and through the softmax activation function, each output neuron predicts the conditional probability of a category given the input. In class-incremental learning, new neurons are added to the output layer each time new categories appear (these neurons are initialized with a zero-mean normal distribution for the weight matrix and zero for the bias term).

The meta-controller is specialized in generating an architecture search policy to sample new candidate architectures for the task network when new classes arrive. The controller is designed to be a deep reinforcement learning agent. The role of the meta-controller is only to guide the architecture sampling process, by selecting promising architectures to try out based on experiences gathered from previous time steps. The selection of the best architecture out of the sampled ones is based on a validation set. This can be seen as a one-step ahead planning guided by the meta-controller to explore good candidates.

Lastly, the heuristic function considers the validation performance of all the sampled architectures and decides if an expansion is beneficial in the current step. Preventing unnecessary expansions will reduce the computational time in subsequent steps as well as increase the parameter efficiency of the task network.

4.1. Training Procedure

Algorithm 1 describes the training procedure for CNAS when a new dataset arrives. The task network is first trained with a combined dataset of new samples and samples stored in memory and is then used as the starting point for *ArchSearch* (Algorithm 2). *ArchSearch* then outputs the validation accuracies of all the sampled architectures and the best performing candidate architecture. *HeuristicFunc* (Algorithm 3) then decides if expanding the current task network is beneficial based on the validation performance differences between the sampled candidate architectures and the existing architecture. If no expansion is needed, the current architecture is kept, otherwise the best performing sampled architecture becomes the new task network structure. This new task network is then further trained on the available data to ensure it has converged. The number of candidate architectures that

Algorithm 2: CNAS ArchSearch

Hyper-parameter: Sample size n . Epoch limit l .

Input: Dataset $D_{|Y \in \mathcal{Y}_1 \cup \dots \cup \mathcal{Y}_t}$. Current architecture β_{t-1} .

Output: Best performing architecture β_* . List of validation accuracy of sampled architectures $V_{sampled}$.

for $i = 1, \dots, n$ **do**

 Generate candidate architecture $\hat{\beta}^i$ from RL agent ;

 Initialize $\hat{\beta}^i$ by weights from β_{t-1} using Net2Net;

 Train $\hat{\beta}^i$ for l epochs with $D_{|Y \in \mathcal{Y}_1 \cup \dots \cup \mathcal{Y}_t}$;

 Append validation accuracy $\hat{v}_i$ of $\hat{\beta}^i$ to $V_{sampled}$;

end

Update RL agent with all $(\hat{\beta}^i, \hat{v}_i)$ pairs;

// Return the candidate architecture with best accuracy:

Let $\beta_* = \hat{\beta}^j$ where $j = \arg \max_{i=1, \dots, n} \{\hat{v}_i\}$;

Return $(\beta_*, V_{sampled})$;

Algorithm 3: CNAS HeuristicFunc

Input: Current architecture β_{t-1} . Best performing sampled architecture β_* .

Performance of current architecture v_{t-1} . Performance of sampled architectures $V_{sampled}$.

Output: New task network architecture β_t

// Calculate number of negative improvements:

$N_{negative} = |\{v \in V_{sampled} \mid v < v_{t-1}\}|$;

if $N_{negative} < \frac{|V_{sampled}|}{2}$ and $\text{mean}(V_{sampled}) > v_{t-1}$

 Return β_* ; // Architecture is expanded.

else

 Return β_{t-1} ; // No expansion.

end

can be sampled per time step is a hyper-parameter of our algorithm and it controls the trade-off between computational complexity and exploration depth.

One could greedily expand the continual learner at each time step (i.e. always set β_t to β_* in Algorithm 1). However, such strategy would lead to rapid over-parametrization thus causing increased computational cost as well as additional memory space required to store a large model. To avoid such greedy expansion, the heuristic function (*HeuristicFunc*, see Algorithm 3) is designed to evaluate the benefit of expansion based on the difference in validation performance between all sampled architectures and the existing architecture. If capacity saturation occurs, it is important to expand the architecture and the added capacity will result in performance gain. However, when only a small portion of expanded structures shows gains in performance then it is likely that these improvements are due to the randomness in network training and architecture expansion is not required.

4.2. Net2Net Transformations

To save the computational cost of training each sampled architecture from scratch, we use a transfer learning technique called Net2Net [11]. Net2Net enables a rapid transfer of information from one neural network to another by expanding/creating fully-connected and

convolutional layers using two types of operations. Net2WiderNet operations replace a given layer by a wider one (more units for fully-connected layers or more filters for convolutional layers) while preserving the function computed by the network. Net2DeeperNet operations insert a new layer that is initialized as an identity mapping between two existing layers, thus preserving the function computed by the neural network. More formally, Net2DeeperNet replaces a layer $h^{(i)} = \phi(W^{(i)}h^{(i-1)})$ with two layers $h^{(i)} = \phi(I\phi(W^{(i)}h^{(i-1)}))$ where I is the identity matrix. However, the last equality is true only if the activation function ϕ is such that $\phi(I\phi(v)) = \phi(v)$ for all vectors v , which holds for the rectified linear activation (ReLU). Therefore, we use ReLU activation for all hidden layers.

Net2WiderNet and Net2DeeperNet operations can be applied sequentially to grow the original network in both width and depth. In this way, any architecture that is strictly larger than the original can preserve the function computed by the original network. This allows CNAS to use a trained network as a starting point for architecture search and quickly initialize new larger architectures. By using Net2Net, the capacity of the task network can be expanded efficiently and dynamically for better performance as new data become available. Further details regarding Net2Net transformations are provided in the original paper [11].

5. Reinforcement Learning Agent

We use the policy gradient method REINFORCE [21] and design two independent policy networks for respectively taking Net2WiderNet and Net2DeeperNet actions, with the simplifying assumption that they are independent.

We describe continual architecture design as an RL problem: at each step, an agent observes the current state s_t of the environment and samples actions (=network transformations) a_t according to a stochastic policy $\pi(a_t|s_t)$. For each sampled action, it observes a reward signal r_t , which is used along with a step size α to improve the policy for future time steps. For computational efficiency, a fixed number of architectures is sampled at each time step using the RL agent. The planning horizon is limited to one time step. Limiting the horizon acts as a complexity control method [24] and results in only optimizing for the current distribution. The REINFORCE algorithm is simplified to:

$$\theta \leftarrow \theta + \alpha r_t \nabla_{\theta} \ln \pi(a_t|s_t, \theta) \quad (5.1)$$

where θ represents the parameters of the policy networks for Net2WiderNet and Net2DeeperNet.

The architecture search for each time step is summarized in Algorithm 2. Any sampled architecture is trained for at most l epochs using early stopping. Due to the benefit of weight transfer through Net2Net transformations, sampled architectures only require training for a low number of epochs in practice.

5.1. Policy Networks

The policy networks for Net2WiderNet and Net2DeeperNet, referred to as wider actor and deeper actor respectively, are identical in design as seen in Figure 2, but trained independently. Encoding the task network’s architecture in details into the state s_t might be of little use to the RL agent as such states are almost never repeated (since the architecture is continuously expanding). Therefore, we only include the number of convolutional layers and the number of fully-connected layers of the task network in s_t (denoted by A_{conv} and A_{fc} respectively). Moreover, to measure the disparity between the current training distribution D_t and the previous one D_{t-1} , the difference in validation accuracy of the task network on these two distributions is included in the state space (denoted by V_{diff}). Lastly, the number of new classes received by the continual learner at the current time step is also added (denoted by N_{new}).

Figure 2. Flow chart of the policy network

The wider and deeper actors decide the number of widen and deepen transformations to take respectively, and are implemented as multilayer perceptrons. Both the input and hidden layers have ReLU activations while the output layer of the actor networks has a softmax activation. The i -th output neuron corresponds to the probability of taking $(i - 1)$ transformations and the first neuron always represents not taking any transformation. The predicted probability is then used as input to a categorical distribution out of which actions are selected. In this way, with the same input state space, the number of transformations selected by the actor networks is stochastic.

5.2. Reward Design

To best decide the number of transformations needed for each time step, we design a reward function based on the performance of the newly transformed architecture, compared to the existing one (measured with average incremental accuracy from Equation 3.2). We consider the difference in validation accuracy between the original architecture and the sampled architecture, $r_t = v' - v_{t-1}$. Here r_t is the reward signal given to the agent at time step t after deciding on the number of Net2Net transformations while v' and v_{t-1} respectively stands for the validation accuracy of the sampled architecture and of the original architecture on the current dataset after training. Therefore, any architecture that performs worse than the original will provide a negative reward signal while a better architecture will yield a positive one. To obtain better reward signals for learning, the rewards are normalized into $[-1, 1]$ range within all architectures sampled at time t . Lastly, we add an entropy term to the reward function in order to improve policy optimization [25].

6. Experiments

In this section, we conduct experiments to evaluate CNAS in various incremental learning settings. We repeat each experiment three times with different random seeds and report the standard deviation with error bars.

Dataset We split both the training set and test set of CIFAR-100 by class labels. The CIFAR-100 dataset contains a total of 60,000 images across 100 classes. In our experiments, each class is further split into 450 images as training set, 50 images as validation set and 100 images as test set. When a new class is introduced, all corresponding test data will start to be used for the calculation of average incremental accuracy. The arrival order of the classes is based on the default labels given by the CIFAR-100 dataset. Each experiment starts with some initial classes (known as the base knowledge and considered as the dataset for time step 0). When the memory size of a continual learner is constrained, the same number of examples as the memory size is set aside for early stopping and architecture selection.

Methods We compare CNAS and its variants with the following baselines:

- (1) *CNAS all data*: CNAS with a rehearsal of all previously seen data.

Figure 3. Average Incremental Accuracy (left) and number of parameters (right) for the capacity saturation experiment. CNAS can mitigate capacity saturation to a large extent by adaptively growing the current architecture based on observed data.

- (2) *CNAS memory k* : CNAS with k number of past samples allowed in memory. We follow the same training procedure as ER [17].
- (3) *SA all data*: a continual learner using the starting architecture of CNAS as a static architecture and weights are carried over from one step to another while training is conducted using all past data.
- (4) *SA memory k* : a continual learner with k number of past samples allowed in memory. This is the same as ER [17].
- (5) *full archSearch*: models obtained by completing a full architecture search on 200 randomly sampled convolutional architectures. We only conduct search using the initial 10 classes, 20 classes and 30 classes respectively and identify the best candidate architectures. We only include three data points as these experiment are computationally expensive to complete.

Note that for all methods, new neurons are added to the output layer each time new categories appear (as discussed in Section 4).

Implementation All models are trained using the ADAM [26] optimizer with a learning rate of 10^{-4} and other parameters set to default values. Training with all past data is conducted with mini-batch size 128 and training with ER [17] is conducted with mini-batch size 16. The task network is trained until convergence (with early stopping) both before and after the architecture search at each time step. Both the wider and deeper actors are implemented as a multilayer perceptron with 2 hidden layers, each having 128 neurons. The learning rate for RL agent is 0.001 and the entropy regularization term is multiplied by 0.01.

6.1. Capacity Saturation

To empirically understand the capacity saturation problem, we select a starting architecture with a small number of parameters and compare it with the best architecture found through full architecture search on the initial 10 classes. In this experiment, CNAS samples 20 architectures at each time step and can take at most 3 Net2WiderNet and 3 Net2DeeperNet actions. The continual learners start with a base knowledge of two image classes and then incrementally learns two new classes at each time step.

In Figure 3(left), “*SA all data(small arch)*” (resp. “*SA all data(large arch)*”) corresponds to the continual learner using the small starting architecture (resp. the starting architecture found by full architecture search) as a static architecture. Even when the effect of catastrophic forgetting is minimized through full rehearsal, capacity saturation has a direct impact on the performance of the model especially as more classes are introduced. CNAS starts with the same small architecture but is able to mitigate capacity saturation to a large extent and exhibit greatly improved performances as compared to “*SA all data(small arch)*”. Figure 3(right) shows that CNAS adaptively expands the architecture based on current data thus performing closer to “*SA all data(large arch)*” than “*SA all data(small arch)*”.

Figure 4. Performance comparison in 2-class (left) and 10-class (right) incremental learning

Figure 5. Parameter growth curve in 10-class incremental learning

6.2. k -class incremental

We compare the performances of CNAS and SA in k -class incremental learning experiments on CIFAR-100 for $k = 2$ and $k = 10$. For 2-class incremental learning, CNAS samples 20 architectures at each time step and can take at most 3 Net2WiderNet and 3 Net2DeeperNet actions. For 10-class incremental learning, 50 architectures are sampled at each time step and at most 10 Net2WiderNet and 5 Net2DeeperNet transformations can be taken. Both experiments start with a base knowledge of 10 classes and the initial architecture is selected through a full architecture search.

From Figure 4, we can see that when all past data is available, CNAS performs slightly better than SA in terms of average incremental accuracy and even achieves comparable performance to full architecture search. In Figure 5, there are steps where CNAS chooses not to expand and maintains its architecture as the heuristics function ensures that only beneficial expansions are taken.

When enough memory is available, CNAS performs better than SA as there are more validation samples available for selecting architectures. Figure 4 shows that the effect of catastrophic forgetting becomes more severe as memory size decreases. Capacity saturation might have less impact on performance as many neurons are under-utilized due to forgetting.

Computational Time For all experiment, a single TITAN Xp GPU is used and we report the average process time. Figure 6 shows the cumulative computational time in 10-class incremental experiment and 2-class incremental experiment. CNAS’s computational cost grows significantly slower than full architecture search. Based on the observed trend, using full search at each step is at least one order of magnitude more computationally expensive than CNAS after learning 100 classes. This difference in efficiency is more drastic when there are more time steps such as in 2-class incremental learning. We also observe that compute time is longer as more memory space is available (more samples in training).

7. Discussion

As seen from Section 6, capacity saturation can have a negative impact on model performance. In particular, in applications where parameter efficiency is essential, it is important

Figure 6. Computational time in 2-class (left) and 10-class (right) incremental experiment

to solve the continual architecture design problem and identify the most efficient architecture with the best possible performance. From Figure 3 (right), CNAS expands the number of model parameters to approximately match the one found by full architecture search.

CNAS requires very few hyper-parameters to be effective. The starting architecture of CNAS is optimized on the base knowledge (training data at time step 0). The number of sampled architectures and the maximum number of Net2Net transformations are selected based on the available computational resources. Given a larger search space and more sampled architectures, CNAS is likely to find stronger models.

CNAS avoids greedily picking the architecture with the best validation performance, which can quickly lead to overparametrized models. The validation accuracy of any sampled architecture is not only determined by the effectiveness of the architecture but also affected by the stochasticity of parameters optimization. As limited past examples are reused, the lack of diversity in validation samples also increases the variance of validation accuracy.

8. Conclusion

In this paper, we studied capacity saturation in class-incremental learning and introduced the continual architecture design problem which requires expanding architectures to avoid capacity saturation. To tackle this problem, we proposed CNAS, an efficient and economical autoML approach for continual learning. CNAS (i) reuses trained weights through Net2Net, (ii) implements an RL meta-controller to adjust the architecture space and (iii) uses a heuristic function to decide when to expand the architecture. We demonstrate that CNAS can mitigate capacity saturation to a large extent. In addition, when all past data is available, we show that CNAS achieves comparable performance to a full architecture search while being at least one order of magnitude more computationally efficient.

Acknowledgements

This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Canadian Institute for Advanced Research (CIFAR).

References

- [1] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter. “Continual lifelong learning with neural networks: A review”. In: *Neural Networks* (2019).
- [2] M. McCloskey and N. J. Cohen. “Catastrophic interference in connectionist networks: The sequential learning problem”. In: *Psychology of learning and motivation*. Vol. 24. Elsevier, 1989, pp. 109–165.
- [3] A. Chaudhry, M. Ranzato, M. Rohrbach, and M. Elhoseiny. “Efficient lifelong learning with a-gem”. In: *arXiv preprint arXiv:1812.00420* (2018).
- [4] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert. “icarl: Incremental classifier and representation learning”. In: *Proceedings of CVPR*. 2017, pp. 2001–2010.

- [5] J. Yoon, E. Yang, J. Lee, and S. J. Hwang. “Lifelong learning with dynamically expandable networks”. In: *arXiv preprint arXiv:1708.01547* (2017).
- [6] J. Xu and Z. Zhu. “Reinforced continual learning”. In: *NeurIPS*. 2018, pp. 899–908.
- [7] J. Yoon, S. Kim, E. Yang, and S. J. Hwang. “ORACLE: Order Robust Adaptive Continual LEarning”. In: *arXiv preprint arXiv:1902.09432* (2019).
- [8] P. Pan, S. Swaroop, A. Immer, R. Eschenhagen, R. E. Turner, and M. E. Khan. “Continual deep learning by functional regularisation of memorable past”. In: *arXiv preprint arXiv:2004.14070* (2020).
- [9] S. Sodhani, S. Chandar, and Y. Bengio. “On Training Recurrent Neural Networks for Lifelong Learning”. In: *arXiv preprint arXiv:1811.07017* (2018).
- [10] S. Lee, J. Ha, D. Zhang, and G. Kim. “A Neural Dirichlet Process Mixture Model for Task-Free Continual Learning”. In: *arXiv preprint arXiv:2001.00689* (2020).
- [11] T. Chen, I. J. Goodfellow, and J. Shlens. “Net2Net: Accelerating Learning via Knowledge Transfer”. In: *ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. 2016. URL: <http://arxiv.org/abs/1511.05641>.
- [12] H. Mendoza, A. Klein, M. Feurer, J. T. Springenberg, and F. Hutter. “Towards Automatically-Tuned Neural Networks”. In: *Proceedings of the Workshop on Automatic Machine Learning*. Ed. by F. Hutter, L. Kotthoff, and J. Vanschoren. Vol. 64. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, 2016, pp. 58–65.
- [13] M. Feurer, A. Klein, K. Eggenberger, J. Springenberg, M. Blum, and F. Hutter. “Efficient and robust automated machine learning”. In: *NeurIPS*. 2015, pp. 2962–2970.
- [14] A. Krizhevsky, G. Hinton, et al. *Learning multiple layers of features from tiny images*. Tech. rep. Citeseer, 2009.
- [15] D. Lopez-Paz et al. “Gradient episodic memory for continual learning”. In: *NeurIPS*. 2017, pp. 6467–6476.
- [16] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.
- [17] A. Chaudhry, M. Rohrbach, M. Elhoseiny, T. Ajanthan, P. K. Dokania, P. H. S. Torr, and M. Ranzato. “On Tiny Episodic Memories in Continual Learning”. In: (2019). arXiv: [1902.10486](https://arxiv.org/abs/1902.10486) [cs.LG].
- [18] C.-Y. Hung, C.-H. Tu, C.-E. Wu, C.-H. Chen, Y.-M. Chan, and C.-S. Chen. “Compacting, Picking and Growing for Unforgetting Continual Learning”. In: *NeurIPS*. 2019, pp. 13647–13657.
- [19] B. Zoph and Q. V. Le. “Neural architecture search with reinforcement learning”. In: *arXiv preprint arXiv:1611.01578* (2016).
- [20] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang. “Efficient architecture search by network transformation”. In: *AAAI*. 2018.
- [21] R. J. Williams. “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. In: *Machine learning* 8.3-4 (1992), pp. 229–256.
- [22] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.
- [23] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. “Dropout: a simple way to prevent neural networks from overfitting”. In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.
- [24] N. Jiang, A. Kulesza, S. Singh, and R. Lewis. “The dependence of effective planning horizon on model accuracy”. In: *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems. 2015, pp. 1181–1189.
- [25] Z. Ahmed, N. Le Roux, M. Norouzi, and D. Schuurmans. “Understanding the impact of entropy on policy optimization”. In: *ICML*. 2019, pp. 151–160.
- [26] D. P. Kingma and J. Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).