



# Cornell Bowers C-IS

## College of Computing and Information Science

Join by  
Web

**PollEv.com**  
**/kilianweinberger352**

Join by  
Text

Send **kilianweinberger352** to **076**  
**943 91 62**



# Modern Convolutional Neural Networks 2

CS4782: Intro to Deep Learning

Varsha Kishore, Justin Lovelace, Anissa Dallmann, Stephanie Ginting

# Convolutional Neural Networks (CNNs)

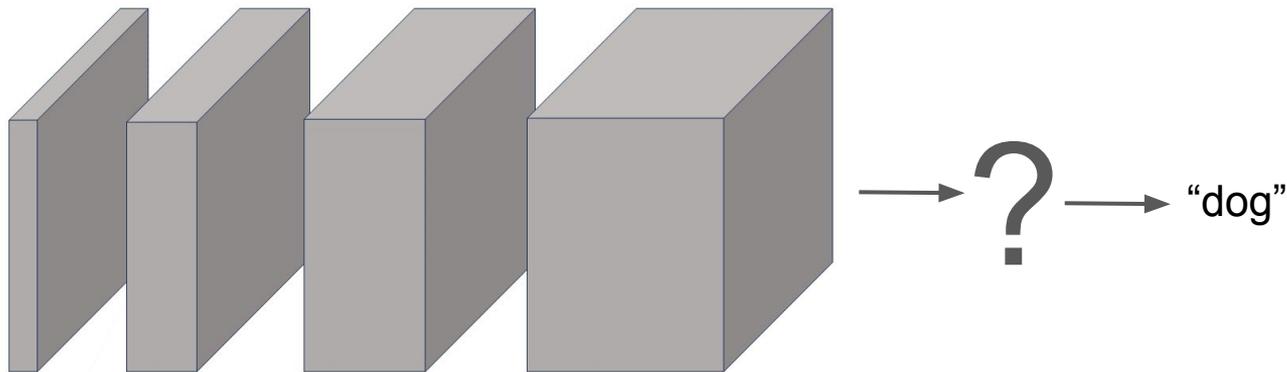
## ✓ Convolutions

Maintain spatial relation between pixels

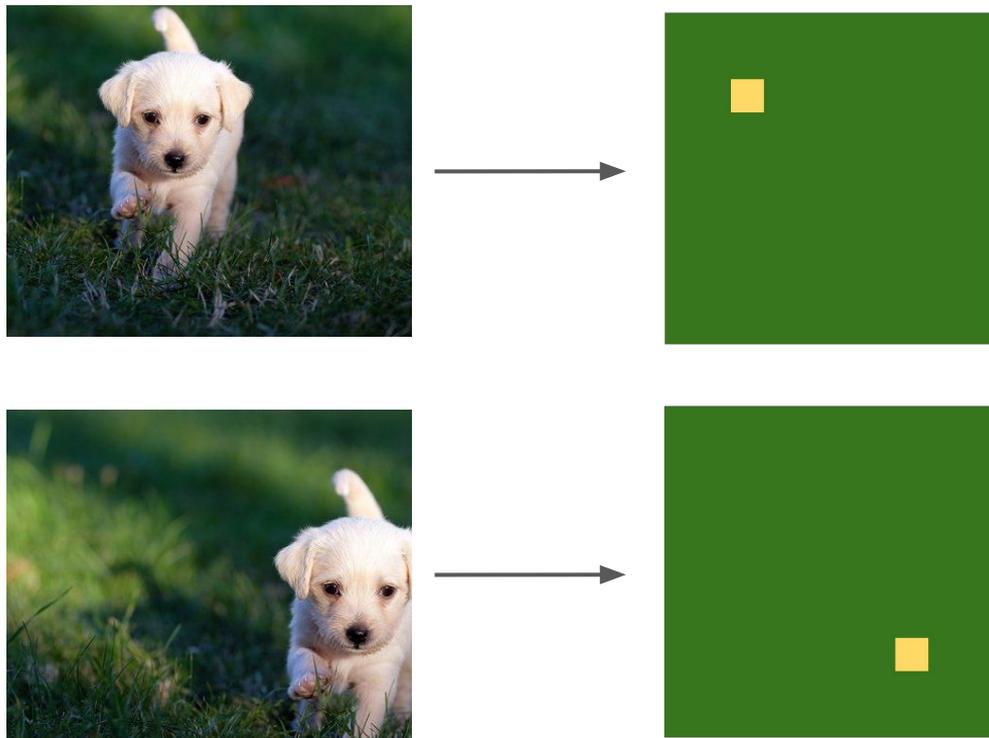
Reduce number of parameters through weight sharing



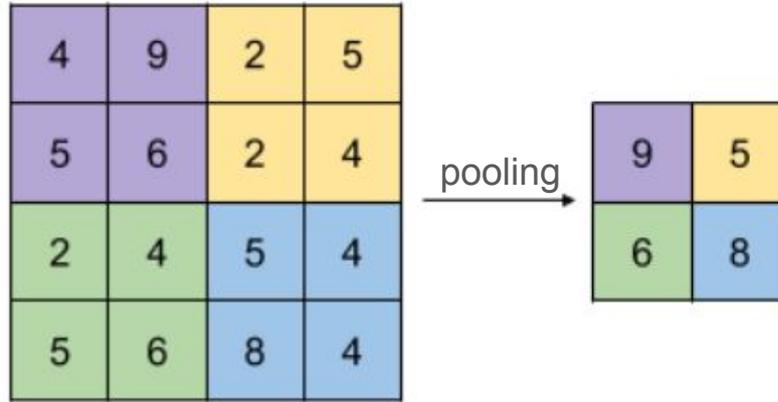
input image



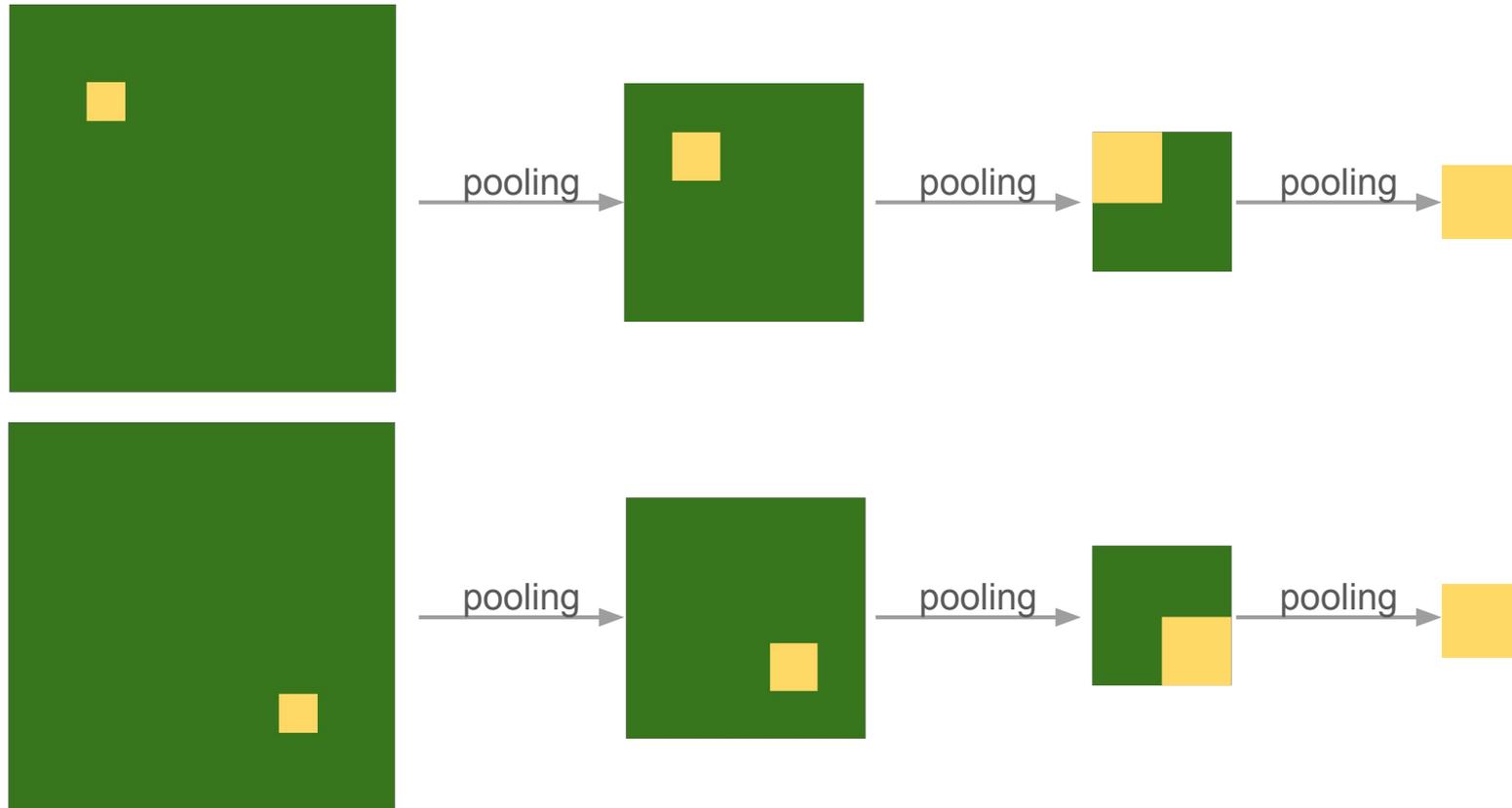
# Ensuring translational invariance



# Max Pooling



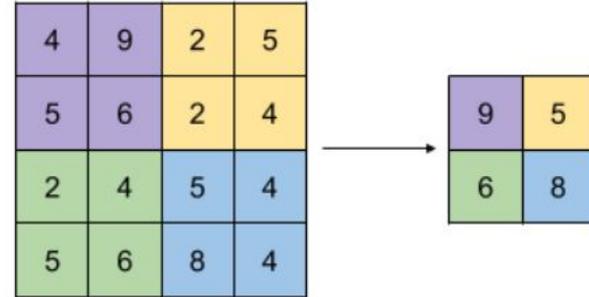
# CNNs - Pooling



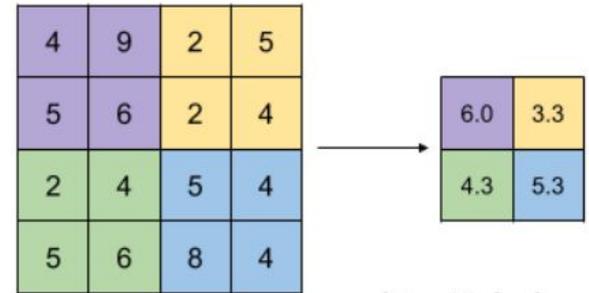
# CNNs - Pooling

- ❖ Down sample feature maps that highlight the most present feature in the patch
- ❖ Improve efficiency by reducing computations with downsampling
- ❖ Increase receptive field size

**Max Pooling**



**Avg Pooling**



# Convolutional Neural Networks (CNNs)

✓ Convolutions

Maintain spatial relation between pixels

Reduce number of parameters through weight sharing

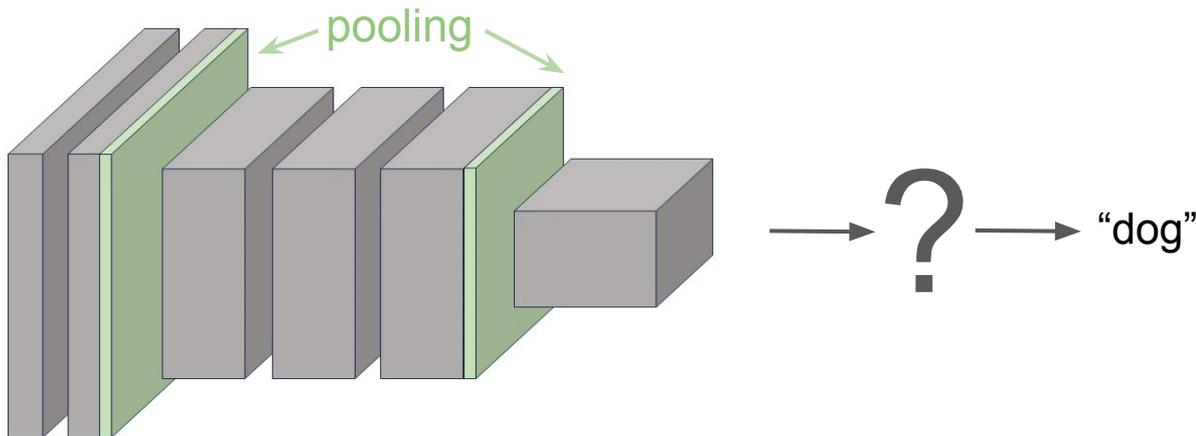
✓ Pooling

Captures key information from across different areas of the feature maps

Together with convolutions allows for translational invariance

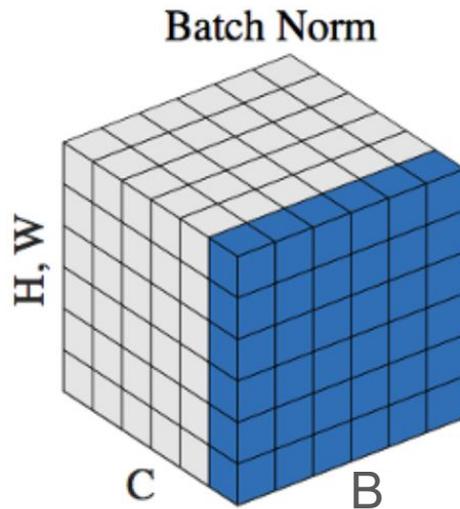


input image



# Normalization

- ❖ Normalize channels to mean 0 and variance 1 across each training batch
- ❖ Increases speed of training by enabling the use of larger learning rates
- ❖ Improves stability of training
- ❖ **B=Bach, C=Channels, H=Height, W=Width**



This cube is 4D (but we can collapse H,W into one dimension.)

# The Batch Normalization Algorithm

**Input:** Values of  $x$  over a mini-batch:  $\mathcal{B} = \{x_{1\dots m}\}$ ;

Parameters to be learned:  $\gamma, \beta$

**Output:**  $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{ mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{ mini-batch variance}$$

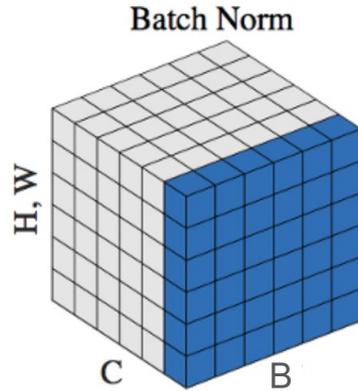
$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{ normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \quad // \text{ scale and shift}$$

**Algorithm 1:** Batch Normalizing Transform, applied to activation  $x$  over a mini-batch.

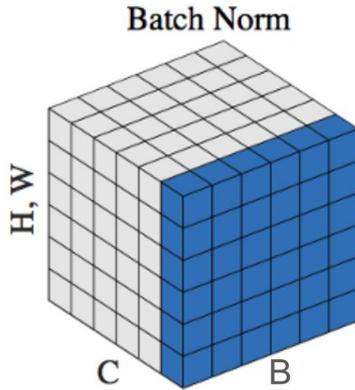
# Discuss!

What is the dimension of the mean when you compute the batch norm of a volume of dimension  $(b \times c \times h \times w)$ ?



# Discuss!

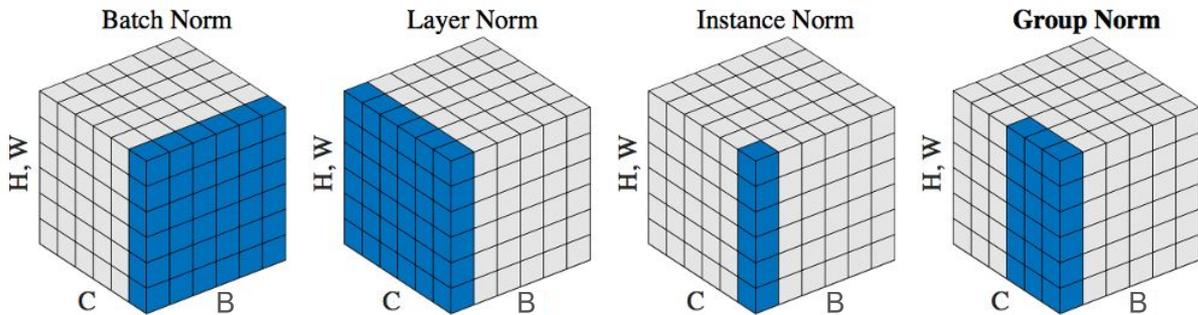
What is the dimension of the mean when you compute the batch norm of a volume of dimension  $(b \times c \times h \times w)$ ?



**The mean should be  $(1 \times c \times 1 \times 1)$ !**

# Normalization Layers

- Normalization layers improve training stability
- Can train with larger learning rates
  - Faster training
- A large learning rate acts as an implicit regularizer
  - Better generalization
- Normalization can also be applied across different dimensions for different use cases



# Convolutional Neural Networks (CNNs)

✓ Convolutions

Maintain spatial relation between pixels  
Reduce number of parameters through weight sharing

✓ Pooling

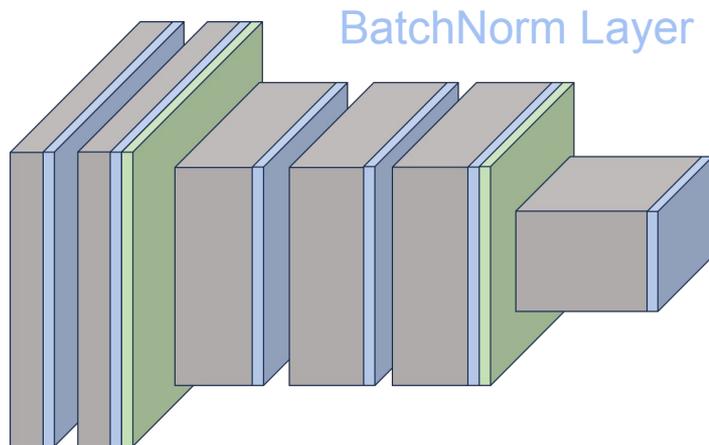
Captures key information from across different areas of the feature maps  
Together with convolutions allows for translational invariance

✓ BatchNorm

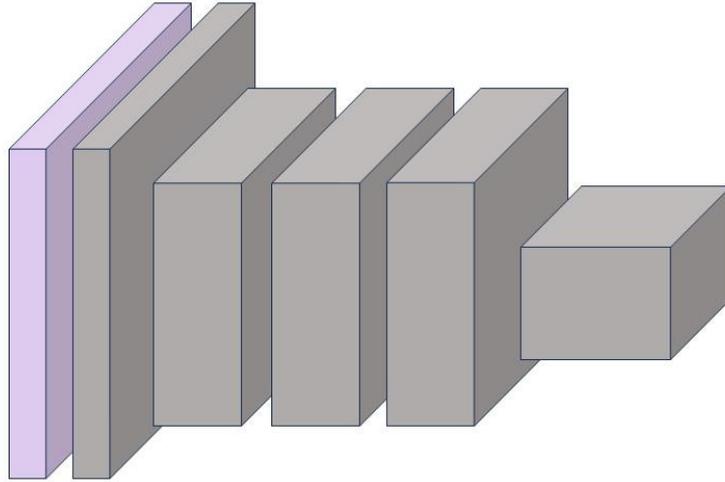
Increases speed and stability of training



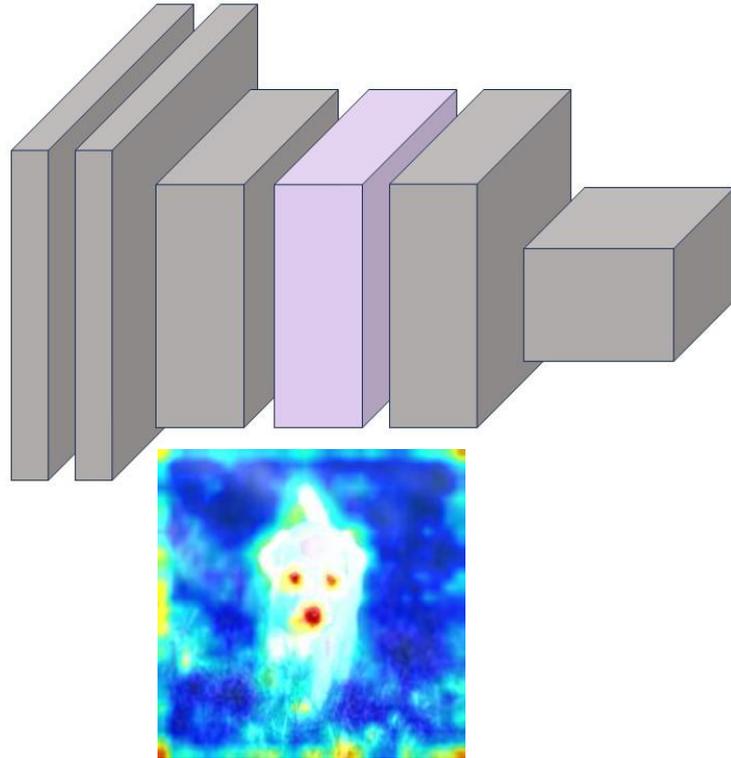
input image



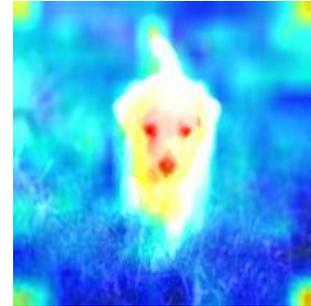
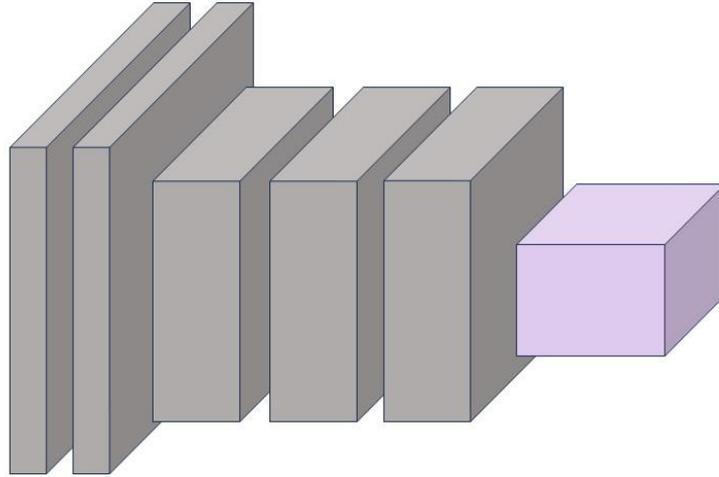
# Convolutional Neural Networks (CNNs)



# Convolutional Neural Networks (CNNs)



# Convolutional Neural Networks (CNNs)



# Convolutional Neural Networks (CNNs)

✓ Convolutions

Maintain spatial relation between pixels  
Reduce number of parameters through weight sharing

✓ Pooling

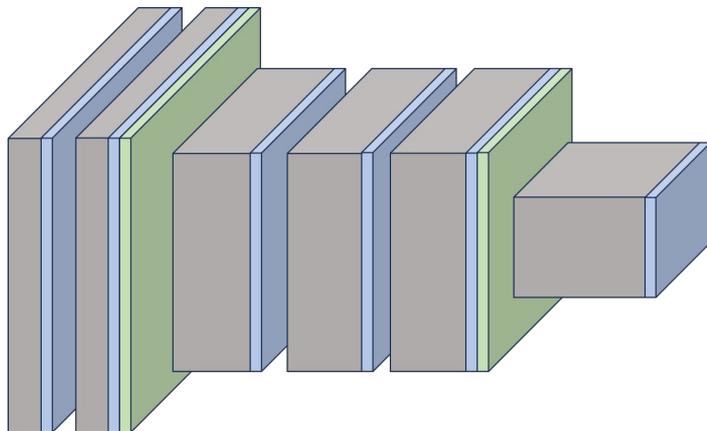
Captures key information from across different areas of the feature maps  
Together with convolutions allows for translational invariance

✓ BatchNorm

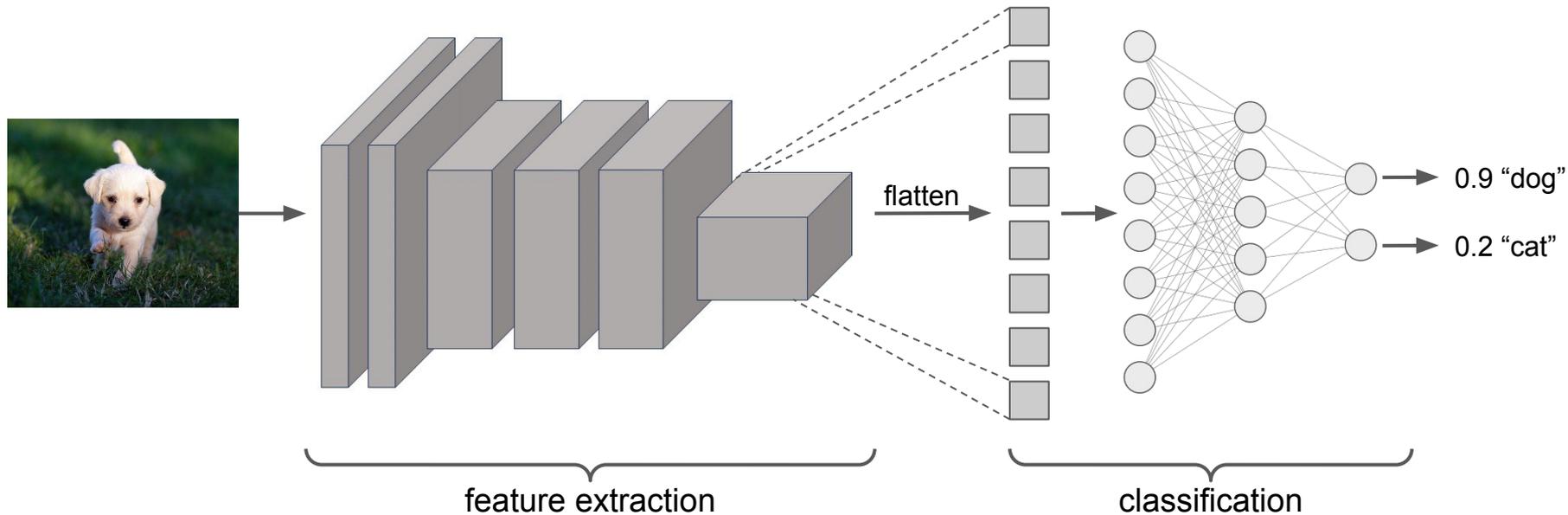
Increases speed and stability of training



input image



# Image Classification



# Practical Guide

- Input image dimensions is divisible by 2
- Small conv filters (3x3 or 5x5)
- Zero padding is used to maintain spatial resolution
- Max pooling for downsampling
- Pooling layers have a receptive field of 2 and stride of 2

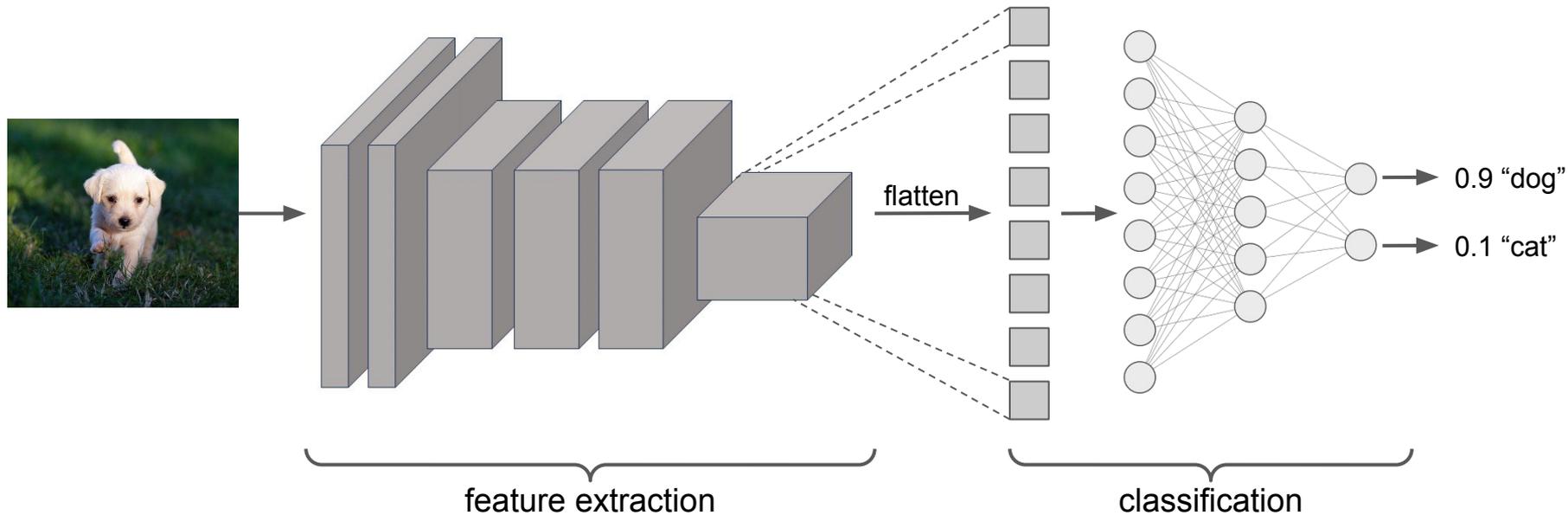
# Summary

- CNNs are primarily designed to process and analyze visual data, such as images and videos.
- Key components: convolution layers, pooling layers, activation functions, normalization layers
- Advantages:
  - Translational Invariance
  - Parameter sharing
  - Feature learning
- Can be trained with backprop
- Used for tasks such as segmentation, classification, object detection, etc.

# Logistics

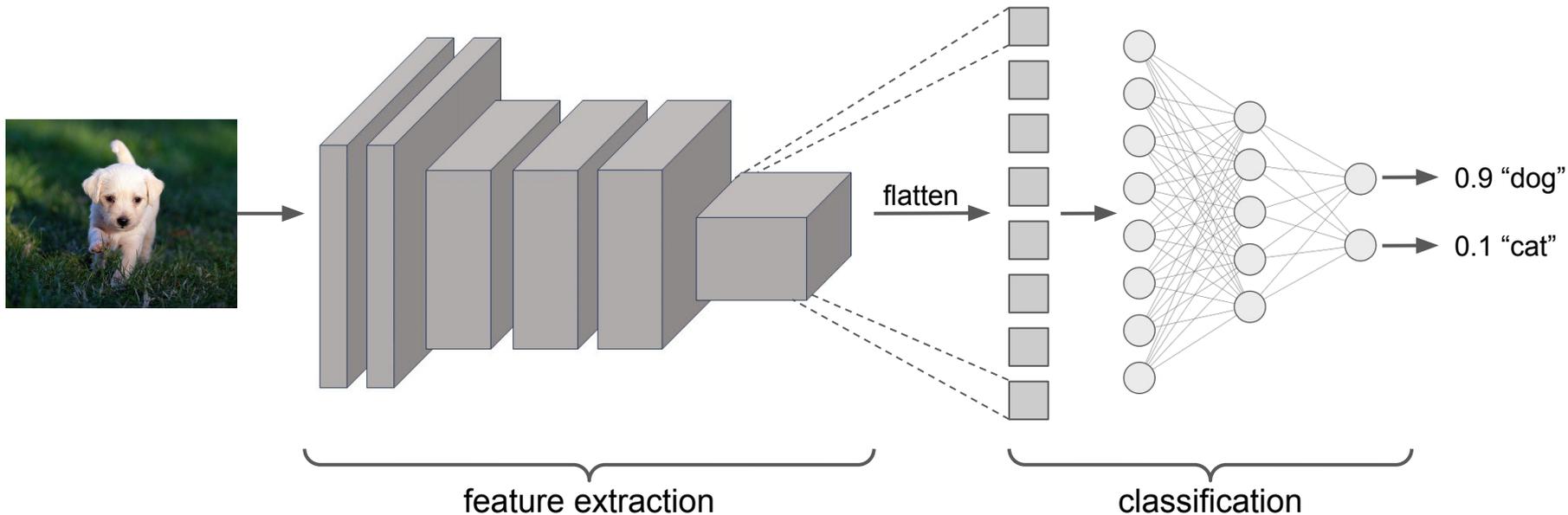
- **HW1+P1** is due Thursday (February 12) 11:59 PM
- Late submissions accepted until Saturday (February 14) 11:59 PM
  
- **HW2** to be released this Thursday (February 12) - due Thursday (March 5)
- **P2** to be released this Thursday (February 12) - due Thursday (February 26)
  
- Office hours are listed on the course website
- Homework clarifications are listed as pinned posts under HW1 on Ed
- Post questions on Ed

# Review: Image Classification



# Image Classification

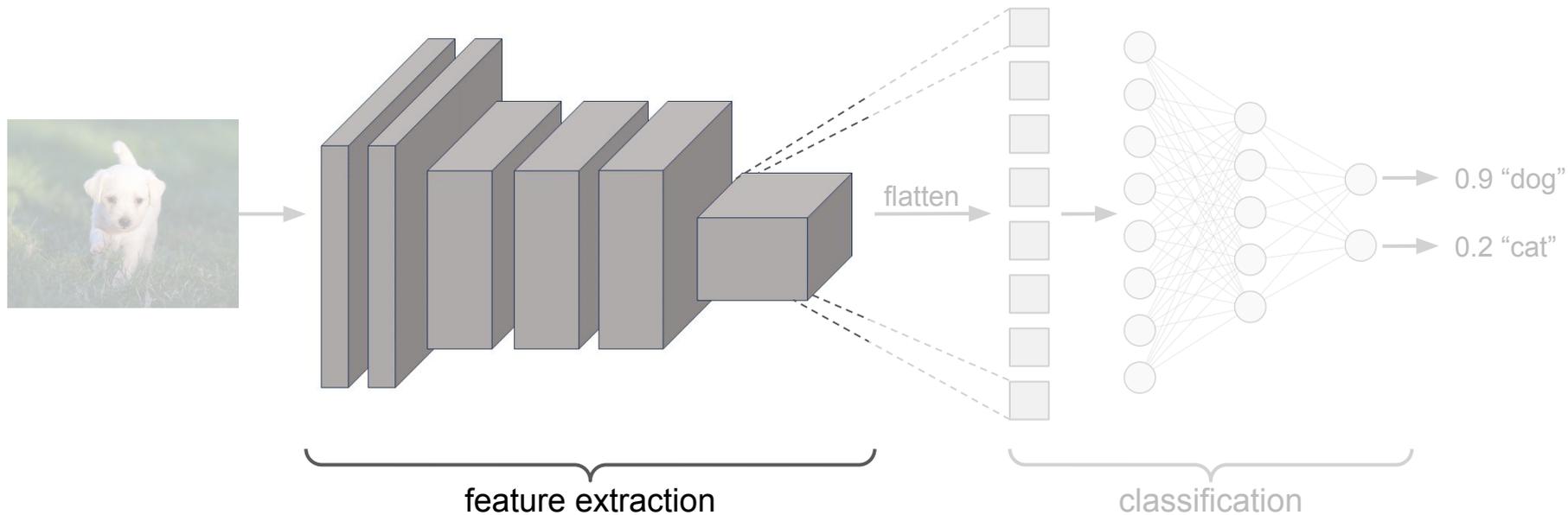
- Important: Everything is differentiable!
- Can calculate gradient of the loss with backpropagation
  - Train with SGD/Adam/etc.
  - Learn convolutional filters and classification head end-to-end!



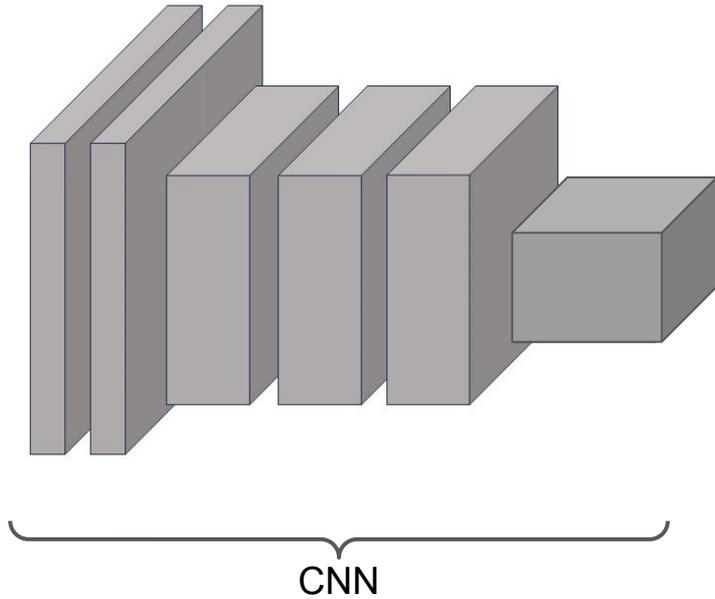
# Discuss: Padding

- Given a 5x5 feature map and a 3x3 convolution:
  - How much padding do I need to maintain the spatial size of the feature map (i.e., 5x5)?
- What about when using a 5x5 convolution?

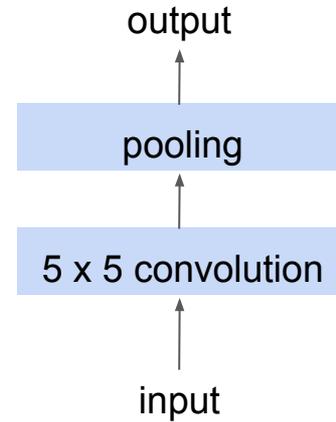
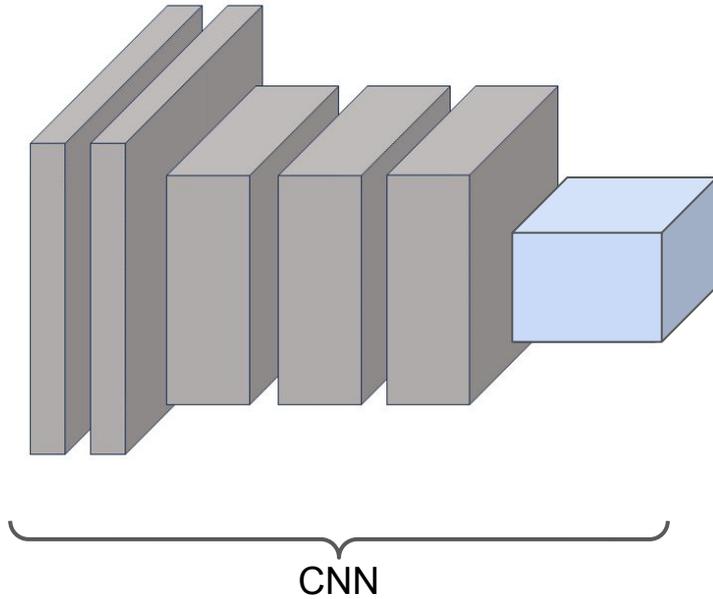
# Deeper CNN Architectures



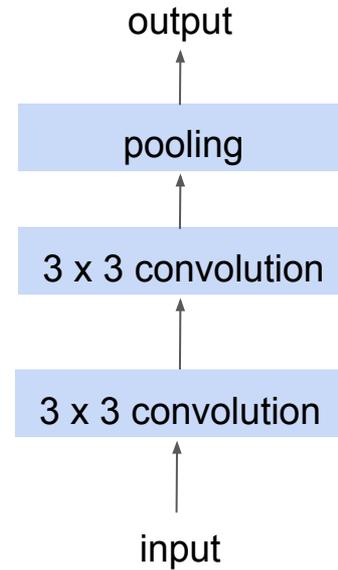
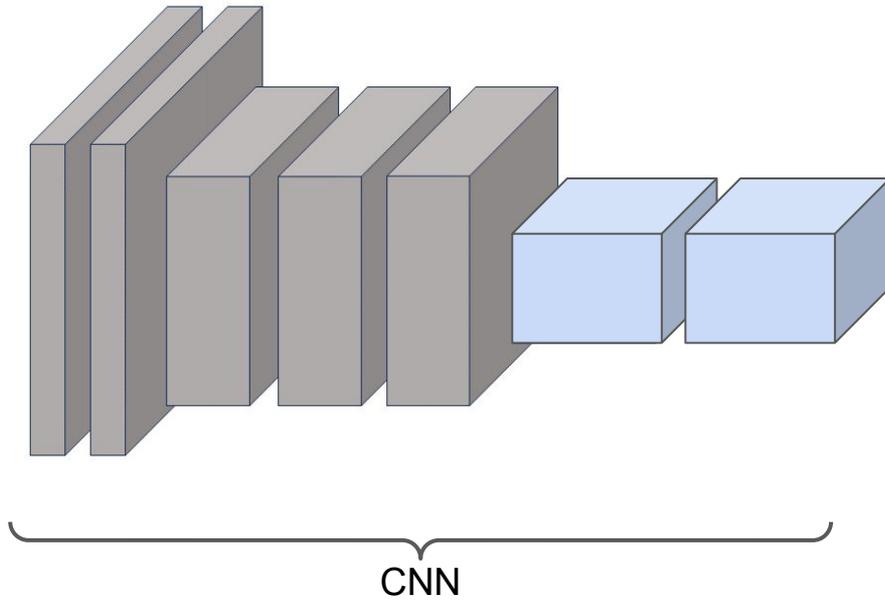
# Deeper CNN Architectures



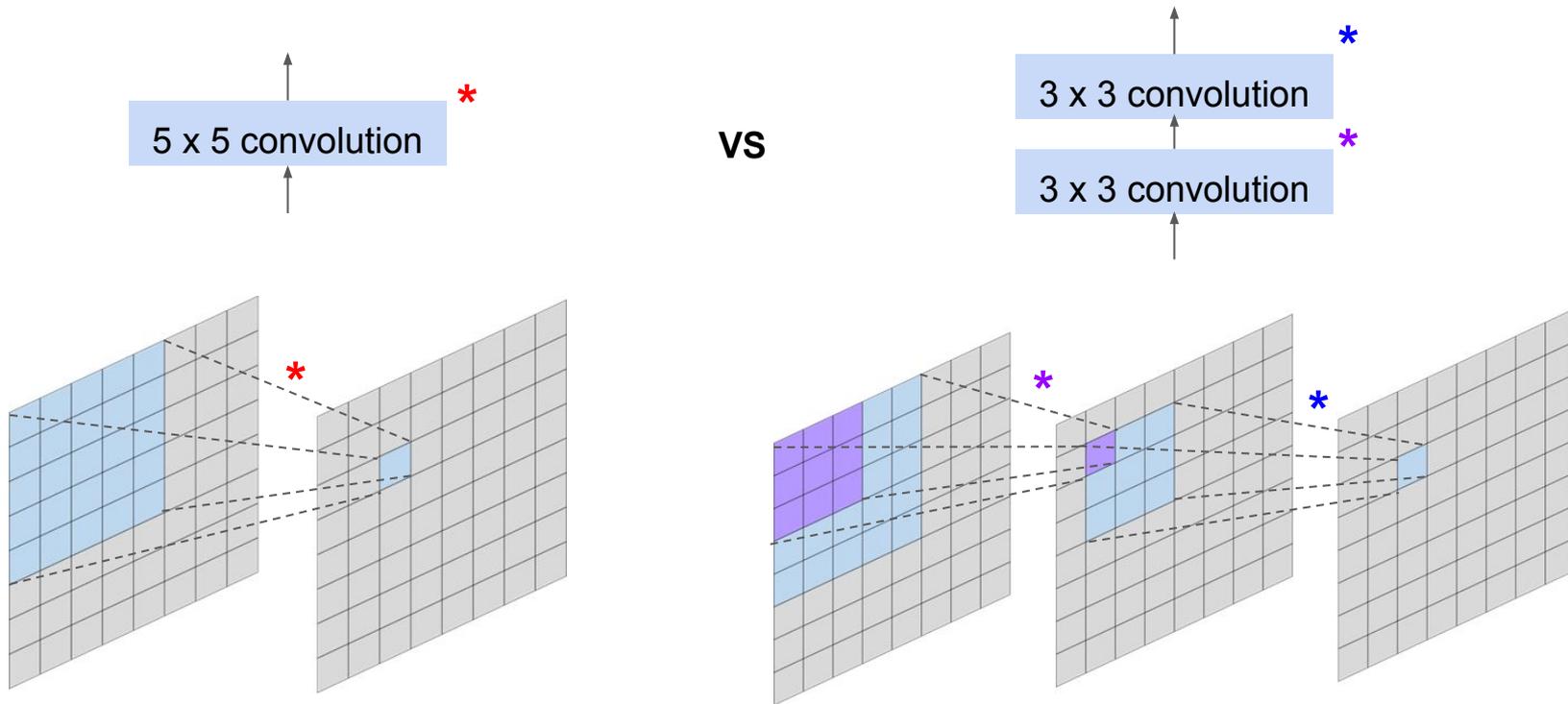
# Deeper CNN Architectures



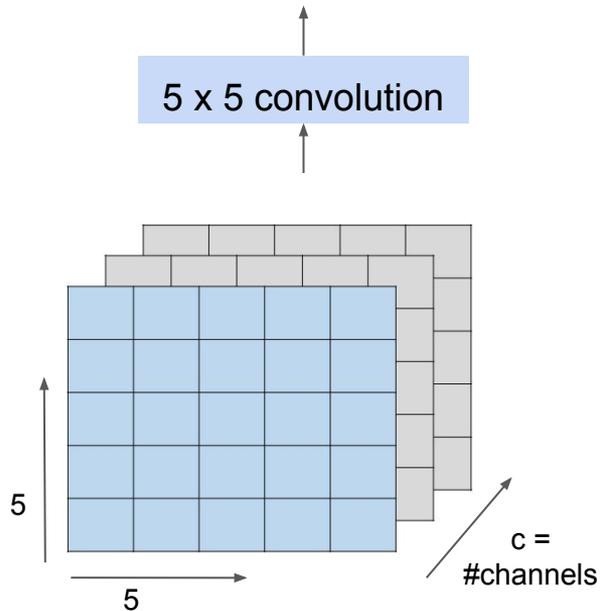
# Deeper CNN Architectures



# Deeper CNN Architectures

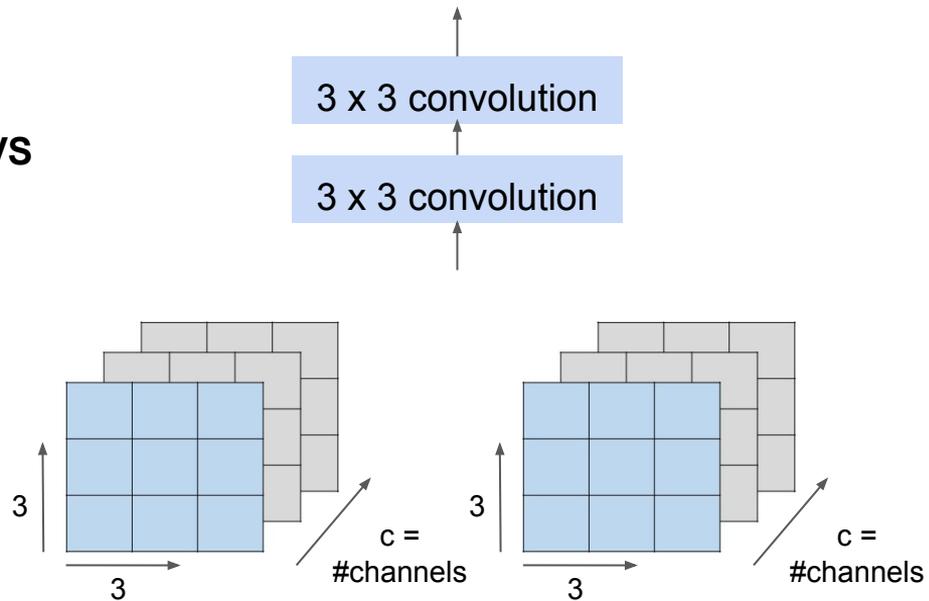


# Deeper CNN Architectures



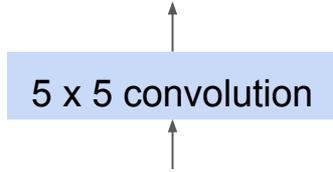
$$5 * 5 * c^2 = 25c^2 \text{ parameters}$$

VS

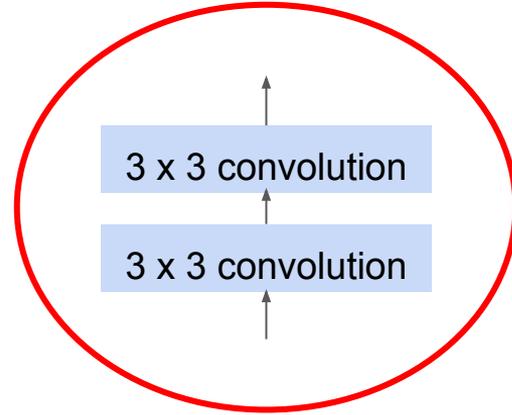


$$2 * 3 * 3 * c^2 = 18c^2 \text{ parameters}$$

# Deeper CNN Architectures

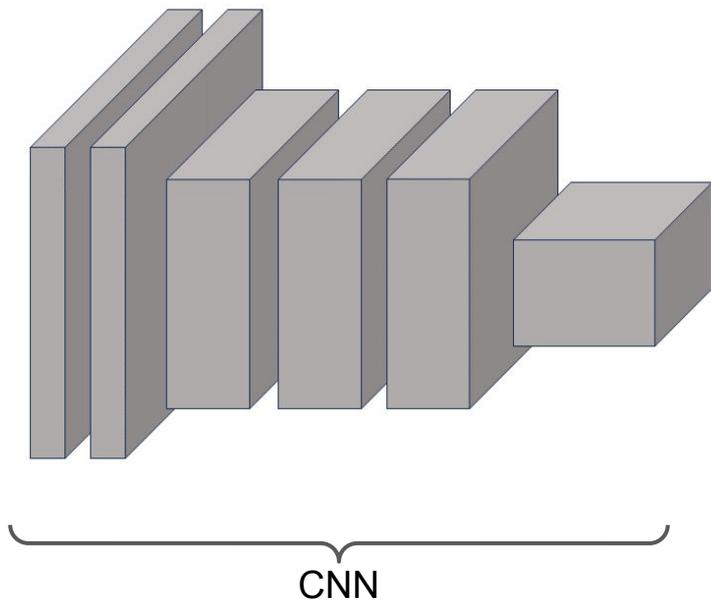


VS

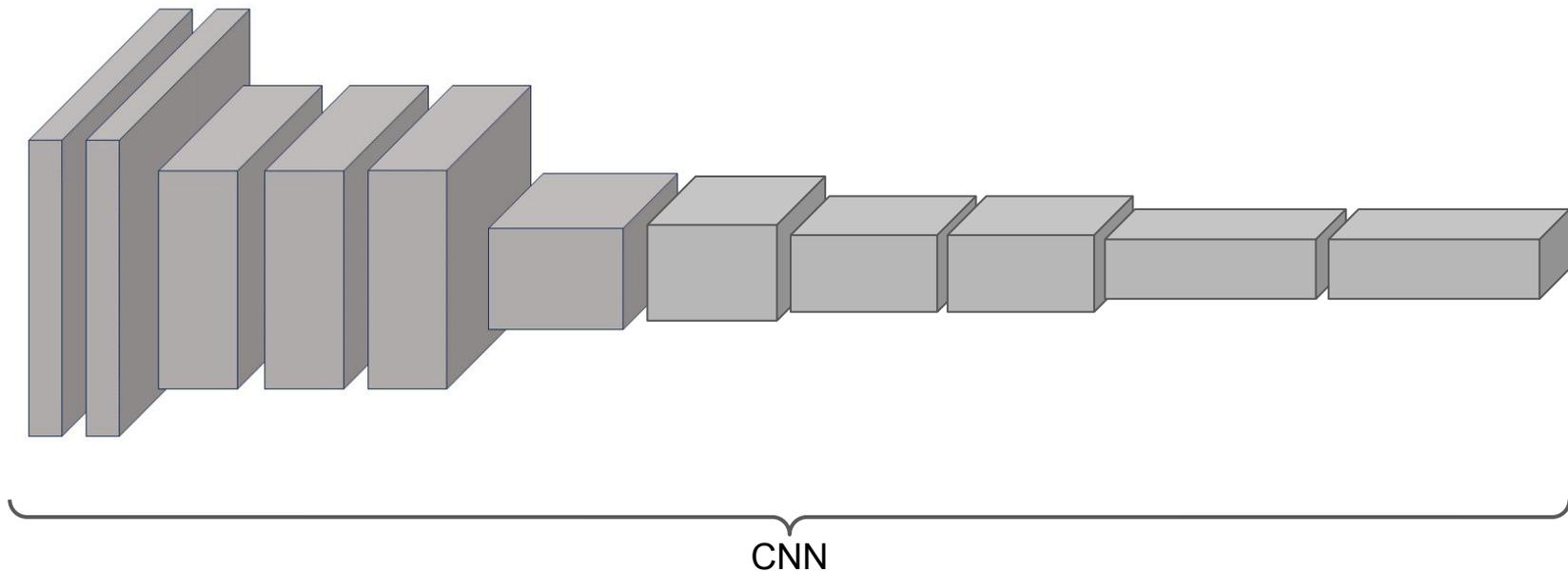


**Performed better!**

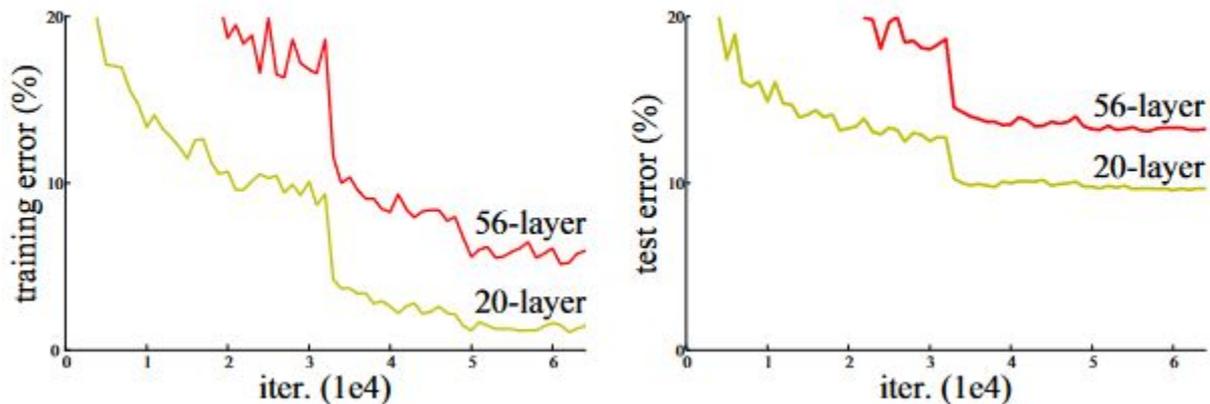
Deeper == better



Deeper == better



# Deeper == better?



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

# Cornell Bowers C-IS

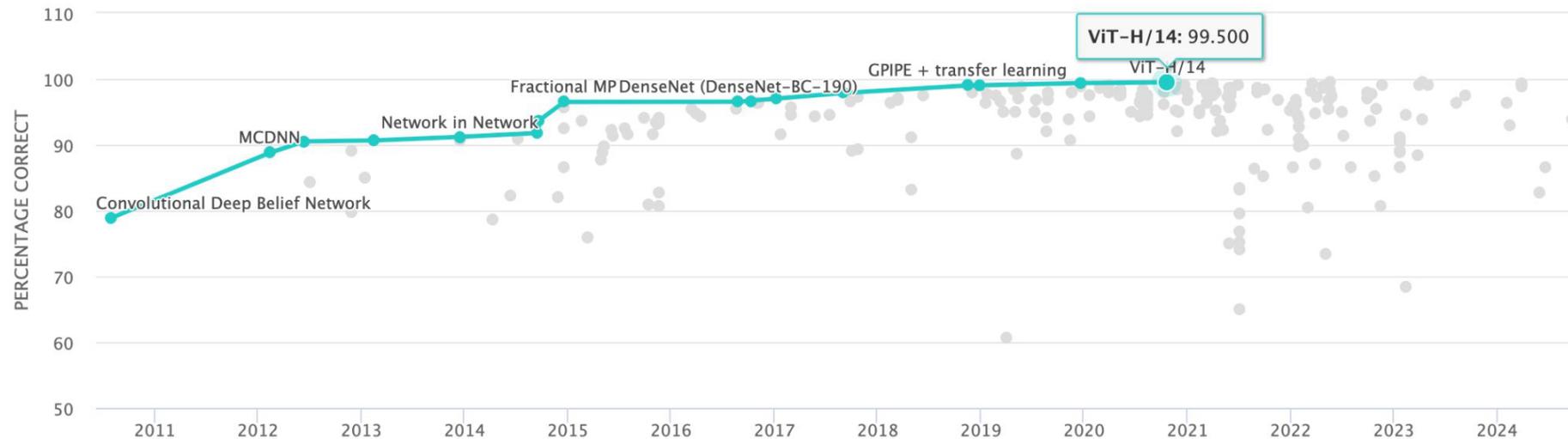
airplane



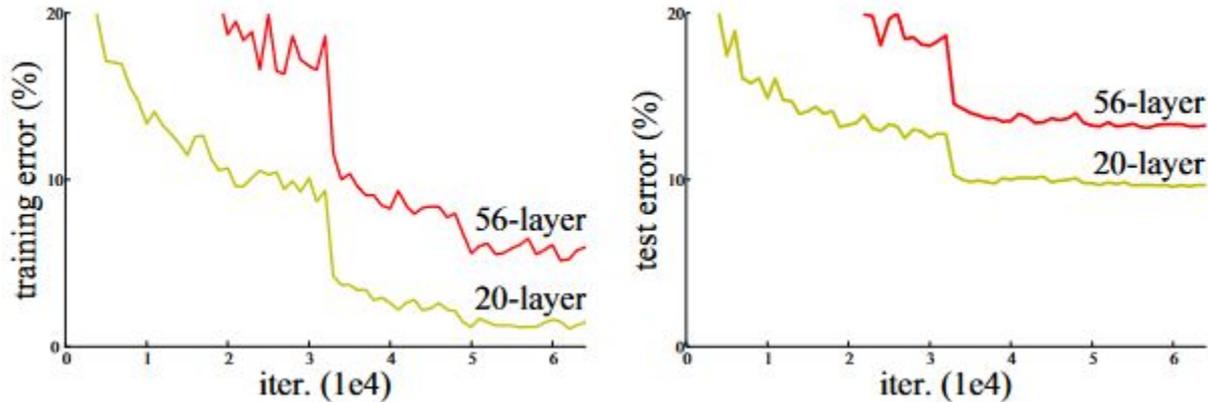
automobile



bird



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

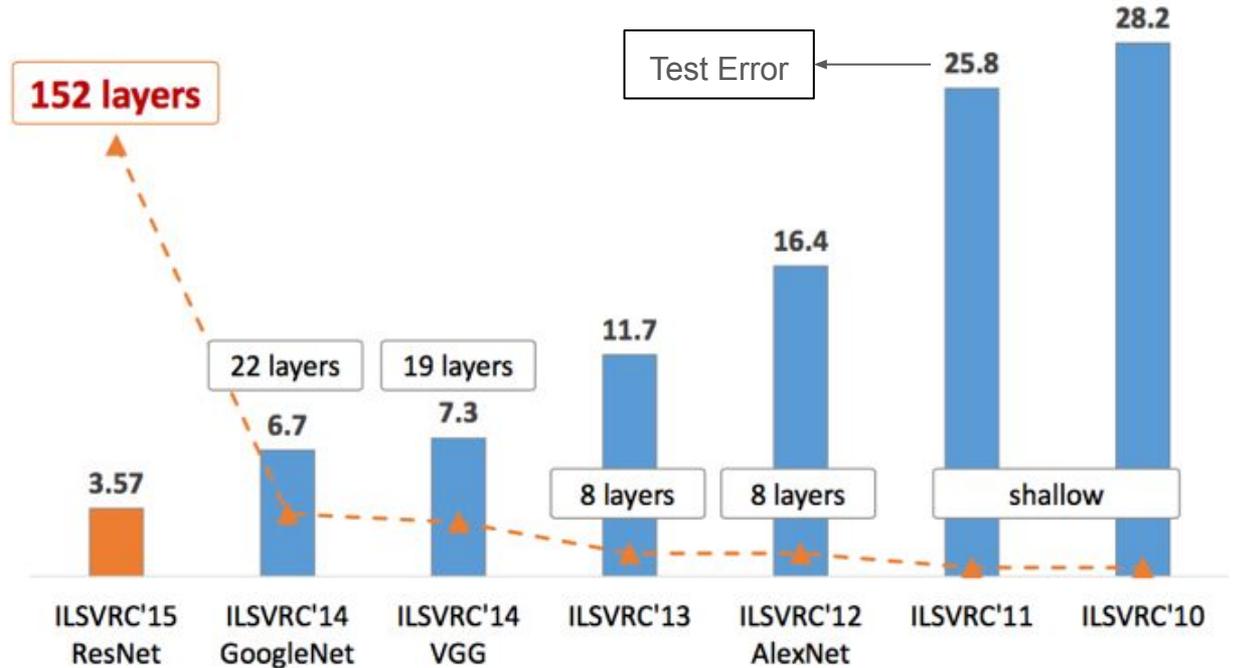
# Deeper != better

- Long training times
- Vanishing gradient problem
  - Recall backpropagation to update weights

$$\frac{\partial z}{\partial z_i} = \frac{\partial z}{\partial z_{m-1}} \frac{\partial z_{m-1}}{\partial z_{m-2}} \dots \frac{\partial z_{i+1}}{\partial z_i}$$

- If each term  $\lll 1$ , gradient “vanishes” as the entire multiplication goes towards 0
- => Weights not updated properly

# ImageNet Classification Challenge: Deeper == better



# GoogLeNet/Inception Net

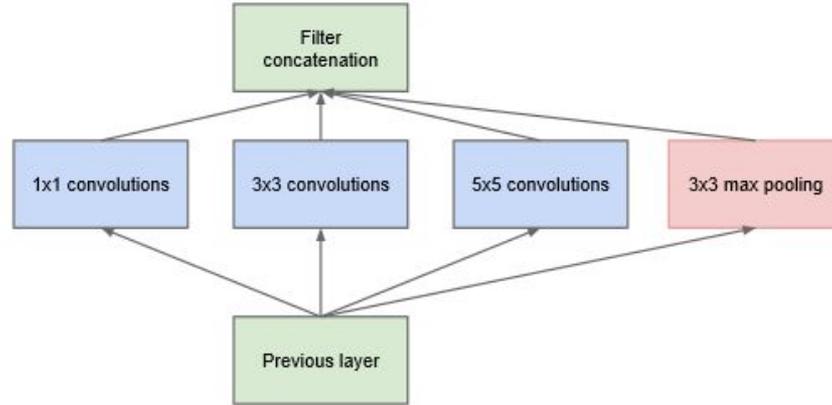
Goal: given a fixed computational budget  
network

=> Deeper networks with computational



In this paper, we will focus on an efficient deep neural network architecture for computer vision, codenamed Inception, which derives its name from the Network in network paper by Lin et al [12] in conjunction with the famous “we need to go deeper” internet meme [1]. In our case, the word

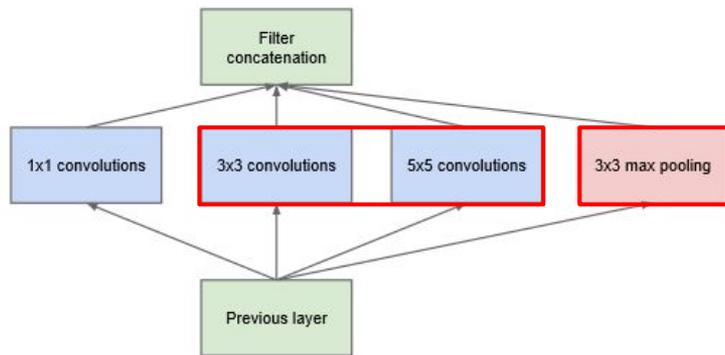
# Inception Module



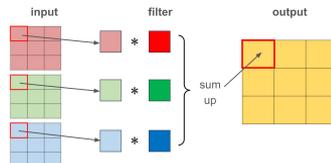
Inception module = main  
building blocks

# Inception Module

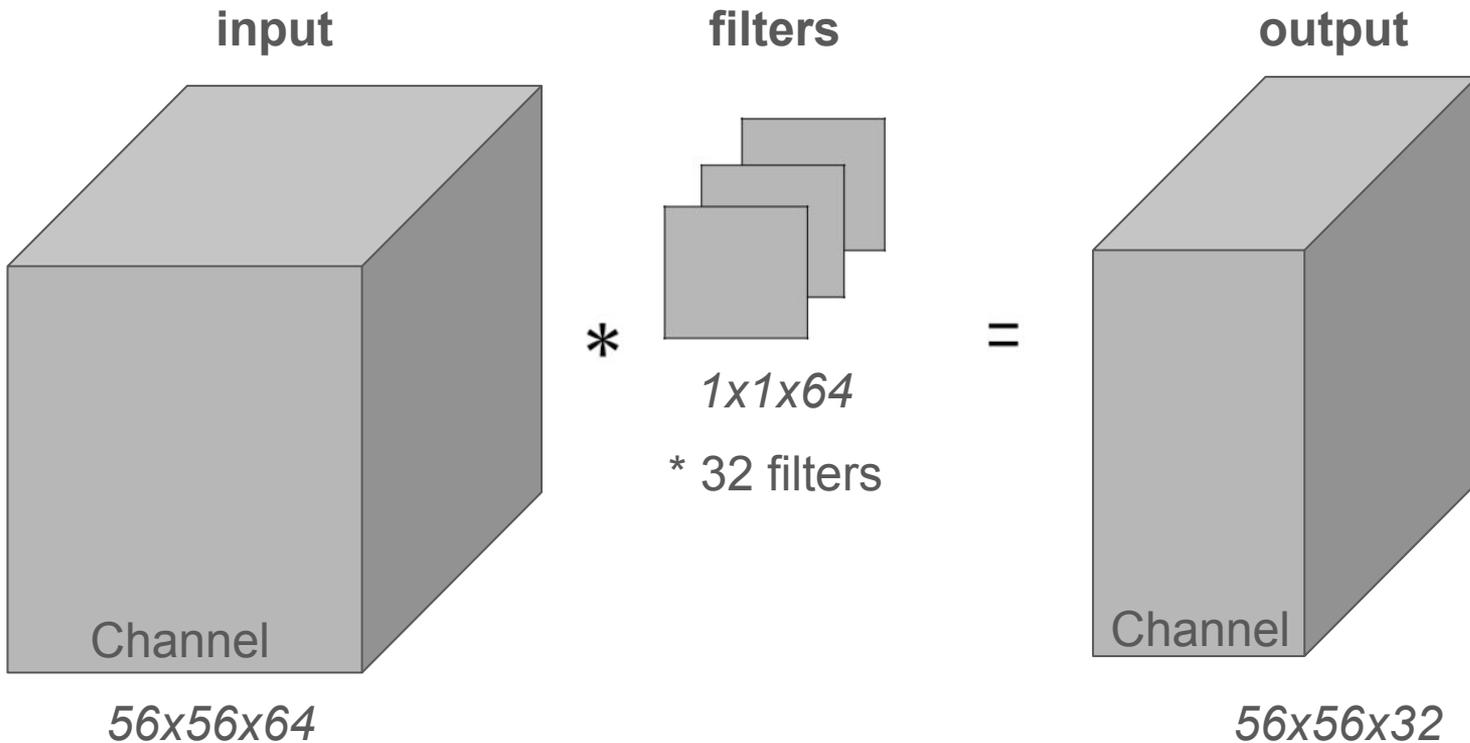
Still expensive!



- 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

## 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

$$3*3*256*256 = \sim 590k \text{ parameters}$$

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

## 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

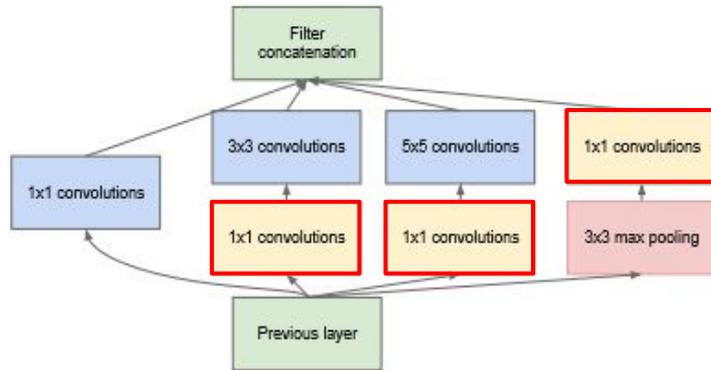
$$3*3*256*256 = \sim 590k \text{ parameters}$$

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

$$1*1*256*64 + 3*3*64*64 + 1*1*64*256 = \sim 70k \text{ parameters}$$

# Inception Module

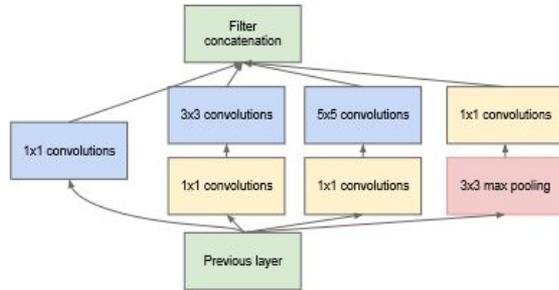
Solution: Inception module with dimension reduction



- “Bottleneck” with 1x1 convolutions to reduce dimensions

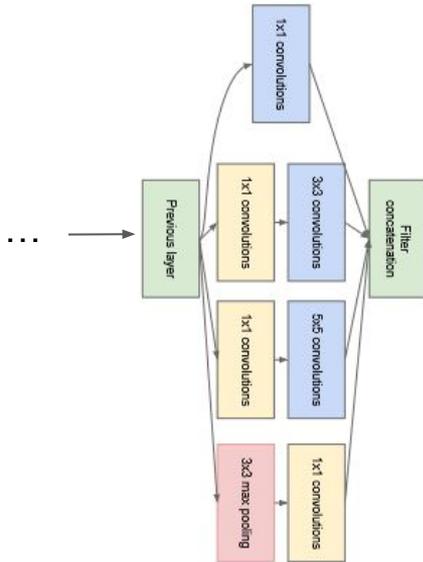
# GoogLeNet Architecture

Key idea: stack inception modules together



# GoogLeNet Architecture

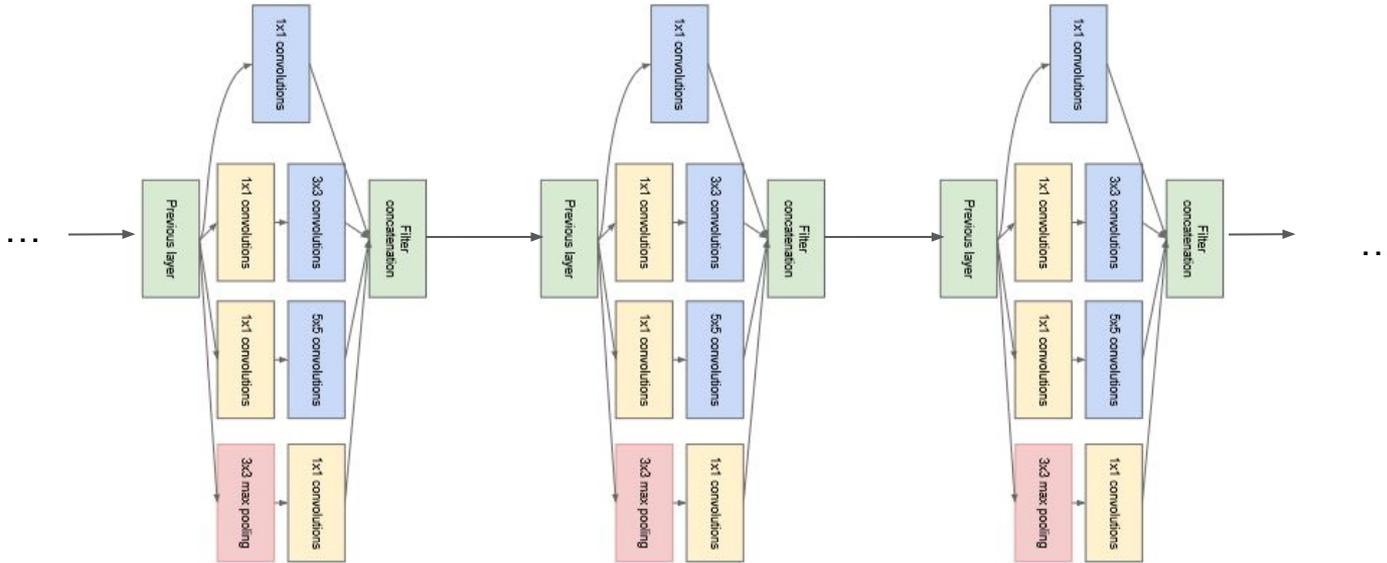
Key idea: stack inception modules together



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

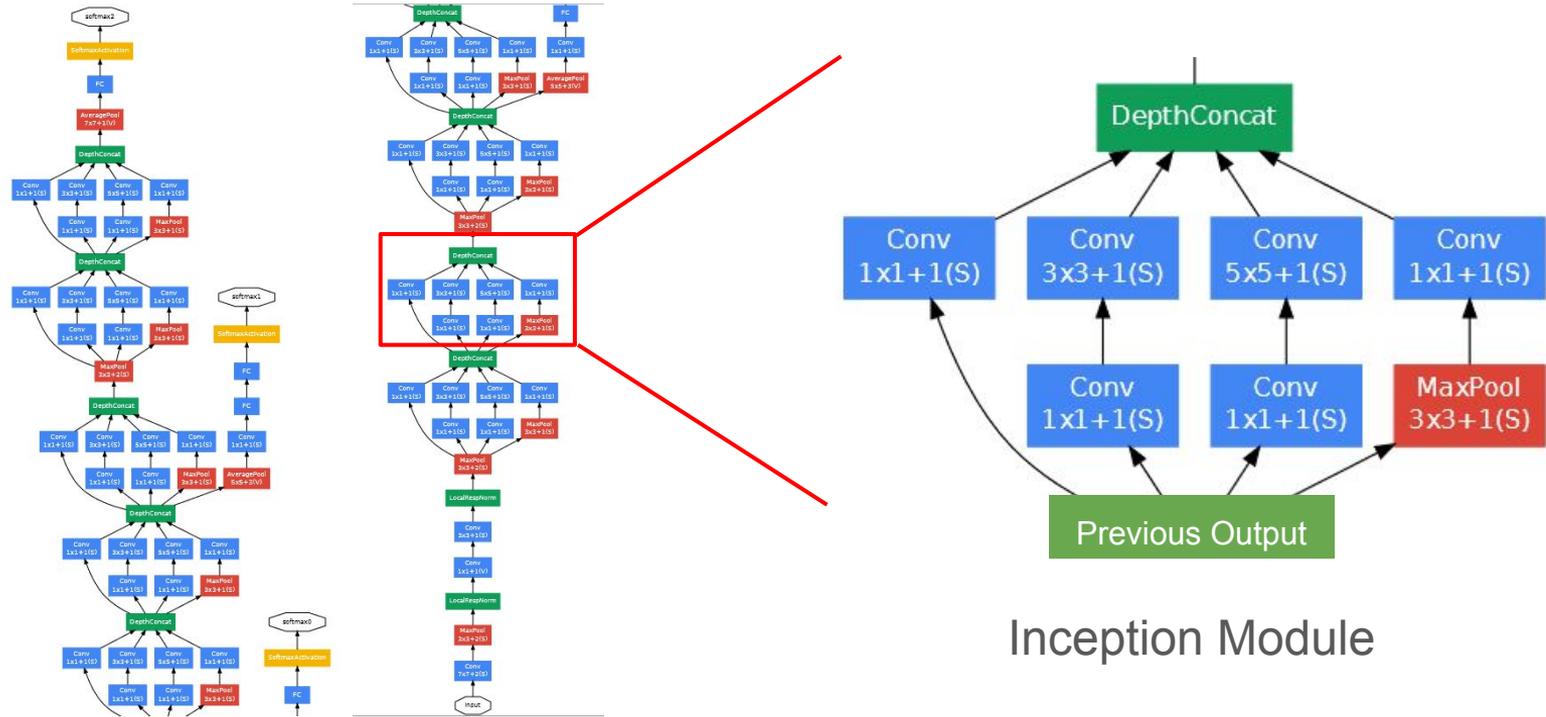
# GoogLeNet Architecture

Key idea: stack inception modules together



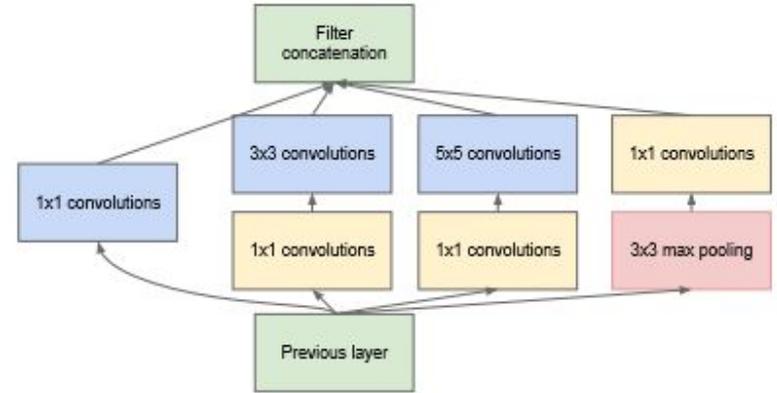
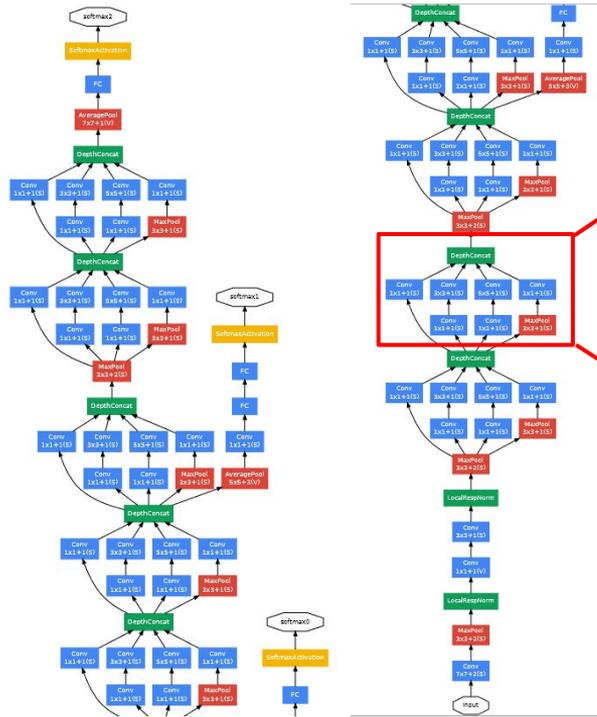
[Szegedy, Christian, et al. "Going deeper with convolutions." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015.]

# The Entire GoogLeNet Architecture



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

# The Entire GoogLeNet Architecture



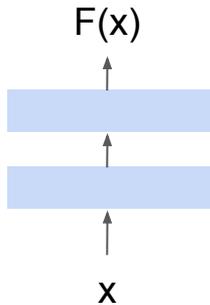
Inception Module

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

# CNN Architectures

## “Plain” CNN

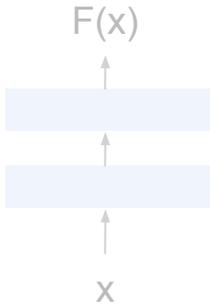
Simple connection  
from previous to next  
layer



# CNN Architectures

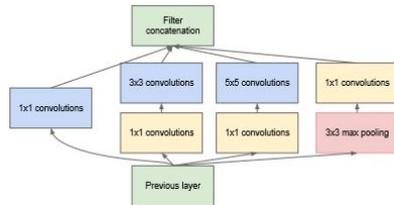
## “Plain” CNN

Simple connection  
from previous to next  
layer

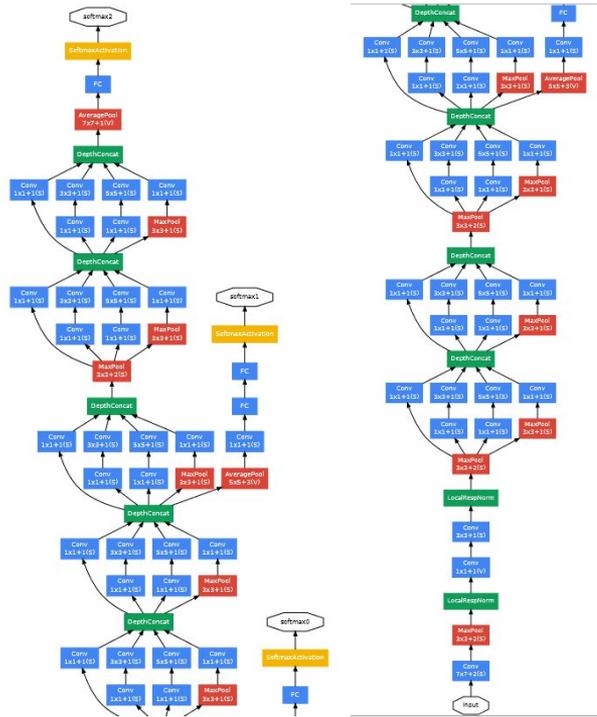


## GoogLeNet

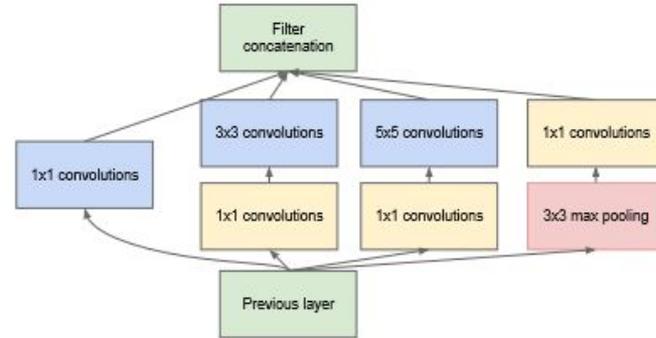
1x1, 3x3, 5x5  
convolutions and  
pooling between each  
layer



# The Entire GoogleNet Architecture

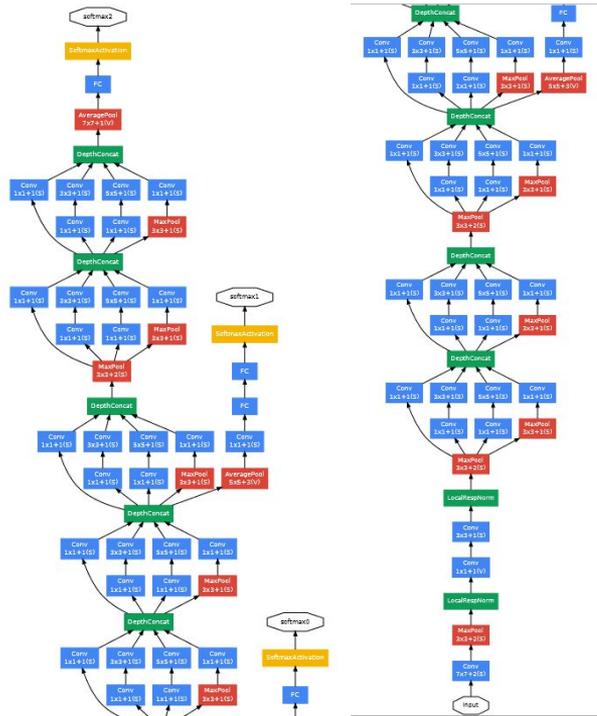


Very complicated - how exactly did this architecture solve the problem?

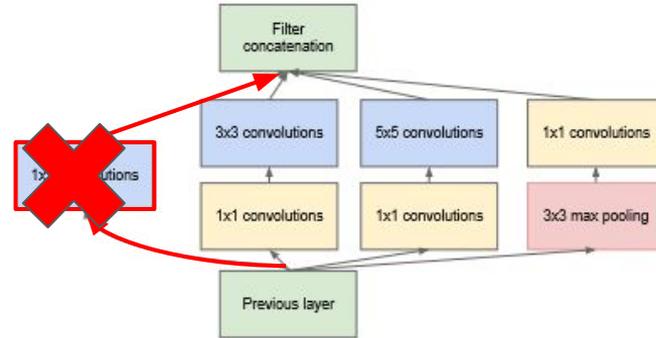


[Szegedy, Christian, et al. "Going deeper with convolutions." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015.]

# The Entire GoogleNet Architecture

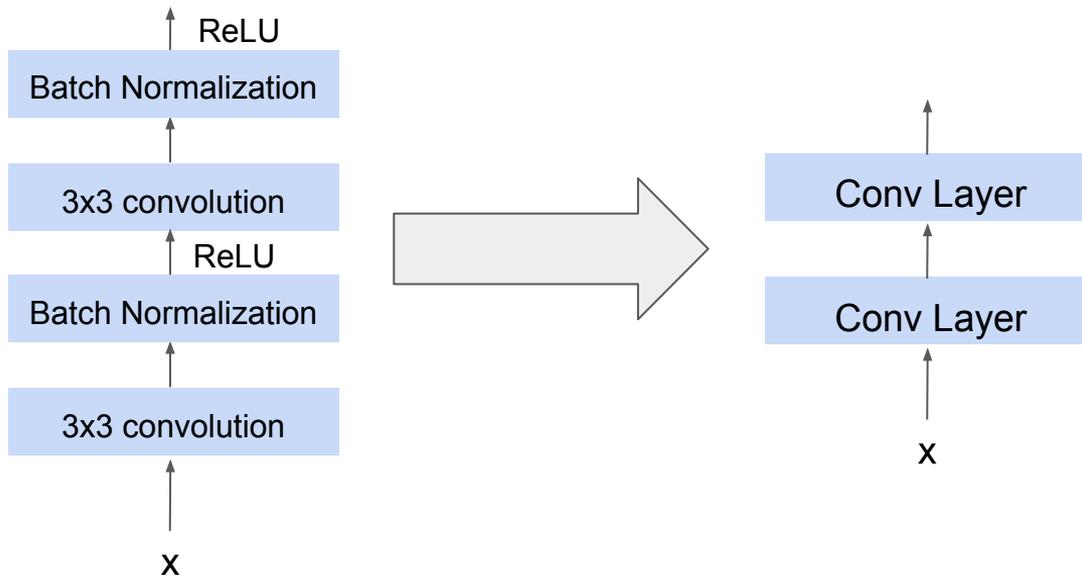


Very complicated - how exactly did this architecture solve the problem?



Residual connections: connect layers directly

# Aside: Conv Layer Abstraction

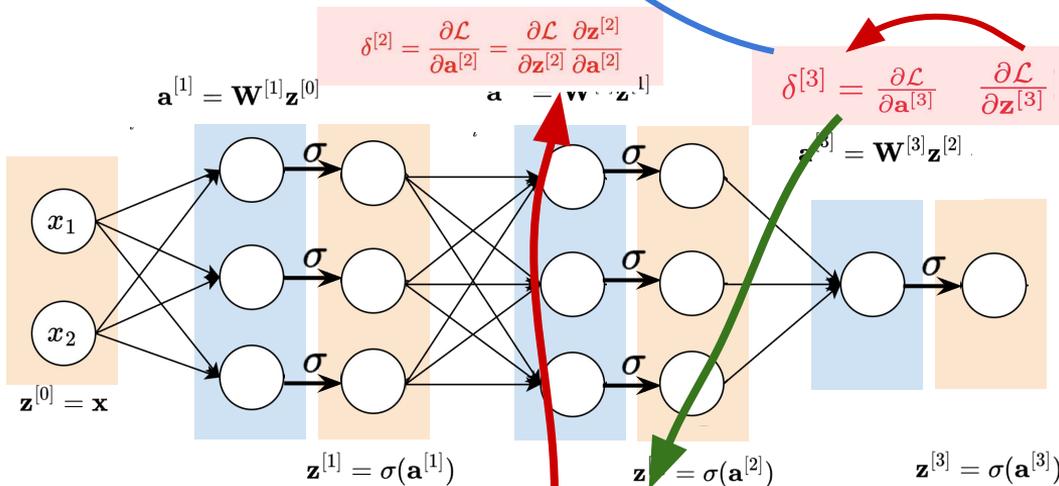


# Backpropagation

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

## 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:**  $\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[1:L]}}, \frac{\partial \mathcal{L}}{\partial \mathbf{b}^{[1:L]}}$

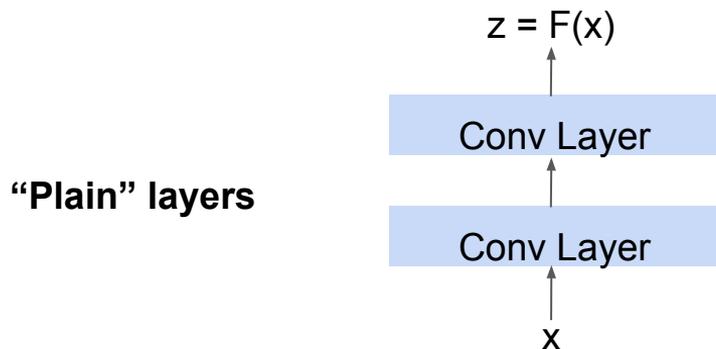


$$\mathcal{L}(\mathbf{z}^{[3]}, \mathbf{y})$$

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

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

# Backpropagation through Residual blocks

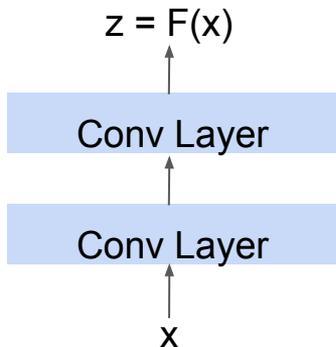


$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial x} = \frac{\partial L}{\partial z} F'(x)$$



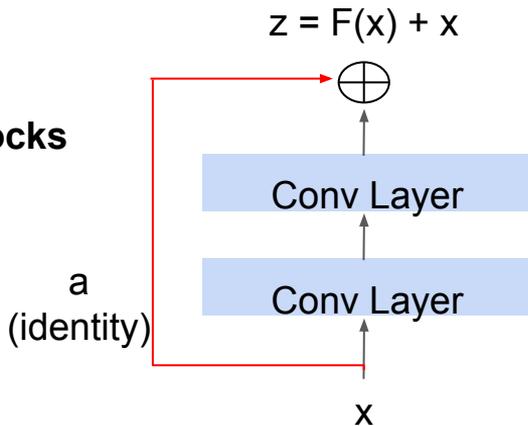
# Backpropagation through Residual blocks

“Plain” layers



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

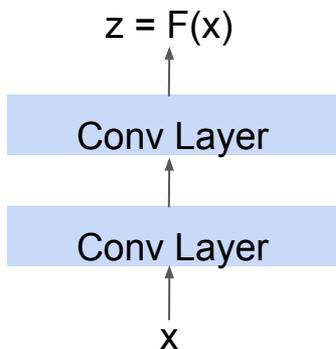
Residual Blocks



$$\frac{\partial L}{\partial x} = \boxed{\phantom{\frac{\partial L}{\partial z} \frac{\partial z}{\partial x}}}$$

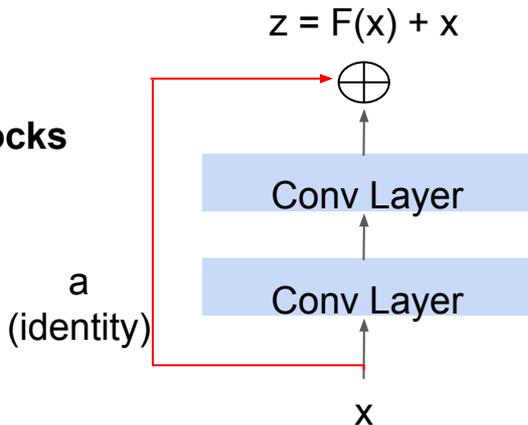
# Backpropagation through Residual blocks

“Plain” layers



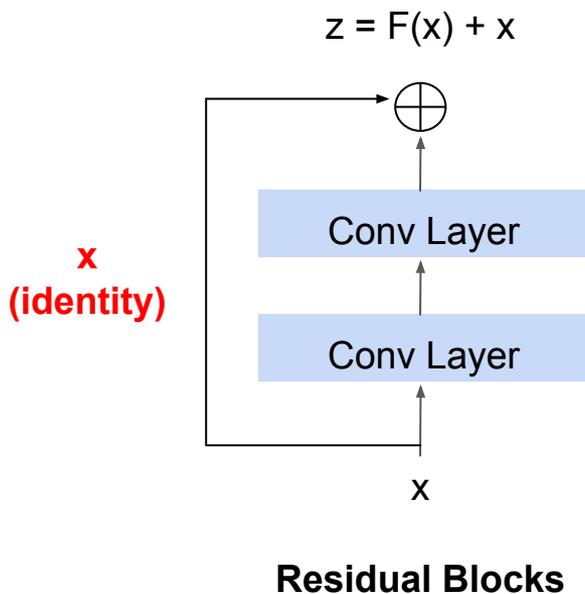
$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial x} = \frac{\partial L}{\partial z} F'(x)$$

Residual Blocks



$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial x} = \frac{\partial L}{\partial z} (1 + F'(x))$$

# Residual Connections



## Identity mapping

- can propagate features forward
- only learn *difference* of feature maps

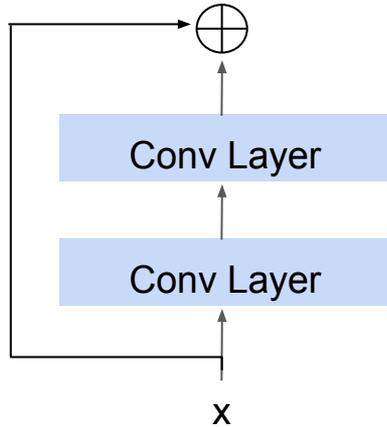
## Additive component of identity

- alleviates vanishing gradients

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial x} = \frac{\partial L}{\partial z} (1 + F'(x))$$

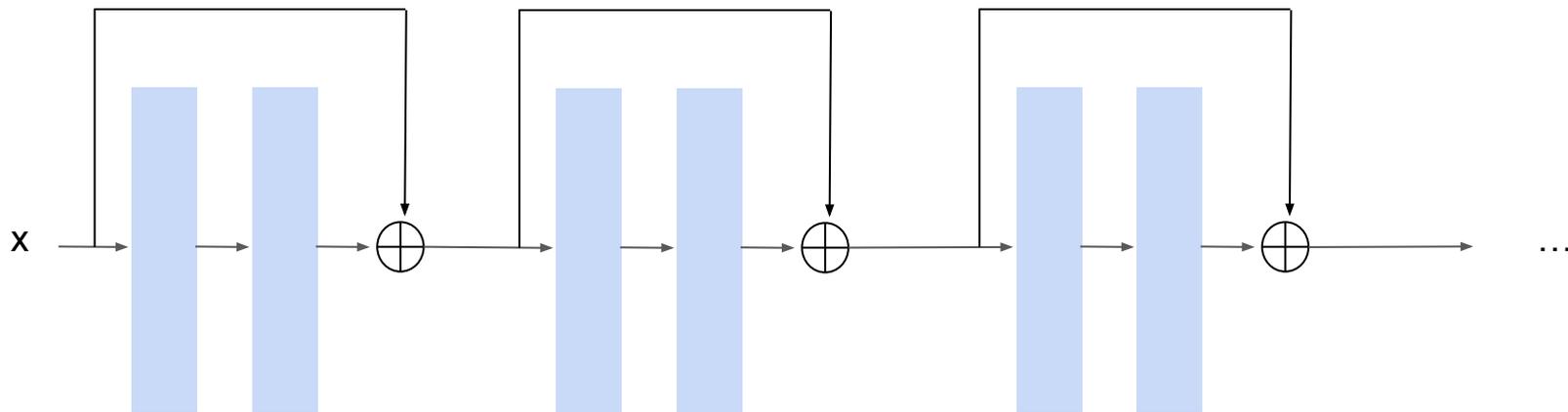
# ResNet

Stack residual blocks together!



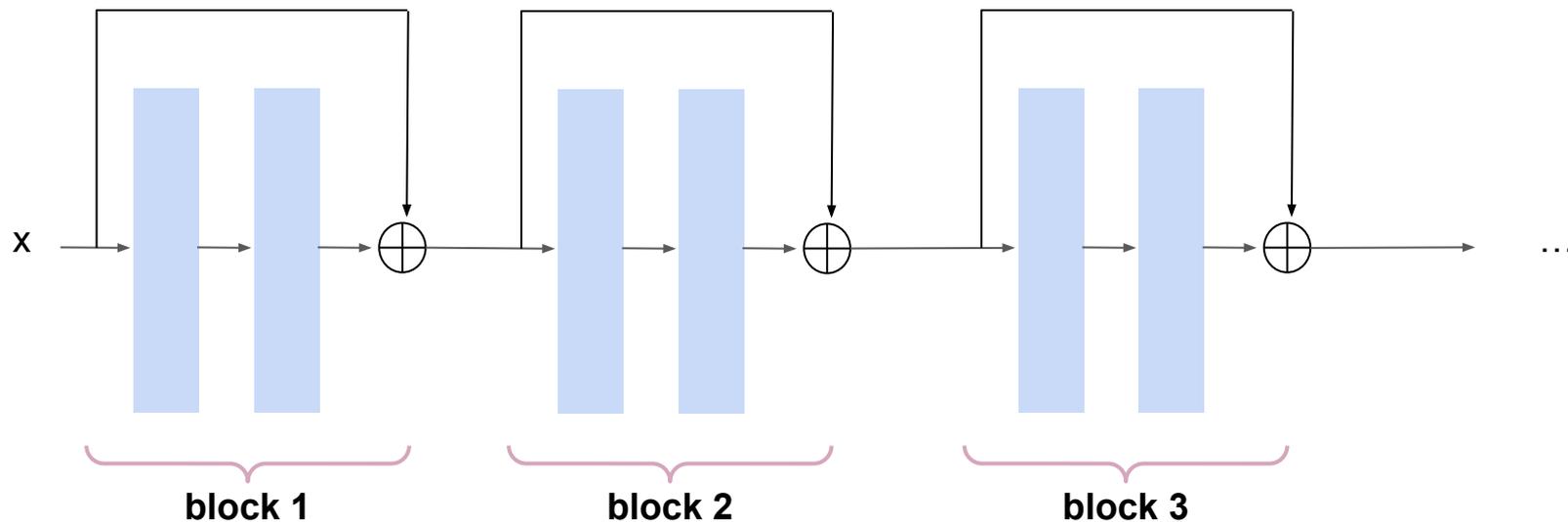
# ResNet

Stack residual blocks together!



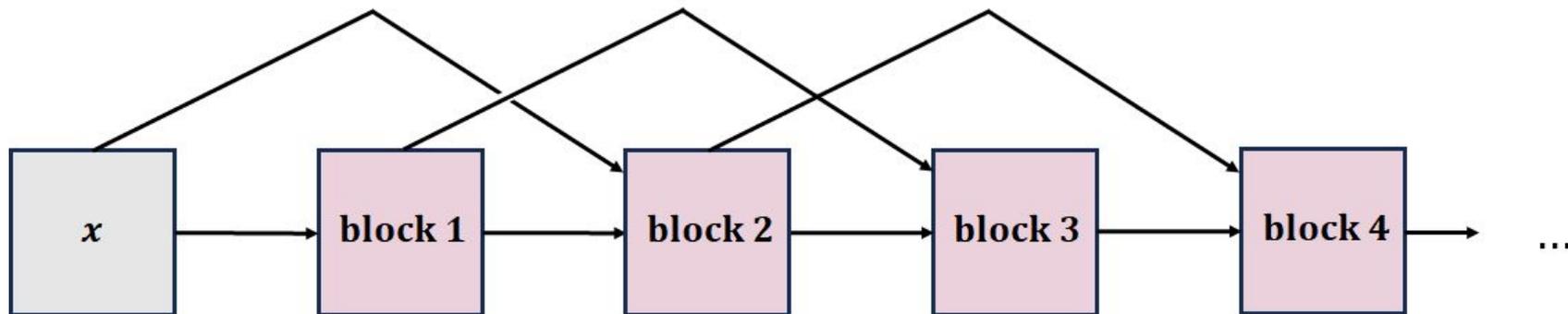
# ResNet

Stack residual blocks together!



# ResNet

Stack residual blocks together!

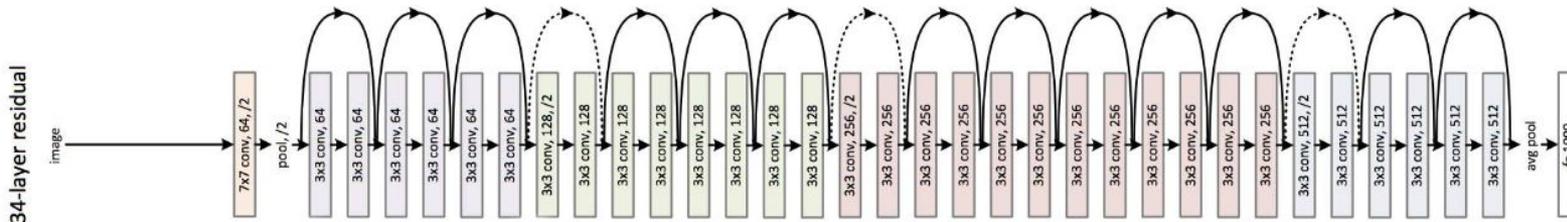


# Full ResNet Architecture

## “Plain” Network

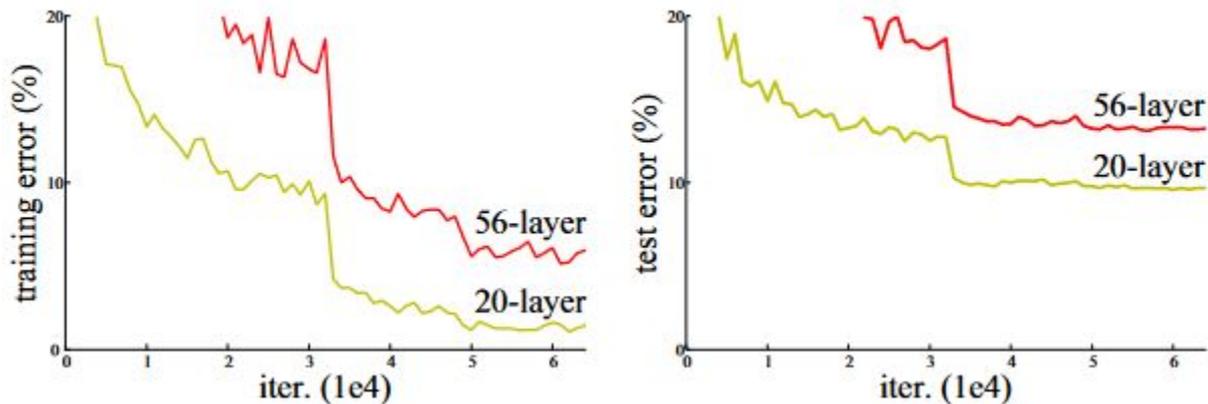


## ResNet



[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!

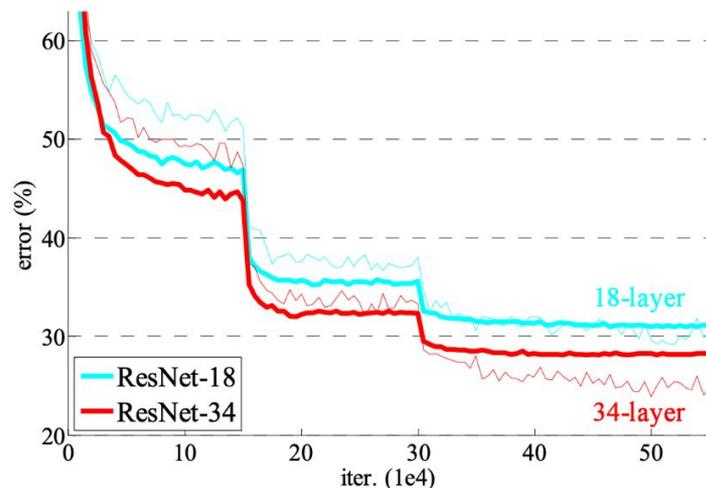
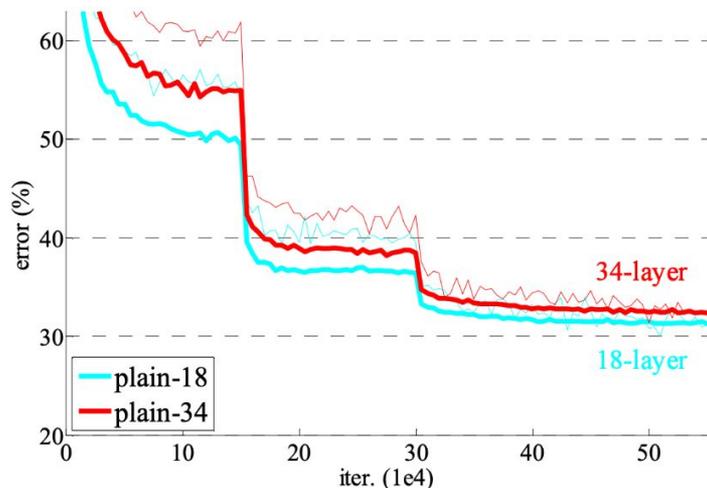


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

# Visualizing the Effect of Skip Connections

Makes optimization easier!

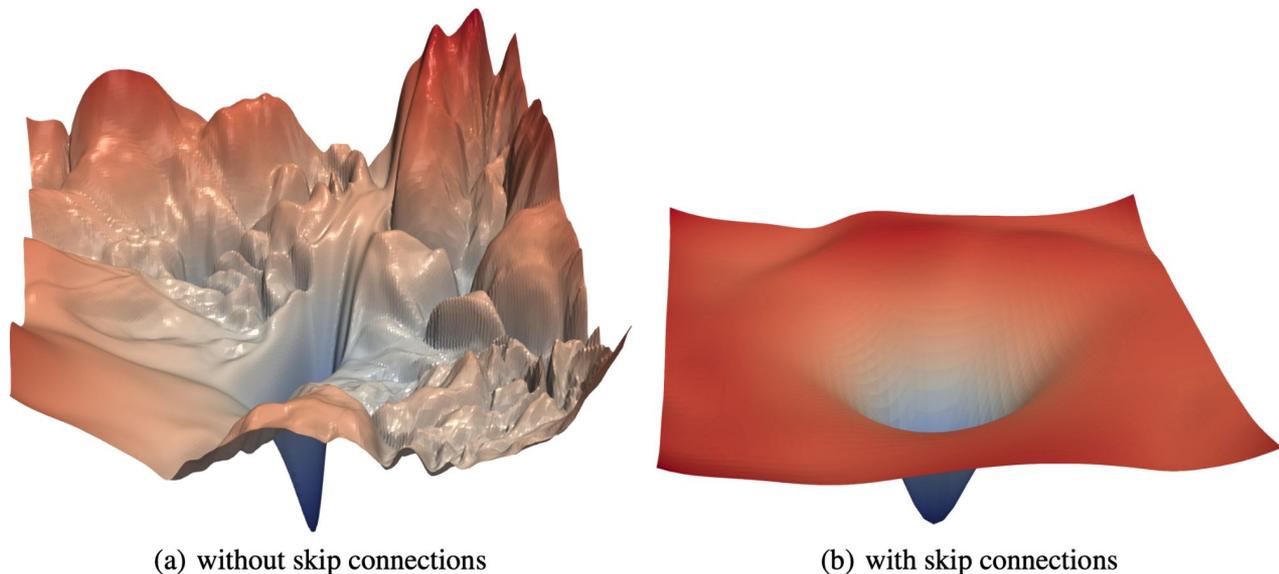


Figure 1: The loss surfaces of ResNet-56 with/without skip connections. The proposed filter normalization scheme is used to enable comparisons of sharpness/flatness between the two figures.

# Stochastic Depth

Still have long training times! Solution: stochastic depth

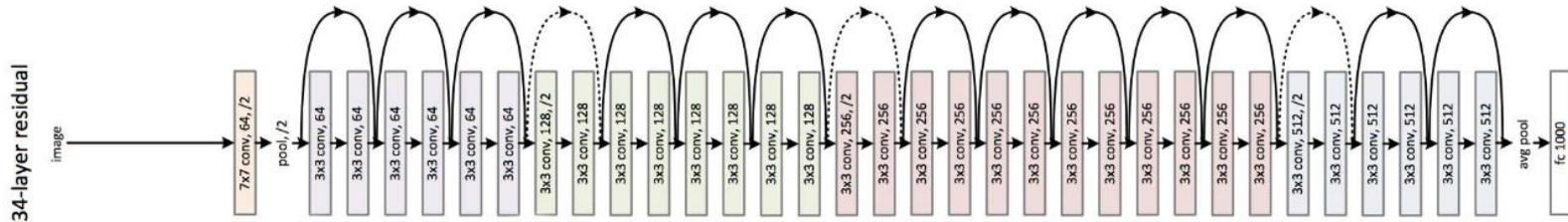
# Stochastic Depth

During training, randomly drop Residual Blocks using skip connections

Like dropout but with residual blocks instead of individual neurons

Another benefit: robustness/mitigating overfitting

Drop probability for layer  $l$  (out of  $L$ ): 
$$p_l = 1 - \frac{l}{L}(1 - p_L)$$



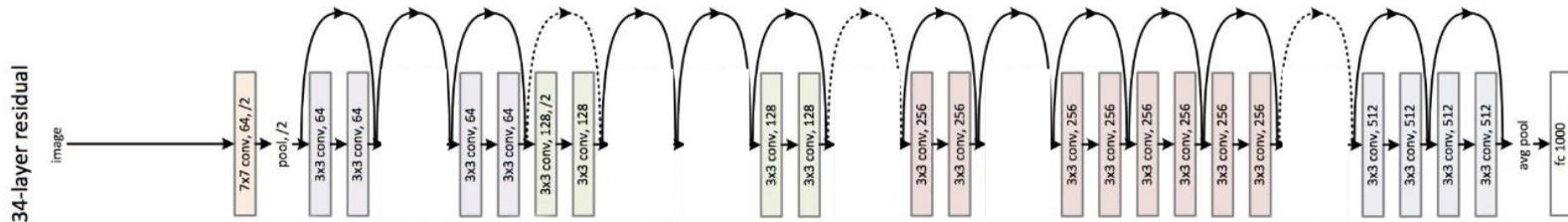
# Stochastic Depth

During training, randomly drop Residual Blocks using skip connections

Like dropout but with residual blocks instead of individual neurons

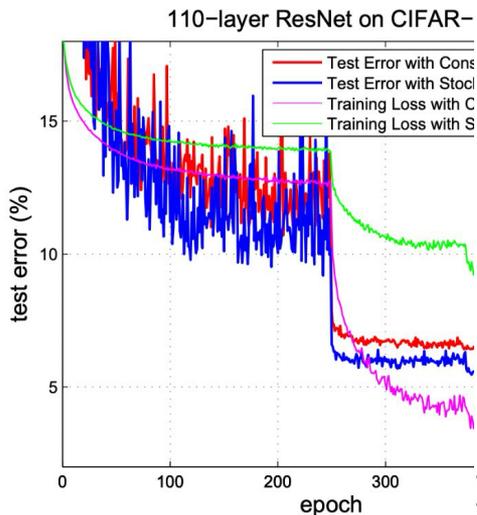
Another benefit: robustness/mitigating overfitting

Drop probability for layer  $l$  (out of  $L$ ): 
$$p_l = 1 - \frac{l}{L}(1 - p_L)$$

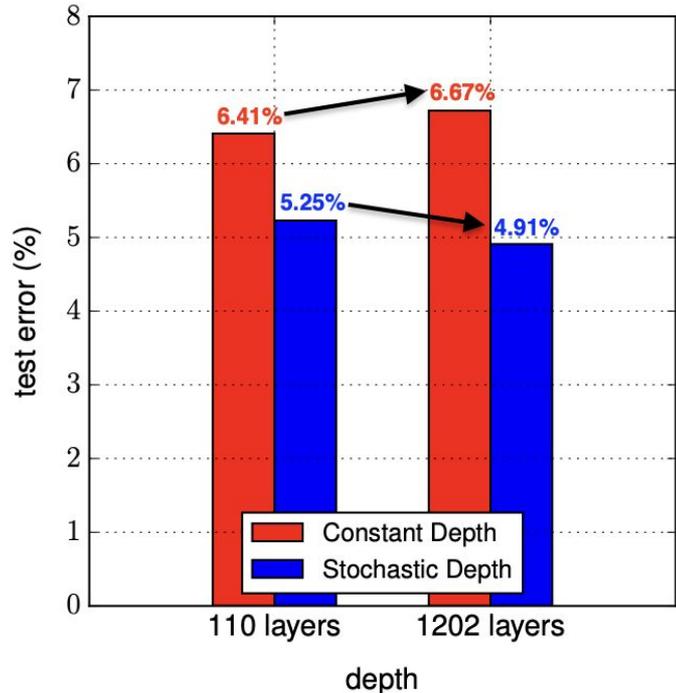


# Stochastic Depth

Increases training loss, but... decreases test error



**Fig. 3.** Test error on CIFAR-10 data augmentation, correspond

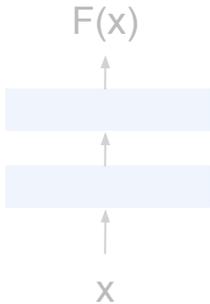


**Fig. 5.** With stochastic depth, the 1202-layer ResNet still significantly improves over the 110-layer one.

# CNN Architectures

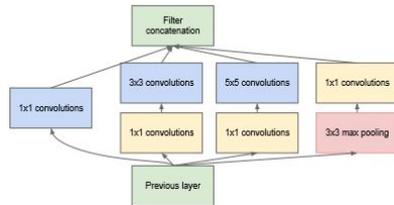
## “Plain” CNN

Simple connection  
from previous to next  
layer



## GoogLeNet

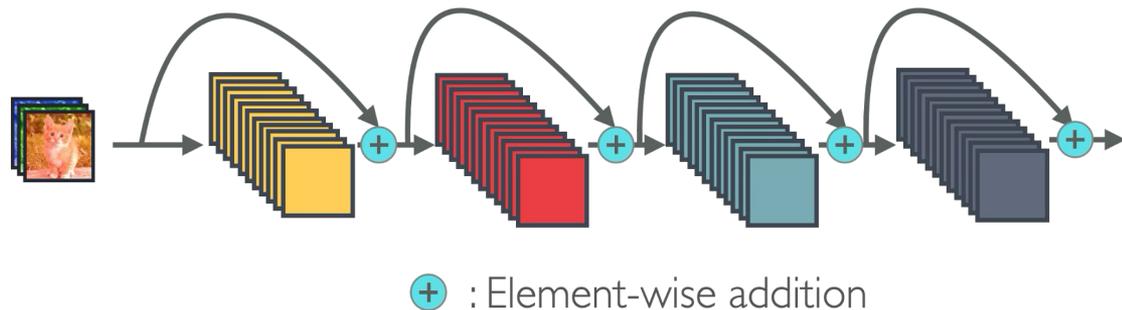
1x1, 3x3, 5x5  
convolutions and  
pooling between each  
layer



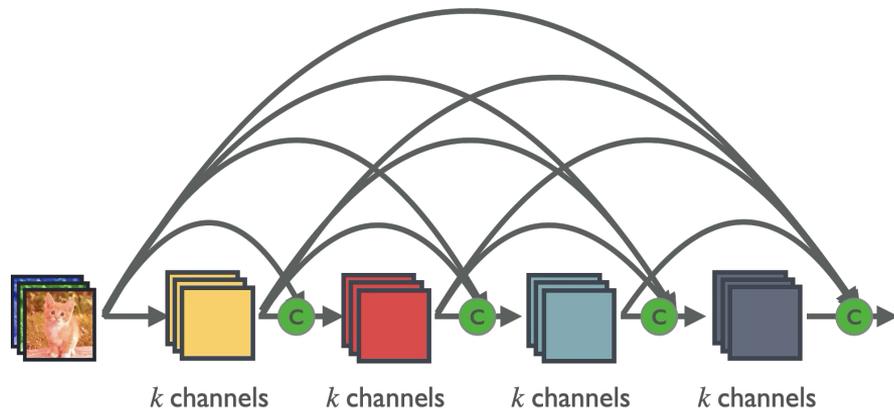


# From ResNets to DenseNets

ResNet

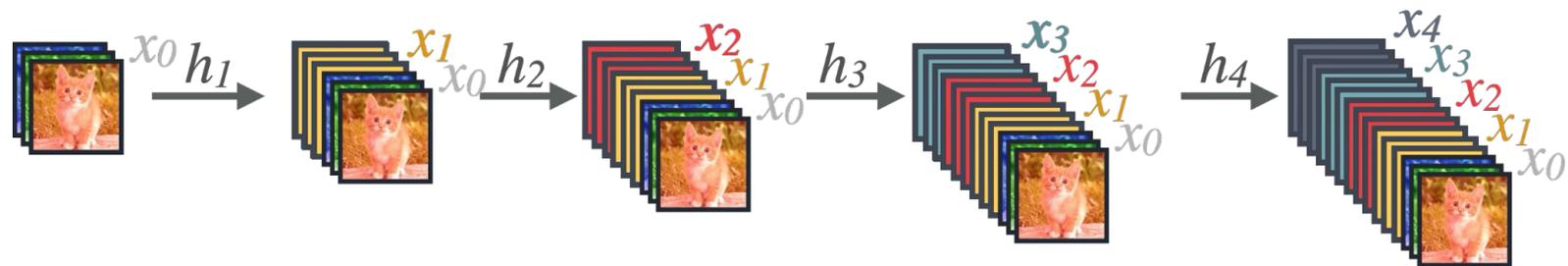


DenseNet



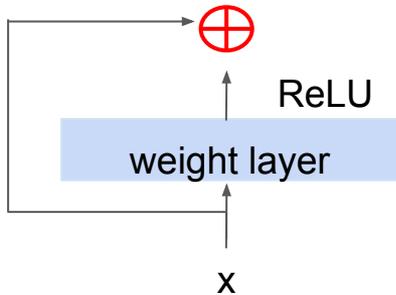
# DenseNets

Feature concatenation

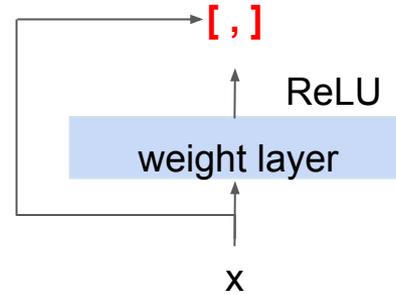


# Dense Blocks

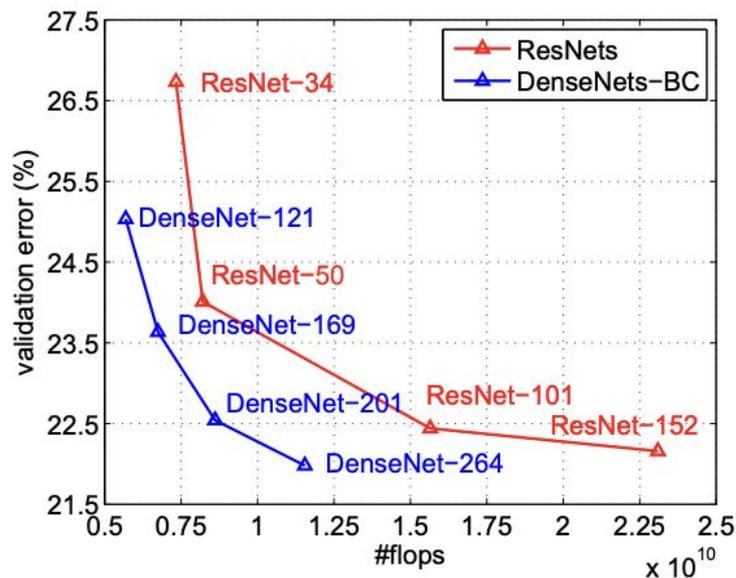
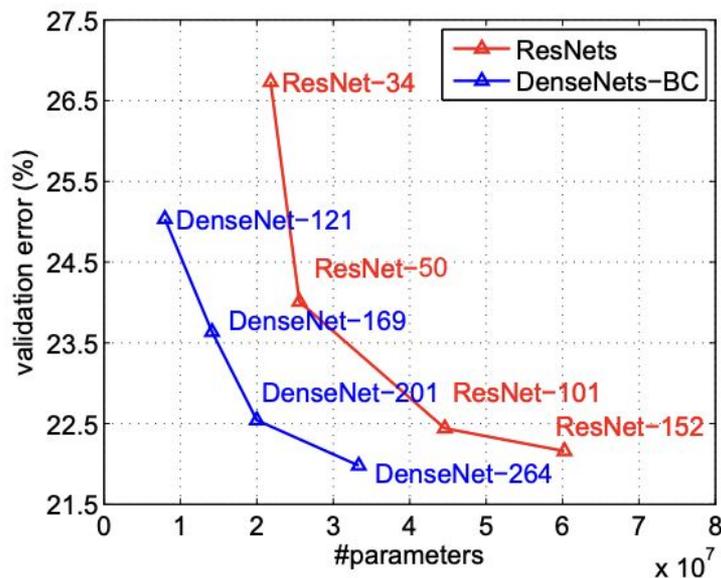
To create dense connections, dense blocks use the same structure as residual blocks, but concatenate (denoted by  $[ , ]$ ) inputs instead of simply adding them



**Residual Blocks**



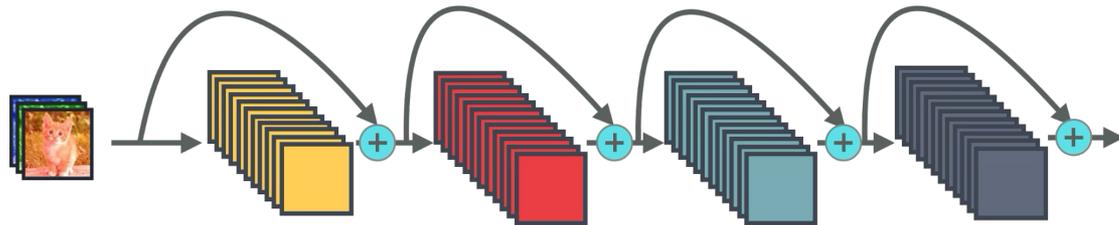
**Dense Blocks**



**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*).

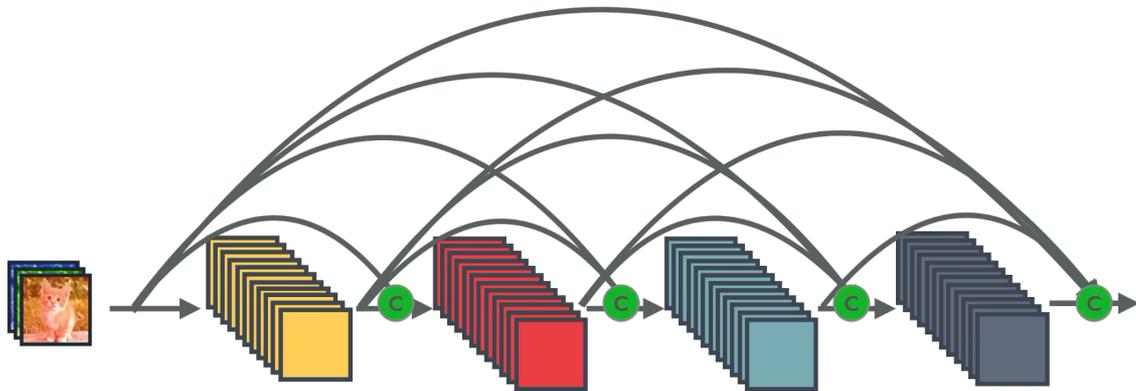
Discussion: What design choices might allow a ~100-layer DenseNet to have fewer parameters than a ~100-layer ResNet?

ResNet



⊕ : Element-wise addition

DenseNet

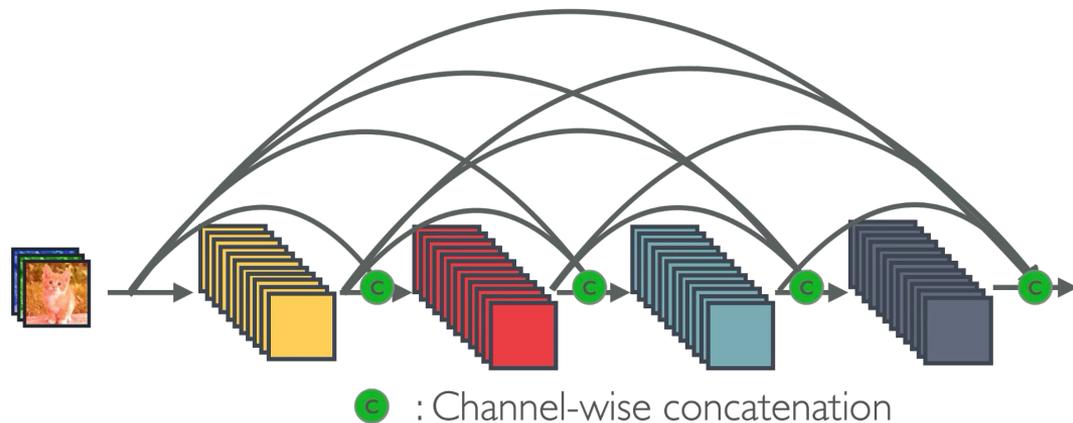


⊕ : Channel-wise concatenation

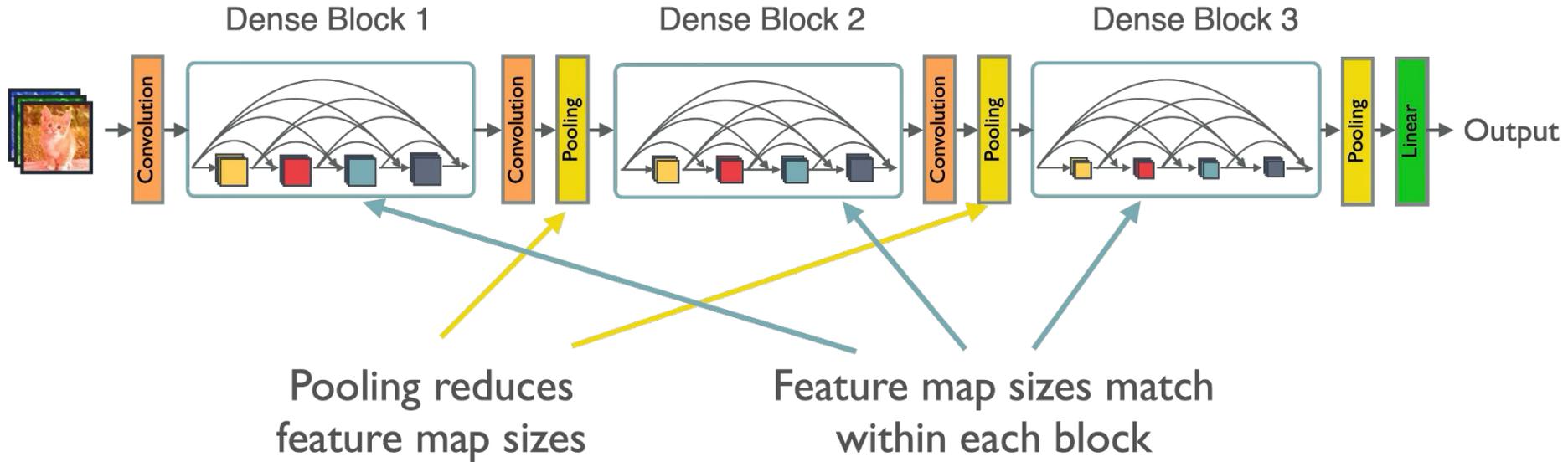
# Dense Connections

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



Pooling reduces feature map sizes

Feature map sizes match within each block

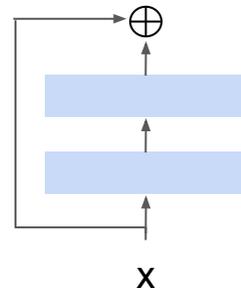
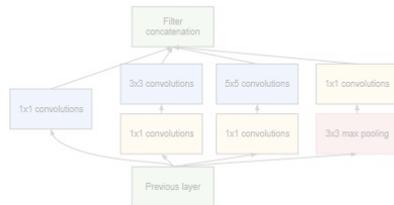
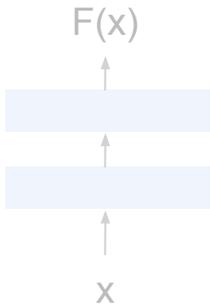
# CNN Architectures

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

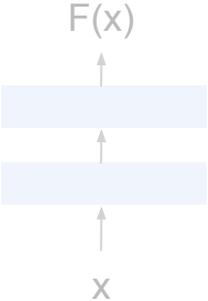
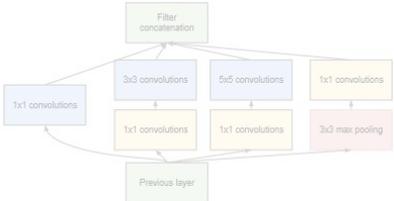
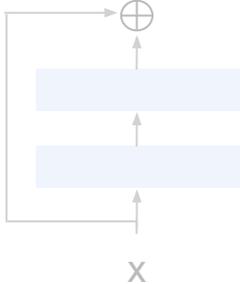
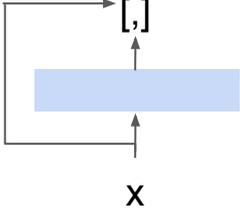
Simple connection from previous to next layer

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

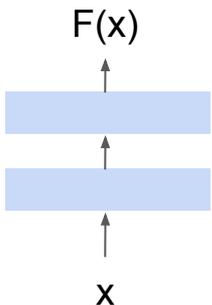
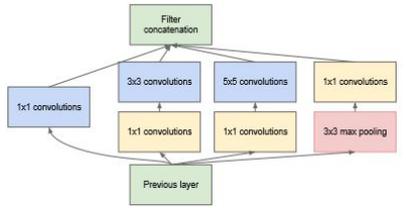
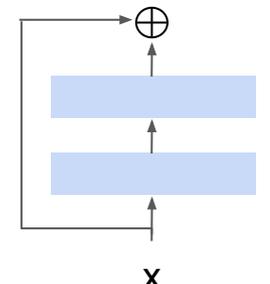
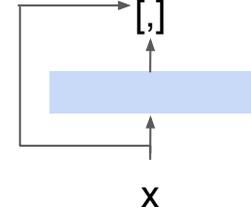
Skip connections  
Add output of previous layer to next layer



# CNN Architectures

"Plain" CNN	GoogLeNet	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 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