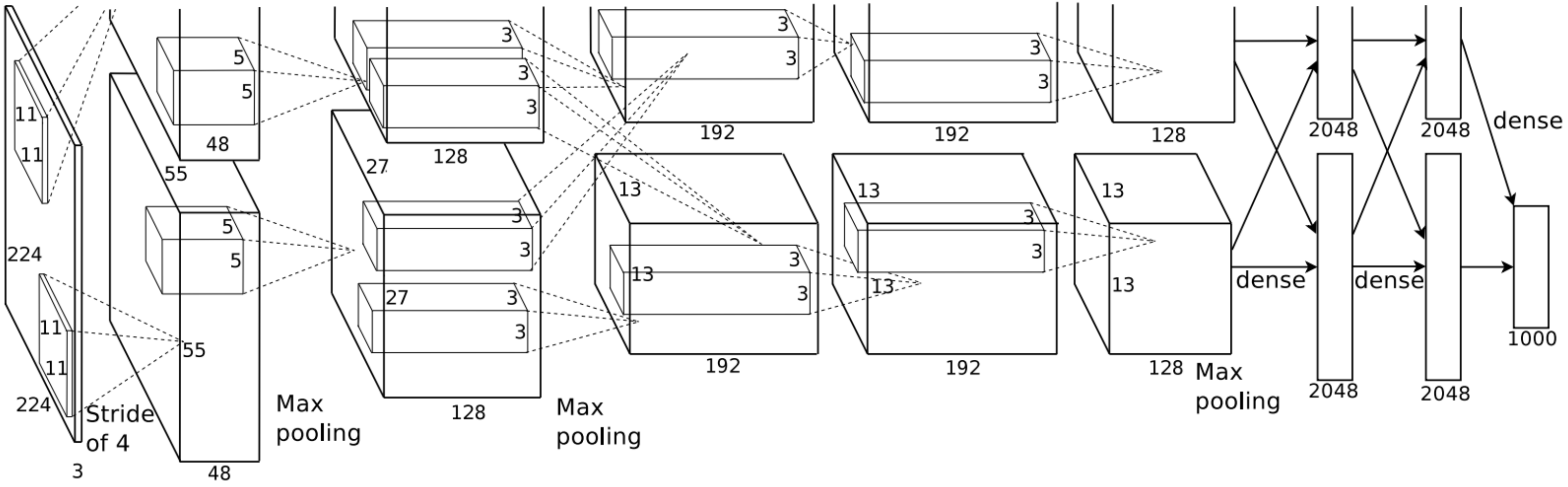
Convolutional Neural Net Architectures

AlexNet

- Created in 2012 for the ImageNet Large Scale Visual Recognition Challenge (ILSVRC)
- Task: predict the correct label from among 1000 classes
- Dataset: around 1.2 million images
- Considered the “flash point” for modern deep learning
- Demolished the competition.
- Top 5 error rate of 15.4%
- Next best: 26.2%

AlexNet - Model Diagram



AlexNet - Details

- They performed *data augmentation* for training
- Includes Cropping, horizontal flipping, and other manipulations

AlexNet - Details

- They performed *data augmentation* for training
  - Cropping, horizontal flipping, and other manipulations

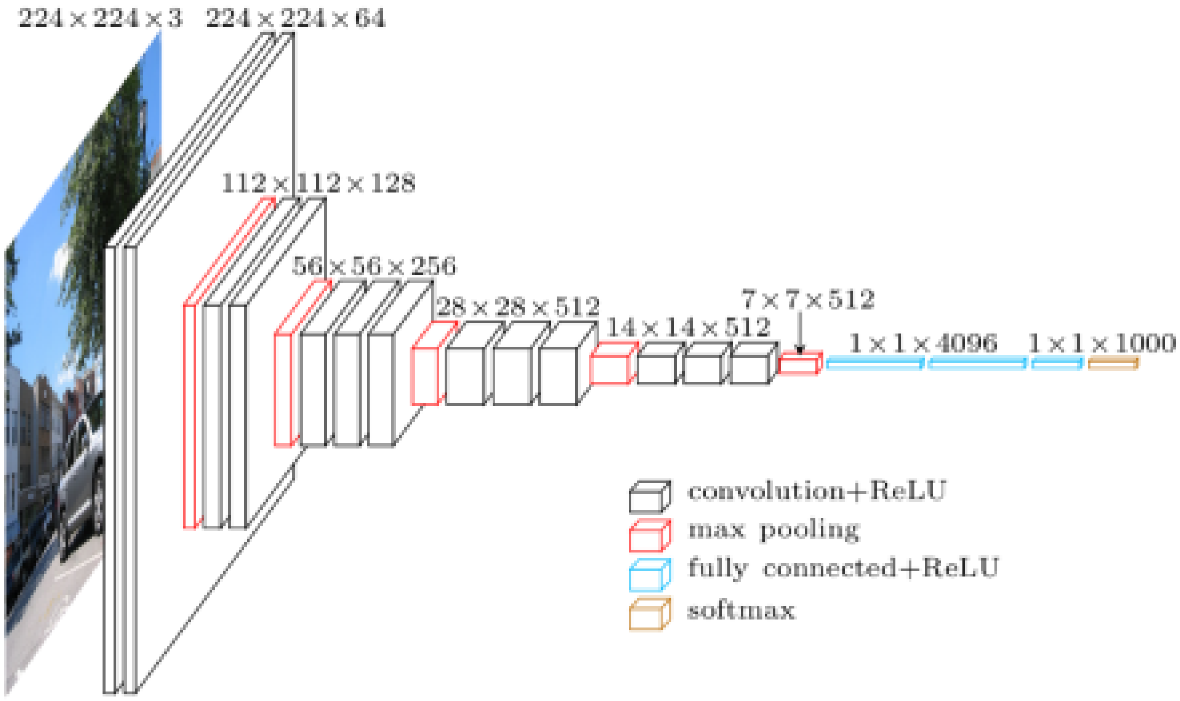
- Basic Template:
  - Convolutions with ReLUs
  - Sometimes add maxpool after convolutional layer
  - Fully connected layers at the end before a softmax classifier

VGG

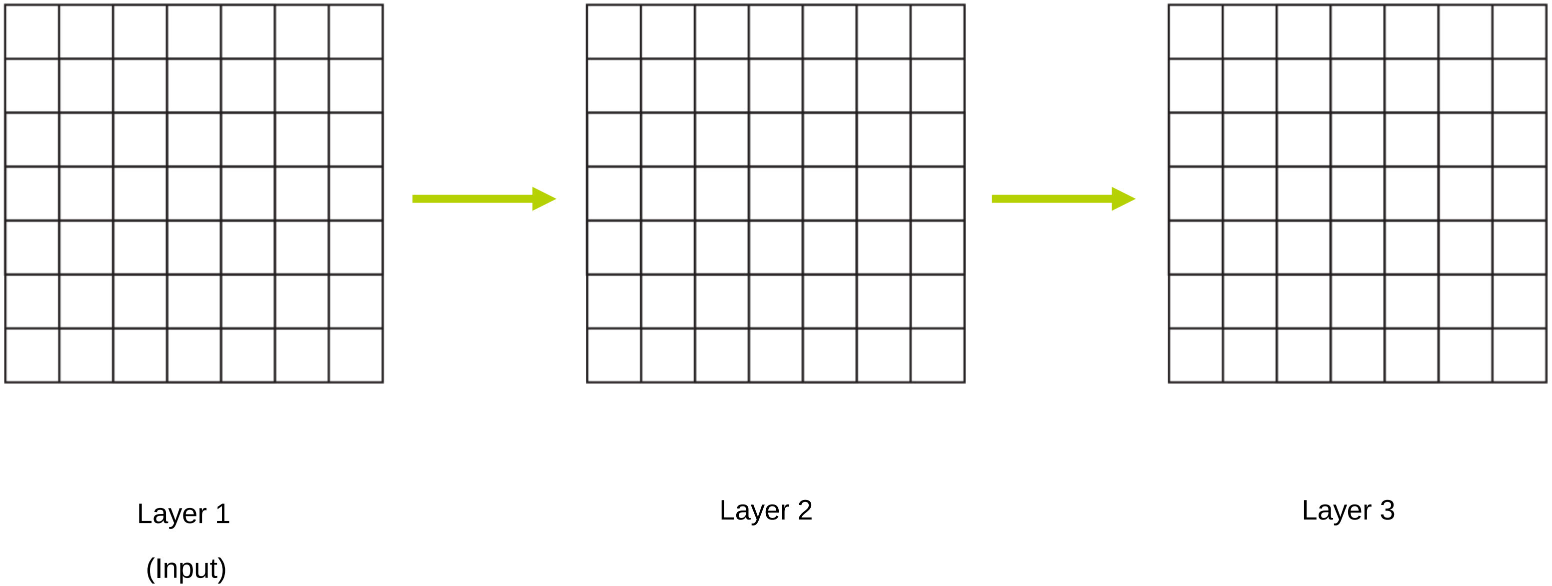
- Simplify Network Structure
- Avoid Manual Choices of Convolution Size
- Very Deep Network with 3x3 Convolutions
- These “effectively” give rise to larger convolutions

Reference:
Very Deep Convolutional Networks for Large-Scale Image Recognition
Karen Simonyan and Andrew Zisserman, 2014

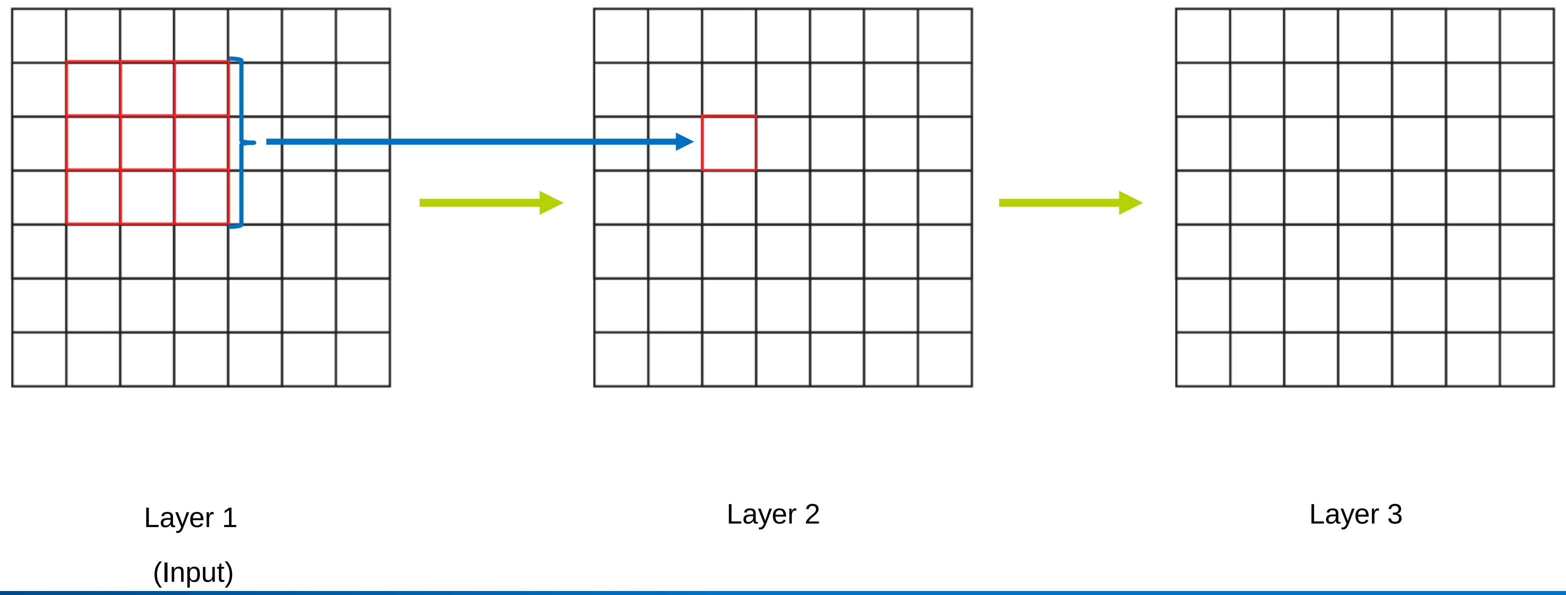
VGG16 Diagram



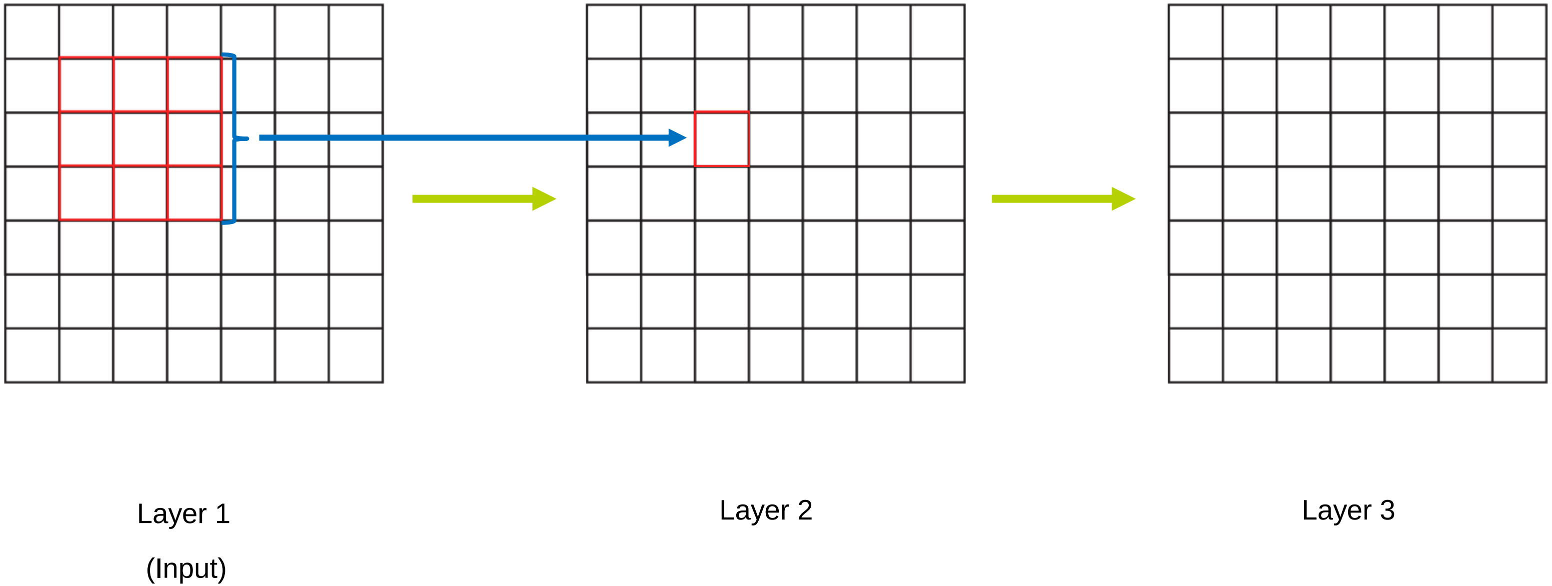
VGG



VGG



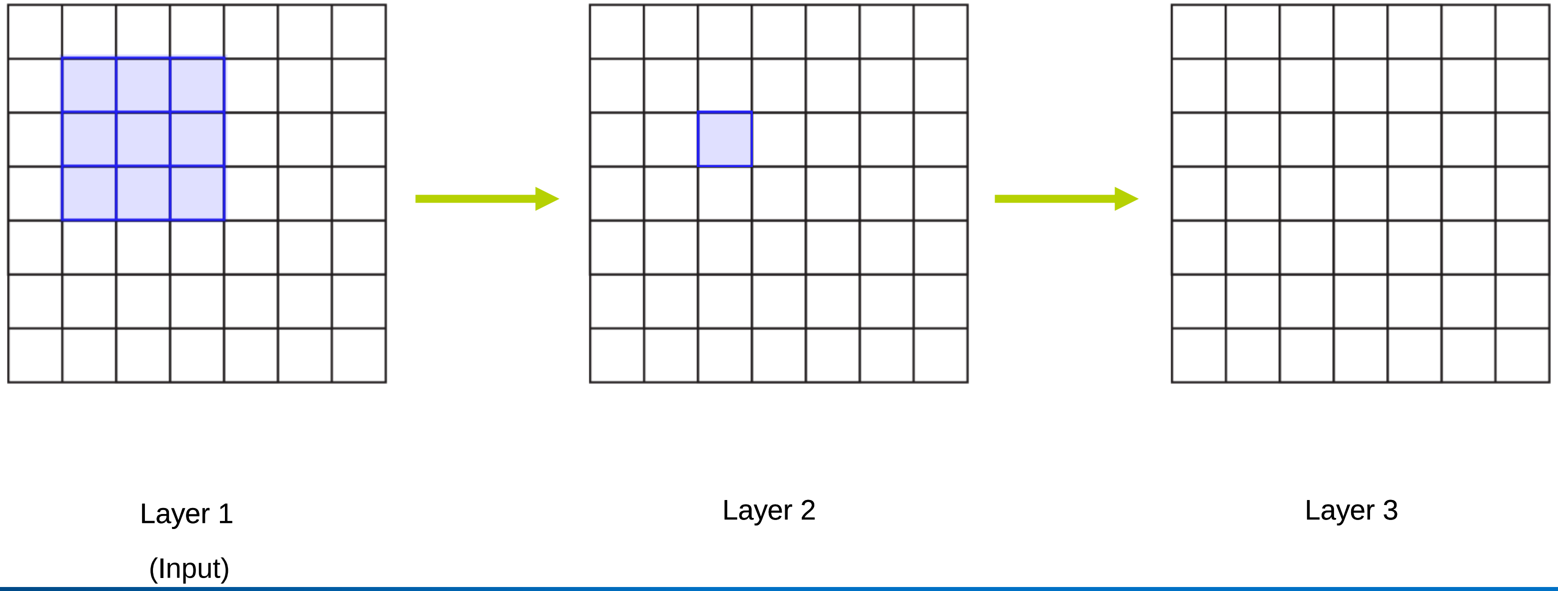
VGG



VGG

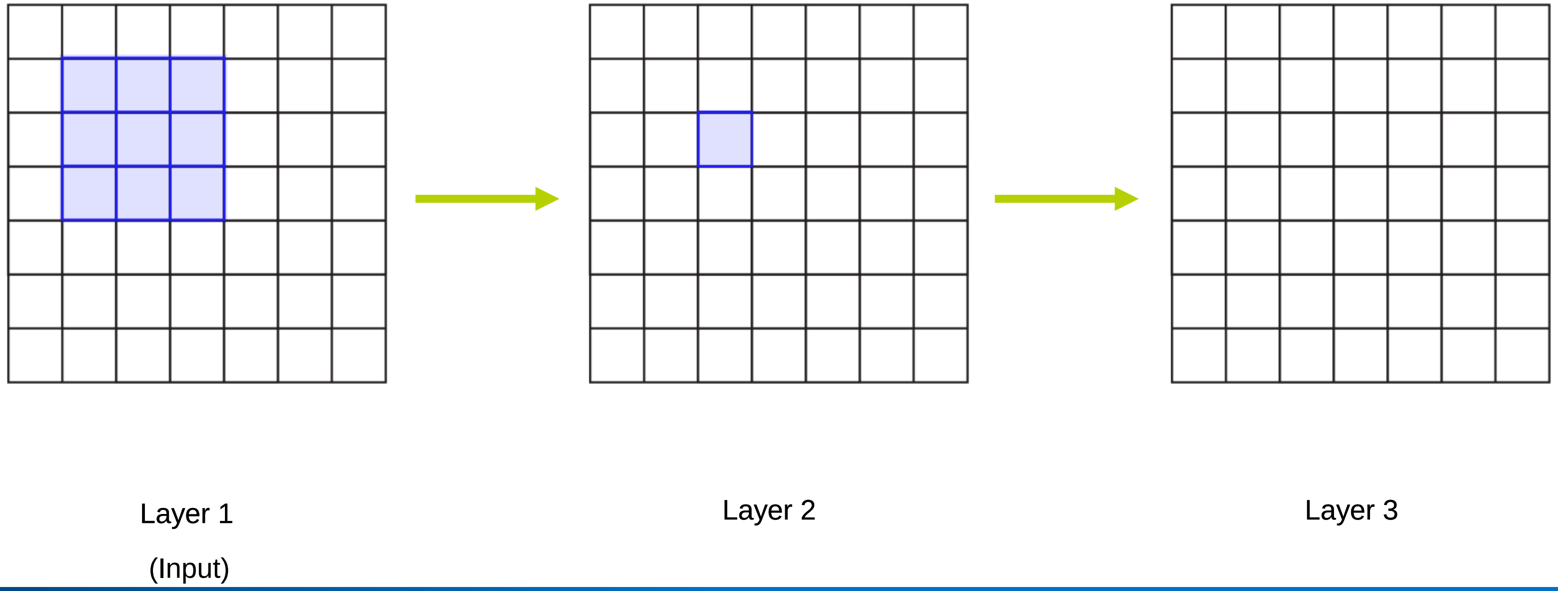
We can say that the “receptive field” of Layer 2 is 3x3

Each output has been influenced by a 3x3 patch of inputs



VGG

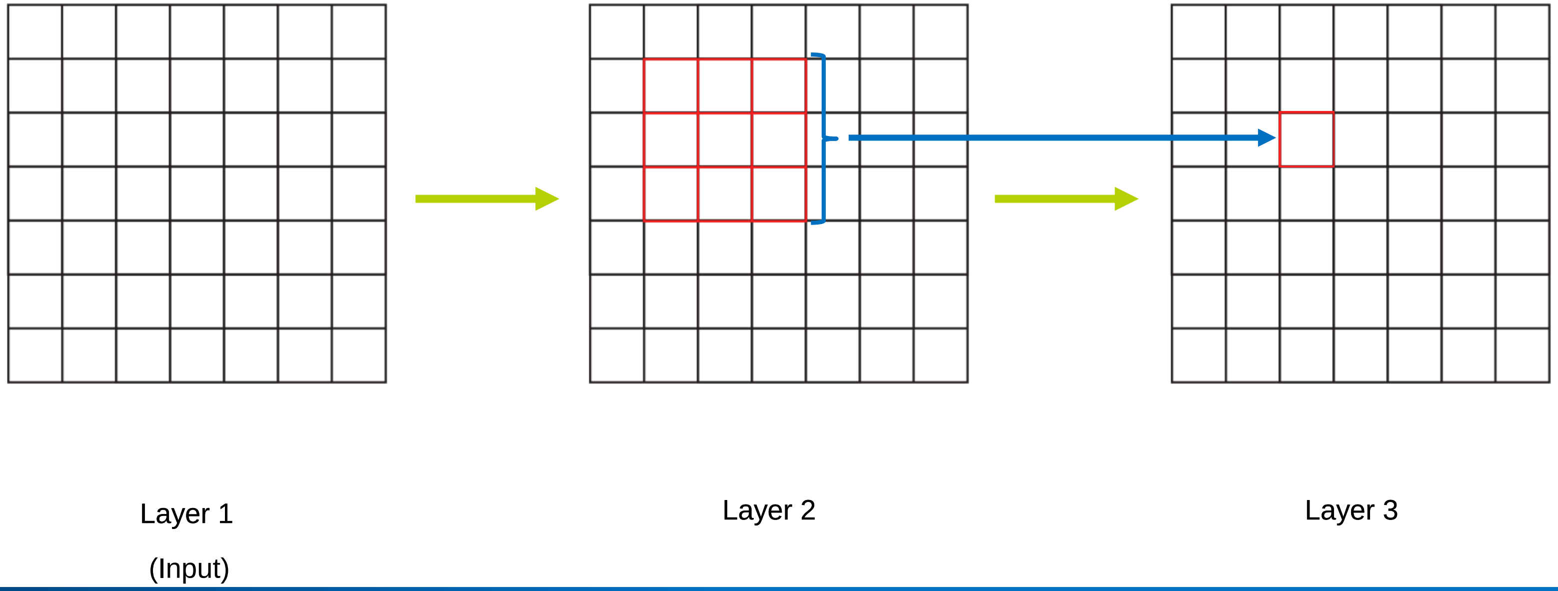
What about on Layer 3?



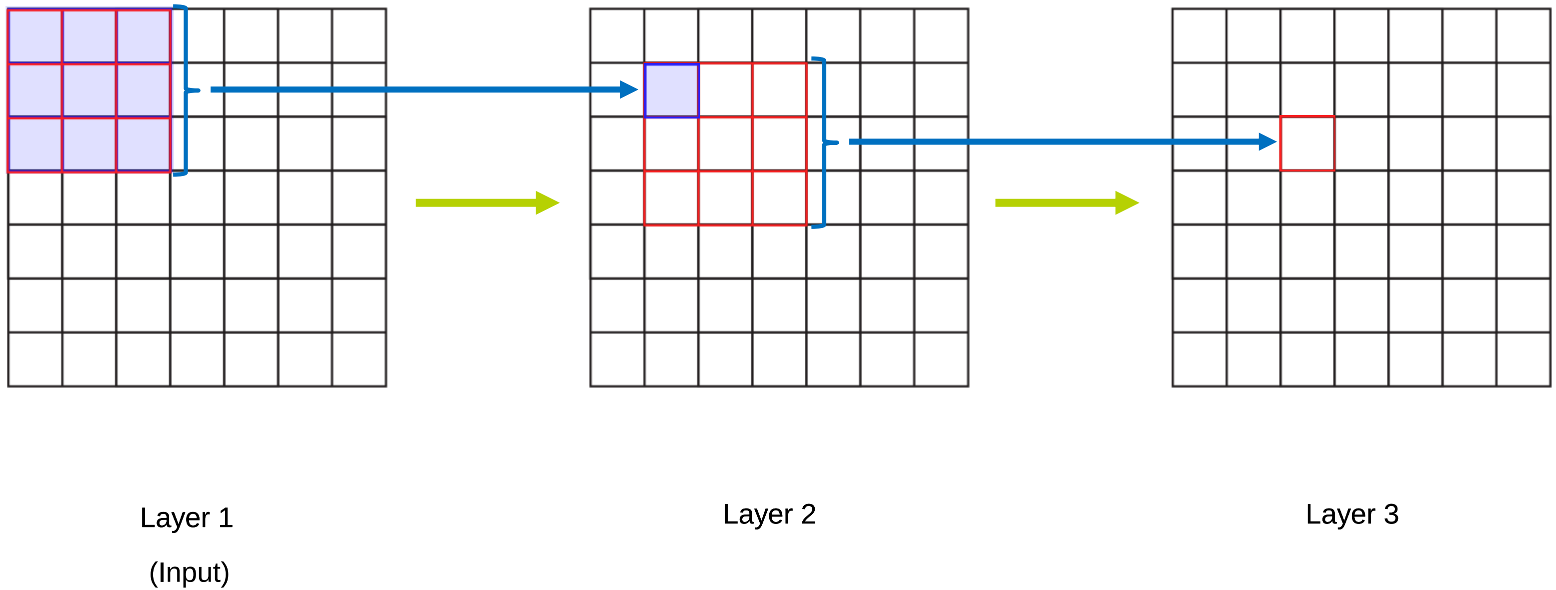
VGG

This output on Layer 3 uses a 3x3 patch from Layer 2

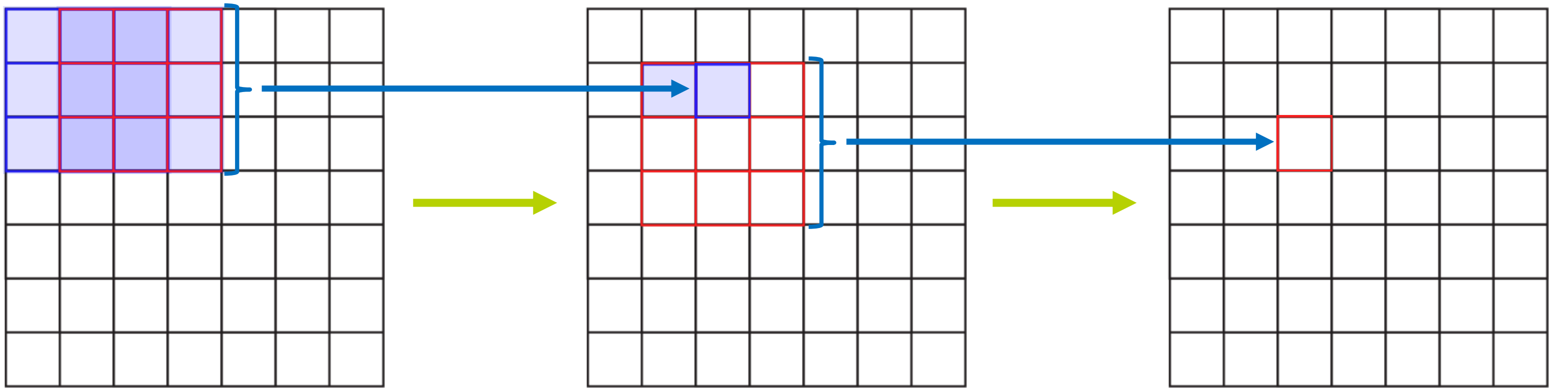
How much from Layer 1 does it use?



VGG



VGG



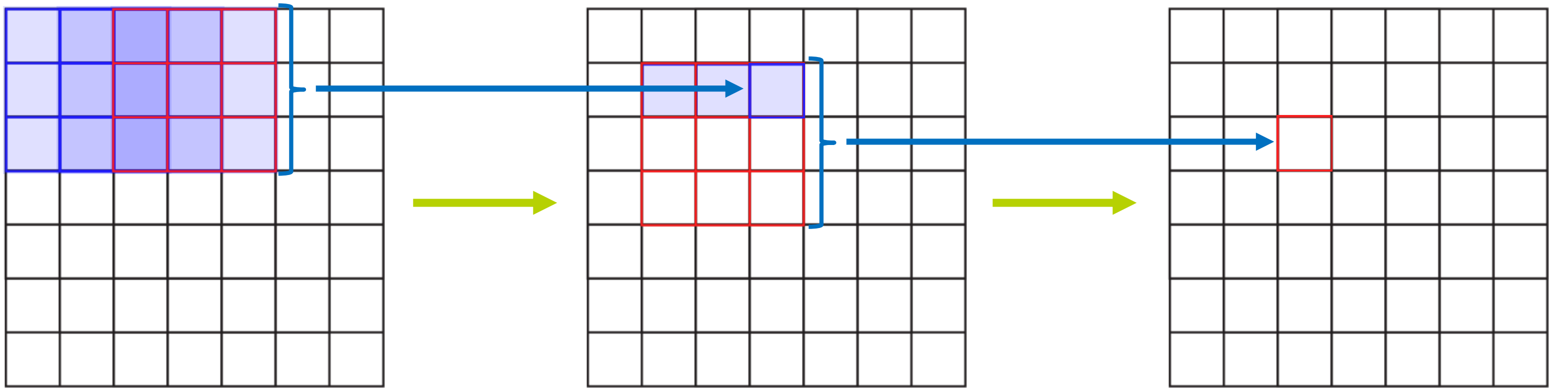
Layer 1

(Input)

Layer 2

Layer 3

VGG



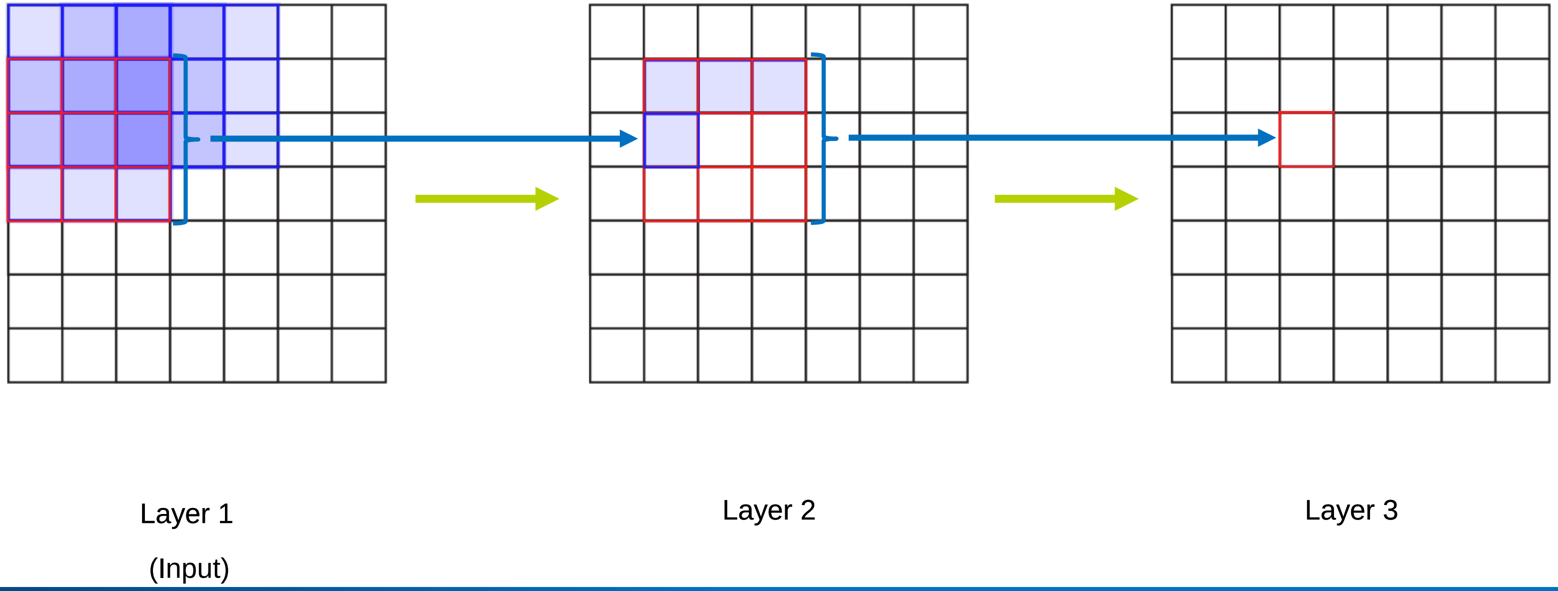
Layer 1

(Input)

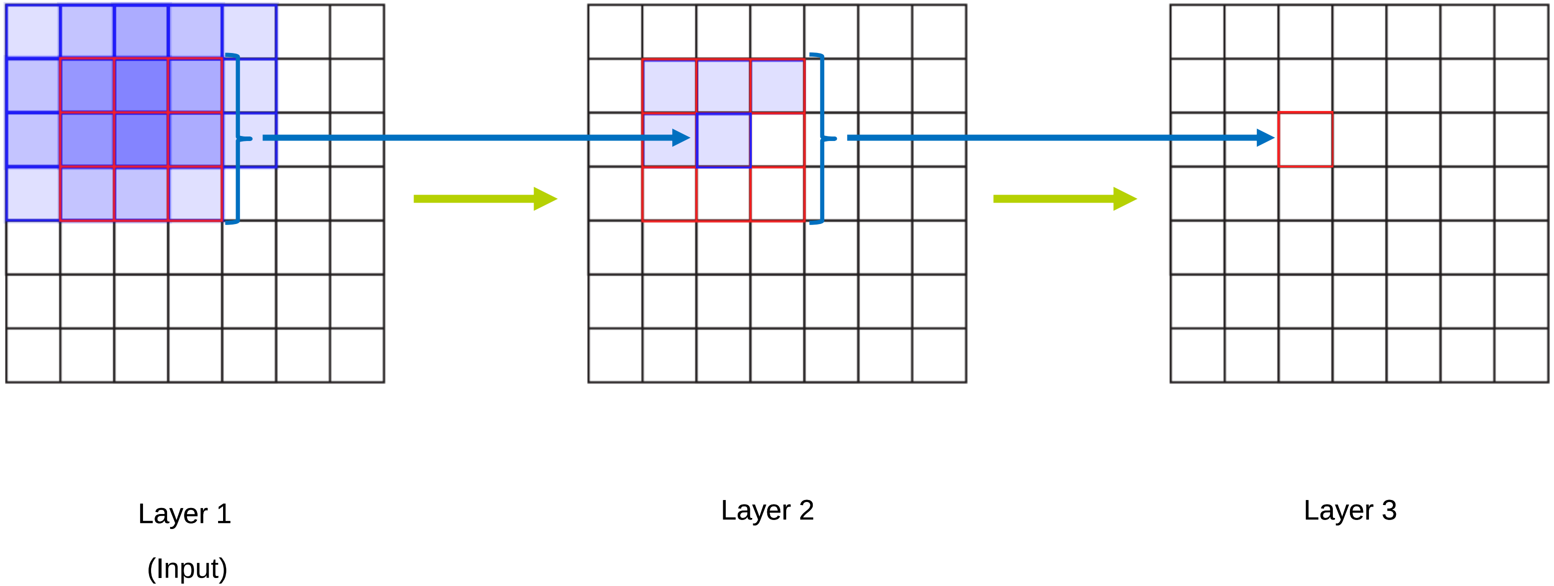
Layer 2

Layer 3

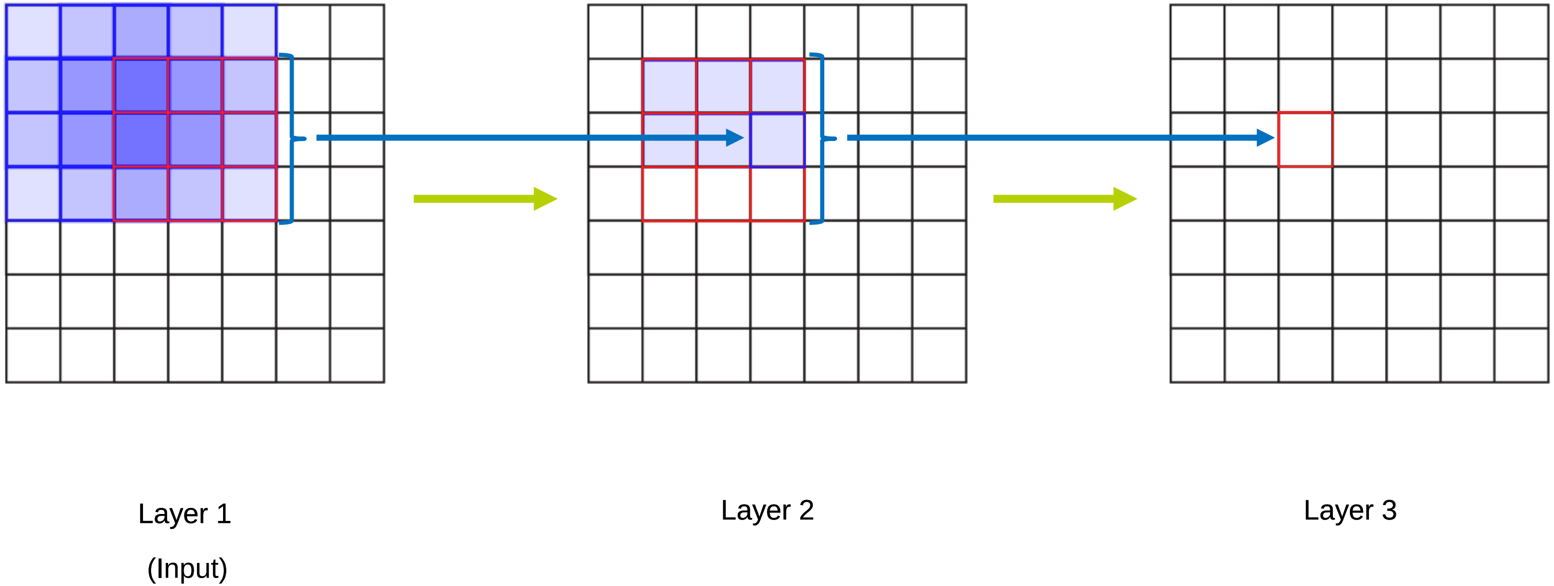
VGG



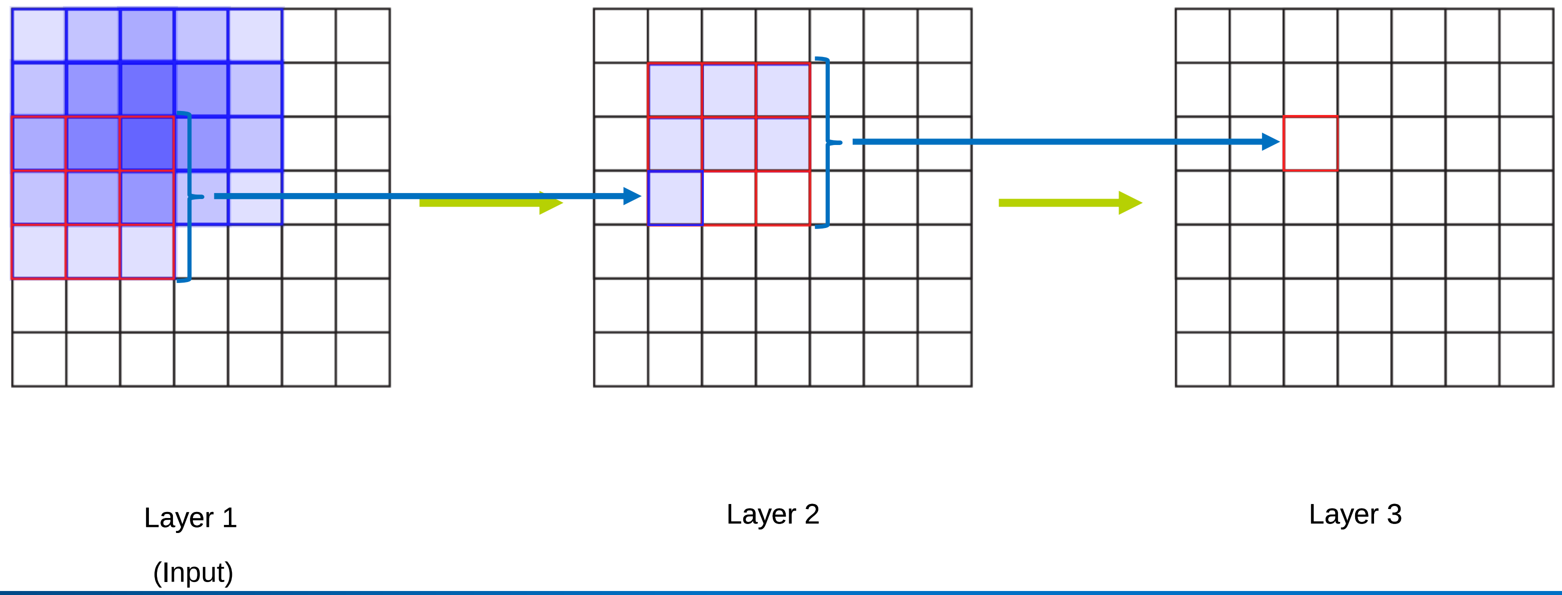
VGG



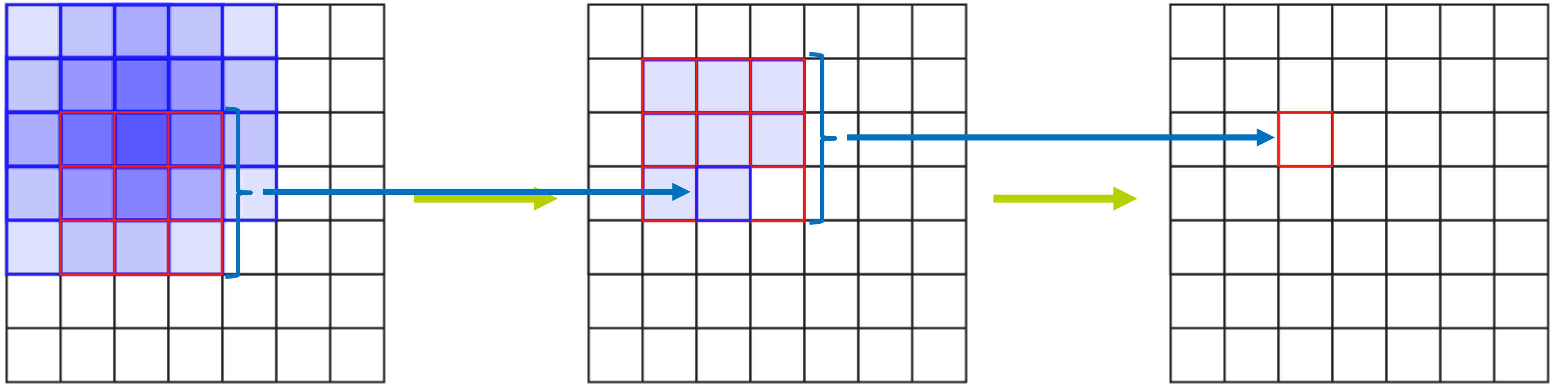
VGG



VGG



VGG



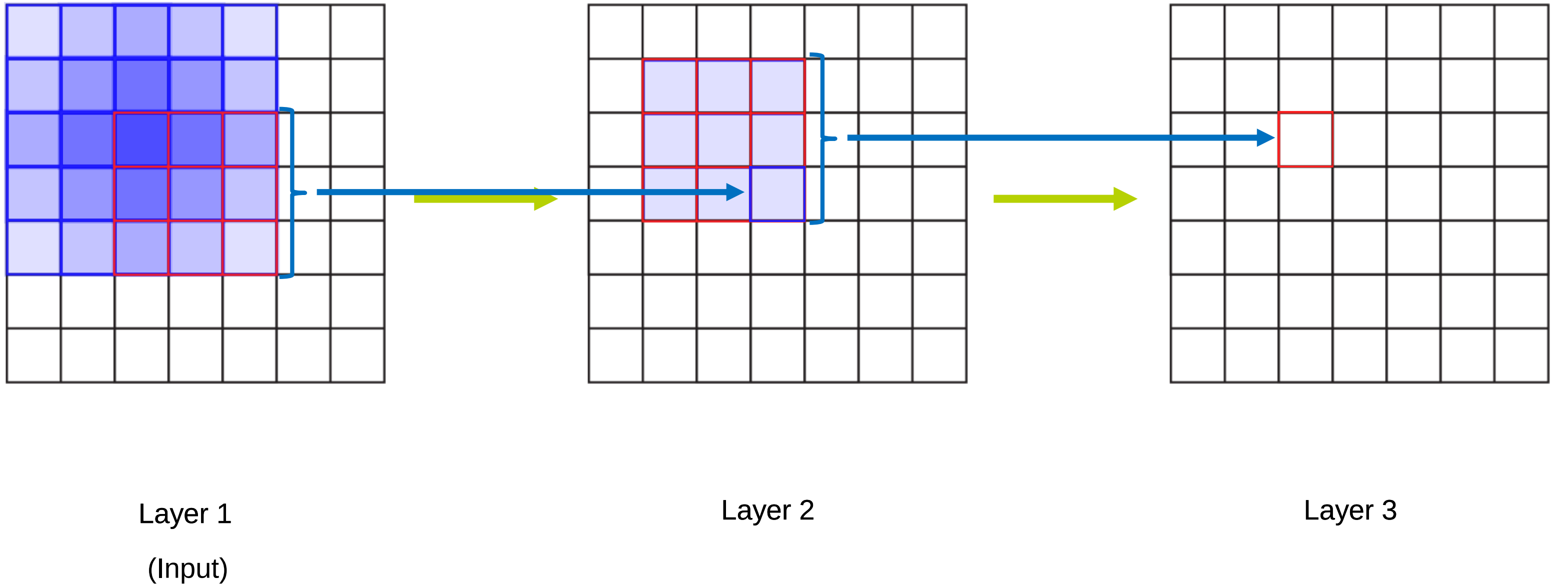
Layer 1

(Input)

Layer 2

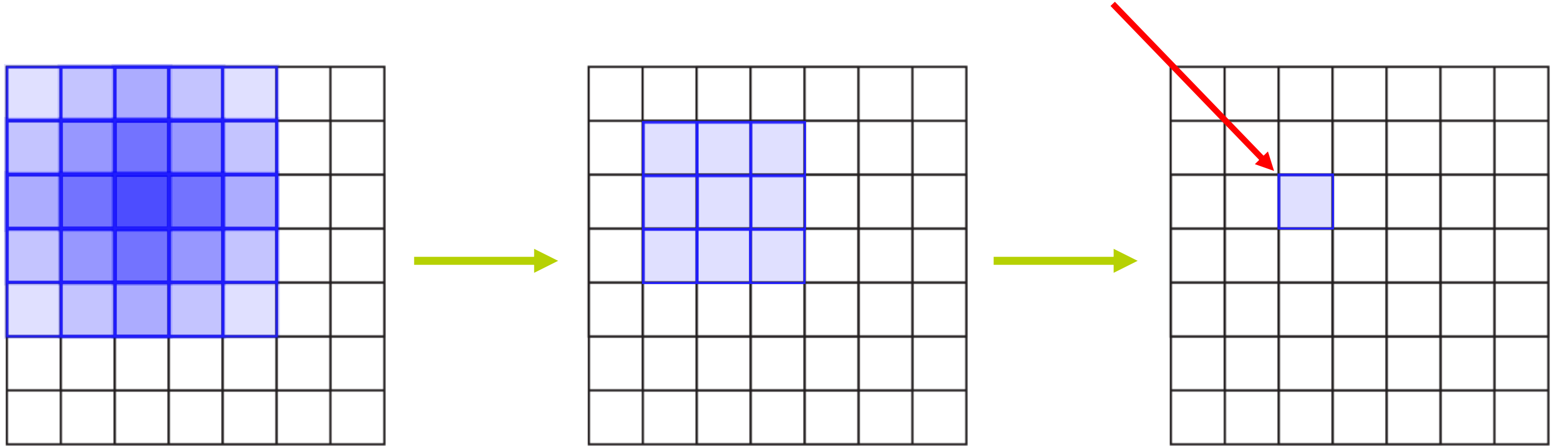
Layer 3

VGG



VGG

Each square in Layer 3 “sees” a 5x5 grid from Layer 1



Layer 1

(Input)

Layer 2

Layer 3

VGG

Two 3x3, stride 1 convolutions in a row → one 5x5

Three 3x3 convolutions → one 7x7 convolution

Benefit: fewer parameters

One 3x3 layer

$$3 \times 3 \times C \times C = 9C^2$$

One 7x7 layer

$$7 \times 7 \times C \times C = 49C^2$$

Three 3x3 layers

$$3 \times (9C^2) = 27C^2$$

$49C^2$ → $27C^2$ → ≈45% reduction!

VGG

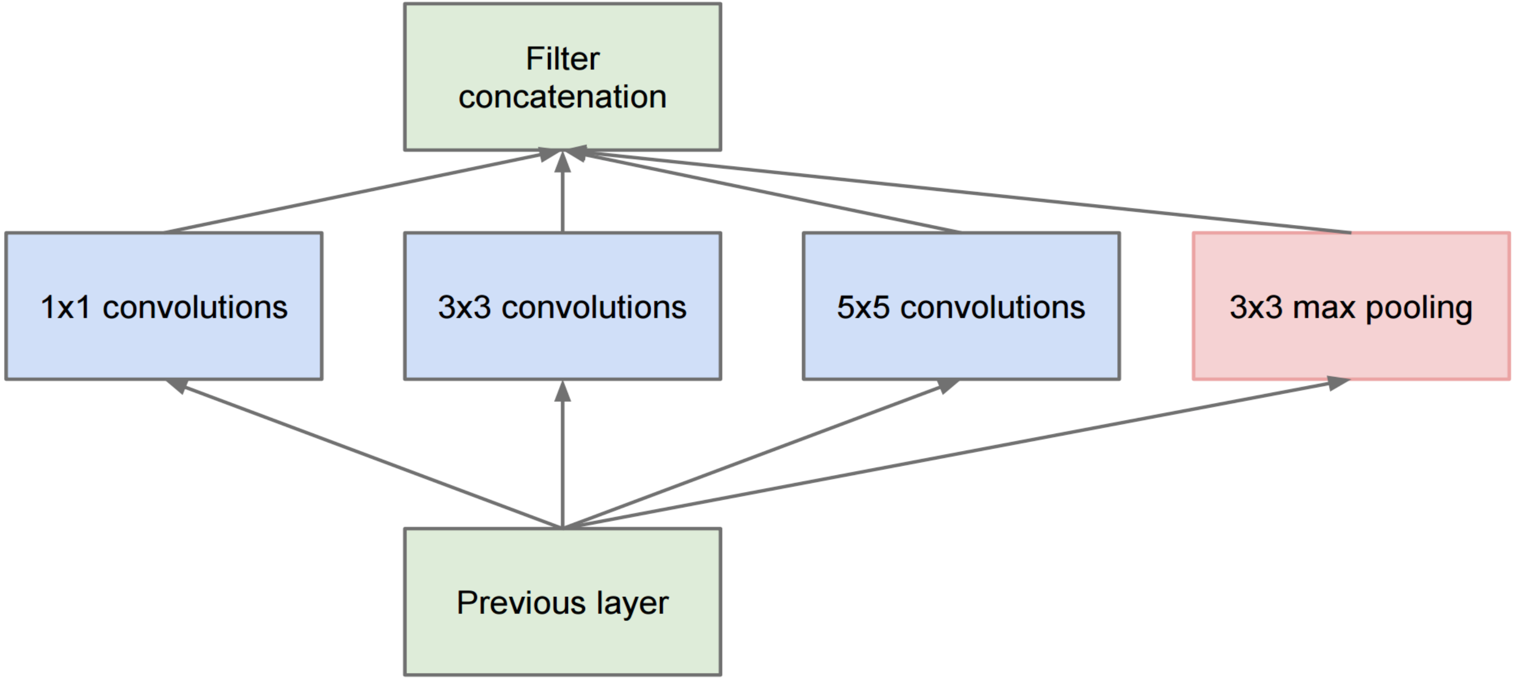
- One of the first architectures to experiment with many layers (More is better!)
- Can use multiple 3x3 convolutions to simulate larger kernels with fewer parameters
- Served as ”base model” for future works

Inception

- Szegedy et al 2014
- Idea: network would want to use different receptive fields
- Want computational efficiency
- Also want to have sparse activations of groups of neurons
- Hebbian principle: “Fire together, wire together”
- Solution: Turn each layer into branches of convolutions
- Each branch handles smaller portion of workload
- Concatenate different branches at the end

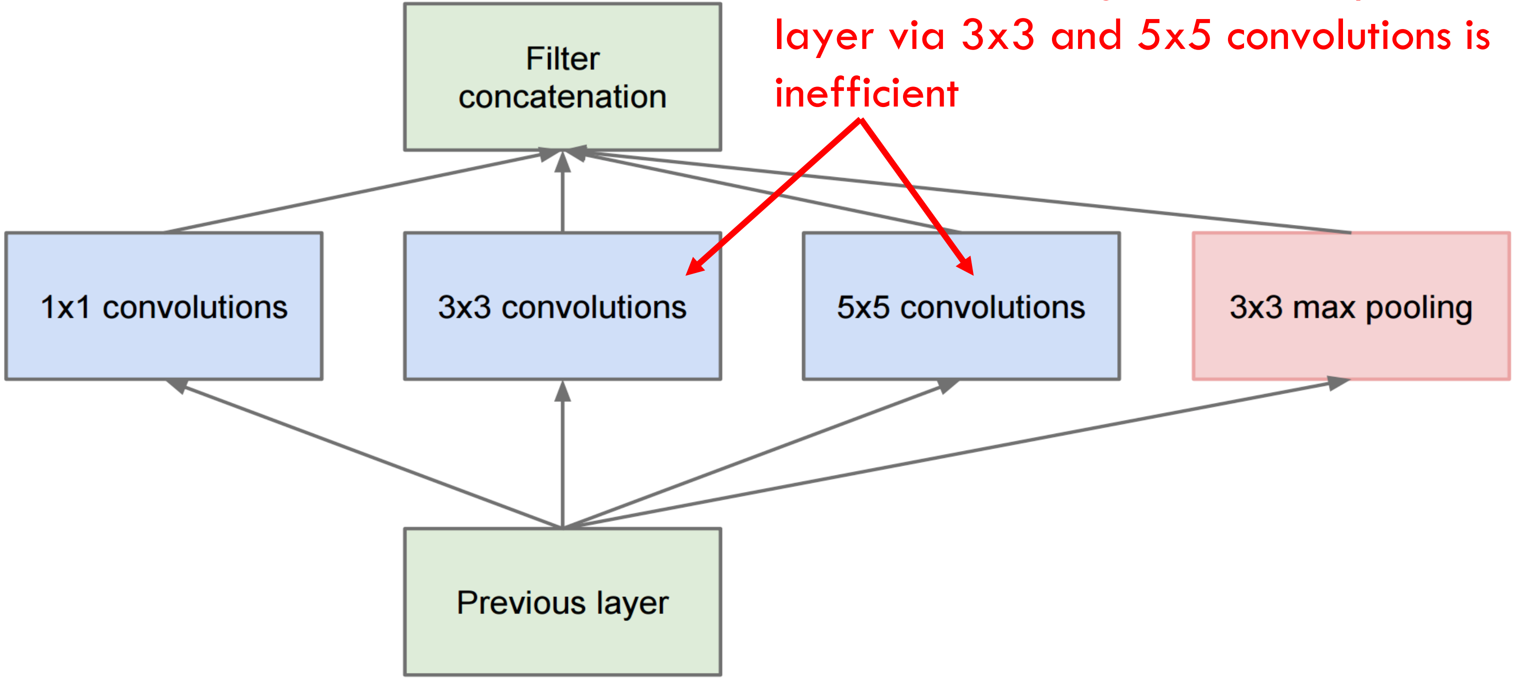
Inception

Basic idea: replace single 3x3 convolutions with module

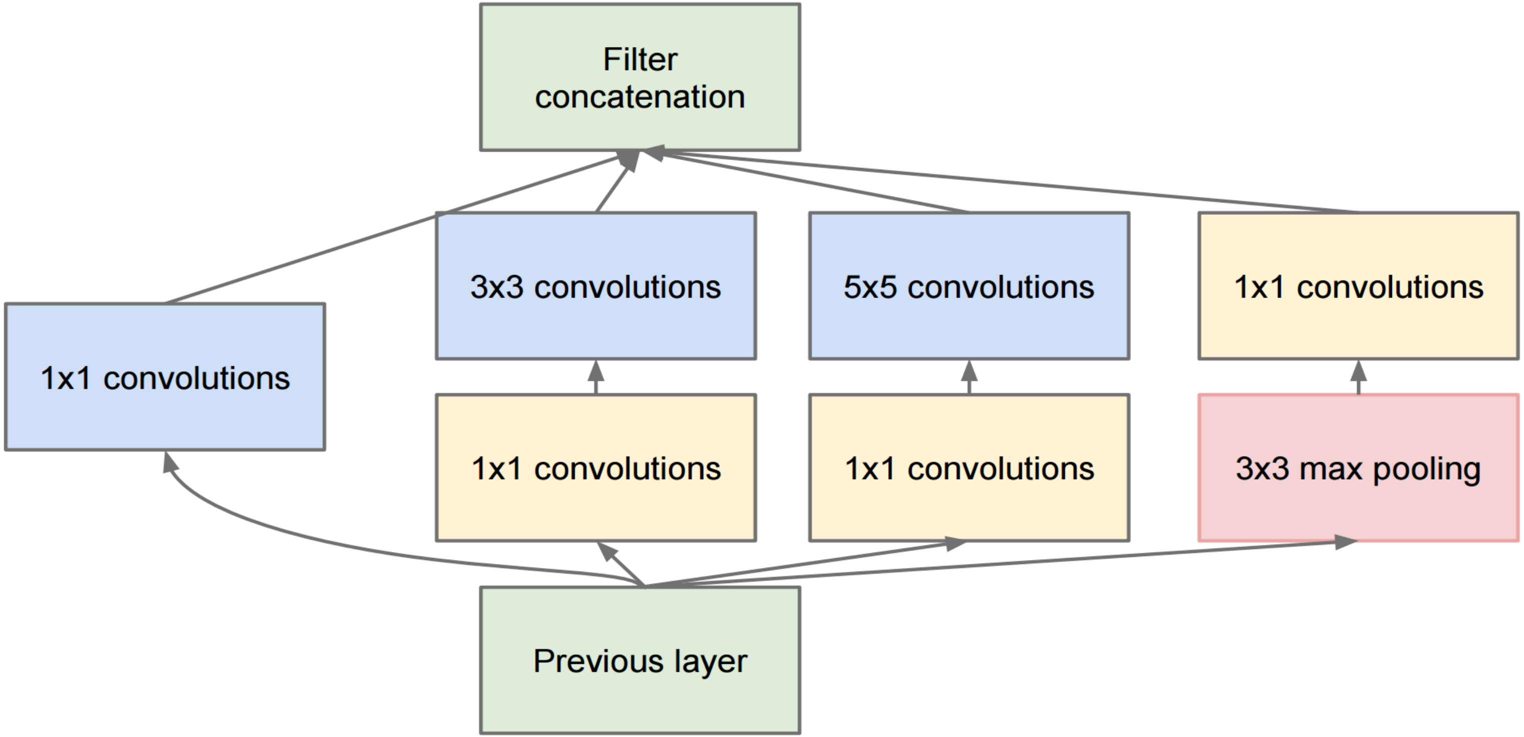


Inception

Problem: reducing filters from previous layer via 3x3 and 5x5 convolutions is inefficient



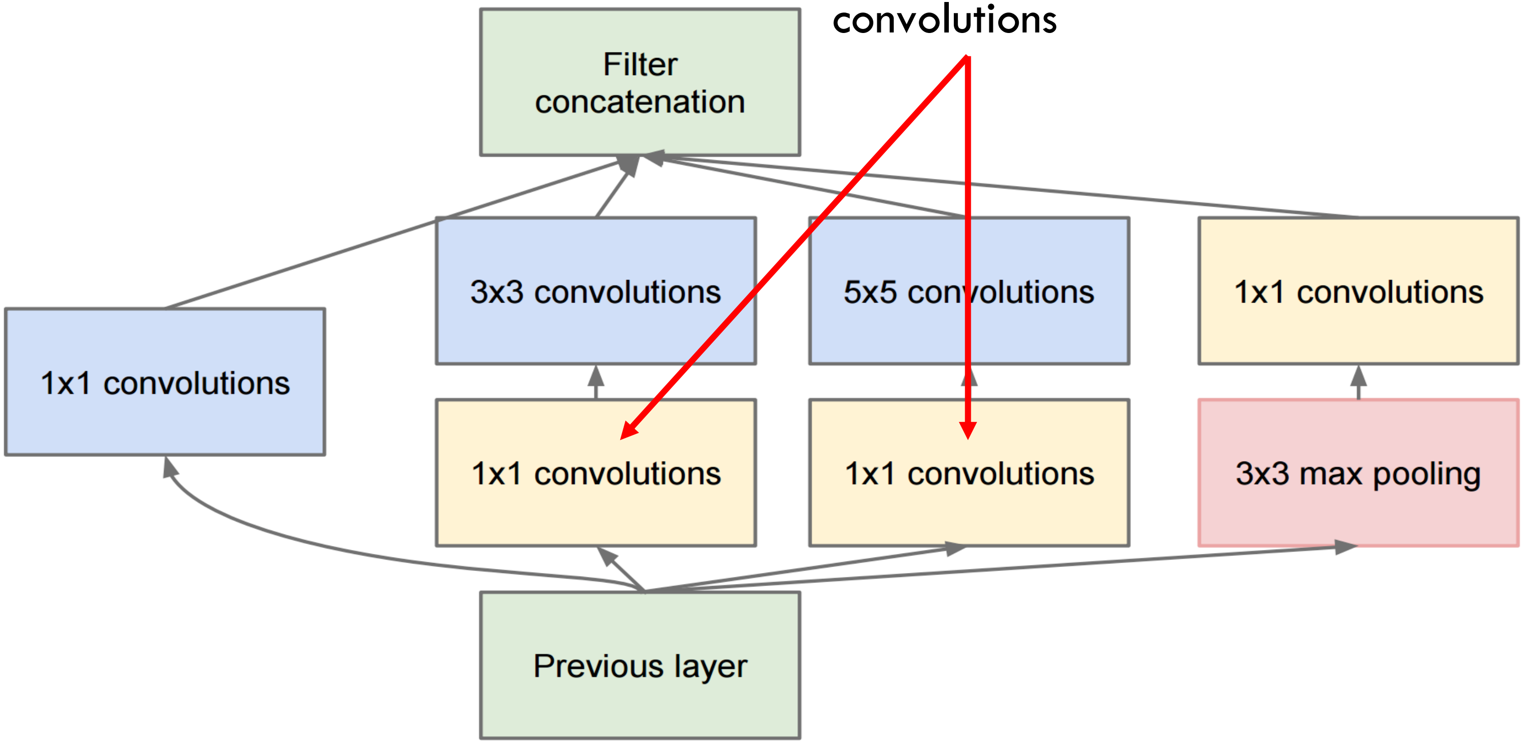
Inception



Filter concatenation
1x1 convolutions
3x3 convolutions
5x5 convolutions
1x1 convolutions
1x1 convolutions
1x1 convolutions
3x3 max pooling
Previous layer

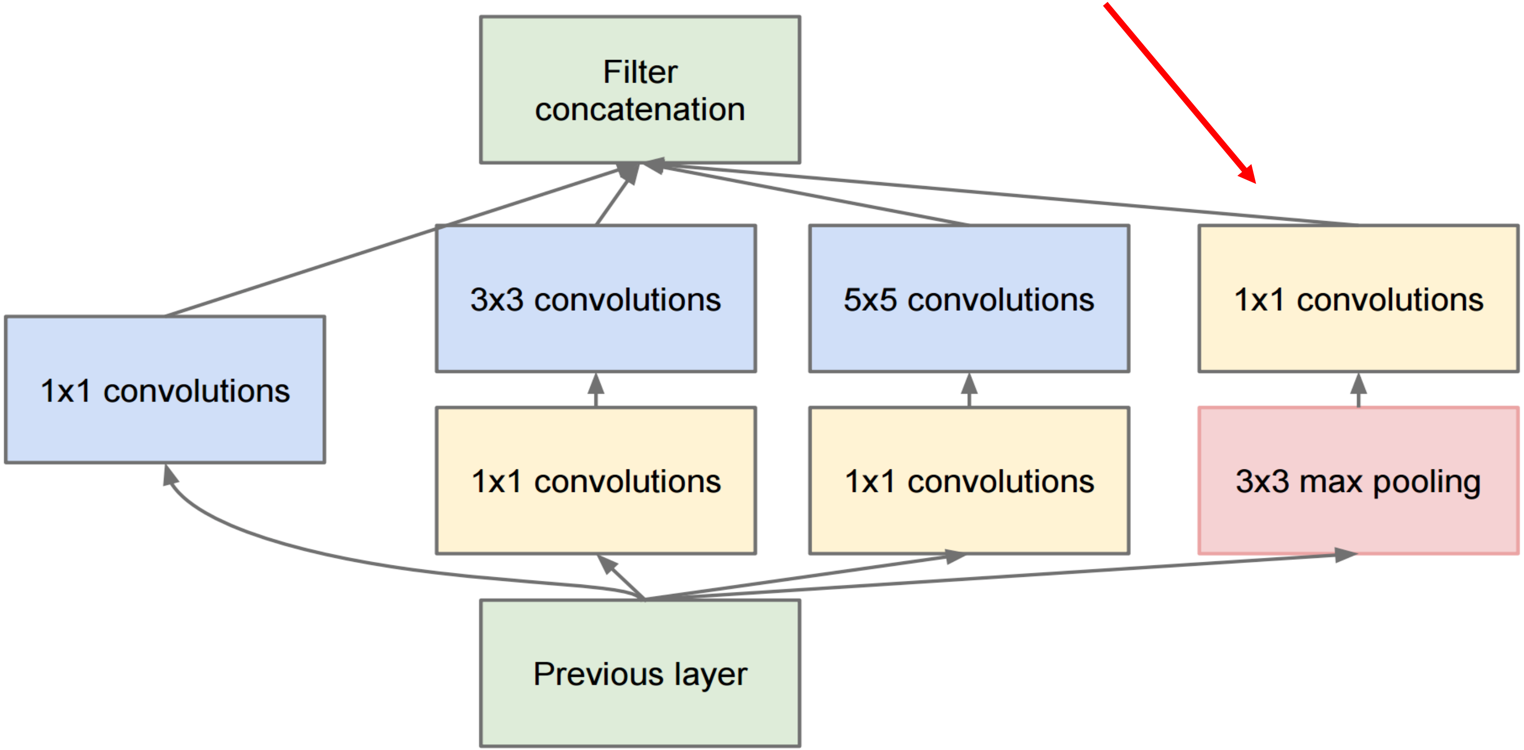
Inception

Instead, reduce parameters by reducing filter depth with 1x1 convolutions

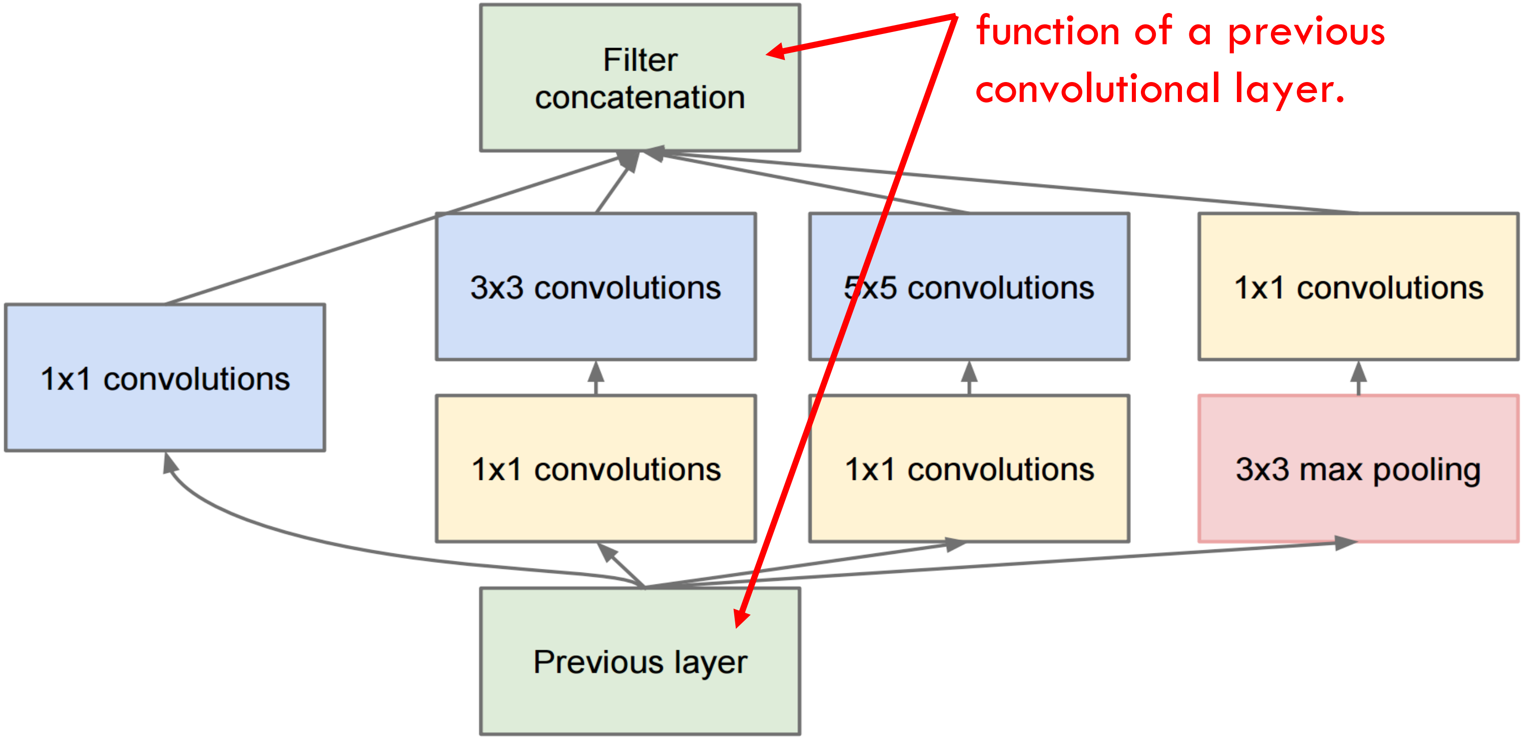


Inception

We also control the number of filters from max pool branch with a 1x1 conv

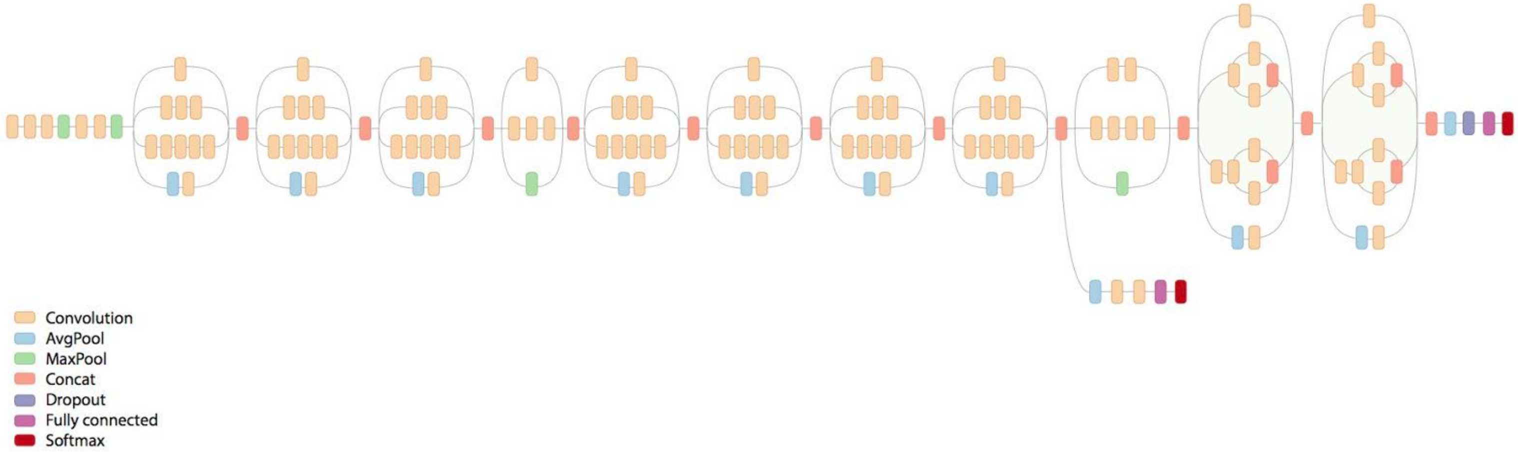


Inception



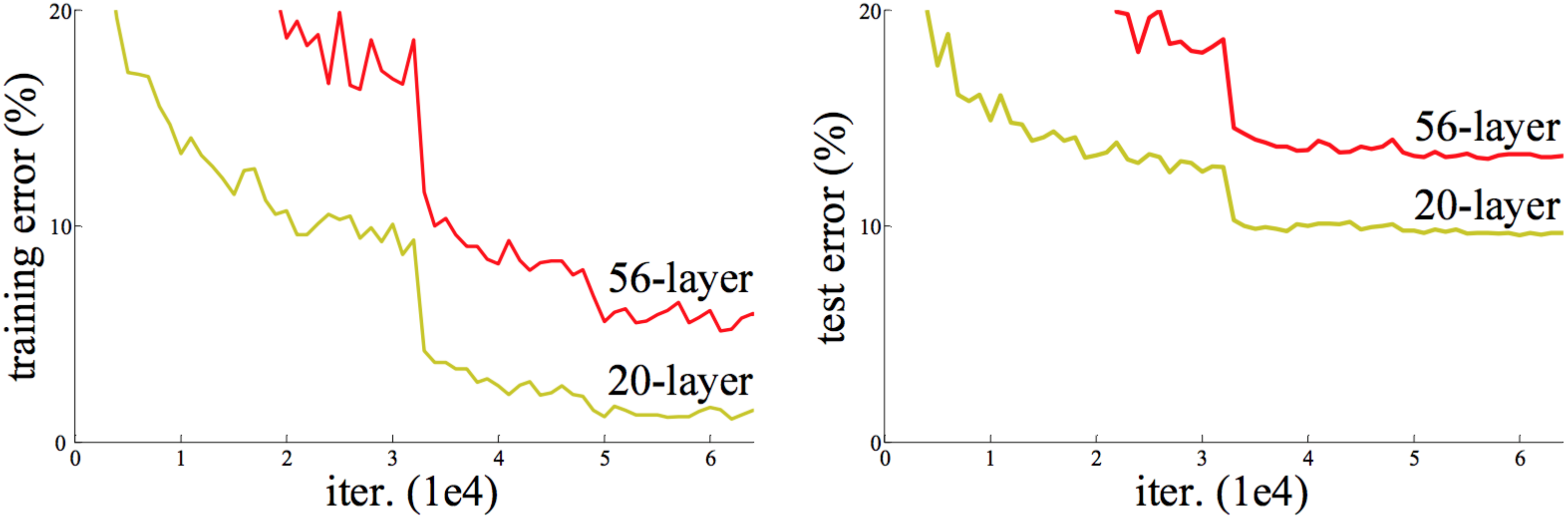
This whole "block" serves the function of a previous convolutional layer.

Inception V3 schematic



ResNet - Motivation

Issue: Deeper Networks performing worse on **training** data!
(as well as test data)

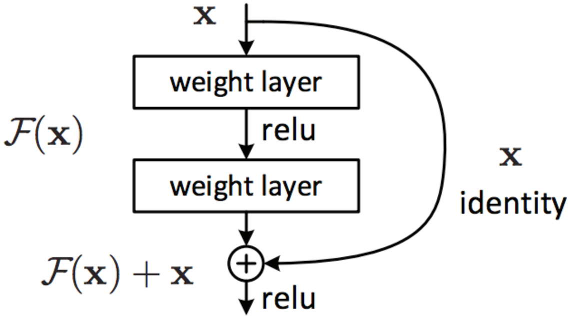


ResNet

- Surprising because deeper networks should overfit more.
- So what’s happening?
- Early layers of Deep Networks are very slow to adjust.
- Analogous to “Vanishing Gradient” issue.
- In theory, should be able to just have an “identity” transformation that makes the deeper network behave like a shallower one

ResNet

- Assumption: best transformation over multiple layers is close to $\mathcal{F}(x) + x$
- x → input to series of layers
- $\mathcal{F}(x)$ → function represented by several layers (such as convs)
- Enforce this by adding "shortcut connections"
- Add the inputs from an earlier layer to the output of current layer



ResNet

- Add previous layer back in to current layer!
- Similar idea to “boosting”