Convolutional Neural Networks

Janghun Jo
Computer Graphics Lab.
jhjo432@postech.ac.kr

POSTECH

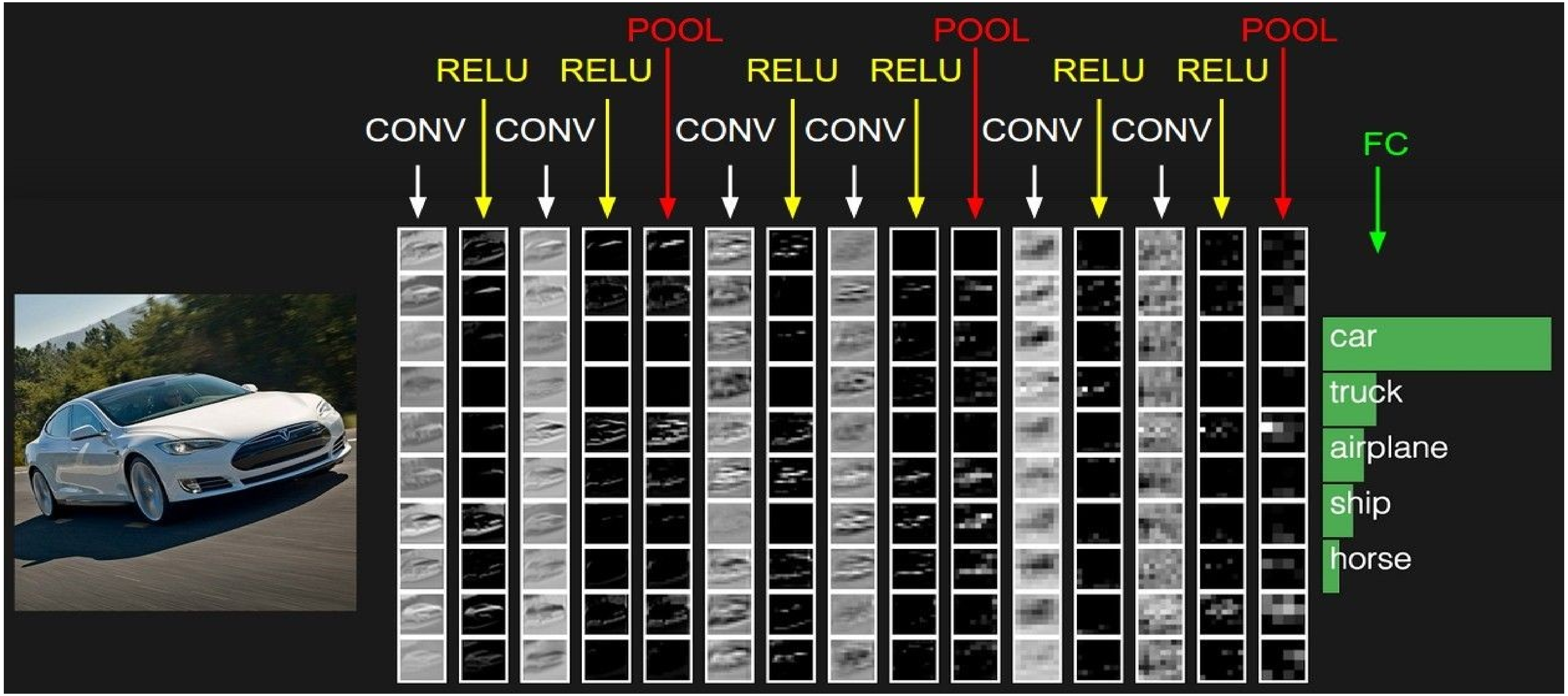
Contents

- TODO
  - quiz: CIFAR 10
  - quiz: VGG 19
  - quiz: Resnet

- CNN
  - Training a CNN classifier on CIFAR-10

- VGG 19
  - Implement
  - Training
- ResNet
  - Implement
  - Training

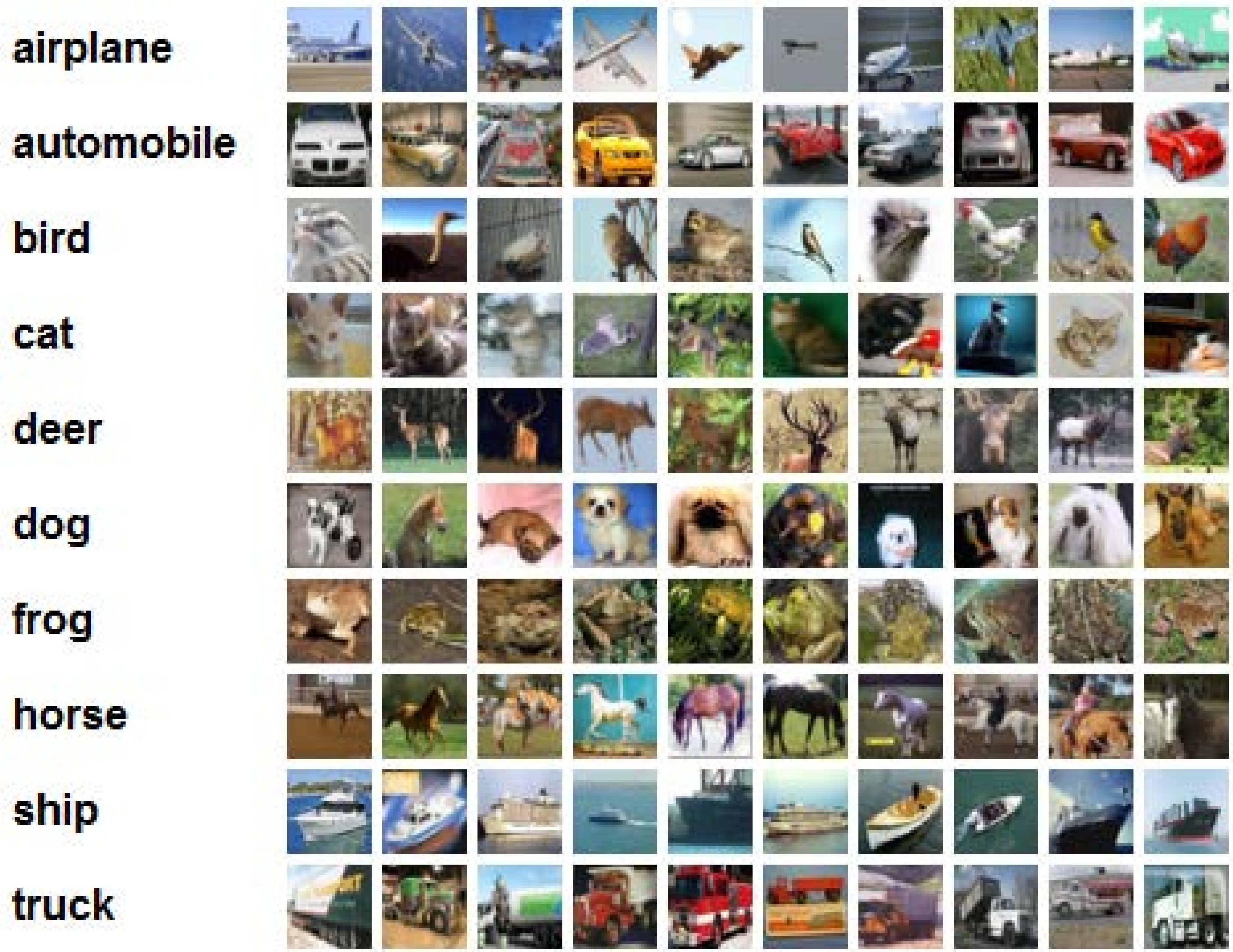
Convolutional Neural Networks



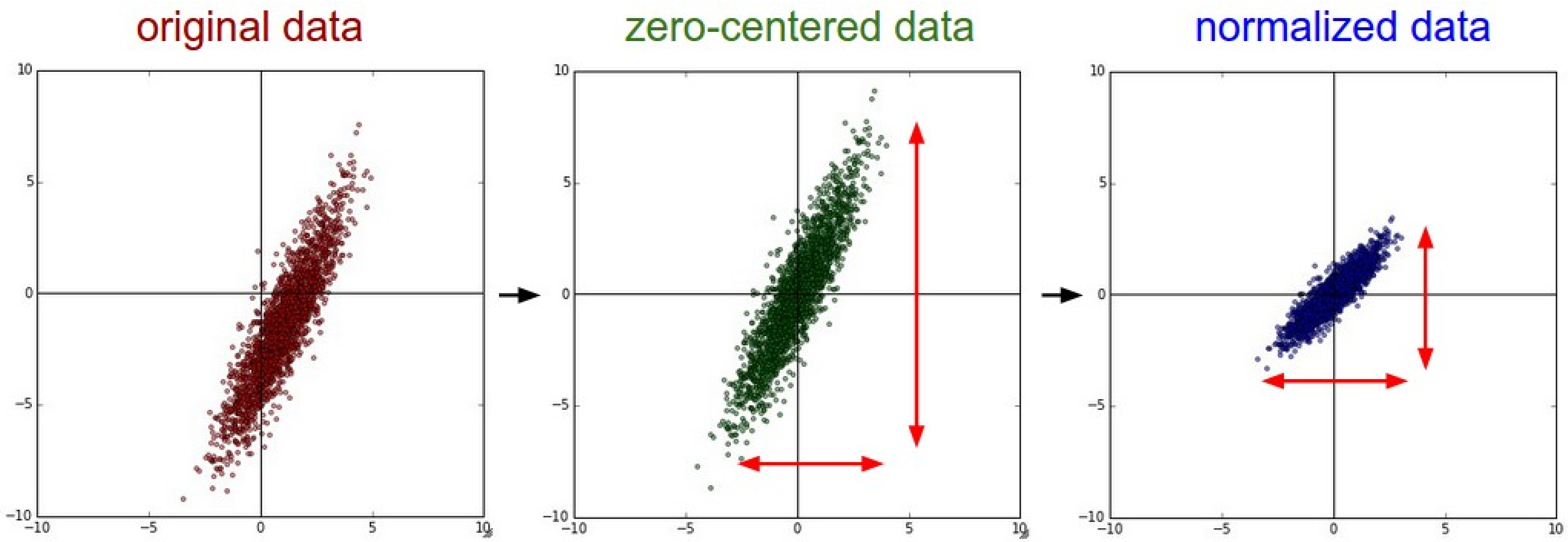
CIFAR-10 dataset

- Image classification dataset
  - 10 classes
  - 32 x 32 image size, RGB images

  - Training sample : 50,000
  - Test sample : 10,000



Note: Image Normalization



Quiz1. Training a CNN classifier on CIFAR-10

- Steps)
  - Load and normalize the CIFAR 10 training and test datasets using torchvision

  - 1. Define a convolutional Neural Network
    - Convolution – input channel: 3, output channel: 6, kernel_size: 5
    - Maxpoling – size: 2, stride: 2
    - Convolution – input channel: 6, output channel: 16, kernel_size: 5
    - Maxpoling – size: 2, stride: 2
    - Fully connected layer – in_features: 400, out_features: 120
    - Fully connected layer – in_features: 120, out_features: 84
    - Fully connected layer – in_features: 84, out_features: 10
    - Note: Apply ReLU activation function for hidden layers.

  - 2. Define a Loss function and optimizer
    - Cross-Entropy loss
    - SGD with learning rate 0.001 and momentum 0.9

  - 3. Train the network on the training data

  - Test the network on the test data

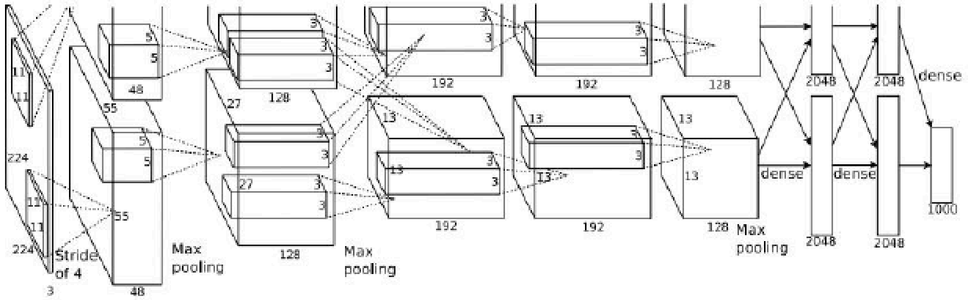
AlexNet [Krizhevskyet al. 2012]

- Successful CNN image classification model
  - Based on LeNet5 CNN design (Yann Lecun et al, 1989)
  - Computationally expensive, but feasible due to GPUs
    - Parallel computation
  - Winner of ILSVRC-2012 competition by a large margin,
    - ImageNet Large-Scale Visual Recognition Challenge
    - a top-5 error of 15.3%, more than 10.8 percentage points lower than that of the runner up.
  - Transfer to significant gains in a variety of domains

AlexNet [Krizhevskyet al. 2012]

Case Study: AlexNet

[Krizhevsky et al. 2012]

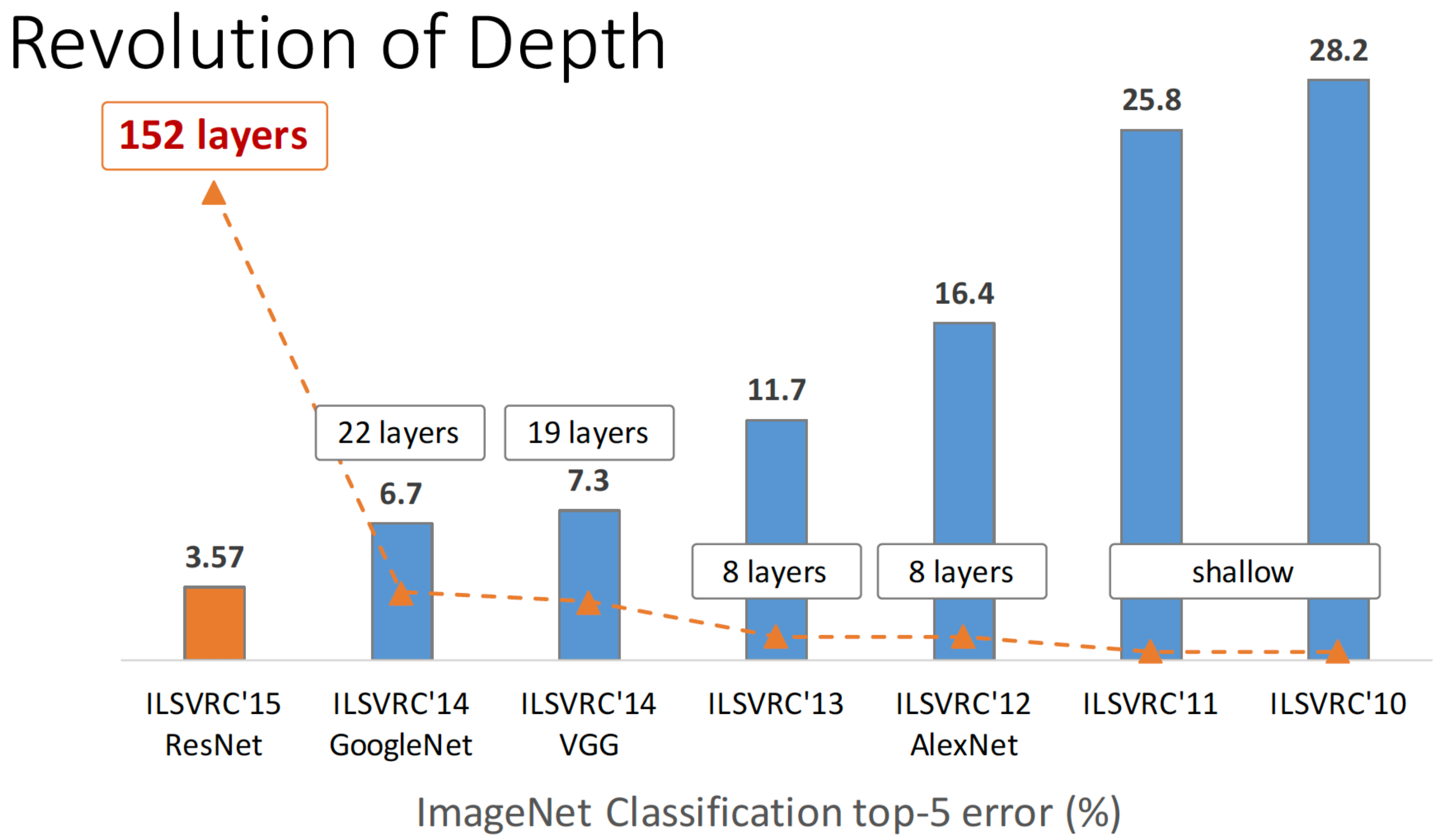


Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Details/Retrospectives:
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

- Architecture
  - 5 Conv + 3 fc layers, ReLU
  - Dropout for fc layers
    - For regularization to deal with overfitting

Common trend : Revolution of depth



(Slides from Kaiming He's recent presentation)

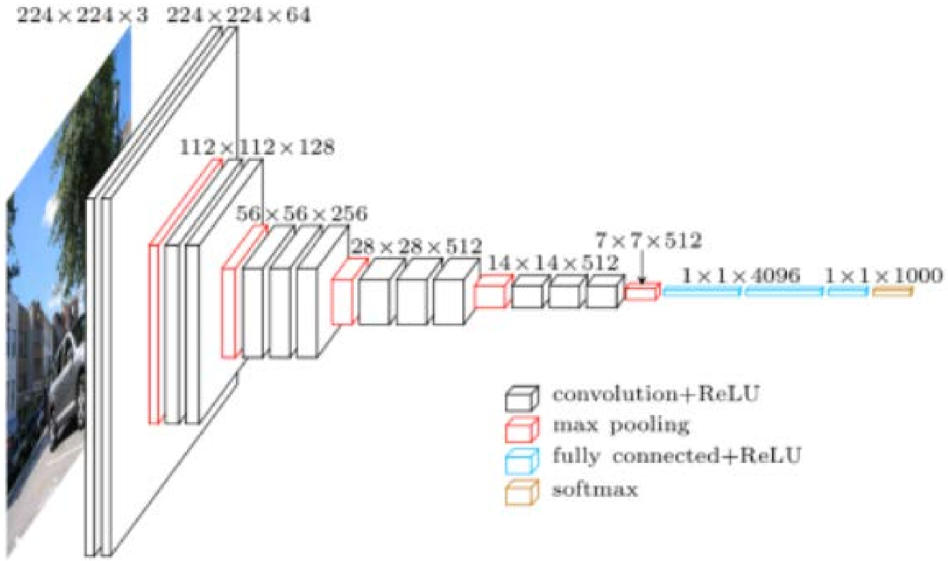
Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. Deep Residual Learning for Image Recognition. CVPR 2016.

VGGNet [Simonyan & Zisserman, 2015]

- Proposed by Oxford VGG team in 2014 ILSVRC (2nd rank)

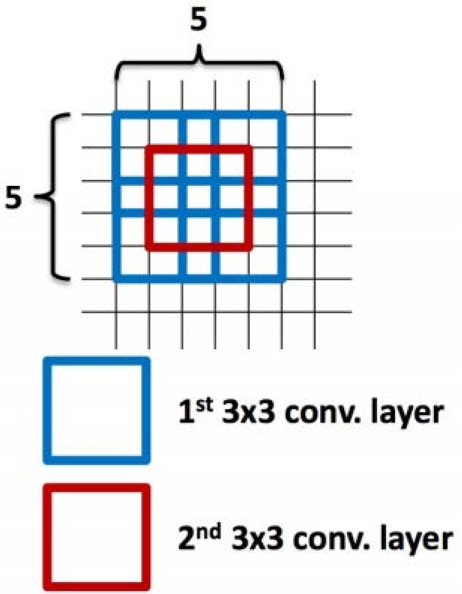
- Comparison
  - More deeper network than AlexNet
  - More simple structure than GooleNet (1st rank) (covered later!)
  - More computational budget than GooleNet (parameter & computation)

- Filter
  - Only 3 x 3 filter & 1 x 1 filter
    - Less parameter for same receptive field -> Regularization

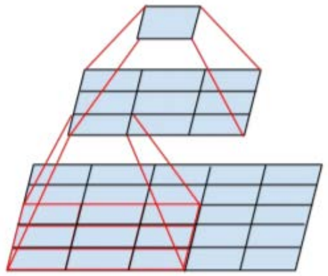


VGGNet [Simonyan & Zisserman, 2015]

- Receptive field
  - the region in the input space that a particular CNN's feature is looking at

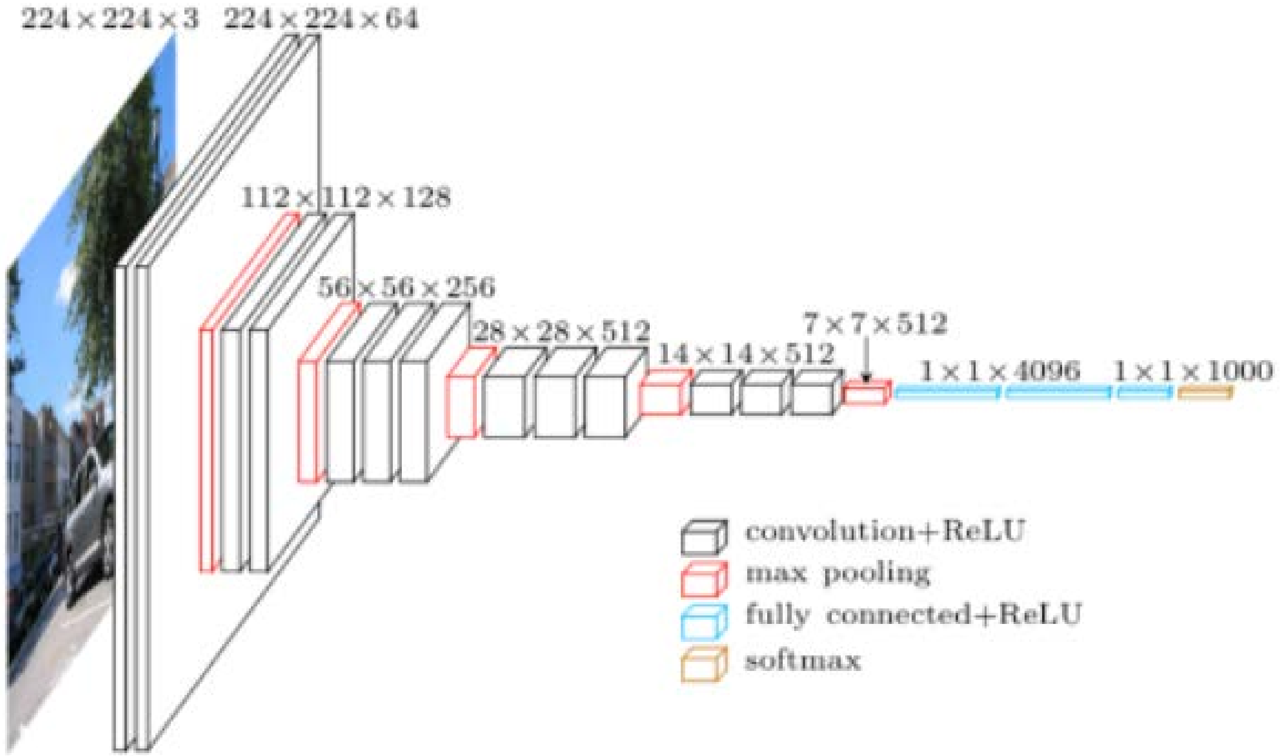


- Example)
  - Stack of 3 filters(3x3) = 1 filter (7 x 7)
    - # of parameters
      - 3-(3x3) filter : 3 * ($3^2 * C^2$) = 27 * C^2
      - 1-(7x7) filter : ($7^2 * C^2$) = 49 * C^2



VGGNet [Simonyan & Zisserman, 2015]

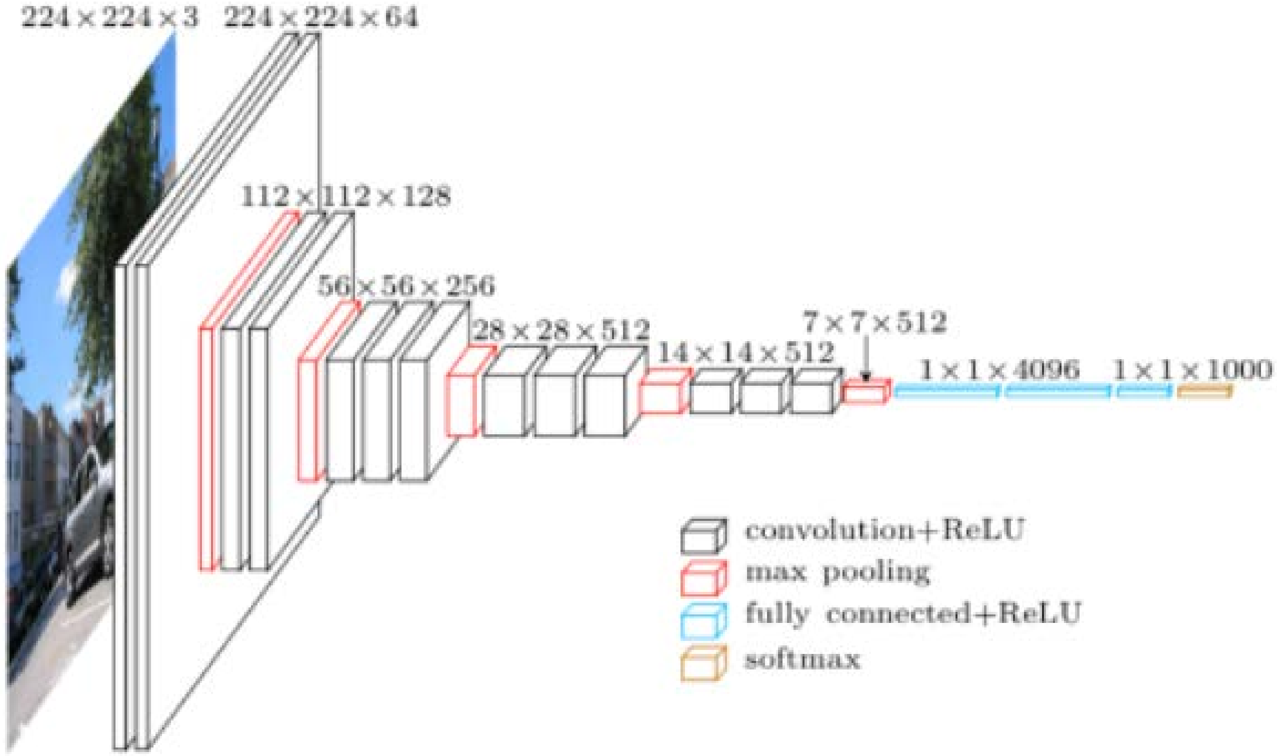
- Configuration
  - Image size : 224 * 224 * 3
  - Stride : 1
  - Padding : 1
  - Max – pooling
    - 2 x 2 window, 2 stride
  - Filter size : 3 x 3, 1 x 1



VGGNet [Simonyan & Zisserman, 2015]

- Convolutional filters
  - Starting from 64, double after each max-pooling layer until 512

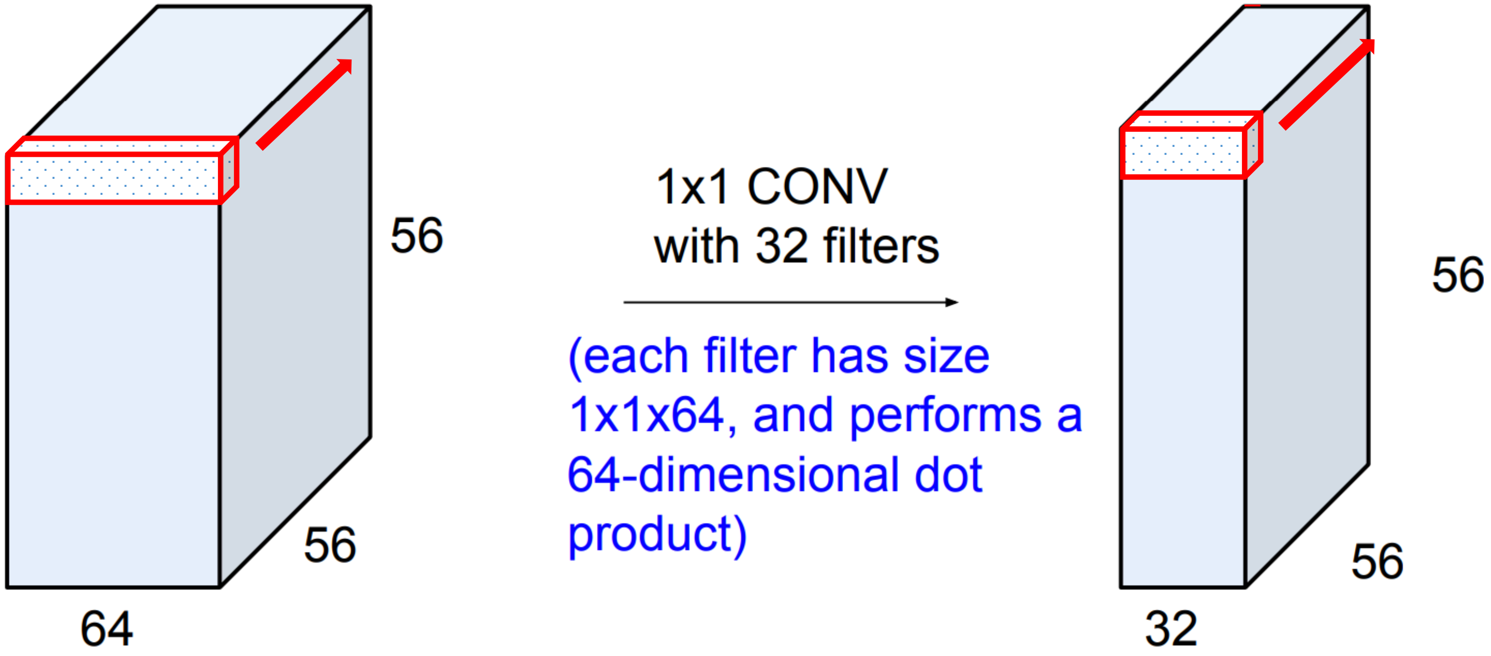
- In 3x3 filter, use 1 padding & 1 stric
  - Preserve spatial resolution (H * W size)



1 x 1 Convolution

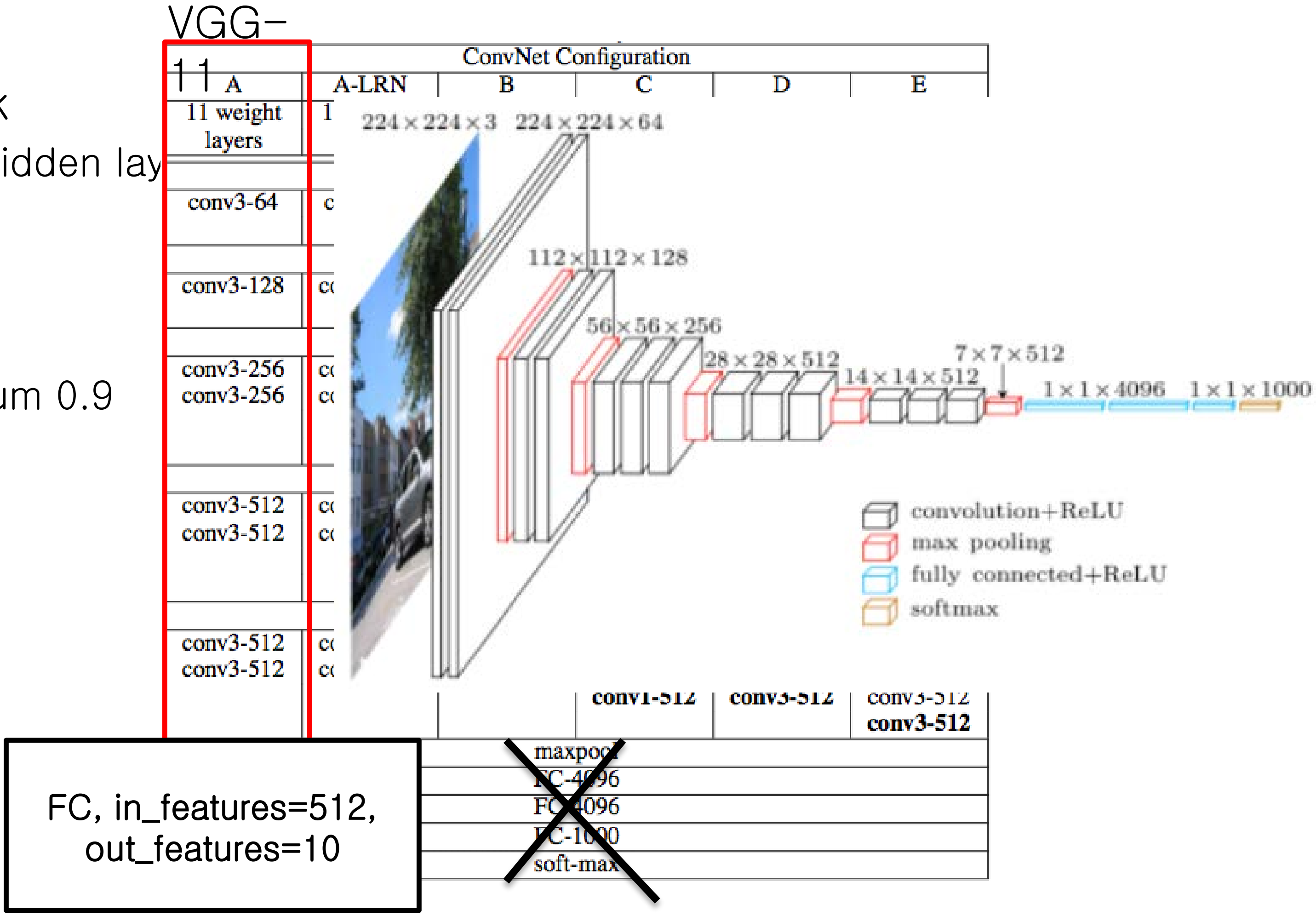
- Increase non-linearity without affecting receptive field
- When input channels == output channels
  - Projection onto space of same dim
- Another perspective : Fully connected with weight sharing
  - Ex) Fully Convolutional Networks for Semantic Segmentation (Jonathan Long, et al) (cover later in Semantic Segmentation)
- Simple interpretation : Change the channel
  - Used in Inception network to reduce computational budget

1 x 1 Convolution

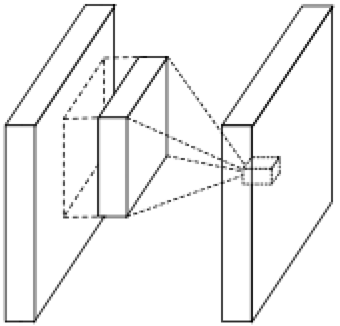


Quiz2. **Training VGG-11 on CIFAR-10**

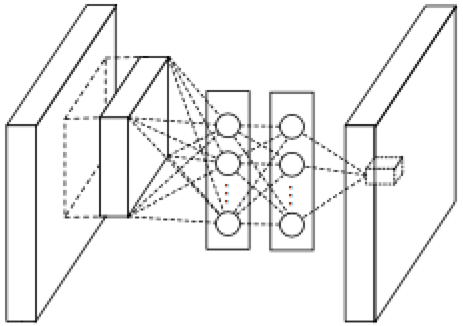
- Steps)
  - Load and normalize the CIFAR 10 training and test datasets using torchvision
  - 1. Define a convolutional Neural Network
    - Note: Apply ReLU activation function for hidden lay
  - 2. Define a Loss function and optimizer
    - Cross-Entropy loss
    - SGD with learning rate 0.01 and momentum 0.9
  - 3. Train the network on the training data
  - Test the network on the test data



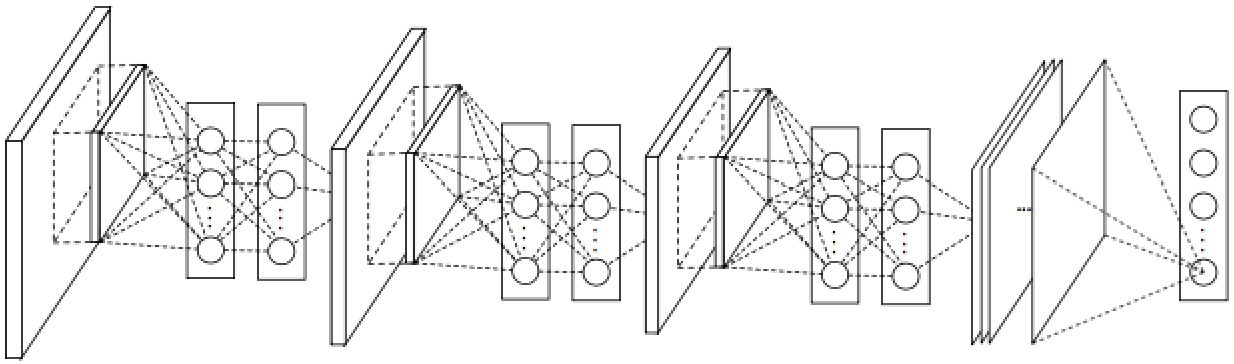
Network In Network



(a) Linear convolution layer



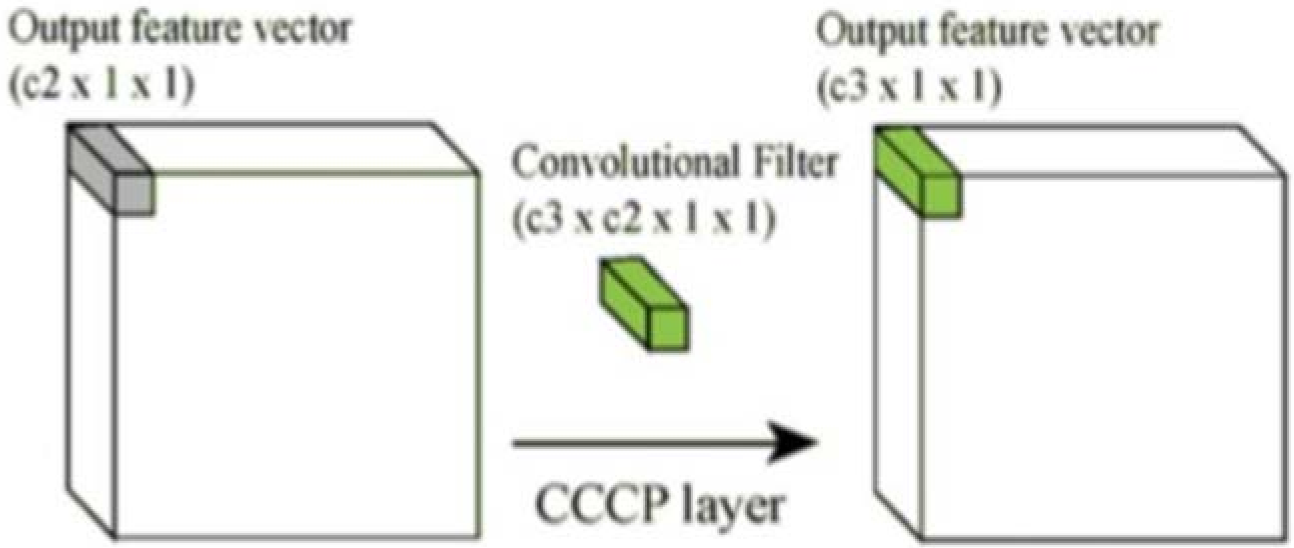
(b) Mlpconv layer



Network In Network

Main Points

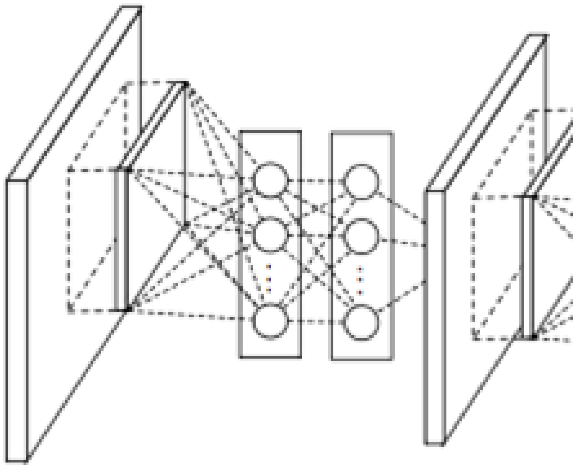
- The MLP as convolution
- 1x1 Conv to reduce number of channel
- Global avgpool in last layer (Drop fully connected layers)



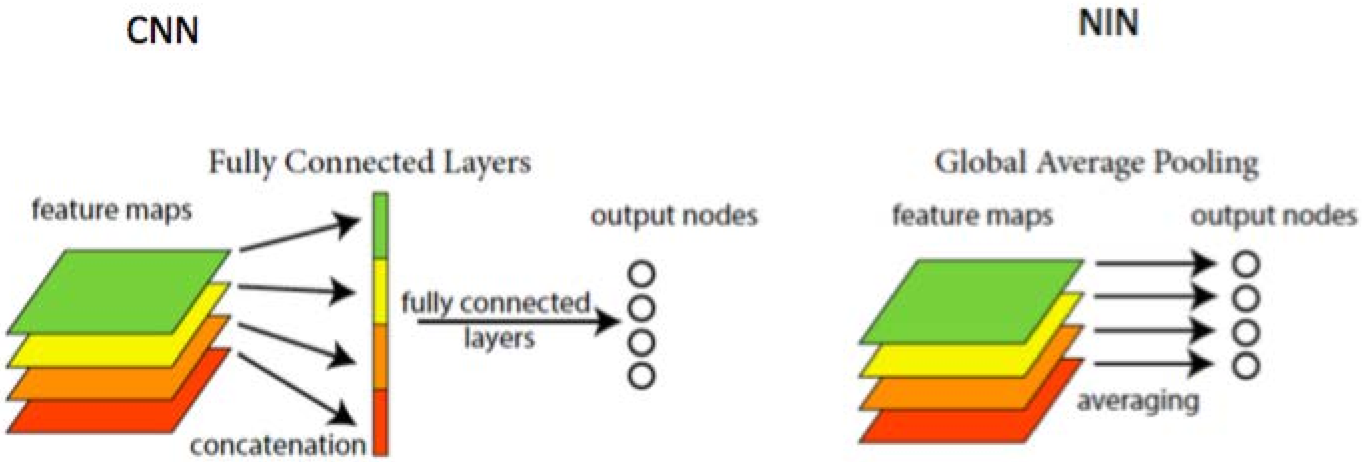
Network In Network

- Implementation

```
nn.Conv2d(3, 192, kernel_size=5, stride=1, padding=2),
nn.ReLU(inplace=True),
nn.Conv2d(192, 160, kernel_size=1, stride=1, padding=0),
nn.ReLU(inplace=True),
nn.Conv2d(160,  96, kernel_size=1, stride=1, padding=0),
nn.ReLU(inplace=True),
nn.MaxPool2d(kernel_size=3, stride=2, padding=1),
nn.Dropout(0.5),
```



Network In Network



Network In Network

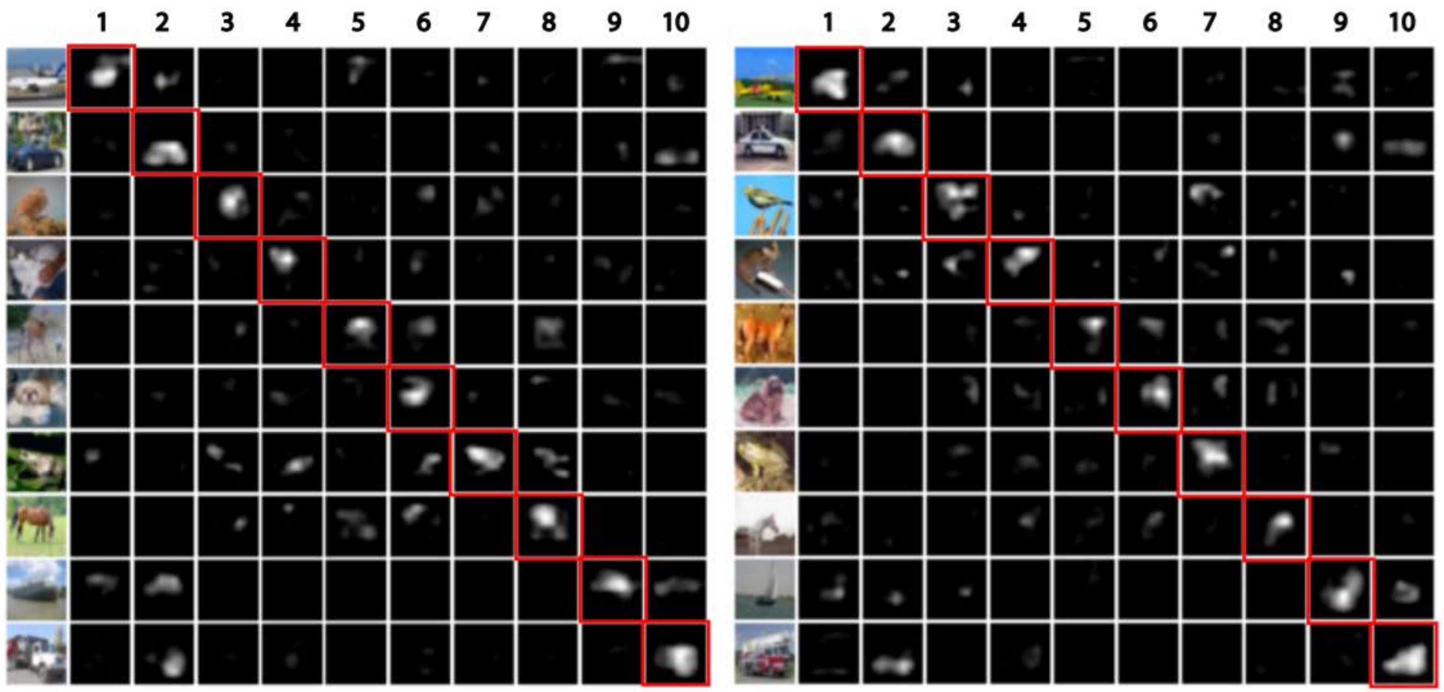
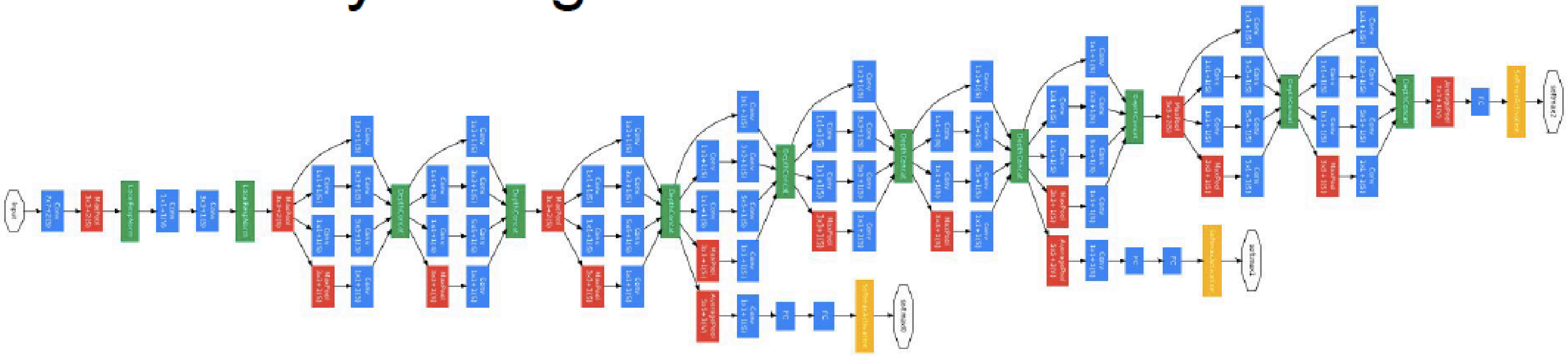


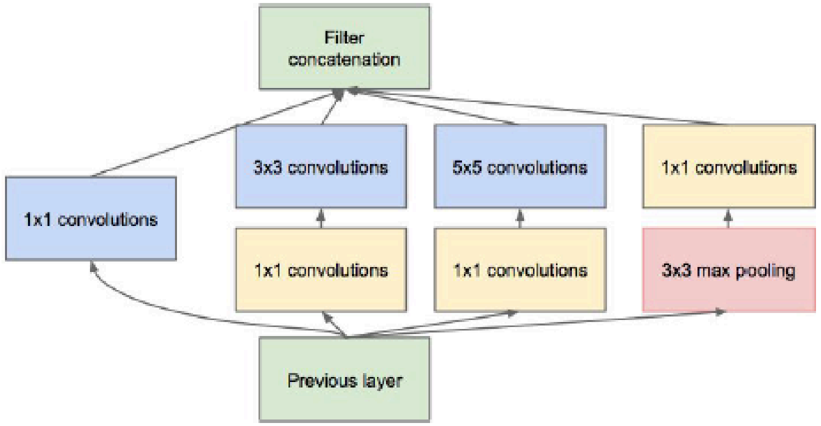
Figure 4: Visualization of the feature maps from the last mlpconv layer. Only top 10% activations in the feature maps are shown. The categories corresponding to the feature maps are: 1. airplane, 2. automobile, 3. bird, 4. cat, 5. deer, 6. dog, 7. frog, 8. horse, 9. ship, 10. truck. Feature maps corresponding to the ground truth of the input images are highlighted. The left panel and right panel are just different examplars.

GoogLeNet [Szegedy et al., 2014]

Case Study: GoogLeNet

[Szegedy et al., 2014]

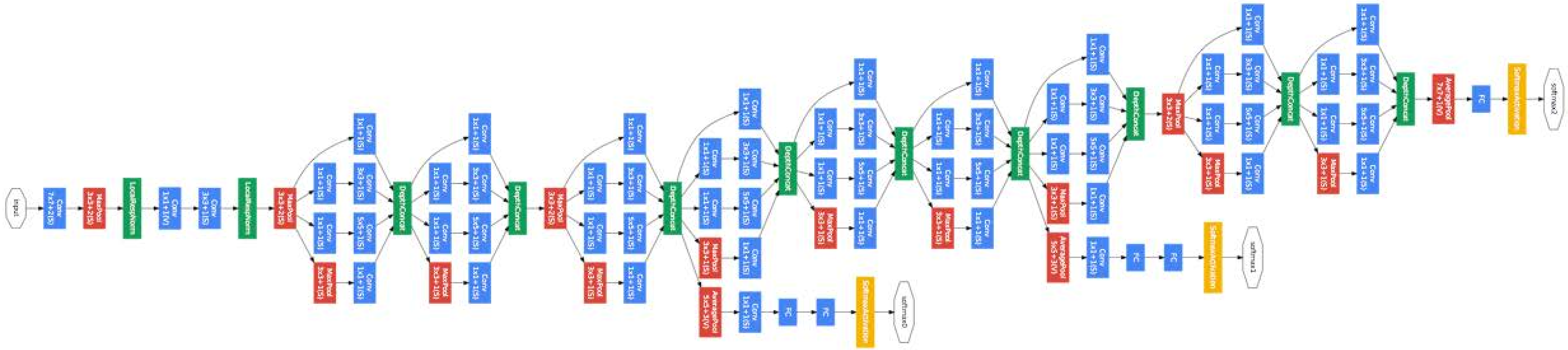




Inception module

ILSVRC 2014 winner (6.7% top 5 error)

GoogLeNet [Szegedyet al., 2014]



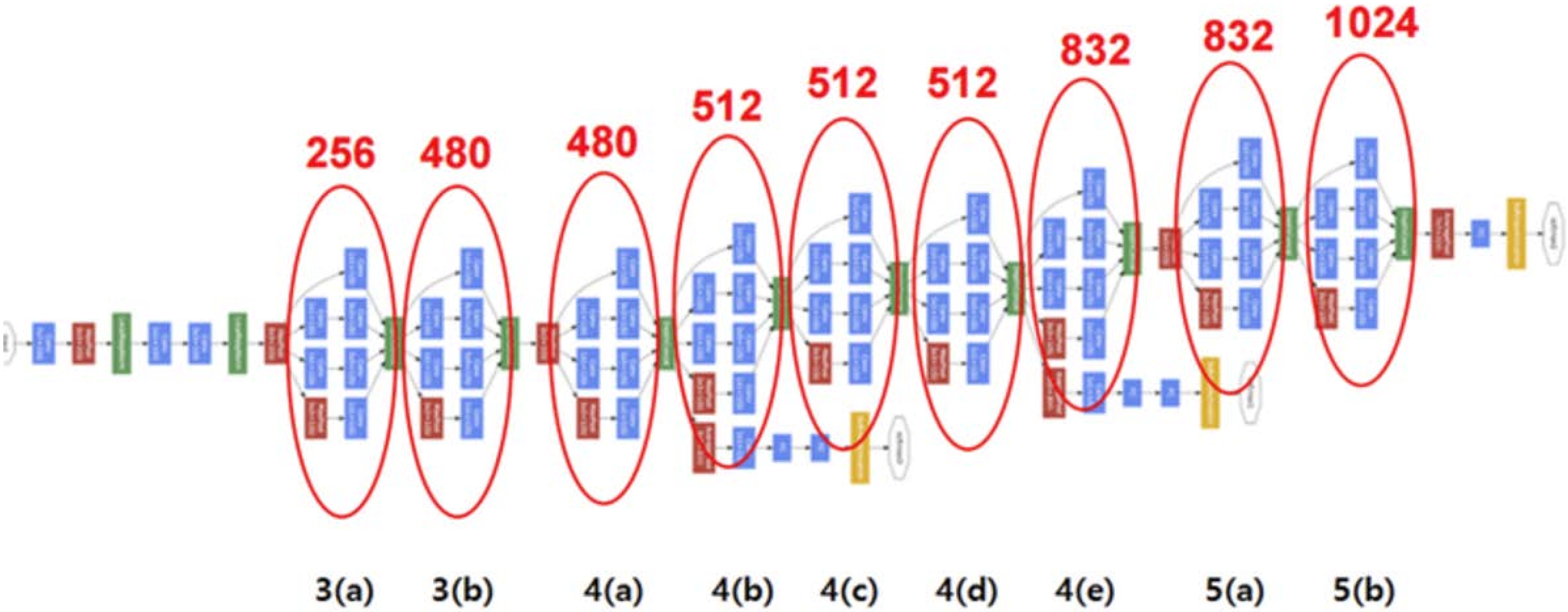
Convolution Pooling Softmax Others

- Network in network: inception modules
- Auxiliary classifiers to facilitate training
- 22 layer network: 27 layers if pooling layers are counted
- The winner of ILSVRC 2014 classification task

Uses 12x fewer parameters than AlexNet
Used 9 Inception modules in the whole architecture
No use of fully connected layers! They use average pool instead
(Save huge number of parameters)
Many size of kernels (1x1, 3x3, 5x5, 7x7)

[Szegedy15] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich: **Going deeper with convolutions**. CVPR 2015

GoogLeNet [Szegedyet al., 2014]



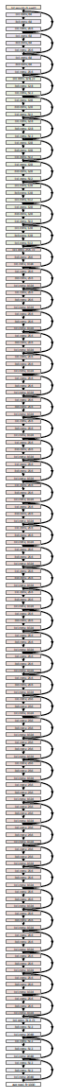
ResNet [He et al. 2016]

Revolution of Depth

AlexNet, 8 layers
(ILSVRC 2012)

VGG, 19 layers
(ILSVRC 2014)

ResNet, 152 layers
(ILSVRC 2015)



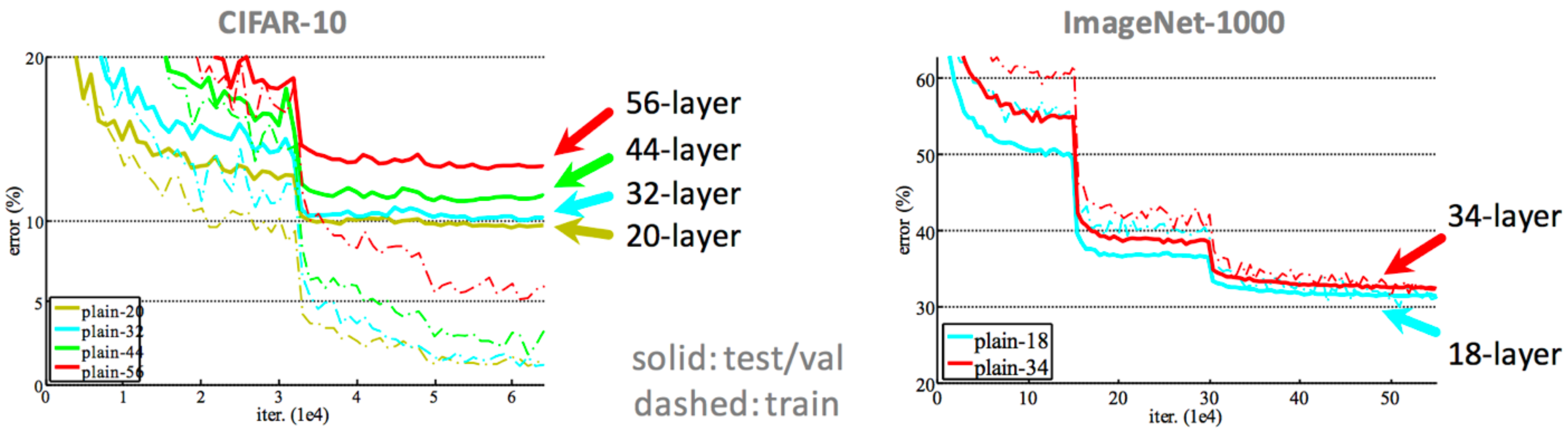
- 2–3 weeks of training on 8-GPU machine
- Running faster than a VGGNet
  - even though it has 8x more layers

(Slides from Kaiming He's recent presentation)

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. Deep Residual Learning for Image Recognition. CVPR 2016.

ResNet [He et al. 2016]

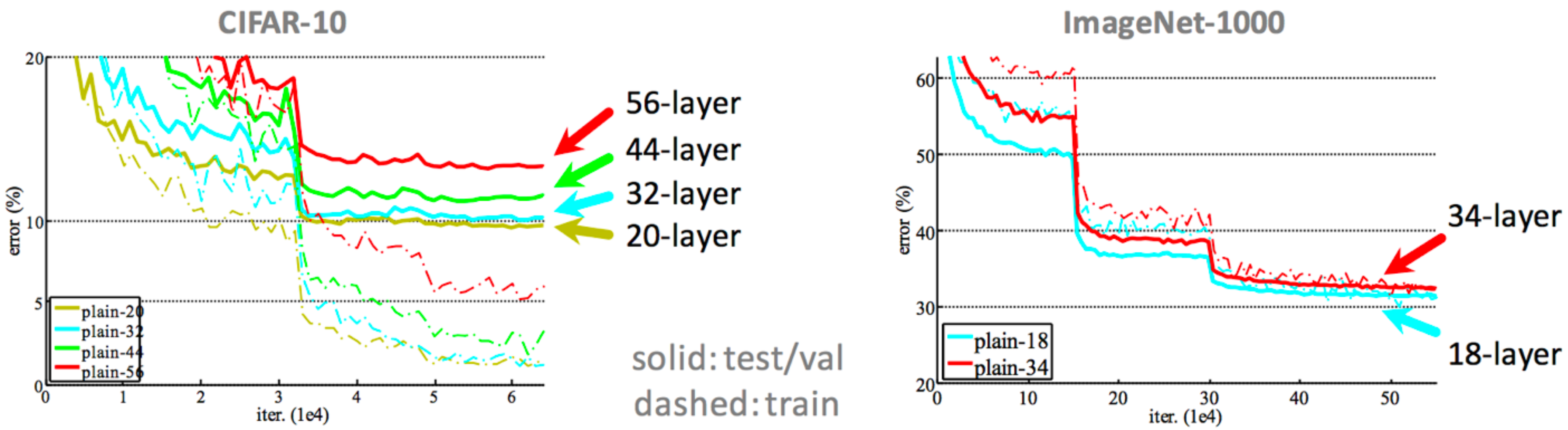
- Simply stacking layers?



  - "Overly deep" plain nets have higher training error
  - A general phenomenon, observed in many datasets

ResNet [He et al. 2016]

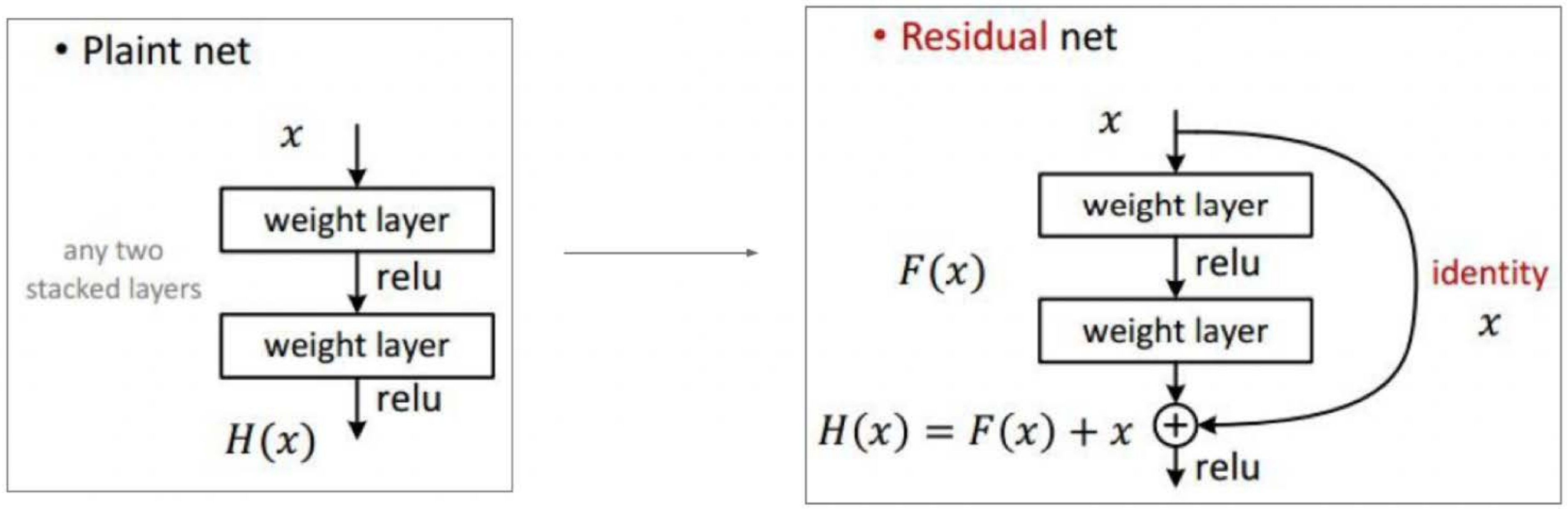
- Simply stacking layers?



  - “Overly deep” plain nets have higher training error
  - A general phenomenon, observed in many datasets

ResNet [He et al. 2016]

- Residual learning
  - Main idea: make it easy to learn the identity mapping!



  - Very smooth backward propagations by preconditioning

ResNet [He et al. 2016]

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

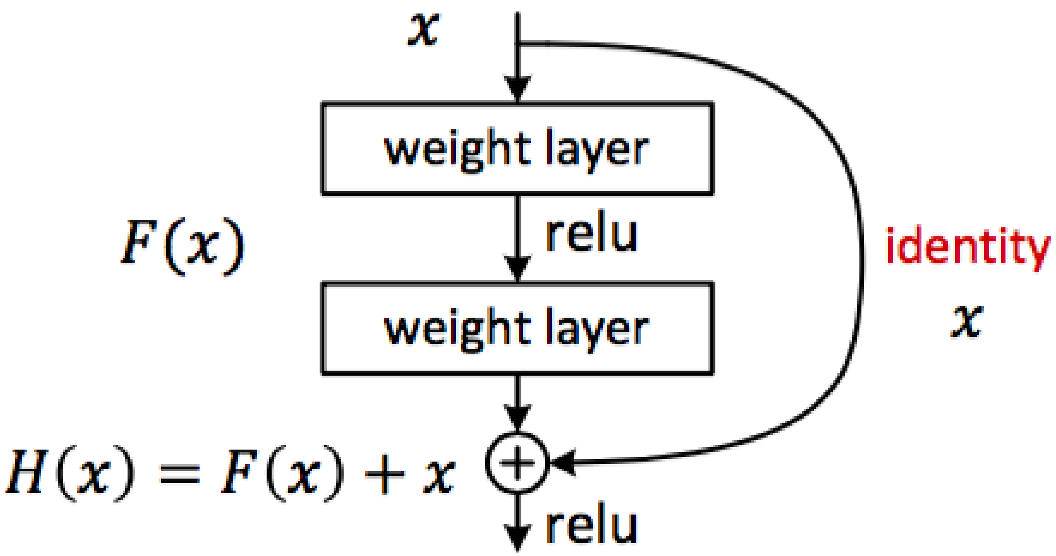
ResNet [He et al. 2016]

method	top-5 err. (**test**)
VGG [41] (ILSVRC'14)	7.32
GoogLeNet [44] (ILSVRC'14)	6.66
VGG [41] (v5)	6.8
PReLU-net [13]	4.94
BN-inception [16]	4.82
ResNet (ILSVRC'15)	**3.57**

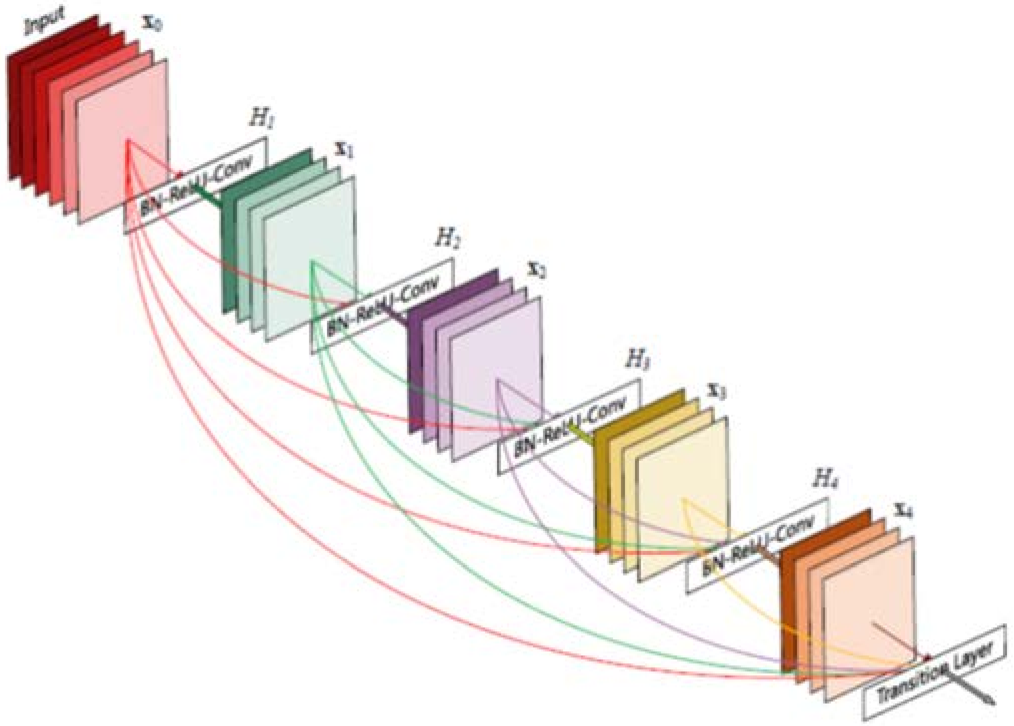
Quiz 3. Implement a CNN with Residual Block on CIFAR-10

- Steps)
  - Load and normalize the CIFAR 10 training and test datasets using torchvision
  - 1. Define a convolutional Neural Network
    - Convolution – input channel: 3, output channel: 64, kernel_size: 3
    - Residual block – hidden channel: 256, kernel_size: 3
    - Maxpoling – size: 2, stride: 2
    - Residual block – hidden channel: 256, kernel_size: 3
    - Maxpoling – size: 2, stride: 2
    - Fully connected layer – in_features: 4096, out_features: 120
    - Fully connected layer – in_features: 120, out_features: 84
    - Fully connected layer – in_features: 84, out_features: 10
    - Note
      - Apply ReLU activation function for hidden layers.
      - Residual block have same number of input channels and output channels.
  - 2. Define a Loss function and optimizer
    - Cross-Entropy loss
    - Adam optimizer with learning rate 0.001
  - 3. Train the network on the training data
  - Test the network on the test data

- Residual net



DenseNet (2017)



DenseNet (2017)

Layers	Output Size	DenseNet-121 $(k=32)$	DenseNet-169 $(k=32)$	DenseNet-201 $(k=32)$	DenseNet-161 $(k=48)$
Convolution	112×112	7×7 conv, stride 2			
Pooling	56×56	3×3 max pool, stride 2			
Dense Block (1)	56×56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56×56	1×1 conv			
	28×28	2×2 average pool, stride 2			
Dense Block (2)	28×28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28×28	1×1 conv			
	14×14	2×2 average pool, stride 2			
Dense Block (3)	14×14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 36$
Transition Layer (3)	14×14	1×1 conv			
	7×7	2×2 average pool, stride 2			
Dense Block (4)	7×7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$
Classification Layer	1×1	7×7 global average pool			
		1000D fully-connected, softmax			

Table 1. DenseNet architectures for ImageNet. The growth rate for the first 3 networks is $k = 32$, and $k = 48$ for DenseNet-161. Note that each "conv" layer shown in the table corresponds the sequence BN-ReLU-Conv.

DenseNet (2017)

Method	Depth	Params	C10	C10+	C100	C100+	SVHN
Network in Network [22]	-	-	10.41	8.81	35.68	-	2.35
All-CNN [31]	-	-	9.08	7.25	-	33.71	-
Deeply Supervised Net [20]	-	-	9.69	7.97	-	34.57	1.92
Highway Network [33]	-	-	-	7.72	-	32.39	-
FractalNet [17]	21	38.6M	10.18	5.22	35.34	23.30	2.01
with Dropout/Drop-path	21	38.6M	7.33	4.60	28.20	23.73	1.87
ResNet [11]	110	1.7M	-	6.61	-	-	-
ResNet (reported by [13])	110	1.7M	13.63	6.41	44.74	27.22	2.01
ResNet with Stochastic Depth [13]	110	1.7M	11.66	5.23	37.80	24.58	1.75
	1202	10.2M	-	4.91	-	-	-
Wide ResNet [41]	16	11.0M	-	4.81	-	22.07	-
	28	36.5M	-	4.17	-	20.50	-
with Dropout	16	2.7M	-	-	-	-	1.64
ResNet (pre-activation) [12]	164	1.7M	11.26*	5.46	35.58*	24.33	-
	1001	10.2M	10.56*	4.62	33.47*	22.71	-
DenseNet ($k = 12$)	40	1.0M	**7.00**	5.24	**27.55**	24.42	1.79
DenseNet ($k = 12$)	100	7.0M	**5.77**	**4.10**	**23.79**	**20.20**	1.67
DenseNet ($k = 24$)	100	27.2M	**5.83**	**3.74**	**23.42**	**19.25**	**1.59**
DenseNet-BC ($k = 12$)	100	0.8M	**5.92**	4.51	**24.15**	22.27	1.76
DenseNet-BC ($k = 24$)	250	15.3M	**5.19**	**3.62**	**19.64**	**17.60**	1.74
DenseNet-BC ($k = 40$)	190	25.6M	-	**3.46**	-	**17.18**	-