Nizhny Novgorod State University

Institute of Information Technologies, Mathematics and Mechanics

Department of Computer software and supercomputer technologies

Educational course
«Introduction to deep learning
using the Intel® neon™ Framework»

Convolutional neural networks. Deep residual networks

Supported by Intel

Valentina Kustikova,
Phd, lecturer, department of Computer software
and supercomputer technologies

Content

- The “convolution” operation
- The structure of a convolutional layer
- Input and output data of convolutional neural networks
- Backpropagation algorithm for convolutional neural networks
- Computing the number of network parameters. Estimating the amount of memory required to store a convolutional network
- Example of a convolutional neural network for predicting a person's sex from a photo
- Principles of constructing convolutional neural networks
- The problem of model degradation. Deep residual networks
- Example of a residual neural network for predicting a person's sex from a photo

THE “CONVOLUTION” OPERATION. THE STRUCTURE OF A CONVOLUTIONAL LAYER

Convolutional neural networks

- ***Convolutional neural networks*** are the kind of neural networks that, at least on one of their layers, use the "convolution" operation as a transform

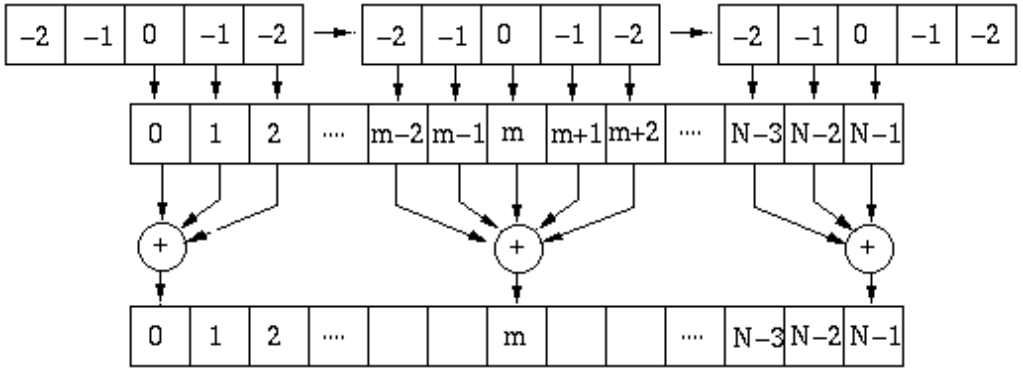
- ***Convolution*** is an operation applied to two real-valued functions

The “convolution” operation (1)

- Computer works with discrete data, and measurements are performed at a certain interval, therefore ***discrete convolution*** is considered:

$$s(t) = \langle x * w \rangle(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)\,,$$

where $x(\cdot)$ is ***an input***, $w(\cdot)$ is ***a convolution kernel***, output is ***a feature map***

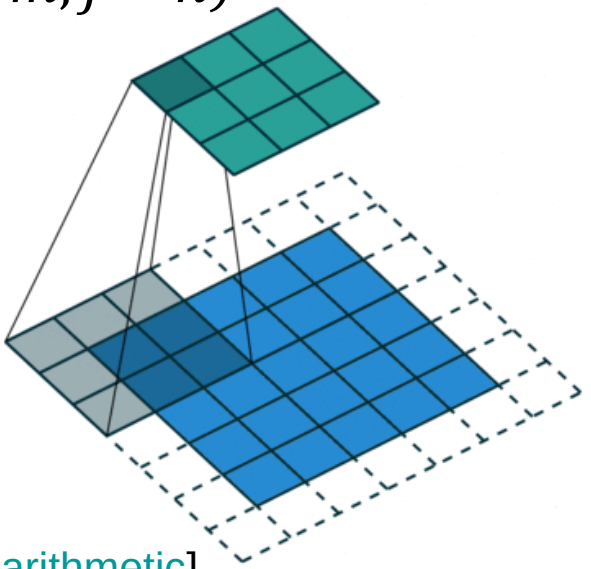


* Digital Convolution -- E186 Handout [http://fourier.eng.hmc.edu/e161/lectures/convolution/index.html]

The “convolution” operation (2)

- In machine learning problems, the ***input*** is a multidimensional array of data (tensor), and the ***kernel*** is a multidimensional array of parameters
- If we have a two-dimensional image I as input and a kernel K, then the convolution operation is described as follows:

$$s(i,j) = \langle I * K \rangle (i,j) = \sum_m \sum_n I(m,n) K(i-m, j-n)$$

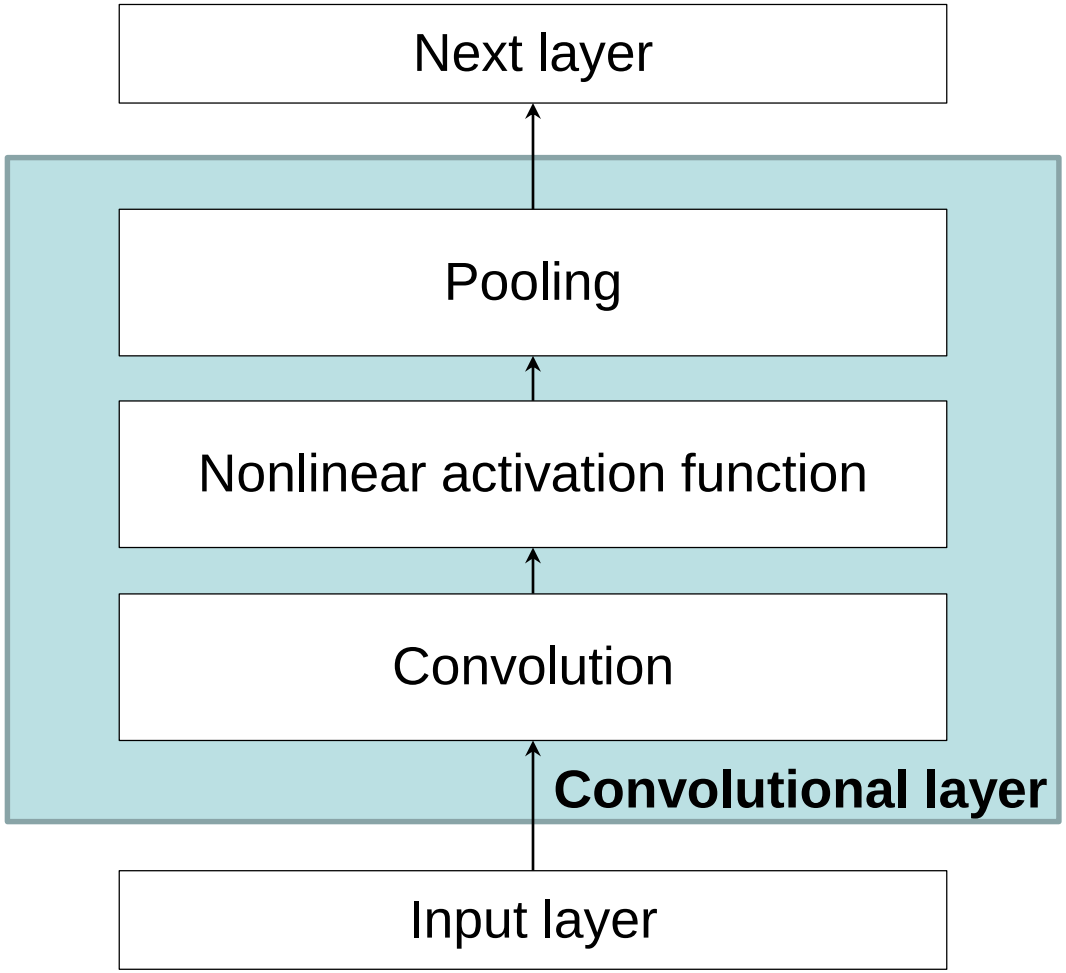


* A technical report on convolution arithmetic in the context of deep learning [https://github.com/vdumoulin/conv_arithmetic]

The structure of a convolutional layer (1)

- The structure of a convolutional layer involves the following transforms:
  - ***Creating a set of linear activations*** by performing one or more parallel convolutions
  - ***Detection*** involves the application of a nonlinear activation function to all linear activations
  - ***Pooling*** to modify the output for transmission to the next layer of the network

The structure of a convolutional layer (2)



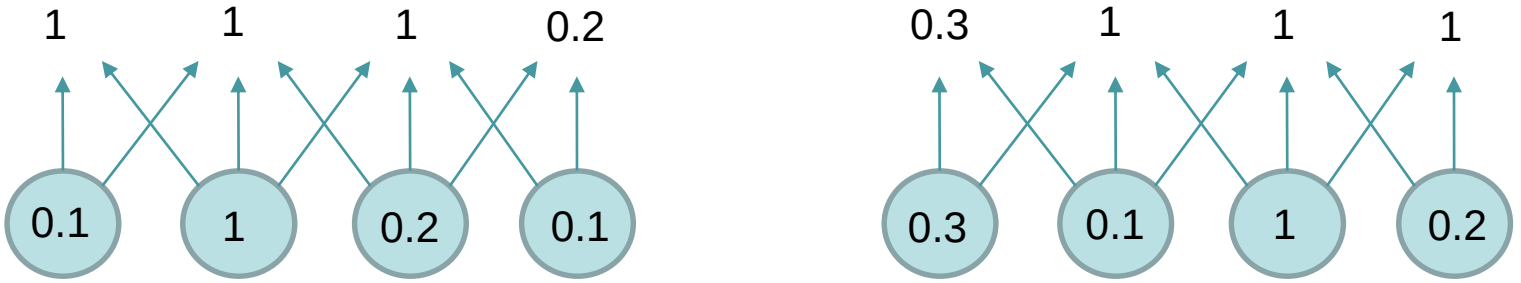
Pooling (1)

- The idea of pooling is to replace the network output by the summary statistics in the output neighborhood

- Examples:
  - max pooling,
  - average pooling,
  - L^2- norm in a rectangular neighborhood,
  - weighted average based on the distance from the center

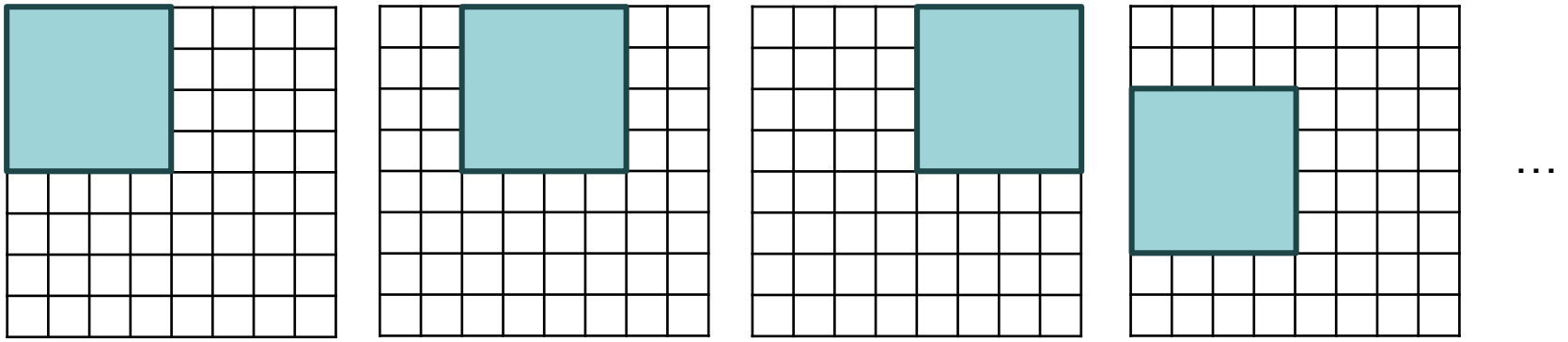
Pooling (2)

- ***Regardless of the pooling choice, it helps to make the representation to be invariant with respect to the offset of the inputs***
- If the object moves slightly in the image, then the output values of the pooling stage will remain practically unchanged
- For example, in the face detection problem, there is enough information that on the left and right side of the face there is one eye
- An example of applying max pooling (differences only at the boundaries)



Pooling (3)

- ***The pooling operation generalizes responses in a certain neighborhood, therefore at this stage it is possible to use a smaller number of neurons***
- It is implemented by combining neighborhoods with a step (stride), greater than one. Example of bypass (stride = 2):



- The computing efficiency of the network increases because the input size of the next layer is smaller (about a stride-fold) than the input size of the previous convolutional layer

INPUT AND OUTPUT DATA OF CONVOLUTIONAL NEURAL NETWORKS

One-dimensional input data

Single-channel data	Multi-channel data
Audio • A discrete signal obtained with some time step • Convolution is calculated along the time axis	***Skeleton animation data*** • At each time, the person's pose is described by angles formed at points corresponding to the joints of the skeleton • Each data channel that is fed to the input of the convolutional network is an angle around one axis of one joint

Two-dimensional input data

Single-channel data	Multi-channel data
Preprocessed audio • Signal after applying a discrete Fourier transform • A two-dimensional matrix in which the rows correspond to different frequencies, the columns correspond to different points in time	***Color image*** • Image in RGB (or BGR) format • The convolution kernel moves in the horizontal and vertical directions simultaneously, thus ensuring invariance with respect to the shift operation

Three-dimensional input data

Single-channel data	Multi-channel data
Spatial data • A typical example is CT scan	***Color video*** • Sequence of color images

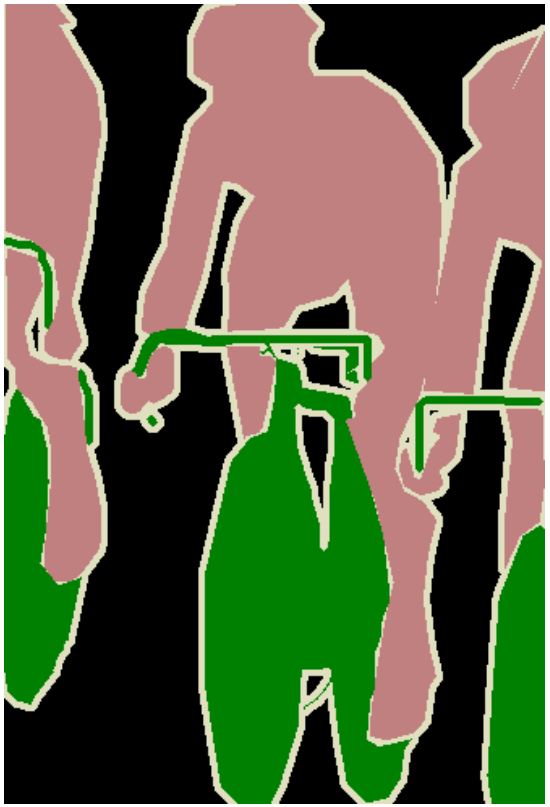
Output data (1)

- Convolutional neural networks allow to generate high-dimensional output data
- As a rule, an output is a tensor obtained at the output of a standard convolutional (or fully-connected) network layer

- For example, the model can generate a three-dimensional tensor with the elements $S_{i,j,k}$, corresponding to the probability of the pixel (i, j) belonging to the class k. The model allows to mark out each pixel and select objects in the image, i.e. to solve the semantic segmentation problem

Output data (2)

Origin image



Layout

Semantic segmentation

* The PASCAL Visual Object Classes Homepage [http://host.robots.ox.ac.uk/pascal/VOC]

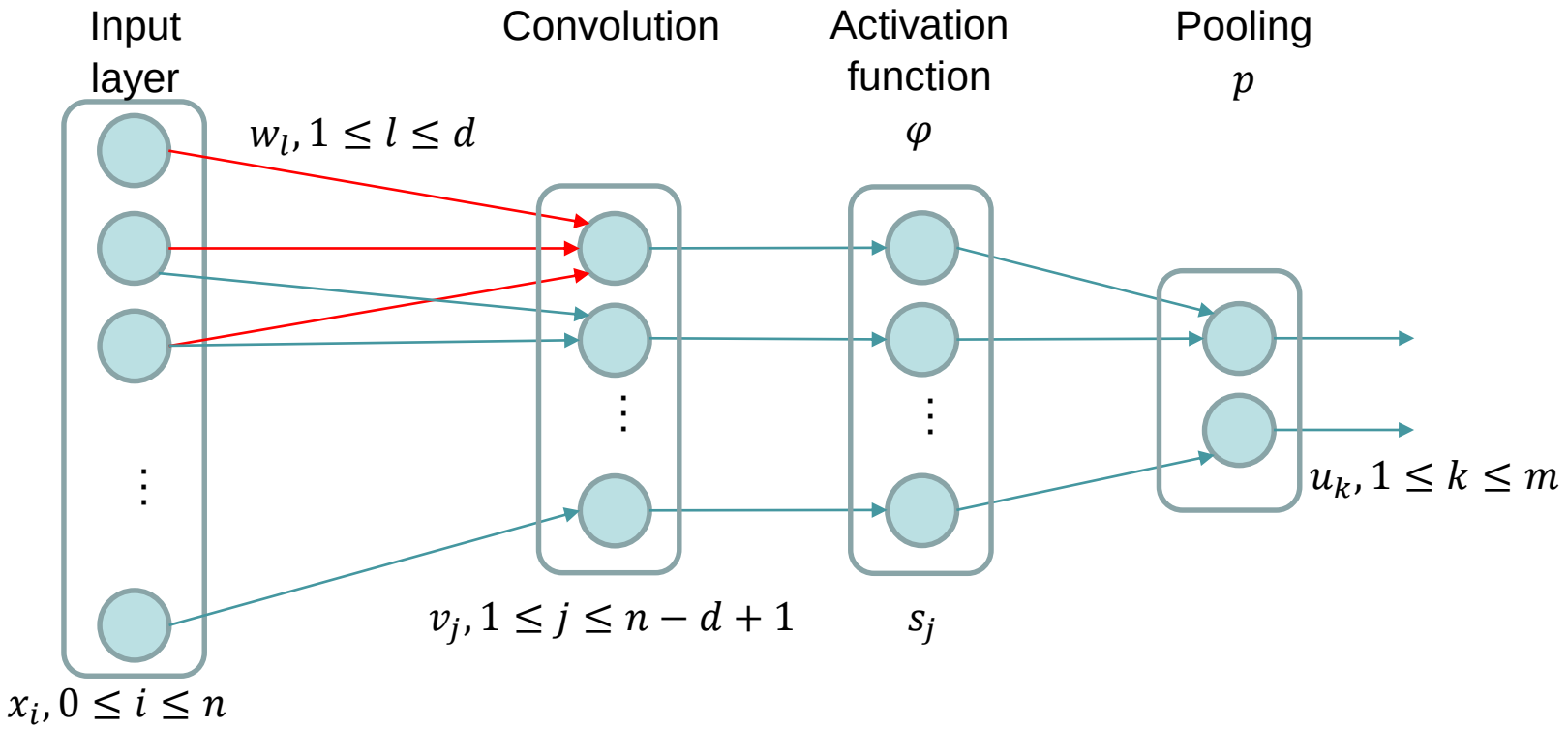
Output data (3)

- The output shape of the convolutional network ***depends on the particular application*** that is being solved

BACKPROPAGATION ALGORITHM FOR CONVOLUTIONAL NEURAL NETWORKS

Backpropagation algorithm for convolutional neural networks. Assumptions and notations

- Let us consider a convolutional network containing only one convolutional layer
- Assume that we have one-dimensional input data
- We denote as $E(w)$ the cost function



Feed forward. Computing derivatives (1)

- The result of applying convolution to the input signal is as follows:

$$v_j = \sum_{i=1}^{d} x_{j+i-1} w_i = w^T x_{j:j+d-1}$$

- Then the derivative of the convolution with respect to the weight coefficients of the kernel is calculated as follows:

$$\frac{\partial v_j}{\partial w_i} = x_{j+i-1}, \forall i = \overline{1, d}$$

Feed forward. Computing derivatives (2)

- The derivative of the cost function related to the kernel weights is as follows:

$$\frac{\partial E(w)}{\partial w_i} = \frac{\partial E}{\partial v}\frac{\partial v}{\partial w_i} = \sum_{j=1}^{n-d+1} \frac{\partial E}{\partial v_j}\frac{\partial v_j}{\partial w_i} = \sum_{j=1}^{n-d+1} \frac{\partial E}{\partial v_j} x_{j+i-1} =$$

$$= \langle \delta^{(conv)} * x \rangle(i) = \langle x * \delta^{(conv)} \rangle(i),$$

$$\delta^{(conv)} = \left(\frac{\partial E}{\partial v_j}\right)_{j=\overline{1,n-d+1}}, \qquad \frac{\partial E}{\partial w} = \langle x * \delta^{(conv)} \rangle$$

Feed forward. Computing derivatives (3)

- The derivatives of the activation function and the pooling function depend on the form of the functions
- Any differentiable real-valued function can be chosen as a polling function

$$
\begin{cases}
\dfrac{\sum_{i=1}^{q} x_i}{q}, \quad \dfrac{\partial p}{\partial x_i} = \dfrac{1}{q}, & average\ pooling \\
\max\limits_{i=\overline{1,q}} x_i, \quad \dfrac{\partial p}{\partial x_i} = \begin{cases} 1, x_i = \max\limits_{i=\overline{1,q}} x_i \\ 0, \text{иначе} \end{cases}, & max - pooling \\
\|x\|_p = \left(\sum_{i=1}^{q} |x_i|^p \right)^{\frac{1}{p}}, \dfrac{\partial p}{\partial x_i} = \left(\sum_{i=1}^{q} |x_i|^p \right)^{\frac{1}{p}-1} |x_i|^{p-1}, & L_p - pooling \\
p\colon \mathbb{R}^q \to \mathbb{R}, \quad \exists \dfrac{\partial p}{\partial x_i}
\end{cases}
$$

Feed forward. Computing derivatives (4)

- The derivative of the cost function is calculated as follows:

$$\frac{\partial E(w)}{\partial w_i} = \sum_{j=1}^{n-d+1} \frac{\partial E}{\partial u_k} \frac{\partial p}{\partial s_j} \frac{\partial \varphi}{\partial v_j} x_{j+i-1} = \langle x * \delta^{(conv)} \rangle (x_i),$$

$$\delta_{j,k}^{(conv)} = \frac{\partial E}{\partial u_k} \frac{\partial p}{\partial s_j} \frac{\partial \varphi}{\partial v_j} = \delta_k^{(pool)} \frac{\partial p}{\partial s_j} \frac{\partial \varphi}{\partial v_j}, \qquad \delta_k^{(pool)} = \frac{\partial E}{\partial u_k}$$

Backward. Weight update

- The type of the pooling function determines in which direction the error propagates
- In the case of max pooling, it is obvious that the gradient of the cost function on the last layer goes towards the neuron s_j, at the output of which the maximum value
- In the general case, the gradient propagates in accordance with the value $\delta_k^{(pool)}, 1 \le k \le m$
- This value is transmitted in the reverse direction and the gradient values on the convolutional layer are determined in accordance with the formula for calculating $\delta_{j,k}^{(conv)}, 1 \le j \le n - d + 1$

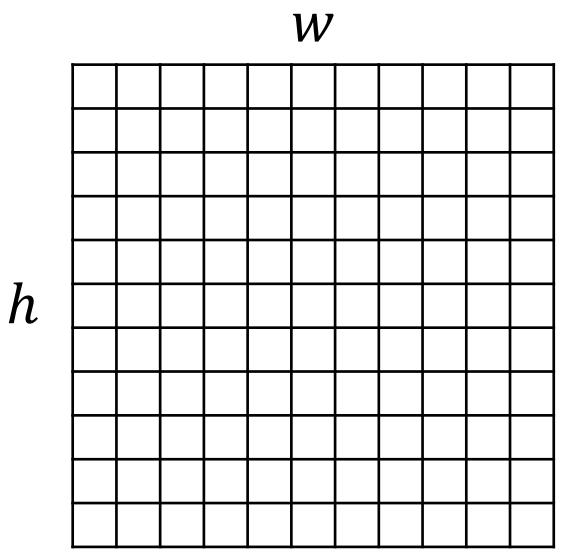
COMPUTING THE NUMBER OF NETWORK PARAMETERS

Why determine the number of network parameters?

- It allows to estimate the parameter space dimension in which the task of minimizing the cost function will be solved
- It allows to estimate the amount of memory required for training / testing a neural network

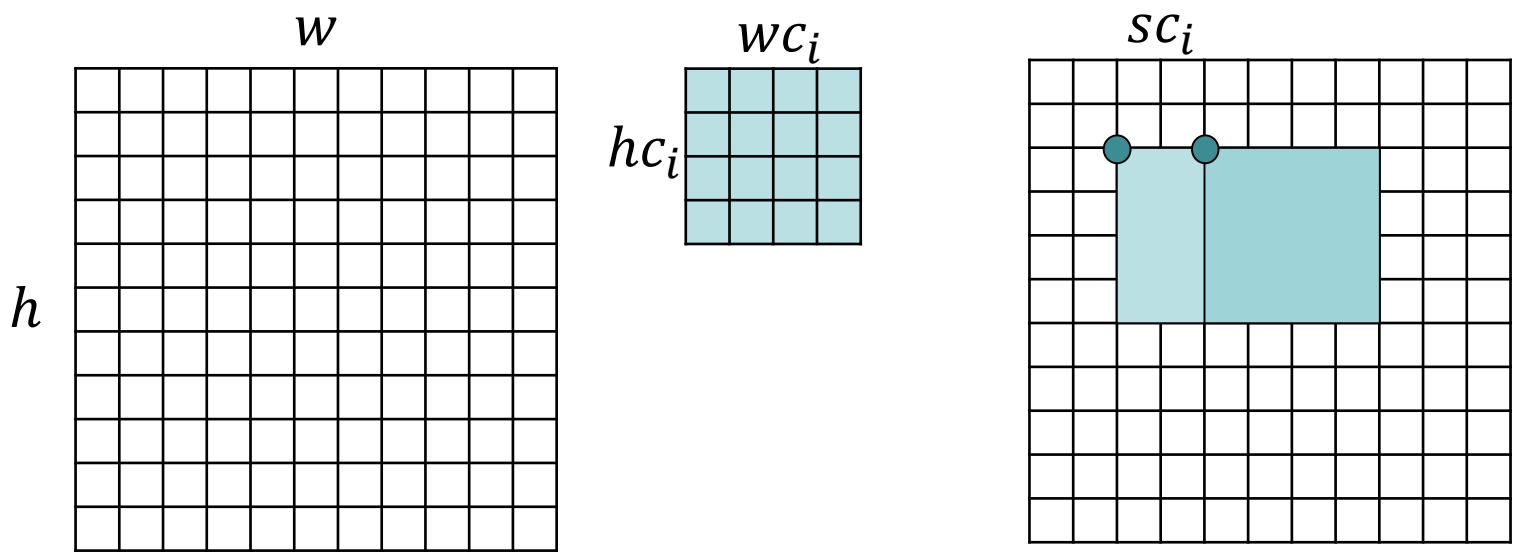
Computing the number of convolutional network parameters (1.1)

- Assumed that the network input is a single-channel image of the resolution $w \times h$



Computing the number of convolutional network parameters (1.2)

- The network contains N convolutional layers (triples consisting of convolution, activation function and pooling)
- On each layer the feature map is convolved with filter whose kernel size is $wc_i \times hc_i$, where $1 \leq i \leq N$ is a layer number
- The image is bypassed by the filter with the stride sc_i



Computing the number of convolutional network parameters (1.3)

- The number of network parameters is described as follows:

$$\sum_{i=1}^{N} wc_i \cdot hc_i$$

Computing the number of convolutional network parameters (2)

- In general case, a convolution can be calculated with several kernels of the same size on each layer
- Assuming that the layer with number i contains k_i convolutions, the total number of parameters is as follows:

$$\sum_{i=1}^{N} k_i \cdot wc_i \cdot hc_i$$

The amount of memory required to store a convolutional neural network (1)

- The input network layer contains $w \times h$ pixels

- Appling k_1 convolutions to the input layer leads to the feature map of the shape which is computed as follows

$$k_1 \times \left(\left[\frac{w - wc_1}{sc_1}\right] + 1\right) \times \left(\left[\frac{h - hc_1}{sc_1}\right] + 1\right)$$

- Appling activation function to each element of this feature map allows to obtain the feature map with the same shape

$$k_1 \times \left(\left[\frac{w - wc_1}{sc_1}\right] + 1\right) \times \left(\left[\frac{h - hc_1}{sc_1}\right] + 1\right)$$

The amount of memory required to store a convolutional neural network (2)

- Let us introduce the notation:

$$w_1 = \left[\frac{w - wc_1}{sc_1}\right] + 1, \qquad h_1 = \left[\frac{h - hc_1}{sc_1}\right] + 1$$

- Assumed the pooling operation is applied to each channel of the feature map (output of activation function)
- Assumed the shape of neighborhood is $wp_1 \times hp_1$, and the stride is sp_1

- The shape of the feature map after pooling is calculated as follows:

$$k_1 \times \left(\left[\frac{w_1 - wp_1}{sp_1}\right] + 1\right) \times \left(\left[\frac{h_1 - hp_1}{sp_1}\right] + 1\right)$$

The amount of memory required to store a convolutional neural network (3)

- The amount of memory required for storing the convolutional layer is described as follows:

$$\left(2k_1w_1h_1 + k_1\left(\left[\frac{w_1 - wp_1}{sp_1}\right] + 1\right)\left(\left[\frac{h_1 - hp_1}{sp_1}\right] + 1\right)\right) \cdot sizeof(type)$$

- If we have N convolutional layers, arranged in the same way, then to store the network the following amount of memory is required:

$$\left(w \cdot h + \sum_{i=1}^{N}\left(2k_iw_ih_i + k_i\left(\left[\frac{w_i - wp_i}{sp_i}\right] + 1\right)\left(\left[\frac{h_i - hp_i}{sp_i}\right] + 1\right)\right)\right) \cdot sizeof(type),$$

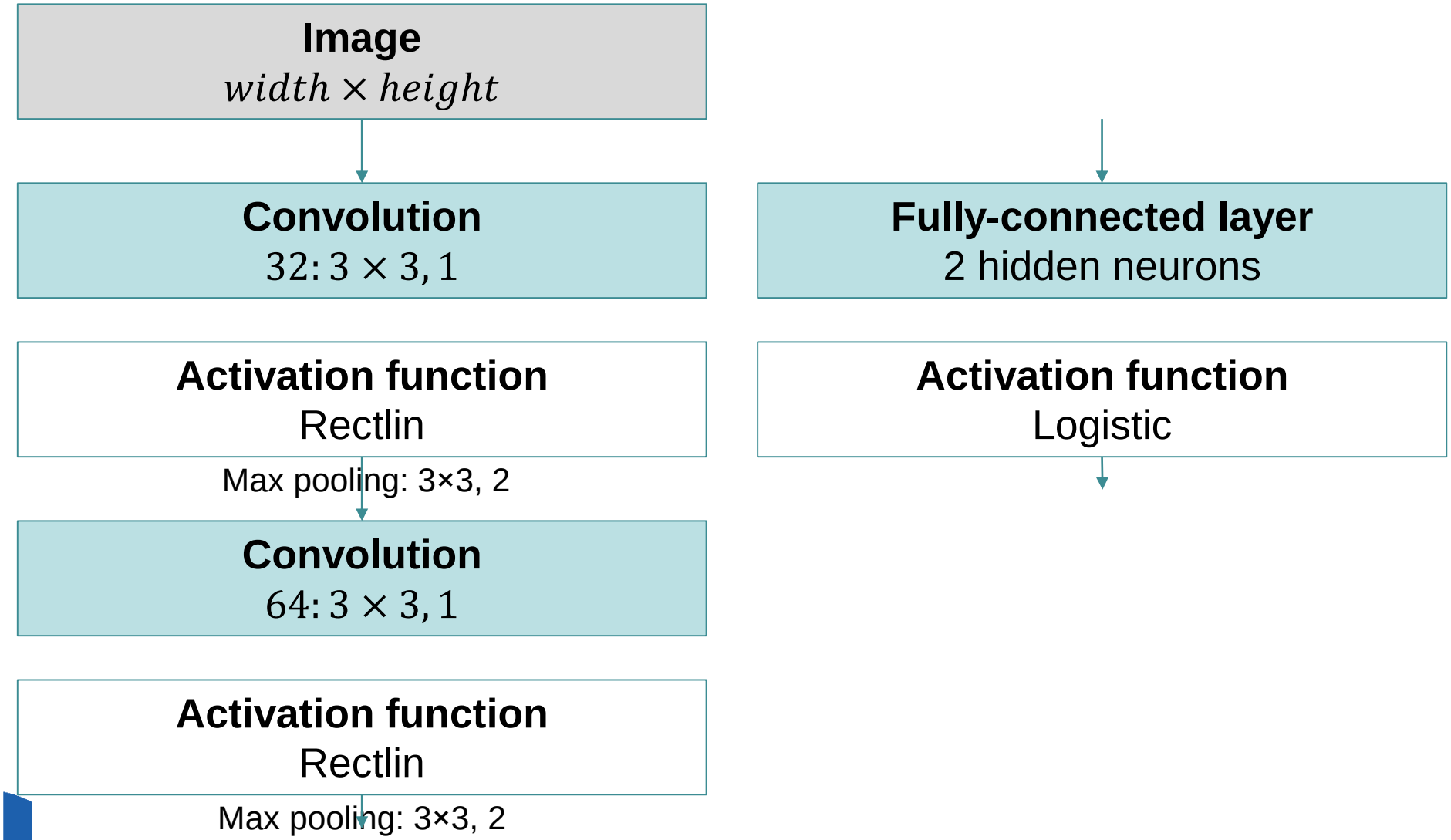
$$w_i = \left[\frac{w_{i-1} - wc_i}{sc_i}\right] + 1, h_i = \left[\frac{h_{i-1} - hc_i}{sc_i}\right] + 1, \qquad w_0 = w, h_0 = h$$

The amount of memory required to store a convolutional neural network (4)

- Described sample is a special case of a convolutional network and it represents the general steps to estimate the amount of memory required for storing this network
  - The network input might be a signal different in structure from the considered one (multidimensional signal)
  - Applying the convolution operation, the input feature map can be supplemented by borders to preserve the shape of the output feature map
  - A convolutional network may contain fully-connected layers
  - During training, a ***batch*** of images can be fed to the network input in order to improve the efficiency of calculations and the learning convergence

EXAMPLE OF A CONVOLUTIONAL NEURAL NETWORK FOR PREDICTING A PERSON'S SEX FROM A PHOTO

Convolutional neural network



Example of a convolutional network for predicting a person's sex from a photo

```
def generate_cnn_model():
    layers = [
     DataTransform(transform=Normalizer(divisor=128.0)),
     Conv(fshape=(3, 3, 32), padding=2, strides=1,
          dilation=2, init=Kaiming(), activation=Rectlin()),
     Pooling(fshape=(3, 3), padding=1, strides=2, op='max'),
     Conv(fshape=(3, 3, 64), padding=2, strides=1,
          dilation=2, init=Kaiming(), activation=Rectlin()),
     Pooling(fshape=(3, 3), padding=1, strides=2, op='max'),
     Affine(nout=class_count, init=Xavier(),
     activation=Logistic(shortcut=True)) ]
    model = Model(layers=layers)
    cost = GeneralizedCost(costfunc=CrossEntropyBinary())
    return (model, cost)
```

Infrastructure

- CPU: Intel® Xeon® CPU E5-2660 0 @ 2.20GHz
- GPU: Tesla K40s 11Gb
- OS: Ubuntu 16.04.4 LTS
- Frameworks:
  - Intel® neon™ Framework 2.6.0
  - CUDA 8.0
  - Python 3.5.2
  - Intel® Math Kernel Library 2017 (Intel® MKL)

Experiments

Network id	Network structure	Learning parameters	Accuracy, %	Training time, s
CNN-1	Conv((3,3,32),0,Rectlin), Pooling((3,3),2,'max'), Conv((3,3,64),0,Rectlin), Pooling((3,3),2,'max'), Affine(2,0), Logistic)	batch_size = 128 epoch_count = 30 backend = gpu GradientDescentMomentum(0.01, momentum_coef= 0.9, wdecay=0.0005)	79.3	1582
CNN-2	Conv((3,3,32),Explin), Pooling((5,5),2,'max'), Conv((3,3,32),Explin), BatchNorm, Conv((1,1,64),Explin), BatchNorm, Conv((3,3,64),Explin), BatchNorm, Conv((1,1,128),Explin), BatchNorm, Conv((3,3,128),Explin), BatchNorm, Conv((3,3,2),Explin), BatchNorm, Conv((1,1,2),0,Explin), BatchNorm, Pooling('avg'), Affine(2,0,Logistic)		**83.5**	2030

Comparison of the fully-connected neural networks and the convolutional neural networks

Network id	Network structure	Accuracy, %	Training time, s
FCNN-1	Affine(128,0,Tanh), Affine(2,0,Logistic)	71.2	932
FCNN-2	Affine(128,0,Tanh), Affine(64,0,Tanh), Affine(2,0,Logistic)	73.5	977
FCNN-3	Affine(400,0,Rectlin), Affine(50,0,Logistic), Affine(2,0,Logistic)	**77.7**	1013
CNN-1	Conv((3,3,32),0,Rectlin), Pooling((3,3),2,'max'), Conv((3,3,64),0,Rectlin), Pooling((3,3),2,'max'), Affine(2,0), Logistic)	79.3	1582
CNN-2	Conv((3,3,32),Explin), Pooling((5,5),2,'max'), Conv((3,3,32),Explin), BatchNorm, Conv((1,1,64),Explin), BatchNorm, Conv((3,3,64),Explin), BatchNorm, Conv((1,1,128),Explin), BatchNorm, Conv((3,3,128),Explin), BatchNorm, Conv((3,3,2),Explin), BatchNorm, Conv((1,1,2),0,Explin), BatchNorm, Pooling('avg'), Affine(2,0,Logistic)	**83.5**	2030

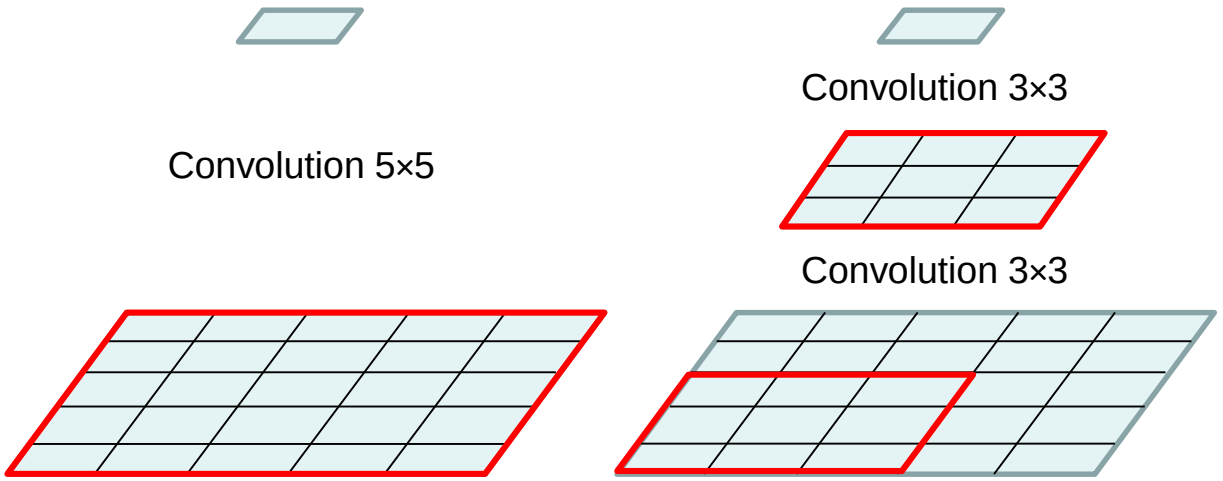
PRINCIPLES OF CONSTRUCTING CONVOLUTIONAL NEURAL NETWORKS

Principles of constructing convolutional neural networks (1)

- ***Perform preprocessing of input data***
  - Subtraction of mean image obtained over all images of the training set
  - Centering images
- ***Avoid "bottlenecks" in the network representation, especially on the first layers***
  - It makes sense to avoid extreme information compression
  - The shape of representation is an approximate estimation of the content

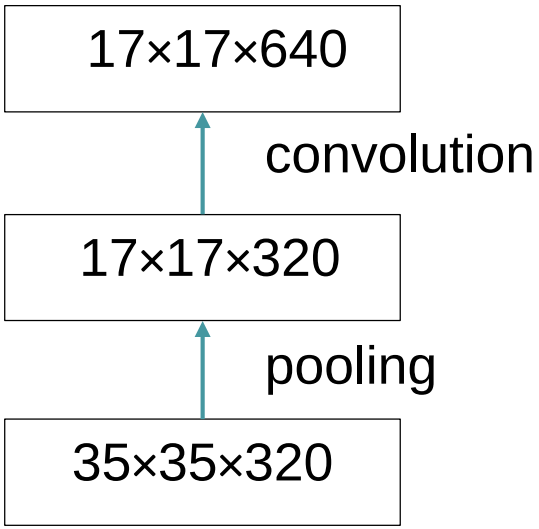
Principles of constructing convolutional neural networks (2)

- ***Replace convolutions of large dimension with a stack of convolutions of lower dimension***
  - A convolution with a filter 5×5 can be replaced by two convolutions with 3×3 filters
  - We create a network with a smaller number of parameters, but with the same input shape and output depth



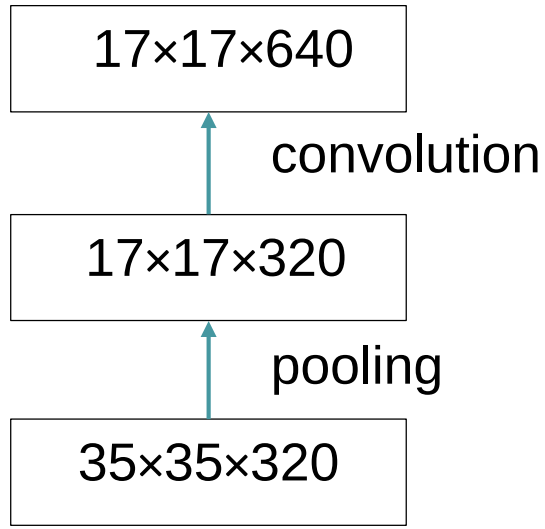
Principles of constructing convolutional neural networks (3.1)

- ***Spatial aggregation should be performed on feature maps of lower dimensionality to reduce computational complexity***
  - Implemented through pooling or the inception-modules



Principles of constructing convolutional neural networks (3.1)

- ***Spatial aggregation should be performed on feature maps of lower dimensionality to reduce computational complexity***
  - Implemented through pooling or the inception-modules

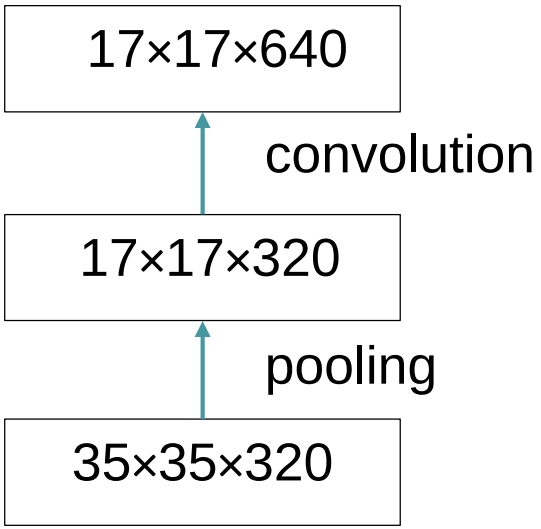


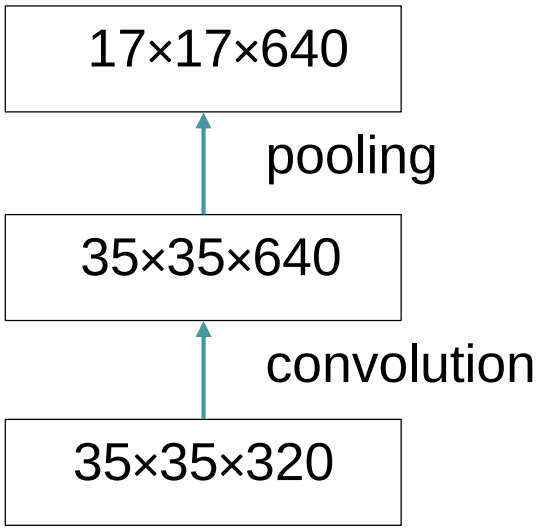
Loss of information about the initial representation

Principles of constructing convolutional neural networks (3.1)

- ***Spatial aggregation should be performed on feature maps of lower dimensionality to reduce computational complexity***
  - Implemented through pooling or the inception-modules

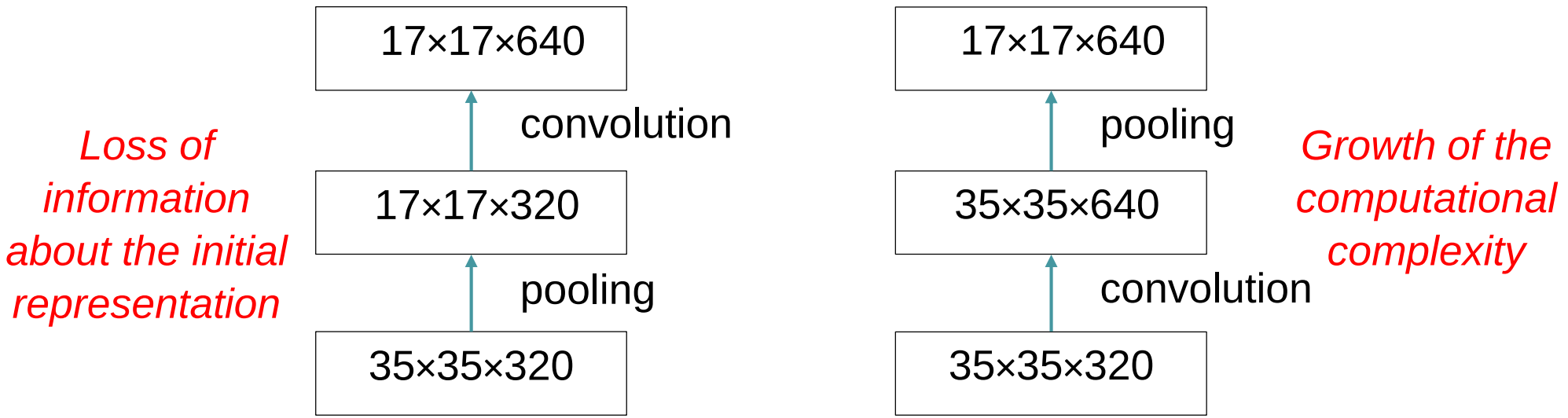
Loss of information about the initial representation





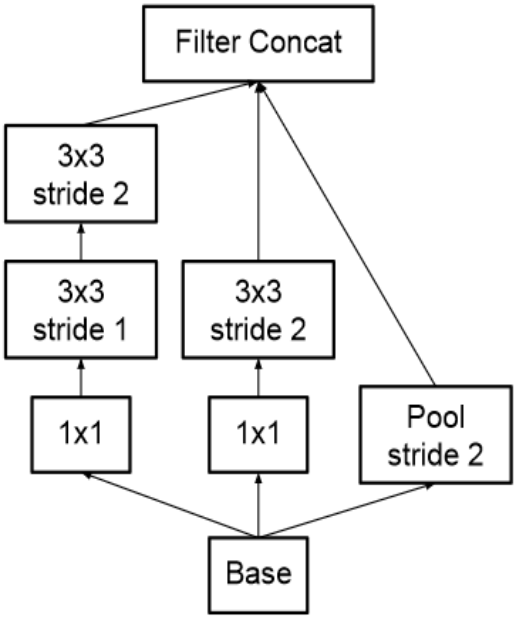
Principles of constructing convolutional neural networks (3.1)

- ***Spatial aggregation should be performed on feature maps of lower dimensionality to reduce computational complexity***
  - Implemented through pooling or the inception-modules



Principles of constructing convolutional neural networks (3.2)

- ***Spatial aggregation should be performed on feature maps of lower dimensionality to reduce computational complexity***
  - Implemented through pooling or the inception-modules



* Szegedy C., Vanhoucke V., Ioffe S., Shlens J. Rethinking the Inception Architecture for Computer Vision – [https://arxiv.org/pdf/1512.00567v3.pdf].

Principles of constructing convolutional neural networks (4)

- ***Balance the depth and width of the network***
  - Increasing the width and depth of the network can help to create networks of higher performance
  - Optimum network performance can be achieved by balancing the number of filters on each convolutional layer and the network depth

THE PROBLEM OF MODEL DEGRADATION. DEEP RESIDUAL NETWORKS

The problem of model degradation

- ***The problem of model degradation:***
  - With the increase of the network depth, the accuracy is saturated and then rapidly begins to decrease (degrade)

- The problem is not a consequence of an overfitting
- Increasing the number of layers leads to a greater training error
- The accuracy degradation indicates that not all deep models are equally easily optimized

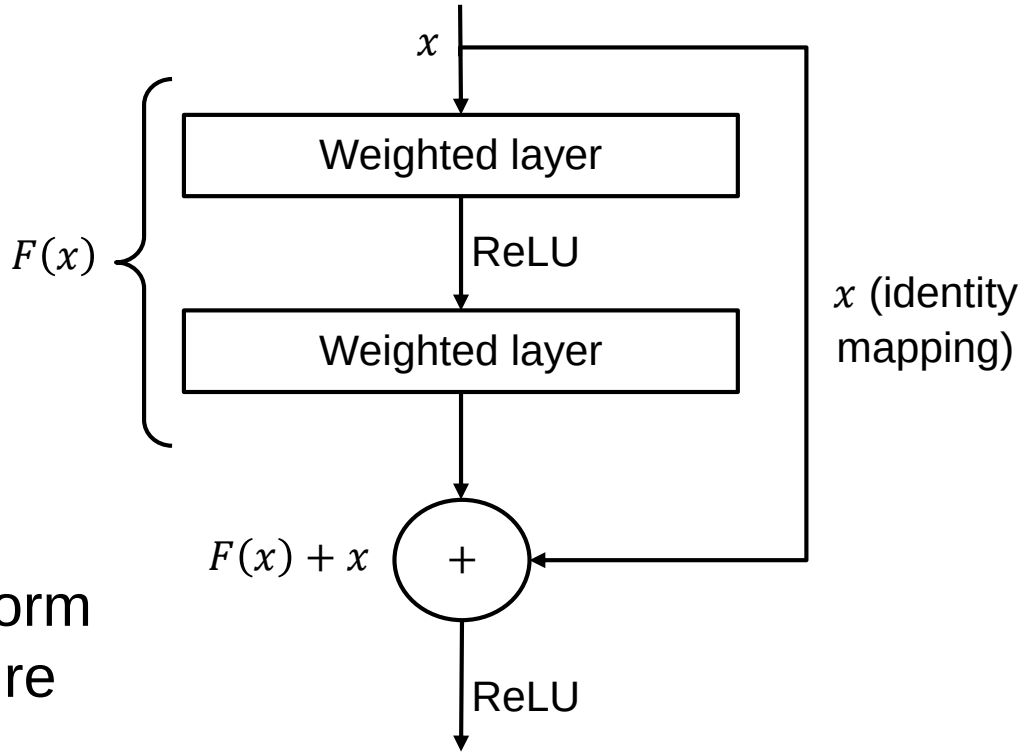
Deep residual networks (1)

- Instead of assuming that a sequence of network layers directly approximates the base mapping, it is assumed that these layers approximate the residual mapping
- Let us introduce
  - $H(x)$ is a base mapping
  - $F(x) = H(x) - x$ is a residual mapping
  - The base mapping can be represented as an element-by-element addition of feature maps $F(x) + x$
- It is assumed that the residual mapping is easier to optimize than the base mapping. In the extreme case, if the identity transform is optimal, then it is simpler to reduce the remainder to zero than to approximate the identical mapping by a set of nonlinear layers

Deep residual networks (2)

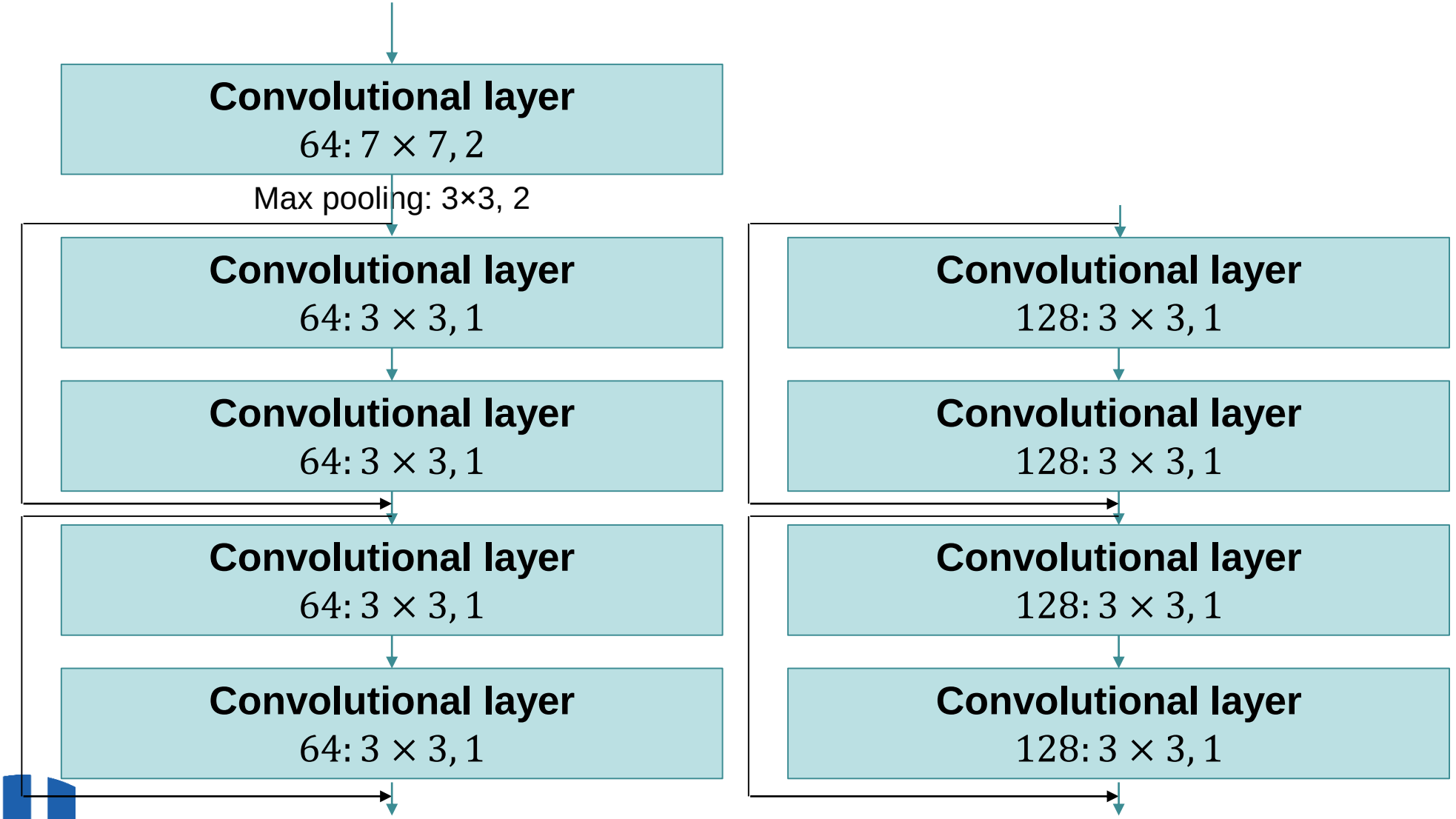
- The mapping $F(x) + x$ can be represented as a feed-forward network with shortcut connections

- For the represented example $y = F(x, W_i) + x = W_2\varphi(W_1 x) + x$, where $\varphi(\cdot)$ is the activation function ReLU
- $F(x, W_i)$ and x can have different dimension. To fix this difference it is sufficient to perform the projection of the input feature vector $y = F(x, W_i) + W_s x$

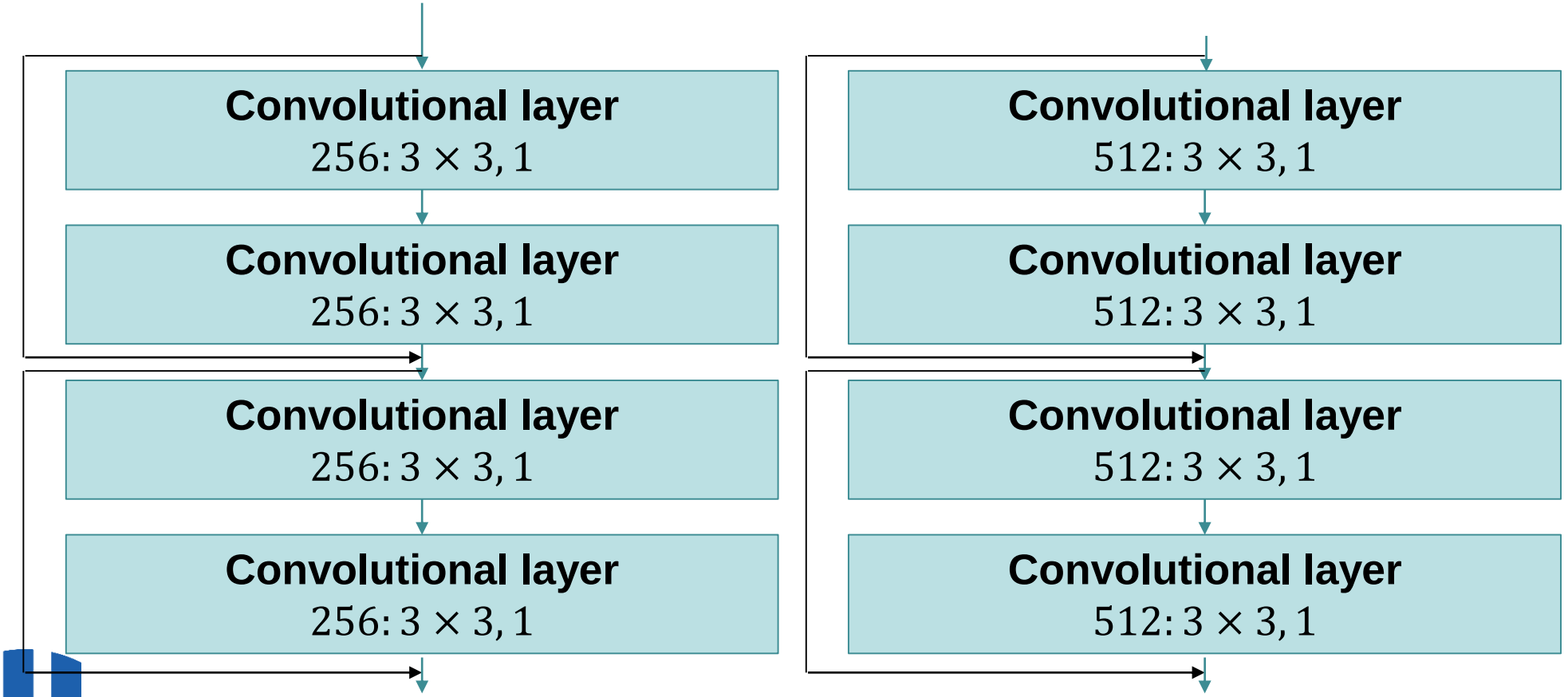


EXAMPLE OF A RESIDUAL NETWORK FOR PREDICTING A PERSON'S SEX FROM A PHOTO

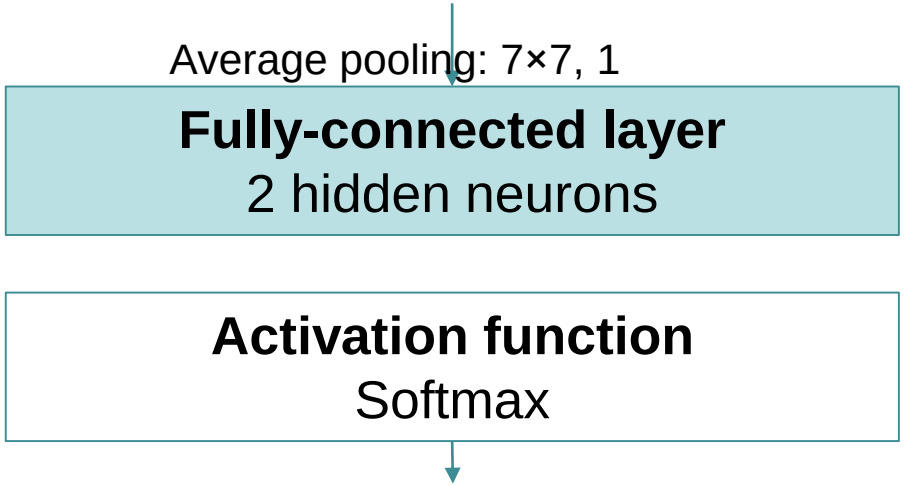
The residual neural network ResNet-18 (1)



The residual neural network ResNet-18 (2)



The residual neural network ResNet-18 (3)



- ***Notes:***
  - The normalization **`BatchNorm`** and the activation function **`ReLU`** follow after each convolutional layer
  - ResNet-34, ResNet-50, ResNet-101 are constructed in the same way

Example of a residual network for predicting a person's sex from a photo (1)

```
def generate_resnet18_model():
    layers = make_resnet_base([2, 2, 2, 2], block='basic')
    layers.append(BatchNorm())
    layers.append(Activation(Rectlin()))
    layers.append(Pooling('all', op='avg'))
    layers.append(Affine(2, init=Kaiming(local=False),
                         activation=Softmax()))

    model = Model(layers=layers)

    cost = GeneralizedCost(costfunc=CrossEntropyMulti())

    return (model, cost)
```

Example of a residual network for predicting a person's sex from a photo (2)

```
def make_resnet_base(stage_depths, block='basic',
                     base_channels=64):
    from math import log2

    def conv_params(fsize, nfm, stride=1, relu=True,
                    batch_norm=True):
        return dict(
            fshape=(fsize, fsize, nfm),
            strides=stride,
            padding=fsize // 2,
            activation=(Rectlin() if relu else None),
            init=Kaiming(local=True),
            batch_norm=batch_norm)
...
```

Example of a residual network for predicting a person's sex from a photo (3)

```
def module_basic(nfm, first=False, stride=1):
        # building block for ResNet-18
        mainpath = [] if first else [BatchNorm(),
                                     Activation(Rectlin())]
        mainpath.append(Conv(**conv_params(3, nfm,
                             stride=stride)))
        mainpath.append(Conv(**conv_params(3, nfm,
                             relu=False, batch_norm=False)))
        sidepath = Conv(**conv_params(1, nfm, stride=stride,
                             relu=False, batch_norm=False))
                   if (first or (stride != 1)) else SkipNode()

        return MergeSum([sidepath, mainpath])
...
```

Example of a residual network for predicting a person's sex from a photo (4)

```
def module_bottleneck(nfm, first=False, stride=1):
        # building block for ResNet-50, -101, -152
        mainpath = [] if first else [BatchNorm(),
                                     Activation(Rectlin())]
        mainpath.append(Conv(**conv_params(1, nfm,
                                     stride=stride)))
        mainpath.append(Conv(**conv_params(3, nfm)))
        mainpath.append(Conv(**conv_params(1, nfm * 4,
                                     relu=False, batch_norm=False)))

        sidepath = Conv(**conv_params(1, nfm * 4,
                stride=stride, relu=False, batch_norm=False))
                if (first or (stride != 1)) else SkipNode()
        return MergeSum([sidepath, mainpath])
```

Example of a residual network for predicting a person's sex from a photo (5)

```
blocks = {'basic': module_basic, 'bottleneck':
           module_bottleneck}
# 18:  [2, 2, 2, 2], output = 512, block = 'basic‘
# 34:  [3, 4, 6, 3], output = 512, block = 'basic'
# 50:  [3, 4, 6, 3], output = 2048, block = 'bottleneck'
stage_depths_populated = []
for stage, depth in enumerate(stage_depths):
    stage_depths_populated.extend([stage] * depth)
# nfms is a list of channel counts in blocks
nfms = [2**(stage + int(log2(base_channels))) for stage
       in stage_depths_populated]
strides = [1 if cur == prev else 2 for cur, prev
           in zip(nfms[1:], nfms[:-1])]
module = blocks[block]
```

Example of a residual network for predicting a person's sex from a photo (6)

```
    # Now construct the network
    layers = [
        Conv(**conv_params(7, base_channels, 2)),
        Pooling(fshape=(3, 3), padding=1, strides=2,
        op='max')
    ]
    layers.append(module(nfms[0], first=True))
    for nfm, stride in zip(nfms[1:], strides):
        layers.append(module(nfm, stride=stride))

    return layers
```

Infrastructure

- CPU: Intel® Xeon® CPU E5-2660 0 @ 2.20GHz
- GPU: Tesla K40s 11Gb
- OS: Ubuntu 16.04.4 LTS
- Frameworks:
  - Intel® neon™ Framework 2.6.0
  - CUDA 8.0
  - Python 3.5.2
  - Intel® Math Kernel Library 2017 (Intel® MKL)

Comparison of the fully-connected neural networks, the convolutional and residual neural networks

Network id	Accuracy, %	Training time, s
FCNN-1	71.2	932
FCNN-2	73.5	977
FCNN-3	**77.7**	1013
CNN-1	79.3	1582
CNN-2	**83.5**	2030
ResNet-18 (90 epochs)	**81.3**	15127
ResNet-50 (30 epochs)	**80.9**	11849

Conclusion

- The use of convolutional networks makes it possible to improve the accuracy of solving the problem
- The construction of residual neural networks does not allow us to obtain a significant gain relative to the fully-connected models for the problem of person's sex classification
- The effectiveness of the residual network application in solving other practical problems should be checked experimentally

Literature

- Haykin S. Neural Networks: A Comprehensive Foundation. – Prentice Hall PTR Upper Saddle River, NJ, USA. – 1998.
- Osovsky S. Neural networks for information processing. – 2002.
- Goodfellow I., Bengio Y., Courville A. Deep Learning. – MIT Press. – 2016. – [http://www.deeplearningbook.org].

Authors

- **Kustikova Valentina Dmitrievna**
  Phd, lecturer, department of Computer software and supercomputer technologies, Institute of Information Technologies, Mathematics and Mechanics, Nizhny Novgorod State University
  valentina.kustikova@itmm.unn.ru
- **Zhiltsov Maxim Sergeevich**
  master of the 1st year training, Institute of Information Technology, Mathematics and Mechanics, Nizhny Novgorod State University
  zhiltsov.max35@gmail.com
- **Zolotykh Nikolai Yurievich**
  D.Sc., Prof., department of Algebra, geometry and discrete mathematics, Institute of Information Technologies, Mathematics and Mechanics, Nizhny Novgorod State University
  nikolai.zolotykh@itmm.unn.ru