Deep Neural Networks

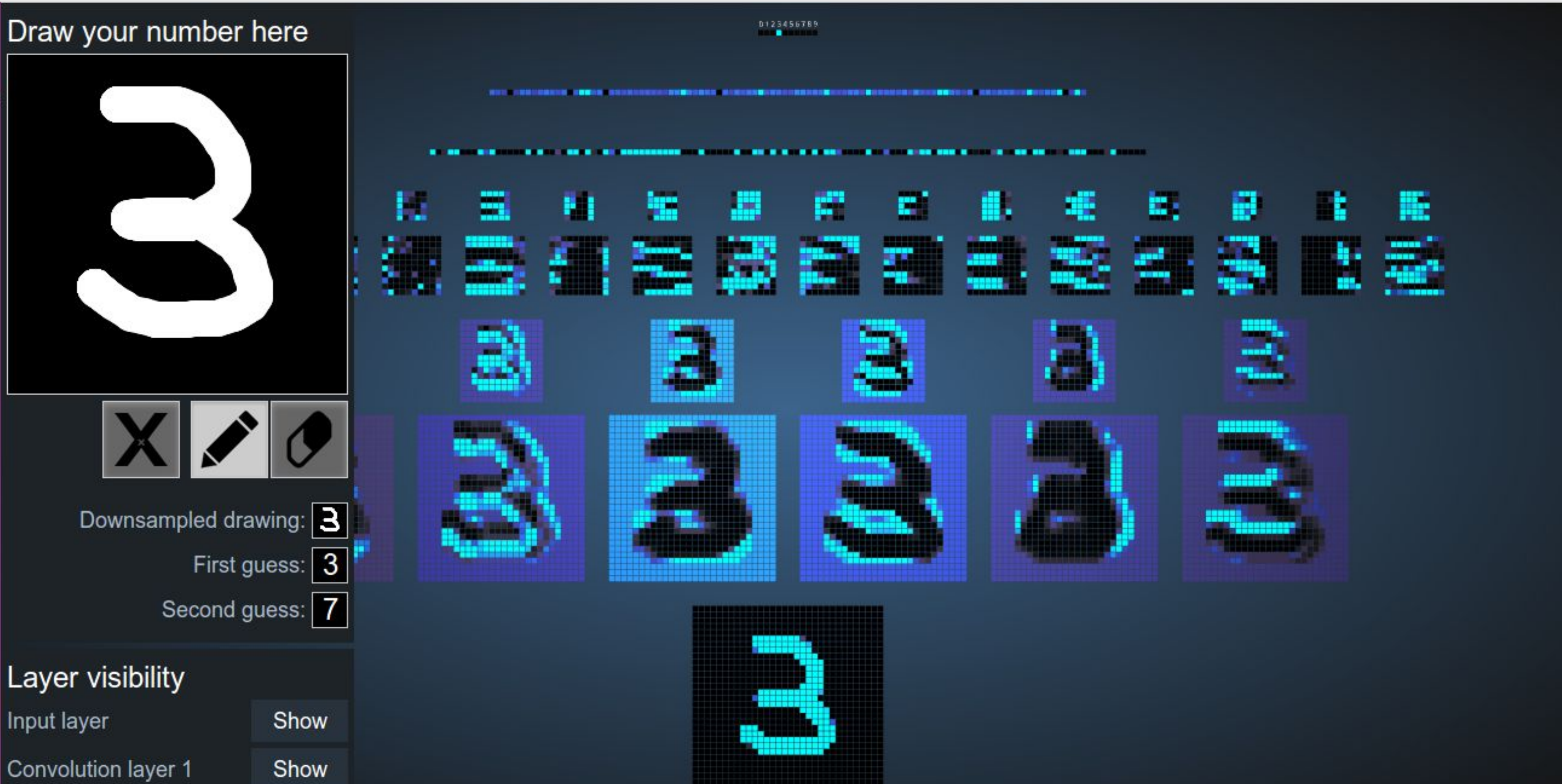
Machine Learning and Pattern Recognition

(Largely based on slides from Fei-Fei Li & Justin Johnson & Serena Yeung)

Prof. Sandra Avila
Institute of Computing (IC/Unicamp)

MC886/MO444, October 4, 2018

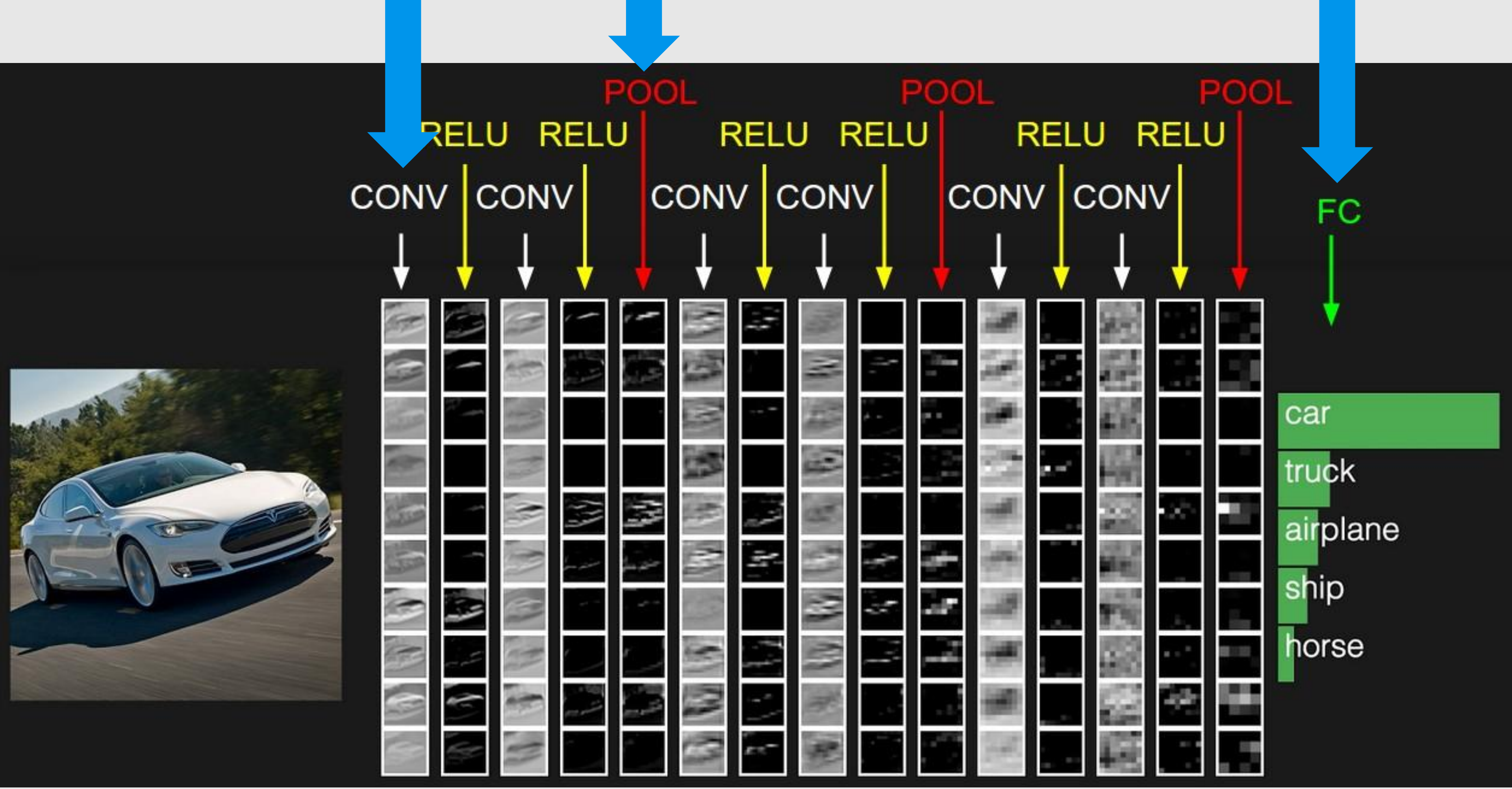
http://scs.ryerson.ca/~aharley/vis/conv/flat.html



Visualizing a CNN trained on Handwritten Digits

- Input image: 1024 pixels (32 x 32 image)
- CONV 1 (+ RELU): 6 5 × 5 (stride 1) filters
- POOL 1: 2 × 2 max pooling (with stride 2)
- CONV 2 (+ RELU): 16 5 × 5 (stride 1) filters
- POOL 2: 2 × 2 max pooling (with stride 2)
- 2 FC layers:
  - 120 neurons in the first FC layer
  - 100 neurons in the second FC layer
- Output layer: 10 neurons in the third FC

Convolutional Neural Networks (CNNs)



Credit: http://cs231n.github.io/convolutional-networks/

Convolutional Layer

The size of the **Activation Map** (or Feature Map or Convolved Feature) is controlled by three parameters:

- **Depth**: corresponds to the **number of filters** we use for the convolution operation.
- **Stride**: the **number of pixels by which we slide** our filter matrix over the input matrix.
- **Zero-padding**: sometimes, it is convenient to pad the input matrix with **zeros around the border**.

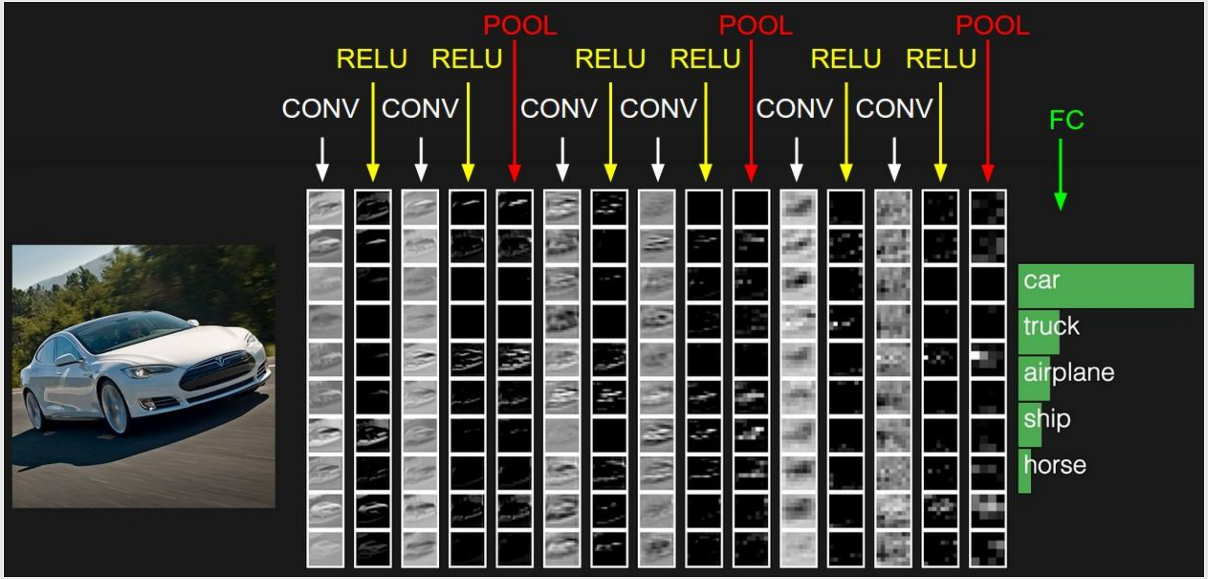
Pooling Layer

The function of Pooling is **to progressively reduce the spatial size of the input representation**. In particular, pooling

- **makes the input representations** (feature dimension) **smaller** and more manageable
- **reduces the number of parameters** and computations in the network, therefore, controlling overfitting

Fully Connected Layer

- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks



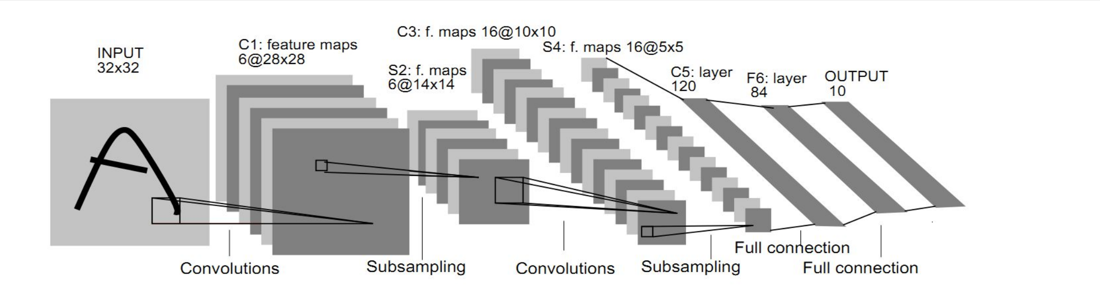
Credit: http://cs231n.github.io/convolutional-networks/

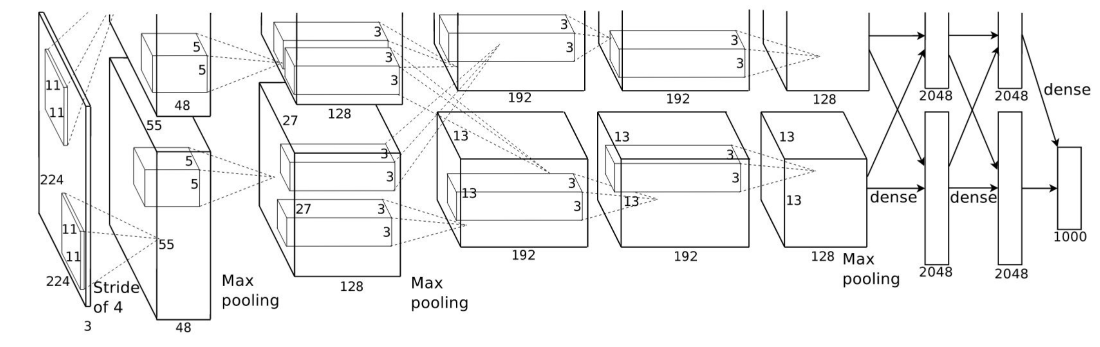
CNNs Architectures

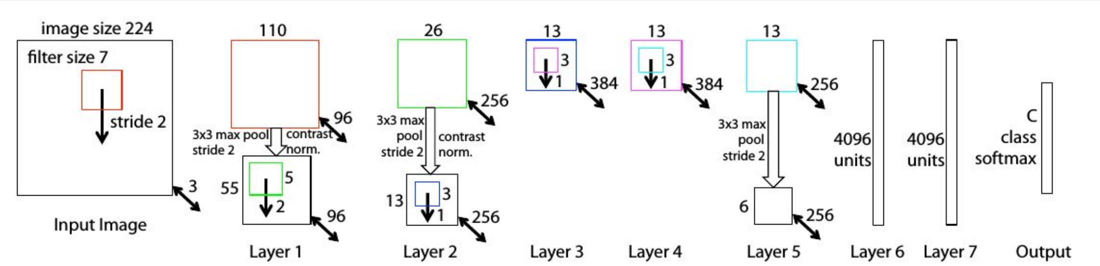
CNNs Architectures

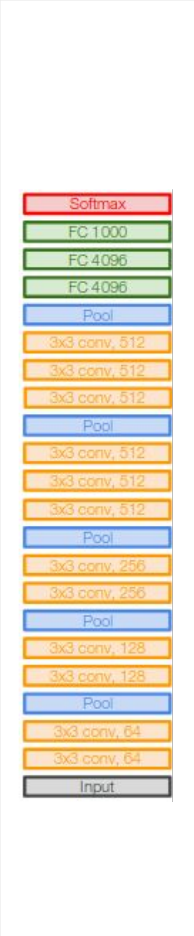
- **LeNet** by Yann LeCun, Léon Bottou & Yoshua Bengio (1998)
- **AlexNet** by Alex Krizhevsky, Ilya Sutskever & Geoff Hinton (2012)
- **ZF Net** by Matthew Zeiler & Rob Fergus (2013)
- **GoogLeNet** by Szegedy et al. (2014)
- **VGGNet** by Karen Simonyan & Andrew Zisserman (2014)
- **ResNet** by Kaiming He et al. (2015)

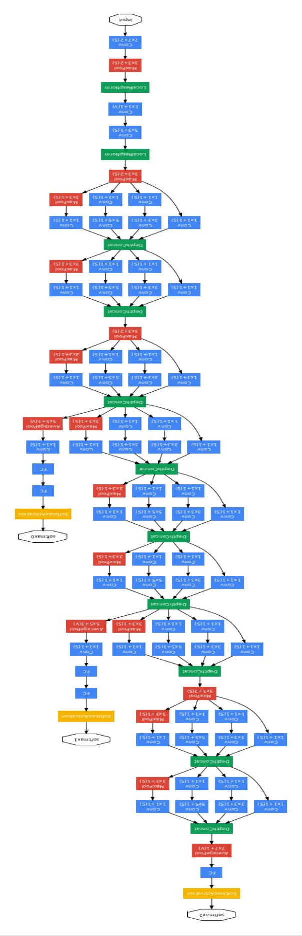
CNN-based Architectures

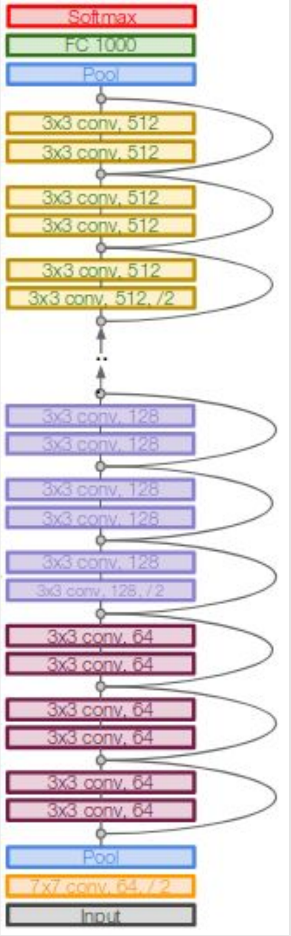




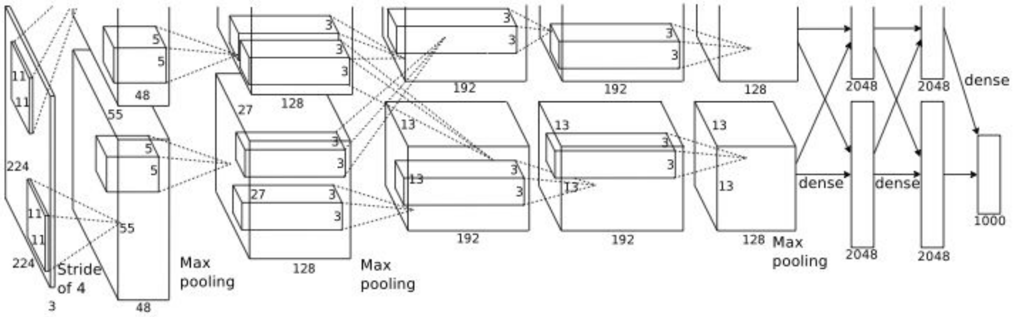




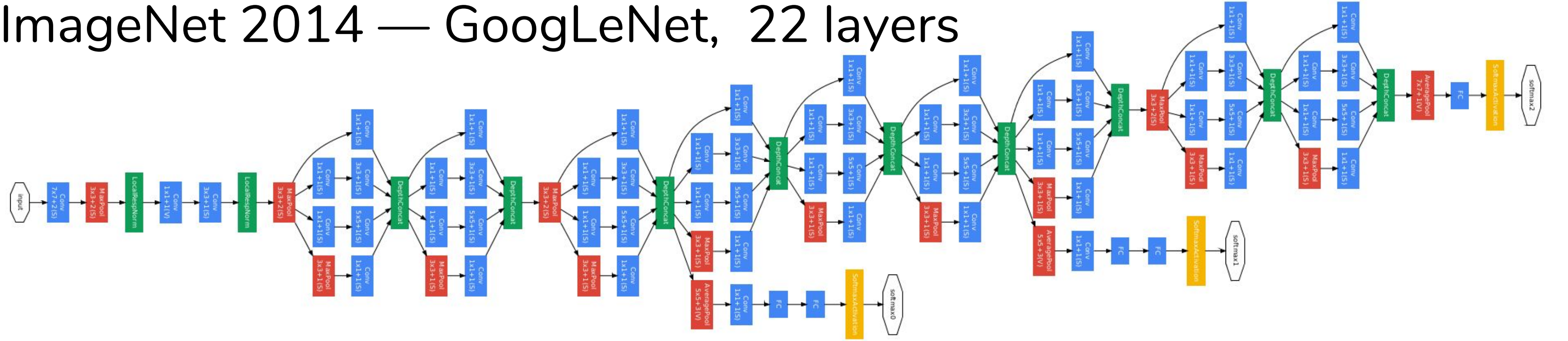




ImageNet 2012 — AlexNet, 8 layers



ImageNet 2014 — GoogLeNet, 22 layers

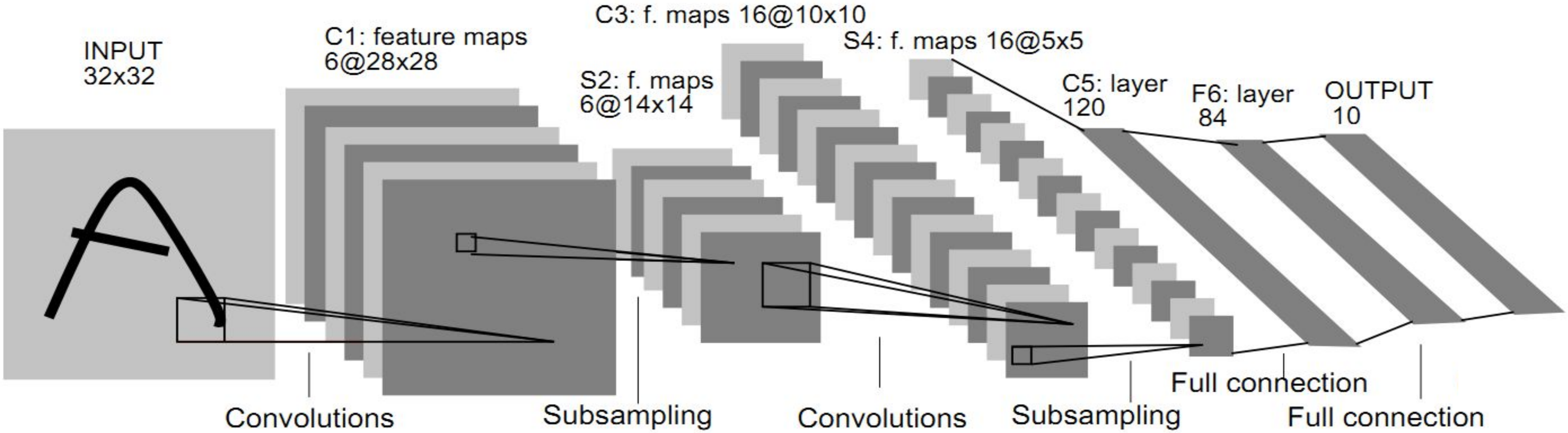


ImageNet 2015 — ResNet, 152 layers

CNNs Architectures

- **LeNet** by Yann LeCun, Léon Bottou & Yoshua Bengio (1998)
- **AlexNet** by Alex Krizhevsky, Ilya Sutskever & Geoff Hinton (2012)
- **ZF Net** by Matthew Zeiler & Rob Fergus (2013)
- **VGGNet** by Karen Simonyan & Andrew Zisserman (2014)
- **GoogLeNet** by Szegedy et al. (2014)
- **ResNet** by Kaiming He et al. (2015)

LeNet-5 [LeCun et al., 1998]



Convolution filters: 5x5 with stride 1

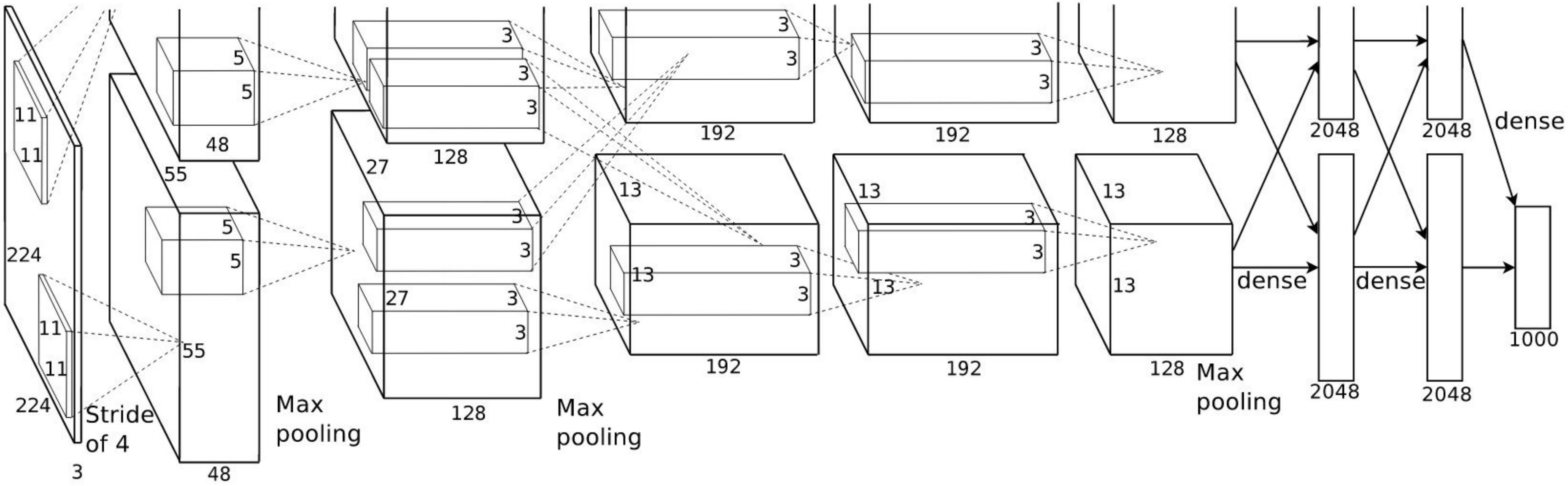
Subsampling (Pooling) layers: 2x2 with stride 2

[CONV-POOL-CONV-POOL-FC-FC]

CNNs Architectures

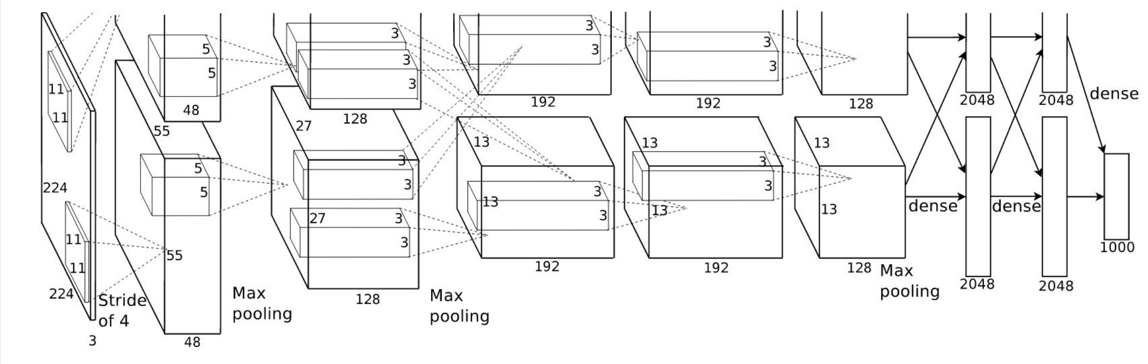
- **LeNet** by Yann LeCun, Léon Bottou & Yoshua Bengio (1998)
- **AlexNet** by Alex Krizhevsky, Ilya Sutskever & Geoff Hinton (2012)
- **ZF Net** by Matthew Zeiler & Rob Fergus (2013)
- **VGGNet** by Karen Simonyan & Andrew Zisserman (2014)
- **GoogLeNet** by Szegedy et al. (2014)
- **ResNet** by Kaiming He et al. (2015)

AlexNet [Krizhevsky et al., 2012]



AlexNet [Krizhevsky et al., 2012]

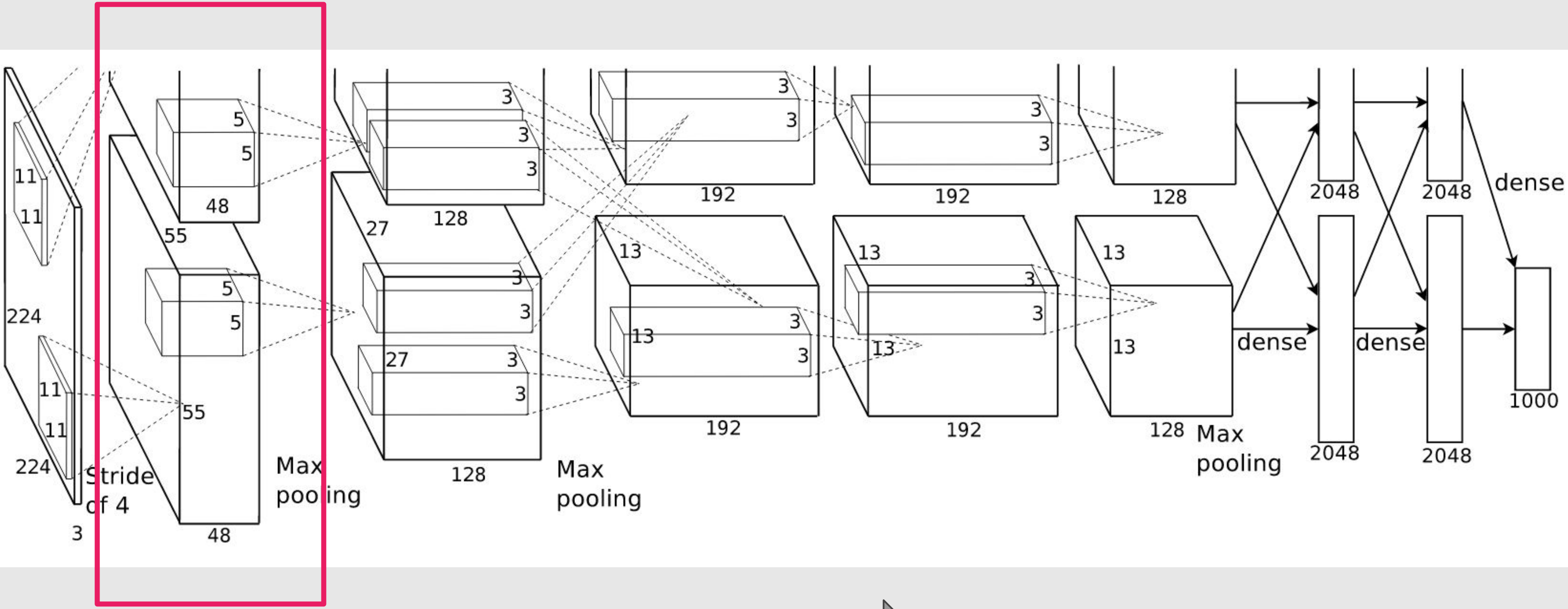
Architecture:
CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
MAX POOL3
FC6
FC7
FC8



AlexNet [Krizhevsky et al., 2012]

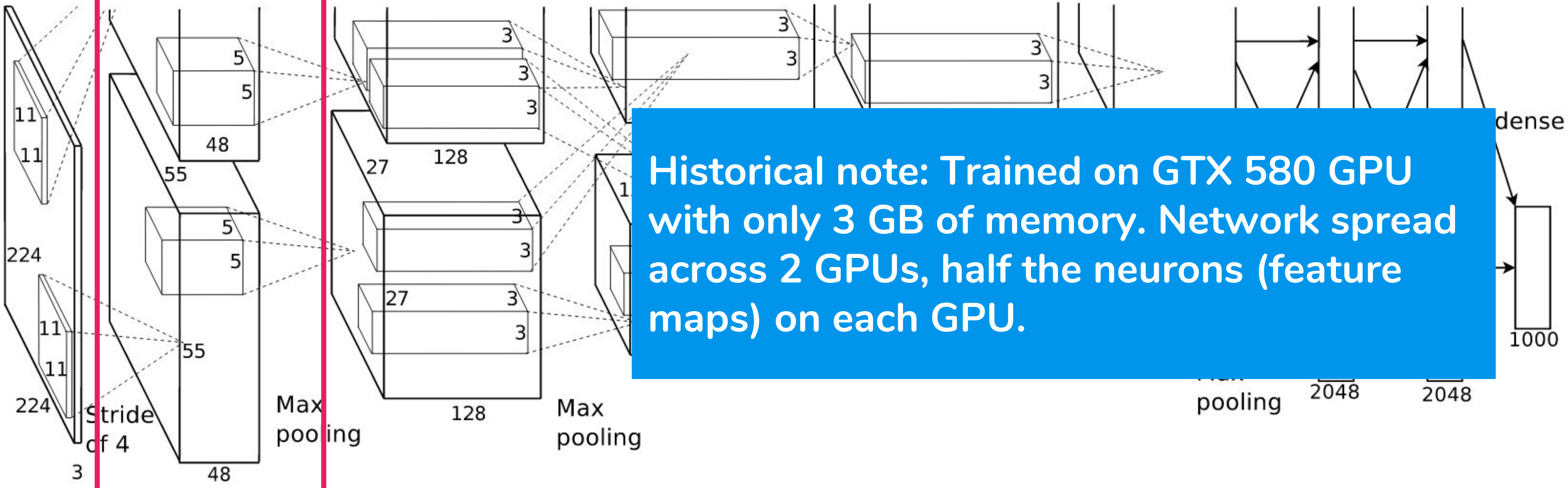
[227x227x3] INPUT
[55x55x96] **CONV1**: 96 11x11 filters at stride 4, pad 0
[27x27x96] **MAX POOL1**: 3x3 filters at stride 2
[27x27x96] **NORM1**: Normalization layer
[27x27x256] **CONV2**: 256 5x5 filters at stride 1, pad 2
[13x13x256] **MAX POOL2**: 3x3 filters at stride 2
[13x13x256] **NORM2**: Normalization 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)

AlexNet [Krizhevsky et al., 2012]



[55x55x96] **CONV1** → [55x55x48] x 2

AlexNet [Krizhevsky et al., 2012]



Historical note: Trained on GTX 580 GPU with only 3 GB of memory. Network spread across 2 GPUs, half the neurons (feature maps) on each GPU.

[55x55x96] **CONV1** → [55x55x48] x 2

AlexNet [Krizhevsky et al., 2012]

Details:

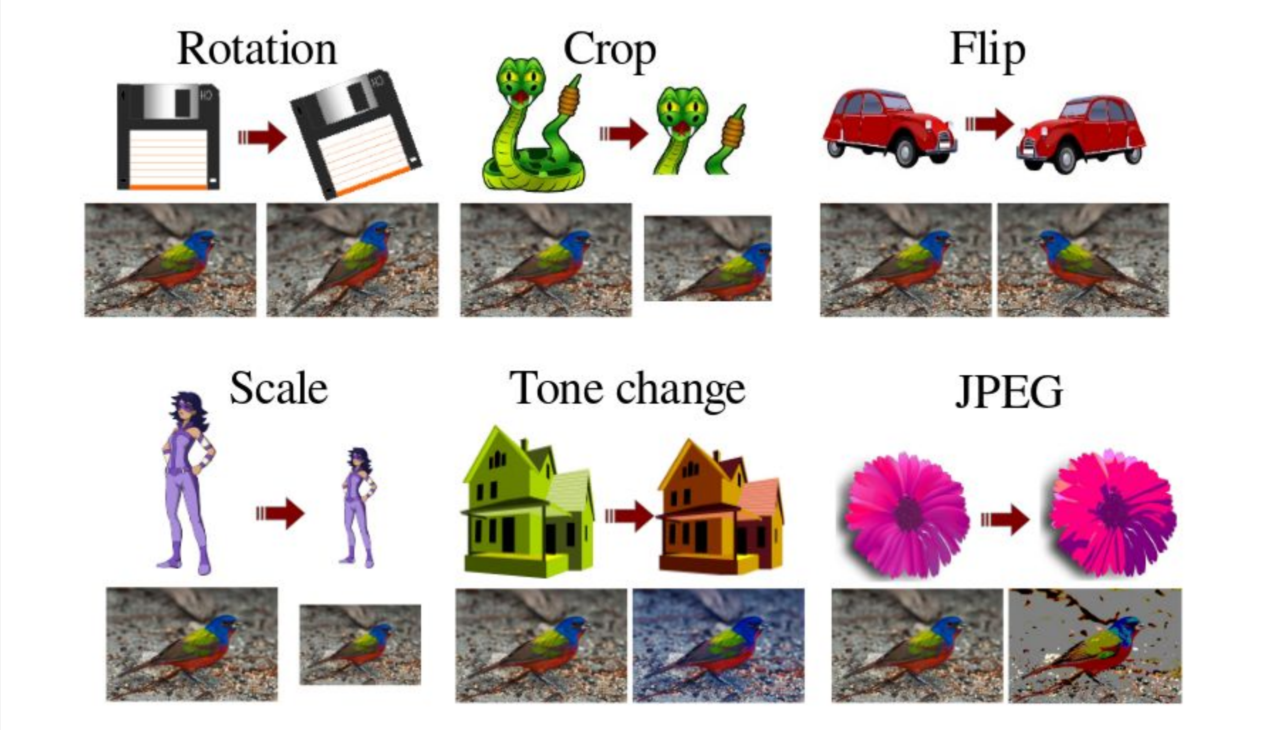
- 60 million learned parameters
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- 7 CNN ensemble: 18.2% -> 15.4%
- 5-6 days to train on 2 GTX 580 3GB GPUs

AlexNet [Krizhevsky et al., 2012]

Details:

- 60 million learned parameters
- first use of ReLU
- used Norm layers (not common anymore)
- heavy **data augmentation**
- dropout 0.5
- batch size 128
- 7 CNN ensemble: 18.2% -> 15.4%
- 5-6 days to train on 2 GTX 580 3GB GPUs

Data Augmentation



Credit: Plauin et al., Transformation Pursuit for Image Classification, CVPR 2014.

Data Augmentation

Get creative!

- Random mix/combinations of :
  - Translation
  - Rotation
  - Stretching
  - Shearing
  - Lens distortions
  - Go crazy!

Data Augmentation

- Simple to implement, use it
- Especially useful for small datasets
- Apply on training and testing

AlexNet [Krizhevsky et al., 2012]

Details:

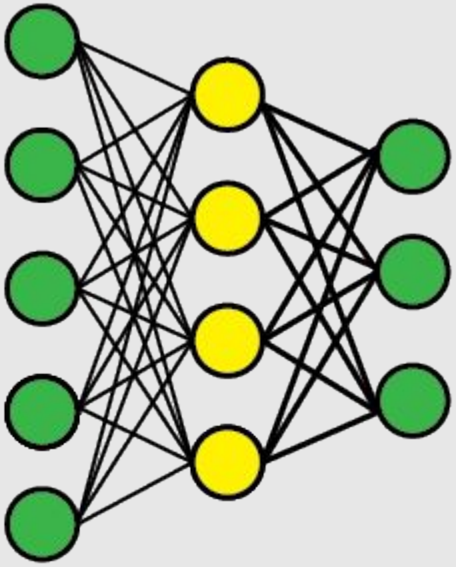
- 60 million learned parameters
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- **dropout** 0.5
- batch size 128
- 7 CNN ensemble: 18.2% -> 15.4%

Dropout [Hinton et al., 2012]

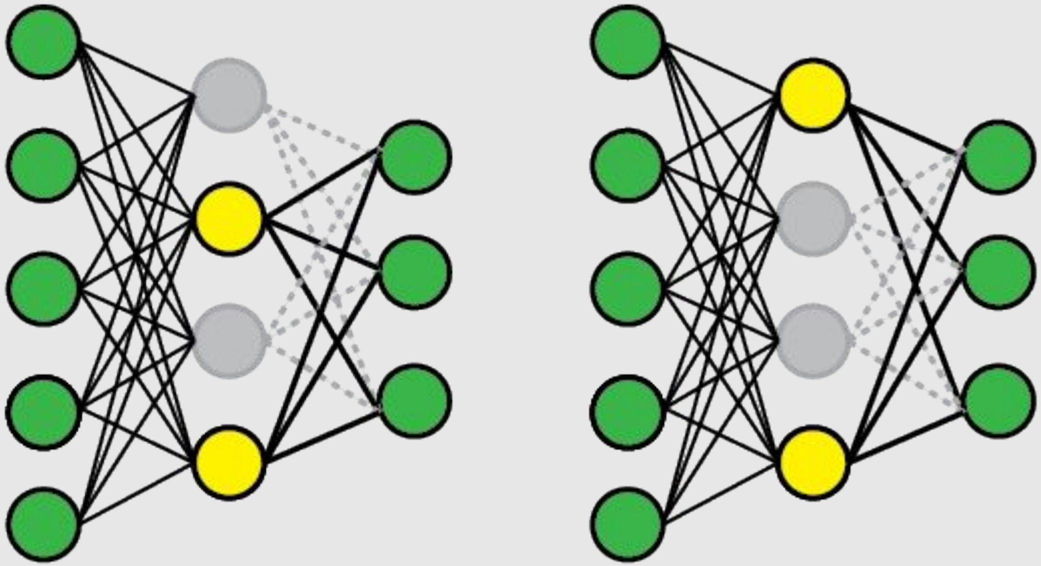
- Dropout is a radically different technique **for regularization**.
- Dropout doesn't rely on modifying the cost function.
  Instead, **in dropout we modify the network itself**.

[Srivastava et al., 2014] Dropout: A Simple Way to Prevent Neural Networks from Overfitting
https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf

Dropout [Hinton *et al.*, 2012]

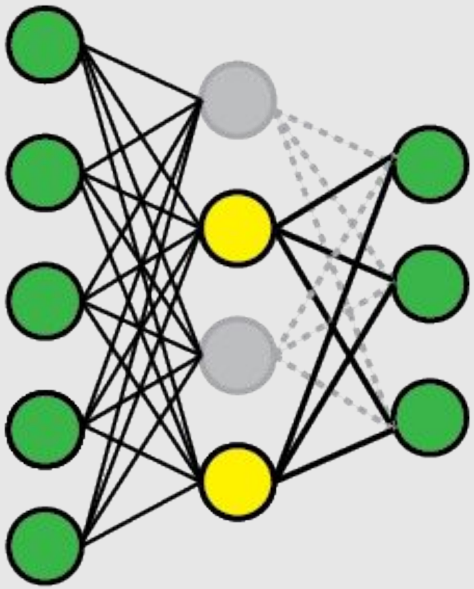


Standard Network

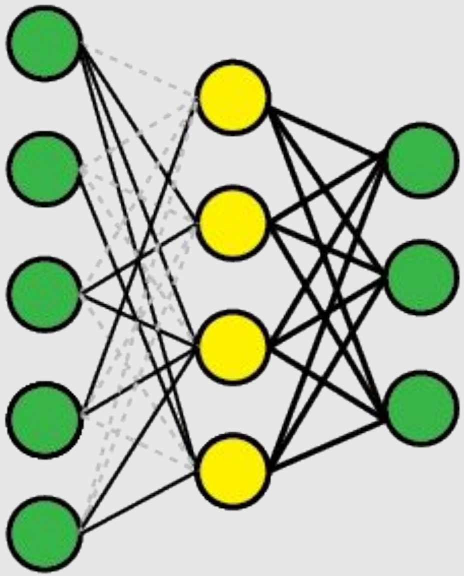


After applying dropout

Dropout [Hinton *et al.*, 2012] *vs.* DropConnect [Wan *et al.*, 2013]



Dropout



DropConnect

AlexNet [Krizhevsky et al., 2012]

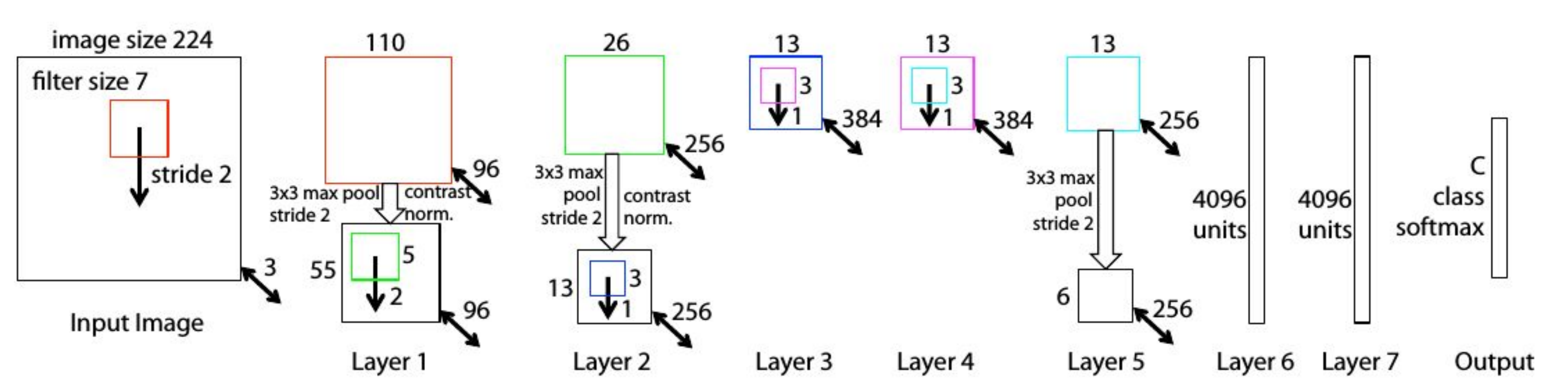
Details:

- 60 million learned parameters
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- 7 CNN **ensemble**: 18.2% -> 15.4%

CNNs Architectures

- **LeNet** by Yann LeCun, Léon Bottou & Yoshua Bengio (1998)
- **AlexNet** by Alex Krizhevsky, Ilya Sutskever & Geoff Hinton (2012)
- **ZF Net** by Matthew Zeiler & Rob Fergus (2013)
- **VGGNet** by Karen Simonyan & Andrew Zisserman (2014)
- **GoogLeNet** by Szegedy et al. (2014)
- **ResNet** by Kaiming He et al. (2015)

ZFNet [Zeiler & Fergus, 2013]



AlexNet but:

CONV1: change from (11x11 stride 4) to (7x7 stride 2)

CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512

CNNs Architectures

- **LeNet** by Yann LeCun, Léon Bottou & Yoshua Bengio (1998)
- **AlexNet** by Alex Krizhevsky, Ilya Sutskever & Geoff Hinton (2012)
- **ZF Net** by Matthew Zeiler & Rob Fergus (2013)
- **VGGNet** by Karen Simonyan & Andrew Zisserman (2014)
- **GoogLeNet** by Szegedy et al. (2014)
- **ResNet** by Kaiming He et al. (2015)

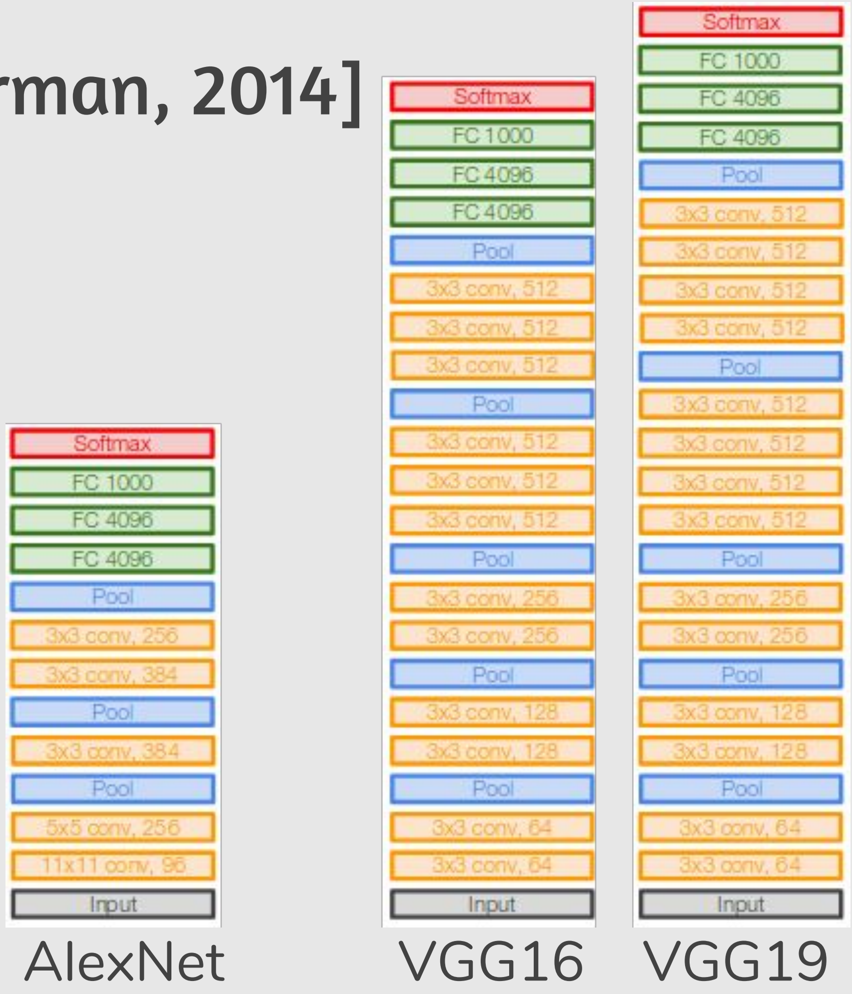
VGGNet [Simonyan & Zisserman, 2014]

Small filters, Deeper networks

8 layers (AlexNet)
16 - 19 layers (VGG16Net)

Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

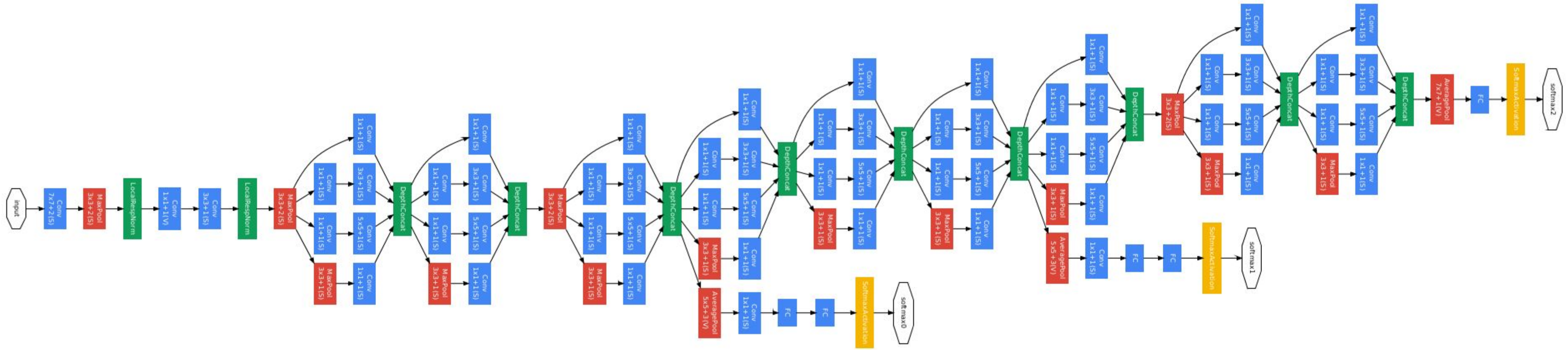
11.7% in ILSVRC'13 (ZFNet)
7.3% in ILSVRC'14



CNNs Architectures

- **LeNet** by Yann LeCun, Léon Bottou & Yoshua Bengio (1998)
- **AlexNet** by Alex Krizhevsky, Ilya Sutskever & Geoff Hinton (2012)
- **ZF Net** by Matthew Zeiler & Rob Fergus (2013)
- **VGGNet** by Karen Simonyan & Andrew Zisserman (2014)
- **GoogLeNet** by Szegedy et al. (2014)
- **ResNet** by Kaiming He et al. (2015)

GoogLeNet [Szegedy et al., 2014]



11.7% in ILSVRC’13 (ZFNet)

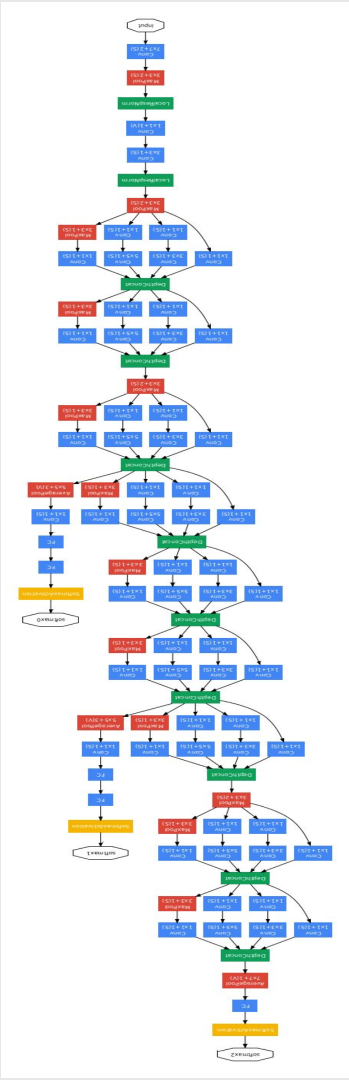
7.3% in ILSVRC’14 (VGGNet)

6.7% in ILSVRC’14 (GoogLeNet)

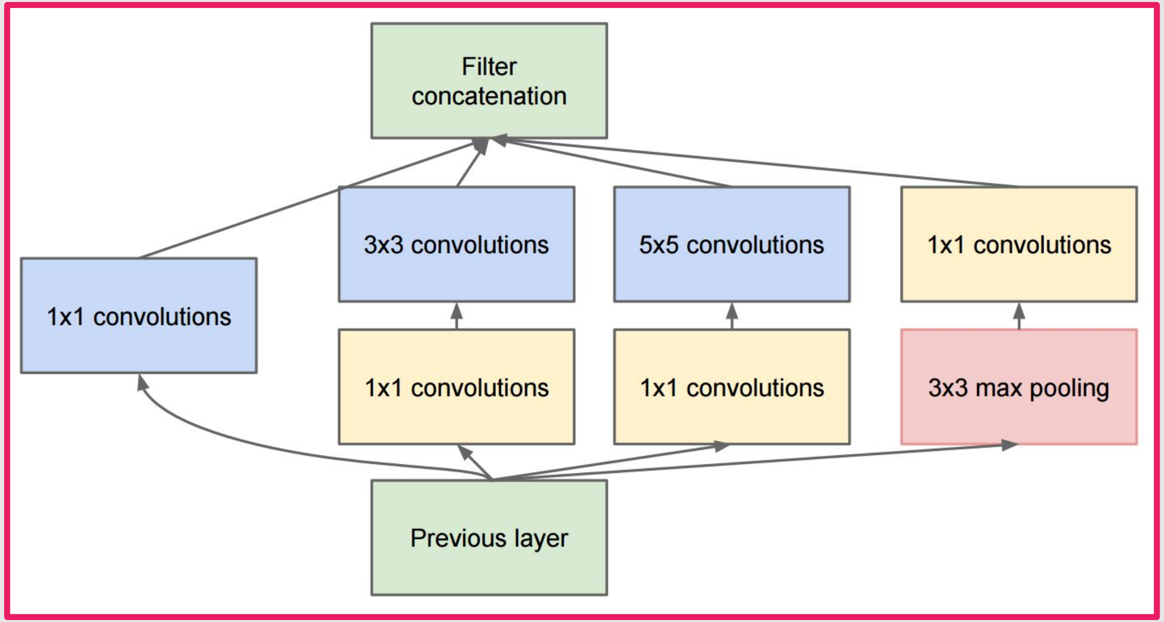
GoogLeNet [Szegedy et al., 2014]

Deeper networks, with computational efficiency

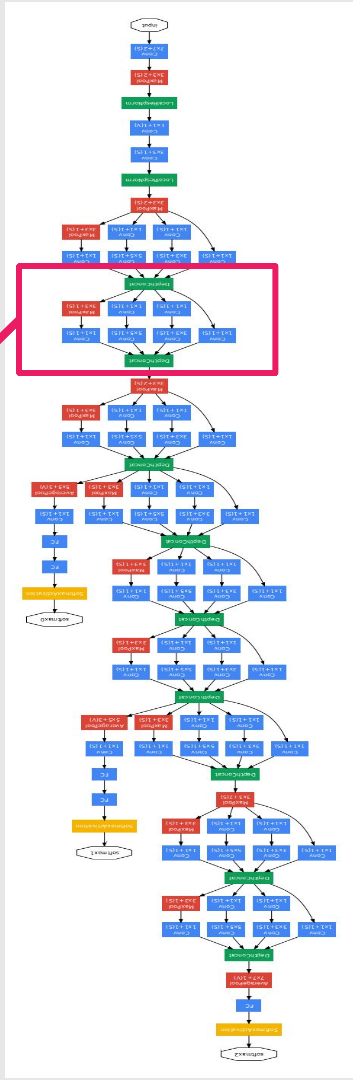
- 22 layers
- Efficient “Inception” module
- No FC layers
- Only 5 million parameters!
  12x less than AlexNet



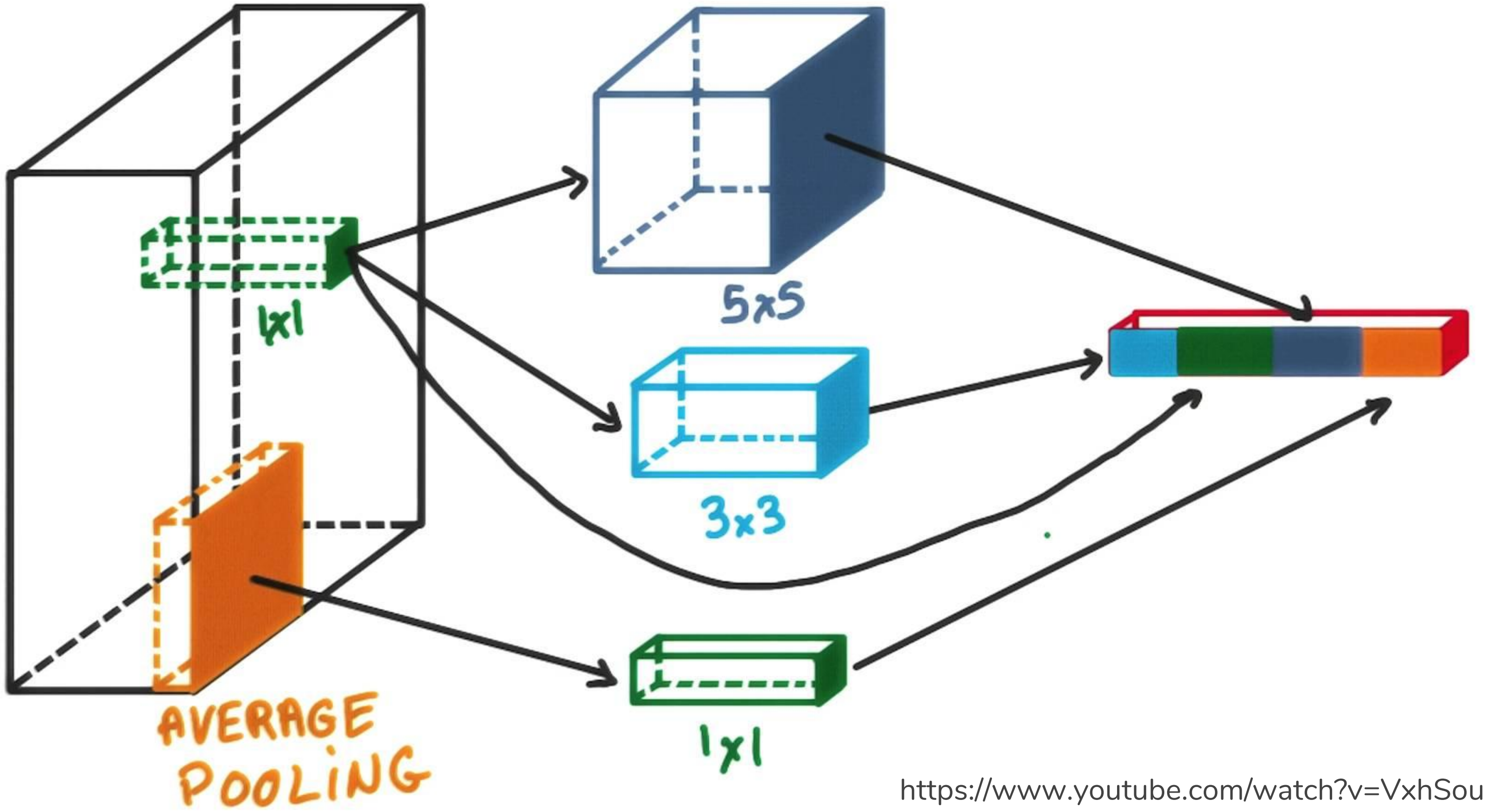
GoogLeNet [Szegedy et al., 2014]



Inception Module

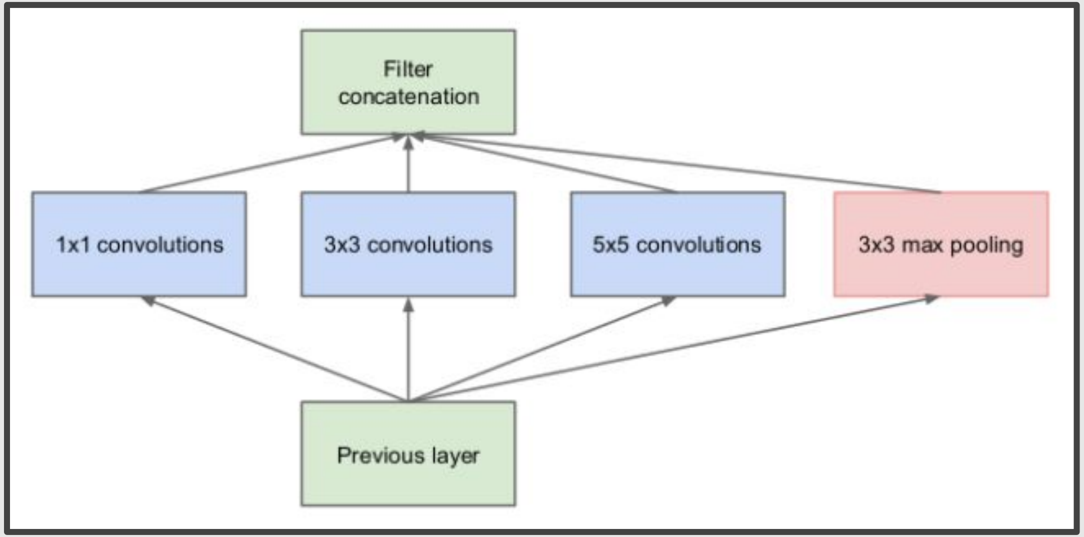


INCEPTION MODULES



https://www.youtube.com/watch?v=VxhSouuSZDY

GoogLeNet [Szegedy et al., 2014]



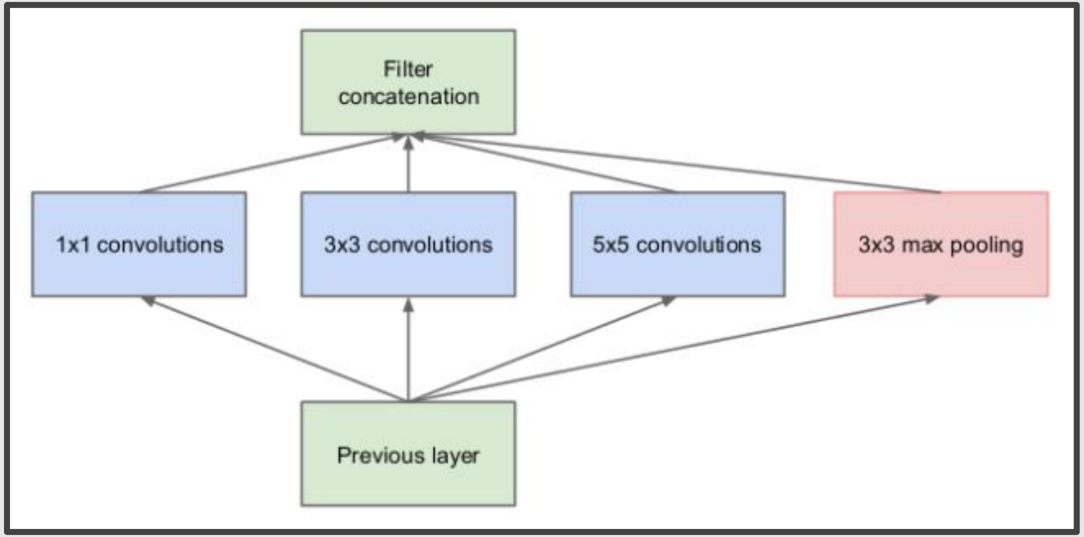
Naive Inception Module

Apply parallel filters on the input from previous layer:

- Multiple receptive field sizes for convolution (1x1,3x3,5x5)
- Pooling operation (3x3)

Concatenate all filter outputs together depth-wise

GoogLeNet [Szegedy et al., 2014]



Naive Inception Module

Apply parallel filters on the input from previous layer:

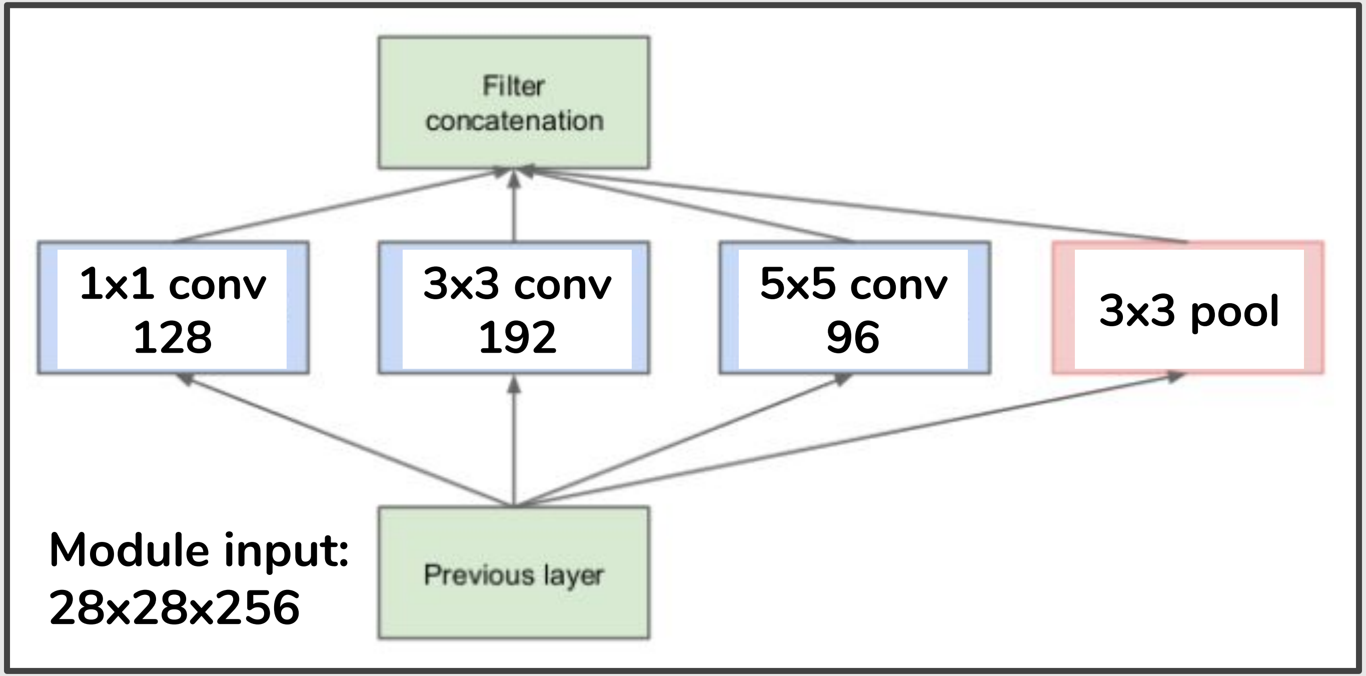
- Multiple receptive field sizes for convolution (1x1,3x3,5x5)
- Pooling operation (3x3)

Concatenate all filter outputs together depth-wise

Q: What is the problem with this?

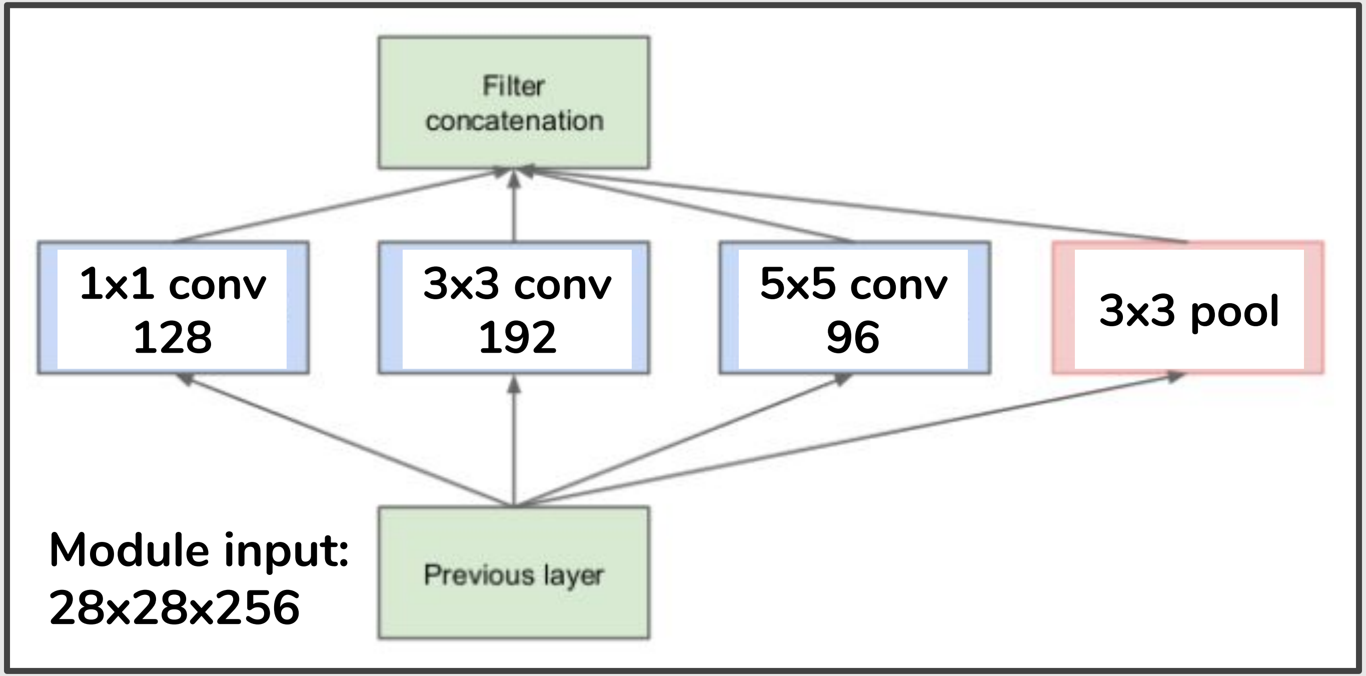
GoogLeNet [Szegedy et al., 2014]

Example:



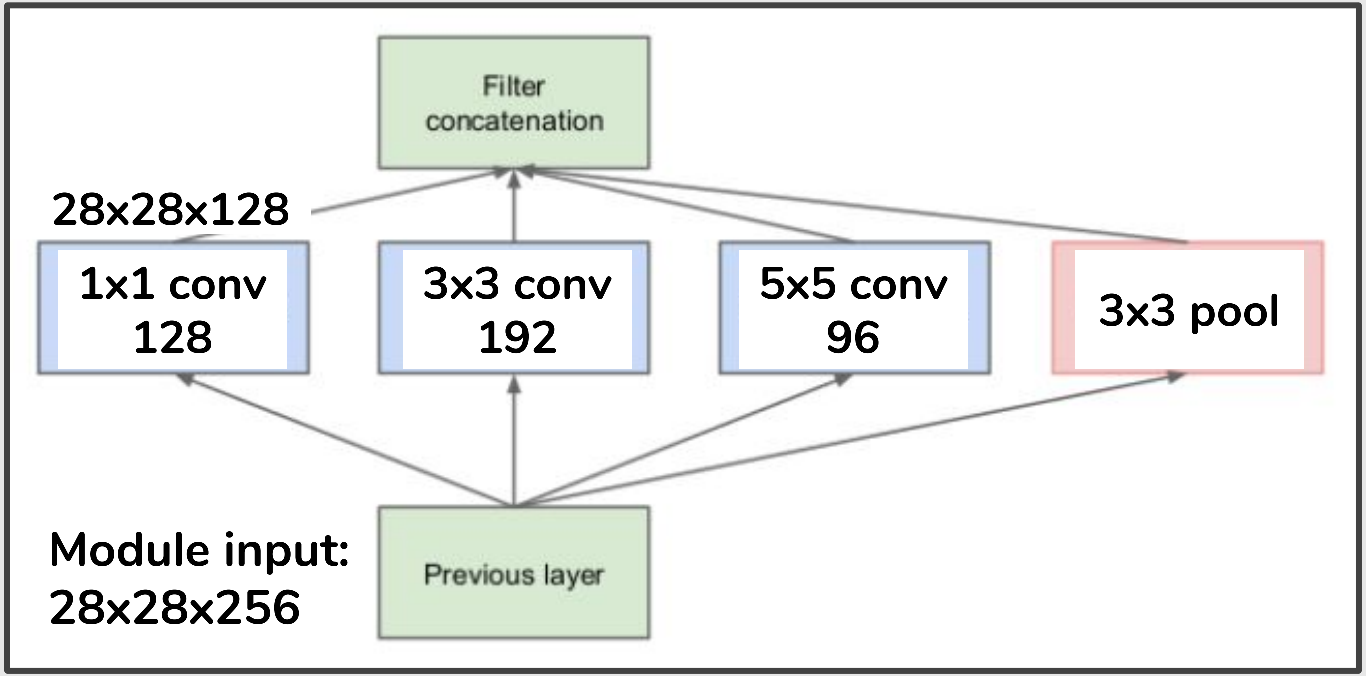
GoogLeNet [Szegedy et al., 2014]

Example: What is the output size of the **1x1 conv, with 128 filters**?



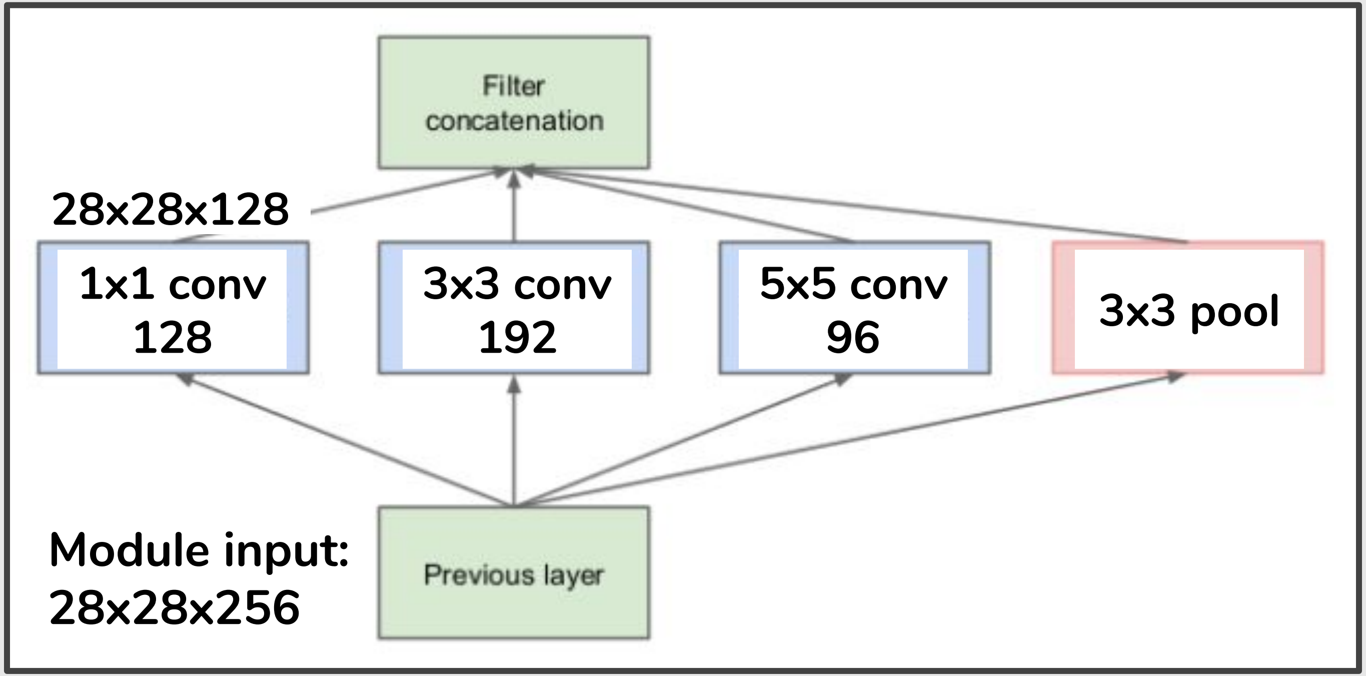
GoogLeNet [Szegedy et al., 2014]

Example: What is the output size of the 1x1 conv, with 128 filters?



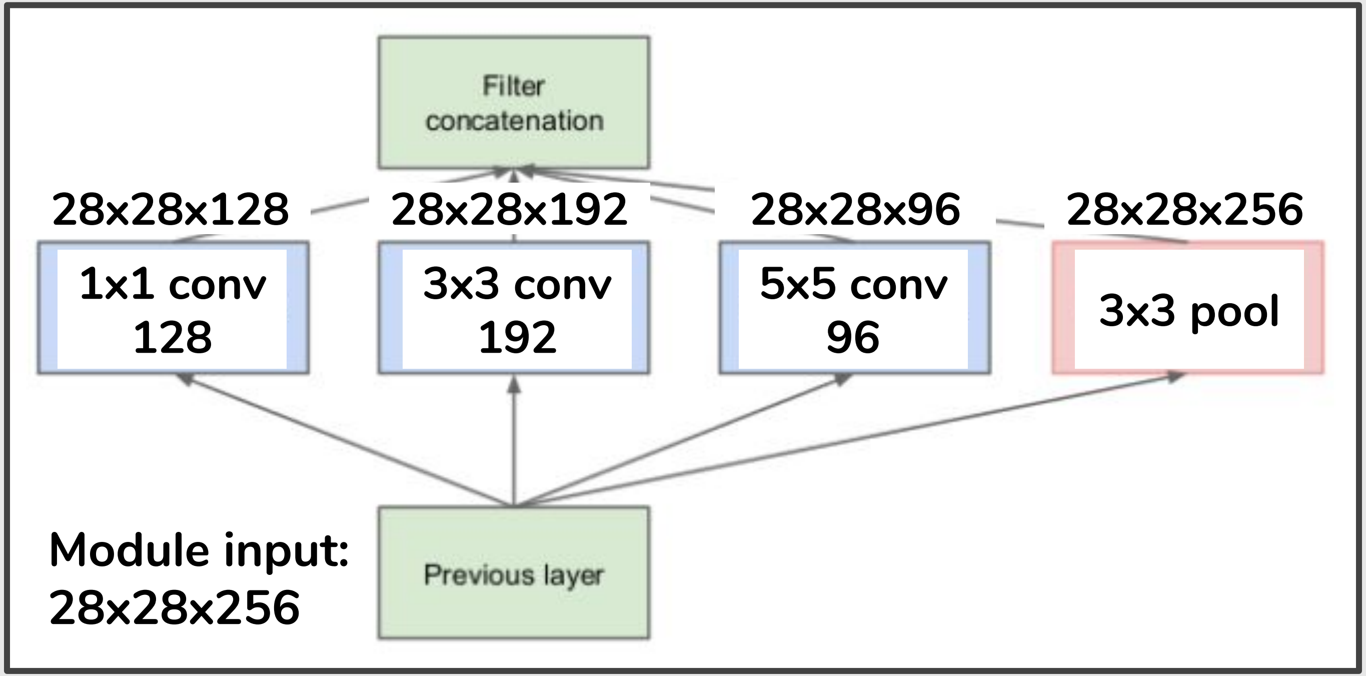
GoogLeNet [Szegedy et al., 2014]

Example: What are the output sizes of all different filter operations?



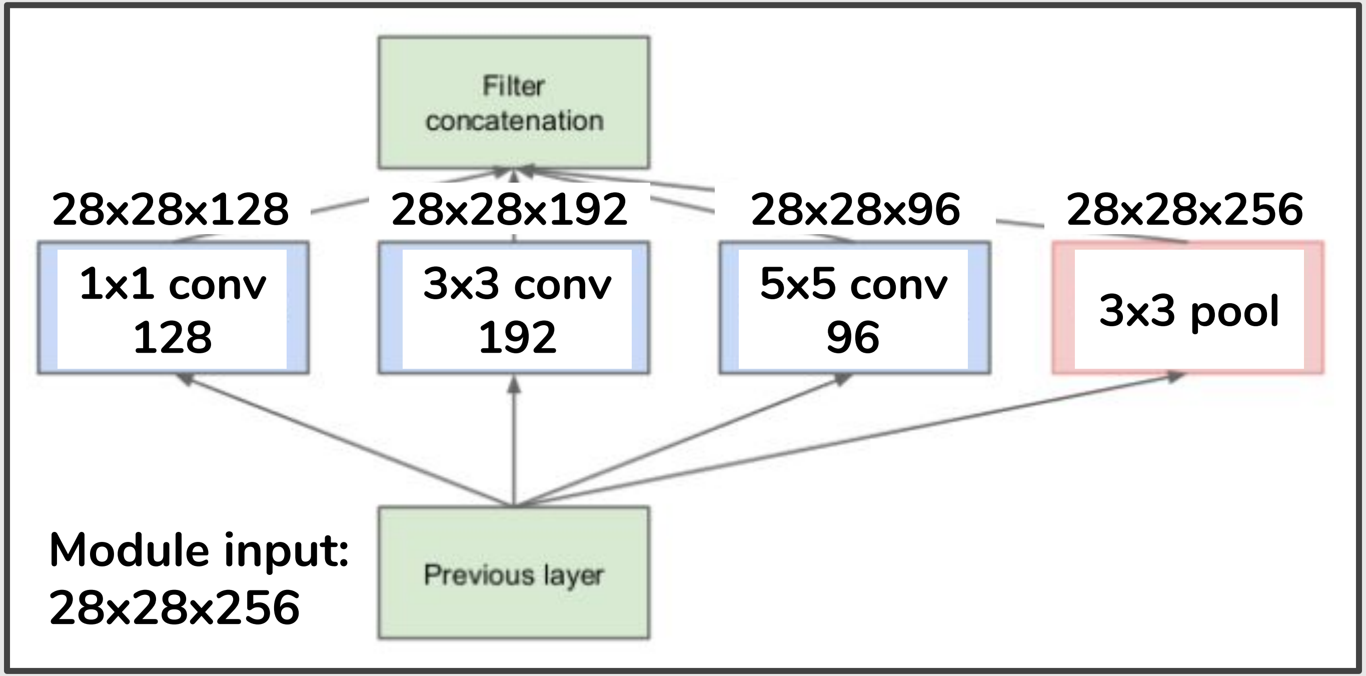
GoogLeNet [Szegedy et al., 2014]

Example: What are the output sizes of all different filter operations?



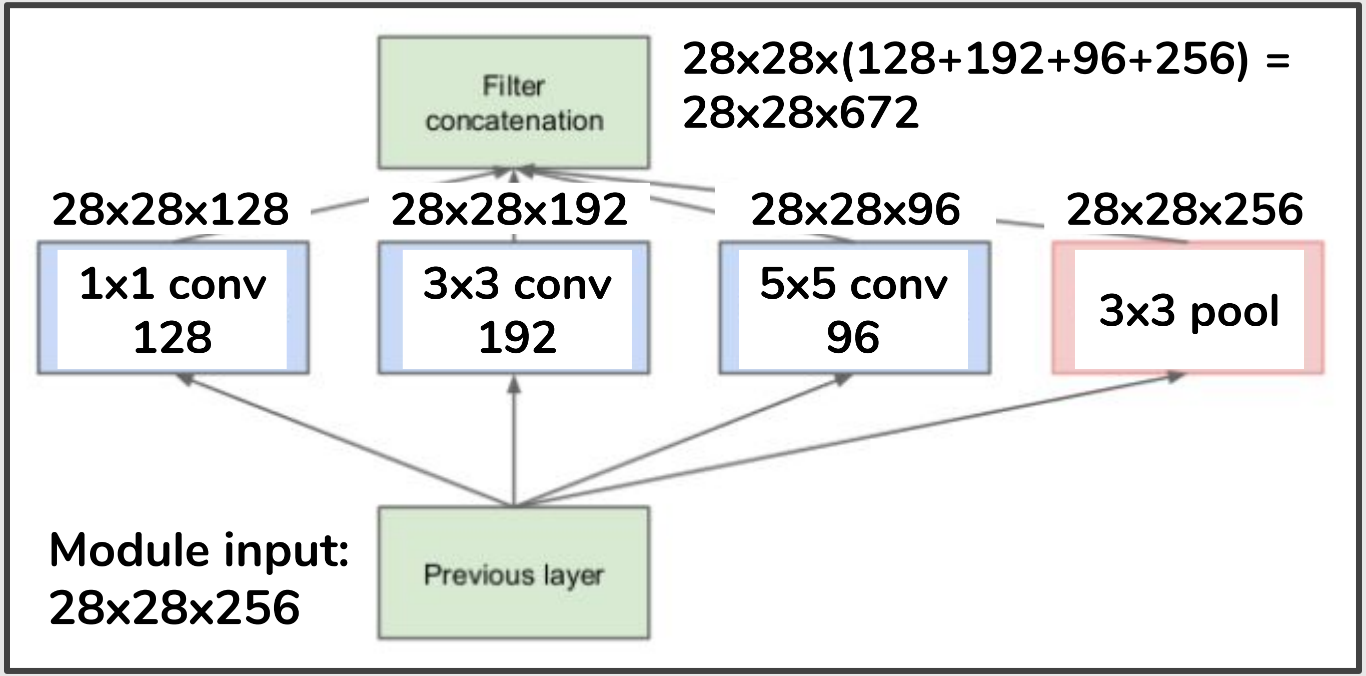
GoogLeNet [Szegedy et al., 2014]

Example: What is output size after filter concatenation?



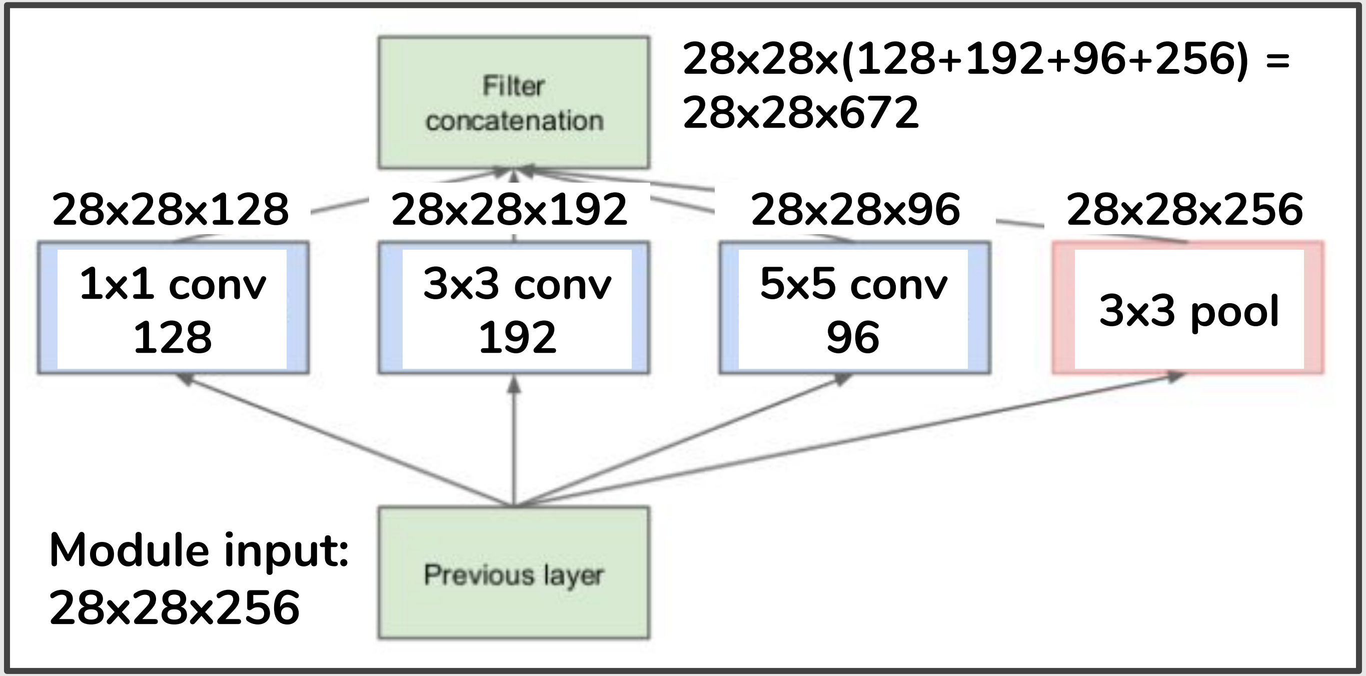
GoogLeNet [Szegedy et al., 2014]

Example: What is output size after filter concatenation?



GoogLeNet [Szegedy et al., 2014]

Example: What is output size after filter concatenation?

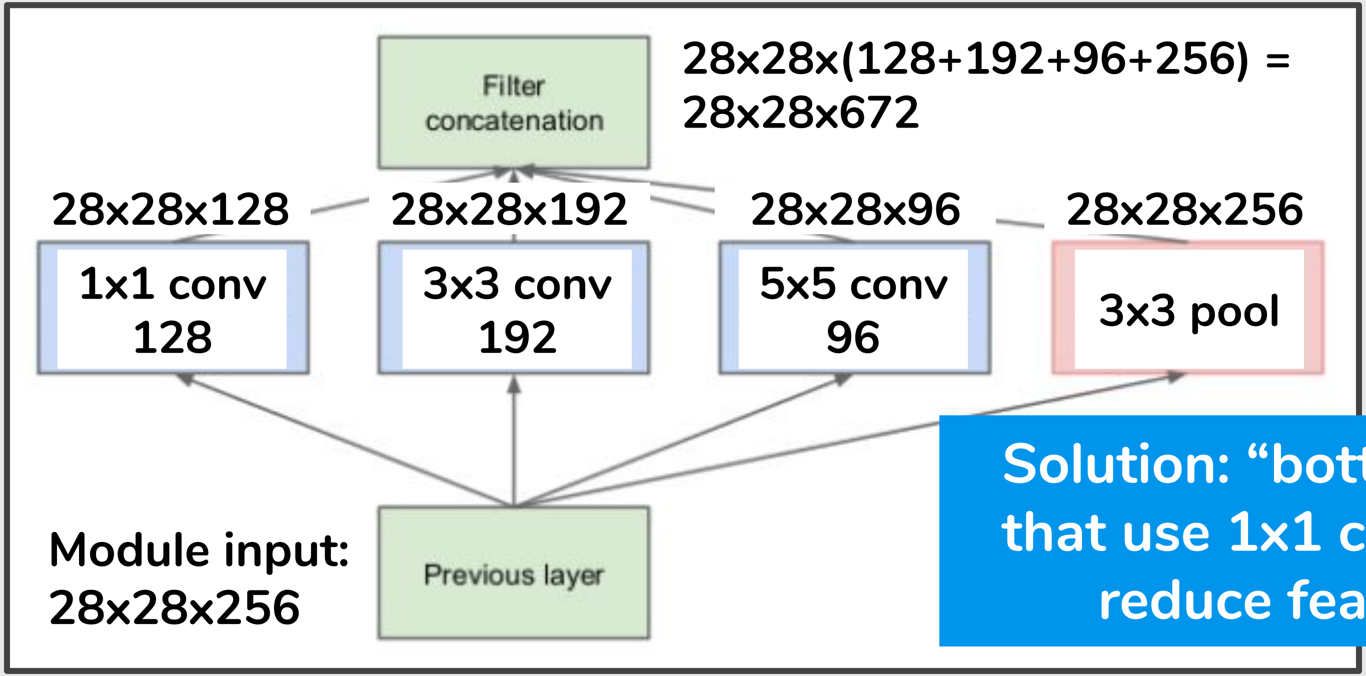


Conv Ops:

854M ops!!

GoogLeNet [Szegedy et al., 2014]

Example: What is output size after filter concatenation?

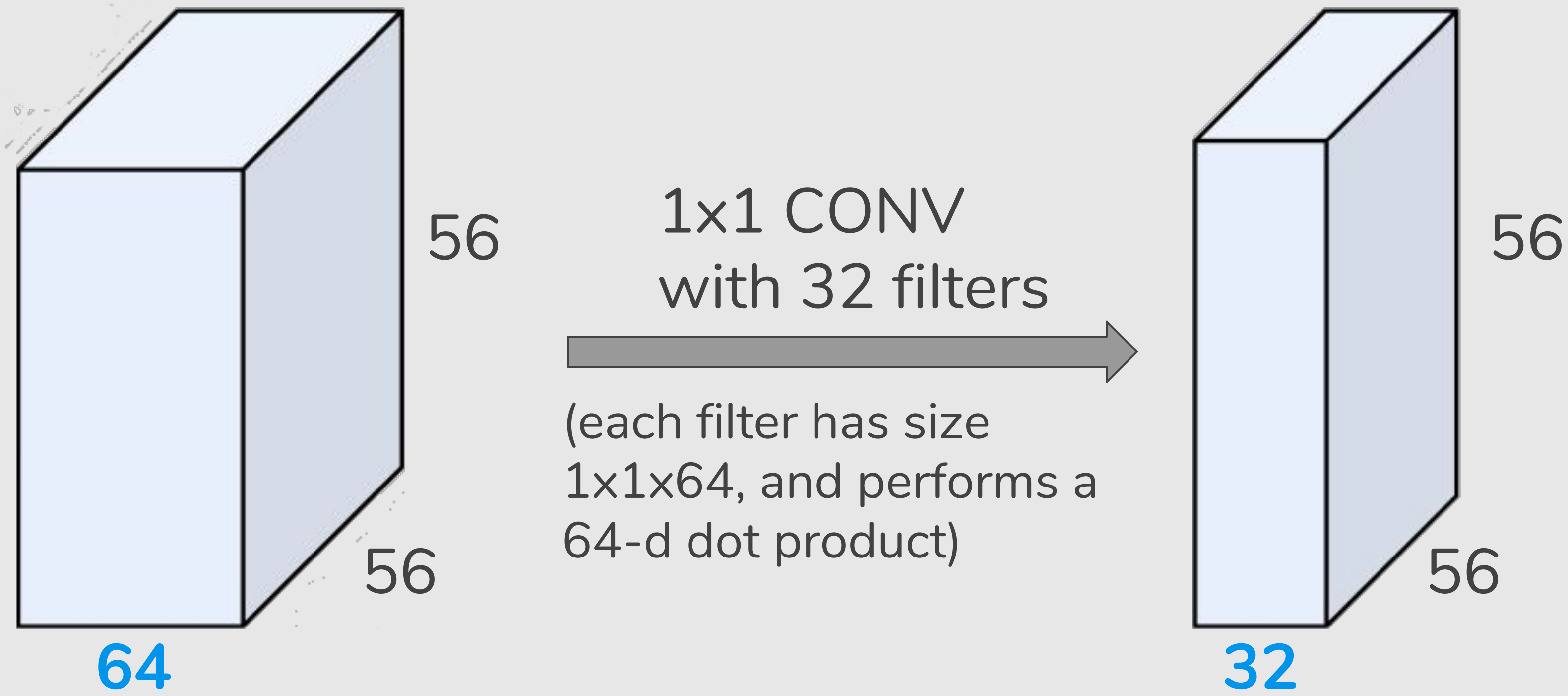


Conv Ops:
854M ops!!

Solution: "bottleneck" layers that use 1x1 convolutions to reduce feature depth

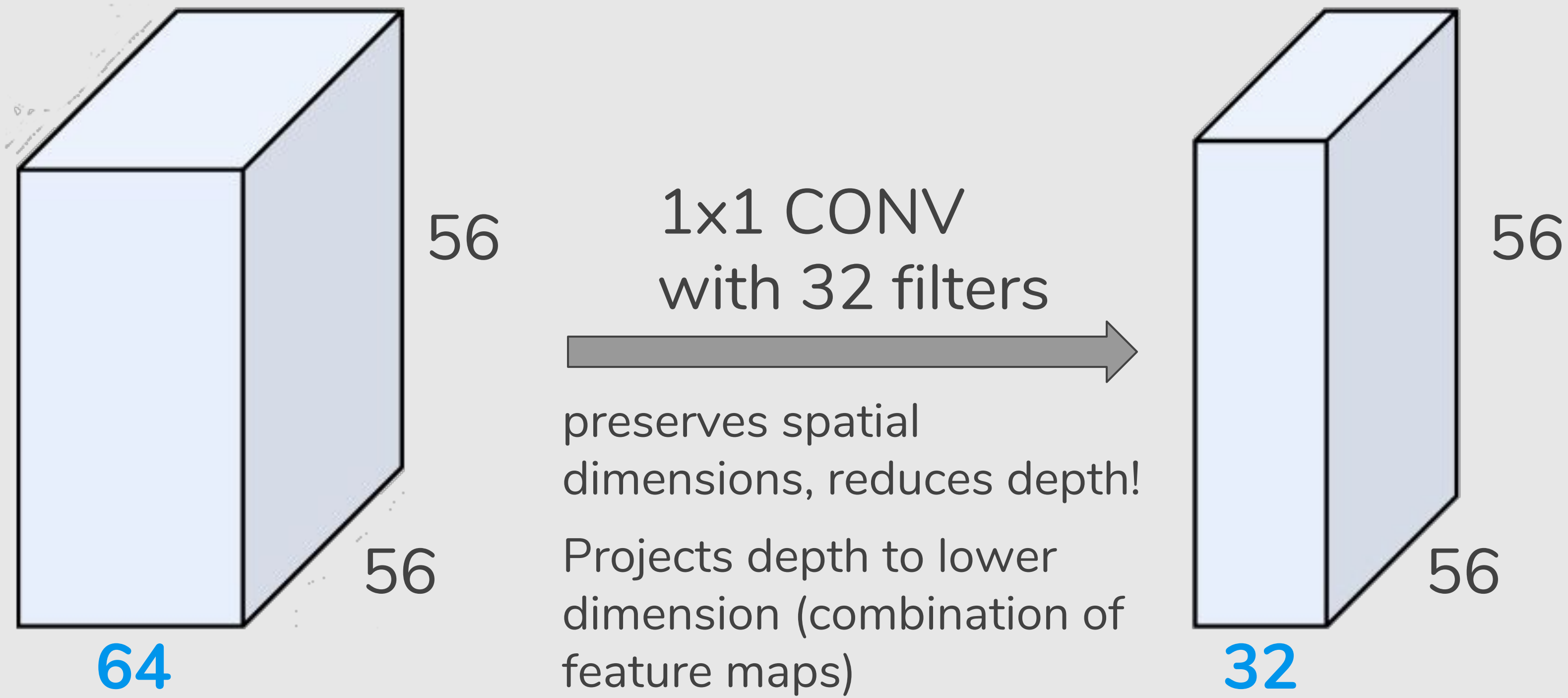
GoogLeNet [Szegedy et al., 2014]

Reminder: 1x1 convolutions



GoogLeNet [Szegedy et al., 2014]

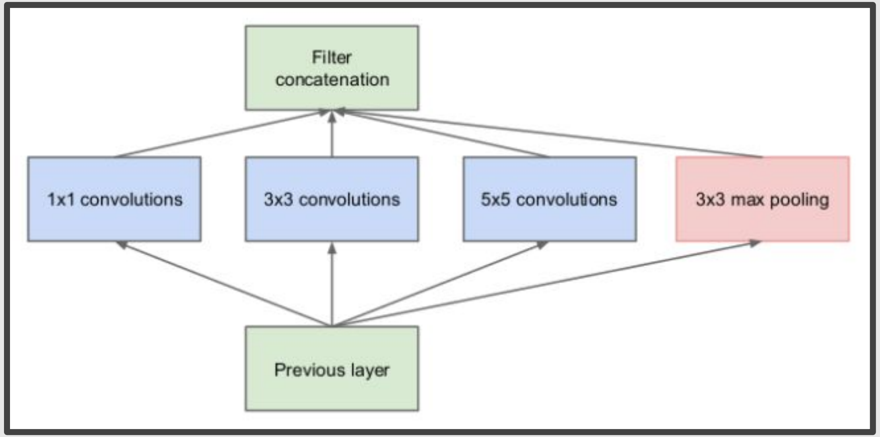
Reminder: 1x1 convolutions



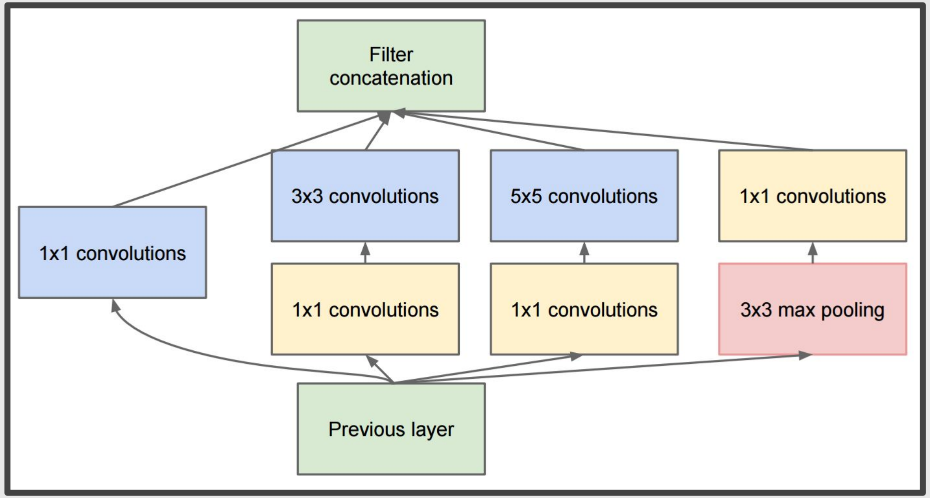
preserves spatial dimensions, reduces depth!

Projects depth to lower dimension (combination of feature maps)

GoogLeNet [Szegedy et al., 2014]



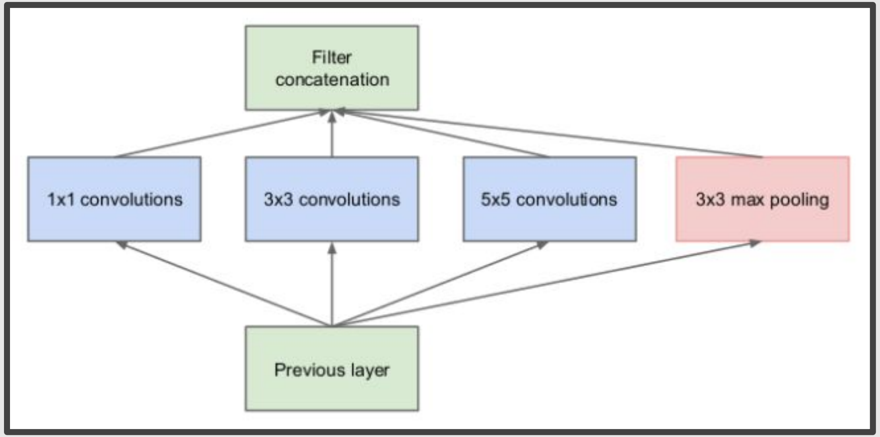
Naive Inception Module



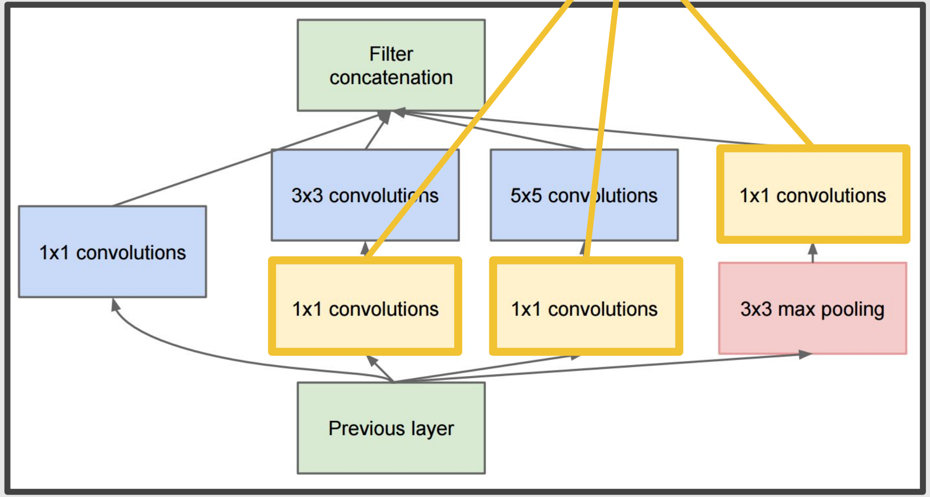
Inception Module

GoogLeNet [Szegedy et al., 2014]

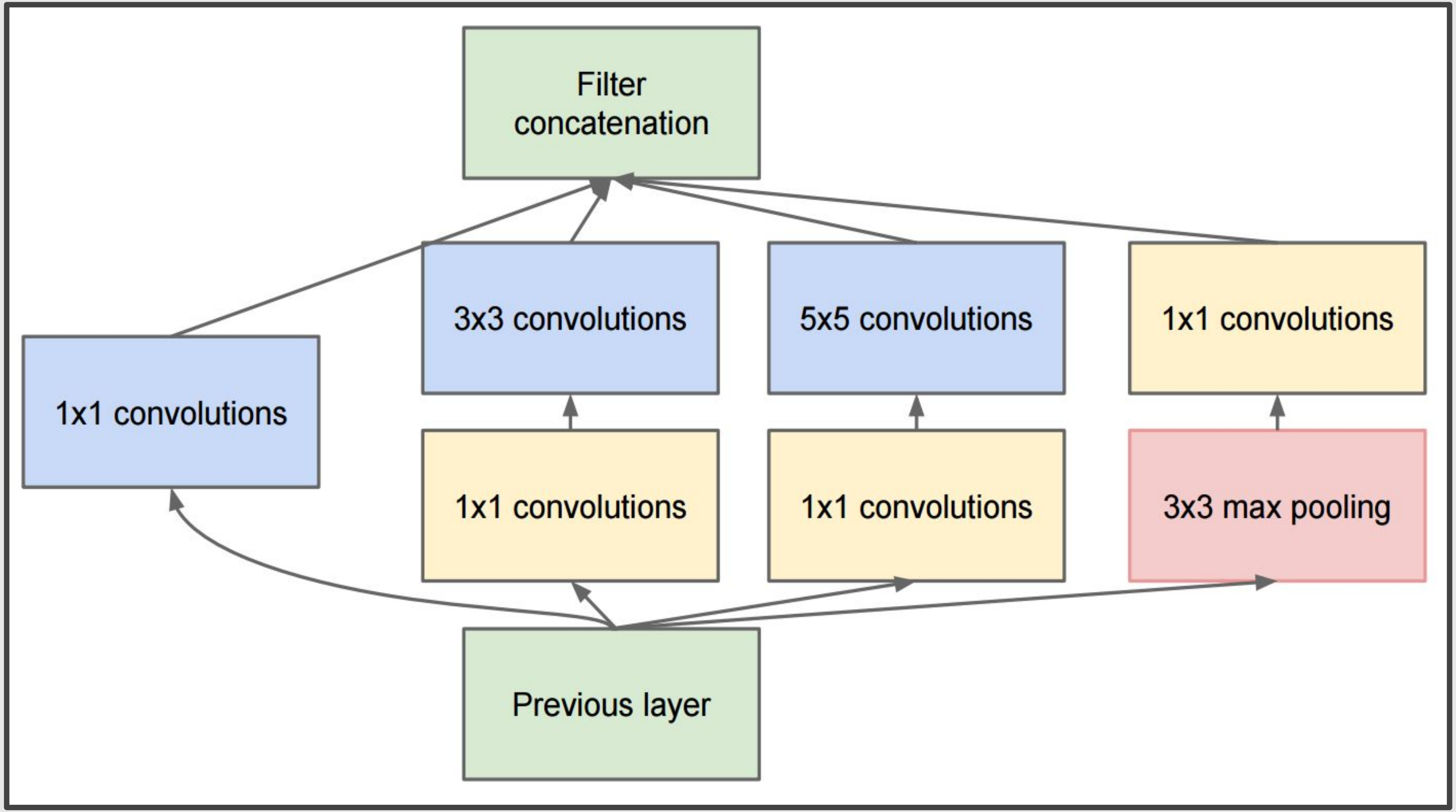
1x1 conv
“bottleneck” layers



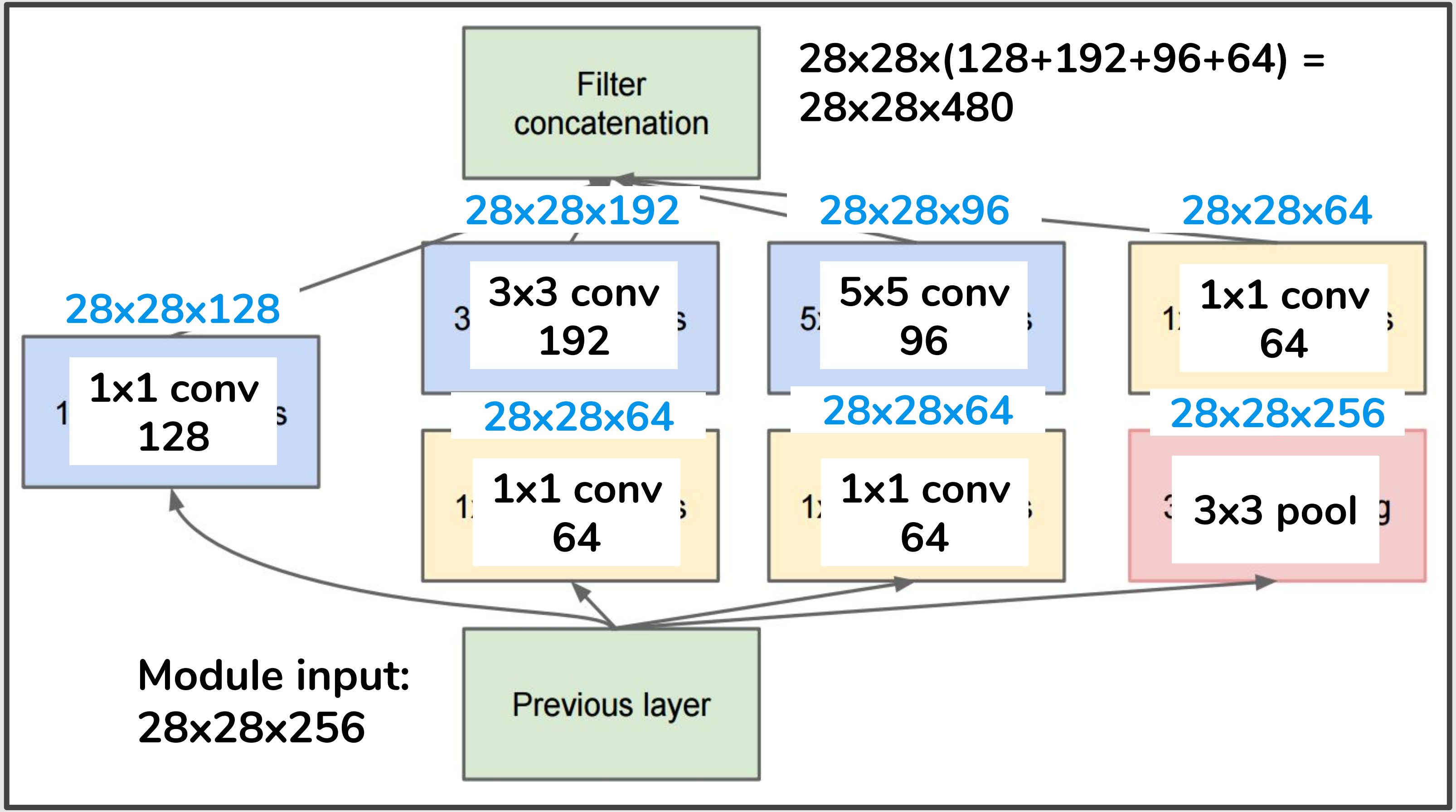
Naive Inception Module



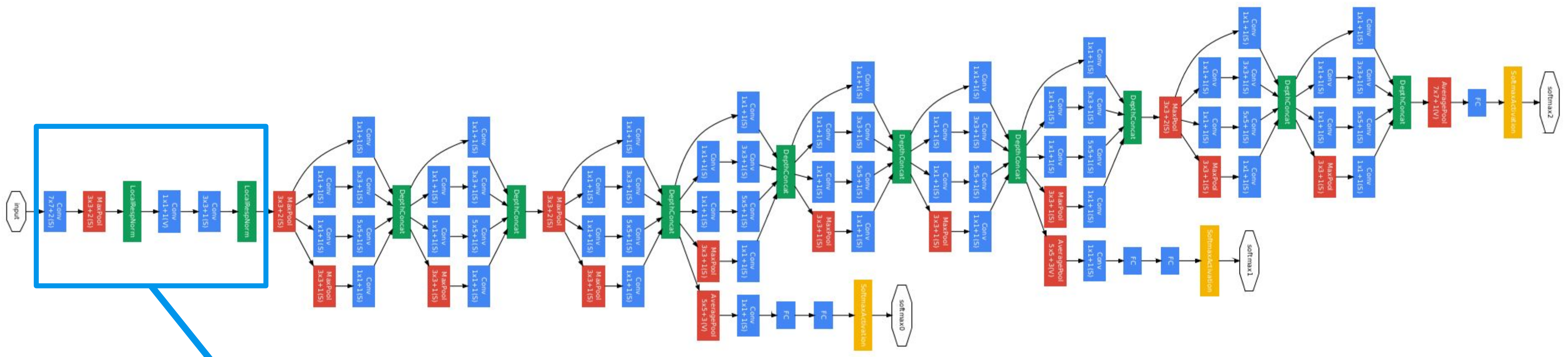
Inception Module


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

Conv Ops: 358M ops

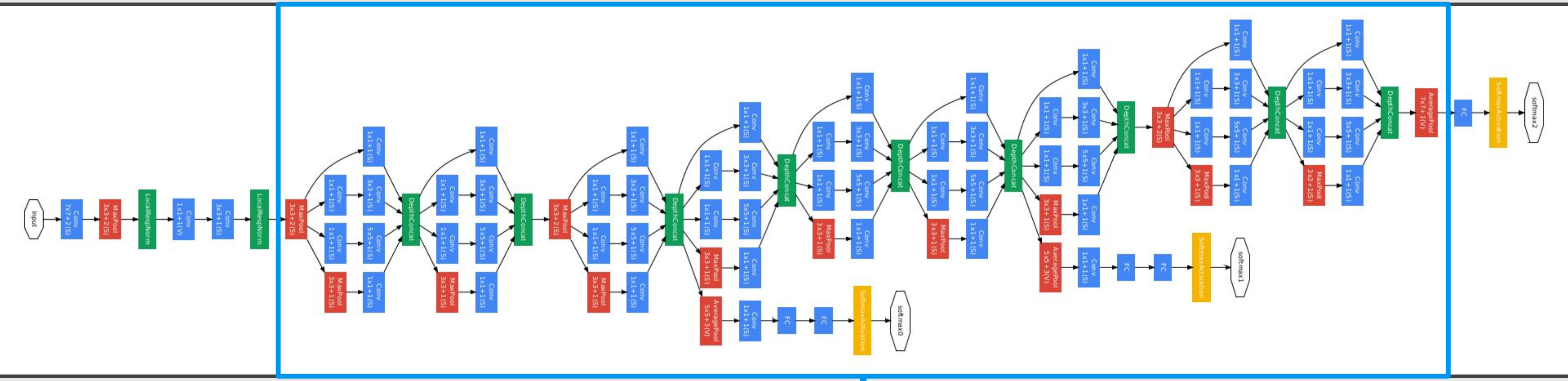


GoogLeNet [Szegedy et al., 2014]



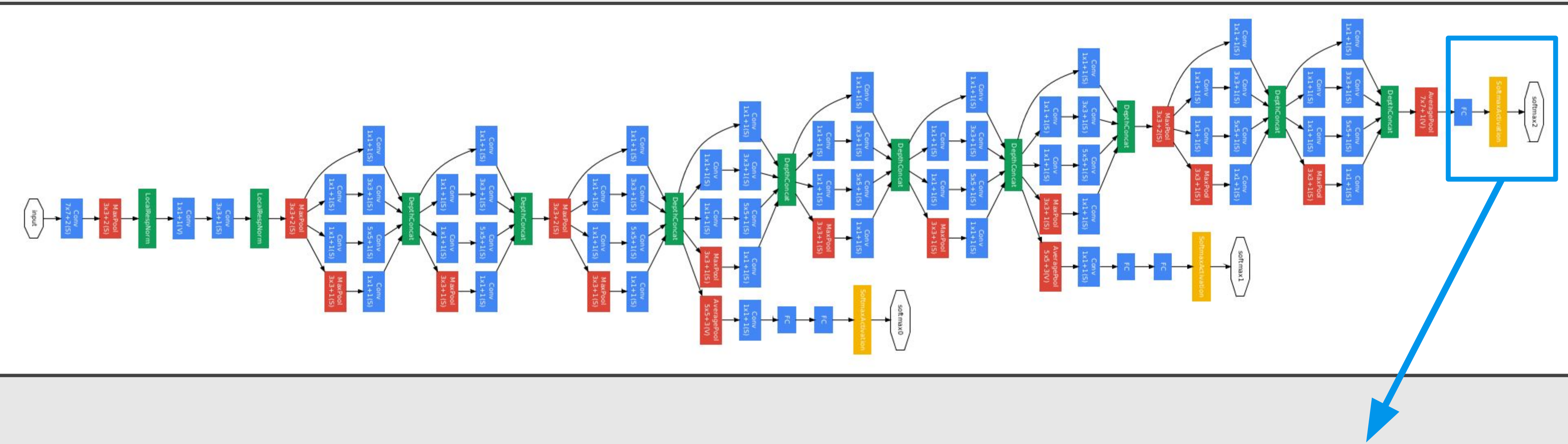
Conv-Pool 2x Conv-Pool

GoogLeNet [Szegedy et al., 2014]



Stacked Inception Modules

GoogLeNet [Szegedy et al., 2014]

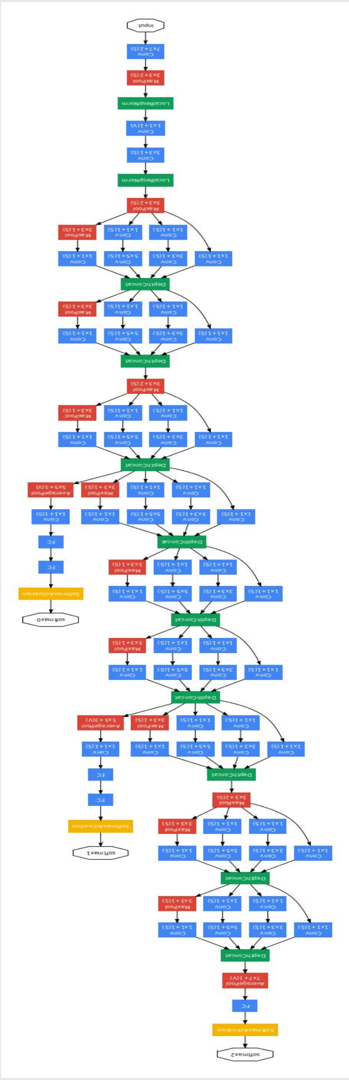


Classifier Output

GoogLeNet [Szegedy et al., 2014]

Deeper networks, with computational efficiency

- 22 layers
- Efficient "Inception" module
- No FC layers
- Only 5 million parameters!
  12x less than AlexNet



References

Machine Learning Books

- Hands-On Machine Learning with Scikit-Learn and TensorFlow, Chap. 11 & 13

Machine Learning Courses

- https://www.coursera.org/learn/neural-networks
- “The 3 popular courses on Deep Learning”: https://medium.com/towards-data-science/the-3-popular-courses-for-deeplearning-ai-ac37d4433bd