Deep Residual Learning for Image Recognition

CVPR
(Computer Vision and Pattern Recognition)
2016

Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun
Microsoft Research
{kahe, v-xiangz, v-shren, jiansun}@microsoft.com

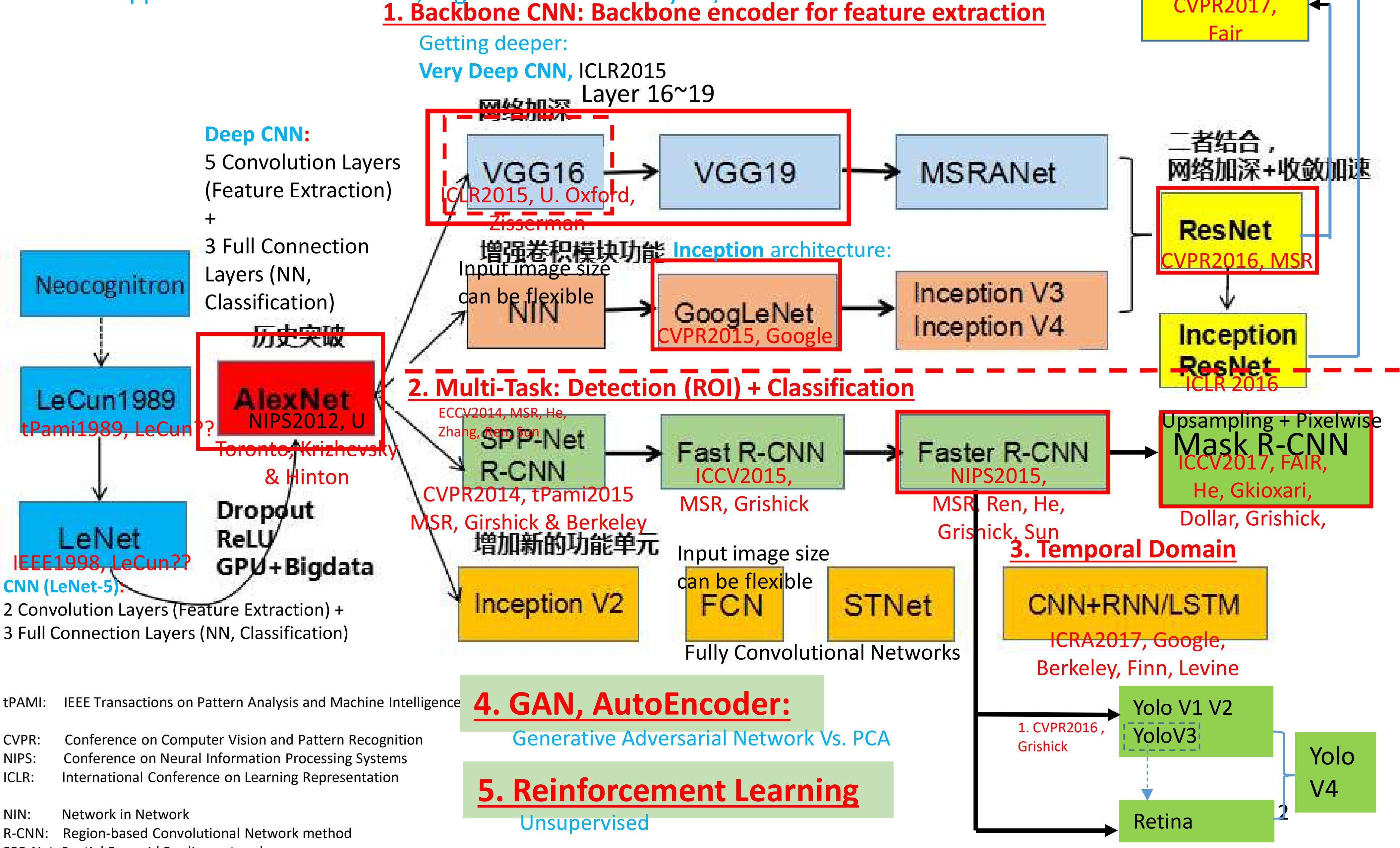
Student: V2 楊佩姍
V3 玄承浩（Tommy）

Advisor: Jenn-Jier James Lien (連震杰)
Robotics Lab, CSIE NCKU

0.1 Deep Learning: History

Which one do I want to be?

1. Basic CNN Creation
2. Modified CNN
3. CNN Application: Data collection, organization and analysis / benchmark



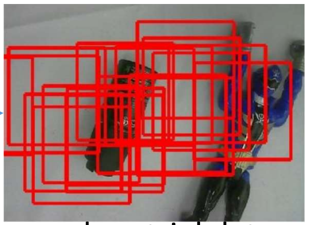
tPAMI:	IEEE Transactions on Pattern Analysis and Machine Intelligence
CVPR:	Conference on Computer Vision and Pattern Recognition
NIPS:	Conference on Neural Information Processing Systems
ICLR:	International Conference on Learning Representation
NIN:	Network in Network
R-CNN:	Region-based Convolutional Network method
SPP-Net:	Spatial Pyramid Pooling networks

AlexNet Example at CMU. Frameork: 5 + 3 Training Stage1

$\delta(j, \theta_i) = 1$ when gripper angle *Θi* corresponds to jth bin for ith image patch
$\delta(j, \theta_i) = 0$, otherwise
B: batch size, J: exists image patch i=1~B
A_{ji}: forward pass binary activations
l_i: the label corresponding to angle *Θi*
AlexNet: 2012年, ImageNet比赛冠军的model
i : current iteration ??

1) Training Stage 1

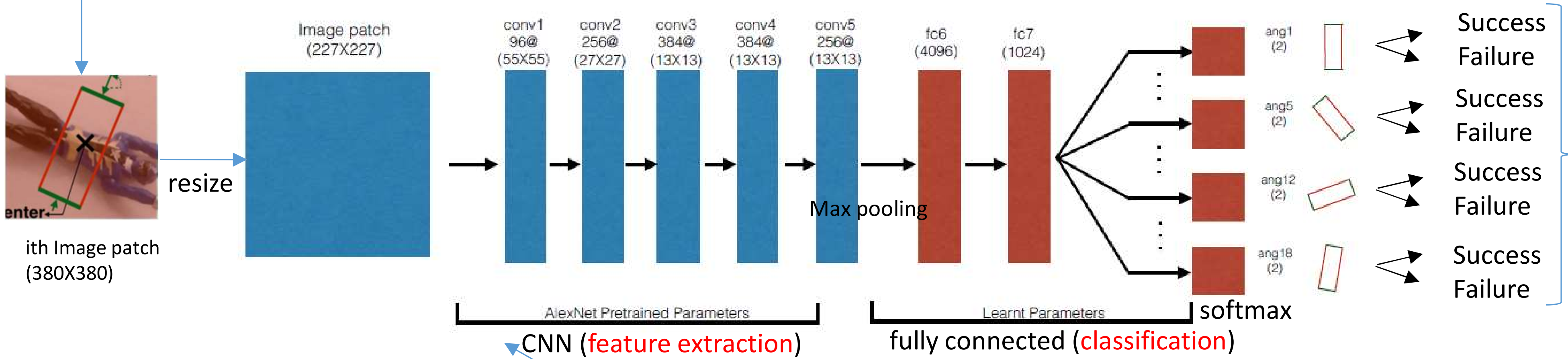
- the network trained use the random trial data (Collection Stage1 data).
- Learning rate :0.01
- trained over 20 epochs


random trial data

-18 angles x 2 (0 or 1) =36 output
-For ith image patch, each time only one of 36 outputs is labelled *li*
F=1 (successful grasp),
others F=0 (fail grasp)
-For ith image patch:
Angle *Θi*=1~18
Bin *j*=1~18

Pretrained using Transfer learning from ImageNet-trained AlexNet

Gaussian initialization



??

$$L_B = \sum_{i=1}^{B} \sum_{j=1}^{N=18} \delta(j, \theta_i) \cdot \mathrm{softmax}(A_{ji}, l_i)$$

Batch loss

deep network learnt in the random trial data (Stage1 data).

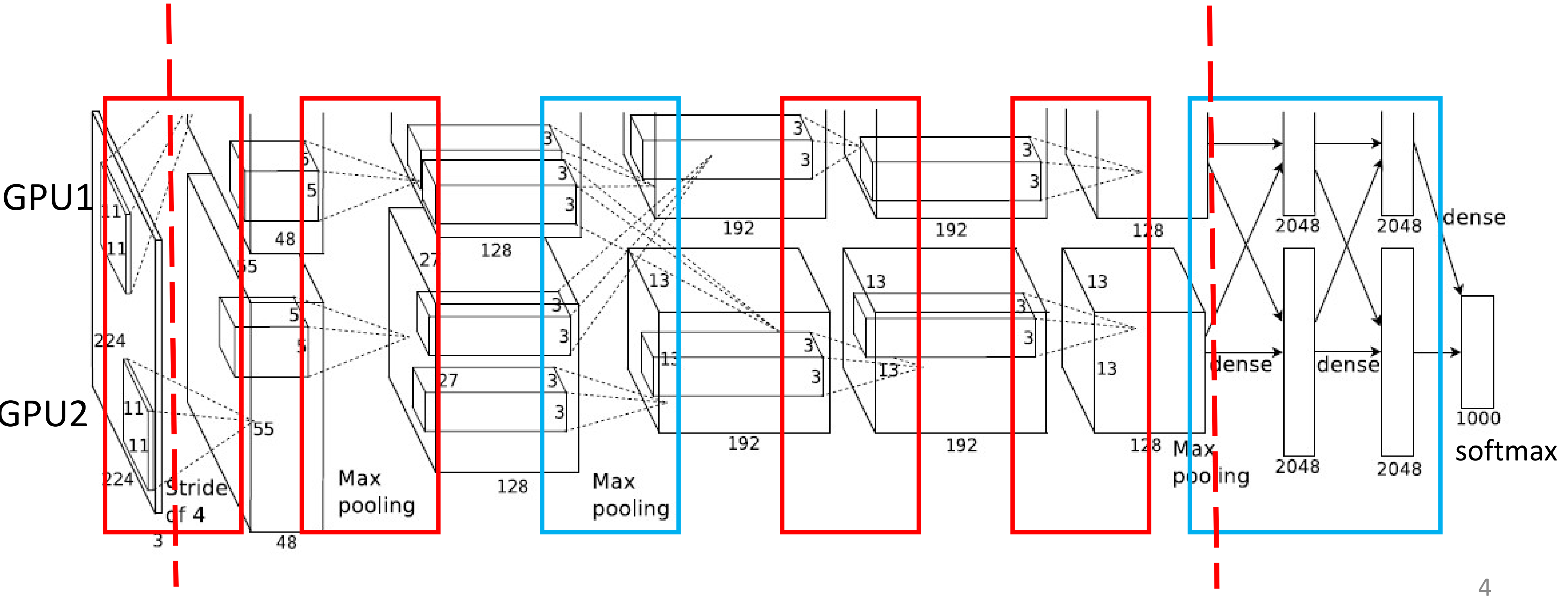
- Used to collect data in Collection Stage2
- As an initialization parameter use in Training Stage 2

Fig. 5. Our CNN architecture is similar to AlexNet [6]. We initialize our convolutional layers from ImageNet-trained Alexnet.

AlexNet The Architecture: (5+3)x2

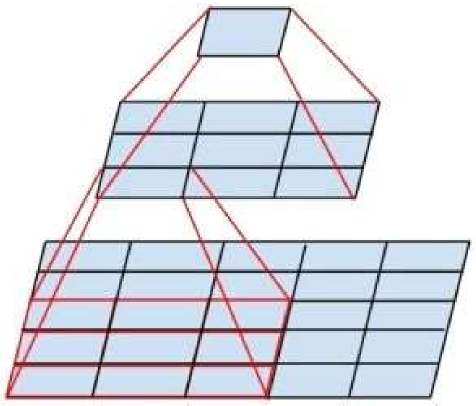
1.2) Training on Multiple GPUs

- A single GTX 580 GPU has only 3GB of memory, which limits the maximum size of the networks that can be trained on it. Therefore we spread the net across two GPUs.
- the GPUs communicate only in certain layers
  (red: no communicate btw GPUs; blue: communicate btw GPUs)
- This scheme reduces our top-1 and top-5 error rates by 1.7% and 1.2%, respectively.



0.0.1 VGG16 & VGG19 (1/3)

1. VGG16 (5 Blocks +3 layers) 相比 AlexNet 的改進為採用連續幾個 3x3 convolution kernel/filter 代替 AlexNet 中較大的 kernel(ex: 11x11, 7x7, 5x5)
   1) 提升深度並提升效果
   2) 減少參數
   3) 3x3 kernel 可以保持較好的圖像性質 reduce para
      - 7x7: 利用 3 個 3x3
      - 5x5: 利用 2 個 3x3
2. Architecture
   1) VGG16: hidden layer x13 (5 blocks) + fully connected x3
   2) VGG19: hidden layer x16 (5 blocks) + fully connected x3


Mini-network replacing the 5x5 convolutions.

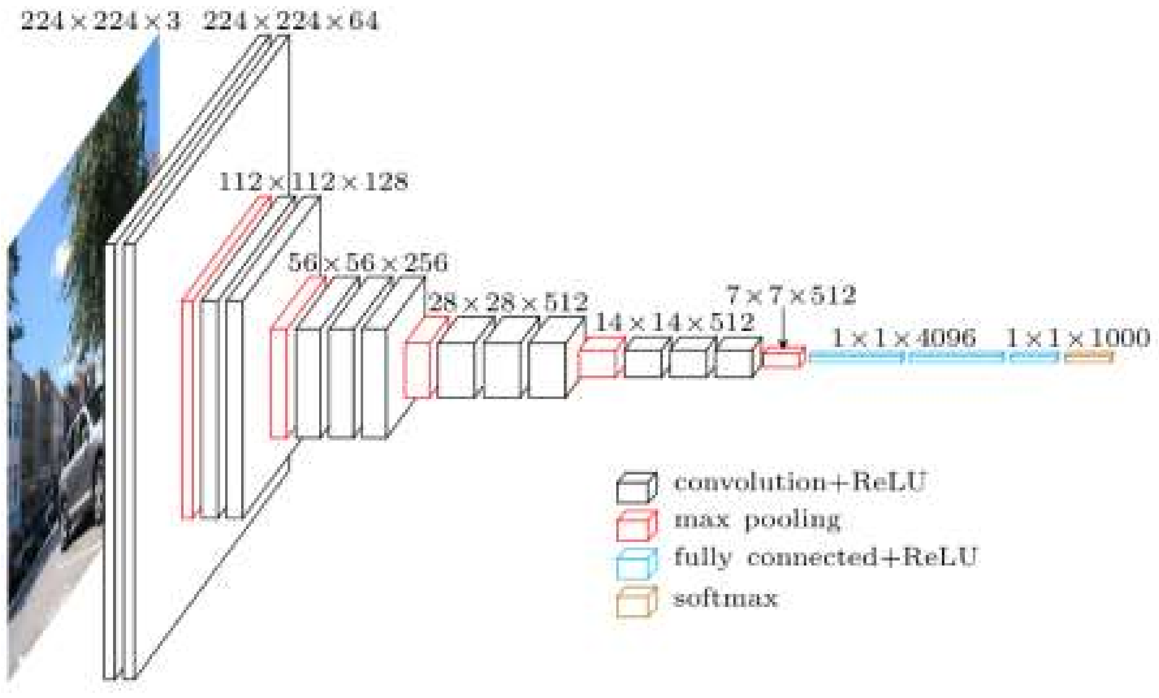
VGG16	VGG19
conv3-64 conv3-64	conv3-64 conv3-64
max pooling	
conv3-128 conv3-128	conv3-128 conv3-128
max pooling	
conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv3-256 conv3-256
max pooling	
conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
max pooling	
conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
max pooling	
FC-4096	
FC-4096	
FC-1000	
softmax	

0.0.1 VGG16 & VGG19 (2/3) ??

Shan: 許多 pretrained 方法使用 VGG model, 相對其他方法, 參數空間很大, 但也因此需要花更長時間 train

3. 優點
   1) 架構簡潔, 皆使用大小同樣的 conv. kernel(3x3) 和 max pooling(2x2) 尺寸
   2) 小 kernel 的組合較大 kernel 好, 減少參數及運算複雜度, 加快運算速度(??)
   3) 通過加深網路可以提升性能
4. 缺點
   1) 耗費更多計算資源, 大多參數來自第一個 fully connected layer(VGG 有三個 FC)

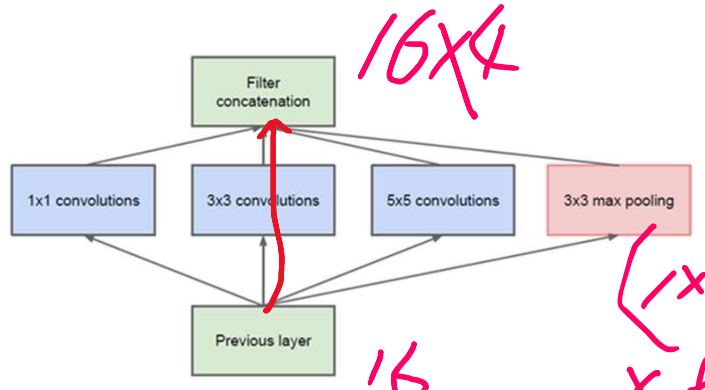
Shan: FC應可砍(??), 不過要先看分類數量決定要使用幾層



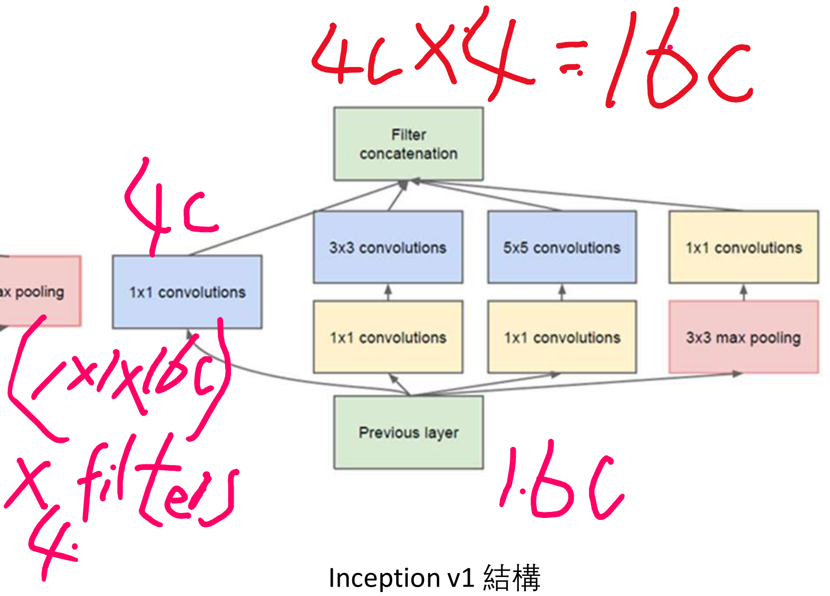
Architecture of VGG16

0.0.2 GoogLeNet (1/n)

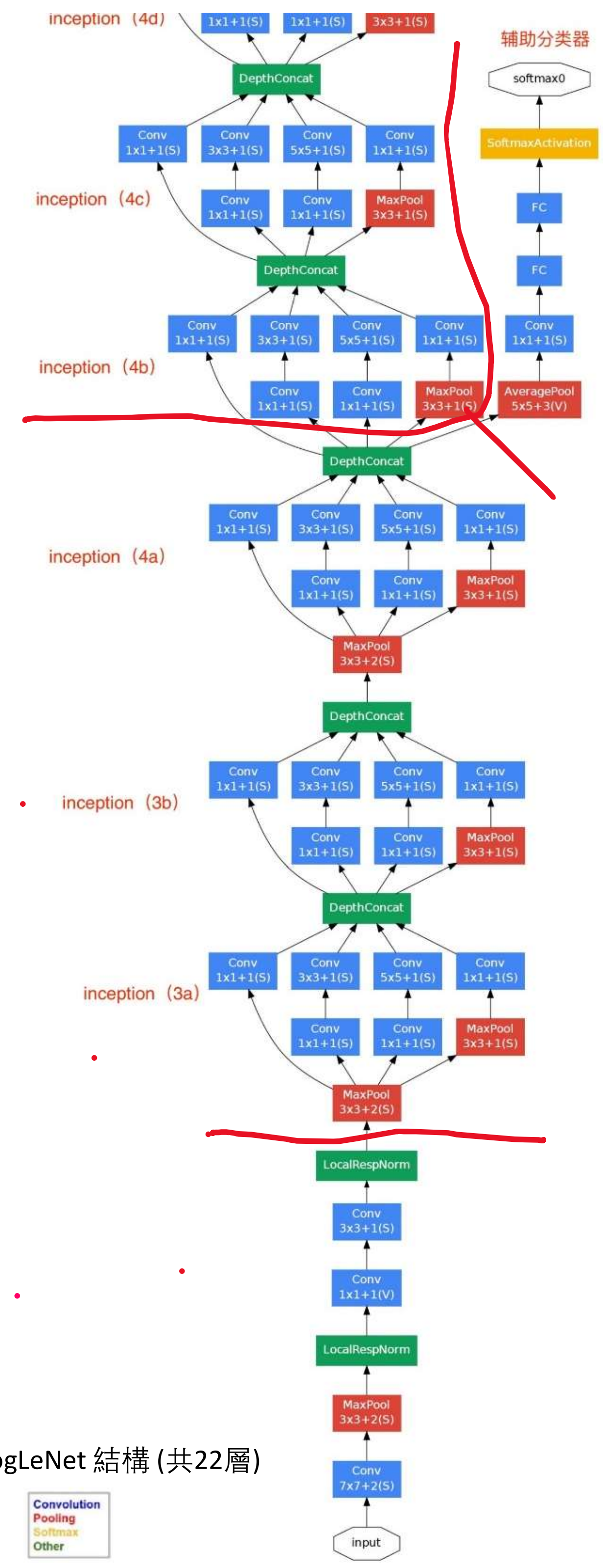
1. GoogLeNet 相比 VGG 加深網路, 選擇加入具稀疏性及高計算效能的 Inception 取代單純的卷積層
2. 多版本發展:
   1) Inception V1: 通過設計一個稀疏網路結構, 但能產生稠密的資料, 既能增加神經網路表現, 又能保證計算資源的使用效率
      (1) 將 CNN 中常用的 convolution kernel, max pooling 堆疊 concatenate 在一起 (通道相連), 增加網路的寬度及網路對尺度的適應性
      (2) Inception 網路能夠提取輸入的每個細節資訊, 5x5 conv 能覆蓋大部分接受層的輸入並 max pooling downsampling 以減少空間大小, 避免 overfitting
      (3) 在每一層 layer 後都要做 ReLU 操作, 增加網路的非線性特徵
      (4) 但是所有的 conv layer都在上一層的所有輸出上來做, 5x5 conv 所需的計算量就太大, 造成 feature map很厚, 為了避免這種情況, 在 conv 和 max pooling 後分別加上了 1x1 conv, 以降低特徵圖厚度 (number of feature map), 這也就形成了Inception v1



Inception 基本結構



Inception v1 結構

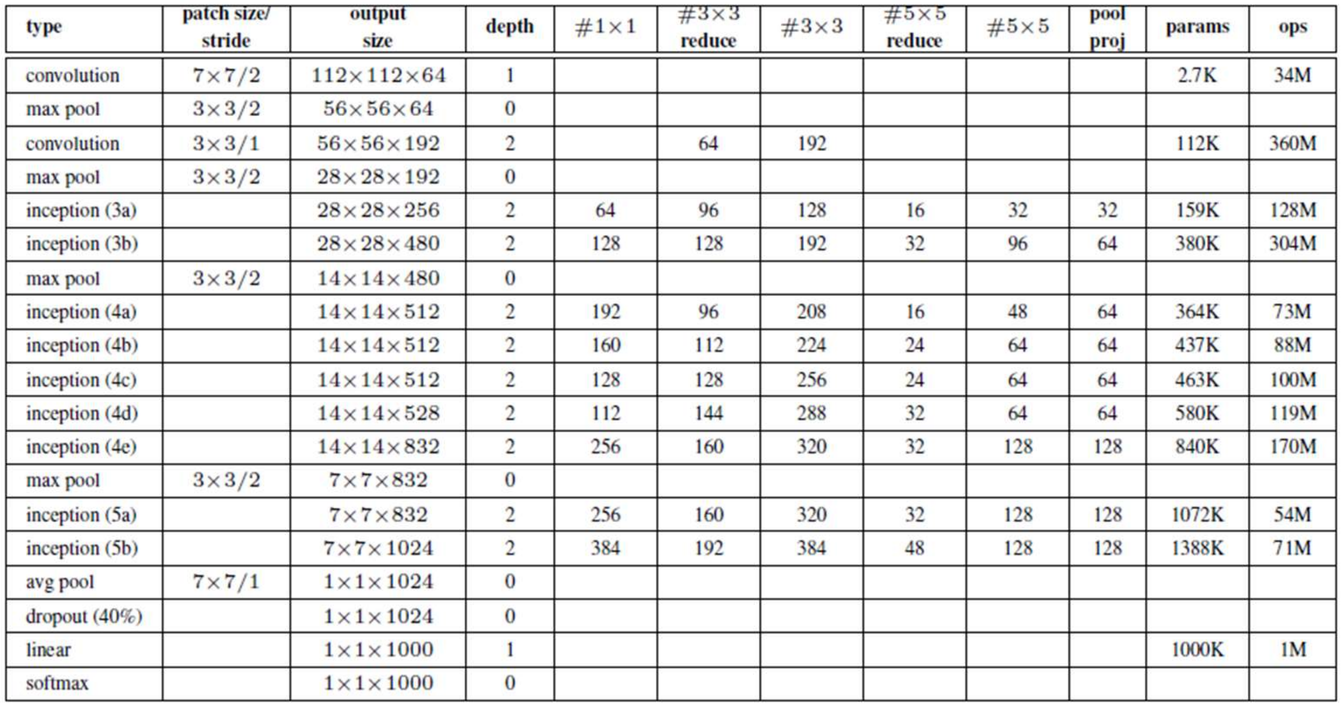


GoogLeNet 結構 (共22層)

Shan: 1x1 conv 可減少維度並修正ReLU(??)
分層stage參數可變, 大多數保留不變(??)

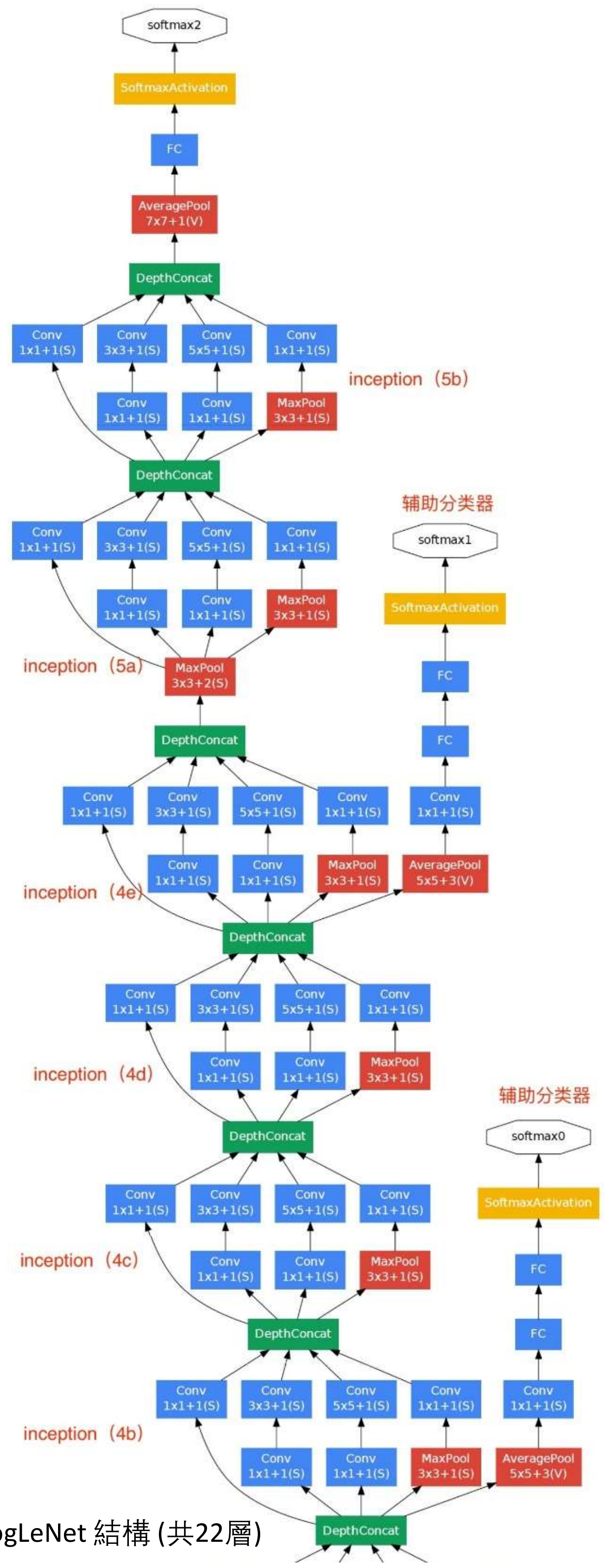
0.0.2 GoogLeNet (2/n)

(5) GoogLeNet 採用了模組化的結構(Inception), 方便增添和修改
(6) 網路最後採用 average pooling 代替 fully connected(想法來自NIN, Network in Network ??), 可將準確率提高0.6%
(7) 實際在最後還是加了一個 fully connected, 為了方便對輸出進行調整
(8) 雖然移除了fully connected layer, 網路中仍使用 Dropout
(9) 避免梯度消失, 網路增加了2個輔助的 softmax 用於向前傳導梯度(輔助分類器)



type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

GoogLeNet 結構細節

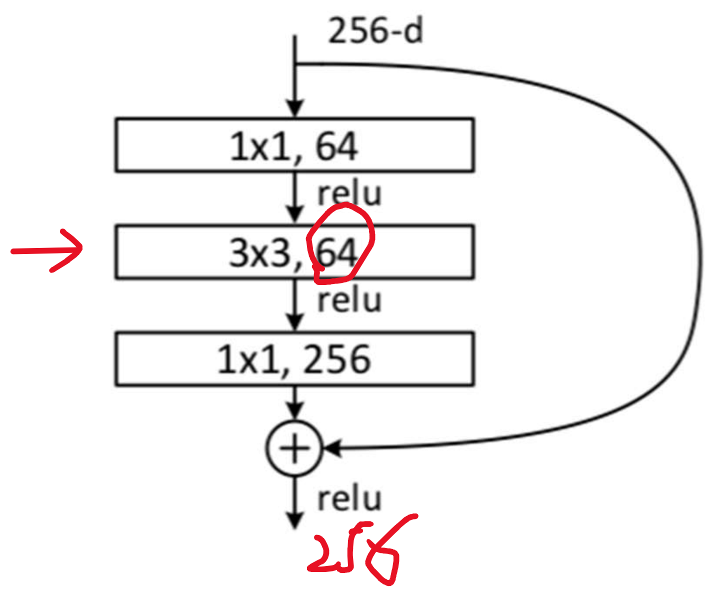


GoogLeNet 結構 (共22層)

Shan: 輔助分類器是將中間某一層的輸出用於分類, 並用一個較小的權重(ex: 0.3) 加到最終分類結果中, 相當做了模型融合, 同時增加 Backpropagation 的梯度訊號。而在實際測試的時候, 這兩個額外的 softmax 會被去掉

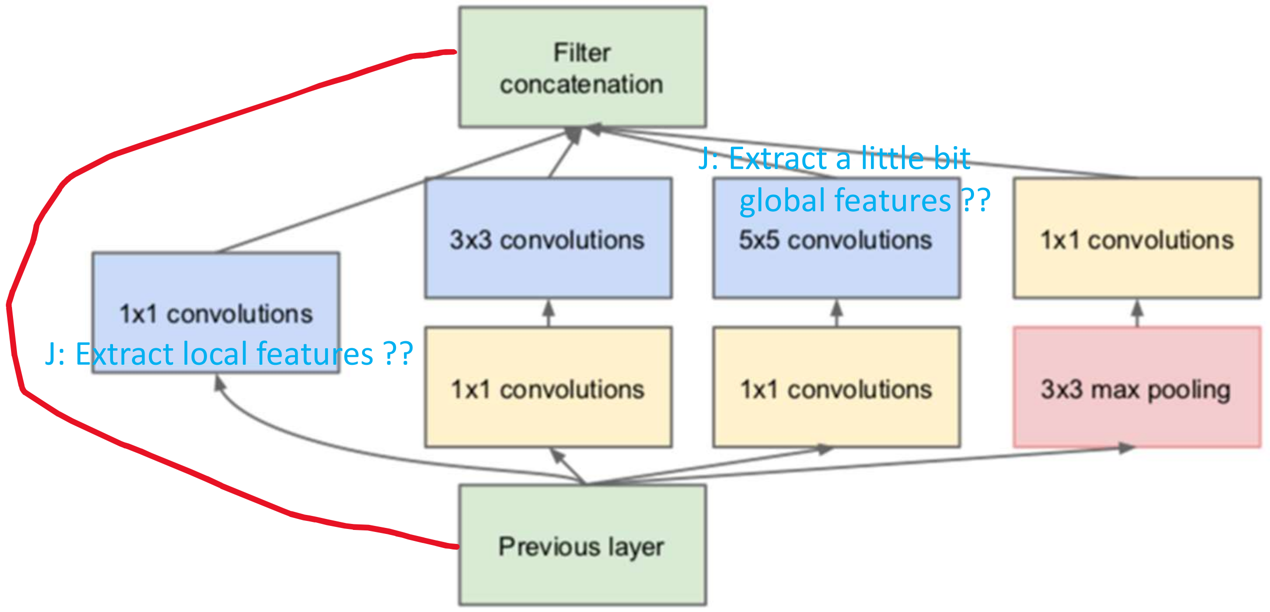
0.2 About ResNet (0/5)

Method Names	Advantages	Disadvantages
1. ResNet/VGG	Stack modules of the same topology → Reduce hyperparameters	Not effective enough (large parameters number)
2. Inception	Carefully designed topology & split-transform-merge strategy → high accuracy & low complexity	1) Too many hyperparameters 2) over-adapting the hyperparameters to a specific dataset



A "bottleneck" residual block for ResNet-50/101/152

RES-51



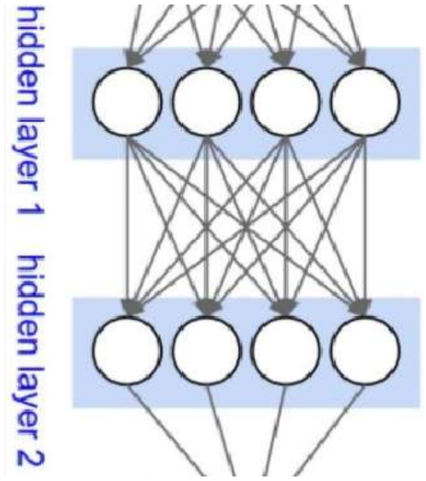
Inception block from GoogLeNet

0.2 About ResNet (1/6) ??

3.1 Residual Learning

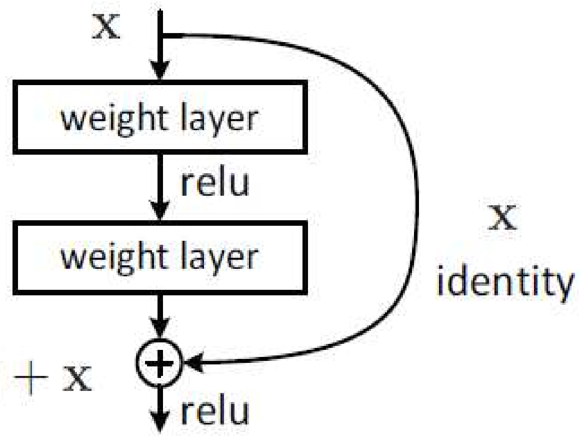
- If the added layers can be constructed as identity mappings, a deeper model should have training error no greater than its shallower counterpart.
- Degradation problem: The degradation problem suggests that the solvers might have difficulties in approximating identity mappings.
- Consider H(x) as an underlying mapping to be fit by a few stacked layers
- Although both forms should be able to asymptotically approximate the desired functions the ease of learning might be different.

$H(x)=\mathcal{F}(\mathbf{x})$



$H(x)-x=\mathcal{F}(\mathbf{x})$

$H(x)=\mathcal{F}(\mathbf{x})+\mathbf{x}$



J: Why shortcut??

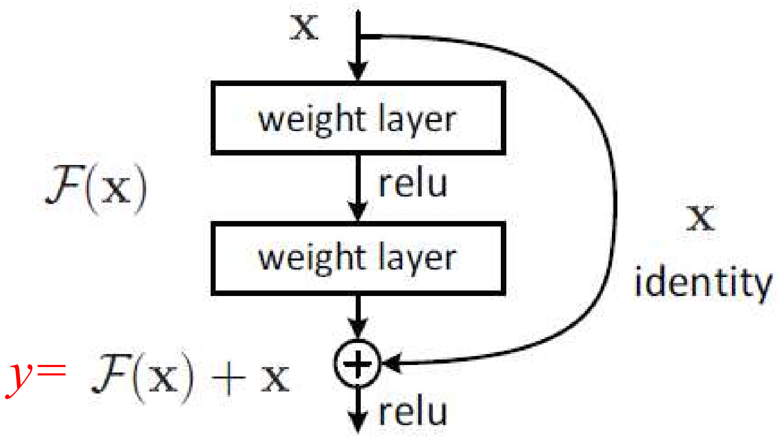
For vanishing gradient problem

0.2 About ResNet (2/6)

3.2 Identity Mapping by Shortcuts

- In this paper we consider a building block defined as

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}. \quad (1)$$



x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$y = \mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

- The shortcut connections in Eqn.(1):
- Introduce neither extra parameter nor computation complexity
- This is not only attractive in practice but also important in our comparisons between plain and residual networks.
- We can fairly compare plain/residual networks that simultaneously have the same number of parameters, depth, width, and computational cost.

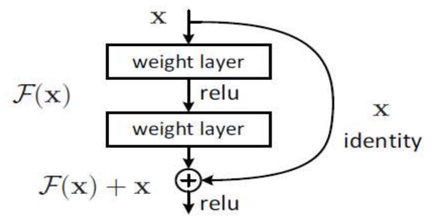
0.2 About ResNet (4/6)

1) Residual Learning - 18, 34,

2) Deep Residual Learning - 50, 101…

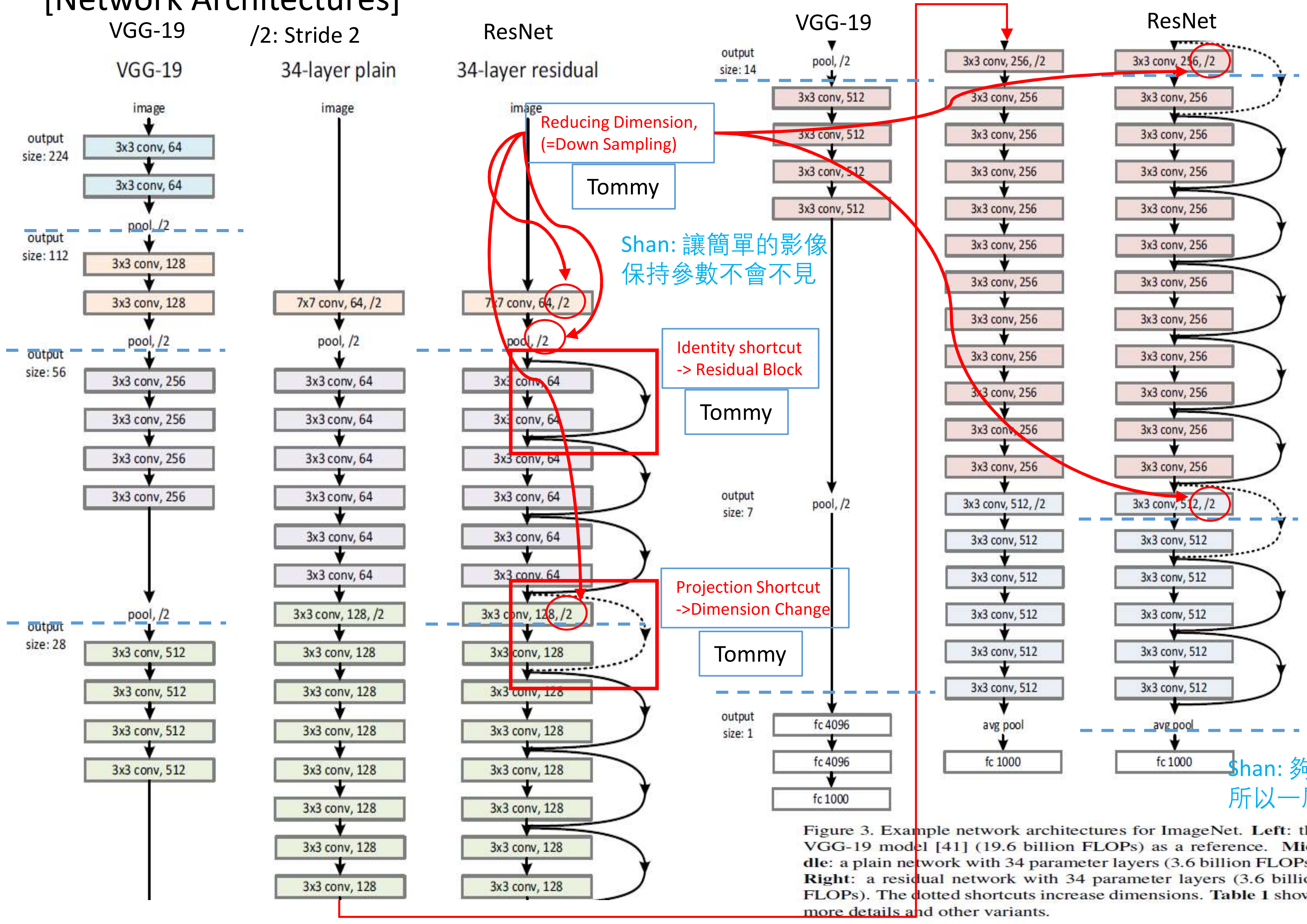
34 w/o bottle neck

[Network Architectures]



x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Tommy



Reducing Dimension,
(=Down Sampling)

Tommy

Shan: 讓簡單的影像保持參數不會不見

Identity shortcut
-> Residual Block

Tommy

Projection Shortcut
->Dimension Change

Tommy

Shan: 夠Deep了，
所以一層就夠

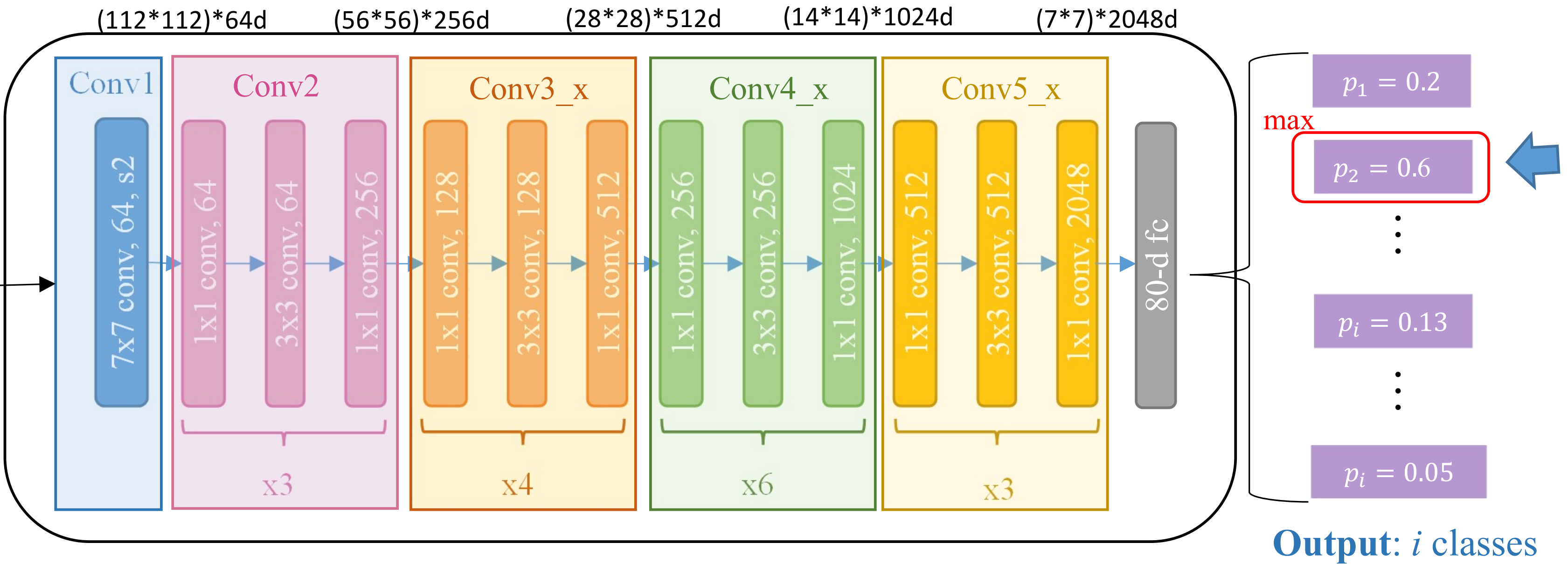
Figure 3. Example network architectures for ImageNet. **Left**: the VGG-19 model [41] (19.6 billion FLOPs) as a reference. **Middle**: a plain network with 34 parameter layers (3.6 billion FLOPs). **Right**: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions. **Table 1** shows more details and other variants.

0.2 About ResNet (6/6)

3.4 ResNet50 Architecture

Input: an RGB image

(take 224x224 pixels for example)
224x224



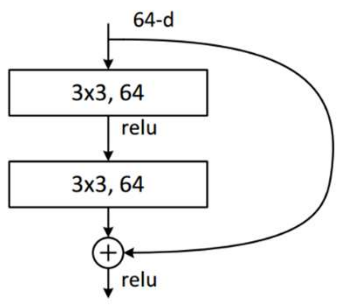
Output: *i* classes

jj: ResNet 50, 101, ...??

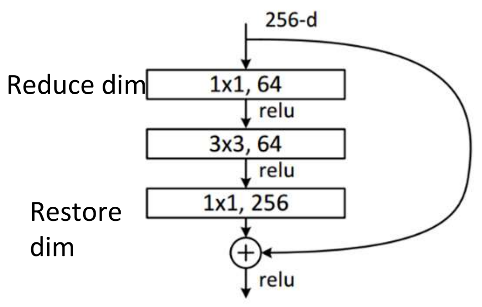
2) Bottleneck: Train the deeper network without additional parameters using bottleneck

(1) The bottleneck architecture is used in very deep networks due to computational considerations.

↓cross過去



A building block



A "bottleneck" building block

Reduce from 256 to 64 for further process, and then recover back to 256

- These bottleneck units have a stack of 3 layers (1x1x256c)x64 filters, (3x3x64c)x64 filters and (1x1x64c)x256 filters).
- The 1x1 layers are just used to reduce (first 1x1 layer) the depth and then restore (last 1x1 layer) the depth of the input.
  - ✓ Suppose the input has a depth of 256c (256-feature map).
    1) Reduce d: The first 1x1 layer (or (1x1x256d) filter) can reduce the depth to 64.
    2) Feature Extraction: The 3x3 convolution layer (or (3x3) filter) can then operate on this less dense (lower depth, such as 64-d) feature vector.
    3) Restore d: The final 1x1 layer (or (1x1x64) filter) will then restore it to its original depth (256).
- The main goal of this design is to increase the efficiency.

Outline

0. Abstract
1. Introduction
2. Related Work
3. Deep Residual Learning
4. Experiments

0. Abstract

Main contribution:

- Present a residual learning framework to ease the training of networks that are substantially deeper than those used previously.
- These residual networks are easier to optimize, and can gain accuracy from considerably increased depth.
- On the ImageNet dataset we evaluate residual nets with a depth of up to 152 layers—8 deeper than VGG nets [41]
- Analysis on CIFAR-10 with 100 and 1000 layers.
- Win the 1st places on the tasks of ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation.

1. Introduction (1/5) - Q

1) Deep networks naturally integrate low/mid/high level features [50] and classifiers in an end-to-end multilayer fashion, and the "levels" of features can be enriched by depth.

2) Recent evidence [41, 44] reveals that network depth is of crucial importance, and the leading results [41, 44, 13, 16] on the challenging ImageNet dataset [36] all exploit "very deep" [41] models.

3) Driven by the significance of depth, a question arises: **Is learning better networks as easy as stacking more layers?**

4) An obstacle to answering this question was the notorious problem of vanishing/exploding gradients. This problem, however, has been largely addressed by normalized initialization [23, 9, 37, 13] and intermediate normalization layers [16].

1. Introduction (2/5) - Q

5) When deeper networks are able to start converging, a degradation problem has been exposed: with the network depth increasing, accuracy gets saturated and then degrades rapidly. Unexpectedly, such degradation is **not caused by overfitting**.

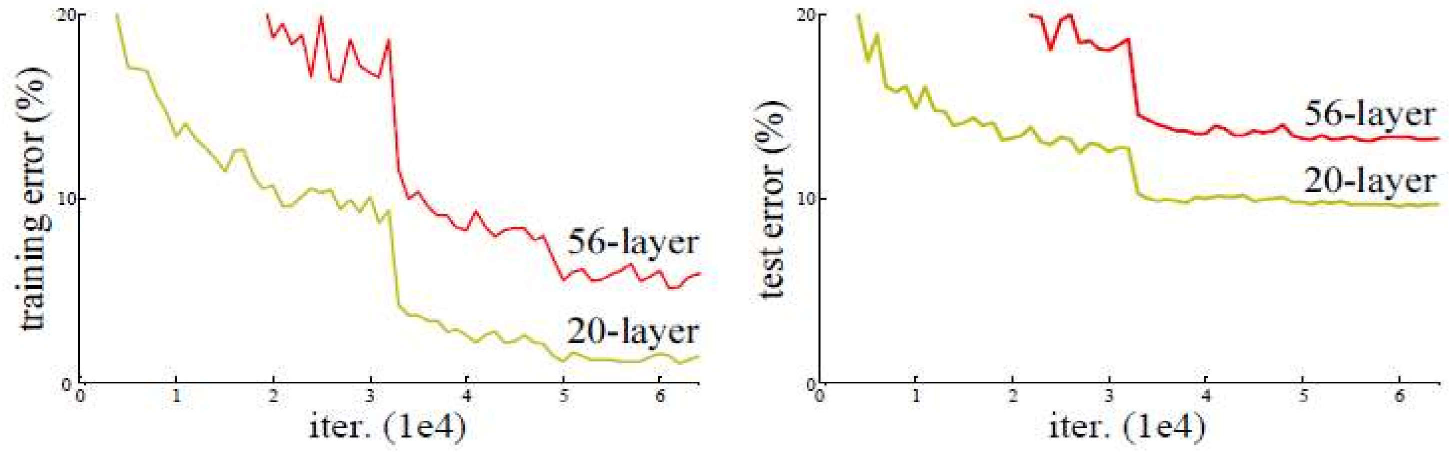
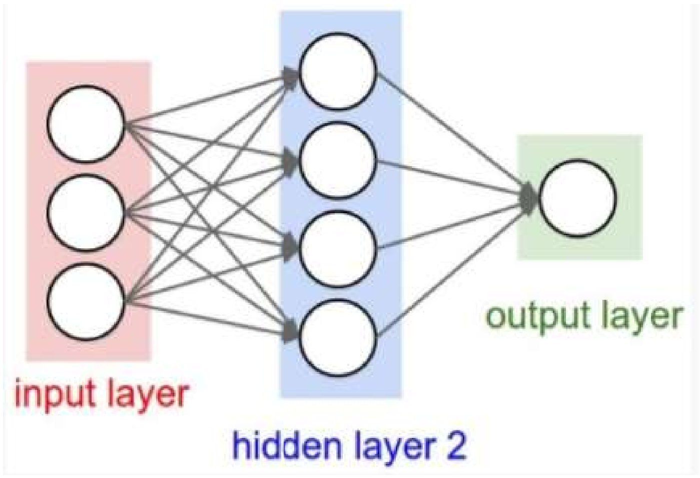
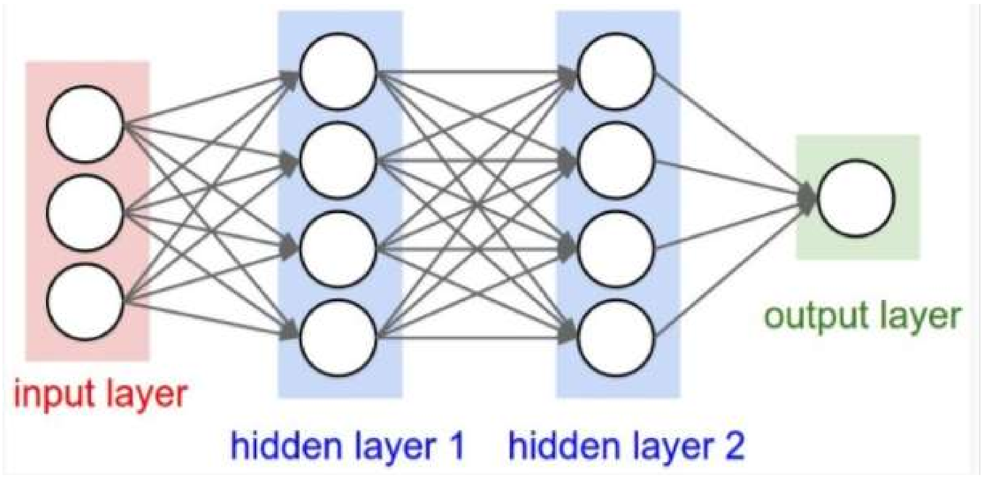


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

1. Introduction (3/5) - Q

6) The degradation (of training accuracy) indicates that not all systems are similarly easy to optimize.





$F(x)_2$

If F(x)1=x, then both networks will be the same

$F(x)_1$ $F(x)_2$

If $F(x)_1 = x$, that a deeper model should produce no higher training error than its shallower counterpart.

J: Original image has complicate texture or simple texture, simple texture doesn't need deep layers to extract

1. Introduction (4/5) - A

1) In this paper, we address the degradation problem by introducing a deep residual learning framework.

- Instead of hoping each few stacked layers directly fit a desired underlying mapping*(H(x))*, we explicitly let these layers fit a residual mapping*(f(x))*.

- To the extreme, if an identity mapping were optimal, it would be easier to push the residual to zero than to fit an identity mapping by a stack of nonlinear layers.

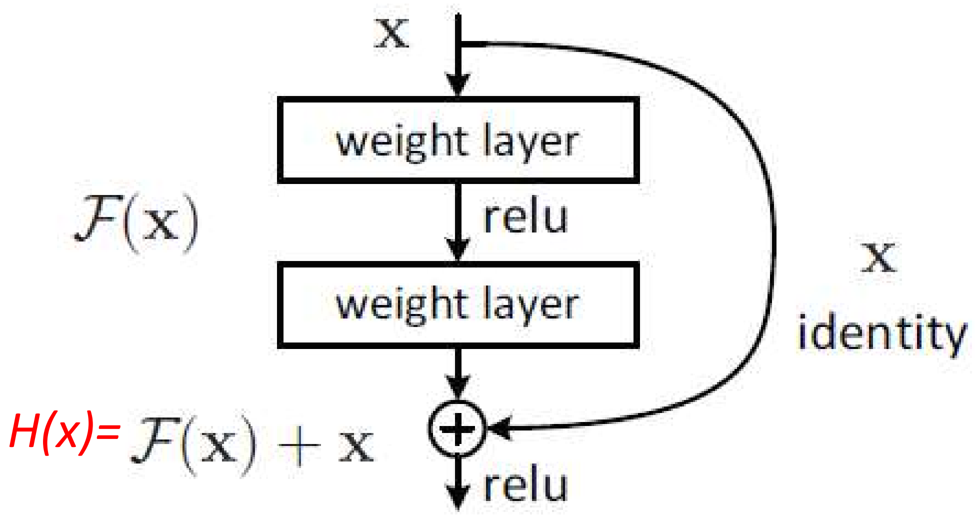

x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
H(x)= $\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 2. Residual learning: a building block.

If F(x)1=x, then both Networks will be the same. Therefore, we can connect as this network by adding x, so when F(x) = 0, H(x) = x (shortcut)

1. Introduction (5/5) - A

2) We present comprehensive experiments on ImageNet

- Our extremely deep residual nets are easy to optimize, but the counterpart “plain” nets (that simply stack layers) exhibit higher training error when the depth increases.
- Our deep residual nets can easily enjoy accuracy gains from greatly increased depth.
- Similar phenomena are also shown on the CIFAR-10 set.
- Our 152-layer residual net is the deepest network ever presented on ImageNet,while still having lower complexity than VGG nets.
- win the 1st places on: ImageNet detection, ImageNet localization, COCO detection and COCO segmentation in ILSVRC & COCO 2015 competitions.

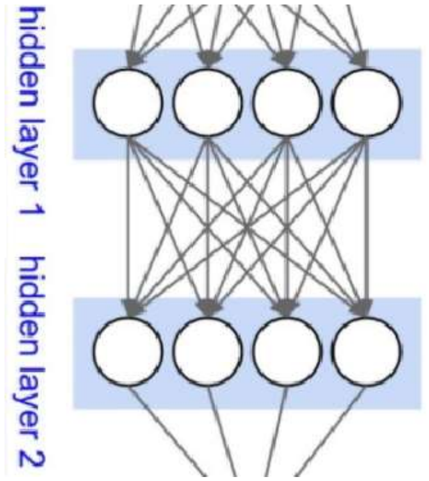
3. Deep Residual Learning

identity mappings: f(x)=x
underlying mapping: Best result

3.1 Residual Learning

- If the added layers can be constructed as identity mappings, a deeper model should have training error no greater than its shallower counterpart.
- The degradation problem suggests that the solvers might have difficulties in approximating identity mappings.
- Consider H(x) as an underlying mapping to be fit by a few stacked layers.
- Although both forms should be able to asymptotically approximate the desired functions the ease of learning might be different.

$H(x)=\mathcal{F}(\mathrm{x})$

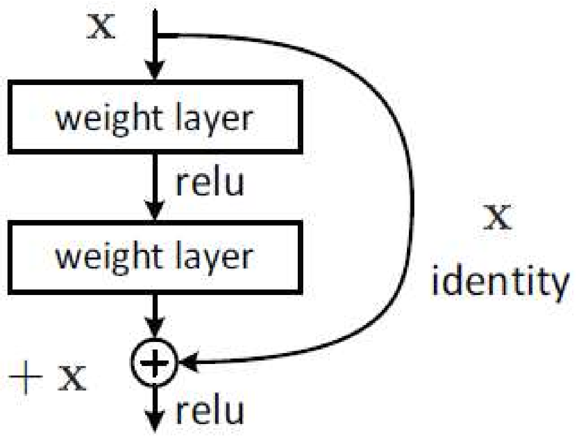


Learn *H(x)*

H(x): Expected result
x: Input or original data
F(x): Residual??

$H(x)-x=\mathcal{F}(\mathrm{x})$

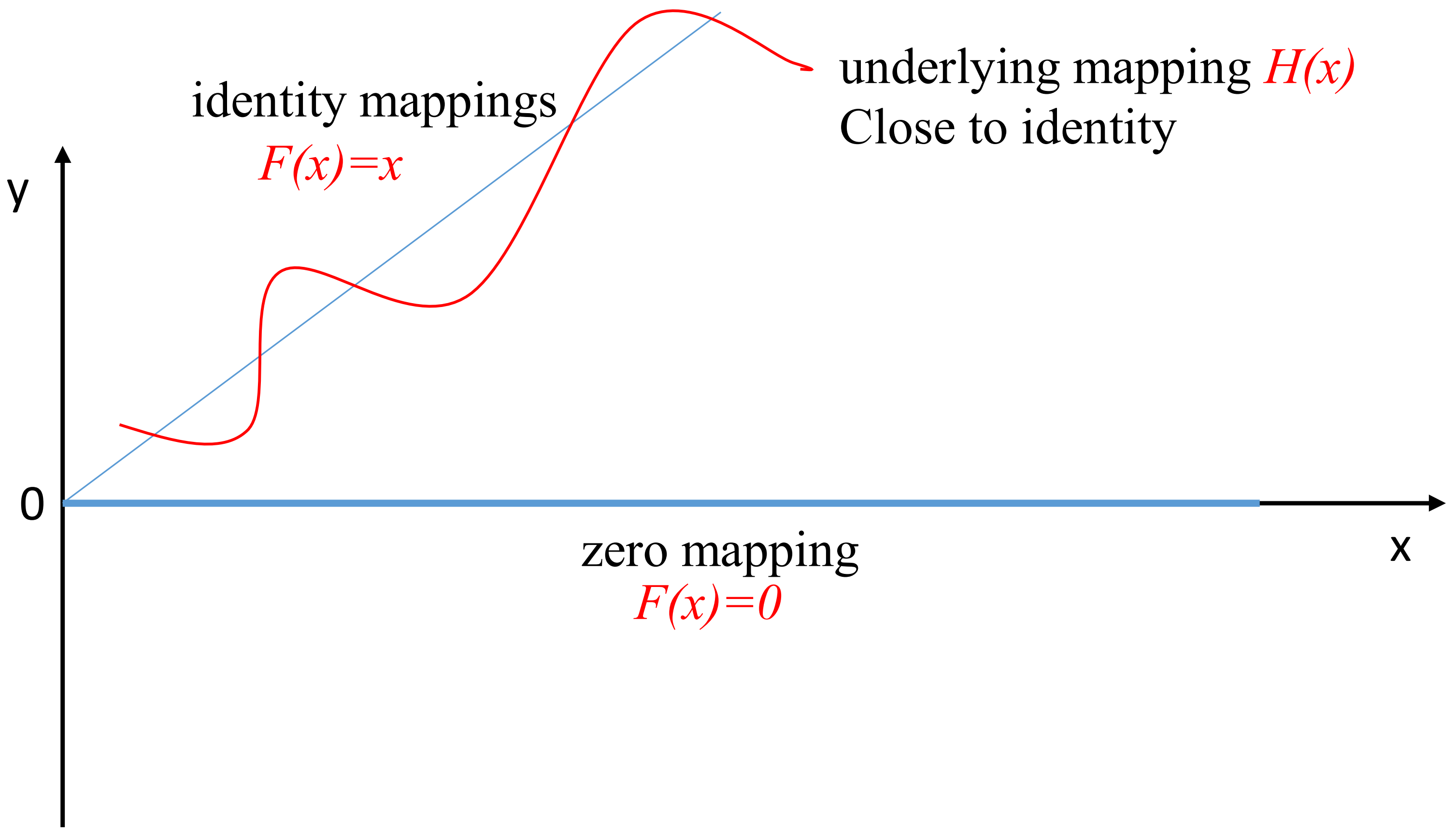
$H(x)=\mathcal{F}(\mathrm{x})+\mathrm{x}$



Learn *H(x)-x*

3. Deep Residual Learning

- In real cases, it is unlikely that identity mappings are optimal， If the optimal function is closer to an identity mapping than to a zero mapping, it should be easier for the solver to find the perturbations with reference to an identity mapping.



3. Deep Residual Learning

- We show by experiments that the learned residual functions in general have small responses, suggesting that identity mappings provide reasonable preconditioning.

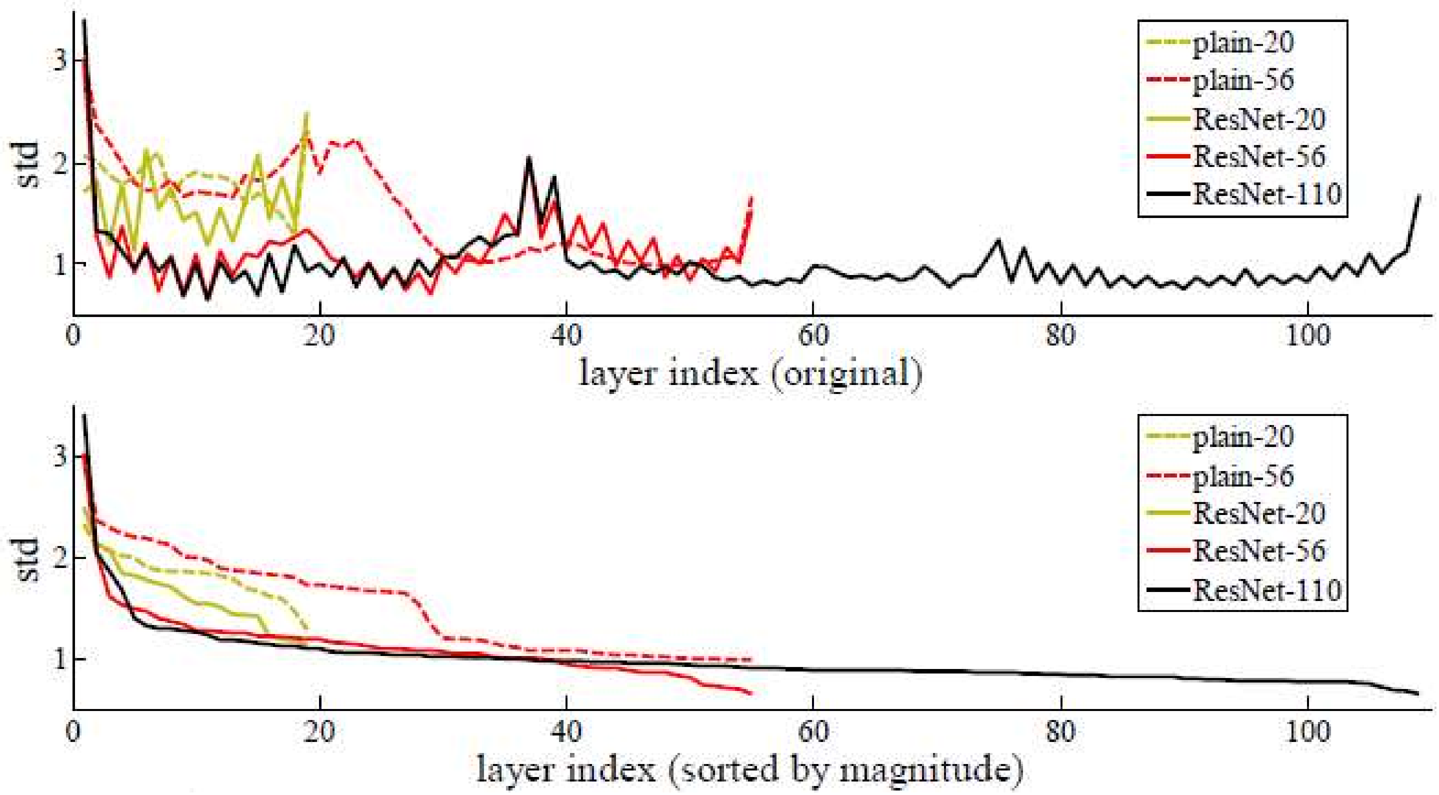
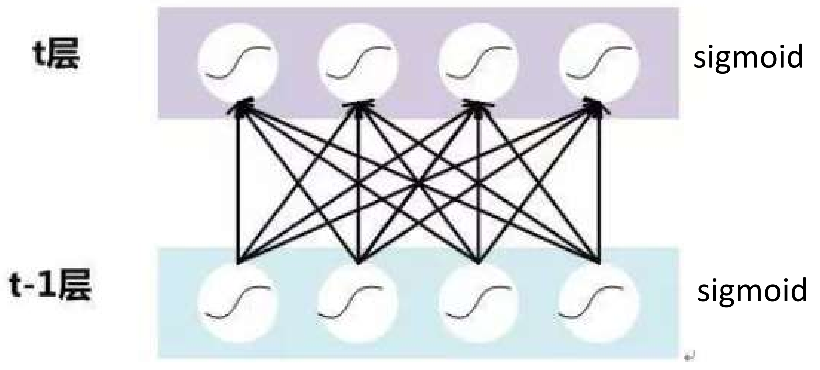


Figure 7. Standard deviations (std) of layer responses on CIFAR-10. The responses are the outputs of each 3×3 layer, after BN and before nonlinearity. **Top**: the layers are shown in their original order. **Bottom**: the responses are ranked in descending order.

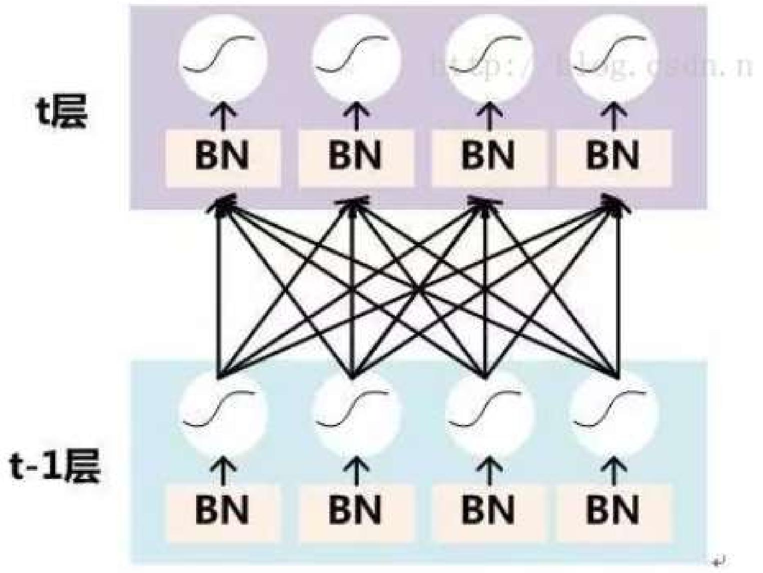
3. Deep Residual Learning

Batch normalization: Reduce vanishing problem

BN 就是通過規範化手段, 把每層神經網絡的輸入值的分佈, 強行拉回到均值為 0, 方差為 1 的標準常態分佈(normal distribution)



DNN without BN



DNN with BN

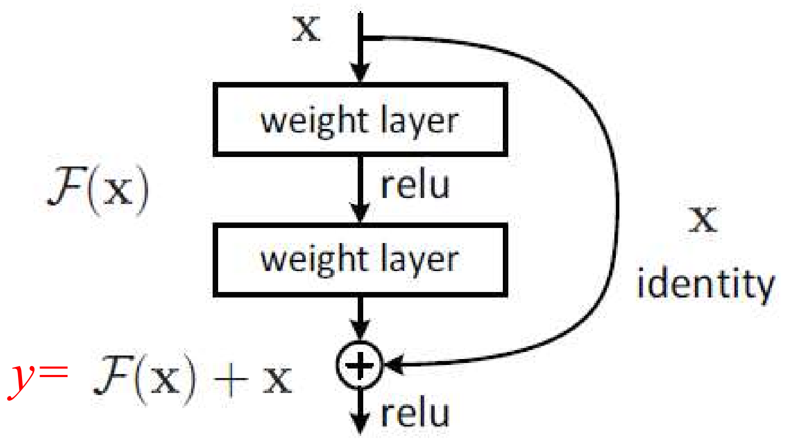
3. Deep Residual Learning

identity mappings: f(x)=x
underlying mapping: Best result

3.2 Identity Mapping by Shortcuts

- in this paper we consider a building block defined as

$$\mathrm{y} = \mathcal{F}(\mathrm{x}, \{W_i\}) + \mathrm{x} \quad (1)$$


x
weight layer
$\mathcal{F}(\mathrm{x})$
relu
weight layer
x
identity
$y = \mathcal{F}(\mathrm{x}) + \mathrm{x}$
relu

- The shortcut connections in Eq.(1) introduce neither extra parameter nor computation complexity, This is not only attractive in practice but also important in our comparisons between plain and residual networks. We can fairly compare plain/residual networks that simultaneously have the same number of parameters, depth, width, and computational cost.

3. Deep Residual Learning

3.3 Network Architectures

/2: Stride 2

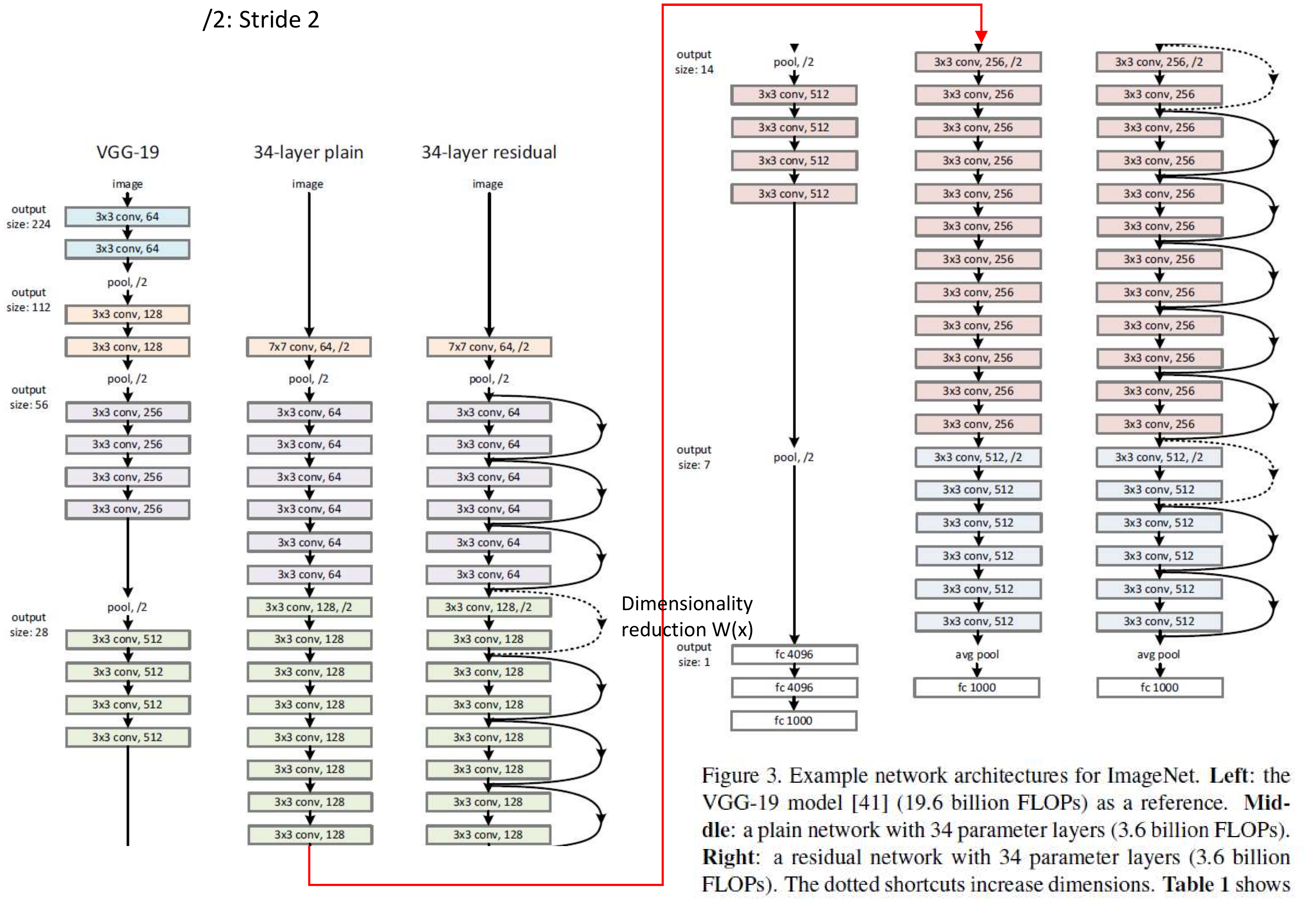


Figure 3. Example network architectures for ImageNet. **Left**: the VGG-19 model [41] (19.6 billion FLOPs) as a reference. **Middle**: a plain network with 34 parameter layers (3.6 billion FLOPs). **Right**: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions. **Table 1** shows more details and other variants.

3. Deep Residual Learning

Plain Network:

- Our plain baselines are mainly inspired by the philosophy of VGG nets[41]. The convolutional layers mostly have 3x3 filters and follow two simple design rules:

  I. For the same output feature map size, the layers have the same number of filters.

  II. If the feature map size is halved, the number of filters is doubled so as to preserve the time complexity per layer.

- We perform down sampling directly by convolutional layers that have a stride of 2.

- The network ends with a global average pooling layer and a 1000-way fully-connected layer with softmax.

3. Deep Residual Learning

Residual Network:

- Based on the above plain network, we insert shortcut connections.
- The identity shortcuts(Eqn.(1)) can be directly used when the input and output are of the same dimensions.
- When the dimensions increase(dotted line shortcuts in Fig.3), we consider two options.

I. The shortcut still performs identity mapping, with extra zero entries padded for increasing dimensions. This option introduces no extra parameter.

II. The projection shortcut in Eq.(2) is used to match dimensions (done by 1x1 convolutions)

3. Deep Residual Learning

3.4 Implementation:

- The image is resized with its shorter side randomly sampled in [256; 480] for scale augmentation[41](VGG).
- A 224*224 crop is randomly sampled from an image or its horizontal flip, with the per-pixel mean subtracted.
- Adopt batch normalization(BN)[16] right after each convolution and before activation.
- Initialize the weights as in[13]
- Mini-batch size of 256.
- The learning rate starts from 0.1 and is divided by 10 when the error plateaus.
- Models are trained for up to 6.0×10^4 iterations.
- Use a weight decay of 0.0001 and a momentum of 0.9.
- Do not use dropout[14], following the practice in[16]
- In testing, for comparison studies we adopt the standard 10-crop testing[21]

VGG TRAINING IMAGE SIZE

- Let S be the smallest side of an rescaled training image, from which the ConvNet input is cropped (224*224). We consider two approaches for setting the training scale S

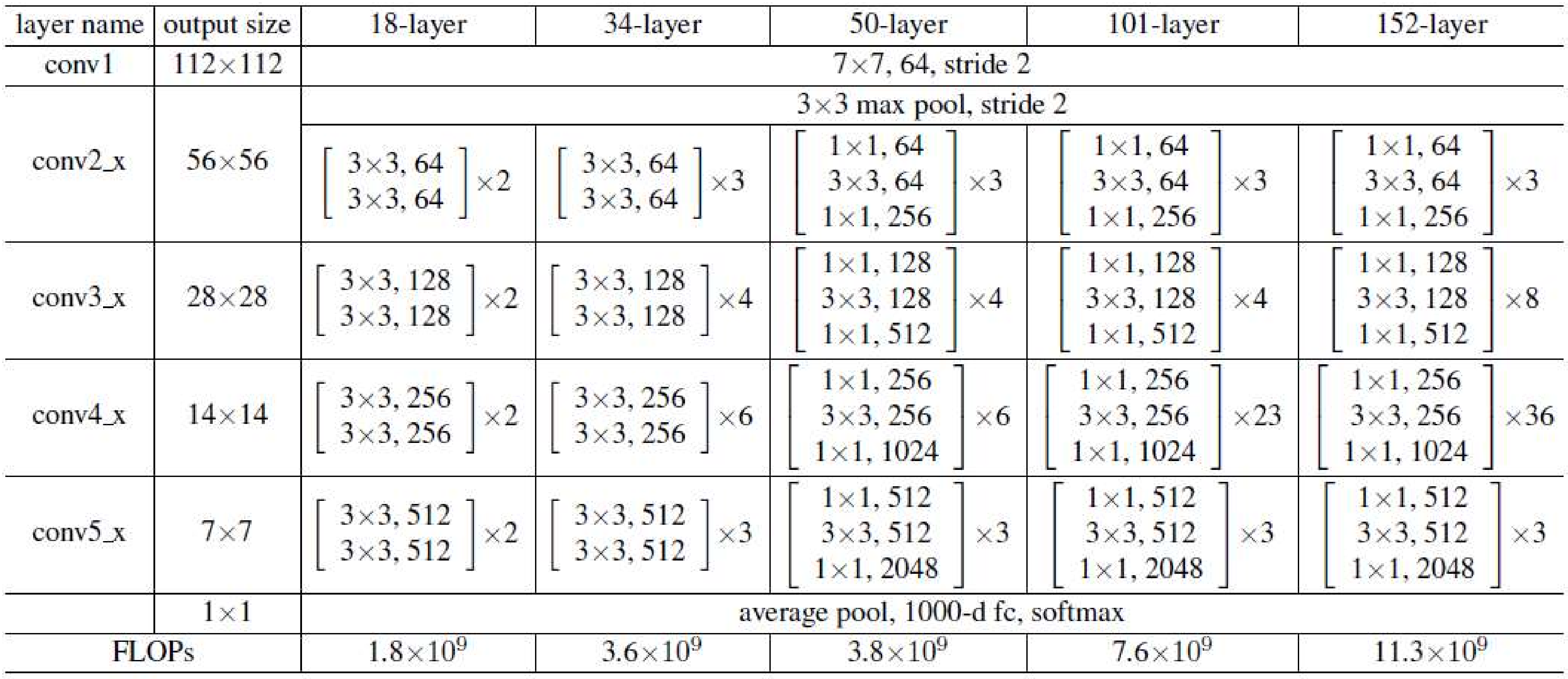
  1) The first is to fix S, which corresponds to single-scale training. We evaluated models trained at two fixed scales: S =256(which has been widely used in the prior art (Krizhevsky et al., 2012)) and S = 384.

  2) The second approach to setting S is multi-scale training, where each training image is individually rescaled by randomly sampling S from a certain range [Smin, Smax] (we used Smin = 256 and Smax = 512).

4. Experiments

4.1 ImageNet Classification

- Evaluate our method on the ImageNet 2012 classification dataset [36] that consists of 1000 classes. We evaluate both top-1 and top-5 error rates.



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

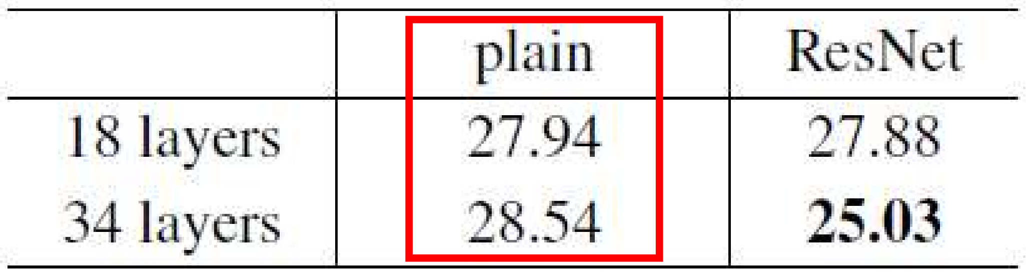
Table 1. Architectures for ImageNet. Building blocks are shown in brackets (see also Fig. 5), with the numbers of blocks stacked. Down-sampling is performed by conv3_1, conv4_1, and conv5_1 with a stride of 2.

J: Why ResNet50 常用??

4. Experiments

Plain Networks

- First evaluate 18-layer and 34-layer plain nets. The results in Table 2 show that the deeper 34-layer plain net has higher validation error than the shallower 18-layer plain net.



	plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	**25.03**

Table 2. Top-1 error (%, 10-crop testing) on ImageNet validation. Here the ResNets have no extra parameter compared to their plain counterparts. Fig. 4 shows the training procedures.

4. Experiments

- To reveal the reasons, in Fig.4(left) we compare their training/validation errors during the training procedure.

    I. We have observed the degradation problem.
    II. We argue that this optimization difficulty is unlikely to be caused by vanishing gradients.
    III. The reason for such optimization difficulties will be studied in the future.

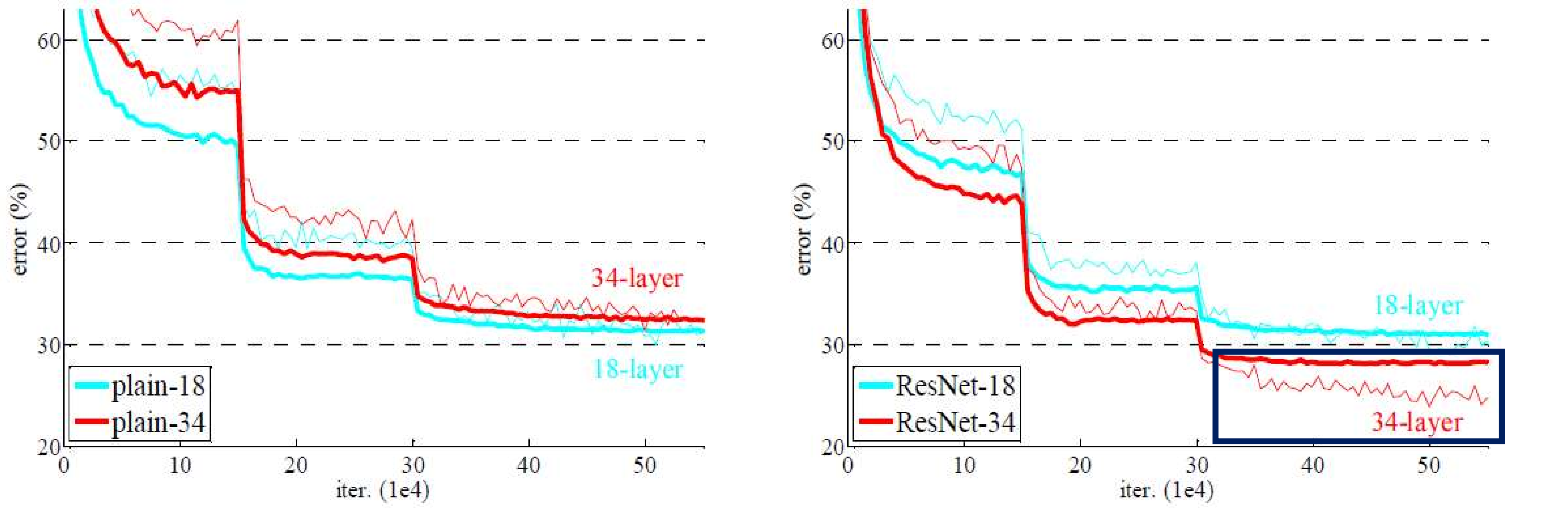


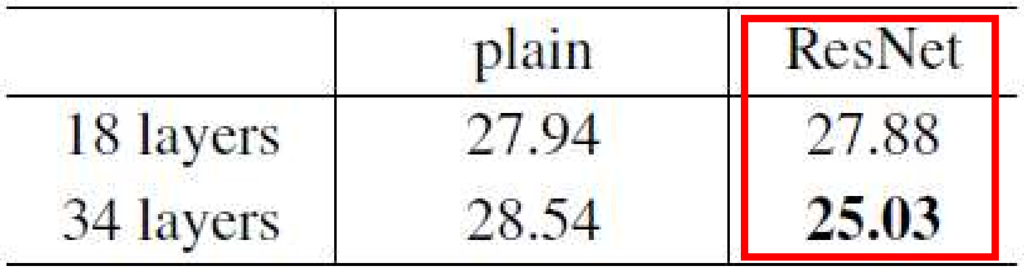
Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

4. Experiments

Residual Networks.

- Evaluate 18-layer and 34- layer residual nets.

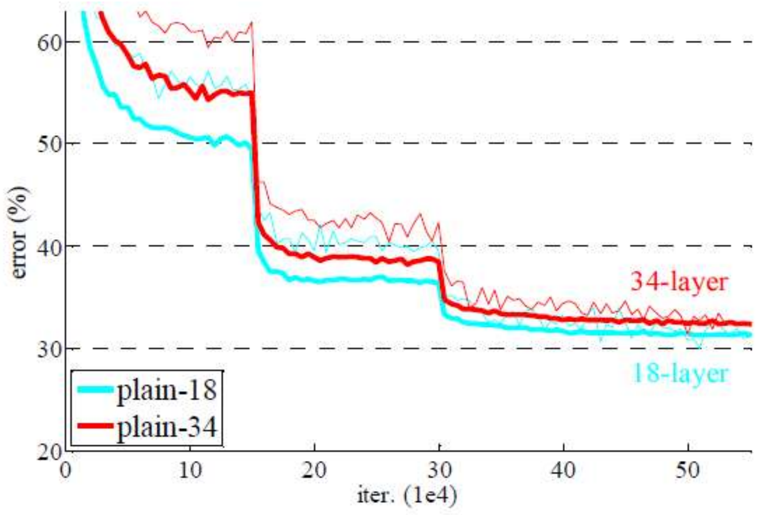
  I. The baseline architectures are the same as the above plain nets.
  II. A shortcut connection is added to each pair of 3x3 filters(zero-padding for increasing dimensions)
  III. Have no extra parameter compared to the plain counterparts.



	plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	**25.03**

Table 2. Top-1 error (%, 10-crop testing) on ImageNet validation. Here the ResNets have no extra parameter compared to their plain counterparts. Fig. 4 shows the training procedures.

4. Experiments



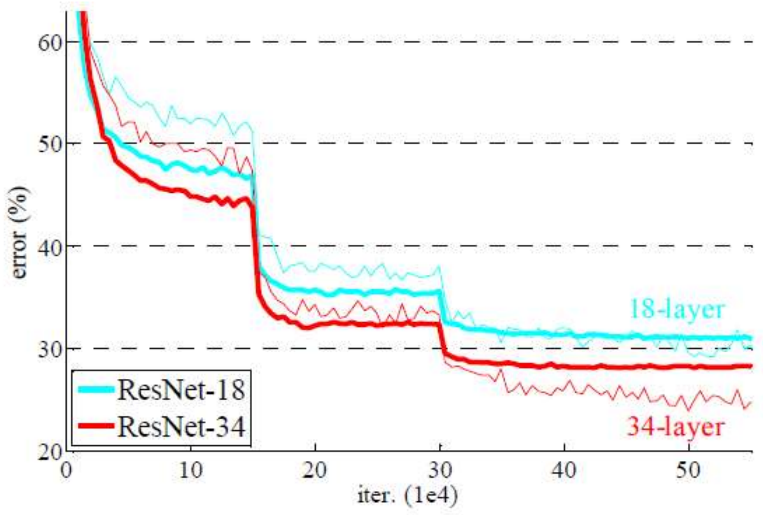


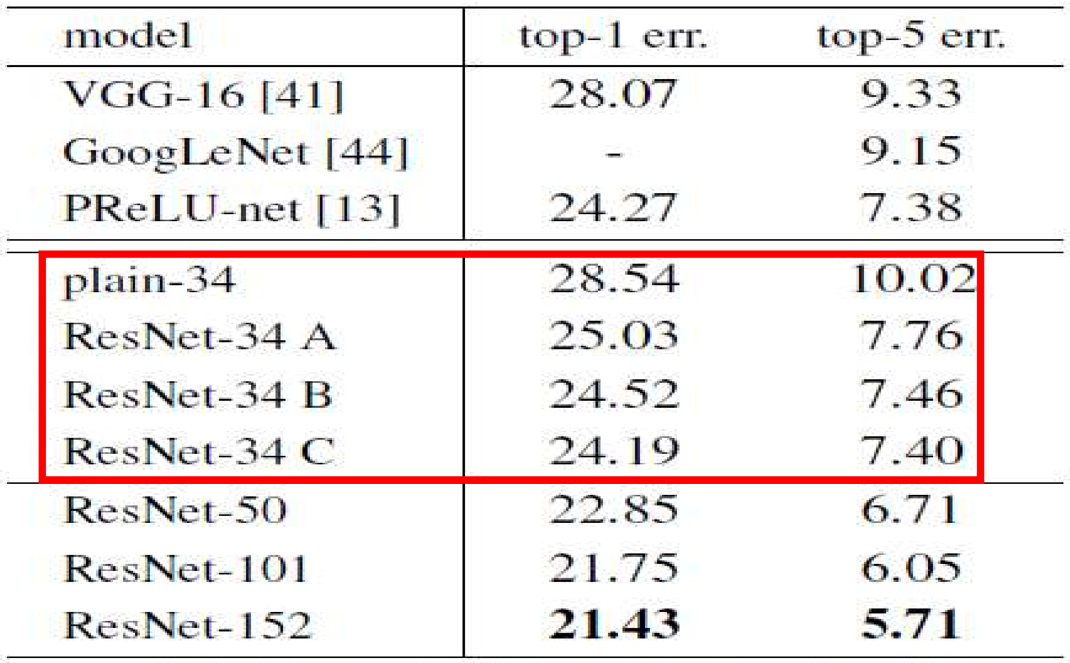
Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

I. the situation is reversed with residual learning – the 34-layer ResNet is better than the 18-layer ResNet (by 2.8%). we manage to obtain accuracy gains from increased depth.

II. compared to its plain counterpart, the 34-layer ResNet reduces the top-1 error by 3.5%.

III. 18-layer plain/residual nets are comparably accurate, but the 18-layer ResNet converges faster.

4. Experiments



model	top-1 err.	top-5 err.
VGG-16 [41]	28.07	9.33
GoogLeNet [44]	-	9.15
PReLU-net [13]	24.27	7.38
plain-34	28.54	10.02
ResNet-34 A	25.03	7.76
ResNet-34 B	24.52	7.46
ResNet-34 C	24.19	7.40
ResNet-50	22.85	6.71
ResNet-101	21.75	6.05
ResNet-152	**21.43**	**5.71**

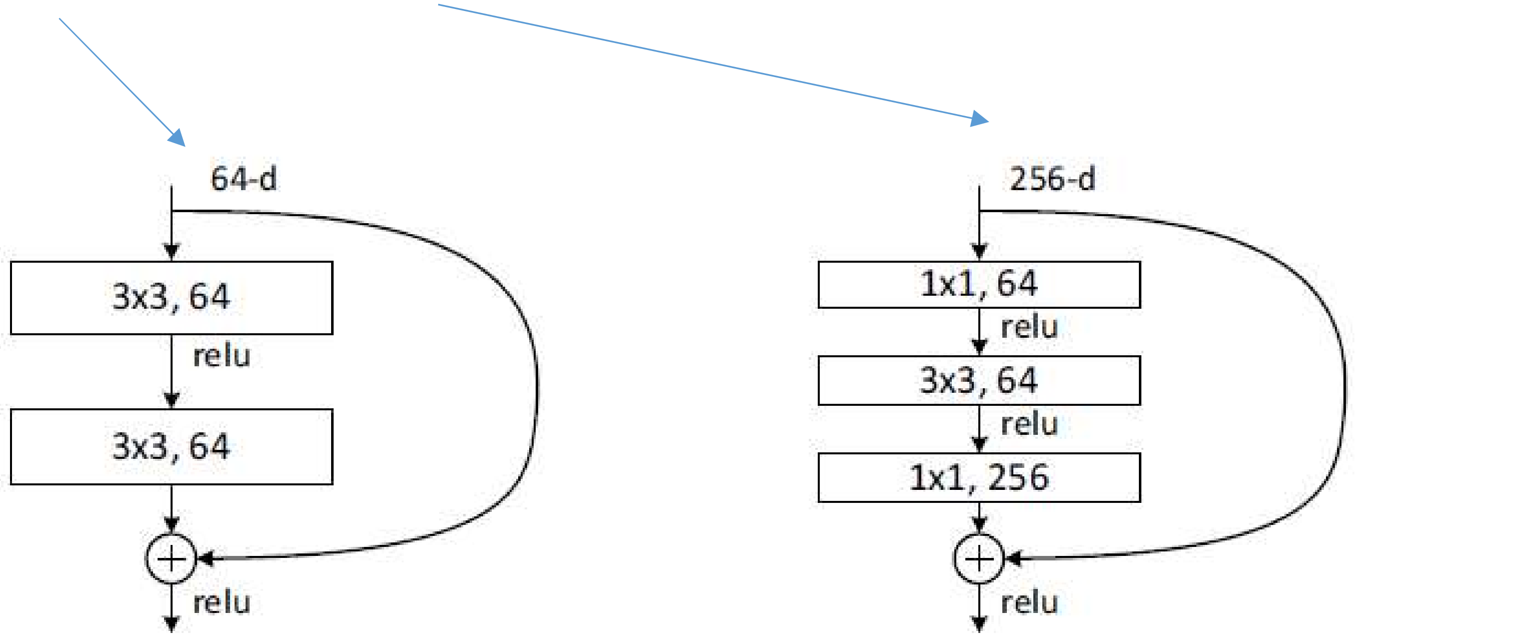
Table 3. Error rates (%, **10-crop** testing) on ImageNet validation. VGG-16 is based on our test. ResNet-50/101/152 are of option B that only uses projections for increasing dimensions.

I. All three options are considerably better than the plain counterpart
II. B is slightly better than A (because the zero-padded dimensions in A indeed have no residual learning)
III. C is marginally better than B (because the extra parameters introduced)
IV. the small differences among A/B/C indicate that projection shortcuts are not essential for addressing the degradation problem. So we do not use option C in the rest of this paper, to reduce memory/ time complexity and model sizes.

4. Experiments

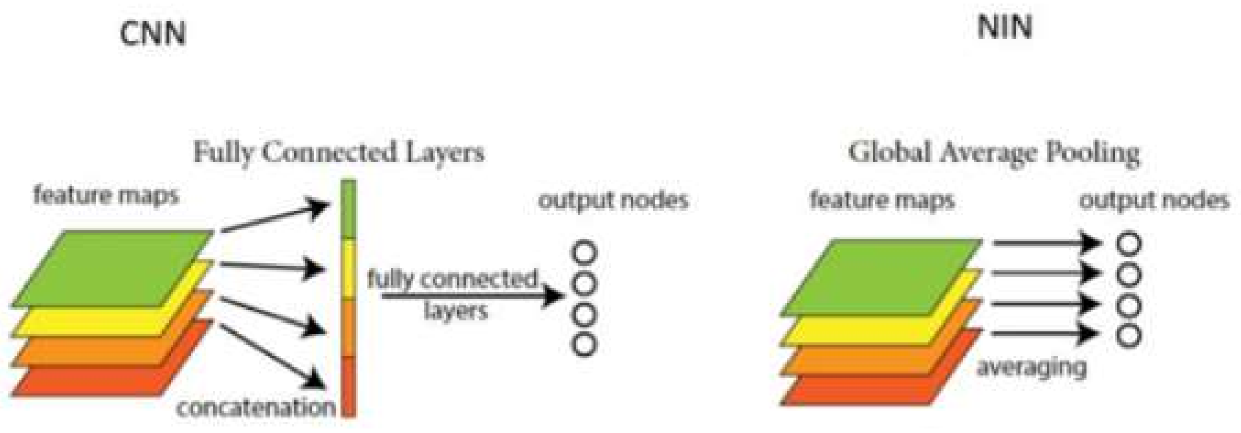
Deeper Bottleneck Architectures.

- Because of concerns on the training time that we can afford, we modify the building block as a bottleneck.



- The three layers are 1x1, 3x3, and 1x1 convolutions, where the 1x1 layers are responsible for reducing and then increasing dimensions.
- 50-layer ResNet: (Table 1)
- 101-layer and 152-layer ResNets: (Table 1)

Global average pooling ??



- 假設最後的一層的 data 是 4 張 6x6 的 feature map, global average pooling 是將每一張 feature map 計算所有 pixel 的均值, 輸出一個值, 這樣4張 feature map 就會輸出4個值

References

[1] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[2] C. M. Bishop. *Neural networks for pattern recognition*. Oxford university press, 1995.

[3] W. L. Briggs, S. F. McCormick, et al. *A Multigrid Tutorial*. Siam, 2000.

[4] K. Chatfield, V. Lempitsky, A. Vedaldi, and A. Zisserman. The devil is in the details: an evaluation of recent feature encoding methods. In *BMVC*, 2011.

[5] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The Pascal Visual Object Classes (VOC) Challenge. *IJCV*, pages 303–338, 2010.

[6] S. Gidaris and N. Komodakis. Object detection via a multi-region & semantic segmentation-aware cnn model. In *ICCV*, 2015.

[7] R. Girshick. Fast R-CNN. In *ICCV*, 2015.

[8] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014.

[9] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010.

[10] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. *arXiv:1302.4389*, 2013.

[11] K. He and J. Sun. Convolutional neural networks at constrained time cost. In *CVPR*, 2015.

[12] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *ECCV*, 2014.

[13] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015.

[14] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv:1207.0580*, 2012.

[15] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[16] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[17] H. Jegou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *TPAMI*, 33, 2011.

[18] H. Jegou, F. Perronnin, M. Douze, J. Sanchez, P. Perez, and C. Schmid. Aggregating local image descriptors into compact codes.

[19] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv:1408.5093*, 2014.

[20] A. Krizhevsky. Learning multiple layers of features from tiny images. *Tech Report*, 2009.

[21] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[22] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to hand-written zip code recognition. *Neural computation*, 1989.

[23] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient backprop. In *Neural Networks: Tricks of the Trade*, pages 9–50. Springer, 1998.

[24] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu. Deeply-supervised nets. *arXiv:1409.5185*, 2014.

[25] M. Lin, Q. Chen, and S. Yan. Network in network. *arXiv:1312.4400*, 2013.

[26] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: Common objects in context. In *ECCV*. 2014.

[27] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015.

[28] G. Montúfar, R. Pascanu, K. Cho, and Y. Bengio. On the number of linear regions of deep neural networks. In *NIPS*, 2014.

[29] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[30] F. Perronnin and C. Dance. Fisher kernels on visual vocabularies for image categorization. In *CVPR*, 2007.

[31] T. Raiko, H. Valpola, and Y. LeCun. Deep learning made easier by linear transformations in perceptrons. In *AISTATS*, 2012.

[32] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *NIPS*, 2015.

[33] S. Ren, K. He, R. Girshick, X. Zhang, and J. Sun. Object detection networks on convolutional feature maps. *arXiv:1504.06066*, 2015.

[34] B. D. Ripley. *Pattern recognition and neural networks*. Cambridge university press, 1996.

[35] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. Fitnets: Hints for thin deep nets. In *ICLR*, 2015.

[36] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *arXiv:1409.0575*, 2014.

[37] A. M. Saxe, J. L. McClelland, and S. Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv:1312.6120*, 2013.

[38] N. N. Schraudolph. Accelerated gradient descent by factor-centering decomposition. Technical report, 1998.

References

[39] N. N. Schraudolph. Centering neural network gradient factors. In *Neural Networks: Tricks of the Trade*, pages 207–226. Springer, 1998.

[40] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. In *ICLR*, 2014.

[41] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[42] R. K. Srivastava, K. Greff, and J. Schmidhuber. Highway networks. *arXiv:1505.00387*, 2015.

[43] R. K. Srivastava, K. Greff, and J. Schmidhuber. Training very deep networks. *1507.06228*, 2015.

[44] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[45] R. Szeliski. Fast surface interpolation using hierarchical basis functions. *TPAMI*, 1990.

[46] R. Szeliski. Locally adapted hierarchical basis preconditioning. In *SIGGRAPH*, 2006.

[47] T. Vatanen, T. Raiko, H. Valpola, and Y. LeCun. Pushing stochastic gradient towards second-order methods–backpropagation learning with transformations in nonlinearities. In *Neural Information Processing*, 2013.

[48] A. Vedaldi and B. Fulkerson. VLFeat: An open and portable library of computer vision algorithms, 2008.

[49] W. Venables and B. Ripley. Modern applied statistics with s-plus. 1999.

[50] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional neural networks. In *ECCV*, 2014.