Università degli Studi di Pavia

Deep Learning and TensorFlow

Episode 3
Deep Convolutional Neural Networks

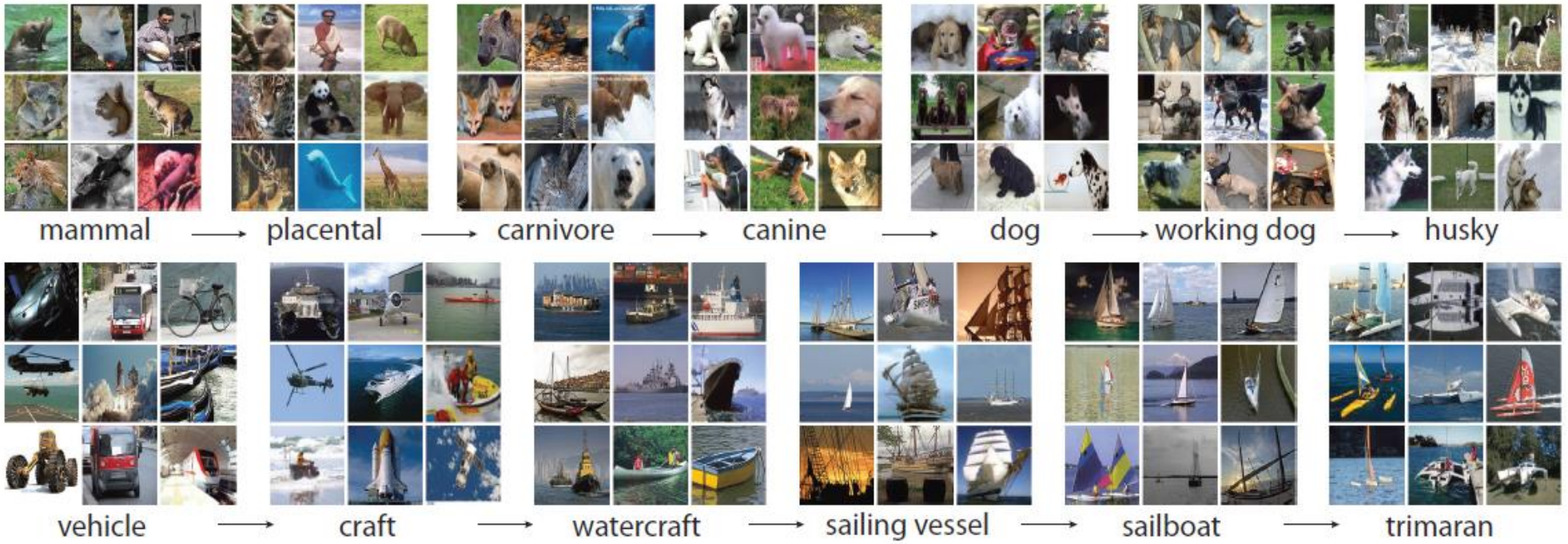
Marco Piastra

This presentation can be downloaded at:
http://vision.unipv.it/AI/AIRG.html

The storm of Deep Convolutional Neural Networks (DCNN)

ImageNet Challenge

- The ImageNet Large Scale Visual Recognition Challenge



1,461,406 full resolution images

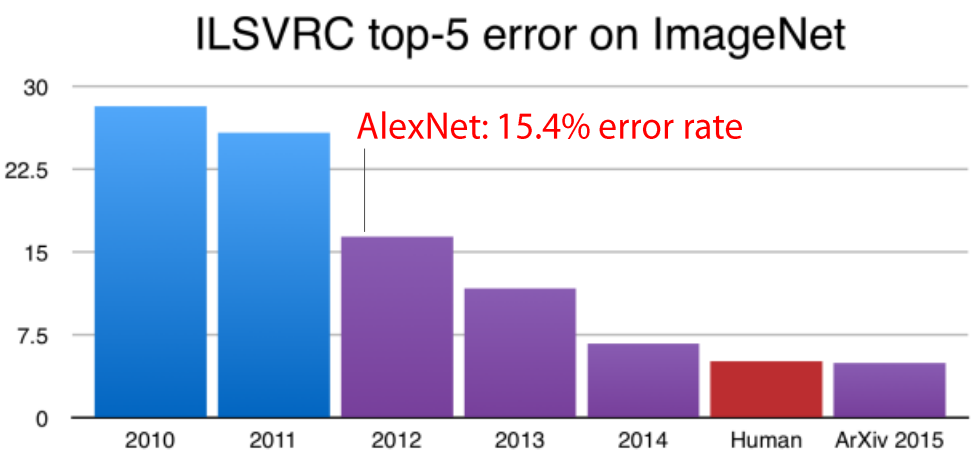
Complex and multiple textual annotation,
hierarchy of 1000 object classes along several dimensions

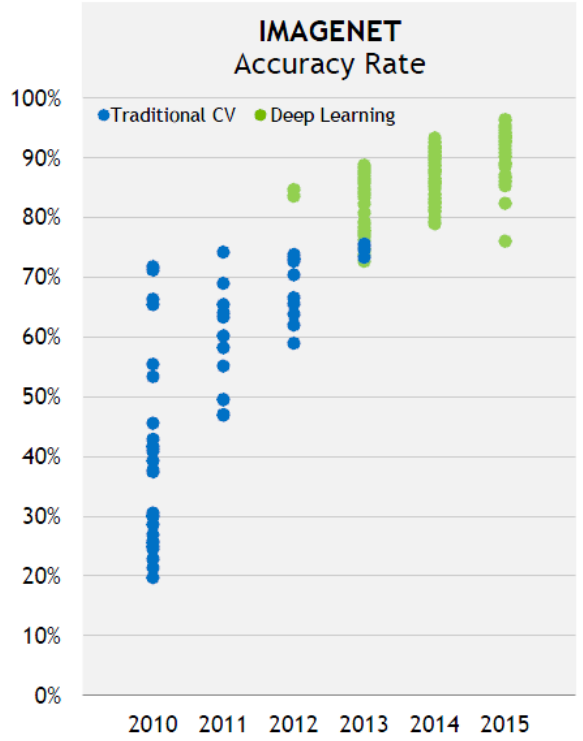
The image classification challenge is run annually since 2010

[figures from www.nvidia.com]

ImageNet Challenge

- The ImageNet Large Scale Visual Recognition Challenge





1,461,406 full resolution images

Complex and multiple textual annotation,
hierarchy of 1000 object classes along several dimensions

The image classification challenge is run annually since 2010

[figures from www.nvidia.com]

The Mother of all DCNNs

Deep Convolutional Neural Network (DCNN)

- ***AlexNet*** [Krizhevsky, Sutskever & Hinton, 2012]

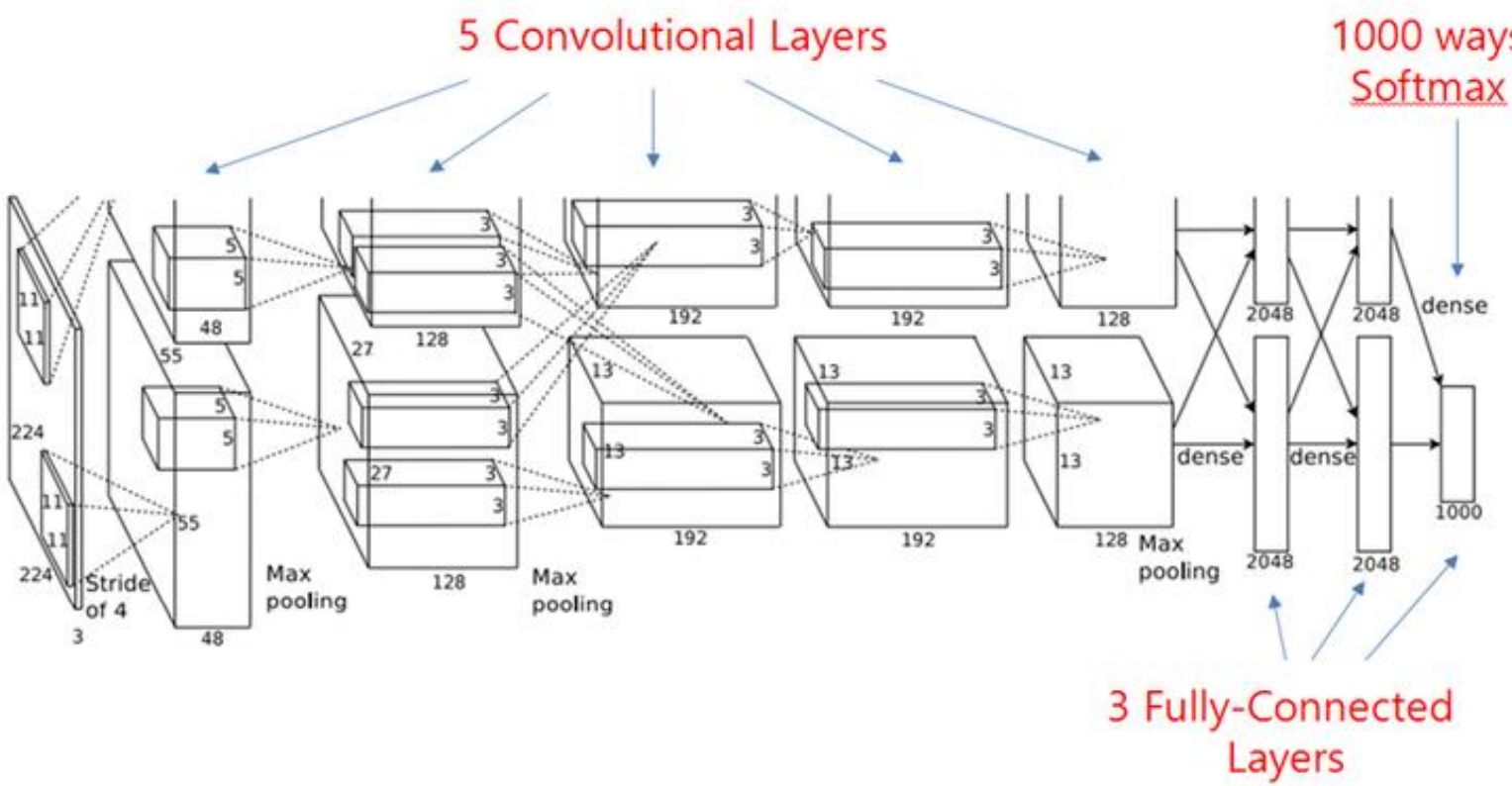


Image from [Krizhevsky, Sutskever & Hinton, 2012]

The Mother of all DCNNs

Deep Convolutional Neural Network (DCNN)

- ***AlexNet*** [Krizhevsky, Sutskever & Hinton, 2012]

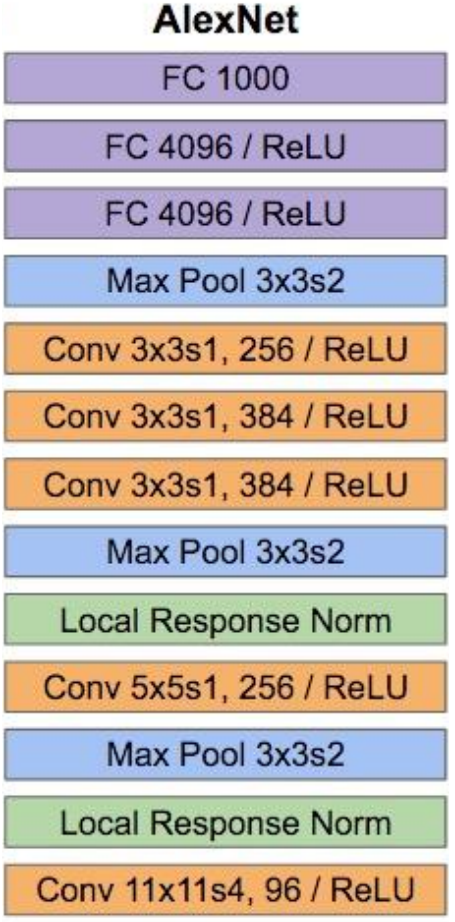
This layer is ***Softmax***

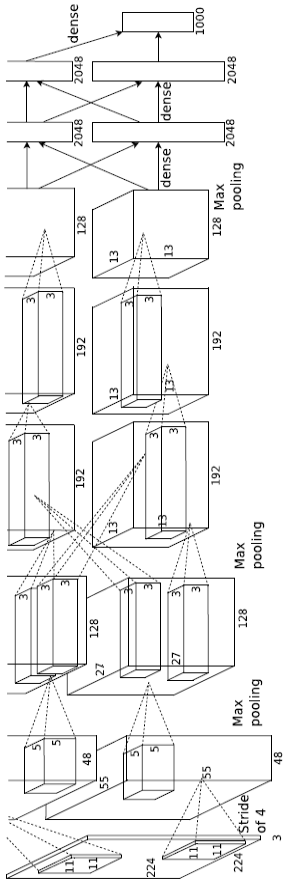
These layers are ***fully connected (FC)*** FF neural networks

This layer is ***Max Pooling***

These layers are ***convolutional*** with ***ReLU*** as non-linearity

This layer is ***Local Response Normalization (LRN)***





DCNN Building Blocks (layerwise)

Convolutional Layer

- ***Convolution operation***

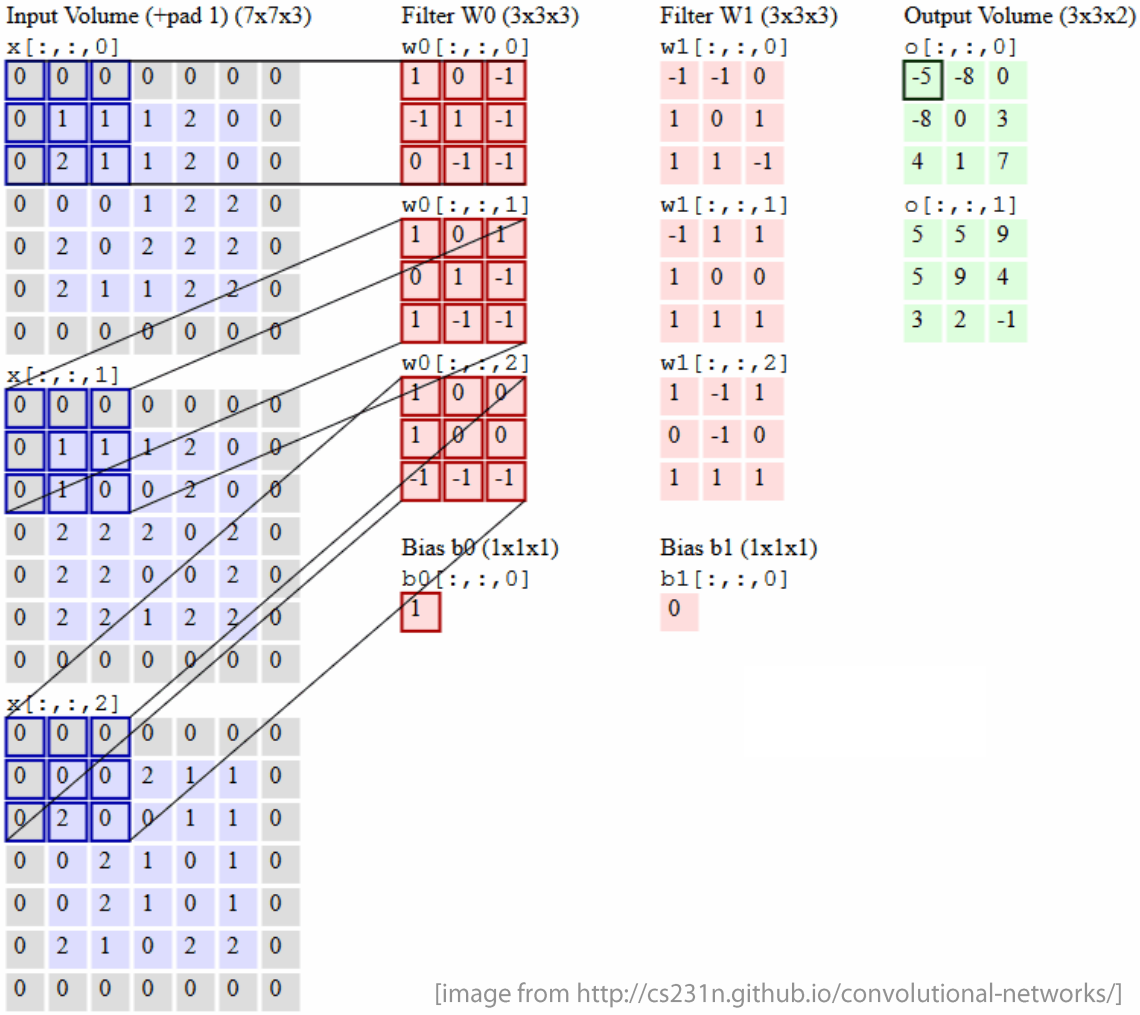
  A ***convolution filter***
  is a square (or cubic) matrix

  It is first centered on a pixel of the input image

  It produces a scalar value: the dot product between the filter and the image region around the pixel

  By mapping the same procedure on all pixels of the input image,

  a new image is produced (i.e. a *feature map*)



[image from http://cs231n.github.io/convolutional-networks/]

Convolutional Layer

- **Convolution operations (on images)**

A ***convolution filter***
is a square (or cubic) matrix

In symbols

$$\boldsymbol{Y}_i := \boldsymbol{W}_i * \boldsymbol{X}$$

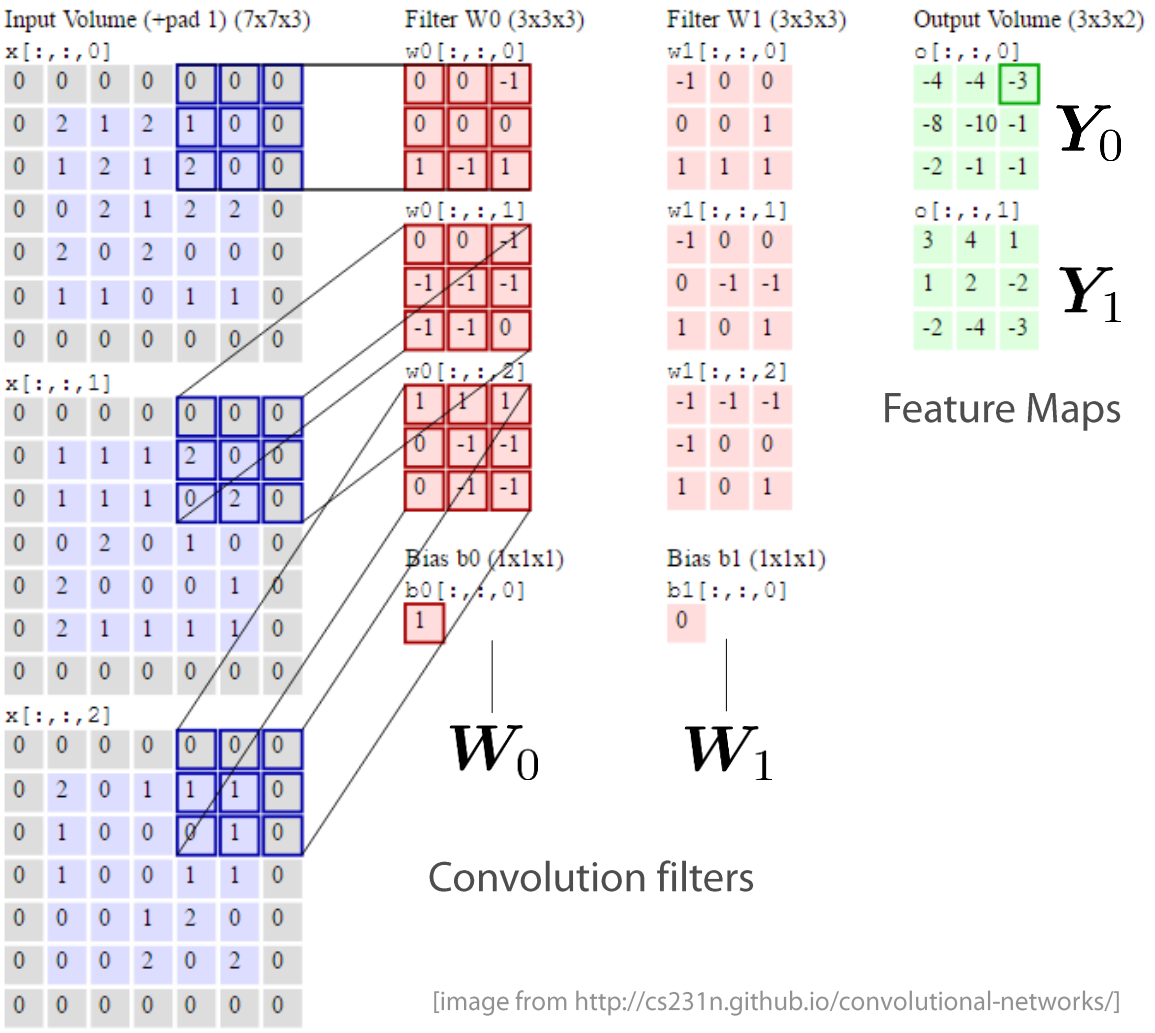
i-th feature map

convolution operator

where:

$\boldsymbol{X}$

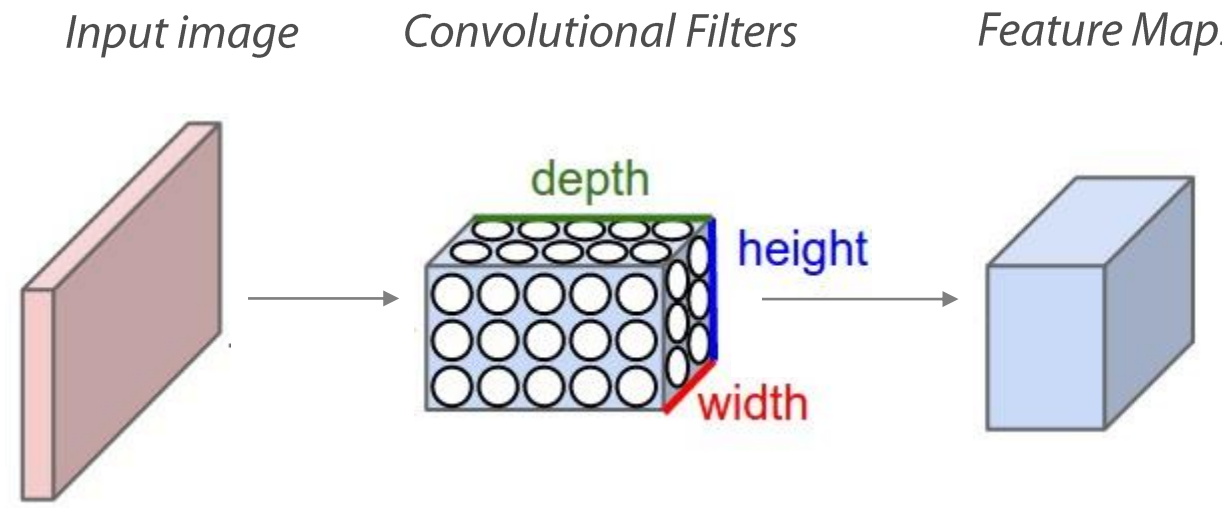
Input image (e.g. RGB)



[image from http://cs231n.github.io/convolutional-networks/]

Convolutional Layer

- **Convolution network (first layer)**



$\boldsymbol{X}$ $[\boldsymbol{W}_0, \cdots, \boldsymbol{W}_h]$ $[\boldsymbol{Y}_0, \cdots, \boldsymbol{Y}_h]$

[image adapted from http://cs231n.github.io/convolutional-networks/]

Convolutional Layer

- ***Convolution operation with non-linearity***

  The linear form for convolution

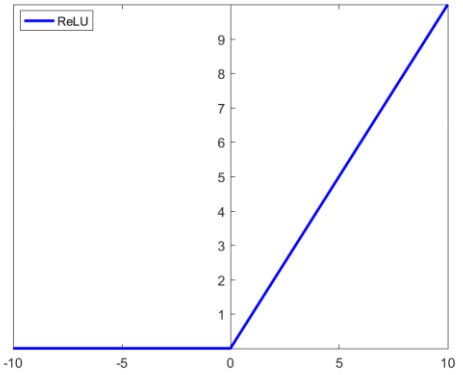
  $$\boldsymbol{Y}_i := \boldsymbol{W}_i * \boldsymbol{X}$$

  in actual networks is composed with a non-linearity

  $$\boldsymbol{Y}_i := \mathrm{ReLU}(\boldsymbol{W}_i * \boldsymbol{X})$$

  Applied elementwise to all matrix components

$$y = \max(0, x)$$



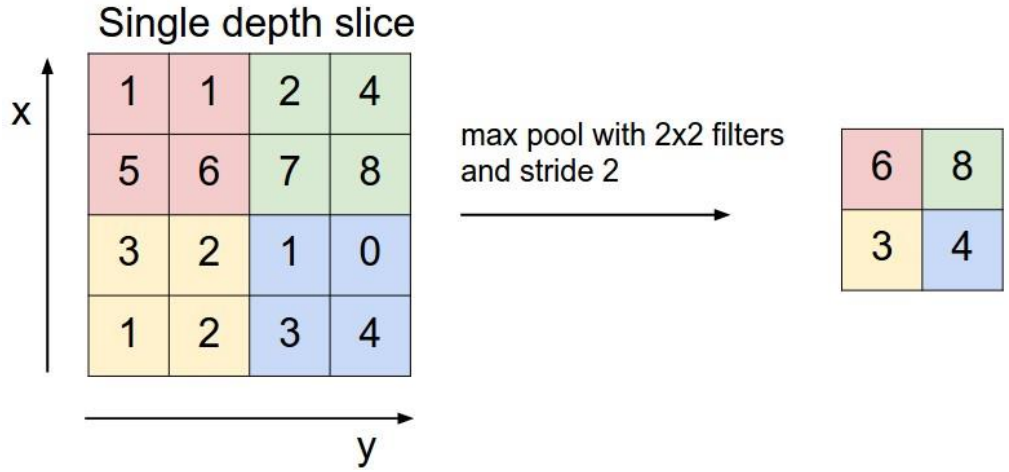
ReLU

Why ReLU?
To be seen later on, when discussing training

Max Pooling Layer

- ***Max Pooling operation***

  Returns the maximum value in a pre-defined region of its input

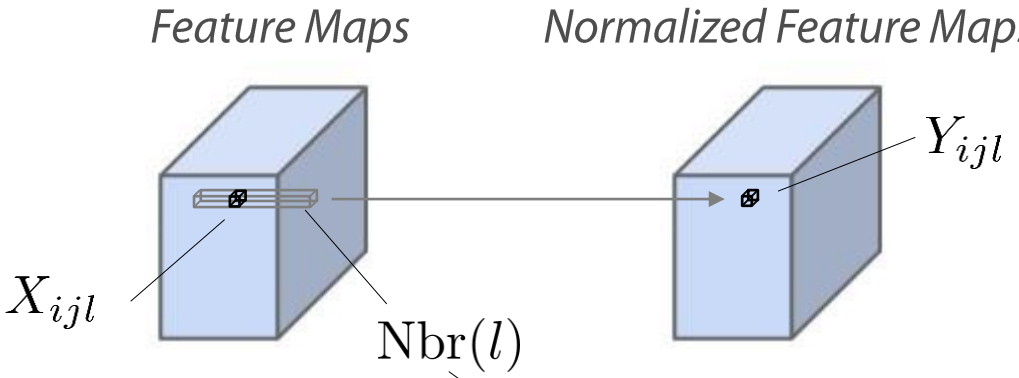


Local Response Normalization Layer

- ***Local Response Normalization (LRN)***

Rationale:

compensating the tendency of ReLU to produce large values in output



Two variants:

- *across feature* maps
  (i.e. as in figure)
- *within feature* map
  (i.e. with neighboring pixels)

$$[\boldsymbol{X}_0, \cdots, \boldsymbol{X}_h] \qquad [\boldsymbol{Y}_0, \cdots, \boldsymbol{Y}_h]$$

$$\boldsymbol{X}_l := [X_{ijl}] \qquad \boldsymbol{Y}_l := [Y_{ijl}]$$

$$Y_{ijl} := \frac{X_{ijl}}{\left(a + \alpha \sum_{k \in \mathrm{Nbr}(l)} (X_{ijk})^2\right)^{\beta}}$$

where a, α, β are fixed *hyperparameters*

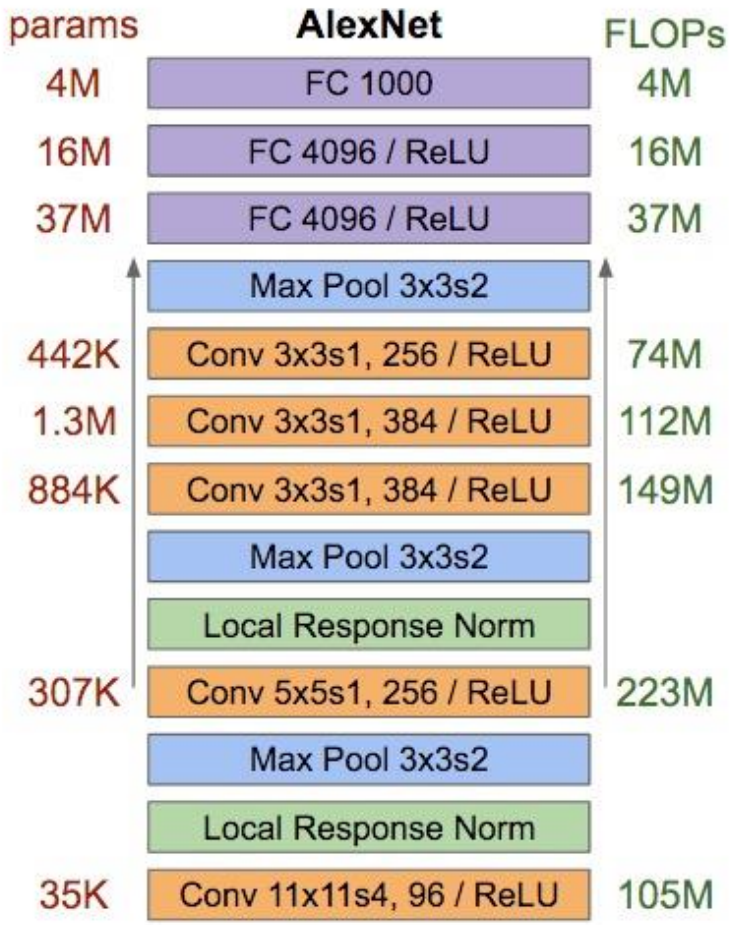
AlexNet Architecture

- ***AlexNet*** [Krizhevsky, Sutskever & Hinton, 2012]
  - number of parameters, per layer
    in red on the left
  - number of floating point operations, (FLOP) per layer in single forward pass
    in green on the right

Higher layers have more parameters
but the bulk of the computation
takes place at lower layers

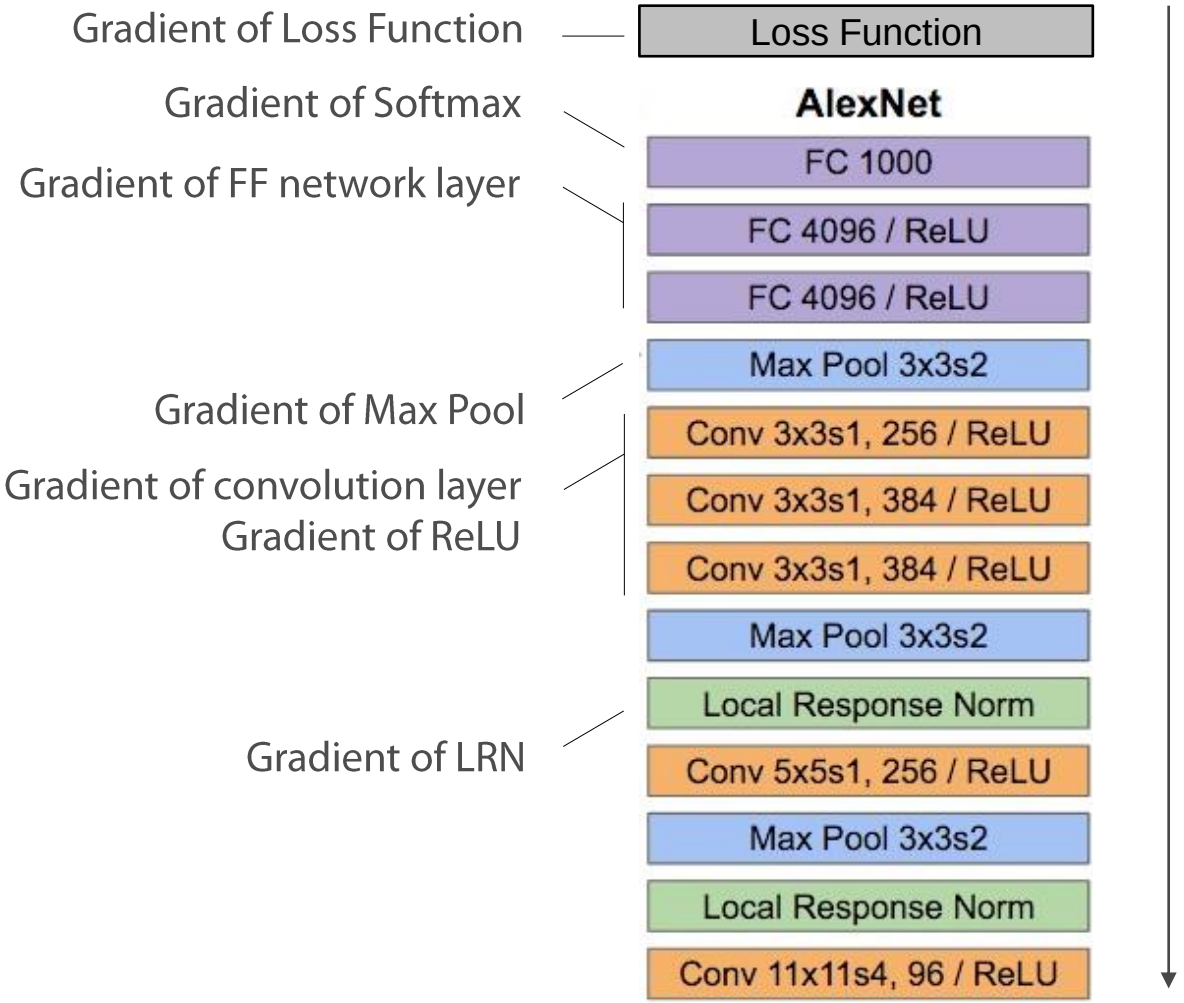
Totals:

- around 60M parameters
- around 837M FLOPs for a single pass



AlexNet Gradient

- ***Computing gradients (backward propagation)***



Convolutional Layer Gradient

- **Gradient of *convolutional layer***

Define

$$\boldsymbol{Y} = (\boldsymbol{W} * \boldsymbol{X})$$

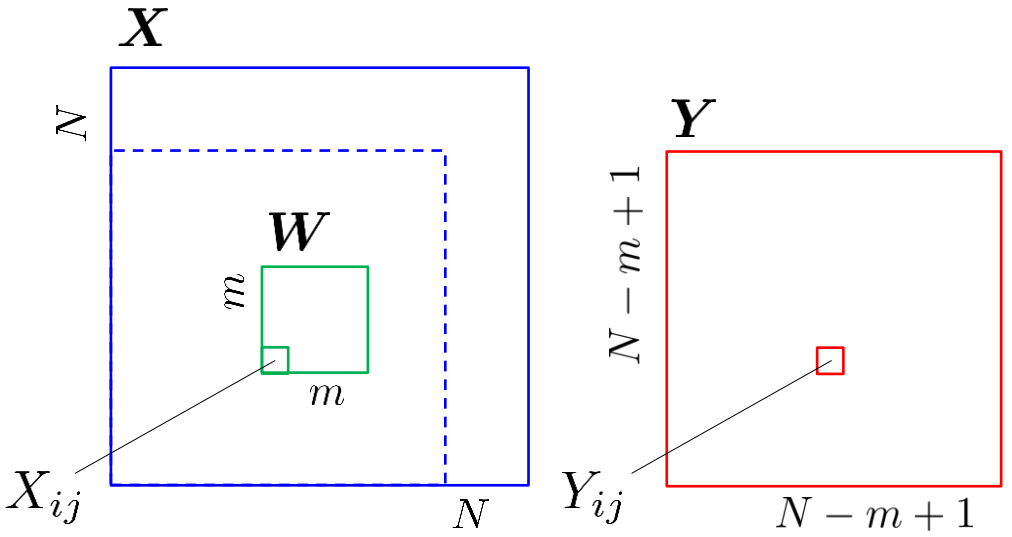
and, for convenience, assume that

$$\boldsymbol{W} \in \mathbb{R}^{m \times m}, \quad \boldsymbol{X} \in \mathbb{R}^{N \times N}$$

the input image is square

$$Y_{ij} = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} W_{ab} X_{(i+a)(j+b)}$$

the convolution operator is 'centered' in the lower left corner



All matrices in this example are indexed as images: i.e. the lower left corner is $0,0$

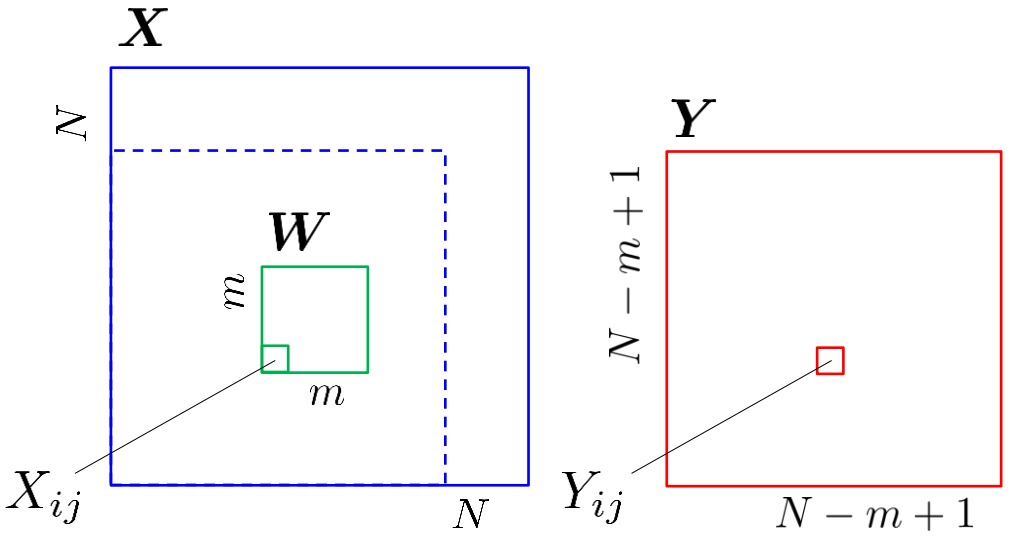
(In general m *is odd and the convolution is 'centered' in the centroid of the filter)*

Convolutional Layer Gradient

- **Gradient of *convolutional layer***

$$\boldsymbol{Y} = (\boldsymbol{W} * \boldsymbol{X})$$

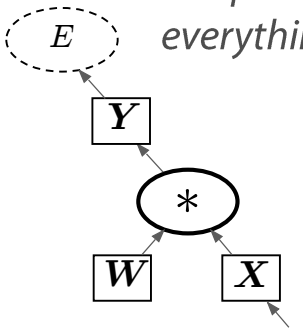
$$Y_{ij} = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} W_{ab} X_{(i+a)(j+b)}$$



Consider the pseudo-graph

The flow is bottom-up, in this example

The pseudo-node E represents everything that stands above



Another subgraph stands below (not represented)

Convolutional Layer Gradient

- **Gradient of *convolutional layer***

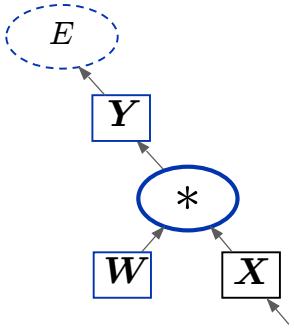
$$\boldsymbol{Y} = (\boldsymbol{W} * \boldsymbol{X})$$

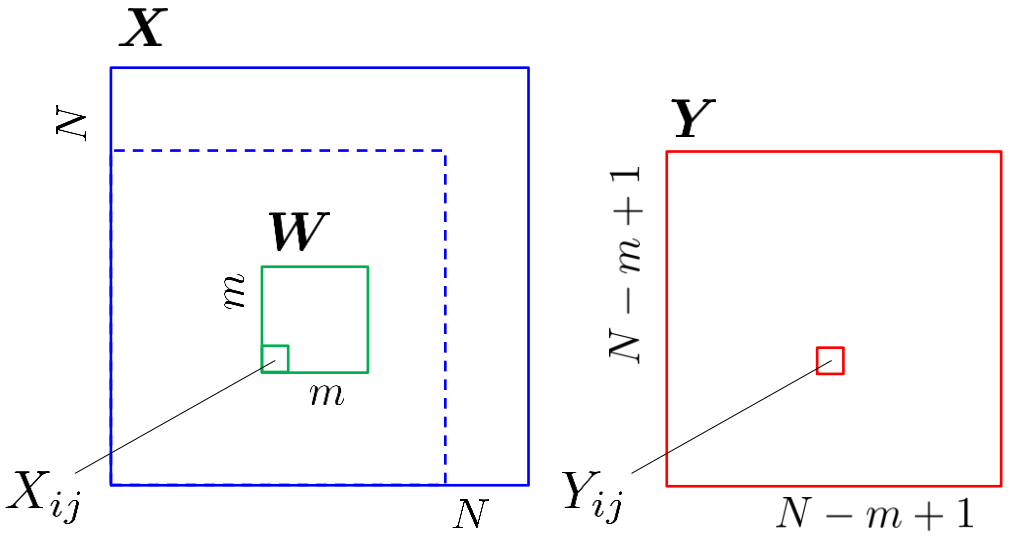
$$Y_{ij} = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} W_{ab} X_{(i+a)(j+b)}$$

Case 1:

$$\frac{\partial}{\partial \boldsymbol{W}} E(\boldsymbol{Y})$$

i.e. the 'end of the chain'



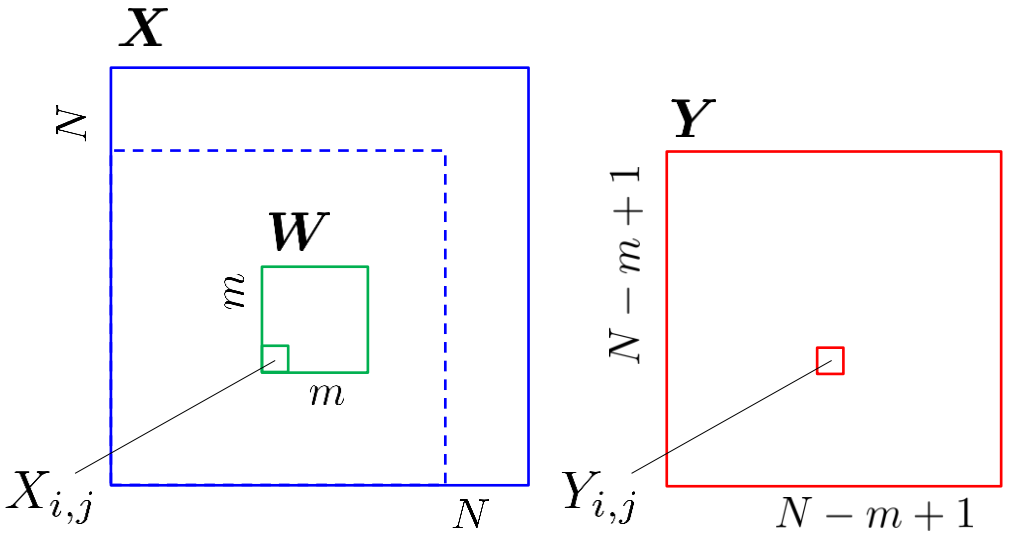


Convolutional Layer Gradient

- **Gradient of *convolutional layer***

$$\boldsymbol{Y} = (\boldsymbol{W} * \boldsymbol{X})$$

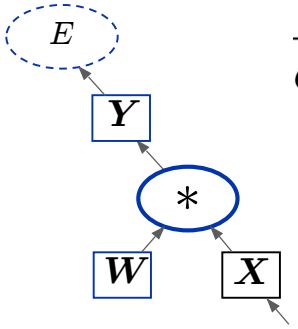
$$Y_{ij} = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} W_{ab} X_{(i+a)(j+b)}$$



Case 1:

$$\frac{\partial}{\partial \boldsymbol{W}} E(\boldsymbol{Y})$$

i.e. the 'end of the chain'



$$\frac{\partial}{\partial W_{lk}} E(\boldsymbol{Y}) = \sum_{i=0}^{N-m} \sum_{j=0}^{N-m} \frac{\partial E(\boldsymbol{Y})}{\partial Y_{ij}} \frac{\partial Y_{ij}}{\partial W_{lk}}$$

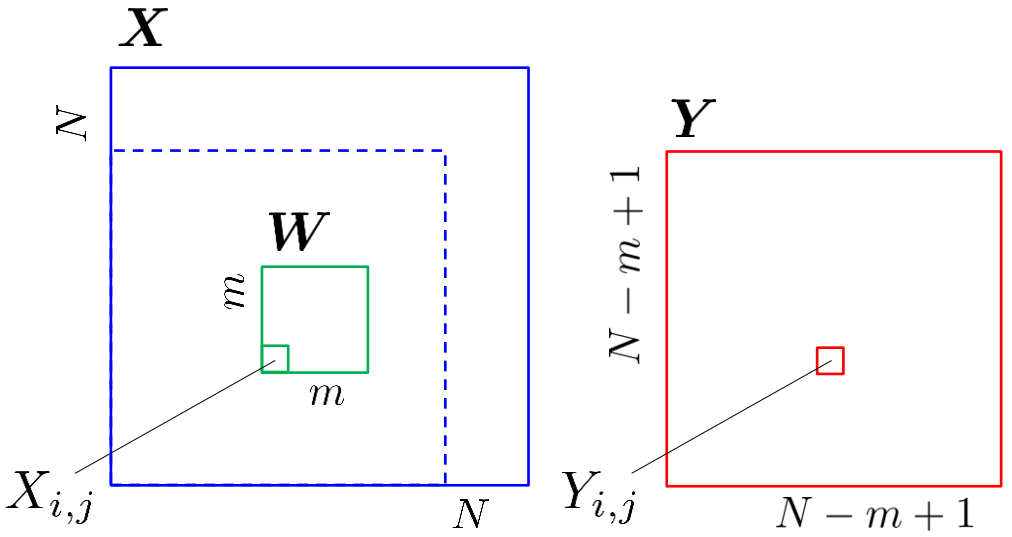
by applying the chain rule
in extended version (see also Episode 1)

Convolutional Layer Gradient

- **Gradient of *convolutional layer***

$$\boldsymbol{Y} = (\boldsymbol{W} * \boldsymbol{X})$$

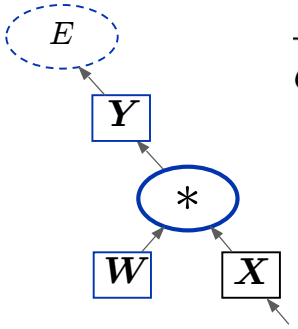
$$Y_{ij} = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} W_{ab} X_{(i+a)(j+b)}$$



Case 1:

$$\frac{\partial}{\partial \boldsymbol{W}} E(\boldsymbol{Y})$$

i.e. the 'end of the chain'



$$\frac{\partial}{\partial W_{lk}} E(\boldsymbol{Y}) = \sum_{i=0}^{N-m} \sum_{j=0}^{N-m} \frac{\partial E(\boldsymbol{Y})}{\partial Y_{ij}} \frac{\partial Y_{ij}}{\partial W_{lk}}$$

$$\partial E_{ij} := \frac{\partial E(\boldsymbol{Y})}{\partial Y_{ij}} \qquad \frac{\partial Y_{ij}}{\partial W_{lk}} = X_{(i+l)(j+k)}$$

i.e. the backpropagation component across Y_{ij}

$$\frac{\partial}{\partial W_{lk}} E(\boldsymbol{Y}) = \sum_{i=0}^{N-m} \sum_{j=0}^{N-m} \partial E_{ij}\, X_{(i+l)(j+k)}$$

Convolutional Layer Gradient

- **Gradient of *convolutional layer***

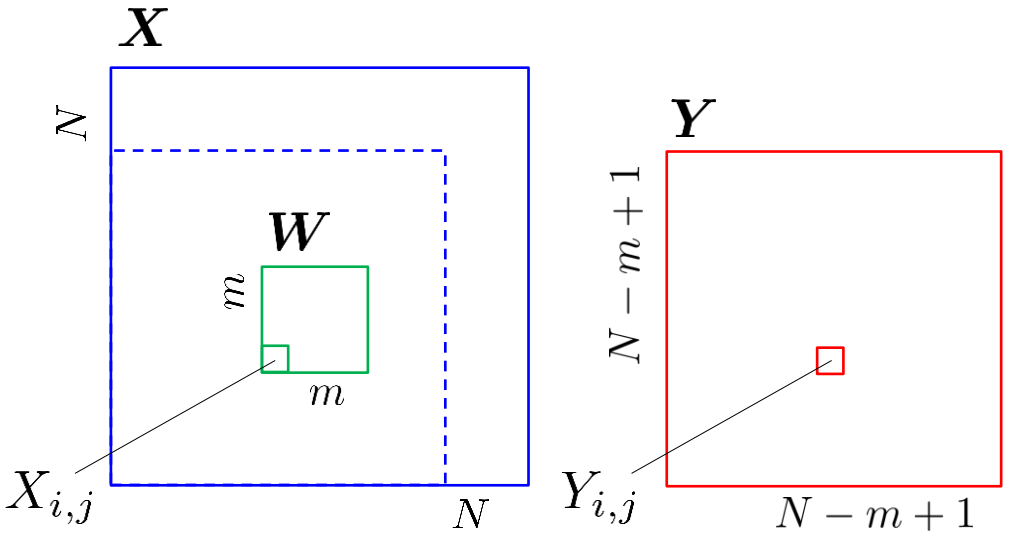
$$\boldsymbol{Y} = (\boldsymbol{W} * \boldsymbol{X})$$

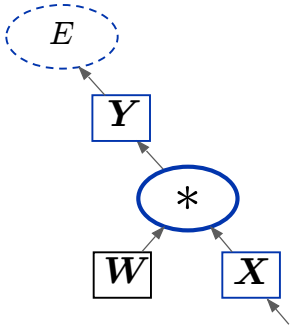
$$Y_{ij} = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} W_{ab} X_{(i+a)(j+b)}$$

Case 2:

$$\frac{\partial}{\partial \vartheta} E(\boldsymbol{Y})$$

$\vartheta \neq \boldsymbol{W}$ *is a generic parameter on which* $\boldsymbol{X}$ *depends*



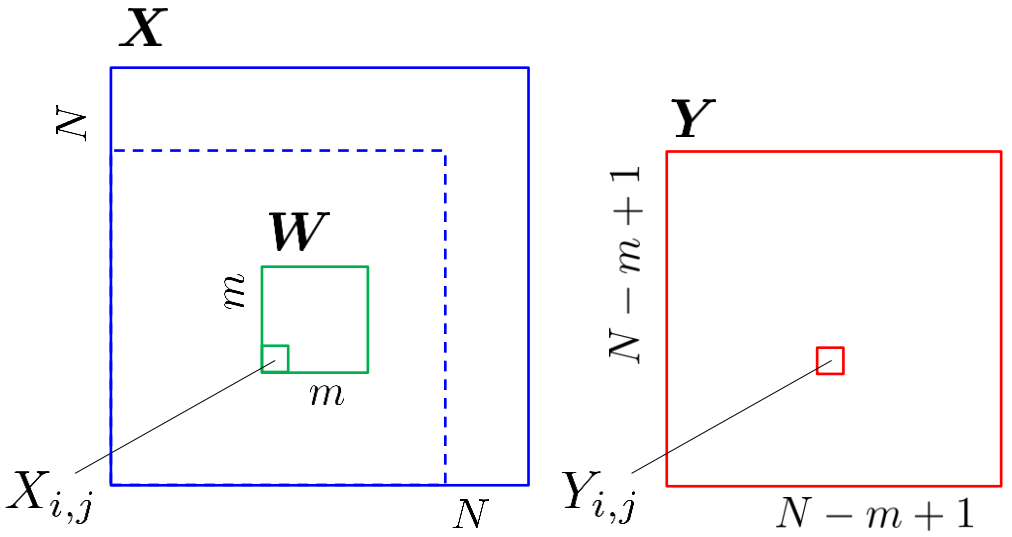


Convolutional Layer Gradient

- **Gradient of *convolutional layer***

$$\boldsymbol{Y} = (\boldsymbol{W} * \boldsymbol{X})$$

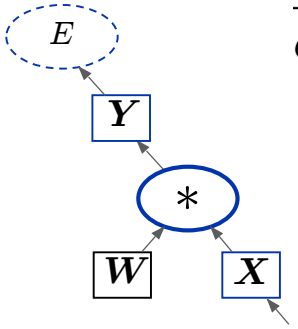
$$Y_{ij} = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} W_{ab} X_{(i+a)(j+b)}$$



Case 2:

$$\frac{\partial}{\partial \boldsymbol{\vartheta}} E(\boldsymbol{Y})$$

$\boldsymbol{\vartheta} \neq \boldsymbol{W}$ is a generic parameter on which $\boldsymbol{X}$ depends



$$\frac{\partial}{\partial \boldsymbol{\vartheta}} E(\boldsymbol{Y}) = \sum_{i=0}^{N-m} \sum_{j=0}^{N-m} \partial E_{ij} \frac{\partial Y_{ij}}{\partial \boldsymbol{\vartheta}}$$

$$= \sum_{i=0}^{N-m} \sum_{j=0}^{N-m} \partial E_{ij} \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} W_{ab} \frac{\partial}{\partial \boldsymbol{\vartheta}} X_{(i+a)(j+b)}$$

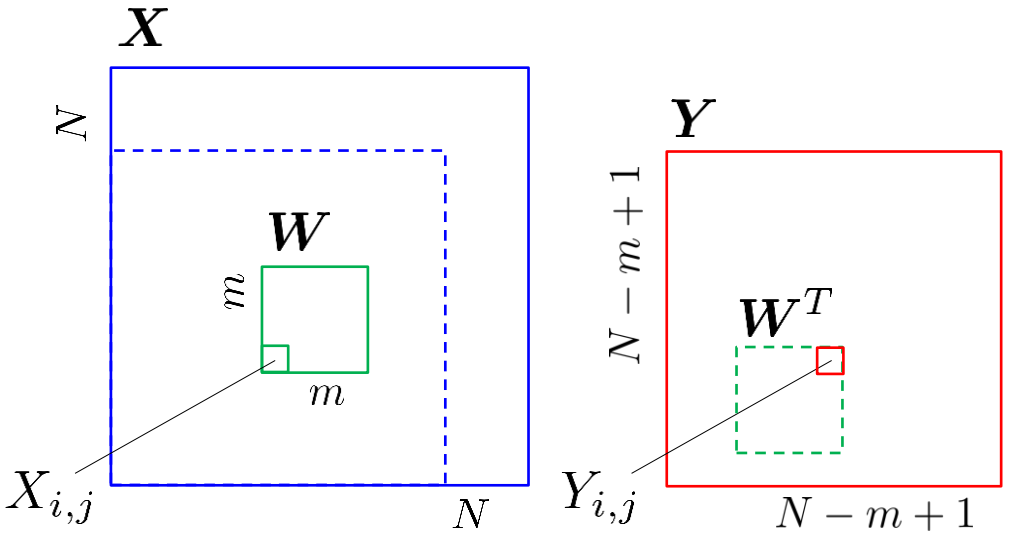
This is inconvenient: the same X components appear multiple times - let's re-factor ….

Convolutional Layer Gradient

- **Gradient of *convolutional layer***

$$\boldsymbol{Y} = (\boldsymbol{W} * \boldsymbol{X})$$

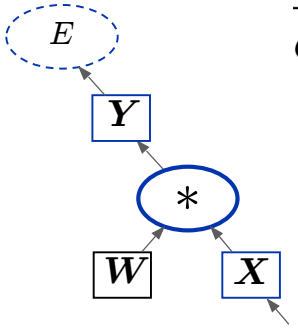
$$Y_{ij} = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} W_{ab} X_{(i+a)(j+b)}$$



Case 2:

$$\frac{\partial}{\partial \vartheta} E(\boldsymbol{Y})$$

$\vartheta \neq \boldsymbol{W}$ is a generic parameter on which $\boldsymbol{X}$ depends



$$\frac{\partial}{\partial \vartheta} E(\boldsymbol{Y}) = \sum_{i=0}^{N-m} \sum_{j=0}^{N-m} \partial E_{ij} \frac{\partial Y_{ij}}{\partial \vartheta}$$

$$= \sum_{i=0}^{N-m} \sum_{j=0}^{N-m} \partial E_{ij} \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} W_{ab} \frac{\partial}{\partial \vartheta} X_{(i+a)(j+b)}$$

$$= \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \frac{\partial}{\partial \vartheta} X_{ij} \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} W_{ba} \, \partial E_{(i-a)(j-b)} \qquad (i-a), (j-b) \geq 0$$

note the inversion of indexes

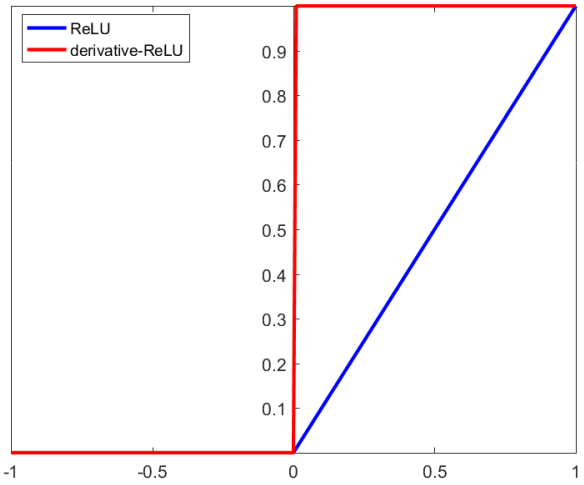
Convolutional Layer Gradient

- **Gradient of *ReLU*** *(see also Episode 1)*

$$\boldsymbol{Y} = \mathrm{ReLU}(\boldsymbol{X})$$

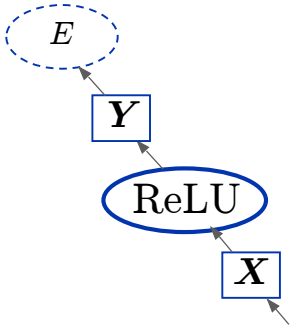
(ReLU has no parameters)

$$\frac{\partial}{\partial x}\mathrm{ReLU}(x) = \frac{\partial}{\partial x}\max(x, 0) \approx \mathrm{step}(x)$$



So the gradient of ReLU acts like a 'switch'

When is it open? *Backpropagation alone 'does not know'*



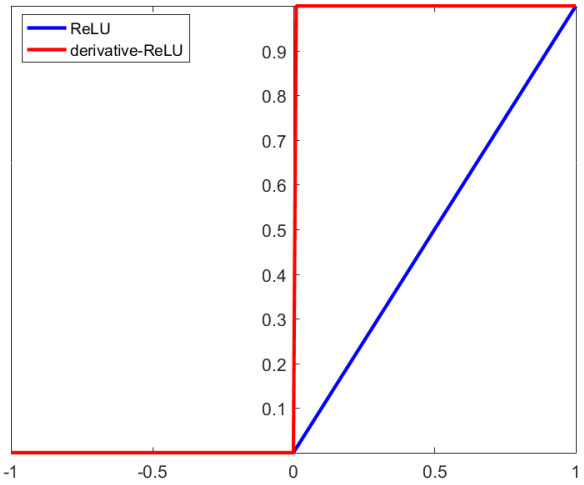
Convolutional Layer Gradient

- **Gradient of *ReLU*** *(see also Episode 1)*

$$\boldsymbol{Y} = \mathrm{ReLU}(\boldsymbol{X})$$

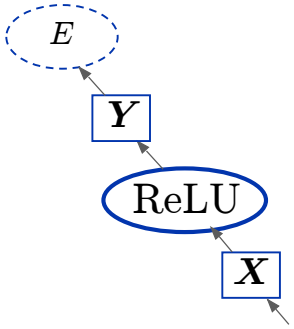
(ReLU has no parameters of its own)

$$\frac{\partial}{\partial x}\mathrm{ReLU}(x) = \frac{\partial}{\partial x}\max(x, 0) \approx \mathrm{step}(x)$$



So the gradient of ReLU acts like a 'switch'

When is it open? *Backpropagation alone 'does not know'*



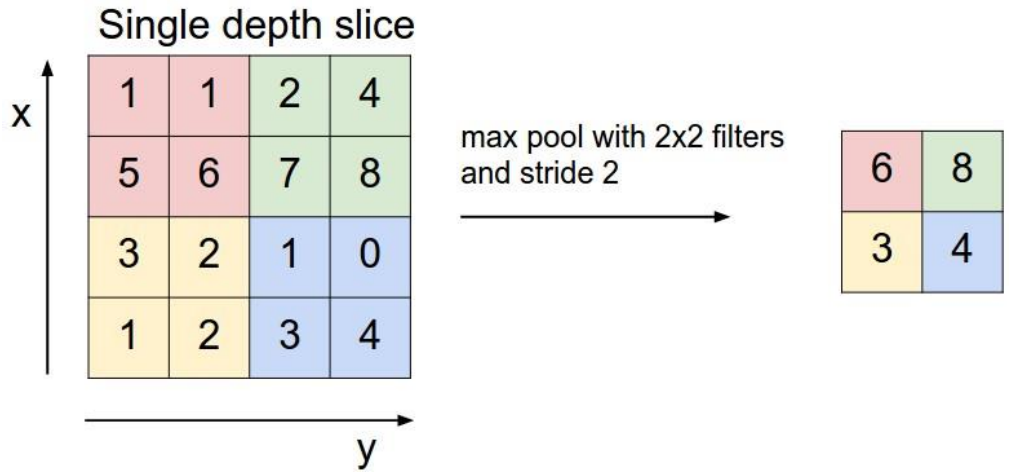
$\frac{\partial}{\partial \boldsymbol{\vartheta}} E(\boldsymbol{Y})$ *This is the gradient we want to compute*

as we have to apply it to each specific data item $\left(\frac{\partial}{\partial \boldsymbol{\vartheta}} E(\boldsymbol{Y})\right)(\boldsymbol{X}^{(i)})$

Moral: we need to perform one forward pass (i.e. activation)
to decide which component Y_{ij} is open (i.e. $= 1$) and which is not (i.e. $= 0$)

Max Pooling Gradient

- **Gradient of *Max Pooling***

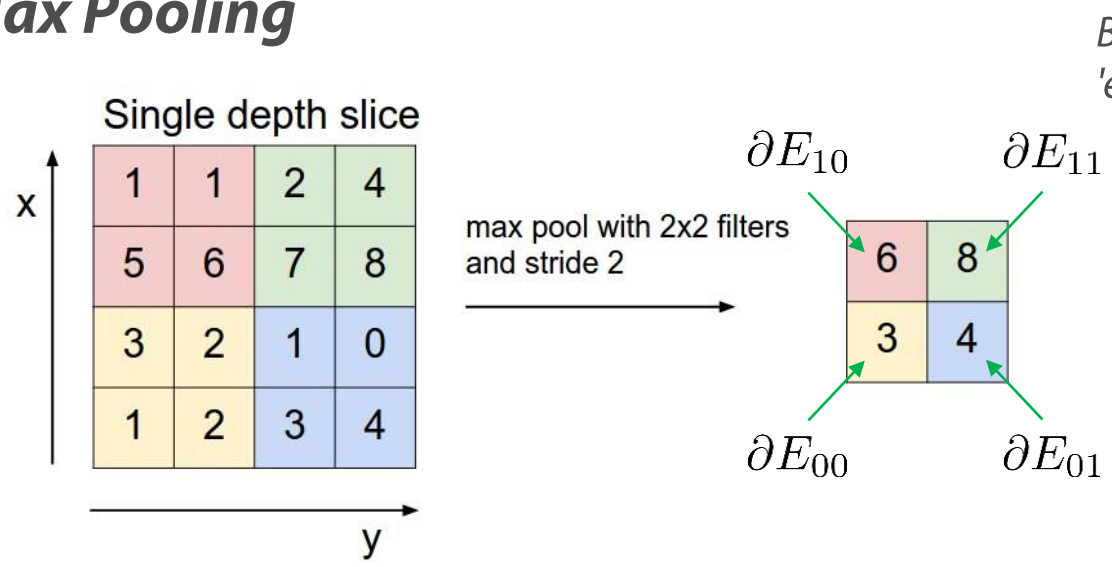


The gradient of max pooling acts as *multiplexer*

As with ReLU, one forward pass is required to determine which channel is selected

Convolutional Layer Gradient

- **Gradient of *Max Pooling***

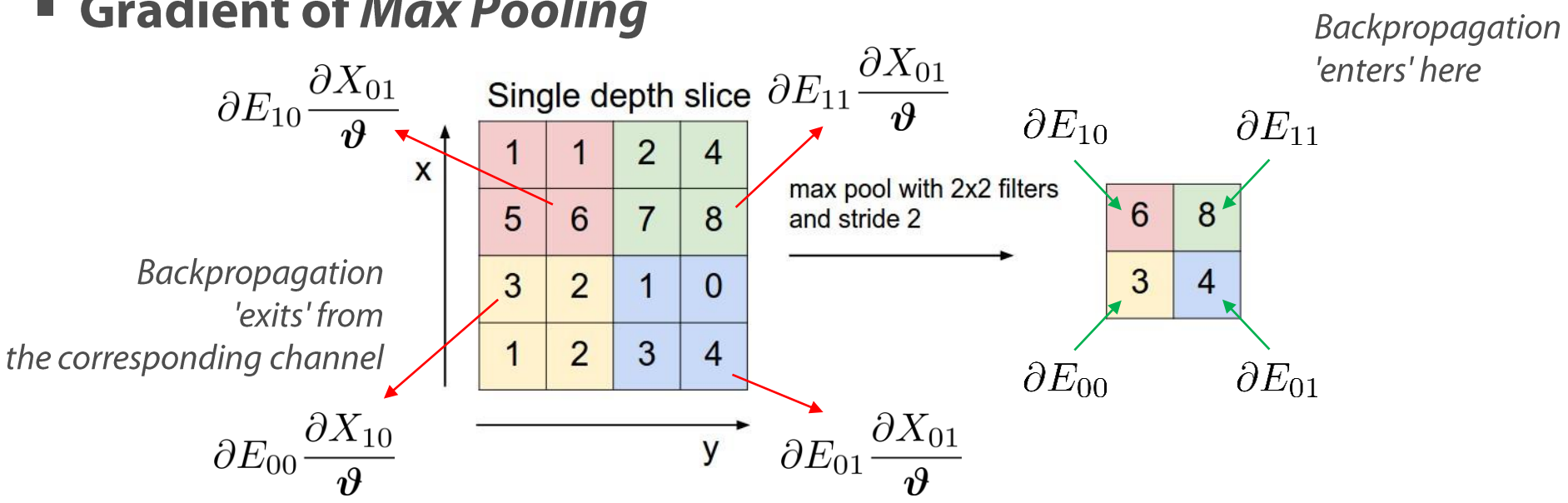


The gradient of max pooling acts as *multiplexer*

As with ReLU, one forward pass is required to determine which channel is selected

Convolutional Layer Gradient

- **Gradient of *Max Pooling***



The gradient of max pooling acts as *multiplexer*

As with ReLU, one forward pass is required to determine which channel is selected

LRN Gradient

- **Gradient of *Local Response Normalization***

$$Y_{ijl} := \frac{X_{ijl}}{\left(a + \alpha \sum_{k \in \mathrm{Nbr}(l)} (X_{ijk})^2\right)^{\beta}}$$

where a, α, β are fixed hyperparameters

This formula is quite inconvenient: let's simplify….

LRN Gradient

- **Gradient of *Local Response Normalization***

$$Y_{ijl} := \frac{X_{ijl}}{\sum_{k=1}^{h} X_{ijk}}$$

i.e. plain, cross-map normalization

LRN Gradient

- **Gradient of *Local Response Normalization***

$$Y_{ijl} := \frac{X_{ijl}}{\sum_{k=1}^{h} X_{ijk}}$$

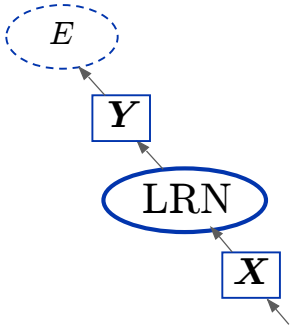
i.e. plain, cross-map normalization

i.e. the backpropagation component across Y_{ijl}

$$\frac{\partial}{\partial \boldsymbol{\vartheta}} E(\boldsymbol{Y}) = \sum_{i,j} \sum_{l} \partial E_{ijl} \frac{\partial Y_{ijl}}{\partial \boldsymbol{\vartheta}} \qquad \text{where} \quad \partial E_{ijl} := \frac{\partial E(\boldsymbol{Y})}{\partial Y_{ijl}}$$

$$= \sum_{i,j} \sum_{l} \partial E_{ijl} \frac{\partial}{\partial \boldsymbol{\vartheta}} \frac{X_{ijl}}{\sum_k X_{ijk}}$$

$$= \sum_{i,j} \sum_{l} \partial E_{ijl} \left(\frac{1}{\sum_k X_{ijk}} \frac{\partial X_{ijl}}{\partial \boldsymbol{\vartheta}} - \frac{X_{ijl}}{\left(\sum_k X_{ijk}\right)^2} \sum_k \frac{\partial X_{ijk}}{\partial \boldsymbol{\vartheta}} \right)$$



$$= \sum_{i,j} \sum_{l} \partial E_{ijl} \left(\frac{1}{c} \frac{\partial X_{ijl}}{\partial \boldsymbol{\vartheta}} - \frac{Y_{ijl}}{c} \sum_k \frac{\partial X_{ijk}}{\partial \boldsymbol{\vartheta}} \right)$$

where

$$c := \sum_k X_{ijk}$$

LRN Gradient

- **Gradient of *Local Response Normalization***

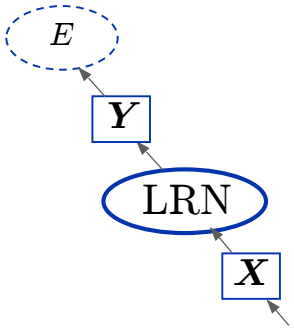
$$Y_{ijl} := \frac{X_{ijl}}{\sum_{k=1}^{h} X_{ijk}}$$

i.e. plain, cross-map normalization

$$\frac{\partial}{\partial \boldsymbol{\vartheta}} E(\boldsymbol{Y}) = \sum_{i,j} \sum_{l} \partial E_{ijl} \left(\frac{1}{c} \frac{\partial X_{ijl}}{\partial \boldsymbol{\vartheta}} - \frac{Y_{ijl}}{c} \sum_{k} \frac{\partial X_{ijk}}{\partial \boldsymbol{\vartheta}} \right)$$

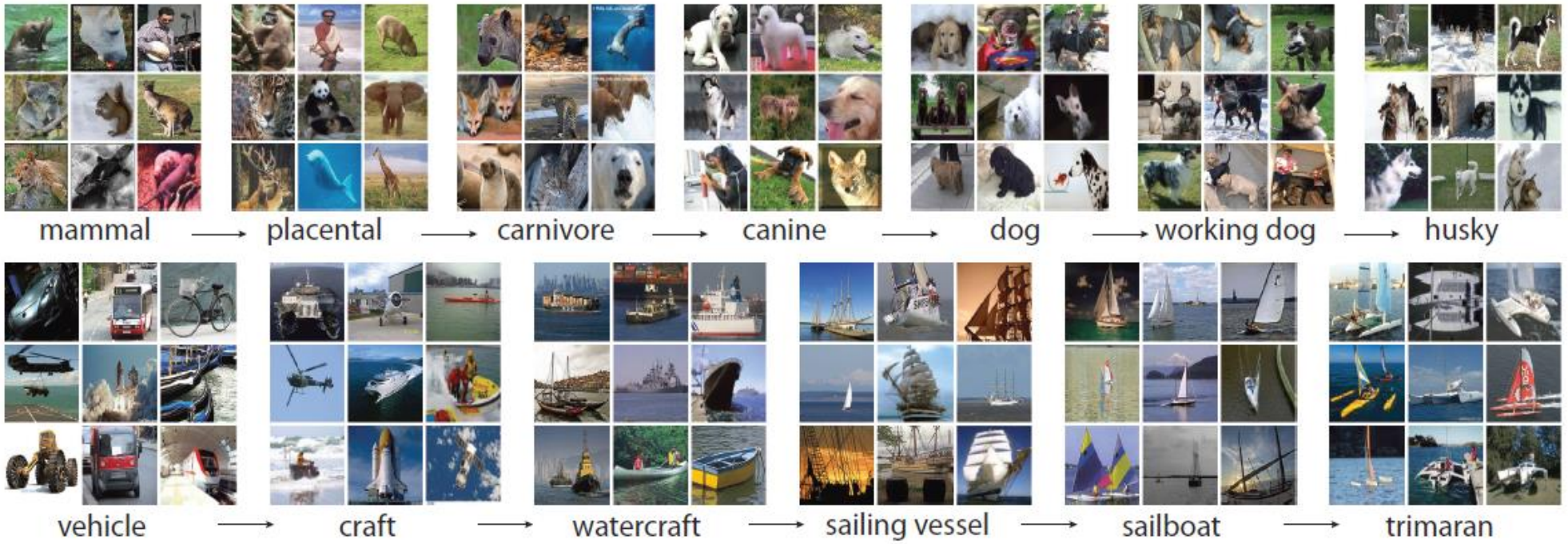
This is inconvenient: the same X components appear multiple times - let's re-factor ….

$$= \sum_{i,j} \sum_{l} \left(\frac{\partial E_{ijl}}{c} - \sum_{k} \left(Y_{ijk} \frac{\partial E_{ijk}}{c} \right) \right) \frac{\partial X_{ijl}}{\partial \boldsymbol{\vartheta}}$$



ImageNet Challenge

- The ImageNet Large Scale Visual Recognition Challenge



1,461,406 full resolution images

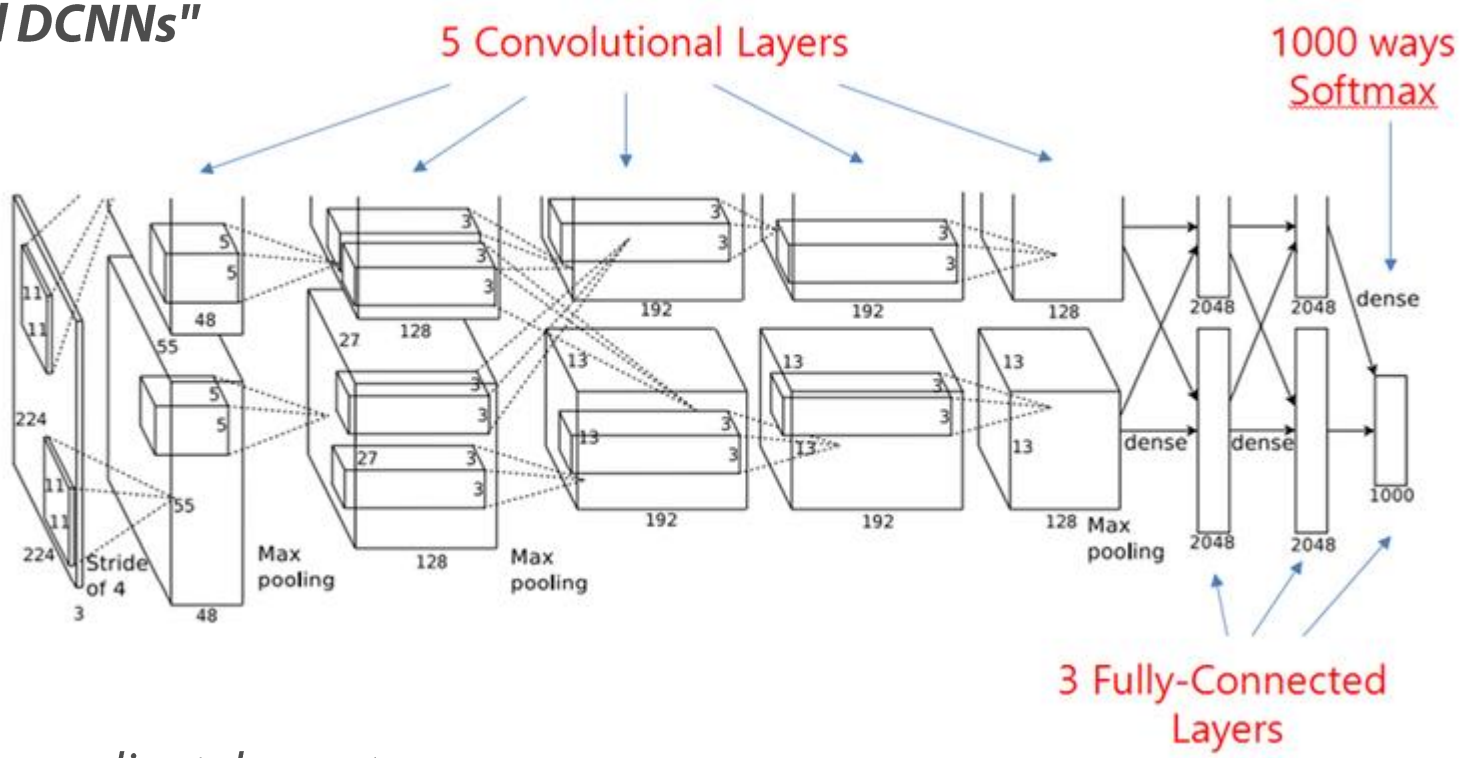
Complex and multiple textual annotation,
hierarchy of 1000 object classes along several dimensions

The image classification challenge is run annually since 2010

[figures from www.nvidia.com]

AlexNet (Krizhevsky, Sutskever & Hinton, 2012)

"The Mother of all DCNNs"



Trained with *batch gradient descent*

- the final supervised training set contained 15M images
- training was performed on two NVIDIA GTX 580 GPUs for six days

[image from https://world4jason.gitbooks.io/research-log/content/deepLearning/CNN/Model%20&%20ImgNet/alexnet/alexnet.html]

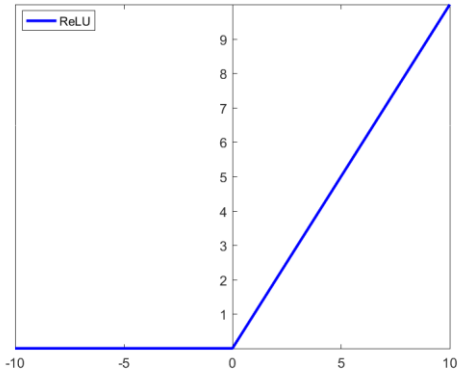
Deep Convolutional Neural Networks (DCNN)

- **AlexNet**

  Why ReLU and not another non-linearity?

  Because it is much faster to train.

$$y = \max(0, x)$$



ReLU

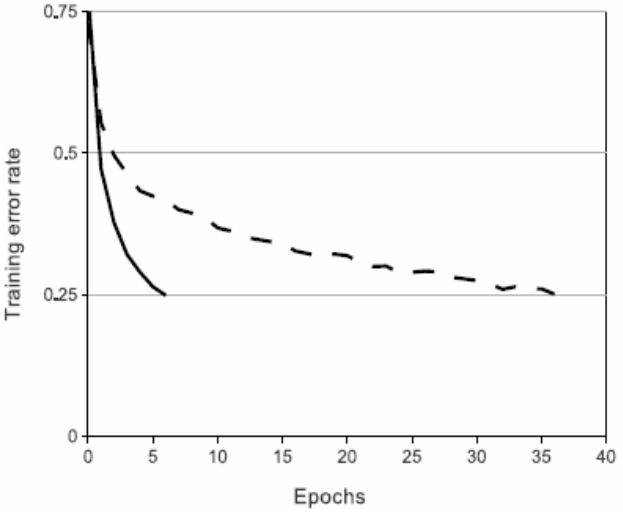
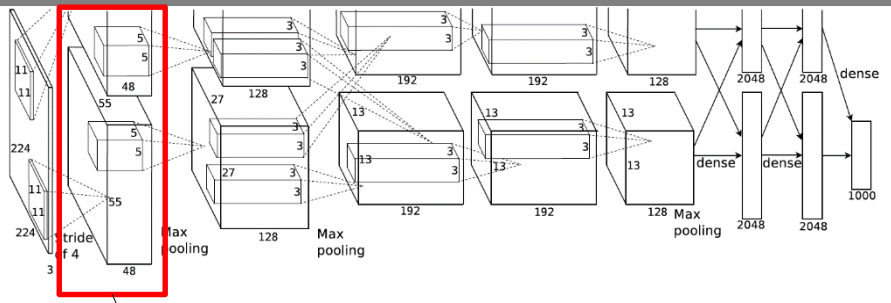


Figure 1: A four-layer convolutional neural network with ReLUs (**solid line**) reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons (**dashed line**). The learning rates for each network were chosen independently to make training as fast as possible. 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.

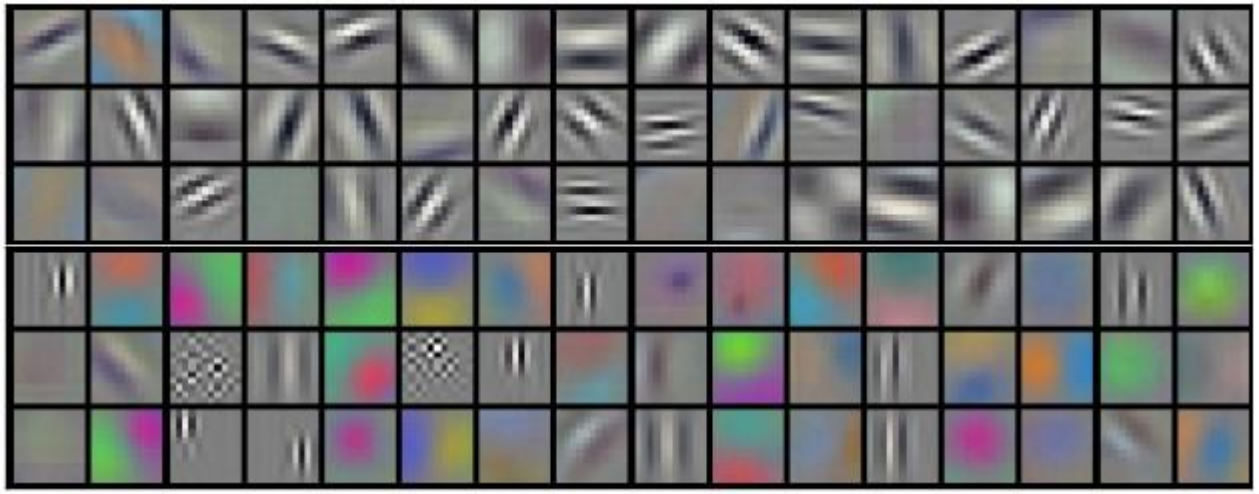
Image from [Krizhevsky, Sutskever & Hinton, 2012]

Inside AlexNet (after training)

AlexNet Filters (after training)



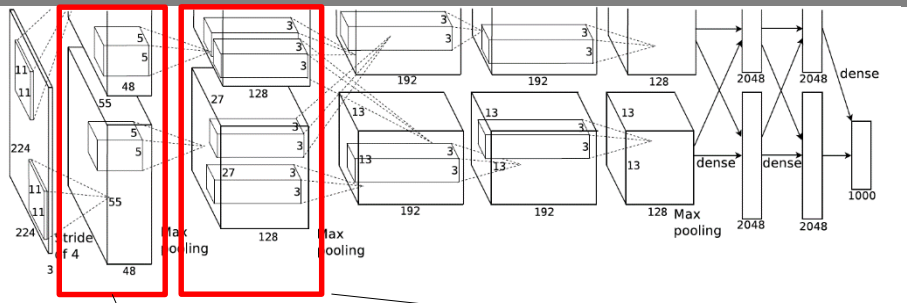
Layer 1



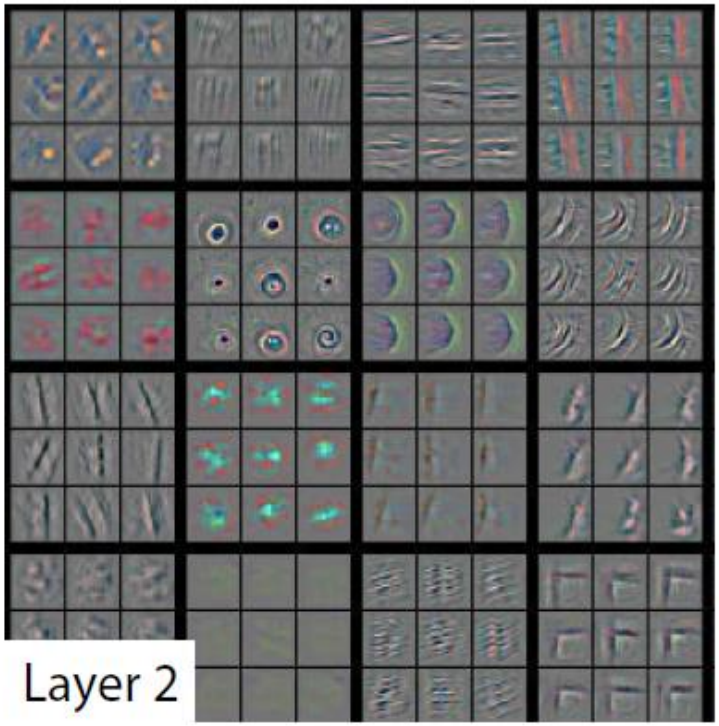
These are 96 real examples of convolutive filters for RGB images

[image from http://cs231n.github.io/convolutional-networks/]

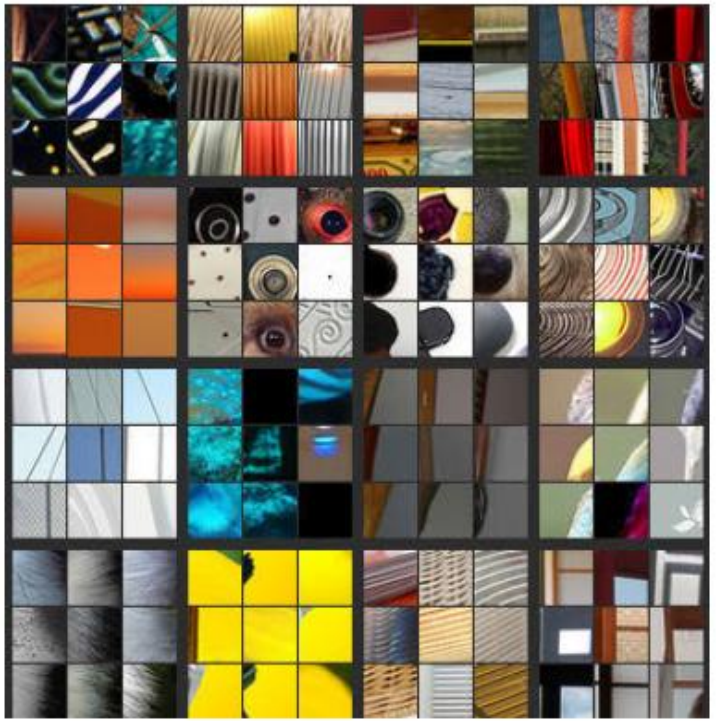
AlexNet Filters- DeconvNet



Layer 1

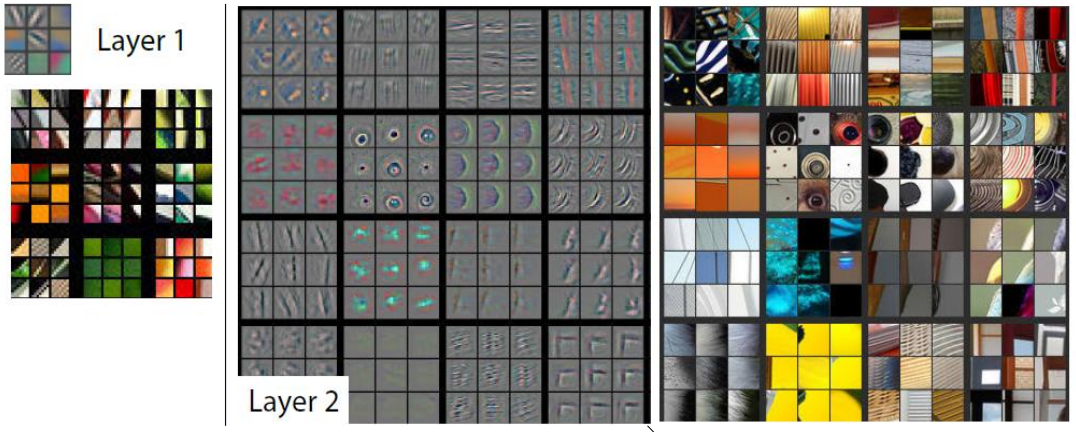


Layer 2



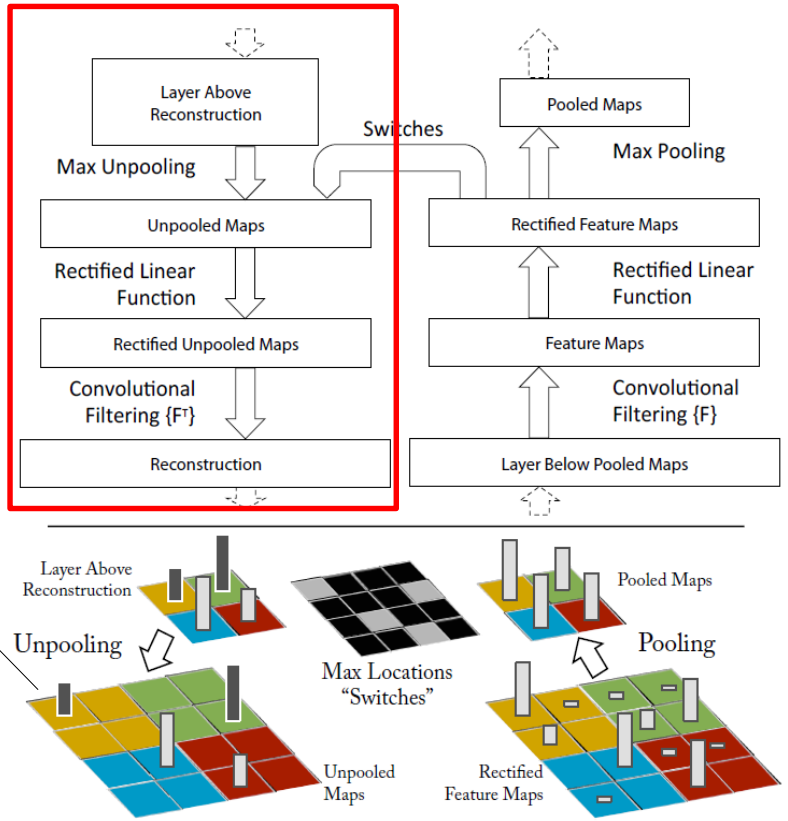
[images from https://arxiv.org/pdf/1311.2901.pdf]

AlexNet Filters- DeconvNet



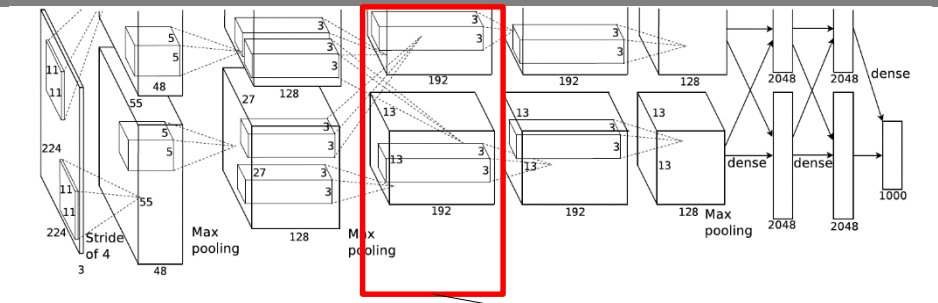
DeconvNet:
using a DCNN in reverse

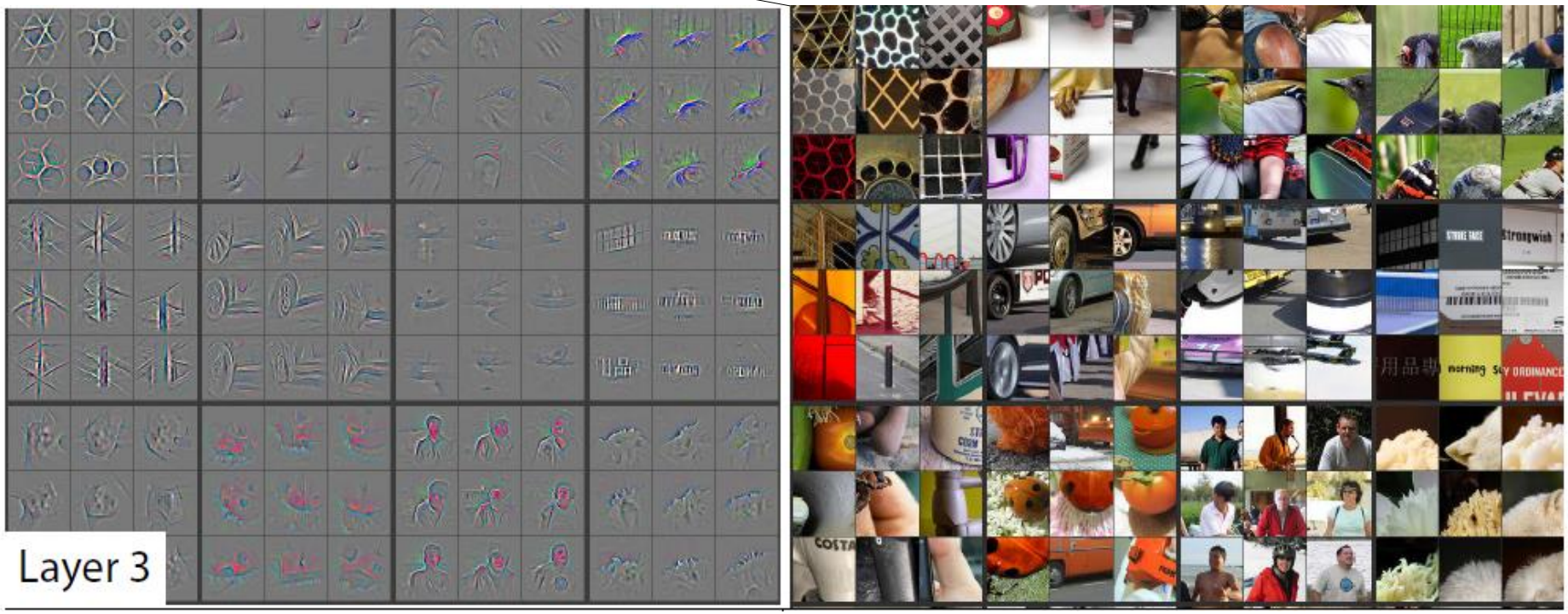
DeconvNet



[images from https://arxiv.org/pdf/1311.2901.pdf]

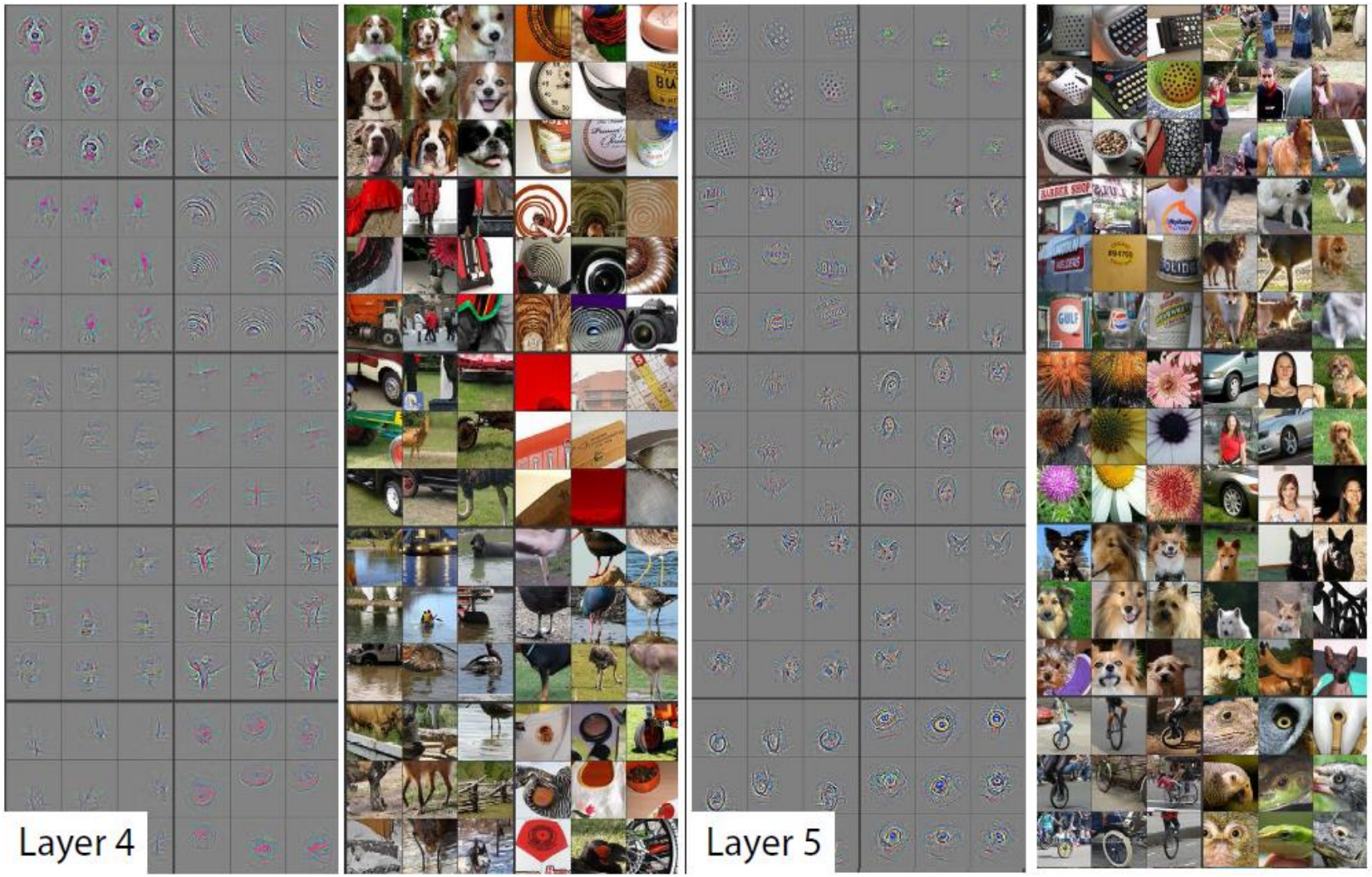
AlexNet Filters- DeconvNet





[images from https://arxiv.org/pdf/1311.2901.pdf]

AlexNet Filters- DeconvNet



[images from https://arxiv.org/pdf/1311.2901.pdf]

Beyond AlexNet: The DCNN storm

VGG Architecture

Several variants

Only 3x3 convolutional filters used
(each with ReLU)

LRN used in only one variant

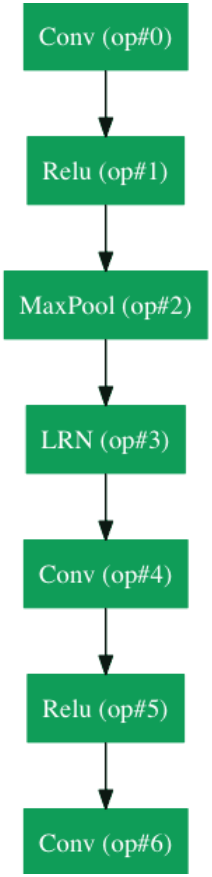
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128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

Image from
[https://arxiv.org/pdf/1409.1556.pdf]

Inception Acrhitecture

- The ImageNet Large Scale Visual Recognition Challenge

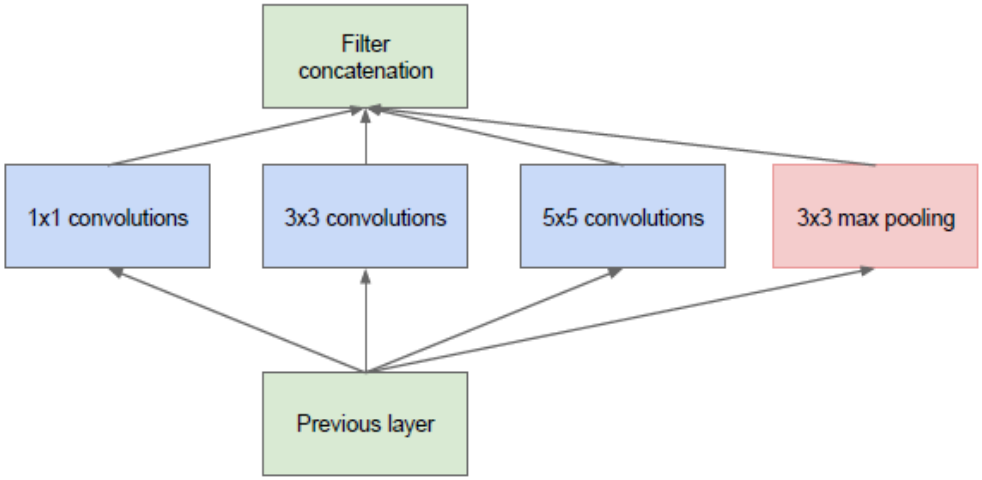
  How deep is a deep neural network, for a task like this?



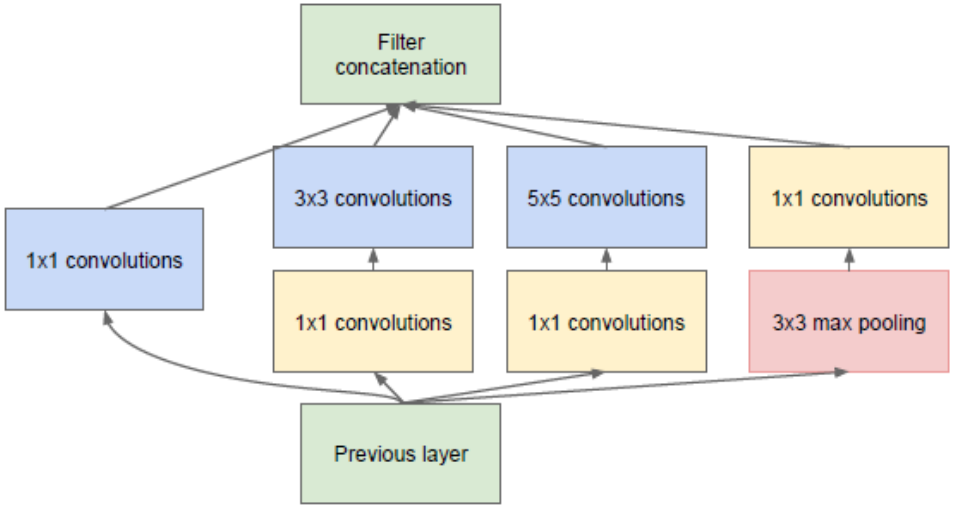
GoogLeNet (Inception v4) winner of two out of three categories in 2014: 154 network layers

Inception Architecture

- Inception modules



(a) Inception module, naïve version

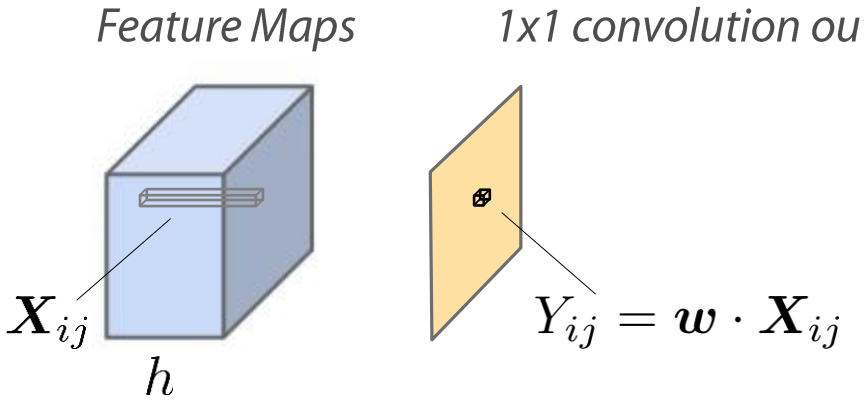


(b) Inception module with dimension reductions

Image from [https://arxiv.org/pdf/1409.4842.pdf]

Inception Architecture

- 1x1 convolution?



(It is a kind of misnomer)
Each filter has dimension $1 \times 1 \times h$
where h is the depth of the set of filter maps

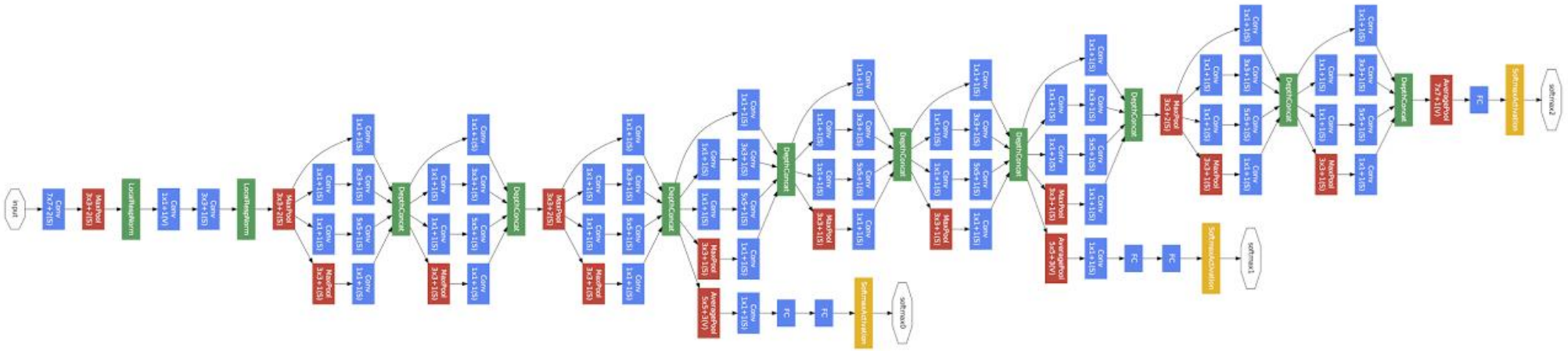
Using d 1x1 convolution filters allows changing depth h into d

Clearly the assumption is $d < h$

It mimics a fully connected layer (across channels)

Inception Architecture

- GoggLeNet architecture

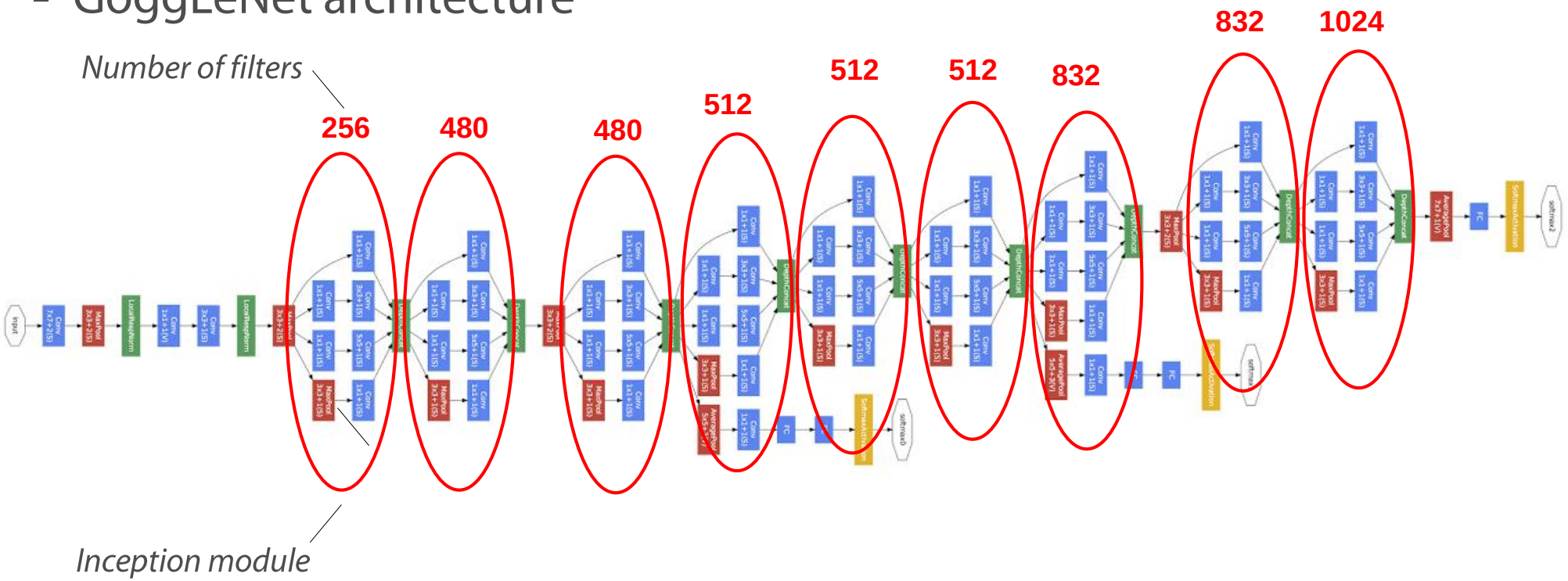


Convolutive
Max Pool
Softmax
Filter Concat

Image from [https://arxiv.org/pdf/1409.4842.pdf]

Inception Architecture

- GoggLeNet architecture

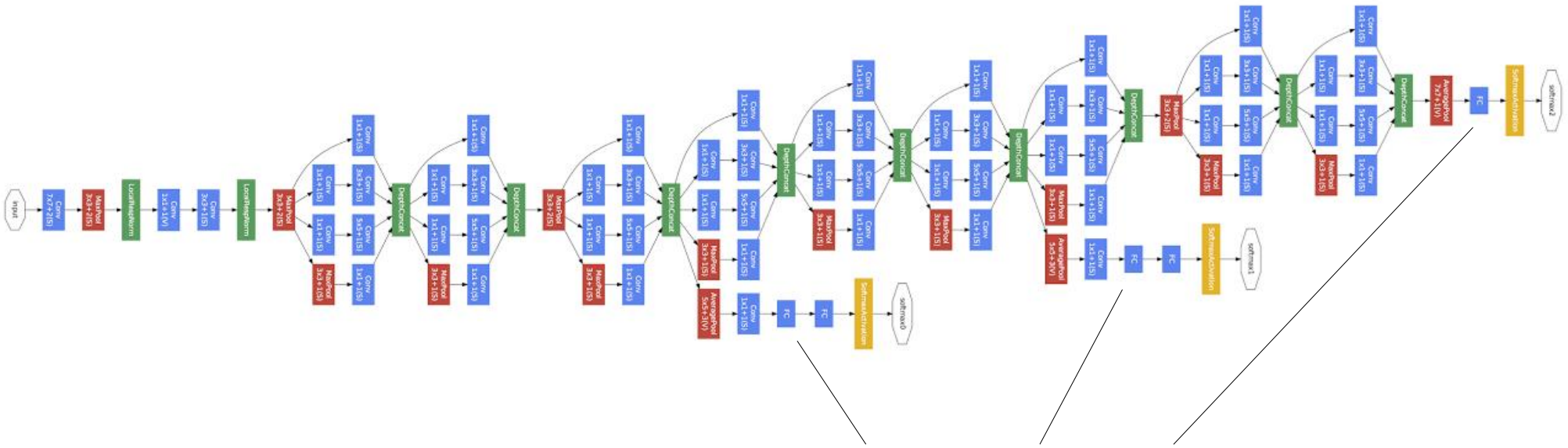


Convolutive
Max Pool
Softmax
Filter Concat

Image from [https://arxiv.org/pdf/1409.4842.pdf]

Inception Architecture

- GoggLeNet architecture



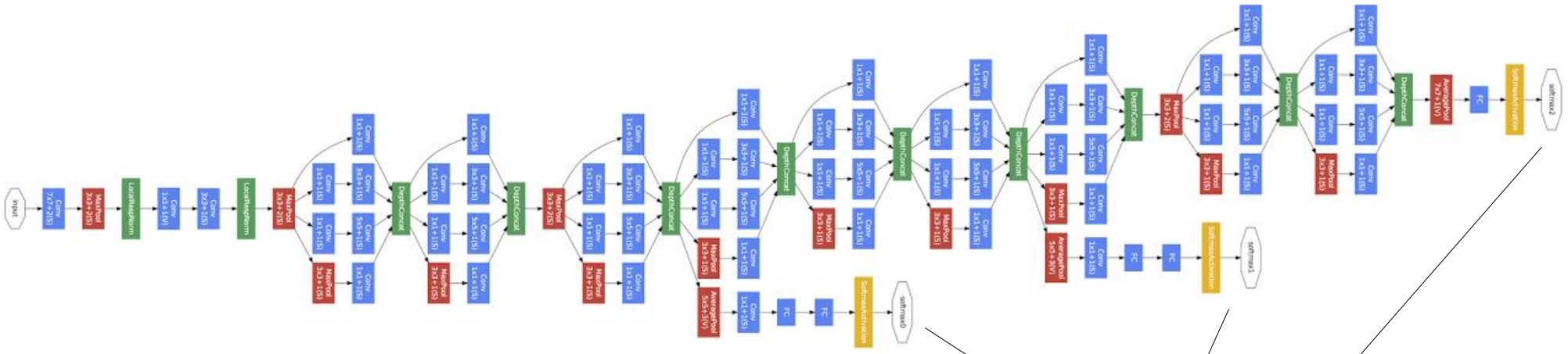
Much smaller FC layers

Convolutive
Max Pool
Softmax
Filter Concat

Image from [https://arxiv.org/pdf/1409.4842.pdf]

Inception Architecture

- GoggLeNet architecture



Three softmax outputs

They are trained to produce the same output, simultaneously

Convolutive
Max Pool
Softmax
Filter Concat

Image from [https://arxiv.org/pdf/1409.4842.pdf]

ResNet Architecture

- ResNet block

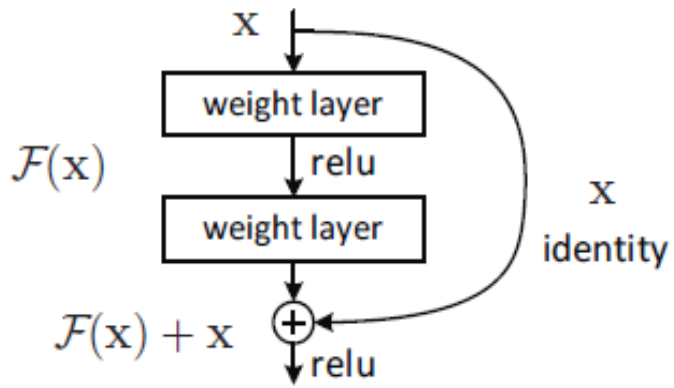


Figure 2. Residual learning: a building block.

Image from [https://arxiv.org/pdf/1512.03385.pdf]

ResNet Architecture

- ResNet architecture

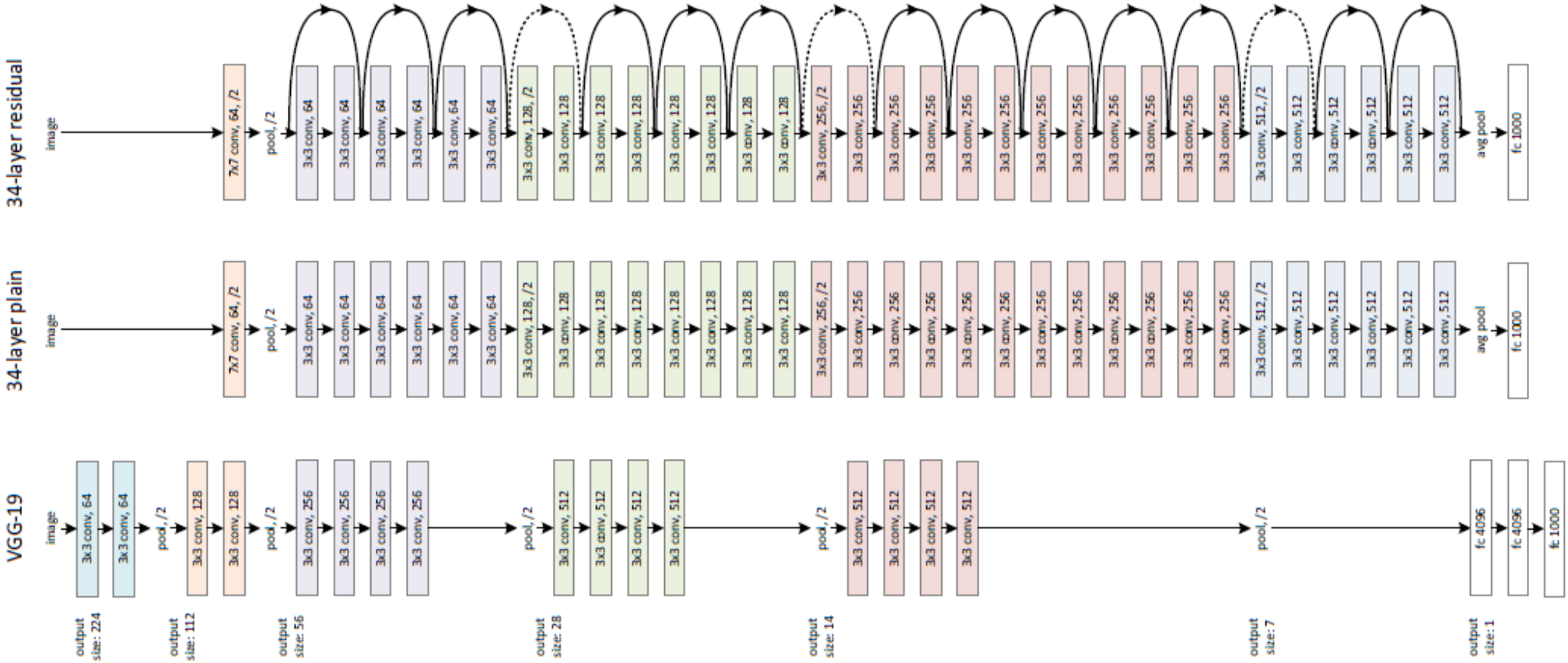
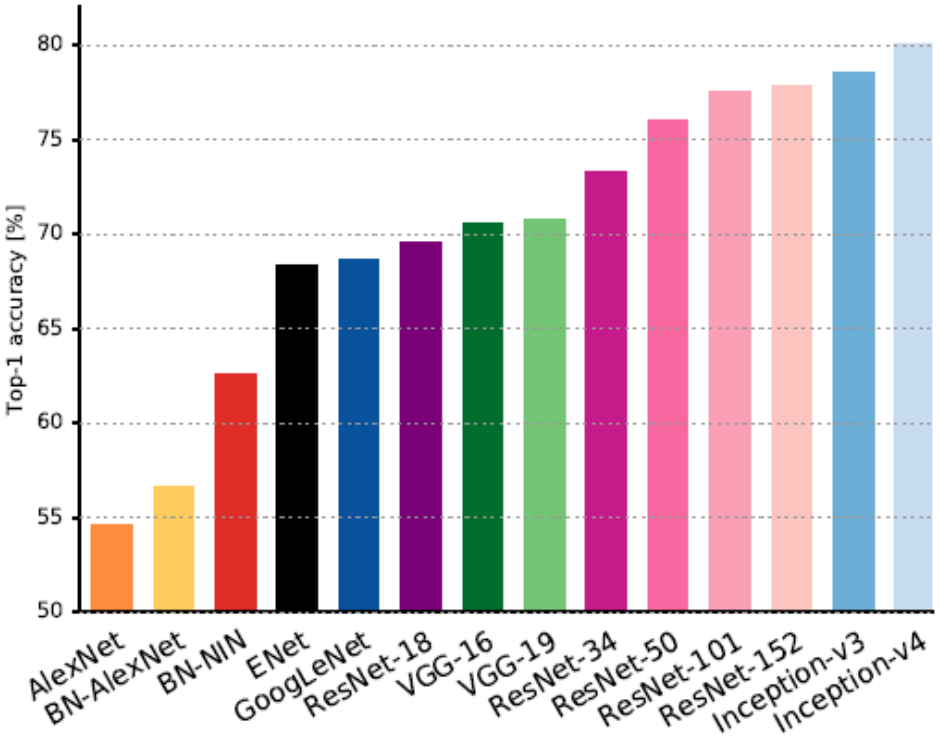


Image from [https://arxiv.org/pdf/1512.03385.pdf]

Comparing Different DCNNs

- Comparative charts at Top-1 accuracy

  i.e. how often the DCNN is right with ImageNet with its top prediction



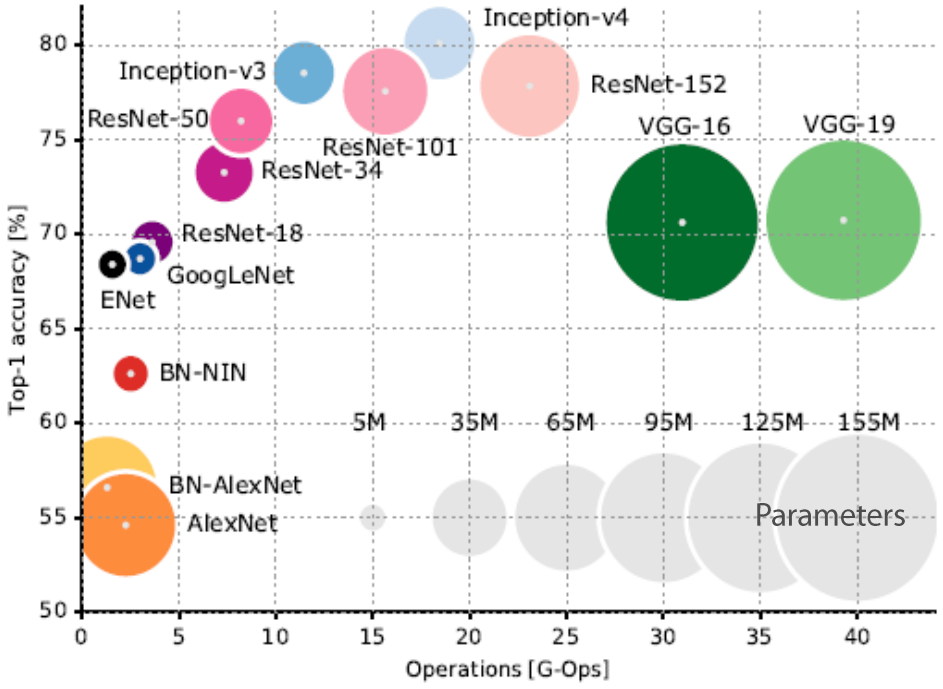
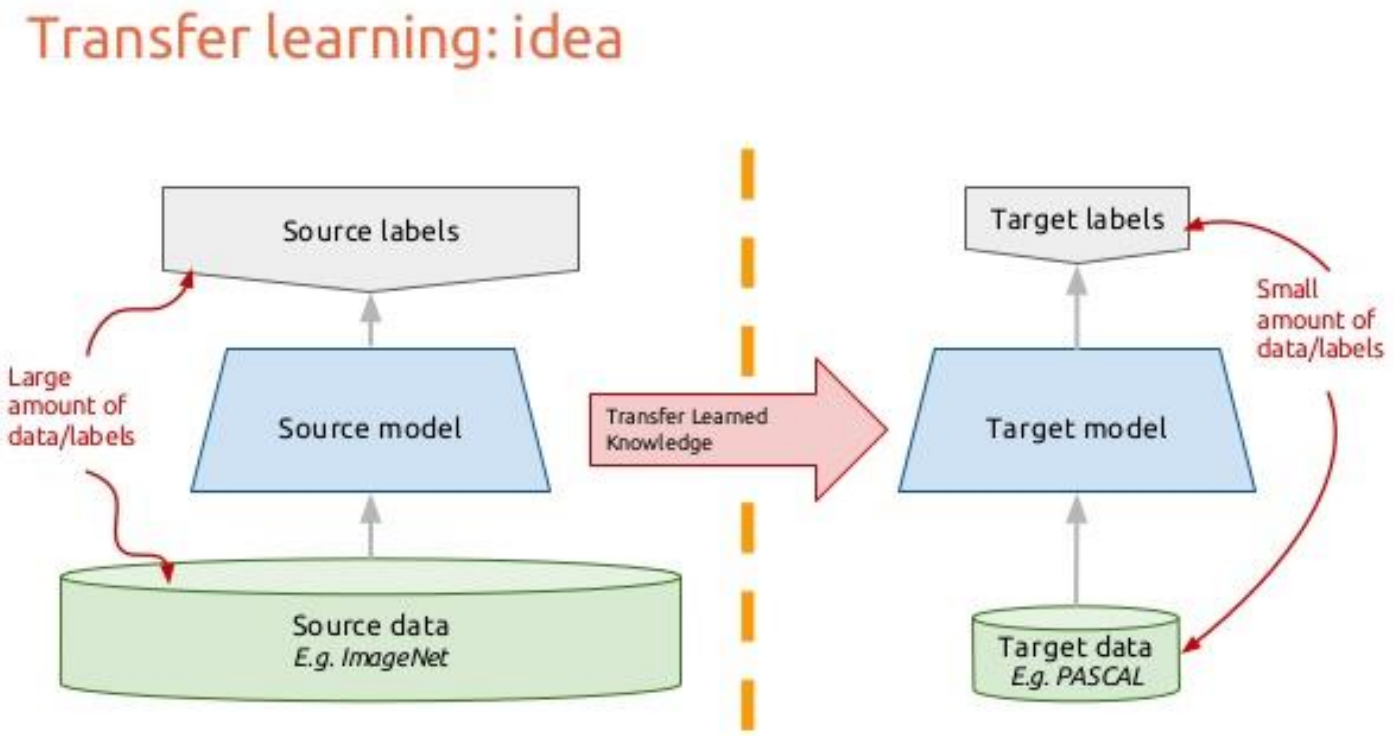


Image from [https://arxiv.org/abs/1605.07678, 2017]

Transfer Learning

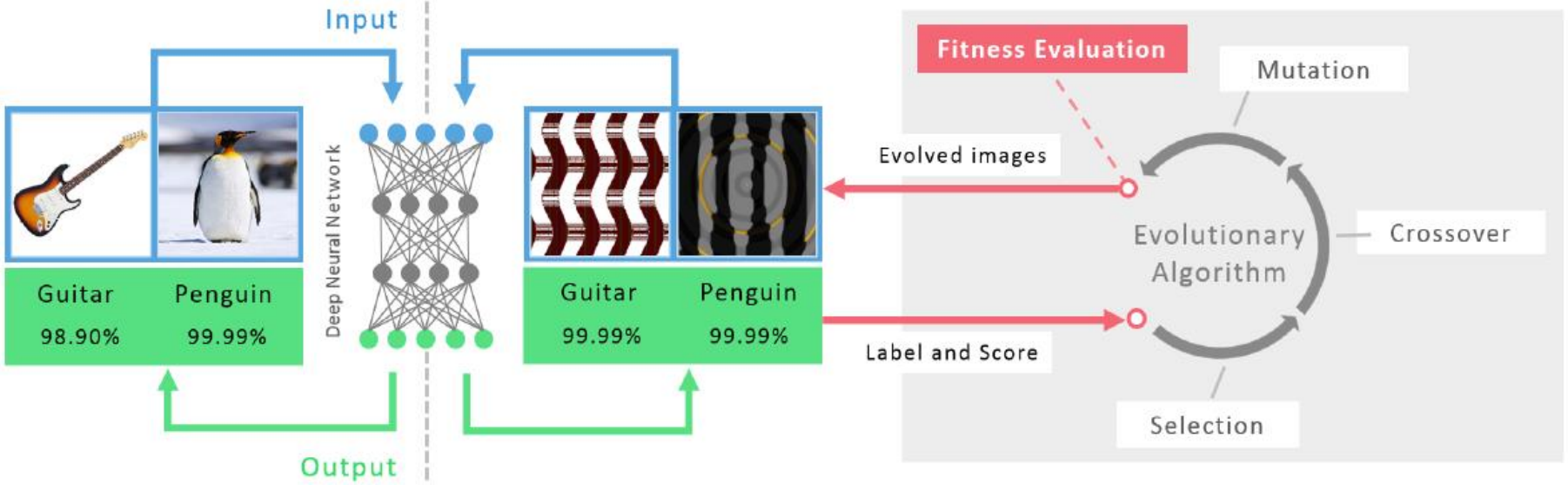


Do DCNNs Dream of Electric Sheep?

A DCNN can be fooled…

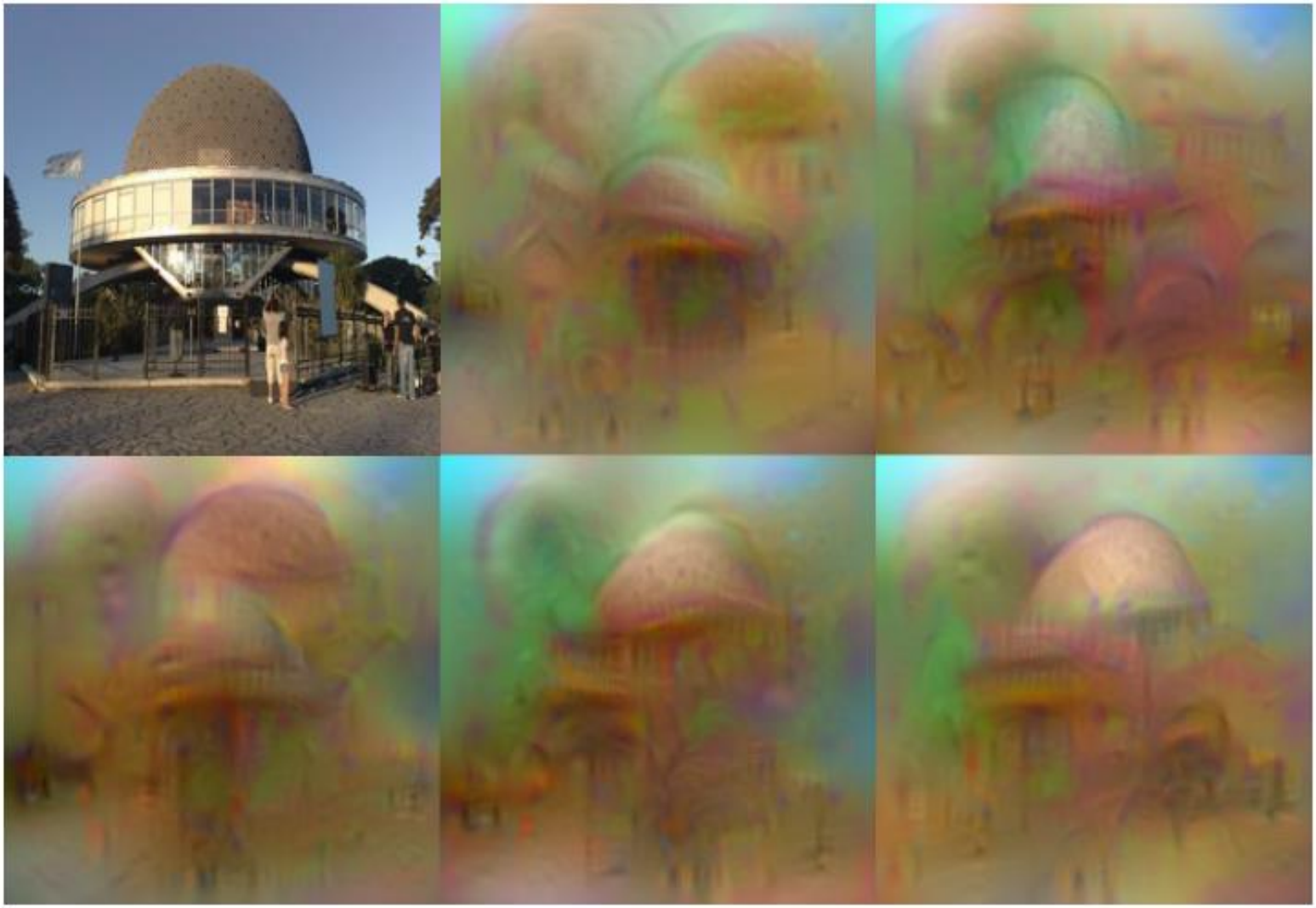
1 State-of-the-art DNNs can recognize real images with high confidence

2 But DNNs are also easily fooled: images can be produced that are unrecognizable to humans, but DNNs believe with 99.99% certainty are natural objects



[images from https://arxiv.org/pdf/1412.1897v4.pdf]

Reconstructing Images from Feature Maps



[images from https://arxiv.org/pdf/1412.0035v1.pdf]

Reconstructing Images from Feature Maps

- **Image Space Gradient Descent**

  Define

  $$\boldsymbol{\Phi}_{k,l}(\boldsymbol{I})$$

  as the response of a DCNN at a layer k, filter l to an image $\boldsymbol{I}$

  Given a specific image $\hat{\boldsymbol{I}}$, we define the loss function

  $$L(\hat{\boldsymbol{I}}, \boldsymbol{I}) := \|\boldsymbol{\Phi}_{k,l}(\hat{\boldsymbol{I}}) - \boldsymbol{\Phi}_{k,l}(\boldsymbol{I})\|^2$$

  and the optimization problem

  $$\boldsymbol{I}^* := \operatorname{argmin}_{\boldsymbol{I}} \left(L(\hat{\boldsymbol{I}}, \boldsymbol{I}) + \rho P(\boldsymbol{I}) + \lambda \|\boldsymbol{I}\|^2 \right)$$

  L2 Regularization

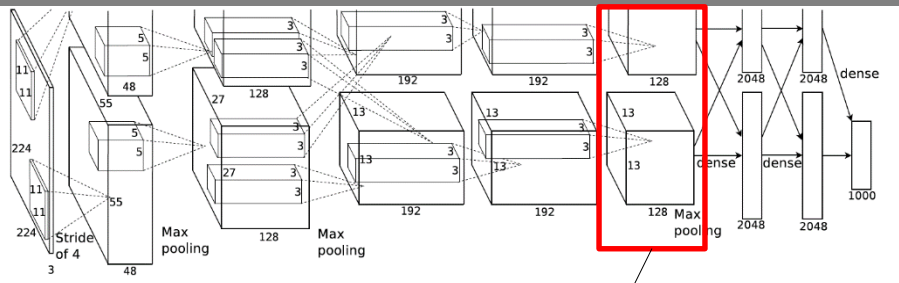
  'Statistical Realism'

  To solve this, we can compute

  $$\frac{\partial}{\partial \boldsymbol{I}} \left(L(\hat{\boldsymbol{I}}, \boldsymbol{I}) + \rho P(\boldsymbol{I}) + \lambda \|\boldsymbol{I}\|^2 \right)$$

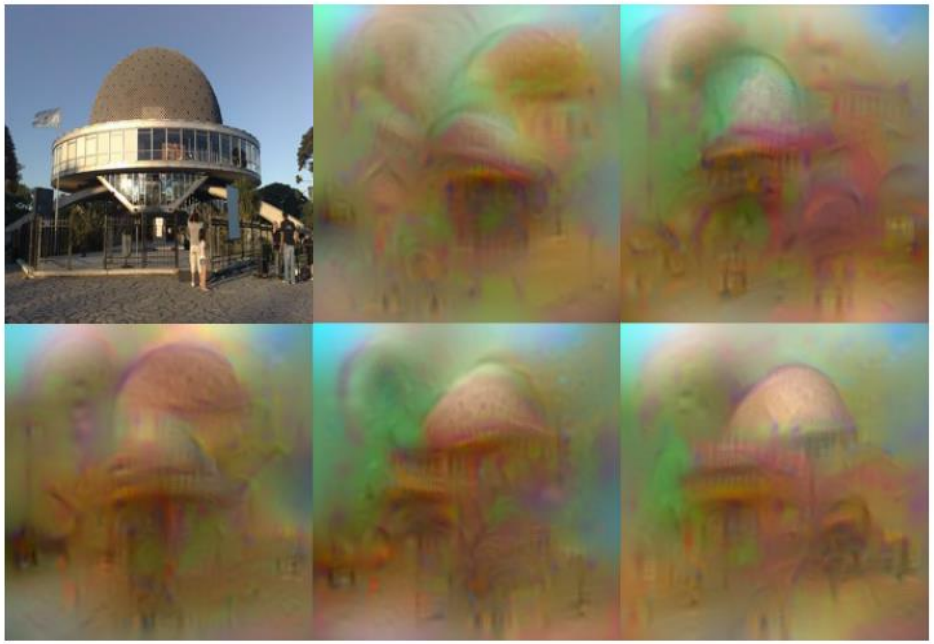
  and apply a gradient descent procedure, starting from a random image $\boldsymbol{I}^{(0)}$

Reconstructing Images from Feature Maps



$\Phi_{k,l}(\hat{I})$ *is taken here*

This is $\hat{I}$



The remaining five images were generated using image space gradient descent with different initial images $I^{(0)}$

[images from https://arxiv.org/pdf/1412.0035v1.pdf]

Can DCNNs 'dream'?

sign in | become a supporter | subscribe | search | jobs | dating | more | International edition

theguardian

UK | world | sport | football | opinion | culture | business | lifestyle | fashion | environment | tech | travel | browse all sections

home › tech

Artificial intelligence (AI)

Yes, androids do dream of electric sheep

Google sets up feedback loop in its image recognition neural network - which looks for patterns in pictures - creating hallucinatory images of animals, buildings and landscapes which veer from beautiful to terrifying

This article is 1 year old

109,591 | 445

Alex Hern

@alexhern

Thursday 18 June 2015 12.57 BST

A hallucinatory filter over a red tree. Spot the animals. Photograph: Google

Advertisement

Office 365 Business

Passa a Outlook 2016 e accedi alla posta ovunque, online e offline.

Outlook

A soli €10,70 al mese*

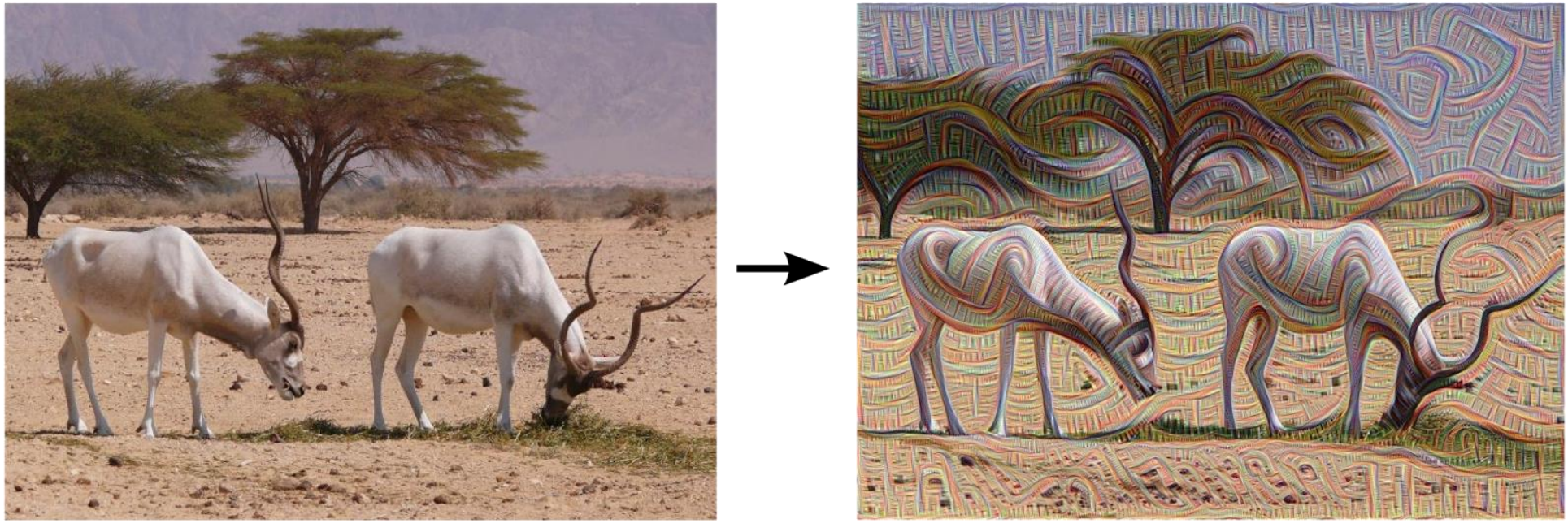
*Il prezzo non include l'IVA

Microsoft

Scopri di più

Can DCNNs 'dream'?

Enhancing lower layers



[images from https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html]

Feature Enhancement

- **Image Space Gradient Descent**

  Define

  $$\boldsymbol{\Phi}_{k,l}(\boldsymbol{I})$$

  as the response of a DCNN at a layer k, filter l to an image $\boldsymbol{I}$

  Given a specific image $\hat{\boldsymbol{I}}$, we define the loss function

  $$L(\hat{\boldsymbol{I}}, \boldsymbol{I}) := \|\gamma\, \boldsymbol{\Phi}_{k,l}(\hat{\boldsymbol{I}}) - \boldsymbol{\Phi}_{k,l}(\boldsymbol{I})\|^2$$

  Amplification factor

  The optimization problem

  $$\boldsymbol{I}^* := \operatorname{argmin}_{\boldsymbol{I}} \left(L(\hat{\boldsymbol{I}}, \boldsymbol{I}) + \lambda \|\boldsymbol{I}\|^2 \right)$$

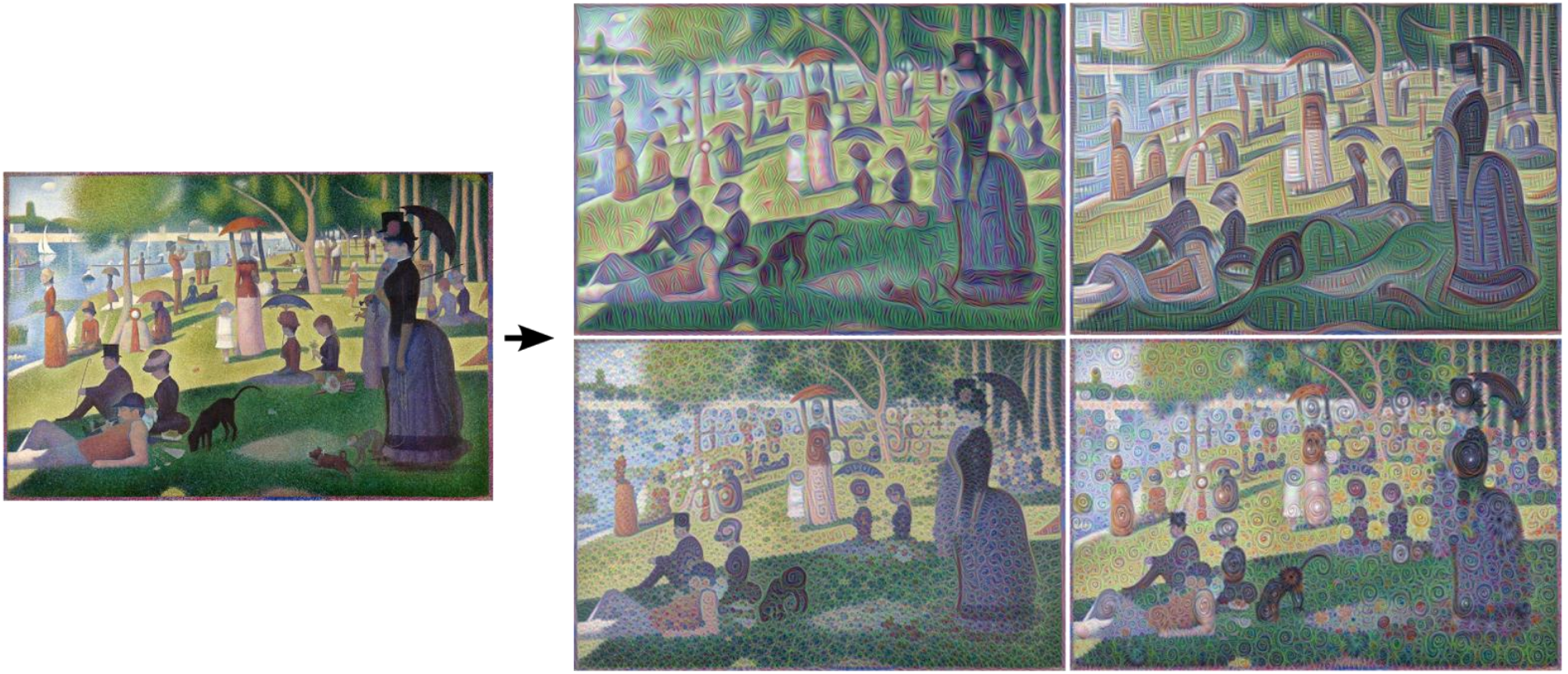
  is solved via gradient descent by computing

  $$\frac{\partial}{\partial \boldsymbol{I}} \left(L(\hat{\boldsymbol{I}}, \boldsymbol{I}) + \lambda \|\boldsymbol{I}\|^2 \right)$$

  and starting from $\boldsymbol{I}^{(0)} = \hat{\boldsymbol{I}}$

Can DCNNs 'dream'?

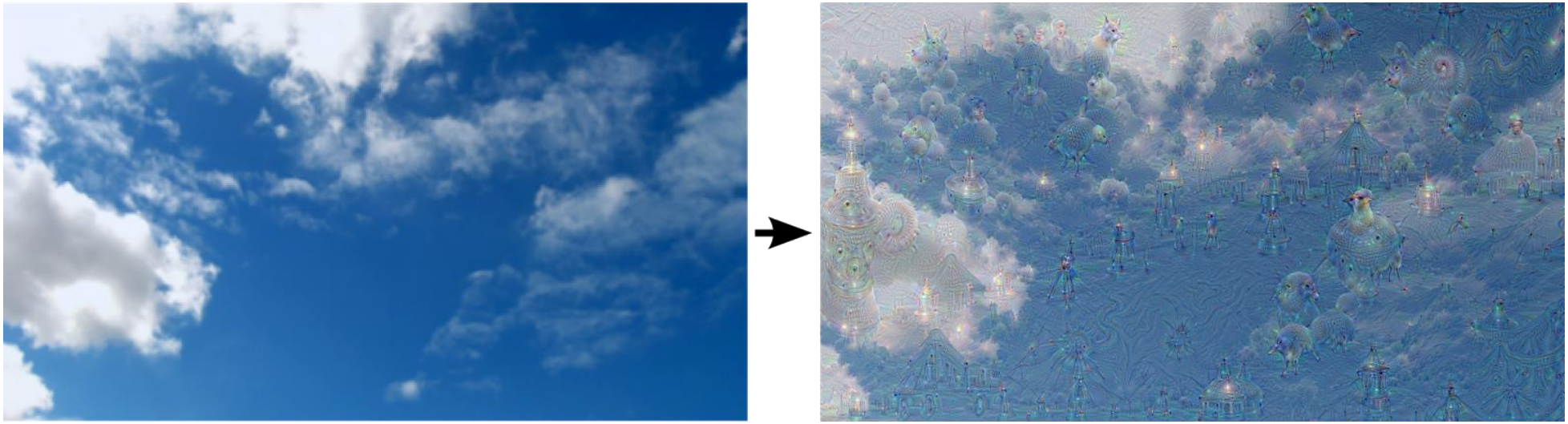
Enhancing lower layers



[images from https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html]

Can DCNNs 'dream'?

Enhancing upper layers



[images from https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html]

Can DCNNs 'dream'?

Letting the DCNN go on its own

[images from https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html]

Can DCNNs 'dream'?

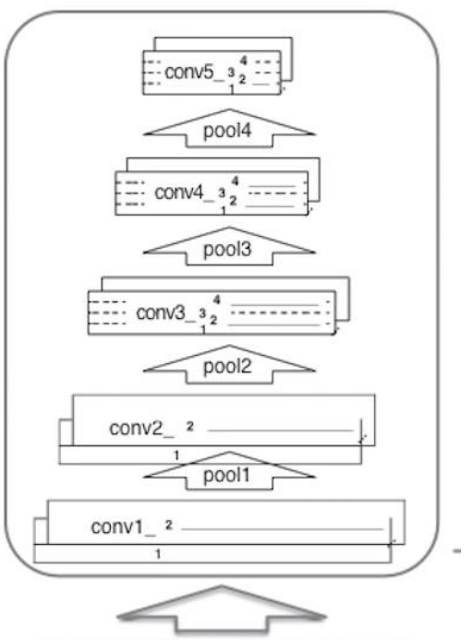
Letting the DCNN go on its own

[images from https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html]

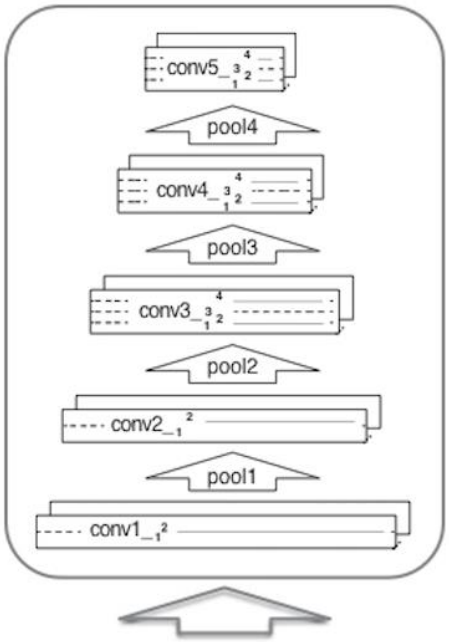
The Power of Abstraction

- ***Different Layers of a Deep Convolutional Neural Network***

  What kind of information does each layer 'store'?



'Content' image



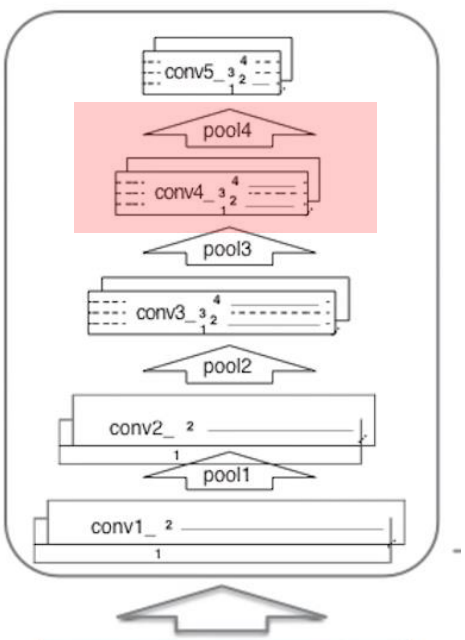
'Style' image

[images from https://www.youtube.com/watch?v=N14_w2RG1A8]

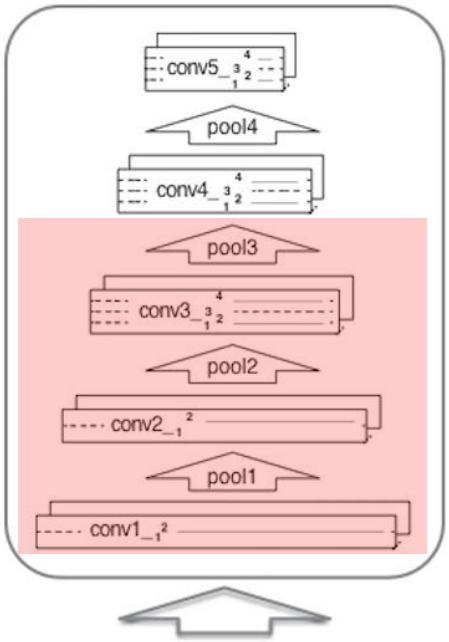
The Power of Abstraction

- ***Different Layers of a Deep Convolutional Neural Network***

  What kind of information does each layer 'store'?



'Content' image



'Style' image

Create a new image by combining more of the 'Content' top layer and more of 'Style' low layers

[images from https://www.youtube.com/watch?v=N14_w2RG1A8]

Mixing Two Images

- **Image Space Gradient Descent**

  Define

  $$\boldsymbol{\Phi}_{k,l}(\boldsymbol{I})$$

  as the response of a DCNN at a layer k, filter l to an image $\boldsymbol{I}$

  Given a specific image $\hat{\boldsymbol{I}}_1$ and $\hat{\boldsymbol{I}}_2$, we define the loss function

  $$L(\hat{\boldsymbol{I}}, \boldsymbol{I}) := \sum_{k,l} \| \boldsymbol{M}_{k,l}(\boldsymbol{\Phi}_{k,l}(\hat{\boldsymbol{I}}_2), \boldsymbol{\Phi}_{k,l}(\hat{\boldsymbol{I}}_1)) - \boldsymbol{\Phi}_{k,l}(\boldsymbol{I}) \|^2$$

  Weighted Merge Function

  The optimization problem

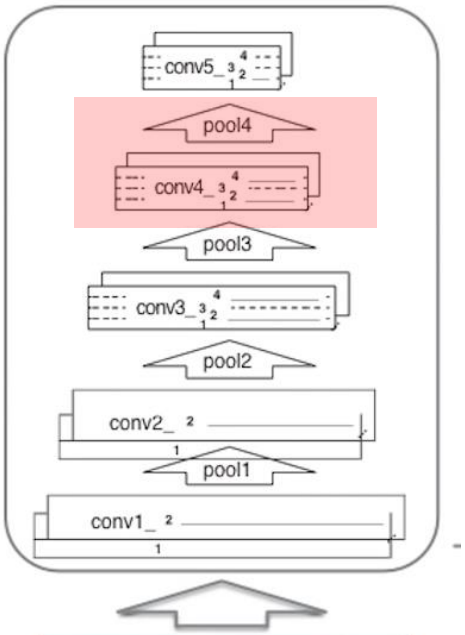
  $$\boldsymbol{I}^* := \operatorname{argmin}_{\boldsymbol{I}} \left(L(\hat{\boldsymbol{I}}, \boldsymbol{I}) + \lambda \|\boldsymbol{I}\|^2 \right)$$

  is solved via gradient descent starting from $\boldsymbol{I}^{(0)} = \hat{\boldsymbol{I}}_1$

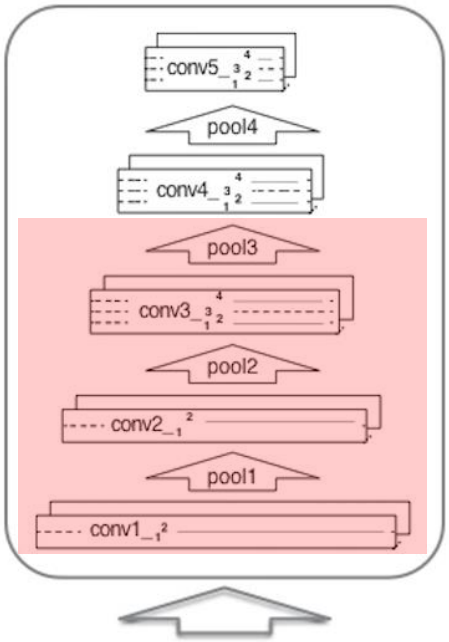
The Power of Abstraction

- ***Different Layers of a Deep Convolutional Neural Network***

  What kind of information does each layer 'store'?



'Content' image



'Style' image

Create a new image by combining more of the 'Content' top layer and more of 'Style' low layers

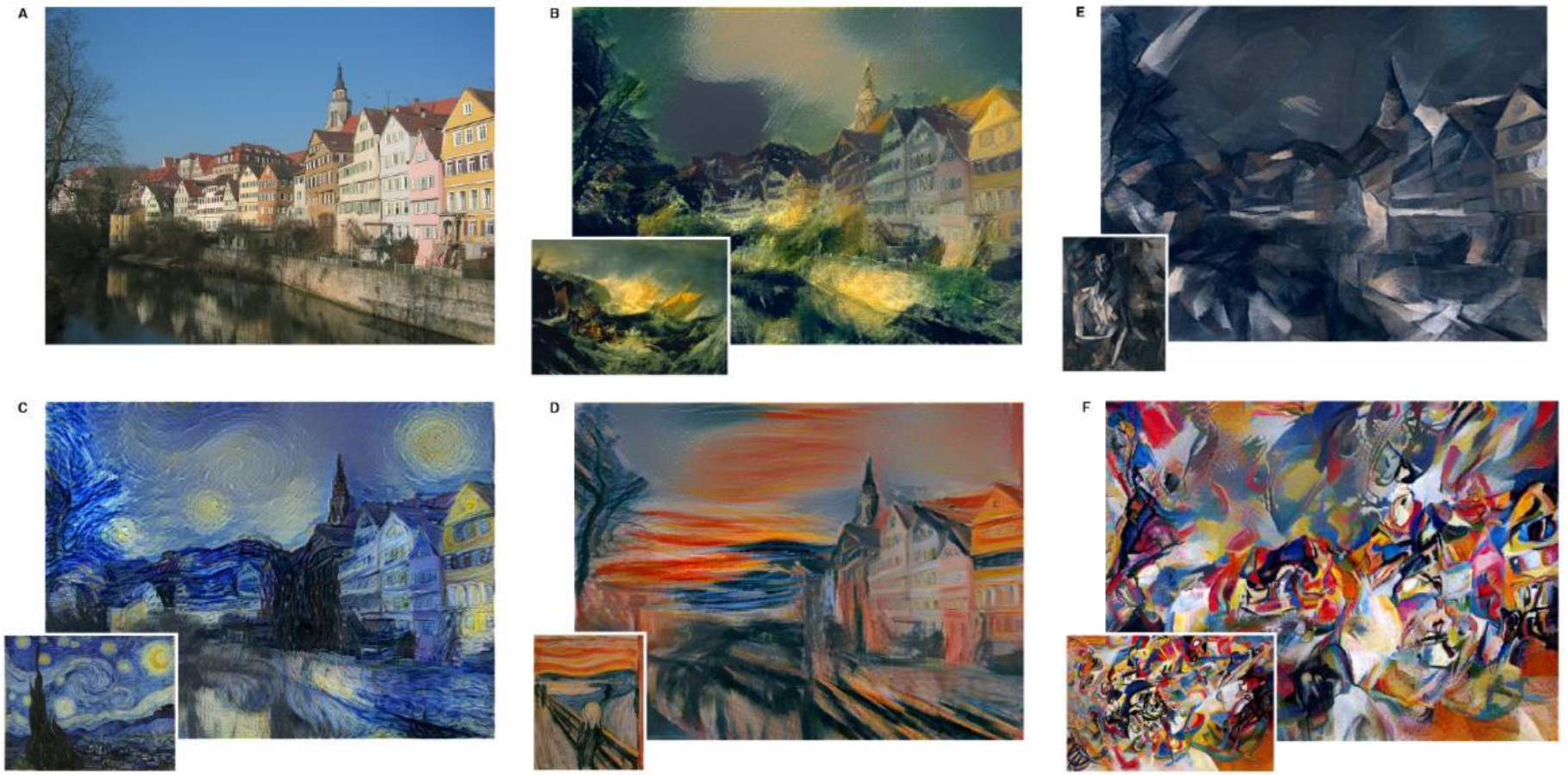
This is the result

[images from https://www.youtube.com/watch?v=N14_w2RG1A8]

The Power of Abstraction

- ***Different Layers of a Deep Convolutional Neural Network***

  Further examples:



[images from https://arxiv.org/pdf/1508.06576.pdf]

Reproducing Non-Existing Masterpieces

[figures from the website]

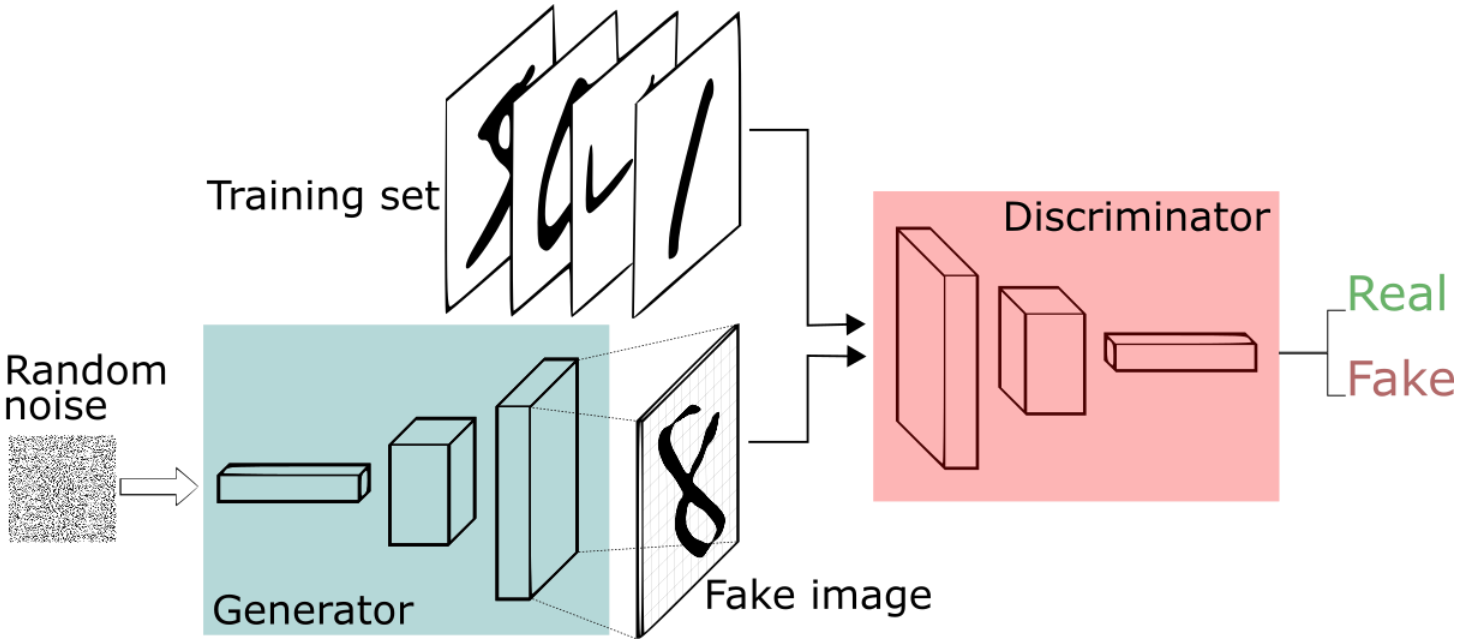
Next Rembrandt project [https://www.nextrembrandt.com/, 2016].

No Free Lunch:
having an annotated dataset

Generative Adversarial Network

- ***Two competing networks***
  a) A *discriminator* learns to classify images while detecting fake ones
  b) A *generator* learns how to fool the discriminator

  Each network is trained in turn, while keeping the other fixed

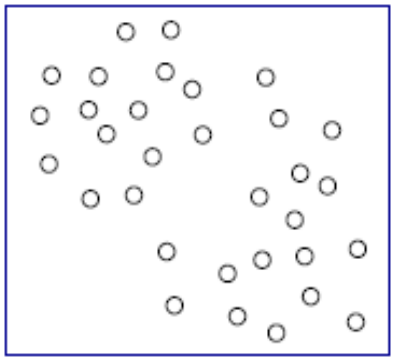


[image from https://www.oreilly.com/learning/generative-adversarial-networks-for-beginners]

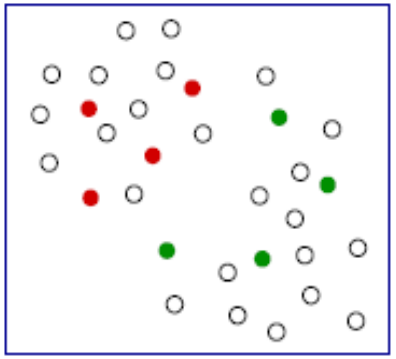
Active Learning

When the network decides which annotations should be made

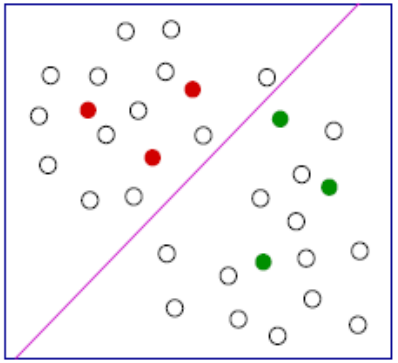
1) Consider a large non-annotated *dataset*



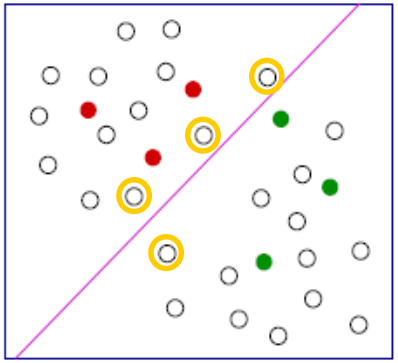
2) Annotate a few images at random



3) Train a classifier on annotated images only



4) Annotate borderline cases

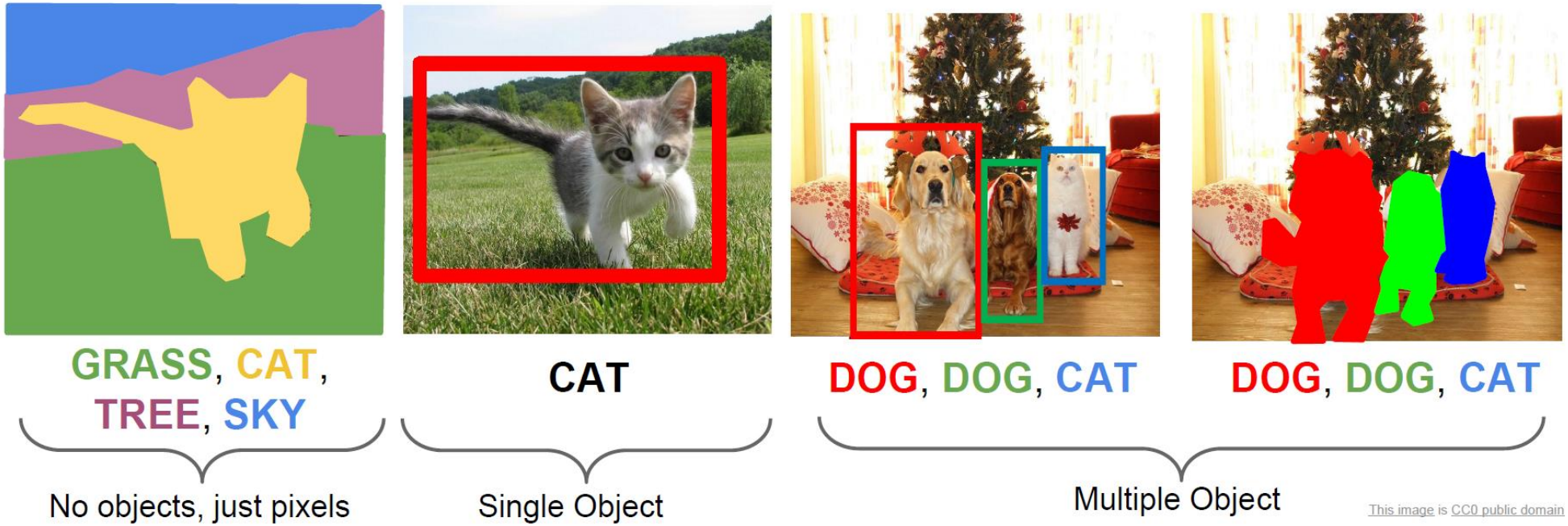


5) Repeat training

Image Classification
Object Detection
Segmentation

Deep Learning for different imaging tasks

Beyond simple image classification



[images from http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf]

Semantic segmentation

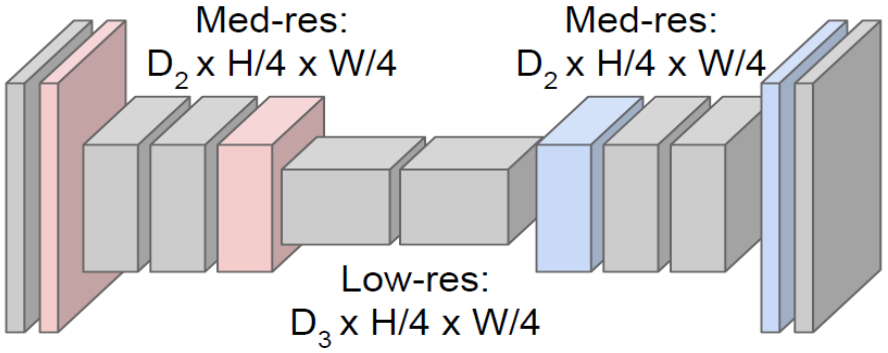
Beyond simple image classification

- ***Similar network architecture, different arrangement***

  Fully Convolutional Networks (FCN)

  Downsampling first, *upsampling* afterwards

Input:
3 x H x W



High-res:
D1 x H/2 x W/2

High-res:
D1 x H/2 x W/2

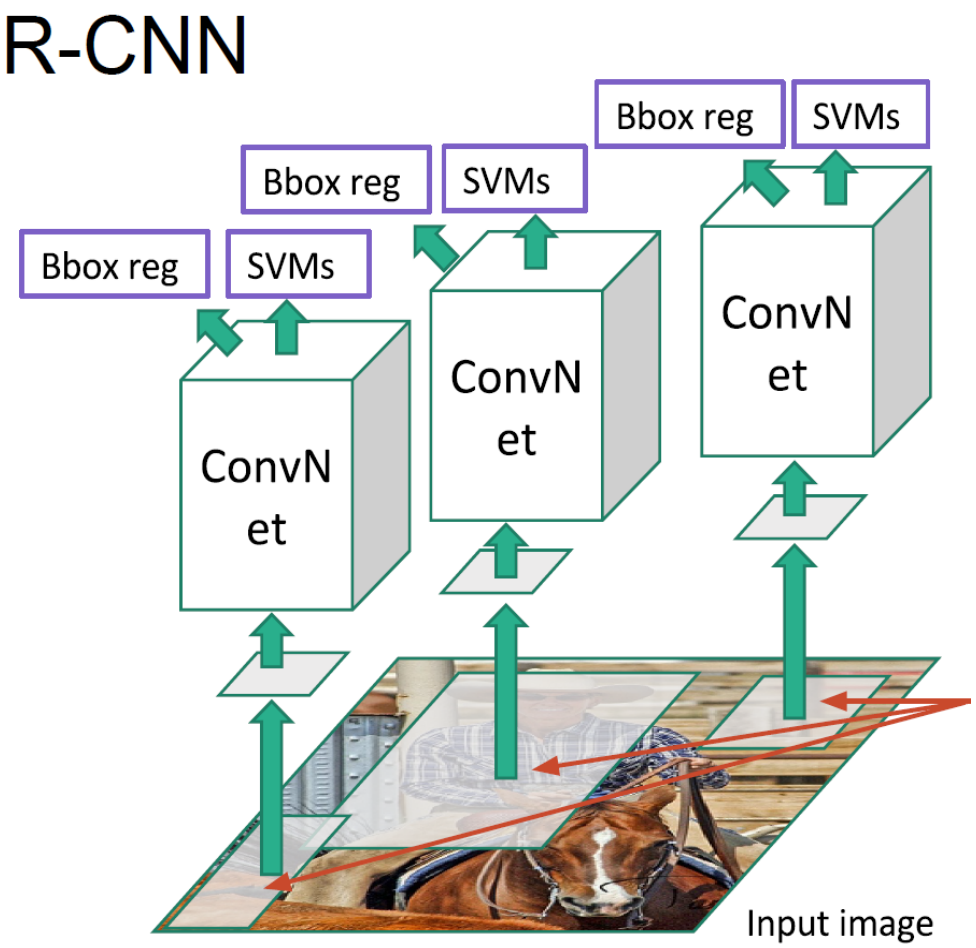
Predictions:
H x W

[images from http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf]

Object detection and positioning

Generate boxes and classifications

- ***Two-stage Process***
  - *Generate bounding box candidates*
  - *Pass each candidate through a DCNN*
  - *Select those candidates that are classified with higher certainty*



[images from http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf]

Object detection and positioning

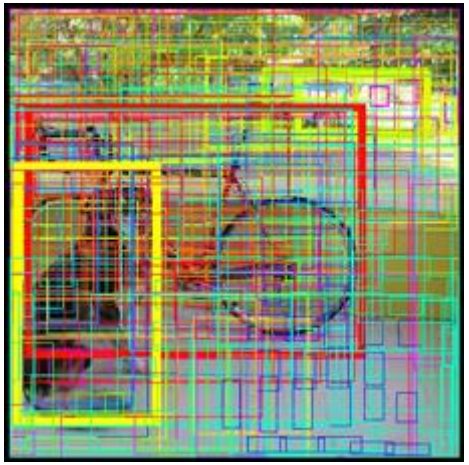
- ***YOLO: a unique convolutional network***

  Generate boxes and classifications <u>at once</u>

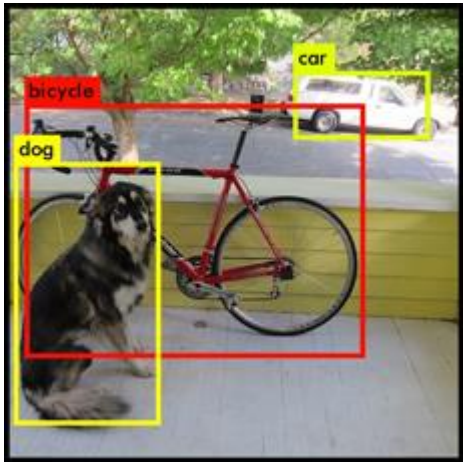
1) Impose a fixed grid over the input image

2) Generate possible bounding boxes

3) Classify each of them



4) Keep the boxes at highest confidence



[images from http://machinethink.net/blog/object-detection-with-yolo/]