Terminology

Sungjoon Choi
(sungjoon.choi@cpslab.snu.ac.kr)

Terminologies are Important

Input Data


Dies ist ein Blindtext. An ihm lässt sich
vieles über die Schrift ablesen, in der er ge-
setzt ist. Auf den ersten Blick wird der
Grauwert der Schriftfläche sichtbar. Dann

Output / Class / Label

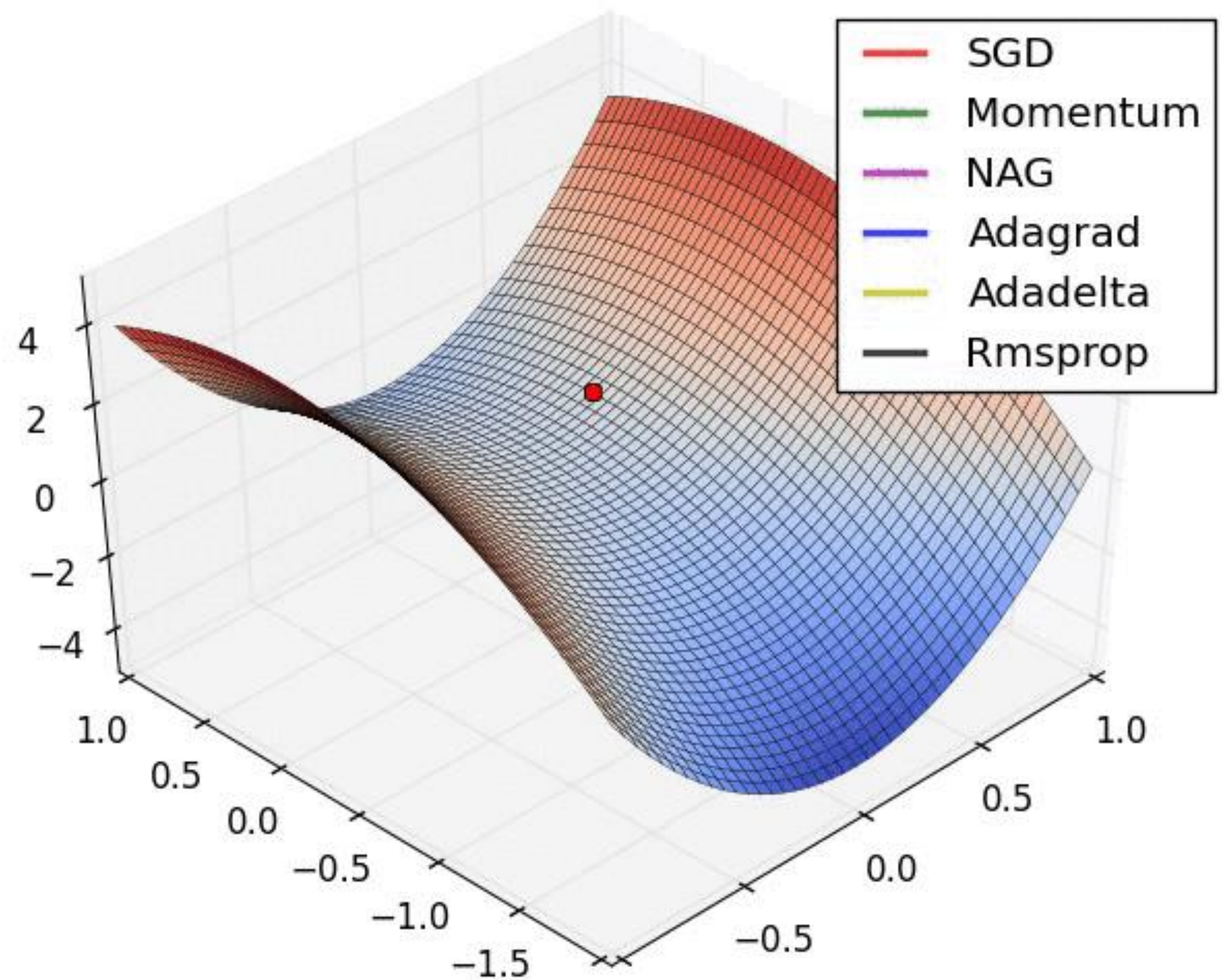



Cat	Dog	Cow	Horse
[1 0 0 0]	[0 1 0 0]	[0 0 1 0]	[0 0 0 1]

One-hot coding

Cat	Dog	Cow	Horse
[1 0 0 0]	[0 1 0 0]	[0 0 1 0]	[0 0 0 1]

Training / Learning



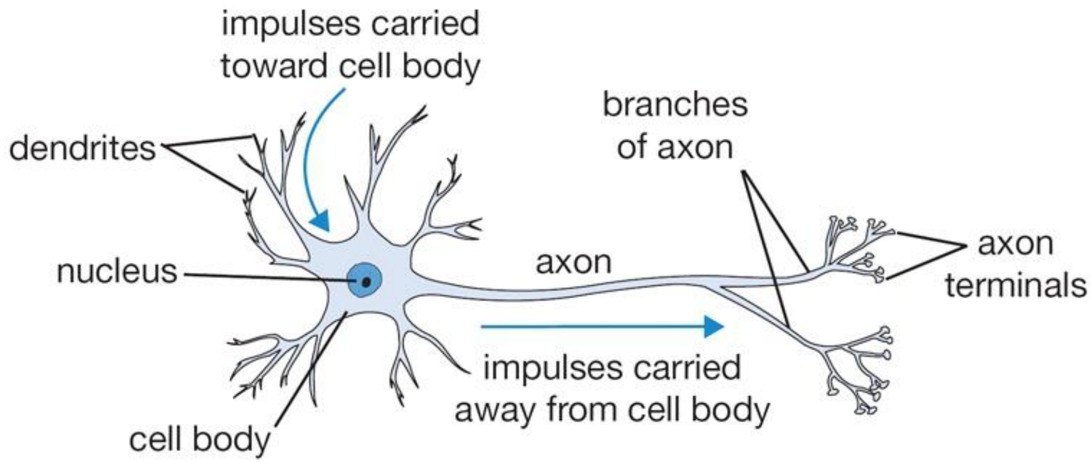
Dataset

Training Data

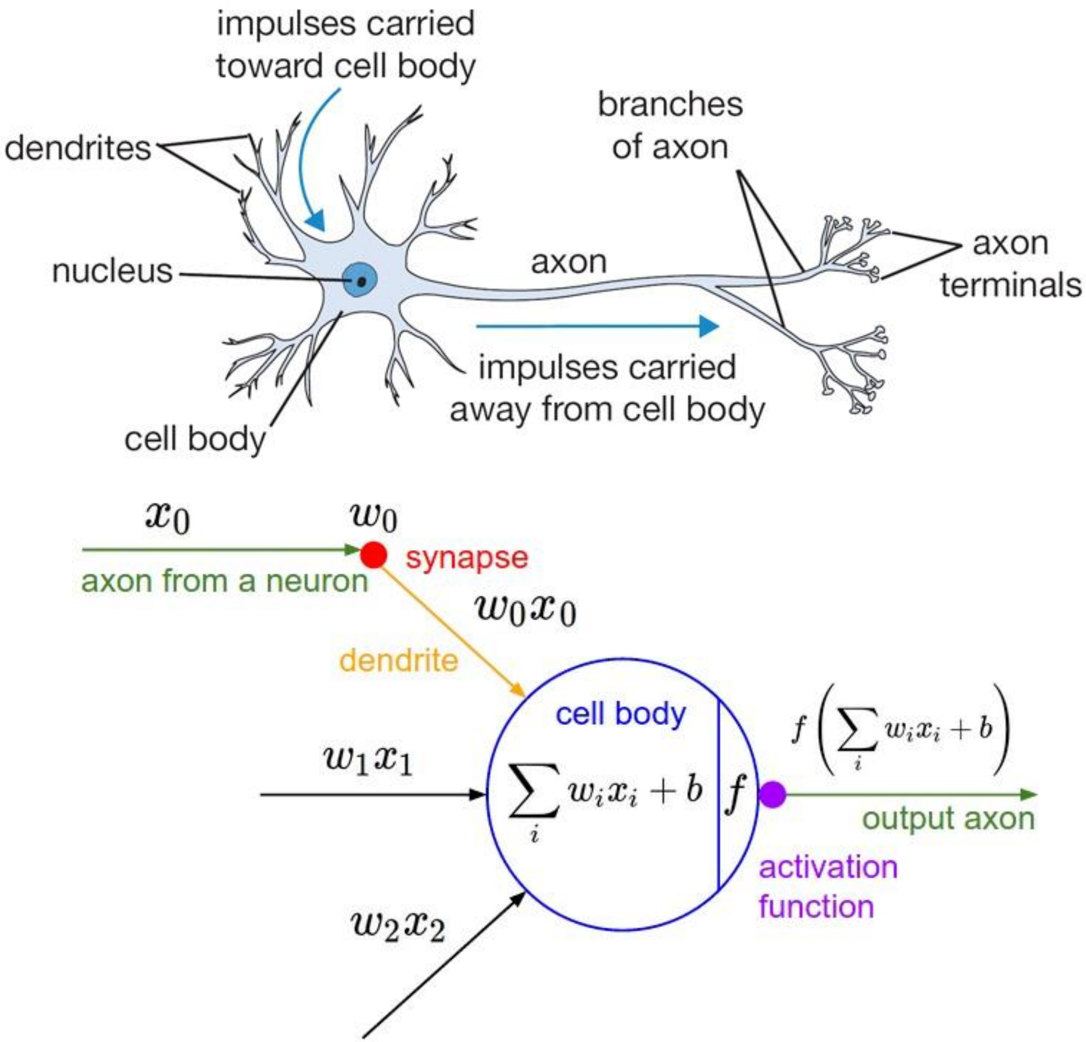
Validation Data (Cross-validation)

Test Data

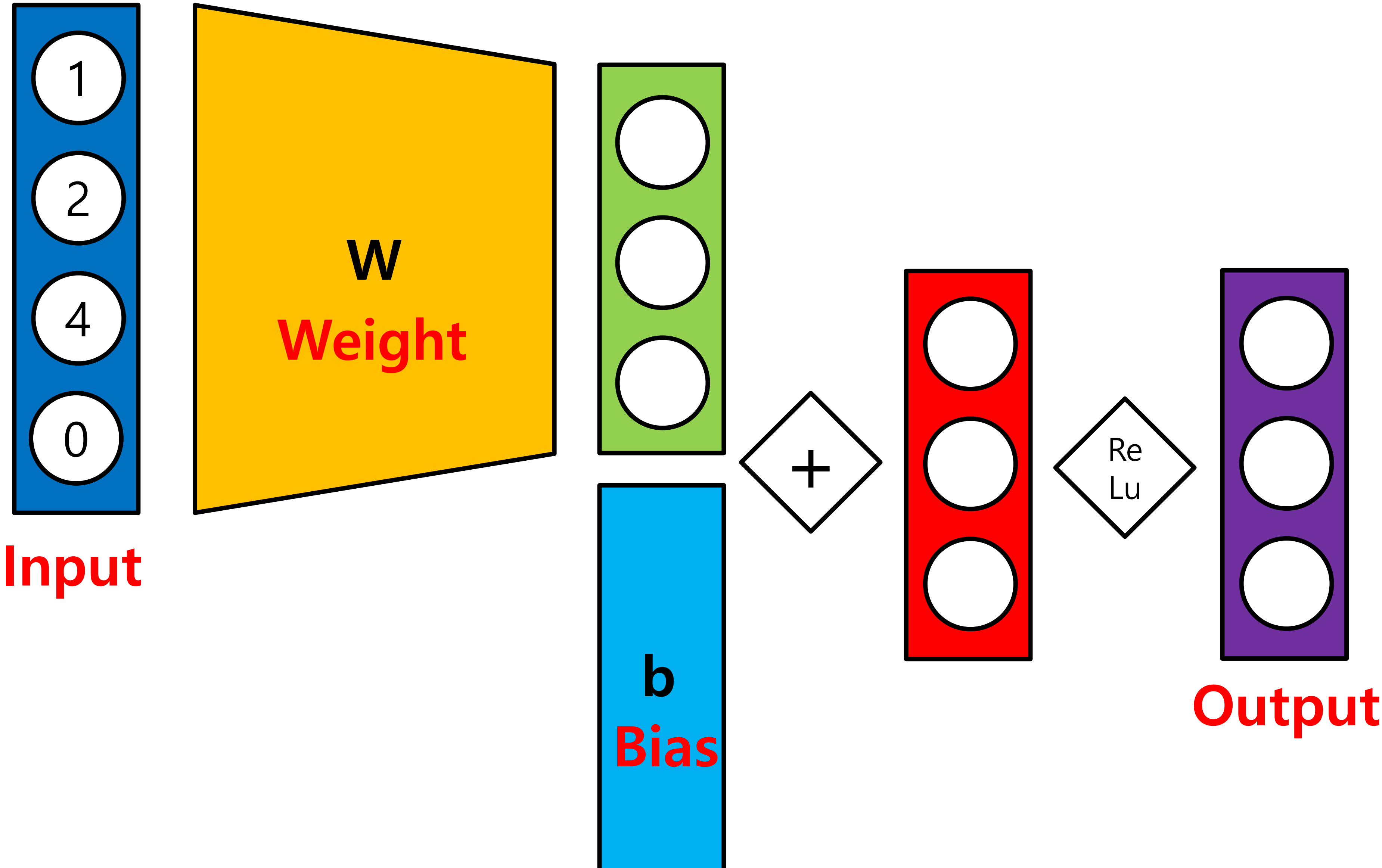
Neuron


impulses carried
toward cell body
dendrites
branches
of axon
nucleus
axon
axon
terminals
impulses carried
away from cell body
cell body

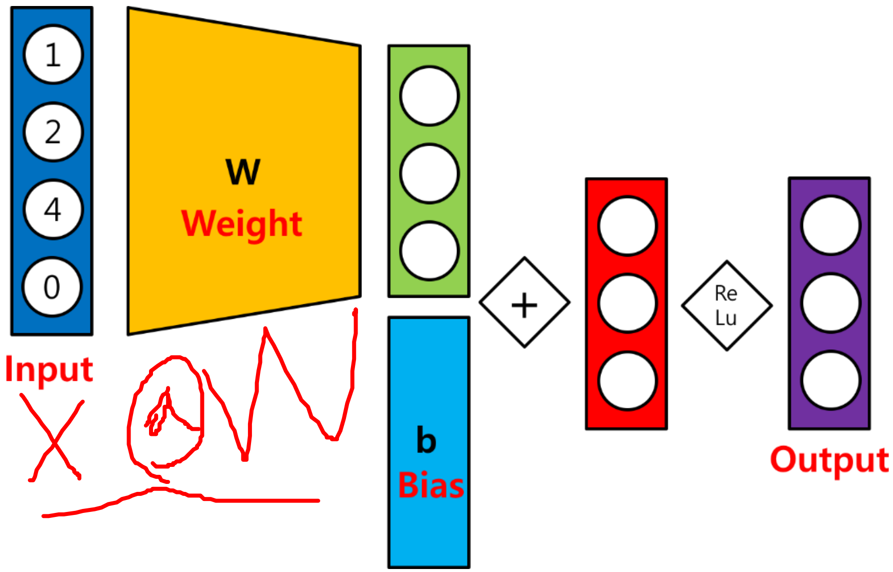
Perceptron


impulses carried
toward cell body
dendrites
branches
of axon
nucleus
axon
axon
terminals
impulses carried
away from cell body
cell body
x_0
w_0
synapse
axon from a neuron
$w_0 x_0$
dendrite
cell body
$f\left(\sum_i w_i x_i + b\right)$
$w_1 x_1$
$\sum_i w_i x_i + b$
f
output axon
activation
function
$w_2 x_2$

Basic single layer network



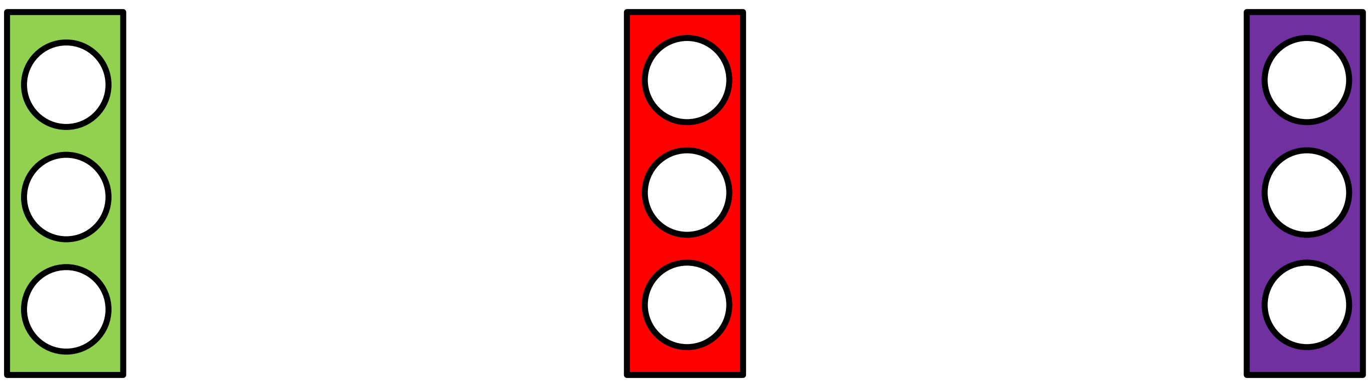
Basic single layer network



Input: $[1 \quad 2 \quad 4 \quad 0]$ (1,4)

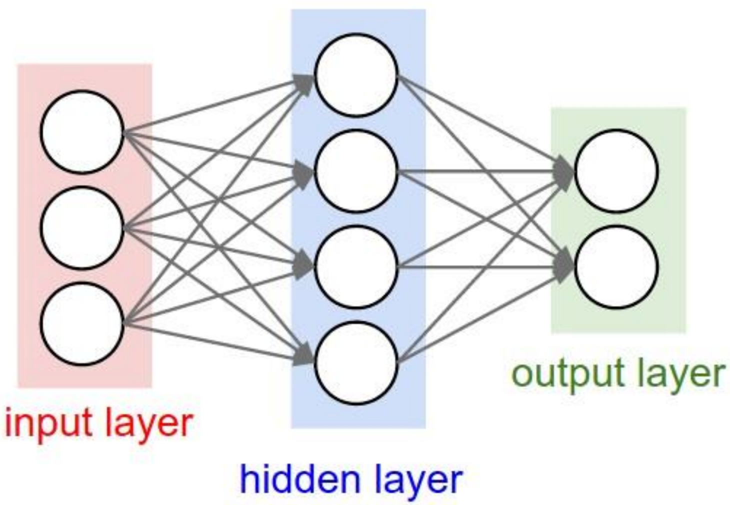
W: $\begin{bmatrix} 1 & 0 & 1 \\ 3 & 0 & 2 \\ -2 & -2 & 0 \\ 1 & 1 & 0 \end{bmatrix}$ (4,3)

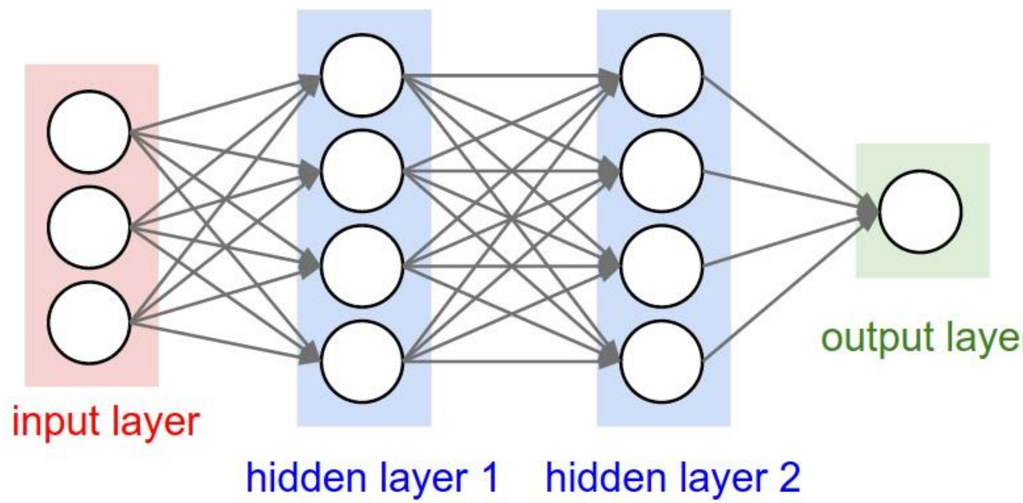
Bias: $[2 \quad 1 \quad 3]$ (1,3)



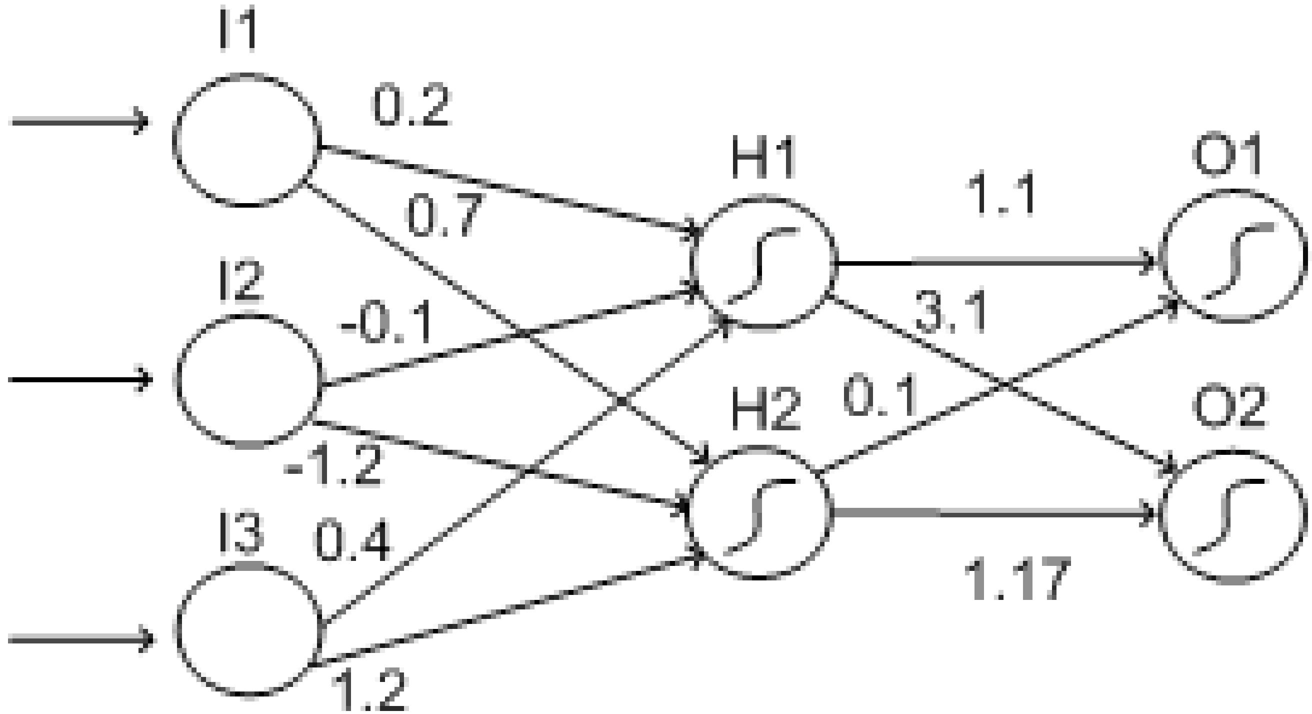
$[-1 \quad -8 \quad 5]$ $\qquad$ $[1 \quad -7 \quad 8]$ $\qquad$ $[1 \quad 0 \quad 8]$

(Neural) Network

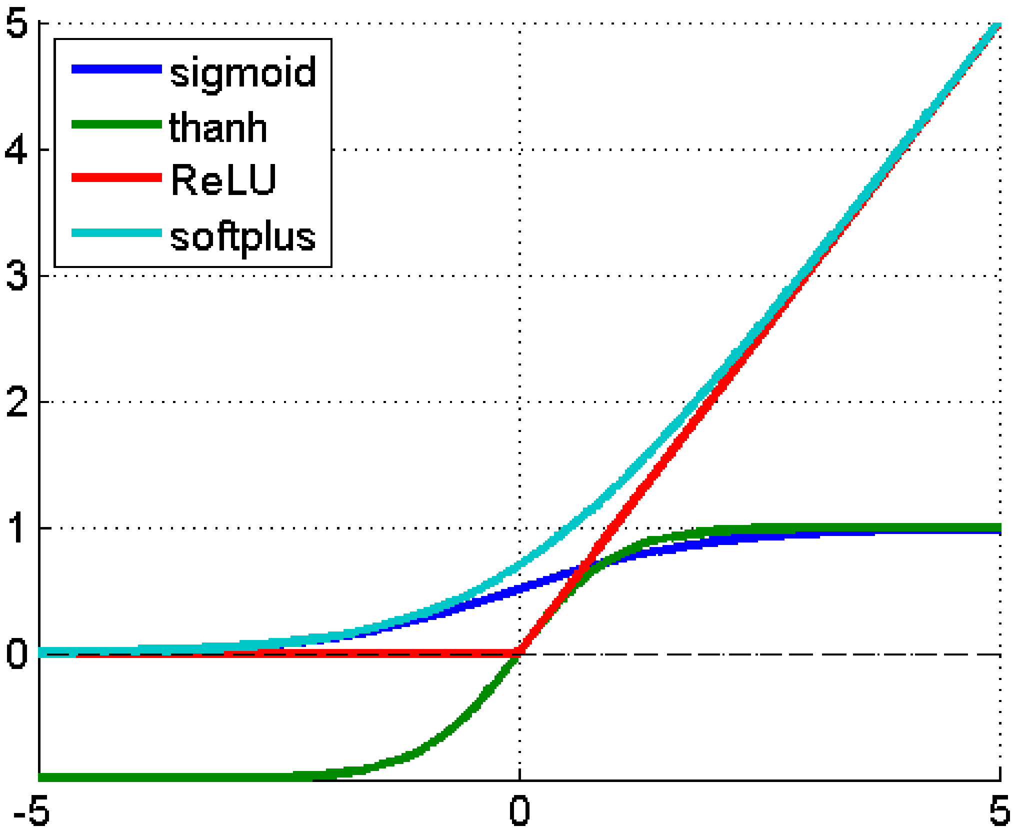




Multi-Layer Perceptron



Activation Function



Epoch / Batch size / Iteration

One **epoch**: one forward and backward pass of **all training data**

Batch size: the number of training examples in **one forward and backward pass**

One **iteration**: number of passes

If we have 55,000 training data, and the batch size is 1,000. Then, we need 55 iterations to complete 1 epoch.

Cost function

This is just a function that we want to minimize.

There is **no guarantee** that it will bring us to the best solution!

It should be **differentiable**!

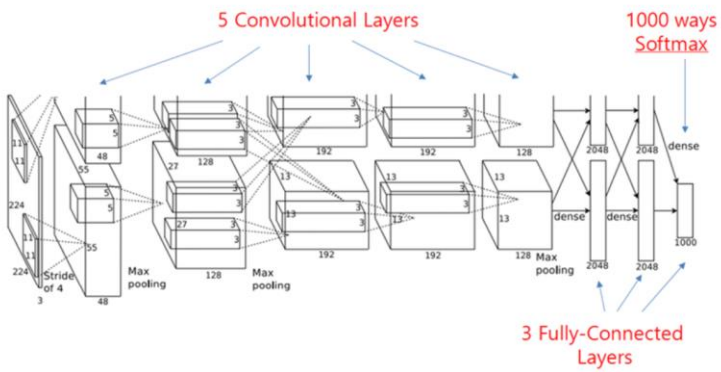
MNIST (Mixed National Institute of Standards and Technology)

Modern CNNs

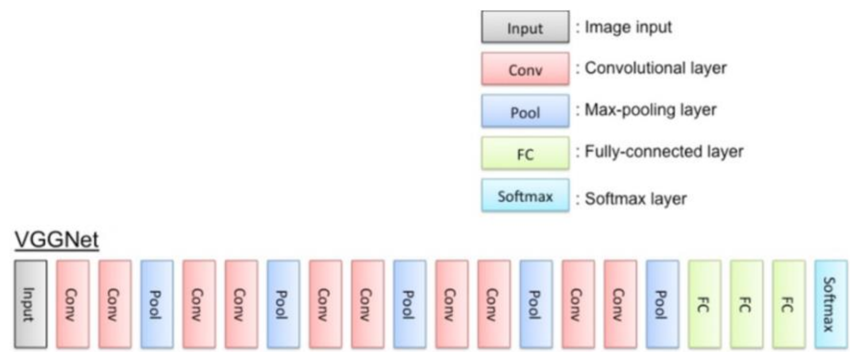
Sungjoon Choi
(sungjoon.choi@cpslab.snu.ac.kr)

CNN Architectures

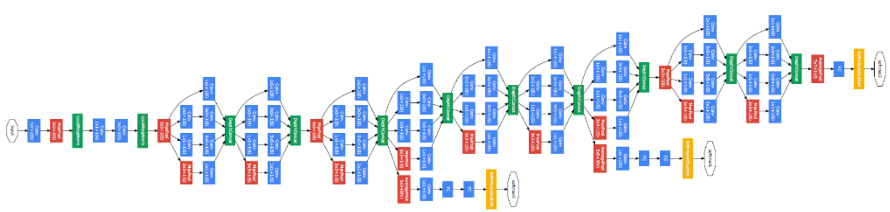
AlexNet



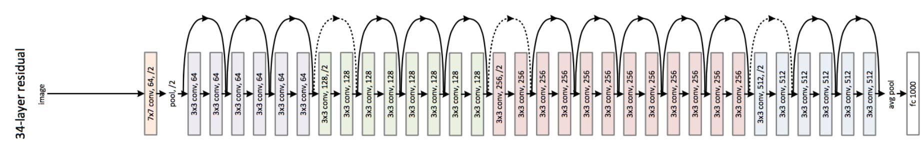
VGG



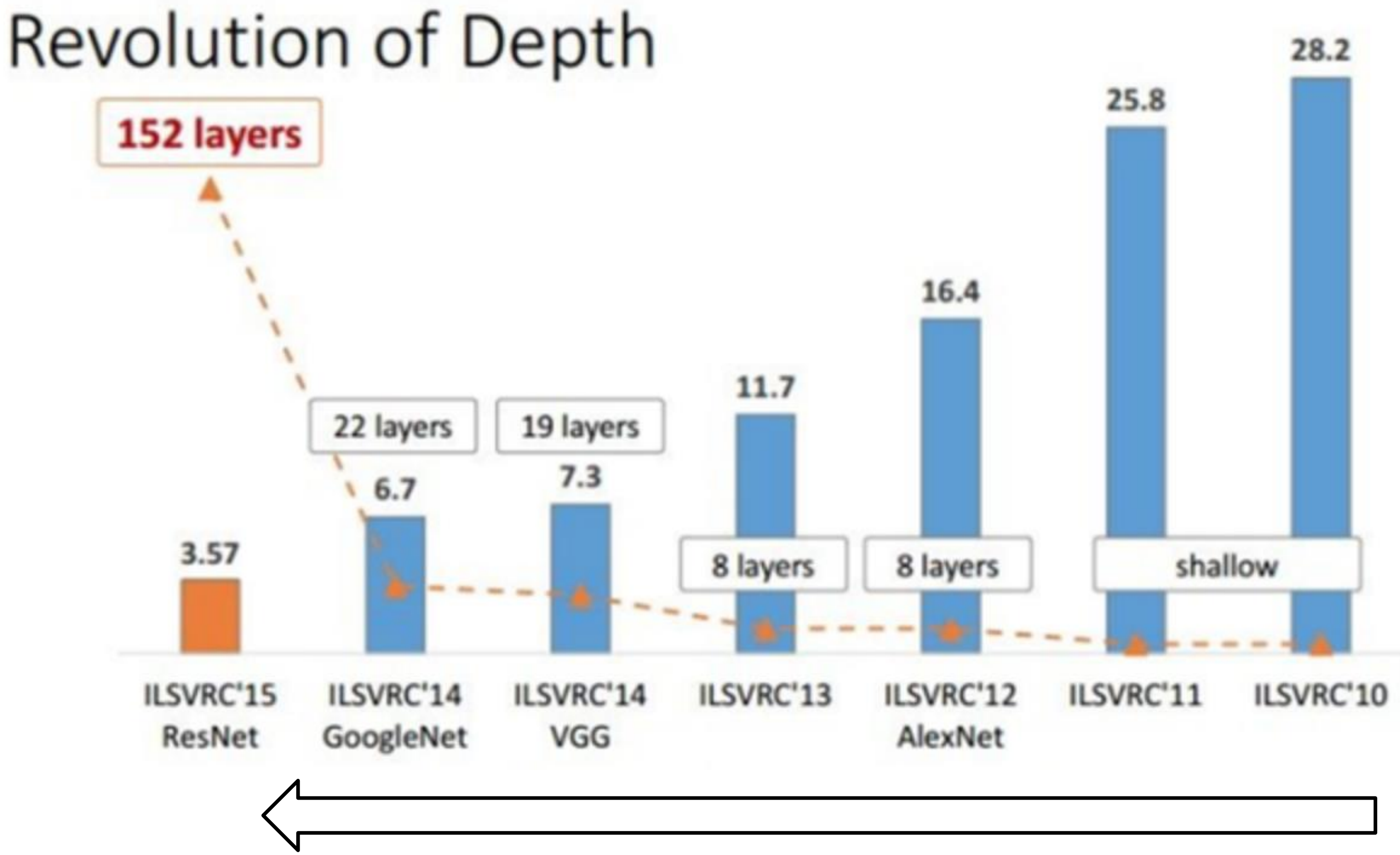
GoogLeNet



ResNet

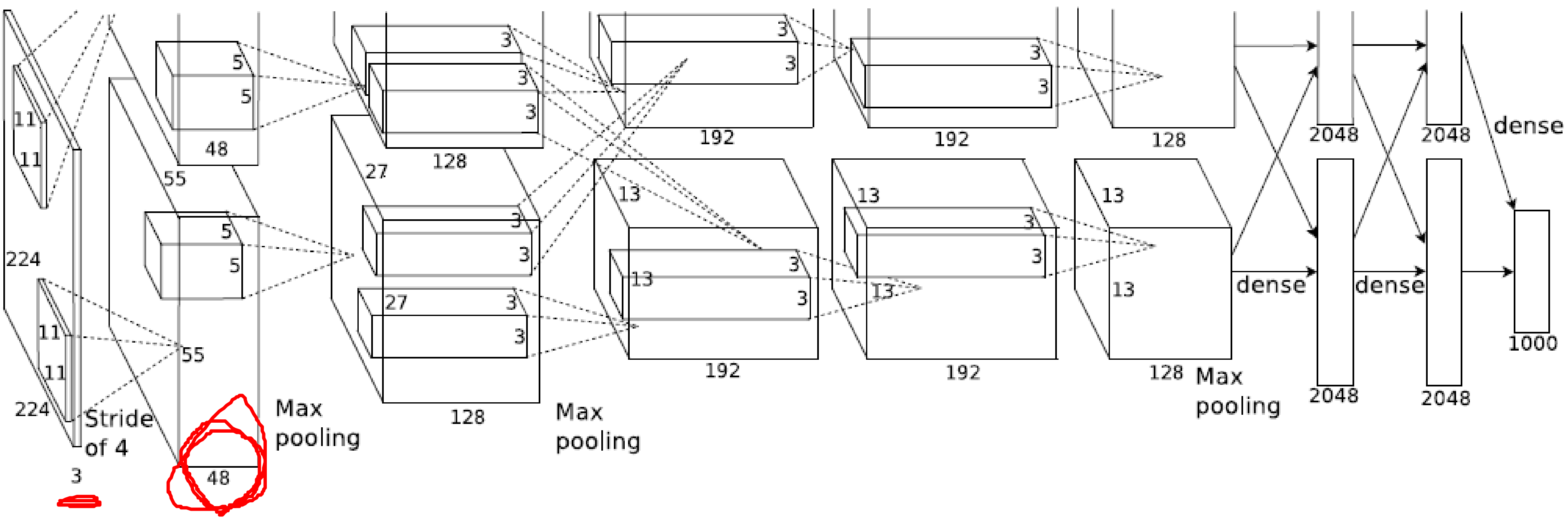


Top-5 Classification Error



AlexNet

AlexNet



What is the number of parameters?

Why are layers divided into two parts?

AlexNet

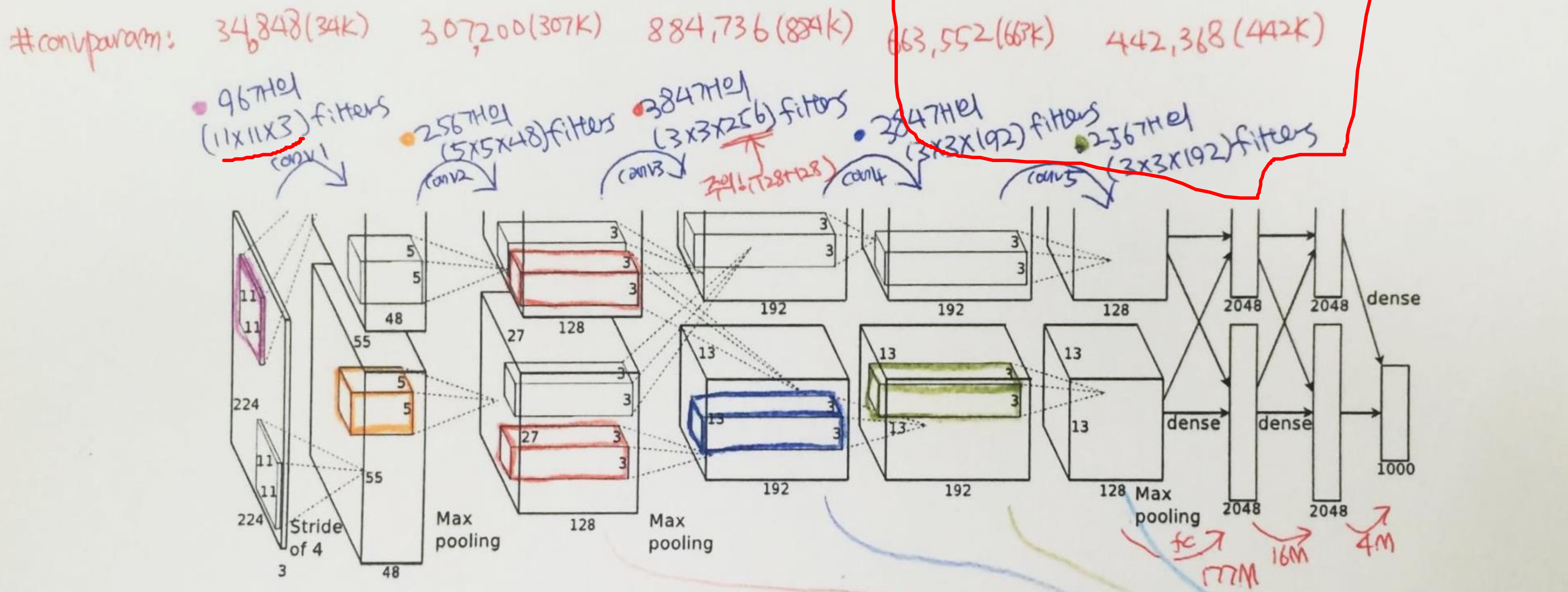
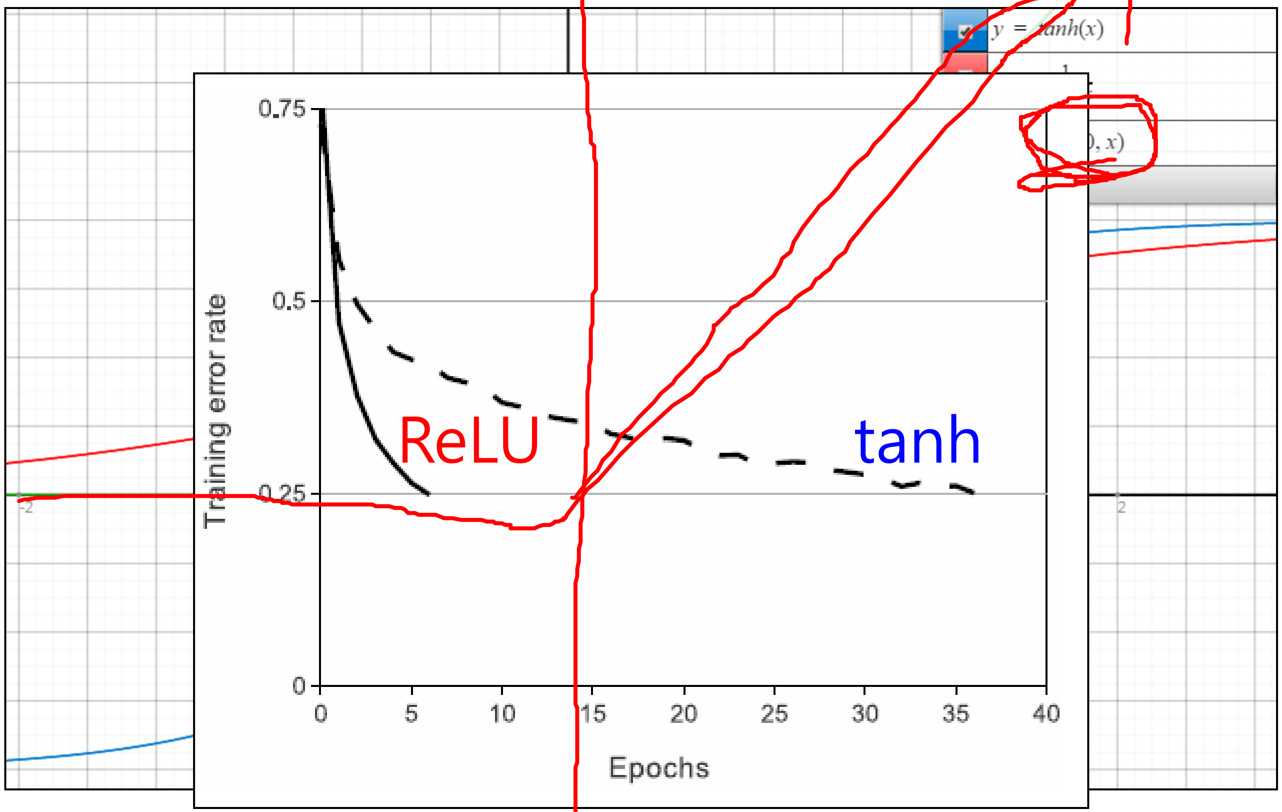


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

ReLU

Rectified **L**inear **U**nit



Faster Convergence!

LRN

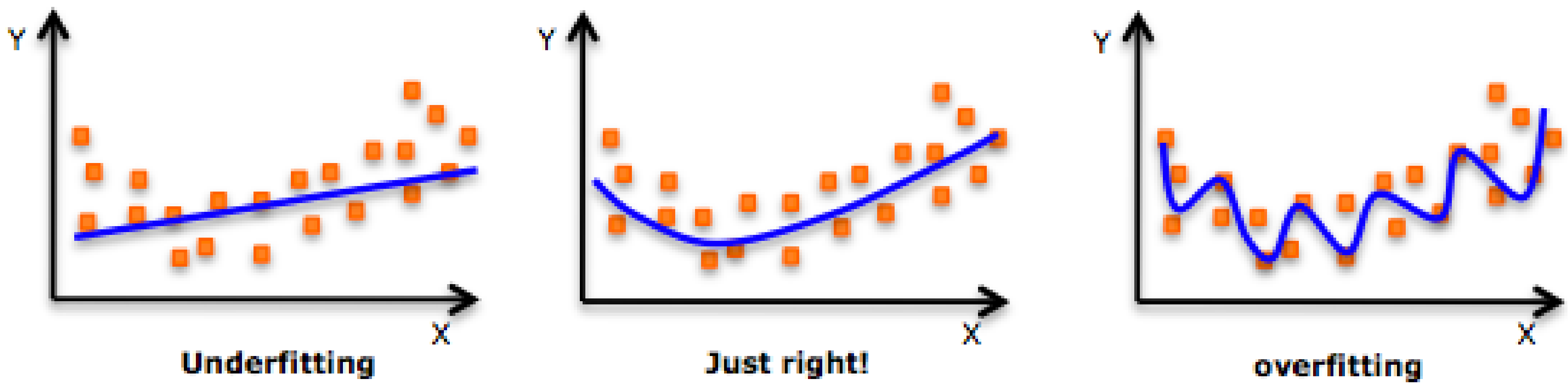
Local **R**esponse **N**ormalization

The response-normalized activity is given by:

$$b_{x,y}^{i} = \frac{a_{x,y}^{i}}{\left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} \left(a_{x,y}^{j}\right)^{2}\right)^{\beta}}$$

It implements a form of **lateral inhibition** inspired by real neurons.

Regularization in AlexNet



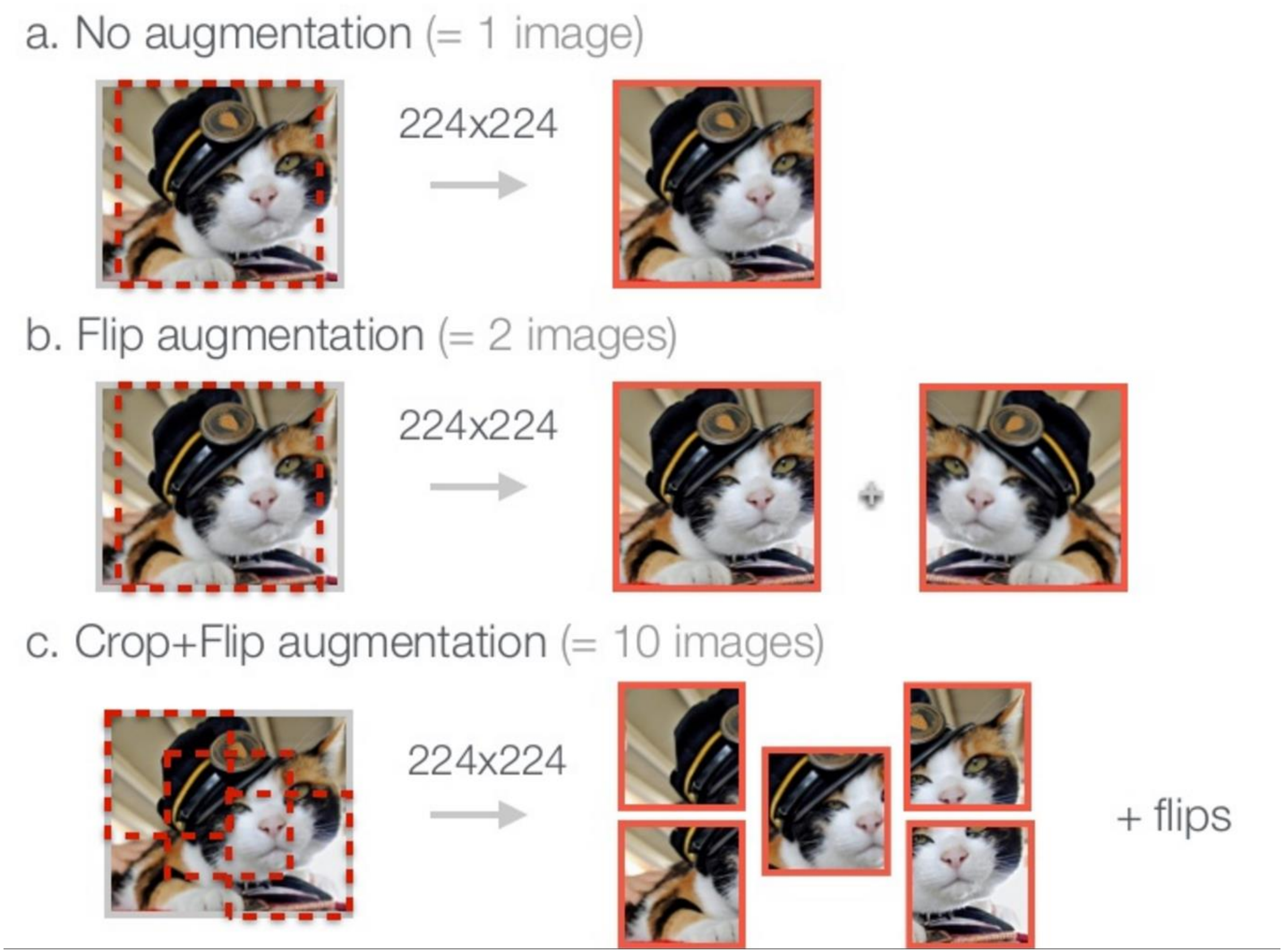
Main objective is to reduce **overfitting**.

More details will be handled in next week.

In the **AlexNet**, two regularization methods are used.

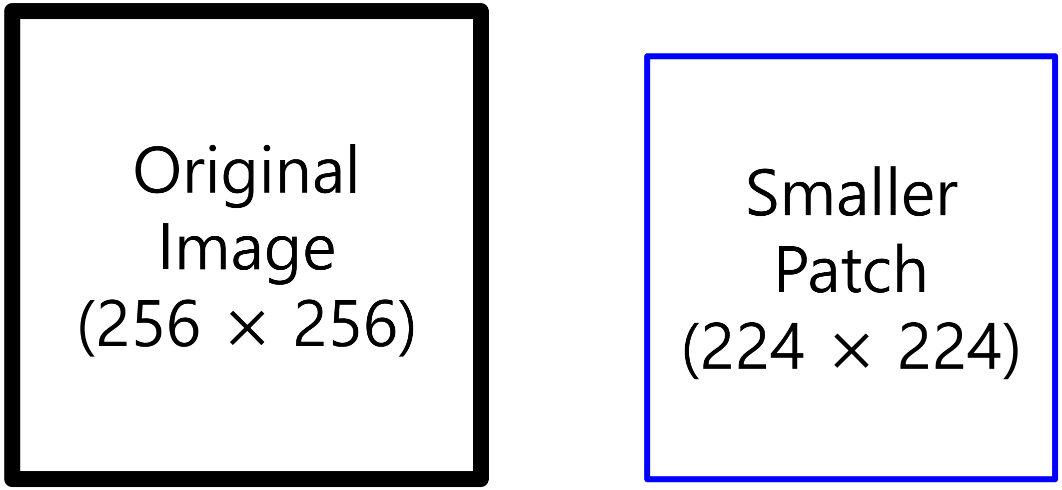
- Data augmentation
- Dropout

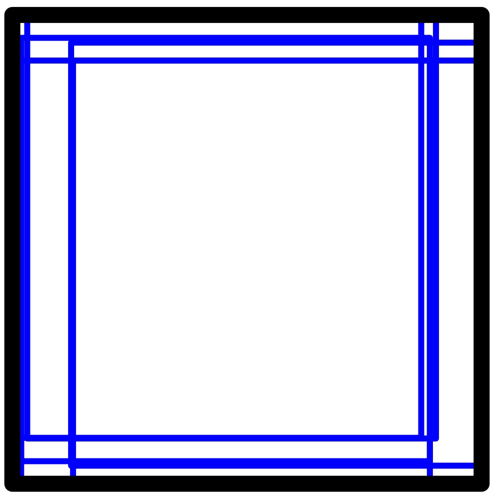
Data augmentation



http://www.slideshare.net/KenChatfield/chatfield14-devil

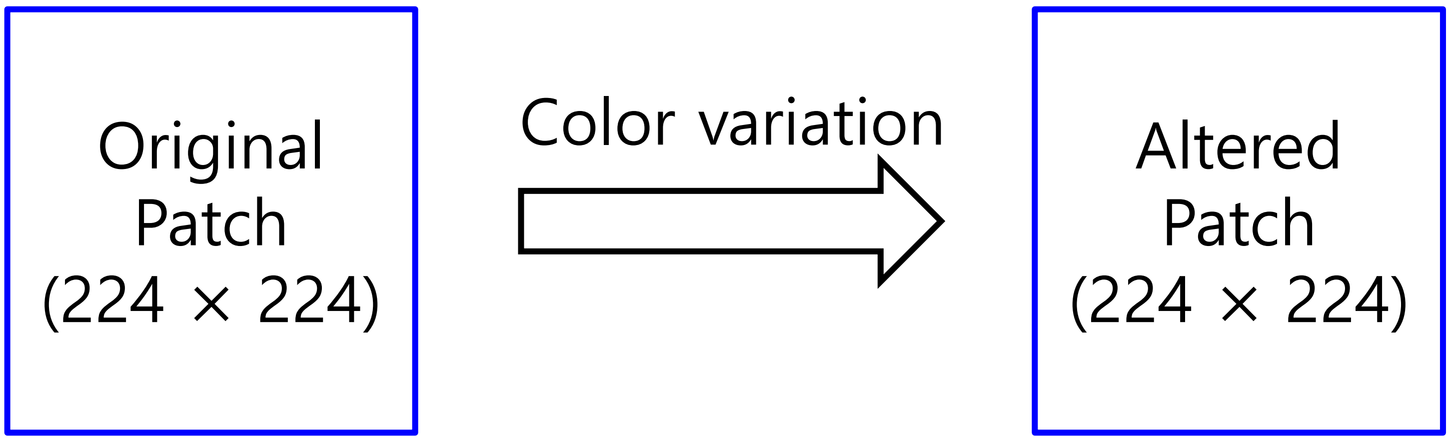
Data augmentation in AlexNet





This increases the size of the training set by a **factor of 2048** $(32 * 32 * 2)$.
Two comes from horizontal reflections.

Data augmentation in AlexNet



To each RGB image pixel, following quantity is added:

$$[p_1, p_2, p_3][\alpha_1\lambda_1, \alpha_2\lambda_2, \alpha_3\lambda_3]^T$$

where p_i and λ_i are ith eigenvector and eigenvalue of 3×3 covariance matrix of RGB pixel values.

Probabilistically, not a single patch will be same at the training phase! (a **factor of infinity**!)

Dropout

layer

Original dropout [1] sets the output of each hidden neuron with certain probability.

In this paper, they simply multiply the outputs by 0.5.

[1] G.E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R.R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. arXiv, 2012.

http://www.eddyazar.com/the-regrets-of-a-dropout-and-why-you-should-drop-out-too/

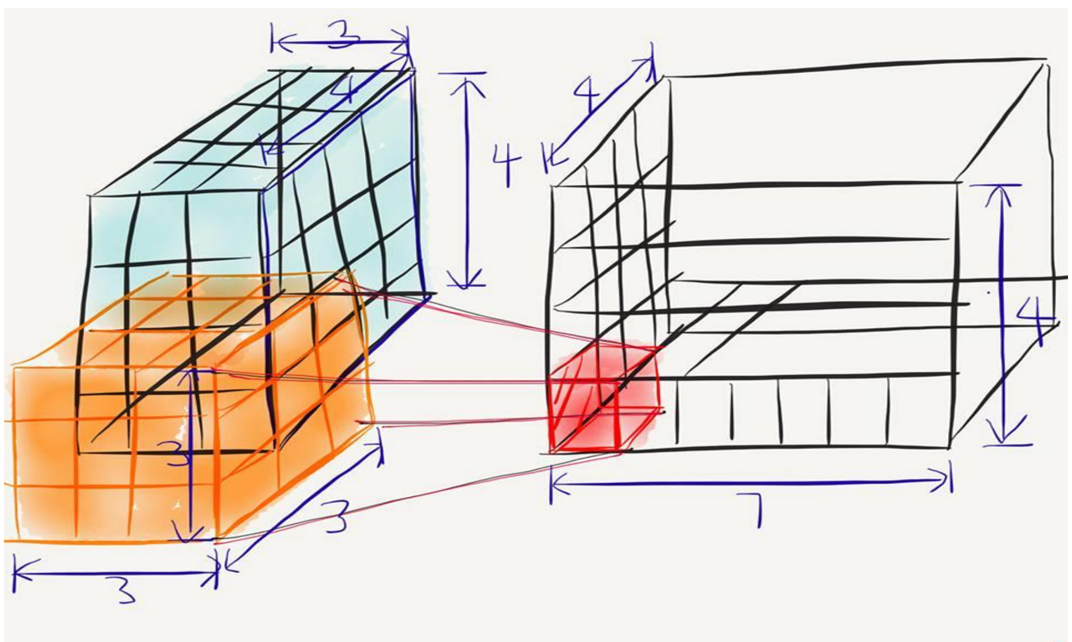
Must remember!

Conv2D

`tf.nn.conv2d(input, filter, strides, padding, use_cudnn_on_gpu=None, name=None)`

Computes a 2-D convolution given 4-D `input` and `filter` tensors.

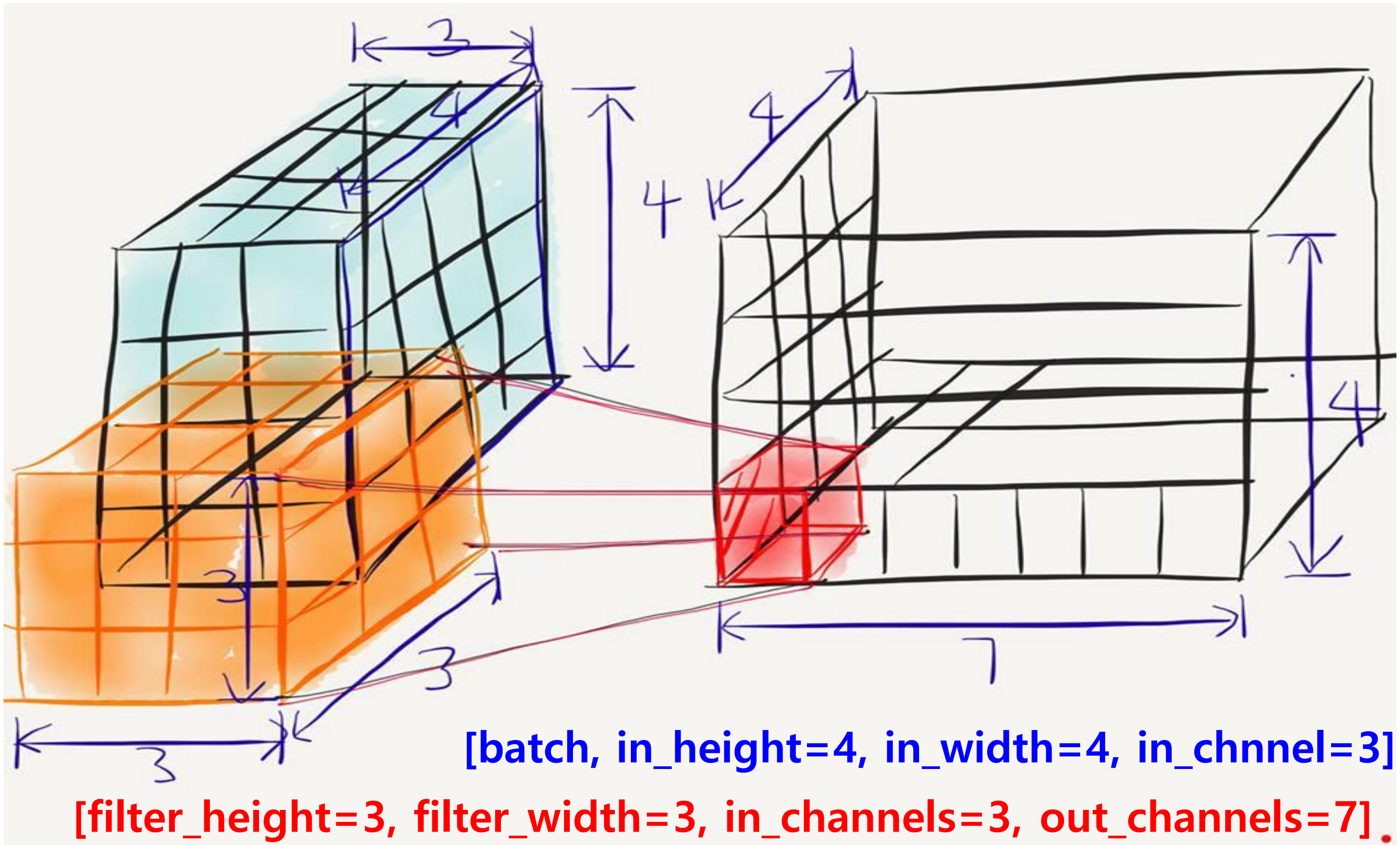
Given an input tensor of shape **[batch, in_height, in_width, in_chnnel]** and a filter / kernel tensor of shape **[filter_height, filter_width, in_channels, out_channels]** this op performs the following:



[batch, in_height=4, in_width=4, in_chnnel=3]

[filter_height=3, filter_width=3, in_channels=3, out_channels=7]

Conv2D



What is the **number of parameters** in this convolution layer?

→ $\mathbf{189 = 3 * 3 * 3 * 7}$

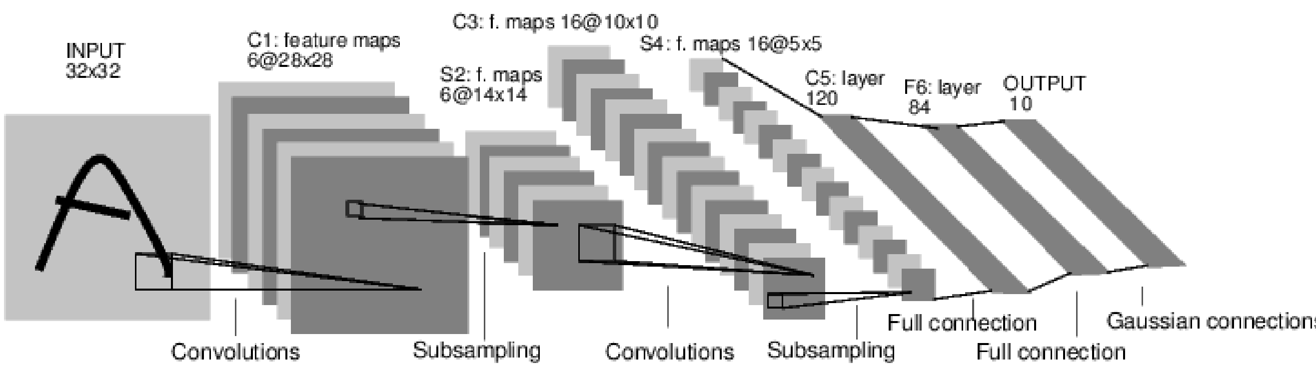
VGG

VGG?

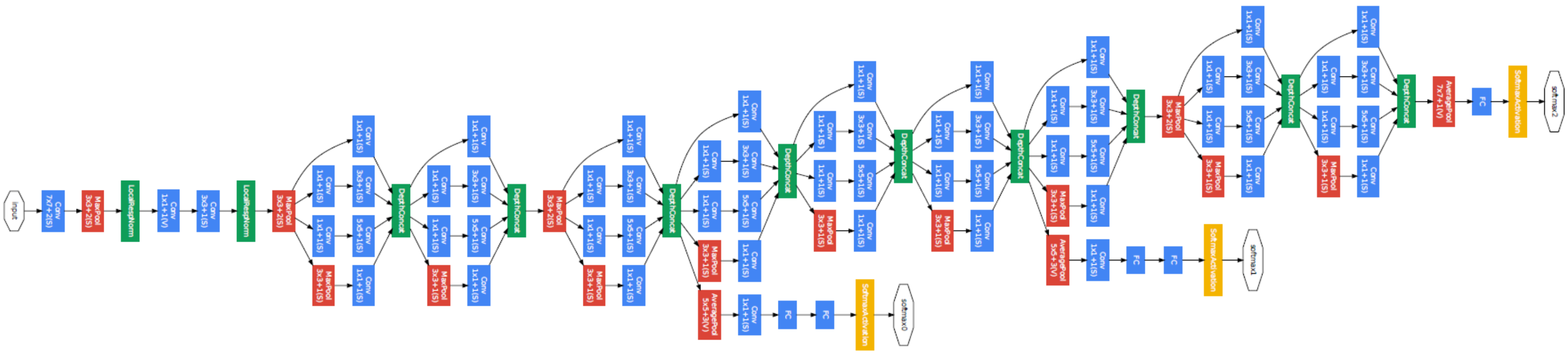
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

GoogLeNet

GoogLeNet


INPUT
32x32
C1: feature maps
6@28x28
S2: f. maps
6@14x14
C3: f. maps 16@10x10
S4: f. maps 16@5x5
C5: layer
120
F6: layer
84
OUTPUT
10
Convolutions
Subsampling
Convolutions
Subsampling
Full connection
Full connection
Gaussian connections

GoogLeNet

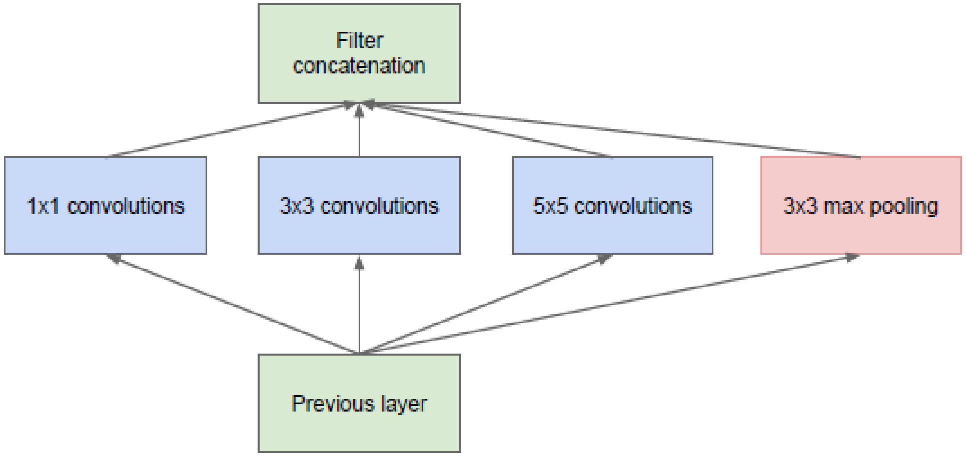


22 Layers Deep Network

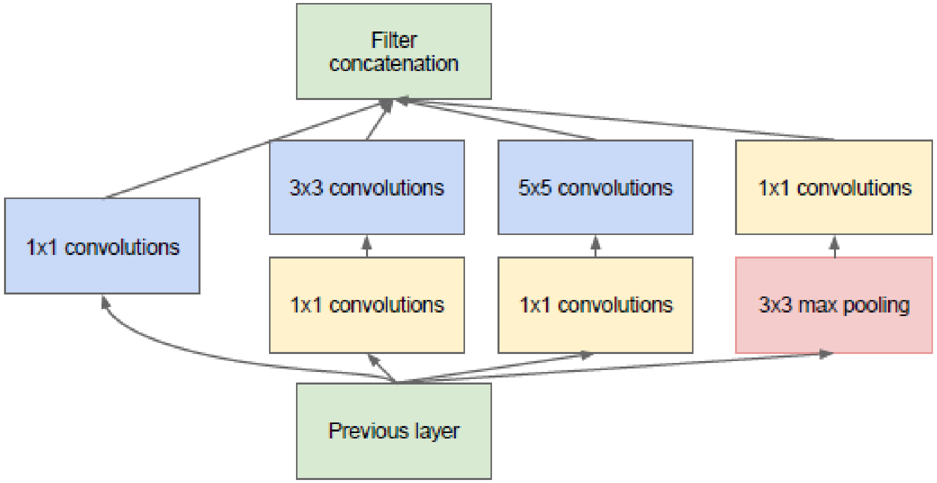
Efficiently utilized computing resources, “Inception Module”

Significantly outperforms previous methods on ILSVRC 2014

Inception module

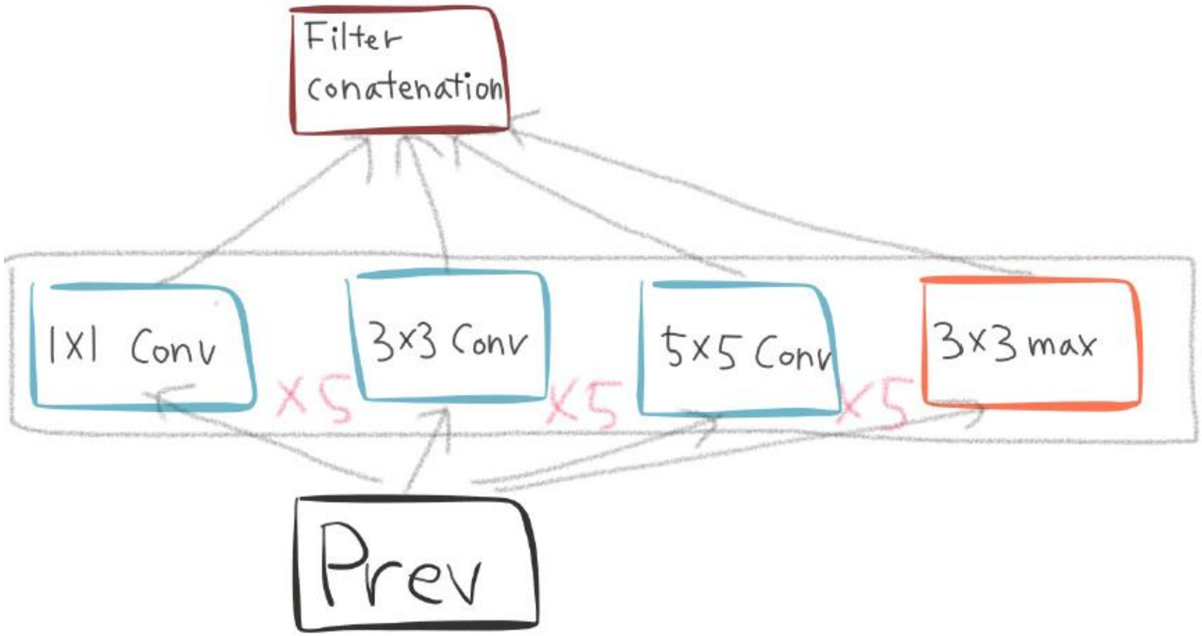


(a) Inception module, naïve version

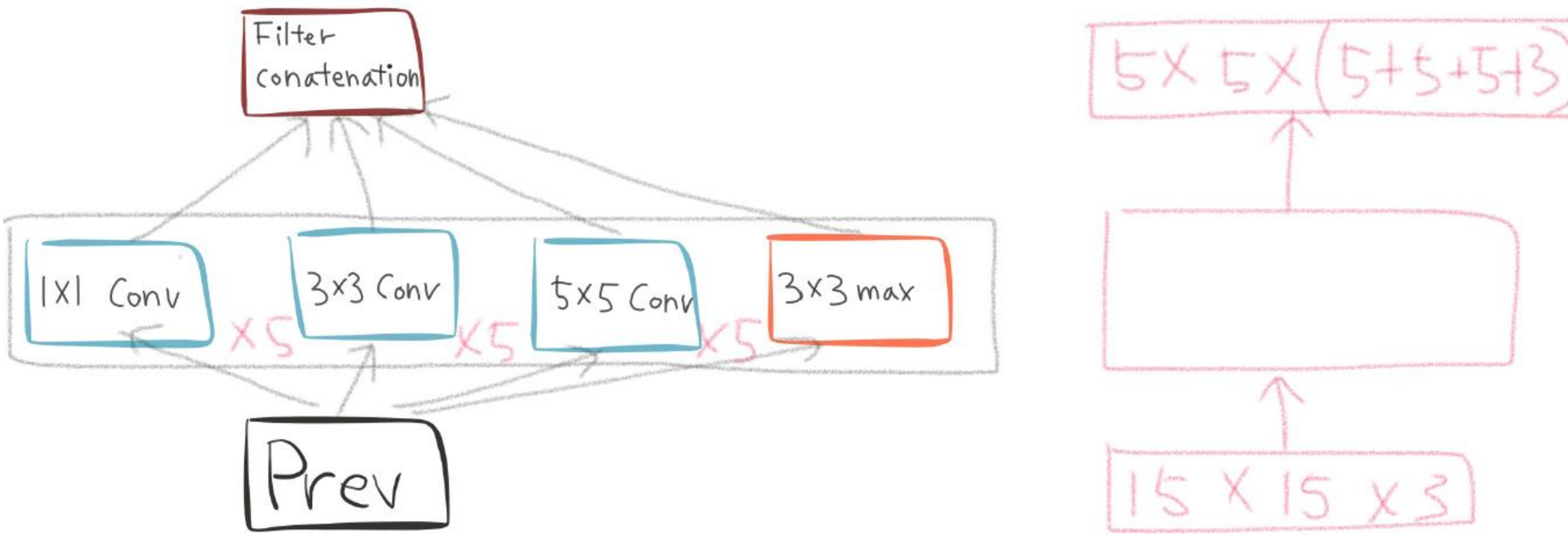


(b) Inception module with dimensionality reduction

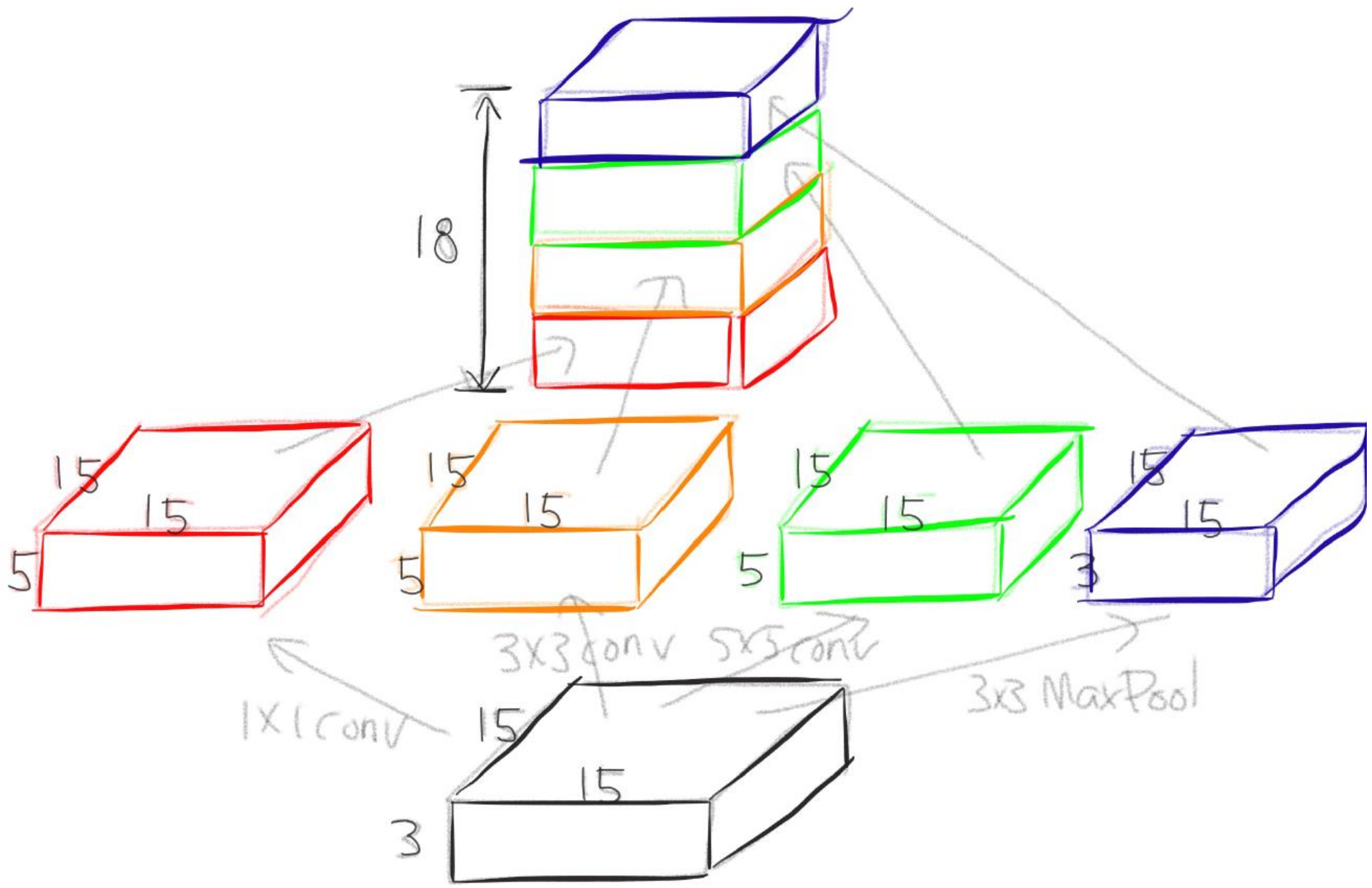
Naïve inception module



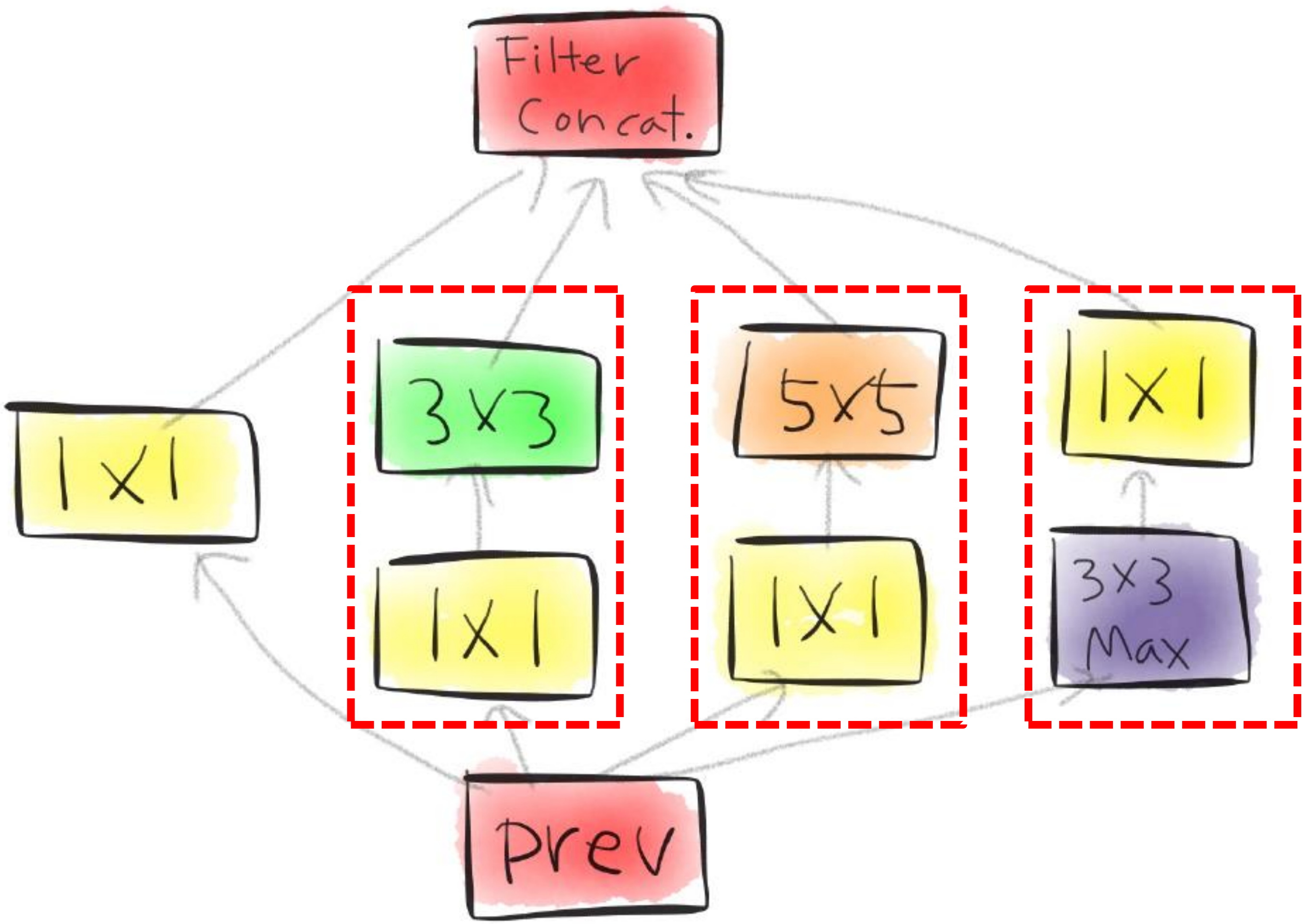
Naïve inception module



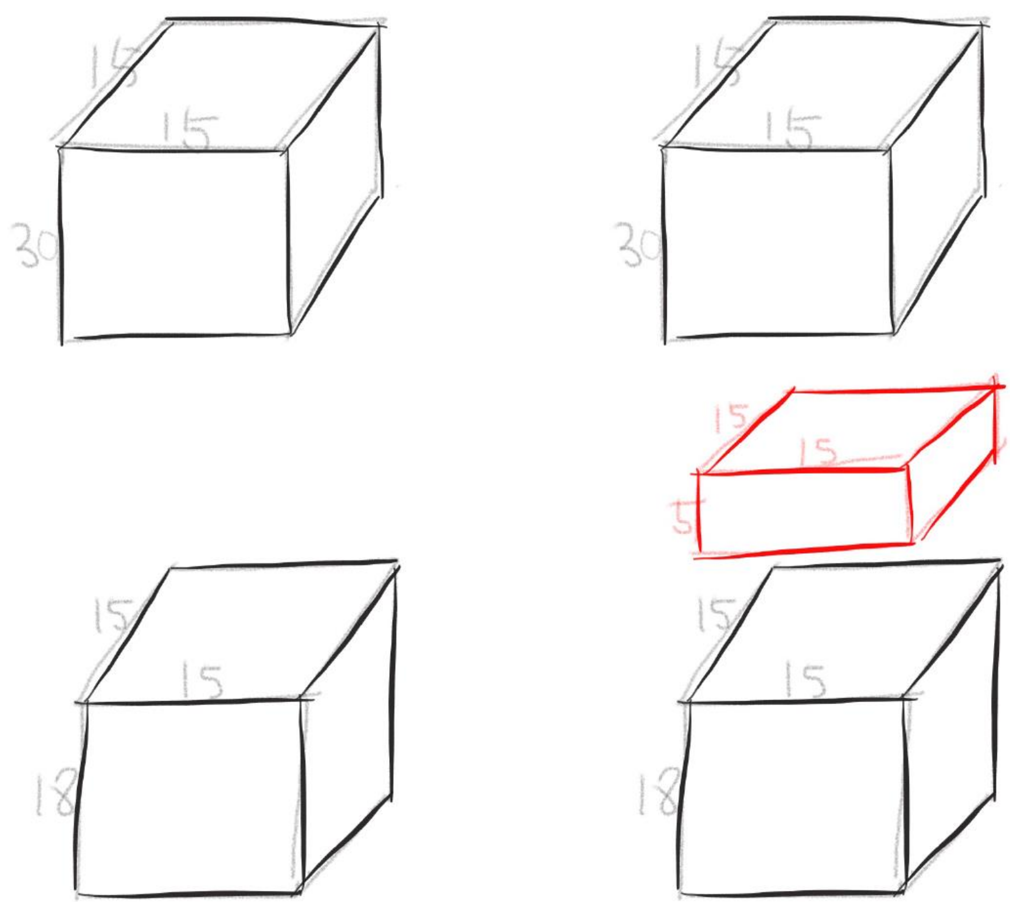
Naïve inception module



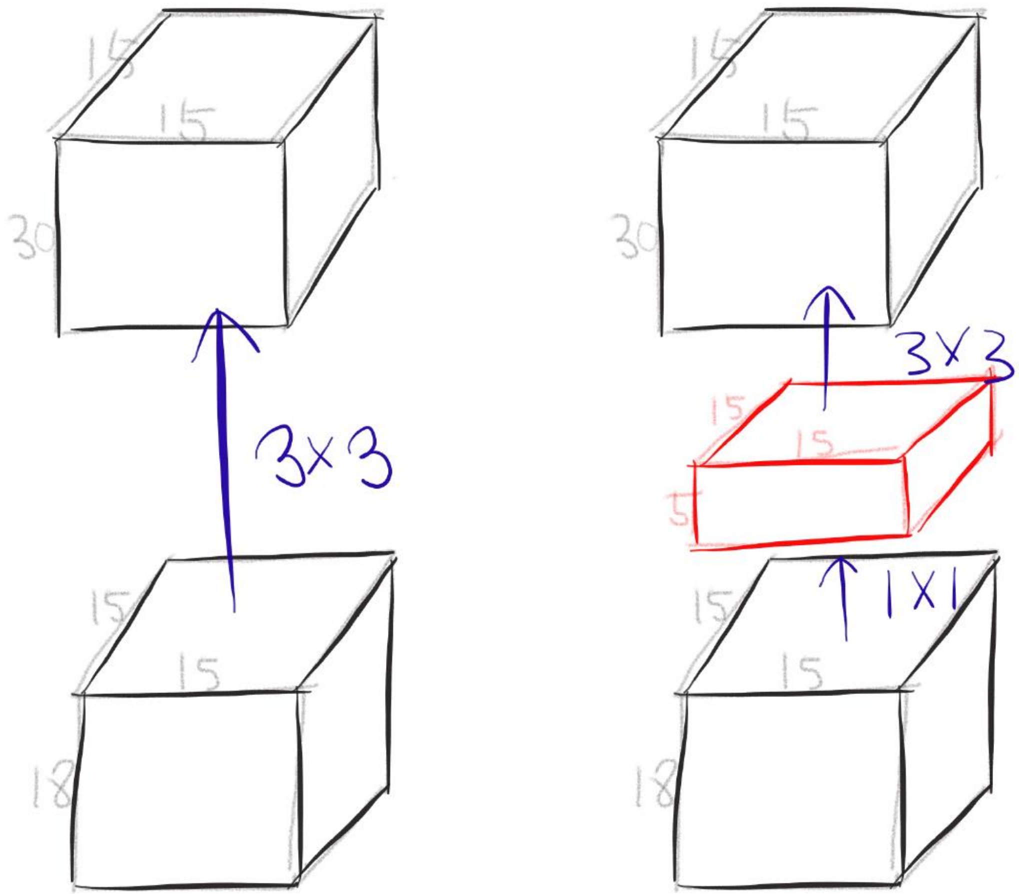
Actual inception module



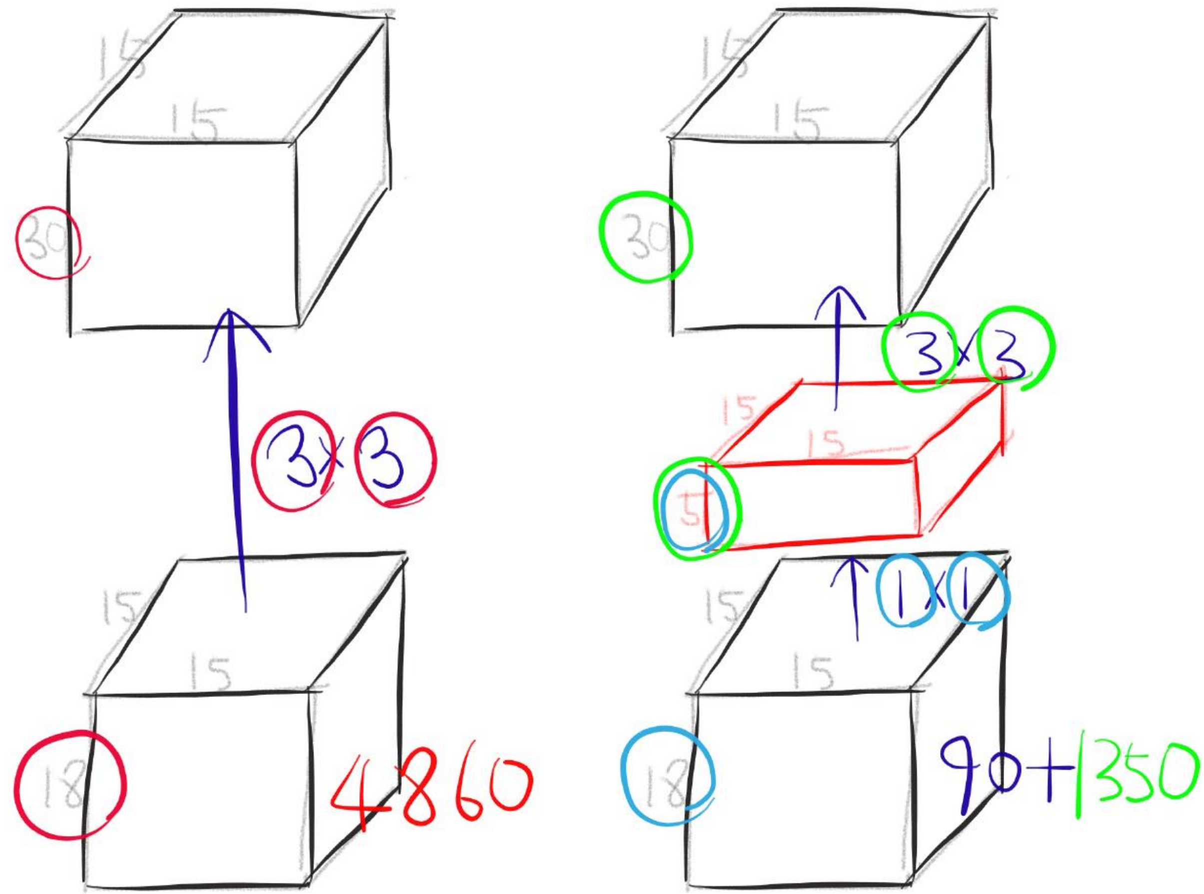
One by one convolution



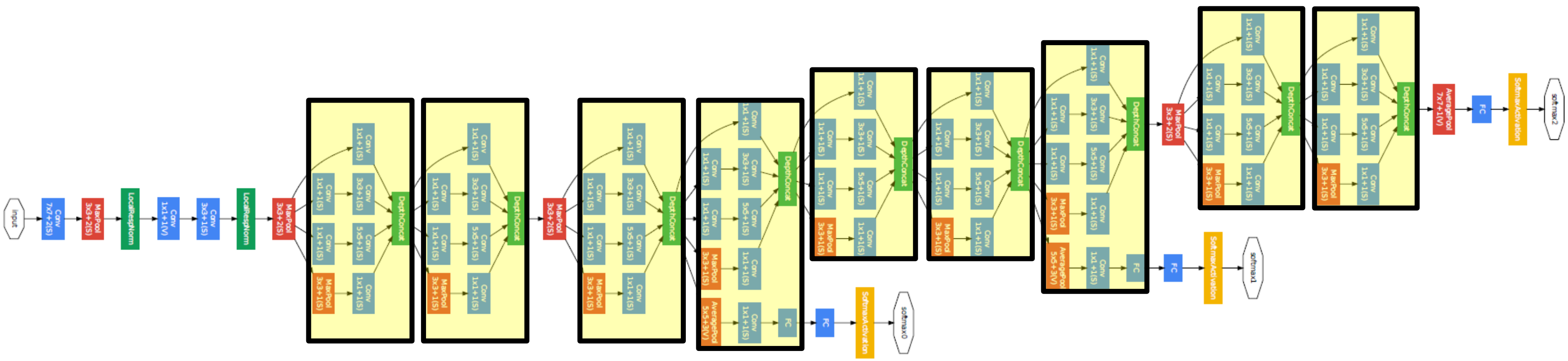
One by one convolution



One by one convolution



GoogLeNet



Network in Network!

Conclusion

Very clever idea of using **one by one convolution** for dimension reduction!

Other than that…

Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning

Inception v4

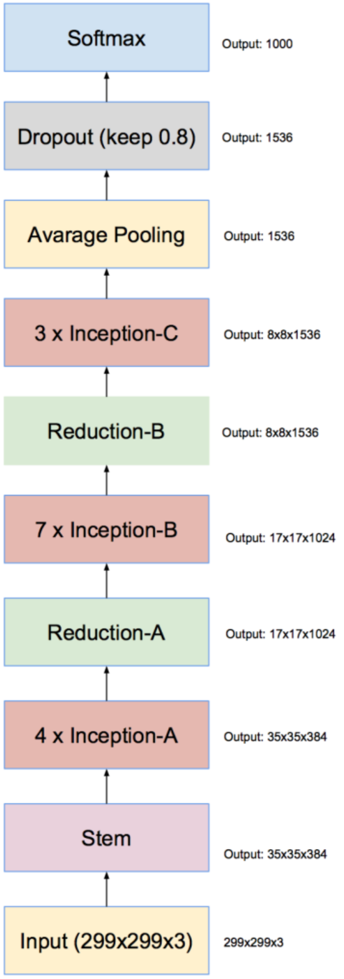
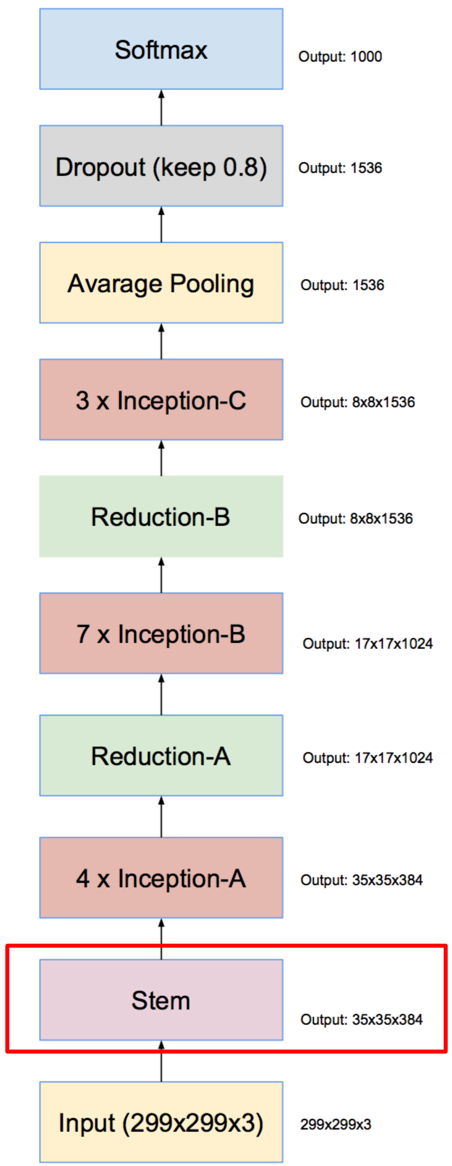


Figure 9. The overall schema of the Inception-v4 network. For the detailed modules, please refer to Figures 3, 4, 5, 6, 7 and 8 for the detailed structure of the various components.

Inception v4



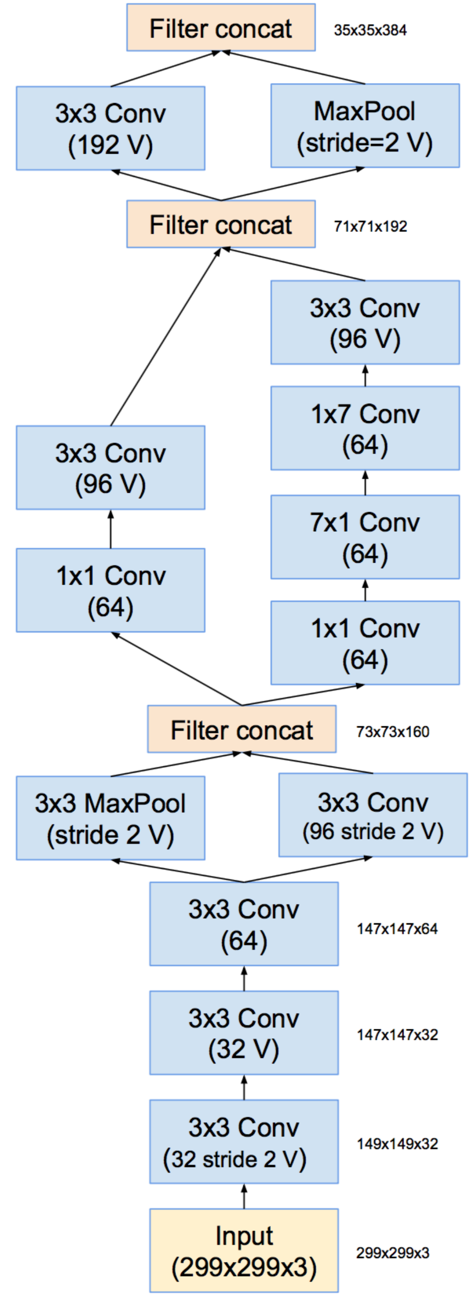
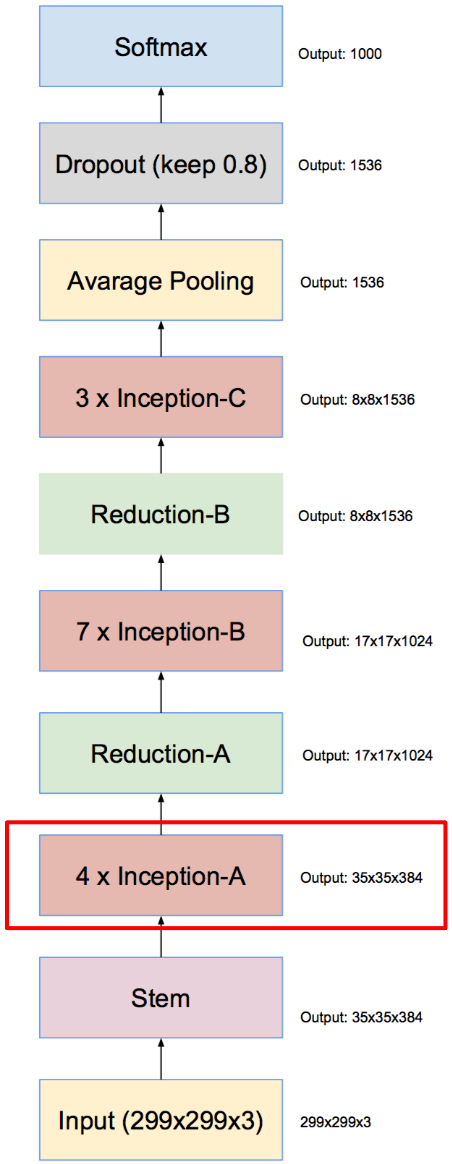


Figure 3. The schema for stem of the pure Inception-v4 and Inception-ResNet-v2 networks. This is the input part of those networks. Cf. Figures 9 and 15

Inception v4



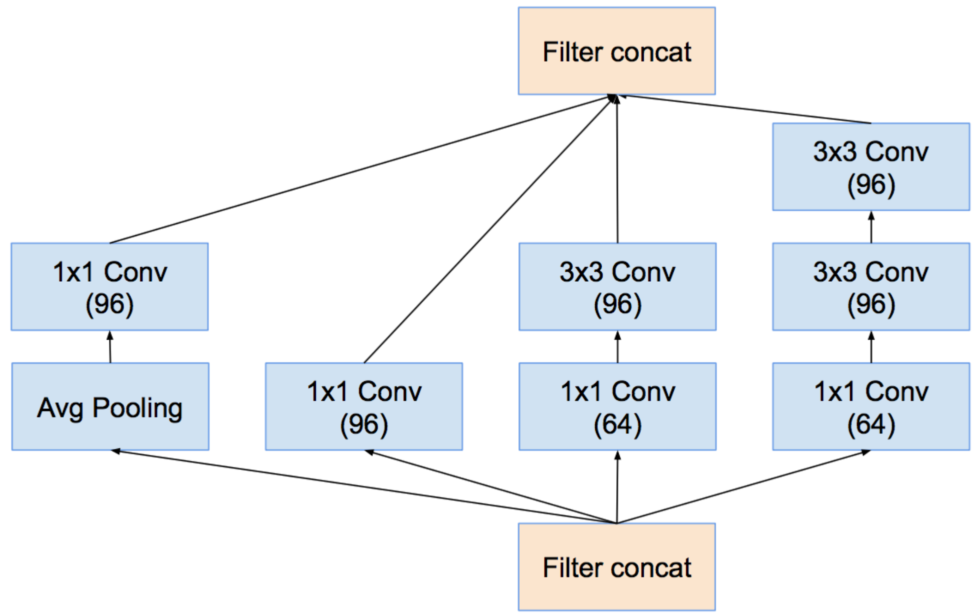
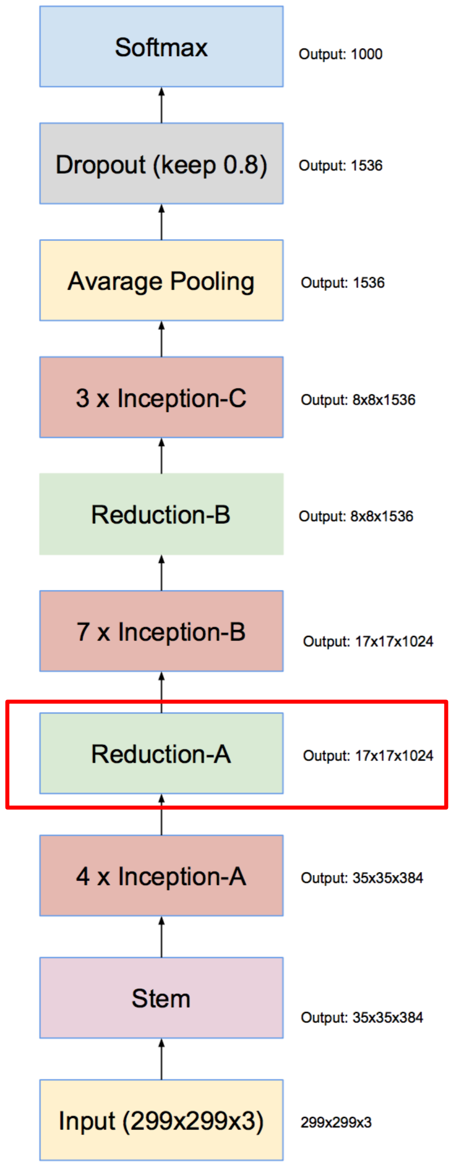


Figure 4. The schema for 35 × 35 grid modules of the pure Inception-v4 network. This is the Inception-A block of Figure 9.

Inception v4



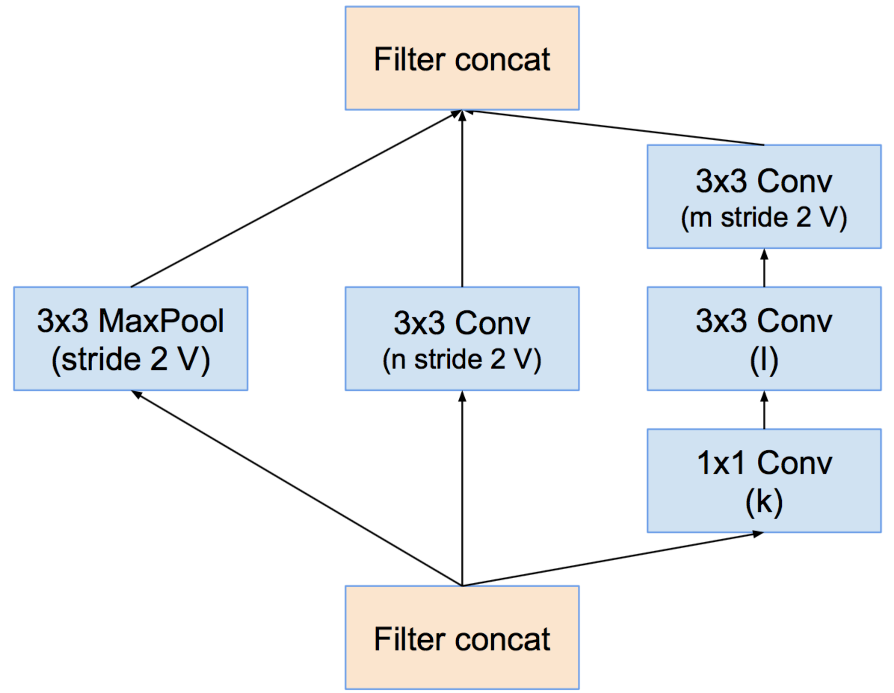
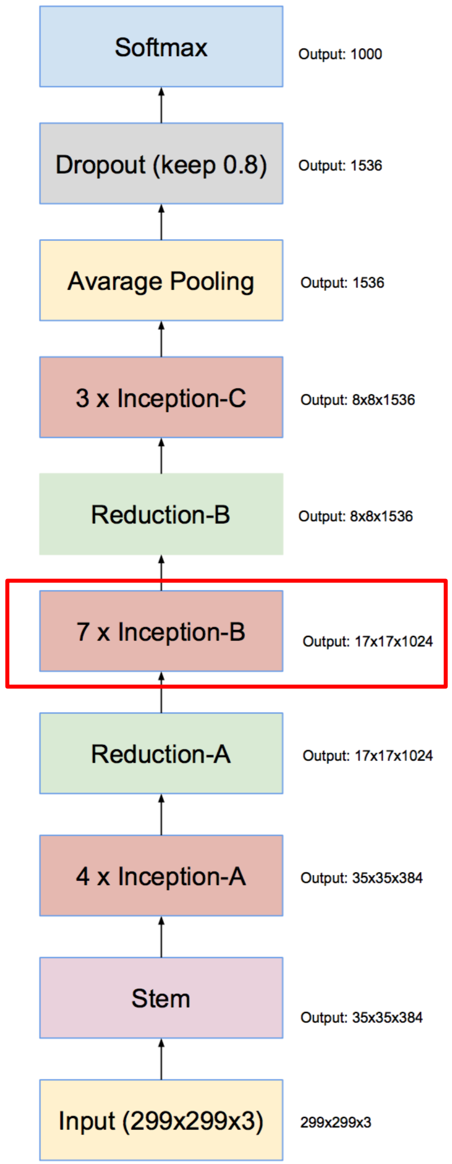


Figure 7. The schema for 35×35 to 17×17 reduction module. Different variants of this blocks (with various number of filters) are used in Figure 9, and 15 in each of the new Inception(-v4, -ResNet-v1, -ResNet-v2) variants presented in this paper. The k, l, m, n numbers represent filter bank sizes which can be looked up in Table 1.

Inception v4



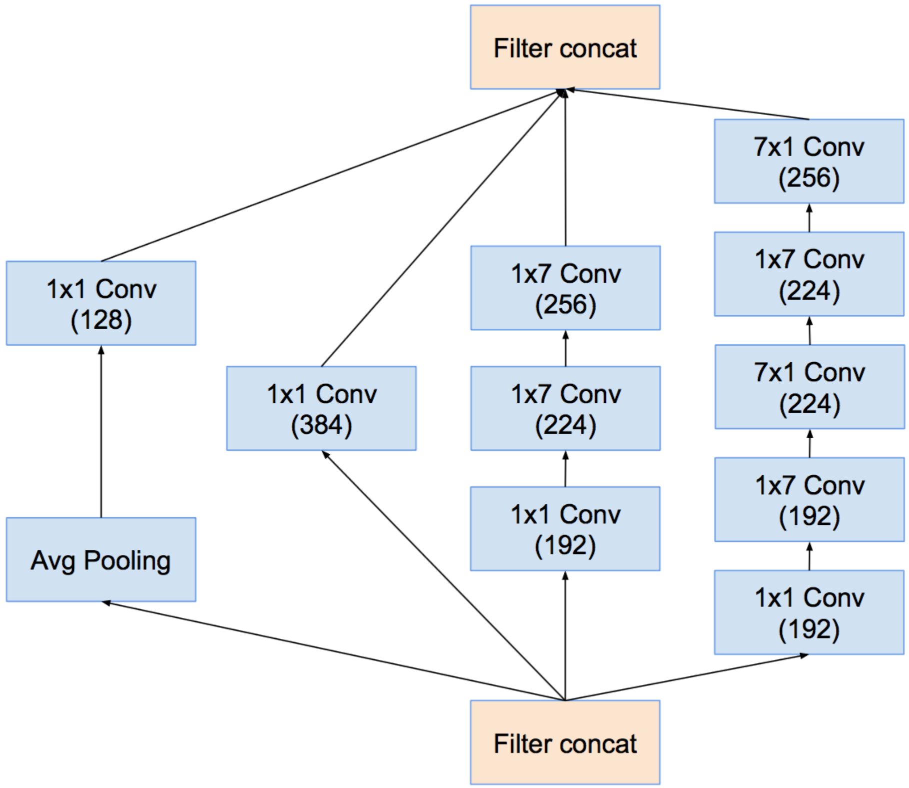
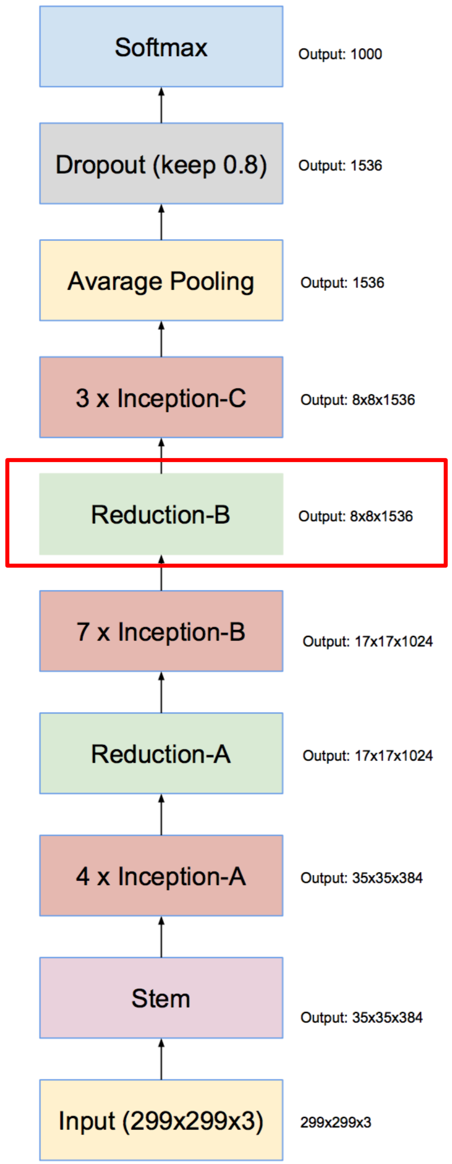


Figure 5. The schema for 17 × 17 grid modules of the pure Inception-v4 network. This is the Inception-B block of Figure 9.

Inception v4



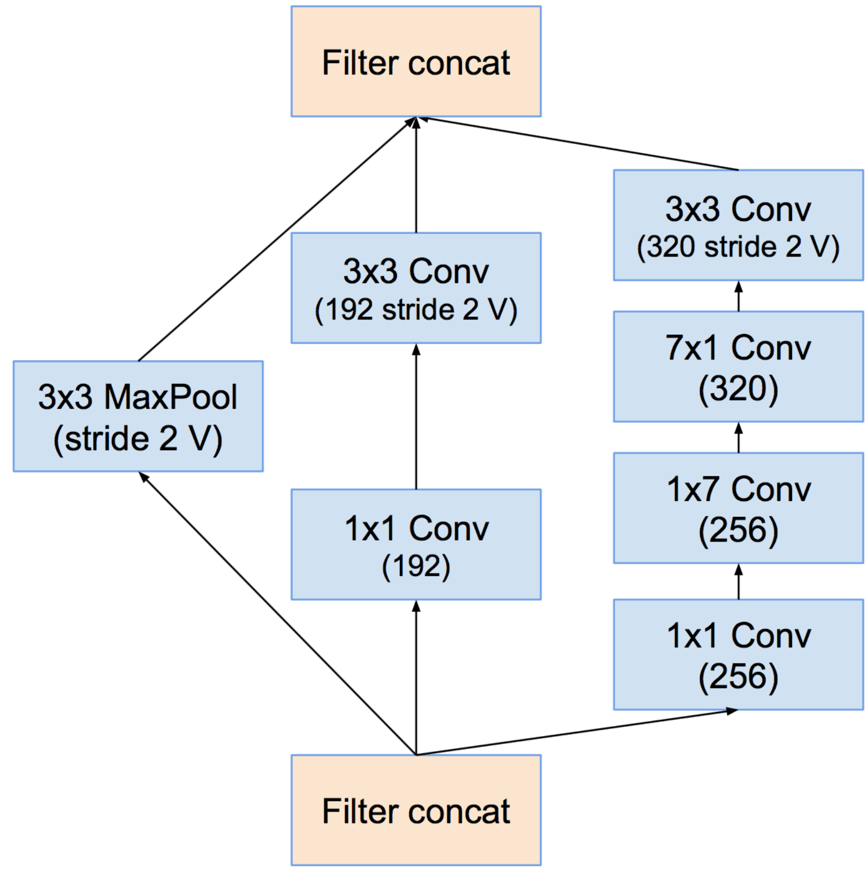
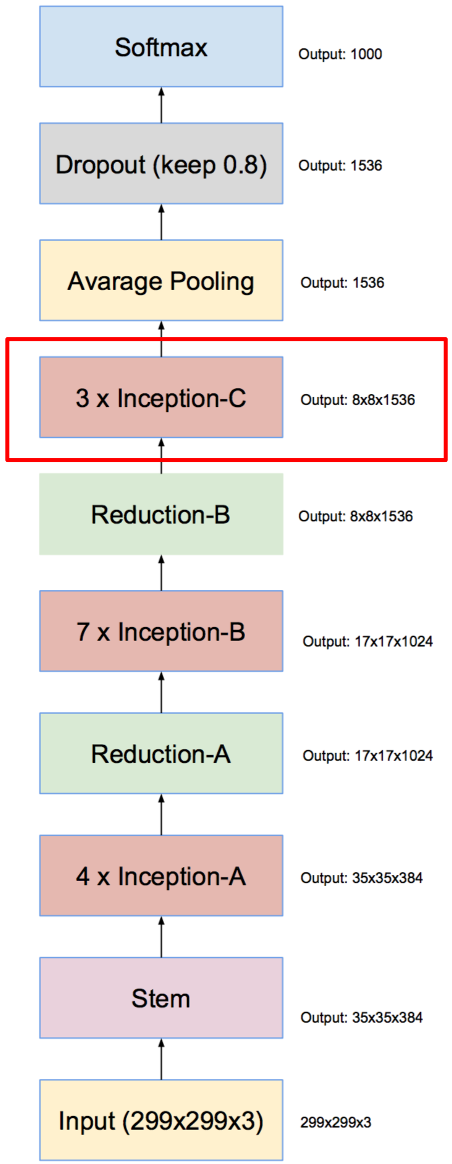


Figure 8. The schema for 17 × 17 to 8 × 8 grid-reduction module. This is the reduction module used by the pure Inception-v4 network in Figure 9.

Inception v4



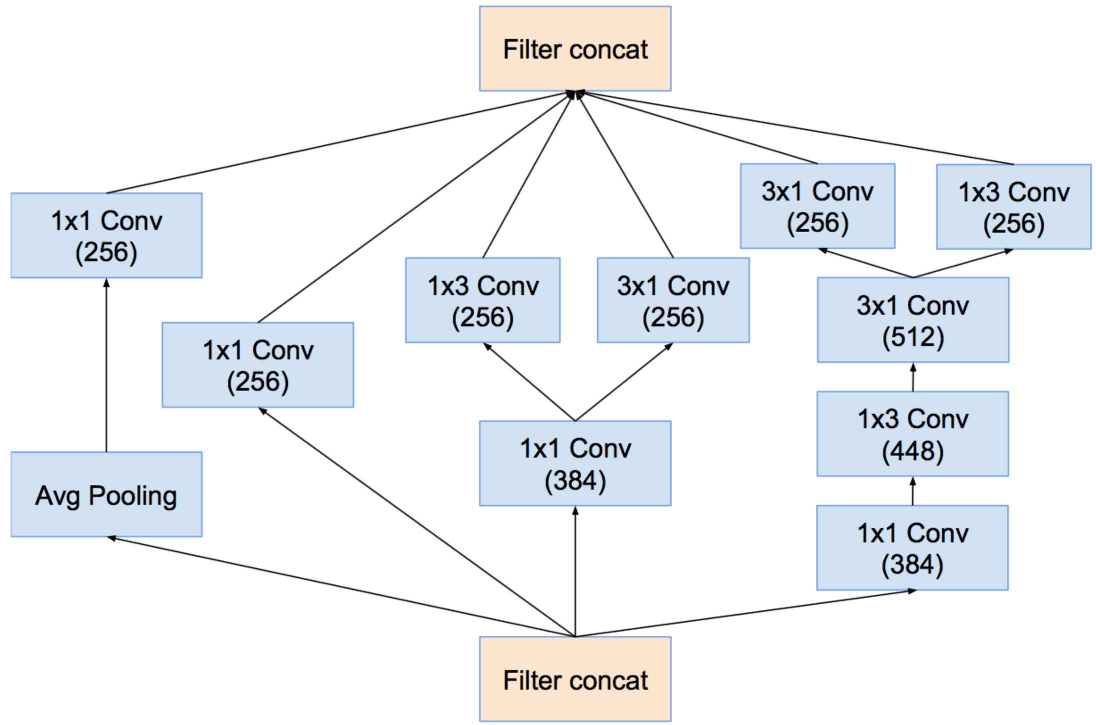


Figure 6. The schema for 8×8 grid modules of the pure Inception-v4 network. This is the Inception-C block of Figure 9.

Inception-ResNet-v1

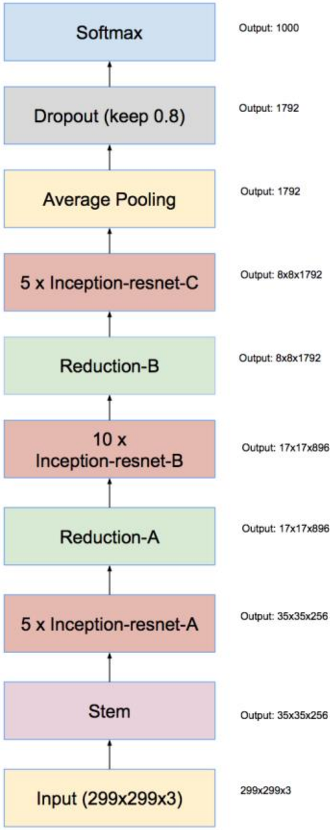
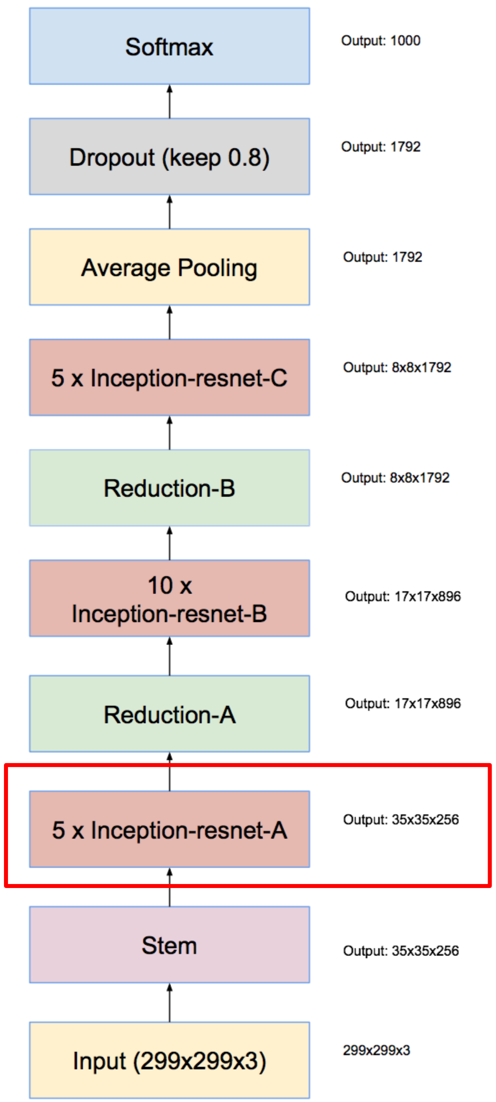


Figure 15. Schema for Inception-ResNet-v1 and Inception-ResNet-v2 networks. This schema applies to both networks but the underlying components differ. Inception-ResNet-v1 uses the blocks as described in Figures 14, 10, 7, 11, 12 and 13. Inception-ResNet-v2 uses the blocks as described in Figures 3, 16, 7, 17, 18 and 19. The output sizes in the diagram refer to the activation vector tensor shapes of Inception-ResNet-v1.

Inception-ResNet-v1



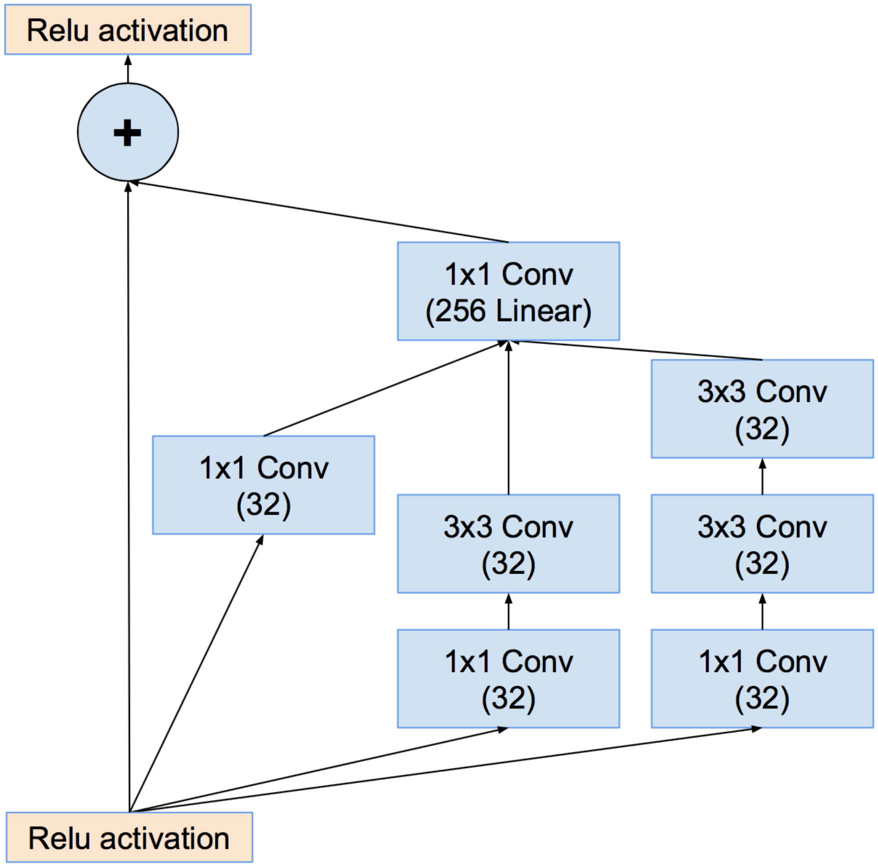
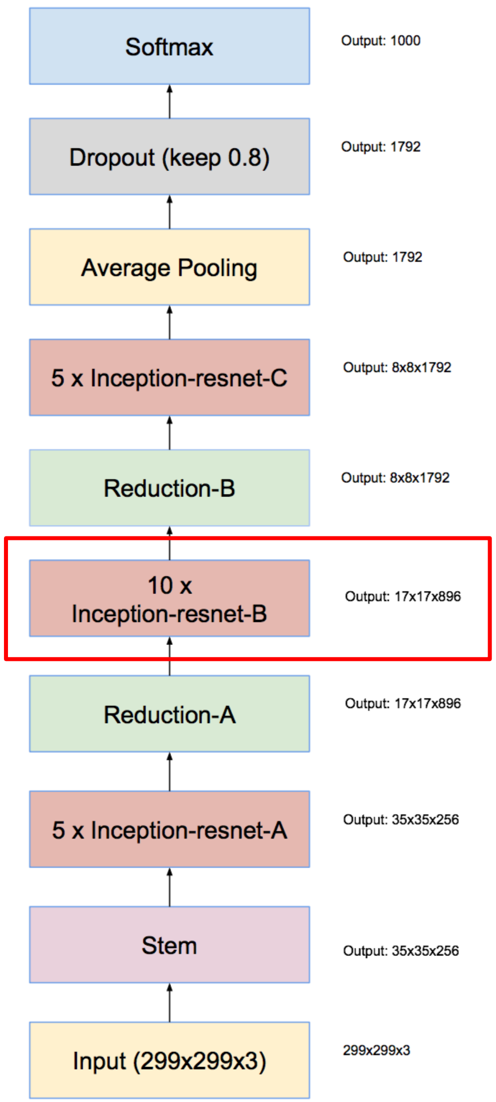


Figure 10. The schema for 35 × 35 grid (Inception-ResNet-A) module of Inception-ResNet-v1 network.

Inception-ResNet-v1



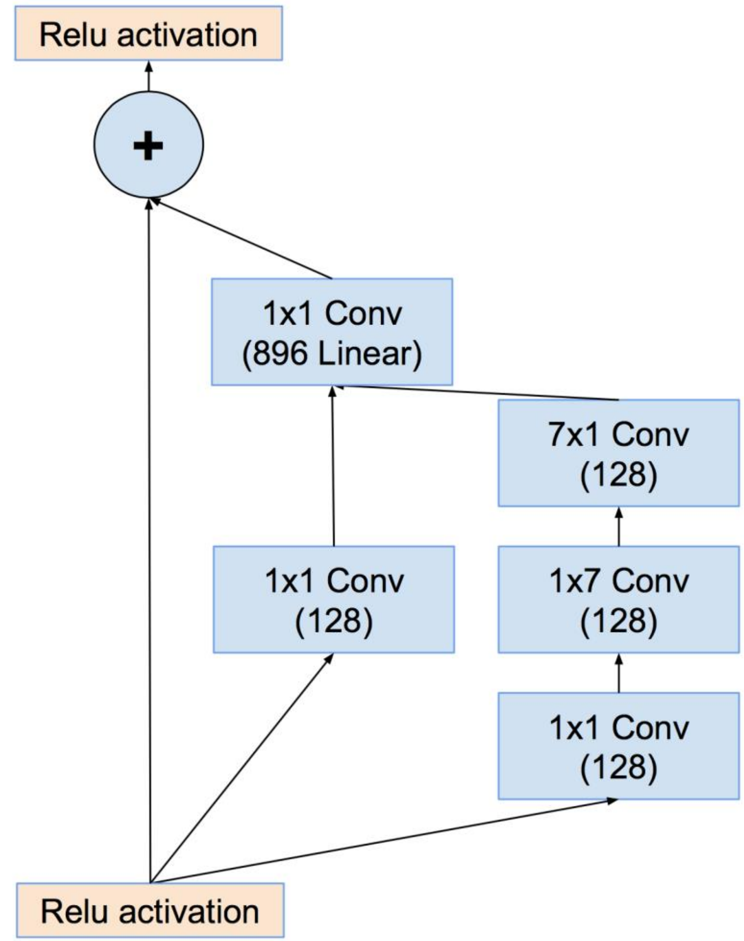
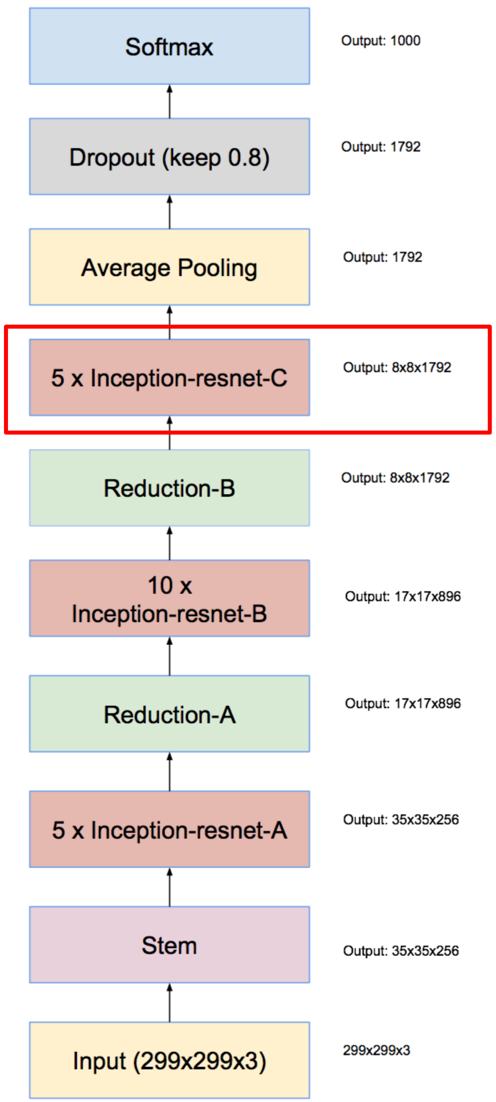


Figure 11. The schema for 17 × 17 grid (Inception-ResNet-B) module of Inception-ResNet-v1 network.

Inception-ResNet-v1



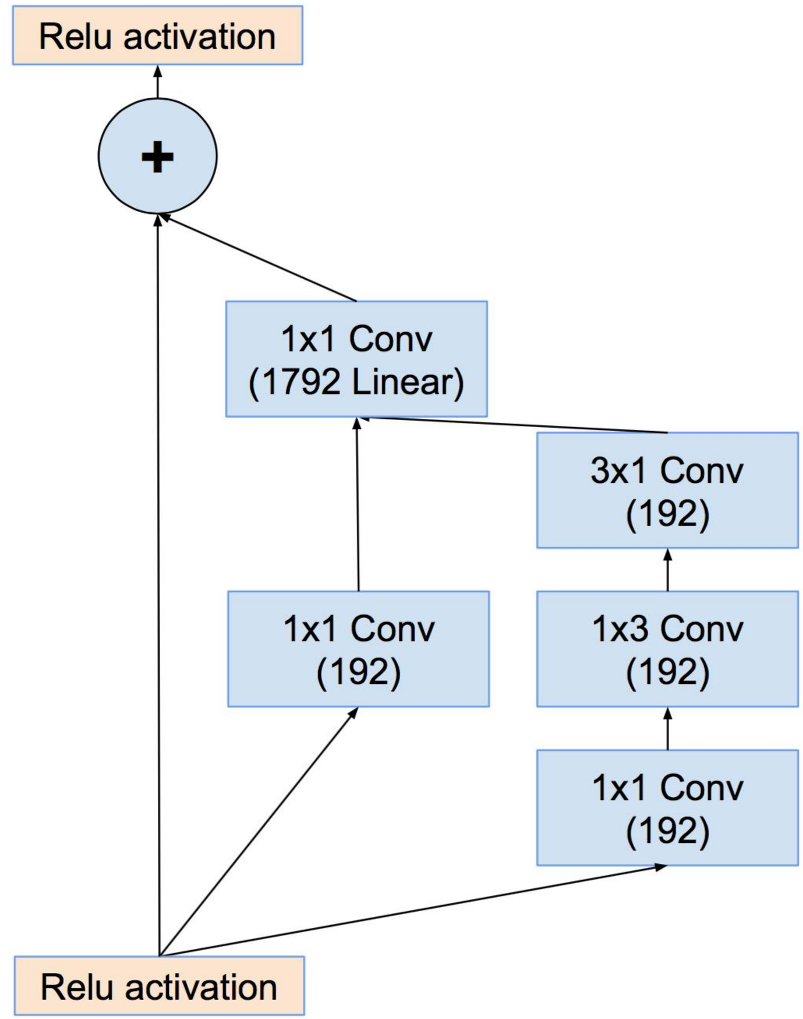


Figure 13. The schema for 8×8 grid (Inception-ResNet-C) module of Inception-ResNet-v1 network.

ResNet

Deep residual networks

152 layers network

1st place on ILSVRC 2015 classification task

1st place on ImageNet detection

1st place on ImageNet localization

1st place on COCO detection

1st place on COCO segmentation

Deeper Network?

Is deeper network always better?

What about vanishing/exploding gradients?

  Better initialization methods / batch normalization / ReLU

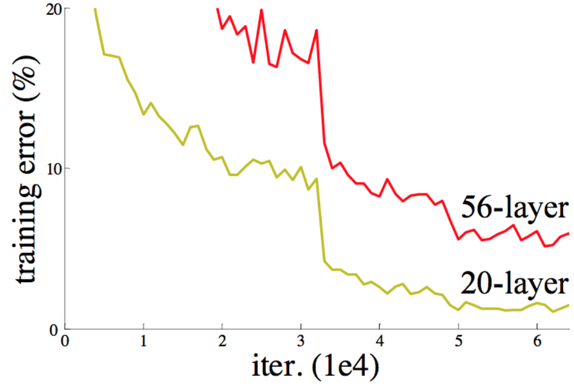
Any other problems?

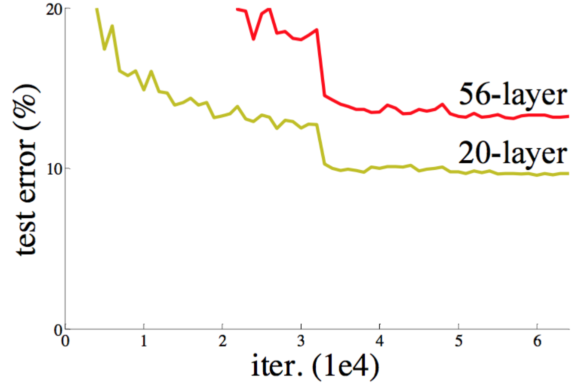
  Overfitting?

  Degradation problem: more depth but lower performance

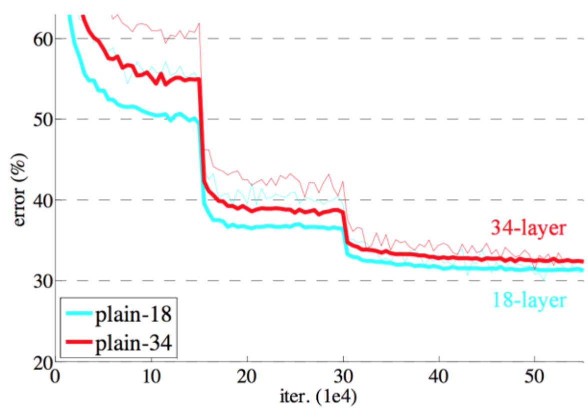
Degradation problem

CiFAR 100 Dataset

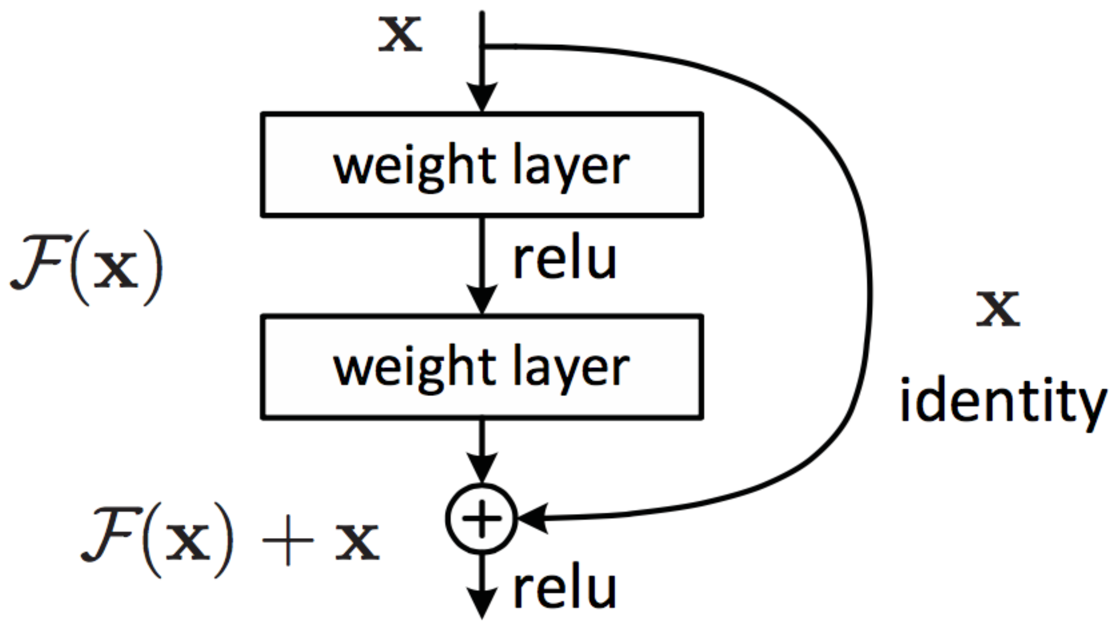




ImageNet

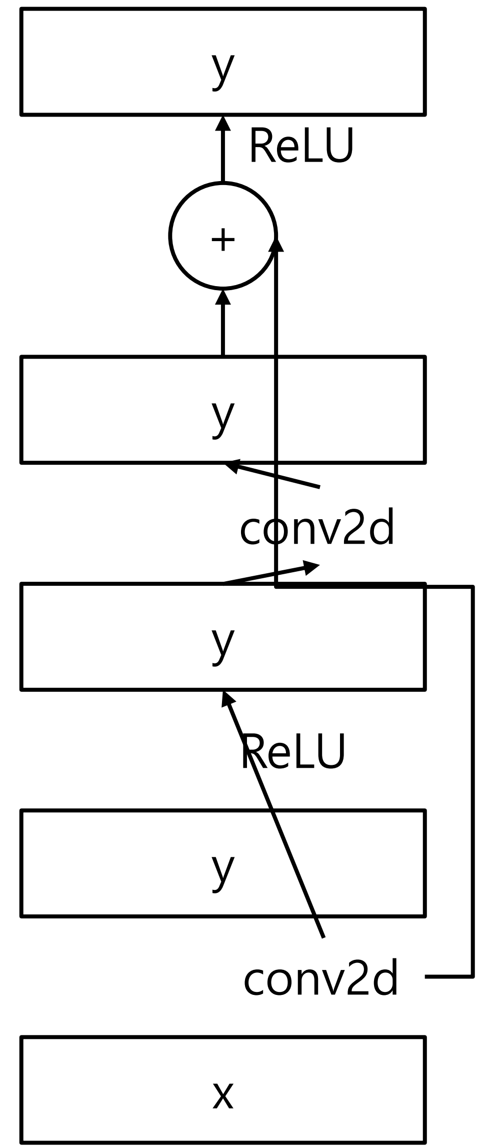


Residual learning building block



Residual learning building block

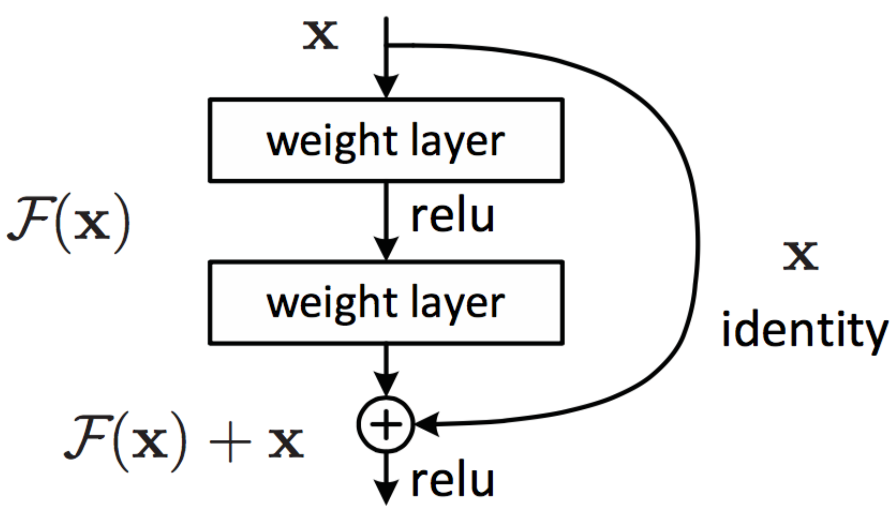
```python
def residual_block(x, n_in, n_out, subsample, phase_train, scope='res_block'):
  with tf.variable_scope(scope):
    if subsample:
      y = conv2d(x, n_in, n_out, 3, 2, 'SAME', False, scope='conv_1')
      shortcut = conv2d(x, n_in, n_out, 3, 2, 'SAME',
                        False, scope='shortcut')
    else:
      y = conv2d(x, n_in, n_out, 3, 1, 'SAME', False, scope='conv_1')
      shortcut = tf.identity(x, name='shortcut')
    y = batch_norm(y, n_out, phase_train, scope='bn_1')
    y = tf.nn.relu(y, name='relu_1')
    y = conv2d(y, n_out, n_out, 3, 1, 'SAME', True, scope='conv_2')
    y = batch_norm(y, n_out, phase_train, scope='bn_2')
    y = y + shortcut
    y = tf.nn.relu(y, name='relu_2')
  return y


def conv2d(x, n_in, n_out, k, s, p='SAME', bias=False, scope='conv'):
  with tf.variable_scope(scope):
    kernel = tf.Variable(
      tf.truncated_normal([k, k, n_in, n_out],
        stddev=math.sqrt(2/(k*k*n_in))),
      name='weight')
    tf.add_to_collection('weights', kernel)
    conv = tf.nn.conv2d(x, kernel, [1,s,s,1], padding=p)
    if bias:
      bias = tf.get_variable('bias', [n_out], initializer=tf.constant_initializer(0.0))
      tf.add_to_collection('biases', bias)
      conv = tf.nn.bias_add(conv, bias)
  return conv
```



Why residual?

We hypothesize that it is easier to optimize the residual mapping than to optimize the original, unreferenced mapping.

Shortcut connections are used.



Why residual?

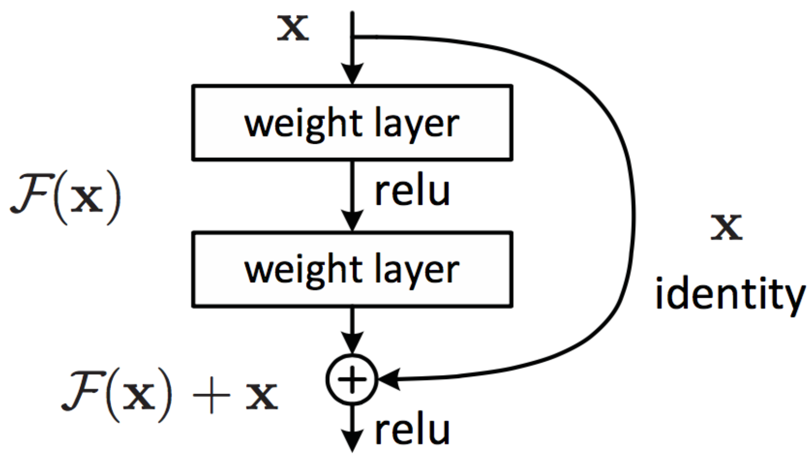
“The extremely deep residual nets are easy to optimize. “

“The deep residual nets can easily enjoy accuracy gains from greatly increased depth, producing results substantially better than previous networks. “

Residual mapping

Basic residual mapping (same dim.)

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}.$$



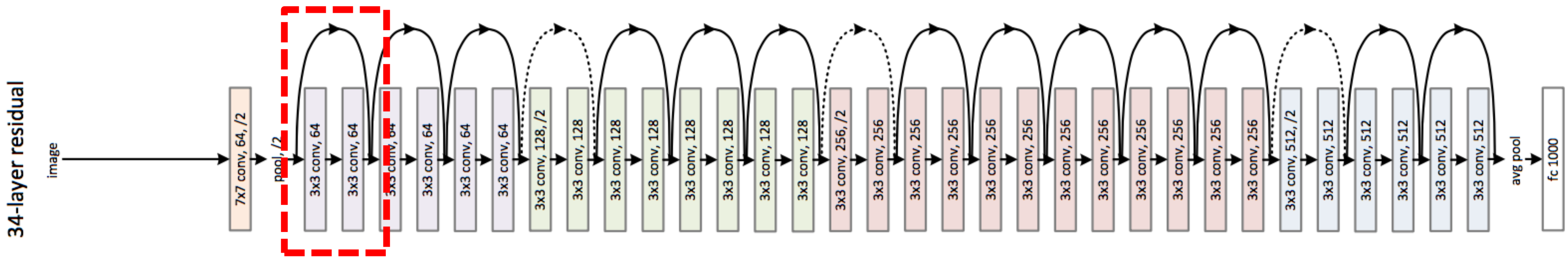
$$\mathcal{F} = W_2\sigma(W_1\mathbf{x})$$

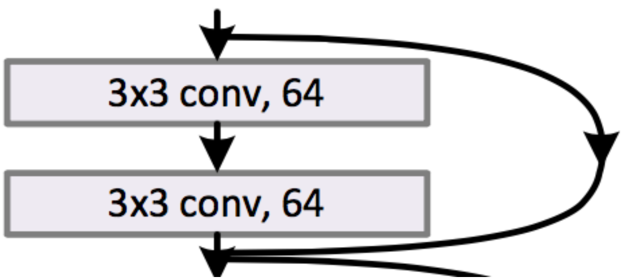
Basic residual mapping (different dim.)

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + W_s\mathbf{x}.$$

"But we will show by experiments that the identity mapping is sufficient for addressing the degradation problem and is economical, and thus W is only used when matching dimensions."

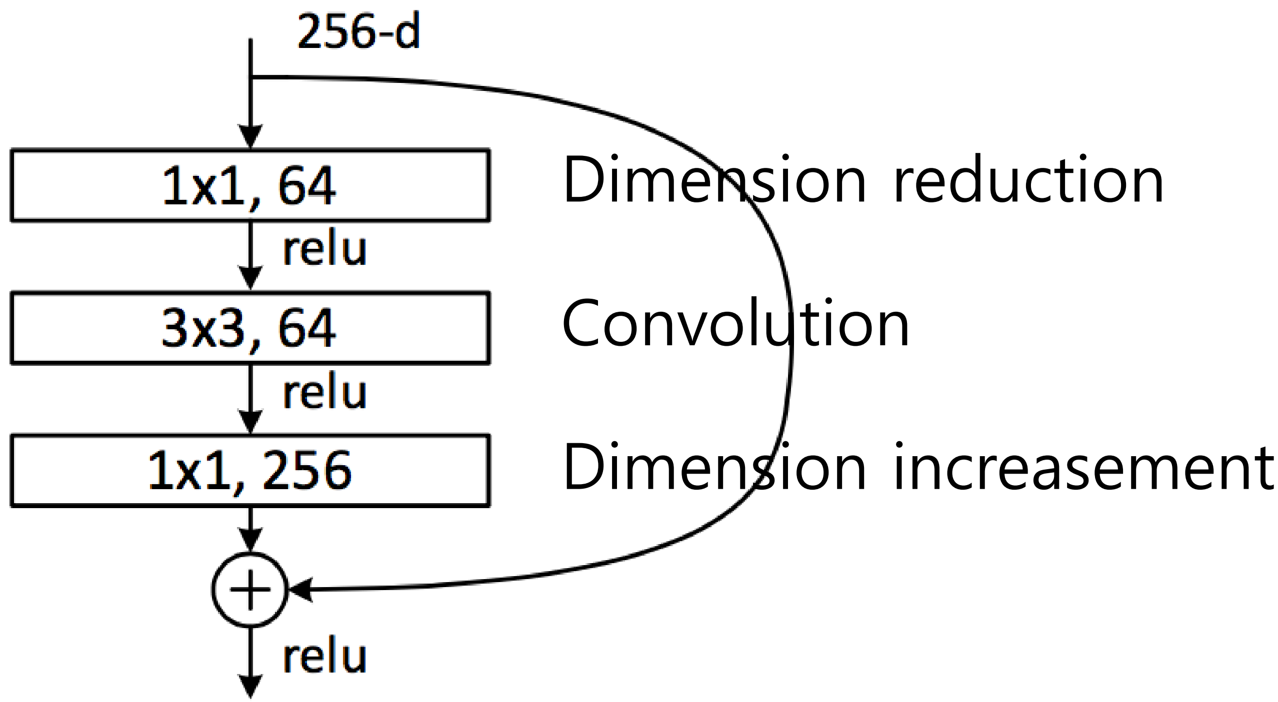
Deep residual network



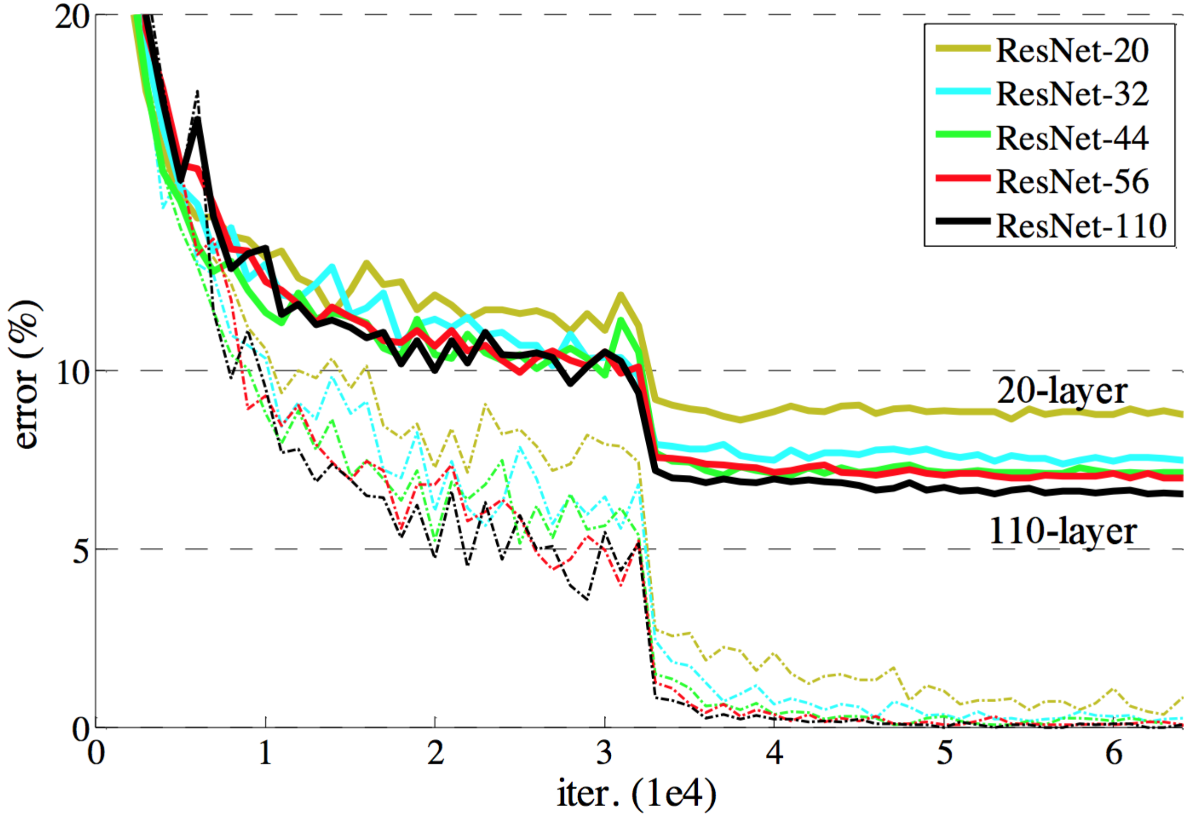



```
def residual_block(x, n_in, n_out, subsample, phase_train, scope='res_block'):
  with tf.variable_scope(scope):
    if subsample:
      y = conv2d(x, n_in, n_out, 3, 2, 'SAME', False, scope='conv_1')
      shortcut = conv2d(x, n_in, n_out, 3, 2, 'SAME',
                 False, scope='shortcut')
    else:
      y = conv2d(x, n_in, n_out, 3, 1, 'SAME', False, scope='conv_1')
      shortcut = tf.identity(x, name='shortcut')
    y = batch_norm(y, n_out, phase_train, scope='bn_1')
    y = tf.nn.relu(y, name='relu_1')
    y = conv2d(y, n_out, n_out, 3, 1, 'SAME', True, scope='conv_2')
    y = batch_norm(y, n_out, phase_train, scope='bn_2')
    y = y + shortcut
    y = tf.nn.relu(y, name='relu_2')
  return y
```

Deeper bottle architecture



Experimental results



Experimental results