Convolutional Neural Networks

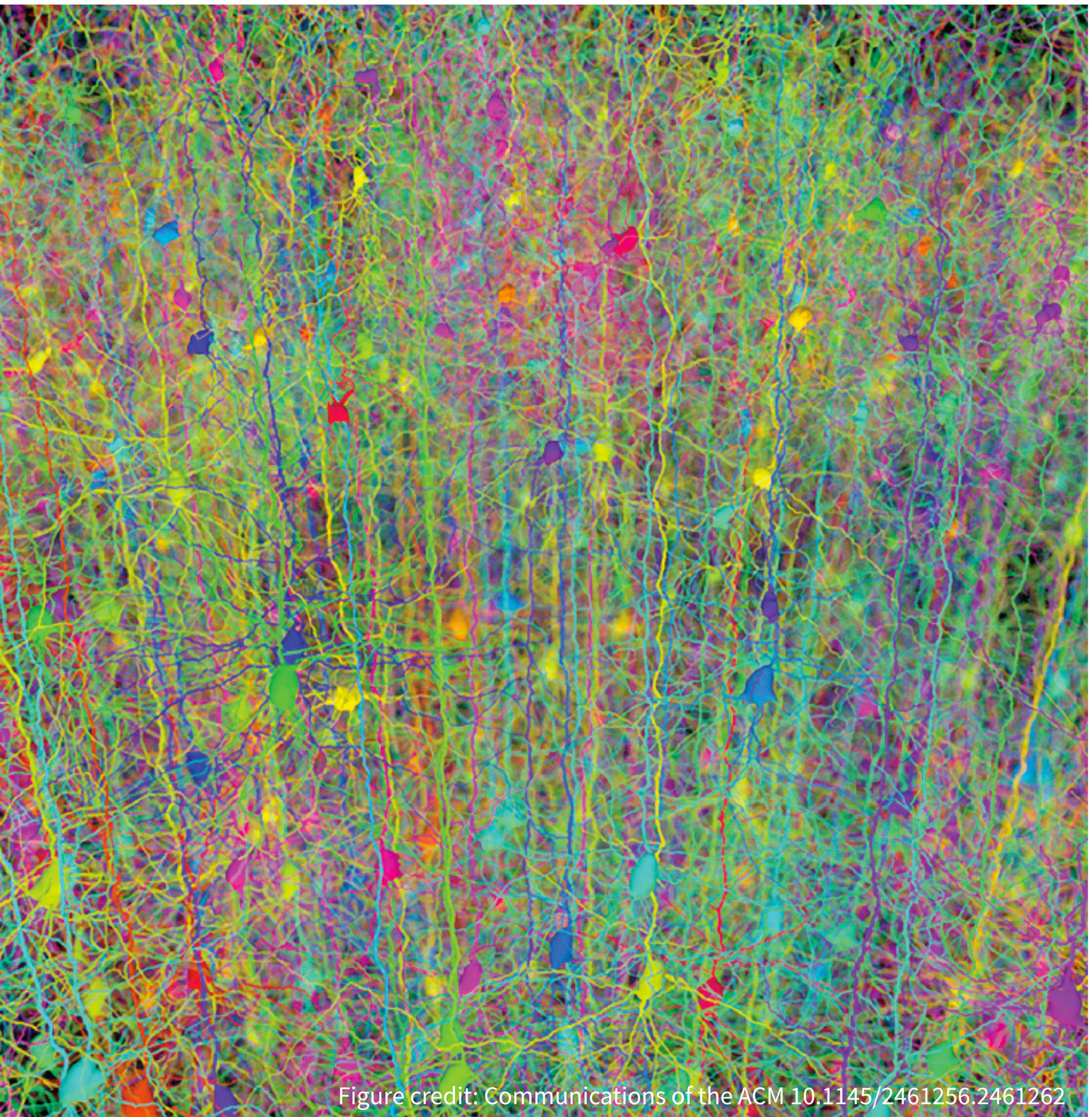


Figure credit: Communications of the ACM 10.1145/2461256.2461262

GRAHAM TAYLOR

VECTOR INSTITUTE

SCHOOL OF ENGINEERING
UNIVERSITY OF GUELPH

CANADIAN INSTITUTE
FOR ADVANCED RESEARCH

UNIVERSITY of GUELPH
CHANGING LIVES
IMPROVING LIFE

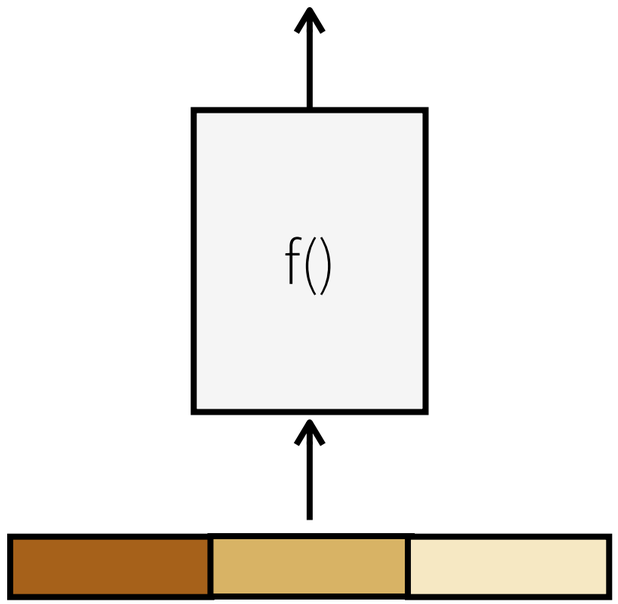
CIFAR
CANADIAN
INSTITUTE
FOR
ADVANCED
RESEARCH

DNNs for Visual Recognition

- For a long time, images have been a common “testbed” for deep learning
- Computer vision and machine learning have been intimately linked, moreso in recent years
- The application of feedforward neural networks to image data has typically been patch-based, or on very tiny images, e.g.
  - MNIST (28 × 28)
  - CIFAR-10 (32 × 32)
  - SVHN (32 × 32)
  - etc…

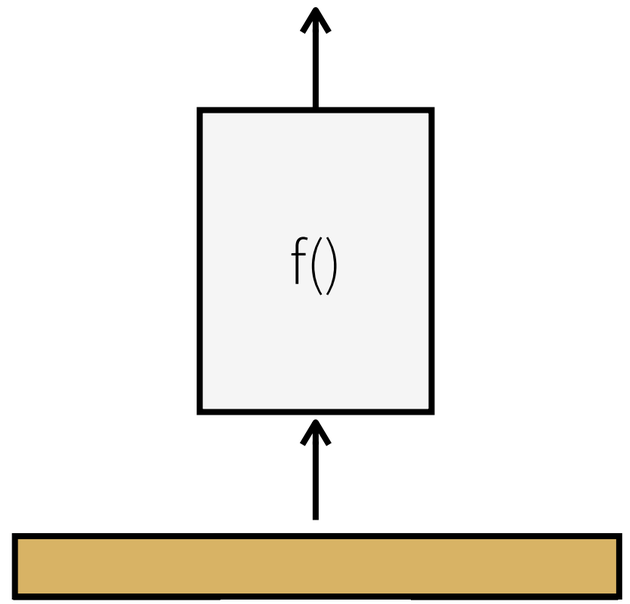
Encoding Domain Knowledge

- We've so-far operated on vectors
  - Paying no heed to structured relationships among elements
- Images can also be processed as vectors
  - Ignoring that we are dealing with pixels
- But this ignores readily available domain knowledge about the structure of the problem



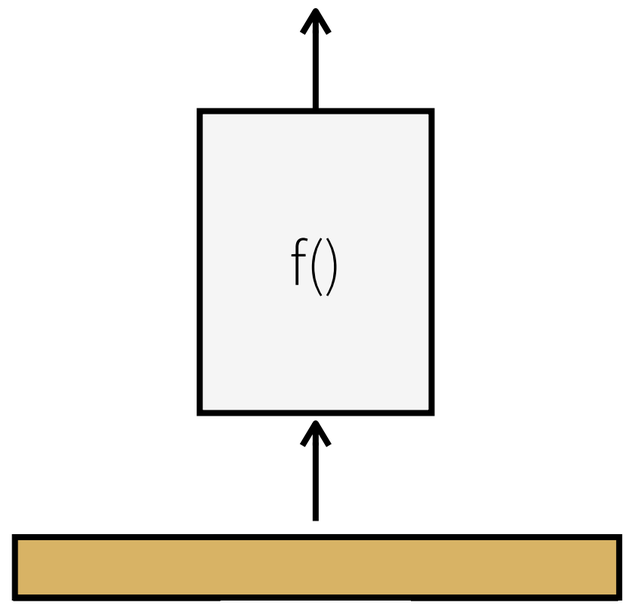
Encoding Domain Knowledge

- We've so-far operated on vectors
  - Paying no heed to structured relationships among elements
- Images can also be processed as vectors
  - Ignoring that we are dealing with pixels
- But this ignores readily available domain knowledge about the structure of the problem

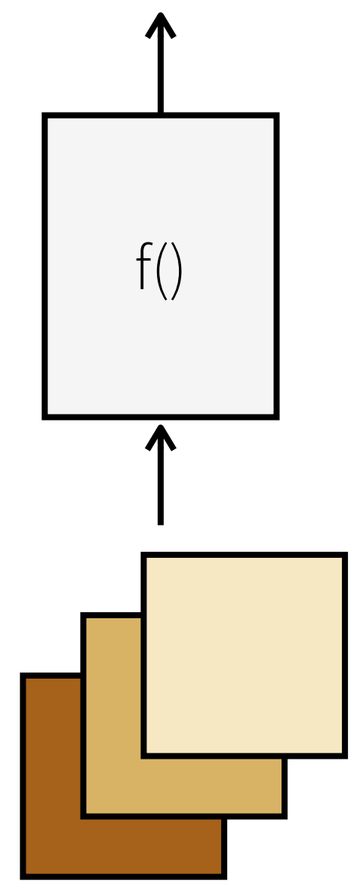


Encoding Domain Knowledge

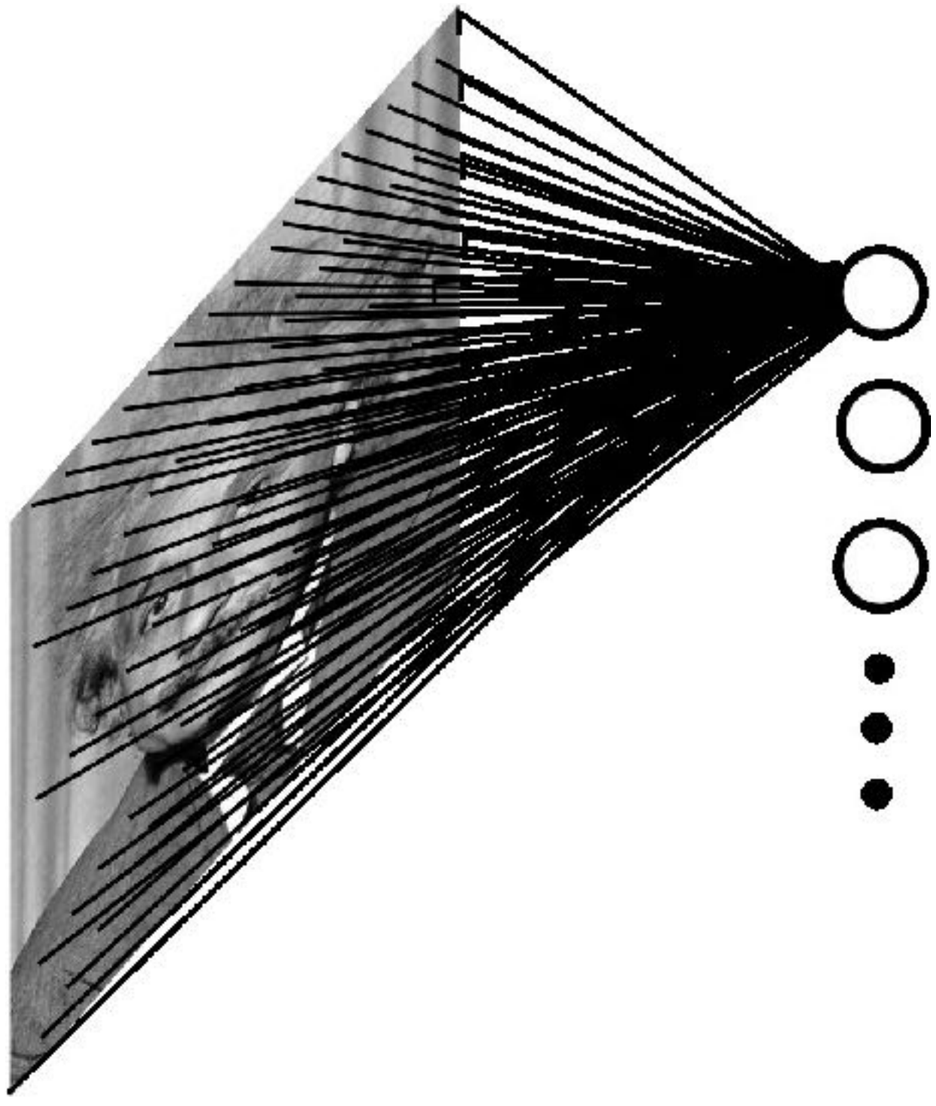
- We've so-far operated on vectors
  - Paying no heed to structured relationships among elements



- Images can also be processed as vectors
  - Ignoring that we are dealing with pixels
- But this ignores readily available domain knowledge about the structure of the problem



Fully-Connected Layer



Example: 200 ×200 image

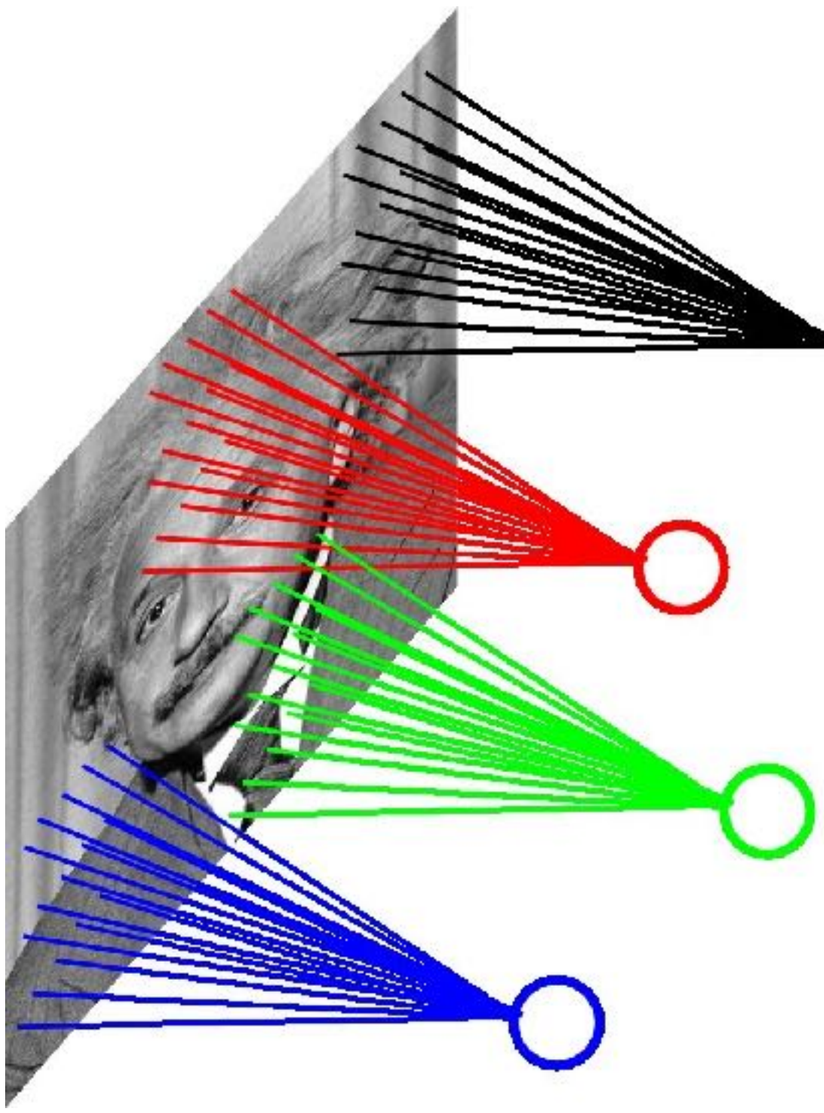
40k hidden units

➡ ~2B parameters!

- Spatial correlation is local
- Waste of resources
- We don’t have enough training examples to fit!

Credit: Marc’Aurelio Ranzato

Locally-Connected Layer



Example: 200 × 200 image

40k hidden units

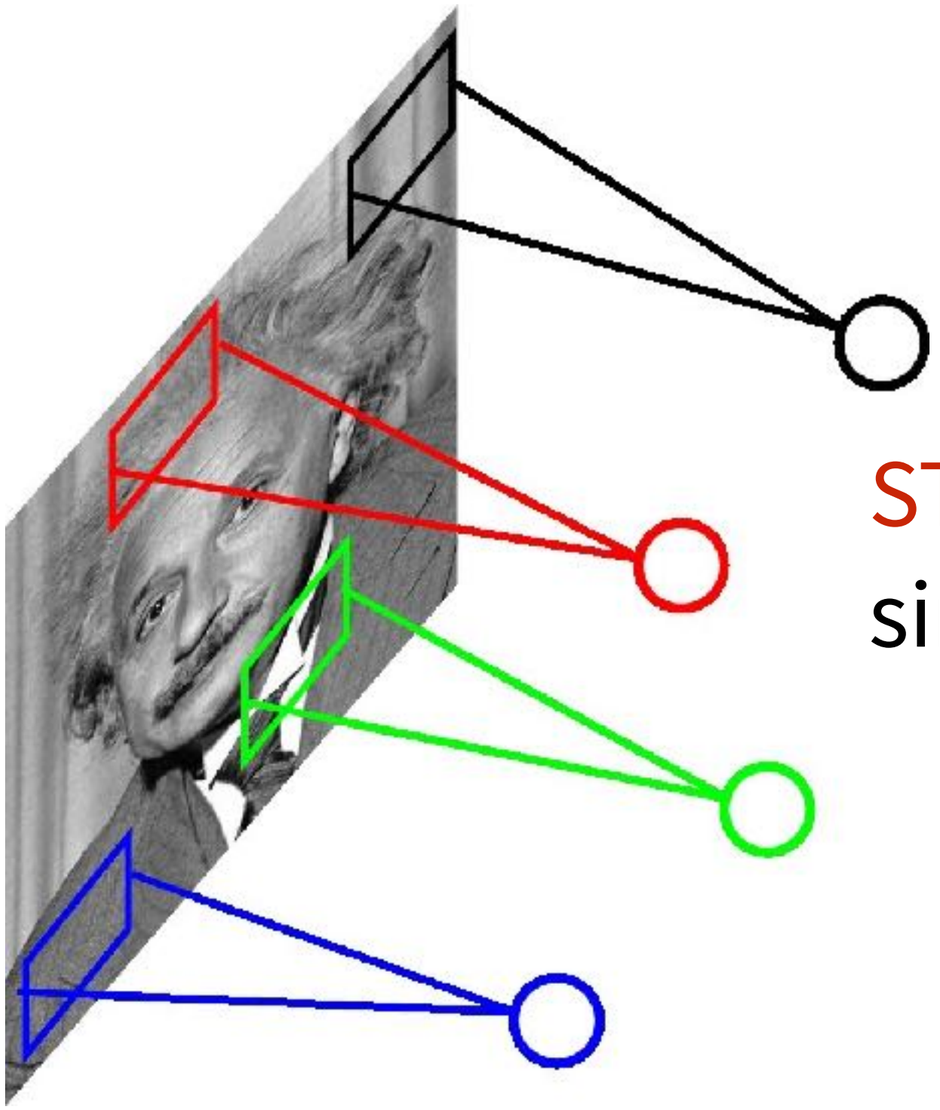
Filter size: 10 × 10

4M parameters

Note: this parameterization is good when the input images are registered (e.g. face recognition)

Credit: Marc'Aurelio Ranzato

Locally-Connected Layer



Example: 200 × 200 image

40k hidden units

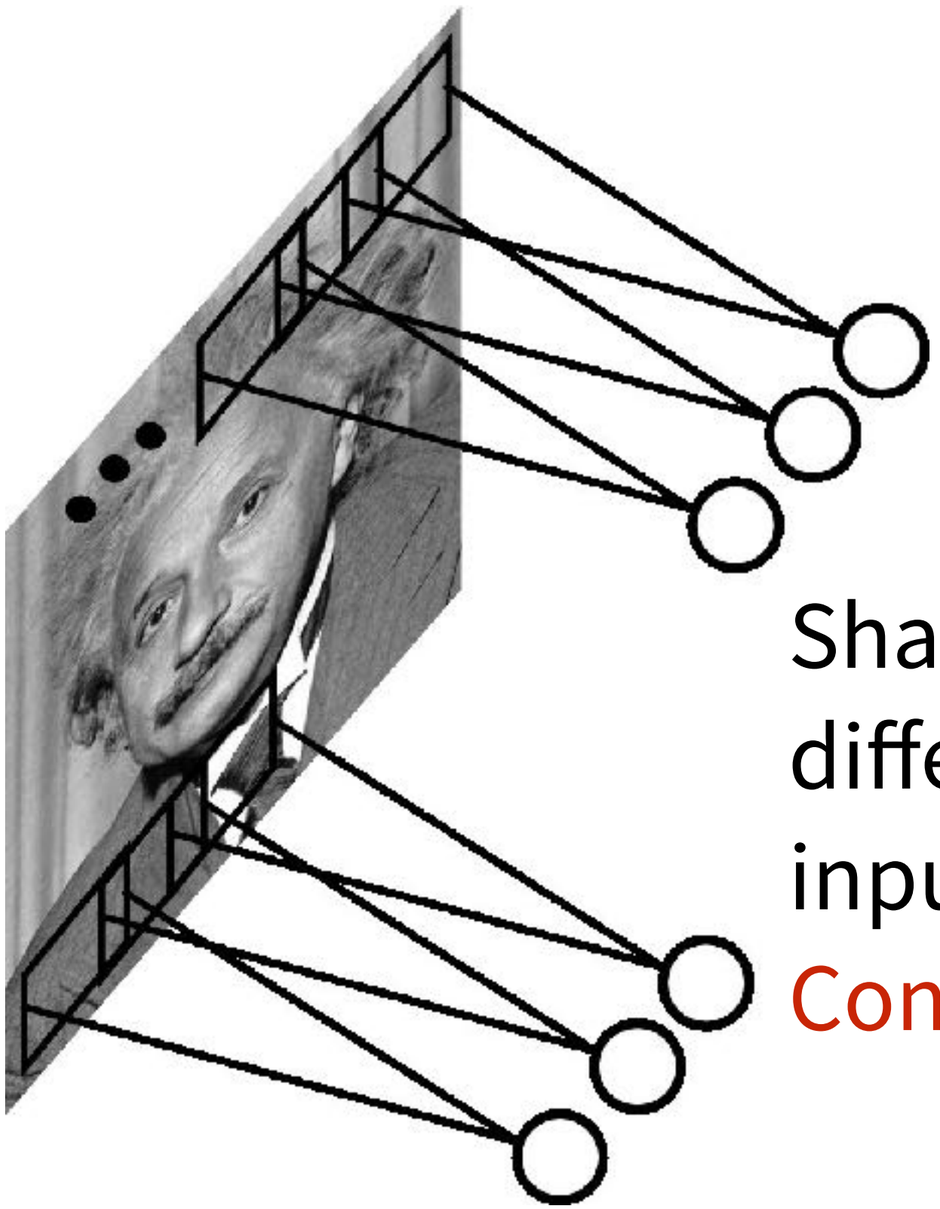
Filter size: 10 × 10

4M parameters

STATIONARITY? Statistics are similar at different locations.

Credit: Marc'Aurelio Ranzato

Weight Sharing



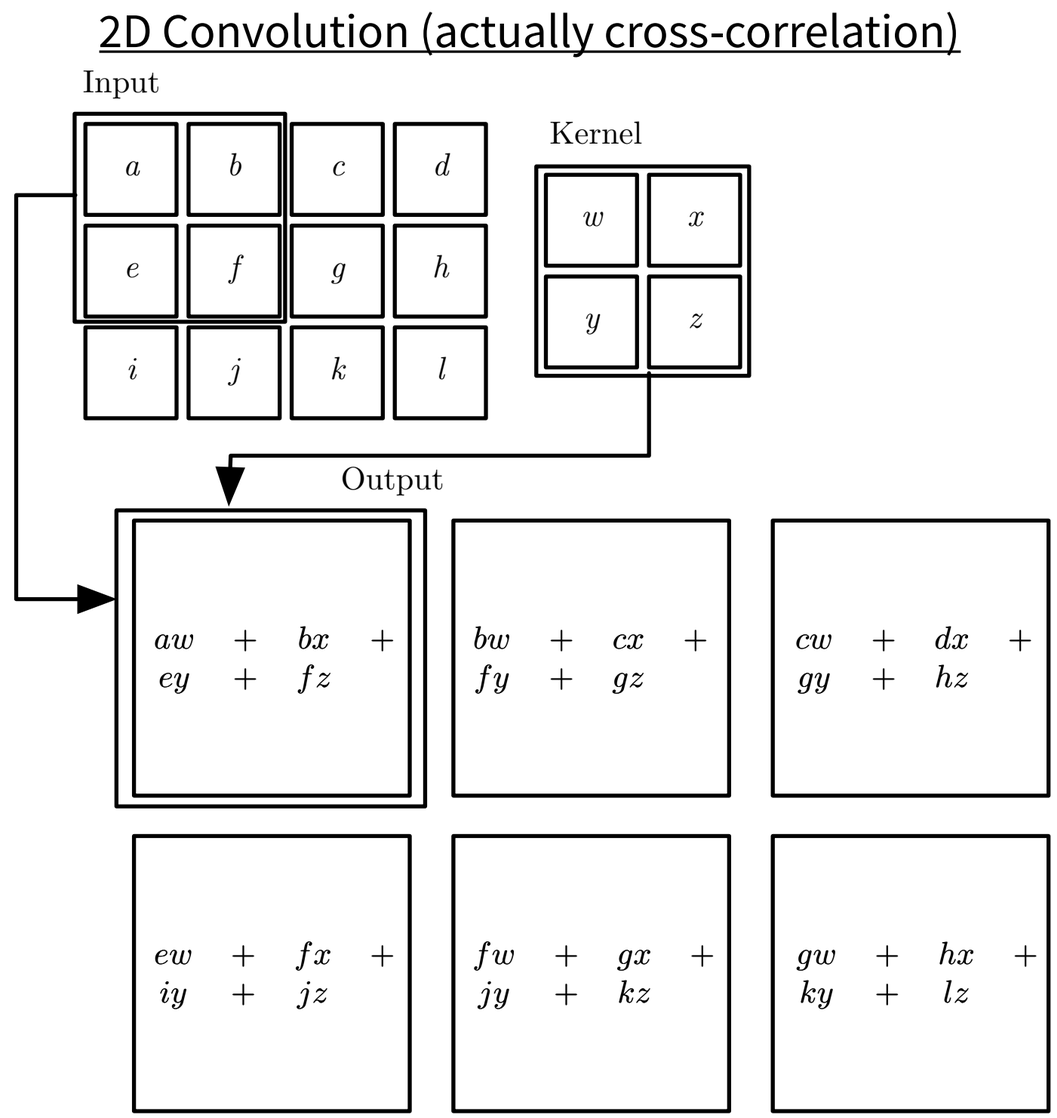
Share the same parameters across different locations (assuming input is stationary)

Convolutions with learned kernels

Credit: Marc'Aurelio Ranzato

What is Convolution?

- Convolution is a specialized kind of linear operation
- Can think of it as a weighted averaging operation in time or space
- For vision, we “convolve” two-dimensional convolution kernels with two-dimensional images:



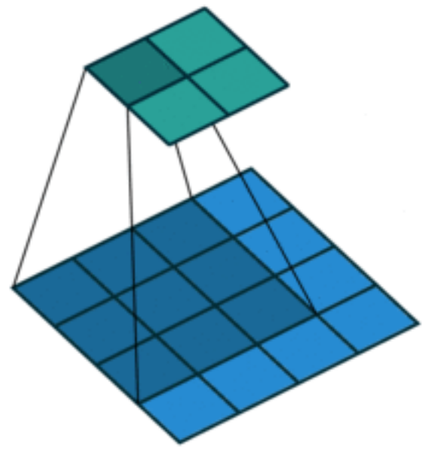
$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$$

Reproduced from Goodfellow et al.

Convolution (Animation)

- Note: hidden units are spatially organized into a “feature map”

- Each hidden unit only receives input from a local region of the input, its “receptive field”

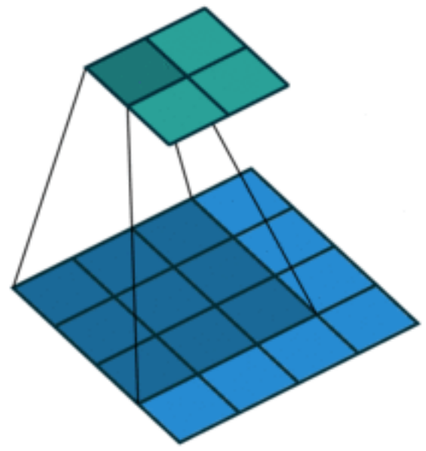


Animation from Vincent Dumoulin
https://github.com/vdumoulin/conv_arithmetic

Convolution (Animation)

- Note: hidden units are spatially organized into a “feature map”

- Each hidden unit only receives input from a local region of the input, its “receptive field”



Animation from Vincent Dumoulin
https://github.com/vdumoulin/conv_arithmetic

Convolutional Layer

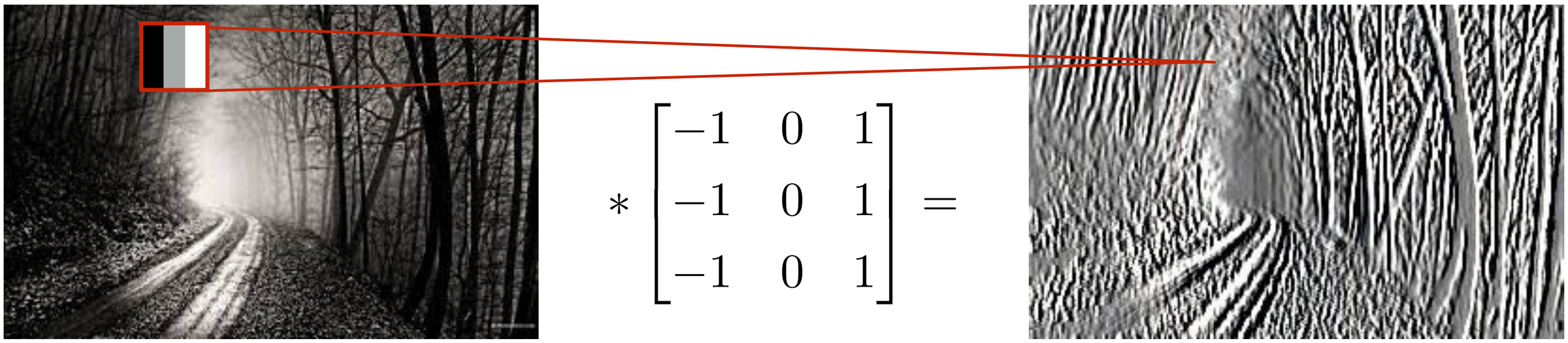


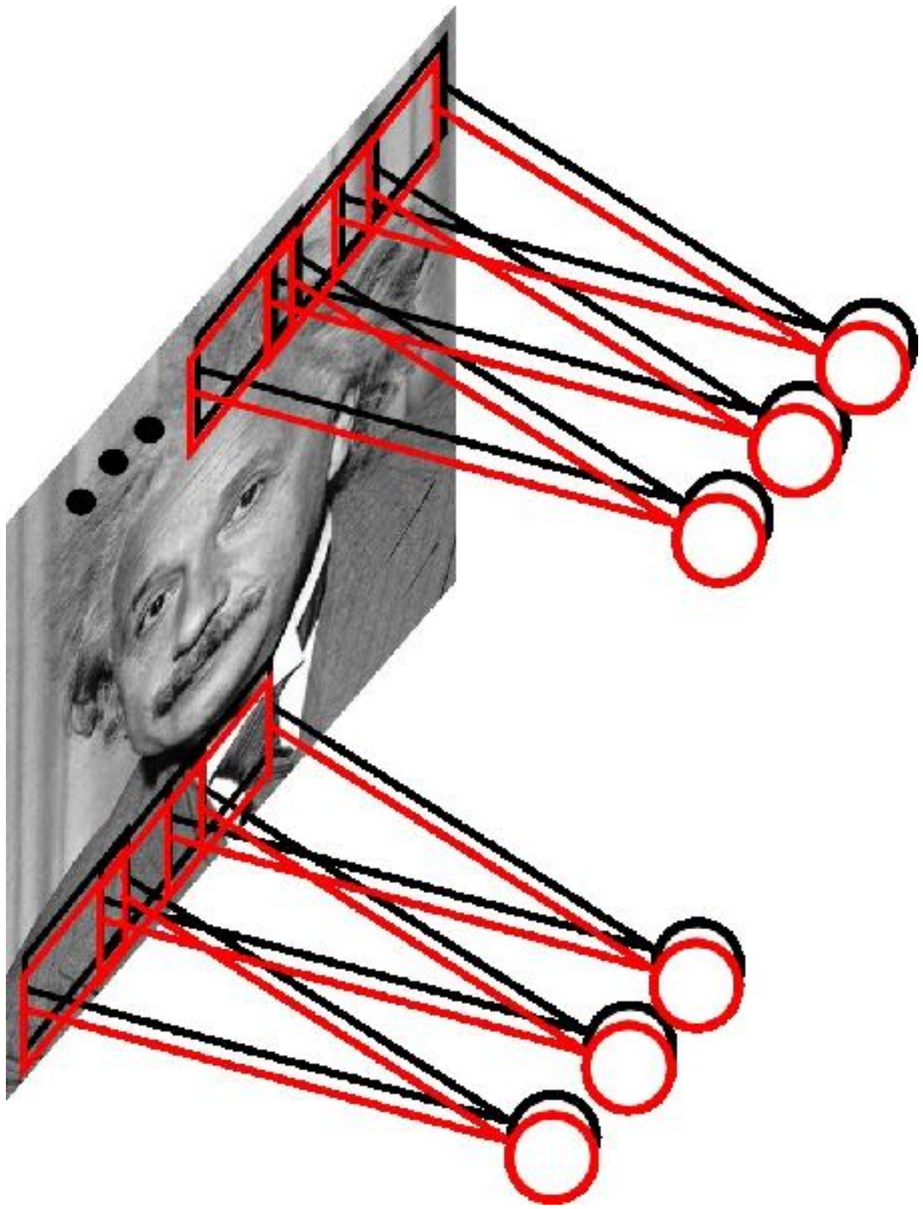
image (input)

convolution kernel

“feature map” (output)

Credit: Marc’Aurelio Ranzato

Convolutional Layer



Learn multiple filters.

Example: 200 × 200 image

100 filters

Filter size: 10 × 10

10k parameters

Credit: Marc'Aurelio Ranzato

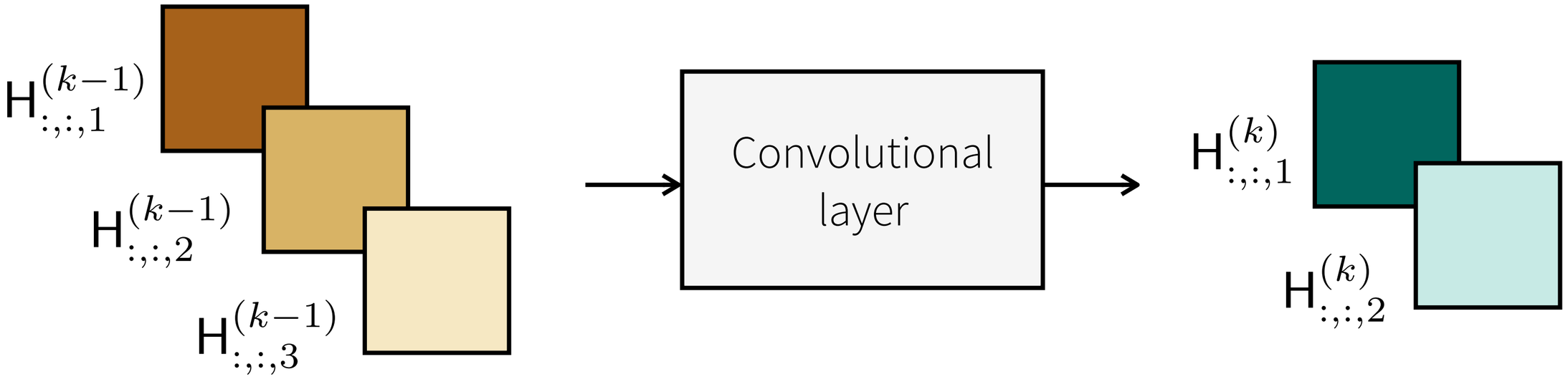
Convolutional Layer

$$\mathrm{H}^{(k)}_{:,:,i} = g\left(\sum_j \mathrm{H}^{(k-1)}_{:,:,j} * \mathrm{W}^{(k)}_{:,:,i,j}\right)$$

output feature map

input feature map

kernel



k indexes layers

i, j index maps (features)

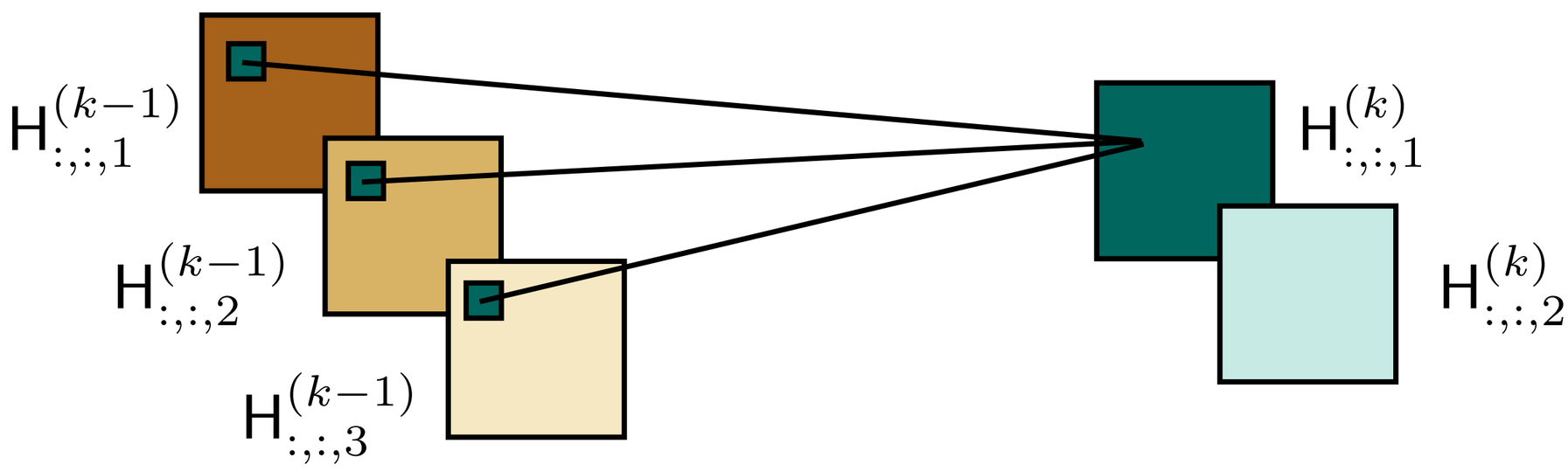
Convolutional Layer

$$\mathsf{H}^{(k)}_{:,:,i} = g\left(\sum_j \mathsf{H}^{(k-1)}_{:,:,j} * \mathsf{W}^{(k)}_{:,:,i,j}\right)$$

output feature map

input feature map

kernel



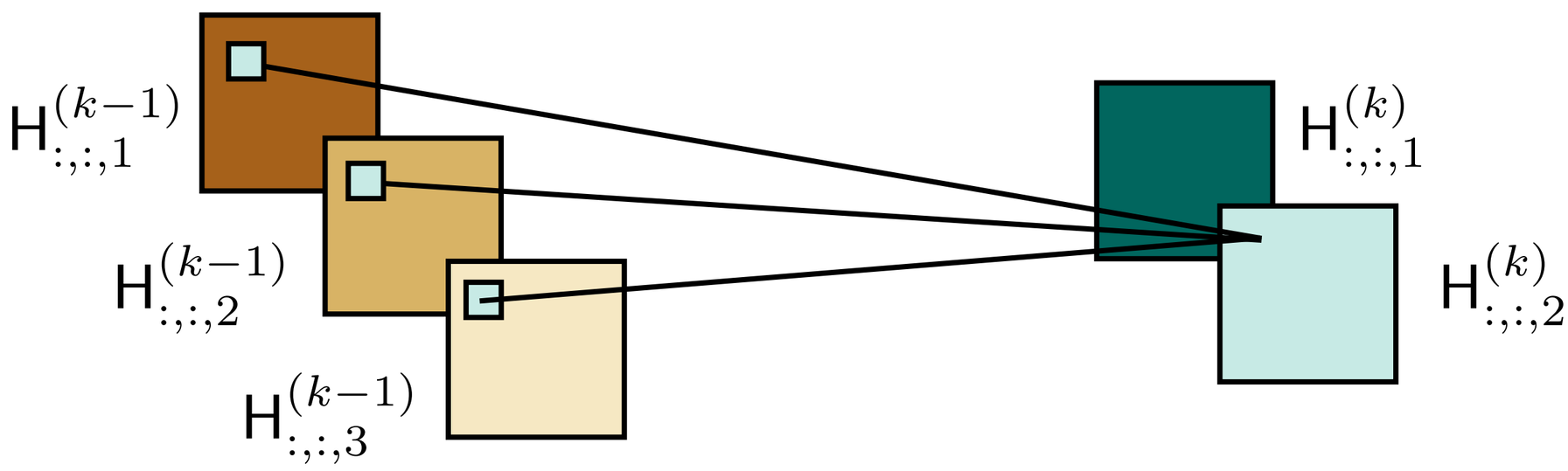
Convolutional Layer

$$\mathsf{H}^{(k)}_{:,:,i} = g\left(\sum_j \mathsf{H}^{(k-1)}_{:,:,j} * \mathsf{W}^{(k)}_{:,:,i,j}\right)$$

output feature map

input feature map

kernel

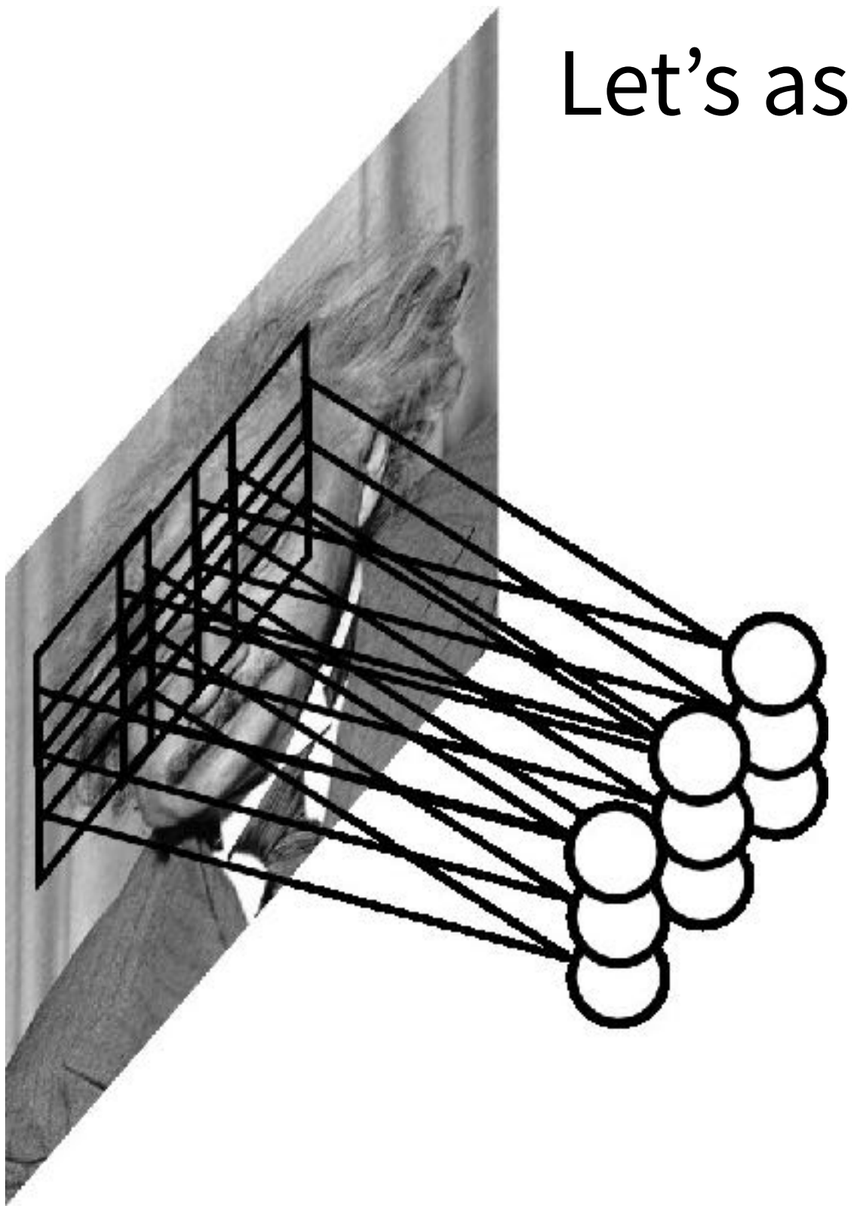


Convolutional Net - Recap

- A fully-connected neural network applied to images
  - scales quadratically with the size of the input
  - does not leverage stationarity
- Solution
  - connect each hidden to a small patch of input
  - share the weights across space
- This is called a convolutional layer
- A network with convolutional layers is called a convolutional net

Credit: Marc'Aurelio Ranzato

Pooling Layer

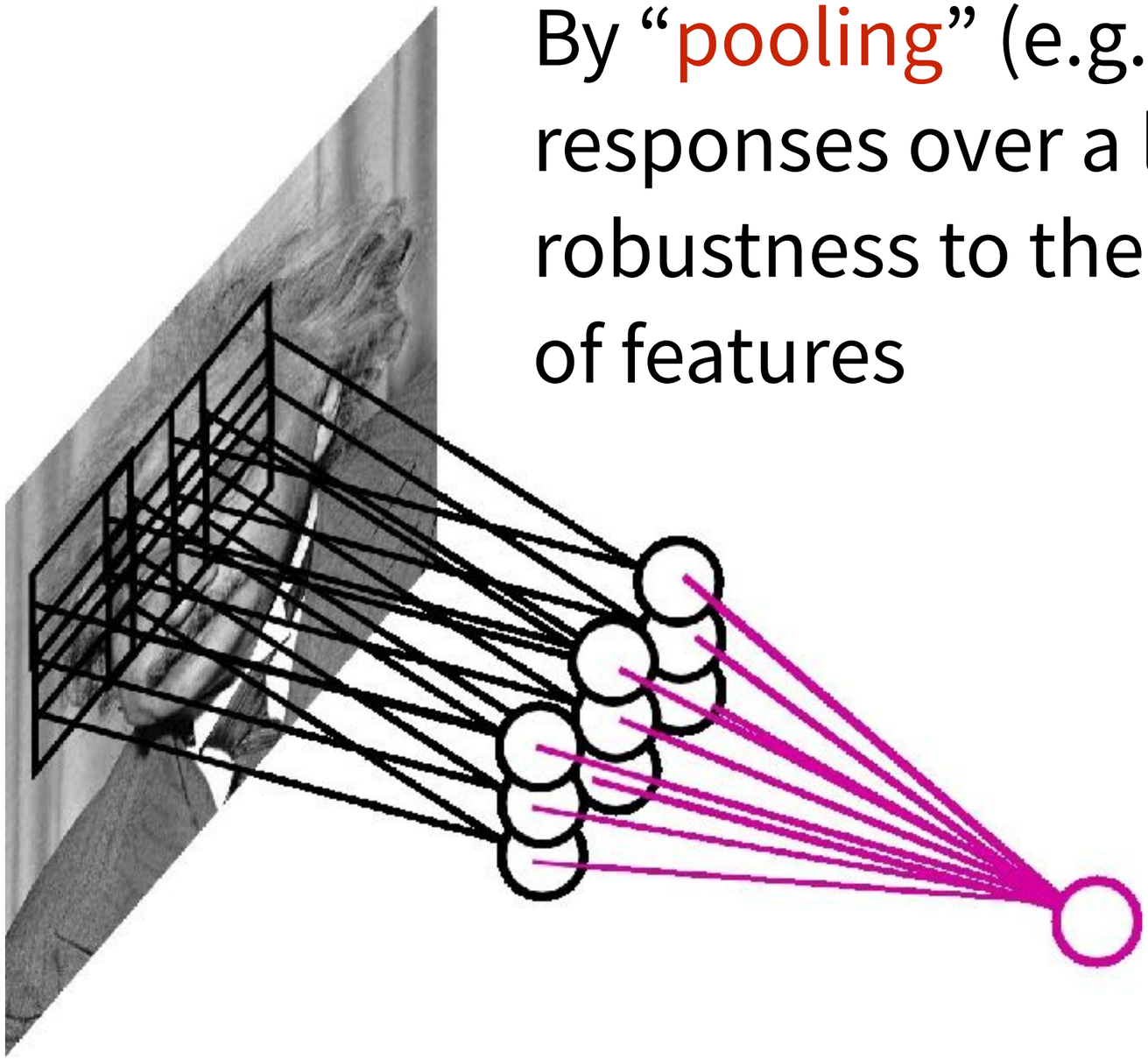


Let’s assume the filter is an “eye” detector

How can we make the detection robust to the exact location of the eye?

Credit: Marc’Aurelio Ranzato

Pooling Layer



By “pooling” (e.g. taking max) the filter responses over a local region we gain robustness to the exact spatial location of features

“Shift invariance”

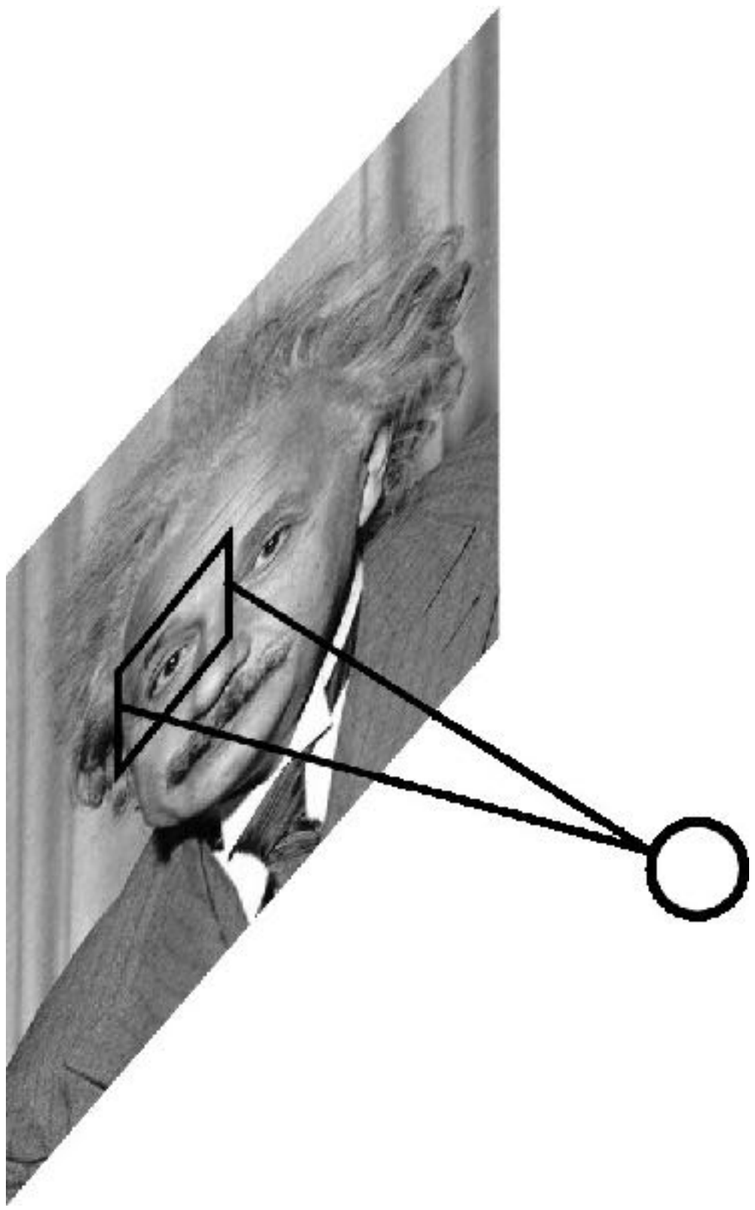
Credit: Marc’Aurelio Ranzato

Types of Pooling

Max-Pooling	$\mathsf{H}^{(k)}_{m,n,i} = \max_{m' \in \mathbb{N}(m), n' \in \mathbb{N}(n)} \mathsf{H}^{(k-1)}_{m',n',i}$
Average-Pooling	$\mathsf{H}^{(k)}_{m,n,i} = \frac{1}{\lvert\mathbb{N}(m)\rvert\lvert\mathbb{N}(n)\rvert} \sum_{m' \in \mathbb{N}(m), n' \in \mathbb{N}(n)} \mathsf{H}^{(k-1)}_{m',n',i}$
L2-Pooling	$\mathsf{H}^{(k)}_{m,n,i} = \sqrt{\sum_{m' \in \mathbb{N}(m), n' \in \mathbb{N}(n)} \left(\mathsf{H}^{(k-1)}_{m',n',i}\right)^2}$
L2-Pooling Over Features	$\mathsf{H}^{(k)}_{m,n,i} = \sqrt{\sum_{j \in \mathbb{N}(i)} \left(\mathsf{H}^{(k-1)}_{m,n,j}\right)^2}$

Credit: Marc'Aurelio Ranzato

Local Contrast Normalization



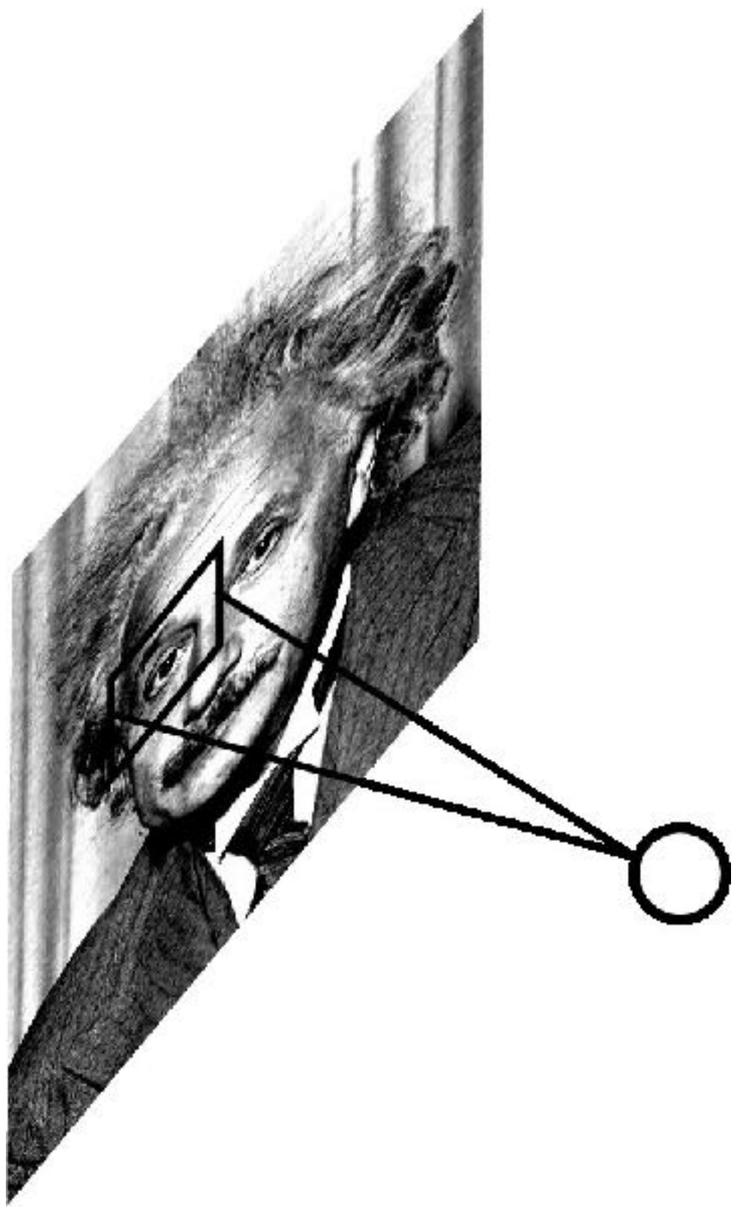
$$\mathsf{H}^{(k)}_{m,n,i} = \frac{\mathsf{H}^{(k-1)}_{m,n,i} - \mathsf{M}^{(k-1)}_{m,n,i}}{\mathsf{S}^{(k-1)}_{m,n,i}}$$

$\mathsf{M}^{(k)}_{m,n,i}$ mean of a local neighbourhood around pixels m, n

$\mathsf{S}^{(k)}_{m,n,i}$ standard deviation of a local neighbourhood around pixels m, n

Credit: Marc'Aurelio Ranzato

Local Contrast Normalization

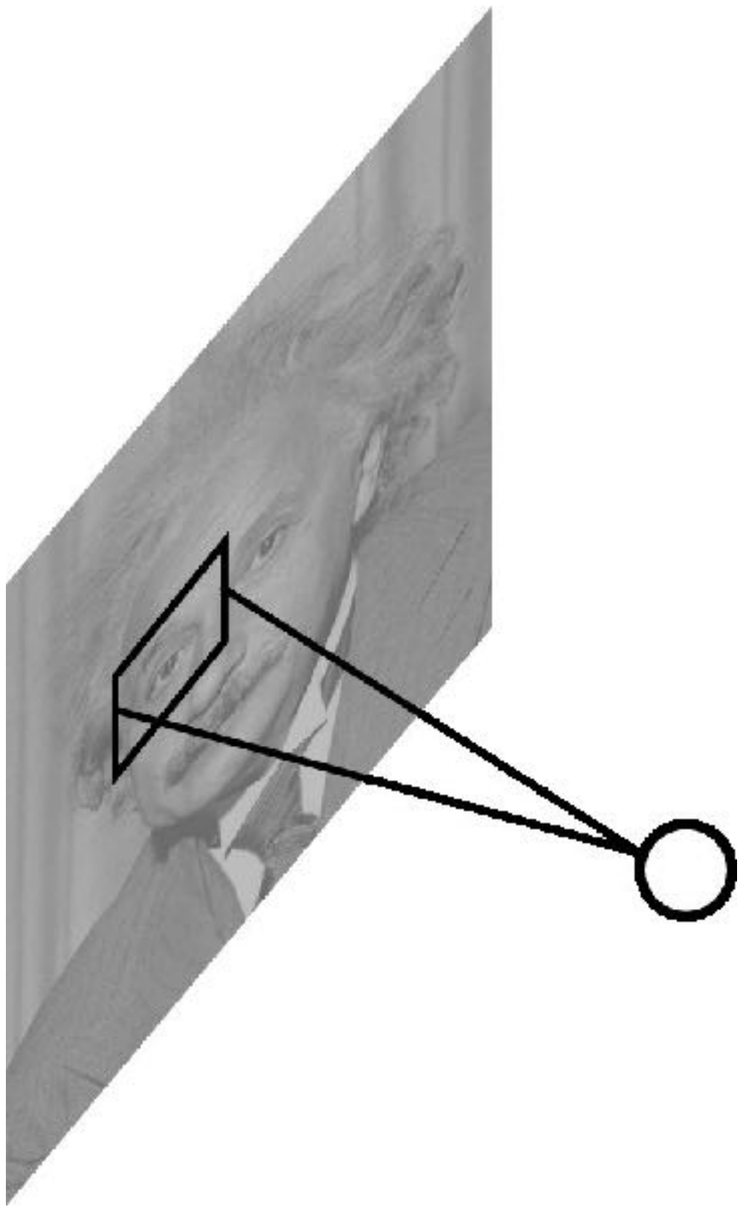


$$\mathsf{H}^{(k)}_{m,n,i} = \frac{\mathsf{H}^{(k-1)}_{m,n,i} - \mathsf{M}^{(k-1)}_{m,n,i}}{\mathsf{S}^{(k-1)}_{m,n,i}}$$

Controls the dynamic range of each feature map.

Credit: Marc'Aurelio Ranzato

Local Contrast Normalization



$$\mathsf{H}^{(k)}_{m,n,i} = \frac{\mathsf{H}^{(k-1)}_{m,n,i} - \mathsf{M}^{(k-1)}_{m,n,i}}{\mathsf{S}^{(k-1)}_{m,n,i}}$$

Performed also across features and in the higher layers…
Effects:

- improves invariance
- improves optimization
- increases sparsity

Note: computational cost is negligible compared to convolutional layer.

Credit: Marc'Aurelio Ranzato

Convnets: Single Stage

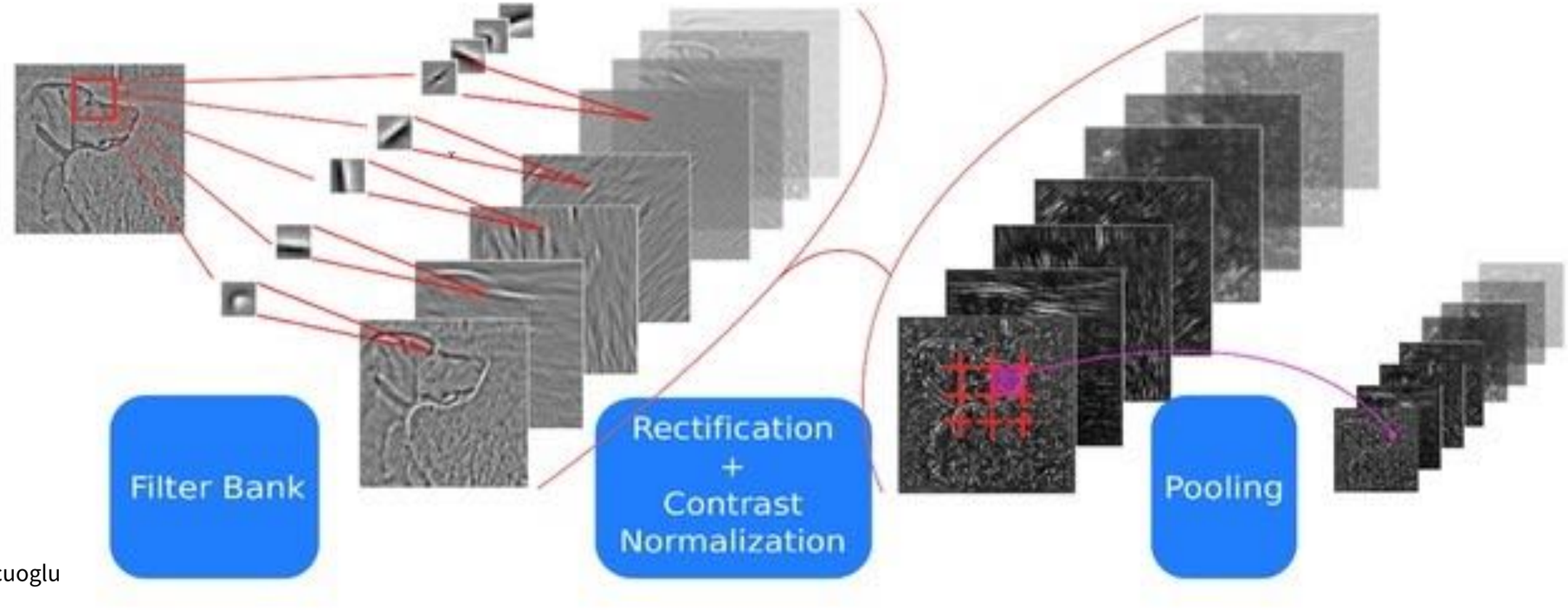


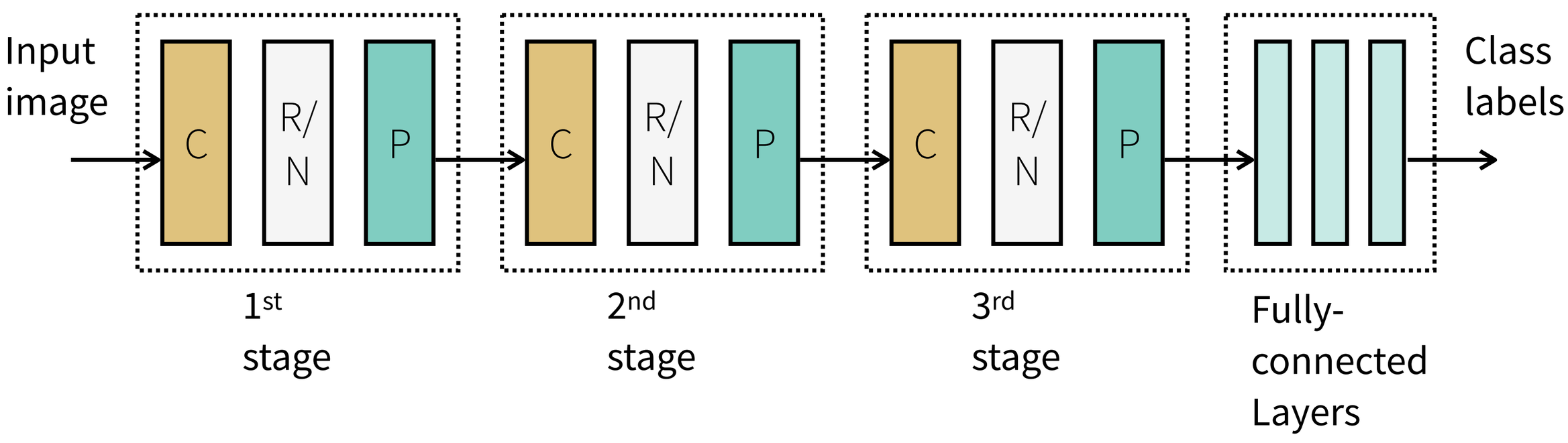
Image credit: Koray Kavukcuoglu

Credit: Marc'Aurelio Ranzato

Convnets: Typical Architecture

Single stage

Whole system



Credit: Marc'Aurelio Ranzato

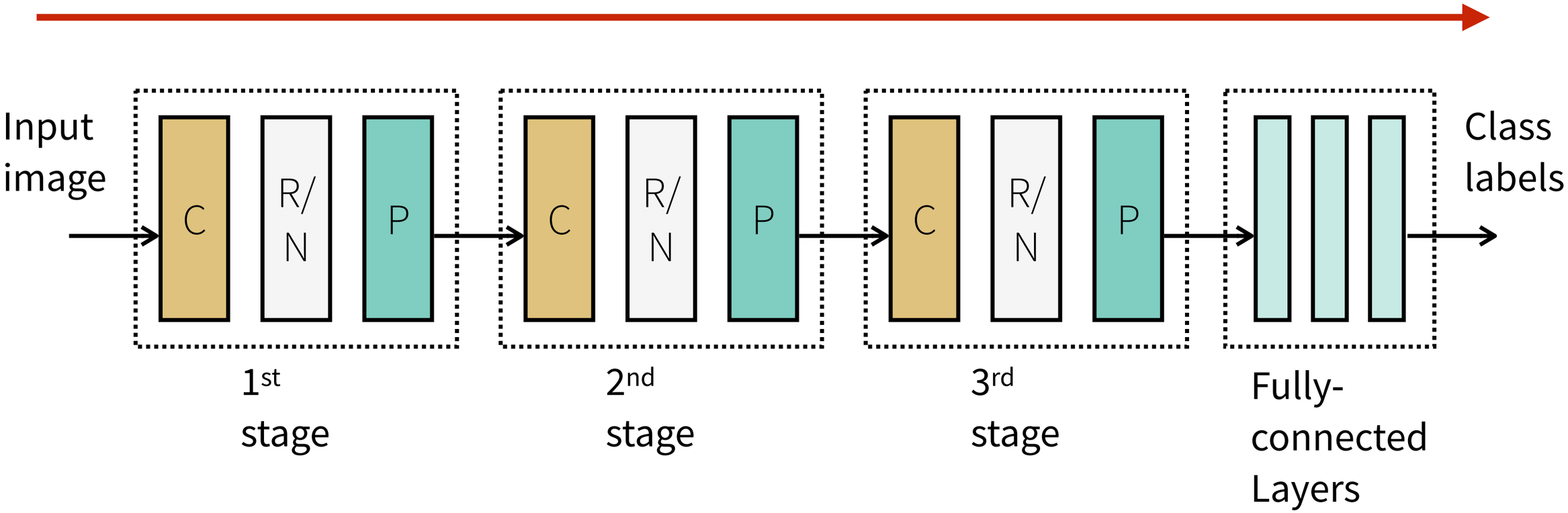
Convnets: Training

- All layers are differentiable
- We can use standard back-propagation
- Algorithm:
  - Given a small mini-batch:
    - F-Prop
    - B-Prop
    - Parameter updates

Credit: Marc'Aurelio Ranzato

Convnets: Testing

At test time, only run forward propagation



Convnets can naturally process larger images at little cost.
Traditional methods use inefficient sliding windows.

Credit: Marc'Aurelio Ranzato

Convnets: today

- Until 2012, the common wisdom was that training didn't work because we would "get stuck in local minima"
- Local minima are all similar, there are long plateaus, and it can take a long time to break symmetries
- Optimization is not the real problem, when:
  - the dataset is large
  - units do not saturate much
  - we use normalization layers

Credit: Marc'Aurelio Ranzato

Scalability

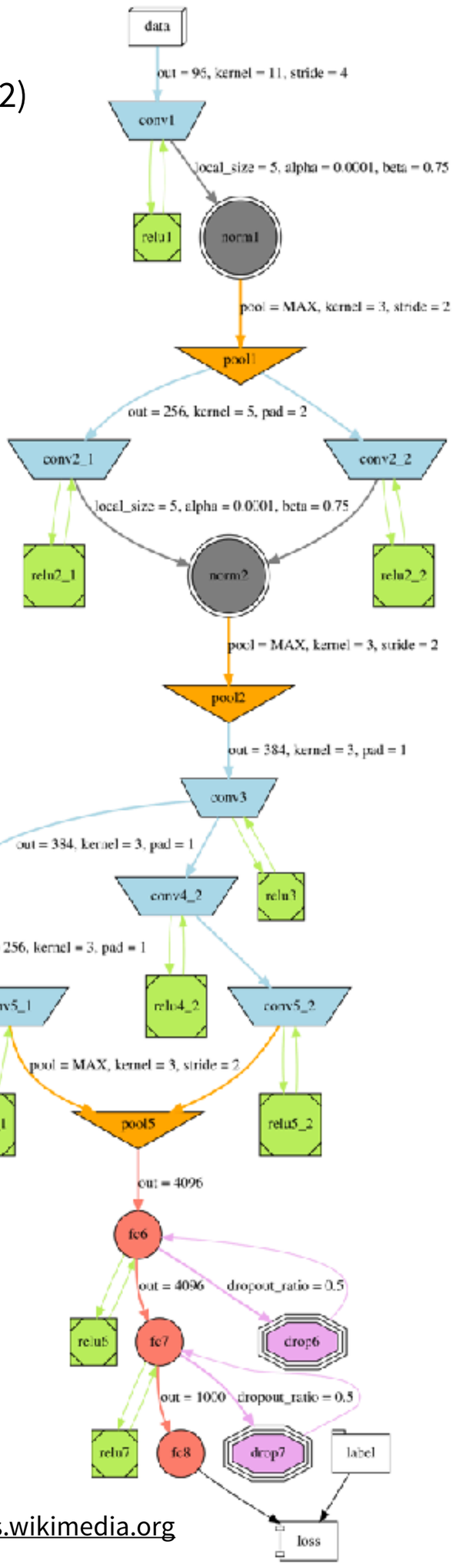
- The real challenges are:
  - Generalization
    - How many training examples to fit 1B params?
    - How many parameters/samples to model spaces with 1M dimensions?
  - Scalability

Credit: Marc'Aurelio Ranzato

Profile: AlexNet

(Krizhevsky et al. 2012)

- First work to popularize DNNs in vision
- Astonishing result on ILSVRC 2012 (16% top-5 error compared to 26% for runner-up)
- Similar to early LeNet architectures but deeper, bigger, and stacked convolutional layers (without pooling)
- Uses dropout regularization
- Optimized for 2 GPUs



via commons.wikimedia.org

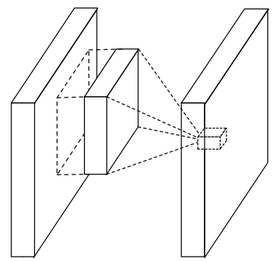
(Lin et al. 2013)

Profile: Network-in-Network

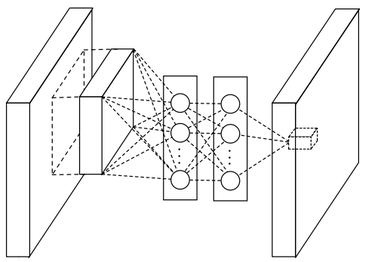
Not an ILSVRC competitor, this paper proposed a number of novel ideas which were utilized by later architectures:

- Replaces linear convolution layer with a “micro network” structure which is a general nonlinear operation
- Noting that fully connected layers are prone to overfitting, proposed “Global Average Pooling”

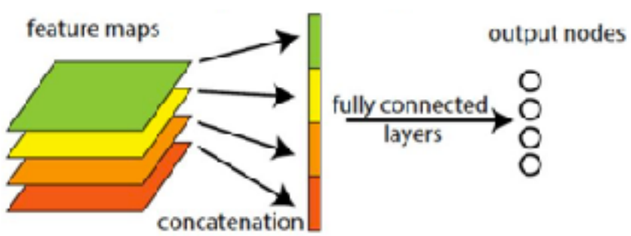
Linear convolution layer



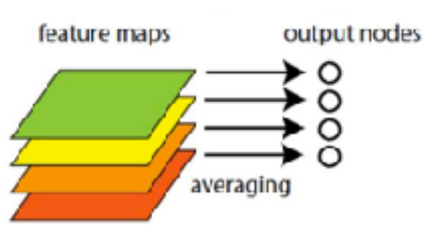
Mlpconv layer



CNN



GAP

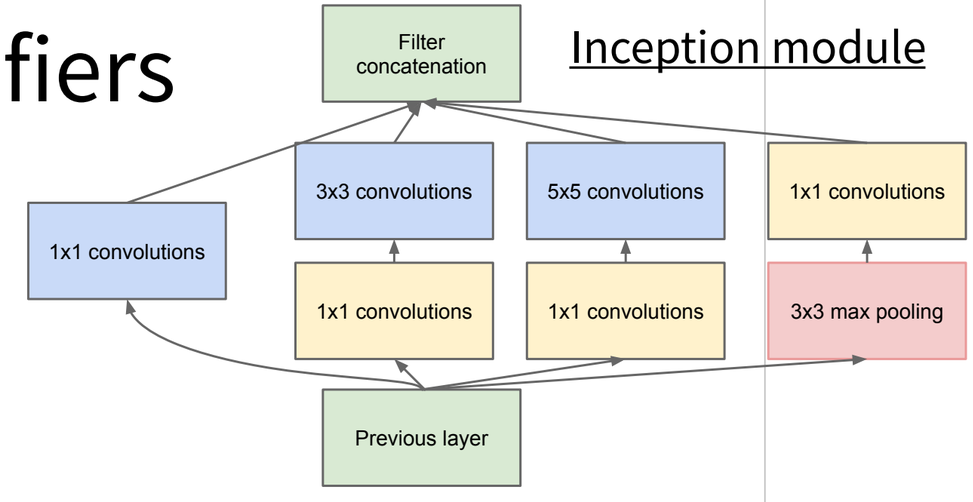


via Lin et al., caffecn.cn

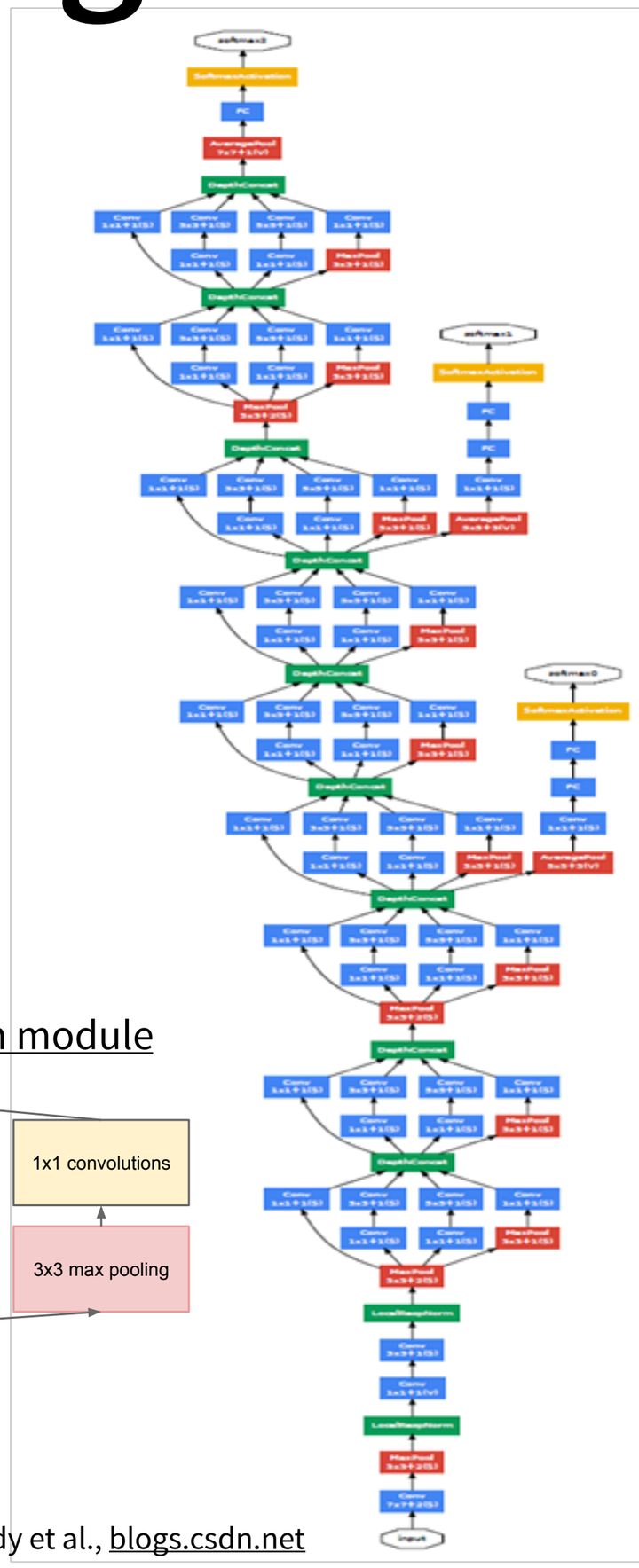
(Szegedy et al. 2015)

Profile: Inception/GoogLeNet

- Winner of ILSVRC 2014
- Main contribution is the “Inception Module” that dramatically reduced the number of parameters in the network (4M compared to 60M for AlexNet)
- Uses auxillary classifiers



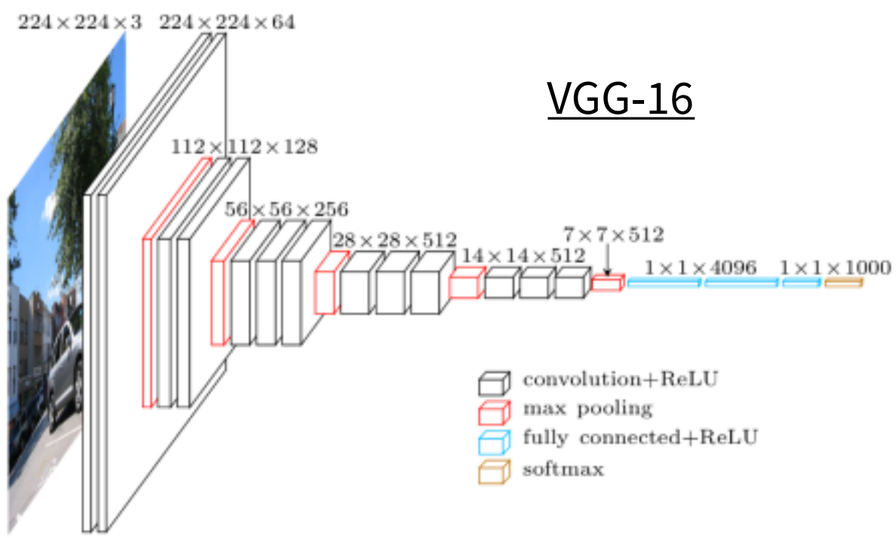
Inception module



via Szegedy et al., blogs.csdn.net

Profile: VGGNet (Simonyan and Zisserman 2014)

- Runner-up to ILSVRC 2014
- Main contribution: showing that depth of the network is a critical component for good performance (16 CONV/FC layers)
- Extremely homogeneous architecture that only performs 3x3 convolutions and 2x2 pooling from beginning to end
- Expensive to evaluate, uses a lot of memory and has 140M parameters



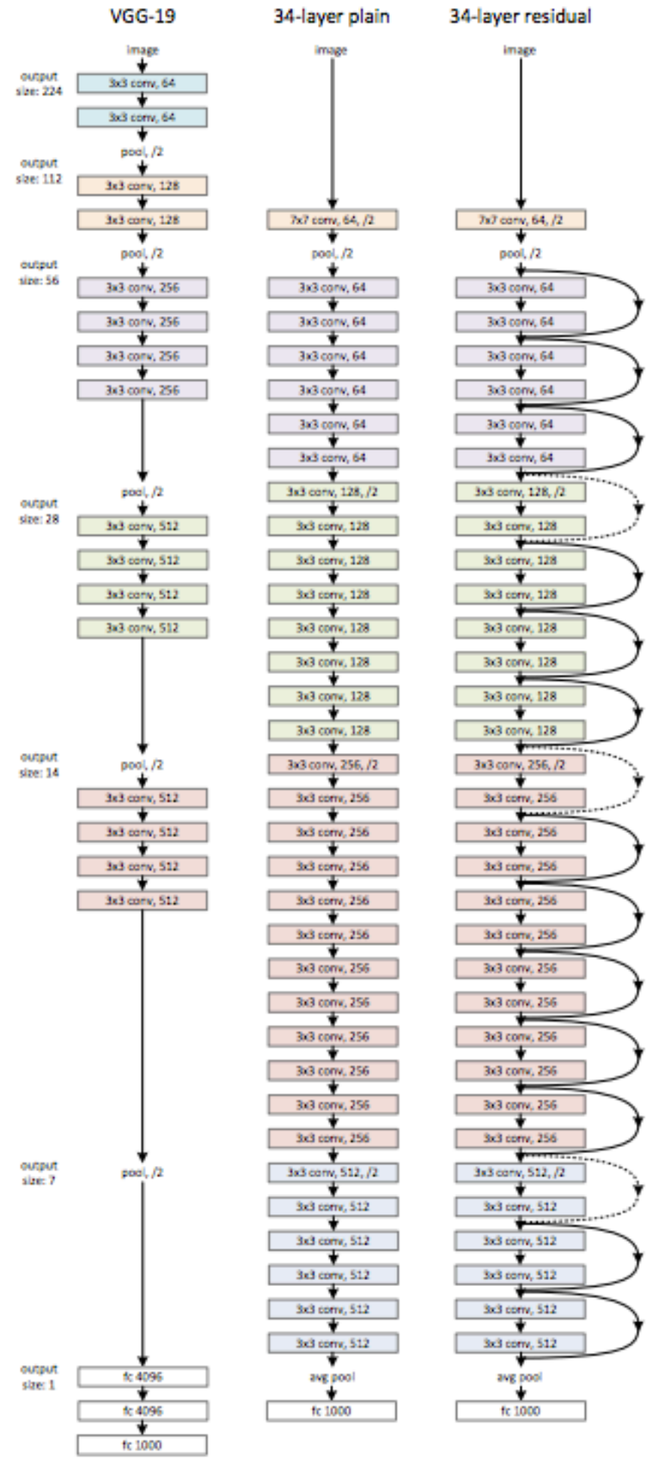
VGG-16

via Alexis Cook

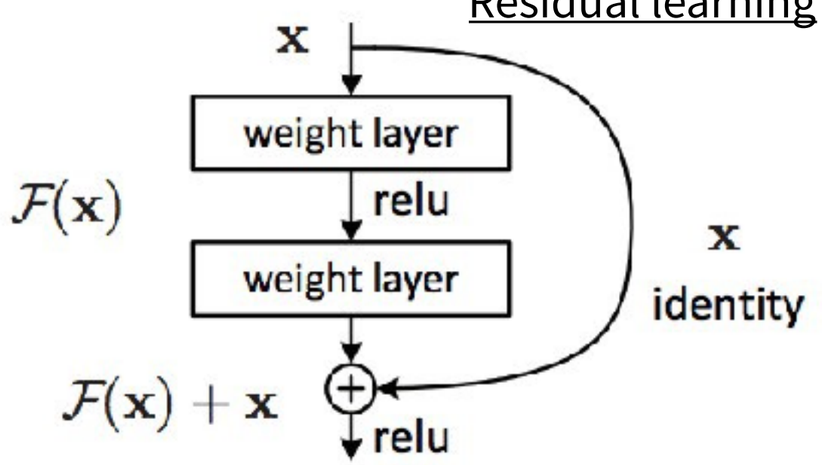
Profile: ResNet

(He et al. 2015)

- Winner of ILSVRC 2015
- It features special skip connections and heavy use of batch normalization
- The architecture is also missing fully connected layers at the end of the network (note use of GAP)
- Winning network had 152 layers; 1000-layer models have been trained

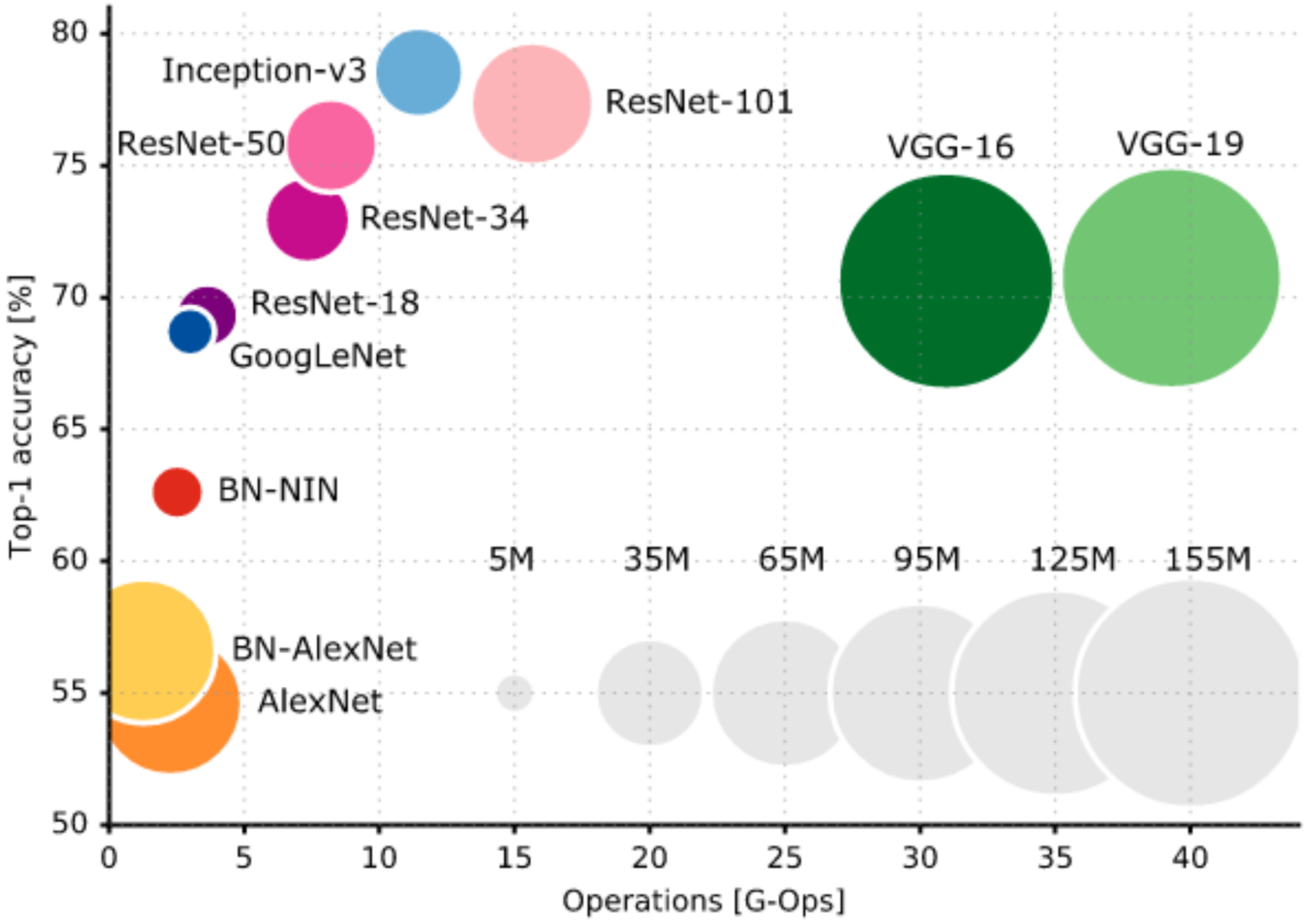


Residual learning



via He et al.

Modern Architectures



Eugenio Culurciello's blog:
https://culurciello.github.io/tech/2016/06/04/nets.html