UNIVERSITAT POLITÈCNICA DE VALÈNCIA

MIARFID Official Master's Degree in Artificial Intelligence, Pattern Recognition and Digital Imaging

Chapter 4. Deep Learning

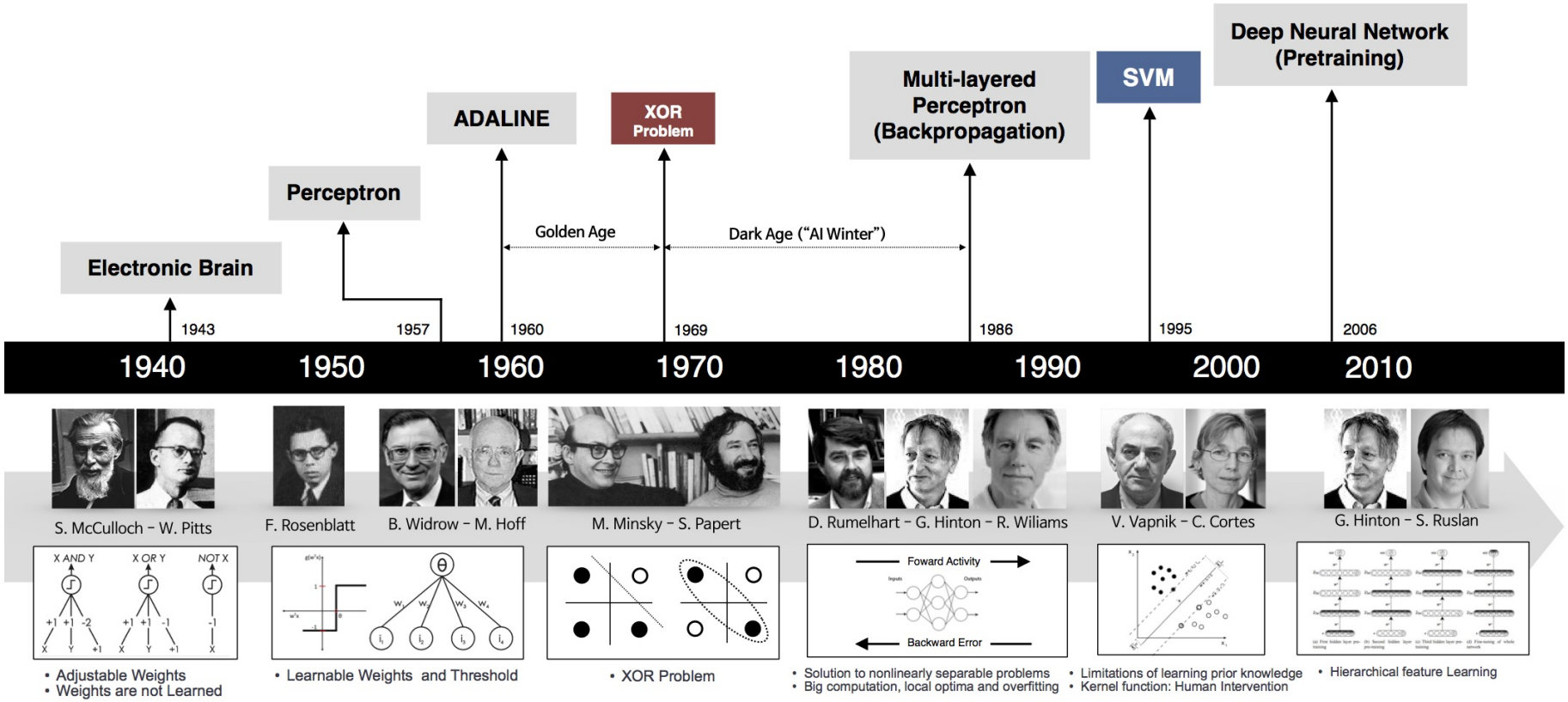
Neural Networks

2023/2024

Máster Universitario en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

Departamento de Sistemas Informáticos y Computación

NN Timeline


Deep Neural Network (Pretraining)
SVM
Multi-layered Perceptron (Backpropagation)
XOR Problem
ADALINE
Perceptron
Electronic Brain
Golden Age
Dark Age ("AI Winter")
1943
1957
1960
1969
1986
1995
2006
1940
1950
1960
1970
1980
1990
2000
2010
S. McCulloch - W. Pitts
F. Rosenblatt
B. Widrow - M. Hoff
M. Minsky - S. Papert
D. Rumelhart - G. Hinton - R. Wiliams
V. Vapnik - C. Cortes
G. Hinton - S. Ruslan
X AND Y
X OR Y
NOT X
Foward Activity
Backward Error
• Adjustable Weights
• Weights are not Learned
• Learnable Weights and Threshold
• XOR Problem
• Solution to nonlinearly separable problems
• Big computation, local optima and overfitting
• Limitations of learning prior knowledge
• Kernel function: Human Intervention
• Hierarchical feature Learning

Index

Index

Feed Forward Networks

- Complex structures
- Forward connections (strictly)
- Add layers to the original Perceptron architecture
- Activation units to overcome the linear limitations
- Target outputs

Deep Feedforward Models

- Goal: perform representation learning
- Solution: deep structures
- Problems arise:
  - Optimization: local minima and random weights
  - Generalization: too many parameters
  - Computation: too expensive

Deep Feedforward Models

- Goal: perform representation learning
- Solution: deep structures
- Problems arise:
  - Optimization: local minima and random weights
  - Generalization: too many parameters
  - **Computation: too expensive** $\rightarrow$ GPUs

Deep Feedforward Models

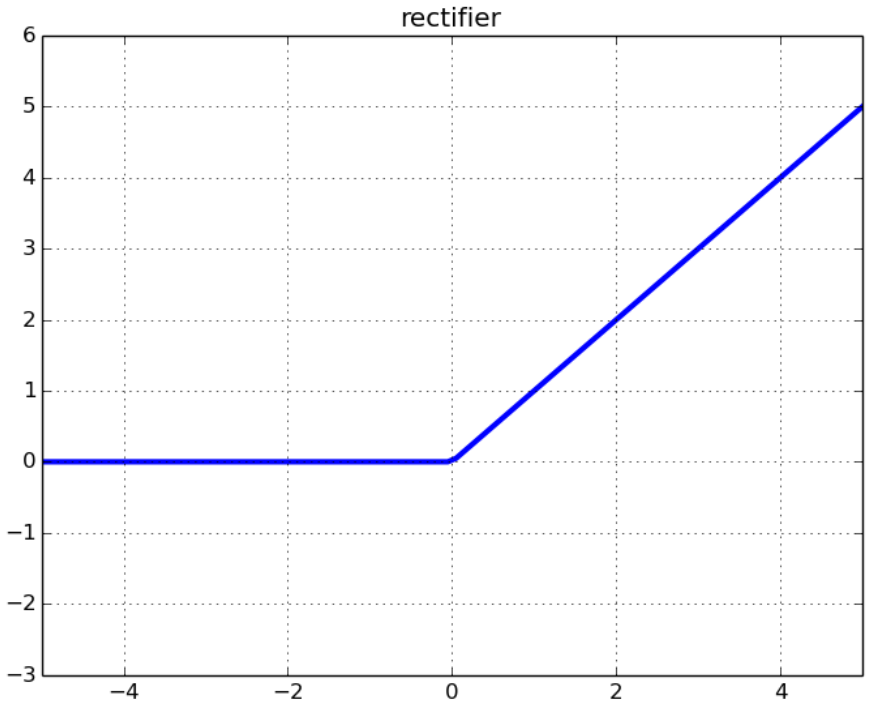
- Goal: perform representation learning
- Solution: deep structures
- Problems arise:
  - **Optimization: local minima and random weights**
  - Generalization: too many parameters
  - Computation: too expensive

Some solutions. Local minima

- Partially solved with Deep Belief Networks (Stacked RBM)
- Solutions:
  - Rectifier Linear Units
  - Max-out Units

ReLu activation function

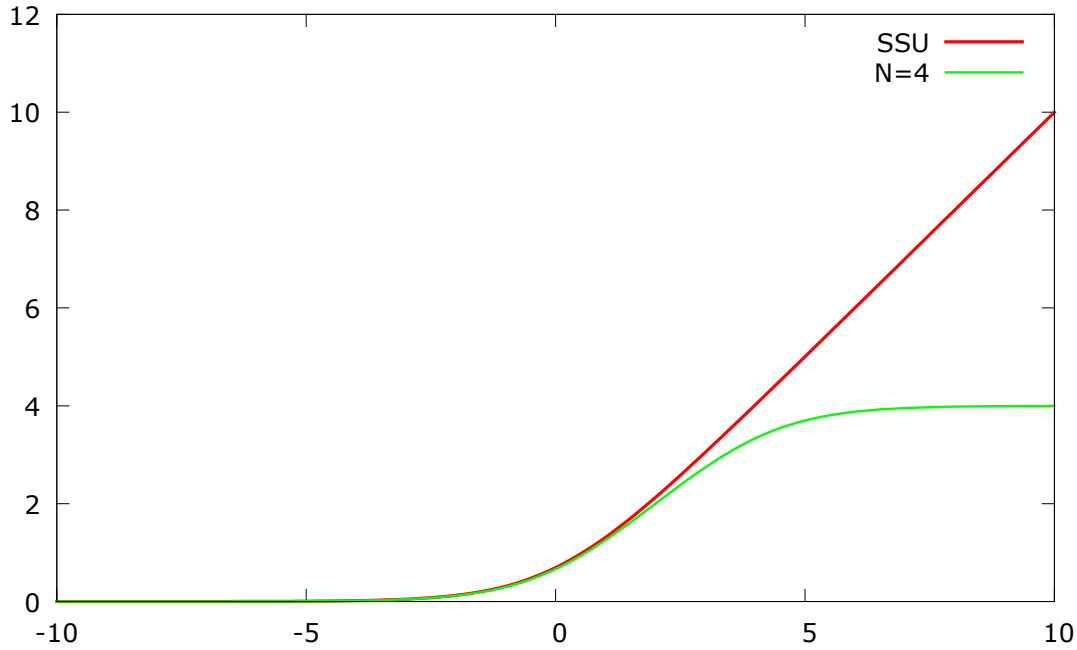
- $f(z) = max(0, z)$
- $f'(z) = 1$ if $z > 0$
- $f'(z) = 0$ if $z \leq 0$



ReLu activation function

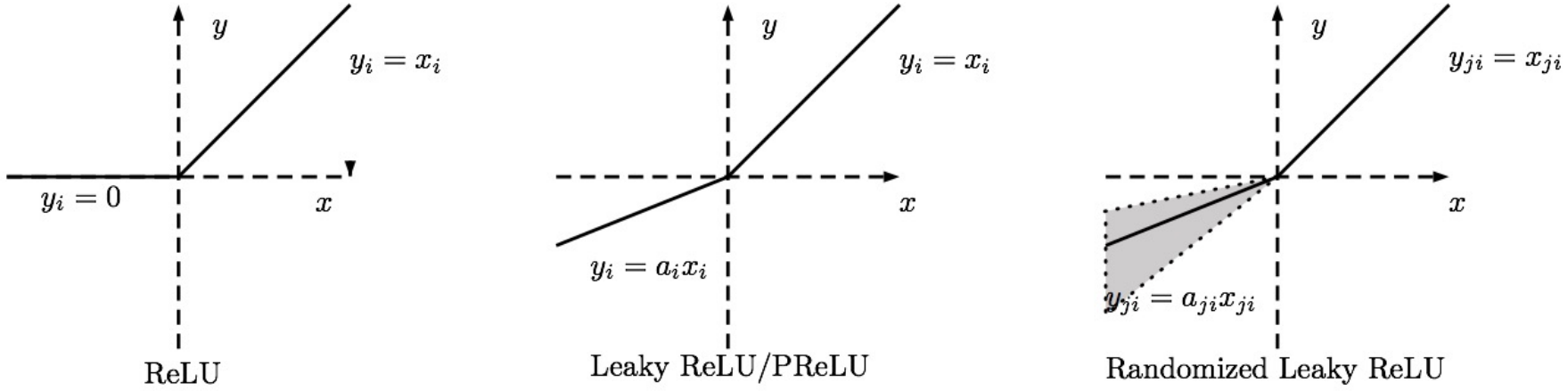
- Inspired on this the stepped sigmoid units (SSU):

$$\sum_{i=1}^{N} \sigma(z - i + 0.5) \approx log(1 + e^z)$$



ReLu variations

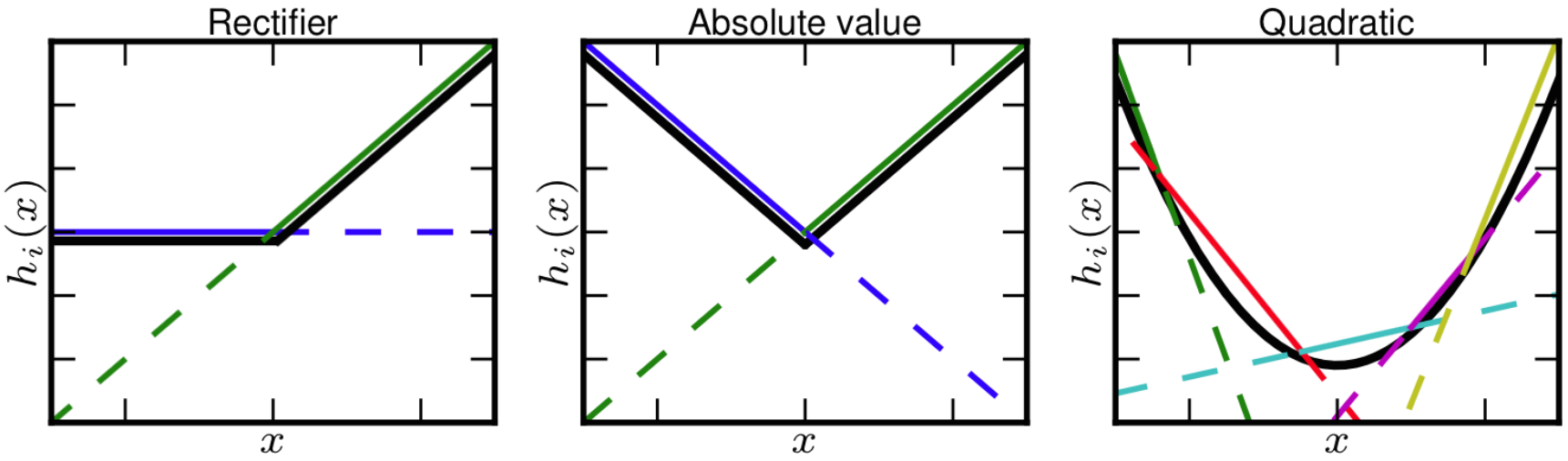
- ReLu, Leaky ReLu, Parametric ReLu and Random ReLu:



https://arxiv.org/pdf/1505.00853.pdf

Max-out

- Activation function is $max_k(W_k x)$
- Take the maximum of linear functions
- The connection between layers are tensors (W_{kij})



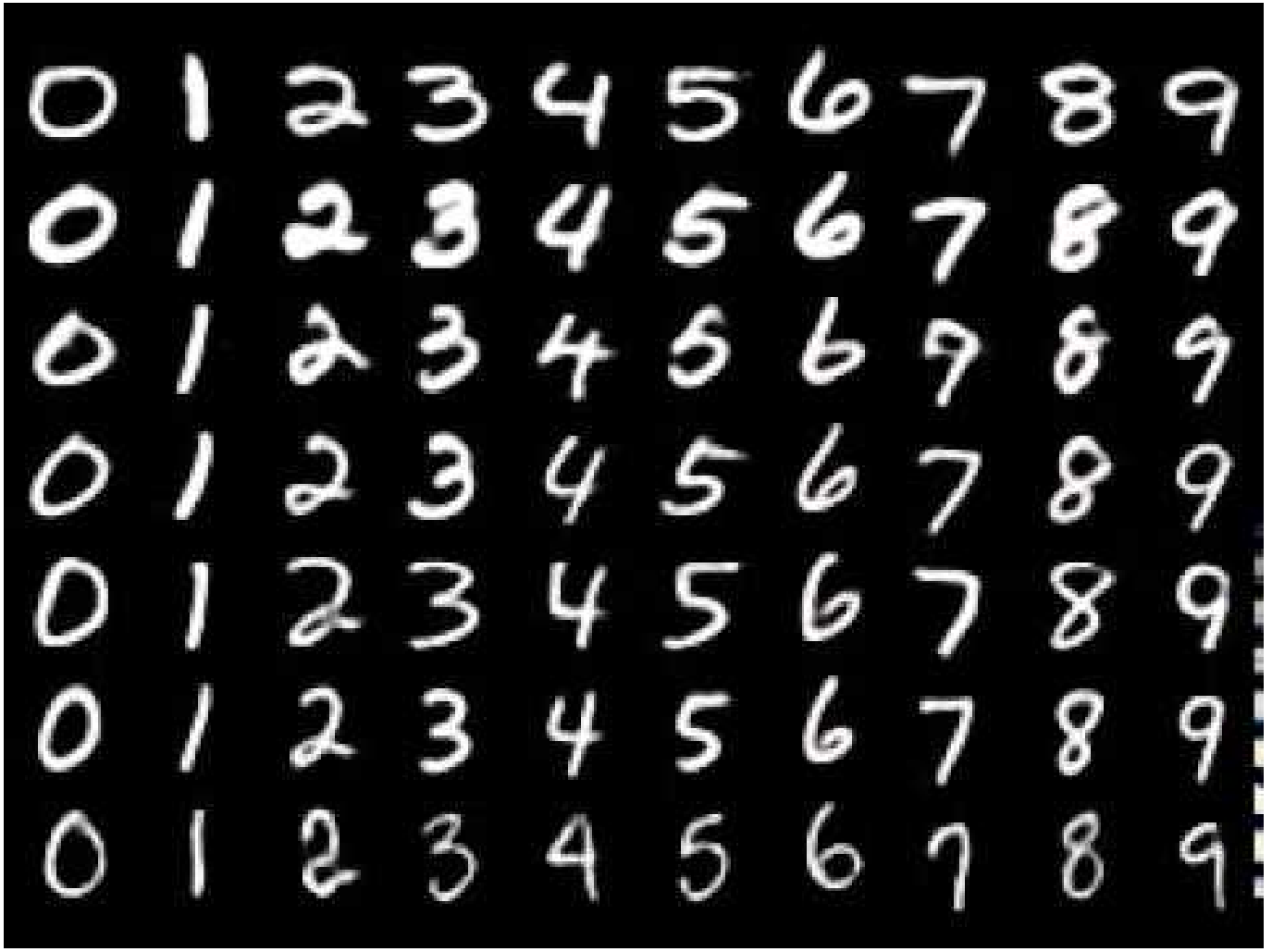
Deep Feedforward Models

- Goal: perform representation learning
- Solution: deep structures
- Problems arise:
  - Optimization: local minima and random weights
  - **Generalization: too many parameters**
  - Computation: too expensive

Some solutions. Generalization

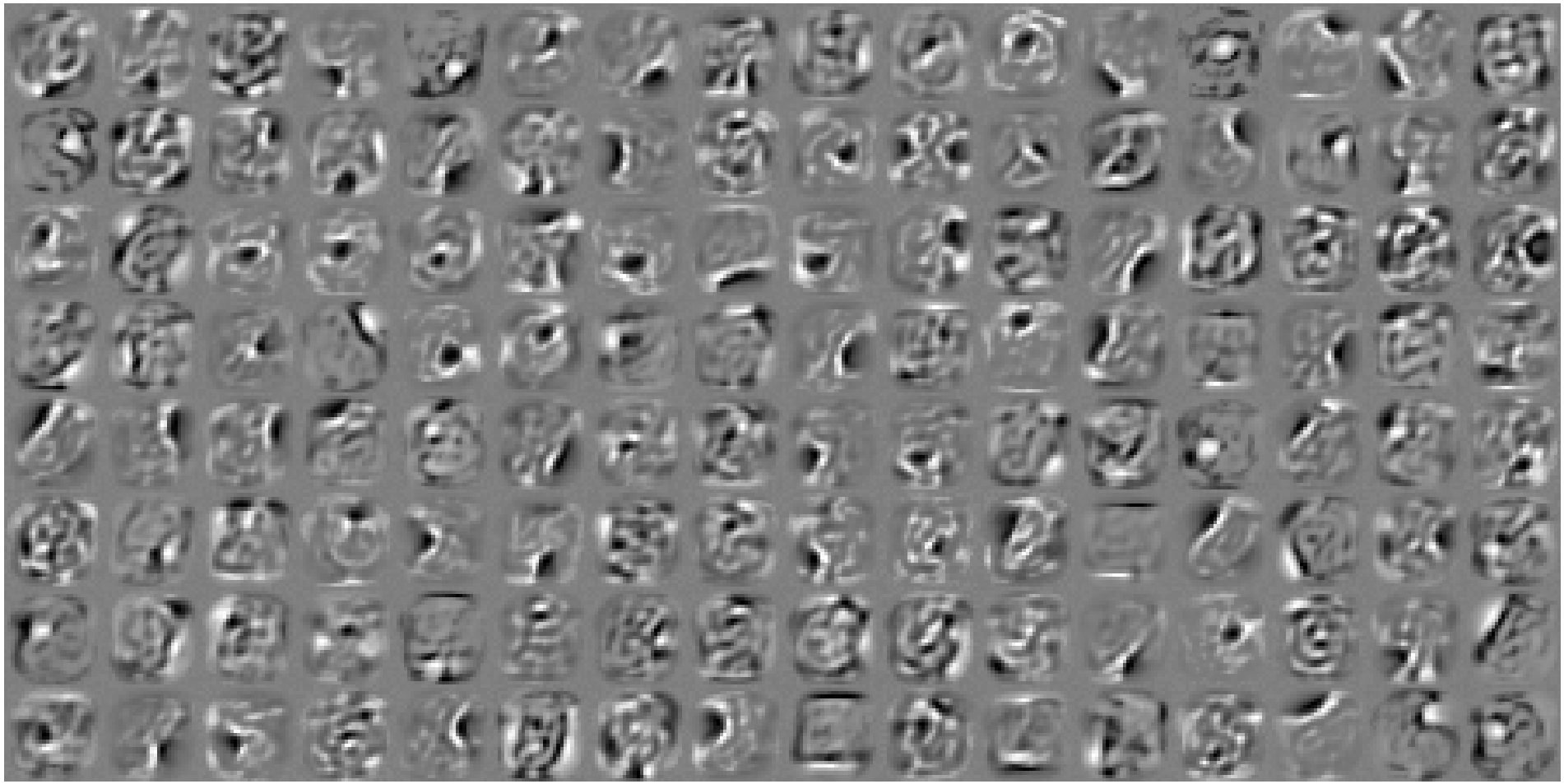
- Partially solved with Deep Belief Networks (Stacked RBM)
- Potential solutions:
  - Regularization penalty
  - Maxnorm regularization
  - Sparsity regularization
  - Noise regularization
  - Dropout
  - Batch-norm

Examples MNIST



Examples MNIST

- MLP one hidden layer of 128 dim



Regularization

- Weight Regularization:

  $E_A(\mathbf{w}) = \frac{1}{2n}\sum_{p=1}^{n}\sum_{k=1}^{N_2}\left(t_{p,k} - s_k^2(\mathbf{x}_p)\right)^2 + \lambda \mid\mid W \mid\mid_r^r$

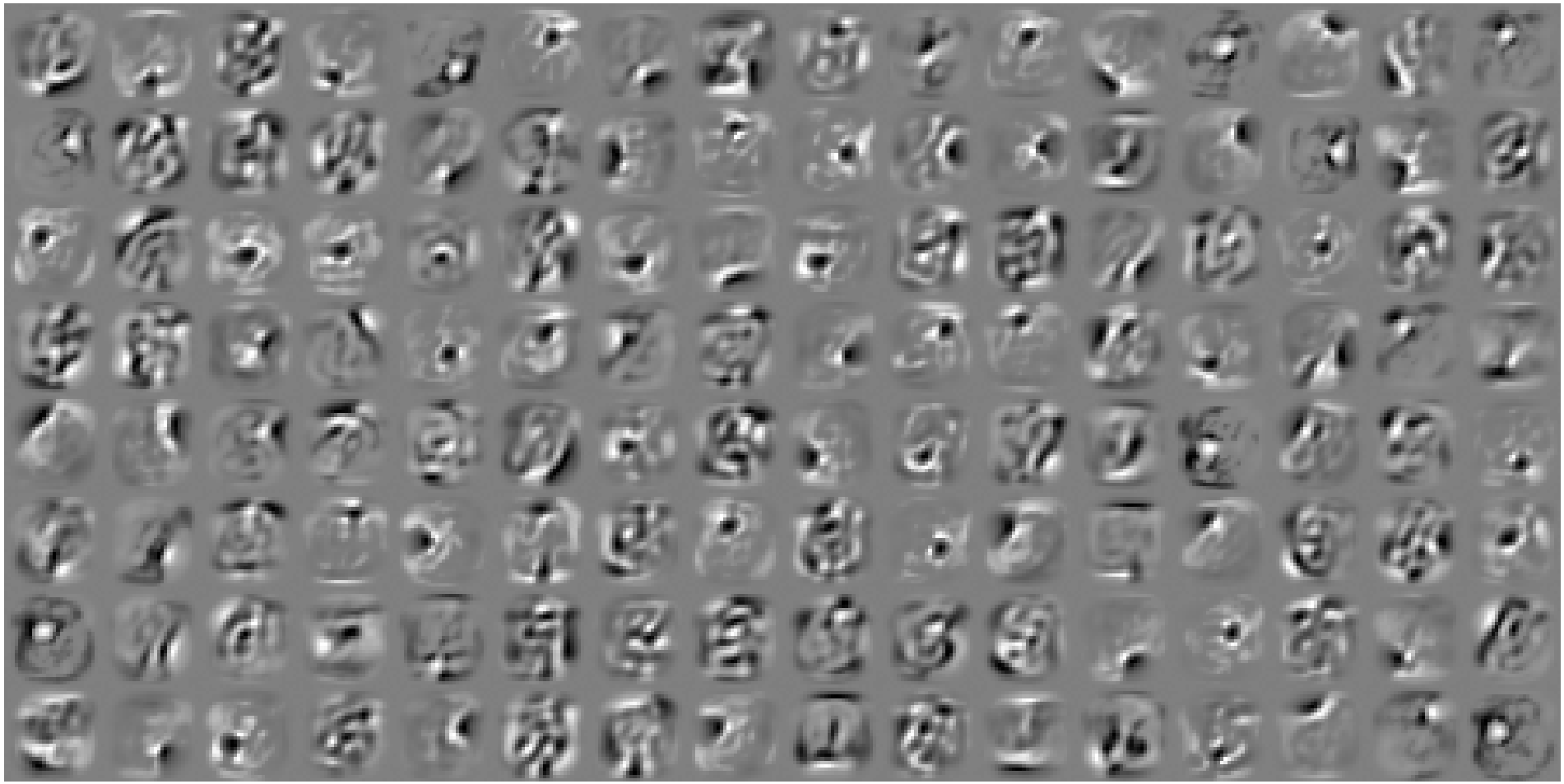
  where $\mid\mid W \mid\mid_r = \left(\sum_{j=1}^{|W|} \mid w_j \mid^r\right)^{1/r}$

  - L2 Regularization ("weight decay"): $\mid\mid W \mid\mid_2^2 = \sum_{j=1}^{|W|} w_j^2$
  - L1 Regularization: $\mid\mid W \mid\mid_1 = \sum_{j=1}^{|W|} \mid w_j \mid$

- Gradients:
  - L2: λw_j
  - L1: $\lambda sign(w_j)$ (no derivative at 0)

Examples MNIST

- MLP one hidden layer of 128 dim with L2 regularization



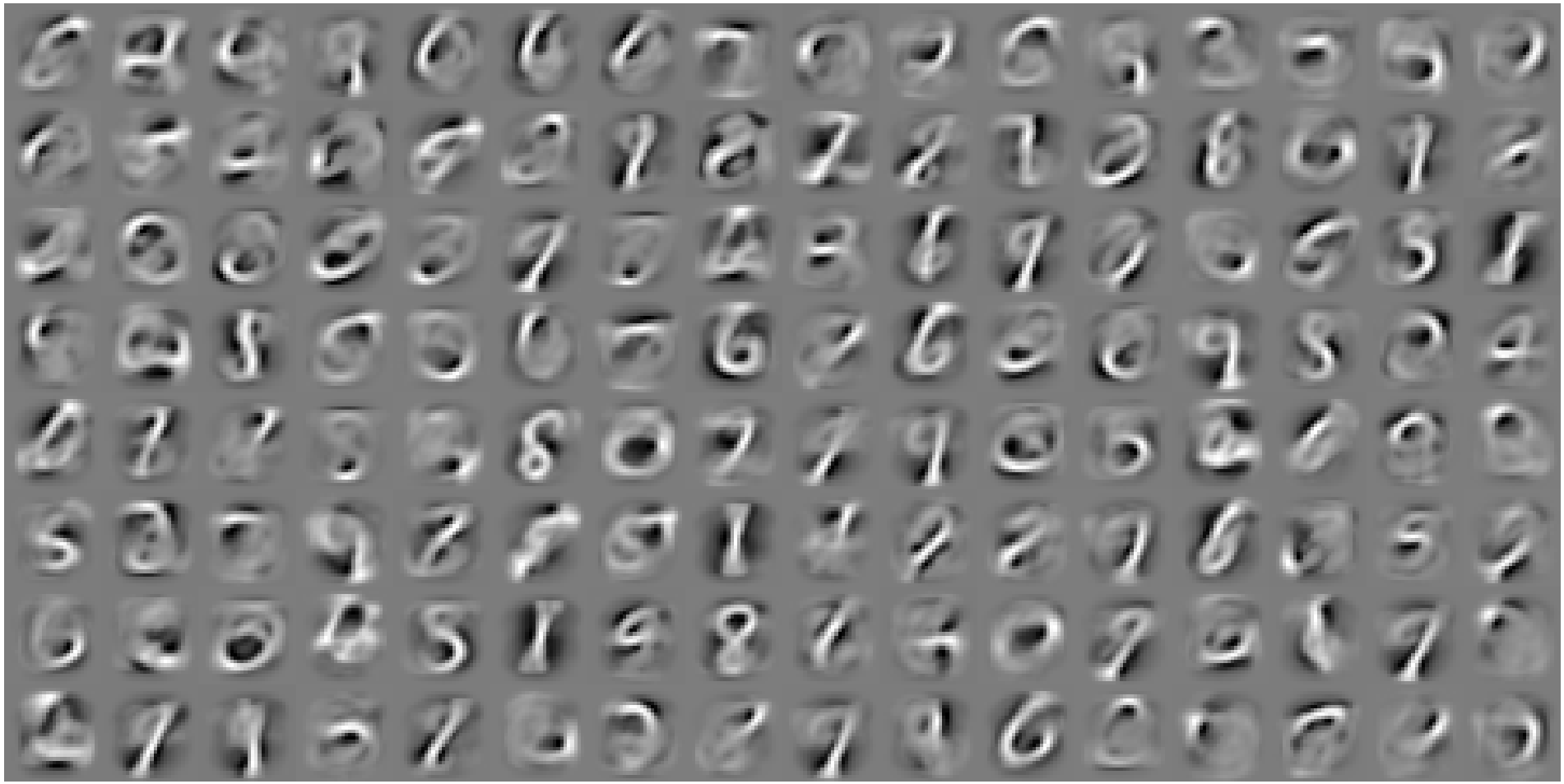
Sparsity regularization

- Sparsity is good, force hidden units to be specialized
- Sparsity means that only few hidden units are "activated"
- Set a sparsity target p
- Compute the average activation per hidden unit q_i
- Introduce a cross-entropy penalty:

$$-p\ log(q_i)\ -\ (1-p)log(1-q_i)$$

Examples MNIST

- MLP one hidden layer of 128 dim L2 regularization and **Sparsity**



MaxNorm Regularization

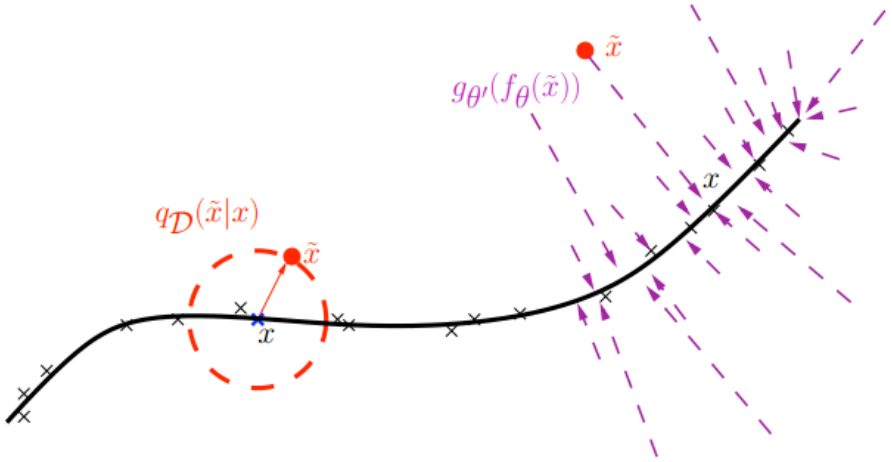
- Maxnorm is generally used with ReLu
- Maxnorm limits the magnitude of the weight parameter
- Projects the weight vector to the hypersphere of radius c

$$W = \begin{cases} W & \text{if } || W ||_2 < c \\ \frac{cW}{||W||_2} & otherwise \end{cases}$$

- Common values $c = \{2, 3\}$

Noise regularization

- Add some noise to the activation function
- The goal is to avoid very "local" solutions (overfitting)
- Usually a normal noise is used $N(0,1)$. Best results combined with batch-norm



Dropout

- A simple way to prevent neural networks from overfitting

  http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

- Essentially a random variable $r_j^l \sim Bernoulli(p)$ controls the activation of neuron j of layer l:

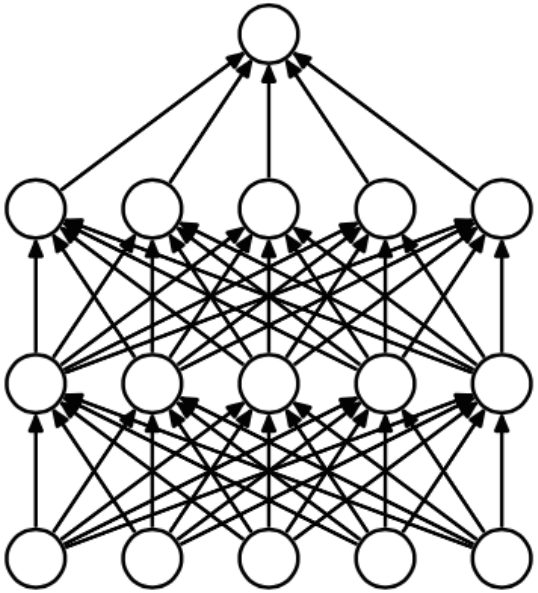
  Then, $\hat{s}_j^l = r_j s_j^l$ is the new output of neuron j of layer l

- The idea behind is the "model combination"

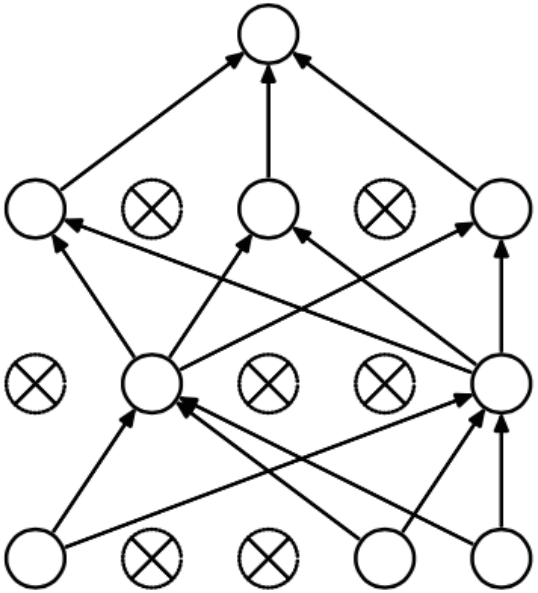
- Only in hidden and visible units, these units must be more *generally* useful

- In test time multiply the outgoing weights by the prob. that the neuron was retained

Dropout



(a) Standard Neural Net



(b) After applying dropout.

Batch-Norm

- Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.

http://proceedings.mlr.press/v37/ioffe15.pdf

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

$$\frac{\partial \ell}{\partial \widehat{x}_i} = \frac{\partial \ell}{\partial y_i} \cdot \gamma$$

$$\frac{\partial \ell}{\partial \sigma^2_{\mathcal{B}}} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot (x_i - \mu_{\mathcal{B}}) \cdot \frac{-1}{2}(\sigma^2_{\mathcal{B}} + \epsilon)^{-3/2}$$

$$\frac{\partial \ell}{\partial \mu_{\mathcal{B}}} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{-1}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}}$$

$$\frac{\partial \ell}{\partial x_i} = \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{1}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} + \frac{\partial \ell}{\partial \sigma^2_{\mathcal{B}}} \cdot \frac{2(x_i - \mu_{\mathcal{B}})}{m} + \frac{\partial \ell}{\partial \mu_{\mathcal{B}}} \cdot \frac{1}{m}$$

$$\frac{\partial \ell}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i} \cdot \widehat{x}_i$$

$$\frac{\partial \ell}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i}$$

Batch-Norm, Where?

- Where to apply batch-norm, before or after activation function?
  - https://www.reddit.com/r/MachineLearning/comments/67gonq/d_batch_normalization_before_or_after_relu/
  - https://github.com/ducha-aiki/caffenet-benchmark/blob/master/batchnorm.md

BN -- before or after ReLU?

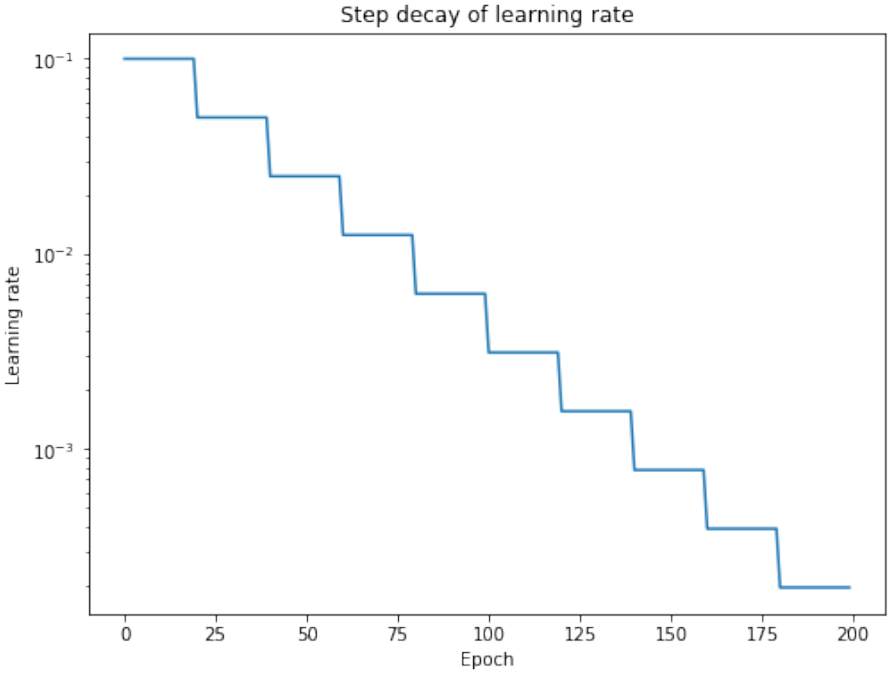
Name	Accuracy	LogLoss	Comments
Before	0.474	2.35	As in paper
Before + scale&bias layer	0.478	2.33	As in paper
After	**0.499**	**2.21**	
After + scale&bias layer	0.493	2.24	

Other tricks

- Improve results with:
  - Learning rate annealing
  - Modifying the batch size
  - Data augmentation

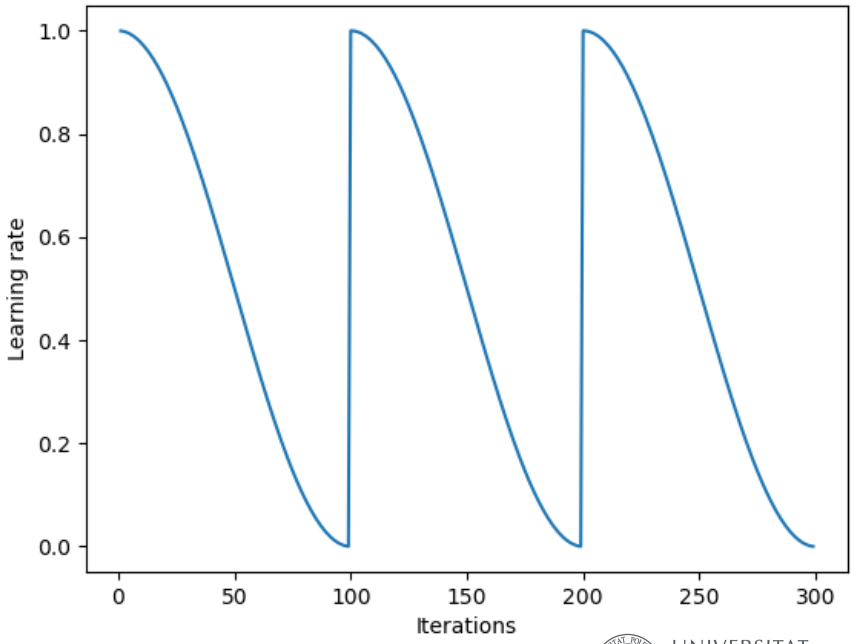
Learning rate annealing

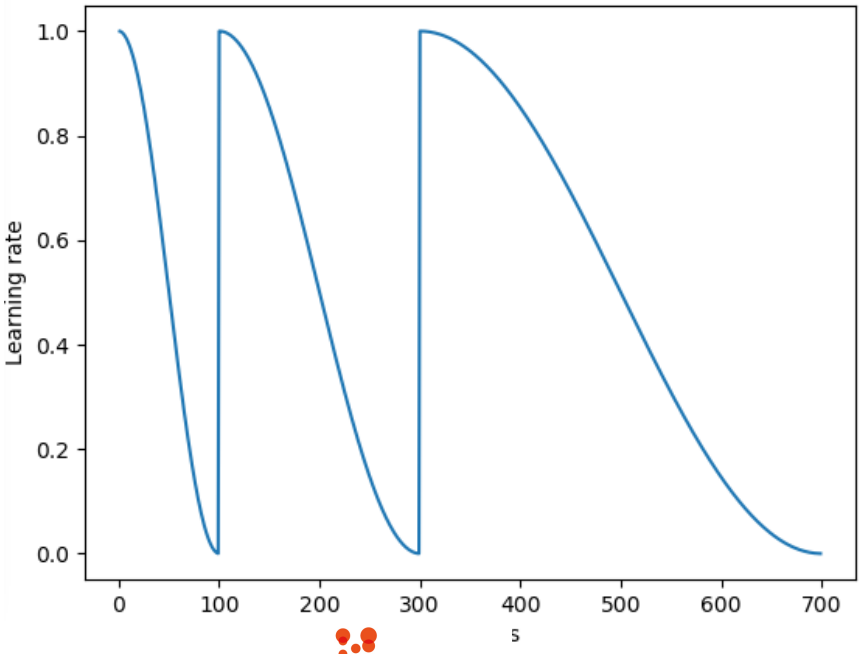
- Learning rate schedulers
- Decrease the learning rate



Learning rate annealing

- Learning rate schedulers
- Decrease the learning rate
- Restart learning rate
- **Potential model combination**





Large batch size

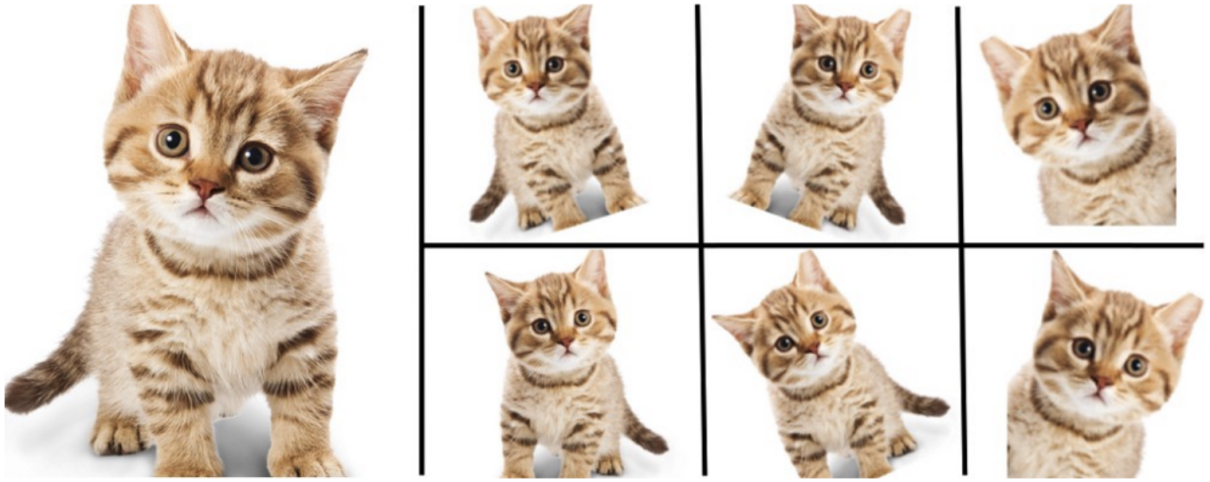
- Benefit from data parallelism. Faster converge
- In some cases better results
- Batch sizes $[1K, 64K]$
- Several GPUs in parallel

  Or

- N forwards, N backwards but 1 gradient update

Data augmentation

- Given $(\mathbf{x}, \mathbf{y})$ input and target of the network
- Provide a new pair $(\tilde{\mathbf{x}}, \mathbf{y})$
- $\tilde{\mathbf{x}}$ is a modified version of $\mathbf{x}$ that have the same $\mathbf{y}$
- Usually in image recognition $\tilde{\mathbf{x}}$ is obtained applying affine transformations to the image $\mathbf{x}$



Mixup data augmentation

- Agnostic data augmentation
- Model the inter-class representation space

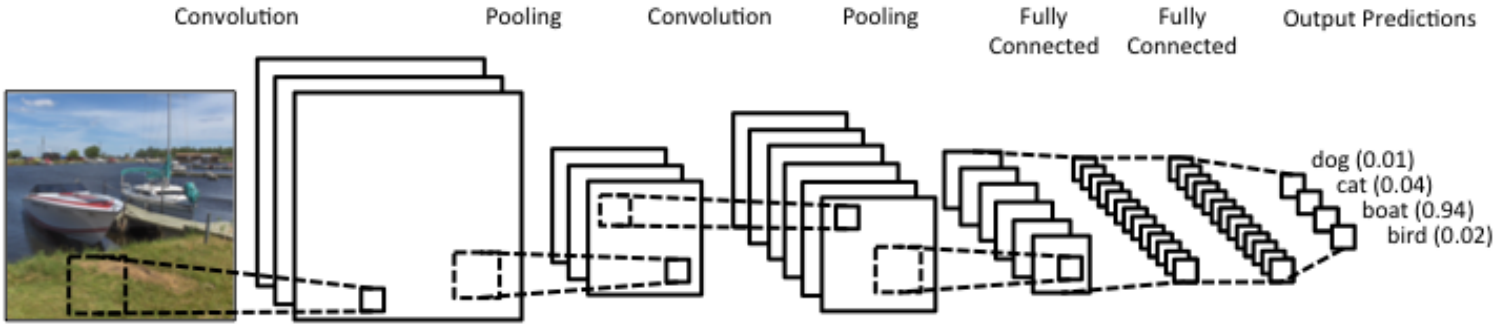
$$\tilde{x} = \lambda x_i + (1-\lambda)x_j, \qquad \text{where } x_i, x_j \text{ are raw input vectors}$$
$$\tilde{y} = \lambda y_i + (1-\lambda)y_j, \qquad \text{where } y_i, y_j \text{ are one-hot label encodings}$$

- Typical values $\lambda = [0.2 - 0.4]$

Index

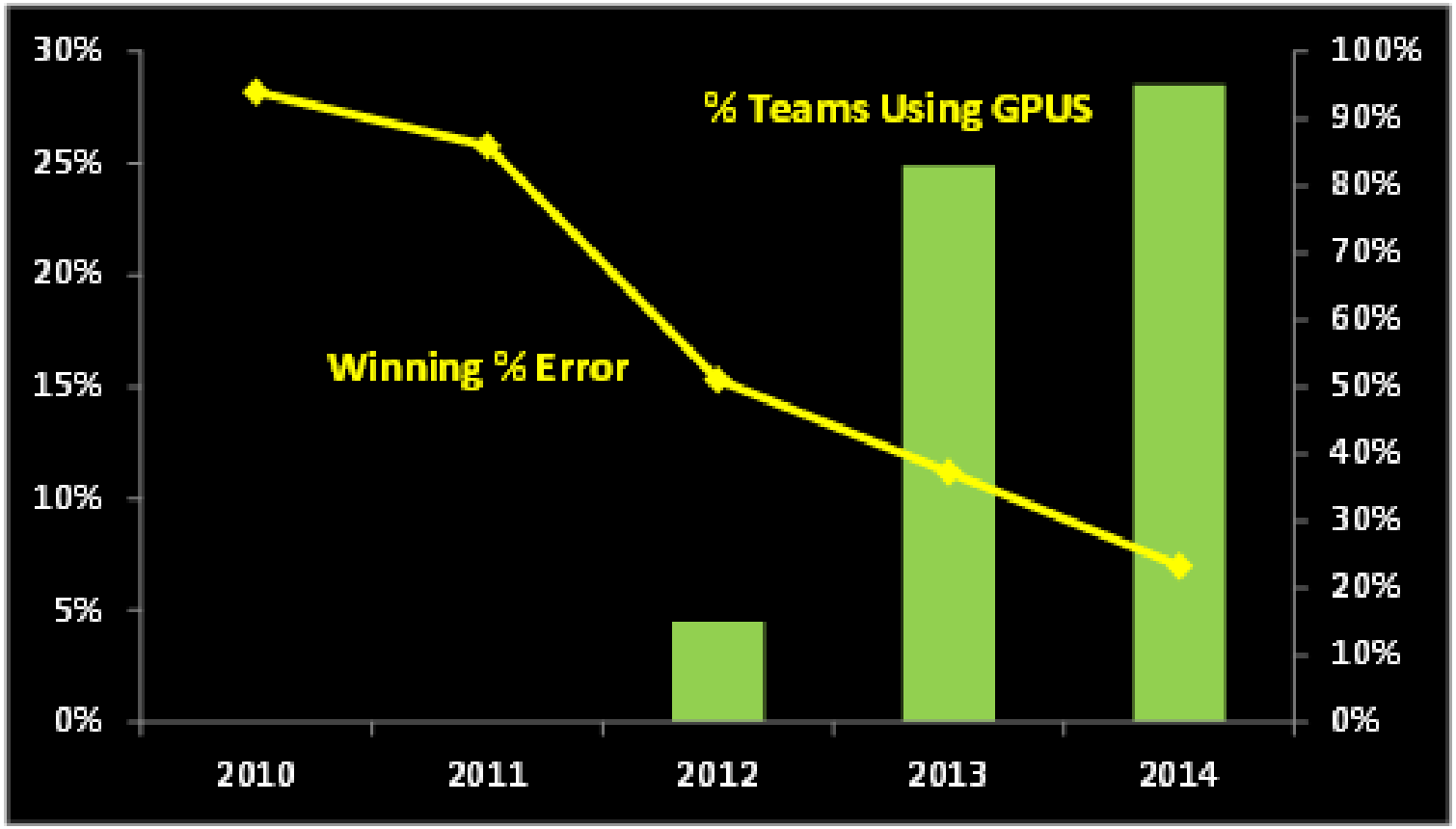
Convolutional Networks

- Deep Learning $\rightarrow$ Bridge the gap between raw representation and categories

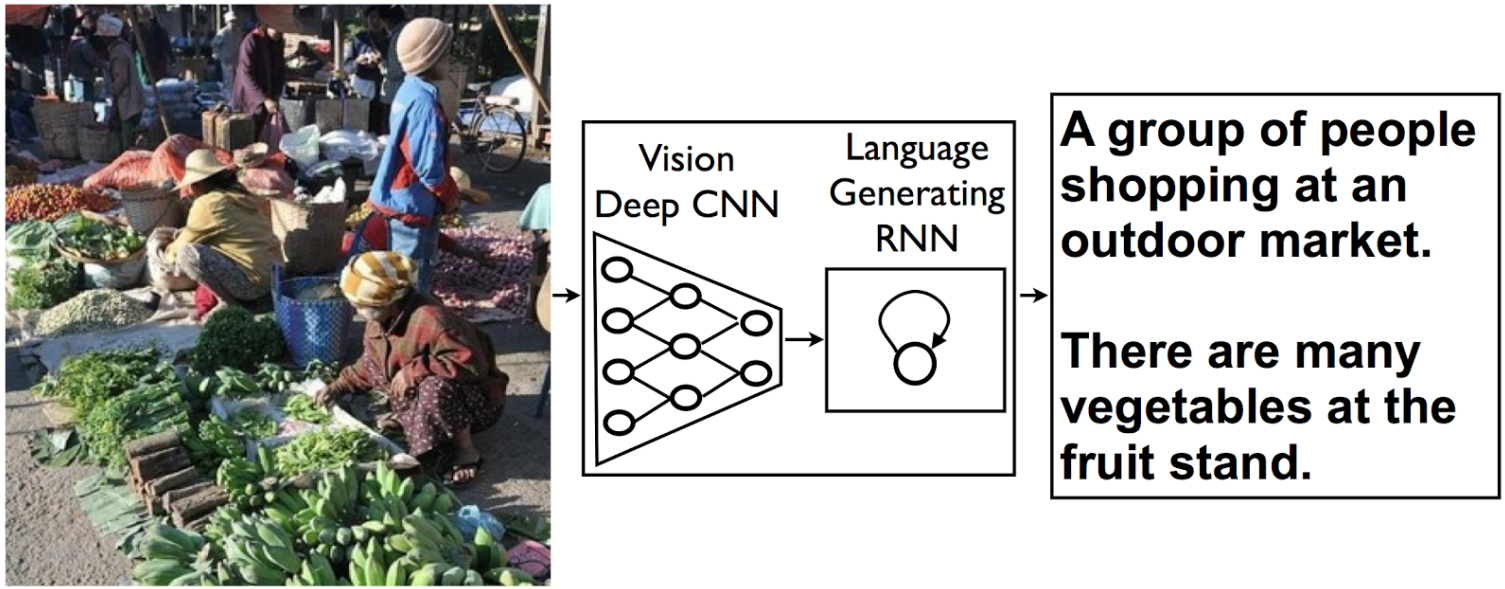


Convolutional Networks

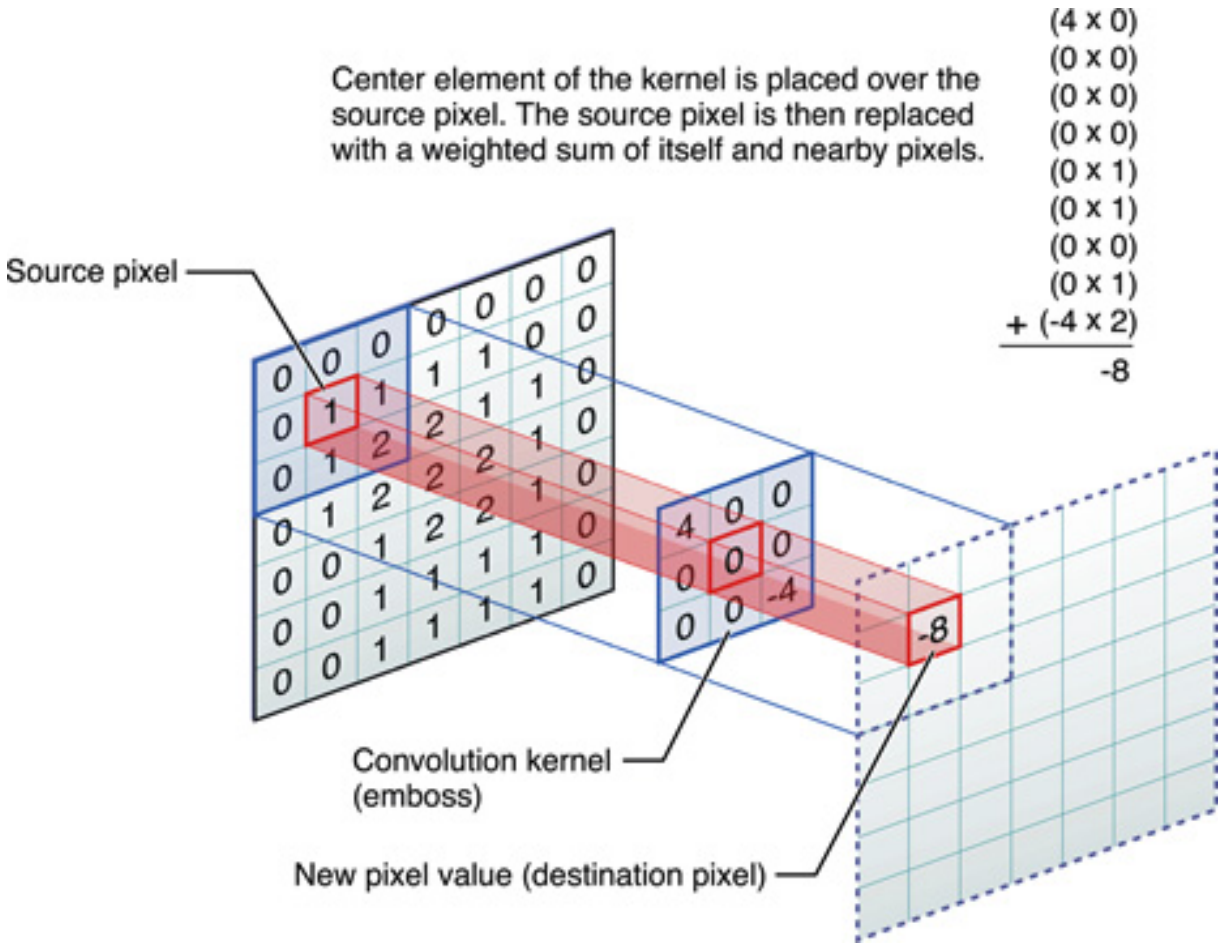
- ImageNet Challenge



Convolutional Networks



Convolution Operator



Convolution Operator

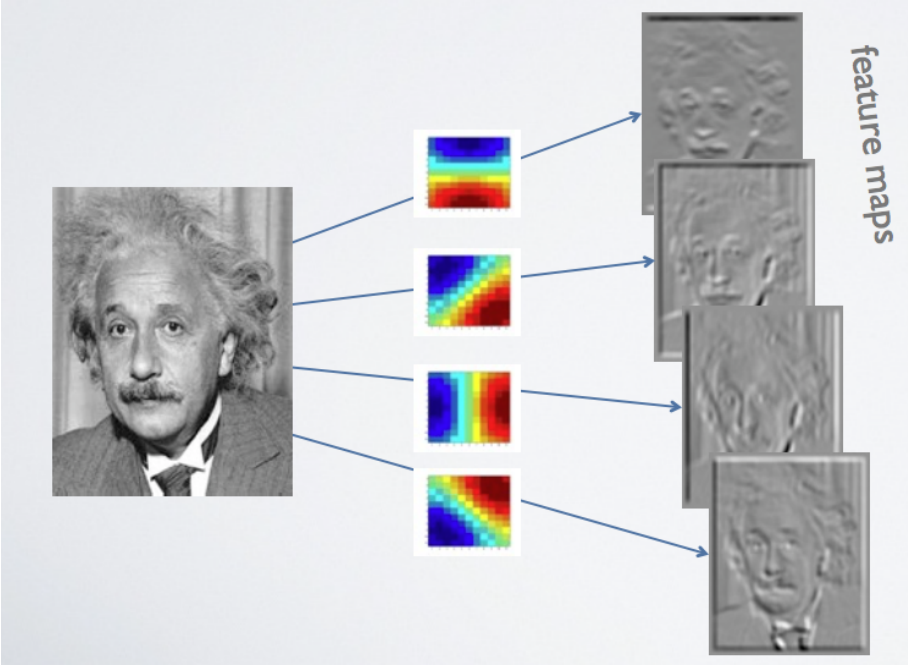
- The simplest case:
  - Size of input image: $I_R \times I_C$
  - Size of kernel: $k_r \times k_c$
  - Size of output image: $O_R \times O_C$
    - $O_R = (I_R - k_r) + 1$
    - $O_C = (I_C - k_c) + 1$
  - Convolution cost: $O_R \times O_C \times k_r \times k_c$

Convolution Operator

- Padding, same output size than input size
  - $O_R = I_R$
  - $O_C = I_C$
  - Add a frame of $k_r/2 \times k_c/2$ of 0's to the input image
- Stride, jump scanning the input image
- In general:
  - $O_R = \lfloor (I_R + 2 * pad - k_r)/stride_r + 1 \rfloor$
  - $O_C = \lfloor (I_C + 2 * pad - k_c)/stride_c + 1 \rfloor$

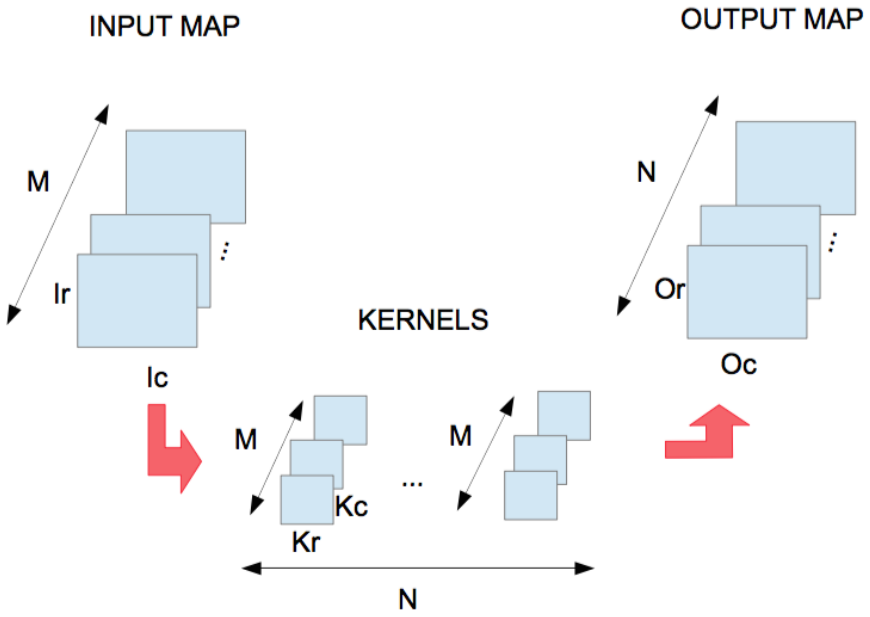
Convolution Operator

- The general case:
  - Input: Images (2D)
  - Apply more than one kernel (3D)
  - Output: Maps (3D)



Convolution Operator

- The general case:
  - Input: Maps (3D)
  - Apply more than one kernel (4D)
  - Output: Maps (3D)



Convolution Operator

- Input Map: $M \times I_R \times I_C$
- Kernels: $N \times (K_r \times K_c \times K_M)$
- Output Map: $N \times O_R \times O_C$

- Kernels are 3D, so it is a 3D convolution
- Cost: $N \times O_R \times O_C \times K_r \times K_c \times K_M$
- Parameters: $N \times K_r \times K_c \times K_M + N$ (bias)

Convolution Operator

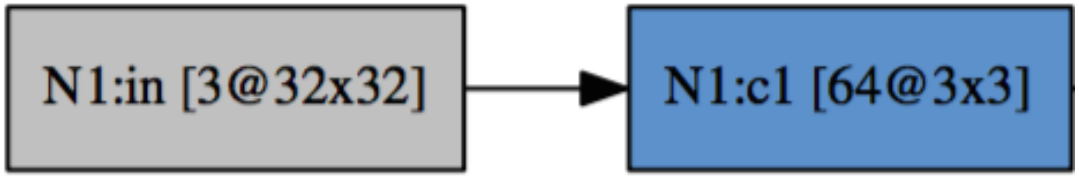
- Implementation tricks: **LOWERING**
  - A convolution becomes a standard multiplication $I \times K$
- Implementation tricks: **Multi-threading**
  - For instance, split the batch into different threads
- Implementation tricks: **FFT**
  - A convolution is a multiplication in the frequency domain
- Implementation tricks: **Winograd** algorithm:
  - A fast method to obtain the convolution with less multiplication and addition operations

Convolution Operator

- Lowering steps:
  - Reshape input maps into I
  - Reshape kernels into K
  - Perform multiplication $O = I \times K$
  - Reshape O into output maps

Convolution Operator as a layer

- Input map $M = 3$, $O_R = 32$ and $O_C = 32$
- Kernels $N = 64$, $K_r = 3$ and $k_c = 3$



- Output map sizes?

Convolution Operator - Exercises

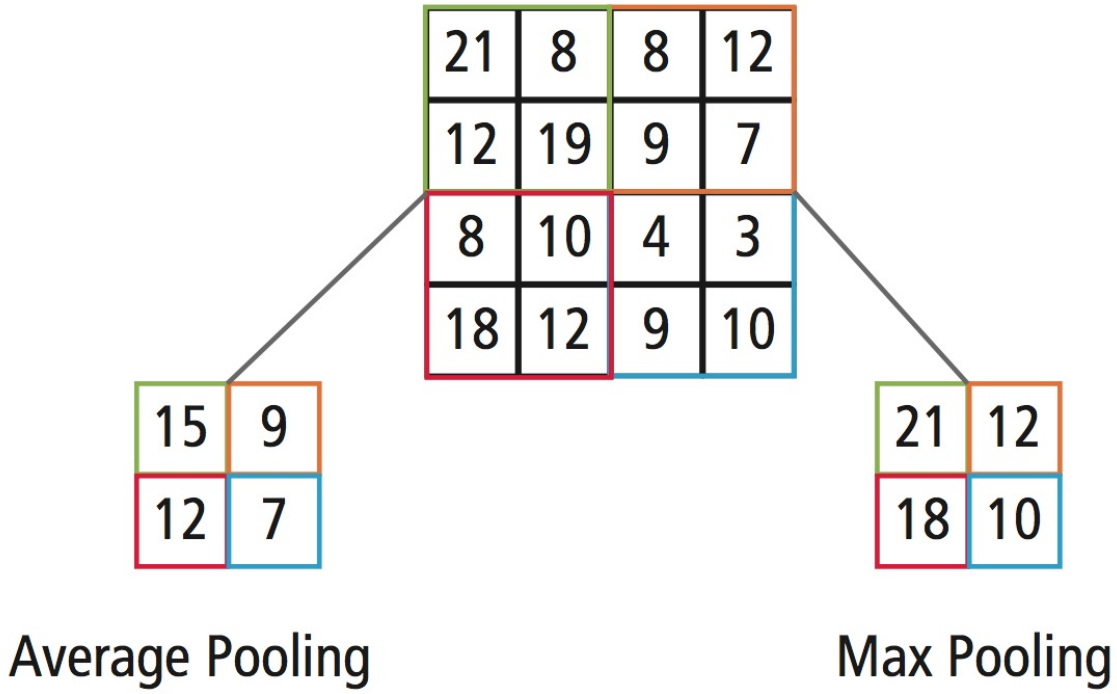
- Given an input map $32@64\times64$ compute the size of the output map and the computational cost of the convolution with the following kernels:
  - $64@5\times5$
  - $64@3\times3$
  - $64@3\times3$ and again $64@3\times3$

Pooling Operator

- The main goal is to reduce the size of the maps:
  - Reduce the computational cost
  - Deal with multiscale
  - Capture higher level features

Pooling Operator

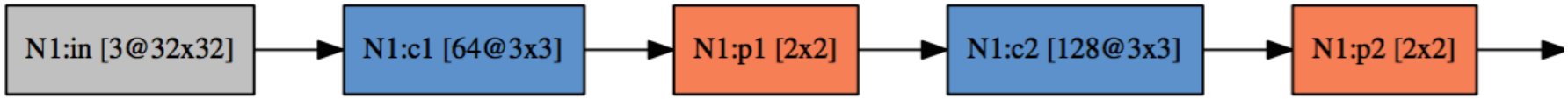
- Maxpool and Average Pool



- Normally $stride = size$

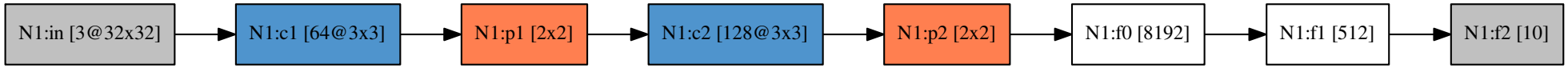
Pooling Operator - Exercises

- Compute the size of the resulting maps after applying the following convolutional and pooling layers:



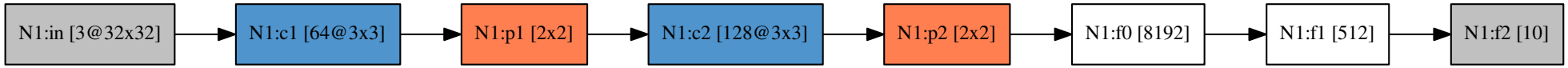
Reshape Layers

- The goal of the reshape layer is to present the maps to the next layers Fully Connected layers
- The maps become a raw vector and the spatial relationship is not considered
- After the reshape several hidden layers can be stacked to reach the output layer, conforming a **Convolutional Network**:



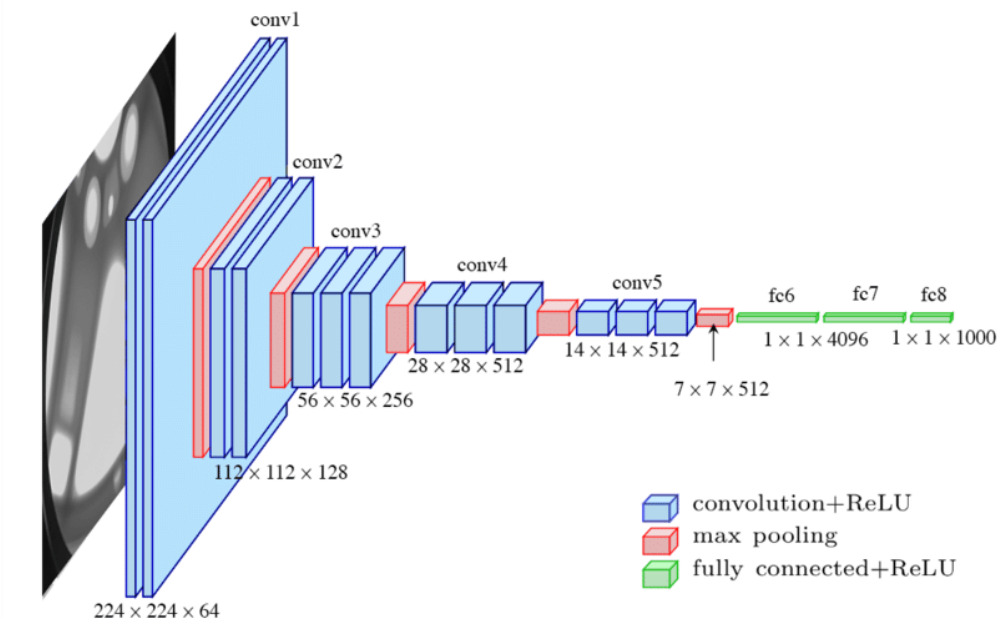
Reshape Layers - Exercise

- Given the following CNN, How many parameters (weights) are?
- Where is the big amount of parameters?

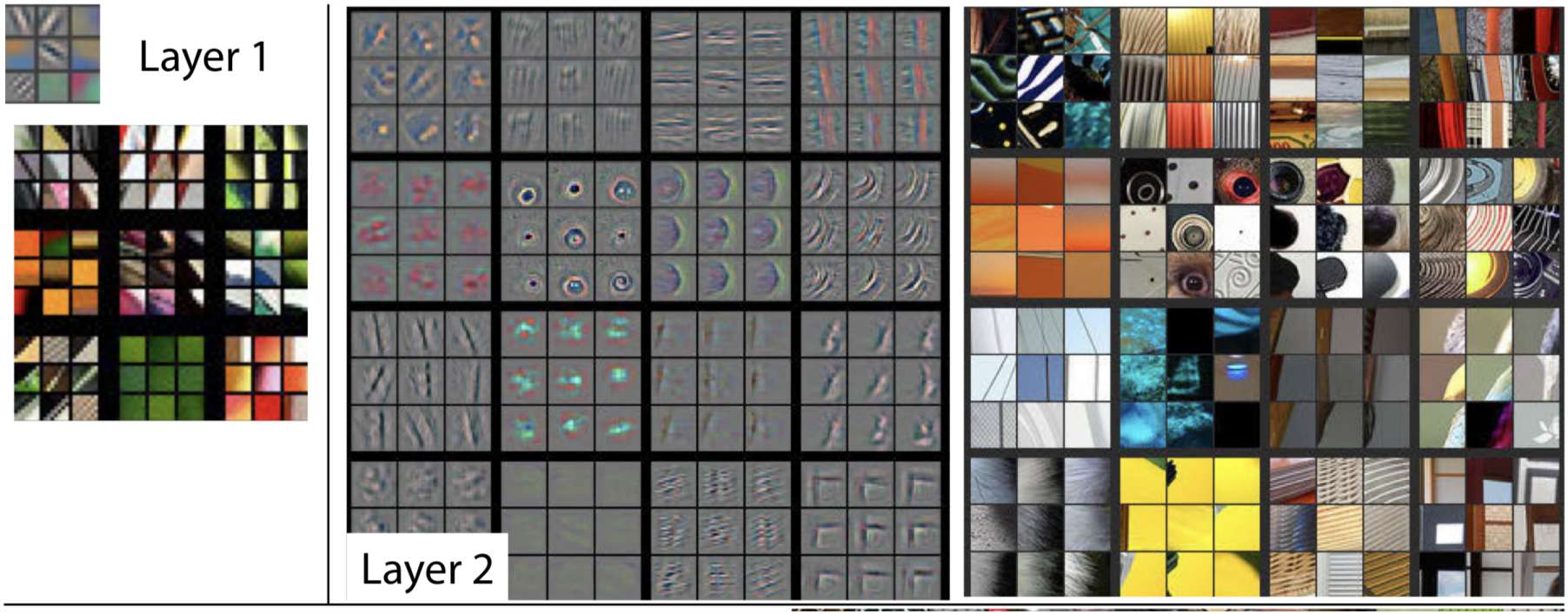


Convolutional Neural Network

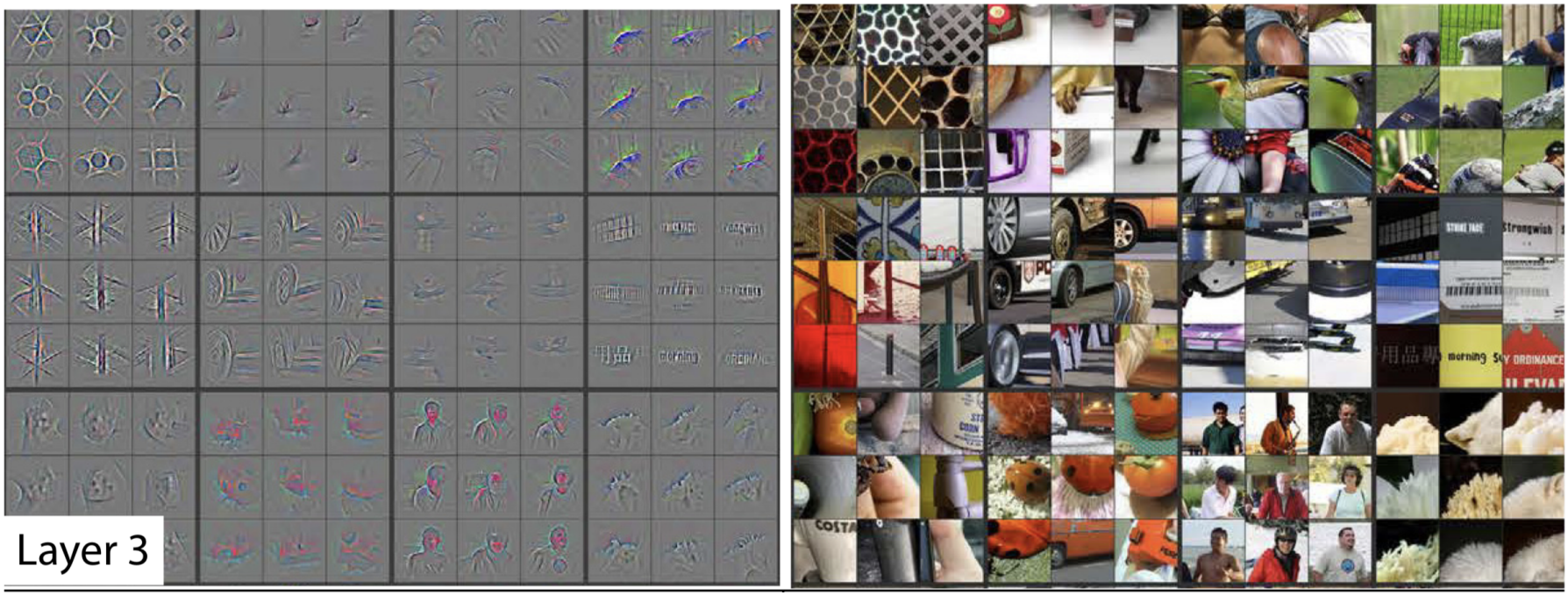
- Tipically the size of the map is reduced but the depth is increased
- The depth size represent the ability to detect different structures (objects)
- The reduction of the map size allow to cover larger spatial dependencies using the same receptive field, i.e. 3×3 filters
- The maxpool allows to keep detections (max) in the spatial reduction operations



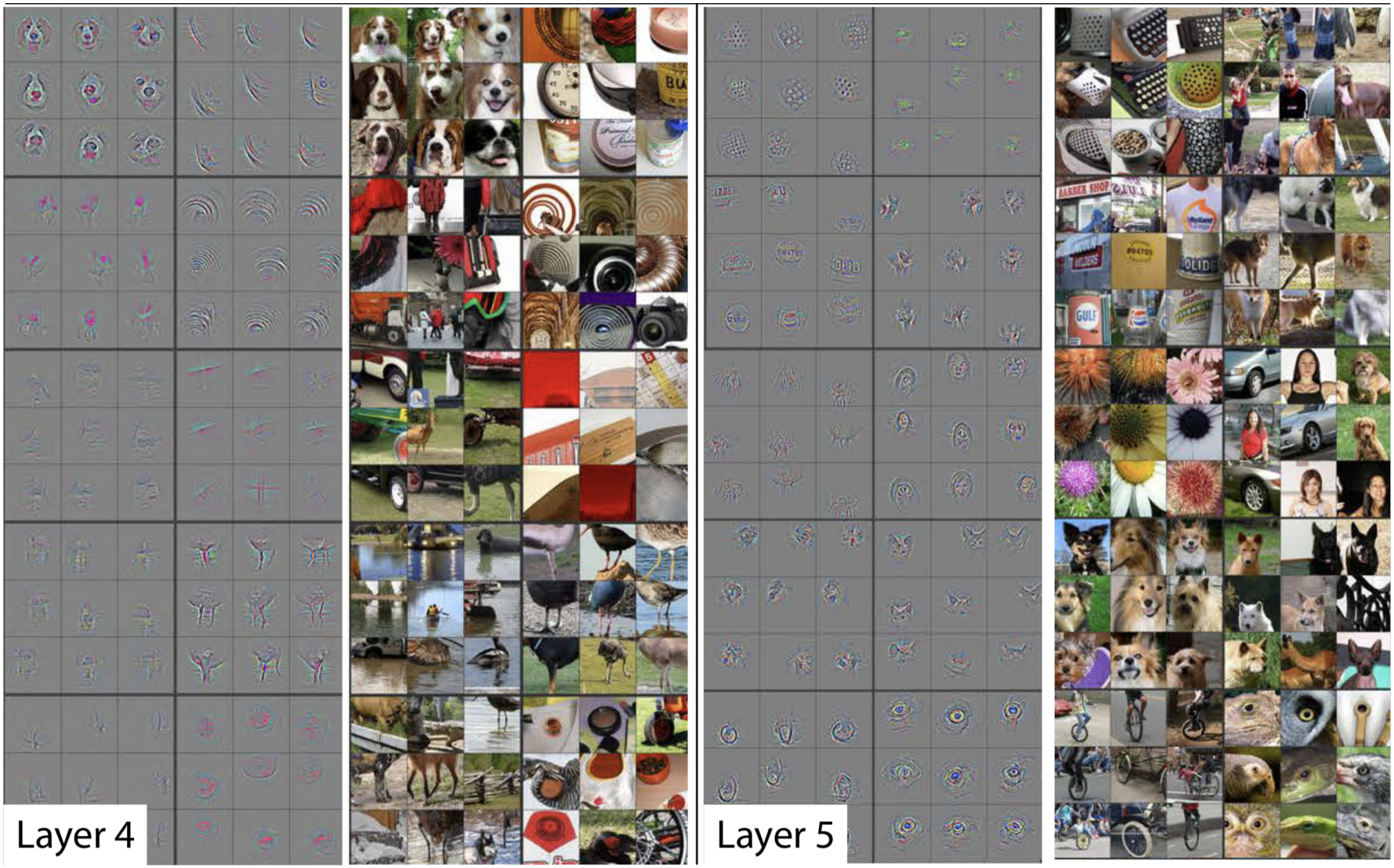
CNN Features



CNN Features

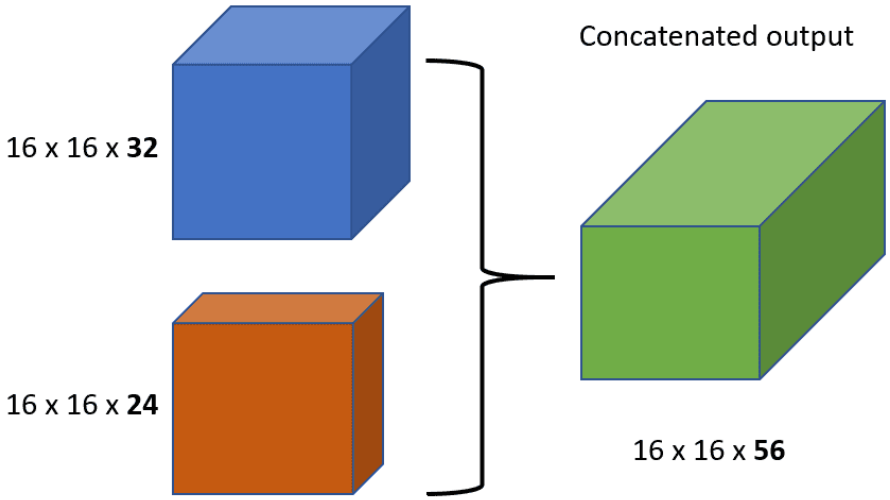


CNN Features



Special Layers - Cat Layers

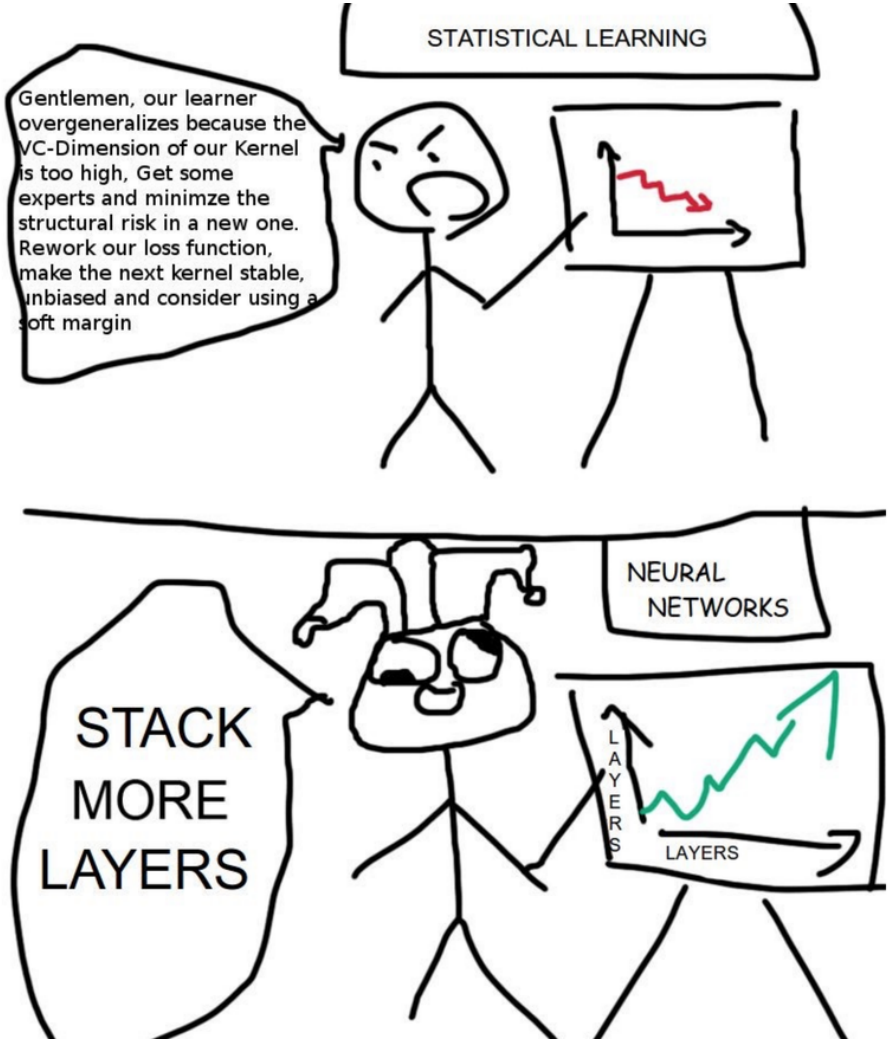
- A layer that cat the maps of the layers that are connected to
- The sizes of the maps must be equivalent
- Use to be necessary the use of padding in previous convolutional layers
- Used in the Inception Model (GoogleNet)



Special Layers - Aggregation Layers

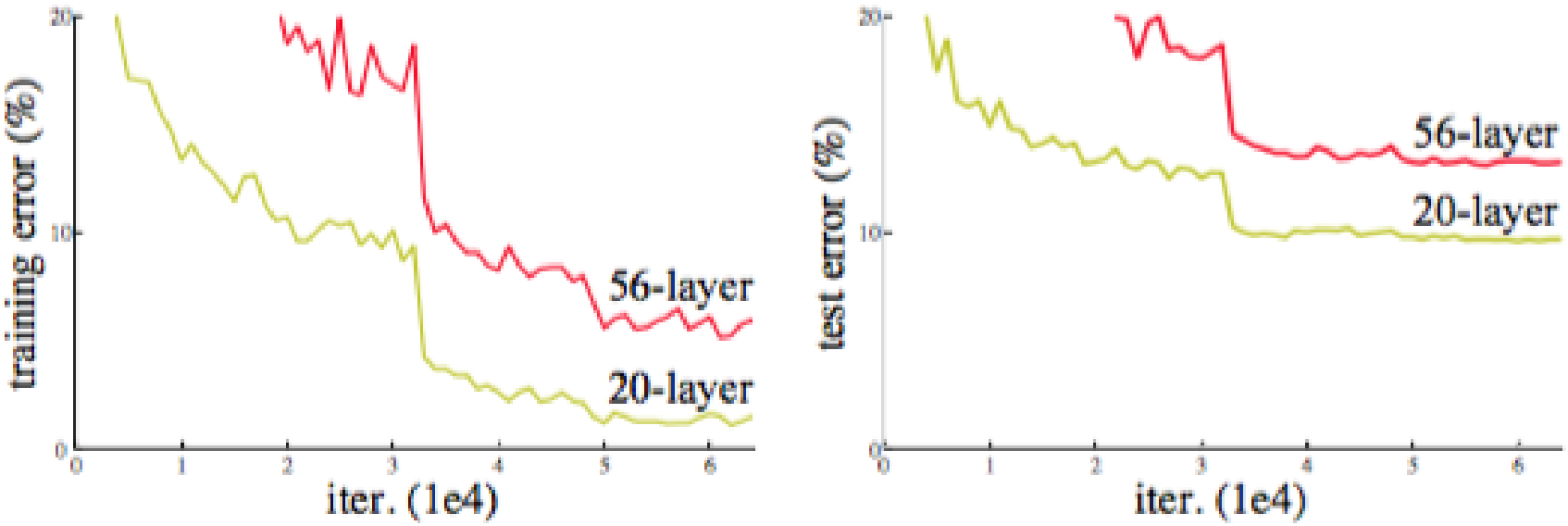
- Similar to cat layers
- A layer that sum the maps of the layers that are connected to
- The size and number of maps must be the same
- Used on the Residual Nets (Microsoft Research)

Going deeper



Problems going deeper

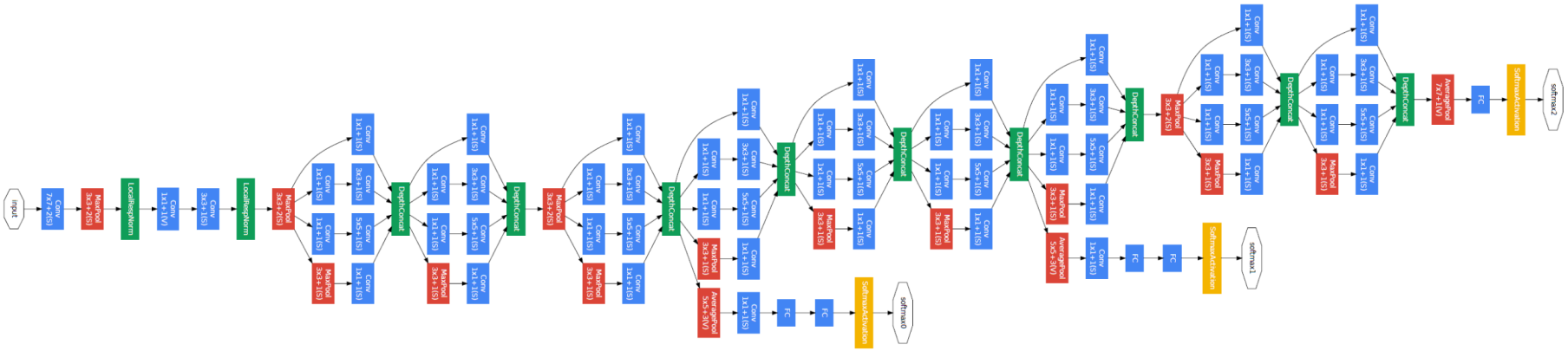
- Train Deep Networks is hard:



Solutions

- Increasing network depth does not work by simply stacking layers together
- Still vanishing gradient problem
- Potential solutions: Optimizers, Initializers (Layer-sequential unit-variance, LSUV), Activation Functions (PReLU, ELU, SELU)
- Potential solutions: GoogleNet with auxiliary loss in a middle layer as extra supervision

Solutions. GoogleNet. Inception Model

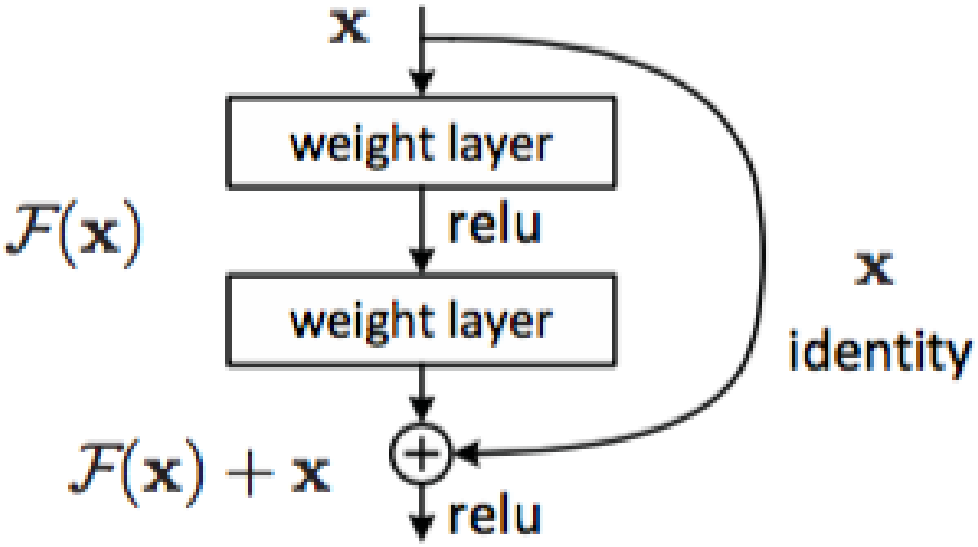


Solutions

- Increasing network depth does not work by simply stacking layers together
- Still vanishing gradient problem
- Potential solutions: Optimizers, Initializers (Layer-sequential unit-variance, LSUV), Activation Functions (PReLU, ELU, SELU)
- Potential solutions: GoogleNet with auxiliary loss in a middle layer as extra supervision
- Better Solutions: identity shortcut connections, **Residual Nets**

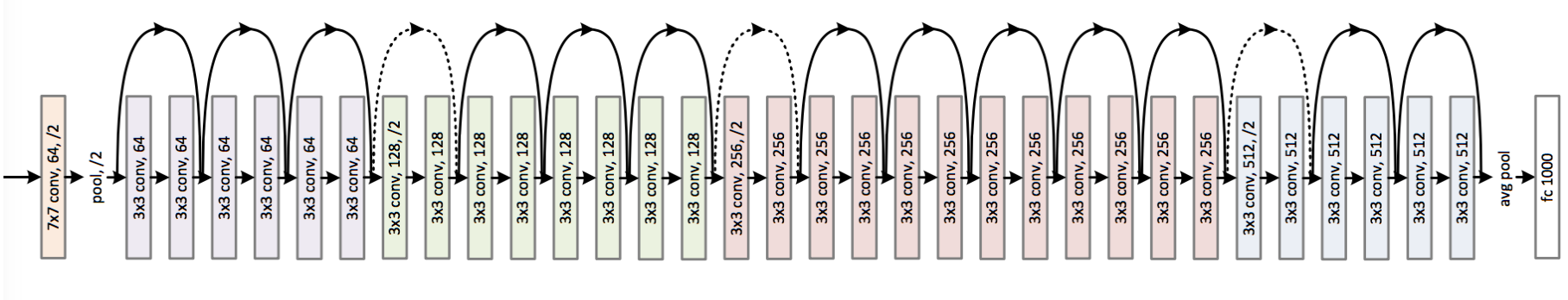
Residual Nets

- Key idea: fit residual mappings instead of mappings



Residual Nets

- ResNet 34-layers



- Dotted lines, two options:
  - Identity mapping, stride 2, zero padding
  - Identity mapping, stride 2, 1x1 convolution

- Convolution + BN + Activation

Residual Nets

- ResNet with different depth

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

Residual Nets

- Results on ImageNet

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**

Follow-up?

Attend to my **Computer Vision** lectures