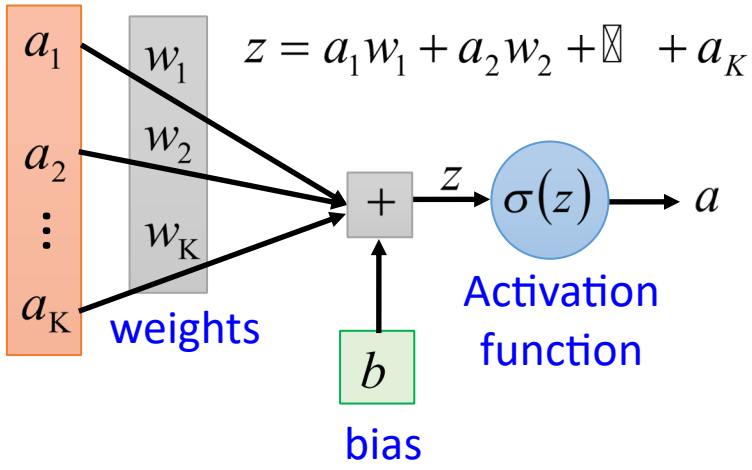
Basics of Deep Learning

By: Dr. Puneet Gupta,
Associate Professor,
Department of Computer Science and Engineering
IIT Indore

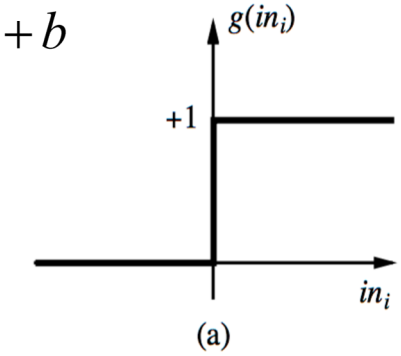
Neural Network: A Neuron

A neuron is a computational unit in the neural network that exchanges messages with each other.

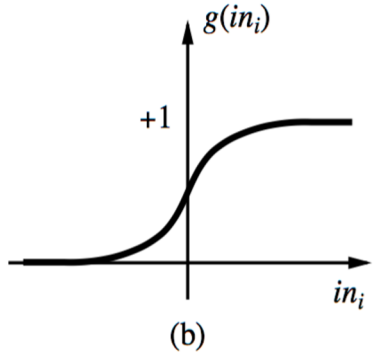
Neuron $f: R^K \rightarrow R$



Possible activation functions:

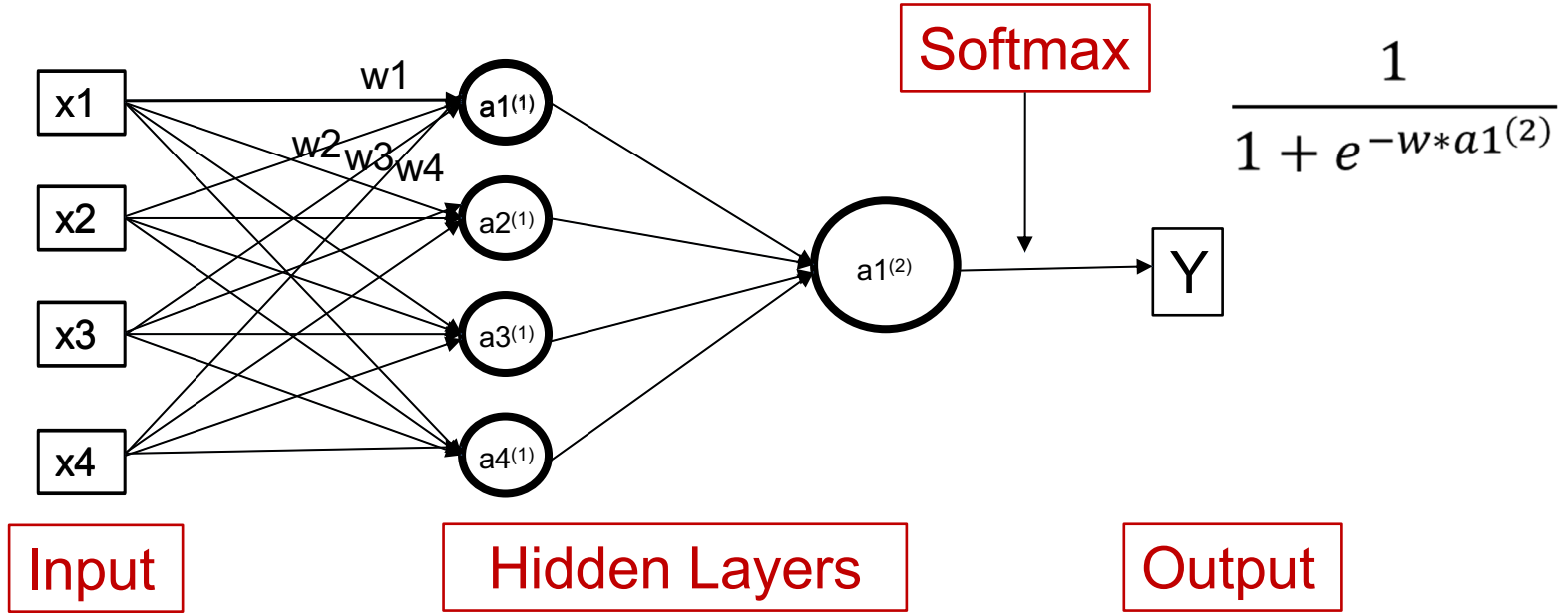


Step function/threshold function



Sigmoid function (a.k.a, logistic function)

A simple neural network



$$a1^{(1)} = f(w1 * x1 + w2 * x2 + w3 * x3 + w4 * x4)$$

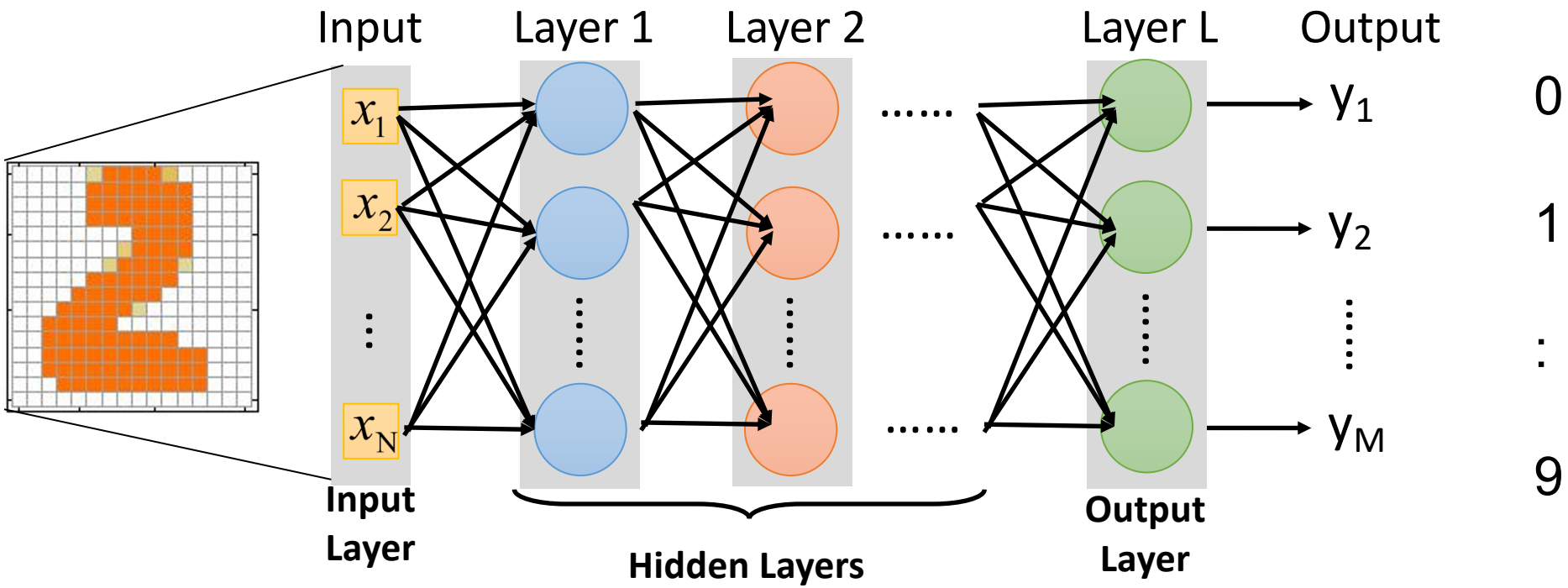
Number of parameters:
4*4 + 4 +1

$f()$ is activation function: Relu or sigmoid

$$Relu: \max(0, x)$$

$$a1^{(1)} = max(0, w1 * x1 + w2 * x2 + w3 * x3 + w4 * x4)$$

A case study



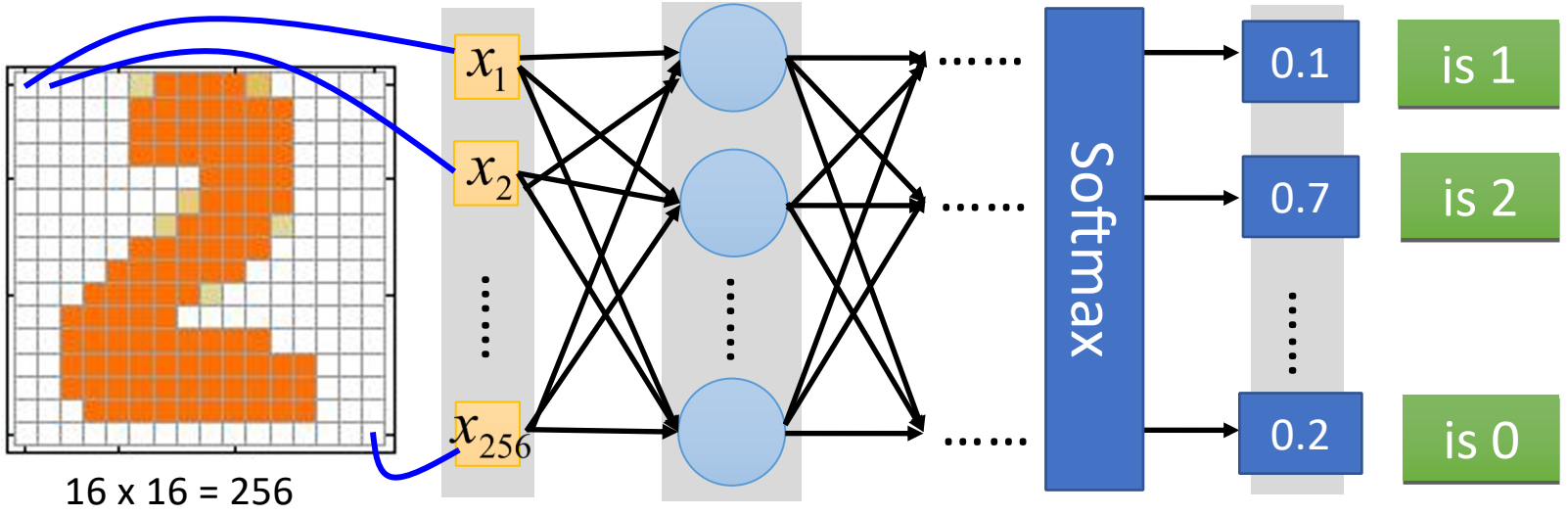
: denotes neuron

$$f: R^{256} \rightarrow R^{10}$$

Deep learning (DL) means many hidden layers.
DL represents the function f by neural network

Learning network parameters

$$\theta = \{W^1, b^1, W^2, b^2, \cdots W^L, b^L\}$$



Orange→ 1
Black → 0

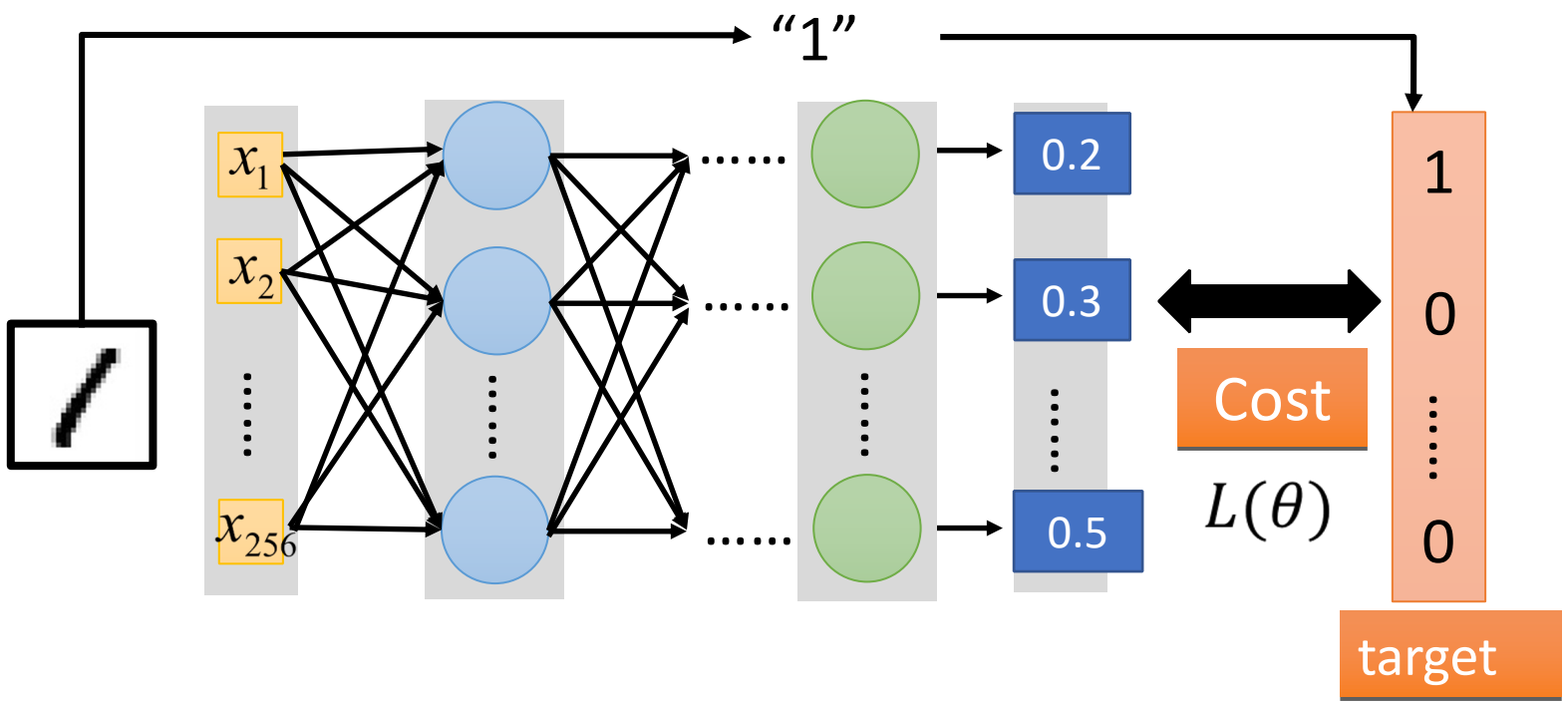
Aim: Find network parameters such that ……

Input :  → y_1 has the maximum value

Input :  → y_2 has the maximum value

How can the neural network achieve this ?

Computing cost of a sample



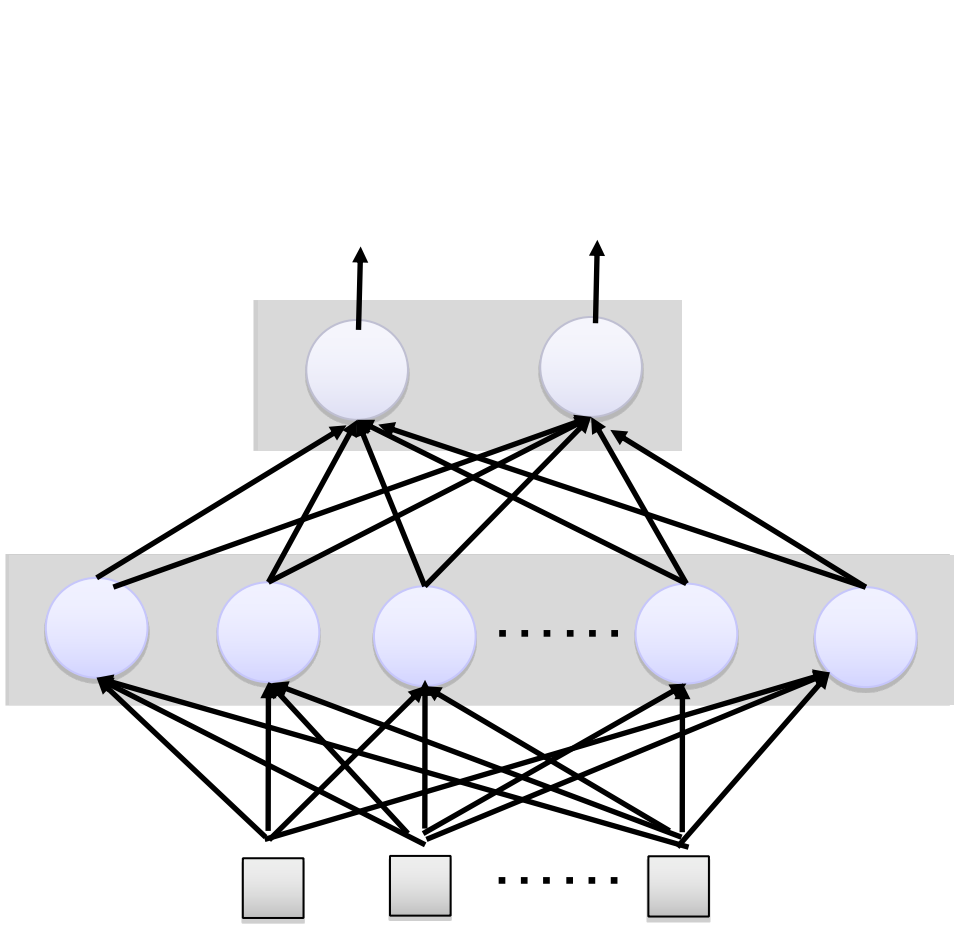
Cost can be Euclidean distance or cross entropy between network output and target
Given a set of network parameters, each example has a cost value.

How to evaluate that how bad the network parameters performs on this task?

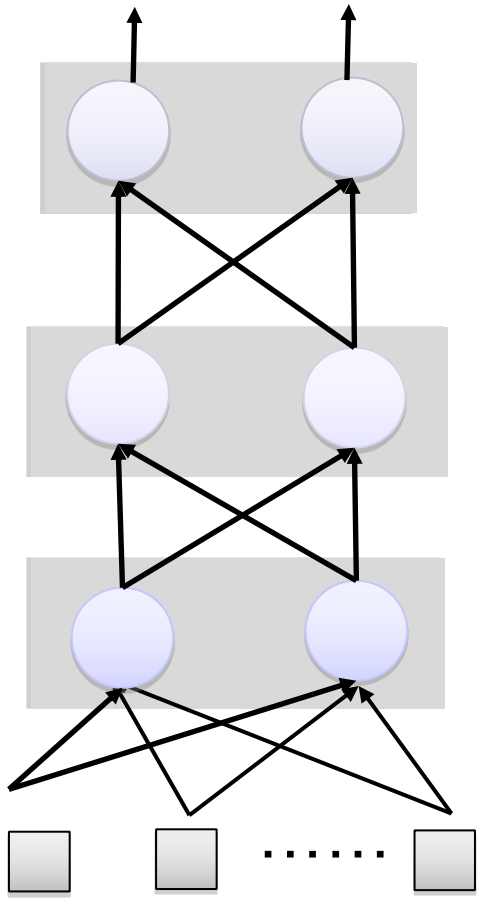
Total Cost: $C(\theta) = \sum_{r=1}^{R} L^r(\theta)$

Find the network parameters that minimize total cost

Fat + Short Vs Thin + Tall



Shallow



Deep

Which one is better, even when both use same number of parameters?

Fat + Short Vs Thin + Tall

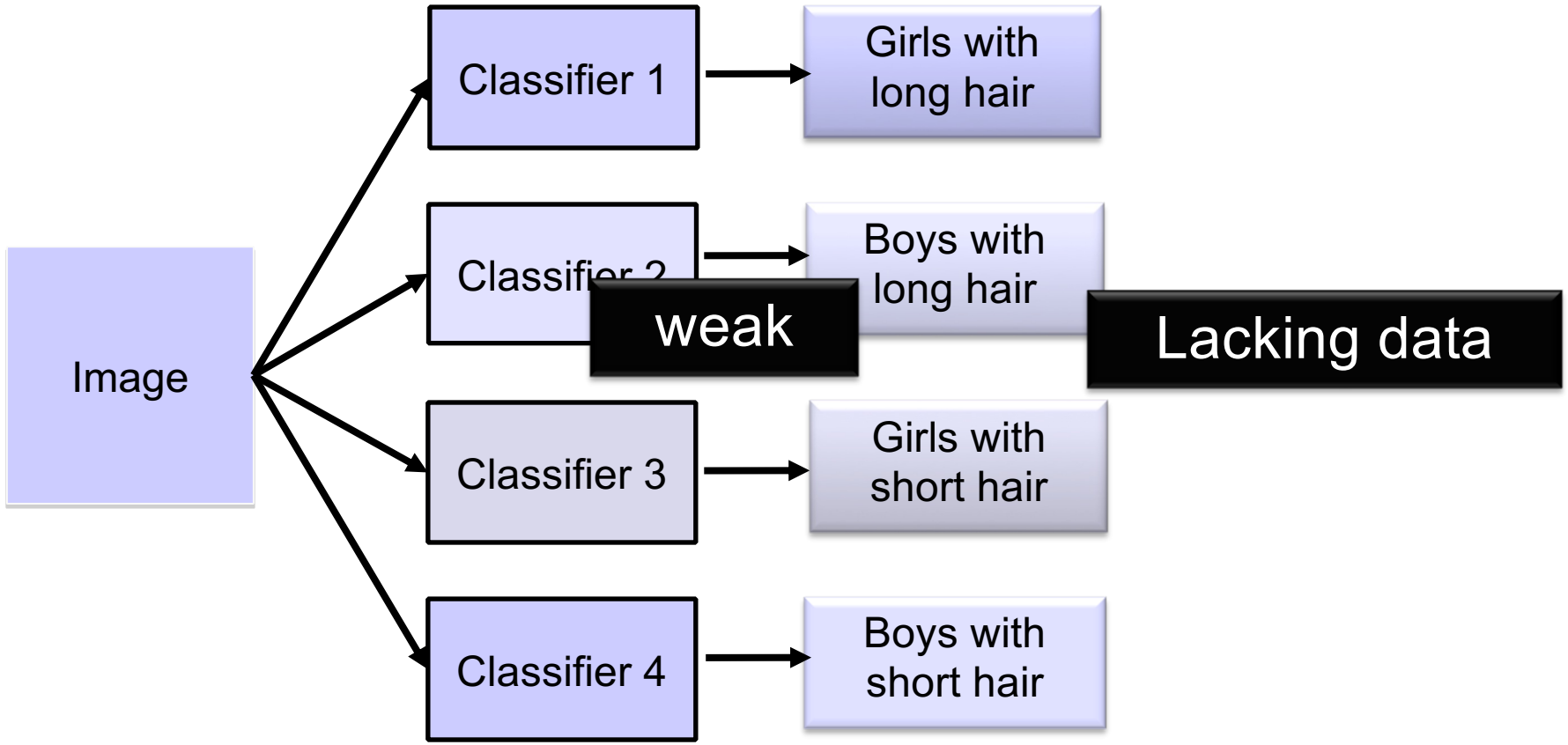
Layer X Size	Word Error Rate (%)	Layer X Size	Word Error Rate (%)
1 X 2k	24.2		
2 X 2k	20.4		
3 X 2k	18.4		
4 X 2k	17.8		
5 X 2k	17.2	1 X 3772	22.5
7 X 2k	17.1	1 X 4634	22.6
		1 X 16k	22.1

Why?

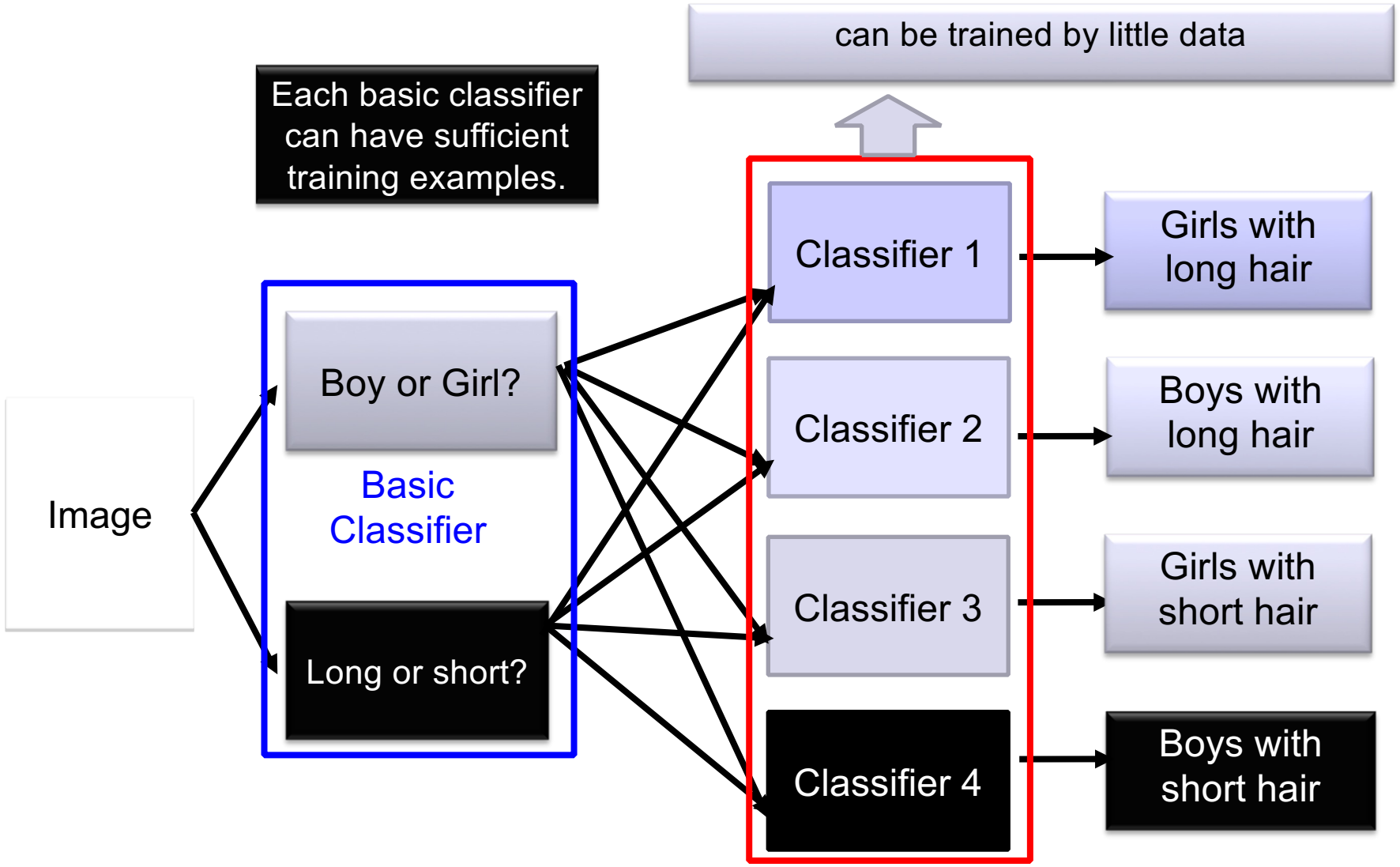
Seide, Frank, Gang Li, and Dong Yu. "Conversational Speech Transcription Using Context-Dependent Deep Neural Networks." *Interspeech*. 2011.

Shallow network

Intuitive example:



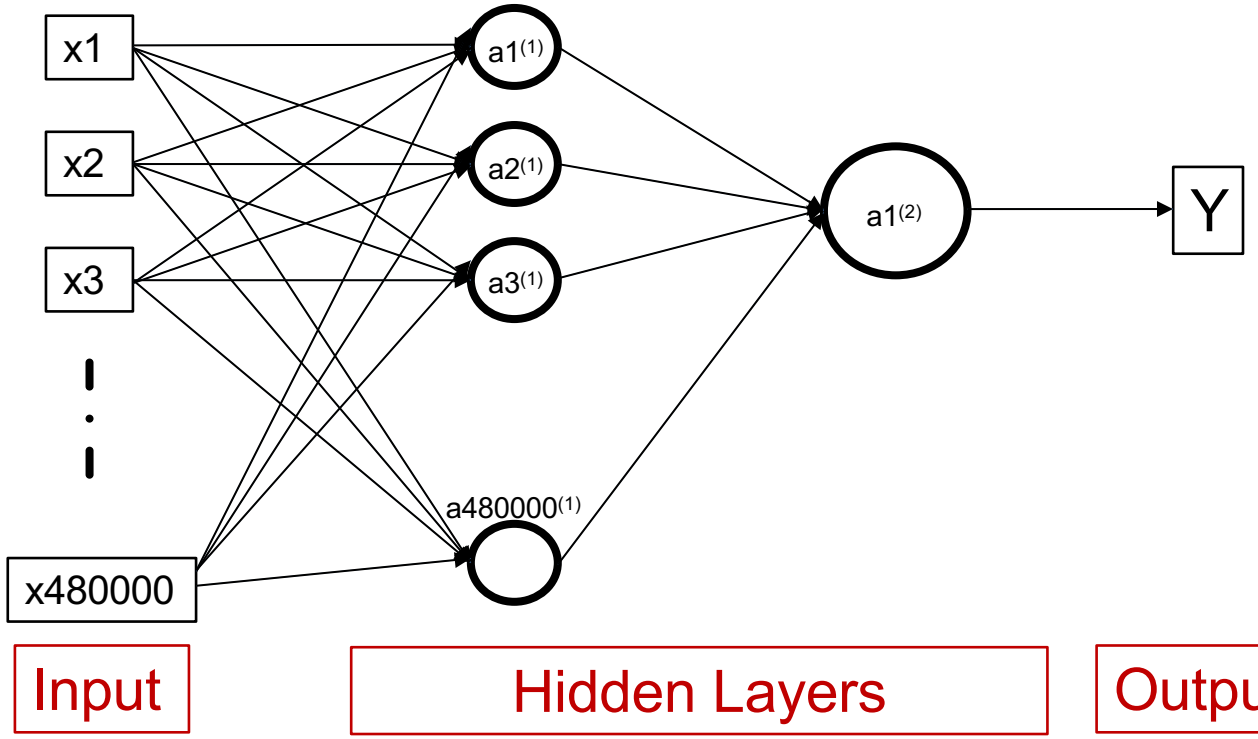
Deep Network



Intuitive example:

Unknown Parameters in a simple Neural Network

400 X 400 X 3



How many parameters are required even after avoiding bias parameters?

Number of Parameters

480000*480000 + 480000 +1 = **approximately 230 Billion !!!**

480000*1000 + 1000 +1 = approximately **480 million !!!** (If we reduce nodes in first hidden layer to 1000)

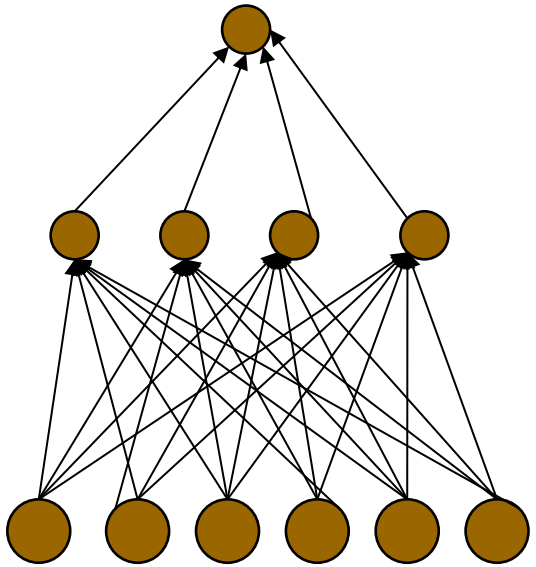
Limitations of Neural Networks

Random initialization and fully connected networks lead to:

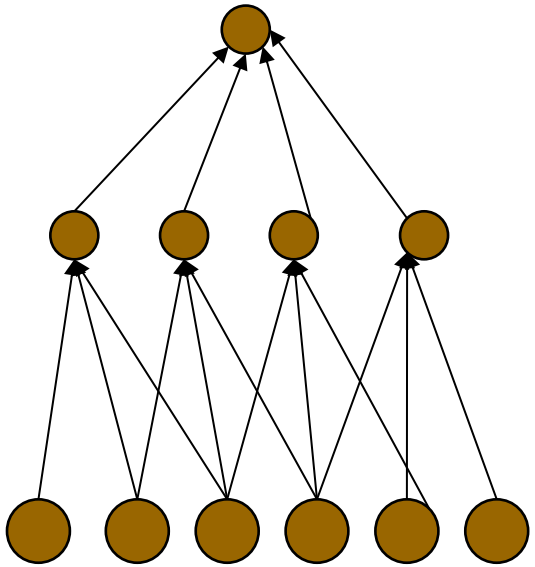
- High cost: Due to large number of neurons
- Difficult to train as the number of hidden layers increases: vanishing gradient problem
  - In backpropagation, gradient is progressively getting more dilute. That is, below top layers, the correction signal is minimal.
- Stuck in local optima
  - The objective function of the neural network is usually not convex.
  - The random initialization does not guarantee global optima.

- Bengioet al., Identifying and attacking the saddle point problem in high-dimensional non-convex optimization, 2014
- LeCunet. al., The Loss Surfaces of Multilayer Networks, 2015
- Goodfellowet al., Qualitatively characterizing neural network optimization problems, 2015

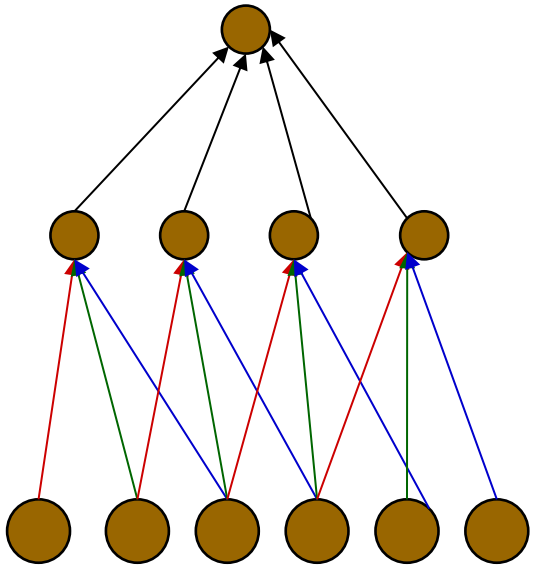
What is Convolutional Neural Network?



6*4+4

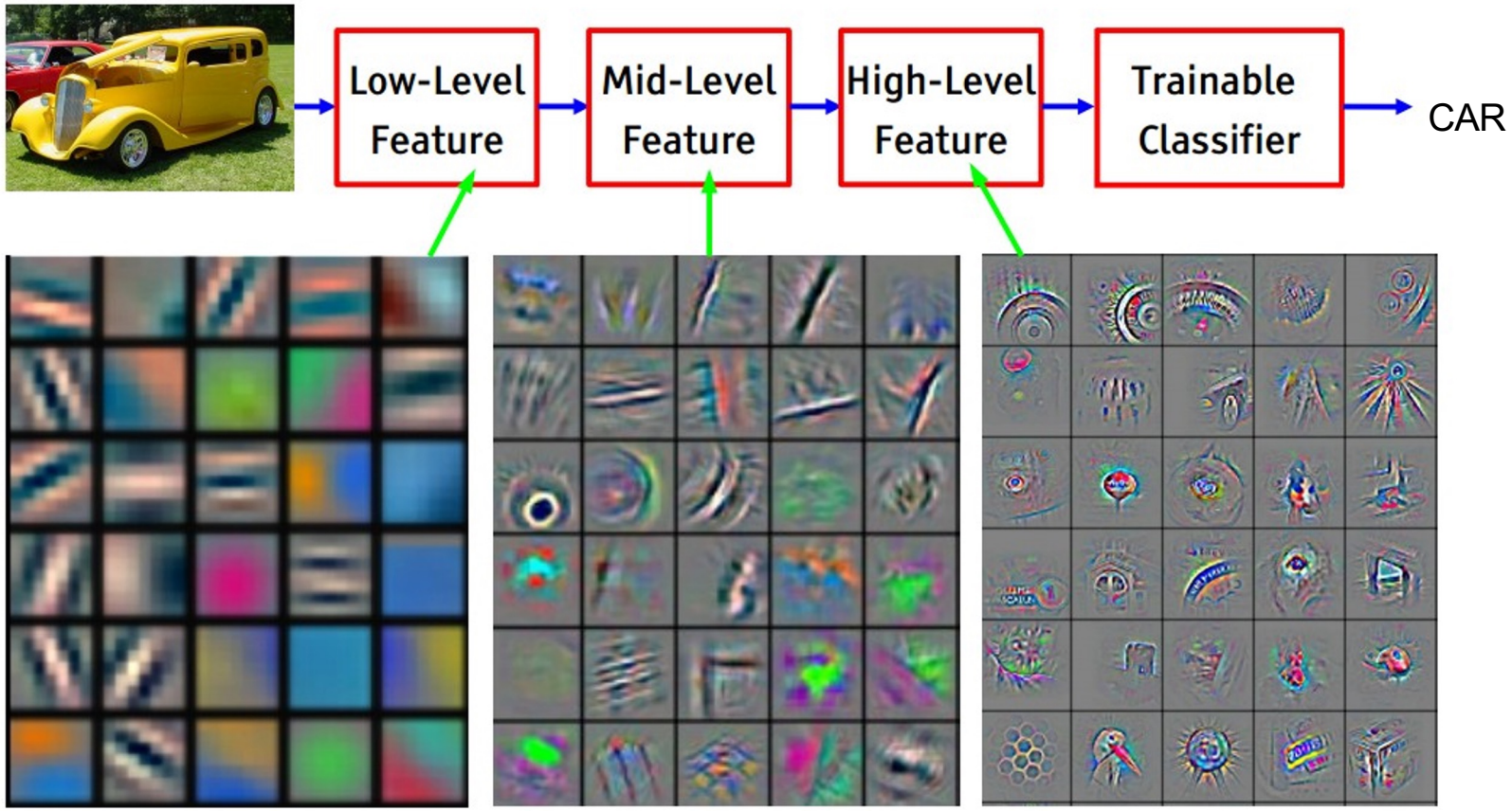


3*4+4



3+4

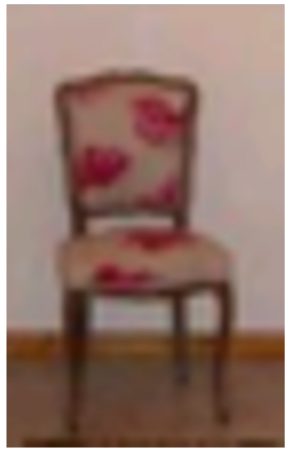
A CNN Classification Network



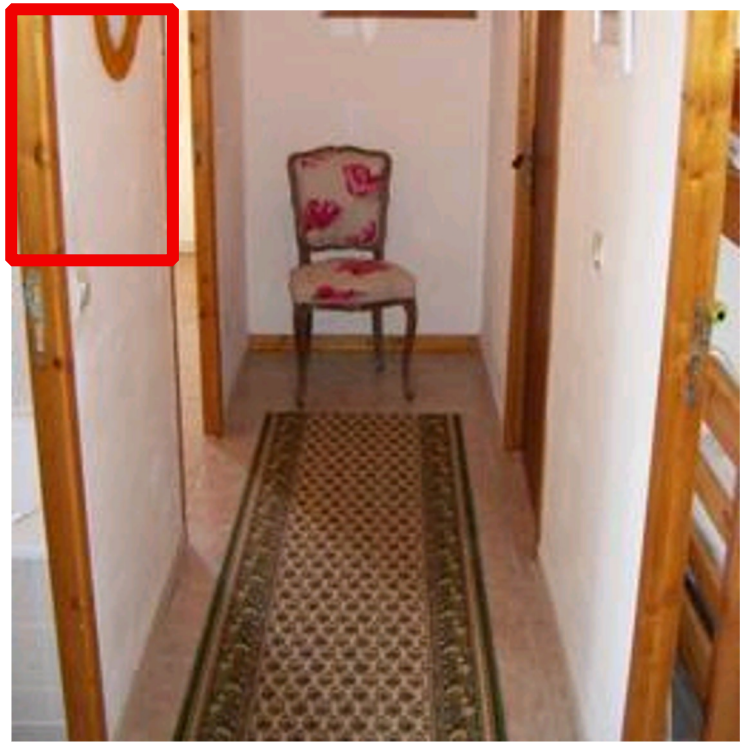
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Global Matching

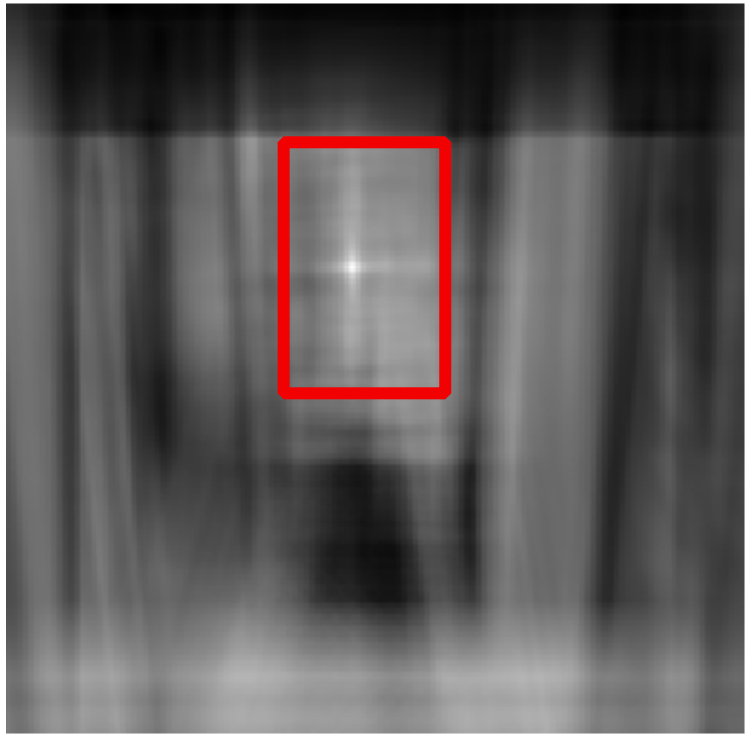
This is a chair

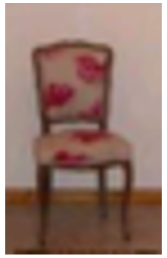


Find the chair in this image



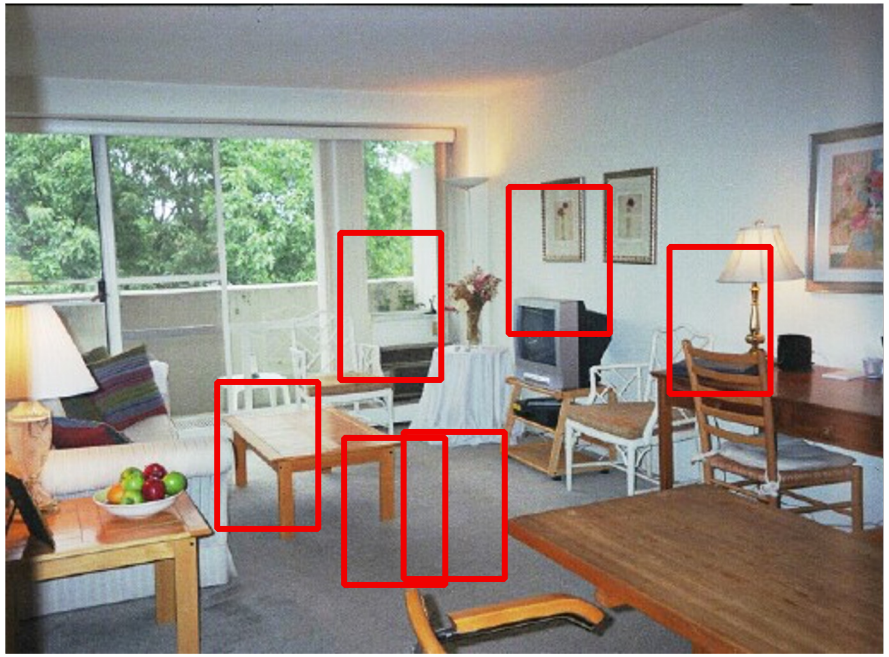
Output of normalized correlation

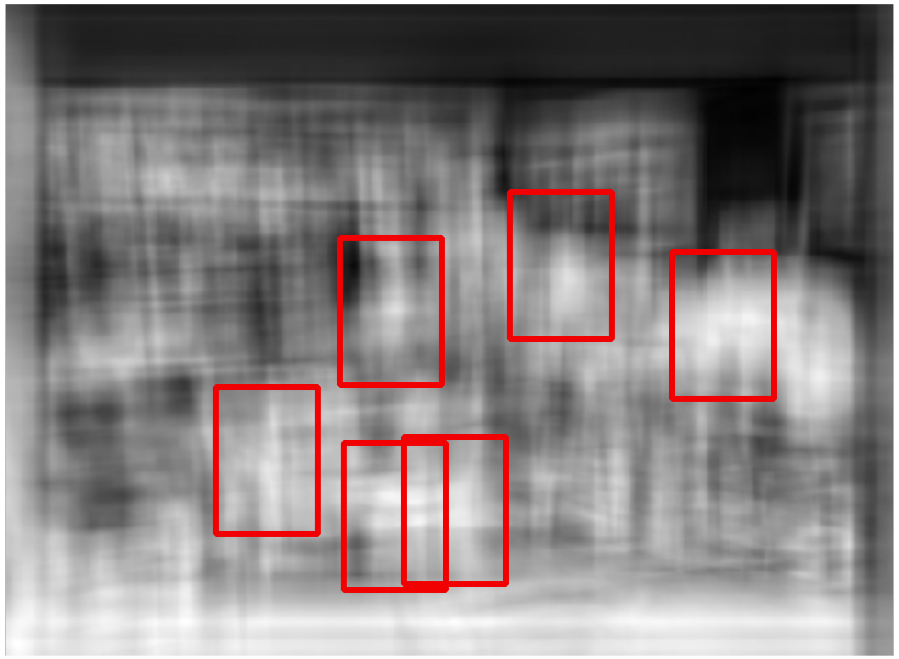




Object recognition
Is it really so hard?

Find the chair in this image

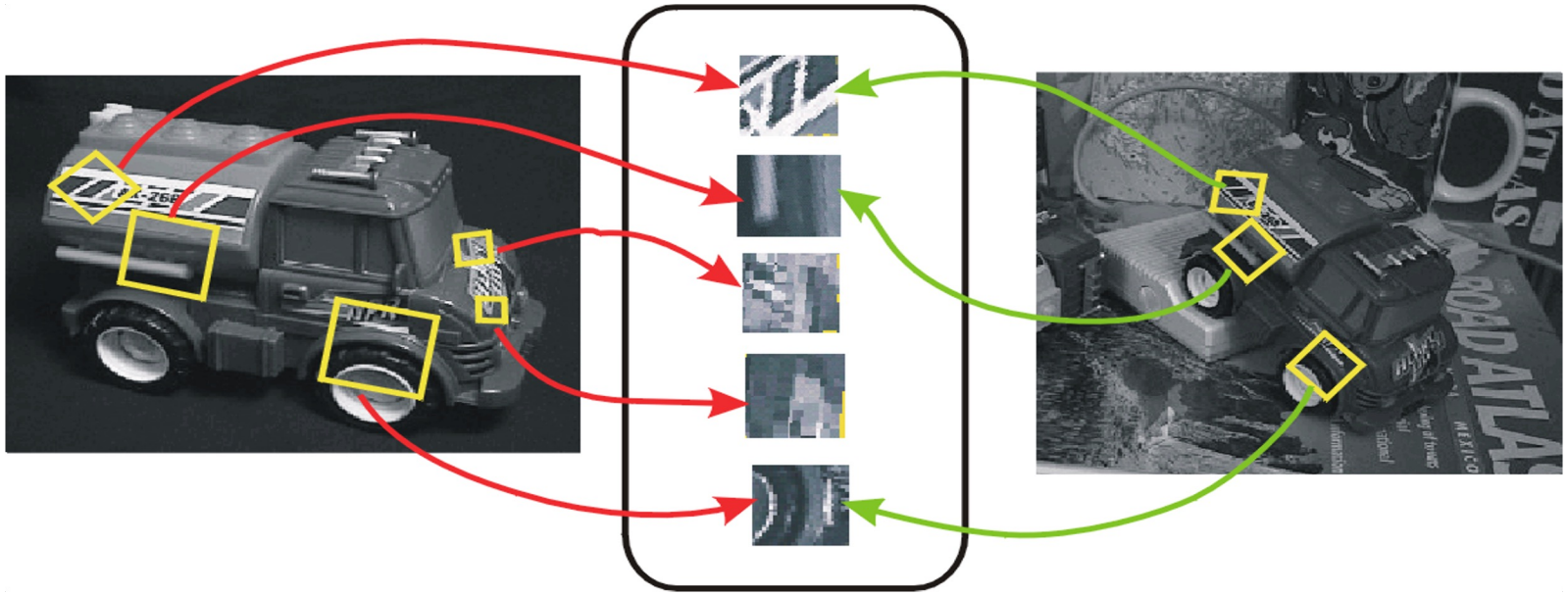




Simple template matching is not useful

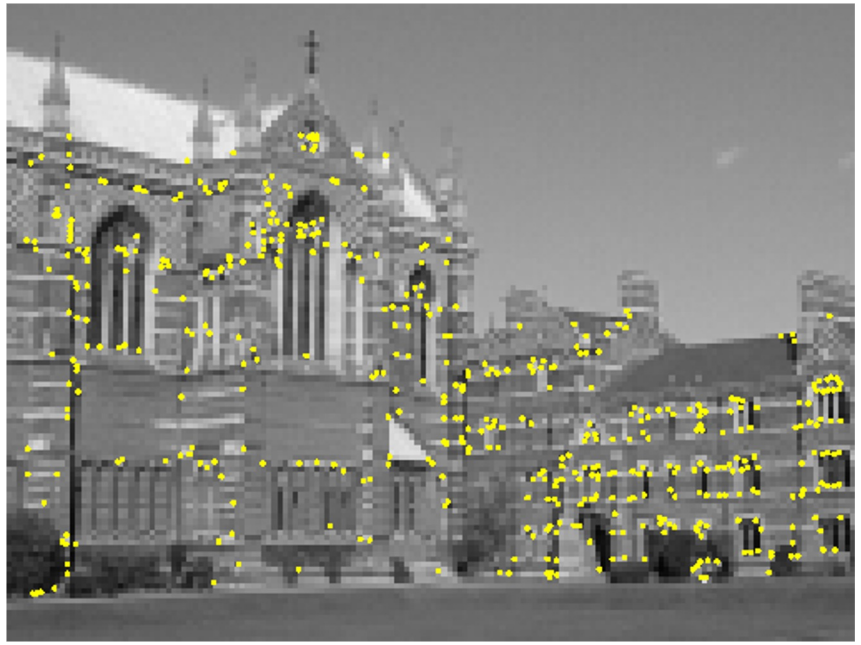
Invariant Local Features

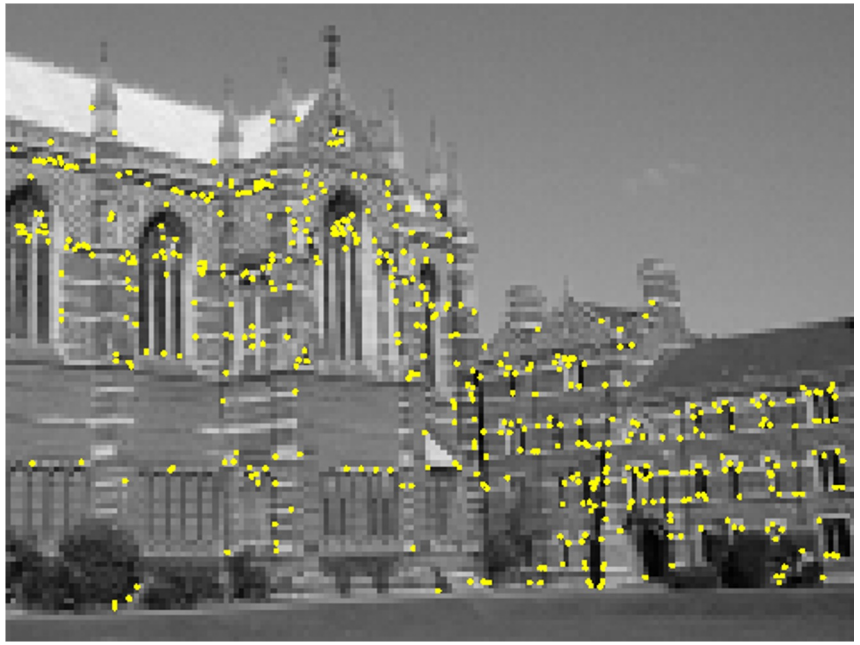
Image content is transformed into local feature coordinates that are invariant to translation, rotation, scale, and other imaging parameters



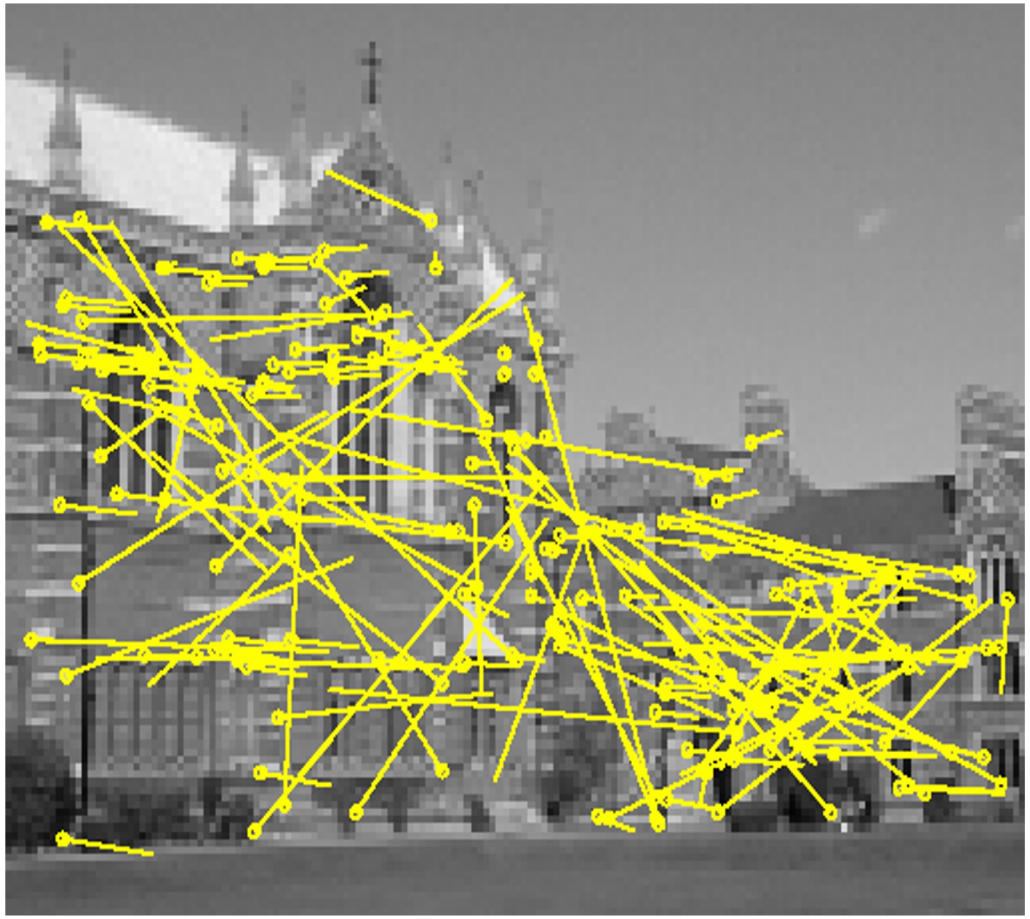
SIFT Features

Harris detector



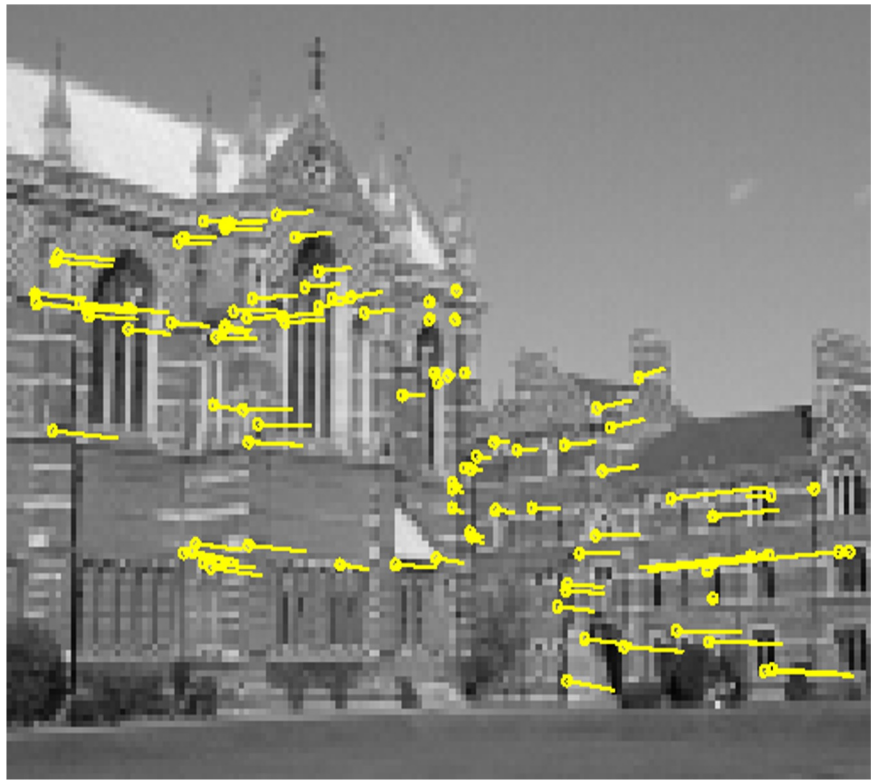


Cross-correlation matching

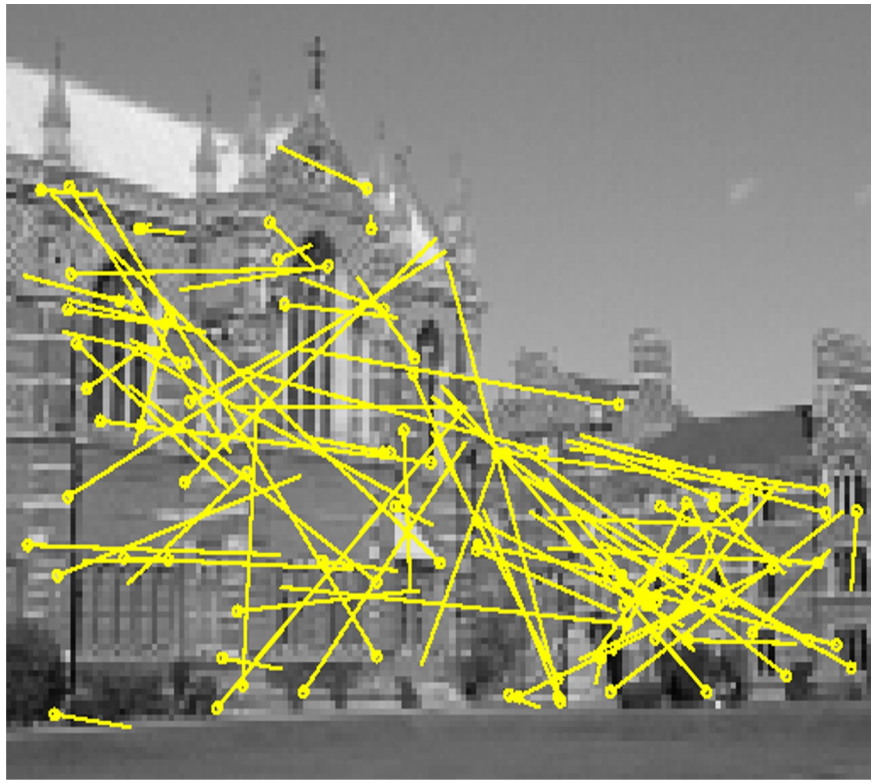


Initial matches (188 pairs)

Global constraints



99
inliers



89 outliers

Context to the rescue!

We know there is a keyboard present in this scene even if we cannot see it clearly.

We know there is no keyboard present in this scene

… even if there is one indeed

Slide by Antonio Torralba

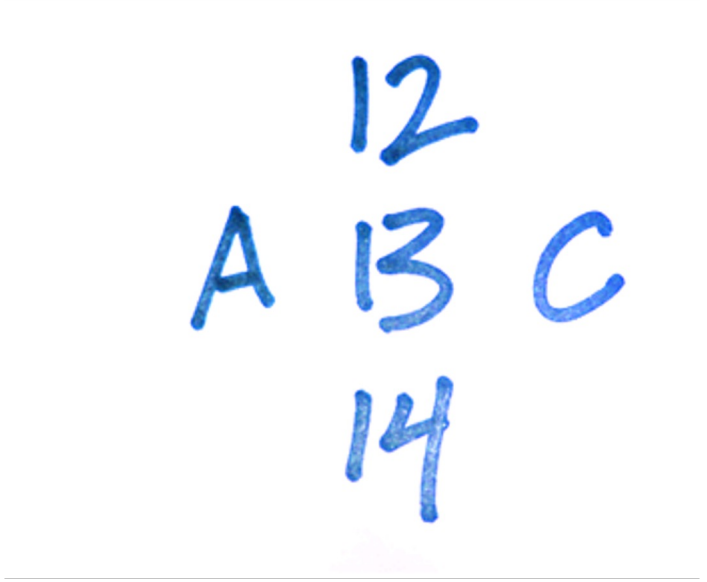
Is it just for small / blurry things?

Slide by Antonio Torralba

Is it just for small / blurry things?

Slide by Antonio Torralba

Is it just for small / blurry things?



Slide by Antonio Torralba

Don’t even need to see the object

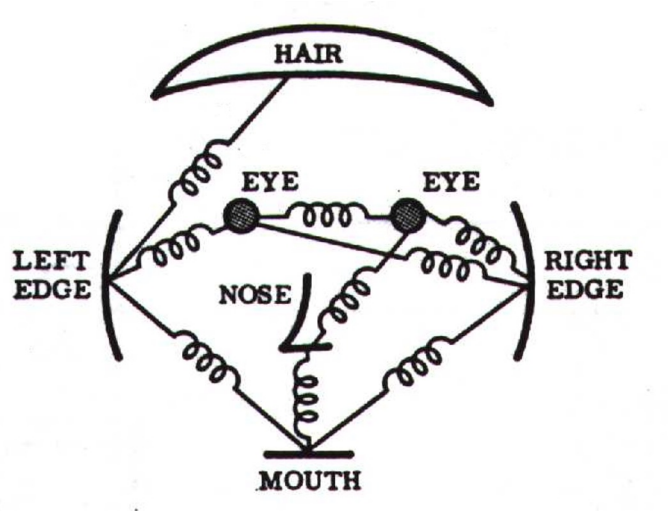
Slide by Antonio Torralba

Don’t even need to see the object

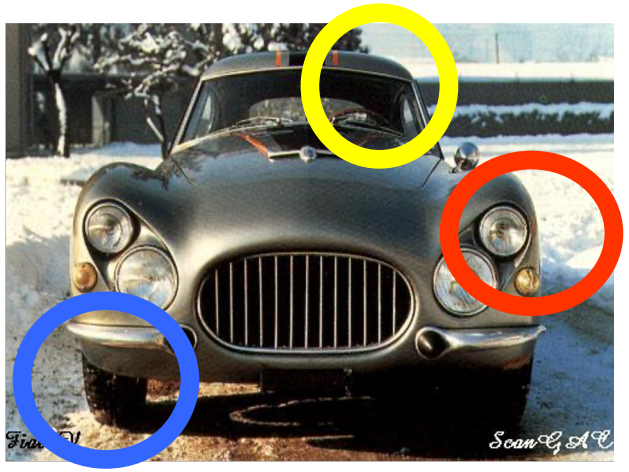
Slide by Antonio Torralba

Constellation of Parts Model

- Constellation of parts models
- Object detection part, category recognition

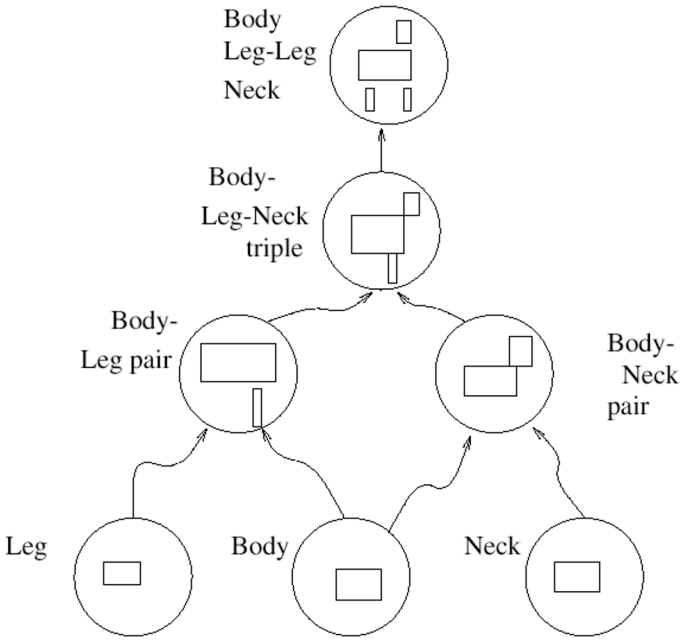


Fischler & Elschlagar, 1973



- [Coots, Taylor et al'95] Univ. of Manchester
- [G.Csurka et al'04] Xerox Research Europe
- [Webber et al'00, Fei-Fei Li et al 03] Caltech
- [Fergus et al'03,04] Oxford

Horses


Body
Leg-Leg
Neck
Body-
Leg-Neck
triple
Body-
Leg pair
Body-
Neck
pair
Leg
Body
Neck

Motivation

Problem with global matching algorithm, like PCA:

- PCA is heavily affected by varying pose and illumination.
- PCA cannot handle extreme change in expression.
- PCA requires uniform background that will not be satisfied in most natural scenes.

Another Approach: Local Image matching.

It works because:

- Locality: features are local, so robust to clutter or neighbours.
- Distinctiveness: They can be easily differentiated from a large database of objects
- Quantity: hundreds or thousands in a single image

Problems:

- These features can match with other object sharing similar feature. Any workaround?

Local + Global image matching, a.k.a Part-based matching:

- Initilally, perform local image matching.
- Find the matching pairs.
- Analyze the global configration/ shape/ topology of the matching pairs.
- Decision will be taken by incorporating both constraints.

Example: Facial Image Analysis

Input

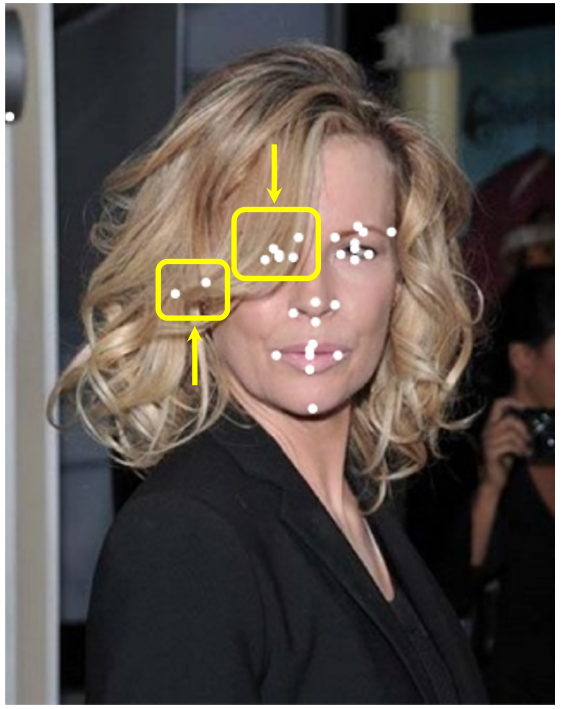
Eye detector

Nose detector

Mouth detector

Chin detector

Shape constraint

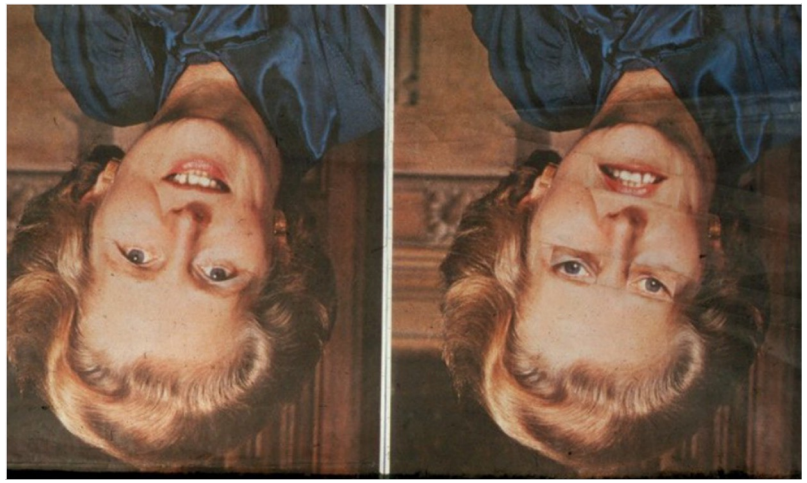


Aim: Detecting facial landmarkpoints

P. N. Belhumeur, D. W. Jacobs, D. J. Kriegman, and N. Kumar. "Localizing parts of faces using a consensus of exemplars". In CVPR, 2011.

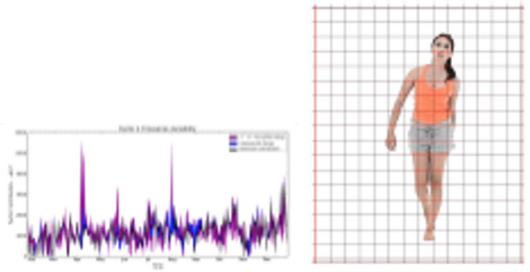
Motivation-II

- Certain research results show that the human brain also processes the images hierarchically
  - Neurons in the visual cortex fire when seeing certain simple graphical primitives like edges having a certain direction
  - Consider the Thatcher illusion – for the upside-down image our brain concludes that the mouth, nose an eyes seem to be correct and in place (they form a face together), but does not realize the error in the fine details



Convolutional Neural Network (CNN)

- Convolutional Neural Networks (CNNs) are useful for processing data with a grid-like topology
  - 1-D grid: time-series data, sensor signal data
  - 2-D grid: image data

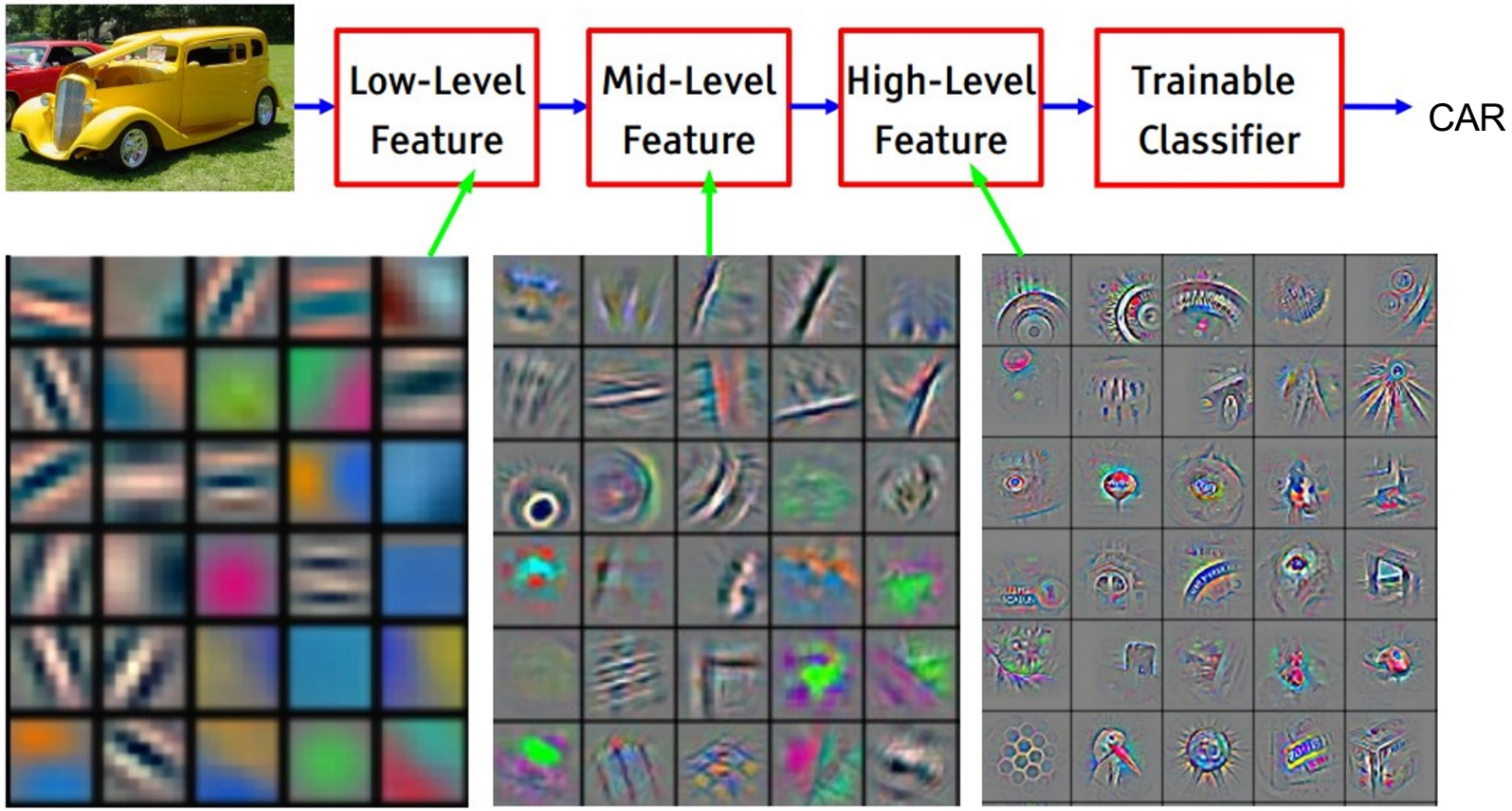


- CNNs are inspired by mammalian visual cortex.
  - The visual cortex contains a complex arrangement of cells, which are sensitive to small sub-regions of the visual field, called a receptive field.
  - These cells act as local filters over the input space and are well-suited to exploit the strong spatially local correlation present in natural images.
  - Two basic cell types:
    - Simple cells respond maximally to specific edge-like patterns within their receptive field.
    - Complex cells have larger receptive fields and are locally invariant to the exact position of the pattern.

Motivations for Deep Architectures

- Insufficient depth can hurt
  - With shallow architecture (SVM, NB, KNN, etc.), the required number of nodes in the graph (i.e. computations, and also number of parameters, when we try to learn the function) may grow very large.
  - Many functions that can be represented efficiently with a deep architecture cannot be represented efficiently with a shallow one.
- The brain has a deep architecture
  - The visual cortex shows a sequence of areas each of which contains a representation of the input, and signals flow from one to the next.
  - Note that representations in the brain are in between dense distributed and purely local: they are **sparse**: about 1% of neurons are active simultaneously in the brain.
- Cognitive processes seem deep
  - Humans organize their ideas and concepts hierarchically.
  - Humans first learn simpler concepts and then compose them to represent more abstract ones.
  - Engineers break-up solutions into multiple levels of abstraction and processing

A Deep Classification Network

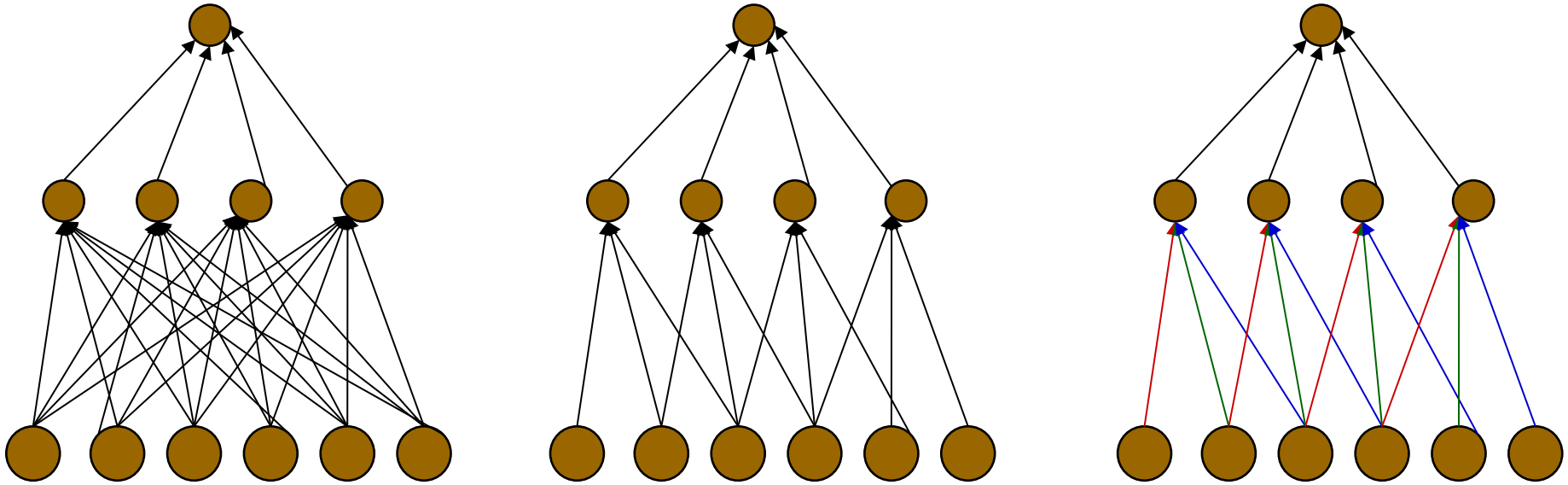


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

CNN Architecture

- Intuition: Neural network with specialized connectivity structure,
  - Stacking multiple layers of feature extractors
  - Low-level layers extract local features.
  - High-level layers extract learn global patterns.
- A CNN is a list of layers that transform the input data into an output class/prediction.
- There are a few distinct types of layers:
  - Convolutional layer
  - Non-linear layer
  - Pooling layer

What is Convolutional Neural Network?



The convolution operation

The convolutional neurons are similar to neurons with the following differences:

- Locality: they process only small parts of the input image
- Weight Sharing: the same neuron is evaluated at many different locations of the image

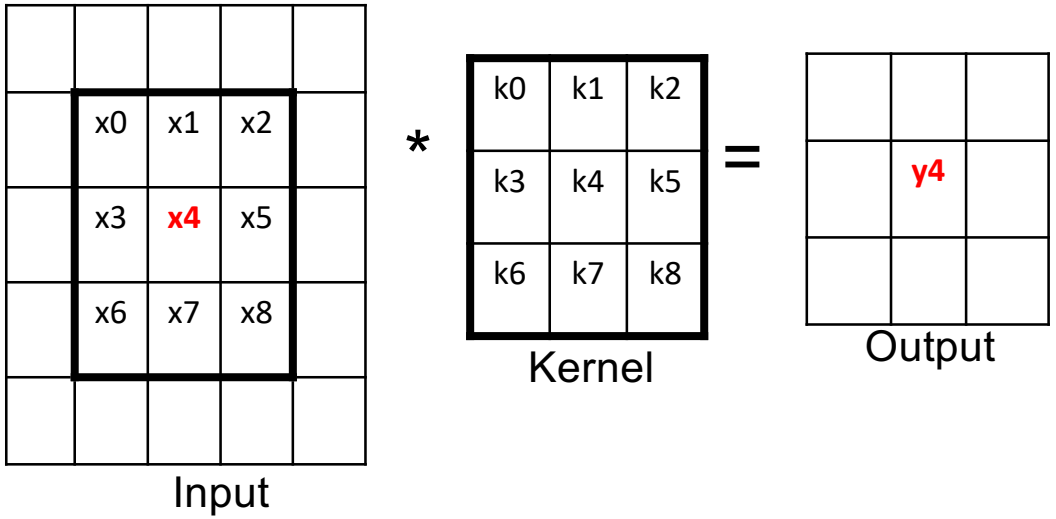


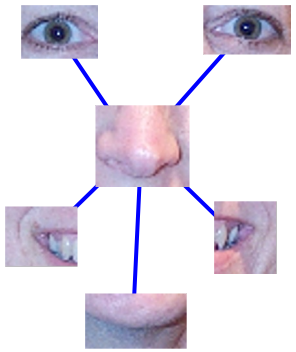
Figure: Computing each output element using 9 pixels

The convolution operation

The operation performed by the neurons is similar to the filters required in image processing, with the differences:

- It apply a nonlinear activation function on the filter output
- Weights will be learnt automatically, not manually
- Strides and padding, just like image processing.
- Several convolutional filters learn several abstract notions (eg. several edge orientations, like Gabor filter at the first layer). Thus, several filters are used and each one provides 1 output. This is referred as the “depth” of the output.

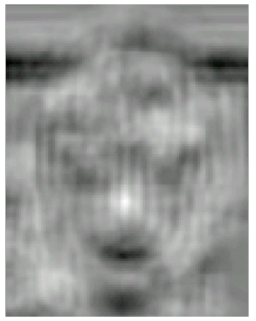
Requirement from the network...

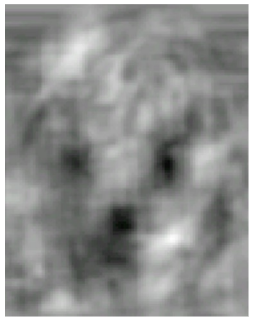


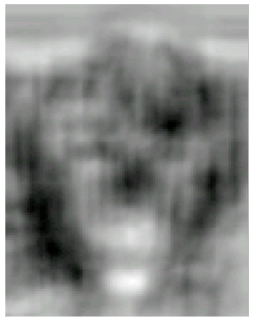
Left eye

Right eye

Nose





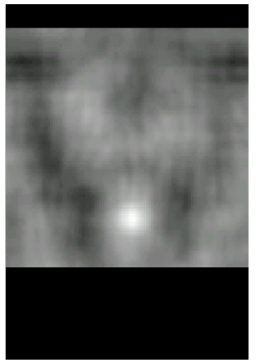


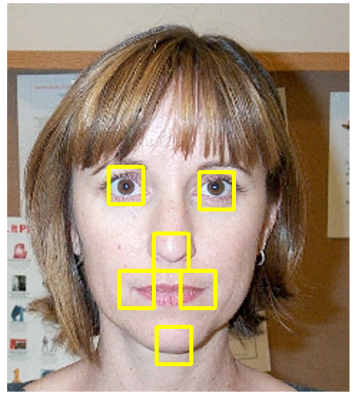
Left mouth

Right mouth

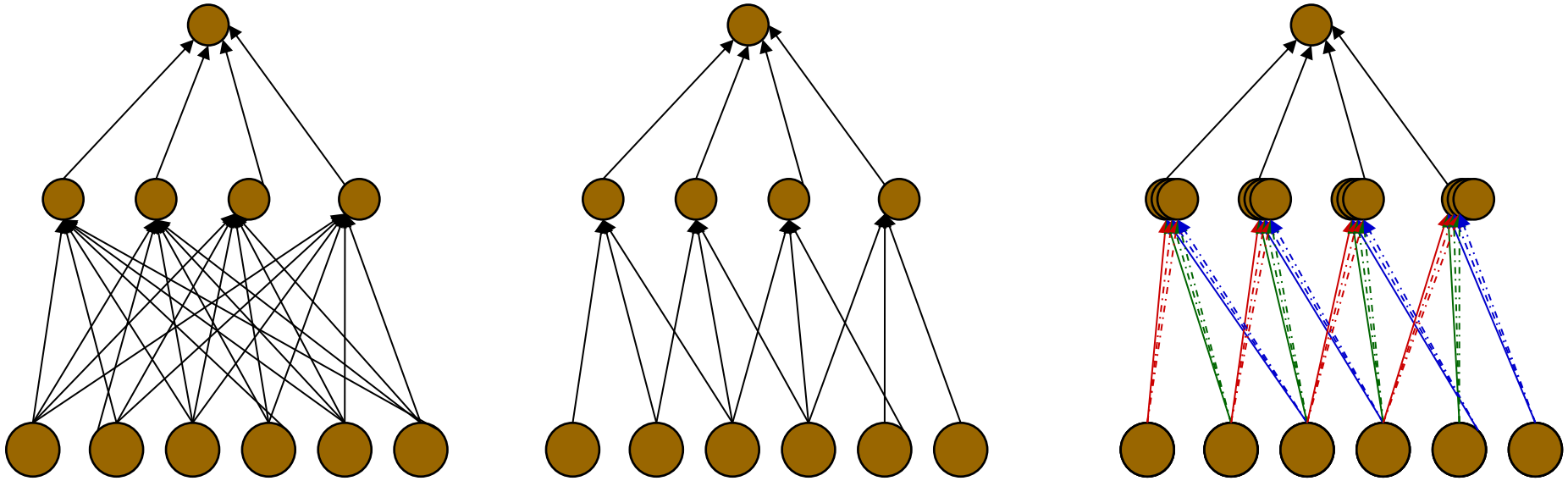
Chin

Marginal on Nose





What is Convolutional Neural Network?



The pooling operation

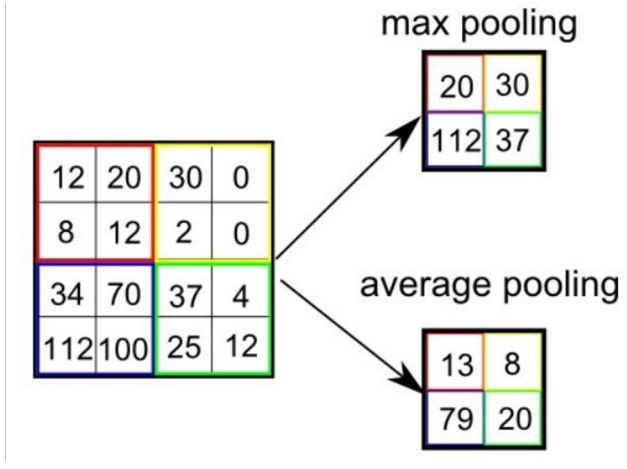
Intuition: As we analyze higher layers, we start neglecting the fine details

- Like, if the nose detector found a nose, then we don't want to keep its precise position and shape in the higher layers

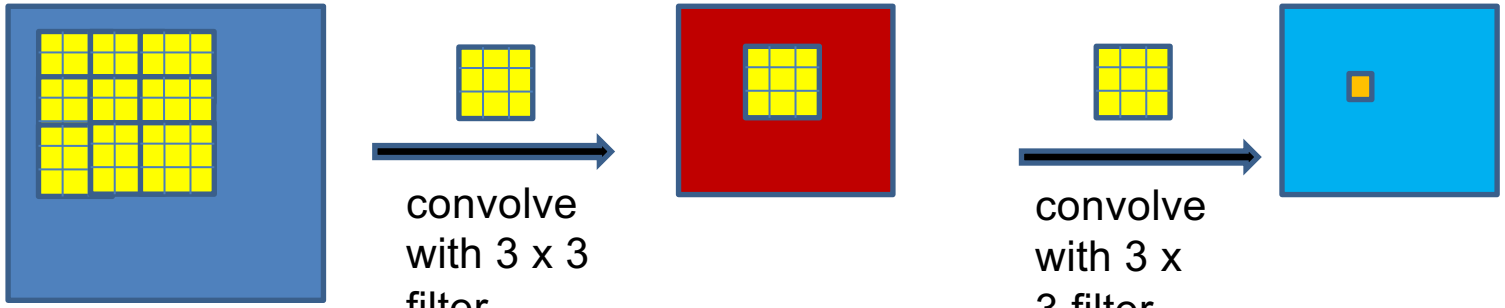
Pooling layer: In a local neighborhood, it neglects the output values

Advantages:

- Shift invariance within the pooling region (no matter where we found the nose, the maximum will be the same)
- We gradually decrease the output size (downsampling) – reducing the number of parameters is always useful

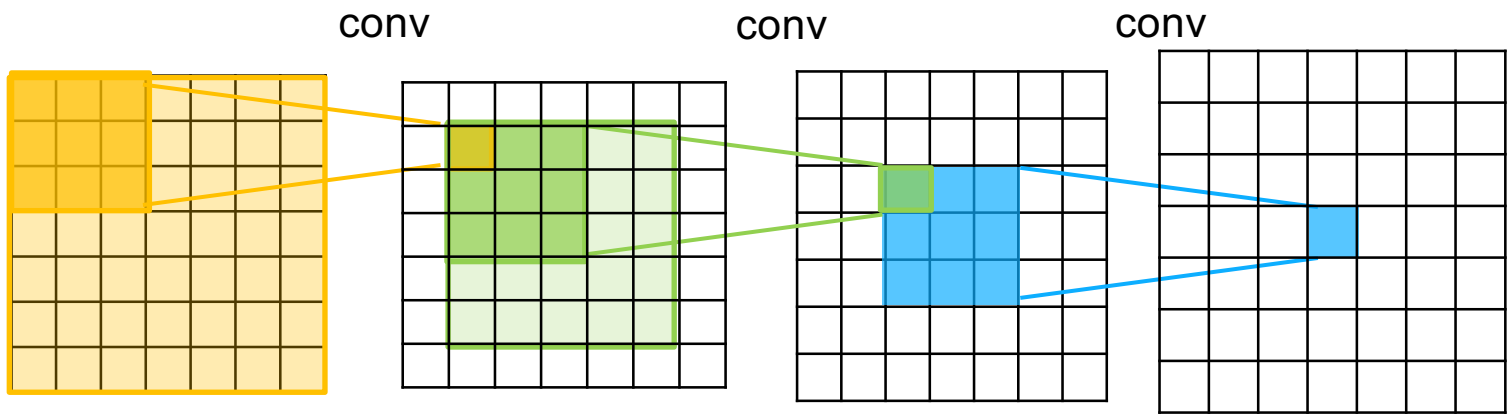


Receptive Field



7x7 receptive field: union of 9 3x3 fields with stride of 2

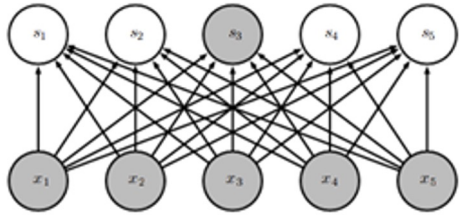
3x3 receptive field



Other Layers

After a few sets, the output is typically sent to one or two **fully connected** layers.

- A fully connected layer is a ordinary neural network layer as in other neural networks.
- Typical activation function is the **sigmoid** function.



The final layer of a CNN is determined based on task:

- Classification: Softmax Layer

The outputs are the probabilities of belonging to each class.

- Regression: Linear Layer

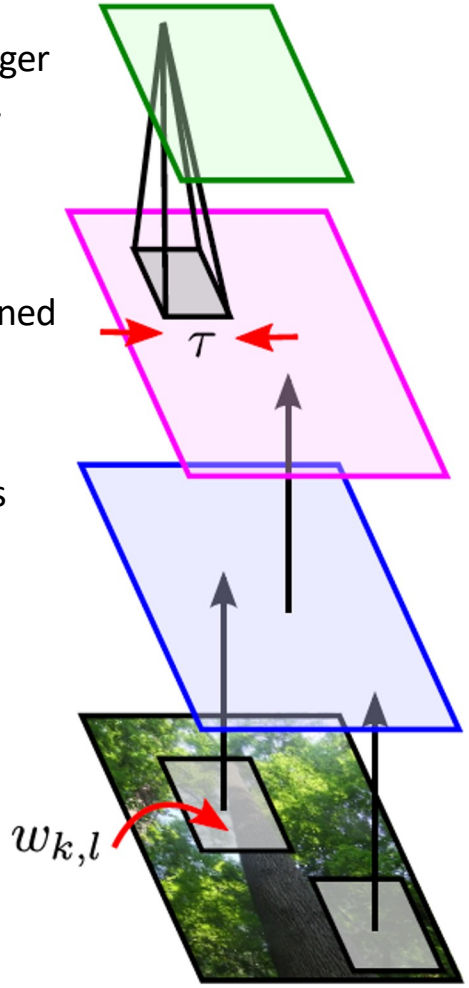
The output is a real number.

Building-blocks for CNNs

Feature maps of a larger region are combined.

Feature maps are trained with neurons.

Each sub-region yields a feature map, representing its feature.



$$x_{i,j} = \max_{|k|<\tau,|l|<\tau} y_{i-k,j-l}$$

mean or subsample also used

pooling stage

$$y_{i,j} = f(a_{i,j})$$

e.g. $f(a) = [a]_+$

$f(a) = \text{sigmoid}(a)$

non-linear stage

$$a_{i,j} = \sum_{k,l} w_{k,l} z_{i-k,j-l}$$

convolutional stage

only parameters

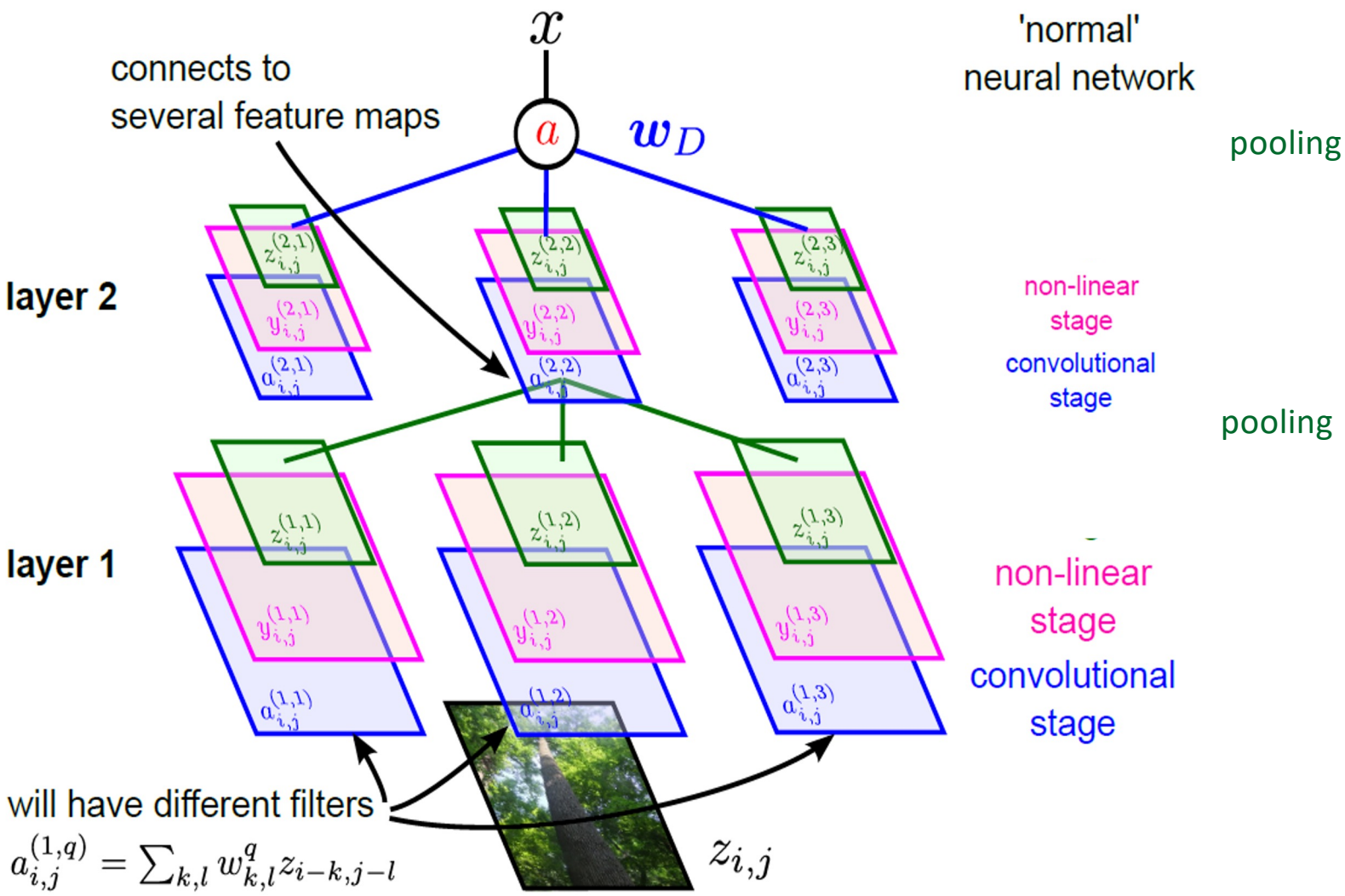
Shared weights

$z_{i,j}$

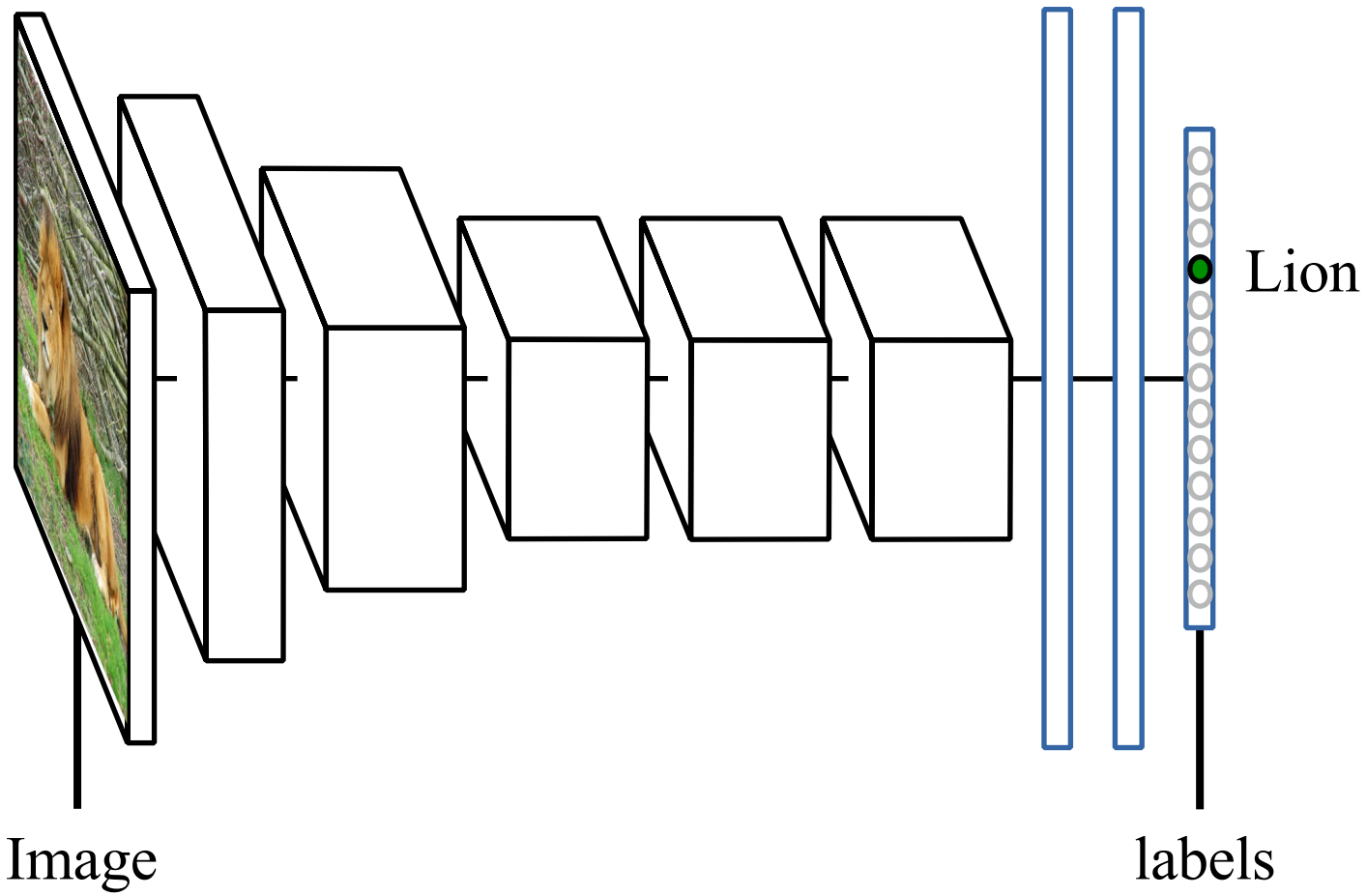
Images are segmented into sub-regions.

input image

Full CNN

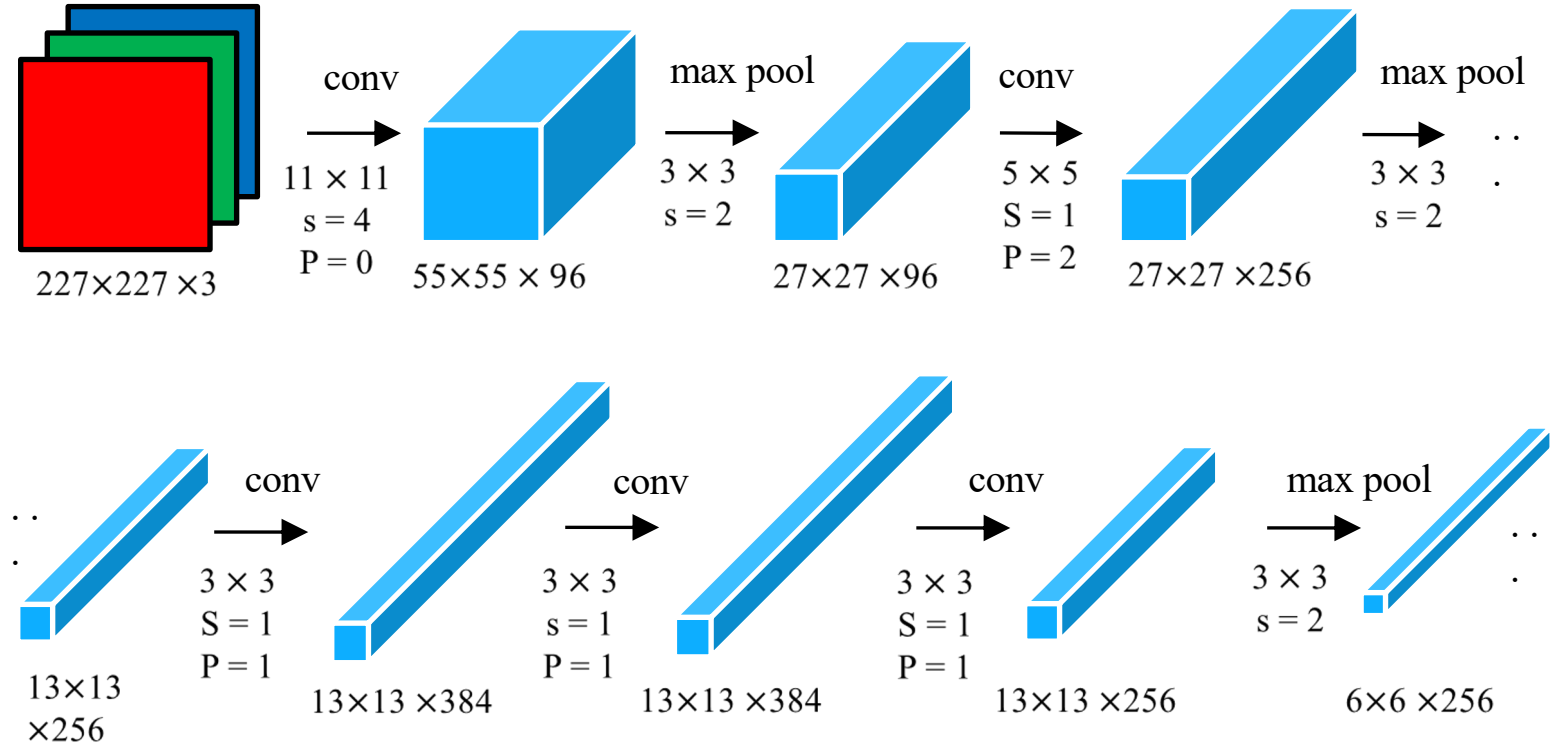

x
connects to
several feature maps
a
w_D
'normal'
neural network
pooling
layer 2
non-linear
stage
convolutional
stage
pooling
layer 1
non-linear
stage
convolutional
stage
will have different filters
$a_{i,j}^{(1,q)} = \sum_{k,l} w_{k,l}^{q} z_{i-k,j-l}$
$z_{i,j}$

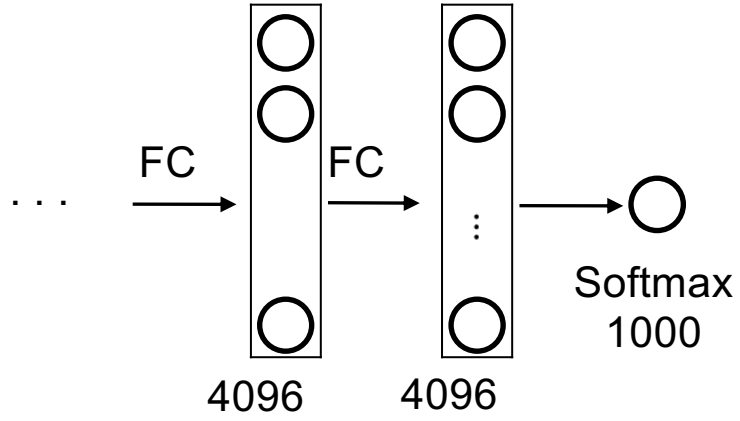
Convolutional Neural Networks: **AlexNet**

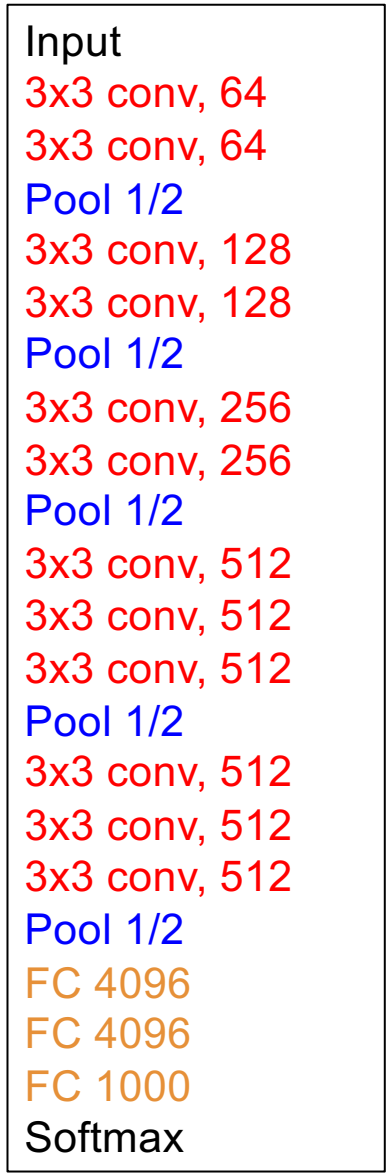


Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton: ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012

AlexNet


227×227 ×3
conv
11 × 11
s = 4
P = 0
55×55 × 96
max pool
3 × 3
s = 2
27×27 ×96
conv
5 × 5
S = 1
P = 2
27×27 ×256
max pool
3 × 3
s = 2
13×13
×256
conv
3 × 3
S = 1
P = 1
13×13 ×384
conv
3 × 3
s = 1
P = 1
13×13 ×384
conv
3 × 3
S = 1
P = 1
13×13 ×256
max pool
3 × 3
s = 2
6×6 ×256


FC
FC
Softmax
1000
4096
4096



VGGNet

- **Smaller filters**
  Only 3x3 CONV filters, stride 1, pad 1 and 2x2 MAX POOL , stride 2
- **Deeper network**
  AlexNet: 8 layers
  VGGNet: 16 - 19 layers
- VGGNet: 7.3% top 5 error in ILSVRC'14

Why use smaller filters? (3x3 conv)
Stack of three 3x3 conv (stride 1) layers has the same effective receptive field as one 7x7 conv layer.

What is the effective receptive field of three 3x3 conv (stride 1) layers?
7x7
But deeper, more non-linearities
And fewer parameters: 3 * (3^2C^2) vs. 7^2C^2 for C channels per layer

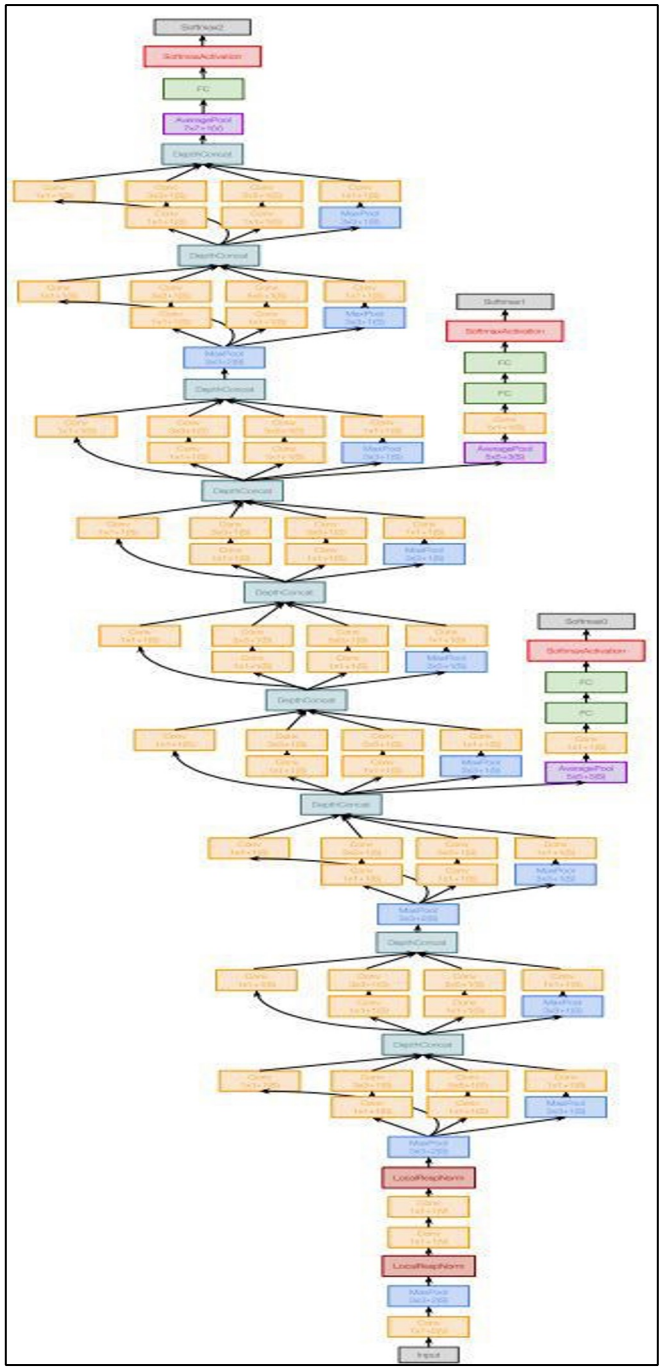
Karen Simonyan, Andrew Zisserman:**Very Deep Convolutional Networks for Large-Scale Image Recognition, ICLR, 2015**

VGG-16 Net

Input	memory: 224*224*3=150K	params: 0
3x3 conv, 64	memory: 224*224*64=3.2M	params: (3*3*3)*64 = 1,728
3x3 conv, 64	memory: 224*224*64=3.2M	params: (3*3*64)*64 = 36,864
Pool	memory: 112*112*64=800K	params: 0
3x3 conv, 128	memory: 112*112*128=1.6M	params: (3*3*64)*128 = 73,728
3x3 conv, 128	memory: 112*112*128=1.6M	params: (3*3*128)*128 = 147,456
Pool	memory: 56*56*128=400K	params: 0
3x3 conv, 256	memory: 56*56*256=800K	params: (3*3*128)*256 = 294,912
3x3 conv, 256	memory: 56*56*256=800K	params: (3*3*256)*256 = 589,824
3x3 conv, 256	memory: 56*56*256=800K	params: (3*3*256)*256 = 589,824
Pool	memory: 28*28*256=200K	params: 0
3x3 conv, 512	memory: 28*28*512=400K	params: (3*3*256)*512 = 1,179,648
3x3 conv, 512	memory: 28*28*512=400K	params: (3*3*512)*512 = 2,359,296
3x3 conv, 512	memory: 28*28*512=400K	params: (3*3*512)*512 = 2,359,296
Pool	memory: 14*14*512=100K	params: 0
3x3 conv, 512	memory: 14*14*512=100K	params: (3*3*512)*512 = 2,359,296
3x3 conv, 512	memory: 14*14*512=100K	params: (3*3*512)*512 = 2,359,296
3x3 conv, 512	memory: 14*14*512=100K	params: (3*3*512)*512 = 2,359,296
Pool	memory: 7*7*512=25K	params: 0
FC 4096	memory: 4096	params: 7*7*512*4096 = 102,760,448
FC 4096	memory: 4096	params: 4096*4096 = 16,777,216
FC 1000	memory: 1000	params: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 96MB / image

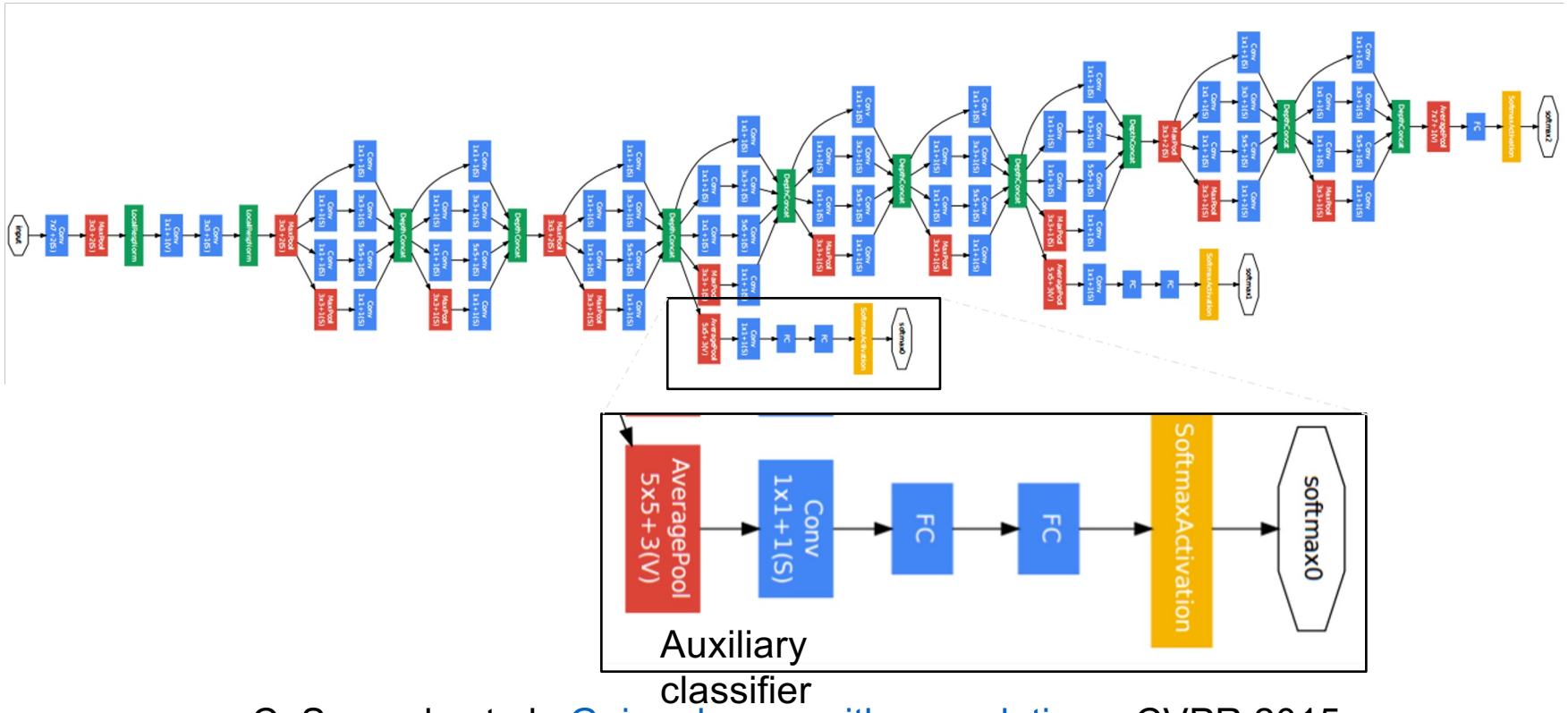
TOTAL params: 138M parameters



GoogleNet

- Significantly deeper than AlexNet (22 layers)
- x12 less parameters than AlexNet
- Focused on computational efficiency
- Efficient **“Inception” module** - strayed from the general approach of simply stacking conv and pooling layers on top of each other in a sequential structure. Instead of simply going deeper in terms of the number of layers, it goes wider. Multiple kernels of different sizes are implemented within the same layer.
- No FC layers. Replaces it with a simple global average pooling which averages out the channel values across the 2D feature map, after the last convolutional layer.
- Only 5 million parameters!
- ILSVRC’14 classification winner (6.7% top 5 error)

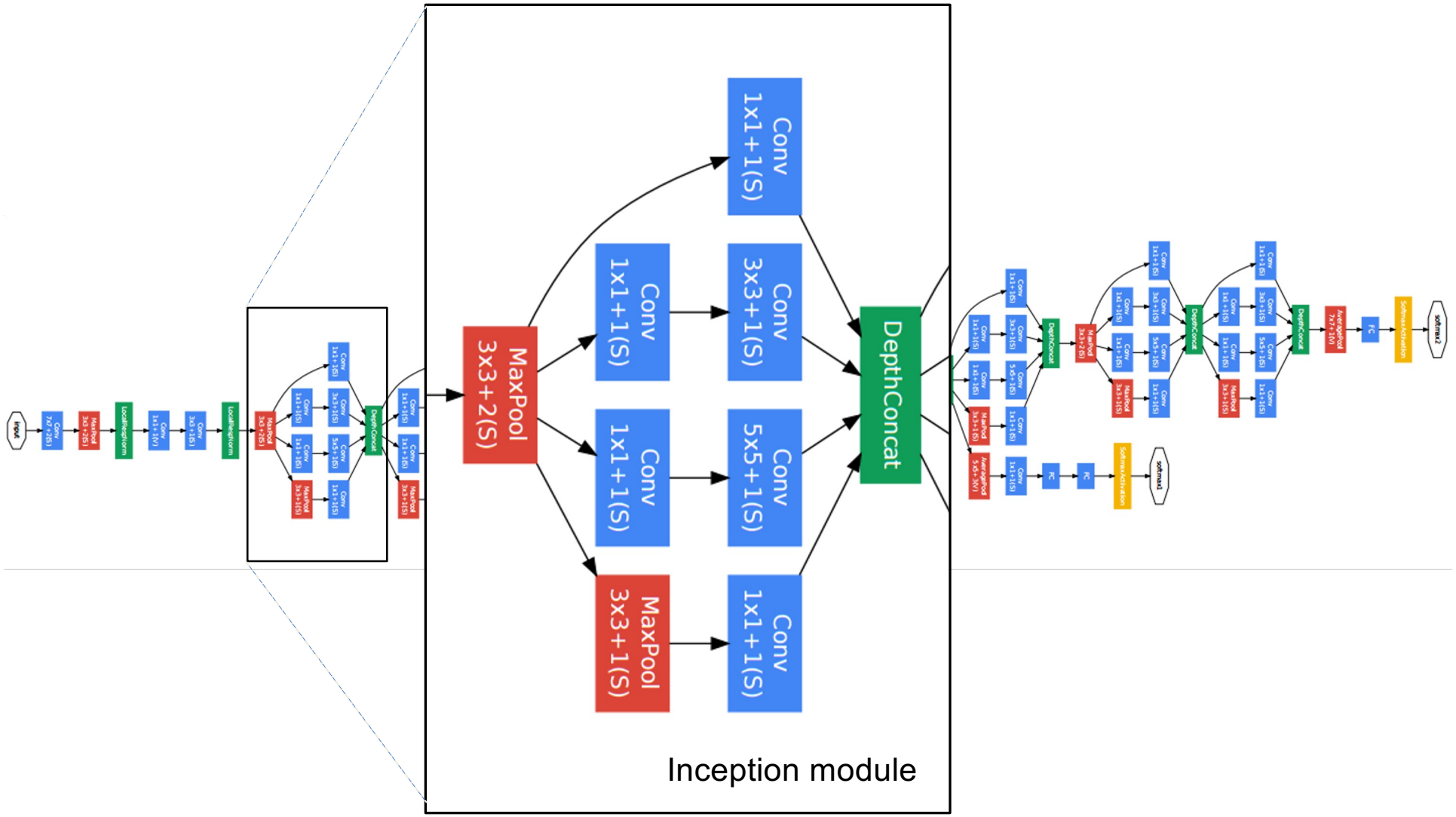
GoogLeNet



C. Szegedy et al., Going deeper with convolutions, CVPR 2015

- By adding auxiliary classifiers connected to these intermediate layers, we expect to encourage discrimination in the lower stages in the classifier and increase the gradient signal that gets propagated back
- During training, their loss gets added to the total loss of the network with a discount weight (the losses of the auxiliary classifiers were weighted by 0.3).
- At inference time, these auxiliary networks are discarded.

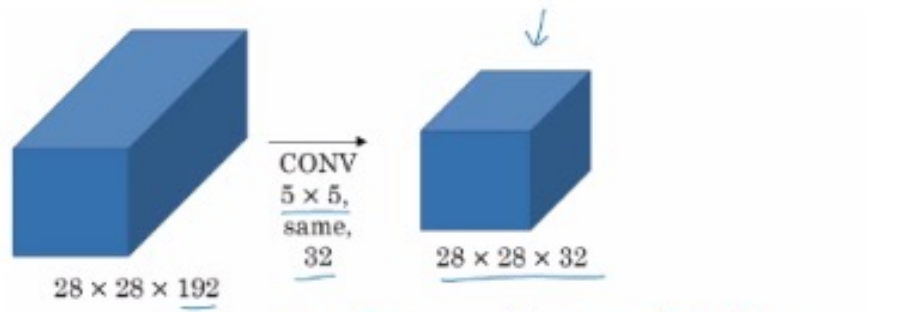
GoogLeNet



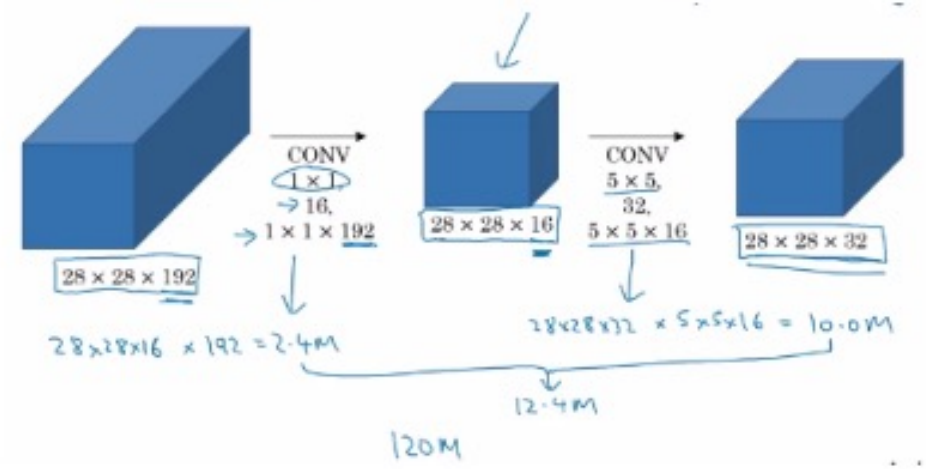
C. Szegedy et al., Going deeper with convolutions, CVPR 2015

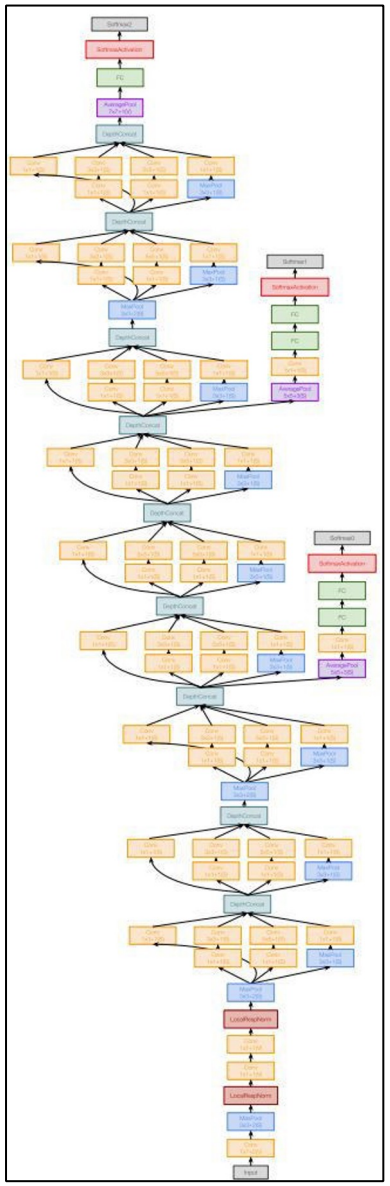
- Also, By using 1*1 filter you can reduce 120M to12M by a factor of 10.

i.



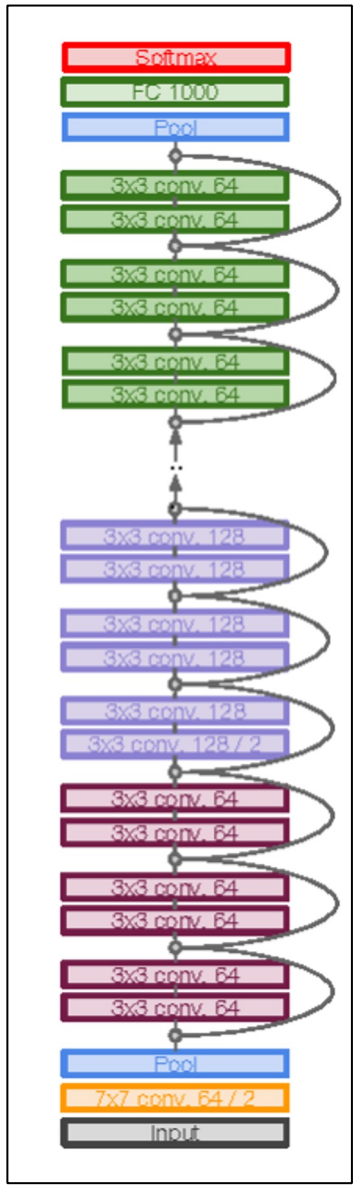
ii.





GoogleNet

- Deeper networks, with computational efficiency
- 22 layers
- Efficient “Inception” module
- No FC layers
- 12x less params than AlexNet
- ILSVRC’14 classification winner (6.7% top 5 error)

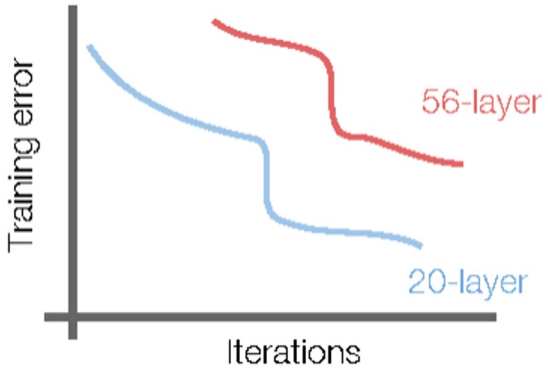


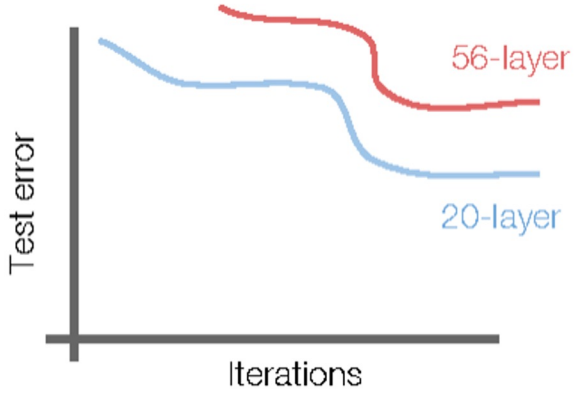
ResNet

- ILSVRC'15 classification winner (3.57% top 5 error, humans generally hover around a 5-10% error rate)
  Swept all classification and detection competitions in ILSVRC'15 and COCO'15!

ResNet

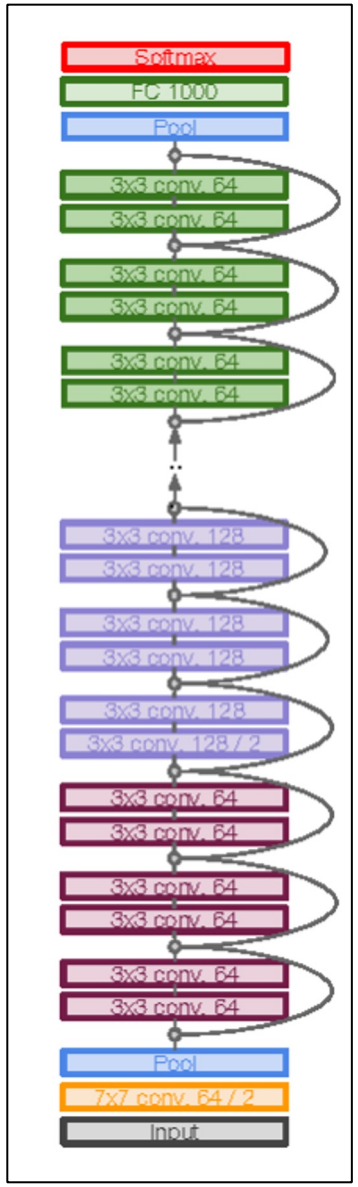
- What happens when we continue stacking deeper layers on a convolutional neural network?





- 56-layer model performs worse on both training and test error

-> The deeper model performs worse (not caused by overfitting)!



ResNet

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double the number of filters and downsample spatially using stride 2 (in each dimension)
- Additional conv layer at the beginning
- No FC layers at the end (only FC 1000 to output classes)

- Total depths of 34, 50, 101, or 152 layers for ImageNet
- For deeper networks (ResNet-50+), use “bottleneck” layer to improve efficiency (similar to GoogLeNet)

Improving Stochastic Gradient Descent

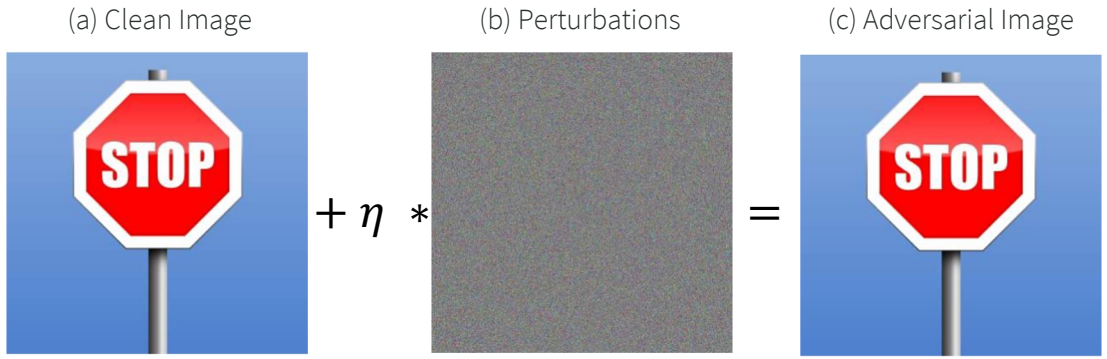
- **Shuffling**: Generally, providing the training examples in a meaningful order to a model is avoided, as it may bias the optimization algorithm. Thus, it is often a good idea to shuffle the training data after every epoch.
- **Curriculum Learning**: For some cases, we aim to solve progressively harder problems by supplying the training examples in a meaningful order may actually lead to improved performance and better convergence. The method for establishing this meaningful order is called Curriculum Learning.
- **Early stopping**: One should always monitor error on a validation set during training and stop (with some patience) if your validation error does not improve enough.
- **Gradient noise**: Gaussian noise is added in the gradients. It is shown that that adding this noise makes networks more robust to poor initialization and helps training particularly deep and complex networks. They suspect that the added noise gives the model more chances to escape and find new local minima, which are more frequent for deeper models.

Several other such tricks can be found from: "LeCun, Y., Bottou, L., Orr, G. B., & Müller, K. R. (1998). Efficient BackProp. Neural Networks: Tricks of the Trade, 1524, 9–50. http://doi.org/10.1007/3-540-49430-8_2"

Limitations of CNN

Susceptibility to adversarial attacks

- CNNs are susceptible to adversarial attacks [1].
- An adversarial attack is addition of well-crafted imperceptible noise to a clean input with an aim to force the model to misbehave.



Ref : [1] Gupta, Puneet, and Esa Rahtu. "MLAttack: Fooling semantic segmentation networks by multi-layer attacks." German Conference on Pattern Recognition. Springer, Cham, 2019.

Limitations of CNN

Reliability on texture rather than shapes

- CNNs, unlike humans, tend to rely on texture more than shape, which degrades performance [1].

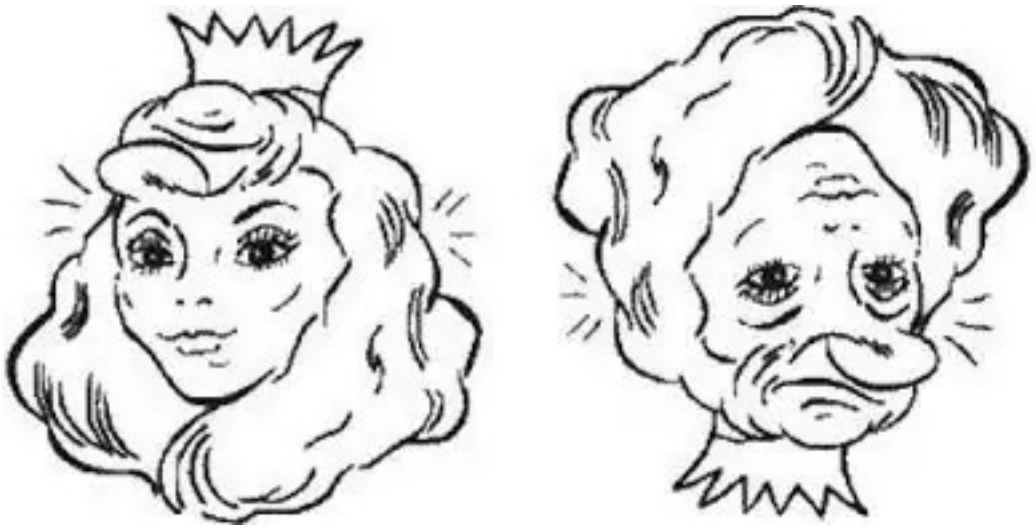
Inability to model global features and long-range dependencies

- CNNs, works to local information and hence fails to encapsulate global features [2].
- CNNs fails to preserve long-range dependencies and hence performs poorly in case of sequential inputs [2].

Ref s: [1] Geirhos, Robert, et al. "ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness." International Conference on Learning Representations, 2019.
[2] Khan, Salman, et al. "Transformers in vision: A survey." ACM computing surveys (CSUR), 2022.

Some limitations that can be mitigated

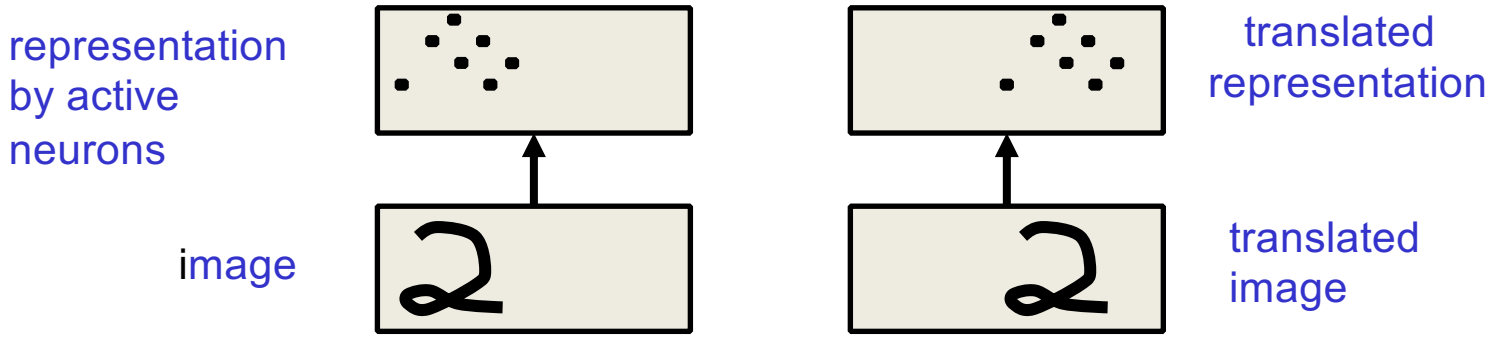
- Classification of Images with different Positions
- Coordinate Frame
- Limited performance
- Gradient vanishing
- Insufficient data



Improving the performance

What data augmentation?

- Equivariant activities: Replicated features do not make the neural activities invariant to translation. The activities are equivariant.



- Invariant knowledge: If a feature is useful in some locations during training, detectors for that feature will be available in all locations during testing.

Data augmentation is a technique to artificially create new training data from existing training data. This is done by applying domain-specific techniques to examples from the training data that create new and different training examples.

Goals for Data Augmentation

- Avoiding overfitting by diversifying the training samples;
- Discovering new principles for given tasks;
- Generalizing network

Data Augmentation

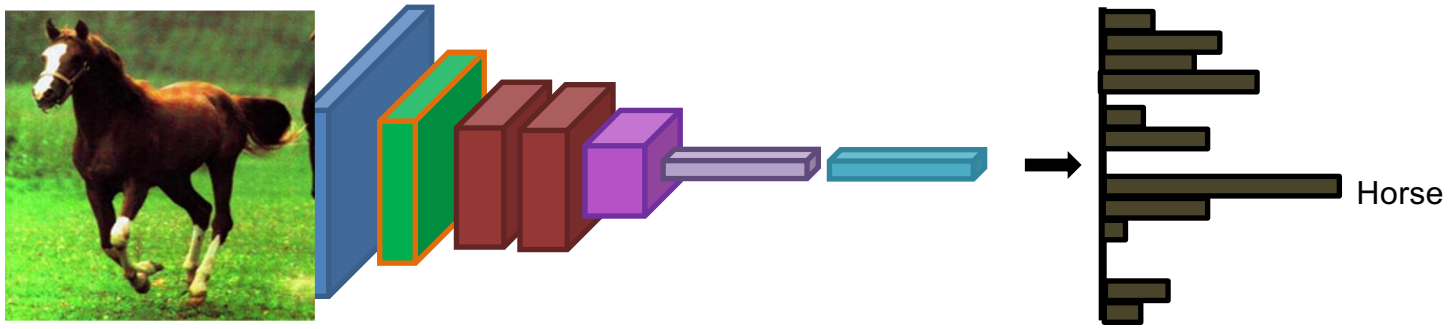
Base Augmentations

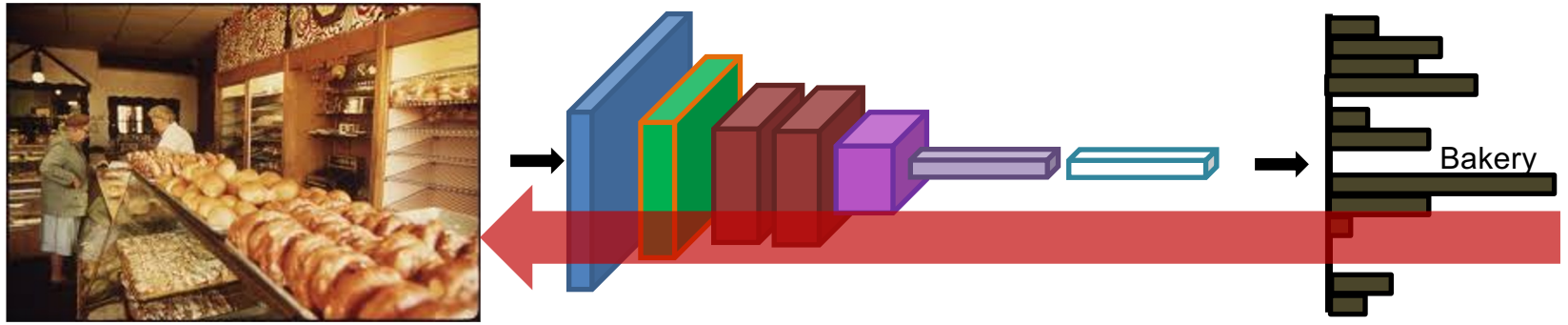
Fine-tuning

We will take a CNN pre-trained on the ImageNet dataset and fine-tune it to perform image classification and recognize classes it was never trained on.
Fine-tuning is a multi-step process:

- Remove the fully connected nodes at the end of the network (i.e., where the actual class label predictions are made).
- Replace the fully connected nodes with freshly initialized ones.
- Freeze earlier CONV layers earlier in the network (ensuring that any previous robust features learned by the CNN are not destroyed).
- Start training, but only train the FC layer heads.
- Optionally unfreeze some/all of the CONV layers in the network and perform a second pass of training.

Fine-tuning

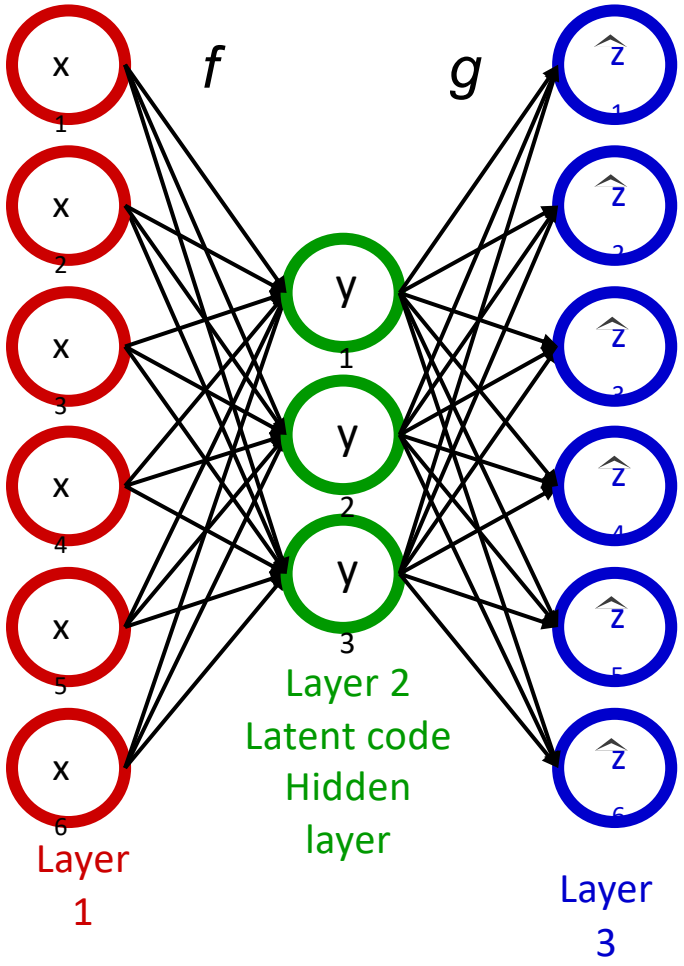




Initialize with pre-trained, then train with low learning rate

$$w^{t+1} = w^t - \alpha \cdot \frac{\delta l}{\delta w}(w^t)$$

Unsupervised DNN: Auto-encoders



Unsupervised. No “label” at the output layer. Output layer simply tries to “recreate” the input.

Defined by two (possibly nonlinear) mapping functions: Encoding function *f* , Decoding function *g*

- y = *f*(x) denotes an encoding (possibly nonlinear) for the input x
- z = *g*(y) = *g* (*f* (x)) denotes the reconstruction (or the “decoding”) for the input x

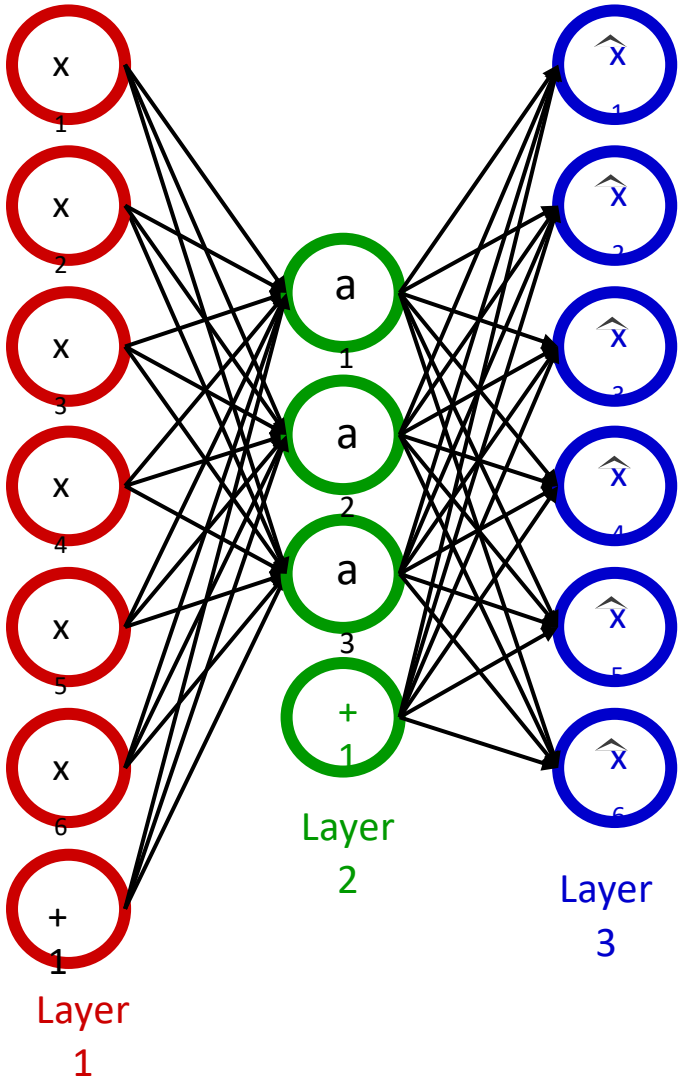
For an Autoencoder, *f* and *g* are learned with a goal to minimize the difference between z and x

$$\mathbf{y} = f_{\theta_1}(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$

$$\mathbf{z} = g_{\theta_2}(\mathbf{W}_2\mathbf{y} + \mathbf{b}_2)$$

$$\theta_1 = \{\mathbf{W}_1, \mathbf{b}_1\}, \theta_2 = \{\mathbf{W}_2, \mathbf{b}_2\}$$

Training Auto-encoders



- To train the autoencoder, we first need to define a loss function
- What is loss function?
- In absence of labeled, how one can define the loss function.
- Example, squared loss when input are real-valued and cross-entropy loss when inputs are binary.
- We find unknown parameters by minimizing the loss function summed over all unlabeled training data, using backpropagation.

Regularized Auto-encoders

Assume dimensions of x and y are DX1 and KX1 respectively.

Undercomplete (K is less than D): Cannot achieve very small reconstruction error.

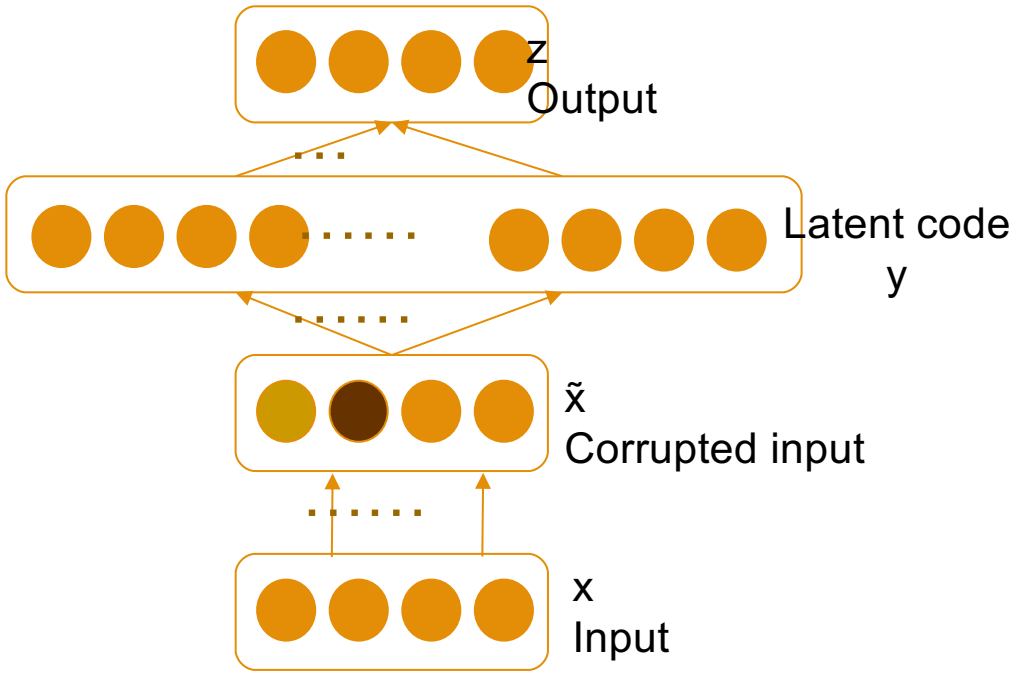
Overcomplete(K is greater than D): Trivial (identity) functions for f and g can result in zero reconstruction error but the learned code will not capture any interesting properties in the data.

It is therefore important to regularize the functions as well as the learned code, and not just focus on minimizing the reconstruction error.

Some examples of regularized auto-encoders are:

- Sparse Autoencoders: Make the learned code sparse
- Denoising Autoencoders: Make the model robust against noisy/incomplete inputs
- Contractive Autoencoders: Make the model robust against small changes in the input

Denoising Auto-encoders



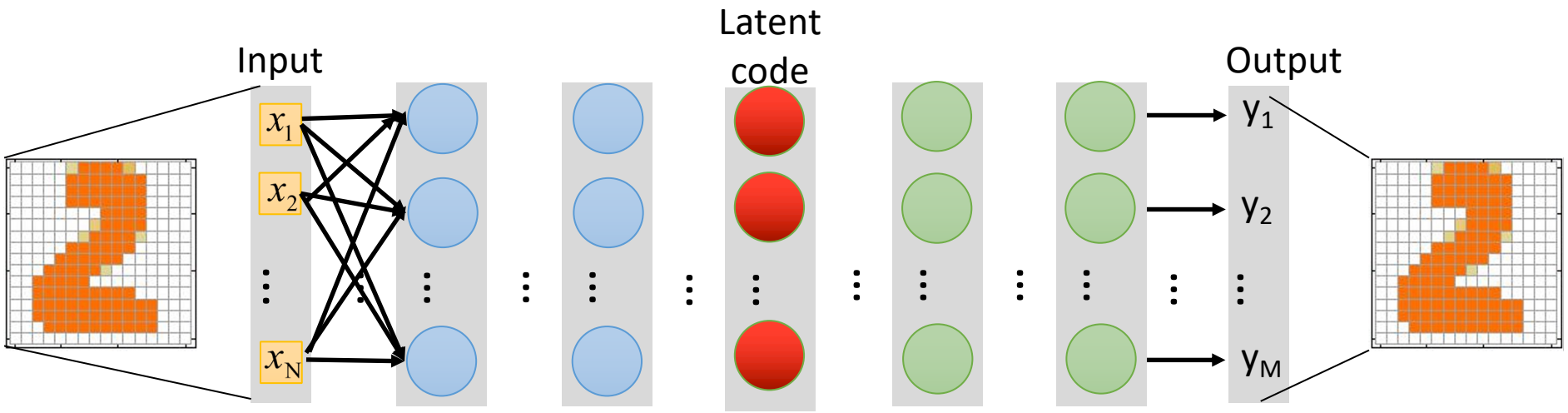
Some noise (e.g., Gaussian noise) can be added to the original input x.

Let's denote $\tilde{x}$ as the corrupted version of x
The encoder f operates on $\tilde{x}$, i.e., $h = f(\tilde{x})$.

However, we still want to reconstruction $\hat{x}$ to be close to the original uncorrupted input x.

Since the corruption is stochastic, we minimize the expected loss:
$E_{\tilde{x}\sim p(\tilde{x}|x)} [L(z,\tilde{x})] + \Omega(y)$

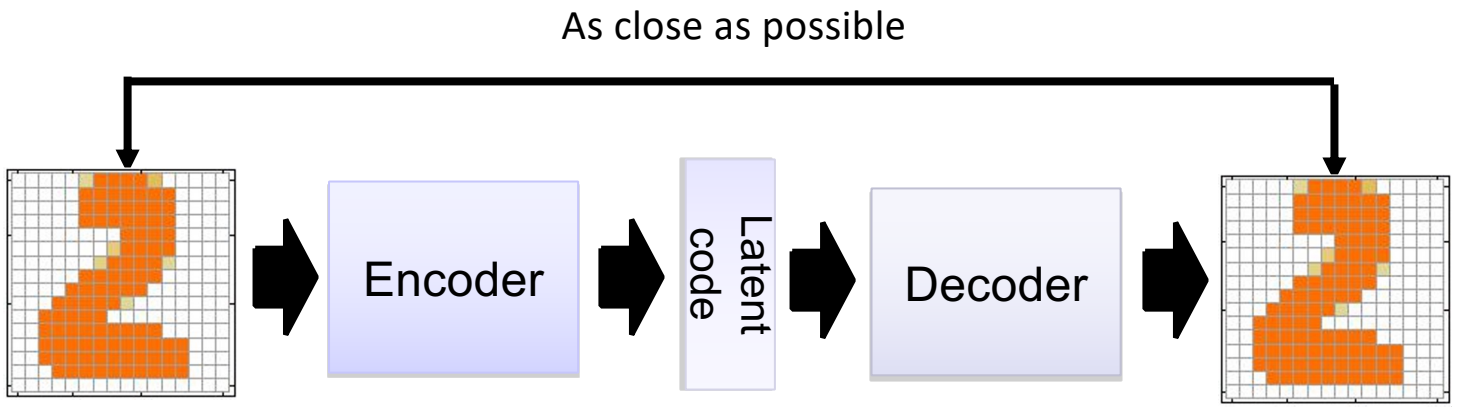
Deep/stacked Auto-encoders



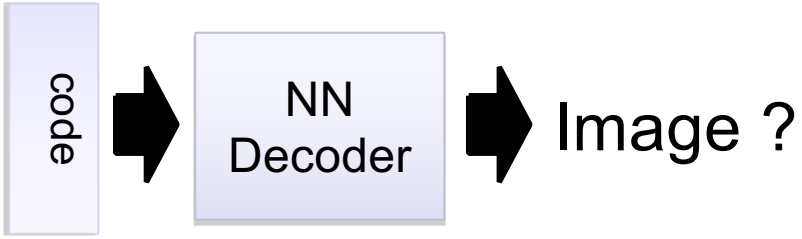
Most autoencoders can be extended to have more than one hidden layer

Using deep encoders and decoders offers the advantages of usual feedforward networks and hierarchical learning

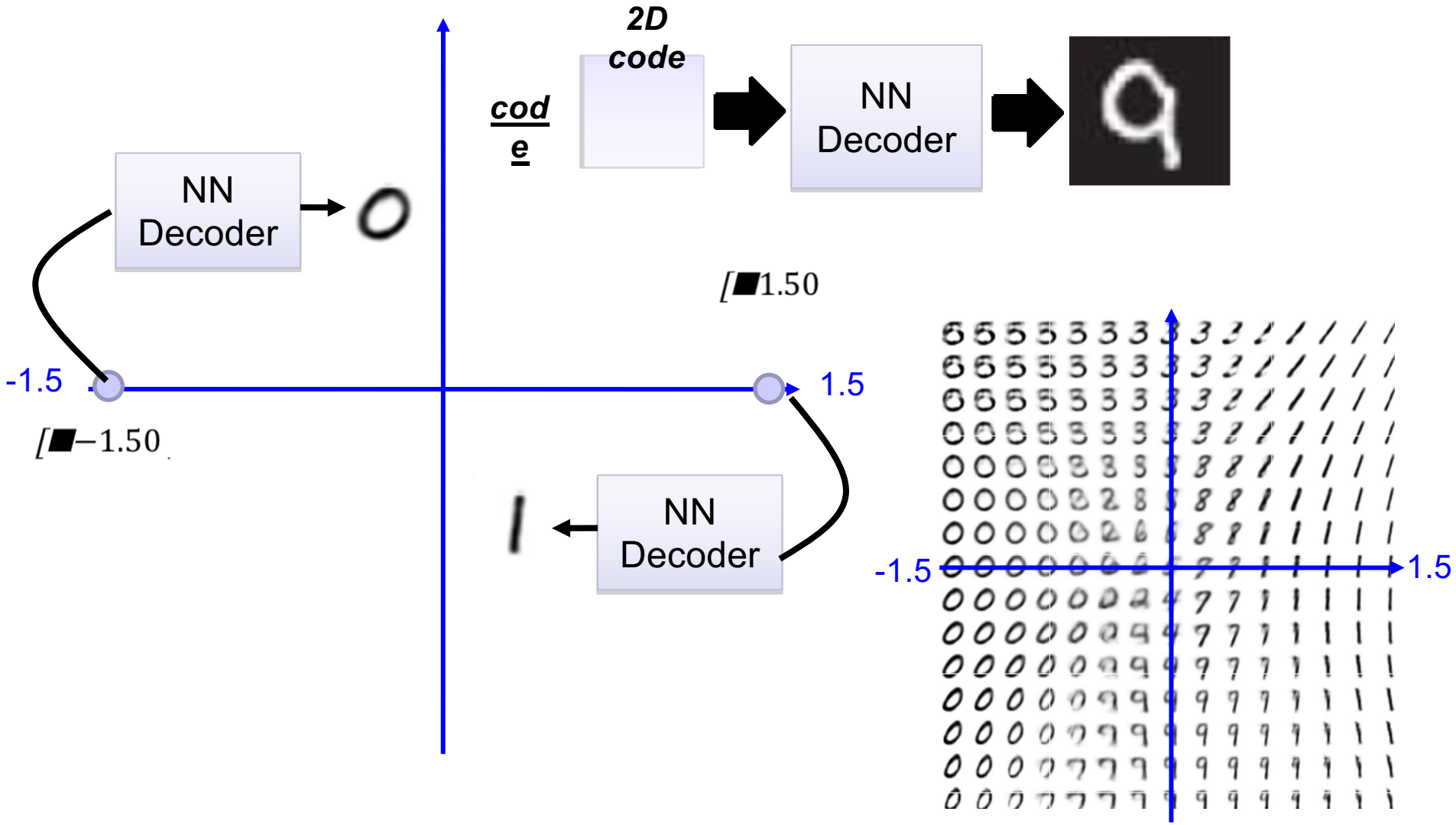
Generating new images



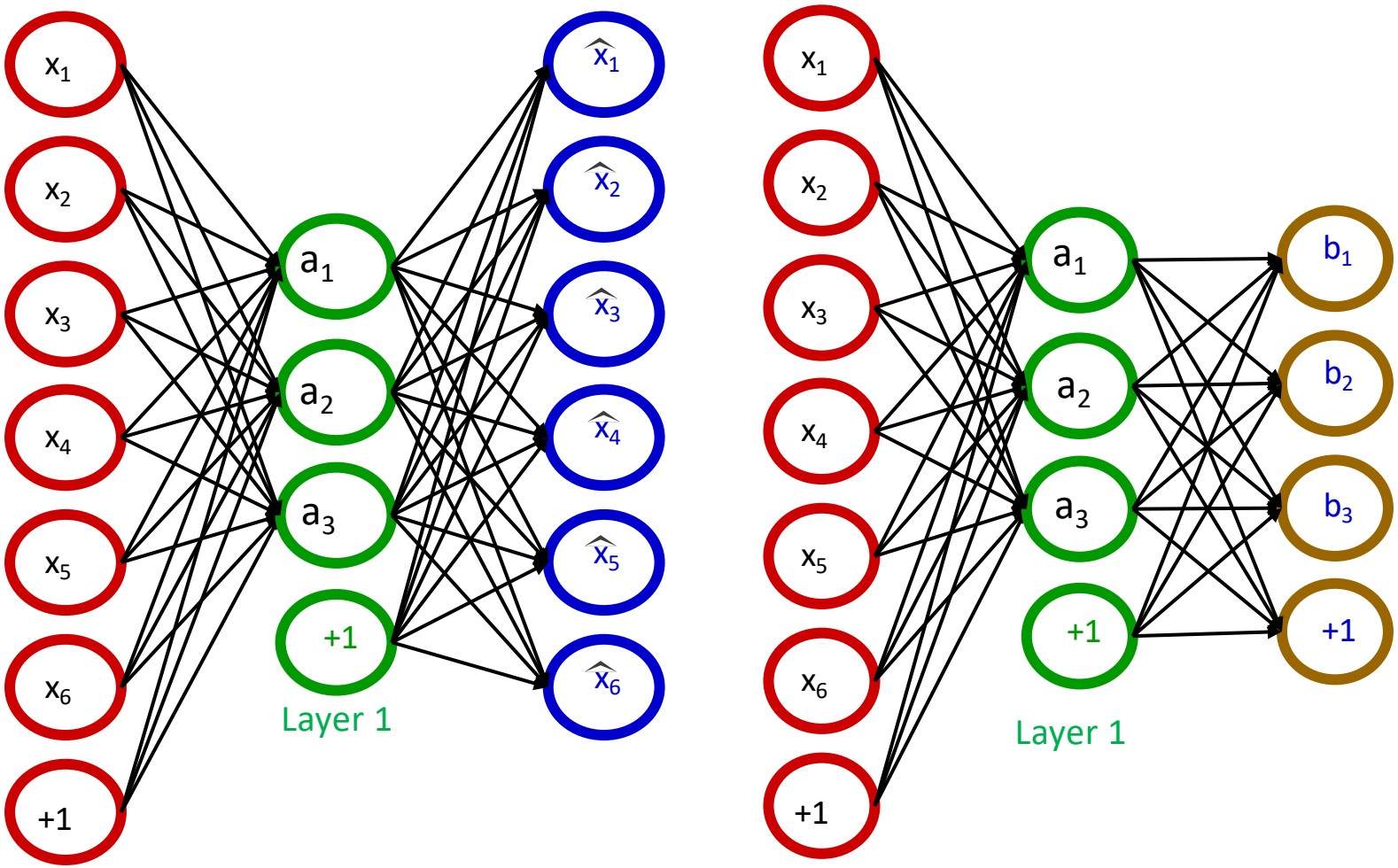
Randomly generate a vector as code



Generating new images

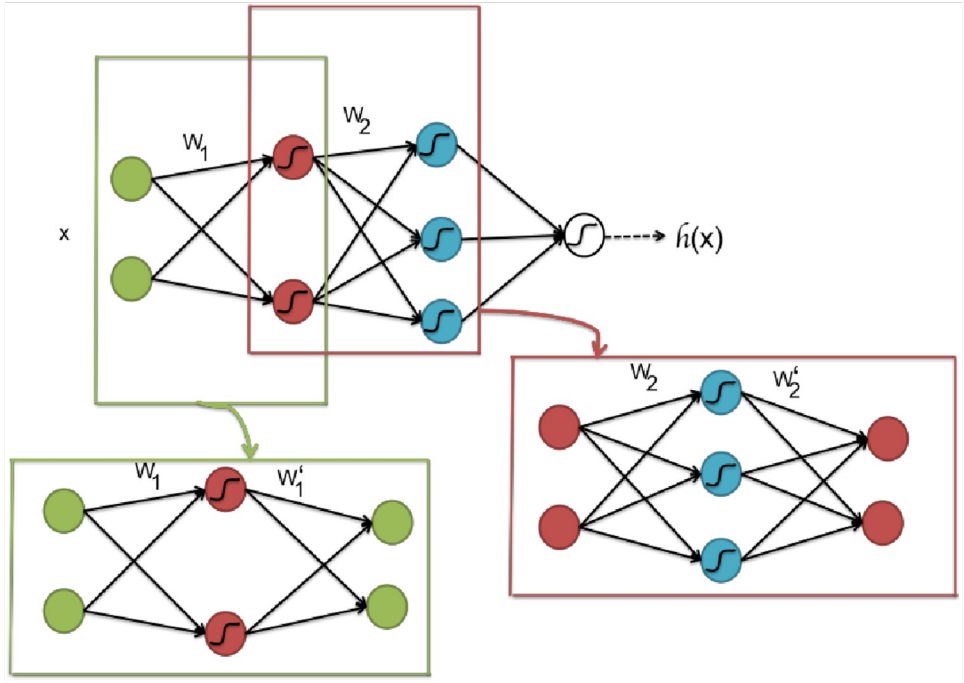


AE as Pretraining Methods



AE as Pretraining Methods

- Pretraining step
  - Train a sequence of shallow autoencoders, greedily one layer at a time, using unsupervised data
- Fine-tuning step 1
  - Train the last layer using supervised data
- Fine-tuning step 2
  - Use backpropagation to fine-tune the entire network using supervised data



Thanks

Questions?