Vision Backbone Session

Basics of CNNs, as well as some important models used in image processing.

Deep Learning Summer school:
Week 2 Session 3

By:
Dharani Govindasamy
Sailesh Kumar T

Convolutional Neural Networks (CNNs)

In general, CNNs are used for working with image data. This includes:

- Image classification
- Image captioning
- Image detection

...and other applications too.

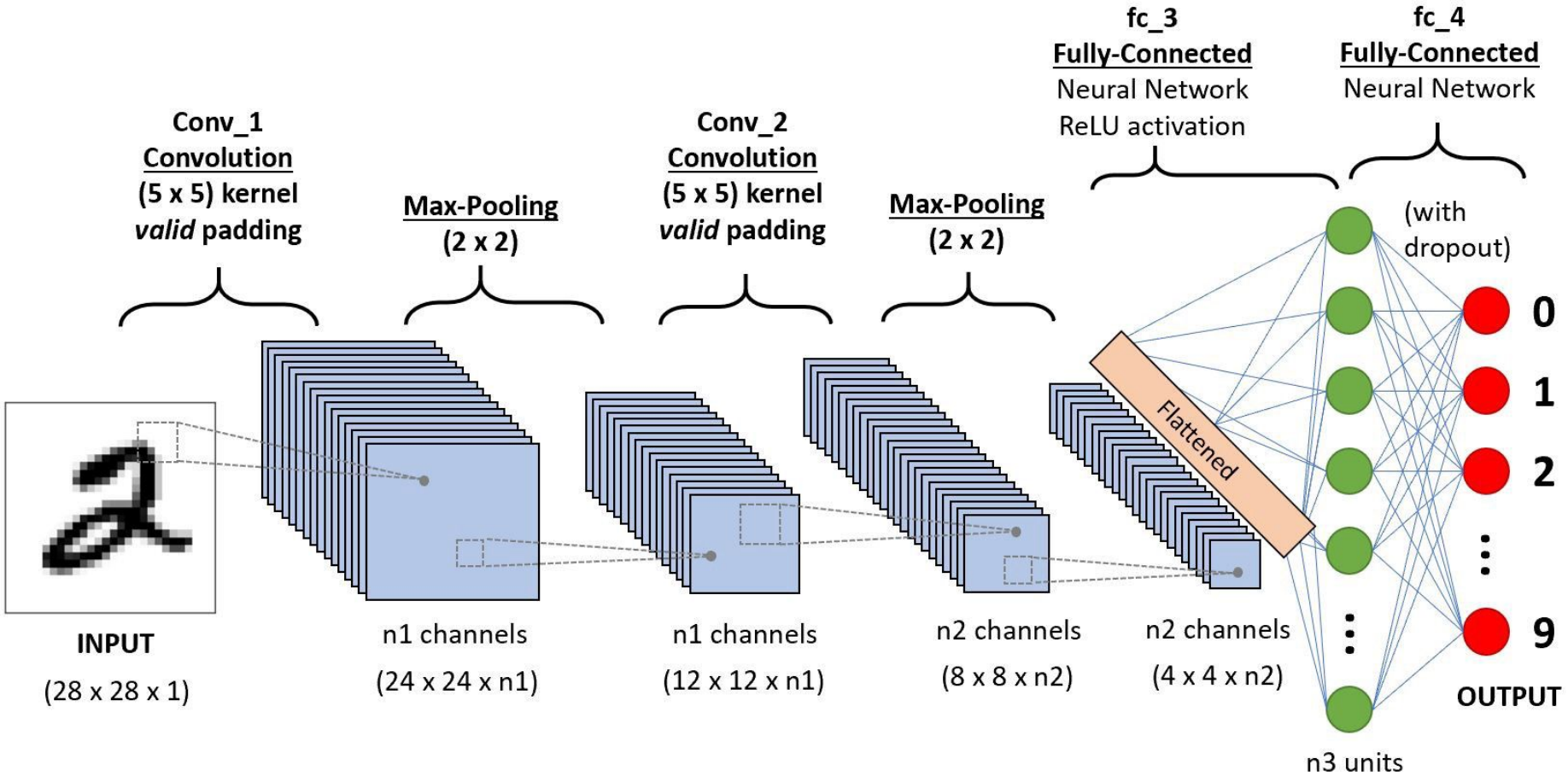
First, we will be looking at the components of a CNN.

What is a convolutional network?

A CNN is a network that consists of multiple layers, which are fundamentally of three types:

- Convolutional layers
- Pooling layers
- Fully connected layers

We will be discussing each of them in detail.

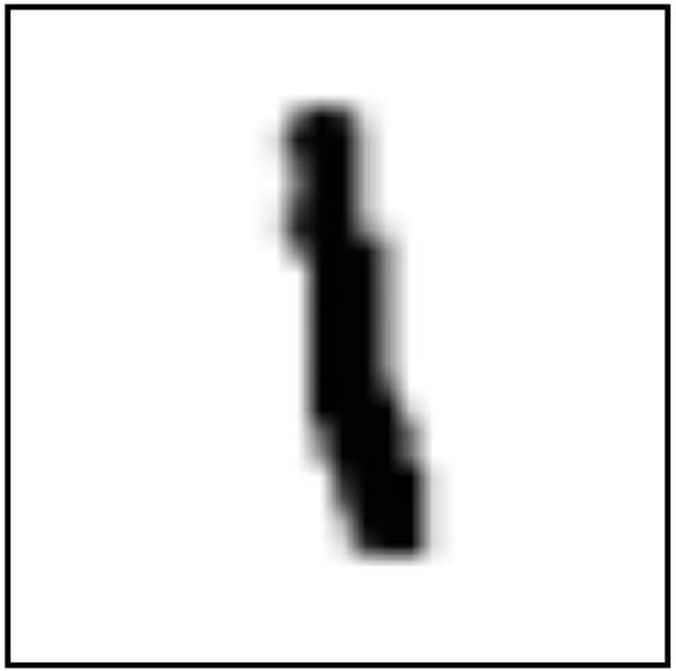

Conv_1
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
Conv_2
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
fc_3
Fully-Connected
Neural Network
ReLU activation
fc_4
Fully-Connected
Neural Network
(with dropout)
Flattened
INPUT
(28 x 28 x 1)
n1 channels
(24 x 24 x n1)
n1 channels
(12 x 12 x n1)
n2 channels
(8 x 8 x n2)
n2 channels
(4 x 4 x n2)
n3 units
0
1
2
9
OUTPUT

How do we express image data?

- Images: expressed as arrays
- Dimensions: length, width, no. of channels (1 for B/W, 3 for RGB)
- Eg: 1920 x 1080 x 3 (full HD color)
- Elements of array: number corresponding to pixel value of image.
- Pre-processing: convert image to tensor for DL tasks.

Transformations

- Transformations - operations to make images easier to work with.
- Common transformations: resizing, converting to tensor, normalization (will discuss later).



≈

$$
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .6 & .8 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .7 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .7 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .5 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .7 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & .9 & 1 & .1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & .3 & 1 & .1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

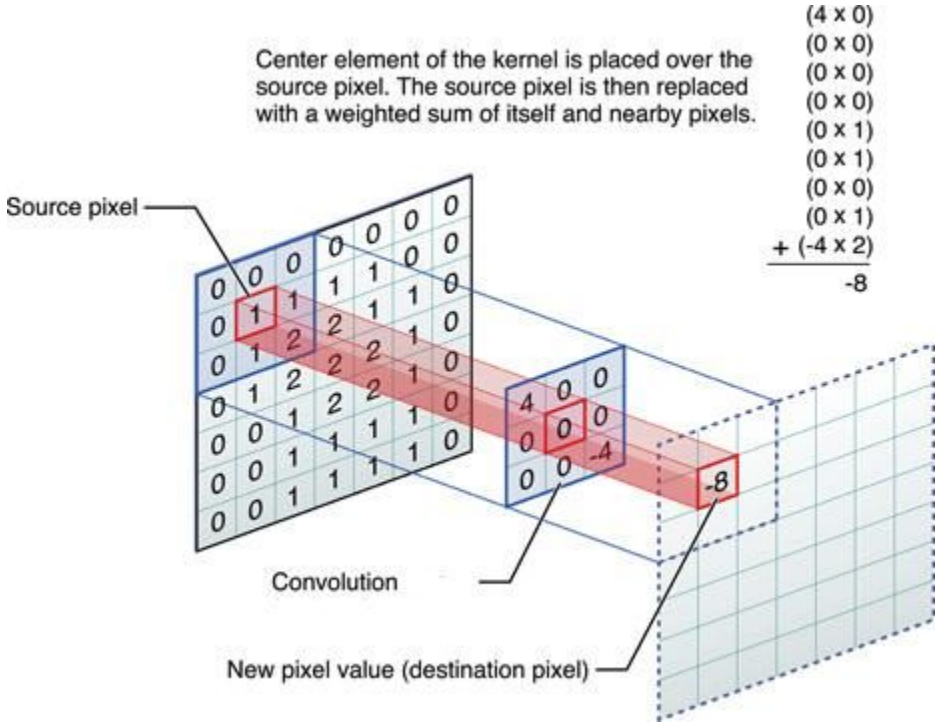
How an image is expressed in data analysis

Convolutional layers

- First step: pass into convolutional layer.
- This layer consists of an array called a filter.
- It moves over the input array like a scanner, covering every pixel at least once, and computing a calculation called a convolution.

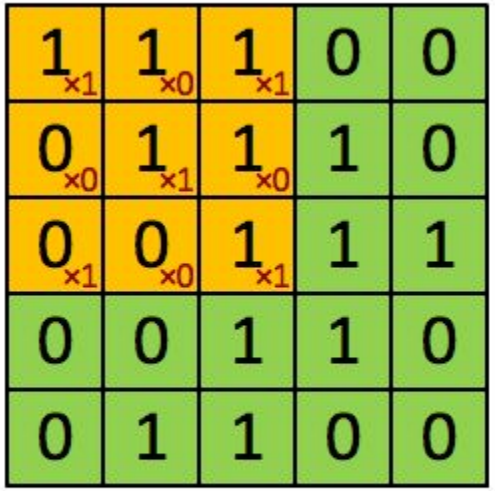
The filter of a convolutional layer

- Parameters: kernel size (filter dimensions)
- Stride (no. of pixels filter moves, after performing convolution operation)
- No. of output channels

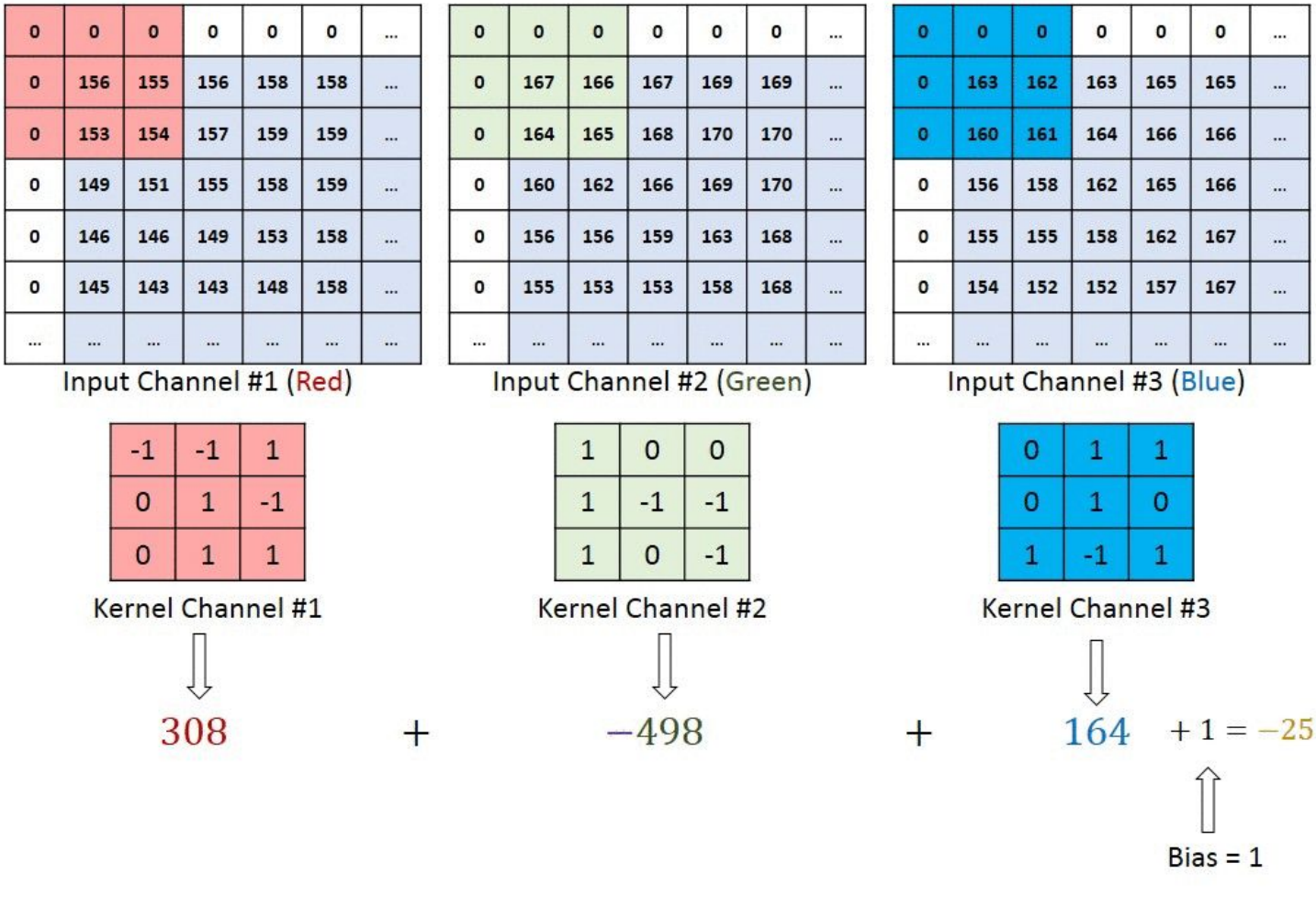


Mathematical explanation of a convolutional operation

Example of a convolution operation



Another example of a convolution operation


0 0 0 0 0 0 ...
0 156 155 156 158 158 ...
0 153 154 157 159 159 ...
0 149 151 155 158 159 ...
0 146 146 149 153 158 ...
0 145 143 143 148 158 ...
... ...
Input Channel #1 (Red)
0 0 0 0 0 0 ...
0 167 166 167 169 169 ...
0 164 165 168 170 170 ...
0 160 162 166 169 170 ...
0 156 156 159 163 168 ...
0 155 153 153 158 168 ...
... ...
Input Channel #2 (Green)
0 0 0 0 0 0 ...
0 163 162 163 165 165 ...
0 160 161 164 166 166 ...
0 156 158 162 165 166 ...
0 155 155 158 162 167 ...
0 154 152 152 157 167 ...
... ...
Input Channel #3 (Blue)
-1 -1 1
0 1 -1
0 1 1
Kernel Channel #1
1 0 0
1 -1 -1
1 0 -1
Kernel Channel #2
0 1 1
0 1 0
1 -1 1
Kernel Channel #3
308
+
−498
+
164
+ 1 = −25
Bias = 1
Output
-25 ... ...
... ...
... ...
... ...
... ...

Feature Image
5x5x1

0.4	0.8	0.1	-0.5	0.1
0.3	-0.7	-0.4	0.2	0.5
0.2	-0.9	-0.3	0.7	-0.6
0.1	0.5	-0.2	-0.5	0.8
-0.1	0.4	0.3	-0.7	0.1

Filter
2x2x1

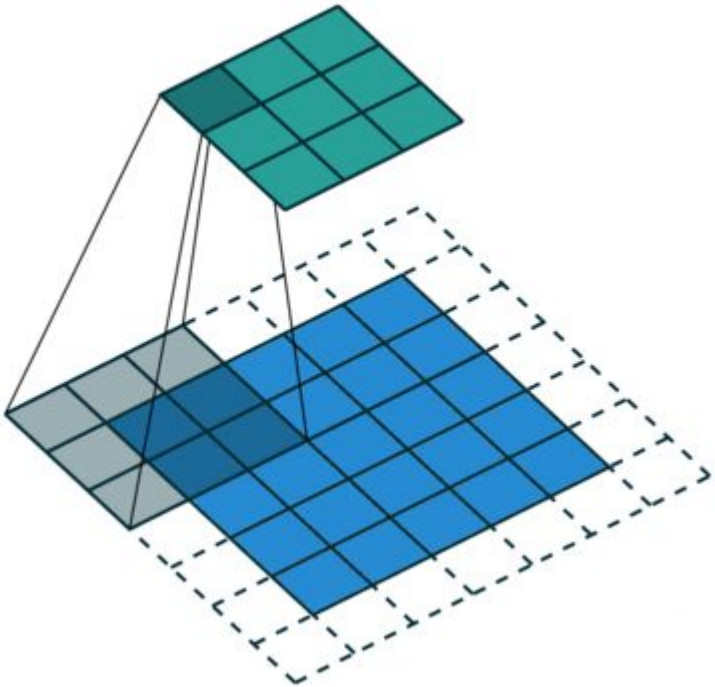
-0.3	0.4
0.8	-0.2

Feature image
4x4x1

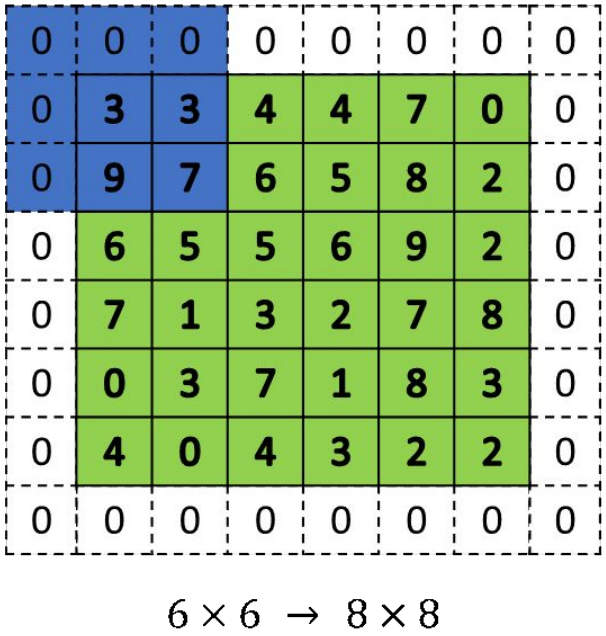
0.58	-0.68	-0.59	0.25
-0.03	-0.64	-0.18	0.82
-0.44	0.59	0.46	-1.01
-0.88	0.03	0.24	-0.11

CONVOLUTION

Stride 1
Padding 0



Convolution with stride of 2



$6 \times 6 \rightarrow 8 \times 8$

$*$

1	0	-1
1	0	-1
1	0	-1

3×3

$=$

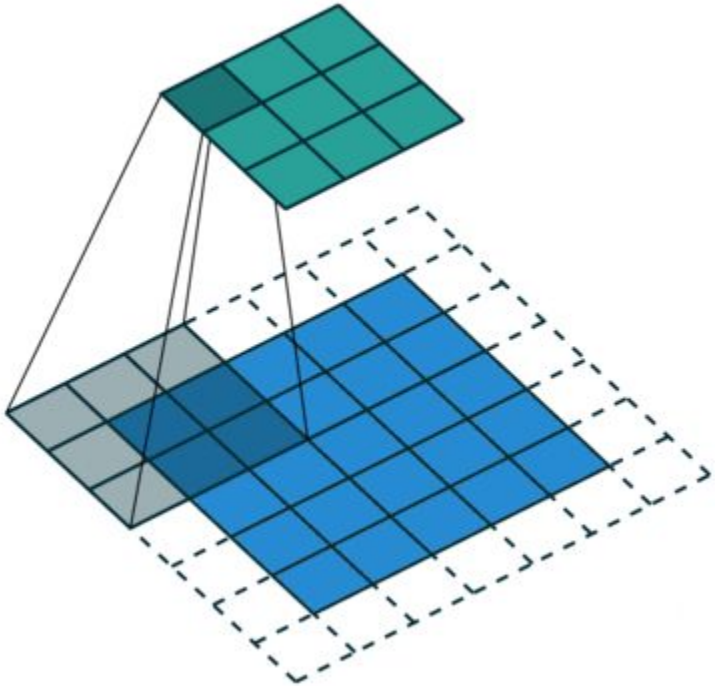
-10	-13	1			
-9	3	0			

6×6

Sometimes, we add padding to the input image to change the dimensions of the output.
Padding adds a border of zeroes to the input, changing the size while unaffecting the content stored.

Types of padding

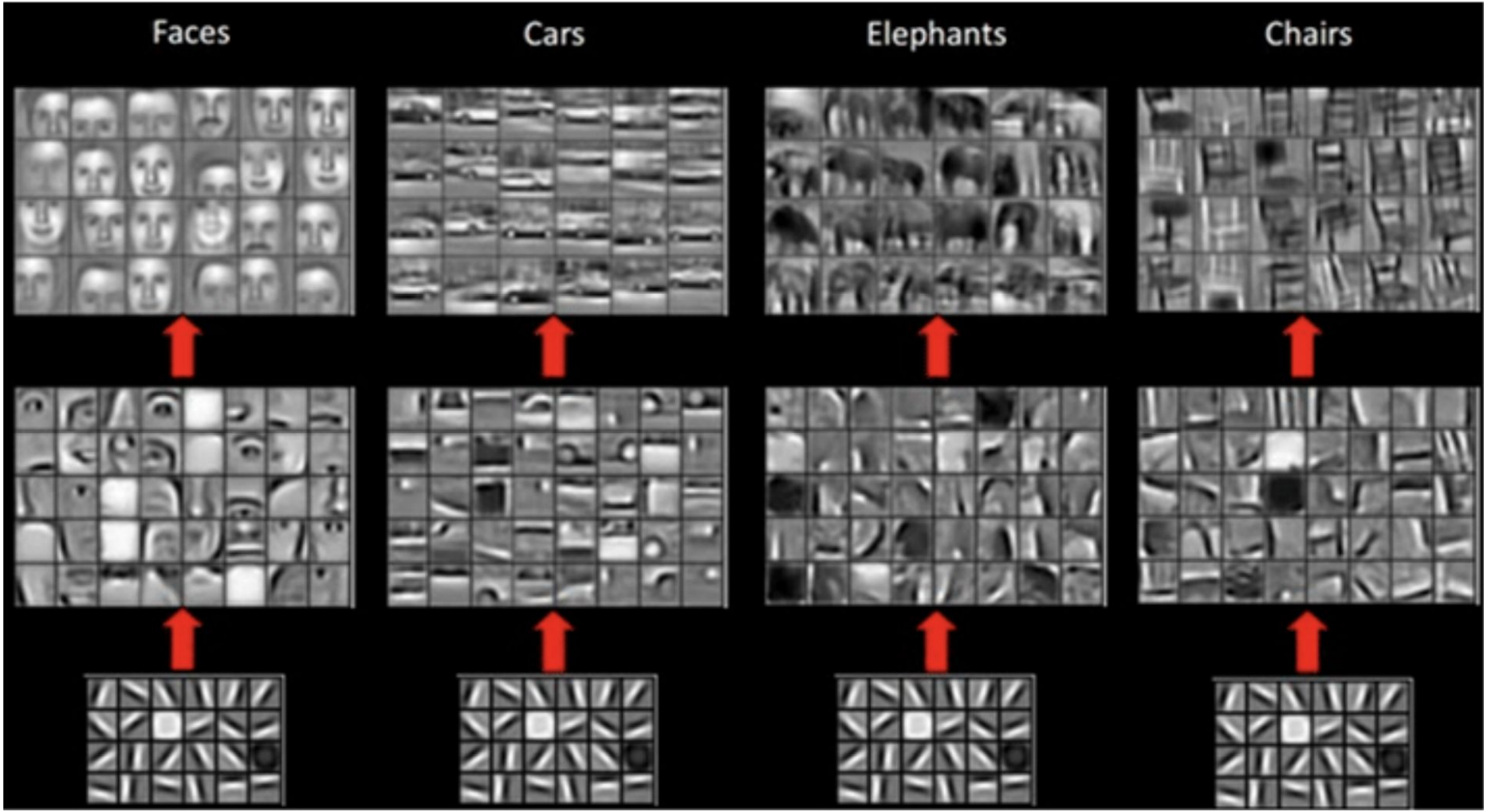
- Valid padding: No padding is applied, input image fed directly; i.e., it uses “valid” image data.
- Same padding: If a stride of 1 is used, then the output’s dimensions are the same as the input image.



Padding added to make 5x5 input into a 6x6 input

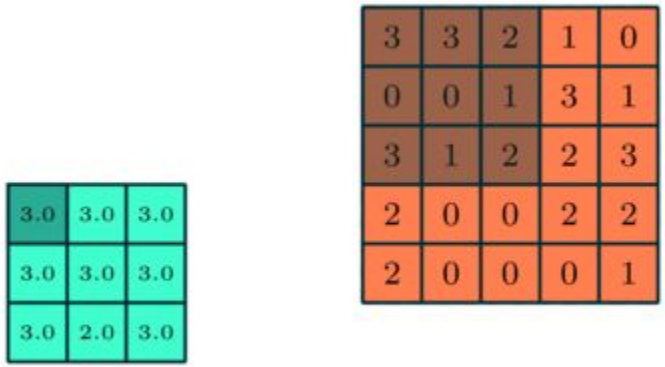
Guiding principles – Hierarchical structure of images

- Images: made of building blocks, growing in complexity.
- Most basic elements: lines, edges, vertices.
- Next level: circles, squares, etc. Complexity increases through layers.
- Convolutional layer - detects these elements in image.
- Each filter learns to recognize a particular element.

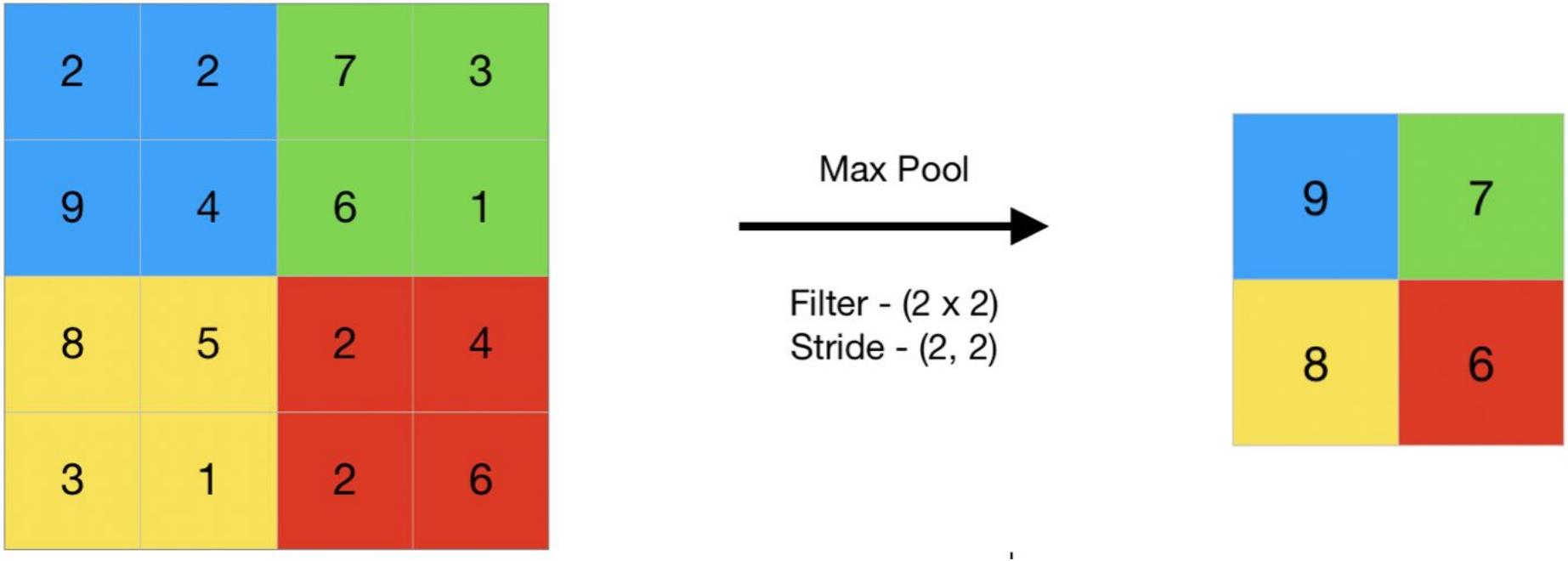

Faces
Cars
Elephants
Chairs

Pooling layers

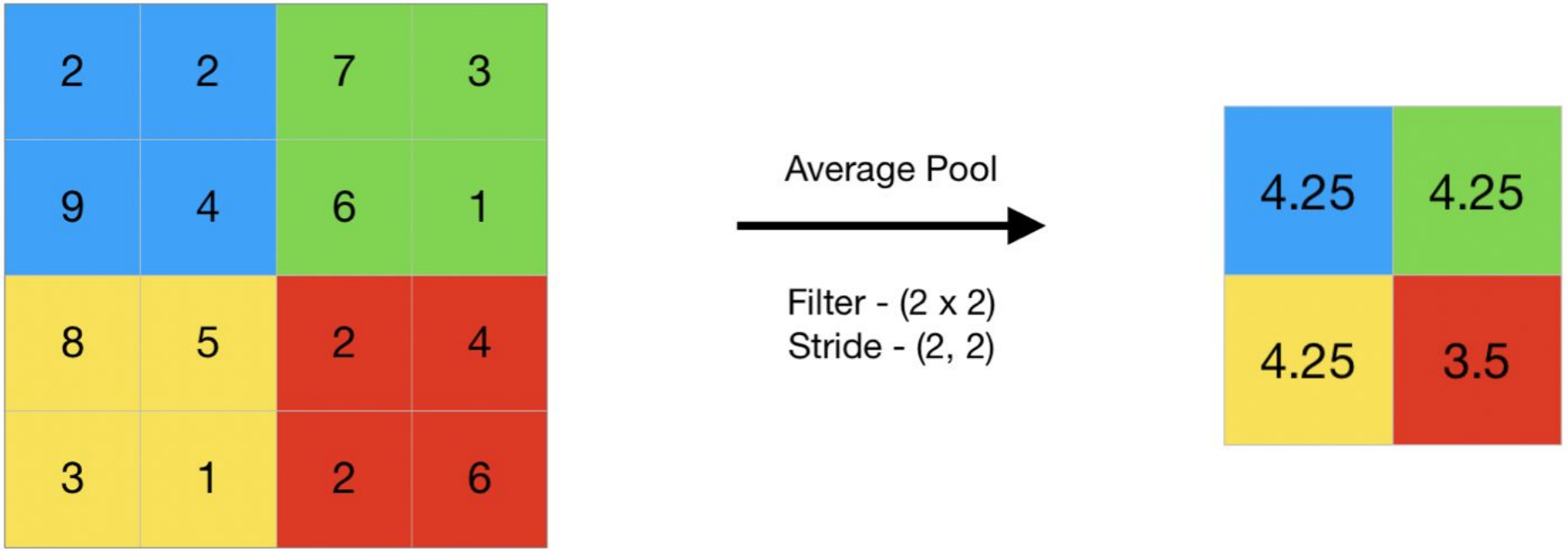
- “Filter”: performs operation, usually maximum or average.
- Condenses image: retains useful features, discards irrelevant ones.
- Have no learnable parameters.
- Arguments: stride, kernel size (same as in convolutional layers).
- Common: maxpool, avgpool.



An example of max pooling



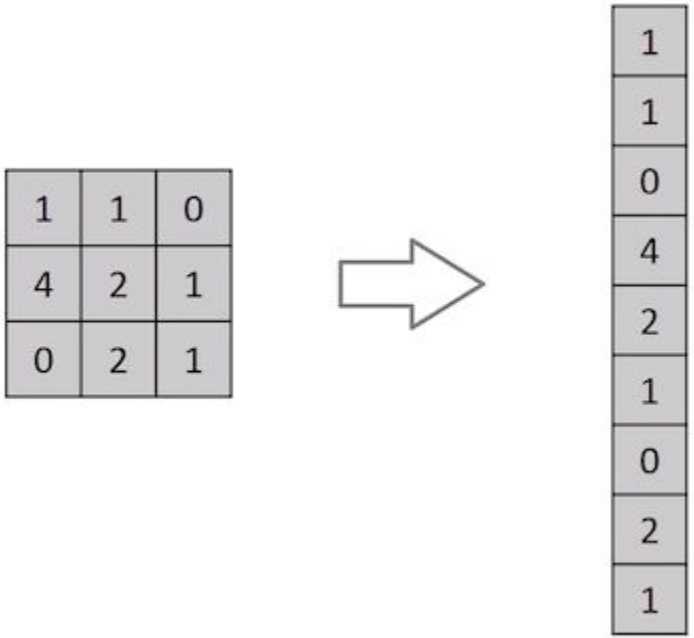
Another example of max pooling



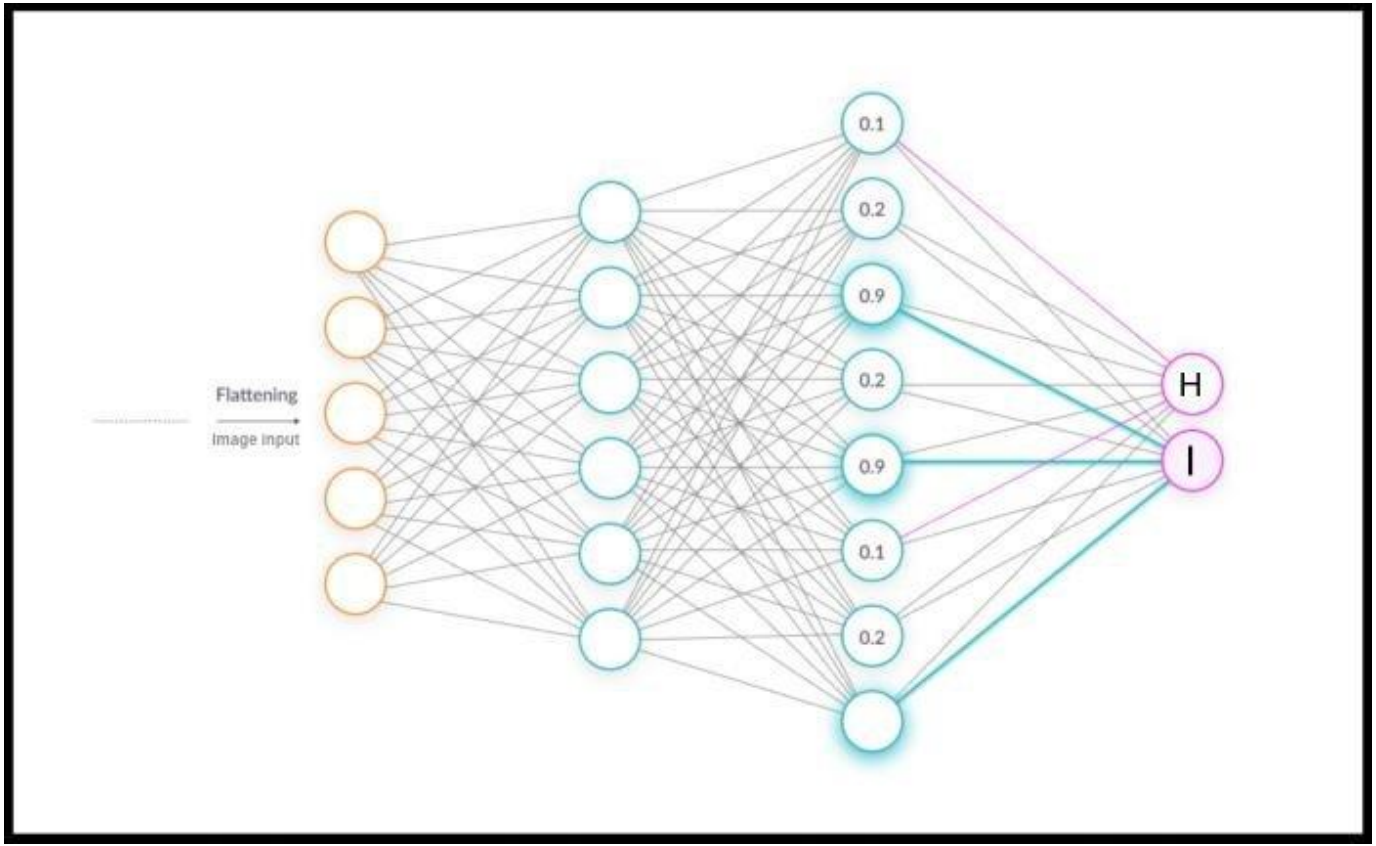
An example of avg pooling

Fully connected layers

- Normal neural network, present after convolutions and pooling.
- Resulting image: flattened into one-dimensional vector.
- Vector passed into fully connected layer.
- Output: processed according to model objective.



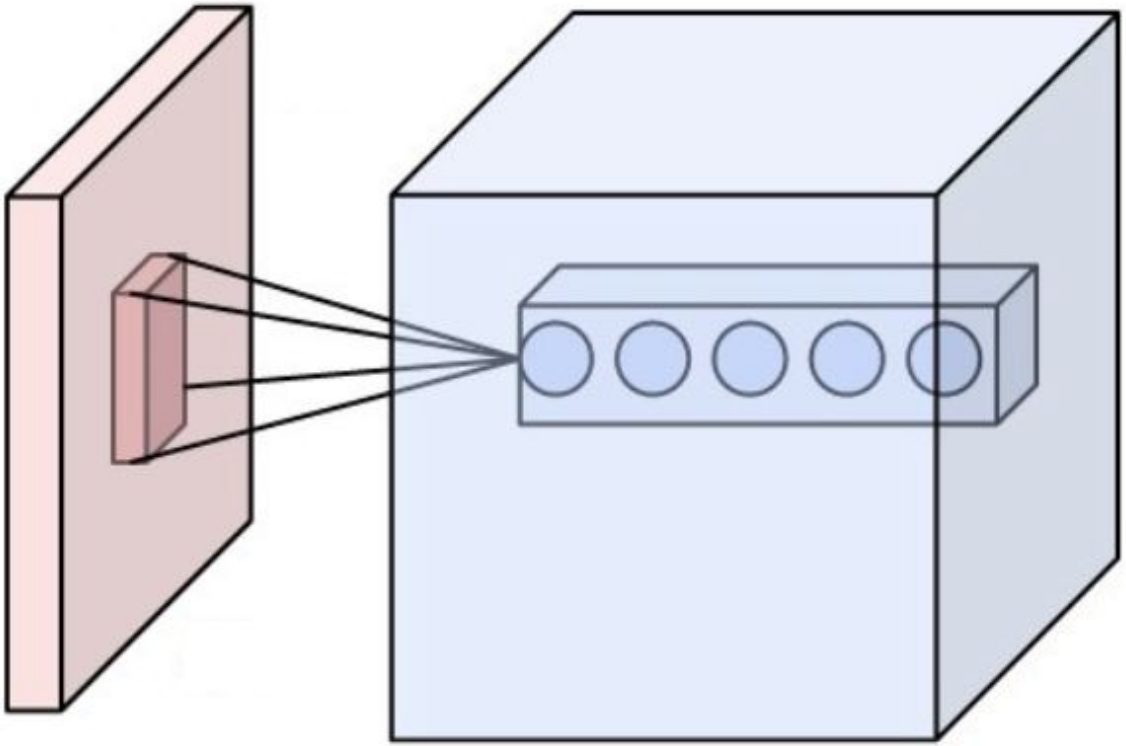
Flattening an image array into a one-dimensional vector



An example of a fully connected layer

The concept of receptive fields

- Receptive field: Part of the input image that is visible to the convolutional filter at a given time.
- Increases linearly as number of convolutional layers increases.
- Images: large number of input features. Impractical to have all neurons interconnected.
- Receptive fields: help focus on particular parts of the image. Less clunky process.



Here, the receptive field is the dark red portion with lines attached.

A common issue in training

- Issue: input features are not of similar values (eg. 0-1, 0-100 ranges)
- Reduces efficiency; weights may differ similarly in magnitude
- Creates an issue called “internal covariate shift”.

The issue of internal covariate shift

- When calculating backpropagation: we assume that previous layer weights remain constant when updating gradients.
- In reality: all layer weights updated simultaneously.
- Result: Updating procedure is endlessly chasing a moving target, in a sense.

Batch Normalization

- Batch normalization (batchnorm): to solve internal covariate shift.
- Outputs of each layer (before or after activation, depending on function) in a mini-batch: standardized by using mean and standard deviation.
- Increases training efficiency and provides regularization effect.

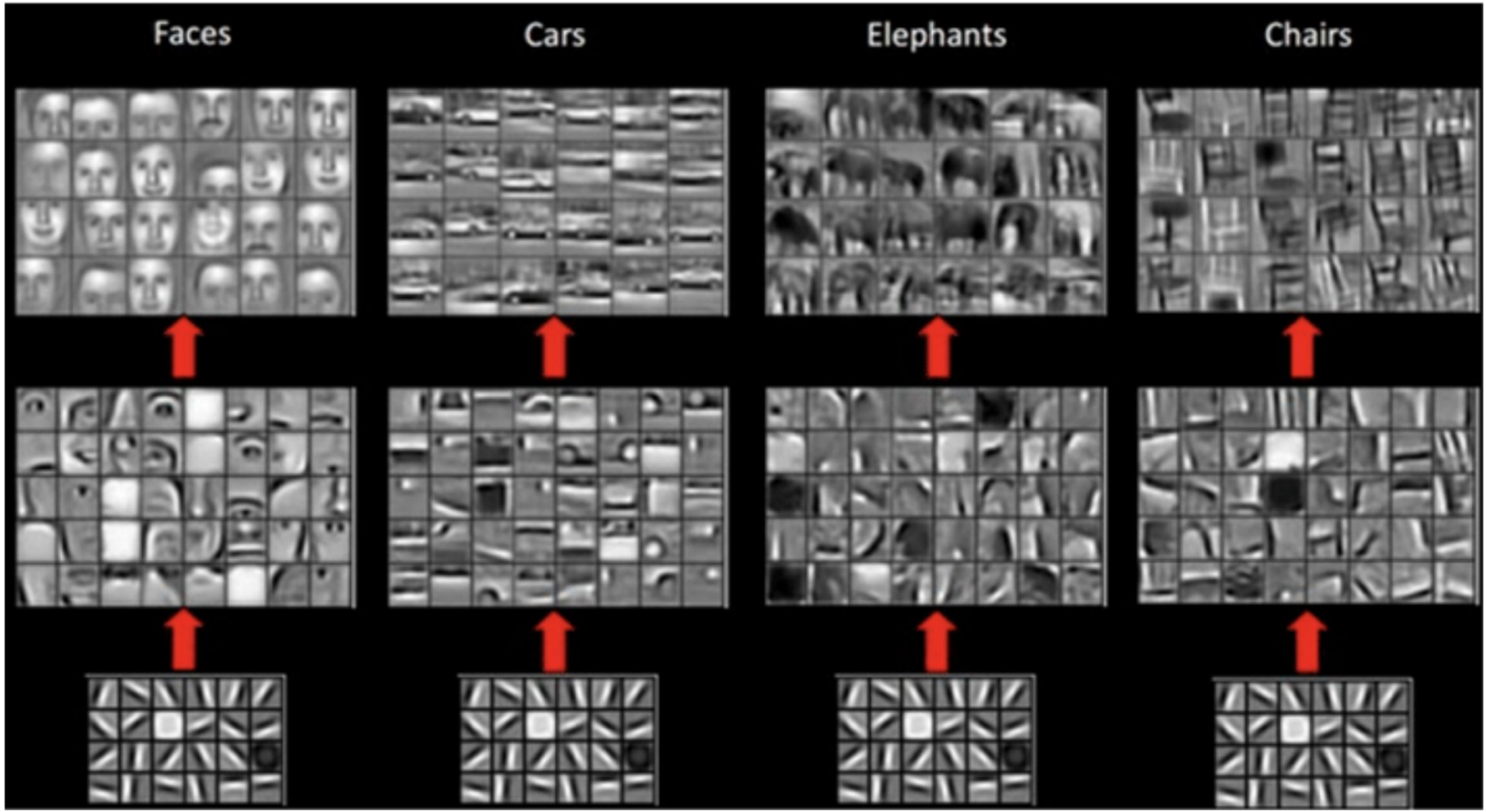
Why do we use CNNs?

1. Parameter sharing: Filter used to detect an edge in one part of image used in another part of the image.

2. Translational invariance: If image is shifted with respect to the original, no trouble in identifying as original.

3. Lower number of parameters to be learned: filter for convolutional layers, none for pooling layers.

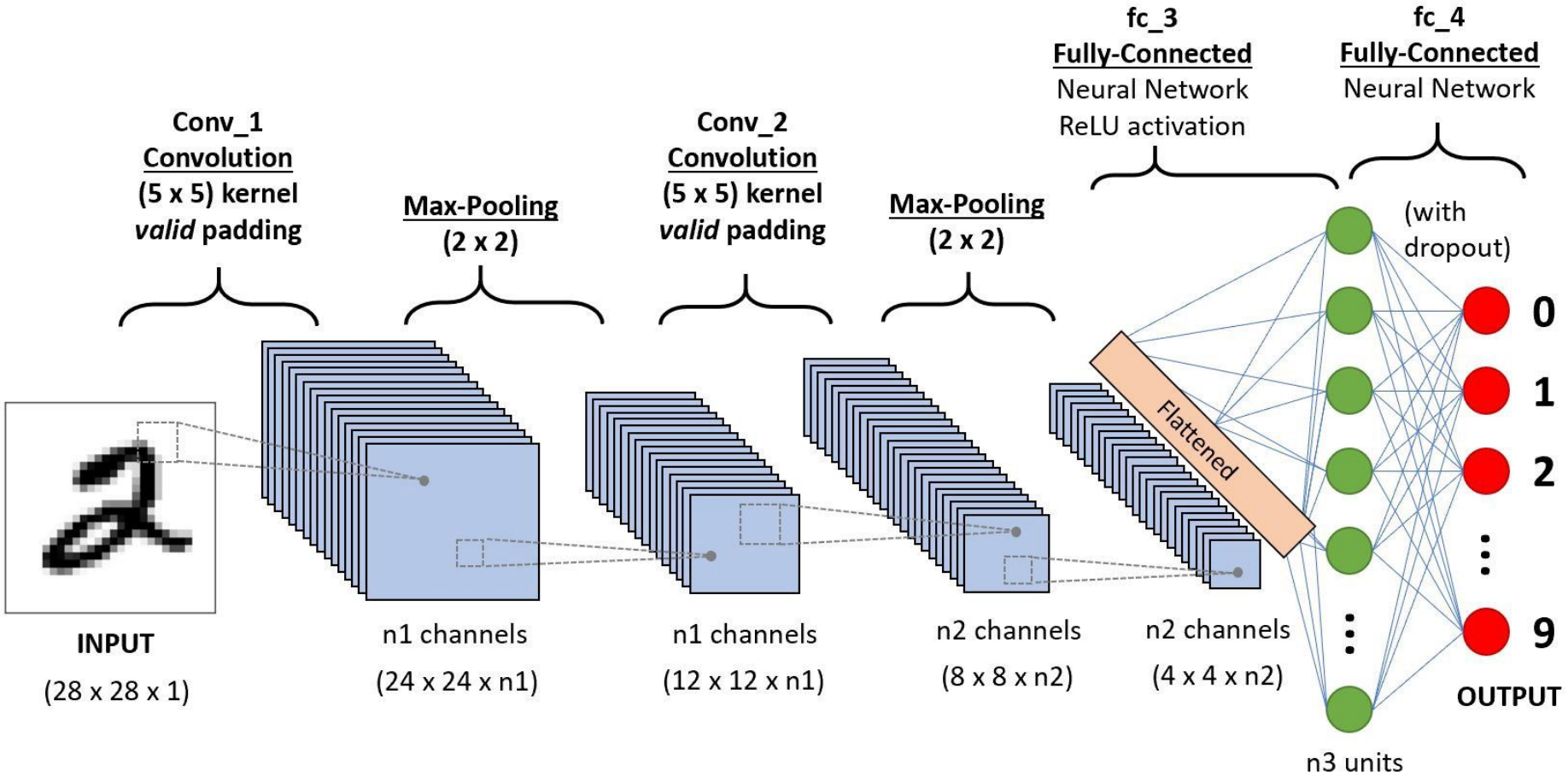
4. Transfer learning (explained in following slides)



As we can see here, the lowest level elements are the same for all the images.

Transfer Learning

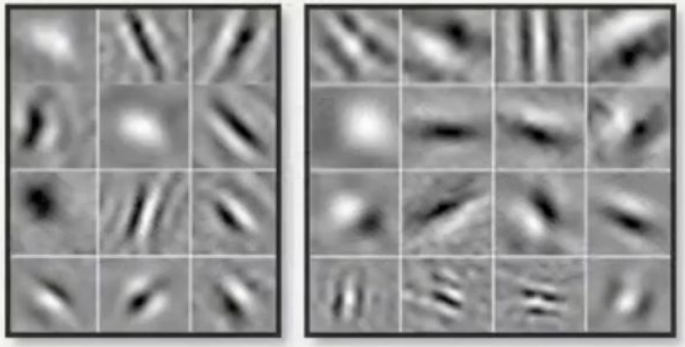
- Lowest level features for all image groups - same.
- Can reasonably assume: weights learned for these filters on one task can be used for other tasks.
- Pretrained weights: for lower levels of hierarchy.
- Only train model for problem-specific layers varying case-by-case.
- Transfer learning: using pre-trained weights for lower layers.


Conv_1
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
Conv_2
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
fc_3
Fully-Connected
Neural Network
ReLU activation
fc_4
Fully-Connected
Neural Network
(with dropout)
Flattened
INPUT
(28 x 28 x 1)
n1 channels
(24 x 24 x n1)
n1 channels
(12 x 12 x n1)
n2 channels
(8 x 8 x n2)
n2 channels
(4 x 4 x n2)
n3 units
0
1
2
9
OUTPUT

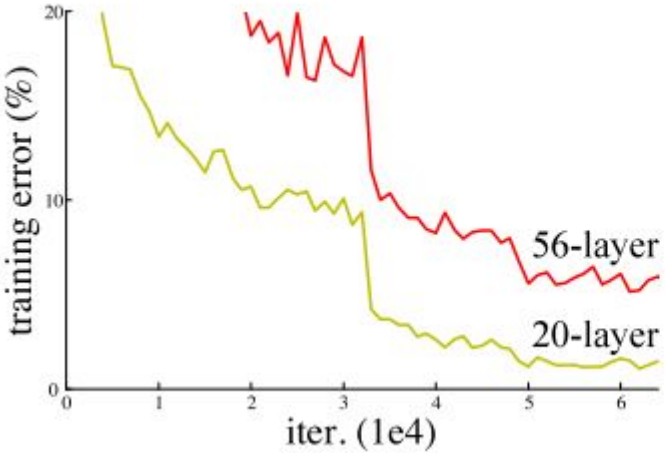
Post-lecture addition: Similarities to monkey retina and image processing

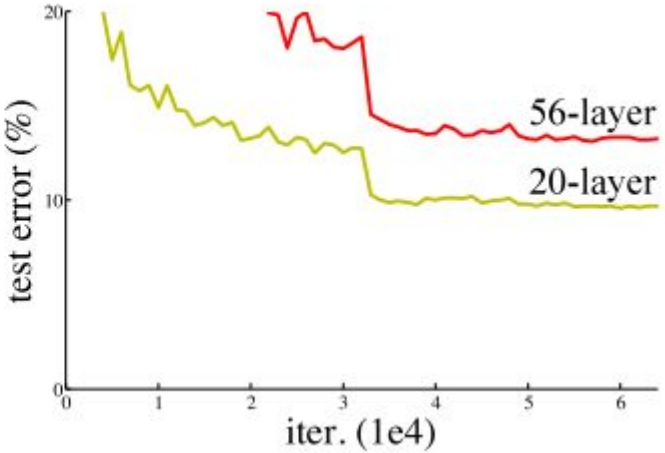
layer 1 filters:
convolutional neural network

neuron receptive fields:
macaque visual cortex



ResNet (Residual Network)

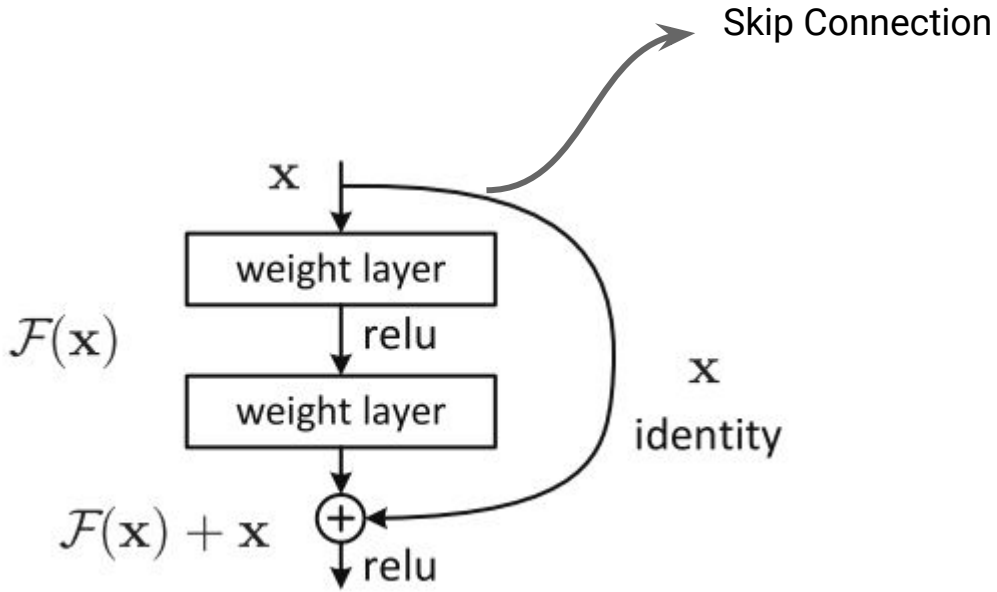




Observe the graph and speculate about the reasons

Reasons:

- Optimization function
- Initialization of the network
- Vanishing gradient problem
- NOT Overfitting



Residual Block

How ResNet helps?

- Skip connections solve the problem of **vanishing gradient** by allowing this ***alternate shortcut path*** for the gradient to flow through.

- By allowing the model to learn the **identity functions** which ensures that the higher layer will perform *at least as good* as the lower layer, and not worse.

Inception Network

Inception Network - v1

Motivation:

- **Salient parts** in the image can have extremely large **variation in size**

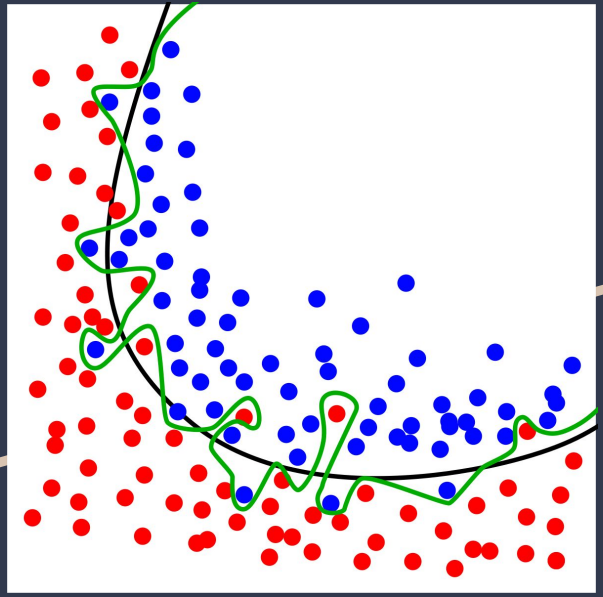


Motivation:

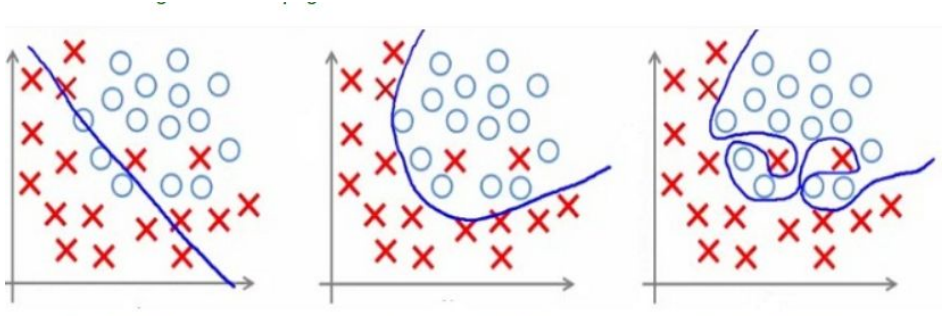
Larger Kernel ⟺ Global information

Smaller Kernel ⟺ Local Information

Motivation:



- Very Deep networks tend to be overfitting



Motivation:

- Naively stacking large convolution operations is computationally expensive.

Solution:

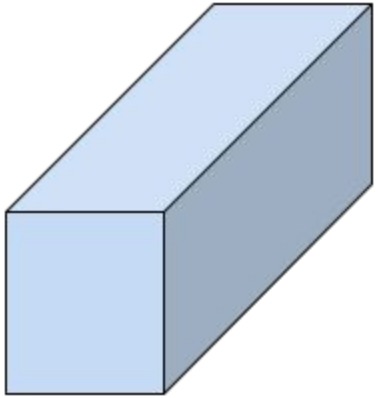
- Why not have filters with multiple size operators on the same level ?

3x3

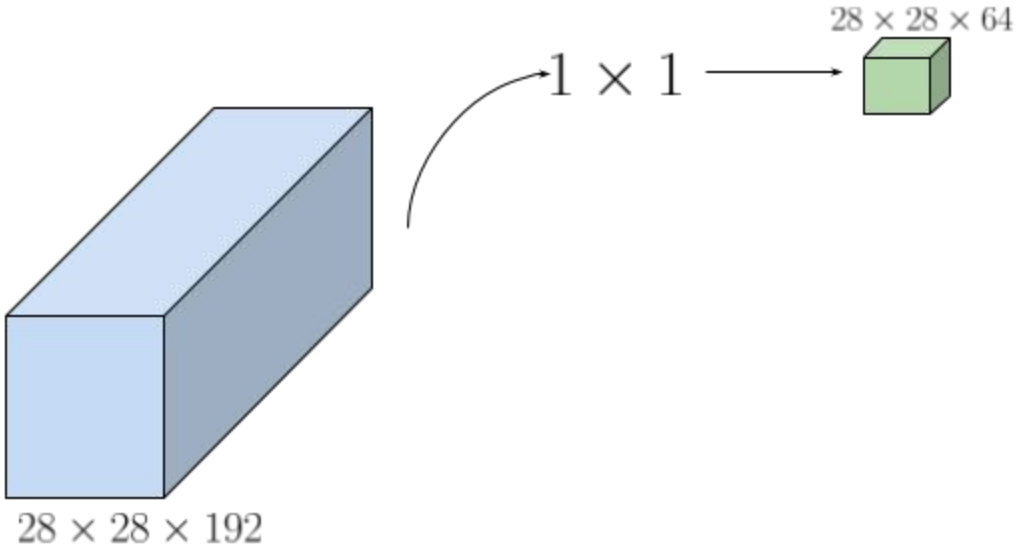
0.91	0.32	0.07
0.73	0.26	0.81
0.53	0.68	0.14

5x5

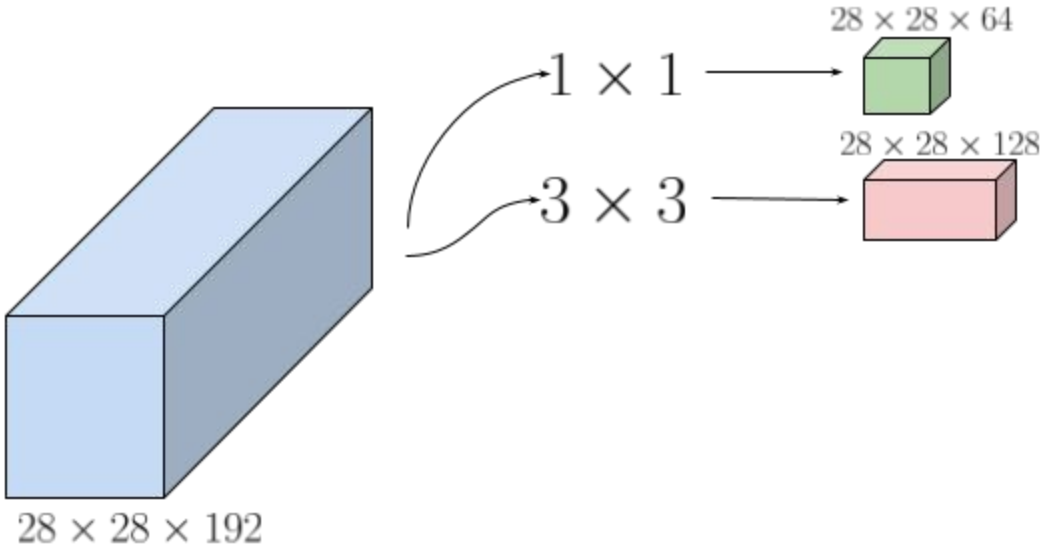
0.27	0.64	0.44	0.84	0.29
0.28	0.06	0.89	0.99	0.33
0.64	0.67	0.08	0.38	0.03
0.04	0.31	0.16	0.57	0.08
0.87	0.85	0.97	0.71	0.96



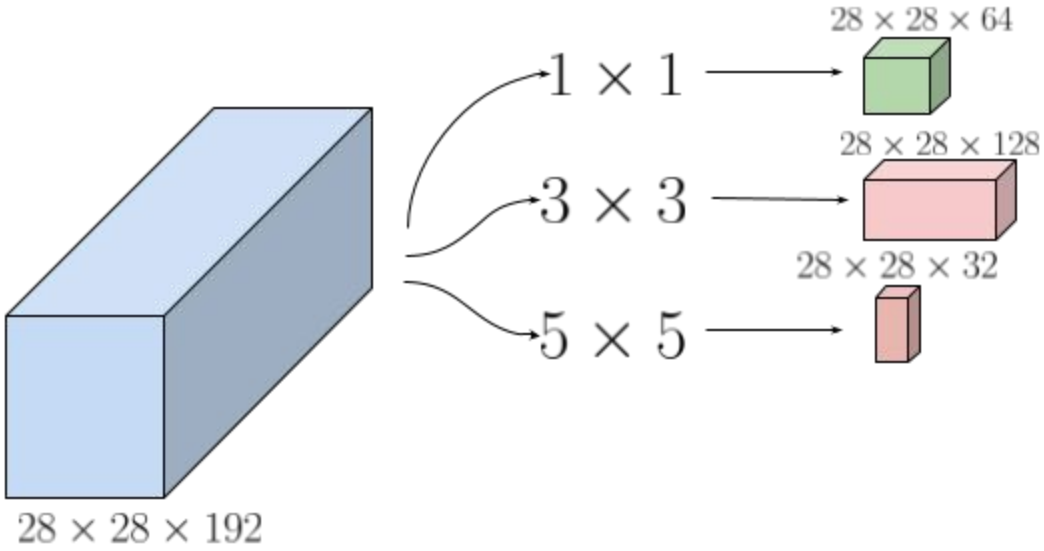
Input dimension



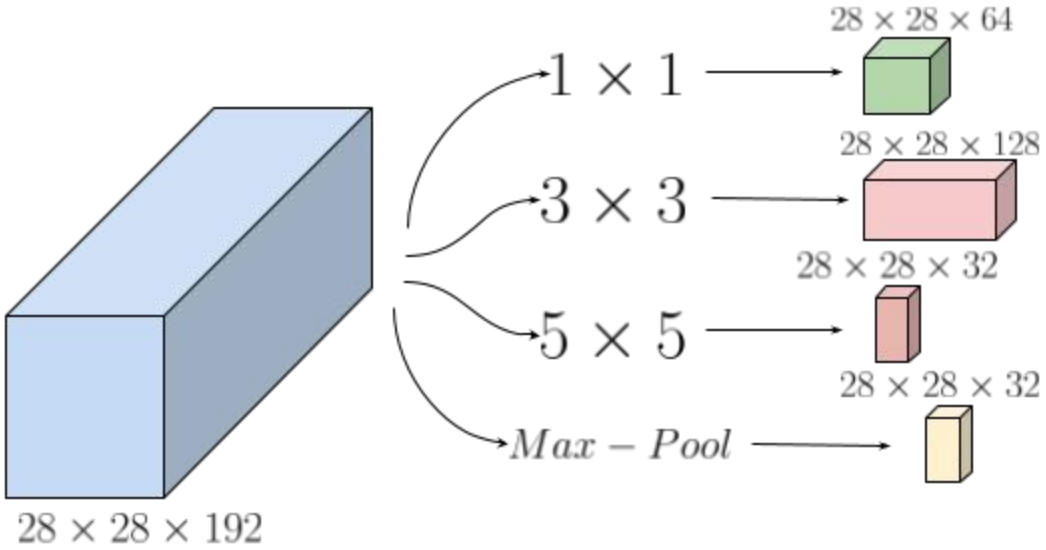
1 ×1 filter - 64



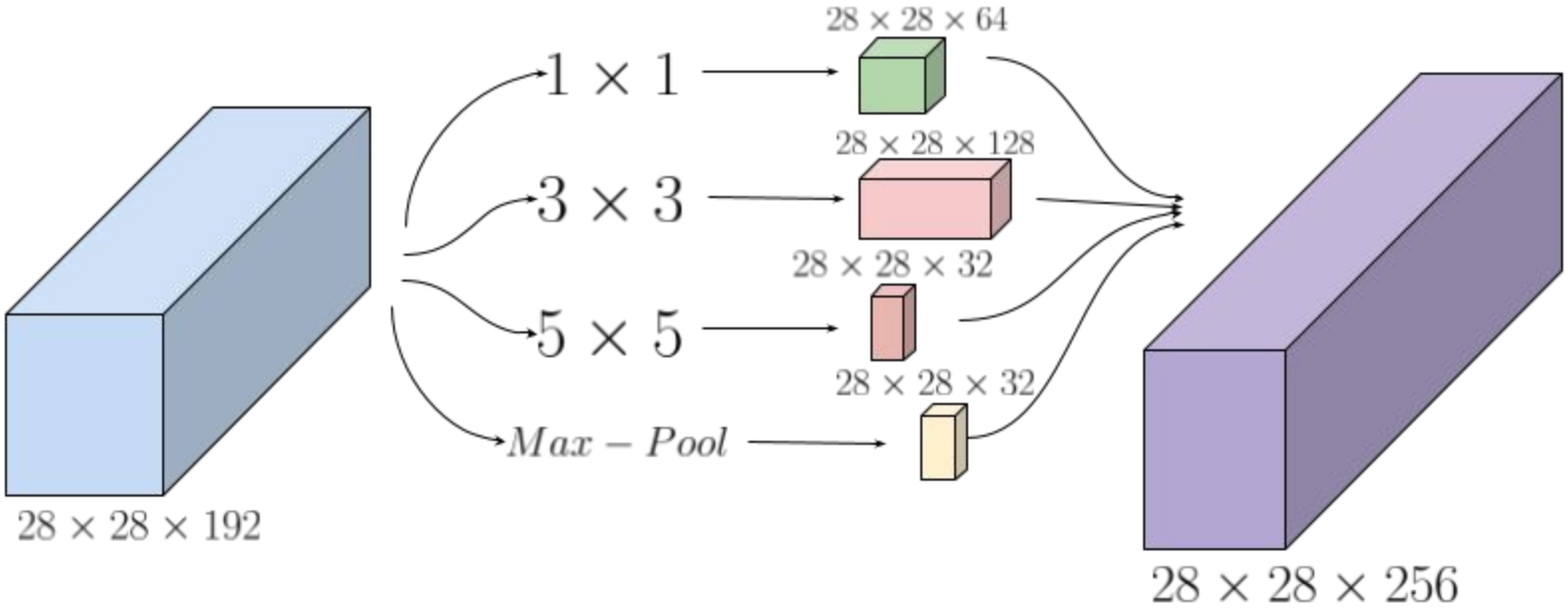
3 x 3 filter - 128



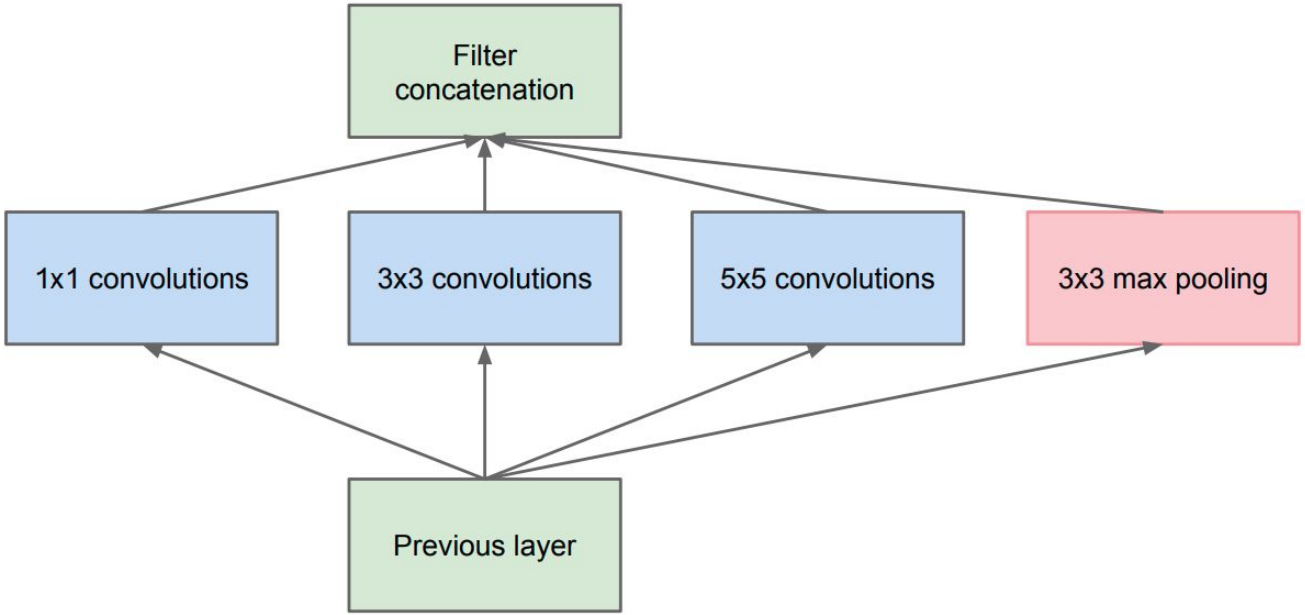
5 x 5 filter - 32



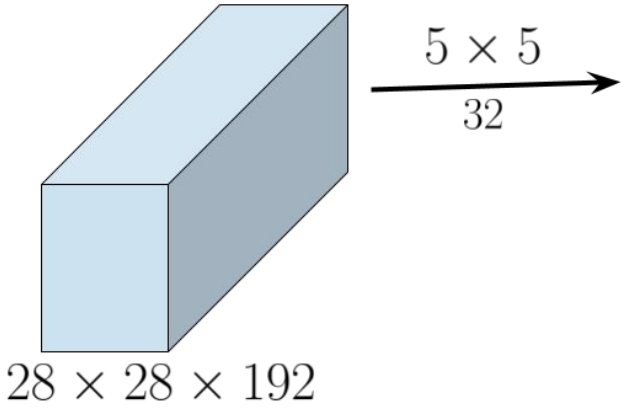
Max - Pooling



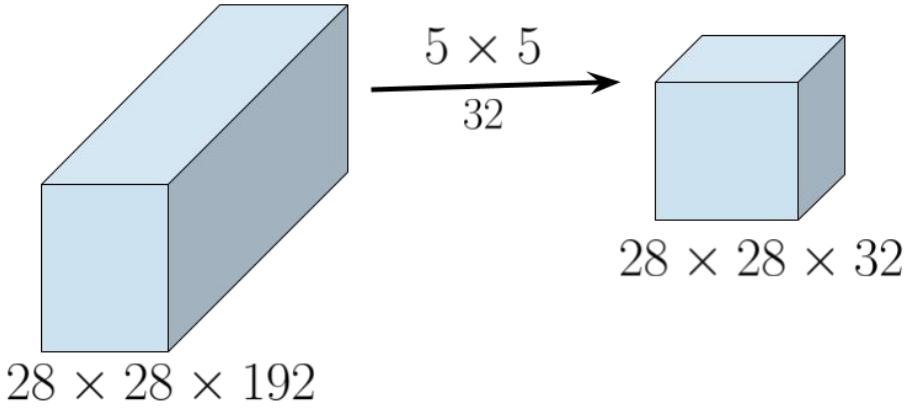
All the channels are concatenated



(a) Inception module, naïve version

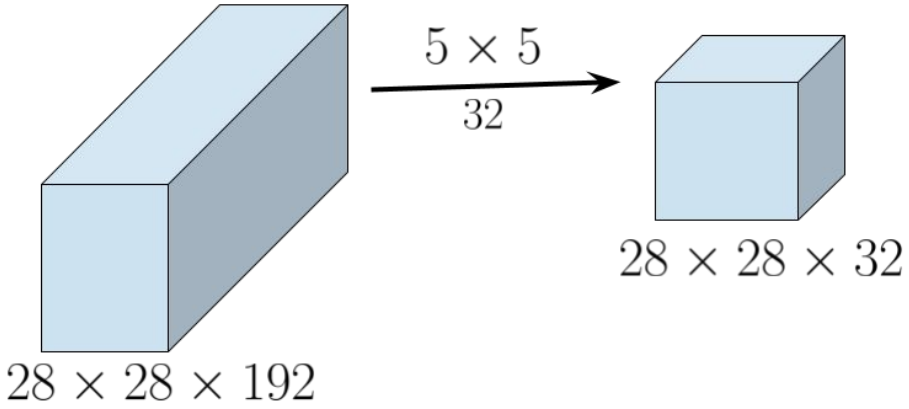


Let’s compute computational cost



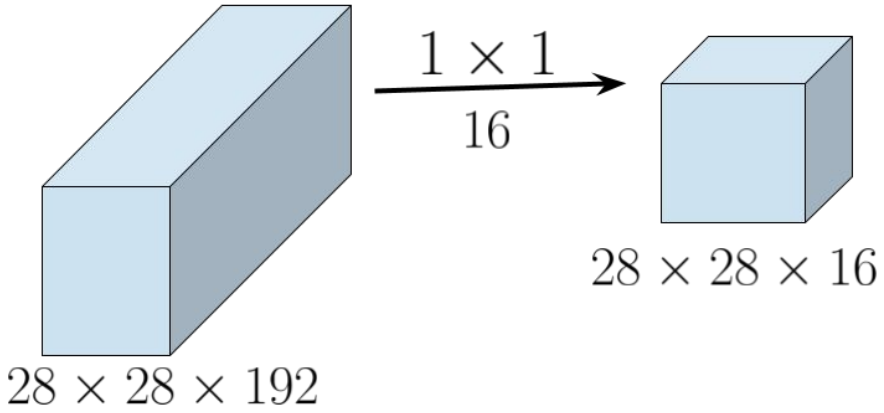
Let’s compute computational cost

$$Filter\ dimension = 5 \times 5 \times 192$$



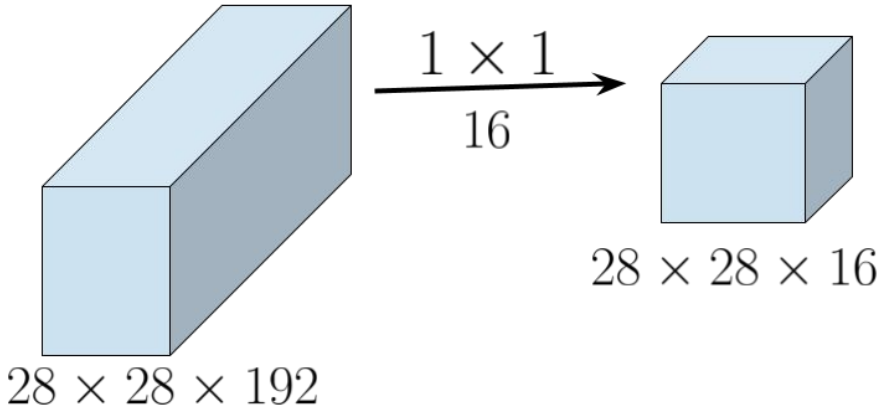
$$Number\ of\ computations = 28 \times 28 \times 32\ \times\ 5 \times 5 \times 192 = 120M$$

High Computation Cost !!

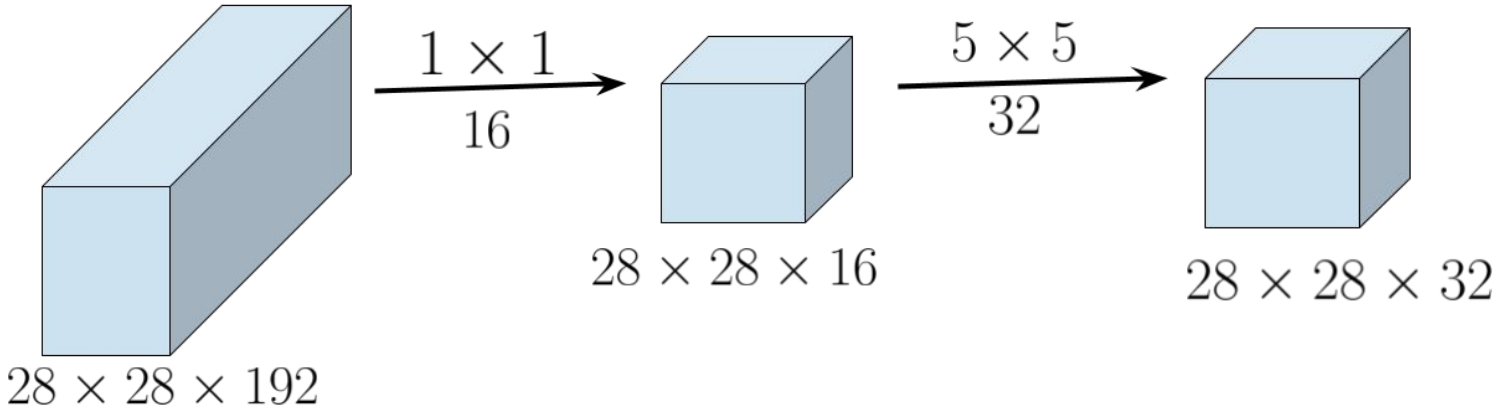


Using 1 × 1 Filters

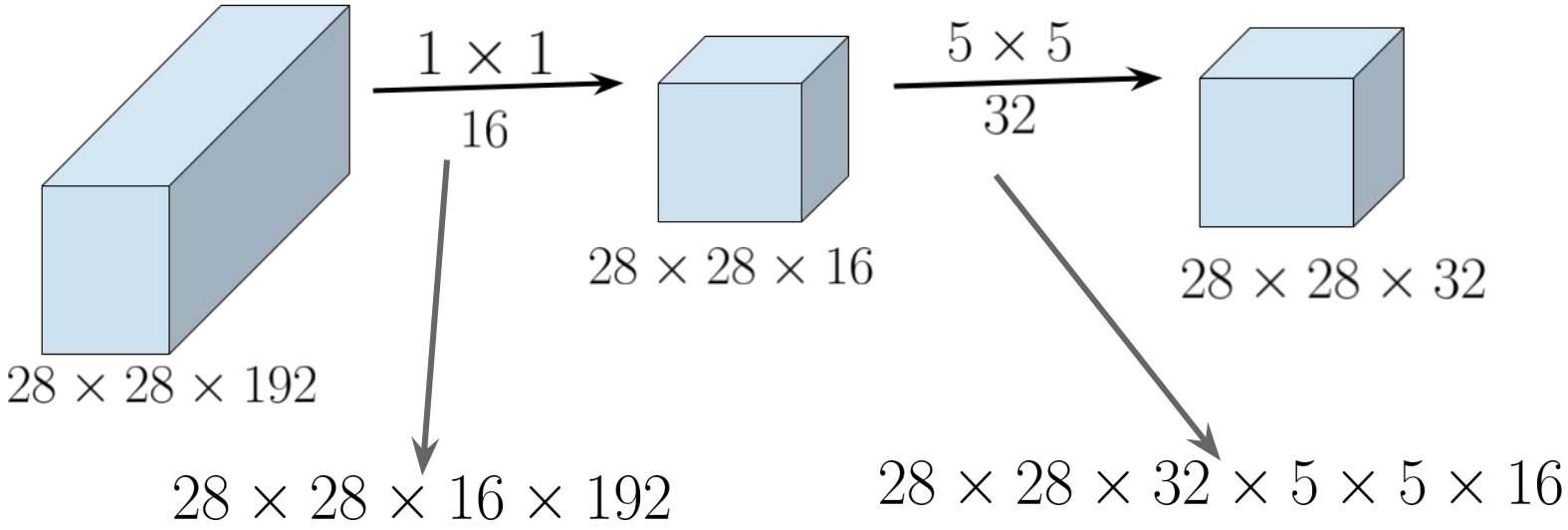
$$Filter\ dimension = 1 \times 1 \times 192$$



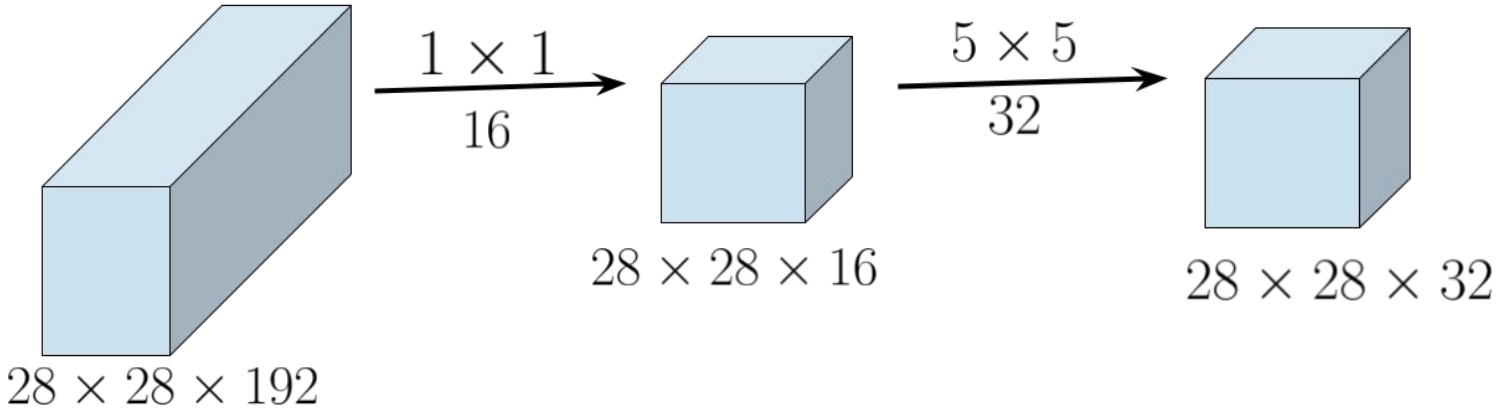
$$Filter\ dimension = 1 \times 1 \times 192$$



$$Filter\ dimension = 1 \times 1 \times 192$$

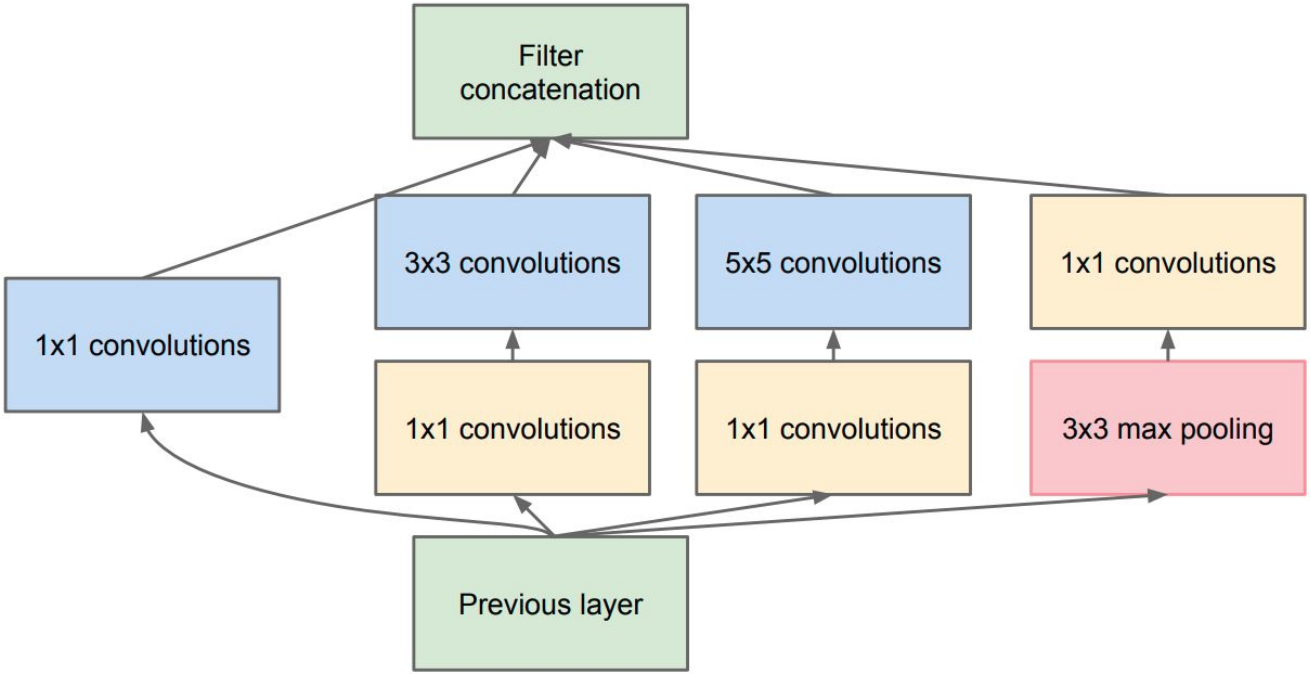


$$Filter\ dimension = 1 \times 1 \times 192$$



$$Total = 12.4M$$

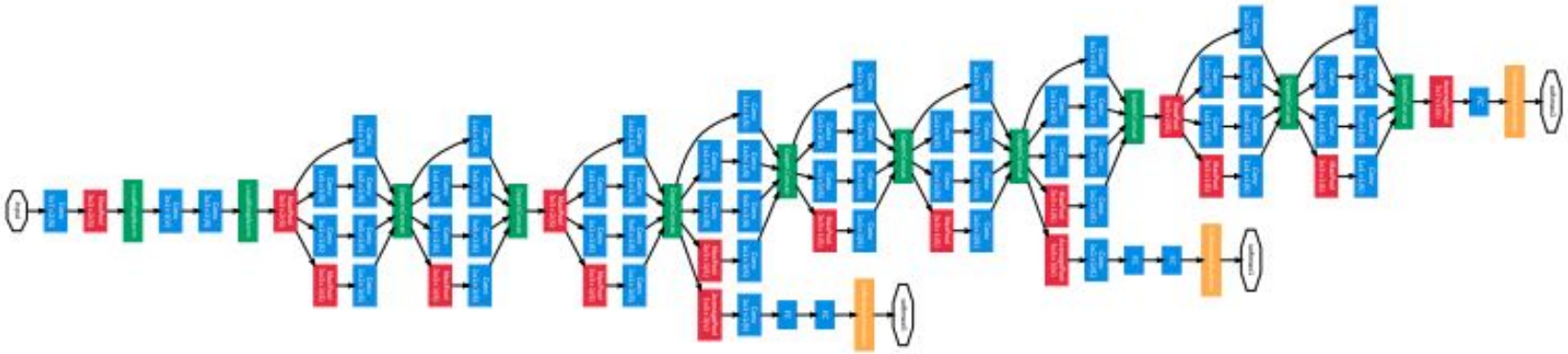
Reduced Computational Cost



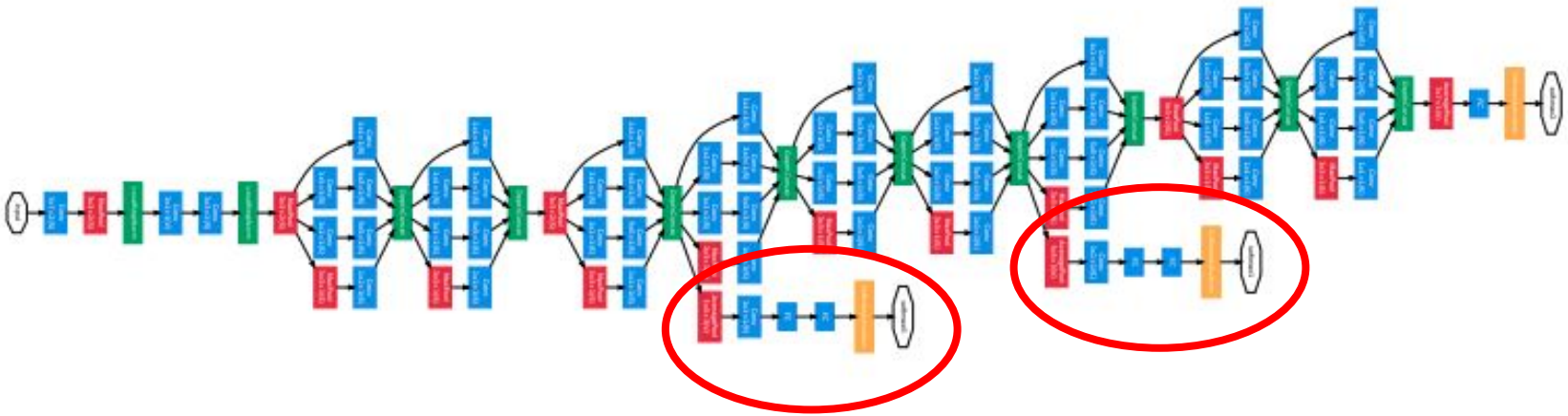
(b) Inception module with dimension reductions

GoogLeNet has 9 such inception modules stacked linearly. It is 22 layers deep (27, including the pooling layers). It uses global average pooling at the end of the last inception module.

Inception -V1 Architecture



GoogleNet

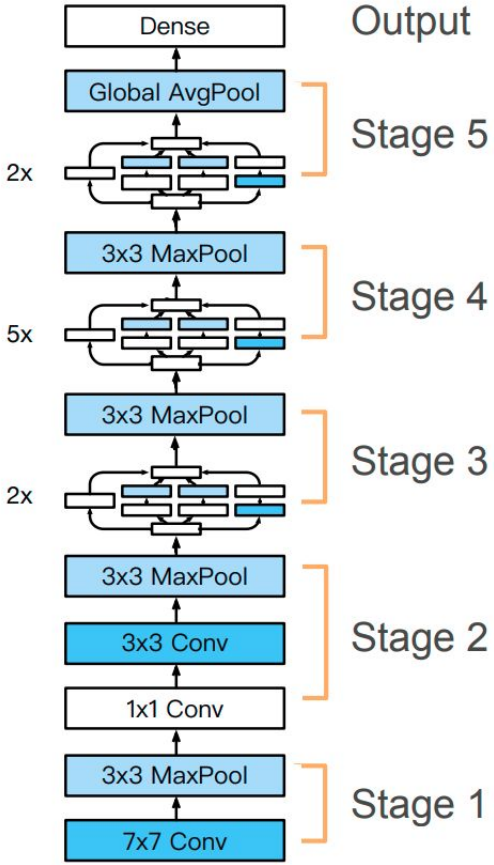


```
# The total loss used by the inception net during training.
total_loss = real_loss + 0.3 * aux_loss_1 + 0.3 * aux_loss_2
```

Auxiliary loss

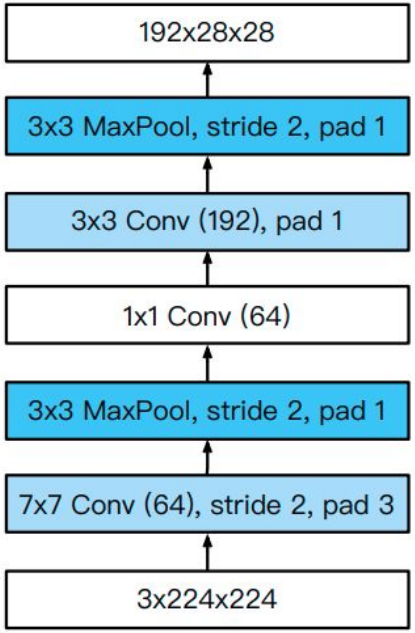
GoogLeNet

- 5 stages with 9 inceptions blocks



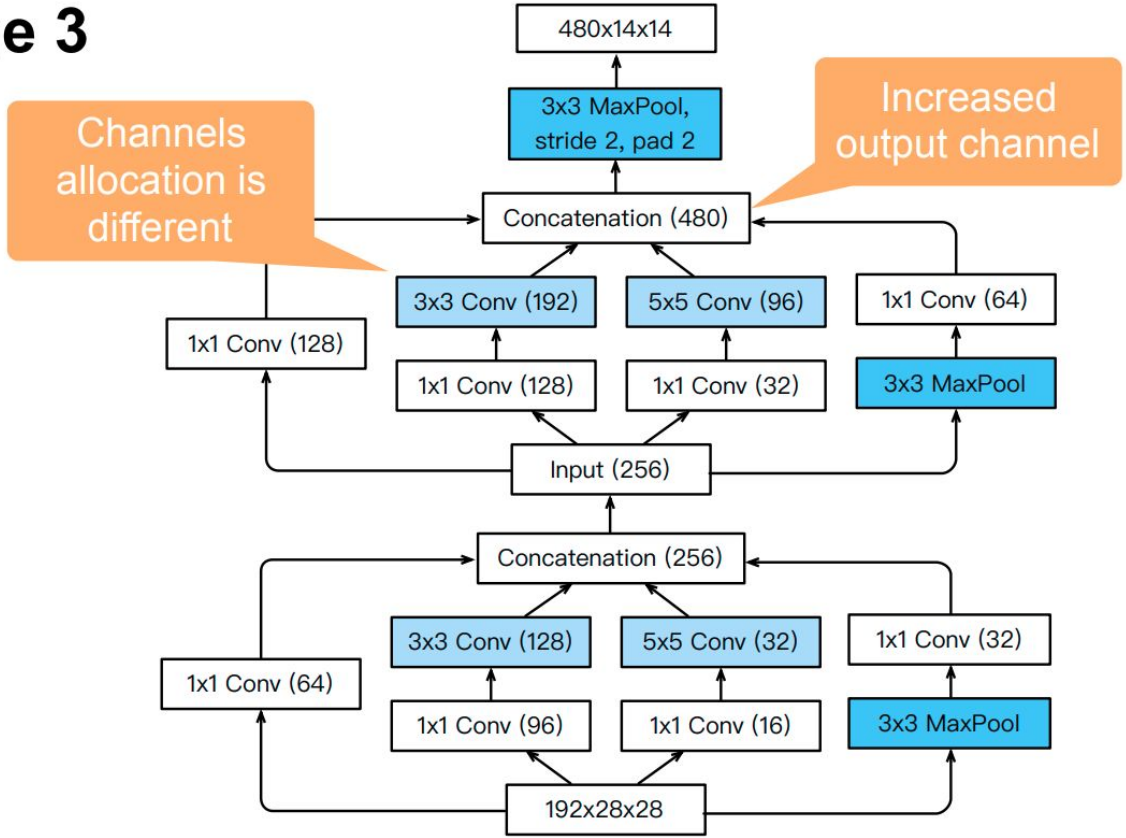
courses.d2l.ai/berkeley-stat-157

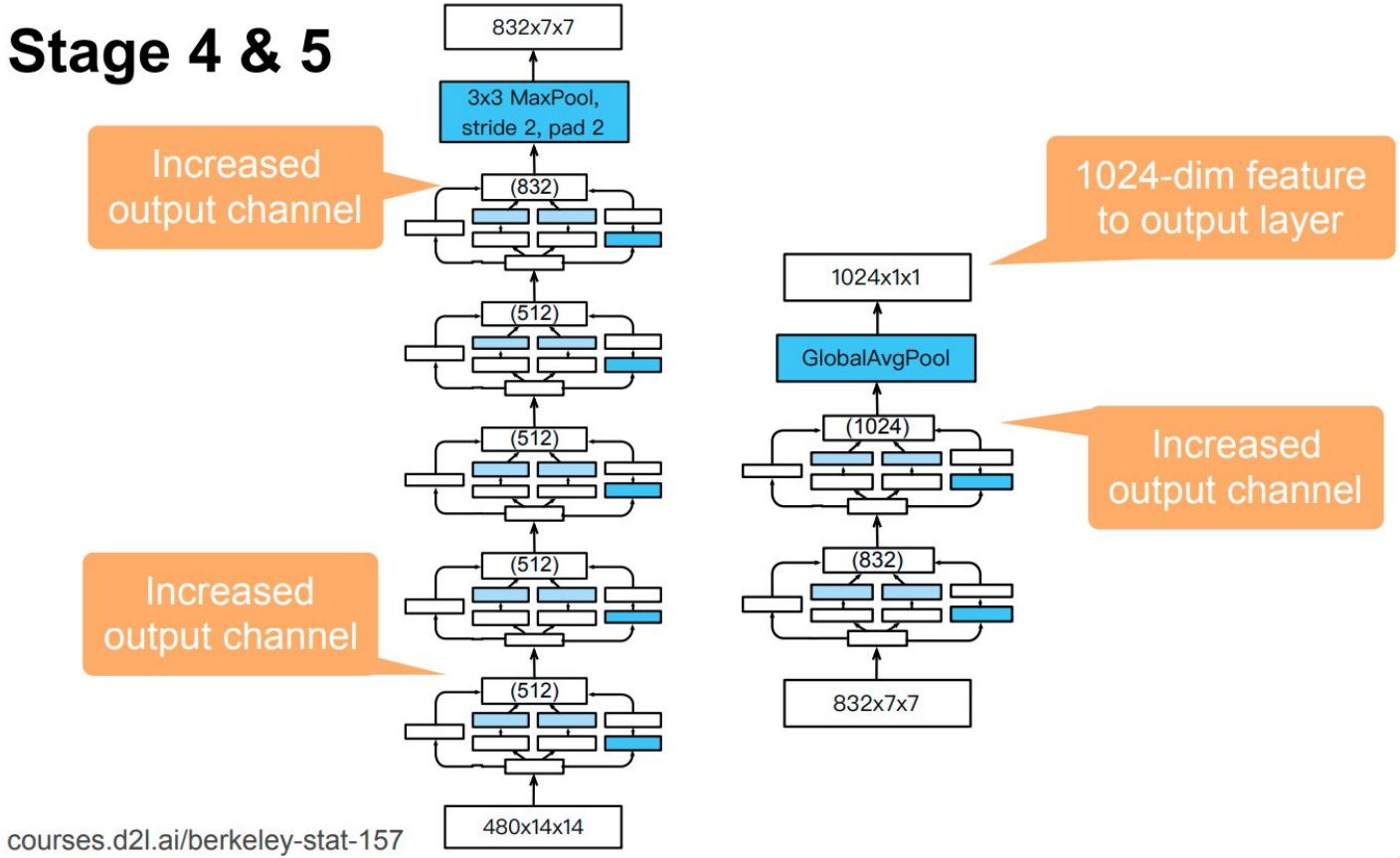
GoogLeNet



Stage - 1 and 2

Stage 3

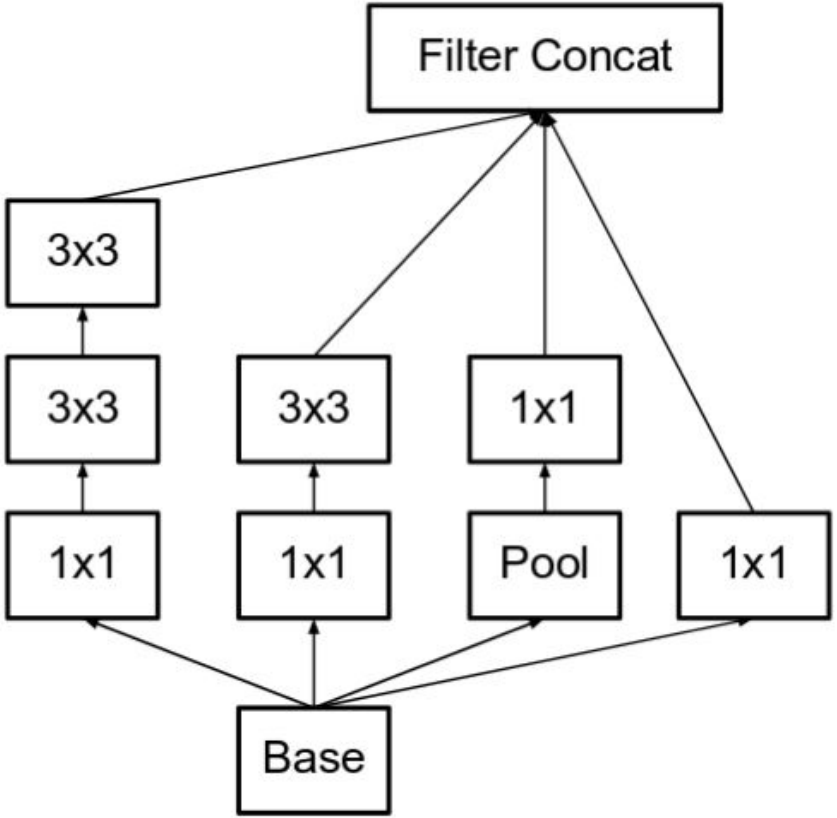

480x14x14
3x3 MaxPool, stride 2, pad 2
Increased output channel
Channels allocation is different
Concatenation (480)
3x3 Conv (192)
5x5 Conv (96)
1x1 Conv (64)
1x1 Conv (128)
1x1 Conv (128)
1x1 Conv (32)
3x3 MaxPool
Input (256)
Concatenation (256)
3x3 Conv (128)
5x5 Conv (32)
1x1 Conv (32)
1x1 Conv (64)
1x1 Conv (96)
1x1 Conv (16)
3x3 MaxPool
192x28x28


Stage 4 & 5
832x7x7
3x3 MaxPool, stride 2, pad 2
Increased output channel
(832)
(512)
(512)
(512)
(512)
Increased output channel
480x14x14
courses.d2l.ai/berkeley-stat-157
1024-dim feature to output layer
1024x1x1
GlobalAvgPool
(1024)
Increased output channel
(832)
832x7x7

Stage - 4 and 5

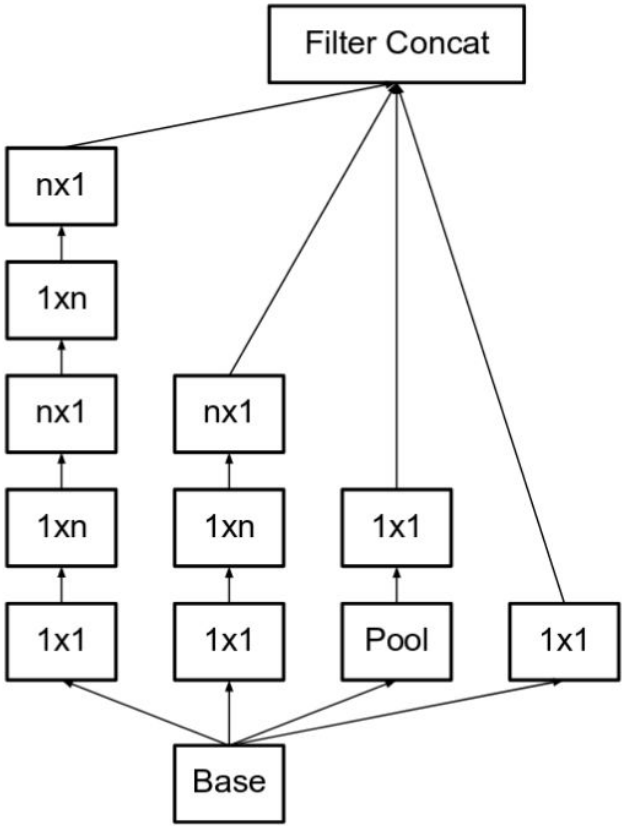
Inception v2

- Avoids reducing the dimension too much as it may cause loss of information
- Makes the network more computationally efficient



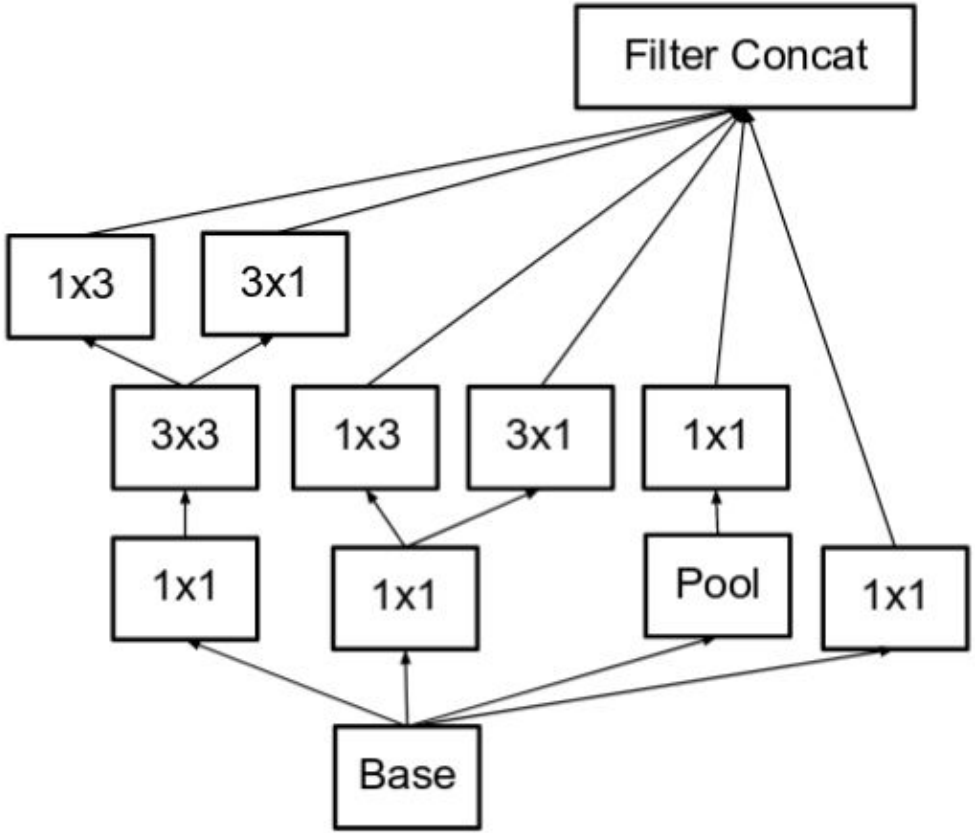
5 x 5 convolution is **2.78 times more expensive** than a 3 x 3

The leftmost 5×5 filter replaced by two 3×3 filters



33% more cheaper than the single 3 x 3 convolution

3 x 3 filters are replaced with 3 x 1 and 1 x 3 filters



Making the inception module wider

type	patch size/stride or remarks	input size
conv	$3\times3/2$	$299\times299\times3$
conv	$3\times3/1$	$149\times149\times32$
conv padded	$3\times3/1$	$147\times147\times32$
pool	$3\times3/2$	$147\times147\times64$
conv	$3\times3/1$	$73\times73\times64$
conv	$3\times3/2$	$71\times71\times80$
conv	$3\times3/1$	$35\times35\times192$
$3\times$Inception	As in figure 5	$35\times35\times288$
$5\times$Inception	As in figure 6	$17\times17\times768$
$2\times$Inception	As in figure 7	$8\times8\times1280$
pool	8×8	$8\times8\times2048$
linear	logits	$1\times1\times2048$
softmax	classifier	$1\times1\times1000$

Inception v3

PREMISE:

- **Auxiliary classifiers** didn't contribute much until near the end of the training process

- Argued that they function as **regularizers**, especially if they have BatchNorm or Dropout operations

Inception v3

Solution:

- RMSProp Optimizer.
- Factorized 7 x 7 convolutions.
- BatchNorm in the Auxiliary Classifiers.
- Label Smoothing (Prevents overfitting)

Inception v4

Premise:

- Make the modules more **uniform**.
- Noticed that modules were **more complicated than necessary**

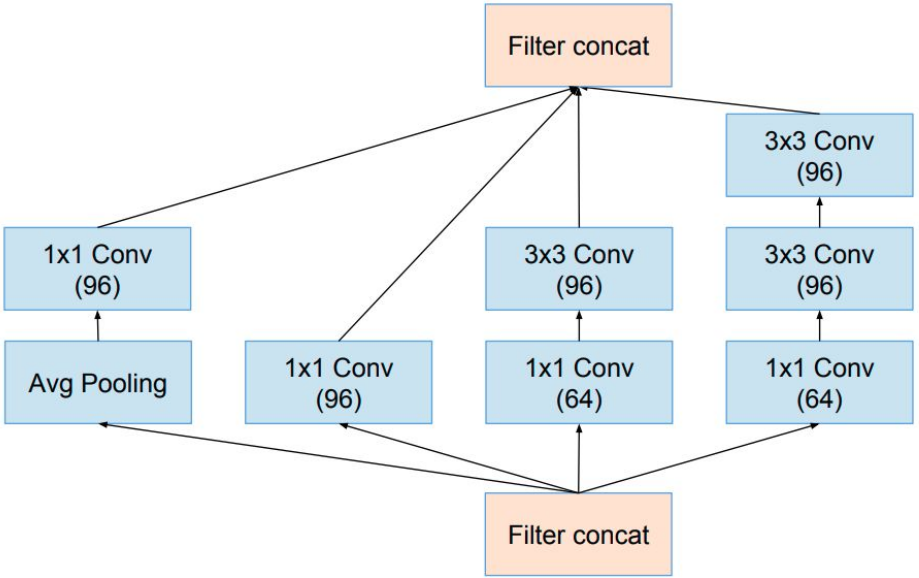


Figure 4. The schema for 35 × 35 grid modules of the pure Inception-v4 network. This is the Inception-A block of Figure 9.

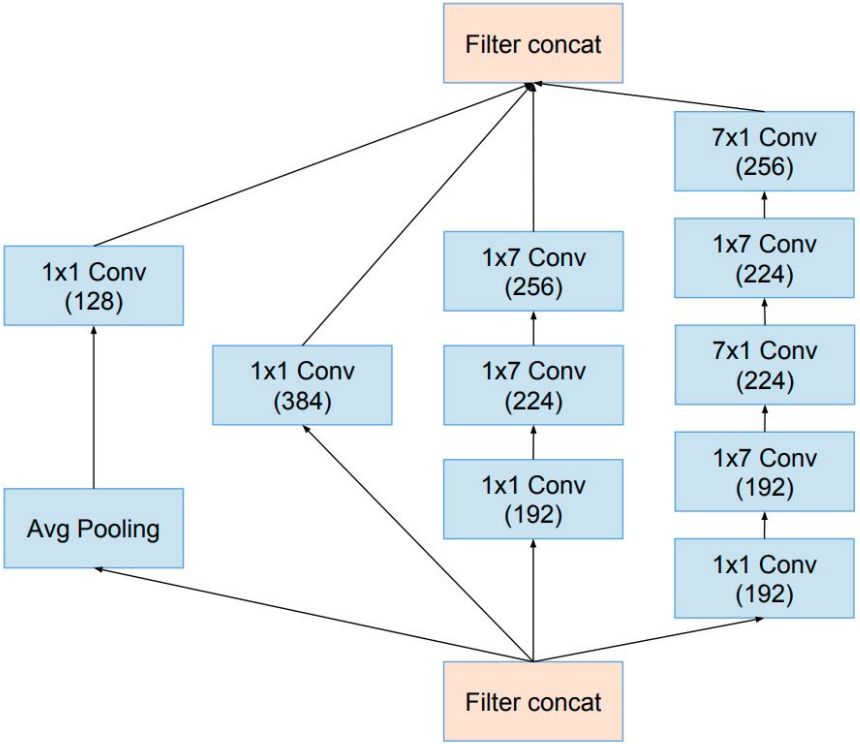


Figure 5. The schema for 17 × 17 grid modules of the pure Inception-v4 network. This is the Inception-B block of Figure 9.

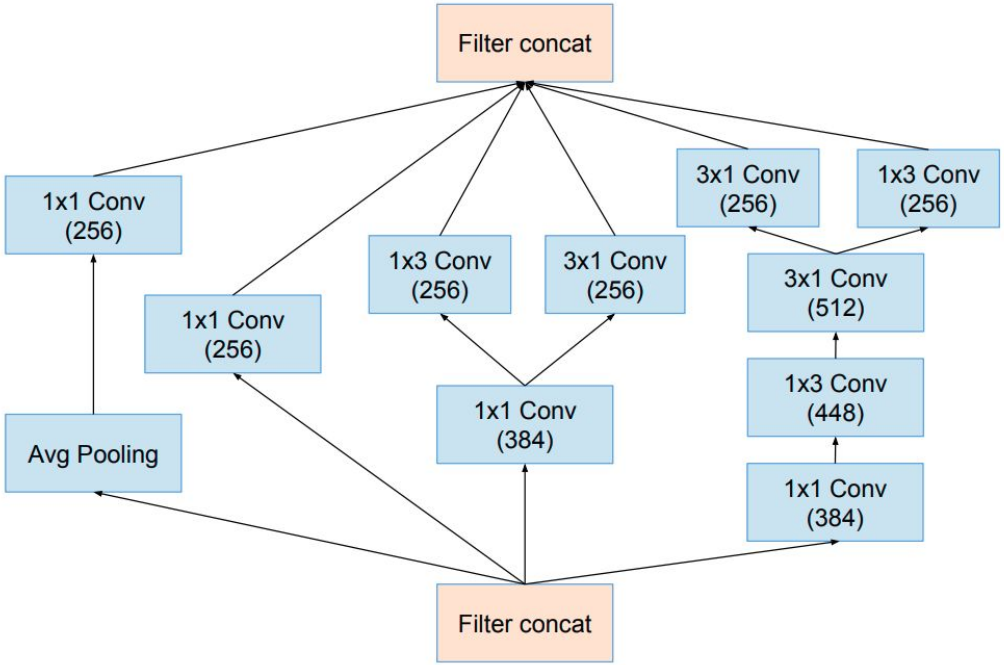
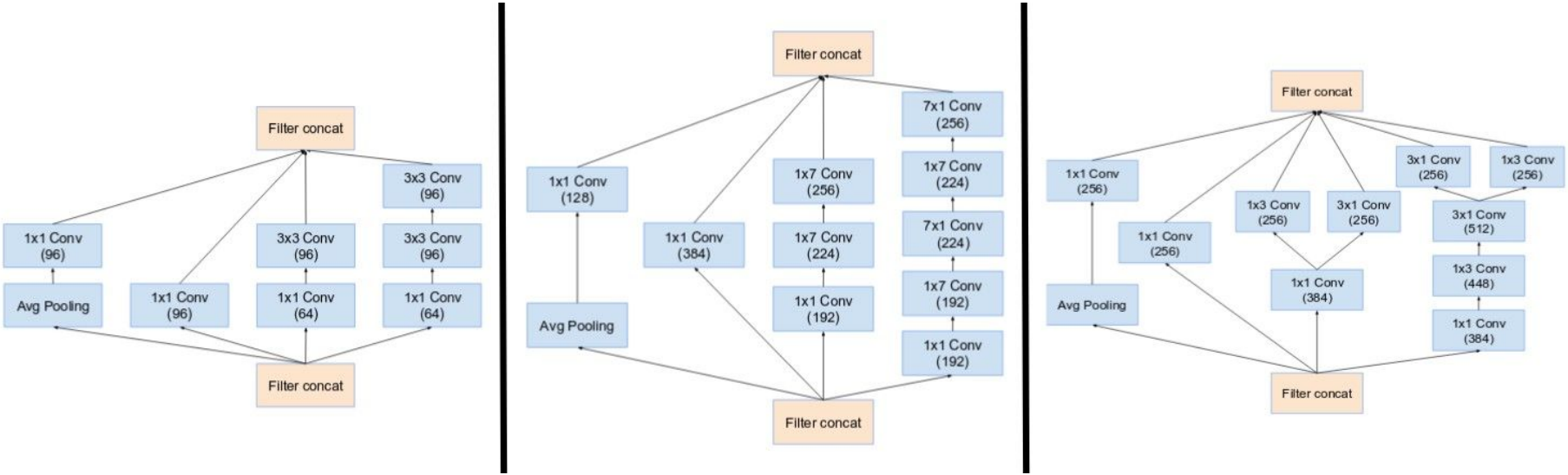
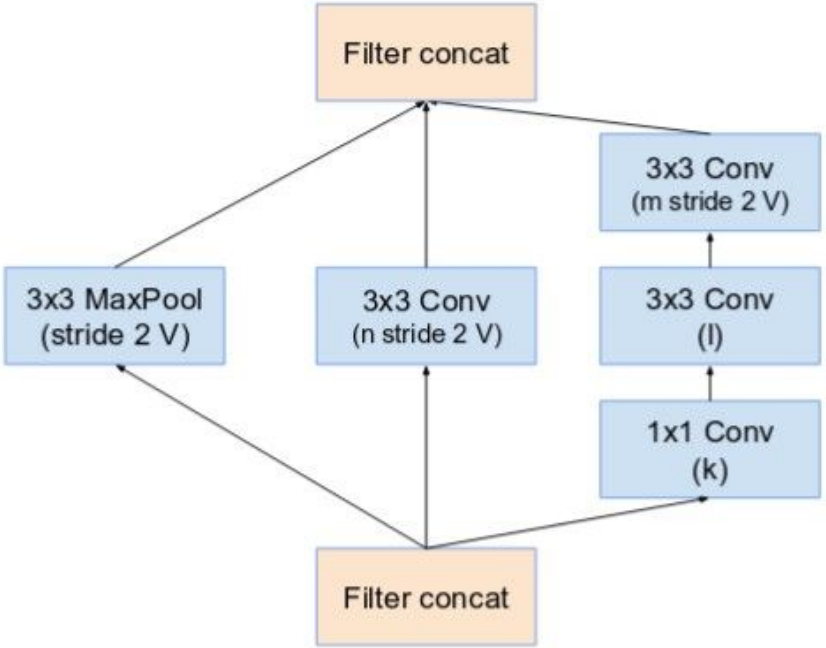
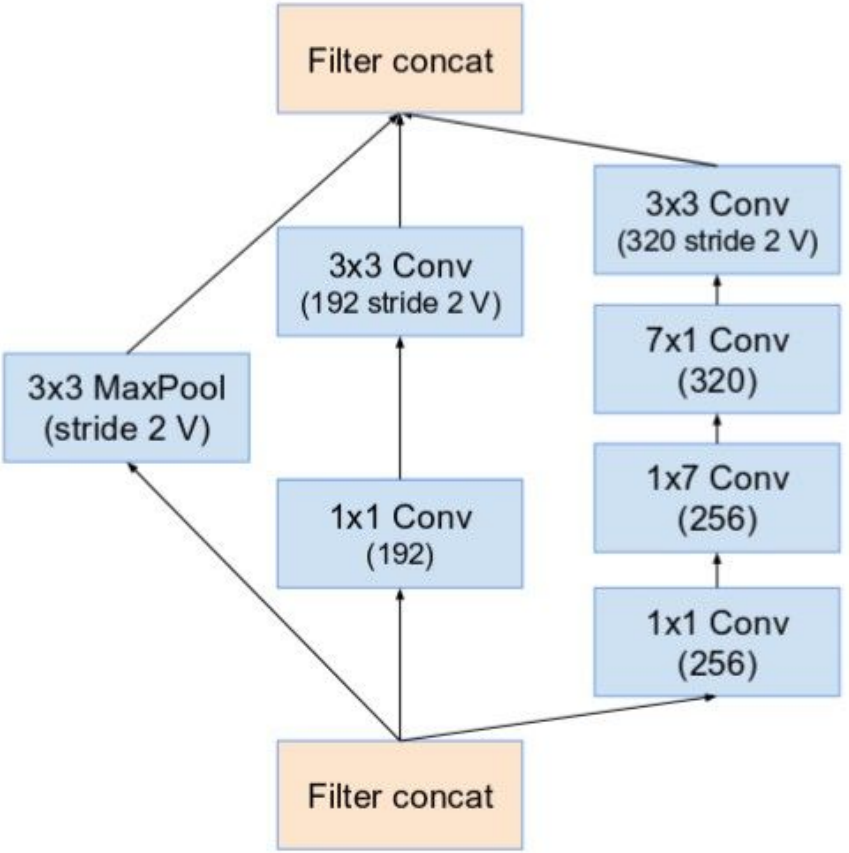


Figure 6. The schema for 8×8 grid modules of the pure Inception-v4 network. This is the Inception-C block of Figure 9.



All the modules together





Reduction Blocks

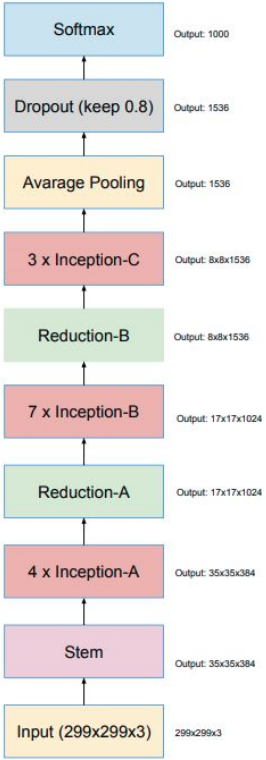
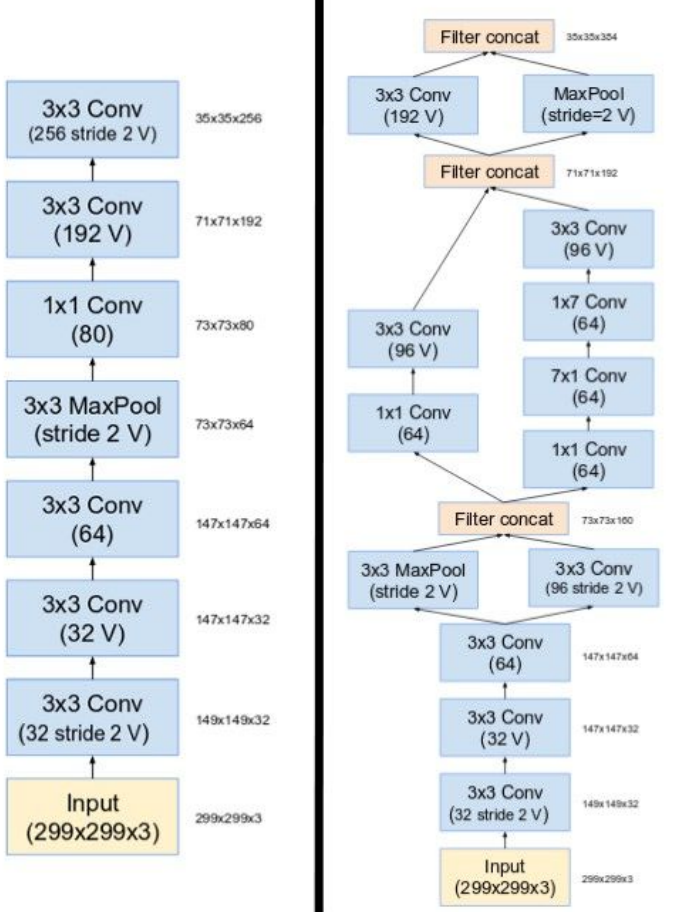


Figure 9. The overall schema of the Inception-v4 network. For the detailed modules, please refer to Figures 3, 4, 5, 6, 7 and 8 for the detailed structure of the various components.

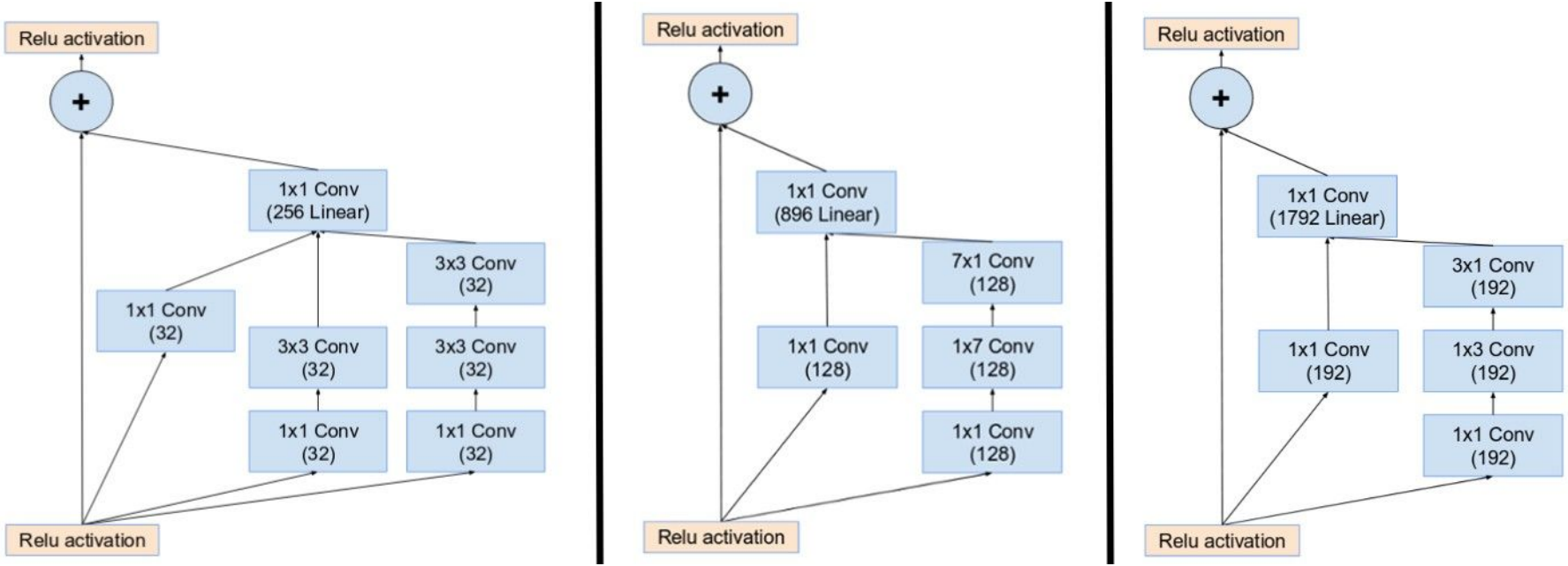
Overall Architecture

Inception - Resnet v1 & v2

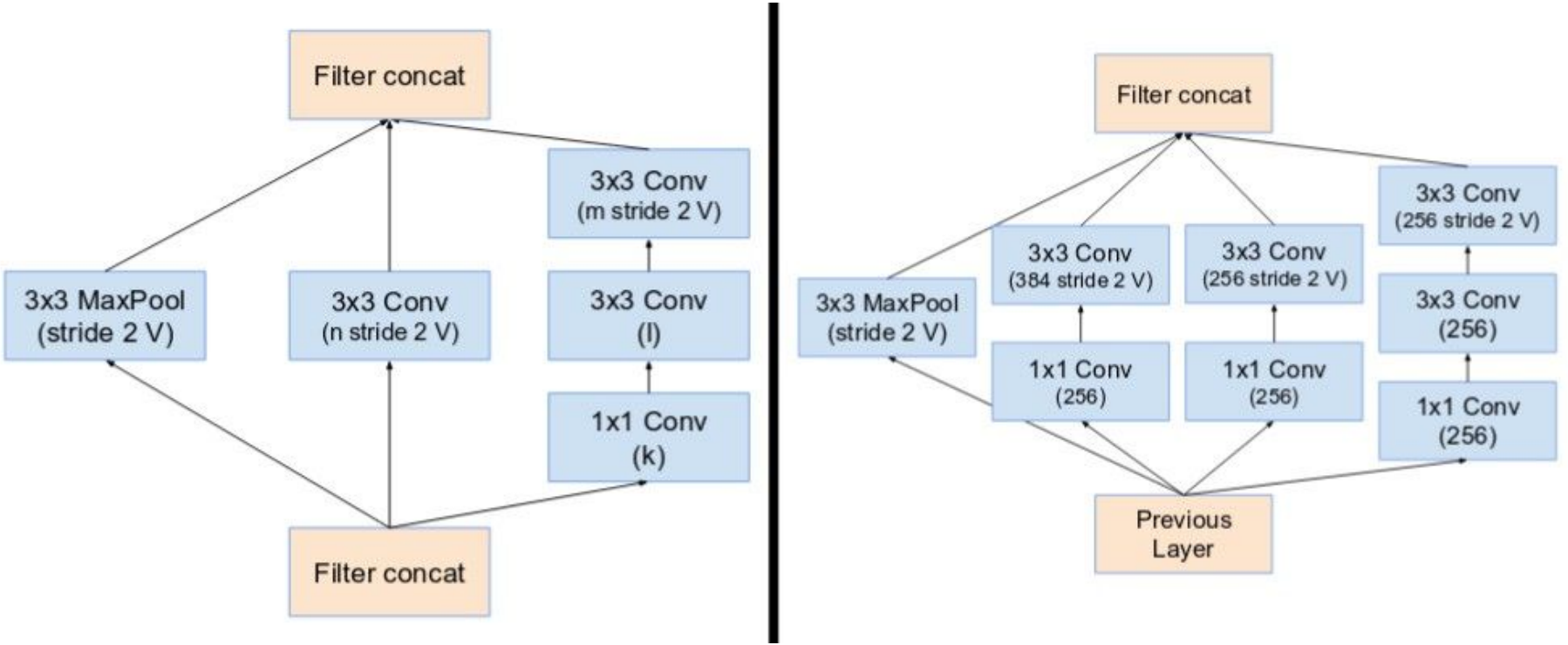
- Inspired by the performance of the ResNet
- They have **different stems**
- Both sub-versions have the **same structure** for the **modules A, B, C** and the **reduction blocks**.
- Only **difference** is the **hyper-parameter** settings.



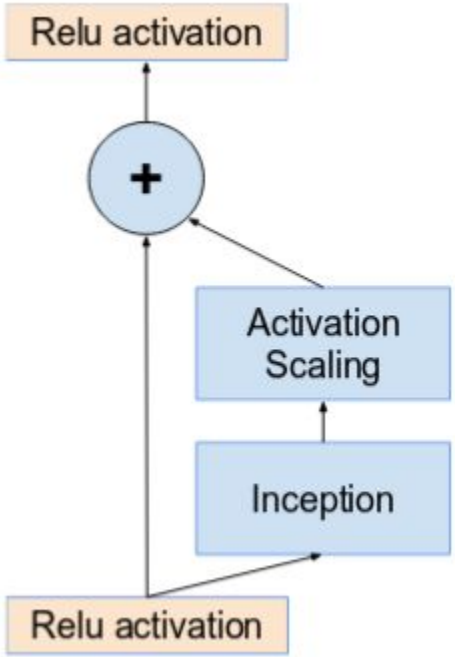
Left v1 and right is v2 and inception v4



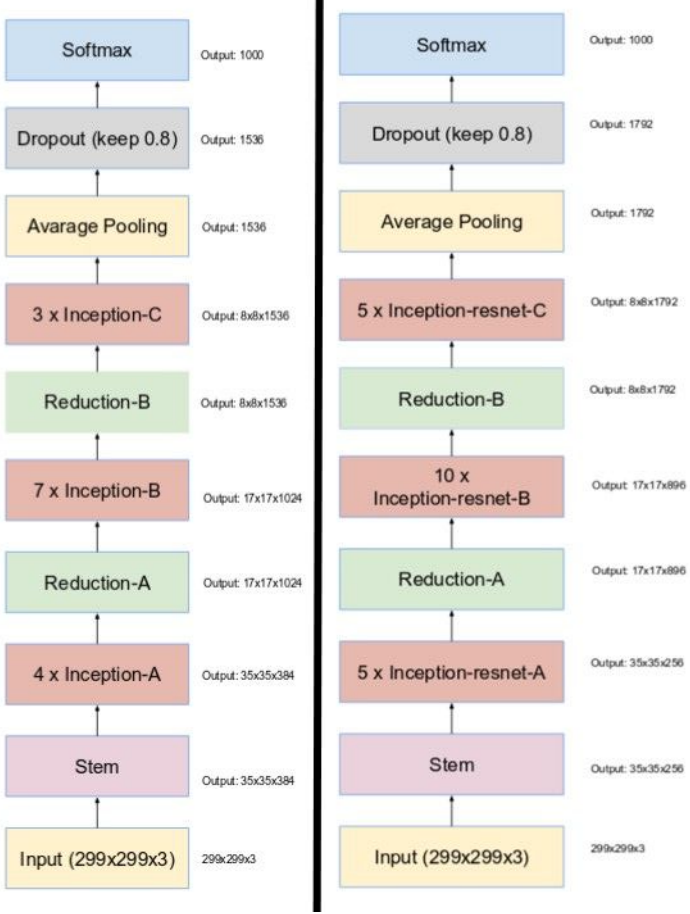
Note how the pooling layer was replaced by the residual connection, and also the additional 1x1 convolution before addition



Reduction Block A (35x35 to 17x17 size reduction) and Reduction Block B (17x17 to 8x8 size reduction).



Activations are scaled by a constant to prevent the network from dying



Inception v4 and Inception-ResNet.