Incremental Learning through Deep Adaption

Zichao Yan[1], Jianing Sun[2]

[1]Department of Computer Science
[2]Department of Electrical Computer Engineering

McGill University

Incremental Learning

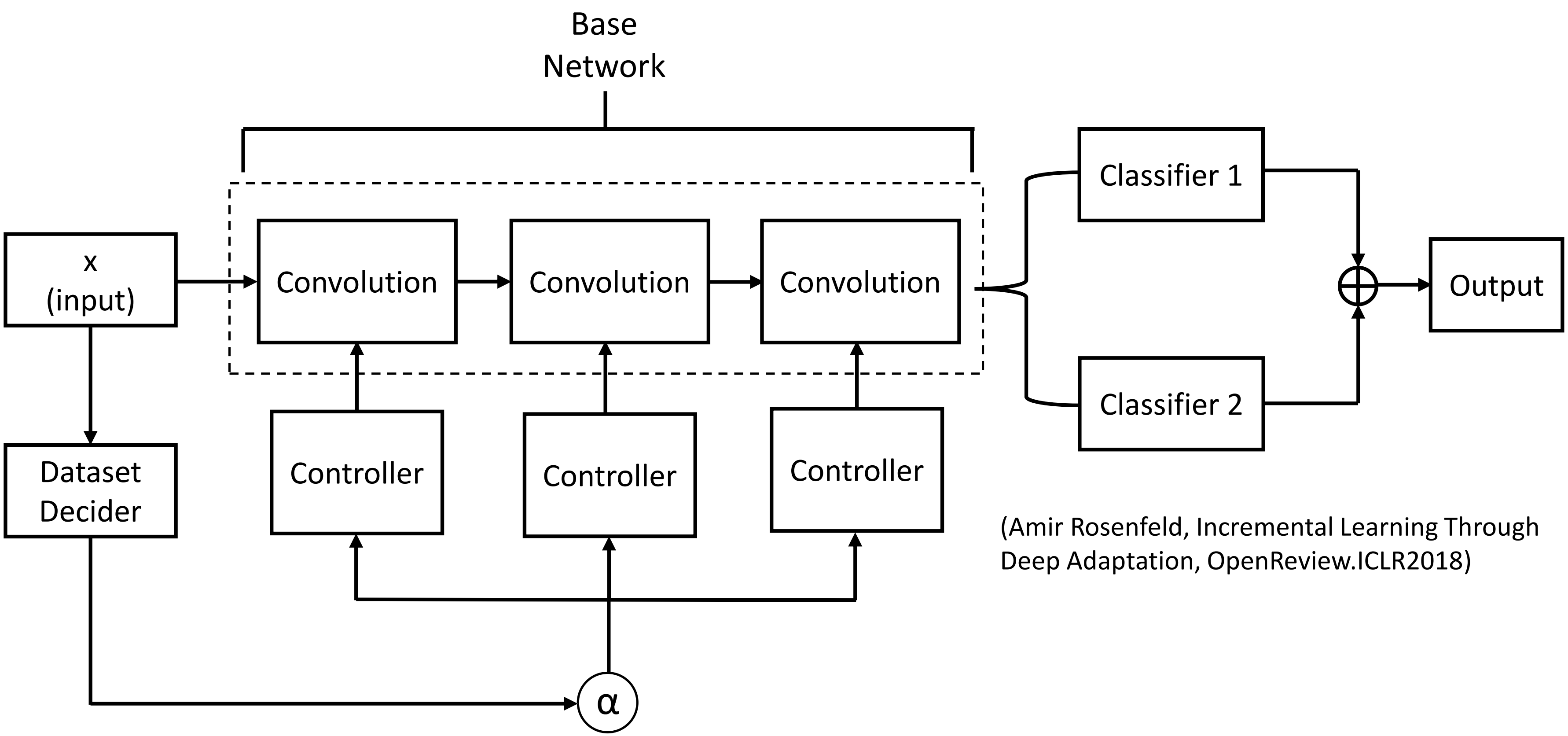
Let’s start from the problem of Deep Neural Network

- Typically a separate model needs to be trained for each new task
- Given two tasks of a similar modality or nature, each would require a similar architecture or computations

Incremental Learning aims at enhancement of knowledge

- Our goal is to enable a network to learn a set of related tasks be learned incrementally, without fine-tuning all parameters again, and achieve a comparable performance
- Compared with full-transfer learning: double the parameters (100%). Our method only requires a 22% to be trained

Approach

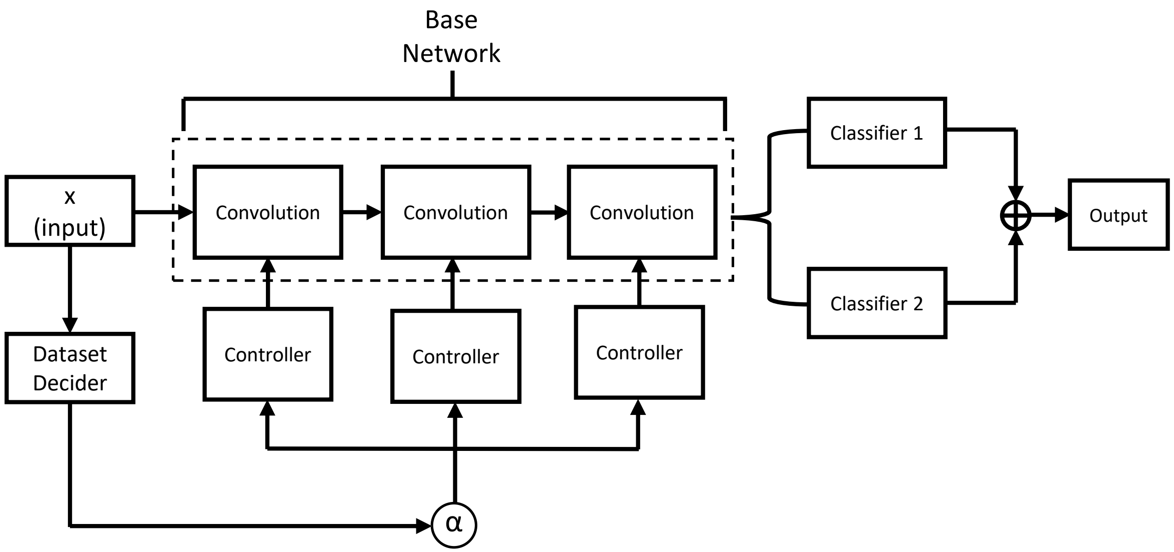


(Amir Rosenfeld, Incremental Learning Through Deep Adaptation, OpenReview.ICLR2018)

- Each convolutional layer of a base network is modified by re-combining its weights through a controller module
- A binary switching vector α controls the output of the network

Adaption Representations

(Amir Rosenfeld, Incremental Learning Through Deep Adaptation, OpenReview.ICLR2018)



- Given two tasks, T1 and T2, we then learn a base network N to solve T1
- We augment N so that it will be able to solve T2 as well

For each convolutional layer $\emptyset_l$ in N, let $F_l \in R^{C_0 \times C_i \times k \times k}$, where C_0 is the number of output features, C_l is the number of inputs, $k \times k$ is the kernel size

Denote by $\widetilde{F_l} \in R^{C_0 \times D}$ the matrix whose rows are the flattened versions of the filters of $\widetilde{F_l}$, where $D = C_i \cdot k \cdot k$. Let $f \in R^{C_i \times k \times k}$ be a filter from F_l whose values are

$$f^1 = \begin{pmatrix} f^1_{11} & \cdots & f^1_{1k} \\ & \ddots & \\ & & f^1_{kk} \end{pmatrix}, \cdots, f^i = \begin{pmatrix} f^i_{11} & \cdots & f^i_{1k} \\ & \ddots & \\ & & f^i_{kk} \end{pmatrix}$$

The flattened version of f is a row vector $\tilde{f} = (f^1_{11}, \ldots, f^1_{kk}, \ldots, \ldots f^i_{11}, \ldots, f^i_{kk}) \in R^D$

Adaption Representations

(Amir Rosenfeld, Incremental Learning Through Deep Adaptation, OpenReview.ICLR2018)

Hence, $\widetilde{F_l^a} = W_l \cdot \widetilde{F}_l$

where $W_l \in R^{C_0 \times C_0}$ is a weight matrix defining linear combinations of the flattened filters of F_l, resulting in C_0 new filters

Unflattening $\widetilde{F_l^a}$ to its original shape results in $F_l^a \in R^{C_0 \times C_i \times k \times k}$, which we call the adapted filters of layer $\emptyset_l$.

Using the symbol $a \otimes b$ as shorthand for *flatten b -> matrix multiply by a -> unflatten*, then we can write

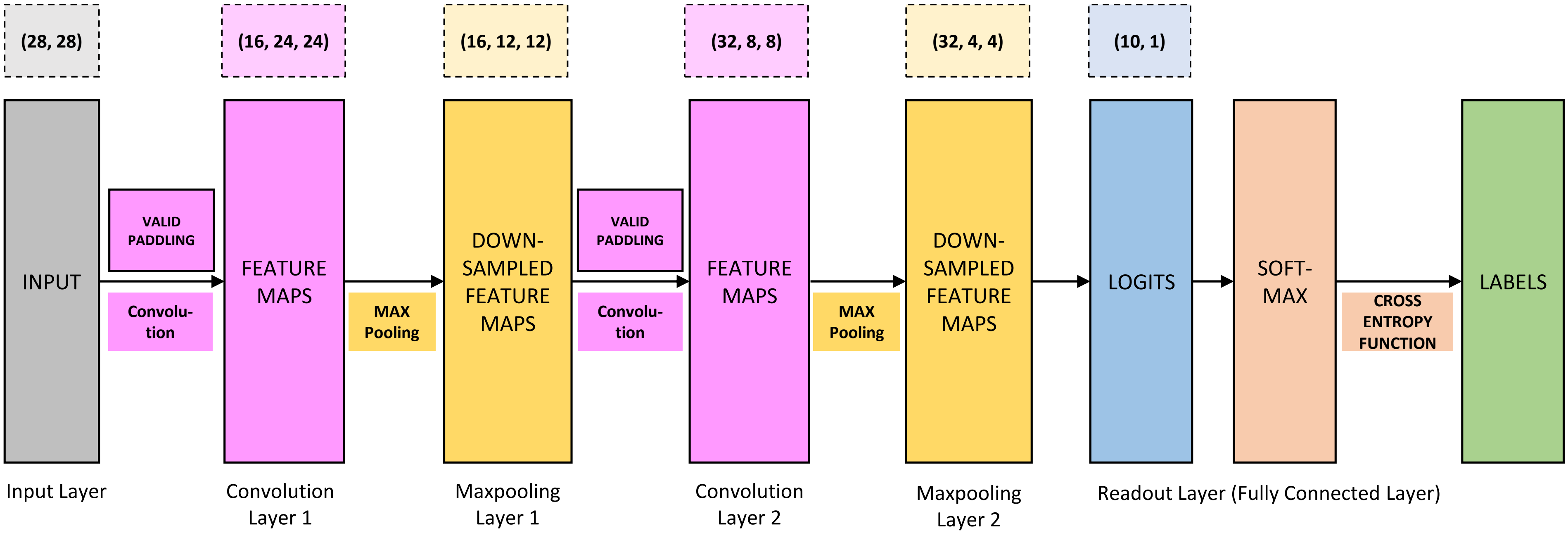
$$F_l^a = W_l \otimes F_l$$

Let x_l be the input of $\emptyset_l$ in the adapted network. For a given switching parameter $\alpha \in \{0,1\}$, we can get the output of the modified layer as follows:

$$\boldsymbol{x_{l+1} = [\alpha(W_l \otimes F_l) + (1-\alpha)F_l] * x_l + \alpha b_l^{\alpha} + (1-\alpha) b_l}$$

Pre Experiments – 2 layer CNN

- Standard 2-layer CNN model:



- New model with controller module:

```
CNNModel (
  (cnn1): controlledConv2 (
    (conv): Conv2d(3, 16, kernel_size=(5, 5), stride=(1, 1))
  )
  (relu1): ReLU ()
  (maxpool1): MaxPool2d (size=(2, 2), stride=(2, 2), dilation=(1, 1))
  (cnn2): controlledConv2 (
    (conv): Conv2d(16, 32, kernel_size=(5, 5), stride=(1, 1))
  )
  (relu2): ReLU ()
  (maxpool2): MaxPool2d (size=(2, 2), stride=(2, 2), dilation=(1, 1))
  (fc1): Linear (800 -> 10)
)
```

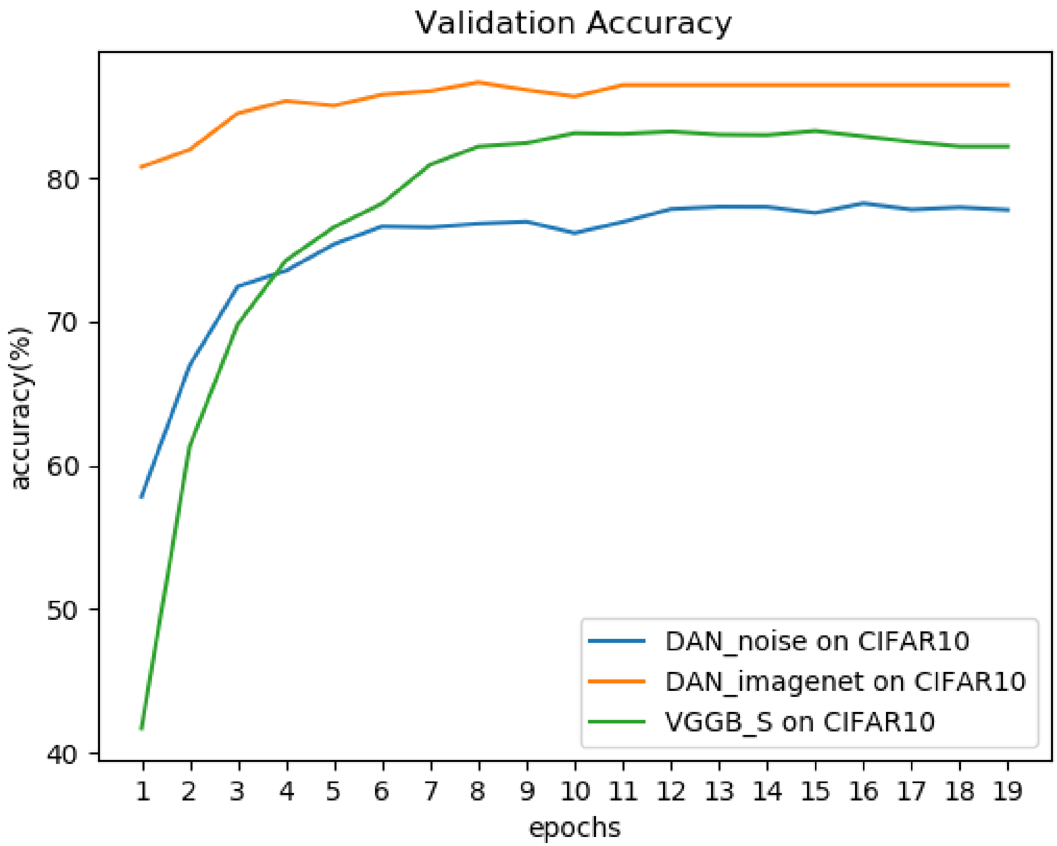
Experiments – VGGB(13_bn)

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Current Results

Dataset

- CIFAR10



Test Accuracy

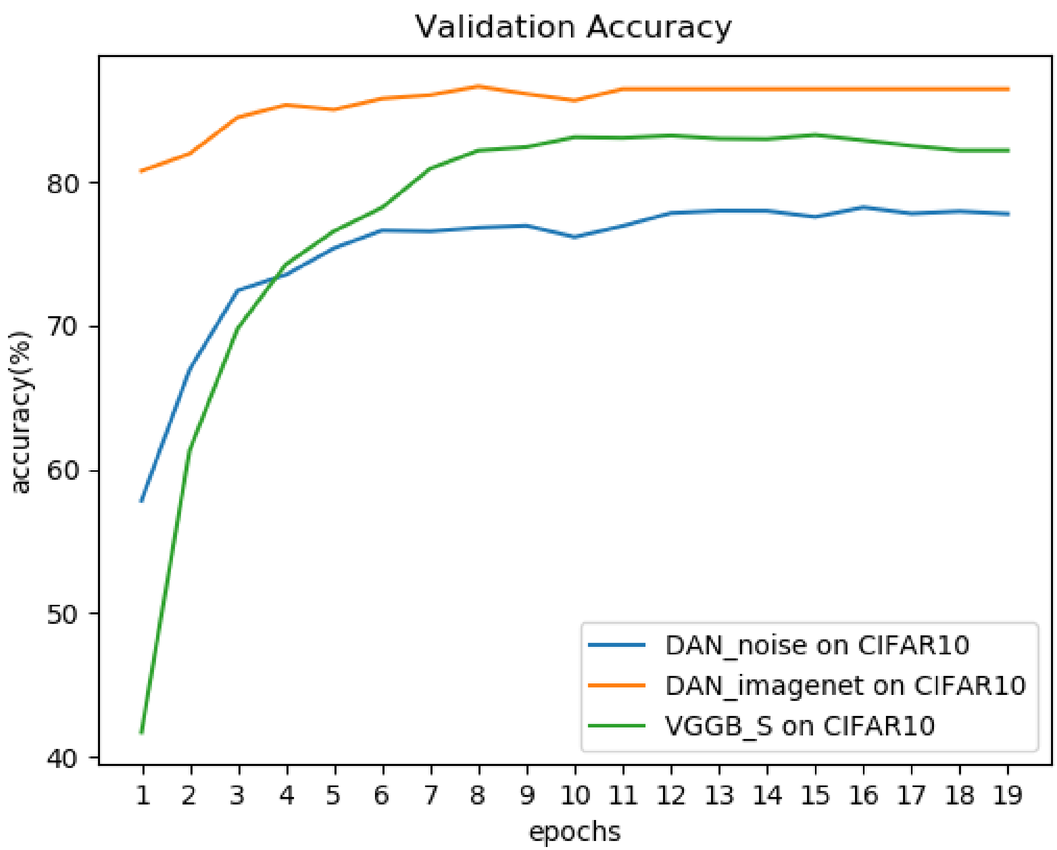
Test Acc	CIFAR10
VGGB S	82.22
DAN noise	77.48
DAN imagenet	86.52

Current Re

Net	C-10	GTSR	SVHN	Caltech	Dped	Oglt	Plnk	Sketch	Perf.	#par
VGG-B(S)	92.5	98.2	96.2	88.2	92.9	86.9	74.5	69.2	87.32	8
VGG-B(P)	93.2	99.0	95.8	92.6	98.7	83.8	73.2	65.4	87.71	8
$\mathbf{DAN_{caltech-256}}$	77.9	93.6	91.8	88.2	93.8	81.0	63.6	49.4	79.91	2.54
DAN_{sketch}	77.9	93.3	93.2	86.9	94.0	85.4	69.6	69.2	83.7	2.54
DAN_{noise}	68.1	90.9	90.4	84.6	91.3	80.6	61.7	42.7	76.29	1.76
$DAN_{imagenet}$	91.6	97.6	94.6	92.2	98.7	81.3	72.5	63.2	86.46	2.76
$DAN_{imagenet+sketch}$	91.6	97.6	94.6	92.2	98.7	85.4	72.5	69.2	**87.76**	3.32

Dataset

- CIFAR10



Test Accuracy

Test Acc	CIFAR10
VGGB S	82.22
DAN noise	77.48
DAN imagenet	86.52

Future work

We will compare our implementation to the authors'.

Complete all experiments on CIFAR10 dataset.

Integrate the binary switch vector α into DAN.

Codes on github:
https://github.com/HarveyYan/ReproDANs
https://github.com/jianingsun21/cnn_incremental-learning
https://github.com/rosenfeldamir/incremental_learning