Computer Vision

Module M6:
Convolutional Neural Networks

Krzysztof Krawiec
http://www.cs.put.poznan.pl/kkrawiec/

Poznan University of Technology, 2021-2023

Module outline

1. Introduction
2. Basic concepts
3. Milestone CNN architectures, designed mostly for object/image classification:
   - AlexNet
   - VGG models
   - Inception Network
   - Residual networks
   - Highway networks and DenseNet
4. Fully convolutional architectures for image segmentation
   - Patch-based segmentation
   - U-Net and UNet++
5. Fully convolutional architectures for image denoising

Introduction to Convolutional Neural Networks

Basic concepts and motivations (Neural Networks 101)

Neural networks: Recap (1)

A directed graph of interconnected (usually stateless (memory-less)) computing units (*units*, neurons), each of which performs the aggregation of input signals by:

1. Weighted sum (scalar product), and then
2. Nonlinear transformation (activation function).

$$y = act(\sum_i w_i x_i + b)$$

where:

- x: vector of input signals
- w: vector of weights (parameters)
- b: free term (bias).

Neural architectures formed from such units are trained from examples using gradient *descent* algorithms.

Neural networks: Recap (2)

- Network units are often grouped into larger entities called layers.
  - Example: *dense* layer, a.k.a. *fully connected*: an ordered list of units, each of which receives the same input signal vector x.
- The architecture (graph/topology of connections) between units is usually preset and does not change during the learning process.
  - The learning process is only concerned with optimizing ('tuning') of the parameters (weights and biases)
  - Other aspects of the model (architecture, training algorithm, ...) are hyperparameters.
- The computational capabilities of a neural model originate in the number of units and the architecture of the connections and aggregation method,
  - rather than the individual capabilities of units – these are very limited.
- Neural networks are universal approximators:
  - Under certain assumptions, a network composed of two layers can model any continuous mapping with any precision [Cybenko 1989].

Cybenko, G. Approximation by superpositions of a sigmoidal function. Math. Control Signal Systems 2, 303-314 (1989).
https://doi.org/10.1007/BF02551274

NN training in a nutshell

Model training/fitting = optimization of network parameters.

Neural networks are a special case of a machine learning model - hence the term 'model' appearing in the following slides.

Inputs:

1. A training set in the form of pairs (x,y), where
   a. The x is the input to the model,
   b. y is the desired (expected) output corresponding to x,
2. A network architecture f, compatible with types of x and y
   a. Parameters initialized by some method, usually small signed values (e.g., in [-0.1,0.1]).
   b. Note: the distribution of initial weights can have a critical impact on the efficiency of the learning process (e.g., different activation functions 'prefer' different distributions).
3. A loss function L that quantifies how the model diverges from the desired behavior exemplified by the training set.

NN training in a nutshell

Typical training loop:

1. Query the model on an example x, obtaining the output of the model y'=f(x)
2. Calculate the loss function L(y,y') and its gradient w.r.t. model parameters.
3. Correct the model parameters guided by the gradient descent (*gradient descent*)

   $$w \leftarrow w - \eta \frac{\partial L}{\partial w}$$

   where w is the vector of model parameters and η is the learning rate hyperparameter.
4. If no stopping conditions are met, jump to 1.

- Known as error backpropagation algorithm and stochastic gradient descent.
- Gradient calculated analytically with specialized automatic differentiation (autodiff) algorithms backed by libraries (Pytorch, TensorFlow, JAX, …).

NN training in a nutshell: Technical comments

- Gradient descent is a heuristic – the gradient vector does not necessarily point toward the global minimum of the loss function.
  - Training may get stuck in a local optimum.
  - However, local optima become less likely when the number of parameters is large.
- Gradient descent is deterministic.
  - However, the starting point (initial parameters) is drawn at random.
  - The order of presenting examples is often randomized too.
  - Advanced low-level implementations can dynamically reschedule operations, leading to indeterminism.
    - Recall that floating-point addition and multiplication is not associative: $(a+b)+c \neq a+(b+c)$
- Examples often given in *batches* (*minibatches*).
- Signals represented as *vectors*.
- Model parameters stored as *matrices*.
- Both vectors and matrices implemented technically as *tensors*.

The concept of tensor

In modern deep learning, operations (as well as their implementations) are often defined for an arbitrary number of dimensions:

- 1-dimensional (1D, e.g. time)
- 2-dimensional (2D, height x width)
- 3-dimensional (3D, height x width x depth)
- 4-dimensional (4D, height x width x depth x time)
- etc.

Hence, the prevailing assumption is that the data processed by the individual components are represented as *tensors*, or multidimensional arrays.

The concept of tensor

- A 2D raster image is a 2- or 3-dimensional tensor (3-dimensional if we assume a separate dimension for the channels, which is the norm).
  - The input images and tensors transmitted between layers in network are technically no different from each other, which is convenient.
- For 2D images, convolutional layers (or entire models) work on batches represented as 4-dimensional arrays (tensors) of dimensions:
  - N: batch (example number in the package)
  - H: height (height, Y coordinate)
  - W: width (width, X coordinate)
  - C: channel (channel, depth).
- The two most common technical conventions: NHWC (channels last) and NCHW (channels first)
  - NHWC often required, and more efficient in hardware implementations (e.g., Tensor Cores)

Examples vs. receptive fields

- In Computer Vision, an *example* is usually an *image* (example = image)
- However, from the unit's point of view, a single example is a fragment of an image visible in its receptive field.
  - Thus, even within a single image, convolutional units learn simultaneously from multiple receptive fields.
  - Each image conveys multiple examples.
  - This makes learning of convolutional layers more efficient (in the sense of 'data-efficient'*) and less prone to overfitting.

*Data efficiency refers to the ability of a machine learning method to produce an effective model with a limited volume of learning data (usually measured by the number of examples).

- More technically: the number of examples required to construct a well-generalizing model.

Why neural networks for computer vision?

The renaissance of neural networks (1)

- Neural models have been present in AI since its inception.
  - Many early neural models were oriented toward processing image information and/or drew inspiration from the organization of visual perception centers in primates and non-human primates; e.g., Perceptron, Neocognitron, and others.
- Interest in neural models declined in the second half of the 1990s: the models did not scale in terms of task complexity.
  - In particular: the lack of efficient learning algorithms for architectures with a large number of layers (indicative: more than 3).
    - So, no possibility of 'complexification' of models (*complexification*).
  - Approximate level of capability at the time: fairly reliable recognition of handwritten (isolated) digits using weave networks (mainly Yann LeCun and his research group).

LeCun, Y.; Bottou, L.; Bengio, Y. & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE. 86(11): 2278 - 2324

The renaissance of neural networks (2)

Since the middle of the first decade of the 21st century, NNs experience an impressive comeback, redressed as deep learning, thanks in part to:

1. Conceptual and theoretical advances (new activation functions, extensions to improve gradient propagation and generalization: dropout, batch normalization, and others)
2. Technological advances: efficient implementations using the GPGPU (General Purpose Computing on Graphics Processing Units) paradigm, enabling strong parallelization of computation.
    - More recently: specialized hardware like Tensor Cores in GPUs, Tensor Processing Units (TPUs), Neural Engines in M1/M2 Apple Chips, and more.

These advances have made it possible to construct more complex models with more layers, that is, deeper, hence the popularity of the terms *deep neural networks* and deep *learning*.

Motivations (1)

Motivations for using NNs/DL in image analysis and computer vision:

1. Many of the operations used in image analysis are differentiable (e.g., convolution), and therefore can be parameterized with gradient algorithms typically used for NNs.
2. Many of the operations used in image analysis are local (based on a local window of the image), and thus naturally parallelizable, which coincides with the parallel/distributed information processing model typical for NNs (multiple independent neurons/units).
3. We represent raster images as vectors, matrices and tensors, the processing of which in NNs is very natural.

Motivations (2)

Motivations for using NNs/DL in image analysis and computer vision:

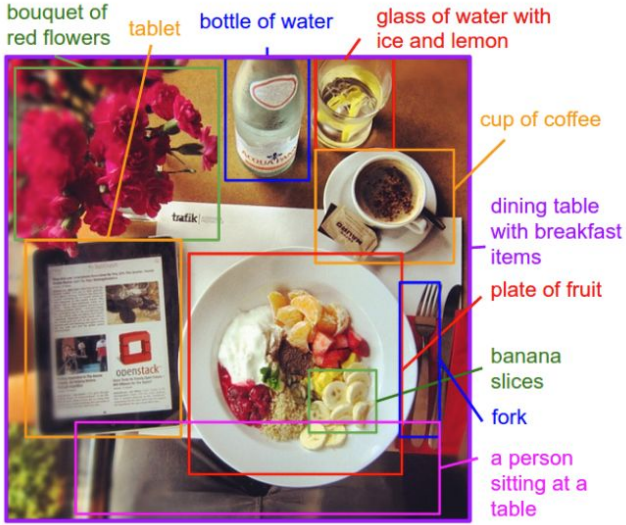
1. Opens the door to the rich repertoire of resources of modern machine learning and deep learning, including architectures, training algorithms, etc.
2. Neural networks (and machine learning more broadly) address one of the fundamental challenges of CV: the difficulty of designing and parameterizing efficient algorithms/systems that solve specific tasks.
    a. DL facilitates semi-automation of that process.
3. Ease of merging subnetworks/components with others, within a single paradigm and technology, facilitating production of end-to-end solutions.
    a. Also: bridging with other data modalities, e.g. automated *image captioning* (see next slide).

Motivations (2): Example

Automatic *image captioning*.

The deep neural network model consists of:

- A stack of convolutional layers for processing of input images.
- A recursive subnetwork/layer for generating tokens representing sentences in natural language.



Advantage: The model can be taught *end-to-end*, and learns intermediate representations (*latent/hidden representations*).

Andrej Karpathy and Li Fei-Fei. 2017. deep Visual-Semantic Alignments for Generating Image Descriptions. IEEE Trans. Pattern Anal. Mach. Intell. 39, 4 (April 2017), 664-676. DOI:https://doi.org/10.1109/TPAMI.2016.2598339

Types of architectures

Two main categories (from the point of view of image analysis):

1. Fully convolutional neural networks (CNNs, ConvNets)
    - Consist only of convolutional layers and other layers that preserve the locality of processing (such as maxpooling; more later).
    - [It would be more appropriate to call them *spatially local* architectures]

2. Mixed architectures
    - On top of layers that process image locally, they feature other components, most often dense (fully connected) layers or other components.

See next 2 slides for details.

Fully convolutional architectures (CNNs, ConvNets)

Typical applications:

- *Image de-noising* (*image denoising*)
- Feature detection
  - E.g. feature detectors learned using generative adversarial networks (GAN) models
- Style transfer: transformation of an original image into a preset aesthetic/artistic style
  - E.g., the well-known Pix2pix model
- Image segmentation, including
  - Semantic *segmentation*
  - [*object*] *instance segmentation*
- Describing images in natural language (*image captioning*)

The list of applications and usage scenarios grows longer every year.

Mixed architectures

Common applications:

- Classification
  - Usually a stack of *convolutional* layers (*convolutional stack*), followed by a dense subnet, aggregating information across the image.
- Regression
  - E.g., assessment of image quality (e.g. in medical applications).
- Generating images (generative models, e.g., generative adversarial networks, GANs)
- Combining with deep natural language processing (NLP) models, e.g., in terms of describing images (*image captioning*)
  - Often involves recurrent subnetworks, such as LSTM (Long Short-Term Memory) or GRU (Gated Recurrent Unit) models.
  - Or transformer-type models (e.g., BERT).

Formalization

The convolution operation

Recall the definition of one-dimensional convolution:

$$g(x) = (f * M)(x) = \sum_{h=-\left\lfloor \frac{k}{2} \right\rfloor}^{\left\lfloor \frac{k}{2} \right\rfloor} M(h) f(x-h)$$

where f is the image, M a mask (filter, kernel) of dimensions kxk.

In traditional image analysis, it is assumed that M is a given, i.e. it was designed to achieve a specific goal/effect, e.g.

- Gaussian mask (normal distribution) purpose of de-noising,
- Sobel or Schar filter for edge detection,
- Laplasian to detect transitions through zero of the second derivative.

In neural approaches, M becomes part of the model and is subject to automatic optimization (learning).

Convolution versus convolution: Differences

In the following slides, we summarize the most significant differences between the convolution operation used in traditional image processing and the convolutional layers used in neural networks.

- Illustrating these differences will help us better understand the 'added value' offered by neural models.

Differences (1)

As in typical layers in neural networks, the outputs of units in convolution layers are passed through a nonlinear activation function:

$$g(x) = act\left(\sum_{h=-\left\lfloor \frac{k}{2} \right\rfloor}^{\left\lfloor \frac{k}{2} \right\rfloor} M(h) f(x-h)\right)$$

This is necessary because the output of a convolutional layer is usually passed to the inputs of the next layer (convolutional or not), and the units in that layer realize the scalar product (dot product, weighted sum) of the inputs. A direct connection to the next layer would be a composite of two linear operations, and this would miss the point, since it could be replaced by a single convolution.

- Typical activation functions are s-shaped (squeezing functions), or the popular Rectified Linear Unit (ReLU), i.e. act(v) = max(v, 0).
- With new advances in deep learning, we can construct and effectively train models with dozens of convolutional layers.

Differences (2)

Like typical units in neural networks, convolutional units in convolutional layers are equipped with a threshold (bias, offset, free parameter) b, which is also subject to learning. That is, a convolution in DL is *de facto* defined as:

$$g(x) = \sum_{h=-\left\lfloor \frac{k}{2} \right\rfloor}^{\left\lfloor \frac{k}{2} \right\rfloor} M(h) f(x-h) + b$$

This is essential because:

- We do not know in advance whether the input signals are zero-centered (i.e. whether the average f=0), especially when the inputs to the layer come from earlier layers in the architecture
  - In the presence of a bias (lack of zero-centering), the algorithm can tune b to compensate for that.

Differences (3)

In traditional image analysis, we usually use single convolutions. In neural networks, almost always:

1. We apply convolution to multiple input channels, summing the result of the convolutions:

$$g(x) = \sum_c \sum_{h=-\left\lfloor \frac{m}{2} \right\rfloor}^{\left\lfloor \frac{m}{2} \right\rfloor} M_c(h) f_c(x-h)$$

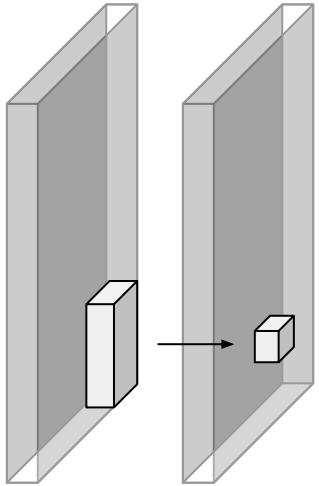
   where M_c is the mask for channel c and f_c is the channel c of the input image.
2. We conduct many such multi-channel convolutions in parallel.

So a convolutional layer accepts an n-channel image at the input and produces an m-channel image at the output.

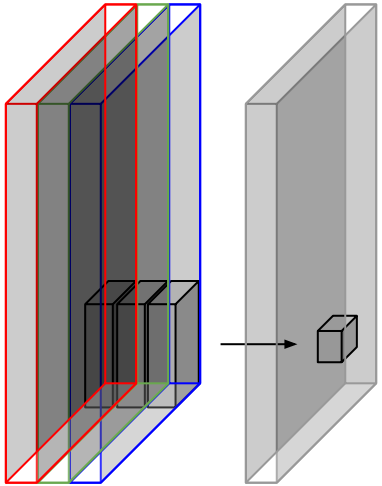
- Each output channel is a linear combination (before the activation function) of the input channels ⇒ a convolutional layer can 'mix' channels.
- The raster associated with one output channel is known as a feature map.
- The number of channels: depth [tensor] (depth).

Illustration

Traditional convolution of a single-channel (scalar) signal (image)

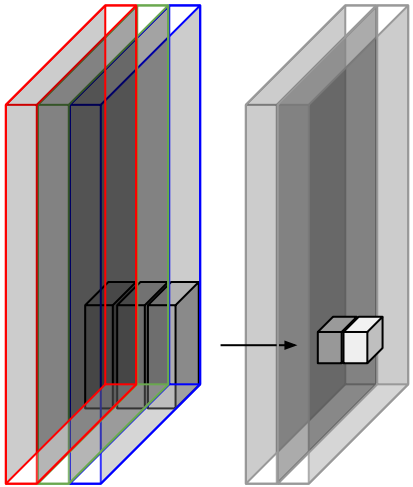


A convolutional layer applied to a 3-channel image: the unit aggregates the input channels into a single output channel.

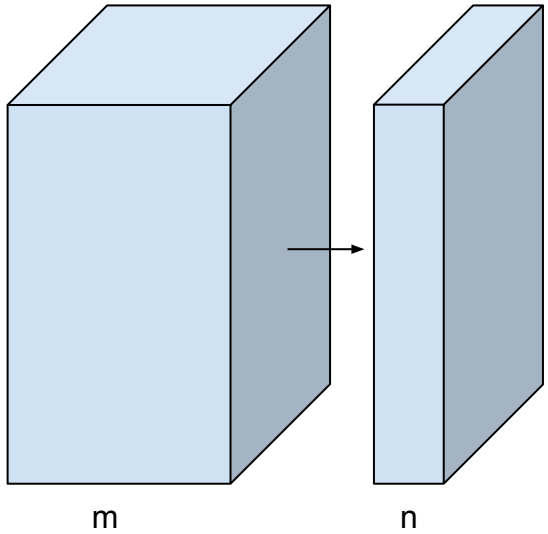


Illustration

General operator: applying a convolution to an m-channel image (tensor) results in an n-channel tensor (here m=3, n=2)



Any m and n



Note that this case should not be equated with a three-dimensional convolution: m and n are constant.

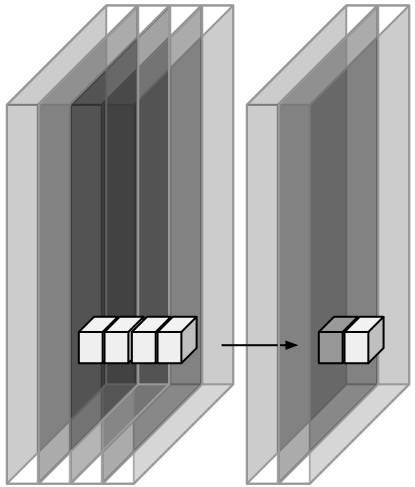
Differences (4)

Implication of multichannel processing (previous slide): it becomes reasonable to use convolution with a 1x1 mask/kernel.

- Such a mask does not realize spatial aggregation of the signal.
- Nevertheless, it aggregates the signal across channels
  - (and transforms the result of this aggregation nonlinearly).
- Useful for reducing the number of channels, which in turn helps reduce the number of parameters (weights) in subsequent network layers.
  - The number of channels used in contemporary DL is often in orders of tens or even hundreds.
- Known in the literature as "1 by 1 convolutions"
  - Of course, there is no point in applying 1x1 convolutions to single-channel images.

Illustration: 1x1 convolution

Channels in the output tensor aggregate the channel values in the input tensor. No spatial aggregation.



Observation:

- A 1x1 convolution is equivalent to applying a dense layer (dense, fully connected) to each pixel independently.
- For this diagram, it would be a dense layer consisting of 2 units, each of which is equipped with 4 inputs.

The number of parameters of a convolution layer

For a mask (kernel) of size kxk pixels, a convolutional layer with m input channels and n output channels has a number of parameters equal to:

$$(k^2 m + 1)n$$

For example, (3^2 *16+1)*32 = 4640

- Not much by today's DL standards; dense layers often have many more parameters.
- Moreover, this does not depend on the size of the input image.
  - Technical implication: we can apply software components implementing convolutional layers (e.g., Conv2D objects in TensorFlow) to images of any size.
  - Image sizes can be determined 'on the fly'.
- Many studies have shown that k=3 is often sufficient, hence in practice, the main determinants of the number of parameters are m and n.
  - ... which indicates the usefulness of 1x1 splices (see previous slides).

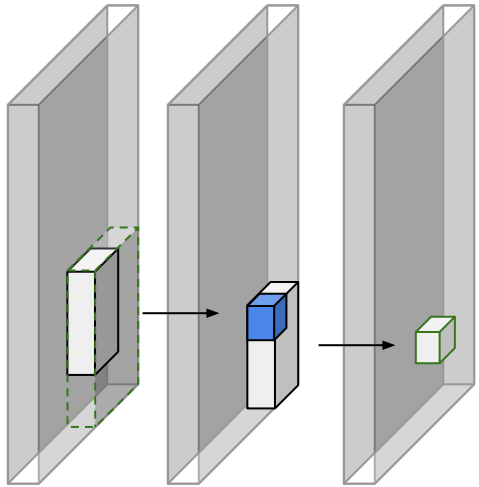
The concept of an effective receptive field

Example: Consider composing two convolutional layers A and B, both with mask sizes of 3x3.

- Each unit in B depends on a 5x5 area in the image fed to A.
- We say that the *effective receptive field* of the unit in B is 5x5.

In models with many convolutional layers,
the ERF can be very large.

- ... and often covers the entire input image.
- This is often desirable, allowing the model to develop global features of the image.



The green unit in Layer 3 aggregates values within a mask in Layer 2. Each unit in Layer 2 aggregates signals in a mask from Layer 1 (among others, the highlighted blue unit). The highlighted green unit in Layer 3 has an ERF in Layer 1 marked with a dashed green line.

Hyperparameters of convolutional layers (1)

Kernel size, k:

- The dominant setting in current practice: k=3
- Provides minimal spatial aggregation (for odd k; even k is not likely to be used),
- Reduces the total number of required parameters.
  Example: a 5x5 ERF can achieved with:
  - One layer with a 5x5 kernel ⇒ $5^2 + 1 = 26$ parameters
  - Two layers with 3x3 kernels ⇒ $2(3^2 + 1) = 20$ parameters
    The two-layer solution has fewer parameters, while involving additional nonlinearity between the layers, which may allow more sophisticated features to be modeled.

In most implementations, k can be set independently for each spatial dimension (height, width, ...), i.e. the masks do not have to be square.

Hyperparameters of convolutional layers (2)

Padding: How to handle pixels for which the mask 'extends' beyond the input tensor? Several variants:

- *valid*: we do not allow mask to extend beyond the tensor; consequently, the output tensor is k-1 pixels smaller on each dimension
- *same*: we allow the extension: in such cases, the mask also aggregates 'virtual' pixels from beyond the image.
  The values of such pixels have to be set somehow; the default solution: complementing with zeros.
  - May cause a sudden change in brightness at the edge of the image, resulting, for example, in the appearance of apparent edges.
  - Better* output: simulating image continuation outside the raster range by
    - Mirroring the edge portion of the image (odd or even mirroring), or
    - Repeating edge pixels.
  - *From experience: Not necessarily always better.

Hyperparameters of convolutional layers (3)

Stride (step) of shifting the mask relative to the input tensor:

- stride = 1:
  Default setting: as in traditional image processing.
- stride > 1:
  The output tensor becomes *stride* times smaller.
  - Note: when stride > k, certain pixels in the input tensor will be completely ignored.
- 0 < stride < 1:
  *Transposed/fractional convolution**.
  - Often implemented independently, with a separate parameter (dilation)

In most implementations, stride can be set independently for each spatial dimension (height, width, ...).

*Sometimes called (incorrectly) deconvolution - this term translates to 'unbundling', and it means a completely different operation in signal processing.

Hyperparameters of convolutional layers (4)

Other parameters:

- Activation function
  - It is often embedded in a programming function (or object) representing the conv layer, allowing a more efficient low-level implementation.
    - For example, process profiling in TensorFlow: specialized low-level functions that combine implementation of scalar product and thresholding (for ReLU activation)
- Channel grouping
  - It forces aggregation only in groups of channels (unlike a typical layer, where each output channel aggregates all input channels).
  - A special case: *channel separable convolution*.

Parameters related to the learning algorithm:

- Initialization of weights
  - The bias is sometimes treated differently.
- Normalizations (batch normalization, BN; layer normalization, LN)

Visualizations of convolutions

https://github.com/vdumoulin/conv_arithmetic

Presents also the *dilated convolutions*.

Other local operations

Similarly to convolution, they rely on spatial aggregation of signals.

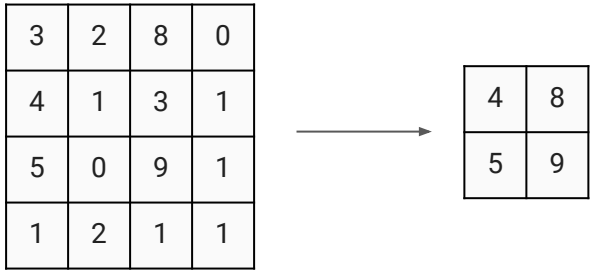
- The result is still a spatial tensor (although it may have a different size).
- The number of channels usually preserved (processing takes place in each channel independently).

Frequently used operations:

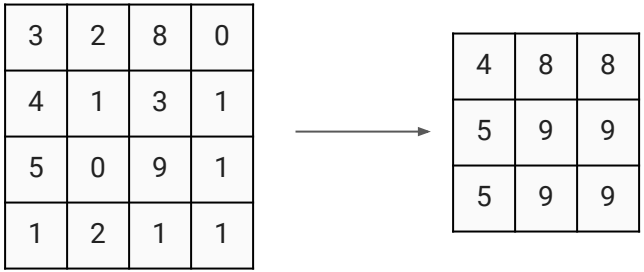
- Grouping/aggregation (pooling):
  - Particularly popular: max pooling: output is the maximum from the values in the kxk window
  - Similarly: min pooling, average pooling, ...
  - The 'max' operator used particularly often: strong response signals the presence of a feature at a given image location.
    - Particularly true for activation functions like ReLU.
- Image/tensor resampling
- Normalizations: per batch, per layer, per channel, etc.

Max pooling: Examples

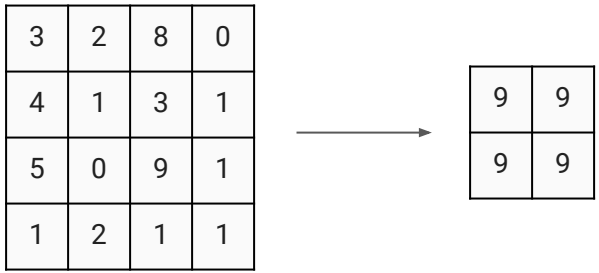
For a window size of 2x2, stride=2:



For a window size of 2x2, stride=1:



For a window size of 3x3, stride=1:



Max pooling is usually used with the stride equal to the size of the mask (like the example in the upper left corner of this slide).

Pooling and gradient propagation

Signal pooling can impede backward gradient propagation.

The line of argument: consider the composition of the maximum operator with three other functions f, g, and h:

$$h(\max(f(x_1), g(x_2))$$

When $f(x_1) < g(x_2)$, we have:

$$\frac{\partial \max}{\partial f} = 0$$

The chain of partial derivatives defining the partial derivative of h is zeroed:

$$\frac{\partial h}{\partial \max} \frac{\partial \max}{\partial f} \frac{\partial f}{\partial x_1} = 0$$

As a result, the parameters of f are not updated (in a given training step).
In max pooling, this means that the gradient is propagated to only one pixel/element (location) in the input mask.

Covariance of convolutional layers

Convolution, pooling and other operations typical for fully convolutional architectures have in common the algebraic property of *covariance*.

An operation f is *covariant* with respect to some transformation operator T when it *commutes* with it, i.e.

$$f \circ T = T \circ f$$

For ConvNet architectures, T is image translation.

- As a result, it does not matter whether we translate the input image and then apply f to it, or the other way round (f being an arbitrarily complex ConvNet architecture)

Learning process

Training convolutional layers

The mask/kernel M becomes part of the model and is subject to automatic optimization (training).

More precisely: M is ‘unrolled’ (flattened) into a vector of weights (parameters) of a single unit (neuron), which is applied in parallel to all locations in the image.

The result: the goal of optimizing M is not explicitly specified - it is only part of a more general goal, which is to minimize the loss function of the entire model (e.g., classification or regression error). Thus, the learning process can:

- Explore the rich design space of kernels, going far beyond typical filters,
- Learn from examples (rather than from ‘first principles’), which can greatly facilitate design and parameterization.

Example: Single pixel classification (segmentation)

Goal: Assign each pixel to one of two decision classes (binary classification), based on its local environment*.

This problem can be approached with a very simple (essentially minimal) model:

1. One convolutional layer with kxk mask
2. The outcome of convolution transformed by a sigmoidal activation function, which maps it to the positive class probability.
3. Loss function: cross-entropy: penalizes the discrepancy between unit output (in (0,1)) and desired output (0 or 1).

*Example practical application: pixel classification in aerial and satellite imaging, such as vegetation types, water bodies, etc.

A different perspective: weight sharing

Conceptually, convolution involves moving a mask (or unit) ‘over' the image.

Alternatively, we can assume that:

- There are as many units are as pixels in the input image,
- Each unit is 'suspended' over a single pixel in the image,
- Units share the weights in the kernel/mask.

This is called *weight sharing*. It allows rephrasing the training process:

- For two units (at different locations), the gradient processes the *i*th weights in their kernels, in_i' and in_i’’, as separate parameters.
- Weight corrections are calculated independently for w_i' and w_i’’, but accumulate finally in the same parameter w_i shared across the entire layer.

Training convolutional layers

Practical implication: training convolutional layers is not different from training other neural components (layers, subnetworks).

- We can use the rich repertoire of algorithms available for 'generic' neural models.
- We can combine convolutional layers (quite arbitrarily) with other components/layers (e.g., pooling layers, sampling layers, full layers, recursive layers, ...), provided that the 'types' (dimensionality and dimensions of tensors, i.e., their 'shapes') are preserved.

Concluding remarks

Implications

Convolutional filters 'designed' by neural models can be better suited to the characteristics/specificity of a particular task (or more precisely: the learning data representing to the task).

- E.g., a convolutional layer implementing de-noising can adjust to the structure of noise present in a given problem – in contrast to, e.g. traditional Gaussian filter, which lacks adaptivity.

On the other hand, as always in the case of learning from examples, we are in danger of overfitting.

- For fully convolutional ConvNets, this risk is usually less severe, thanks to the relatively small number of parameters and each effective receptive field forming an example.

Pre-training (1)

- In the course of learning, a typical ConvNet develops feature detectors* at successive levels of abstraction (and spatial resolution), associated with successive layers.
- Some of those features may be universal in nature, as certain 'percepts' are universal by virtue of, among others:
  - Perception model: imaging in visible light, color perception in RGB space, etc.
  - Characteristics of scenes: objects are usually compact, have well-defined boundaries (edges, contours, corners, ...)
  - The way the radiation light (or other medium) interacts with objects in the scene.

Therefore, convolutional layers acquired in training on problem (dataset) A can be attempted to be used in problem B.

*Not literally *binary* detectors, of course.

Pre-training (2)

The process of 'transplanting' layers from model A to model B: knowledge transfer.

- From the point of view of domain B, we say that the transferred component has been pre-trained on domain A.
- This is often followed by the component (or entire model it has been embedded in) being fine-tuned on data from domain B (a second training process, which does not always is ‘fine’).
- Deep learning environments (TensorFlow, PyTorch) often offer ready-made pretreated models; see also https://huggingface.co/

Common for ConvnNets, LLMs and more; particularly useful for low-level representations.

- Effective knowledge transfer at higher levels of abstraction is more difficult: high-level features are often more application-specific.

Technical realizations

Currently dominant technologies in the market:

PyTorch

- Facebook's AI Research lab (FAIR).
- License: BSD
- https://pytorch.org/

TensorFlow

- Developed initially by Google Brain for Google's internal use.
- In the public domain since 2015.
- License: Apache
- https://www.tensorflow.org/

Technical realizations

Dominant opinions:

- PyTorch is more research-based, while TensorFlow is more technologically mature on the 'production' side.
- New architectures (proposed in scientific articles and at conferences) are prototyped slightly more frequently and quicker in PyTorch.

However, the balance of advantages and disadvantages depends on the context and application.

- Both environments currently offer very similar capabilities, and require similar efforts in preparing models.
- Attempts to integrate and communicate across platforms: Open Neural Network Exchange (ONNX) https://onnx.ai/

Relation with programming languages

Deep learning coincides with new programming paradigms:

- Differentiable programming
  - The program is treated as a computational graph, with nodes implementing differentiable operations and edges reflecting the flow of data.
  - The graph may be
    - static (created/compiled once), or
    - dynamic, built in the course of calculations, when data comes in (so-called eager mode).
- Probabilistic programming

  Program variables become random variables.

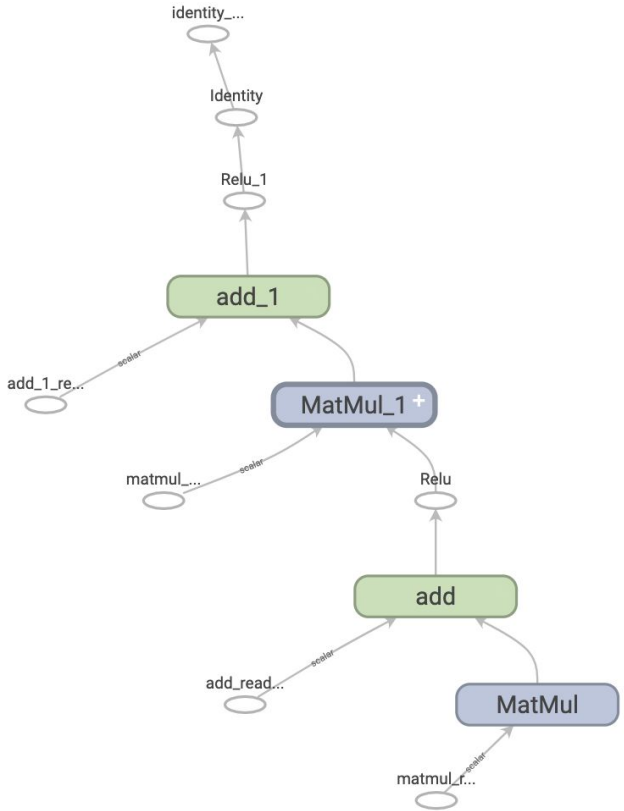
  Known representatives:
  - Church (based on LISP)
  - Figaro (based on Scala)
  - PRISM (based on Prolog)
  - Anglican (based on Clojure)

Example

An example of a computational graph (TensorFlow), involving

- a matrix multiplication,
- adding,
- the ReLU activation function.

Contemporary environments and libraries offer tools for analyzing, visualizing, as well as optimizing such graphs (e.g. Grappler in TensorFlow).

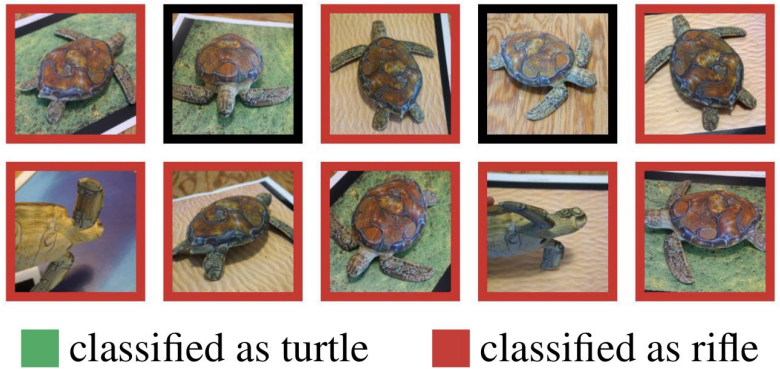


https://www.tensorflow.org/guide/intro_to_graphs

Disadvantages and limitations

- Computational costs for complex models can be unacceptable.
  - Applies not only to training, but sometimes also to querying.
- No guarantee of correctness.

  The Holy Grail of CV: systems that are *provably correct*.
  - Admittedly, this limitation also applies to traditional imaging systems.
  - However, in the case of neural approaches, proving correctness (or, in a more relaxed scenario, estimating the risk of incorrect outcome/response) becomes even more difficult, due to the complexity of these systems.
  - Moreover, neural systems are particularly susceptible to so-called *adversarial examples*).



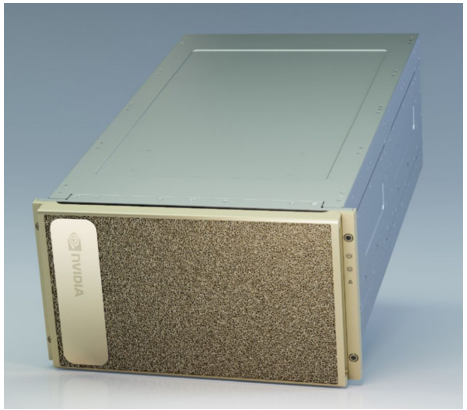
Athalye, A., Engstrom, L., Ilyas, A. & Kwok, K.. (2018). Synthesizing Robust Adversarial Examples. Proceedings of the 35th International Conference on Machine Learning, in Proceedings of Machine Learning Research, 80:284-293.
https://www.csail.mit.edu/news/fooling-neural-networks-w3d-printed-objects

Increasing capabilities of hardware

Hardware manufacturers compete at developing platforms oriented to 'tensor-like' parallel processing.

Architectures are increasingly specialized; examples:

- TensorCores in contemporary GPUs (16b FP precision)
- NVidia's A100 processors: 9.7 TFLOPS
  (used in the DGX A100, among others, pictured right)
- Tensor Processing Unit, TPU (Google): 45 TFLOPS
  (mainly used in Google's cloud solutions).



Increasing capabilities of hardware

The growing capabilities of processors bring with them greater requirements for data, network, and operational and storage infrastructure.

Example: characteristics of the DGX A100 server (right).

These requirements often drive up prices more than the computing units themselves (processors, cores).

SYSTEM SPECIFICATIONS

	NVIDIA DGX A100 640GB	NVIDIA DGX A100 320GB
GPUs	8x NVIDIA A100 80 GB GPUs	8x NVIDIA A100 40 GB GPUs
GPU Memory	640 GB total	320 GB total
Performance	5 petaFLOPS AI 10 petaOPS INT8	
NVIDIA NVSwitches	6	
System Power Usage	6.5 kW max	
CPU	Dual AMD Rome 7742, 128 cores total, 2.25 GHz (base), 3.4 GHz (max boost)	
System Memory	2 TB	1 TB
Networking	8x Single-Port Mellanox ConnectX-6 VPI 200Gb/s HDR InfiniBand 2x Dual-Port Mellanox ConnectX-6 VPI 10/25/50/100/200 Gb/s Ethernet	8x Single-Port Mellanox ConnectX-6 VPI 200Gb/s HDR InfiniBand 1x Dual-Port Mellanox ConnectX-6 VPI 10/25/50/100/200 Gb/s Ethernet
Storage	OS: 2x 1.92 TB M.2 NVME drives Internal Storage: 30 TB (8x 3.84 TB) U.2 NVMe drives	OS: 2x 1.92TB M.2 NVME drives Internal Storage: 15 TB (4x 3.84 TB) U.2 NVMe drives

Leftovers

Some observations about CNNs

- Weight sharing:
  - Features are learned globally
- The dimensions of convolutional layers are not fixed.
  - Convolution is a strictly local operation, it cares only about the part of the input that falls within the dimensions of its receptive field (mask)
  - A CNN composed of only convolutional layers (and other local operations, like max pooling) can process images of arbitrary dimensions.
  - See, e.g., the documentation of Keras.
- In contemporary CNNs, large numbers of parameters originate mostly in:
  - Fully-connected layers
  - Large numbers of filters/channels.

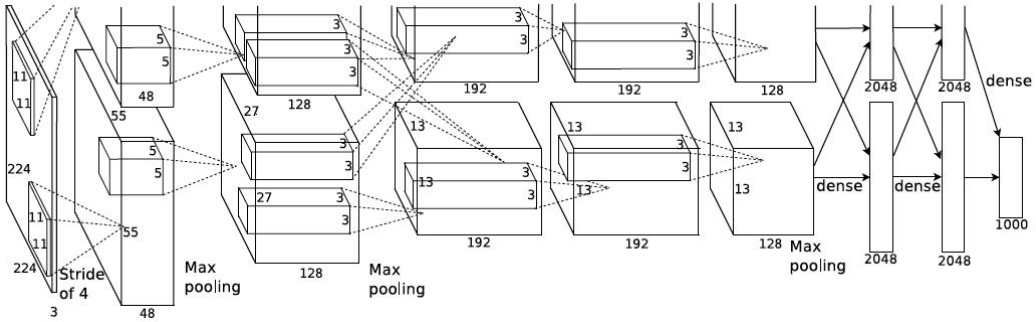
Some observations about CNNs

- CNNs can be used for harvesting ‘universal’ image features.
  - Using pretrained networks is almost always worth considering.
- Super efficient implementation, particularly on GPUs and hardware architectures that target tensor-based computing.
- Can be easily ‘uplifted’ to higher dimensions:
  - 3D
  - 2D+time
  - Supported by most software packages nowadays.
- The concept of convolution can be generalized to other spaces.
  - E.g. graph convolution.

Key facts about CNNs

Essential features. What do CNNs owe their capabilities?

- Multiple filters working in parallel.
- Nonlinear mapping.
- Hierarchical (mostly spatial) organization.
- Helpful: initialization, training algorithms (optimizers), batch normalization, and other developments
- Transfer learning.

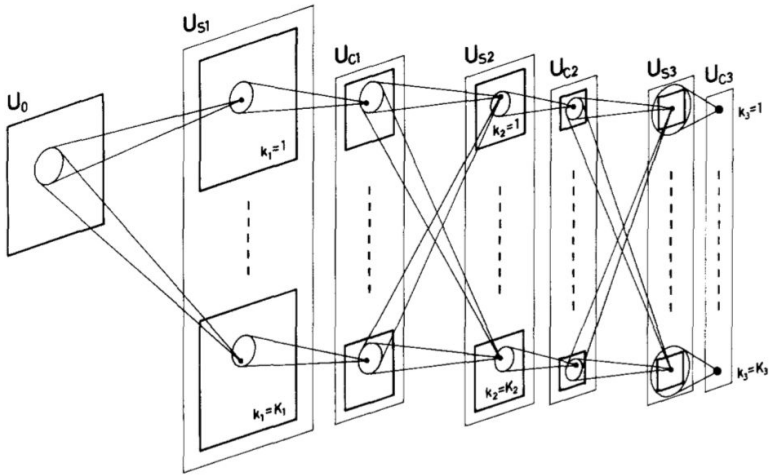


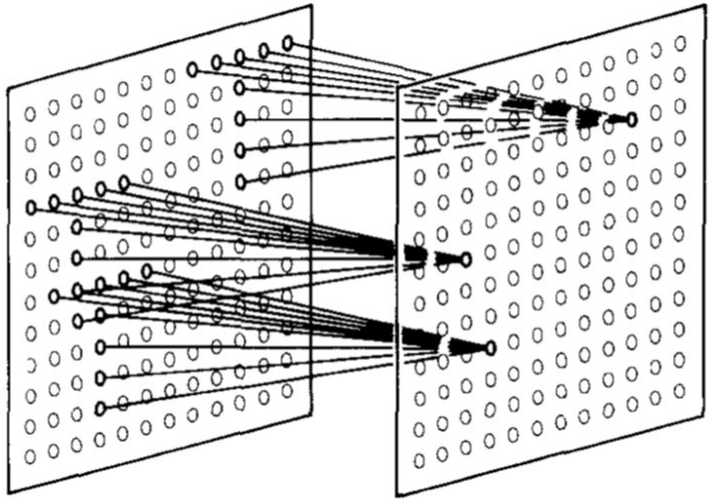
Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks
https://dl.acm.org/doi/10.5555/2999134.2999257

Milestone CNN architectures

Neocognitron

- Neocognitron (Fukushima, 1980)
  - Strongly inspired by the organization of the human/primates visual cortex (interlacing two types of layers, composed of ‘simple’ and ‘complex’ units)
  - Trained mostly in unsupervised manner (only the last layer trained in supervised mode)
  - Empirical assessment on recognition of handwritten digits



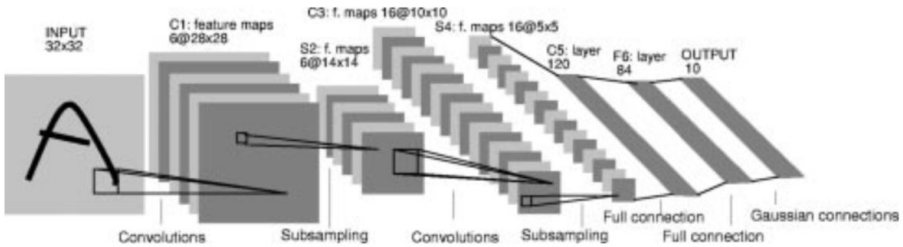


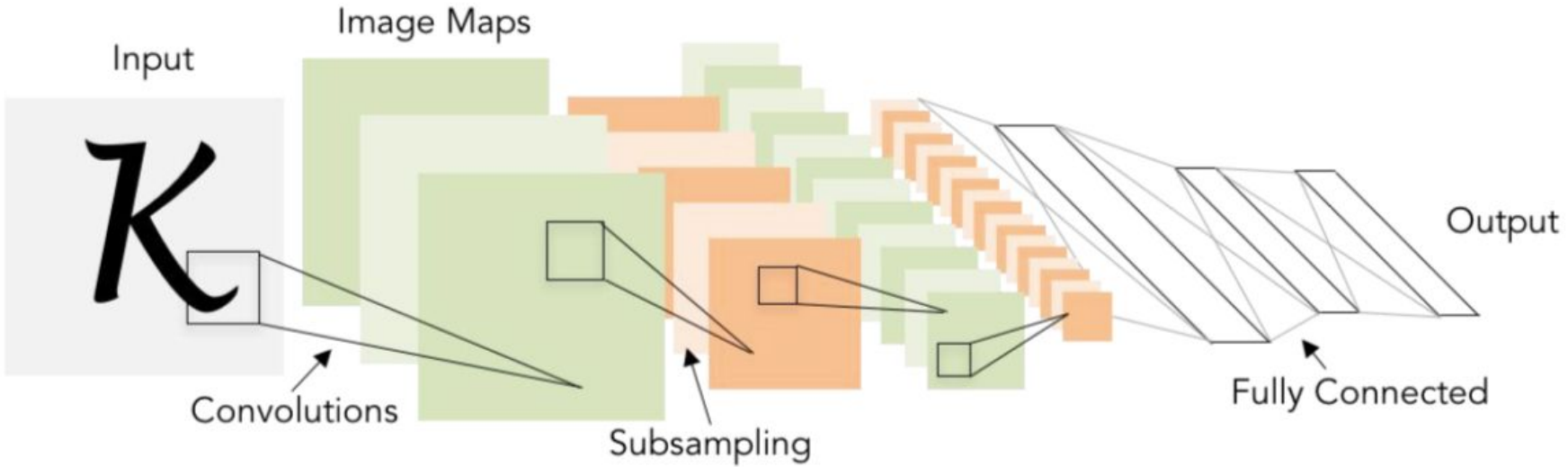
Kunihiko Fukushima, Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position Biological Cybernetics volume 36, 1980.

“LeNets”

LeNets (Lecun et al., 1998), and onward

- A blueprint for contemporary architectures
- Effective character recognition





Yann Lecun, Leon Bottou, Y. Bengio, Patrick Haffner, Gradient-Based Learning Applied to Document Recognition, Proceedings of the IEEE 86(11):2278 - 2324, 1998.

AlexNet

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton

http://arxiv.org/pdf/1409.0575

https://dl.acm.org/doi/10.5555/2999134.2999257

(NIPS 2012) (currently NeurIPS)

Contributions

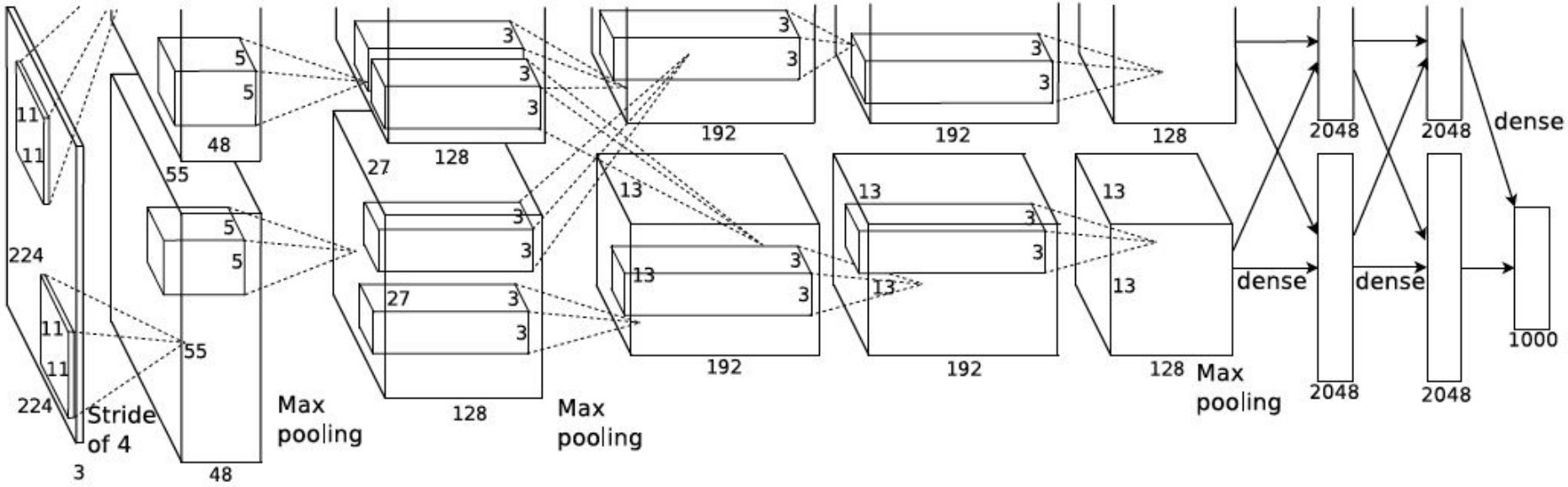
- Large and deep (by 2012 standards)
- Trained on 1.2M images
- Outperformed SoTA methods of the time.
- Second place in the ILSVRC-2012 competition

Architecture: convolutional-dense image classification model

- Five convolutional layers
- Max pooling layers
- Dropout (recently introduced at that time)
- One of the first networks to use ReLU activation functions
- ~60M parameters
- 1000 classes (ImageNet LSVRC contest)

Architecture

Deployed and trained on 2 GPUs (GTX 580, 3GB)



Results

Effectiveness of ReLUs ——————>

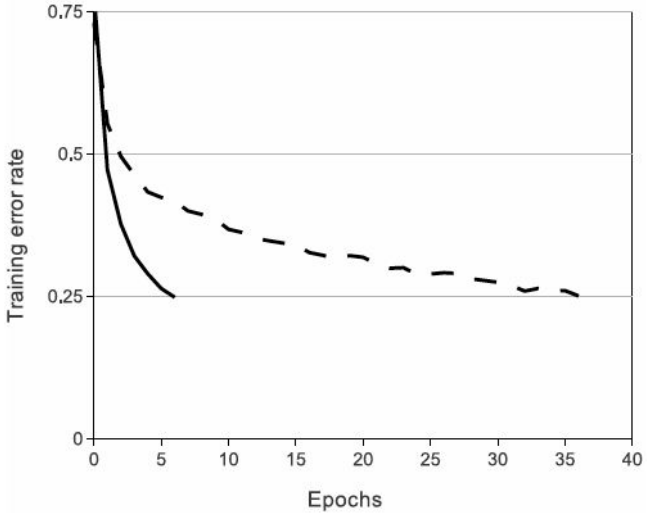


Figure 1: A four-layer convolutional neural network with ReLUs **(solid line)** reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons **(dashed line)**. The learning rates for each network were chosen independently to make training as fast as possible. No regularization of any kind was employed. The magnitude of the effect demonstrated here varies with network architecture, but networks with ReLUs consistently learn several times faster than equivalents with saturating neurons.

Summary of results:

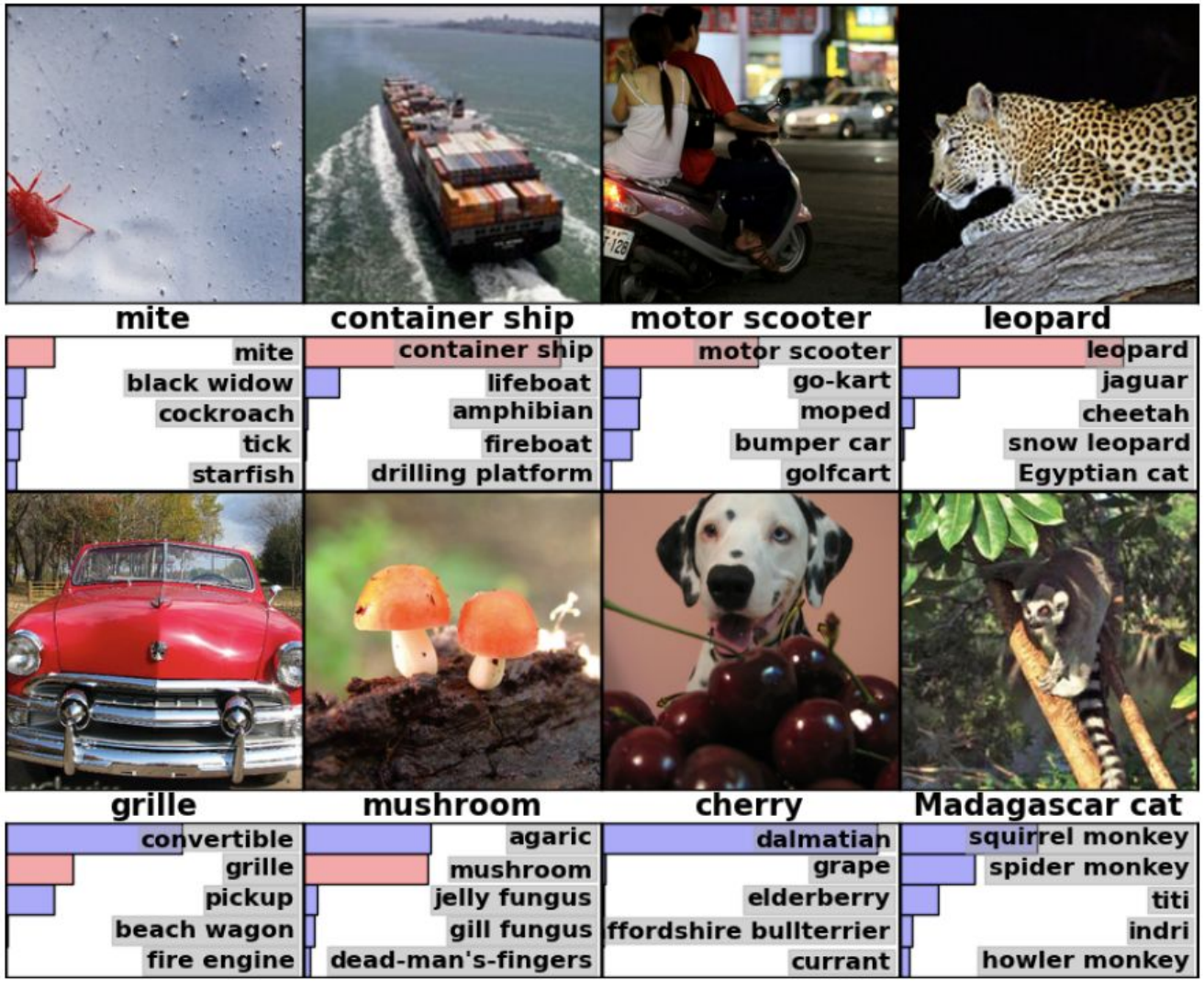
Model	Top-1 (val)	Top-5 (val)	Top-5 (test)
SIFT + FVs [7]	—	—	*26.2%*
1 CNN	40.7%	18.2%	—
5 CNNs	38.1%	16.4%	**16.4%**
1 CNN*	39.0%	16.6%	—
7 CNNs*	36.7%	15.4%	**15.3%**

1 CNN - single CNN

5 CNN - averaging the predictions of 5 CNNs

Another important conclusion: depth is essential.

Top-5 characteristics



VGG models

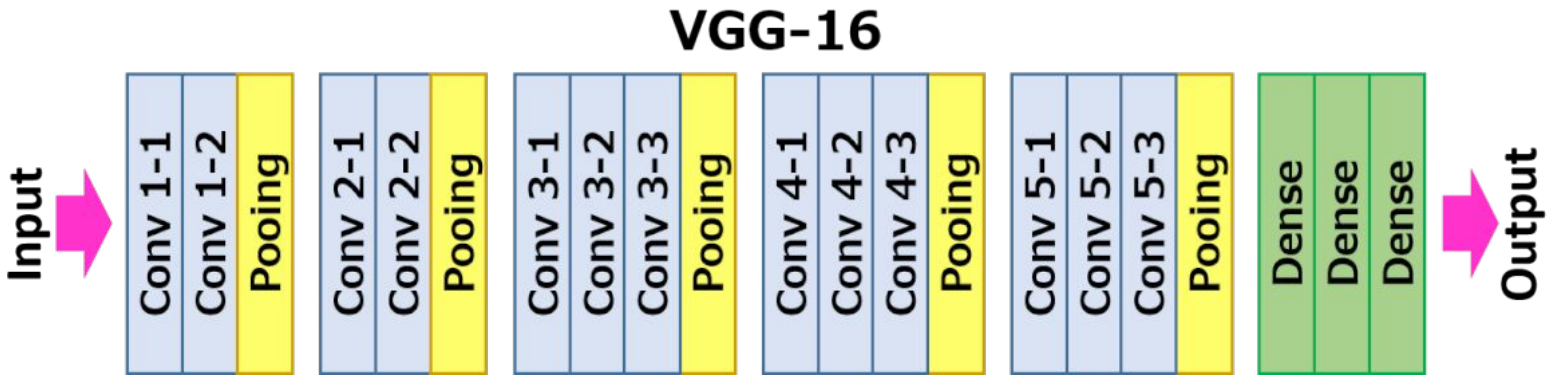
Very Deep Convolutional Networks for Large-Scale Image Recognition
Karen Simonyan, Andrew Zisserman
https://arxiv.org/abs/1409.1556

[The model named after the name of authors' team that won the ILSVRC-2014 contest]

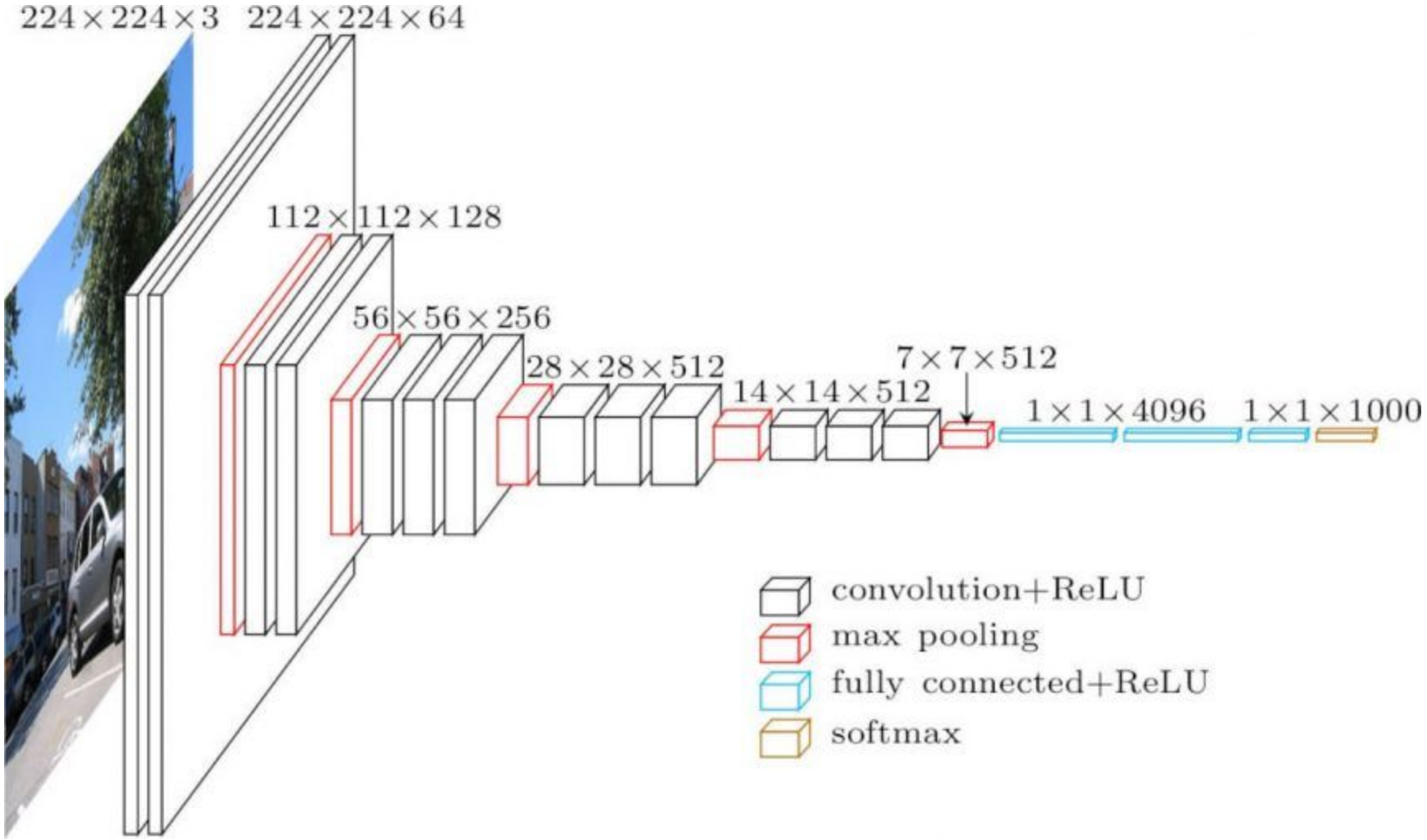
Original contribution

Our main contribution is a thorough evaluation of networks of increasing depth using an architecture with very small (3×3) convolution filters, which shows that a significant improvement on the prior-art configurations can be achieved by pushing the depth to 16–19 weight layers.

Implications: lower number of parameters, more efficient training, less overfitting.



https://neurohive.io/en/popular-networks/vgg16/


224 × 224 × 3
224 × 224 × 64
112 × 112 × 128
56 × 56 × 256
28 × 28 × 512
14 × 14 × 512
7 × 7 × 512
1 × 1 × 4096
1 × 1 × 1000
convolution+ReLU
max pooling
fully connected+ReLU
softmax

Motivation

Compare architectures:

1. One convolutional layer with 5x5 receptive field
2. Two convolutional layers, with 3x3 receptive fields

(assume single-channel layers, for simplicity).

Both have the same *effective* receptive field: 5x5, so the same ‘spatial scope’.
However, in terms of the number of parameters:

1. $5^2+1 = 26$
2. $2(3^2+1) = 20$

The difference becomes only more prominent for greater receptive fields.

Conclusion: Small receptive fields can provide the same ERF at a lower number of parameters. The network becomes deeper, but today we have algorithms that can train deep architectures efficiently.

The Inception Network

Going Deeper with Convolutions

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich

https://arxiv.org/abs/1409.4842

- Further increase of depth, compared to earlier models.
- Modular architecture.
- Multiple output modules.

The origin of the name: the phrase
“we need to go deeper” from the *Inception* movie.

Main features

- Modularity: most of the architecture is made of multiple instances of the same module (Inception module)
  - Same architecture of the module, different parameterization.
- Increased the depth and width of the network while keeping the computational budget constant
  - More precisely: computational cost increases ~proportionally to the depth of the network.
- Architectural decisions based on the Hebbian principle and the intuition of multi-scale processing.
- Specific variant/instance: GoogLeNet, a 22-layer Inception network

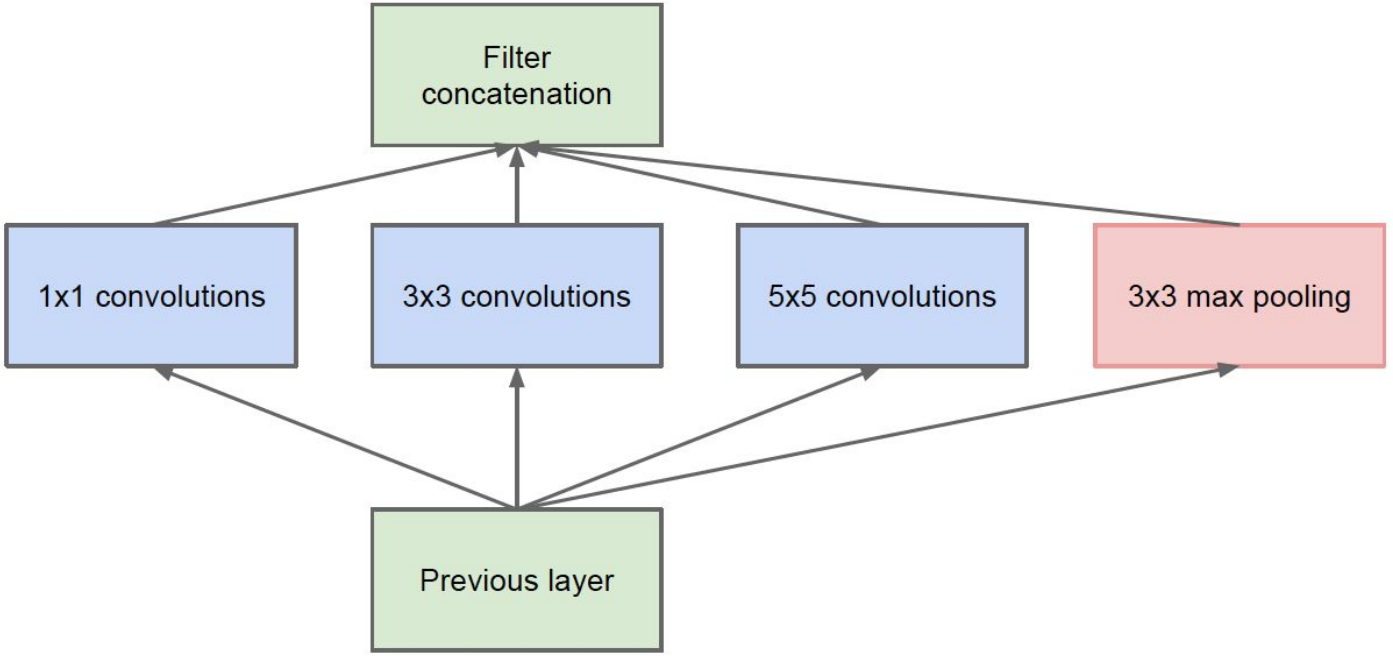
Motivations

- Arora et al. [2]:
  If the probability distribution of the dataset is representable by a large, very sparse deep neural network, then the optimal network topology can be constructed layer after layer by analyzing the correlation statistics of the preceding layer activations and clustering neurons with highly correlated outputs.
  - Related to Hebbian principle – *Neurons that fire together, wire together* [Donald Hebb, 1949]
- If two convolutional layers are chained, an increase in the number of their filters results in a quadratic increase of the number of weights.
  - If the added capacity (weights) is used inefficiently (for example, if most weights end up to be close to zero), then much of the computation is wasted.
  - This could be addressed with sparse data structures, but today's computing architectures are inefficient for such structures.

Motivations

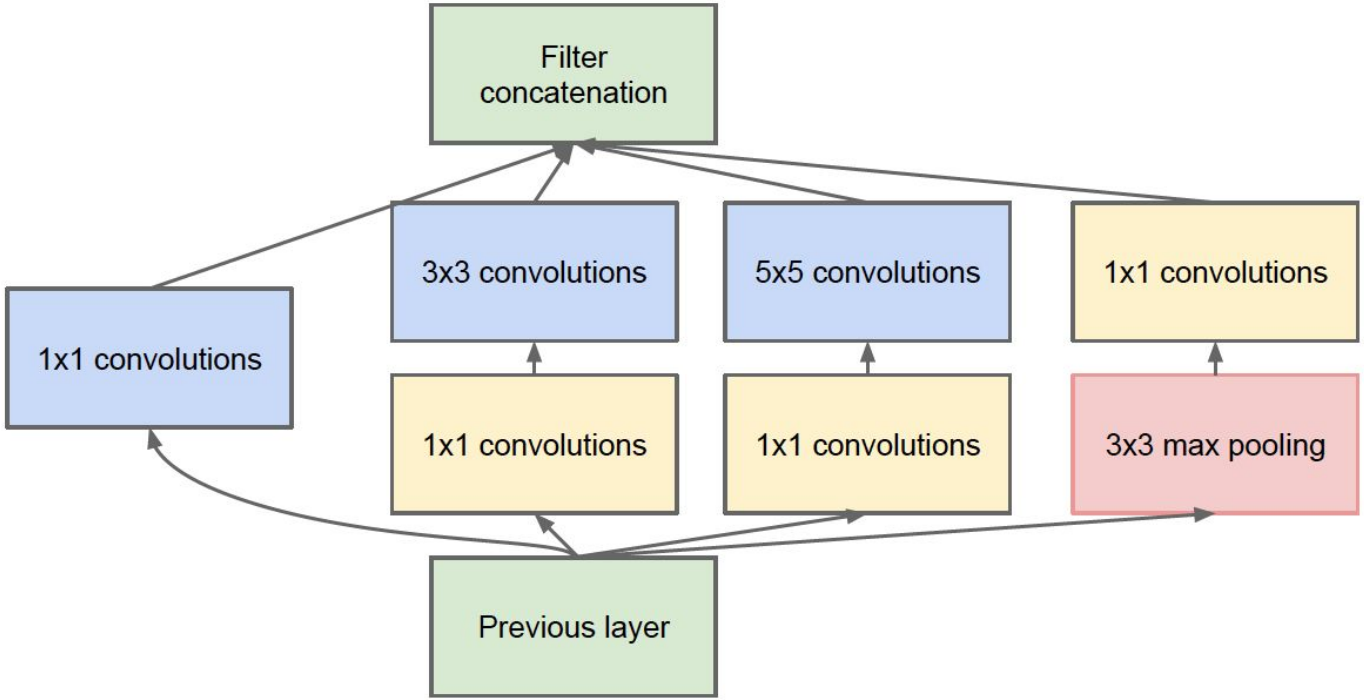
- How an optimal local sparse structure of a convolutional vision network can be approximated and covered by readily available dense components?
  - Assuming translation invariance => use convolutional building blocks.
  - All we need is to find the optimal local construction and to repeat it spatially.
- In the lower layers (the ones close to the input) correlated units would concentrate in local regions.
  - Clusters concentrated in a single region can be covered by a layer of 1x1 convolutions.
- Smaller number of more spatially spread out clusters that can be covered by convolutions over larger patches
  - Hence also 3x3 and 5x5 convolutions.
- All those layers concatenated into a ‘filter bank’
- Additional pooling operations performed in parallel to the above.

Inception module: naive version



- The ratio of 3x3 and 5x5 convolutions to 1x1 should increase with consecutive layers, as features become sparser and more spatially distributed.
- The challenge: large depth of concatenated filters.

Inception module



- 1x1 convolutions embed the stimuli in a lower-dimensional space.
  - Added before convolutions, to preserve spatial sparseness and resolution.
- Implement also ReLU activation functions.

Complete Inception architecture

- A network consisting of modules of the above type, stacked upon each other.
- Occasional max-pooling layers with stride 2 to halve the spatial resolution.
- For memory efficiency: start using Inception modules only at higher layers while keeping the lower layers in traditional convolutional fashion.

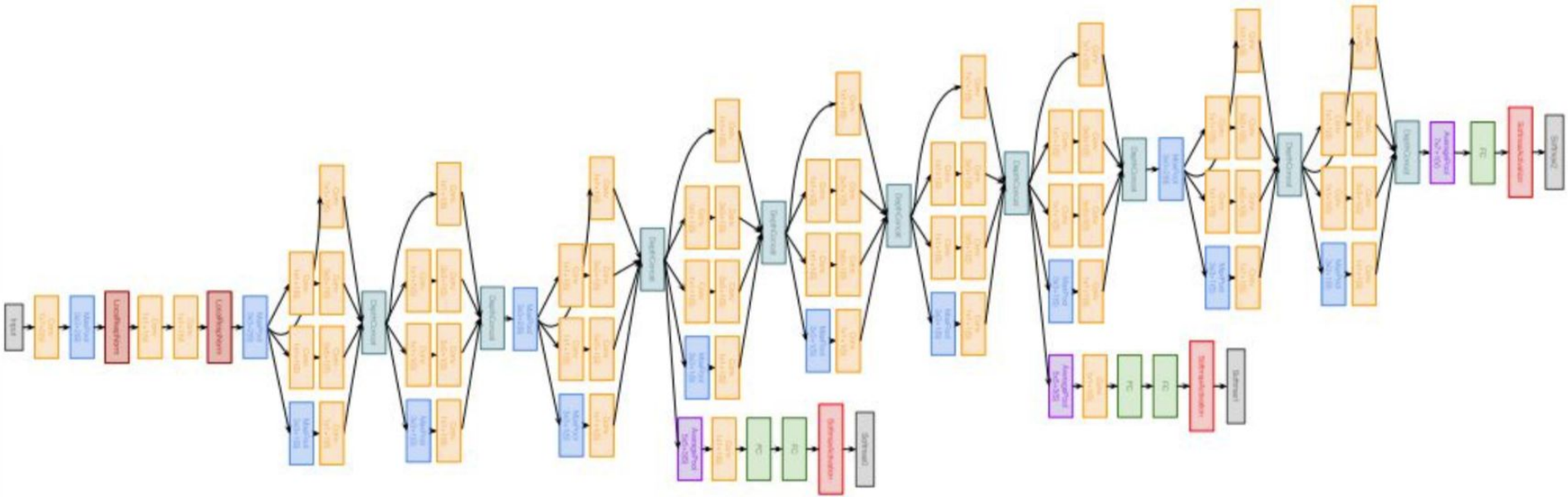
GoogLeNet

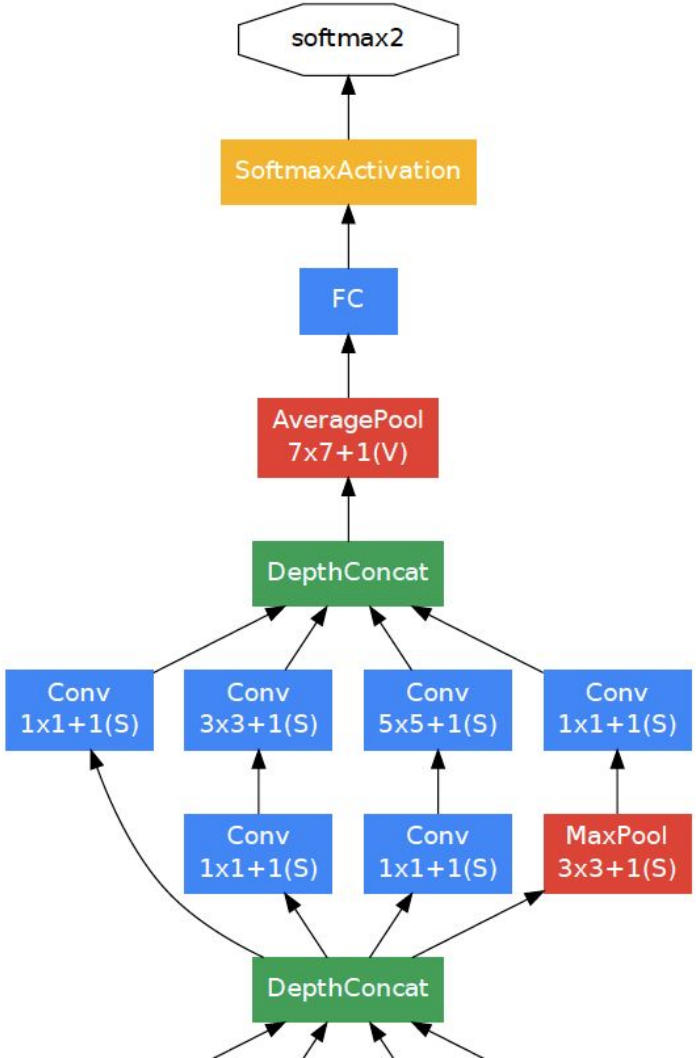
- 22 layers deep (if counting only layers with parameters)
  - 27 layers if also counting pooling
- Uses average pooling before the classifier (with additional linear layer)

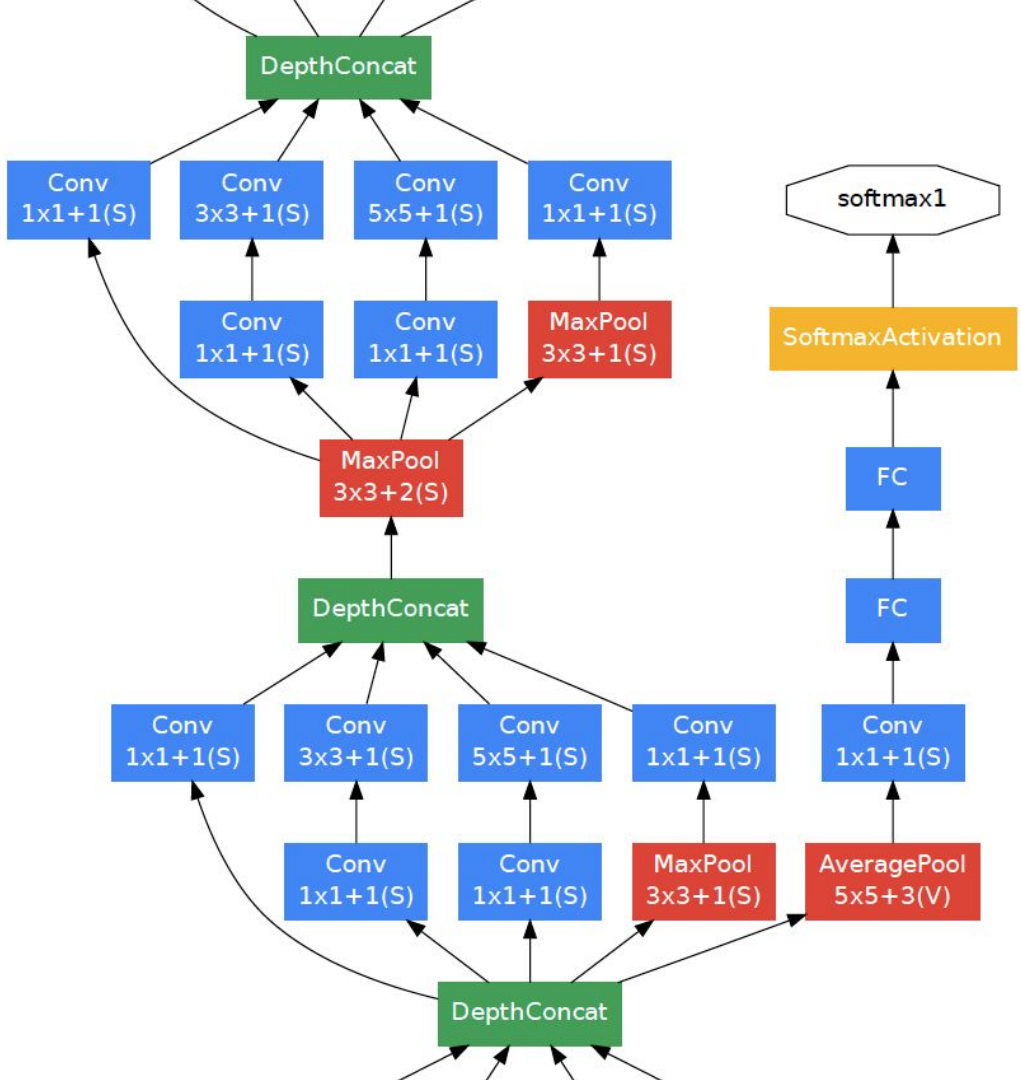
Addressing the vanishing gradient problem:

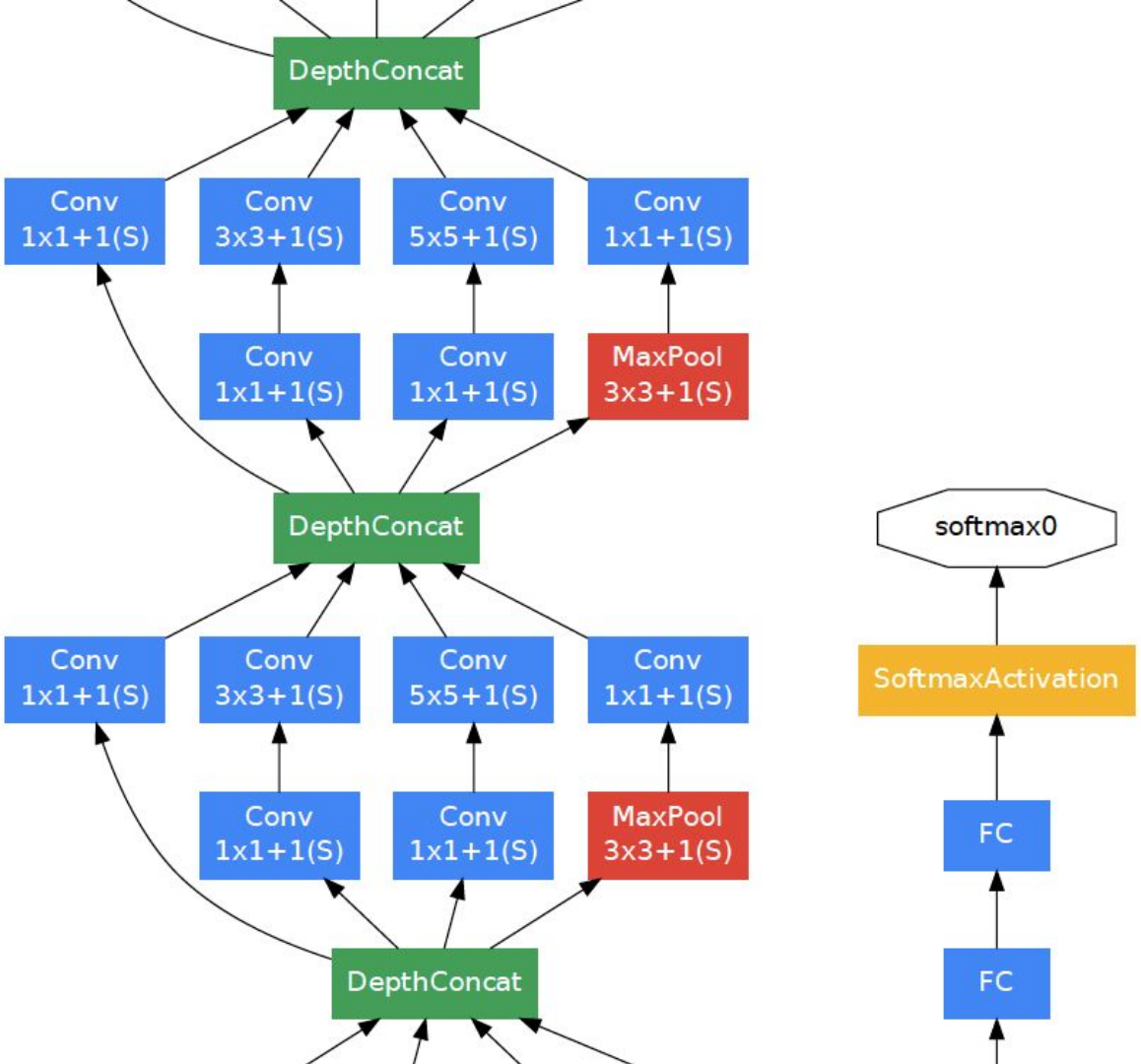
- *The strong performance of shallower networks on this task suggests that the features produced by the layers in the middle of the network should be very discriminative*.
- By adding (to the ‘stem network’) auxiliary classifiers connected to these intermediate layers, discrimination in the lower stages in the classifier was expected.
  - Additional source of training signal (gradient).

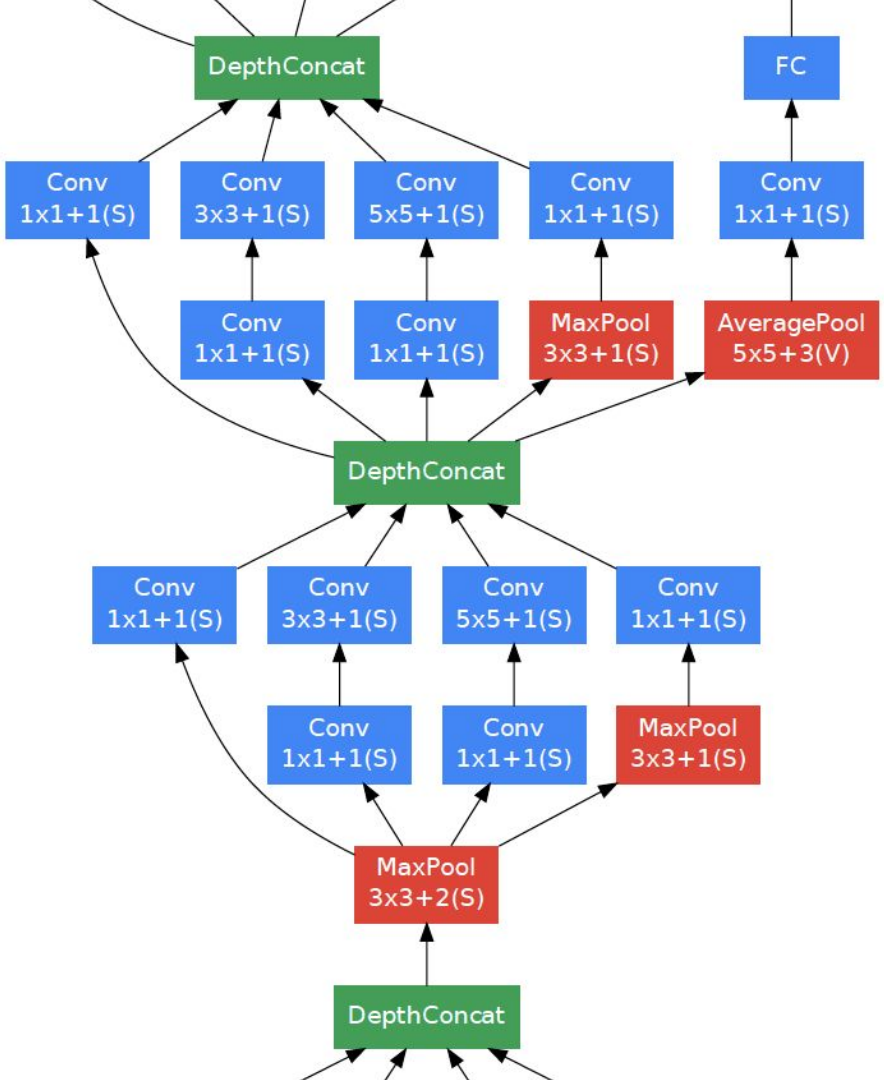
GoogLeNet: Bird’s-eye view

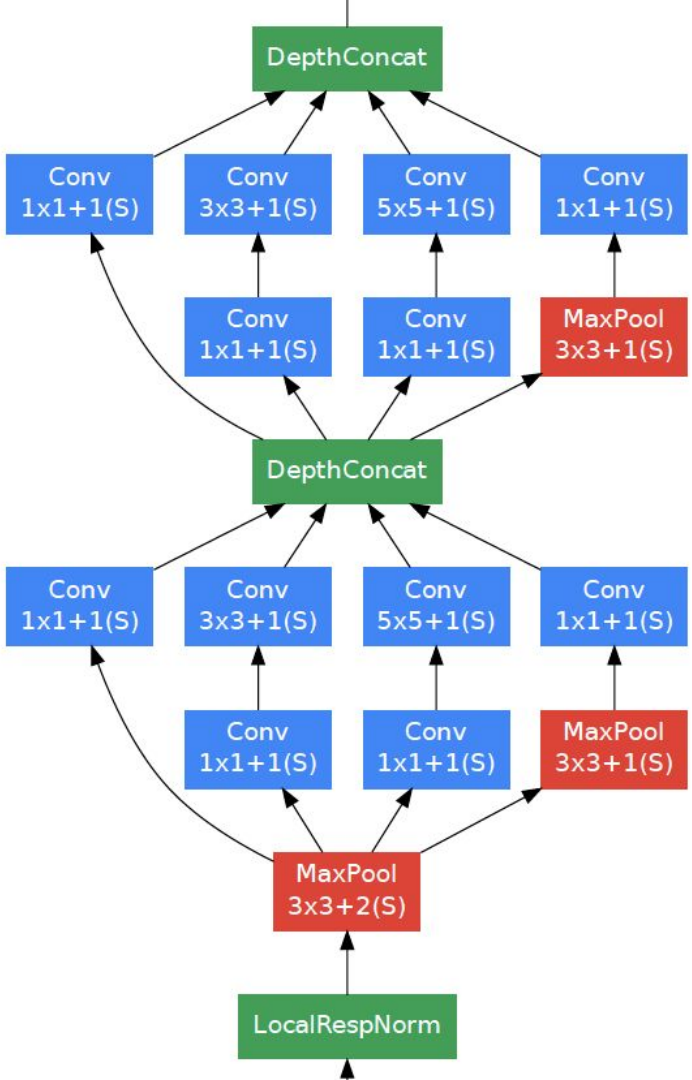


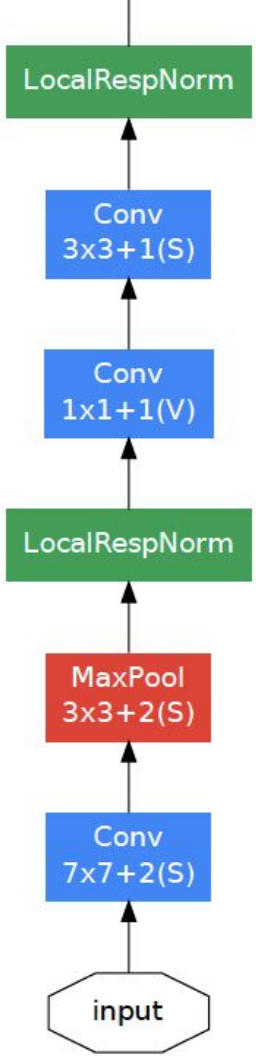

softmax2
SoftmaxActivation
FC
AveragePool
7x7+1(V)
DepthConcat
Conv
1x1+1(S)
Conv
3x3+1(S)
Conv
5x5+1(S)
Conv
1x1+1(S)
Conv
1x1+1(S)
Conv
1x1+1(S)
MaxPool
3x3+1(S)
DepthConcat


DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
MaxPool 3x3+2(S)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
AveragePool 5x5+3(V)
DepthConcat
softmax1
SoftmaxActivation
FC
FC


DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
DepthConcat
softmax0
SoftmaxActivation
FC
FC


DepthConcat
FC
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
AveragePool 5x5+3(V)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
MaxPool 3x3+2(S)
DepthConcat


DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
MaxPool 3x3+2(S)
LocalRespNorm


LocalRespNorm
Conv 3x3+1(S)
Conv 1x1+1(V)
LocalRespNorm
MaxPool 3x3+2(S)
Conv 7x7+2(S)
input

Question

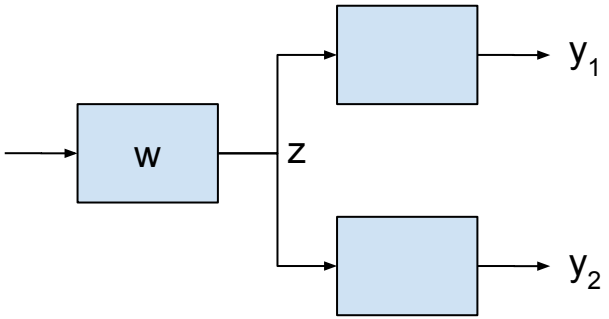
What happens to gradient at the branching points?

Assuming that the total loss is a sum of losses at the ends of branches, where the model returns values y_1 and y_2:

$$L = L_1(y_1) + L_2(y_2)$$

The total derivative of loss function w.r.t. the vector w of weights/parameters of the common part, where z is the output of the common part:

$$\frac{\partial L}{\partial w} = \frac{\partial z}{\partial w}\frac{\partial y_1}{\partial z}\frac{\partial L_1}{\partial y_1} + \frac{\partial z}{\partial w}\frac{\partial y_2}{\partial z}\frac{\partial L_2}{\partial y_2}$$



type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

GoogLeNet

- Trained using asynchronous gradient descent with momentum.
- Quite sophisticated image cropping approach for harvesting the training data.
- Uses an ensemble of 7 independently trained GoogLeNets:
  - Trained with the same initialization, even with the same initial weights, due to an oversight (!)
  - They differed only in sampling applied to input data (involved random shuffling of examples).
  - Same dynamic learning rate policies.
  - The ensemble included one wider version.
- Softmax probabilities are averaged to obtain the final prediction:
  - over multiple crops, and
  - over all the individual classifiers/models.

Residual networks

Deep Residual Learning for Image Recognition

Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun

https://arxiv.org/abs/1512.03385

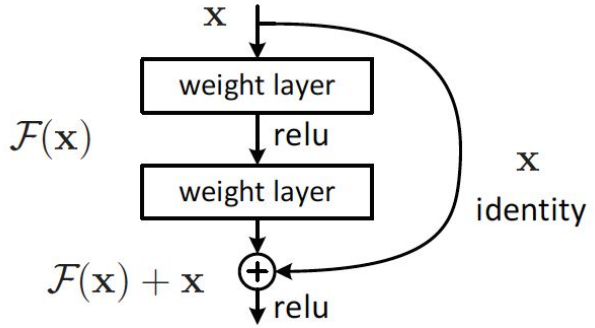
- First well-known study introducing the concept of residual connections.

Residual networks

- Reformulate the training task for a layer as learning a residual function with reference to the layer's inputs, instead of learning the original function.
- The authors use residual nets with a depth of up to 152 layers
  - 8 layers deeper than VGG nets.
- 1st place in the ILSVRC 2015 classification contest.
- 1st places in the following contests:
  - ImageNet detection, ImageNet localization, COCO detection, COCO segmentation.

https://www.image-net.org/ https://cocodataset.org/

Motivations

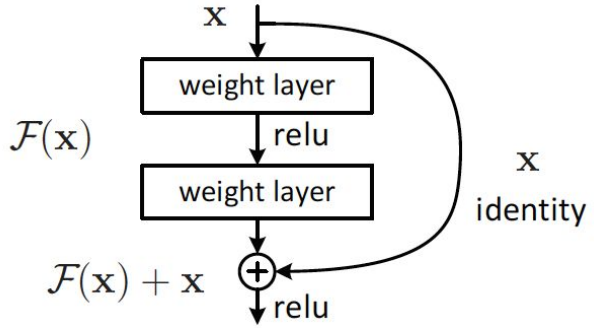


- Let
  - H(x) be the mapping to be fit by a subnetwork/submodel (a few stacked layers, not necessarily the entire net), with x the input to that submodel.
  - Assume that x and H(x) have the same dimensions (‘shapes’).
- If multiple nonlinear layers can asymptotically approximate H(x), then it is equivalent to hypothesize that they can asymptotically approximate the residual function, i.e., H(x) - x
  - Let’s **explicitly let these layers approximate a residual function** F(x) := H(x)-x.
- In other words: after successful training, the subnetwork will implement the function F(x)+x.
- Although both submodels (direct and residual) should be able to asymptotically approximate the desired function (as hypothesized),
  - the difficulty of learning might be different (due to ‘shortcuts’ in gradient flow).
  - This is particularly true when chaining residual modules.

Basic Resnet block

Mapping realized by a Resnet building block:

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



where $\{W_i\}$ are the parameters of F.

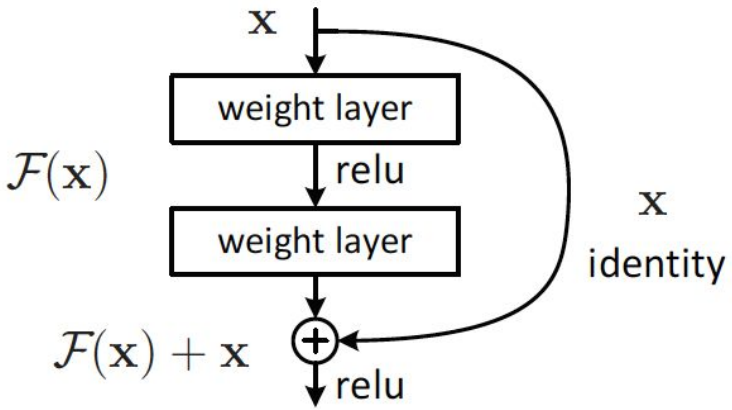
Q: What if, for some reason, the shapes of F and x are different?
A: We can still realize a 'pseudo-shortcut': a linear mapping that uses a (trainable) matrix of parameters W:

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

Also: Notice the presence of ReLU activation *after* the summation.

Basic building block

The specific building block used by the authors:



Similarly to GoogLeNet, Resnets are thus *modular networks.*

- However, this is ‘just’ structural modularity:
  - Each instance of a Resnet block has the same architecture, but has separate weights and thus implements a different function (there’s no weight sharing between individual building blocks/modules).

Overall architecture

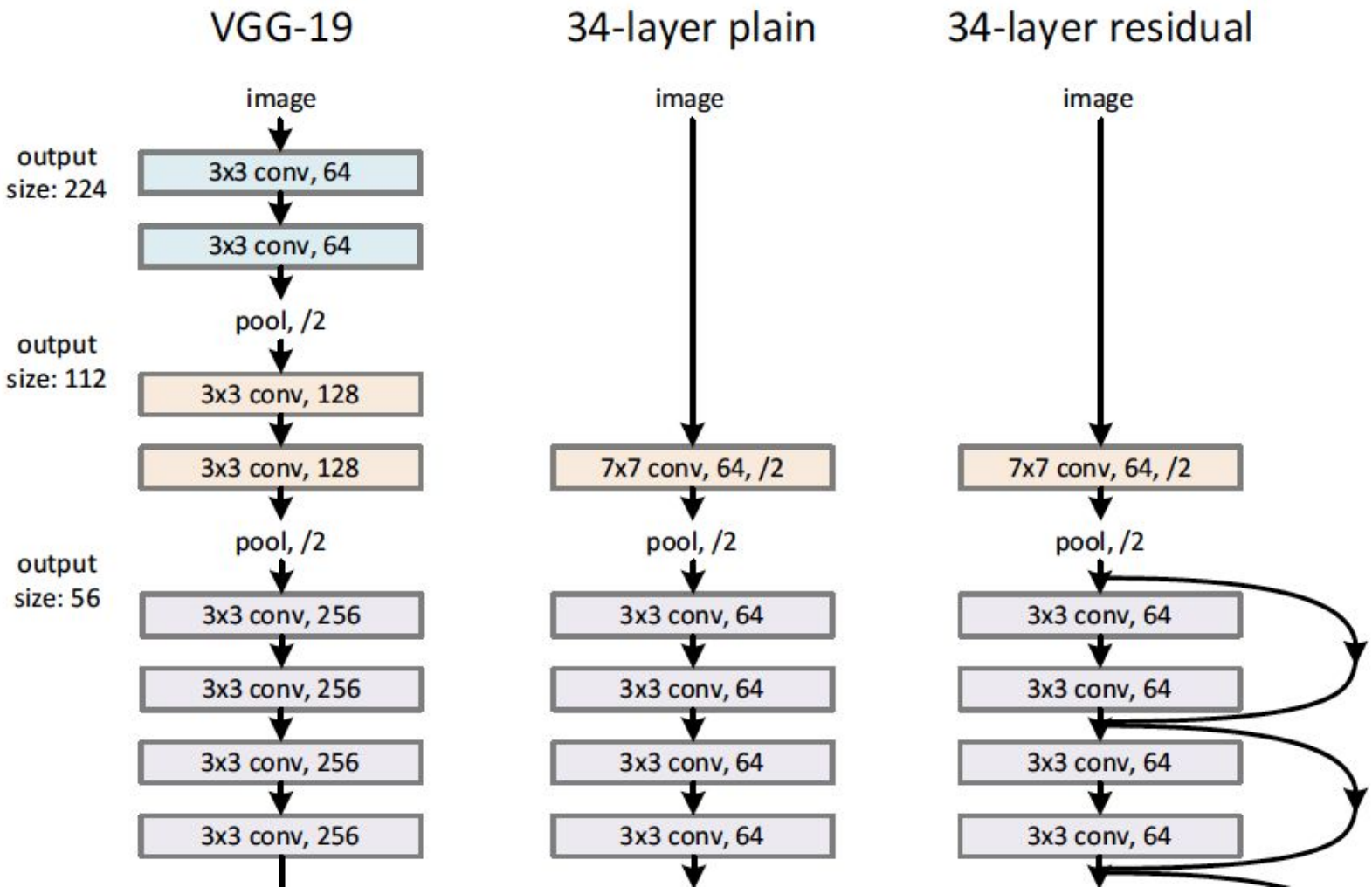
Design closely resembles the VGG networks:

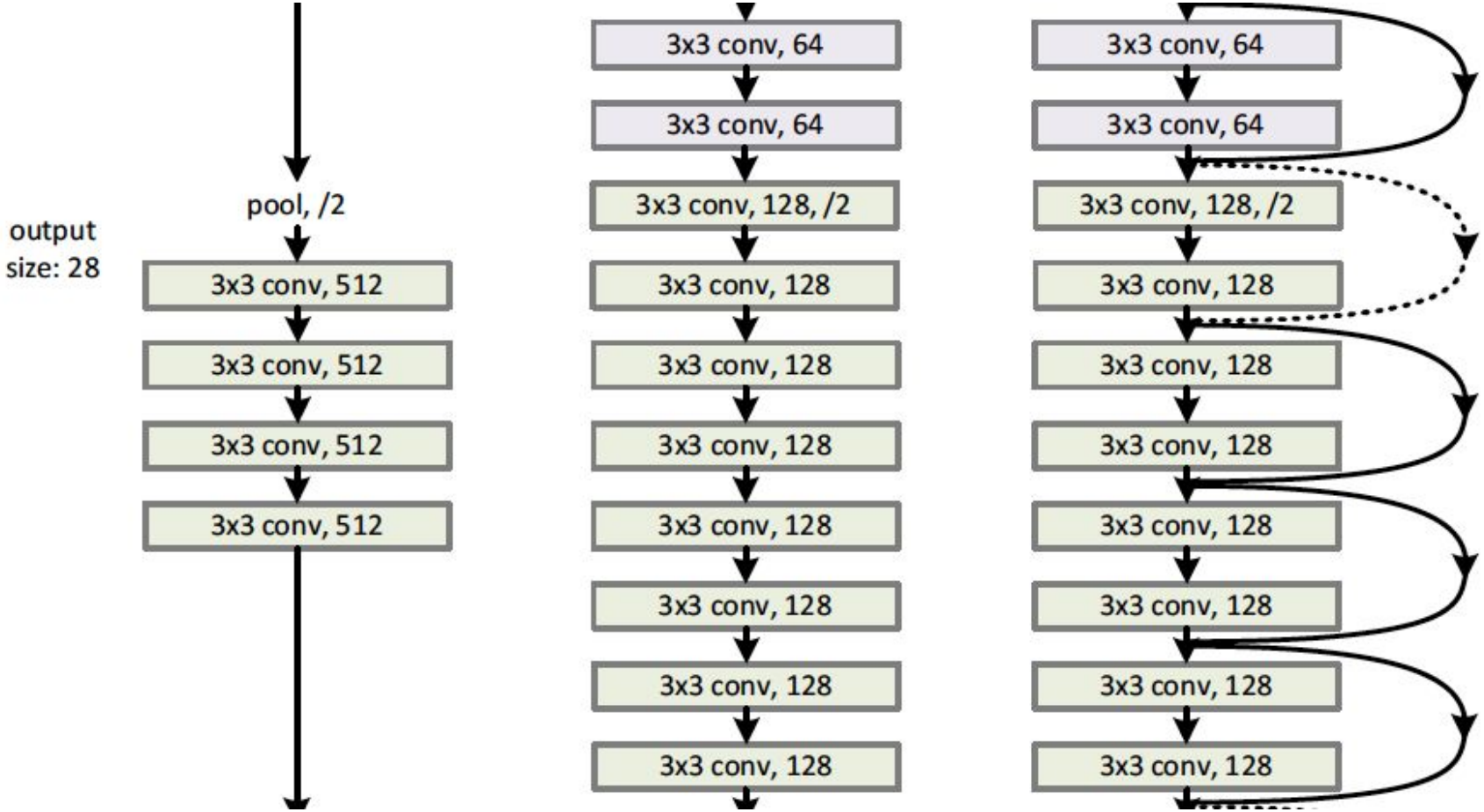
- Mostly 3x3 filters
- Downsampling
  - Using convolution with stride 2 (“/2”).
  - When the feature map size is halved, the number of filters is doubled so as to preserve the time complexity per layer.

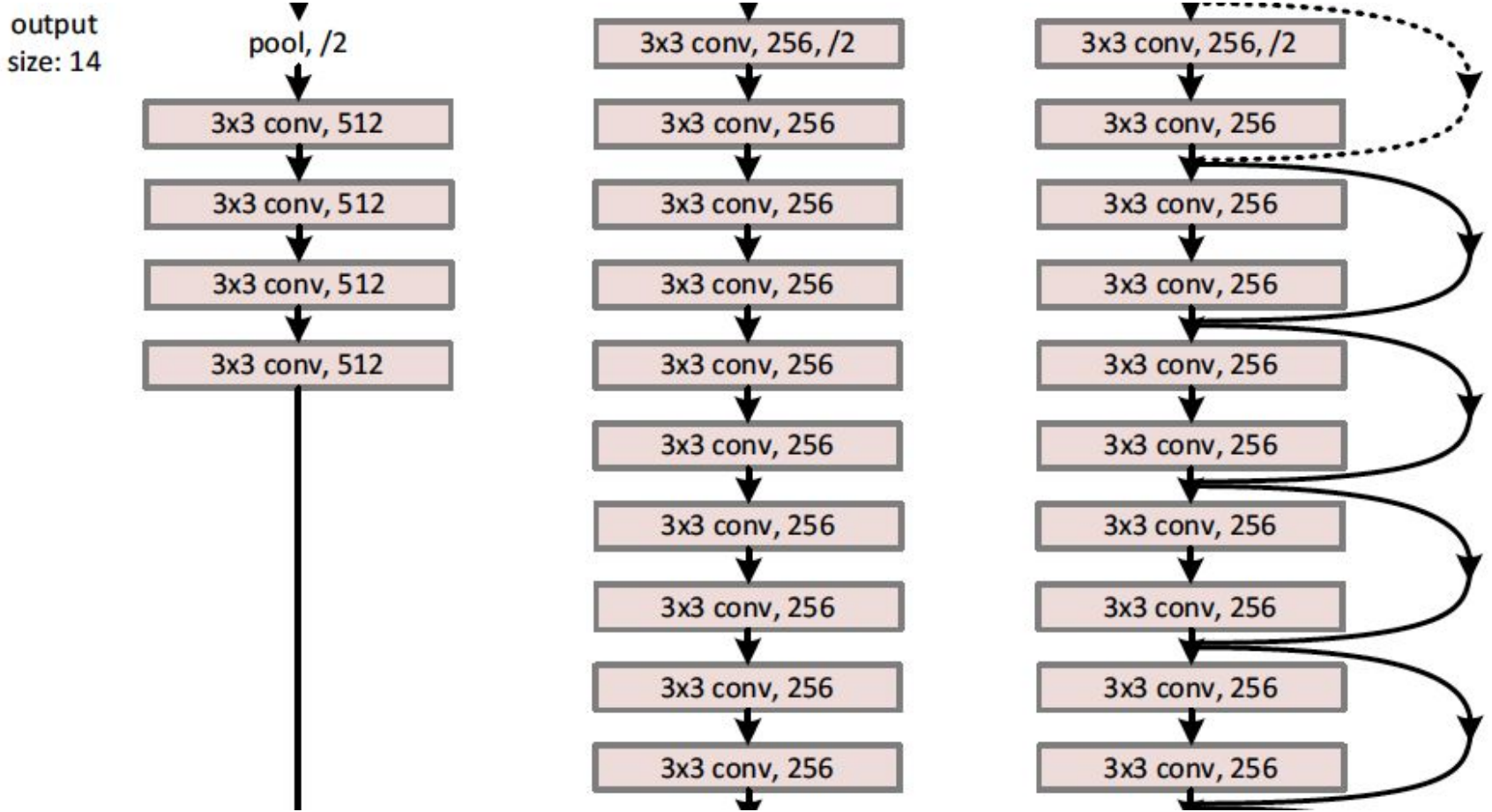
The figure that follows in next slides:

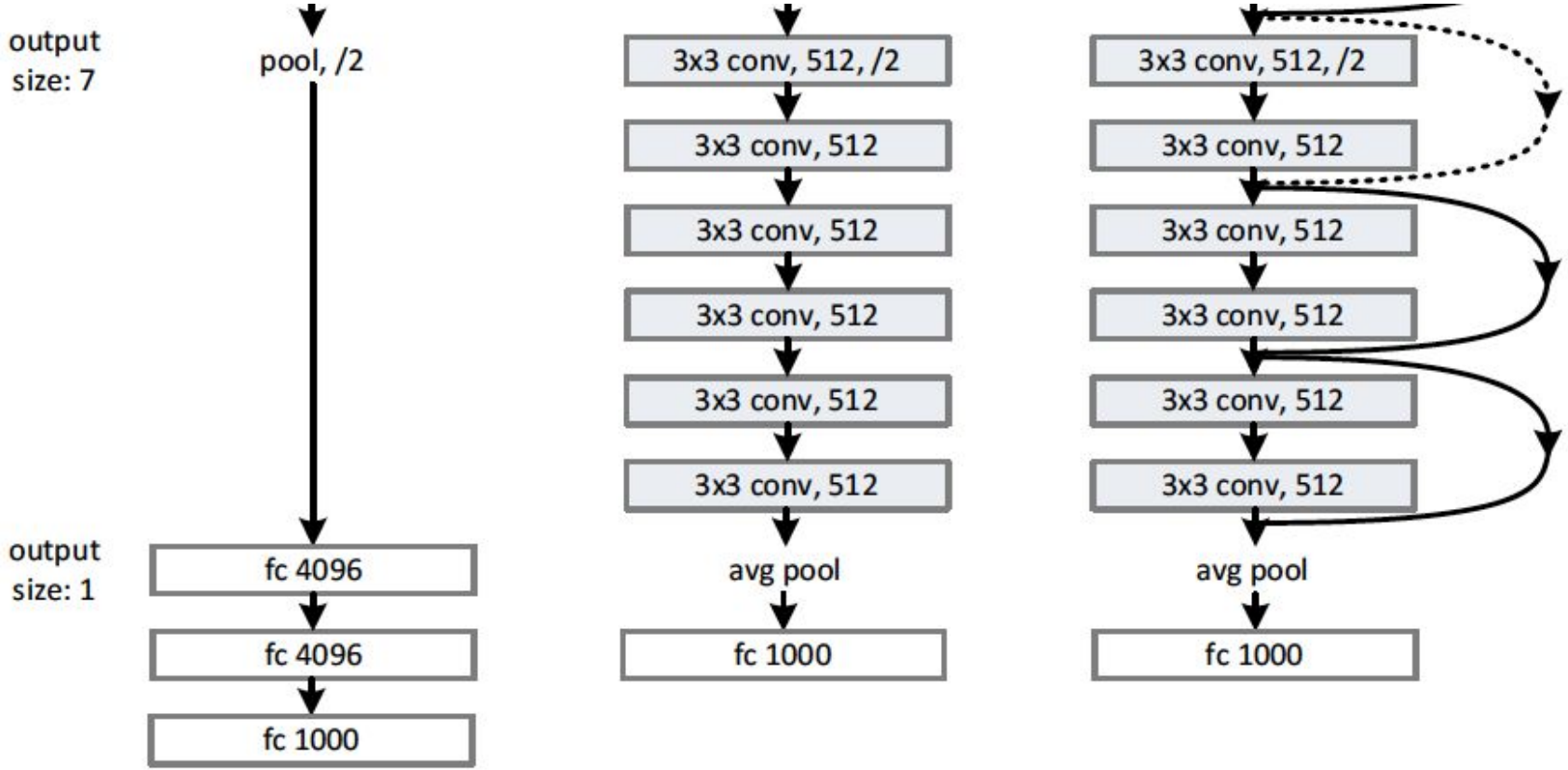
- Left: VGG-19 model: 19.6 billion FLOPs
- Center: A plain network with 34 parameter layers: 3.6 billion FLOPs
- Right: A residual network with 34 parameter layers: 3.6 billion FLOPs

The dotted shortcuts also double the number of channels (description follows).


VGG-19
34-layer plain
34-layer residual
image
output size: 224
3x3 conv, 64
3x3 conv, 64
pool, /2
output size: 112
3x3 conv, 128
3x3 conv, 128
pool, /2
output size: 56
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64


output
size: 28
pool, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 64
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 64
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128


output
size: 14
pool, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256


output
size: 7
pool, /2
output
size: 1
fc 4096
fc 4096
fc 1000
3x3 conv, 512, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
avg pool
fc 1000
3x3 conv, 512, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
avg pool
fc 1000

Other elements

Batch normalization after each convolution (before activation)

Two options for the dotted connections:

- A: Parameter-free: The shortcut still performs identity mapping, with extra zero entries padded for increasing dimensions (no extra parameters)
- B: Parametric: Each output channel is a linear combination of input channels. The projection shortcut is used to match dimensions, using 1x1 convolutions:

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

Considered configurations

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

1 FLOP = 1 multiply+add operation (elementary component of the dot product)

Some results

(A) zero-padding shortcuts are used for increasing dimensions, and all shortcuts are parameter free
(B) projection shortcuts are used for increasing dimensions, and other shortcuts are identity;
(C) all shortcuts are projections

Trained with ordinary SGD with momentum!

Experimented also with ensambles; networks up to 1000 layers (found it to be worse than 110-layer net)

model	top-1 err.	top-5 err.
VGG-16 [41]	28.07	9.33
GoogLeNet [44]	-	9.15
PReLU-net [13]	24.27	7.38
plain-34	28.54	10.02
ResNet-34 A	25.03	7.76
ResNet-34 B	24.52	7.46
ResNet-34 C	24.19	7.40
ResNet-50	22.85	6.71
ResNet-101	21.75	6.05
ResNet-152	**21.43**	**5.71**

Table 3. Error rates (%, **10-crop** testing) on ImageNet validation. VGG-16 is based on our test. ResNet-50/101/152 are of option B that only uses projections for increasing dimensions.

Why do ResNets work?

- They address the vanishing/exploding gradient problem.
  - Each model component with partial derivatives close to zero increases the risk of vanishing gradient.
  - Squeezing activation functions are particularly notorious in that respect (gradient close to zero almost everywhere).
- This implicitly allows maintaining relatively low depth (number of channels/dimensions) along the network
  - Projections used only thrice in the architecture (cf. figure).
- Does modularity help?
  - An open question.
  - Definitely eases automation of architecture optimization: some design choices cease to be available (part of architectural hiperparameters are fixed).

Related: Wide Residual Networks (Wide-Nets) http://arxiv.org/abs/1605.07146: emphasize Resnet width rather than depth. See the module Auxiliary topics.

Highway networks

Highway networks

Rupesh Kumar Srivastava, Klaus Greff, Jürgen Schmidhuber

https://arxiv.org/abs/1505.00387

- Generalizes the residual connections.

Highway networks

- Generalization of residual connections
- Shortcut connections with gating functions (rather than identities): *transform gate* T and *carry gate* C:

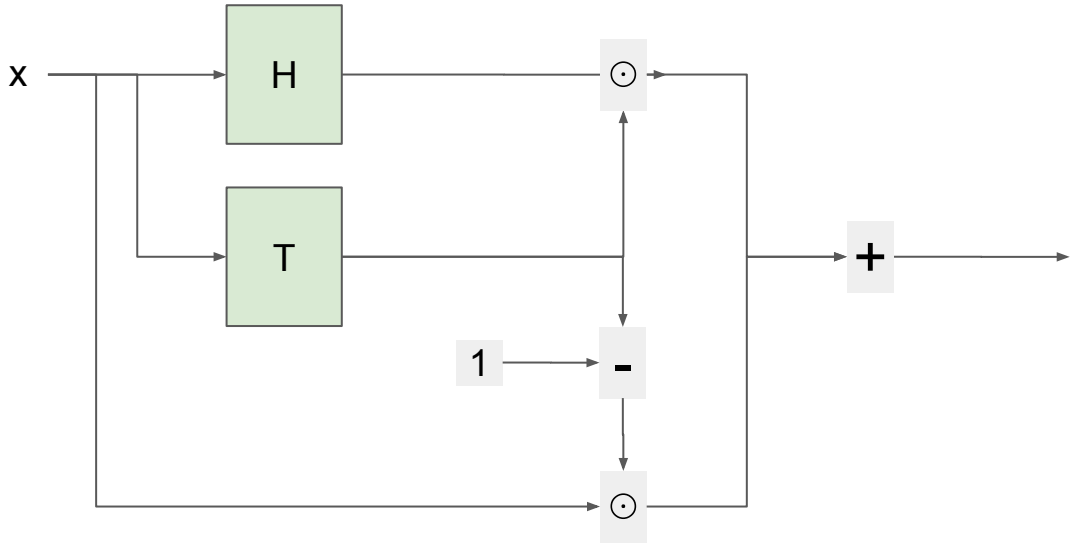
$$\mathbf{y} = H\left(\mathbf{x}, \mathbf{W_H}\right) \cdot T\left(\mathbf{x}, \mathbf{W_T}\right) + \mathbf{x} \cdot C\left(\mathbf{x}, \mathbf{W_C}\right)$$

- The original paper: T is a dense sigmoid layer (so T is in [0,1]) and:

$$\mathbf{y} = H\left(\mathbf{x}, \mathbf{W_H}\right) \cdot T\left(\mathbf{x}, \mathbf{W_T}\right) + \mathbf{x} \cdot \left(1 - T\left(\mathbf{x}, \mathbf{W_T}\right)\right)$$

  (see the diagram on the next slide)
- T's bias initialized so as to implement the carry behavior.
  - That is, the biases of those units is set to large negative values, so that Ts are close to 0.
- Allow unimpeded information flow across several layers on "information highways"

Highway module: conceptual diagram



- T serves as a gate that controls the ‘proportions’ of the carry behavior (x) and processing (H)
- Gating is nowadays a common mechanism in deep learning, in particular in recurrent architectures and attention mechanisms.

Other interesting architectures related to Resnets

- Fractal networks, FractalNet [Larsson et al. 2017]:
  - parallel multi-scale processing,
  - trained by dropping out entire modules/paths
- Deep networks with stochastic depth [Huang et al. 2016]:
  - operate a bit like dropout, but on the level of entire layers.
- DenseNets: see next slides.

DenseNet

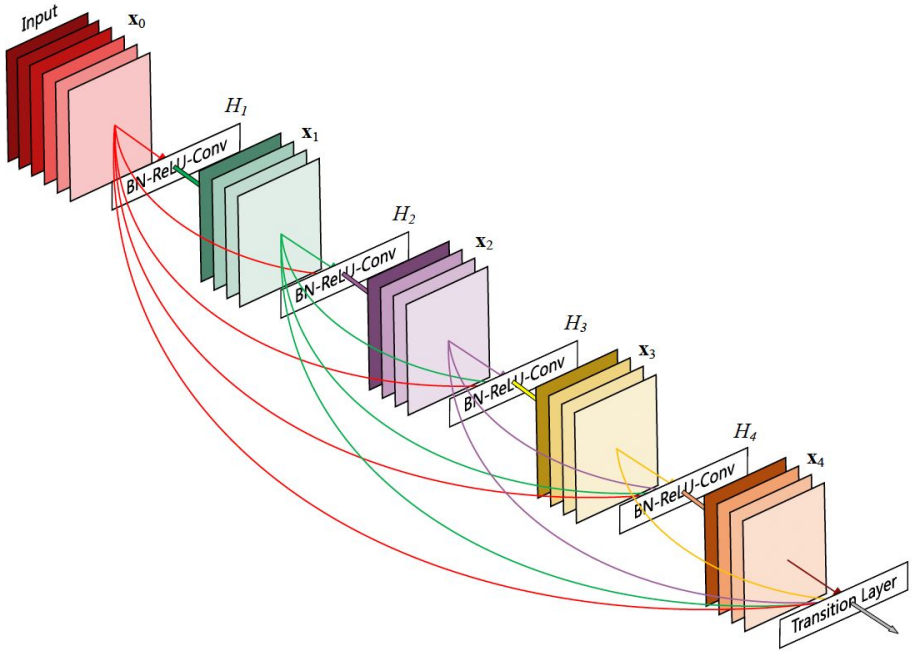
Densely Connected Convolutional Networks

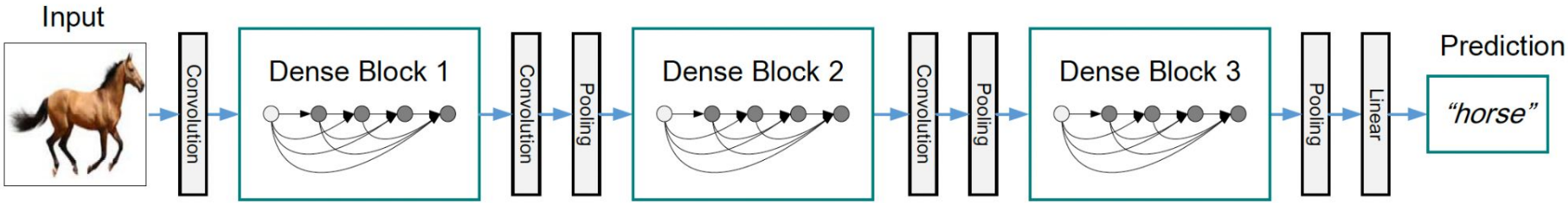
Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger

https://arxiv.org/abs/1608.06993

DenseNet

- Direct connections from each layer to all subsequent layers.
- Applied within dense blocks, which maintain the same spatial dimensions.
- Dense blocks interlaced with transition layers that perform convolution and pooling.





Fully convolutional architectures

Fully convolutional networks (ConvNets, FCN)

Models that comprise exclusively local operations, i.e. mostly:

- Convolutions and transposed convolutions
- Pooling
- Re-sampling (down- and upsampling, interpolation).

A fully convolutional architecture F:

- Can handle images of arbitrary size.
- Is covariant with respect to translation
  - F and translation *commute*.

Motivations

Fully convolutional architectures are applicable in any usage scenario that requires image-to-image mapping, i.e. any *image processing*.

Prominent examples:

- Segmentation
  - Including semantic segmentation
- Denoising
  - The network serves as a denoising filter.
  - Can adapt in training to the characteristics (distributions) of a given class of images/domain.
- Superresolution
- Transfer/translation of local characteristics, e.g. style transfer
  - Also: pseudocoloring, texture filling, and more
  - Semantic transformations, e.g. synthesizing an aerial image based on a map.
    - See, e.g., so-called Pix2Pix models

Applying FCNs to image processing tasks

Advantages:

- Each instance of effective receptive field is a separate example.
  - E.g., a single convolutional layer with 3x3 filters, when applied to a 100x100 pixel image, processes 98^2=9604 examples simultaneously
    - (albeit those examples are not independent).
- Each parameter of the model receives gradient from all locations in its ‘impact field’ (the set of locations in the output image that depend on that parameter)
  - Implication: stronger training signal than when the loss is defined only on the level of entire images (e.g. image classification or regression).

Challenges:

- Border effects: the larger ERF, the more prominent.
- Input-output image pairs have to be precisely spatially coregistered.

Image segmentation

Image segmentation has many faces ...

- ‘Ordinary’ segmentation
  - The algorithm is only required to delineate regions that are perceptually distinct.
- Semantic segmentation
  - The algorithm is expected to work with semantics-rich labels, like sidewalk, car, person, road sign, ...
- Object instance segmentation
  - Multiple instances of the same type of entity should be delineated.
- ...

DNN-based approaches to segmentation

Main categories of representatives:

- Patch-based - a ‘naive’ approach
  - Replaces the segmentation task with classification of individual pixels based of image patches
- Fully convolutional (ConvNet, FCNN)
  - Performs segmentation of all image patches in parallel
- Recurrent
  - Uses recurrent layers/cell/networks to ‘sweep’ the input image.
- Attention-based models
  - Have a separate module that ‘actively seeks’ objects in the image.

Patch-based segmentation

Segmenting Retinal Blood Vessels with Deep Neural Networks
Paweł Liskowski, Krzysztof Krawiec
IEEE Transactions on Medical Imaging, Vol. 35, 11, Nov. 2016
https://ieeexplore.ieee.org/document/7440871?arnumber=7440871

- As of 2016 (and for some time onward) best algorithm for segmentation of fundus imaging.
- Not necessarily a milestone, though nicely cited ;)

A naive approach to segmentation using DL

- A network in a sliding-window setup predicts class label of each pixel by analyzing a local region (patch) around that pixel
- Advantage: each training image gives rise to thousands or more of training examples (patches)

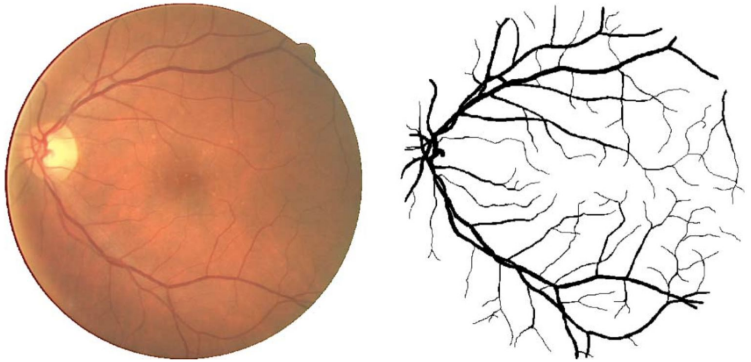


Fig. 1. A training image from the DRIVE database (left) and the corresponding manual segmentation (right).

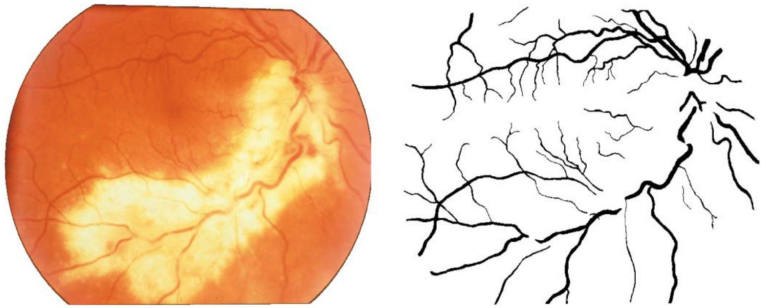


Fig. 2. A pathological image from the STARE database (left) and the corresponding manual segmentation (right).

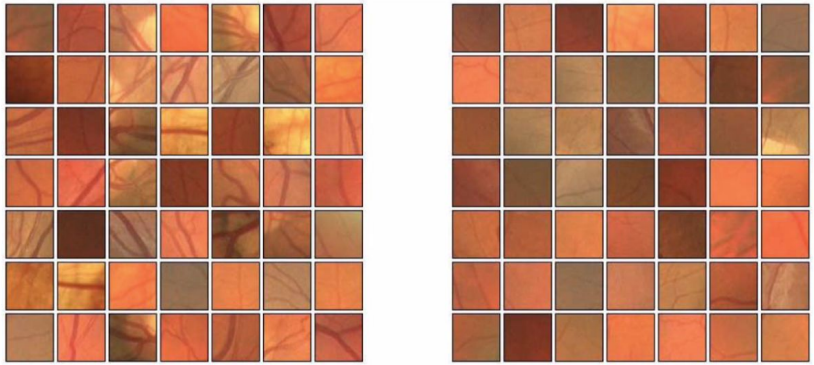


Fig. 4. Examples of positive (left) and negative (right) 27×27 training patches extracted from the DRIVE images.

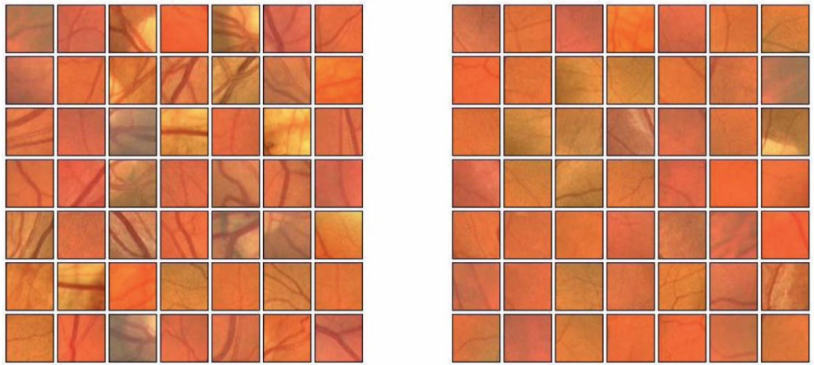


Fig. 5. Examples of positive (left) and negative (right) training patches after applying GCN transformation.

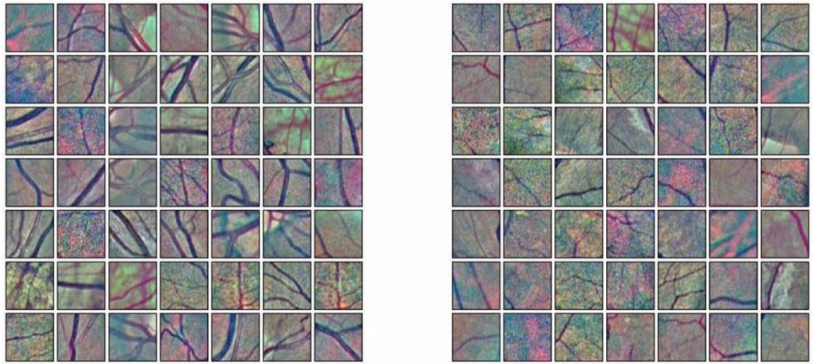


Fig. 6. Positive (left) and negative (right) training patches after applying ZCA whitening transformation.

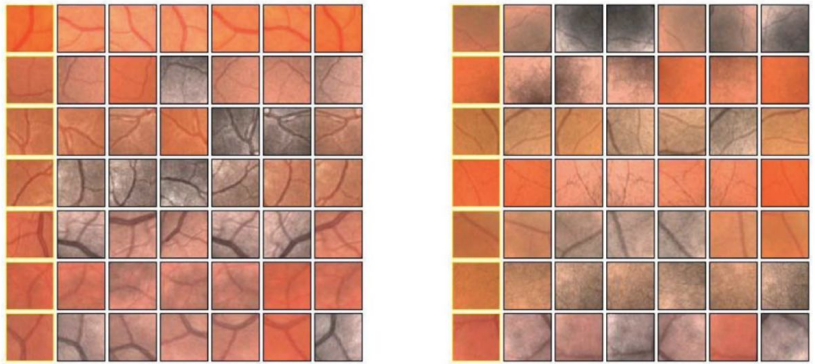
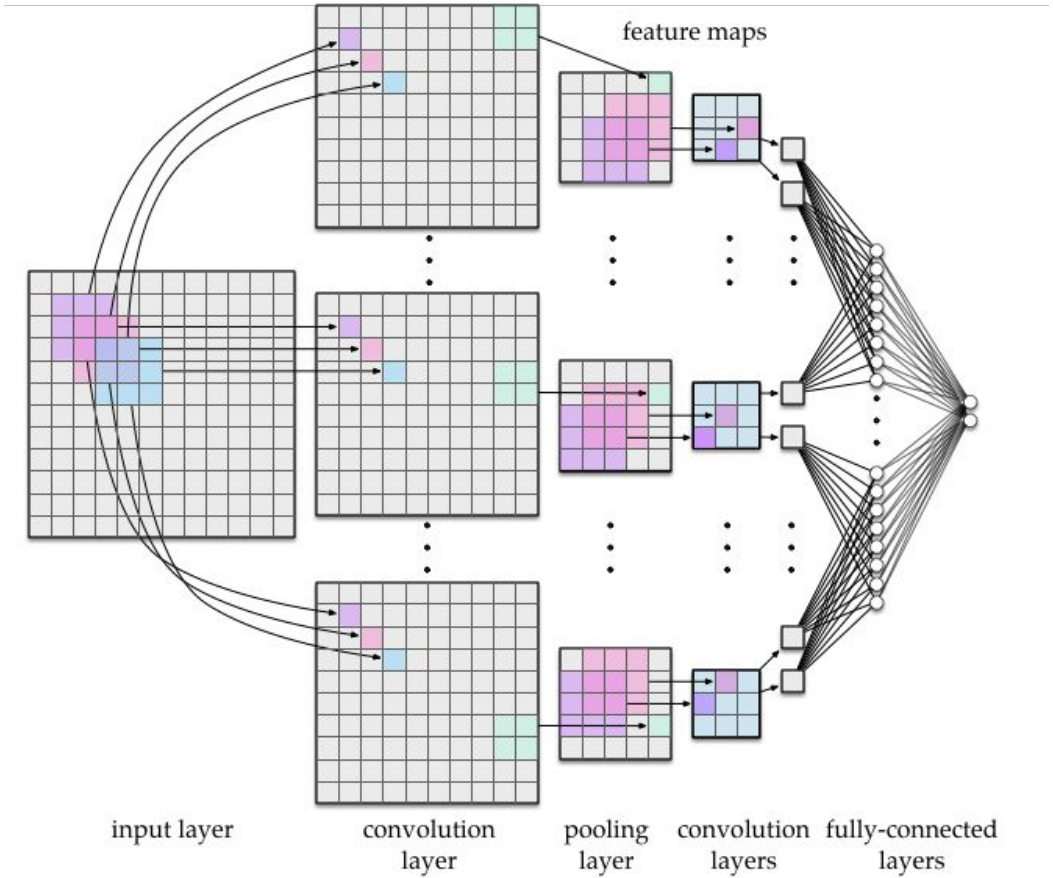


Fig. 7. Augmentations of positive (left) and negative (right) training patches. Each row shows 6 random augmentations of the leftmost patch.

Network architecture

- Two main components:
  - Conventional ConvNet stack
  - Dense stack
- Handles each 27x27 input patch as a separate image/example.



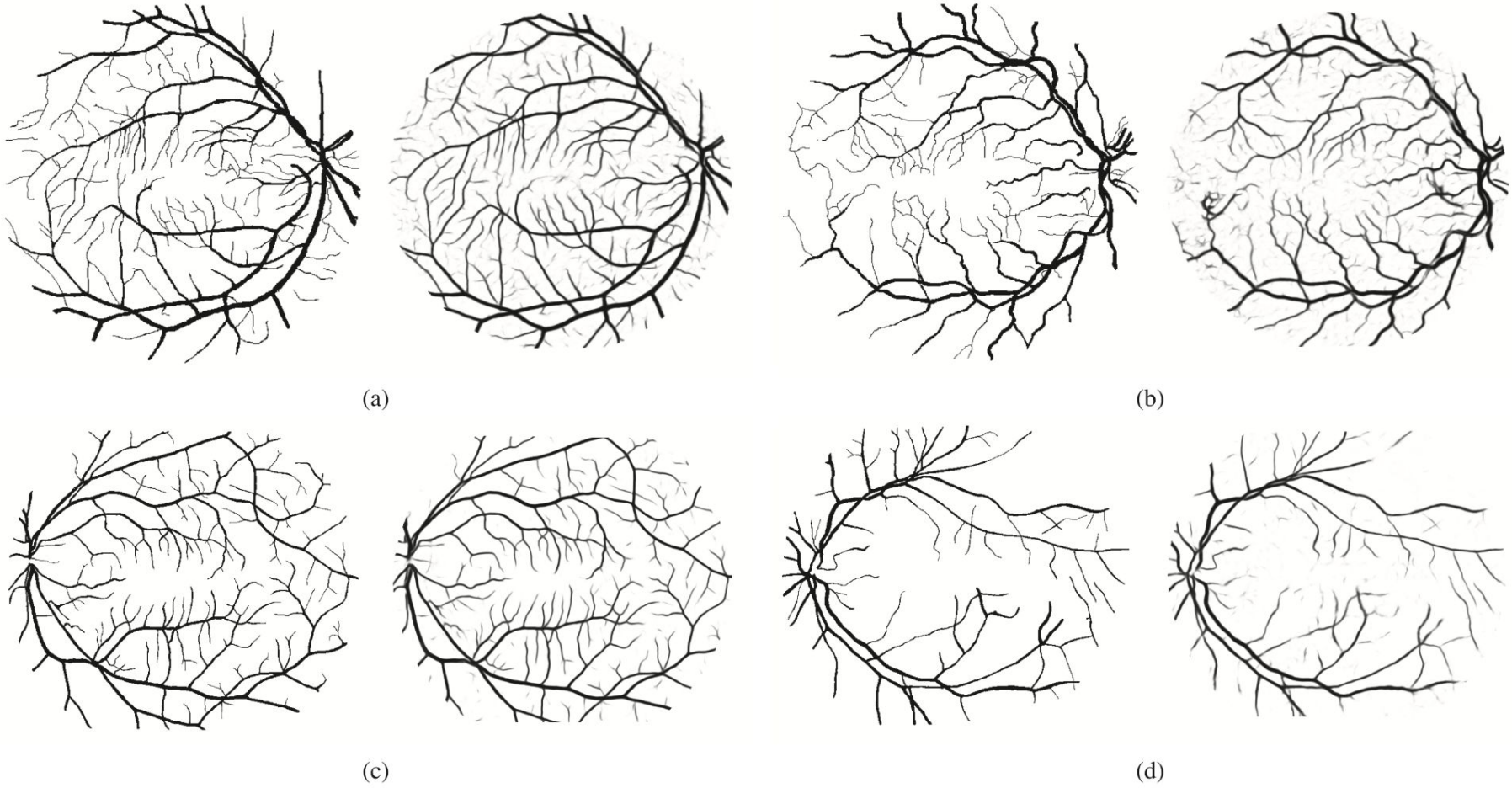


Fig. 9. Ground truth (left) and segmentation result (right) for two healthy subjects: (a) DRIVE image #13, (c) STARE image #0255, and two subjects with pathologies: (b) DRIVE image #8, (d) STARE image #0002.

Some results

TABLE III
PERFORMANCE OF MODELS (POINT ESTIMATES FOR DRIVE, AVERAGES WITH .95 CONFIDENCE INTERVALS FOR STARE)

	DRIVE						STARE					
	AUC	Acc	Acc*	Kappa	Sens	Spec	AUC	Acc	Acc*	Kappa	Sens	Spec
PLAIN	.9683	.9479	.9473	.7653	.7417	.9804	.9767±.0053	.9559±.0071	.9551±.0072	.7477±.0451	.7495±.0721	.9788±.0081
GCN	.9708	.9487	.9475	.7708	.7550	.9792	.9787±.0049	.9571±.0064	.9572±.0064	.7573±.0394	.7620±.0656	.9789±.0072
ZCA	.9719	.9485	.9472	.7756	.7819	.9748	.9783±.0062	.9563±.0064	.9562±.0066	.7598±.0317	.7718±.0490	.9783±.0055
AUGMENT	.9663	.9466	.9453	.7610	.7447	.9784	.9744±.0048	.9527±.0068	.9512±.0069	.7306±.0431	.7376±.0720	.9769±.0086
BALANCED	.9738	.9230	.9251	.7193	.9160	.9241	.9820±.0045	.9309±.0107	.9620±.0051	.7021±.0305	.9307±.0274	.9304±.0133
NO-POOL	.9720	.9495	.9486	.7781	.7763	.9768	.9785±.0066	.9566±.0082	.9568±.0081	.7622±.0415	.7867±.0698	.9754±.0099

TABLE IV
BEST-TO-DATE RESULTS ON DRIVE AND STARE DATABASE. BOLD FONT MARKS THE BEST OVERALL RESULT ACCORDING TO [8]

	DRIVE		STARE	
Method	AUC	Acc	AUC	Acc
2nd. observer [10]	–	.9473	–	.9349
Jiang et al. [27]	.9327	.8911	.9298	.9009
Staal et al. [9]	.9520	.9441	.9614	.9516
Soares et al. [16]	.9614	.9466	.9671	.9480
Ricci et al. [12]	.9558	.9563	.9602	**.9584**
Villalobos-Castaldi et al. [13]*	–	**.9759**	–	–
Osareh et al. [28]	**.9650**	–	–	–
Marin et al. [18]	.9588	.9452	**.9769**	.9526
Roychowdhury et al. [17]	.9620	.9519	.9688	.9515
Azzopardi et al. [14]	.9614	.9442	.9563	.9497
Zhao et al. [15]	.8620	.9540	.8740	.9560

*Result obtained with a different definition of FOV.

Structured prediction

Rather than predicting for a single pixel, predict the labels for a 5x5 patch (centered at the input patch).

- Notice: the model is still partially dense (it's not a fully convolutional NN).

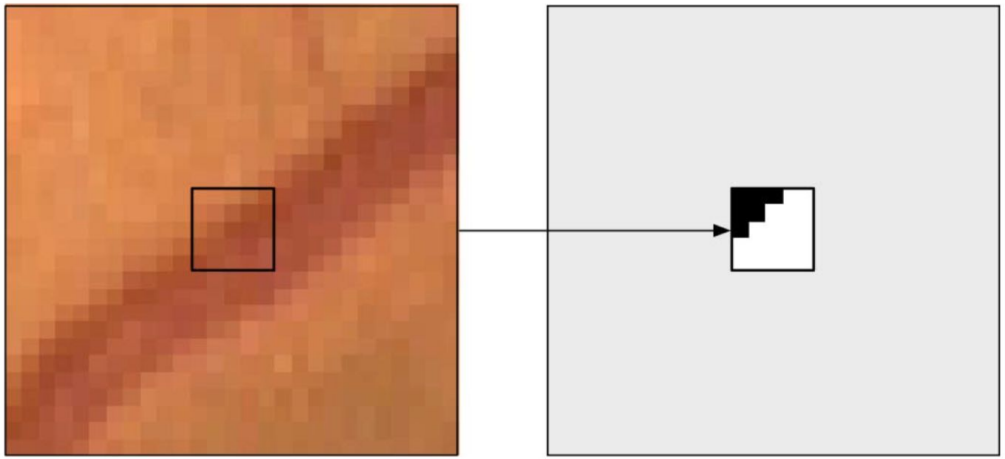


Fig. 8. A training example for the SP approach: the 27×27 patch with the $s \times s = 5 \times 5$ output window (left) and the corresponding desired output (right).

Structured prediction

Some results:

- Significant decrease of the FP rate, while maintaining similar recall.

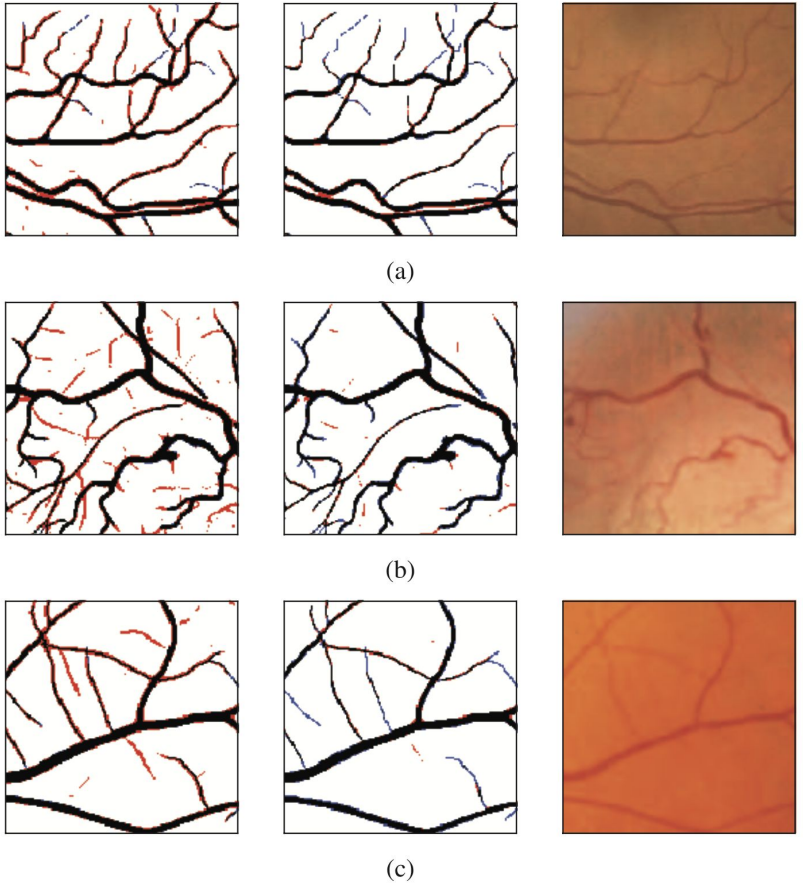


Fig. 10. Examples of segmentation by non-SP (left) and SP (middle) models for images (right) of DRIVE (a) and STARE (b, c) database. Red pixels: FP errors, blue: FN errors, black: TP, white: TN.

Some challenges

Central vessel reflex:

- Vessels are 3D structures
- Part of vessel surface is perpendicular to the light beam and causes strong reflection, much stronger than the other parts of vessel surface.
- As a result, the vessel may render as two dark parallel chains of pixels, which may suggest the presence of two vessels.

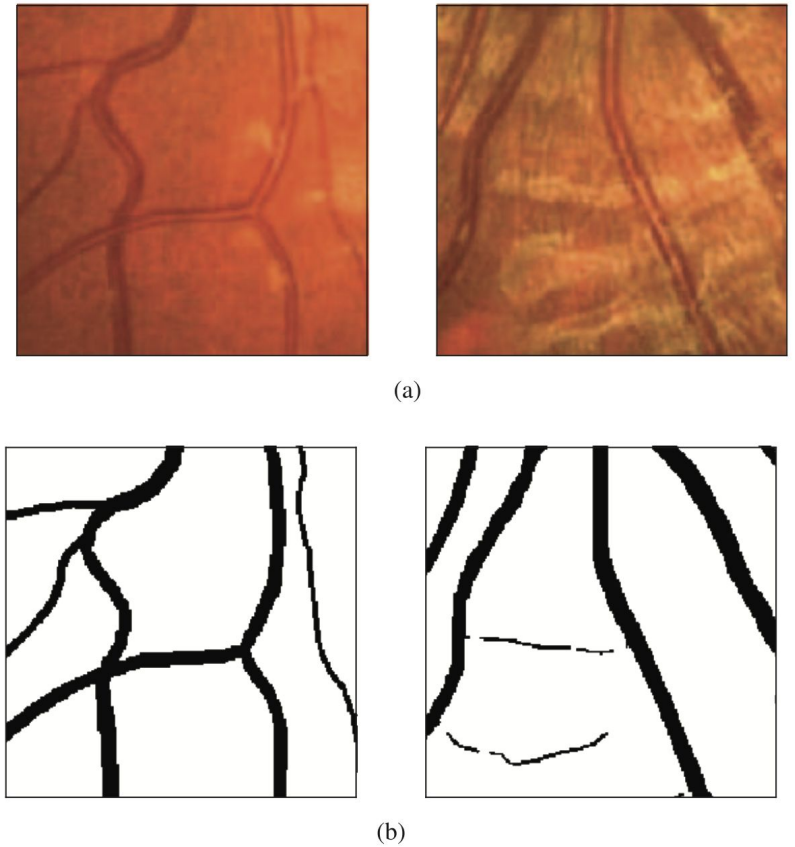


Fig. 11. Image fragments with central vessel reflex (a) and corresponding segmentation results (b); CHASE database.

Patch-based segmentation: Summary

Advantages:

- Conceptually simple.
- Can be easily extended, e.g. with structured prediction.

Disadvantages:

- Slow, in particular at querying, because of explicit scanning of the image.
- Trade-off between the localization accuracy and the use of context:
  - Larger patches require typically more max-pooling layers, which reduces localization accuracy (spatial ‘aliasing’)

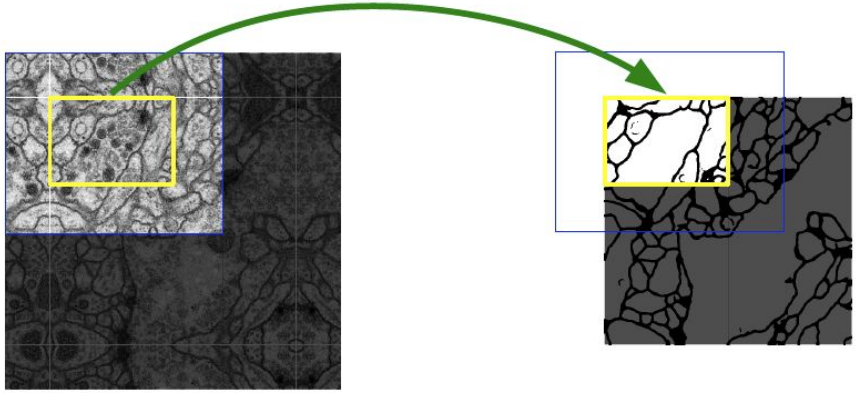
U-Net

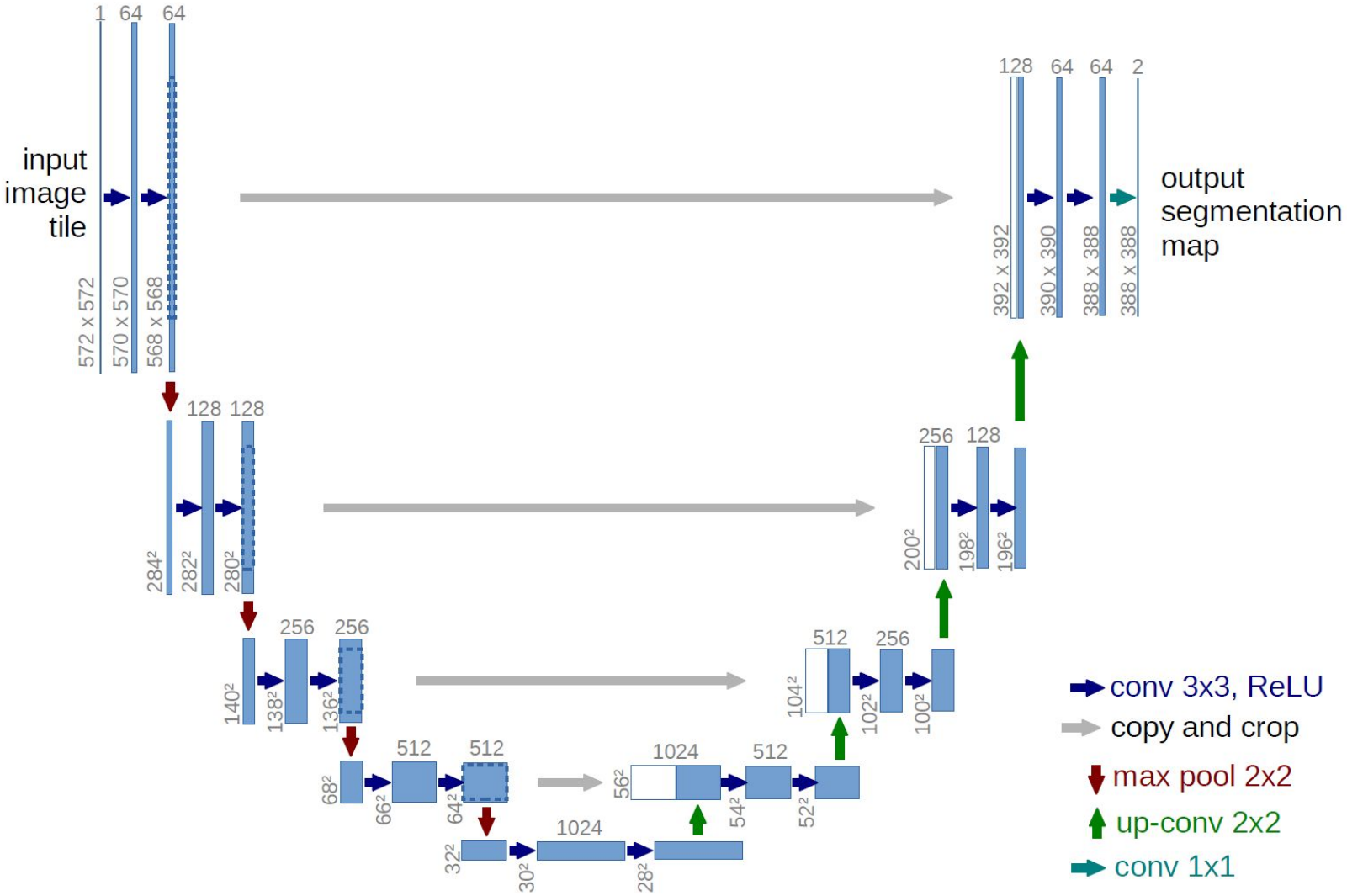
U-Net: Convolutional Networks for Biomedical Image Segmentation
Olaf Ronneberger, Philipp Fischer, Thomas Brox
https://arxiv.org/abs/1505.04597

- One of the most popular ‘design patterns’ for fully convolutional architectures.
- Proved very effective in a large number of applications.
- 10k citations on arXiv (as of Dec 2021)

The architecture

- Uses ConvNet to supplement a usual contracting network by successive expanding layers, where pooling operators are replaced by upsampling operators.
- Main building blocks:
  - a contracting stack to capture context and construct higher-level features,
  - a symmetric expanding stack that enables precise localization,
  - ‘copy&crop’ connections bridging the contracting path with the expanding path.
- The network does not have any fully connected layers and only uses the valid part of each convolution.
  - Padding via mirroring at the edge of the image.




input image tile
output segmentation map
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1

Stride and fractional convolutions

Stride controls the speed of shifting the mask (receptive field) over the input tensor comp *relative to the speed of moving over output tensor*.

$$\text{size(output)} = \text{stride} * \text{size(input)}$$

- stride=1 ⇒ size(output) = size(input)
- stride>1 ⇒ size(output) < size(input)
  - Example: stride=2 implies skipping every second receptive field in the *input*
- stride<1 ⇒ size(output) > size(input)
  - Example: stride=1/2 implies skipping every second receptive field in the *output*

Naming convention:

- stride≥1 : *convolution*
- stride<1 : *fractional convolution* (because in this case stride=1/k, k∈N)

Transposed convolutions

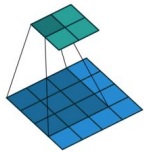
The distinction between the ‘regular’ convolution and the transposed convolution (*up-conv*) concerns the ‘fan-in’ and ‘fan-out’, i.e. the flow of signals.

- Regular convolutions are *contracting*: a k×k receptive field in input is mapped/projected to a single element (unit, location) in output.
- Transposed convolutions are *expanding*: a single input (unit, pixel) is mapped/projected onto a rectangular region in the output tensor.
  - The contributions from individual inputs are subsequently summed.

Transposed convolutions are sometimes (erroneously!) called deconvolutions.

Unet uses transposed fractional convolutions in the expanding path.

Recommended: helpful animated illustrations of various types of convolutions
https://github.com/vdumoulin/conv_arithmetic



Some comments on U-net

- Advantages: Trains effectively from small samples.
  - (like most pure ConvNets)
- Can be trained to label:
  - Regions (interiors)
  - Region borders
- Q: Is this an autoencoder?
- Important: The ‘copy and crop’ connections are not equivalent to residual connections.
  - However, they definitely allow part of gradient flow along a shorter path.
- This is a ‘fully ConvNet’ model:
  - All operations (convolutions, max pooling, up-conv) rely on local receptive fields.
  - Therefore, U-nets are scalable/applicable to images of arbitrary dimensions.

U-Net's weighted target on HeLa cells image

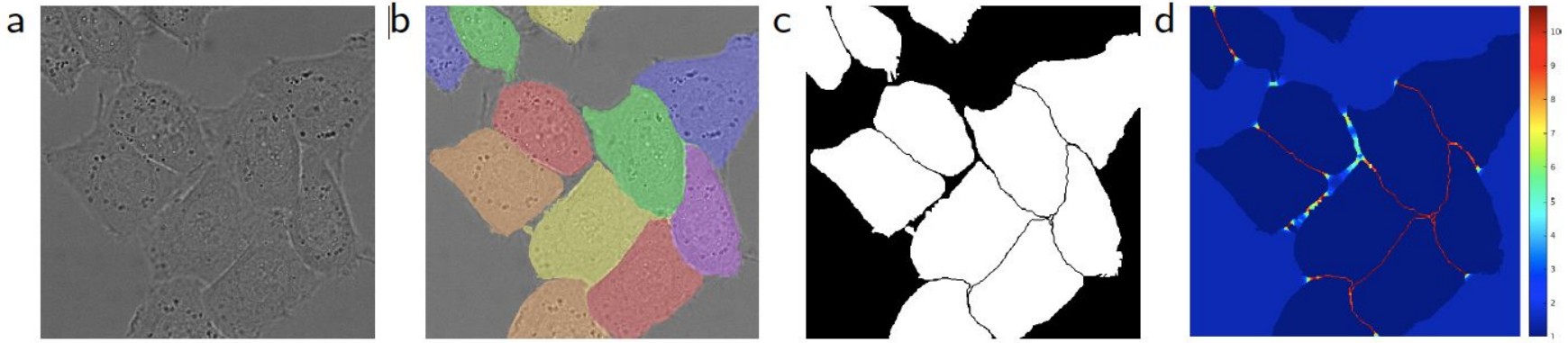
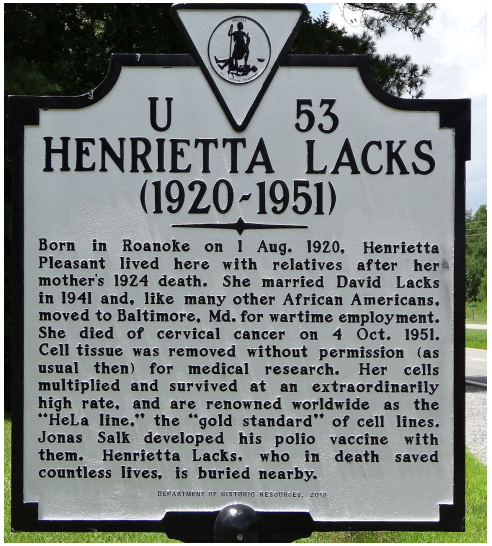


Fig. 3. HeLa cells on glass recorded with DIC (differential interference contrast) microscopy. (**a**) raw image. (**b**) overlay with ground truth segmentation. Different colors indicate different instances of the HeLa cells. (**c**) generated segmentation mask (white: foreground, black: background). (**d**) map with a pixel-wise loss weight to force the network to learn the border pixels.

Digression: HeLa cell line

- HeLa (Henrietta Lacks) is an immortal cell line used in scientific research.
  - The first human cells grown in a lab that were naturally "immortal", meaning that they do not die after a set number of cell divisions
- It is the oldest and most commonly used human cell line (1951).
- Scientists have grown an estimated 50 million metric tons of HeLa cells
- Almost 11,000 patents based on the HeLa line.

Henrietta Lacks (1920-1951)



Other features: per-pixel loss weighing

- Uses weighted target: pixels located at boundaries of target regions (e.g. between touching objects, like cells) obtain relatively large weight (separation boundaries computed using morphological operations):

$$w(\mathbf{x}) = w_c(\mathbf{x}) + w_0 \cdot \exp\left(-\frac{(d_1(\mathbf{x}) + d_2(\mathbf{x}))^2}{2\sigma^2}\right)$$

where:

  - w_c: balances class frequencies,
  - d_1: distance to the border of the nearest object (in the original paper: cell),
  - d_2: distance to the border of the second nearest cell.
  - The authors used w_0=10

Other features

- Loss function: cross-entropy on output (softmax) and target labels

$$p_k(\mathbf{x}) = \exp(a_k(\mathbf{x})) / \left(\sum_{k'=1}^{K} \exp(a_{k'}(\mathbf{x})) \right)$$

$$E = \sum_{\mathbf{x} \in \Omega} w(\mathbf{x}) \log(p_{\ell(\mathbf{x})}(\mathbf{x}))$$

- Notice: ‘soft labels’ resulting from weighing the loss according to the w(x) weights (see previous slide).
- The authors made extensive use of augmentations in training.

Some results

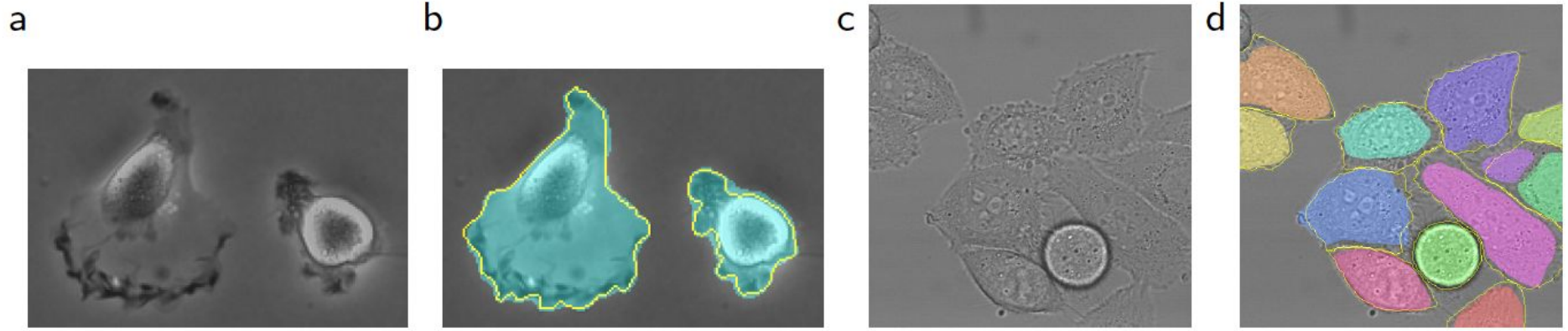


Fig. 4. Result on the ISBI cell tracking challenge. (**a**) part of an input image of the "PhC-U373" data set. (**b**) Segmentation result (cyan mask) with manual ground truth (yellow border) (**c**) input image of the "DIC-HeLa" data set. (**d**) Segmentation result (random colored masks) with manual ground truth (yellow border).

Some results

Name	PhC-U373	DIC-HeLa
IMCB-SG (2014)	0.2669	0.2935
KTH-SE (2014)	0.7953	0.4607
HOUS-US (2014)	0.5323	-
second-best 2015	0.83	0.46
u-net (2015)	**0.9203**	**0.7756**

The metric: IOU = Intersection over Union

- A: detected region
- B: target region

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Final comment:

- The original paper assumed the model has to label *regions*.
- UNet works also quite well for labeling *borders*.

Follow-ups

U-Net is one of the most successful architectures for image segmentation

- In particular in medical applications.

A number of variants/extensions have been proposed; among them:

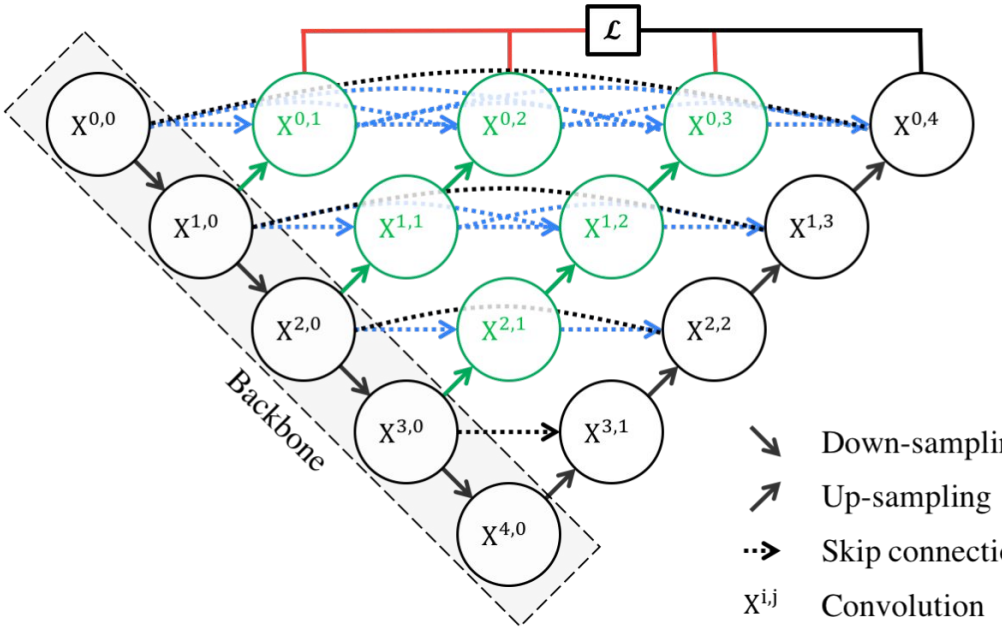
- Replacing max pooling with downsampling
- V-Net: https://arxiv.org/abs/1606.04797
  - works with volumetric (3D) data,
  - adds residual connections between convolutional layers at the same stage of processing,
  - uses histogram matching for augmentation.
- UNet++: see next slide.

UNet++

UNet++: A Nested U-Net Architecture for Medical Image Segmentation
Zongwei Zhou, Md Mahfuzur Rahman Siddiquee, Nima Tajbakhsh, Jianming Liang
https://arxiv.org/abs/1807.10165v1

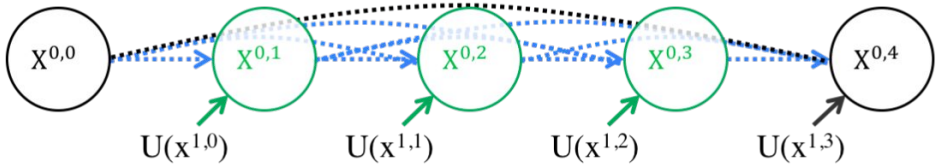
Differences w.r.t. ‘regular’ UNet:

1. Re-designed skip pathways (shown in green and blue in the next slide) that connect the contracting and expanding path.
    - Those are meant to ‘bridge the semantic gap’ between those pathways.
    - The skip connections are dense (see: DenseNet).
2. Use of deep supervision (shown red).



(a)

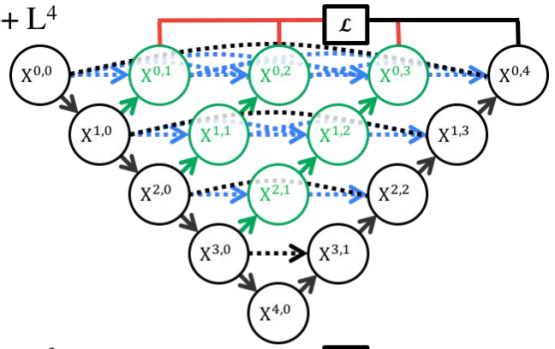
$x^{0,1} = H[x^{0,0}, U(x^{1,0})]$ $x^{0,2} = H[x^{0,0}, x^{0,1}, U(x^{1,1})]$ $x^{0,3} = H[x^{0,0}, x^{0,1}, x^{0,2}, U(x^{1,2})]$



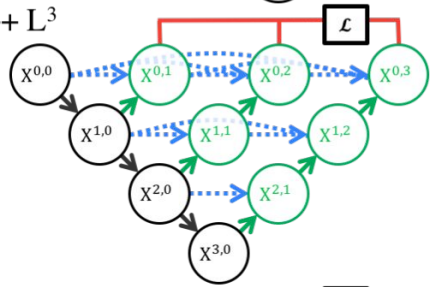
(b)

$x^{0,4} = H[x^{0,0}, x^{0,1}, x^{0,2}, x^{0,3}, U(x^{1,3})]$

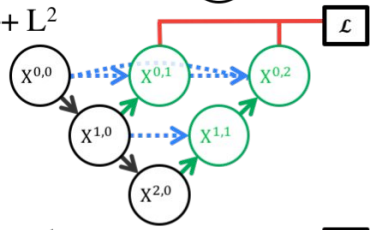
UNet++ L^4



UNet++ L^3



UNet++ L^2



UNet++ L^1

(c)

UNet++

Reported to significantly outperform U-Net on Net in Electron Microscopy, Cell, Nuclei, Brain Tumor, Liver and Lung Nodule medical image segmentation tasks.

Intersection-over-Union (IoU, %):

Architecture	Params	Dataset			
		cell nuclei	colon polyp	liver	lung nodule
U-Net [9]	7.76M	90.77	30.08	76.62	71.47
Wide U-Net	9.13M	90.92	30.14	76.58	73.38
UNet++ w/o DS	9.04M	**92.63**	**33.45**	79.70	76.44
UNet++ w/ DS	9.04M	92.52	32.12	**82.90**	**77.21**

DS = Deep Supervision

Image denoising

A Deep Learning Approach to Denoise Optical Coherence Tomography Images of the Optic Nerve Head
S. K. Devalla et al.
Scientific Reports, vol. 9, no. 1, Art. no. 1
https://www.nature.com/articles/s41598-019-51062-7

- One of quite many works on DL for image denoising.

Image denoising

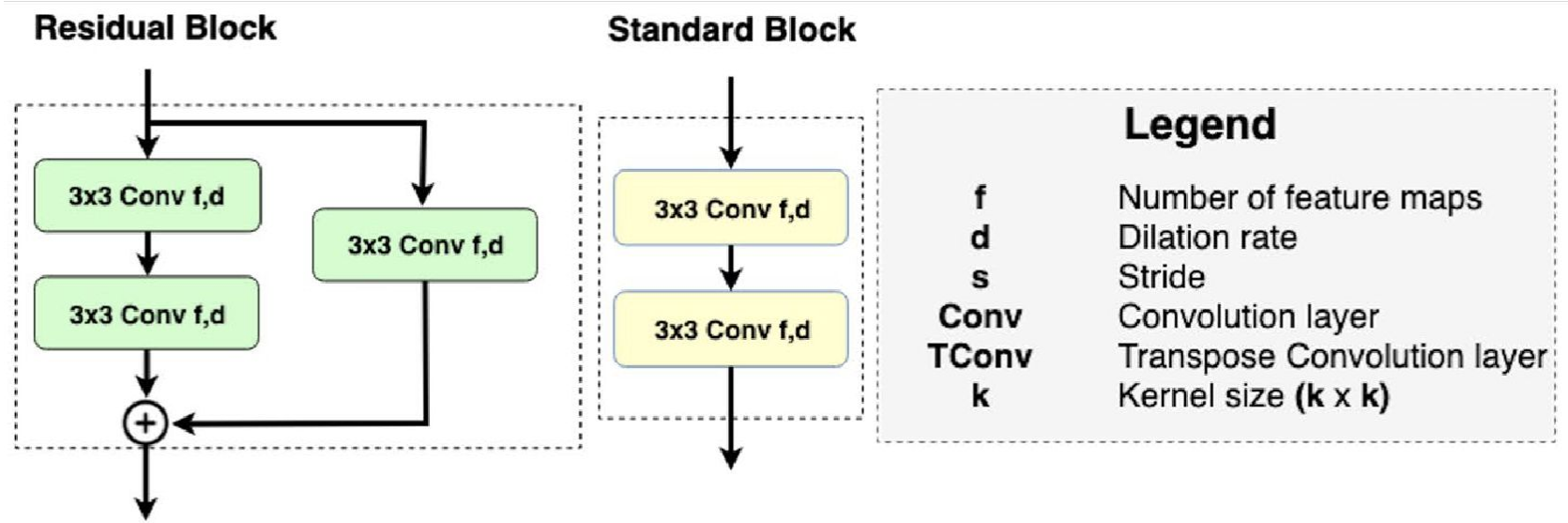
- Noise is pestering all image acquisition techniques.
- Technical means used in practice (sometimes in combination):
  a. Image filtering (typically low-pass filtering)
     Downside: image distortion (loss of detail)
  b. Image/frame averaging
     Downside: requires multiple image acquisitions and well coregistered frames (static scene).
     Often additional co-registering phase is necessary.

Popularity of image averaging (b) provides ready-to-use training data in the form (*set of raw images*, *corresponding target averaged image*).

- Common in medical imaging, for instance MRI and OCT
  - Denoising particularly desirable in medical contexts: patients cannot hold their breath forever, and signal cannot be made stronger.
- Leads naturally to posing the task: learn a mapping from a single raw image to the target image.
  - A single training example has the form (raw image, *corresponding target averaged image*)

Architecture: core components

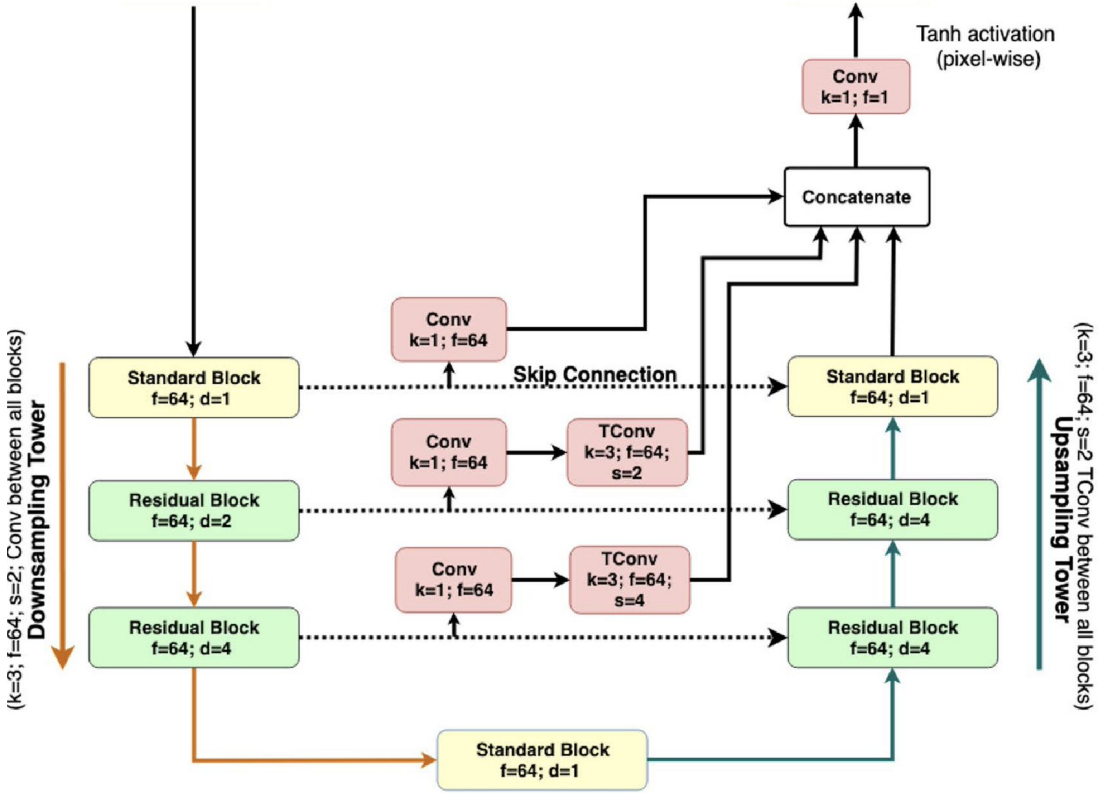
‘Standard’ blocks and ‘residual’ blocks:

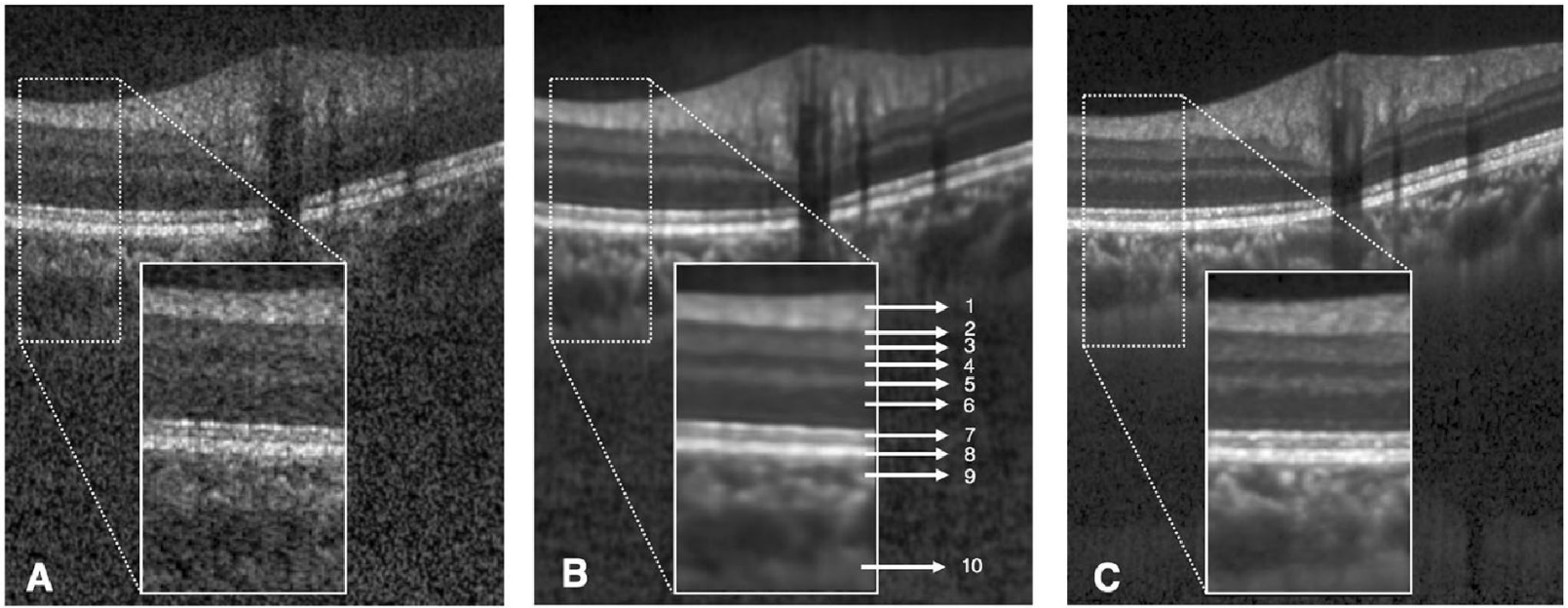


Q: Is this really a residual block?

Architecture

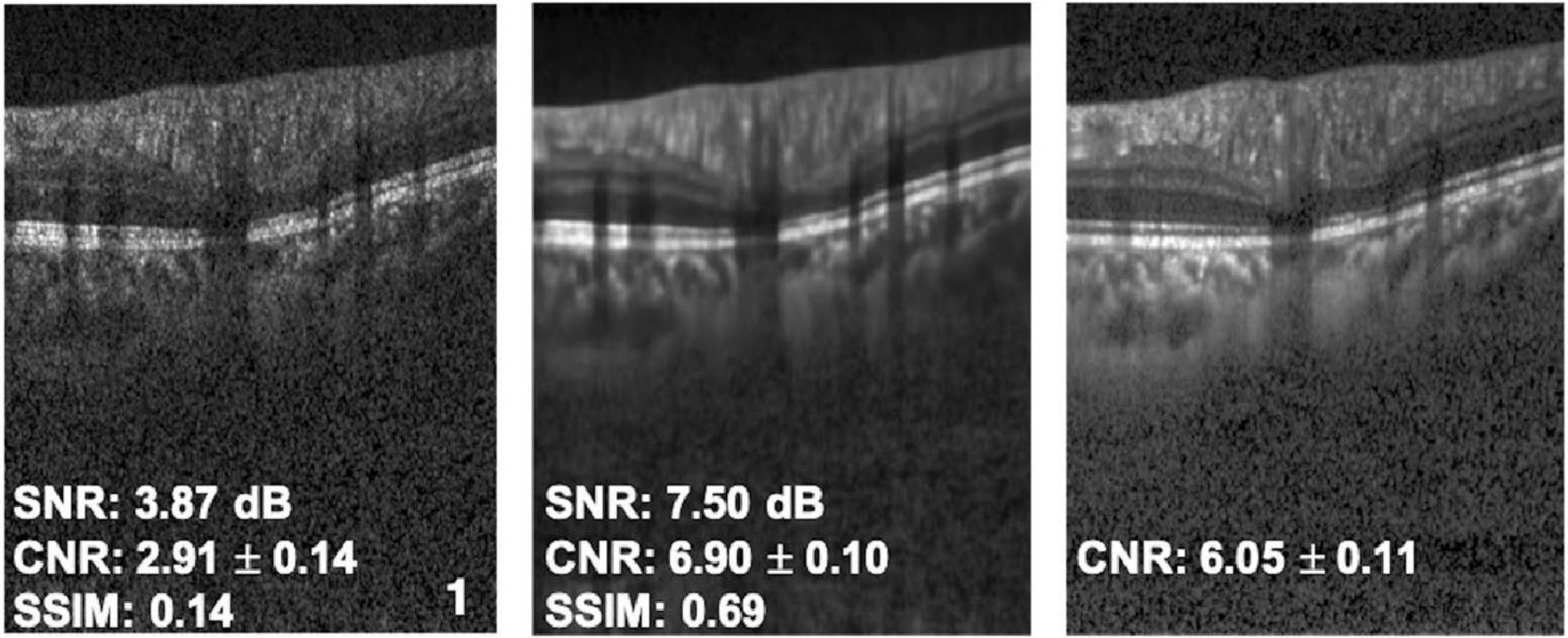
Fully-conv NN inspired by U-net. Also: uses dilated convolutions.





Legend

1. Retinal Nerve Fiber Layer (RNFL)
2. Ganglion Cell Layer (GCL)
3. Inner Plexiform Layer (IPL)
4. Inner Nuclear Layer (INL)
5. Outer Plexiform Layer (OPL)
6. Outer Nuclear Layer (ONL)
7. Photoreceptor Layers
8. Retinal Pigment Epithelium (RPE)
9. Choroid
10. Sclera


Single-Frame
Denoised
Multi-Frame
SNR: 3.87 dB
CNR: 2.91 ± 0.14
SSIM: 0.14
1
SNR: 7.50 dB
CNR: 6.90 ± 0.10
SSIM: 0.69
CNR: 6.05 ± 0.11

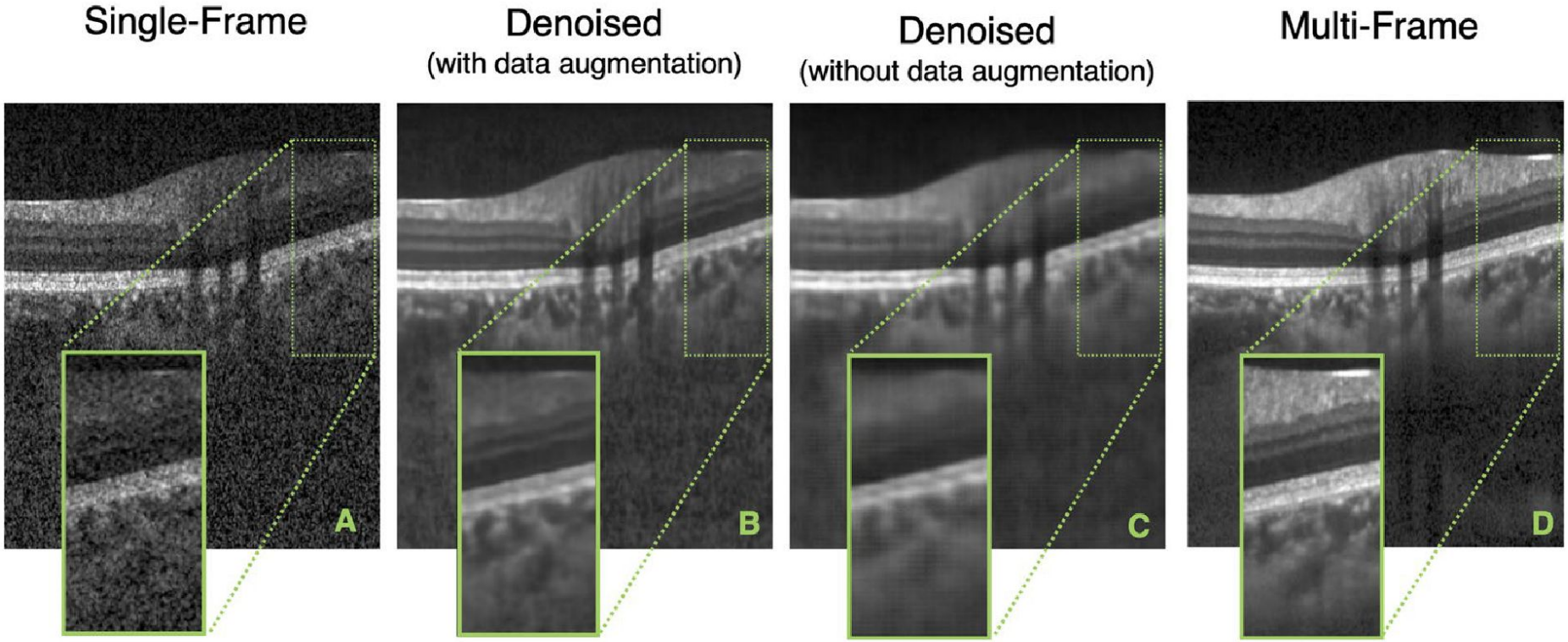
Quantitative analysis

Contrast-to-noise ratio, CNR (the larger, the better):

Tissue	Single-frame	Denoised	Multi-frame
RNFL	2.97 ± 0.42	7.28 ± 0.63	5.18 ± 0.76
GCL + IPL	3.83 ± 0.43	12.09 ± 4.22	11.62 ± 1.85
All other retinal layers	2.71 ± 0.33	5.61 ± 1.46	4.62 ± 0.86
RPE	5.62 ± 0.72	9.25 ± 2.25	8.10 ± 1.44
Choroid	2.99 ± 0.43	5.99 ± 0.45	5.75 ± 0.63
Sclera	2.42 ± 0.39	6.40 ± 1.68	6.00 ± 0.96
LC	4.02 ± 1.23	6.81 ± 1.99	6.46 ± 1.81

Impact of data augmentation

Augmentations used: horizontal flips, rotations, elastic distortions, occlusions



Structural similarity (SSIM) index

- A measure of image similarity adjusted for human perception, meant to address
  - luminance masking (distortions harder to perceive in bright regions) and
  - contrast masking (distortions hard to notice in highly textured regions)
- Used in, among others, superresolution.
- Range: [-1, 1]:

$$\mathrm{SSIM}(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

Other formulation: luminance * contrast * structure:

$$\mathrm{SSIM}(x,y) = \left[l(x,y)^\alpha \cdot c(x,y)^\beta \cdot s(x,y)^\gamma\right]$$

Conclusions

- FCNNs are viable architectures for single-frame denoising.
- Results quantified with CNR (contrast to noise ratio), SNR (signal to noise ratio) and MSSIM (mean SSIM, structural similarity index measure)
  - Five-fold increase of MSSIM compared to raw images.
- Data augmentations improve the quality of denoised images
  - In particular the perceived quality, which is often hard to quantify with metrics.
- Also verified: no negative impact on the accuracy of measurements (of anatomical structures) compared to multi-frame images.
  - No significant difference of mean measurements ($p < 0.05$).

End of module

[Optional: Advanced ConvNets]