StackNet: Stacking feature maps for Continual learning

Jangho Kim*
Seoul National University
Seoul, Korea
kjh91@snu.ac.kr

Jeesoo Kim*
Seoul National University
Seoul, Korea
kimjiss0305@snu.ac.kr

Nojun Kwak†
Seoul National University
Seoul, Korea
nojunk@snu.ac.kr

Abstract

Training a neural network for a classification task typically assumes that the data to train are given from the beginning. However, in the real world, additional data accumulate gradually and the model requires additional training without accessing the old training data. This usually leads to the catastrophic forgetting problem which is inevitable for the traditional training methodology of neural networks. In this paper, we propose a continual learning method that is able to learn additional tasks while retaining the performance of previously learned tasks by stacking parameters. Composed of two complementary components, the index module and the StackNet, our method estimates the index of the corresponding task for an input sample with the index module and utilizes a particular portion of StackNet with this index. The StackNet guarantees no degradation in the performance of the previously learned tasks and the index module shows high confidence in finding the origin of an input sample. Compared to the previous work of PackNet, our method is competitive and highly intuitive.

1. Introduction

The main difference between the human brain and the machine learning methodology is the ability to evolve. Using neurophysiological processing principles, human brains can achieve and organize knowledges throughout their lifespan. Having the neuroplasticity, human brains can transfer an activating region of a given function to a different location or control the creation and destruction of synapses according to its experiences. Usually, artificial neural network (ANN) models consist of a finite number of filters. Also, parameters and operations in the ANN do not possess the ability corresponding to the memory system of human brains. This structural limit leads to the problem called *catastrophic forgetting*, i.e., the newly coming information diverts the model from previously learned knowledge. The field of researches trying to solve this problem is referred to as *continual learning*.

Many researches based on the regularization method have been proposed to solve this problem. [6] and [11] proposed methods regularizing the output and the feature of the newly trained model respectively.

To make a single network deal with a large number of datasets, instead of using all parameters of a single network for each task, PackNet suggested a model which allocates the datasets to specific weights of filters [14]. As only the specialized weights for a particular task are involved in its classification process, PackNet shows a remarkable result in multiple tasks. However, the information about from which task the input comes and which group of weights should be used must be given in advance. Typically, images do not contain such prior knowledges and this makes the PackNet hard to apply in real world situation.

In this paper, we propose a network which efficiently uses the capacity of the network for continual learning without degrading the performance of previous tasks. We have built our method using two complementary components as shown in Figure 1. The *StackNet* keeps the knowledge from several independent tasks. After training the StackNet from the previous task with a certain amount of parameters, additional parameters in the convolutional modules are stacked and trained with the next task. Note that the given capacity of the StackNet is fixed. StackNet stacks the parameters under the given capacity as depicted in Figure 2. To infer the newly coming data, the model utilizes the newly trained filters along with the previously learned filters, whereas the data from the old task is inferred only using the previously learned filters. In order to determine which combination of filters to use, we adopted a index module which can distinguish the origin of a given input sample. It can recall the information not only which group of filters to use but also which group of class labels to use. This endows our method label-expandability. We suggest the method for the index module, which is generative adversarial networks (GAN) [3] and report the properties and performance of this method. Our index module can be

*Equal Contribution
†Corresponding Author

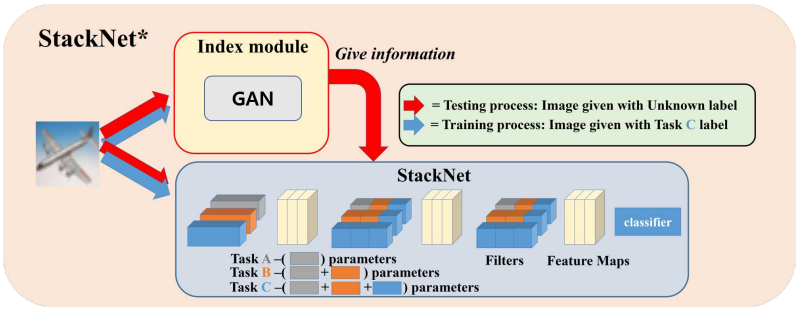

Figure 1. The overall architecture of our method called a StackNet* (Index module + StackNet). Blue arrows depicts the training process for the new task (Task C) of our method, where Task A and Task B have been already trained. Index module is trained each time a dataset for a new task is provided and then the StackNet is trained in a standard way such as using the cross-entropy loss. We allow only the trainable filters (Blue parameters) to be updated. Red arrows shows the inference procedure of our method. Index module finds out the task index $\mathcal{J}$ (Task C) from which the input came and let the StackNet to use the filters specialized for task $\mathcal{J}$ (Blue + Orange + Grey). Class labels are also switched to those of task $\mathcal{J}$.

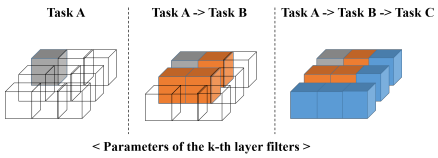

Figure 2. Under the fixed capacity, a certain amount of parameters (Grey) are used in training Task A. After training Task A, additional parameters (Orange) are stacked and trained for Task B. In the same way, The remaining parameters (blue) are trained for Task C.

combined with any other works such as PackNet or Learning without forgetting (LwF) [11]. The combined index module and StackNet can prevent the *catastrophic forgetting* using different parameters for different tasks under the constraint network capacity. The contributions of this paper can be summarized as the followings:

- Filters in the convolutional layers are allocated to each task and parameters from the previously learned tasks are shared among post tasks and used for the initialization of post tasks parameters, which makes the overall model compact and efficient.

- After using constraint parameters, attaching parameters whenever a new dataset is trained allows the model to be expandable as far as the physical constraints, such as memory and processing time constraints, allow.

- To identify which portion of convolutional filters should be used to classify a data, the index module is proposed with several methodologies.

- The overall method is applicable to multi-task continual learning whose number of class labels increases over tasks.

2. Related work

During the past decade, a lot of progress has been made in numerous fields of machine learning. Image classification problem has induced various useful architectures [21, 4, 20] applicable to any other tasks. Many researchers have been struggling to enhance the performance in object detection or image segmentation problems leaving many masterpieces behind [12, 16, 2, 13]. For performance evaluation of the models above, training and test data are usually separated within a single task. However, what we call intelligence must be able to hold the previously-learned knowledge even after learning new knowledge. In the real situation, the model has to remember the learned task without seeing the old task data again.

[11] proposed a method trying to retrain the model by using the outcomes of the previously learned model as anchoring targets. When training a new task, a considerably small learning rate is used to prevent the model from losing the learned knowledge. [6] proposed a method reusing the trained weights of the learned network. Training a new target network, the feature representation at the last layer is enforced to preserve the response of the old network using L_2 regularization. [7] suggested a penalty on the quadratic distance between the old parameters and the new ones. These approaches using regularization basically focus on alleviating the deformation of the original network. Therefore, the more the new knowledge flows into the model, the more the performance on the original task degrades. Moreover, the performance on the new task is not guaranteed to be in its best state.

Dynamic architectures to overcome the *catastrophic forgetting* have been suggested in many researches. [18] proposed a progressive network which grows every time a new task is given. Features in the previous columns are concatenated to the layers of the new column. progressive network and stacknet seem to be similar such that they can increase

their capacity and can learn more continual tasks. However, there are many different things. In the progressive network, each feature is summarized by an additional adapter and passed to the next task network. Therefore, it is not a typical form of CNN and they need to train the additional adapter and increase their capacity with this adapter related specific task. However, the stacknet utilizes the fixed capacity which is given. It uses the task index for splitting the given filters or weights. When the stacknet uses up the given capacity and comes to train a new task, it can then increase the model size beyond the given capacity. [14] proposed a model using the weight pruning method. Leaving the remaining weights fixed after pruning, the model can solely focus on the task on interest. Despite of its efficiency and performance, this method, referred as PackNet, requires the prior knowledge of a given input. Also, once the model is occupied by several tasks, no more task is allowed. This restricts the model from being expandable. Unlike the PackNet, our StackNet simply stack parameters in the convolutional filters instead of a cumbersome pruning procedure while allowing the model to expand further.

In our method, instead of saving all the memory in the main network and trying to access it directly, the index module finds the index of the memory scattered over the main network where knowledges are well organized inside.

3. Method

We propose a method that efficiently stacks parameters for continual learning that satisfies three properties as stated below.

Property 1. The *index module* offers the index about which part of the *StackNet* should be used.

Property 2. The StackNet learns the knowledge on the new task under the presence of the previously learned knowledge.

Property 3. Index module and StackNet are flexible to the expansion of the network without the retraining of previous tasks and as many new tasks can be learned in one network as physical constraints allow.

3.1. StackNet

We propose an efficient way to allocate the capacity of a network according to the given tasks. StackNet is a network that can be separated into several partitions for multi-task and additional filters can be attached if a new task is given. It can be applied to many typical networks such as VGG and ResNet [20, 4]. The only difference with the original versions of those networks is that it uses different parts of filters for different tasks. As shown in [10], typical convolutional neural networks (CNN) are capable of maintaining their performance even when the majority of their parameters are pruned. This implies that filters in a CNN contains a lot of unnecessary or redundant information. Also, sharing

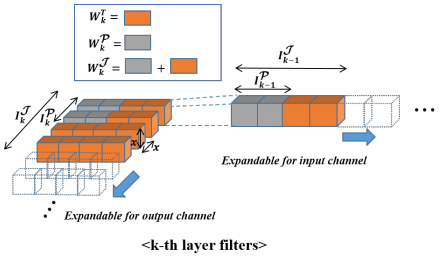

Figure 3. Trainable parameters W_k^T (Orange) and frozen parameters $W_k^{\mathcal{P}}$ (Grey) in the StackNet training process. In the training process, the parameters (Orange), which are trainable parameters except frozen parameters (Grey), are updated. x denotes the spatial dimension of filters.

the filters among different tasks inevitably leads to the *catastrophic forgetting* of the network since the parameters trained by an old task should be replaced with those trained by a new task. For these reasons, we utilize a network divided into several parts and refer it as StackNet where each part takes charge of a particular task.

While training the StackNet with several tasks, each task uses a different part of the StackNet. For the convenience of referring which segments of the StackNet are to be activated, we introduce a filter index $I^{\mathcal{J}}$ for the $\mathcal{J}$-th task. The filter index $I^{\mathcal{J}}$ defines the range of filters to use for the specific task $\mathcal{J}$ in every layers. For example, if the filter index I^1 equals to ten, the first task uses filters from the first to the tenth filter.

Depicted in Figure 3, $I_k^{\mathcal{J}}$ denotes the filter index in the k-th layer corresponding to the task $\mathcal{J}$. This equals to the number of output channels in the sequent feature map. $I_k^{\mathcal{P}}$ denotes the filter index in the k-th layer for the previous task. As the same principle is applied to the precedent $(k-1)$-th layer, the number of input channels for the filters in the k-th layer will change likewise. That is, total of $I_k^{\mathcal{P}}$ filters have the channel length of $I_{k-1}^{\mathcal{P}}$ when the previous task is under training. As the new task is given, the length of the filters become $I_{k-1}^{\mathcal{J}}$ since the filters in the prior layer will deliver a feature map whose length is $I_{k-1}^{\mathcal{J}}$. Therefore, both the number of input and output channels increases as the model grows.

Weight parameters of $\mathcal{J}$-th task in filters of the k-th layer are referred as $W_k^{\mathcal{J}} \in R^{x \times x \times I_{k-1}^{\mathcal{J}} \times I_k^{\mathcal{J}}}$, where x means the size of the kernel while $W_k^{\mathcal{P}} \in R^{x \times x \times I_{k-1}^{\mathcal{P}} \times I_k^{\mathcal{P}}}$ are the weight parameters of the previous task. When training the new task, $W_k^{\mathcal{J}}$ is used for the inference but only the parameters $w \in W_k^{\mathcal{T}}$ ($W_k^{\mathcal{T}} = W_k^{\mathcal{J}} \setminus W_k^{\mathcal{P}}$) are updated while leaving $W_k^{\mathcal{P}}$ fixed. Note that biases in the filters can also be trained likewise as explained above. The fully connected layer located just before the linear classifier can be divided as well into the sections of $I^{\mathcal{J}}$ and $I^{\mathcal{P}}$.

To enhance the performance, slices of trained parameters

Algorithm 1 StackNet Training

Input: $W^{\mathcal{J}}, I^{\mathcal{P}}, I^{\mathcal{J}}$
Output: $W^{\mathcal{J}*}$

1: Freeze the $W^{\mathcal{P}} \subset W^{\mathcal{J}}$ using the information $I^{\mathcal{P}}$
2: Initialize the $W^{\mathcal{T}}$ with $W^{\mathcal{P}}$ and a random noise
3: $W^{\mathcal{J}*} \leftarrow \arg\min(\mathcal{L}_c)$
{Update W^T using backpropagation}

$W^{\mathcal{P}}$ from the old task are duplicated to the newly available task parameters W^T and Gaussian noise is added to them for the sake of good initialization. Then, the network is trained in a standard way, e.g., using a cross-entropy loss $\mathcal{L}_c$. Independent linear classifiers are used for every tasks. This scheme of StackNet can also be applied to structures using shortcut connections such as ResNet. The overall training process is shown in Algorithm 1.

3.2. Index module

Even if the StackNet is well trained separately with several tasks, the task index $\mathcal{J}$ must be given at the time of inference. Especially when the class labels expands, this prior information $\mathcal{J}$ is necessary to estimate which group of class labels to use. In the testing step, existing methods needs the exact origination of the input data. However in the real world, a given data for classification normally does not offer any information about its origination. Therefore, this prior knowledge should be estimated by an independent network.

To solve this problem, we introduce the index module using several different methods, which is able to estimate the task $\mathcal{J}$ of the given input data and inform this to the StackNet to specify which filters to use.

3.2.1 Generative adversarial networks (GAN)

This method is inspired by the on-line replay method used for continual learning [19]. We train a task-specific generative model $G_{\mathcal{J}}$ using a GAN to generate pseudo-samples of task $\mathcal{J}$. The generator is trained using the adversarial loss as follows:

$$\min_{G_{\mathcal{J}}} \max_{D_{\mathcal{J}}} V(D_{\mathcal{J}}, G_{\mathcal{J}}) = \mathbb{E}_{x \sim P^{\mathcal{J}}_{data}}[\log(D_{\mathcal{J}}(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D_{\mathcal{J}}(G_{\mathcal{J}}(z))]. \quad (1)$$

Here, $D_{\mathcal{J}}$ is a discriminator for the task $\mathcal{J}$ and x is a sample from the task $\mathcal{J}$.

After generating samples of all tasks, a task-wise binary classifier $B_{\mathcal{J}}$ is trained to classify whether the given input is from the task $\mathcal{J}$ or not. Note that only the generated samples are involved for the training of each binary classifier $B_{\mathcal{J}}$. This means that positive samples are from the generator $G_{\mathcal{J}}$ for the task $\mathcal{J}$ and negative samples are from all the other task generators $G_{\mathcal{K}}$ ($\forall \mathcal{K} \neq \mathcal{J}$).

Table 1. Details of datasets.

Datasets	#of Train data	#of Test data	#of Class
MNIST	60,000	10,000	10
SVHN	73,275	26,032	10
CIFAR-10	50,000	10,000	10
ImageNet-A	64,750	2,500	50
ImageNet-B	64,497	2,500	50

In the testing phase, each classifier produces the probability of how likely the given unknown input data is from the task $\mathcal{J}$. The classifier B_i with the highest probability assumes that the input data is from the task i. index module can figure out the task $\mathcal{J}$ of input data with maximum probability across the set of classifiers. index module gives this estimated task index $\mathcal{J}$ to the StackNet. The index module using GAN can be summarized as below:

$$\begin{aligned} &\text{Initialization:} \quad B = \{B_1, B_2, ..., B_N\} \\ &\text{Training:} \quad B_i(x) = \begin{cases} 1 & \text{if } x \text{ is from } G_i \\ 0 & \text{if } x \text{ is from } G_j \ \forall j \neq i \end{cases} \\ &\text{Inference:} \quad \mathcal{J} = \arg\max_i B_i(x) \end{aligned} \quad (2)$$

4. Experiment

We evaluate our method on several image classification datasets. First, we verify the effectiveness of our method with MNIST [9], SVHN [15] and CIFAR-10 [8] which are widely used to evaluate image classification performance. Then, we evaluate our method on two subsets of ImageNet [17] which is a real world image dataset. More details are summarized in Table. 1.

We compare our method to various methods such as Learning without forgetting (LwF) and PackNet as well as networks trained for a single target task which shows the performance without conducting continual learning.

4.1. Continual learning using basic image classification datasets

We have built three experimental scenarios to evaluate our method using basic image classification datasets. We compare the performance of learning a consecutive two task pair among MNIST, SVHN and CIFAR-10 dataset. After that, we conduct multiple-task continual learning with these datasets.

4.1.1 Training details

We use the same learning parameters and network architectures suggested in [6] used for 'Tiny image classification'. It is composed of three convolutional layers using 5×5 kernels with the size of 32, 32 and 64 channels respectively,

three max pooling layers, one fully connected layer with 200 nodes and the last softmax classifier layer producing 10 outputs. ReLU is used as the activation function in all the experiments. Also an SGD optimizer with a mini-batch size of 100 has been used for model optimization. When training the old task, the weight decay and the momentum were set to 0.004 and 0.9 respectively. The learning rate starts from 0.01 and the decay of the learning rate with a factor 0.1 is done at the time of 20,000 iterations. After 40,000 iterations, the training is terminated in all the experiments. When training a new task, LwF, PackNet and StackNet have the same settings for the old and new task because they need more iterations as the new parameters are to be trained from the scratch. Note that PackNet needs additional pruning and finetuning step. In the new task training of LwF, we start with a learning rate of 0.0002 and 0.001 which are $0.1 \sim 0.02$ times less than that of old task for the 'MNIST to SVHN' and 'SVHN to CIFAR-10' respectively, as recommended in [11]. The channel split ratio which determines $I^{\mathcal{J}}$ follows the setting of PackNet.

In Table 2, the result of 'StackNet' shows the performance of the StackNet only. That is, it assumes that the index module never fails and the task index $\mathcal{J}$ is always given correctly. Likewise, LwF and PackNet need prior knowledge of a given input to know which classifier to use. As there is no structure like index module in the original paper, The actual comparison must be done with 'StackNet'. On the other hand, StackNet* shows the performance of the StackNet which uses the task index $\mathcal{J}$ given from the index module.

4.1.2 MNIST → SVHN

After selecting a subset of SVHN to equalize the number of training data between MNIST and SVHN, images are resized to 28×28 as in [6]. The 'single network' in Table 2 is a network of full capacity trained using a single dataset. The column 'Old' represents the test accuracy on the previously learned task and 'New' is that of the newly trained task. A knowledge distillation loss with hyper-parameters of $T = 2$ and $T = 1$ [5] is used for LwF.

LwF can control the performance trade-off between the old and new task by changing the temperature parameter T. PackNet almost maintains the performance of the single network for the first task with less than 2% drop in accuracy. StackNet* outperforms all other methods in the table. Note that the prior knowledge to select which task is given in all experiments of other methods. Therefore, StackNet, which shows the highest in performance, should be compared to other baseline methods. Nonetheless, our full model, StackNet* also outperforms PackNet, especially on the second task by around 0.7% even without using the prior knowledge.

Table 2. Mean classification results on the basic datasets (5 runs).

Datasets	Methods	Old(%)	New(%)	Avg. (%)
MNIST ↓ SVHN	single network (MNIST)	99.49	–	–
	single network (SVHN)	–	92.82	–
	LwF ($T = 1$)	98.27	86.40	92.34
	LwF ($T = 2$)	97.33	86.97	92.15
	PackNet	99.45	91.49	95.47
	StackNet*	99.43	92.20	95.82
	StackNet	99.43	92.37	95.90
SVHN ↓ CIFAR-10	single network (SVHN)	92.94	–	–
	single network (CIFAR)	–	79.69	–
	LwF ($T = 1$)	91.76	70.57	81.17
	LwF ($T = 2$)	90.19	71.94	81.07
	PackNet	92.84	76.78	84.81
	StackNet*	90.05	76.62	83.34
	StackNet	92.21	76.70	84.46

Datasets	Methods	MNIST(%)	SVHN(%)	CIFAR(%)	Avg(%)
MNIST ↓ SVHN ↓ CIFAR-10	single network (MNIST)	99.44	–	–	–
	single network (SVHN)	–	92.94	–	–
	single network (CIFAR)	–	–	79.69	–
	LwF ($T = 1$)	95.06	86.80	68.98	83.61
	LwF ($T = 2$)	86.09	85.07	69.63	80.26
	PackNet	99.38	91.93	66.34	85.88
	StackNet*	99.41	89.36	74.93	87.90
	StackNet	99.41	91.84	75.02	88.76

4.1.3 SVHN → CIFAR-10

In this experiment, LwF and PackNet show similar trends as that of 'MNIST to SVHN'. Higher temperature (T) induces better performance in the new task but a bit of degradation in average performance still occurs. PackNet experiences almost no performance degradation. The pruning method of PackNet acts like a generalization method and helps the model to maintain its capacity. The average performance of StackNet is slightly lower than that of PackNet by 0.35%. However, note that PackNet requires additional finetuning after the pruning process and takes additional time for training. On the other hand, StackNet requires no additional pruning or finetuning precedure and yet retains the performance.

4.1.4 MNIST → SVHN → CIFAR-10

Training three or more tasks is unmanageable for the models using regularization methods. No matter how we change the temperature (T), LwF suffers from a drop in average performance, especially in the third task. Also, even though the proportion allocated to each task is equivalent between the PackNet and our methods (StackNet and StackNet*), our method highly outperforms the PackNet in the third task. This implies that StackNet is more suitable for multi-task sequential learning situation.

4.1.5 Multitasks more than three tasks

We also conducted an experiment for 5 tasks to verify effectiveness on multitasks more than three tasks. We used 5 datasets, cifar10, svhn, KMNIST [1], FashionMNIST [22] and MNIST. We used ResNet-32 [4]for the base network. The performance of single networks are 86,95,98,93 and 99

Table 3. Mean classification results for the realistic dataset (2 runs)

datasets	Methods	Old(%)	New(%)	Avg. (%)
ImageNet-A	single network (ImageNet-A)	83.28	–	–
	single network (ImageNet-B)	–	85.28	–
↓	LwF ($T = 1$)	82.2	86.72	84.46
ImageNet-B	LwF ($T = 2$)	80.92	86.96	83.94
	PackNet	82.16	88.72	85.44
	StackNet	83.30	88.66	85.98

Table 4. Performances of the old (MNIST) and new (SVHN) tasks with different initialization methods: 1) initialization with only random Gaussian noise and 2) the pretrained parameter added by random Gaussian noise

Initialization	Old(%)	New(%)	Avg.(%)
Random	99.42	91.67	95.55
Pretrained parameters	99.43	92.37	95.90

respectively. In the same order, the performance of the stacknet are 83.93,95,91 and 99. As we can see, the results have a similar tendency compared to the previous experiments. The accuracy degradation is within the range of 0 3%. Also, StackNet can handle more than 3 tasks.

4.2. Continual learning using realistic datasets

ImageNet contains images more realistic than other datasets used in this paper. Higher resolution and complex backgrounds make the classification even harder. To save the training time, two subsets of the ImageNet dataset each having 50 randomly chosen classes have been used for evaluation. They are referred to as ImageNet-A and ImageNet-B respectively in this paper. To show the adaptability of our method on structures having shortcut connections, ResNet-50 is used for the experiment. We use the same experimental setting as in [4] with a batch size of 128. We compare our StackNet with LwF and PackNet.

Table 3 shows the results on the ImageNet dataset. The 'Old' accuracy of all methods slightly dropped from those of single networks. Like in the 'SVHN to CIFAR-10' experiment, PackNet shows a better result than the LwF regardless of the value of T. The result of StackNet is almost identical to PackNet with a slight increase in the average accuracy.

PackNet actually utilizes more parameters for each task than StackNet does. Since the masks force weights with no influence to be zero and utilize the well trained remaining weights, PackNet can make good use of the entire network. However in StackNet, just adding filters gradually without any other post-processing performs well enough compared to PackNet. When the model of an initially designed size is fully occupied by several tasks, StackNet can just add more filters and train them along with the trained filters while this is not the case of PackNet. Furthermore, PackNet requires additional memories to store the masks for all filters in it. StackNet needs only one integer per layer for each task. For these reasons, StackNet is far more efficient than the PackNet with no loss in performance.

4.3. Ablation study on the efficiency of StackNet

As mentioned above, our StackNet appends parameters whenever a new task is to be trained. In this process, parameters learned from previous tasks are utilized altogether and the parameters which are to be trained newly are initialized using the existing ones. We conduct experiments to show the effect of these methods. Experiments are carried out in 'MNIST to SVHN' case as a representative.

4.3.1 Initialization with pretrained parameters

To analyse the effectiveness of initialization using pretrained old task parameters with random Gaussian noise, we compare the accuracy between a model initializing the new parameters with just a random Gaussian distribution and a model initializing the new task parameters using the pretrained old task parameters with additional random Gaussian noise. In Table 4, the model initialized by our method obviously shows higher accuracy on the SVHN task. This result implies that a good initialization prevents the model from going through a local minima.

4.3.2 Parameter sharing

We verify the effect of the parameter sharing between the old task and the new task. The baseline method is a model where the old parameters are filled with random Gaussian initialization and no further training is done. The new parameters in the model have to learn knowledges from the new task without the aid of old parameters since they are fixed from the beginning. On the other hand, the old parameters in our method is trained by the old task and the new parameters can make use of these learnt knowledges to learn the new task. To solely observe the effect of the parameter sharing, both methods do not use our parameter initialization method mentioned in the previous section. The task indices I^1 and I^2 are $\{16,16,32\}$ and $\{18,18,34\}$ respectively. The model using parameter sharing converges far more faster than the model with no parameter sharing which is shown in Figure 4. Also the final error rate of our method is lower than the other. Increasing the number of indices elevation enhances our method by 6.38% and the other model by 7.91% as shown in Table 5. This implies that the parameter sharing improves the model and allow it to be compact with fewer numbers of filters.

4.4. Index module

To solely examine the influence of index module on our method (StackNet*), we have conducted experiments with the same experimental scenarios in StackNet* for GAN. As a baseline method for the index module, we adopted a highly

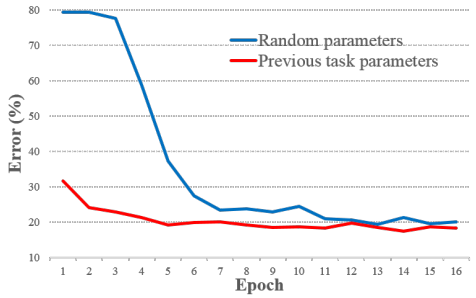

Figure 4. Error rate comparison between sharing the parameters which learned previous task and random parameters.

Table 5. The performances on the new task while increasing the filter index of the model using parameter sharing and the model with random fixed parameters.

#of increasing each index	Ours(%)	Random(%)
2	80.60	78.10
4	85.31	83.85
6	86.98	86.01

Table 6. Experimental results of the index module on the basic datasets

Datasets	Methods	Old(%)	New(%)	Avg. (%)
MNIST→ SVHN	GAN	100	99.94	99.97
	Baseline	61.65	97.17	79.41
SVHN→ CIFAR-10	GAN	97.00	99.90	98.45
	Baseline	89.39	85.86	87.63

Datasets	Methods	MNIST(%)	SVHN(%)	CIFAR(%)	Avg(%)
MNIST→ SVHN CIFAR-10	GAN	100	96.65	99.83	98.83
	Baseline	52.88	90.89	75.7	73.16

naive method that chooses the task index $\mathcal{J}$ according to the confidence of the StackNet. Inferring each task output from the StackNet, the output with the highest probability decides the task index. (e.g, Task index $\mathcal{J} = \arg\max_i(P_i)$ where P_i is the highest probability of i-th classifier)

In continual learning on MNIST to SVHN and SVHN to CIFAR-10 datasets, an index module approximates the results to perfection (See Table 6). The experiment of three datasets also shows the same trend as the experiments above. The baseline definitely shows a poor performance also in all cases, which underlines the necessity of the index module. The generated samples with GAN are depicted in Figure 5,6 and 7.

4.4.1 Limitation

Generative models usually have difficulties in generating realistic images with high resolution and there are few researches experimented on ImageNet data. This incompetence makes the index module hard to be applied to the ImageNet data. Also, when it comes to dealing with datasets of similar distribution, the index module may not be working as good as it is reported in this paper.

(a) Generated samples with the generator of index module (b) Real samples

Figure 5. MNIST dataset

(a) Generated samples with the generator of index module (b) Real samples

Figure 6. SVHN dataset

(a) Generated samples with the generator of index module (b) Real samples

Figure 7. CIFAR-10 dataset

5. Conclusion

In this paper, we have proposed a novel framework which is able to divide its capacity into several parts and utilize them according to the given input. Composed of two networks, the index module is responsible for memorizing "where the data came from" while complicated knowledge such as "what the data is" is engraved to the StackNet. To the best of our knowledge, our work, index module, is the first attempt to estimate the origin of data which used to be assumed as given in the previous works. As well as overcoming the catastrophic forgetting, StackNet allows extra class labels when training a new dataset and also has expandability to the network architecture.

Acknowledgments

This work was supported by Next-Generation Information Computing Development Program through the NRF of Korea (2017M3C4A7077582).

References

[1] Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. Deep learning for classical japanese literature. *arXiv preprint arXiv:1812.01718*, 2018.

[2] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems*, pages 379–387, 2016.

[3] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[5] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[6] Heechul Jung, Jeongwoo Ju, Minju Jung, and Junmo Kim. Less-forgetful learning for domain expansion in deep neural networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[7] 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*, page 201611835, 2017.

[8] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[9] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[10] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[11] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[12] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[13] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[14] Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[15] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, page 5, 2011.

[16] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[17] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[18] 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.

[19] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In *Advances in Neural Information Processing Systems*, pages 2990–2999, 2017.

[20] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[21] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[22] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.