

# Cornell Bowers C-IS

College of Computing and Information Science

## Modern Convolutional Neural Networks 2

CS4782: Intro to Deep Learning

# Thanks to:

Varsha Kishore

Justin Lovelace

Anissa Dallmann

Stephanie Ginting

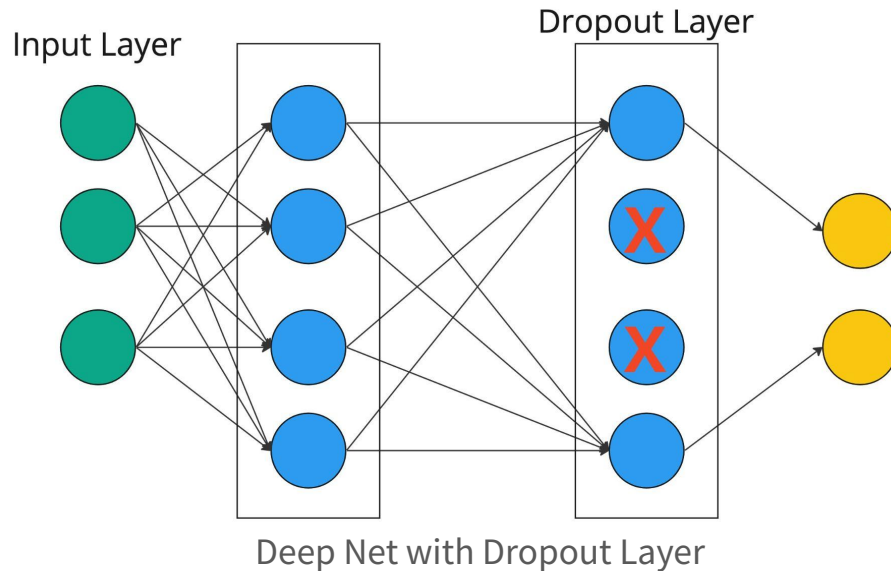
# Logistics

- **HW1+P1** is due Thursday (February 13) 11:59 PM
- Late submissions accepted until Saturday (February 15) 11:59 PM
- **HW2** to be released this Thursday (February 13) - due Thursday (February 27)
- **P2** release timelines to be confirmed soon - due Thursday (February 27)
- Office hours are listed on the course website
- Homework clarifications are listed as pinned posts under HW1 on Ed
- Post questions on Ed

# Clarification: Dropout

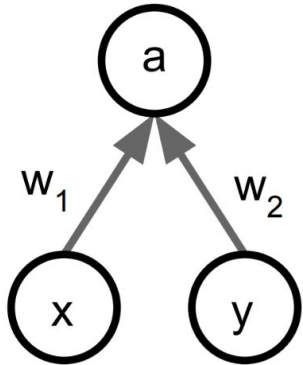
In each forward pass, randomly set some neurons to zero.

The probability of ~~keeping~~ a neuron is a hyperparameter;  $p=0.5$  is common.  
zeroing



# Clarification: Dropout During Test Time

Need to re-scale activations so they are the same (in expectation) during training and testing



Consider a single neuron.

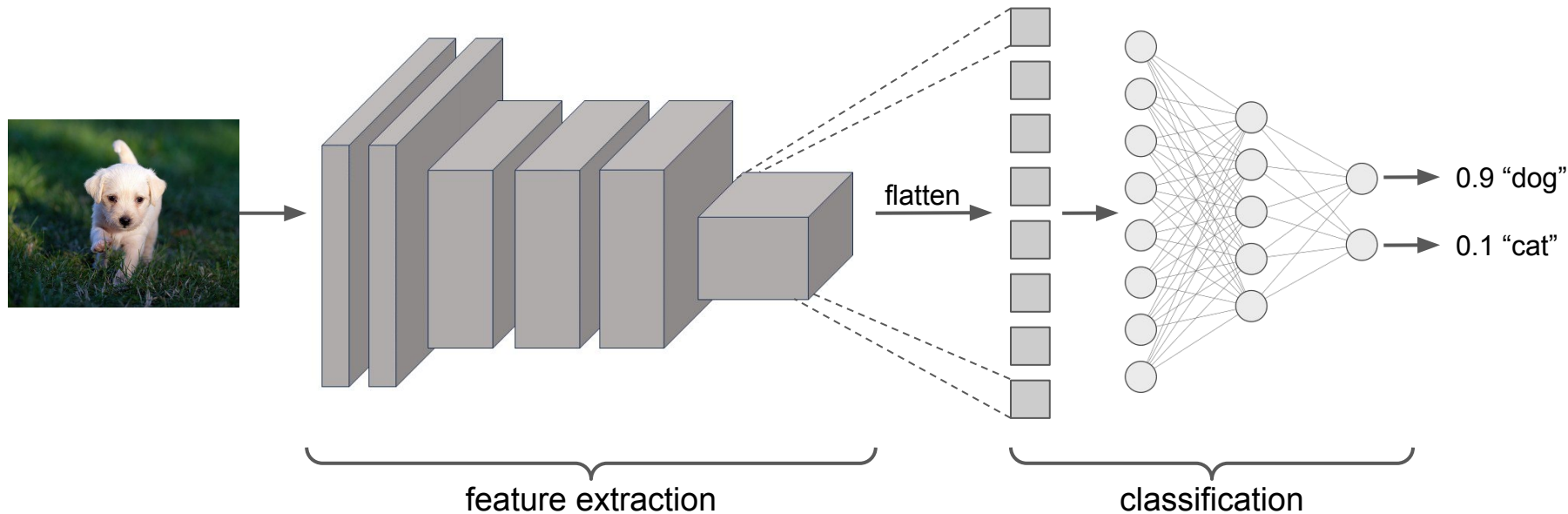
At test time we have:  $E[a] = w_1x + w_2y$

During training we have: 
$$E[a] = \frac{1}{4}(w_1x + w_2y) + \frac{1}{4}(w_1x + 0y) + \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2y) = \frac{1}{2}(w_1x + w_2y)$$

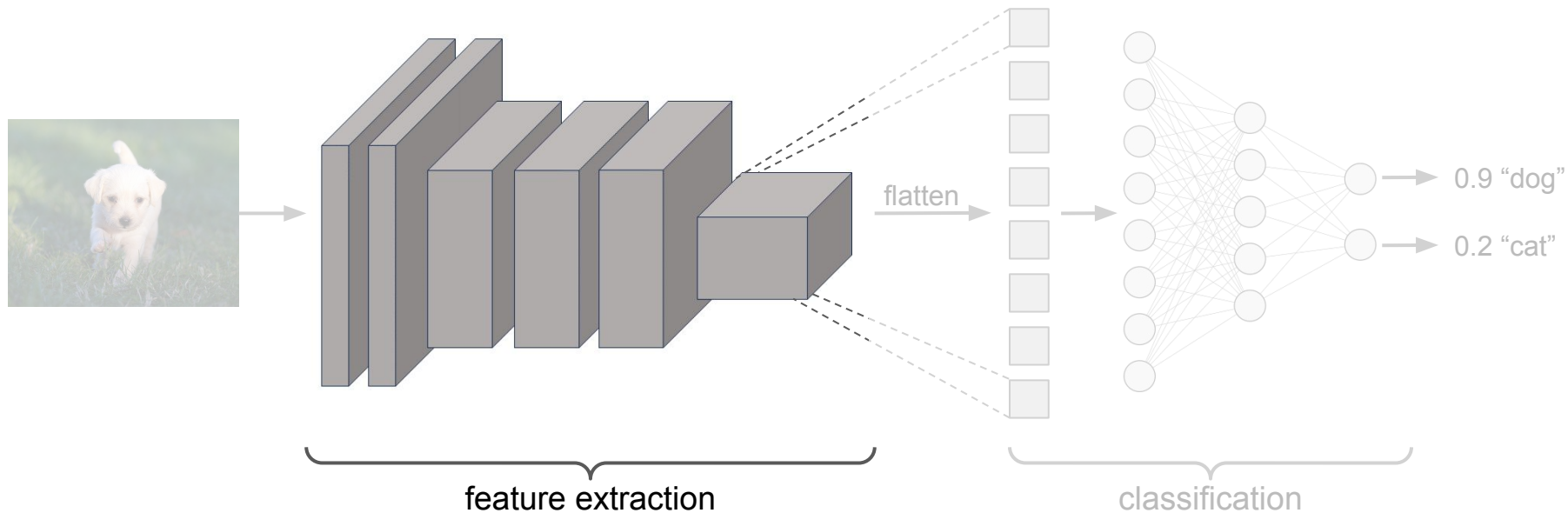
**At test time, multiply by (1 - p)**

# Image Classification

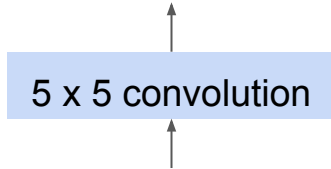
- Important: Everything is differentiable!
- Can calculate gradient of the loss with backpropagation
  - Train with SGD/Adam/etc.
  - Learn convolutional filters and classification head end-to-end!



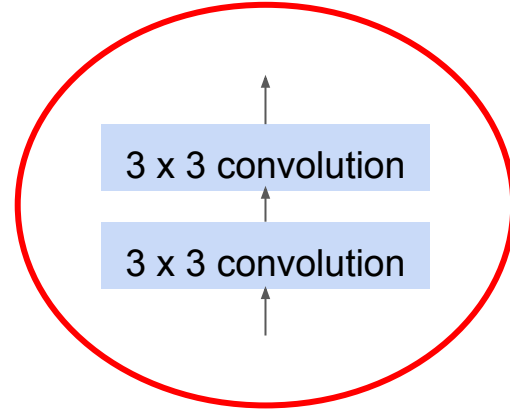
# Deeper CNN Architectures



# Deeper CNN Architectures



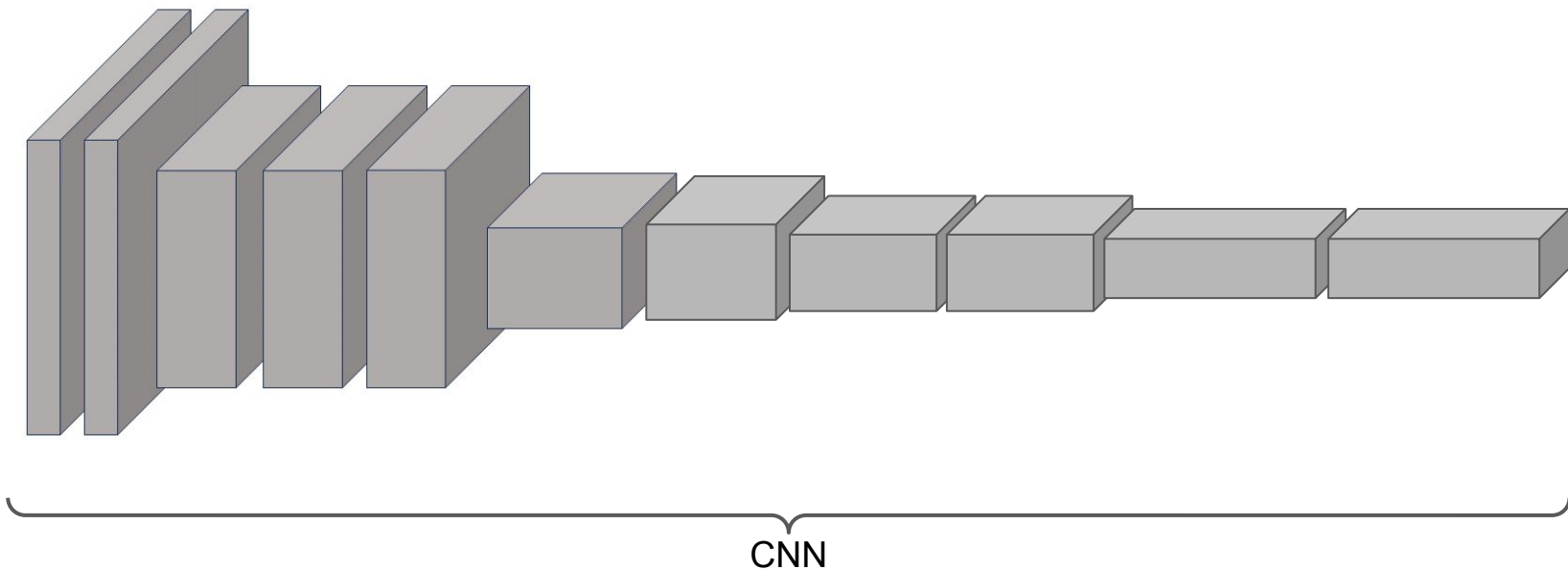
VS



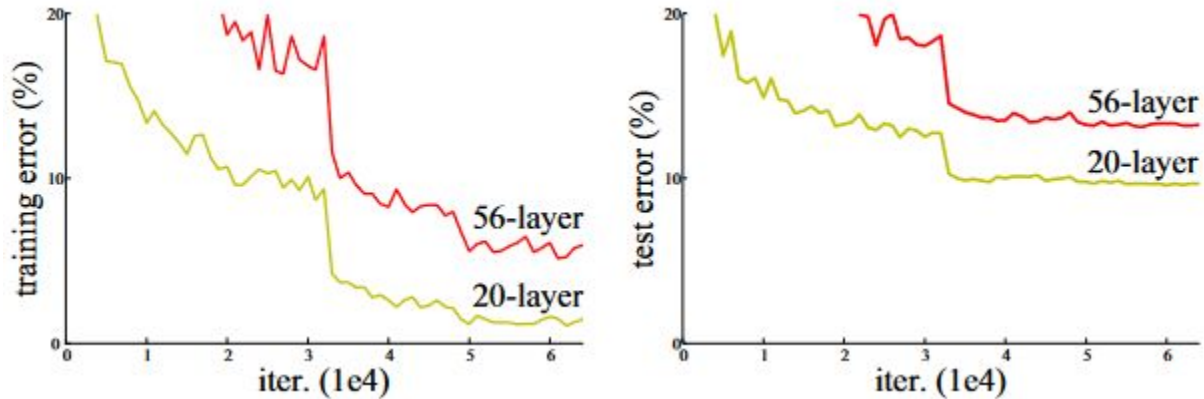
**Performed better!**



Deeper == better

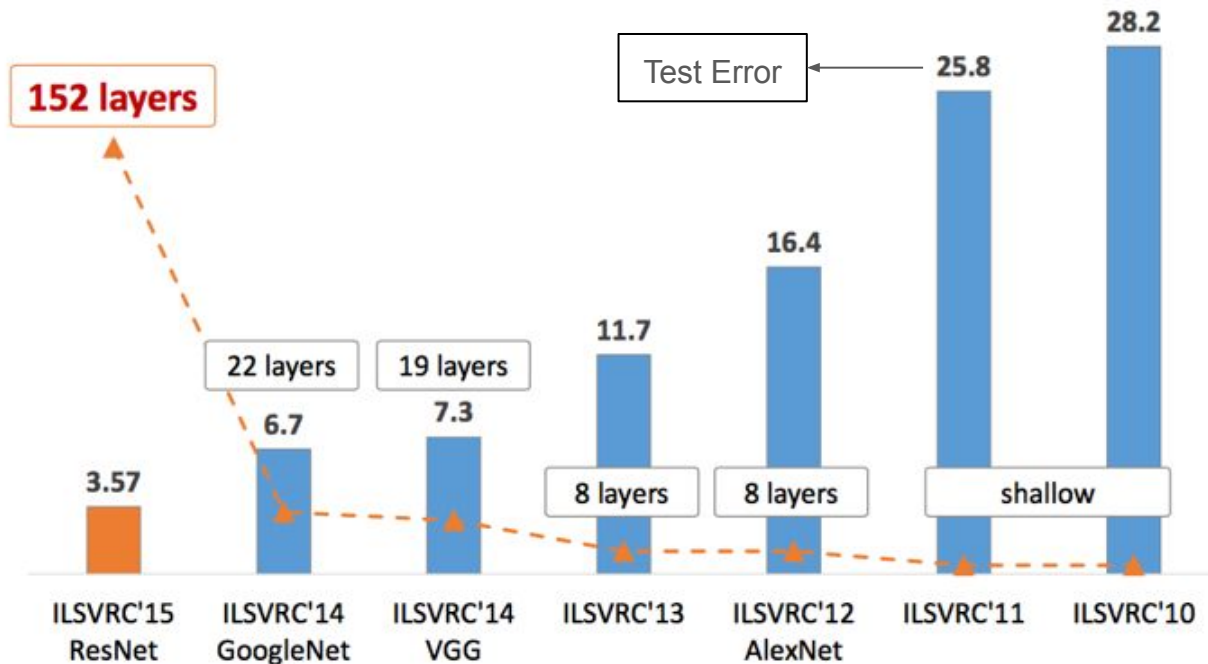


Discuss: How can a larger network achieve a higher training error?



56 layer CNN has higher training and test error than 20 layer CNN on CIFAR-10 dataset for image classification

# ImageNet Classification Challenge: Deeper == better

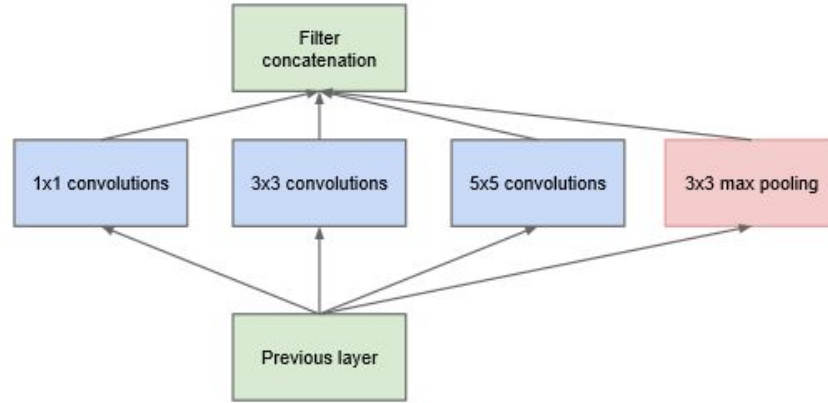


# GoogLeNet/Inception Net

Goal: given a fixed computational budget, optimize the depth and width of the network

=> Deeper networks with computational efficiency

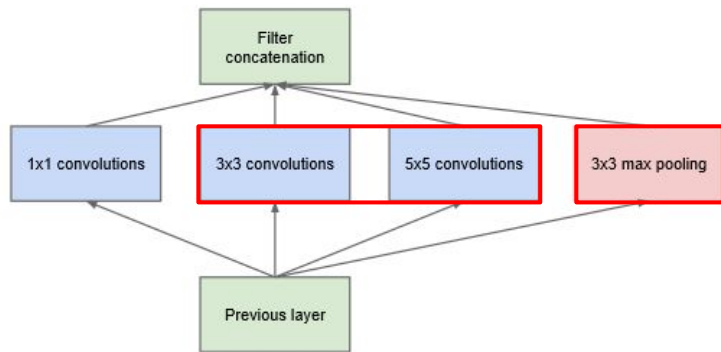
# Inception Module



Inception module = main  
building blocks

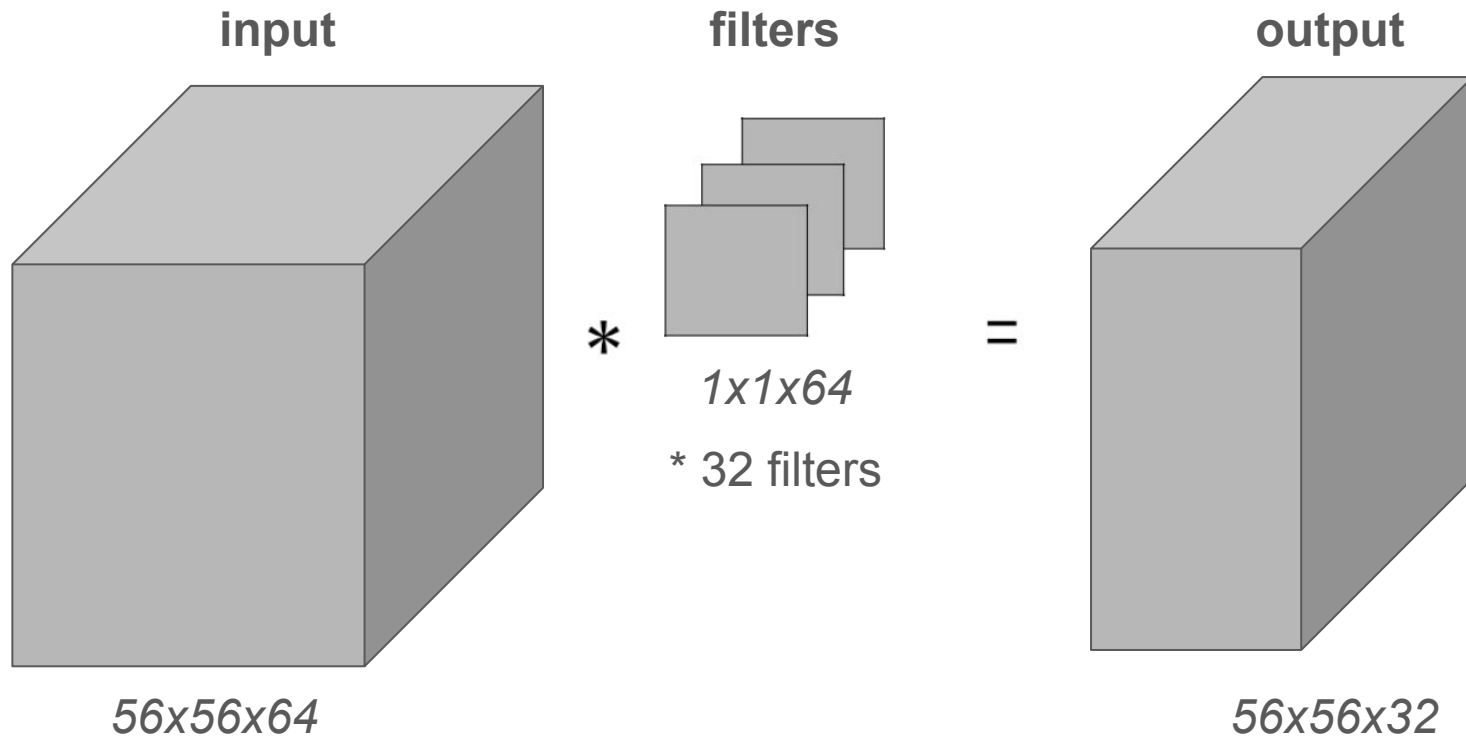
# Inception Module

Still expensive!



- 3x3 and 5x5 convolutions have large number of operations
- Output of pooling layer increases the output channel dimension when concatenated

# Remember: 1x1 convolutions



## Discuss: Impact of Dimension Reduction

Assume you have an input feature map with 256 channels/features.

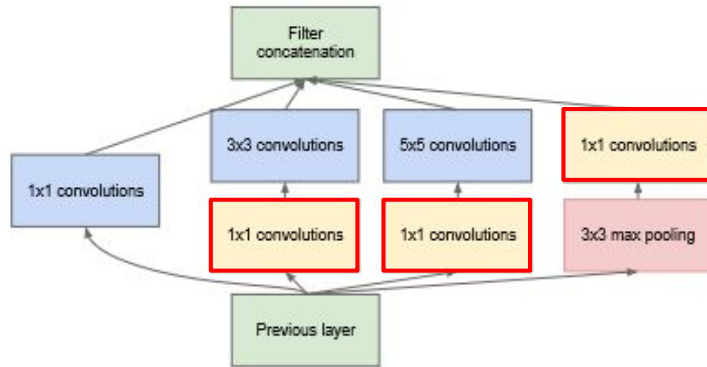
Compare the parameter counts from:

1. 3x3 conv with 256 filters
2. 1x1 conv with 64 filters  $\rightarrow$  3x3 conv with 64 filters  $\rightarrow$  1x1 conv with 256 filters



# Inception Module

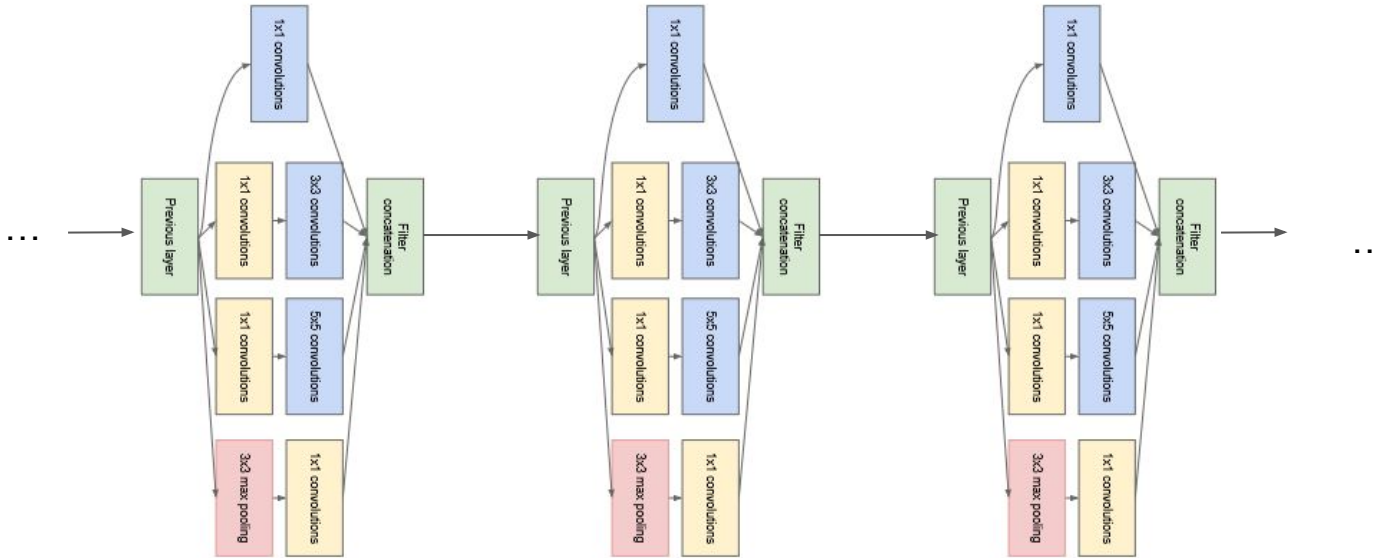
Solution: Inception module with dimension reduction



- “Bottleneck” with 1x1 convolutions to reduce dimensions

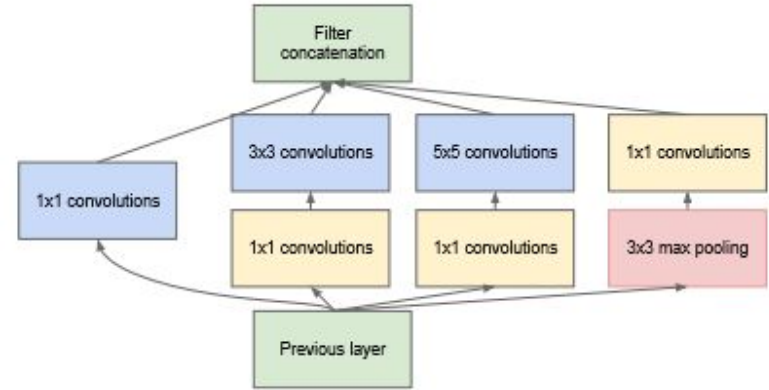
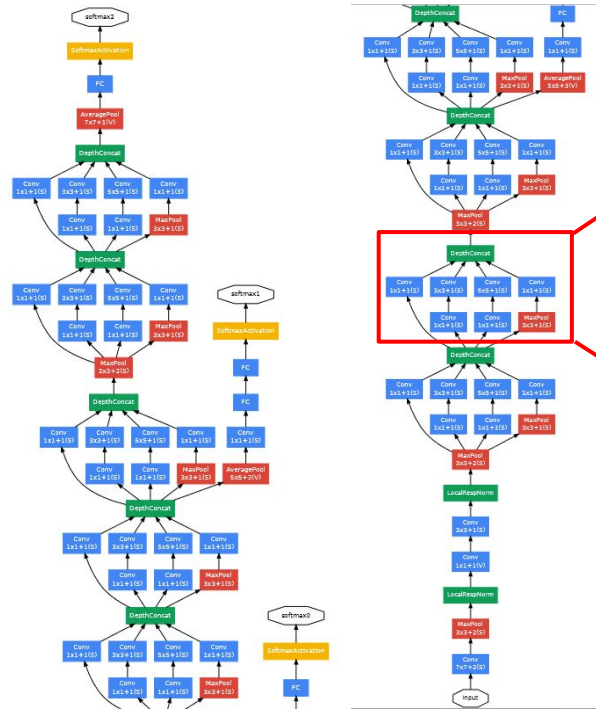
# GoogLeNet Architecture

Key idea: stack inception modules together



[Szegedy, Christian, et al. "Going deeper with convolutions." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015.]

# The Entire GoogLeNet Architecture



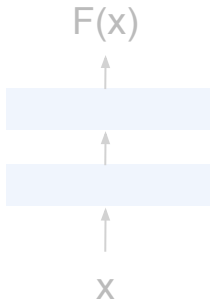
Inception Module

[Szegedy, Christian, et al. "Going deeper with convolutions." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015.]

# CNN Architectures

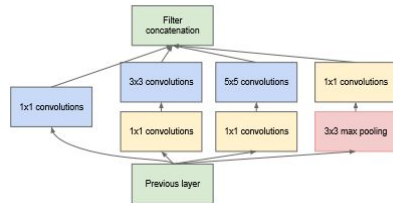
## “Plain” CNN

Simple connection  
from previous to next  
layer

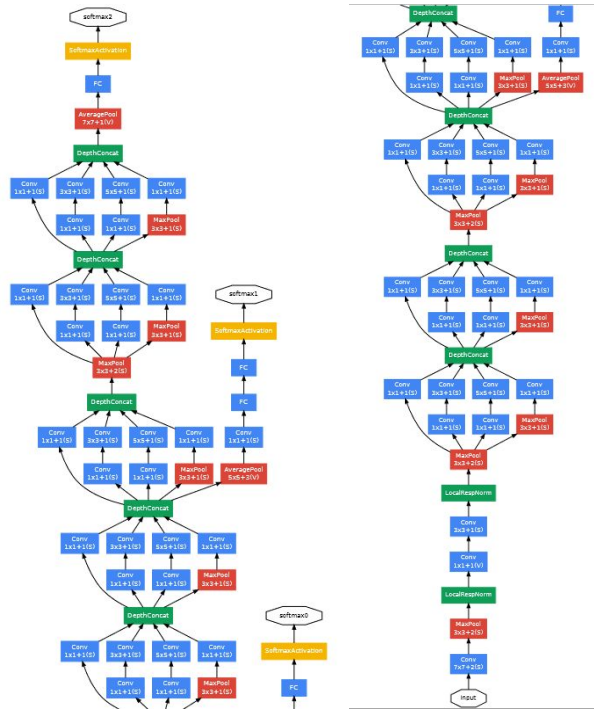


## GoogLeNet

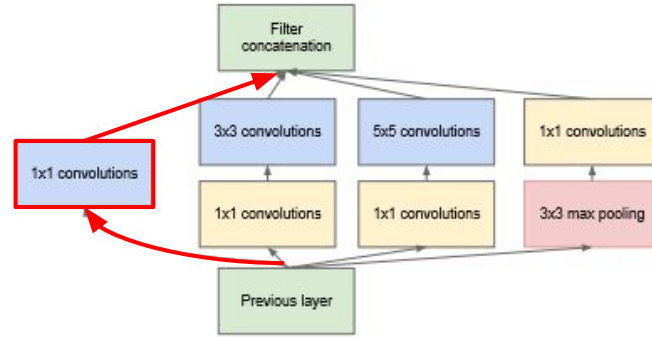
1x1, 3x3, 5x5  
convolutions and  
pooling between each  
layer



# The Entire GoogleNet Architecture

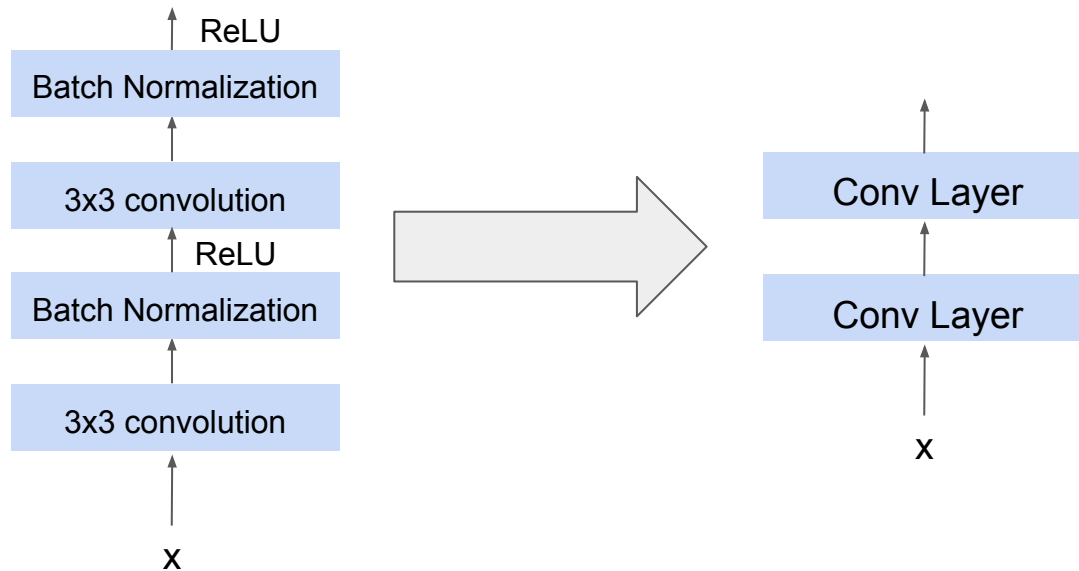


Very complicated - how exactly did this architecture solve the problem?



Residual connections

# Aside: Conv Layer Abstraction

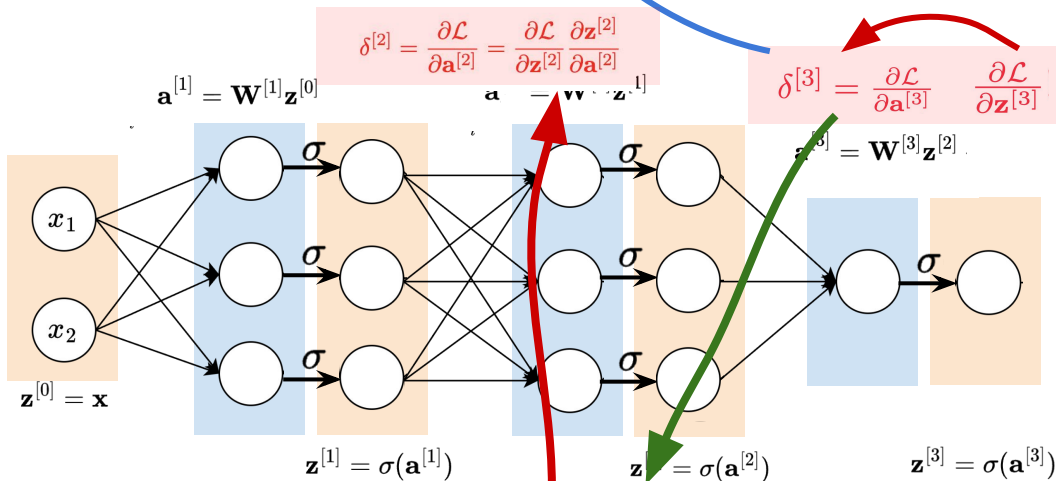


# Backpropagation

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[3]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{W}^{[3]}} = \delta^{[3]} (\mathbf{z}^{[2]})^T$$

## Algorithm Backward Pass through MLP (Detailed)

- 1: **Input:**  $\{\mathbf{z}^{[1]}, \dots, \mathbf{z}^{[L]}\}, \{\mathbf{a}^{[1]}, \dots, \mathbf{a}^{[L]}\},$  loss gradient  $\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[L]}}$
- 2:  $\delta^{[L]} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[L]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[L]}} \frac{\partial \mathbf{z}^{[L]}}{\partial \mathbf{a}^{[L]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[L]}} \odot \sigma^{[L]'}(\mathbf{a}^{[L]})$  ▷ Error term
- 3: **for**  $l = L$  **to** 1 **do**
- 4:  $\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[l]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[l]}} \frac{\partial \mathbf{a}^{[l]}}{\partial \mathbf{W}^{[l]}} = \delta^{[l]} (\mathbf{z}^{[l-1]})^T$  ▷ Gradient of weights
- 5:  $\frac{\partial \mathcal{L}}{\partial \mathbf{b}^{[l]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[l]}} \frac{\partial \mathbf{a}^{[l]}}{\partial \mathbf{b}^{[l]}} = \delta^{[l]}$  ▷ Gradient of biases
- 6:  $\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[l-1]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[l]}} \frac{\partial \mathbf{a}^{[l]}}{\partial \mathbf{z}^{[l-1]}} = (\mathbf{W}^{[l]})^T \delta^{[l]}$
- 7:  $\delta^{[l-1]} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[l-1]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[l-1]}} \frac{\partial \mathbf{z}^{[l-1]}}{\partial \mathbf{a}^{[l-1]}} = ((\mathbf{W}^{[l]})^T \delta^{[l]}) \odot \sigma^{[l-1]'}(\mathbf{a}^{[l-1]})$
- 8: **end for**
- 9: **Output:**  $\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[1:L]}}, \frac{\partial \mathcal{L}}{\partial \mathbf{b}^{[1:L]}}$



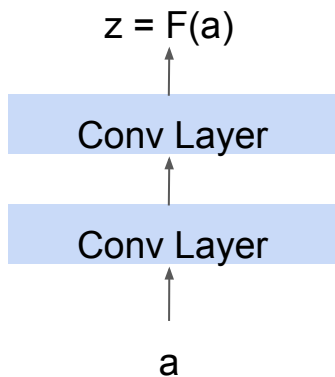
$$\mathcal{L}(\mathbf{z}^{[3]}, \mathbf{y})$$

We can directly compute  $\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[3]}}$ !

$$\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[2]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[2]}} = (\mathbf{W}^{[3]})^T \delta^{[3]}$$

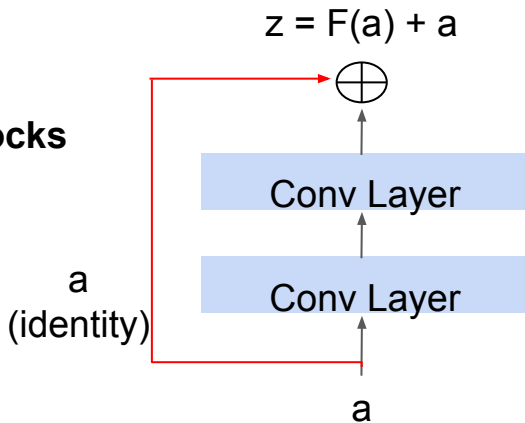
# Discussion: Backpropagation through Residual blocks

“Plain” layers



$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial a} = \frac{\partial L}{\partial z} (F'(a))$$

Residual Blocks

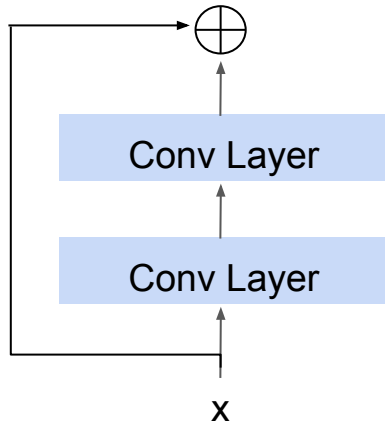


$$\frac{\partial L}{\partial a} =$$



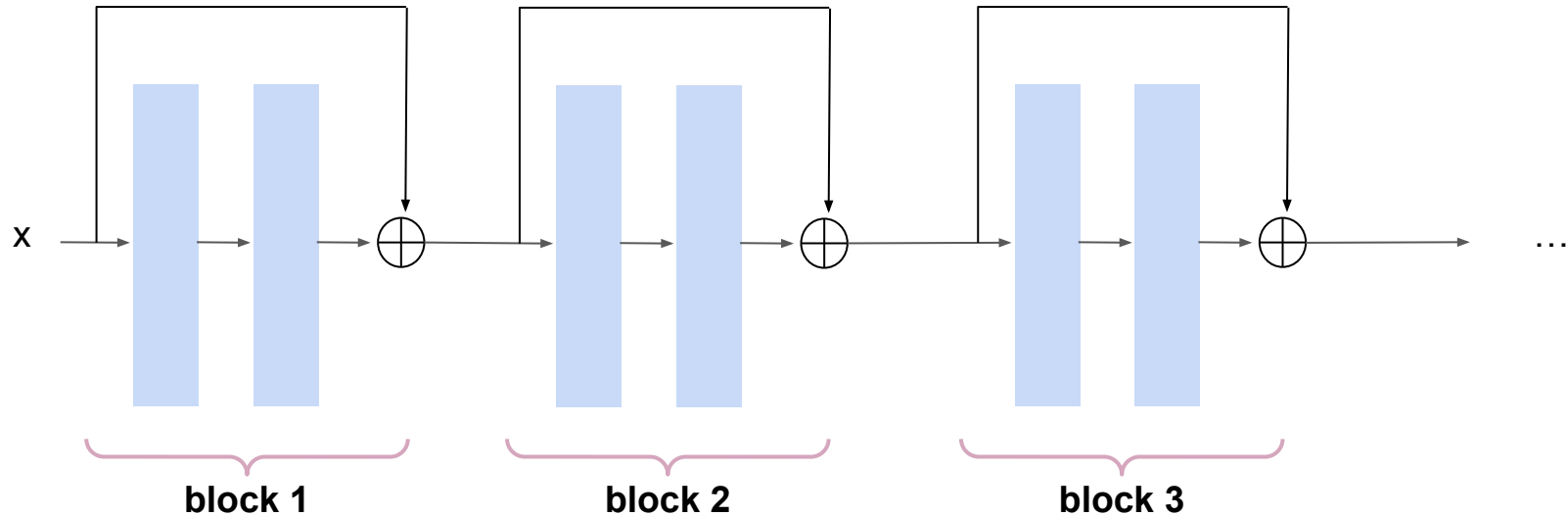
# ResNet

Stack residual blocks together!



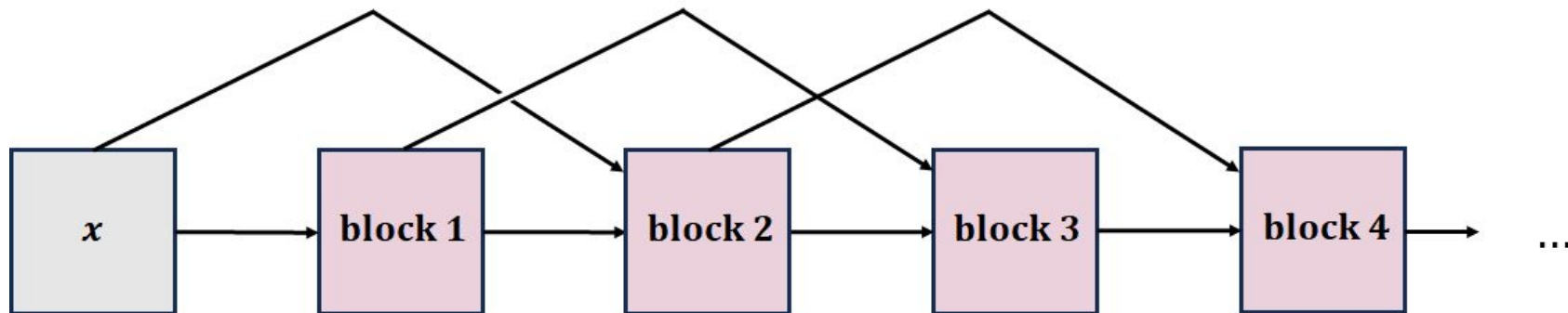
# ResNet

Stack residual blocks together!



# ResNet

Stack residual blocks together!

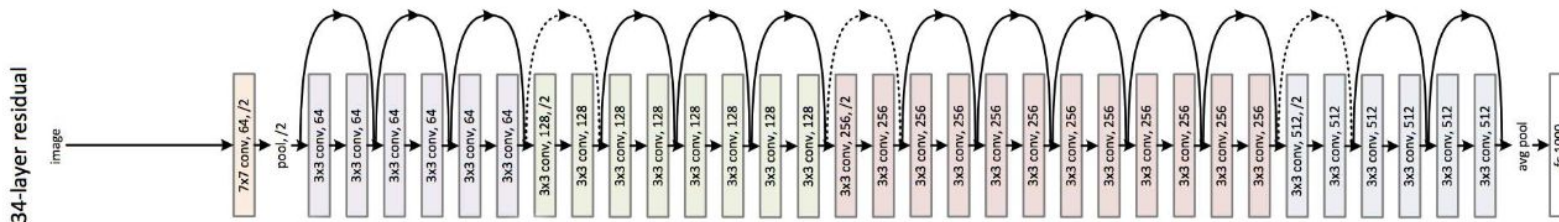


# Full ResNet Architecture

## “Plain” Network

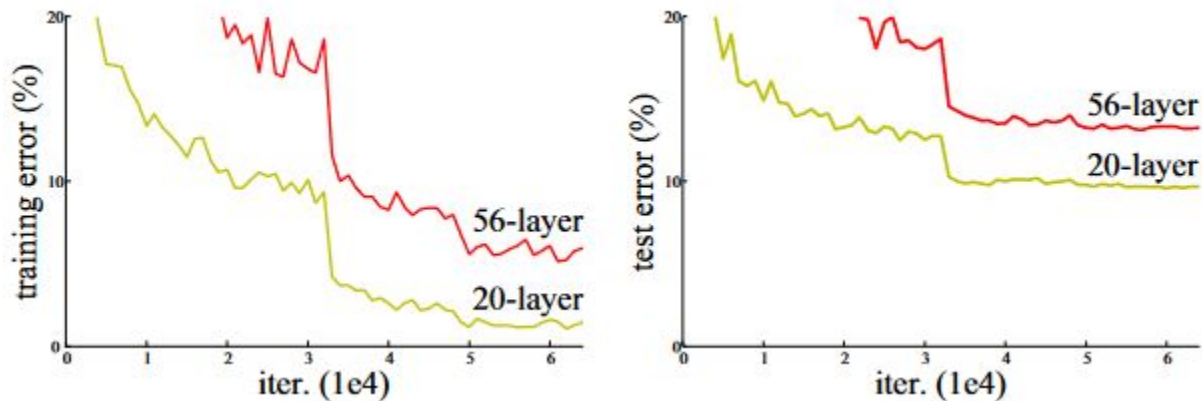


## ResNet



[He, Kaiming, et al. "Deep residual learning for image recognition." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.]

Recall: How can a larger network achieve a higher training error?



56 layer CNN has higher training and test error than 20 layer CNN on CIFAR-10 dataset for image classification

# Deeper == better

Can train deeper models!

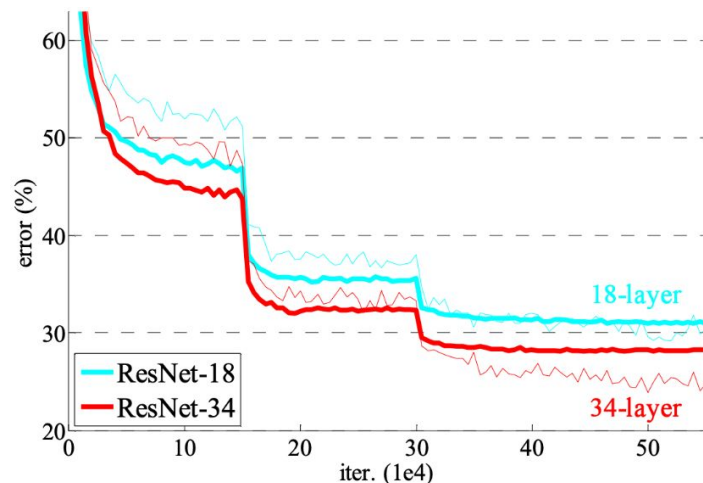
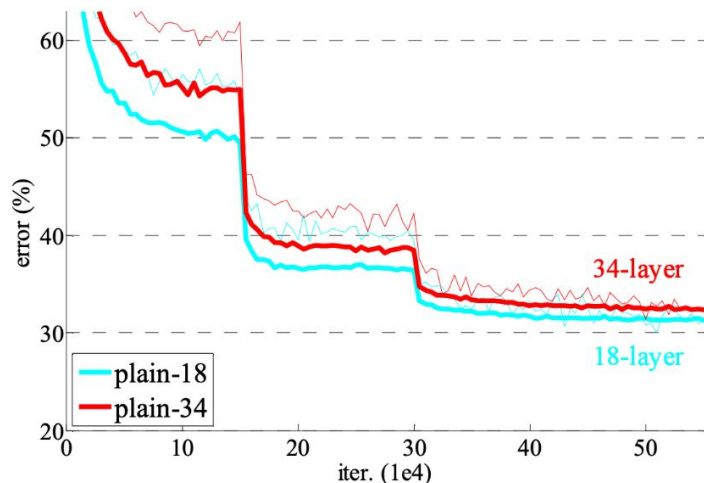
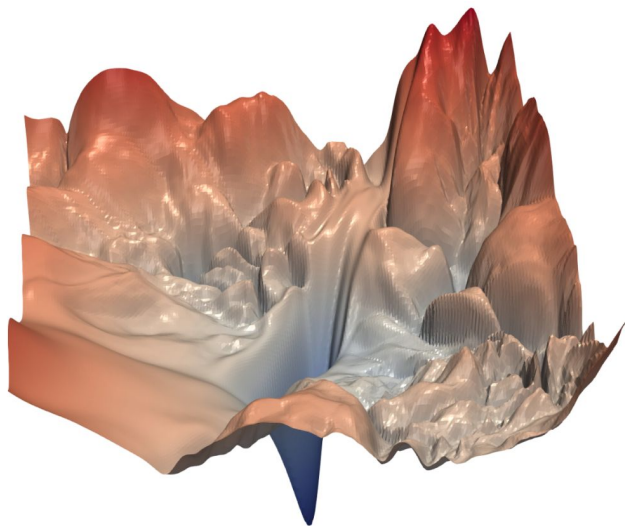


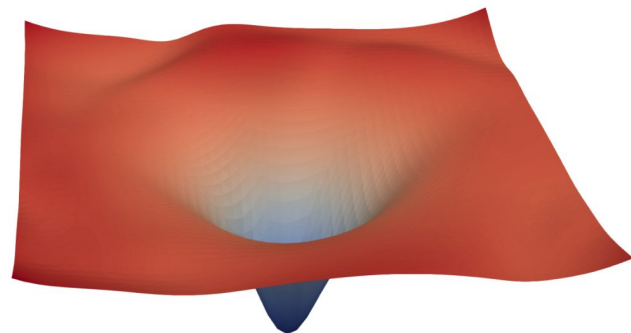
Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

# Visualizing the Effect of Skip Connections

Makes optimization easier!



(a) without skip connections



(b) with skip connections

Figure 1: The loss surfaces of ResNet-56 with/without skip connections. The proposed filter normalization scheme is used to enable comparisons of sharpness/flatness between the two figures.

# Stochastic Depth

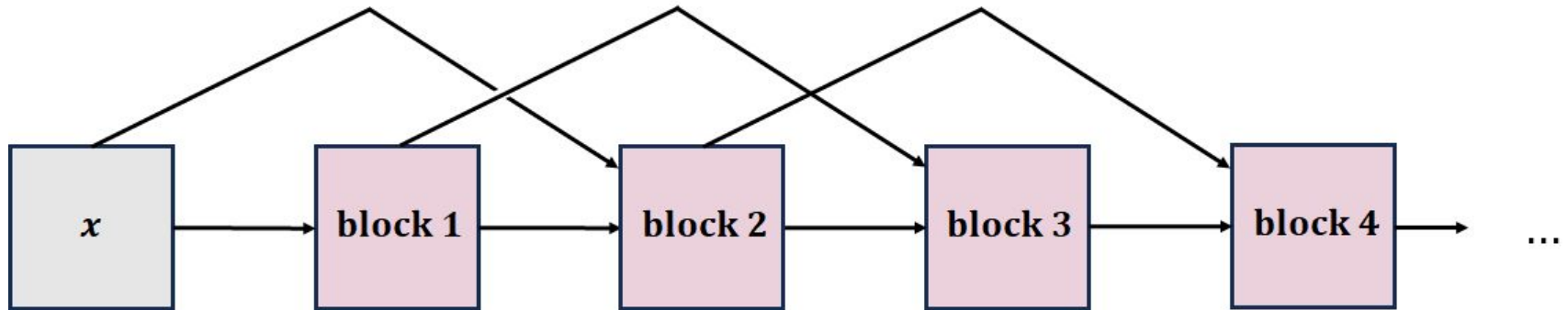
Still have long training times! Solution: stochastic depth



# Stochastic Depth

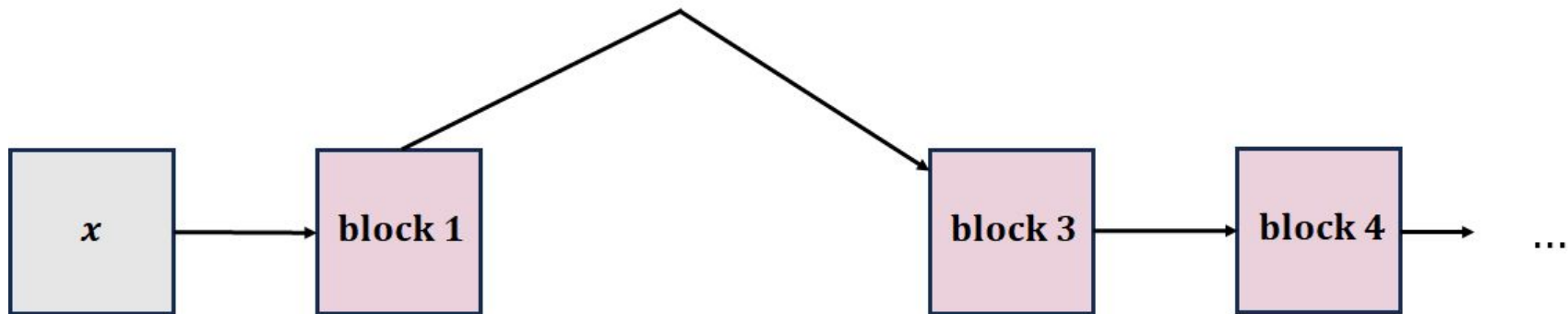
During training, randomly drop Residual Blocks using skip connections

Like dropout but with residual blocks instead of individual neurons



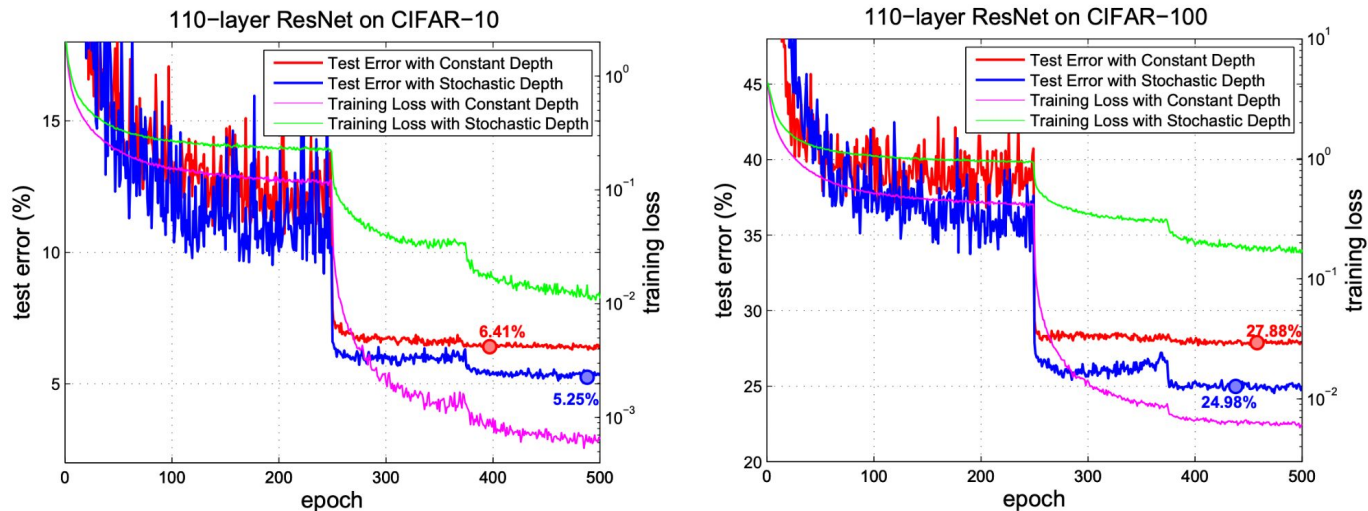
# Stochastic Depth

Another benefit: robustness/mitigating overfitting



# Stochastic Depth

Increases training loss, but... decreases test error!



**Fig. 3.** Test error on CIFAR-10 (*left*) and CIFAR-100 (*right*) during training, with data augmentation, corresponding to results in the first two columns of Table 1.

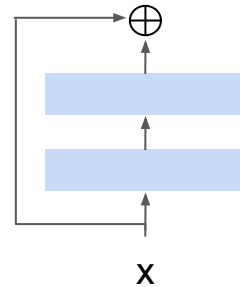
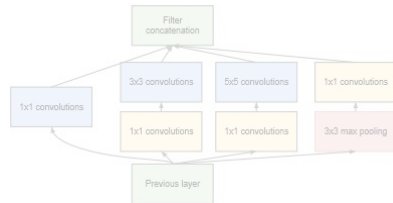
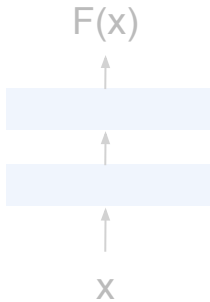
# CNN Architectures

"Plain" CNN	GoogLeNet	ResNet
-------------	-----------	--------

Simple connection from previous to next layer

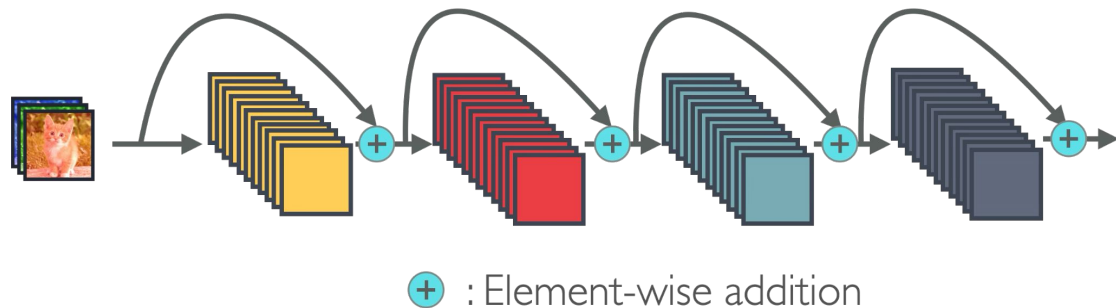
1x1, 3x3, 5x5 convolutions and pooling between each layer

Skip connections  
Add output of previous layer to next layer

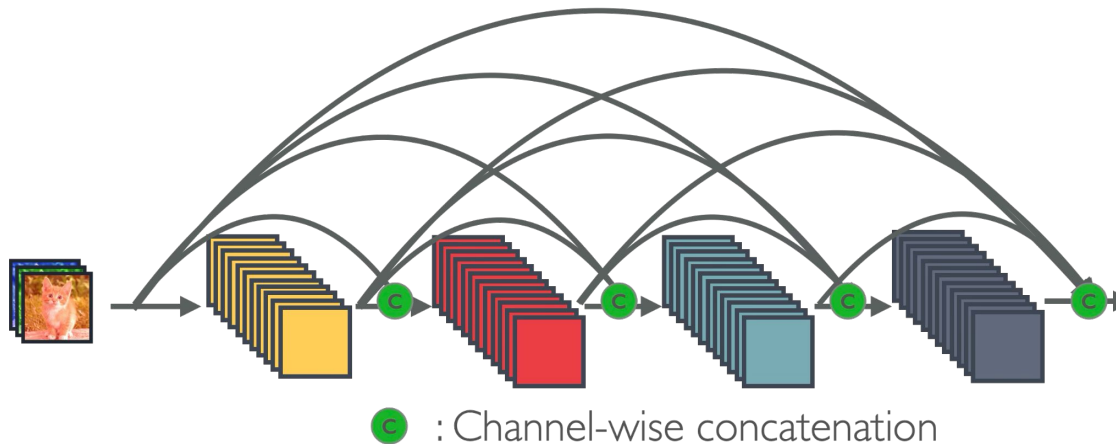


# From ResNets to DenseNets

ResNet

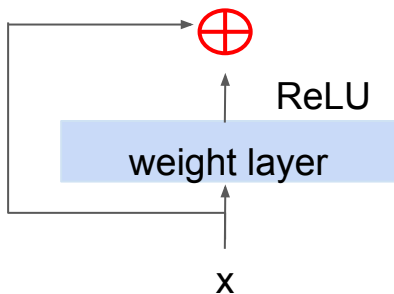


DenseNet

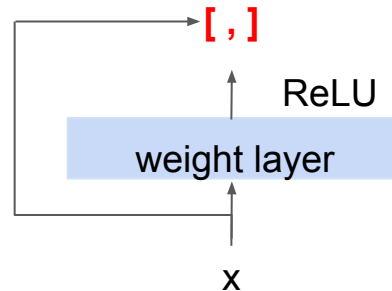


# Dense Blocks

To create dense connections, dense blocks use the same structure as residual blocks, but concatenate (denoted by  $[ , ]$ ) inputs instead of simply adding them



**Residual Blocks**

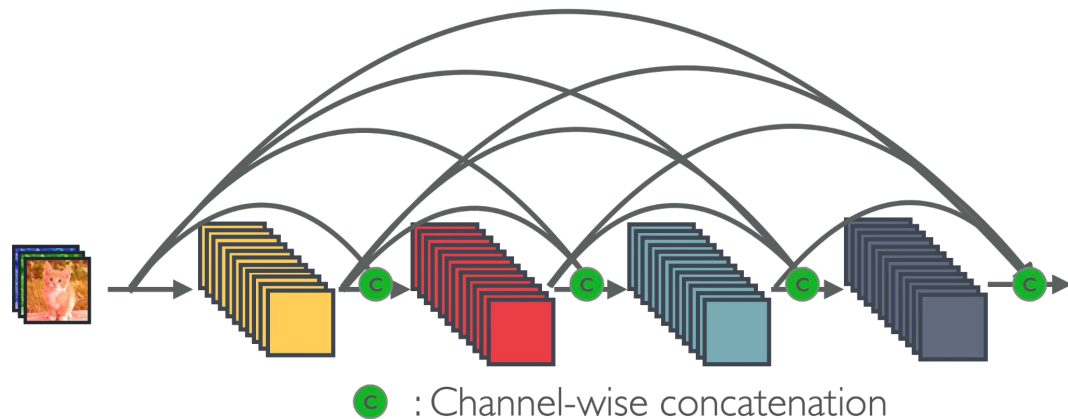


**Dense Blocks**

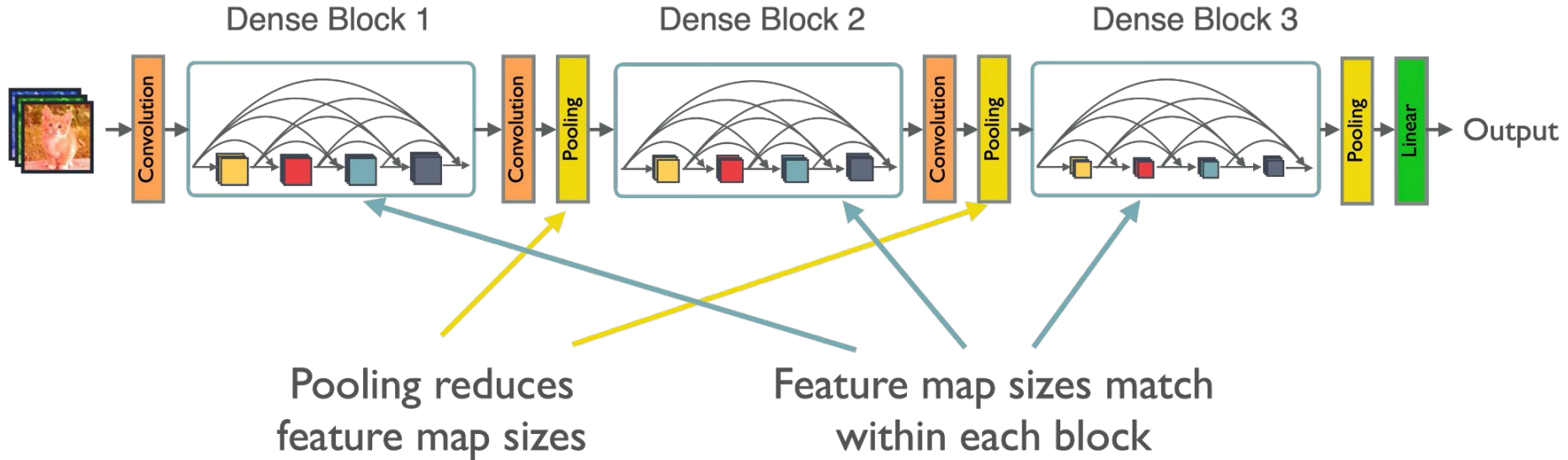
# Dense Connections

Each layer has access to every other layer before it, which:

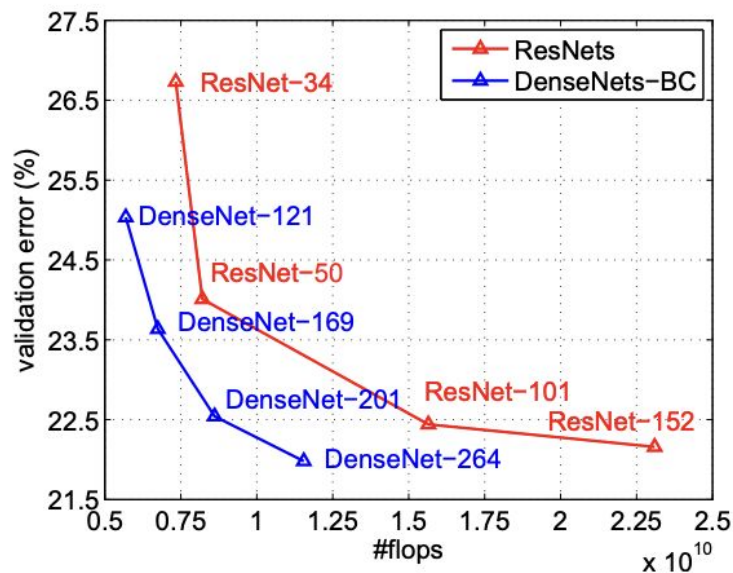
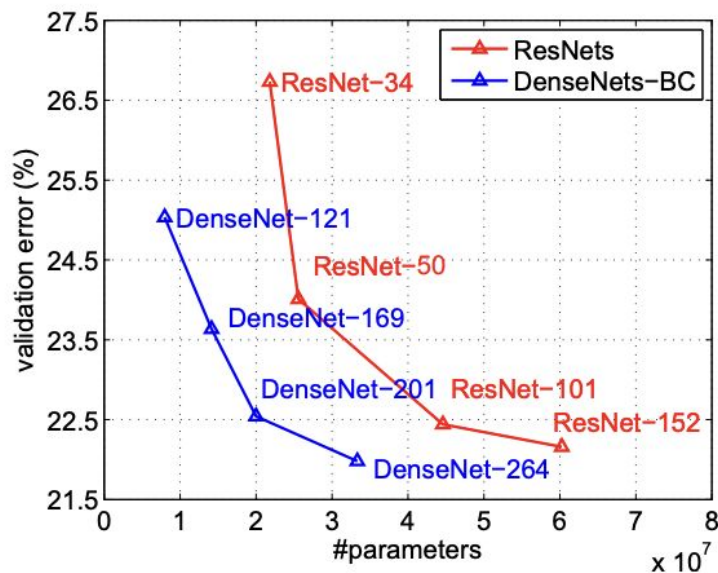
- maximizes information flow
- allows for feature-map reuse
- less parameters to learn
- alleviates vanishing gradient



# DenseNets

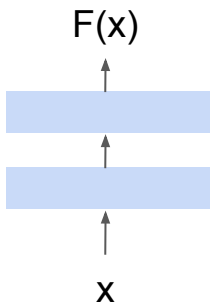
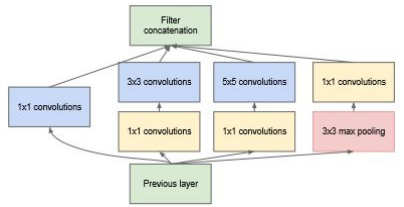
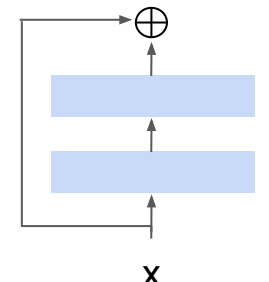
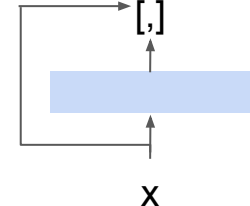






**Figure 3:** Comparison of the DenseNets and ResNets top-1 error rates (single-crop testing) on the ImageNet validation dataset as a function of learned parameters (*left*) and FLOPs during test-time (*right*).

# Summary of Models

"Plain" CNN	Google Net	ResNet	DenseNet
<p>Simple connection from previous to next layer</p>	<p>1x1, 3x3, 5x5 convolutions and pooling between each layer</p>	<p>Skip connections Add output of previous layer to next layer</p>	<p>Dense connections Concatenate output of previous layer to next layer</p>
			

# Summary

- Deep CNNs outperform shallow CNNs
- But...
  - Harder optimization problem!
- Residual (and dense) connections make training easier!
  - Can train networks with 100s of layers!
- Stochastic depth let's you train deeper networks faster
  - 1000+ layers!
- In general...
  - Build large networks as stacks of (many!) simple building blocks