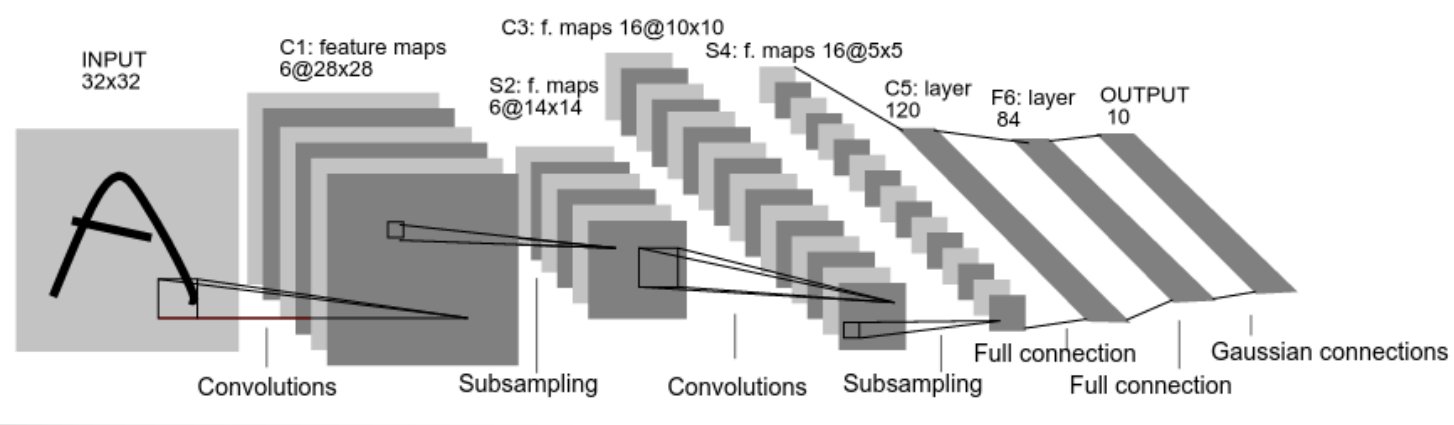
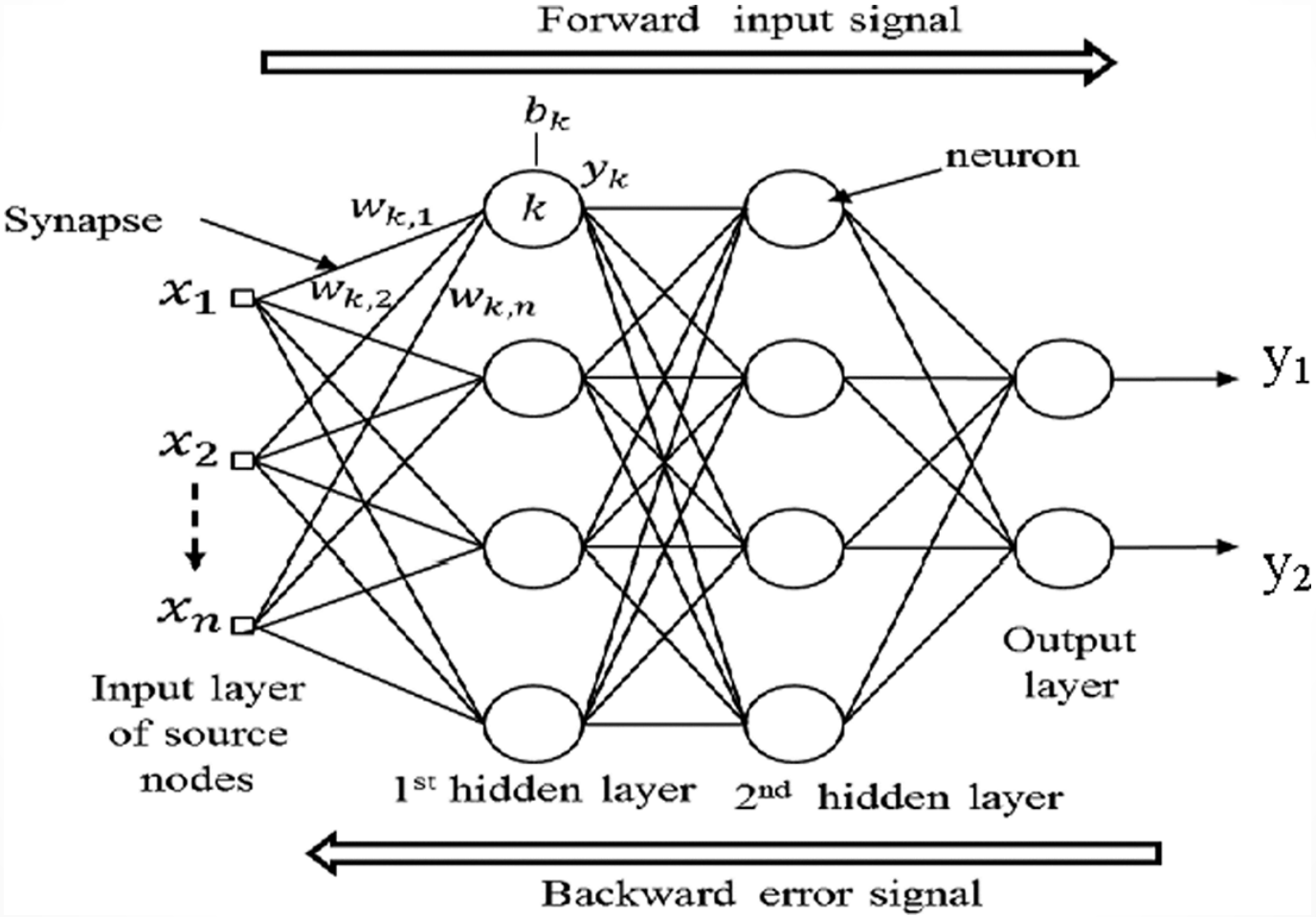
Convolutional Neural Networks



Lenet-5 (Lecun-98), Convolutional Neural Network for digits recognition

Gaurav Mittal
2012CSB1013
IIT Ropar
gauravmi@iitrpr.ac.in

ANN Recap

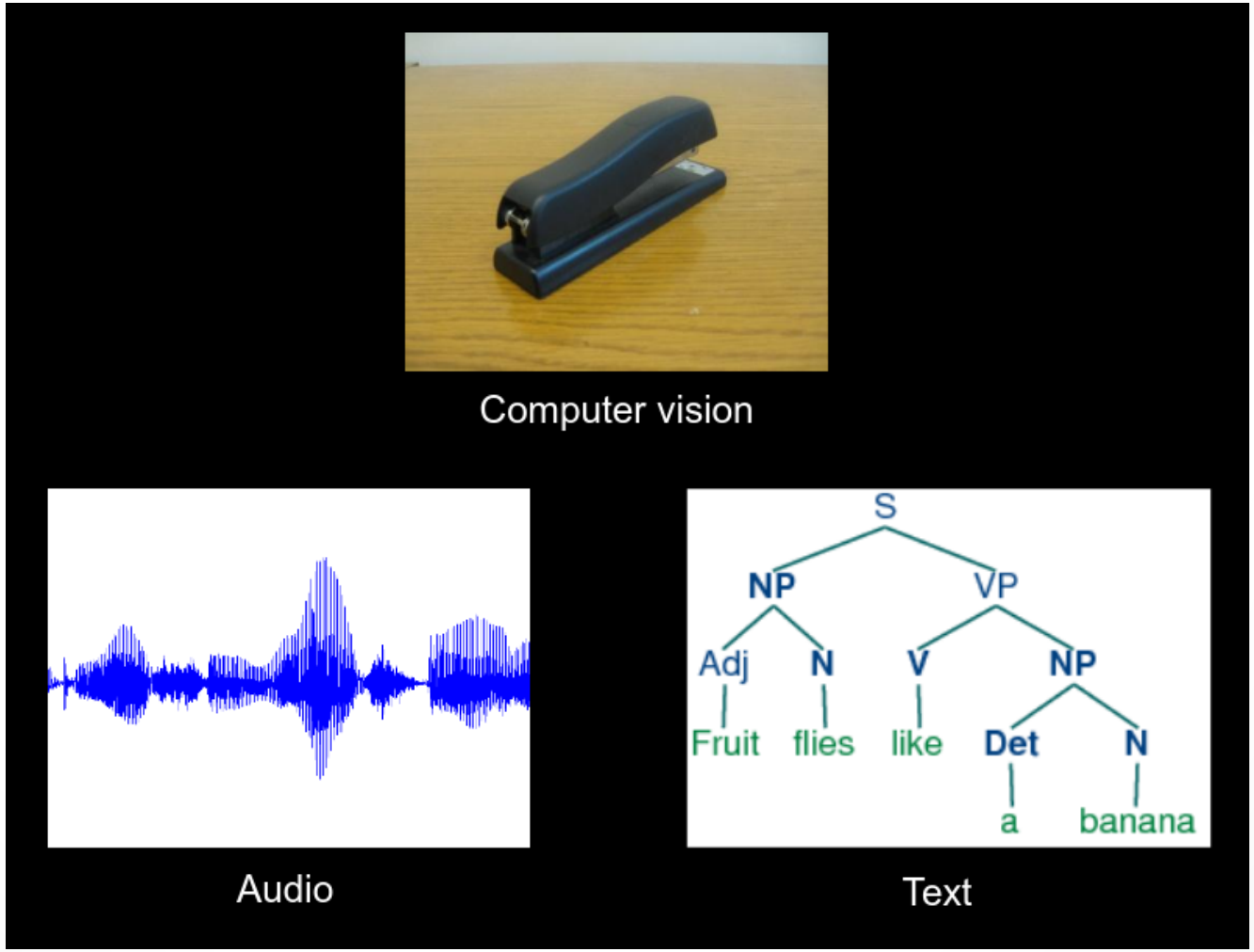

Forward input signal
b_k
y_k
neuron
Synapse
$w_{k,1}$
k
x_1
$w_{k,2}$
$w_{k,n}$
x_2
x_n
y_1
y_2
Input layer of source nodes
1st hidden layer
2nd hidden layer
Output layer
Backward error signal

What are CNNs?

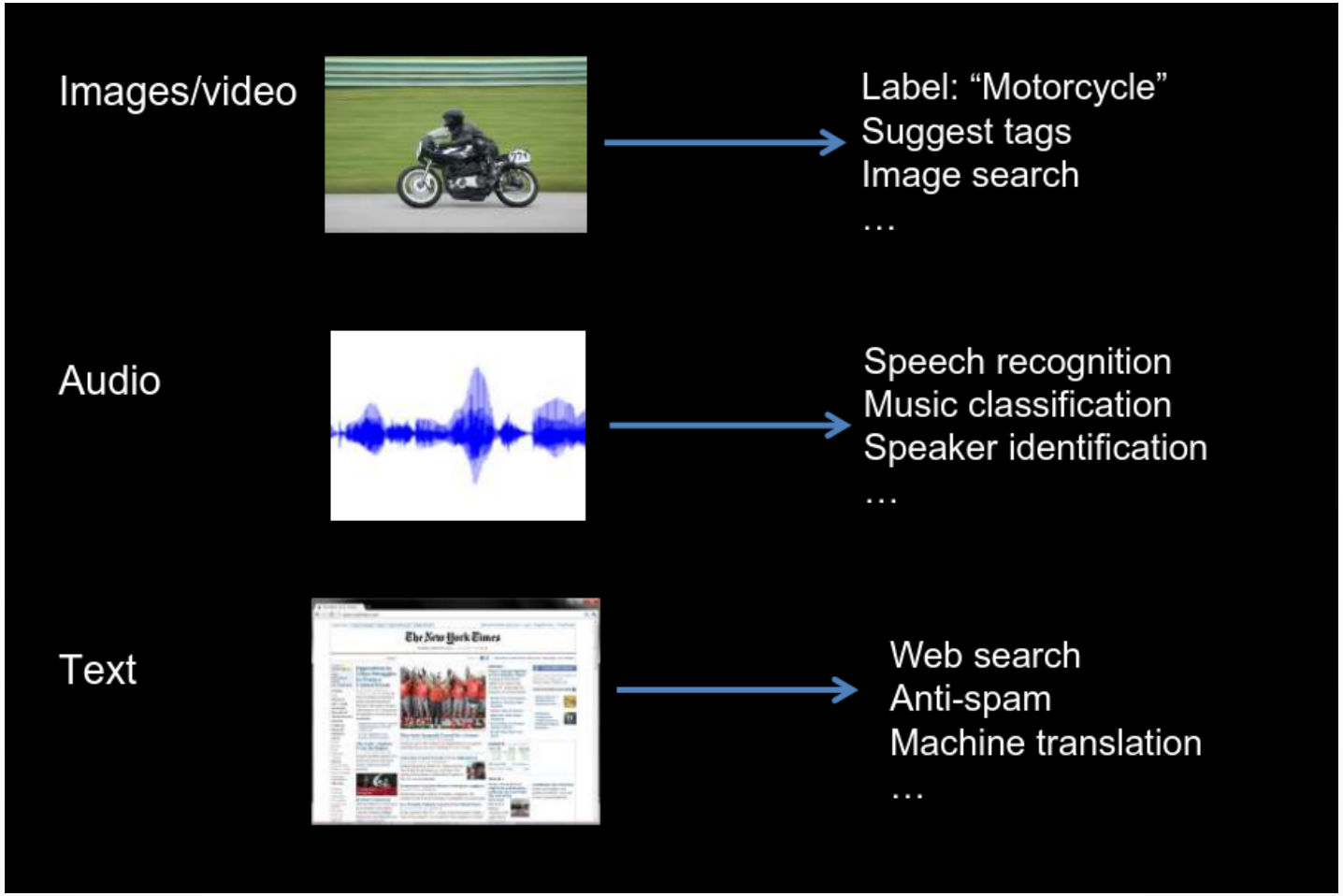
Essentially neural networks that use convolution in place of general matrix multiplication in at least one of their layers.

Motivation

Detection or Classification Tasks

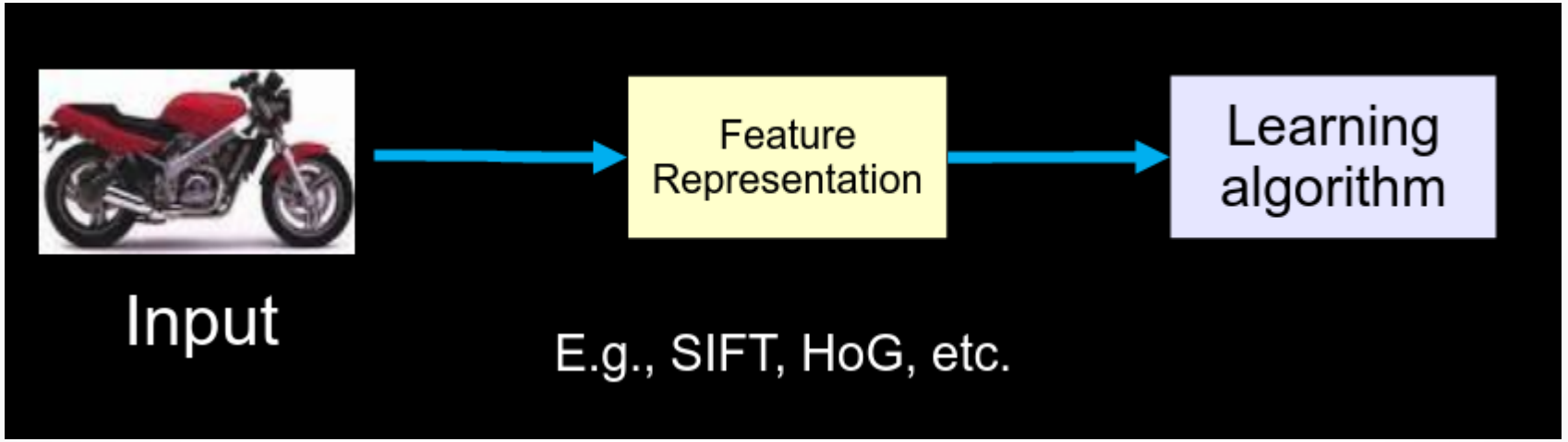


What to do with this data?

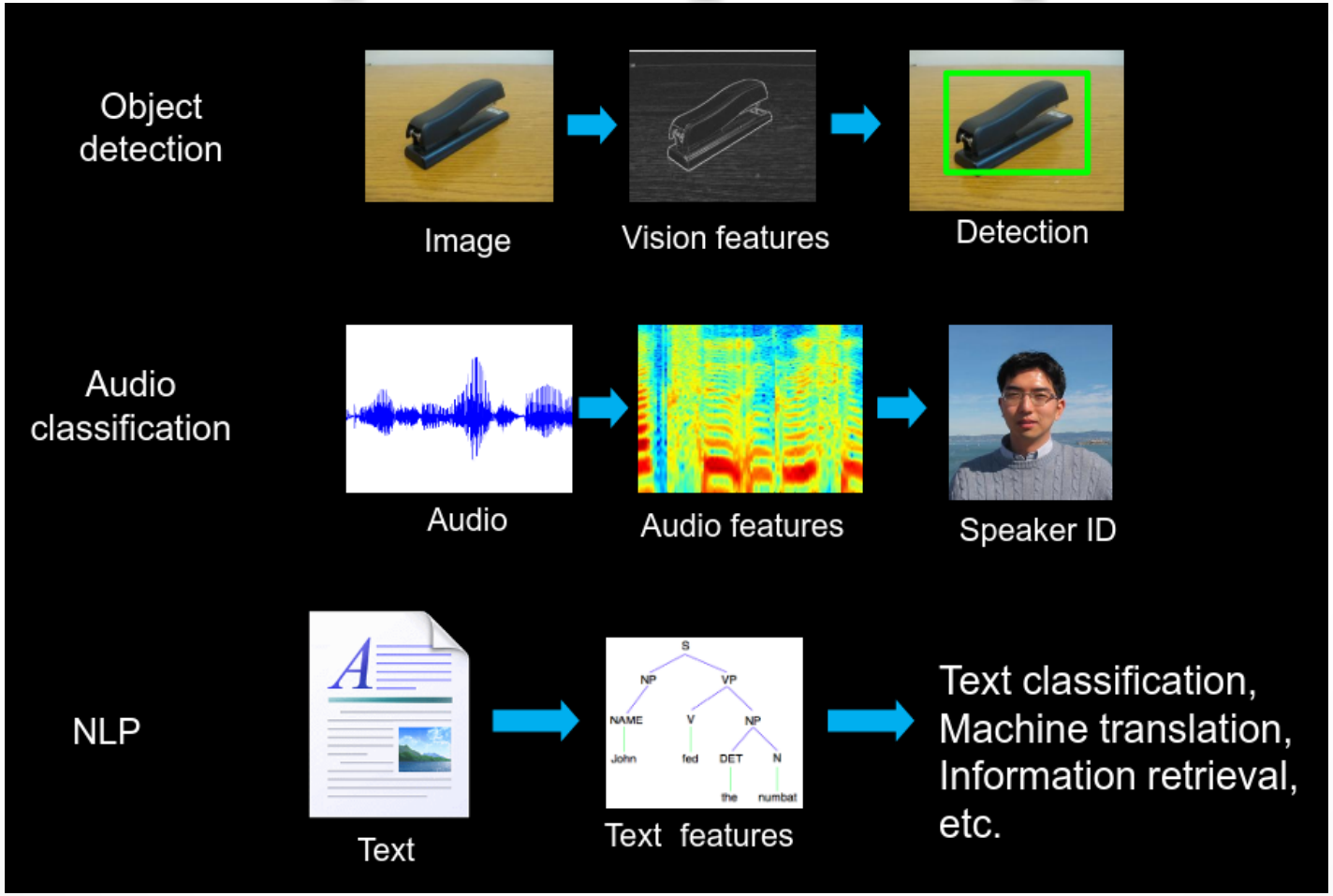


Feature Representations

Feature Representations

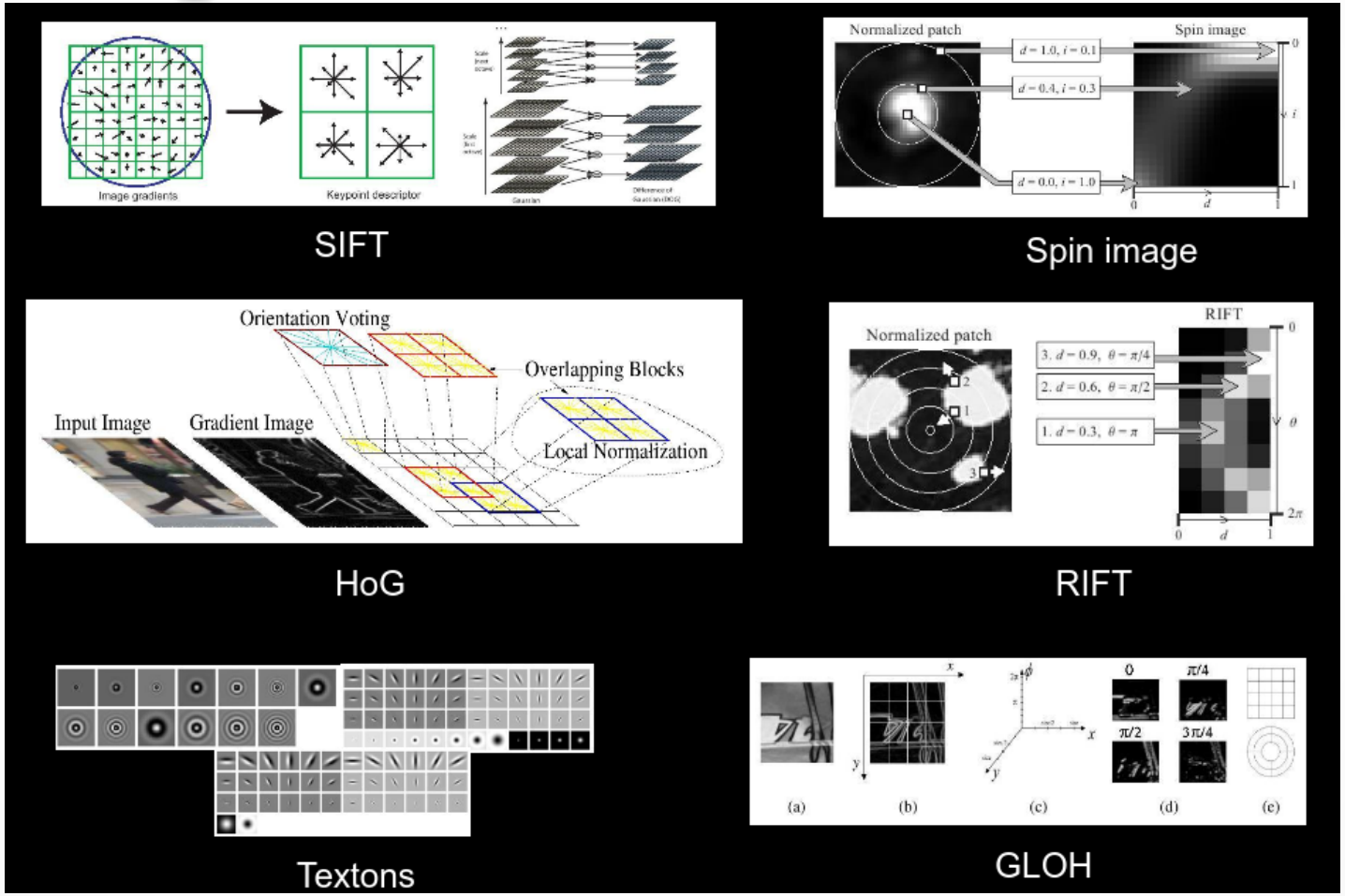


How is computer perception done?

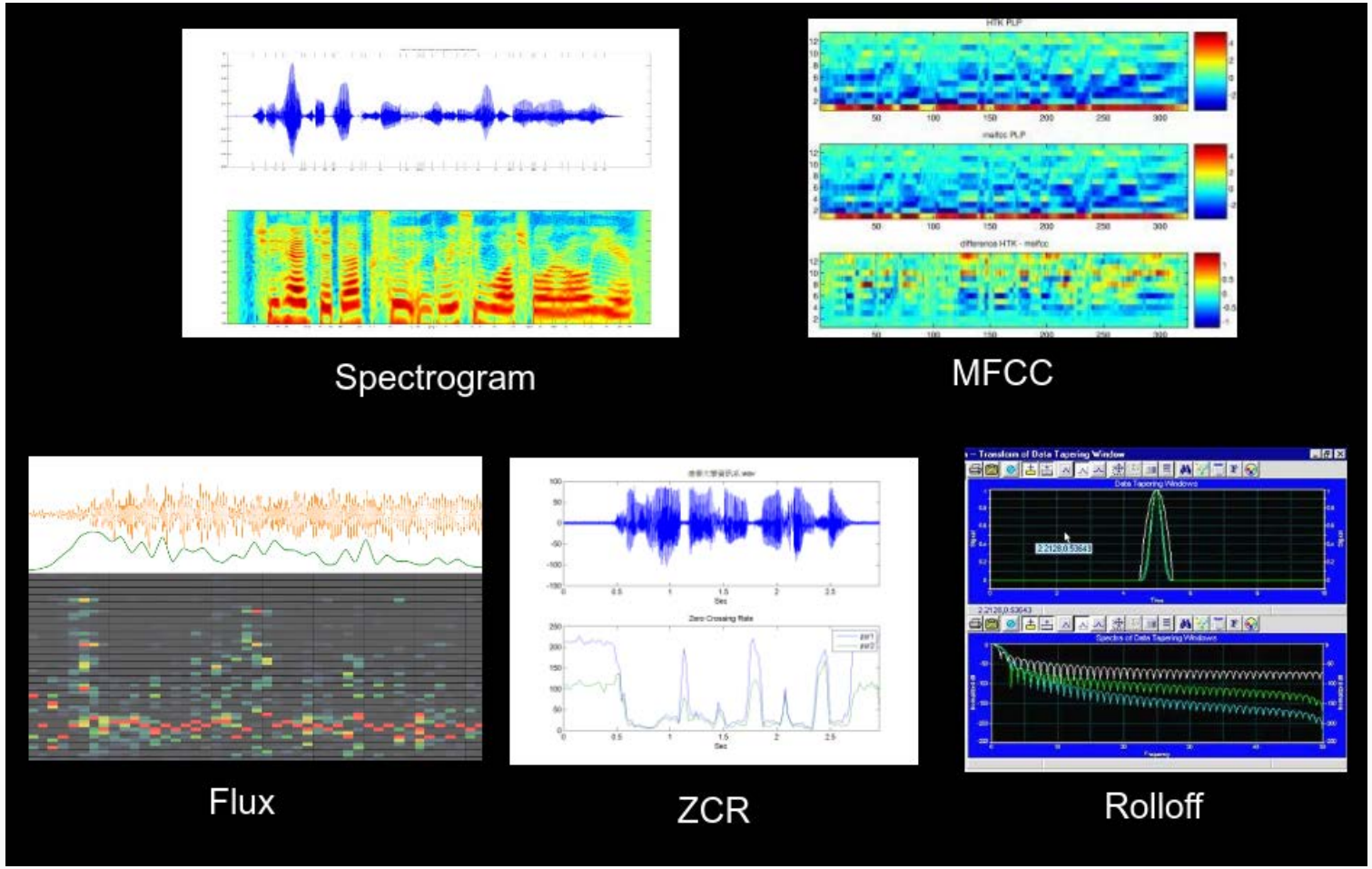


Feature Representations???

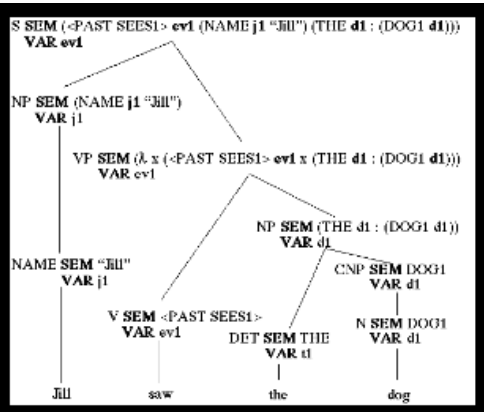
Computer Vision Features



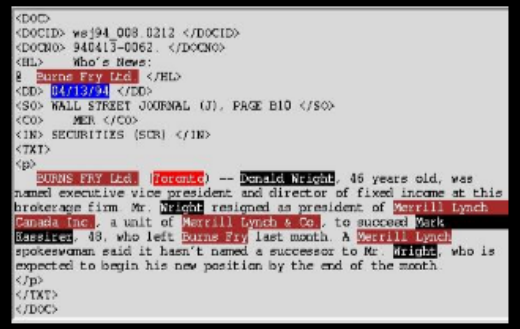
Audio Features



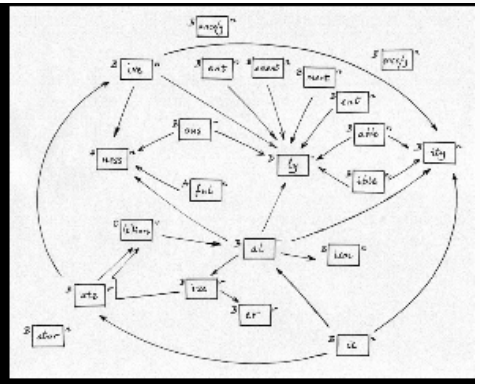
NLP Features



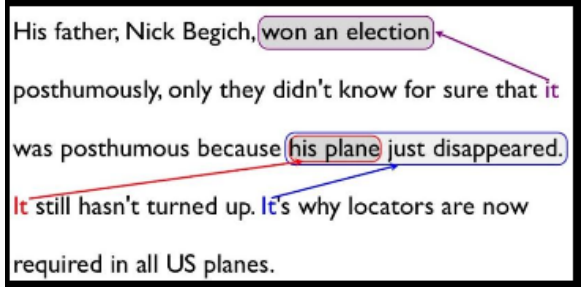
Parser features



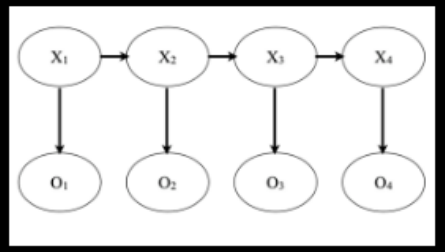
NER/SRL



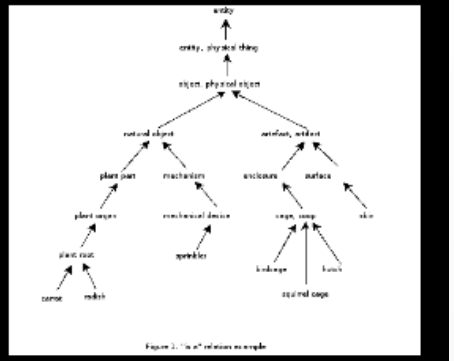
Stemming



Anaphora



POS tagging

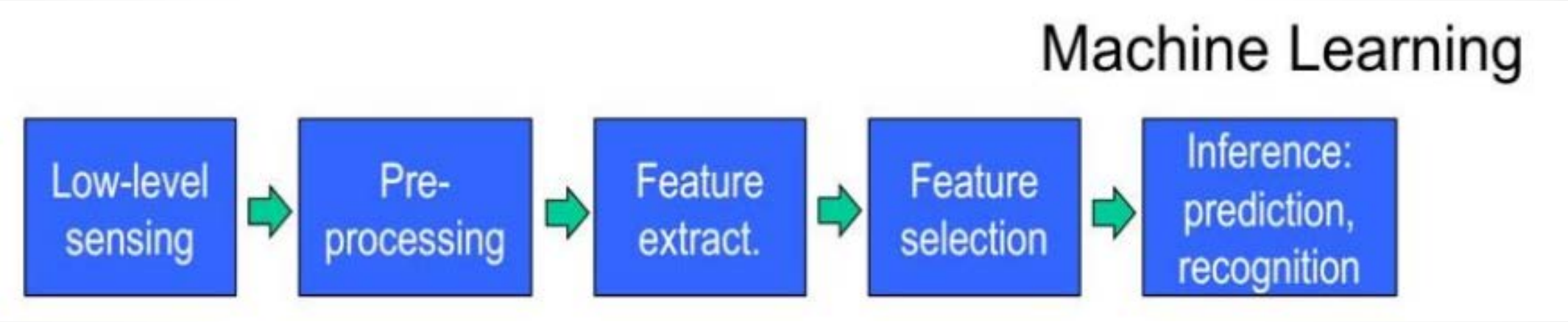


WordNet features

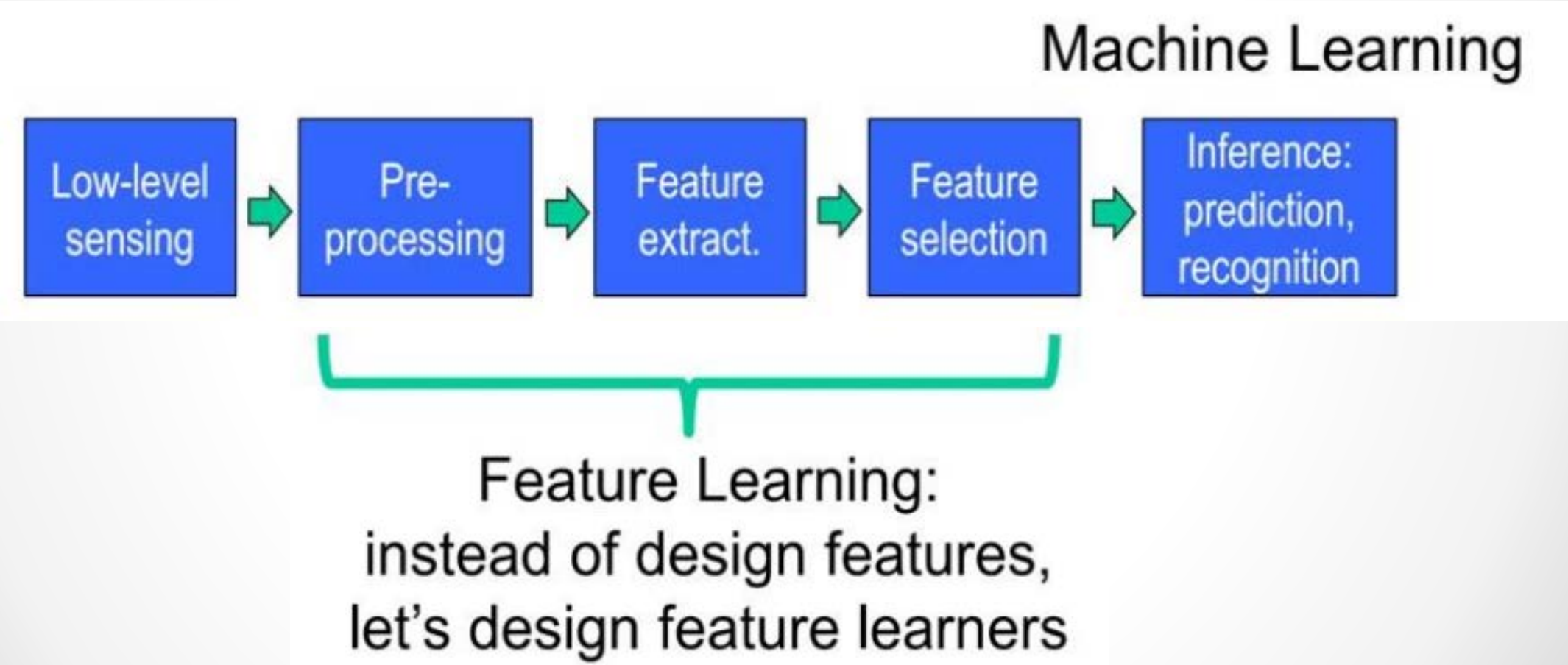
Certainly, coming up with features is difficult, time-consuming and requires expert knowledge.

A lot of time is spend tuning the features which are often hand-crafted!

Feature Representations



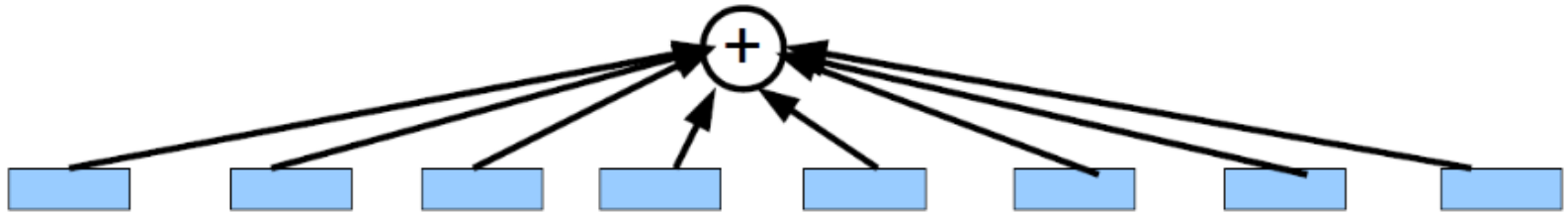
Feature Representations



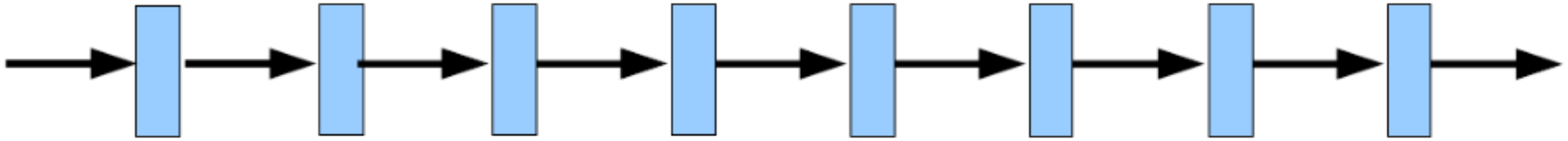
Learning non-linear functions

Given a dictionary of simple non-linear functions: $g_1, \ldots, g_n$

- Proposal 1: linear combination $f(x) \approx \sum_j g_j$



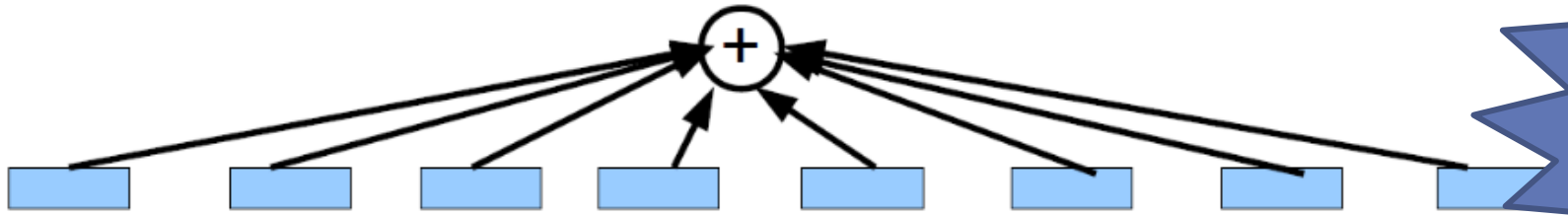
- Proposal 2: composition $f(x) \approx g_1(g_2(\ldots g_n(x)\ldots))$



Learning non-linear functions

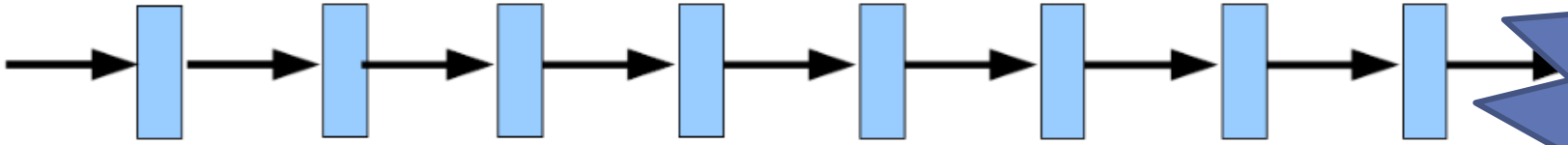
Given a dictionary of simple non-linear functions: $g_1, \ldots, g_n$

- Proposal 1: linear combination $f(x) \approx \sum_j g_j$



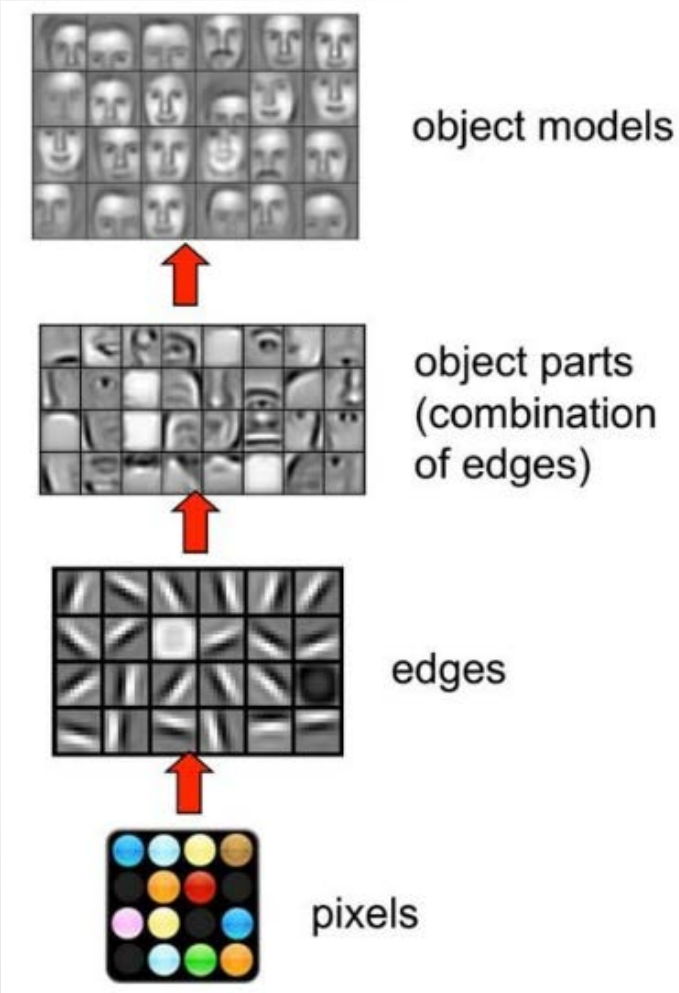
Shallow

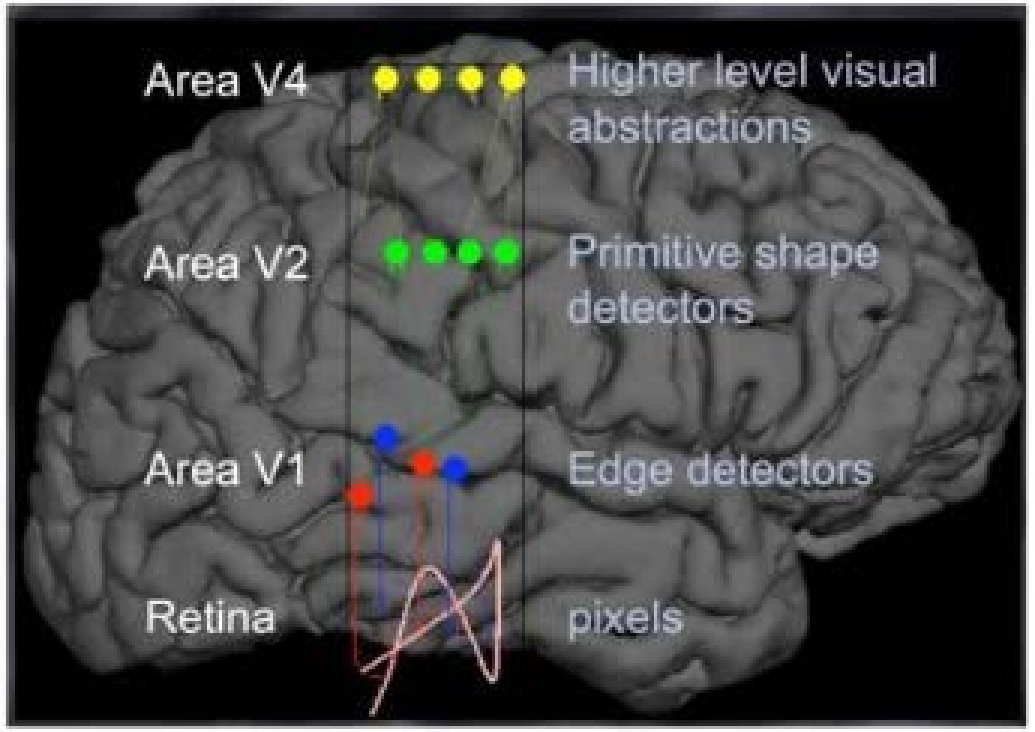
- Proposal 2: composition $f(x) \approx g_1(g_2(\ldots g_n(x)\ldots))$



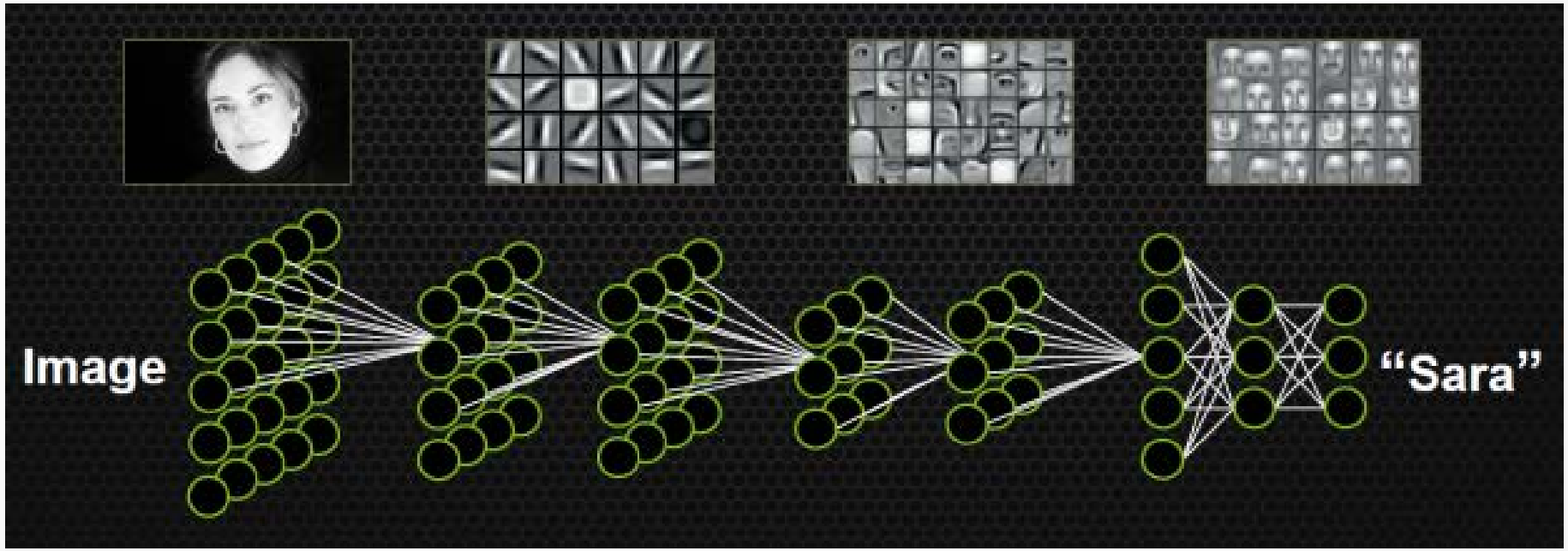
Deep

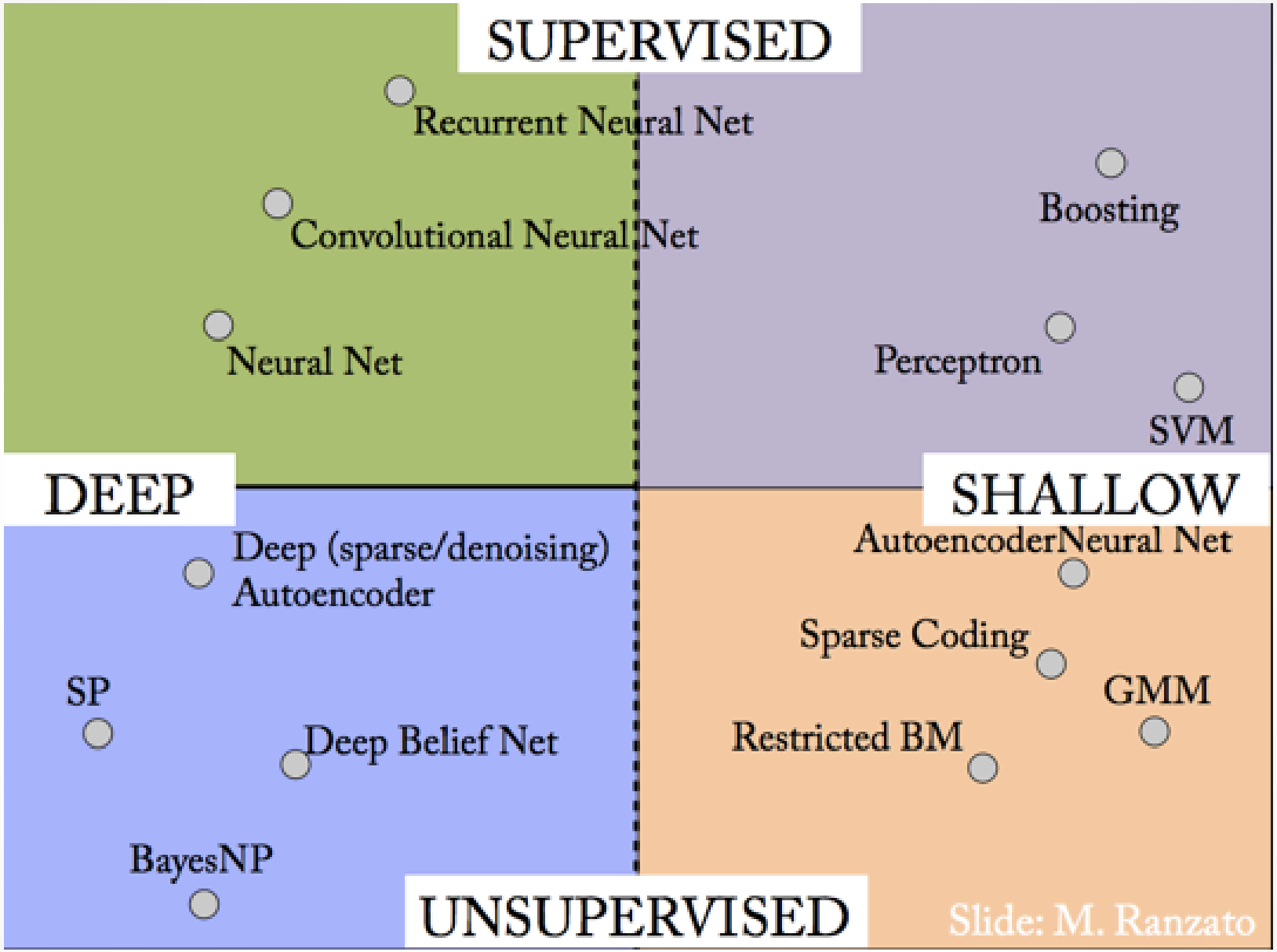
Biologically Inspired!

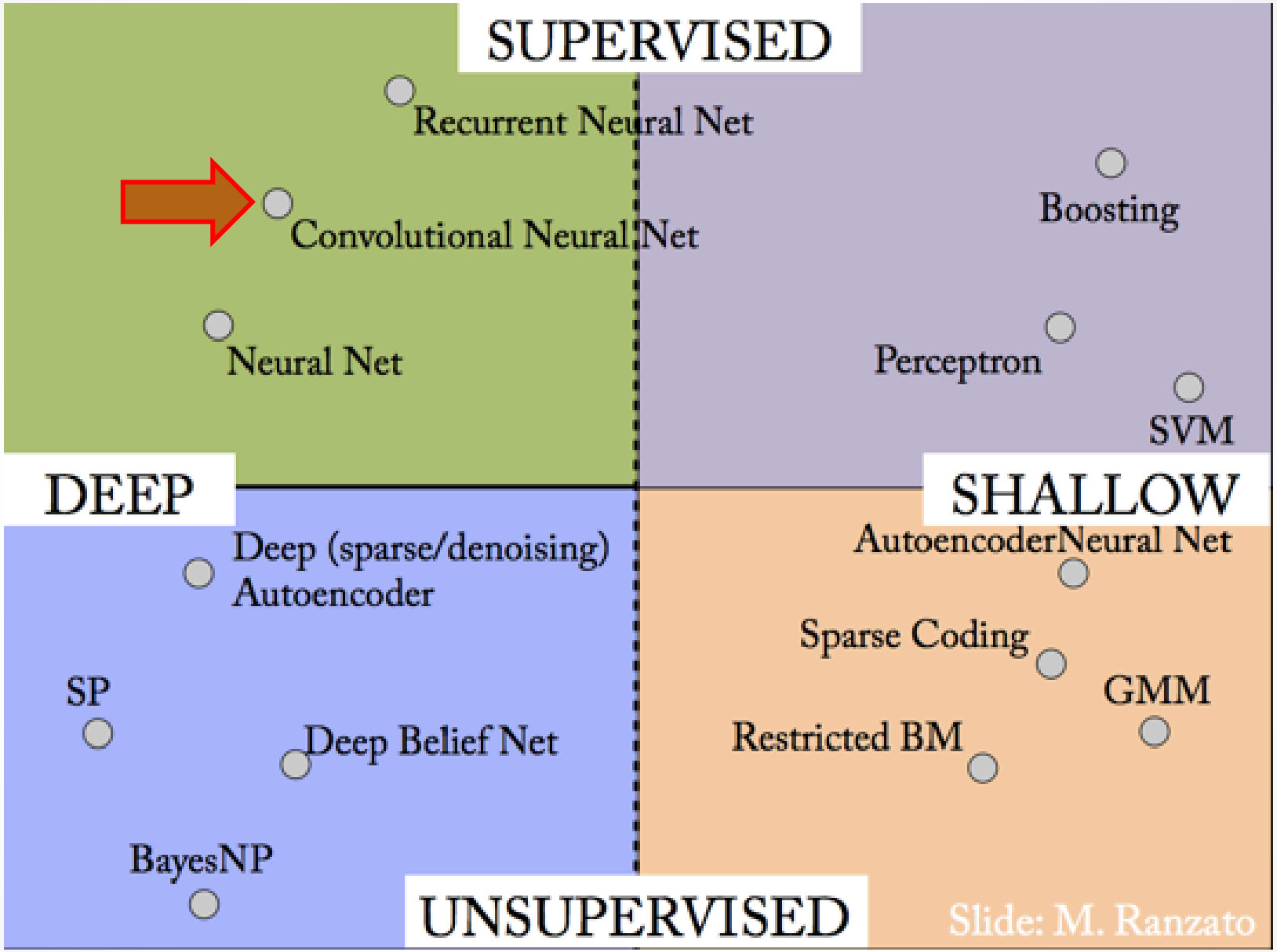




Features Learned by Deep Training




SUPERVISED
Recurrent Neural Net
Convolutional Neural Net
Neural Net
Boosting
Perceptron
SVM
DEEP
SHALLOW
Deep (sparse/denoising) Autoencoder
SP
Deep Belief Net
BayesNP
AutoencoderNeural Net
Sparse Coding
GMM
Restricted BM
UNSUPERVISED
Slide: M. Ranzato


SUPERVISED
Recurrent Neural Net
Convolutional Neural Net
Neural Net
Boosting
Perceptron
SVM
DEEP
SHALLOW
Deep (sparse/denoising) Autoencoder
AutoencoderNeural Net
Sparse Coding
GMM
SP
Deep Belief Net
Restricted BM
BayesNP
UNSUPERVISED
Slide: M. Ranzato


BRACE YOURSELF
CNN IS COMING!!

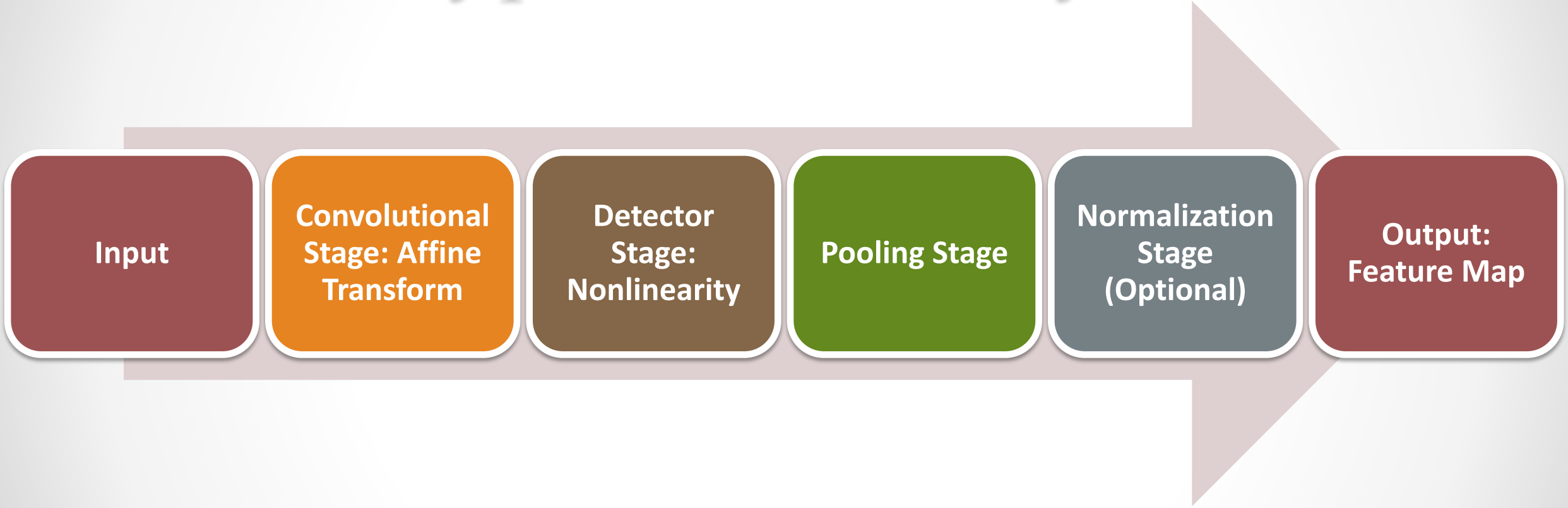
Distinguished Features

Locally Receptive Fields

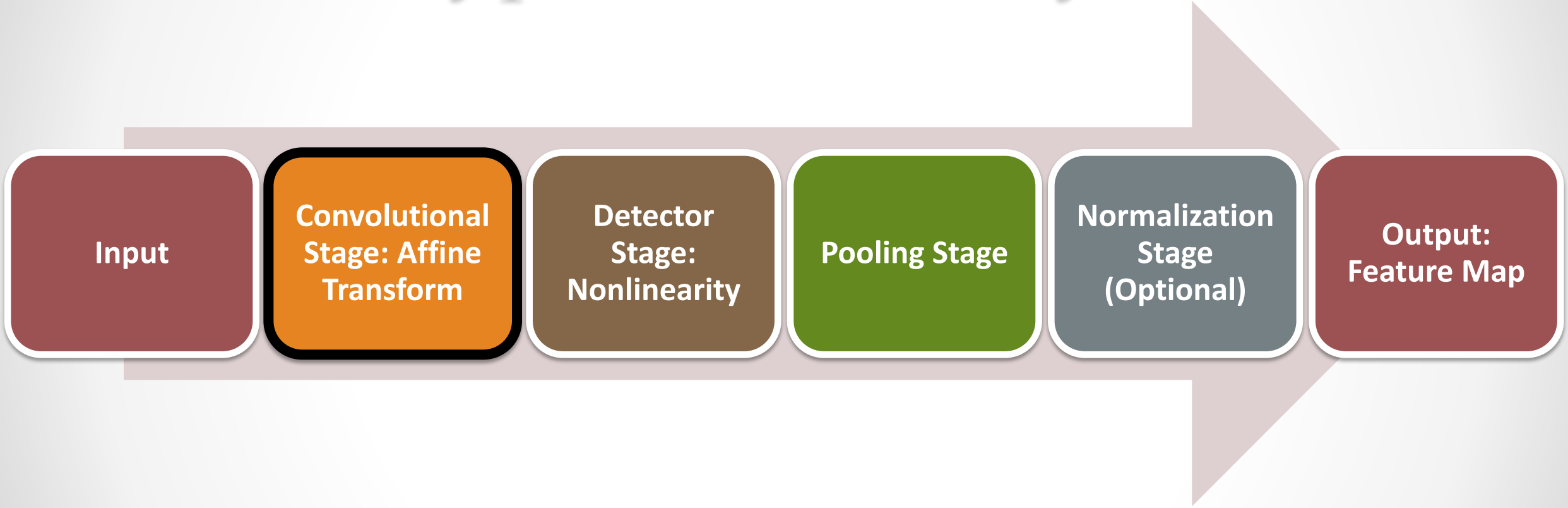
Shared Weights

Spatial or Temporal Sub-sampling

Typical CNN Layer



Typical CNN Layer



Convolution

- The convolution of f and g, written as f∗g, is defined as the integral of the product of the two functions after one is reversed and shifted:

$$s(t) = \int x(a)w(t-a)da \qquad s(t) = (x * w)(t)$$

- Convolution is commutative.
- Can be viewed as a weighted average operation at every moment (for this w need to be a valid probability density function)
- Discrete Convolution (one-axis):

$$s[t] = (x * w)(t) = \sum_{a=-\infty}^{\infty} x[a]w[t-a]$$

Cross-Correlation

- For continuous functions f and g, the cross-correlation is defined as:

$$(f \star g)(\tau) \ \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f^*(t)\ g(t+\tau)\,dt,$$

  where f* denotes the complex conjugate of f and τ is the lag
- Again, cross-correlation is commutative
- For discrete functions, it is defined as:

$$(f \star g)[n] \ \stackrel{\text{def}}{=} \sum_{m=-\infty}^{\infty} f^*[m]\ g[m+n]$$

Convolution and Cross-Correlation in Images

For a 2-D image **H** and a 2-D kernel **F**,

- Convolution Operator: $G = H \star F$

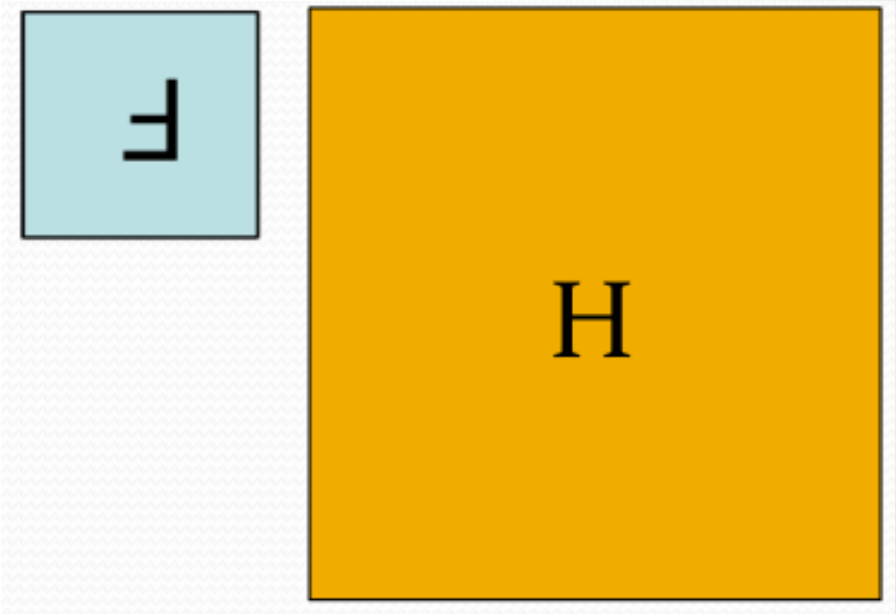
$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i-u, j-v]$$

- Correlation Operator: $G = H \otimes F$

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i+u, j+v]$$

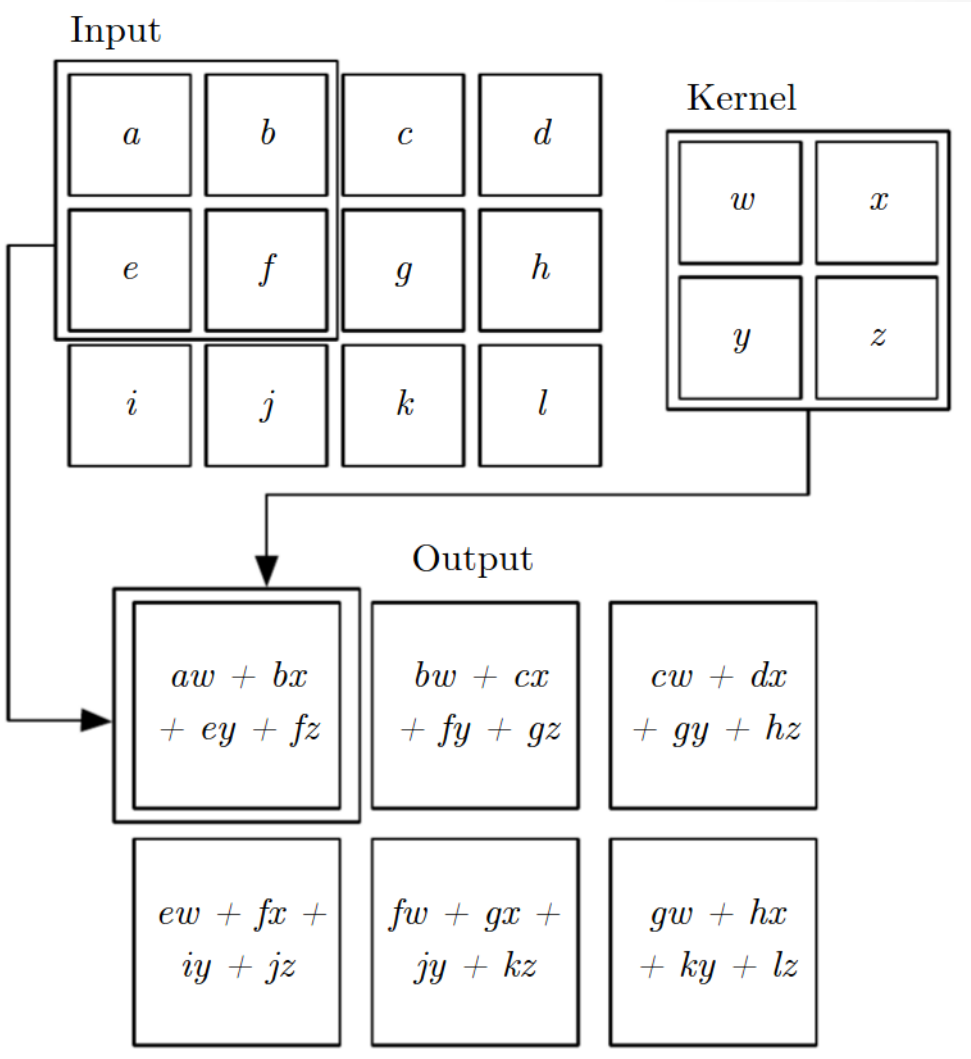
How do they differ?

- Convolution is equivalent to flipping the filter in both dimensions (bottom to top, right to left) and applying cross-correlation
- For symmetric kernels, both result in the same output.
- Many machine learning libraries implement cross-correlation but call it convolution!

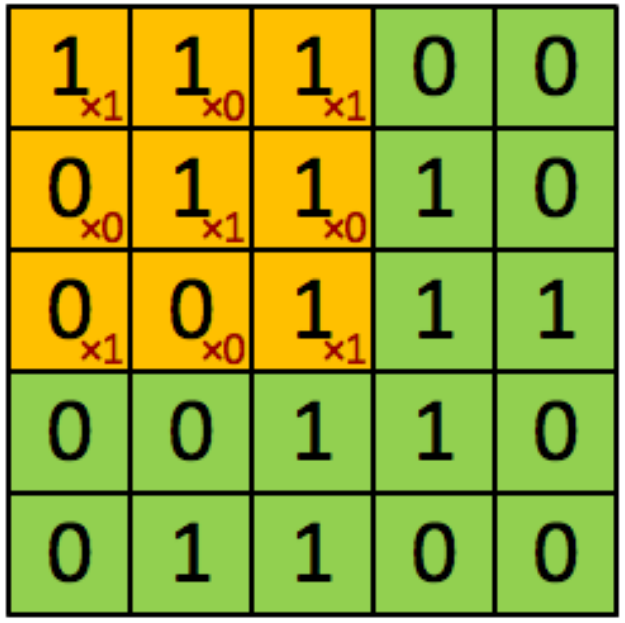


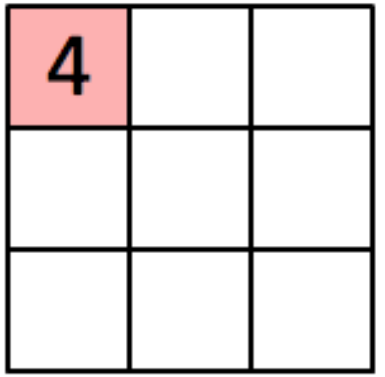
2-D Convolution (without kernel flipping)

Example of 'valid' 2-D convolution (without kernel flipping) where a 3x4 matrix convolved with a 2x2 kernel to output a 2x3 matrix



2-D Convolution in Action!





http://i.stack.imgur.com/I7DBr.gif

Variants

Full

- Add zero-padding to the image enough for every pixel to be visited k times in each direction, with output size: **(m + k - 1) x (m + k - 1)**

Valid

- With no zero-padding, kernel is restricted to traverse only within the image, with output size: **(m - k + 1) x (m - k + 1)**

Same

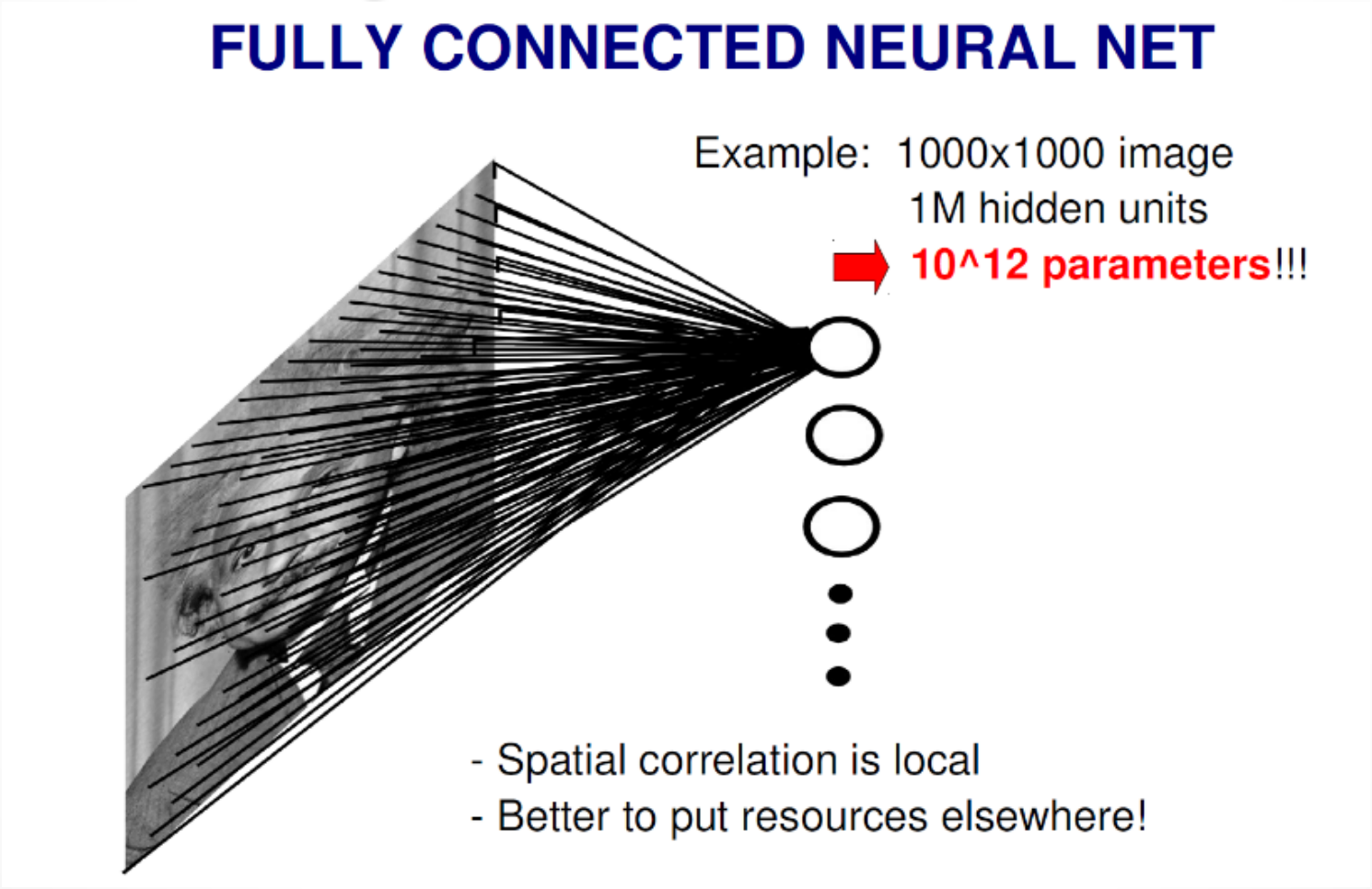
- Add zero-padding to the image to have the output of the same size as the image, i.e., m x m

Stride *s*

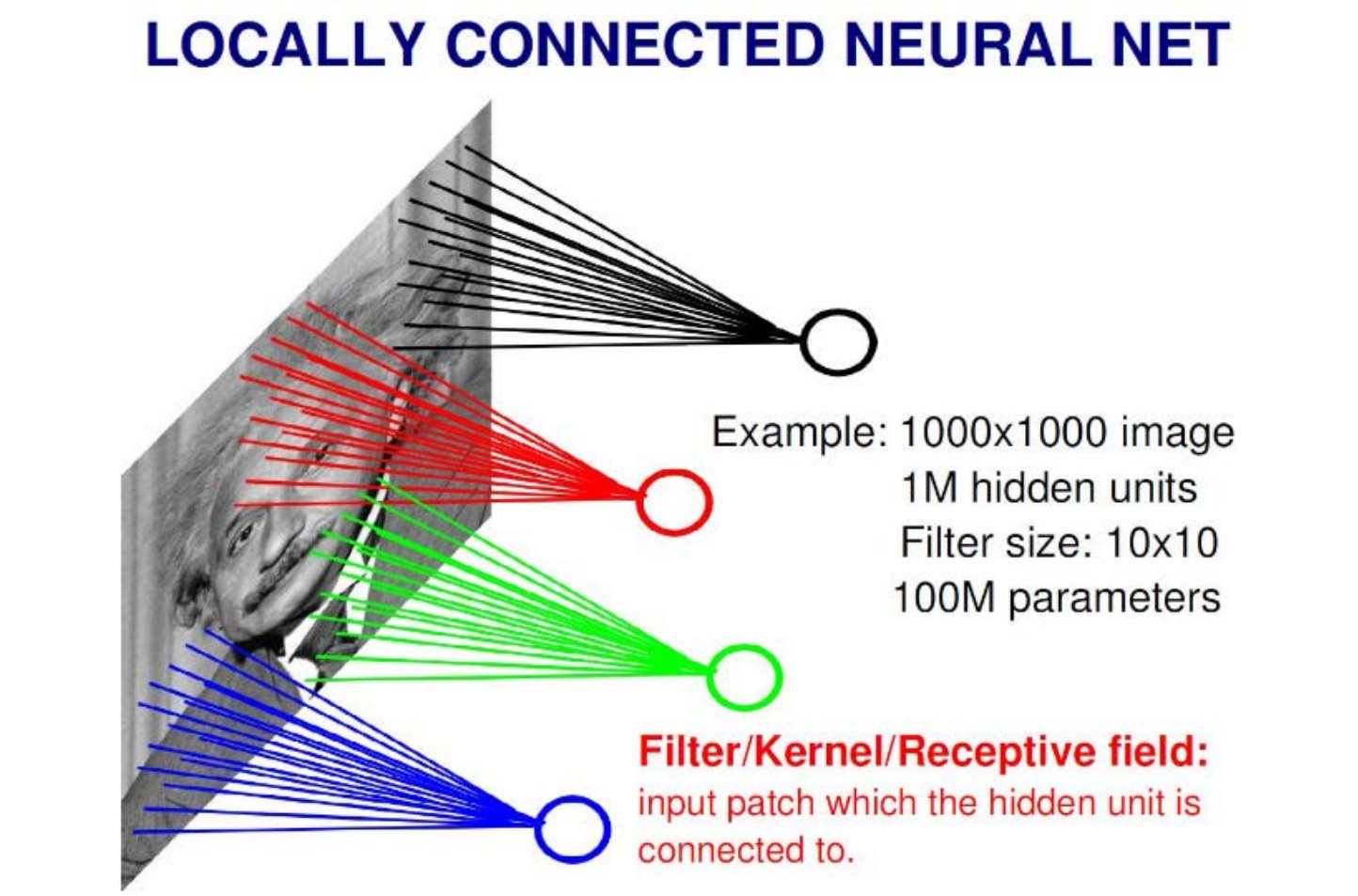
- Down-sampling the output of convolution by sampling only every s pixels in each direction.
- For instance, the output of 'valid' convolution with stride s results in an output of size $\frac{m-k+s}{s}$ **x** $\frac{m-k+s}{s}$

http://goodfeli.github.io/dlbook/contents/convnets.html

Why Convolution?



Why Convolution?



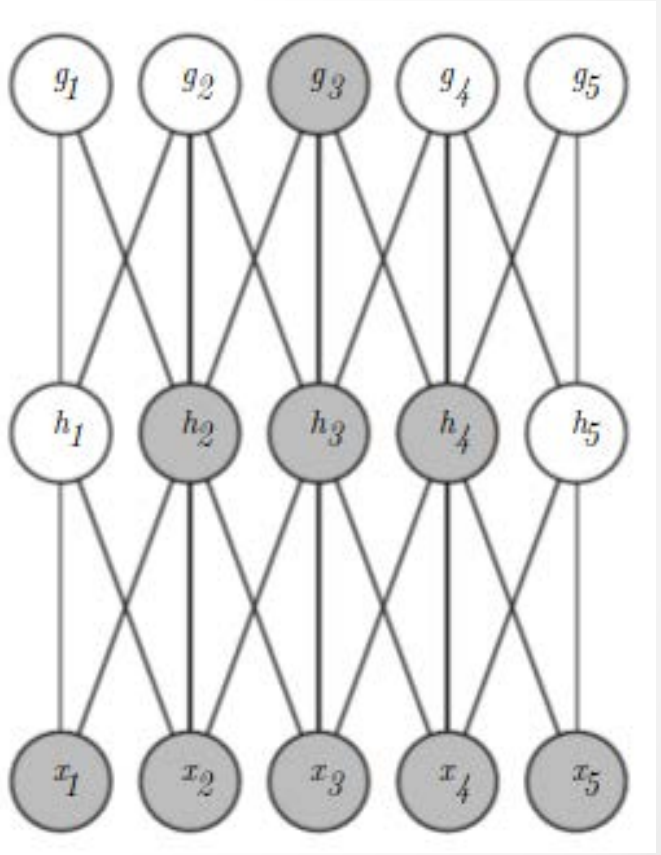
Local Receptive Field/Sparse Connectivity

- Convolution exploits the property of **spatial local-correlations** in the image by enforcing local connectivity pattern between neurons of adjacent layers

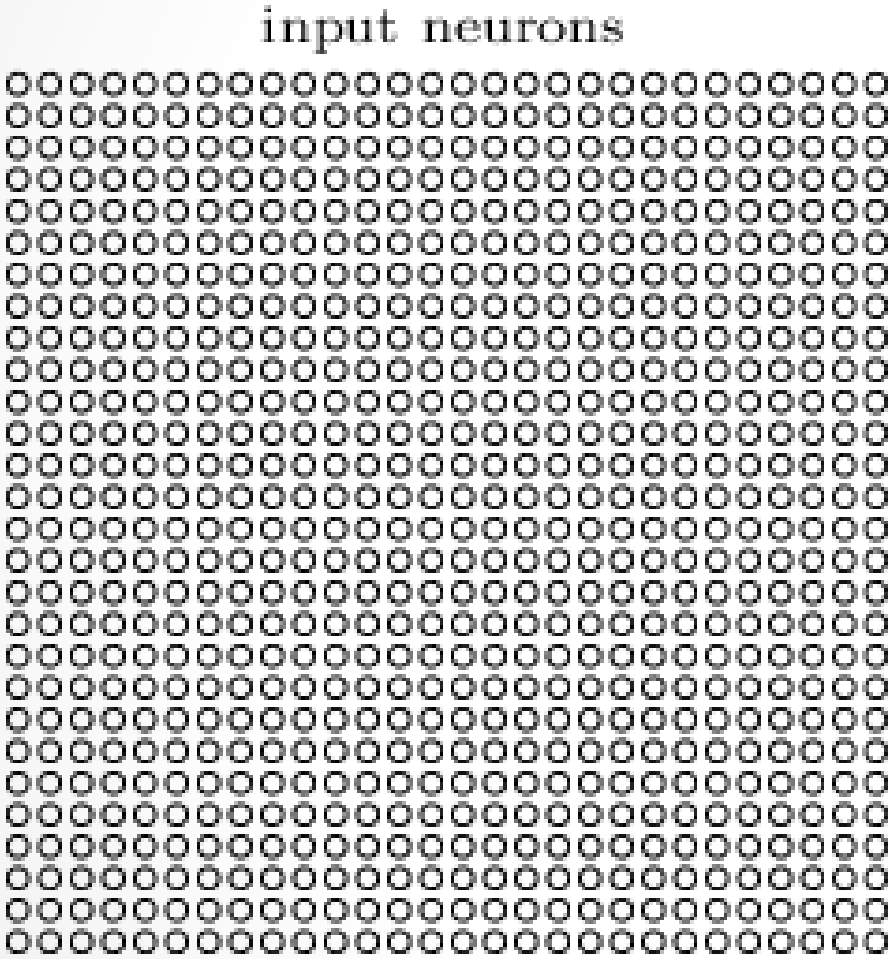
- **Drastic reduce in the number of free parameters** compared to fully connected network **reducing overfitting and more importantly, computational complexity of the network**.

Indirect Global Connectivity

- Receptive fields of units in deeper layers larger than shallow layers
- Though direct connections are very sparse, deeper layers indirectly connected to most of the input image
- Effect increases with strided convolution or pooling

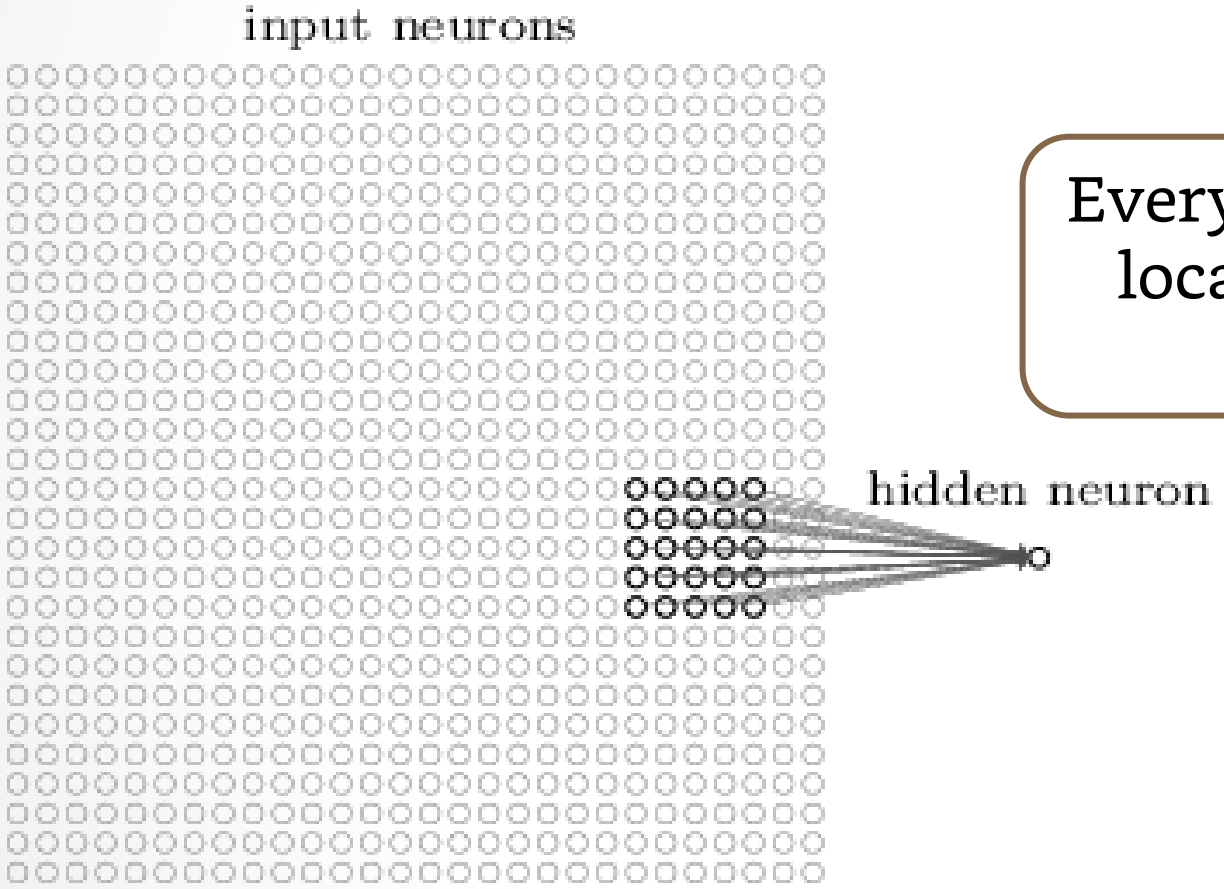


Example



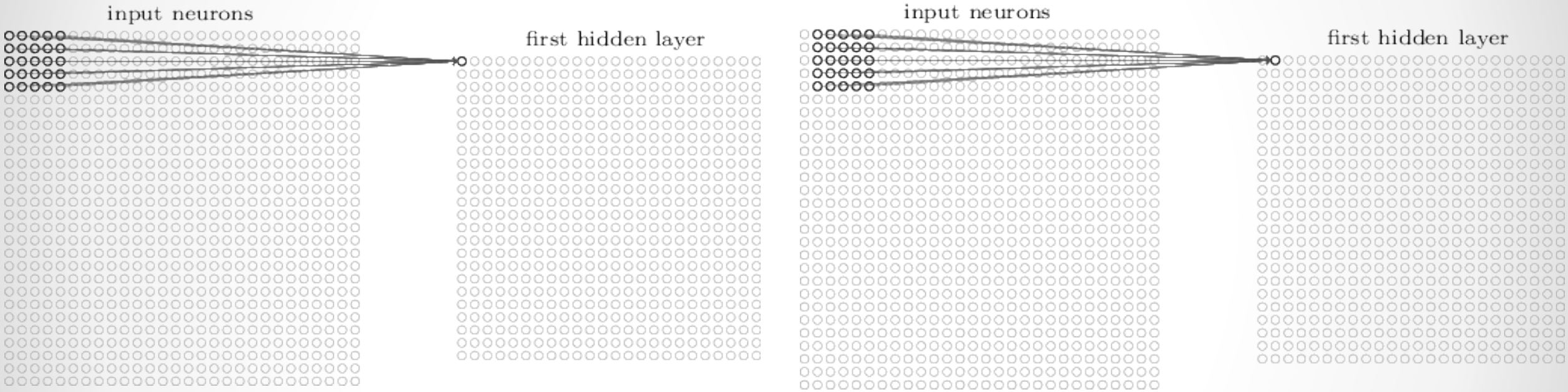
Input neurons representing a 28x28 image (such as from MNIST dataset)

Example



Every hidden layer neuron has a local receptive field of region 5x5 pixels

Example

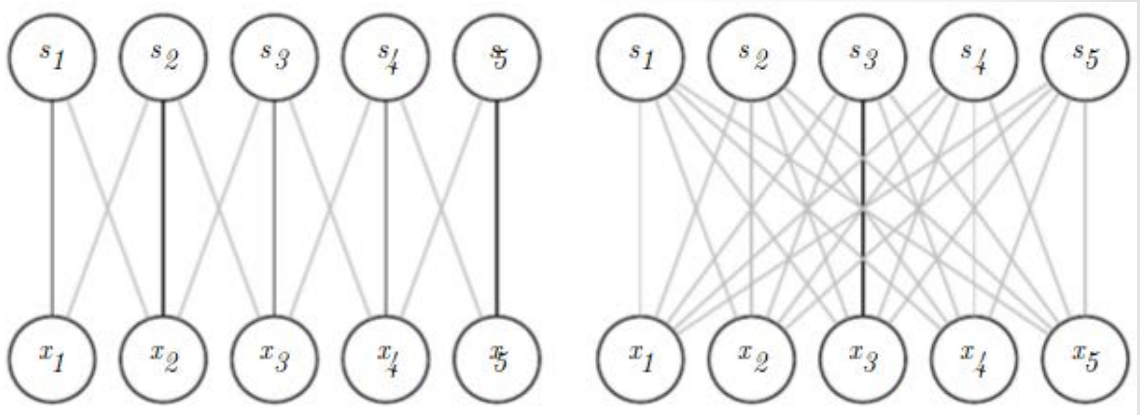


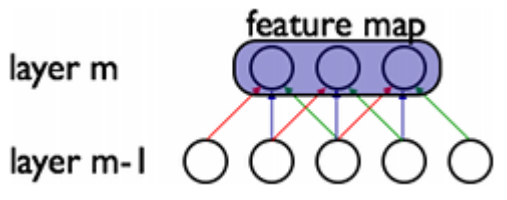
And so on, the first hidden layer is built!

(28 - 5 + 1) = 24 x 24 neurons in the hidden layer on 'valid' convolution
Size of the hidden layer can be changed using another variant of convolution

Shared Weights and Bias

- All neuron in the hidden layer share the same parameterization (weight vector and bias) forming a 'Feature Map‘
- (Shared Weights, Bias) → Kernel/Filter
- Now, the gradient of a shared weight is sum of the gradients of the parameters being shared.

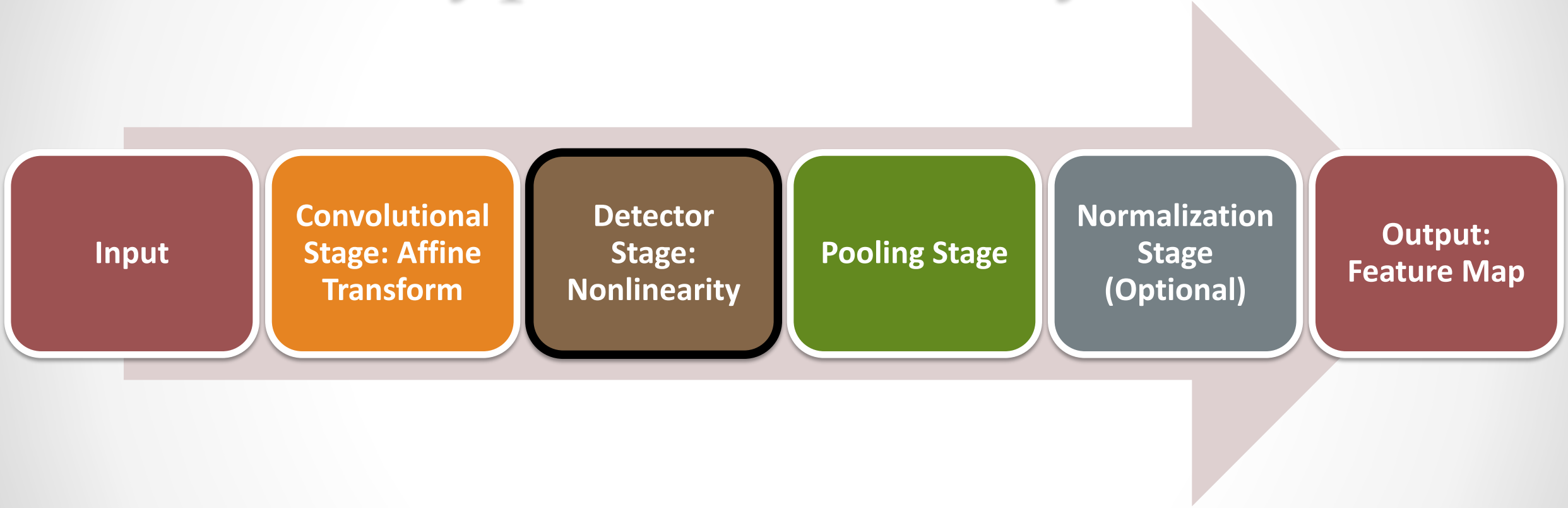




Shared Weights and Bias

- Translation Equivariance
  - Allows features to be detected regardless of their position in the visual field. (Feature is a kind of input pattern that will cause a neuron to activate, for eg. an edge)
  - All neurons in the first hidden layer detect exactly the same feature, just at different locations.
  - CNNs are well adapted to translation invariance of images: move a picture of a cat, and it's still an image of a cat!
- Further reduces the number of free parameters, achieving better generalization and computational performance.

Typical CNN Layer

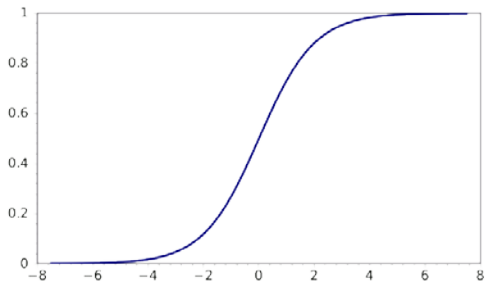


Non-Linear Activation Function

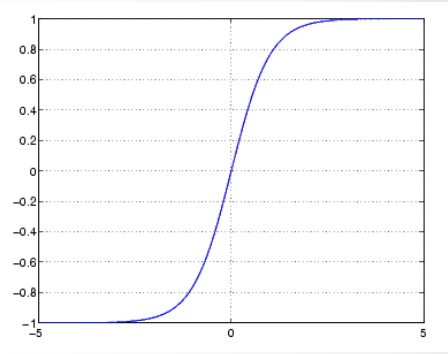
- Sigmoid: $$S(t) = \frac{1}{1 + e^{-t}}.$$
- Tanh: $$\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$
- Rectified Linear Unit (ReLU):

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

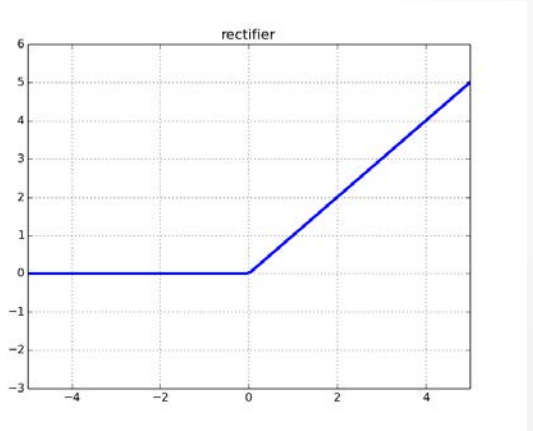
Most popular activation function for DNN as of 2015, avoids saturation issues, makes learning faster



Sigmoid



Tanh



ReLU

- Feature Map - Obtained by convolution of the image with a linear filter, adding a bias term and applying a non-linear function

- Require a number of such feature maps at each layer to capture sufficient features in the image

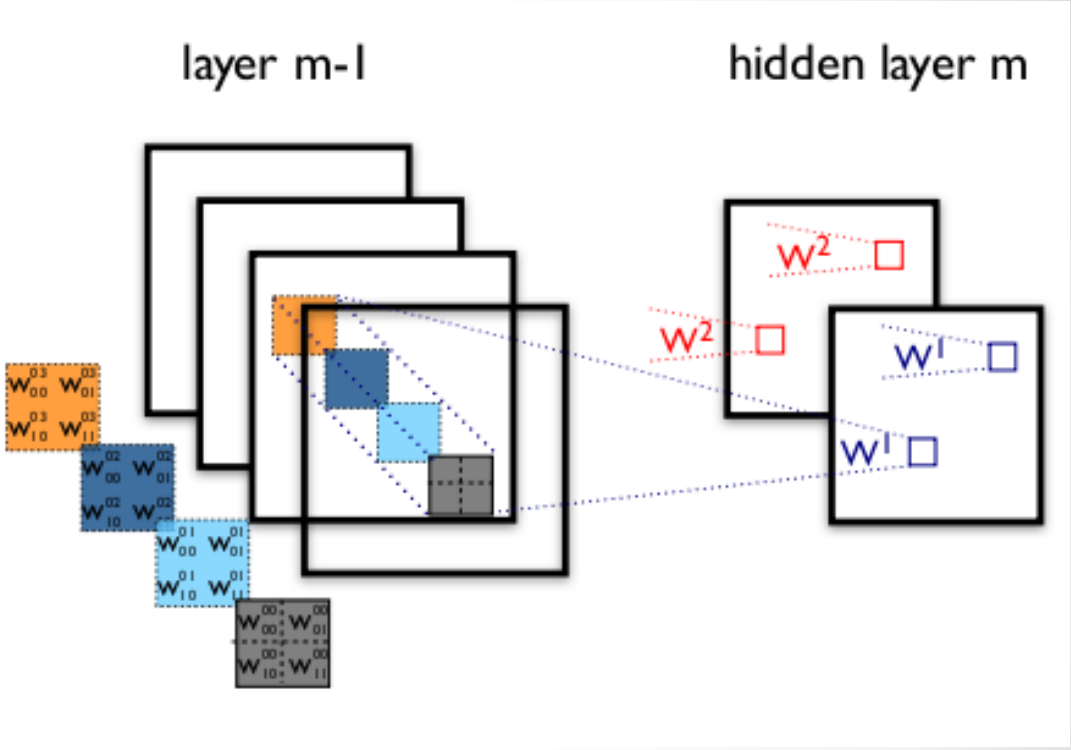
- Let k^{th}feature map at a given layer be x^k, whose filters are determined by W_k and bias b_k, then x^k with sigmoid, σ function for non-linearity and filter of size m x m is obtained as:

$$x^k{}_{ij} = \sigma\left(\left(W^k * a\right)_{\text{ij}} + b_k\right) = \sigma\left[\left(\sum_{a=0}^{m-1}\sum_{b=0}^{m-1} w_{ab} y^{k-1}{}_{(i+a)(j+b)}\right)_{ij} + b_k\right]$$

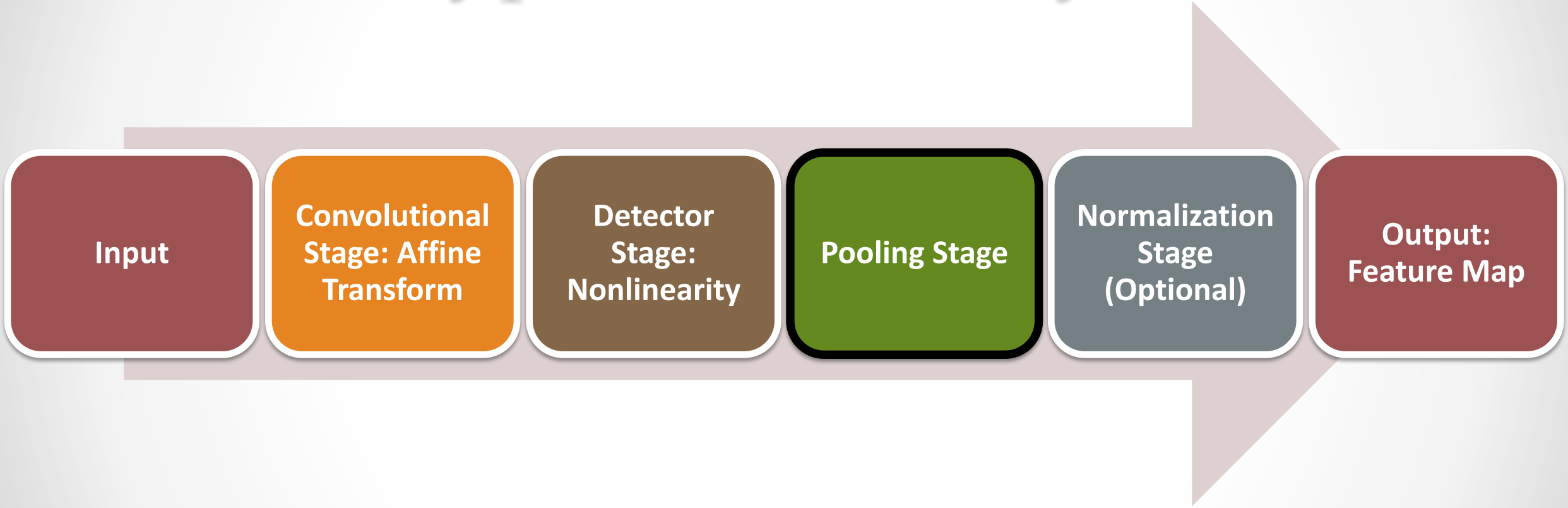
- Each hidden layer is compose of multiple feature maps, $\{x^k, k = 0..K\}$

- Weights, W of a hidden layer can be represented in a 4D tensor containing elements for every combination of **destination feature map, source feature map, source vertical position, and source horizontal position**.

- Biases, b can be represented as a vector containing one element for every destination feature map.

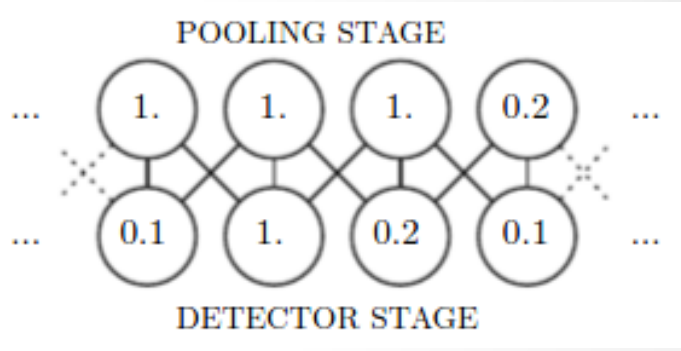


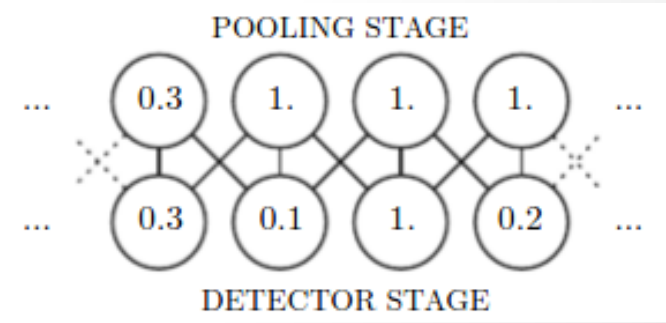
Typical CNN Layer



Pooling

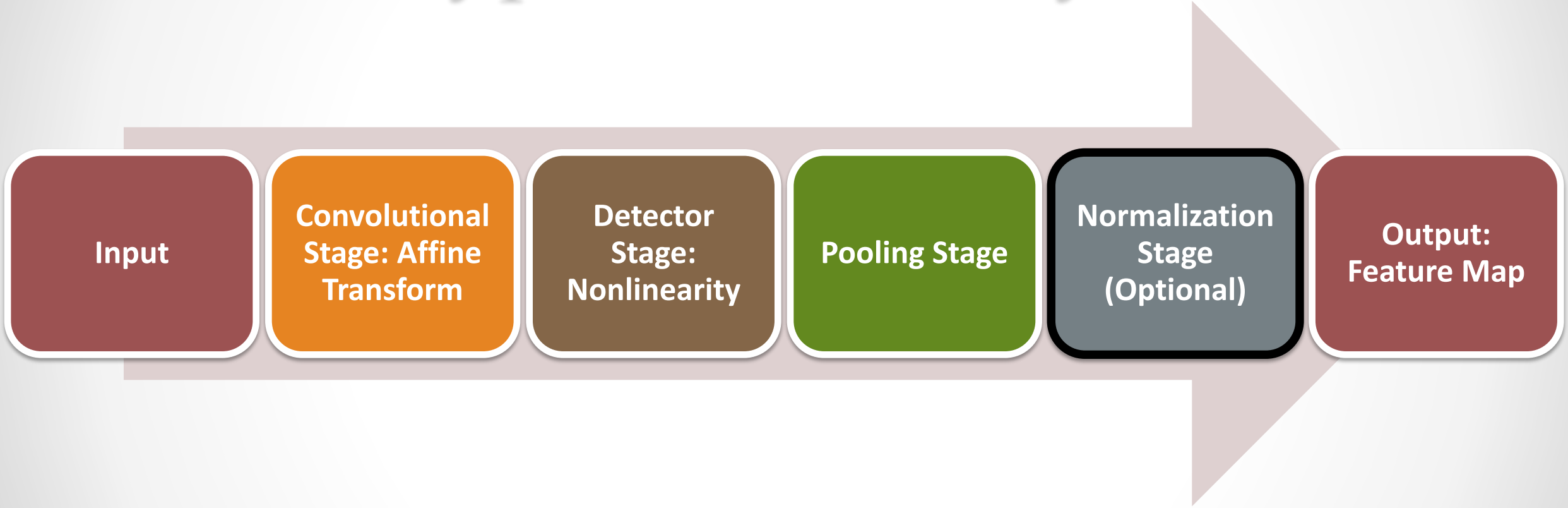
- Non-linear down-sampling to simplify the information in output from convolutional layer.
- Variants:
  - Max pooling (popular)
  - Weighted average based on distance
  - L2 norm of neighborhood
- Reduces computation for upper layers by reporting summary statistics (only with stride > 1)
- Provides translation invariance (Infinitely strong prior that learning must be invariant to small translations)
- Useful property, if we care more about whether some feature is present than exactly where it is, thus adds robustness to position





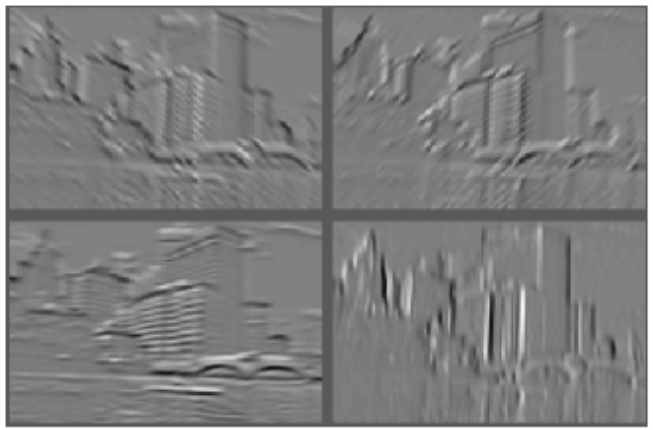
Bottom view has been shifted by 1 pixel w.r.t. Top view.
Every value in the bottom row has changed, but only half the values in the top row has changed!

Typical CNN Layer

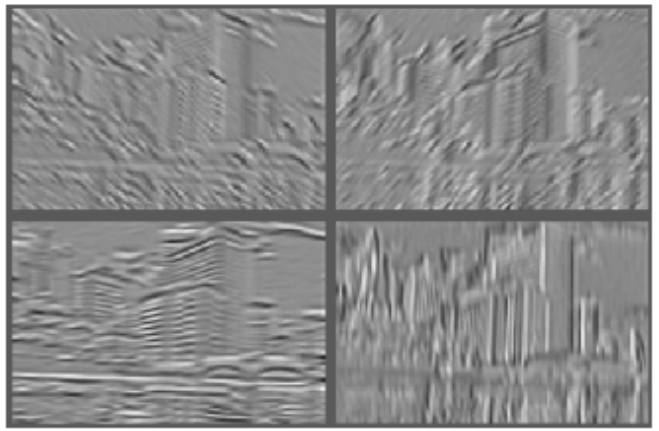


Normalization (Optional)

Locally the response is normalized using some distance based weighted average function

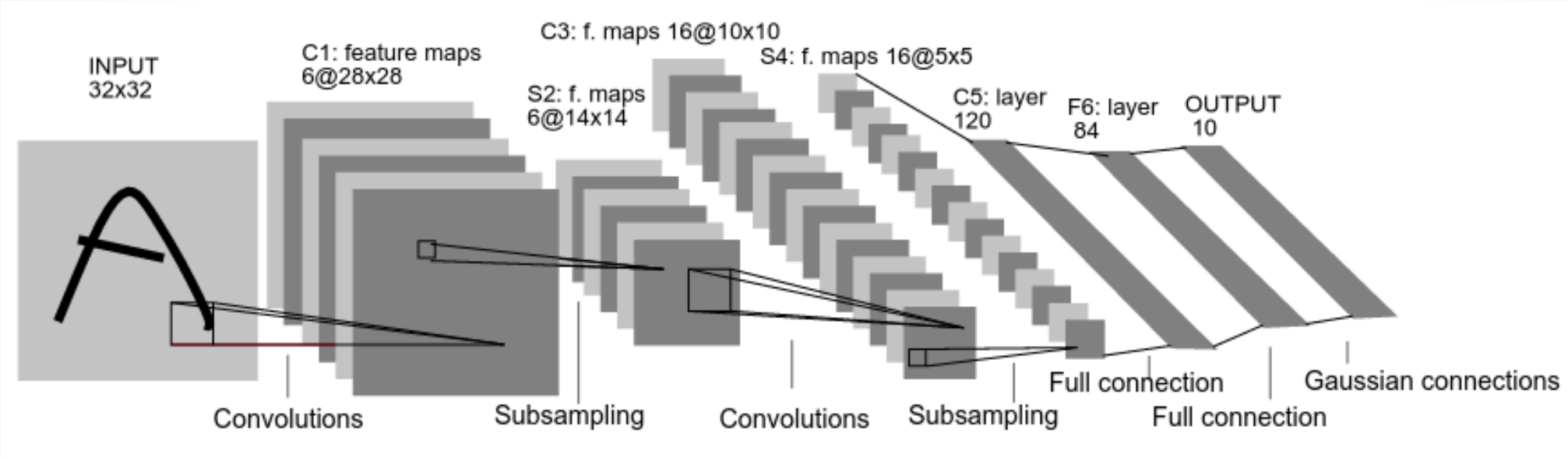


Feature Maps



Feature Maps
After Contrast Normalization

Putting It All Together!



Lenet-5 (Lecun-98), Convolutional Neural Network for digits recognition

Backpropagation

- Loss function
  - **For Classification**
    - Softmax Function $y_j = \frac{e^{\mathbf{x}^\top \mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^\top \mathbf{w}_k}}$ with negative log likelihood $-\sum_{i=1}^{K} t_i \log y_i$
  - **For Regression**
    - Mean squared error $\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(t_i - y_i)^2$
- Weight Update

$$\omega_i \leftarrow \omega_i - \eta\frac{\partial E}{\partial w_i} + \alpha\omega_i - \lambda\eta\omega_i$$

where η - learning rate,
α - momentum,
λ - weight decay

Backpropagation

- *Convolutional Layer*
  - **With error function, E, and filter output x^l,**

$$\frac{\partial E}{\partial \omega_{ab}} = \sum_{i=0}^{N-m} \sum_{j=0}^{N-m} \frac{\partial E}{\partial x_{ij}^{\ell}} \frac{\partial x_{ij}^{\ell}}{\partial \omega_{ab}} = \sum_{i=0}^{N-m} \sum_{j=0}^{N-m} \frac{\partial E}{\partial x_{ij}^{\ell}} y_{(i+a)(j+b)}^{\ell-1}$$

$$\frac{\partial E}{\partial x_{ij}^{\ell}} = \frac{\partial E}{\partial y_{ij}^{\ell}} \frac{\partial y_{ij}^{\ell}}{\partial x_{ij}^{\ell}} = \frac{\partial E}{\partial y_{ij}^{\ell}} \frac{\partial}{\partial x_{ij}^{\ell}} \left(\sigma(x_{ij}^{\ell}) \right) = \frac{\partial E}{\partial y_{ij}^{\ell}} \sigma'(x_{ij}^{\ell})$$

$$\frac{\partial E}{\partial y_{ij}^{\ell-1}} = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} \frac{\partial E}{\partial x_{(i-a)(j-b)}^{\ell}} \frac{\partial x_{(i-a)(j-b)}^{\ell}}{\partial y_{ij}^{\ell-1}} = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} \frac{\partial E}{\partial x_{(i-a)(j-b)}^{\ell}} \omega_{ab}$$

  Thus, the error is propagated to the previous layer.

- *Pooling Layer*
  - **Do not actually learn themselves, just reduce the size of the problem by introducing sparseness.**
  - **Reduces region of k x k size to a single value during forward propagation.**
  - **Error propagated back to the place where it came from, thus errors are rather sparse.**

http://andrew.gibiansky.com/blog/machine-learning/convolutional-neural-networks/

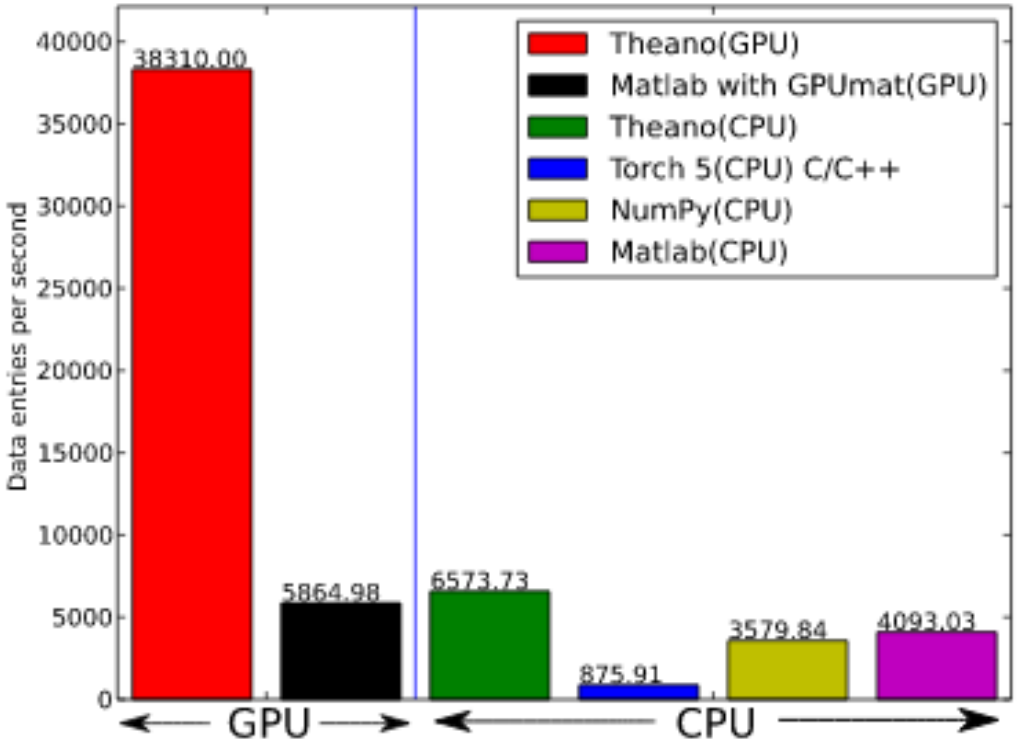
Theano

What is Theano?

- Theano is a Python-based Math Expression Compiler whose syntax is quite similar to NumPy.
- Open-source project developed and maintained by ML group at Université de Montréal.
- User composes mathematical expressions in a high-level description mimicking NumPy's syntax and semantics which allows Theano to provide symbolic differentiation.

Key Features

- Single implementation compatible with both CPU and GPU.
- Theano on its own optimizes using CUDA C++ for GPU.
- **Easy to implement back-propagation** in CNN, as it automatically computes all the mappings involved.
- Creates a graph with the various inputs involved, differentiating using chain rule.



Fitting a multi-layer perceptron to simulated data with SGD having 784 inputs, 500 hidden units, a 10-way classification and training 60 examples at a time

Sneak Peek into Theano...

Theano-based implementations for Deep Learning

- Caffe
- Torch
- Keras

Other Frameworks:

- cuDNN
- DIGITS

Caffe

Key Features

- Deep learning framework (essentially for training CNNs) developed by Berkeley Vision and Learning Center (BVLC)

- **Speed:** Able to process over 60M images per day with a single Nvidia K40 GPU, thus considered to be the fastest convnet implementation available.

- **Expressive Architecture:** Allows models and optimization to be defined as configuration files rather than hard-coding, with ability to switch between CPU and GPU by a single flag.

Sneak Peek into Caffe

Convolutional Layer

```
layers {
  name: "conv1"
  type: CONVOLUTION
  bottom: "data"
  top: "conv1"
  blobs_lr: 1
  blobs_lr: 2
  weight_decay: 1
  weight_decay: 0
  convolution_param {
    num_output: 96
    kernel_size: 7
    stride: 4
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
```

Max Pooling Layer

```
layers {
  name: "pool1"
  type: POOLING
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 2
  }
}
```

Solver

```
net: "trainer.prototxt"
test_iter: 1000
test_interval: 1000
base_lr: 0.001
lr_policy: "step"
gamma: 0.1
stepsize: 10000
display: 20
max_iter: 50000
momentum: 0.9
weight_decay: 0.0005
snapshot: 1000
snapshot_prefix: "snaps/age_train"
solver_mode: GPU
```

Age and Gender Classification using Convolutional Neural Networks

Gil Levi and Tal Hassner
The Open University of Israel

IEEE Workshop on Analysis and Modeling of Faces and Gestures (AMFG), at the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), Boston, June 2015

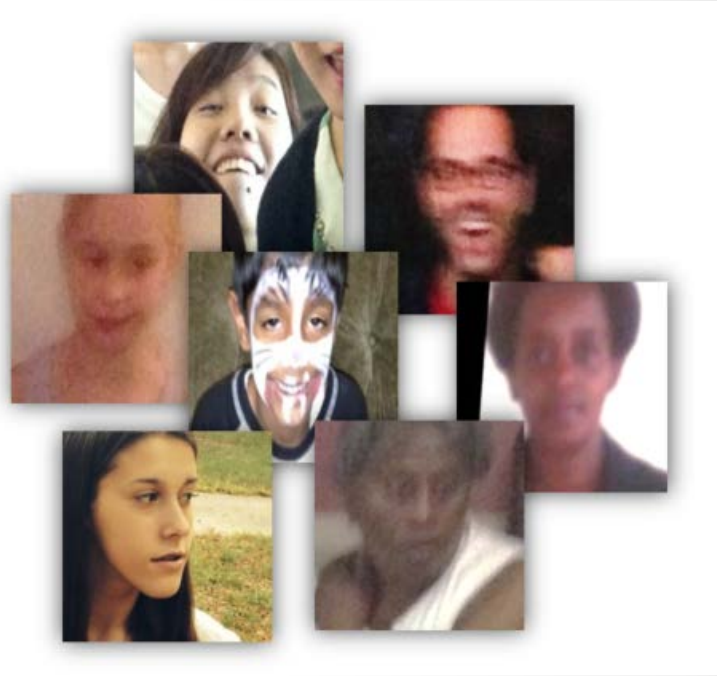
Overview

- Uses deep-convolutional neural networks (CNN) for the task of automatic age and gender classification.

- Despite the very challenging nature of the images in the Adience dataset and the simplicity of the network design used, the method significantly outperforms existing state of the art by substantial margins.

Dataset - The Adience Benchmark

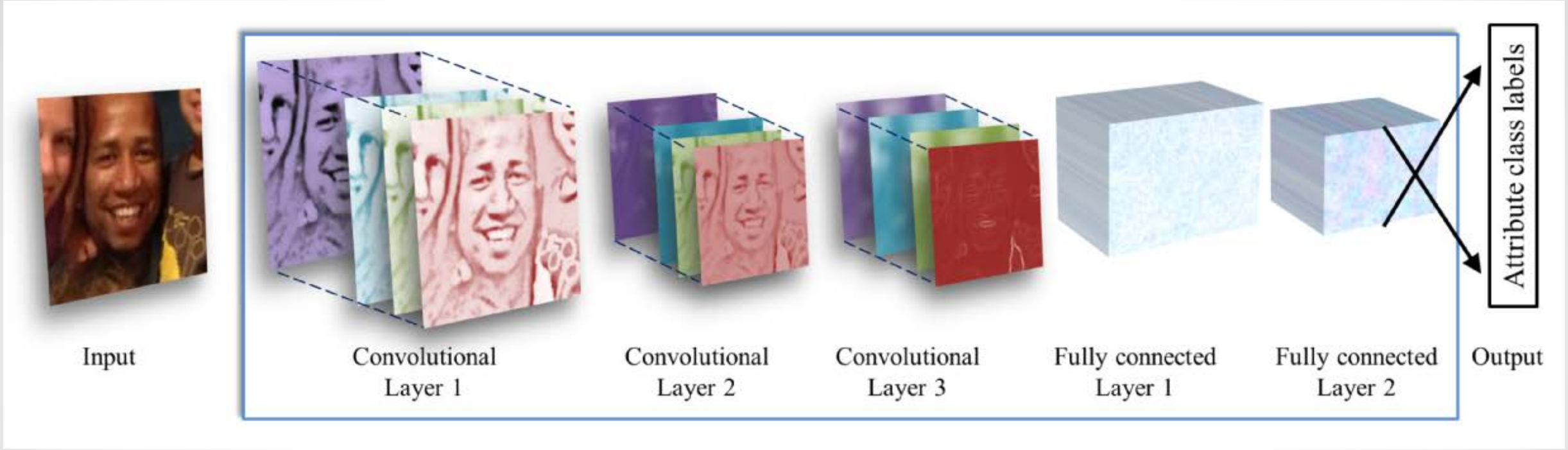
- Consists of images automatically uploaded to Flickr from smartphones.
- Viewing conditions of these images are highly unconstrained, thus capturing extreme variations in head pose, lightning conditions, blur, occlusion, expressions and more.
- Includes roughly 26K images of 2,284 subjects.
- For the tests, in-plane aligned version of the faces is used.



Faces from Adience benchmark (above) and breakdown into different classes (below)

	0-2	4-6	8-13	15-20	25-32	38-43	48-53	60-	Total
Male	745	928	934	734	2308	1294	392	442	8192
Female	682	1234	1360	919	2589	1056	433	427	9411
Both	1427	2162	2294	1653	4897	2350	825	869	19487

Network Architecture



All 3 RGB channels First, resized to 256 x 256, then cropped to 227 x 227

96 filters size 3x7x7

256 filters size 96x5x5

384 filters size 256x3x3

Each convolutional layer is followed by rectified linear operator (ReLU), max pooling layer of 3x3 regions with 2-pixel strides and a local normalization layer

Both fully connected layers contain 512 neurons followed by ReLU and dropout layer

Output to class labels (age / gender)

Measures to reduce overfitting

- Lean network architecture using just 3 convolutional layers and 2 fully connected layers considering the size of the dataset and labels involved (8 age classes and 2 gender classes)
- **Dropout learning:** Randomly set the output value of network neurons to 0 with a dropout ratio of 0.5 (50% chance)
- **Weight decay:** Used to keep the magnitude of weights close to zero
- **Data Augmentation:** Took random crop of 227x227 from image of 256x256 and randomly mirrored it in each forward-backward training pass

All these measures help in keeping the number of free parameters in the network low reducing complexity and thus over-fitting

Experiments

```
net: "trainer.prototxt"
test_iter: 1000
test_interval: 1000
base_lr: 0.001
lr_policy: "step"
gamma: 0.1
stepsize: 10000
display: 20
max_iter: 50000
momentum: 0.9
weight_decay: 0.0005
snapshot: 1000
snapshot_prefix: "snaps/age_train"
solver_mode: GPU
```

Solver

5-fold cross validation based on pre-specified subject exclusive folds distribution

What they used

- Trained on Amazon GPU machine with 1,536 CUDA cores and 4 GB GDDR5 RAM

What I used

- Trained on Nvidia Quadro K2200 with 640 CUDA cores and 4 GB GDDR5 RAM

Results

Gender Classification

Method	Accuracy	
	Paper	Reimplementation
Single-Crop	85.9 ±1.4	86.7 ± 1.5
Over-Sample	86.8 ± 1.4	**87.4 ± 0.9**

Age Estimation

Method	Accuracy			
	Paper		Reimplementation	
	Exact	One-off	Exact	One-off
Single-Crop	49.5 ± 4.4	84.6 ± 1.7	49.5 ± 3.6	85.4 ± 1.8
Over-Sample	50.7 ± 5.1	84.7 ± 2.2	50.6 ± 5.0	**85.8 ± 1.5**

Results - Age Estimation Confusion Matrix

Paper

Actual Labels \ Predicted Labels	0-2	4-6	8-13	15-20	25-32	38-43	48-53	60-
0-2	0.699	0.147	0.028	0.006	0.005	0.008	0.007	0.009
4-6	0.256	0.573	0.166	0.023	0.010	0.011	0.010	0.005
8-13	0.027	0.223	0.552	0.150	0.091	0.068	0.055	0.061
15-20	0.003	0.019	0.081	0.239	0.106	0.055	0.049	0.028
25-32	0.006	0.029	0.138	0.510	0.613	0.461	0.260	0.108
38-43	0.004	0.007	0.023	0.058	0.149	0.293	0.339	0.268
48-53	0.002	0.001	0.004	0.007	0.017	0.055	0.146	0.165
60-	0.001	0.001	0.008	0.007	0.009	0.050	0.134	0.357

Reimplementation

Actual Labels \ Predicted Labels	0-2	4-6	8-13	15-20	25-32	38-43	48-53	60-
0-2	0.741	0.139	0	0.028	0	0	0	0.093
4-6	0.057	0.654	0.135	0.135	0	0	0	0.019
8-13	0	0.114	0	0.828	0.057	0	0	0
15-20	0.018	0.119	0.065	0.653	0.106	0.015	0.010	0.010
25-32	0.009	0.094	0.009	0.471	0.292	0.037	0.037	0.047
38-43	0.02	0	0	0.22	0.56	0.14	0.06	0
48-53	0	0.1	0.033	0.067	0.133	0.267	0.4	0
60-	0.238	0.012	0	0.008	0	0	0	0.740

References

- http://deeplearning.net/tutorial/
- http://goodfeli.github.io/dlbook/contents/convnets.html
- http://neuralnetworksanddeeplearning.com/chap6.html
- http://deeplearning.net/software/theano/tutorial/
- http://andrew.gibiansky.com/blog/machine-learning/convolutional-neural-networks/
- Andrew Ng: Deep Learning, Self-Taught Learning and Unsupervised Feature Learning
- www.caffe.berkeleyvision.org
- http://www.openu.ac.il/home/hassner/Adience/index.html
- https://www.wikipedia.org/
- www.cse.ust.hk/~leichen/courses/FYTG.../FYTGS5101-Guoyangxie.pdf

- LeCun, Yann, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86.11 (1998): 2278-2324.
- Bergstra, James, et al. "Theano: a CPU and GPU math expression compiler." *Proceedings of the Python for scientific computing conference (SciPy)*. Vol. 4. 2010.
- Gil Levi and Tal Hassner, *Age and Gender Classification using Convolutional Neural Networks*, IEEE Workshop on Analysis and Modeling of Faces and Gestures (AMFG), at the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), Boston, June 2015