Continual Learning on CLVision-Challenge

Haoran Zhu, Hua Sun
Department of Electrical and Computer Engineering
New York University
{hz1922, hs4062}@nyu.edu

Abstract

Continual learning is a crucial ability for artificial intelligence. It aims at allowing an agent to learn new tasks without forgetting the knowledge of previous tasks. In real life applications(e.g., autonomous driving cars, robotic applications), due to limited computational and storage resources, it is almost impossible to fully retrain models every time new tasks and new data become available. In this work, we take part in CVPR 2020 Workshop on Continual Learning in Computer Vision Challenge. We reproduce several important continual learning algorithms and apply them on a real life dataset, CORe50, which is a new dataset and benchmark specially designed for continuous object recognition scenarios.

1 Introduction

In today's Deep Learning scheme, the presumption of the training is that all instances of the classes are available, the model will iterate the dataset per epoch to learn the knowledge. This means if there are new instances added to the dataset, we have to retrain the whole model on the whole dataset plus the new instances. This has caused trouble in some scenarios. Specifically, in image classification problems, the pretrained model often encounters new objects or the dataset can be expanded. However, the existence of *Catastrophic Forgetting*(7), i.e. the newly learned parameters will shift the old parameters and weaken its performance on old task, has brought challenge to this problem. This phenomenon can greatly degrade the performance of the model or even totally rewrite the model's parameters, causing the old knowledge being totally forgot. In before, in face of such case, one have to choose either retrain the model on entire dataset or just tolerate such degrading issues. To overcome the dilemma, the concept of continual learning(Lifelong Learning(11)) has emerged. In continual learning, it is required that the model must have not only the ability to acquire new knowledge, but also prevent the novel input to overwhelm the original data.

Continual learning is close to the concept of online learning. But in online learning setting, there will be some dependency on the previous data, while in continual learning, we do not want to access the original dataset because it is costly and time consuming. The focus of continual learning is not only on maintaining the accuracy, but also on training efficiently. Currently, there are three main approaches to apply continual learning,

1. **Retraining** Totally retraining the old parameters θ_o on the new task to obtain the new model θ_n, and then apply regularization to prevent the degrading problem on old task. The typical work in this approach are *Learning without Forgetting*(LwF)(4) and Elastic Weight Consolidation(EWC)(3).
2. **Expansion** Non-Retraining but instead expanding the network. Whenever encounter the new task, just freeze the old weight and expand the network. One of the typical work is *Progressive Network*(9). Another variation of this work is masking, i.e. they focus not only on the weigth itsely, but also considered masking some of the unimportant weights for the new task t. Packenet(6) and Piggyback(5) adopt this approach.

32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

3. **Partial Retraining with Expansion** Another approach is selectively retraining the old network, expanding the capacity when necessary and train dynamically on the optimal solution. One of the outstanding work in this approach is *Dynamically Expandable Networks*(DEN)(12).

In this project, we focus on the *CLVision Challenge*[1], i.e. Continual Learning in Computer Vision, a problem set published by CVPR 2020 Workshop to solve the image classification problem in continual learning. We have reproduced and refined several famous approaches, including *EWC* and *Piggyback*, and tried out them on the CLVision dataset, then we evaluate the performance and analyze the potential improvements that can be done. In Section 2, we have introduced some outstanding previous work on Continual Learning. In Section 3, we introduce the *CLVision Challenge* dataset, and the principle and design of the approaches we use in this project. In Section 4, we present the result of our model and analyze our result. In Section 5, we conclude our work, and propose some potential work that can be done on this topic.

2 Related Work

Continual Learning, also known as Lifelong Learning, is defined to be an algorithm which is capable of extracting knowledge from a continuous stream of data without forgetting the old knowledge, i.e. catastrophic forgetting(8).

PlaceHolder

3 Dataset and Experiment Design

In this section, we introduce the dataset *CORe50* realeased by CVPR Workshop. Their dataset consist of three parts, and due to the constraint of time, we only consider the first two part of the three challenges. Then, we select two representative approaches for Continual Learning, EWC and Piggyback, and introduce the theorem and some core details about them.

3.1 Dataset and Tasks

CORe50 is a new dataset specially designed for *(C)ontinual (O)bject (Re)cognition*. Unlike *permuted MNIST* where new tasks to learn are obtained by simply scrambling the pixel positions, *CORe50* is much more complex and a real life dataset. Datasets such as *ImageNet* and *Pascal VOC* provide a good playground for image classification and detection, but they have been designed with "static" evaluation and lack of multiple views of the same objects taken into different sessions, *CORe50* solves the above problems and meets the requirement for continuous learning scenarios on computer vision.

It consists 50 domestic objects belonging to 10 categories. The dataset is separated into 11 distinct sessions (8 indoors and 3 outdoors) with different background and lightning. For each session and for each object, a 15 seconds video (at 20 fps) has been recorded with a Kinect 2.0 sensor delivering 300 RGB-D frames.

There are three tasks in CVPR 2020 Workshop *CLVision Challenge*:

- **New Classes(NC)**: New training patterns belonging to different classes become available in subsequent batches. In this case the model should be able to deal with the new classes without losing accuracy on the previous ones.
- **New Instances(NI)**: New training patterns of the same classes become available in subsequent batches with new poses and conditions (illumination, background, occlusion, etc.). A good model is expected to incrementally consolidate its knowledge about the known classes without compromising what it has learned before.
- **New Instances and Classes(NIC)**: New training patterns belonging both to known and new classes become available in subsequent training batches. A good model is expected to consolidate its knowledge about the known classes and to learn the new ones.

[1] https://sites.google.com/view/clvision2020/challenge?authuser=0

Figure 1: Example images of the 50 objects in CORe50. Each column denotes one of the 10 categories.

For each task, the input image is $128 \times 128 \times 3$. For case NC and NI, each task batch contains 10k-20k images, which makes it a large dataset.

3.2 EWC

In brains, synaptic consolidation enables continual learning by reducing the plasticity of synapses that are vital to previously learned tasks. *EWC* is an algorithm that performs a similar operation in artificial neural networks by constraining important parameters to stay close to their old values. *EWC* can be used in supervised learning and reinforcement learning problems to train several tasks sequentially without forgetting older ones.

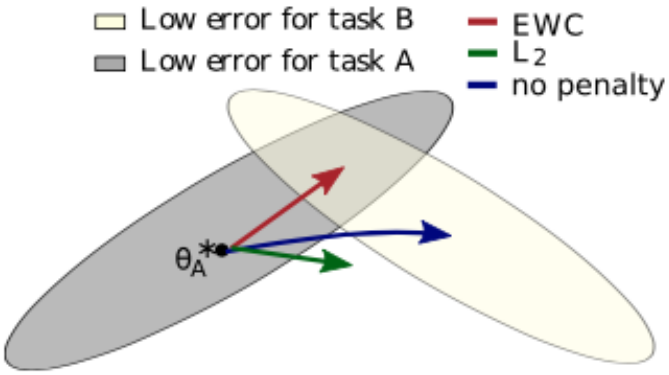

Figure 2: By applying *EWC*, the model of the weights can learn the new task while still maintain in the range of solving the old task. (Red arrow)

3.2.1 Theorem

From a probabilistic perspective: optimizing the parameters is tantamount to finding their most probable values given some data D. We can compute this conditional probability $p(\theta|\mathcal{D})$ from the prior probability of the parameters $p(\theta)$ and the probability of the data $p(\mathcal{D}|\theta)$ by using Bayes' rule:

$$\log p(\theta|\mathcal{D}) = \log p\left(\mathcal{D}_B|\theta\right) + \log p\left(\theta|\mathcal{D}_A\right) - \log p\left(\mathcal{D}_B\right) \tag{1}$$

Similarly we can apply Bayes' rule on two continuous tasks A and B:

$$\log p(\theta|\mathcal{D}) = \log p\left(\mathcal{D}_B|\theta\right) + \log p\left(\theta|\mathcal{D}_A\right) - \log p\left(\mathcal{D}_B\right) \tag{2}$$

While $\log p(\mathcal{D}_B|\theta)$ is negative training loss for task B, we can optimize this by gradient descent. $\log p(\mathcal{D}_B)$ is a constant, we don't need to care about it when optimizing for the overall goal. $\log p(\theta|\mathcal{D}_A)$'s real value is intractable and will be approximated by Laplace approximation. The approximated value can further be the derivation regarding approximated mean value and its corresponding Hessian matrix. Noted that the expected value of Hessian matrix is the negative value of Fisher information matrix F, thus the original goal of maximizing $\log p(\theta|\mathcal{D})$ is equivalent to minimizing the following loss function:

$$\mathcal{L}(\theta) = \mathcal{L}_B(\theta) + \sum_i \frac{\lambda}{2} F_i \left(\theta_i - \theta^*_{A,i}\right)^2 \tag{3}$$

where $\mathcal{L}_\mathcal{B}(\theta)$ is the loss for task B only and hyperparameter λ sets how important the old task is compared to the new task. When the third task comes, you can treat all previous tasks as the old task and repeat the above process. In the end, it can ensure the model can learn new tasks without changing the weights for the previous tasks too much.

3.2.2 Experiment Setting

After several experiments, we set the hyperparameter λ = 100, which we think it is an appropriate value of how important the old task is to the new task. The best results are produced by SGD as the optimizer.

3.3 Piggyback

Piggyback is a upgrading version of Packnet, it uses pure mask layer to mask weights for new task. In this part, we introduce the core idea and the mathemetical representation of Piggyback. We also demonstrate some of the techniques used in training for Piggyback.

3.3.1 Theorem

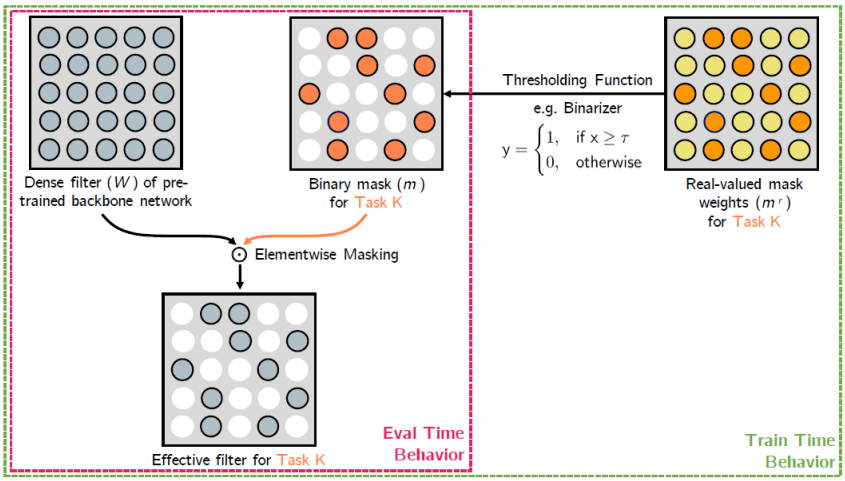

Figure 3: Piggyback Overview

Piggyback adopts masking weight for new task to overcome catastrophic forgetting. The core idea is to selectively mask the fixed weights of a base network, and then use this mask to predict the new task. That is, given model $\mathbf{W}$, whenever there is a new task streaming in, the model will initialize and optimize a binary value matrix $\mathbf{m}$, and use this mask to generate weight $\mathbf{W}^\star$ to be used in prediction for this specific new task. The gradient that backpropagates to the layer will not affect $\mathbf{W}$ but $\mathbf{m}$. In

this way, for a given task, we can obtain filters that is consist of 0/1. For example, a dense weight vector [0.1, 0.2, 0.3, 0.4] could be filtered to [0, 0.2, 0.3, 0] after the binary masking. In short, we does not learn what are the 'right' parameters but learn what is not the 'right' parameters. Therefore, the choice of backbone network is crucial to the performance of piggyback because if the original weights are malfunctioning, pure masking cannot greatly improve the accuracy.

The structure of *Piggyback* is shown in Figure 3. To explain the procedure for training, we consider an end to end fully-connected layer case. Denote $\mathbf{x} = [x_1, x_2, ..., x_m]$ as input, and $\mathbf{y} = [y_1, y_2, ..., y_n]$ as output. The weight matrix for this layer is $\mathbf{W}^{n \times m}$. Without loss of genrality, we can simply assume $\mathbf{y} = \mathbf{W}\mathbf{x}$ by ignoring the bias term. Suppose the loss function is L, the normal backpropagation equation for $\mathbf{W}$ is,

$$\begin{aligned} \delta w_{ji} = \frac{\partial L}{\partial w_{ji}} &= (\frac{\partial L}{\partial y_j}) \cdot (\frac{\partial y_j}{\partial w_{ji}}) \\ &= \delta y_j \cdot x_j \\ \therefore \delta \mathbf{W} &= \delta \mathbf{y} \cdot \mathbf{x^T} \end{aligned} \tag{4}$$

In *Piggyback*, the author has introduced a real value mask matrix $\mathbf{M_r}^{n \times m}$ and a manually set threshold τ. M_r is used to create the binary mask matrix $\mathbf{M} = [m_{ji}]$. We can obtain m_{ji} by,

$$m_{ji} = \begin{cases} 1, & \text{if } m^r_{ji} > \tau \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

Then, for $y_j \in \mathbf{y}$, it gives $y_j = \sum_{i=1}^{m} w_{ji} \cdot m_{ji} \cdot x_i$. The backpropagation equation is used to update the real value matrix M_r. That is, during the whole training procedure, the weight matrix $\mathbf{W}$ is fixed as constant. In *Piggyback*, the modified mask weigths updates as follows. Here, $A \odot B = C = [c_{ji} = A_{ji}B_{ji}]$.

$$\begin{aligned} \delta m_{ji} = \frac{\partial L}{\partial m_{ji}} &= (\frac{\partial L}{\partial y_j}) \cdot (\frac{\partial y_j}{\partial m_{ji}}) \\ &= \delta y_j \cdot w_{ji} \cdot x_j \\ \therefore \delta \mathbf{m} &= (\delta \mathbf{y} \cdot x^T) \odot \mathbf{W} \end{aligned} \tag{6}$$

3.3.2 Experiment Setting

In practical, it is hard to derive the analytical solution to threshold τ. In the original work, the author set $\tau = 5e - 3$. As for the matrix $\mathbf{m_r}$, the author initialized the value to 0.01. The best results are produced by Adam optimizer. In our experiment, we have inherited the parameter settings in the work. However, as mentioned in the previous statement, the crucial part for *Piggyback* is that the base model have to be fine-tuned, otherwise the model will not work very well. We have also verified such phenomon in our experiment. To improve the performance, we modified the logic of the model. At the first task, we use the pure base model to perform the training, as the pre-trained network(ResNet50(1), in our case)cannot perform well on the given dataset. Then we begin performing mask and *Piggyback*. This part will later be discussed in detail in Section 4.

4 Evaluation and Analyze

In this part, we evaluate the performance of *EWC* and *Piggyback*. We will discuss the performance for them on NI(New Instance) case and NIC(New Instance and Class) case respectively. Then we will analyze their performance case by case. According to the guideline of the *CLVision Challenge*, we use average accuracy, i.e. the mean accuracy per task as the metrics. Besides, metrics including maximum RAM usage, average RAM usage and training time should also be taken into account to validate the performance.

4.1 EWC

In this part, due to the requirements of GPU memory, all of our experiments are done with a linux server with Intel Broadwell CPU platform, 52 GB RAM and Tesla T4 GPU with 16GB SDRAM. In all our cases, we set batch size to 32, with learning rate at 0.01 and decay one tenth every 5 epochs. To

Table 1: Performance for EWC/Piggyback

Scenario	Baseline(ResNet50)	EWC	Piggyback(ResNet50)	Piggyback(EWC Combined)
NI-val	0.54	**0.69**	0.21	0.62
NI-test	0.71	**0.75**	-	0.74
NI-RAM(MB)	**15378.31**	22149.00	16287.79	21846.47
NC-val	0.51	0.54	0.10	**0.71**
NC-test	0.81	**0.89**	-	0.82
NC-RAM(MB)	**13970.41**	29170.46	14320.47	14787.31

achieve better performance, we will load the pre-trained ResNet50 and apply *EWC* on all following tasks.

4.2 Piggyback

As mentioned in Section 3, the choice of a well-trained base network is crucial to the performance of $Piggyback$. In this part, all of our experiments are done on a platform with CPU i7 8700K, 32GB RAM, and RTX 2080 GPU with 8GB SDRAM. In all our cases, we set batch size to 32, with learning rate at $1e-4$ and decay one tenth every 5 epochs. At the very first, we have followed the setting in the paper, but failed to reproduce the result. We have tried out ImageNet-pretrained VGG16(10), DenseNet(2) and ResNet50 as base network but the best average accuracy we can reach is 0.1037 in NIC case and 0.2103 in NI case, which is even far worse than the baseline. Considering the principle of Piggyback, we deem that it is the worse performance of the orignal model that causes the bad performance for Piggyback. We test the accuracy before training and it is only 0.0148. This worse performance could be caused by the under-training of the fully connected layer in these three models. As mentioned before, there are 50 classes in our dataset, considering these 50 classes are not similar to what it is in the pre-trained network, there pre-trained weights do not work well on predicting the label of new task. Therefore, modification of the mask can partially solve the problem, but the core cause is the poor performance of the original weight.

To solve this problem, we modified the training procedure. We still load the pre-trained weights for ResNet50 and let the model run EWC at the first task. Then, at the proceeding task, we create mask layer $\mathbf{m_r}$ for the model, and run Piggyback. In this approach, we have raised the final average validation accuracy to 0.53 in NC case, which is slightly better than our baseline. In NI case, the

4.3 Analyze

The summarization of the performance result is shown in Tabel 1. Here *-val represents the performance on validation set, *-test is the performance on test data set. Accuracy on validation dataset is the average of per-task accuracy. Throughout the training process, the validationset is fixed. Therefore, this index can represent how much the model have mitigated the catastrophic forgetting. Since the label for the test dataset is not provided, the accuracy on test dataset is acquired by submitting the result to the official site. Therefore, we do not validate the test performance of Piggyback due to its poor performance.

- **NI** From the perspective of validatoin accuracy, EWC performs the best, scoring 0.69 in validation and 0.75 in test. It demonstrates EWC's ablility to reach and maintain high accuracy after iterating through 8 batches of new instance data. However, due to EWC have to retrain the whole network when encountering new task, it consumes much more RAM than Piggyback and the baseline.
- **NC** In NC case, EWC is still the best in test accuracy. Piggyback combined with EWC outperforms EWC on validation accuracy. This could be caused by the under training in the very beginning. EWC and baseline are under-training in the first several batches, causing the low performance at the beginning. However, EWC ends up with a high accuracy after it sees more and more samples, while Piggyback will train a mask weight separately for each

task, ending up with a more stable performance per task. As for RAM usage, EWC still uses the most RAM. And there is no significant difference between

From this result, we can see that EWC is better than modified Piggyback regarding the accuracy, however its memory consumption is also larger. Besides, in a online setting, Piggyback is more stable than pure EWC. As for test accuracy, Piggyback has no significant advantage over the baseline network, indicating that it only works in some of the dataset, as pointed out in the paper by the author.

5 Conclusion

In this project, we have looked thoroughly into the literature and legacy work of continual learning and then we choose two representative work, EWC and Piggyback, to experiment on the *CLVision Challenge* issued by CVPR 2020 Workshop. For EWC, the performance is slightly improved compared to baseline model. For Piggyback, the initial performance is far worse even than baseline, then we propose a combination of EWC and Piggyback, which led to the best performance in our project in both NI and NC cases. In the future, we could try out and continue to use and refine some dynamical approaches like DEN, and other incremental approaches(13).

References

[1] HE, K., ZHANG, X., REN, S., AND SUN, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 770–778.

[2] HUANG, G., LIU, Z., VAN DER MAATEN, L., AND WEINBERGER, K. Q. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2017), pp. 4700–4708.

[3] KIRKPATRICK, J., PASCANU, R., RABINOWITZ, N., VENESS, J., DESJARDINS, G., RUSU, A. A., MILAN, K., QUAN, J., RAMALHO, T., GRABSKA-BARWINSKA, A., ET AL. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences 114*, 13 (2017), 3521–3526.

[4] LI, Z., AND HOIEM, D. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence 40*, 12 (2017), 2935–2947.

[5] MALLYA, A., DAVIS, D., AND LAZEBNIK, S. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *Proceedings of the European Conference on Computer Vision (ECCV)* (2018), pp. 67–82.

[6] MALLYA, A., AND LAZEBNIK, S. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2018), pp. 7765–7773.

[7] MCCLOSKEY, M., AND COHEN, N. J. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, vol. 24. Elsevier, 1989, pp. 109–165.

[8] PARISI, G. I., KEMKER, R., PART, J. L., KANAN, C., AND WERMTER, S. Continual lifelong learning with neural networks: A review. *Neural Networks* (2019).

[9] RUSU, A. A., RABINOWITZ, N. C., DESJARDINS, G., SOYER, H., KIRKPATRICK, J., KAVUKCUOGLU, K., PASCANU, R., AND HADSELL, R. Progressive neural networks. *arXiv preprint arXiv:1606.04671* (2016).

[10] SIMONYAN, K., AND ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[11] THRUN, S. A lifelong learning perspective for mobile robot control. In *Intelligent Robots and Systems* (1995), Elsevier, pp. 201–214.

[12] YOON, J., YANG, E., LEE, J., AND HWANG, S. J. Lifelong learning with dynamically expandable networks. *arXiv preprint arXiv:1708.01547* (2017).

[13] ZHOU, G., SOHN, K., AND LEE, H. Online incremental feature learning with denoising autoencoders. In *Artificial intelligence and statistics* (2012), pp. 1453–1461.