CPSC 540: Machine Learning

Deep CRFs, Convolutional Neural Networks

Mark Schmidt

University of British Columbia

Winter 2017

Admin

- Assignment 4:
    - 1 late day to hand in tonight, 2 for Monday.

- Assignment 5 coming soon.
- Project description coming soon.
- Final details coming soon.

- Bonus lecture on April 10th (same time and place)?

Last Time: Log-Linear Models

- We discussed log-linear models like the Ising model,

$$p(x_1, x_2, \dots, x_d) = \frac{1}{Z}\exp\left(\sum_{i=1}^{d} x_i w + \sum_{(i,j)\in E} x_i x_j v\right),$$

  where if v is large it encourages neighbouring values to be the same.

- We also discussed CRF variants that generalize logistic regression,

$$p(y^1, y^2, \dots, y^d | x^1, x^2, \dots, x^d) = \frac{1}{Z}\exp\left(\sum_{i=1}^{d} y^i w^T x^i + \sum_{(i,j)\in E} y^i y^j v\right).$$

- We can jointly learn a logistic regression model and the label depencies.

Last Time: Structured SVMs

- In generative UGM models (MRFs) we optimize the joint likelihood,

$$f(w) = -\sum_{i=1}^{n} \log p(y^i, x^i | w).$$

- In discriminative UGM models (CRFs) we optimize the conditional likelihood,

$$f(w) = -\sum_{i=1}^{n} \log p(y^i | x^i, w).$$

- In structured SVMs we penalize decision from decoding conditional likelihood,

$$f(w) = \sum_{i=1}^{n} \max_{y'} [g(y^i, y') - \log p(y^i | x^i, w) + \log p(y' | x^i, w)],$$

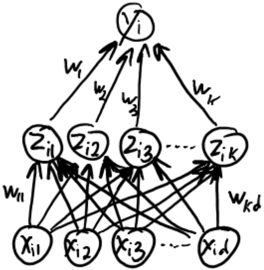
where $g(y^i, y')$ is penalty for predicting y' when true label is y^i.

  - Only requires decoding to evaluate objective/subgradient.

Outline

Feedforward Neural Networks

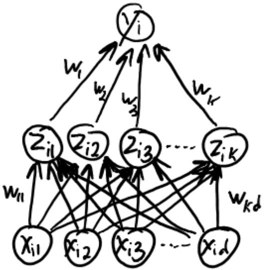
- In 340 we discussed feedforward neural networks for supervised learning.
- With 1 hidden layer the classic model has this structure:



- Motivation:
  - For some problems it's hard to find good features.
  - This learn features z that are good for supervised learning.

Neural Networks as DAG Models

- It's a DAG model but there is an important difference with our previous models:
  - The latent variables z_c are deterministic functions of the x_j.



- Makes inference trivial: if you observe all x_j you also observe all z_c.
  - In this case y is the only random variable.

Neural Network Notation

- We'll continue using our supervised learning notation:

$$X = \begin{bmatrix} \text{——} & (x^1)^T & \text{——} \\ \text{——} & (x^2)^T & \text{——} \\ & \vdots & \\ \text{——} & (x^n)^T & \text{——} \end{bmatrix}, \quad y = \begin{bmatrix} y^1 \\ y^2 \\ \vdots \\ y^n \end{bmatrix},$$

- For the latent features and two sets of parameters we'll use

$$Z = \begin{bmatrix} \text{——} & (z^1)^T & \text{——} \\ \text{——} & (z^2)^T & \text{——} \\ & \vdots & \\ \text{——} & (z^n)^T & \text{——} \end{bmatrix}, \quad w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_k \end{bmatrix}, \quad W = \begin{bmatrix} \text{——} & W_1 & \text{——} \\ \text{——} & W_2 & \text{——} \\ & \vdots & \\ \text{——} & W_k & \text{——} \end{bmatrix},$$

where Z is n by k and W is k by d.

Introducing Non-Linearity

- We discussed how the "linear-linear" model,

$$z^i = Wx^i, \quad y^i = w^T z^i,$$

is degenerate since it's still a linear model.

- The classic solution is to introduce a non-linearity,

$$z^i = h(Wx^i), \quad y^i = w^T z^i,$$

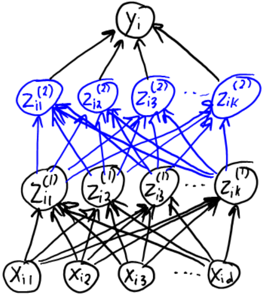
where a common-choice is applying sigmoid element-wise,

$$z_c^i = \frac{1}{1+\exp(-W_c x^i)},$$

which is said to be the "activation" of neuron c on example i.
  - A universal approximator with k large enough.

Deep Neural Networks

- In deep neural networks we add multiple hidden layers,

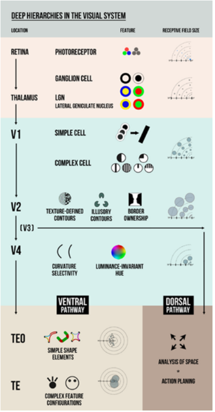


- Mathematically, with 3 hidden layers the classic model uses

$$y^i = w^T \underbrace{h(W^3 \underbrace{h(W^2 \underbrace{h(W^1 x^i)}_{z^{i1}})}_{z^{i2}})}_{z^{i3}} .$$

Biological Motivation

- Deep learning is motivated by theories of deep hierarchies in the brain.



https://en.wikibooks.org/wiki/Sensory_Systems/Visual_Signal_Processing

- But most research is about making models work better, not be more brain-like.

Deep Neural Network History

- Popularity of deep learning has come in waves over the years.
  - Currently, it is one of the hottest topics in science.

- Recent popularity is due to unprecedented performance on some difficult tasks:
  - Speech recognition.
  - Comptuer vision.
  - Machine translation.
- These are mainly due to big datasets, deep models, and tons of computation.
  - Plus some tweaks to the classic models.

- For a NY Times article discussing some of the history/successes/issues, see:

https://mobile.nytimes.com/2016/12/14/magazine/the-great-ai-awakening.html

Training Deep Neural Networks

- If we're training a 3-layer network with squared error, our objective is

$$f(w, W^1, W^2, W^3) = \frac{1}{2}\sum_{i=1}^{n}(w^T h(W^3 h(W^2 h(W^1 x^i))) - y^i)^2.$$

- Usual training procedure is stochastic gradient.
- Highly non-convex and notoriously difficult to tune.
- Recent empirical/theoretical work indicates non-convexity may not be an issue:
  - All local minima may be good for "large enough" networks.
- We're discovering sets of tricks to make things easier to tune.

Training Deep Neural Networks

- Some common data/optimization tricks we discussed in 340:
  - Data transformations.
    - For images, translate/rotate/scale/crop each x^i to make more data.
  - Data standardization: centering and whitening.
  - Adding bias variables.
    - For sigmoids, corresponds to adding row of zeroes to each W^m.
  - Parameter initalization: "small but different", standardizing within layers.
  - Step-size selection: "babysitting", Bottou trick, Adam, momentum.
  - Momentum: heavy-ball and Nesterov-style modifications.
  - Batch normalization: adaptive standardizing within layers.
  - ReLU: replacing sigmoid with $[W_c x^i]^+$.
    - Avoids gradients extremely-close to zero.

Training Deep Neural Networks

- Some common regularization tricks we discussed in 340:
  - Standard L2-regularization or L1-regularization "weight decay".
    - Sometimes with different λ for each layer.
  - Early stopping of the optimization based on validation accuracy.
  - Dropout randomly zeroess z values to discourage dependence.
  - Hyper-parameter optimization to choose various tuning parameters.
  - Special architectures like convolutional neural networks and LSTMs (later).
    - Yields W^m that are very sparse and have many tied parameters.

Backpropagation as Message-Passing

- Computing the gradient in neural networks is called backpropagation.
  - Derived from the chain rule and memoization of repeated quantities.
- We're going to view backpropagation as a message-passing algorithm.
- Key advantages of this view:
  - It's easy to handle different graph structures.
  - It's easy to handle different non-linear transformations.
  - It's easy to handle multiple outputs (as in structured prediction).
  - It's easy to add non-deterministic parts and combine with other graphical models.

Backpropagation Forward Pass

- Consider computing the output of a neural network for an example i,

$$
\begin{aligned}
y^i &= w^T h(W^3 h(W^2 h(W^1 x^i))) \\
&= \sum_{c=1}^{k} w_c h\left(\sum_{c'=1}^{k} W^3_{c'c} h\left(\sum_{c''=1}^{k} W^2_{c''c'} h\left(\sum_{j=1}^{d} W^1_{c''j} x^i_j\right)\right)\right).
\end{aligned}
$$

  where we've assume that all hidden layers have k values.
  - In the second line, the h functions are single-input single-output.

- The nested sum structure is similar to our message-passing structures.

- However, it's easier because it's deterministic: no random variables to sum over.
  - The messages will be scalars rather than functions.

Backpropagation Forward Pass

- Forward propagation through neural network as message passing:

$$
\begin{aligned}
y^i &= \sum_{c=1}^{k} w_c h\left(\sum_{c'=1}^{k} W^3_{c'c} h\left(\sum_{c''=1}^{k} W^2_{c''c'} h\left(\sum_{j=1}^{d} W^1_{c''j} x^i_j\right)\right)\right) \\
&= \sum_{c=1}^{k} w_c h\left(\sum_{c'=1}^{k} W^3_{c'c} h\left(\sum_{c''=1}^{k} W^2_{c''c'} h(M_{c''})\right)\right) \\
&= \sum_{c=1}^{k} w_c h\left(\sum_{c'=1}^{k} W^3_{c'c} h(M_{c'})\right) \\
&= \sum_{c=1}^{k} w_c h(M_c) \\
&= M_y,
\end{aligned}
$$

where intermediate messages are the z before transformation, $z^{i2}_{c'} = h(M_2(c'))$.

Backpropagation Backward Pass

- The backpropagation backward pass computes the partial derivatives.
  - For a loss f, the partial derivatives in the last layer have the form

$$\frac{\partial f}{\partial w_c} = z_c^{i3} f'(w^T h(W^3 h(W^2 h(W^1 x^i)))),$$

where

$$z_{c'}^{i3} = h\left(\sum_{c'=1}^{k} W_{c'c}^{3} h\left(\sum_{c''=1}^{k} W_{c''c'}^{2} h\left(\sum_{j=1}^{d} W_{c''j}^{1} x_j^i\right)\right)\right).$$

  - Written in terms of messages it simplifies to

$$\frac{\partial f}{\partial w_c} = h(M_c) f'(M_y).$$

Backpropagation Backward Pass

- In terms of forward messages, the partial derivatives have the forms:

$$\frac{\partial f}{\partial w_c} = h(M_c)f'(M_y),$$

$$\frac{\partial f}{\partial W^3_{c'c}} = h(M_{c'})h'(M_c)w_c f'(M_y),$$

$$\frac{\partial f}{\partial W^2_{c''c'}} = h(M_{c''})h'(M_{c'})\sum_{c=1}^{k} W^3_{c'c}h'(M_c)w_c f'(M_y),$$

$$\frac{\partial f}{\partial W^1_{jc''}} = h(M_j)h'(M_{c''})\sum_{c'=1}^{k} W^2_{c''c'}h'(M_{c'})\sum_{c=1}^{k} W^3_{c'c}h'(M_c)w_c f'(M_y),$$

which are ugly but notice all the repeated calculations.

Backpropagation Backward Pass

- It's again simpler using appropriate messages

$$\frac{\partial f}{\partial w_c} = h(M_c) f'(M_y),$$

$$\frac{\partial f}{\partial W^3_{c'c}} = h(M_{c'}) h'(M_c) w_c V_y,$$

$$\frac{\partial f}{\partial W^2_{c''c'}} = h(M_{c''}) h'(M_{c'}) \sum_{c=1}^{k} W^3_{c'c} V_c,$$

$$\frac{\partial f}{\partial W^1_{jc''}} = h(M_j) h'(M_{c''}) \sum_{c'=1}^{k} W^2_{c''c'} V_{c'},$$

where $M_j = x_j$.

Backpropagation as Message-Passing

- The general forward message for child c with parents p and weights W is

$$M_c = \sum_p W_{cp} h(M_p),$$

compute weighted combination of non-linearly transformed parents.
  - In the first layer we don't apply h.
- The general backward message from child c to *all* its parents is

$$V_c = h'(M_c) \sum_{c'} W_{cc'} V_{c'},$$

which weights the "grandchildren's gradients".
  - Last layer uses f instead of h.
- The gradient of W_{cp} is $h(M_p)V_c$, which works for general graphs.

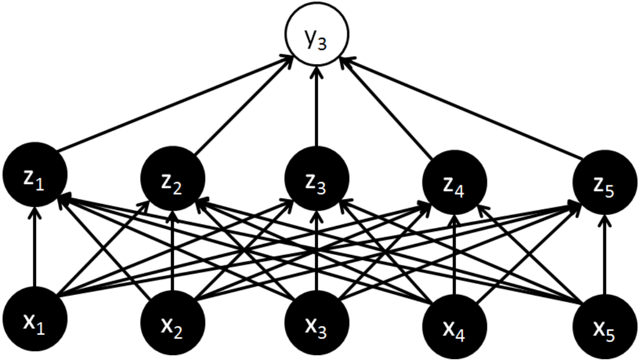
Outline

Back to Structured Prediciton

- Recall that we've been discussing structured prediction
  - We may have multiple labels y_j are a general set of output objects.
- We can also use deep neural networks for structured prediction.
- Provides an alternative/complementary approach to using CRFs.
  - We'll build up to deep CRFs by parts...

Independent Logistic Regression

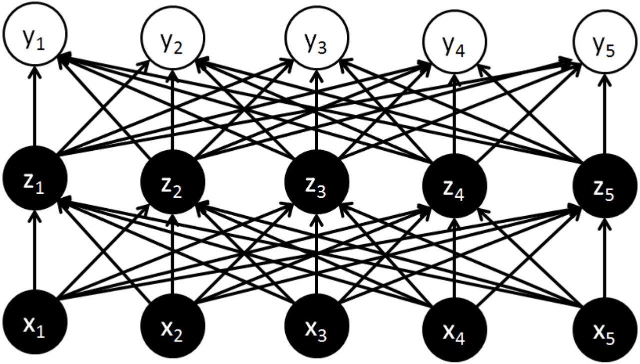
- Our "base" model will be independent logistic regression with fixed featuers z_c.



- For example,
  - We have an image with pixels x_j (usually multiple channels, too).
  - From the pixels x_j we extract features z_c (usually convolutions).
  - From the features we predict label of pixel y_3 (usually linear classifier).

Independent Logistic Regression

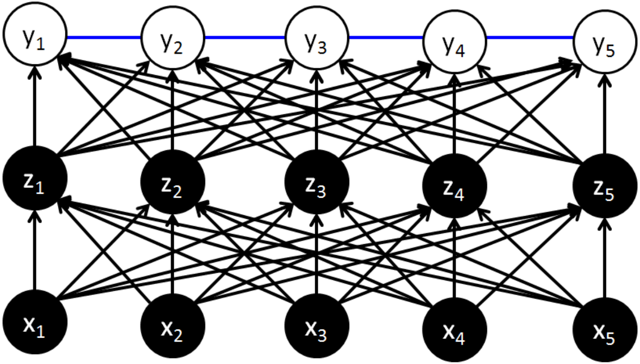
- Strucutred prediction with the independent model classifies each y_j independently.



- This model labels all y_j independently.
  - Although some dependence exists because of the features.

Conditional Random Field

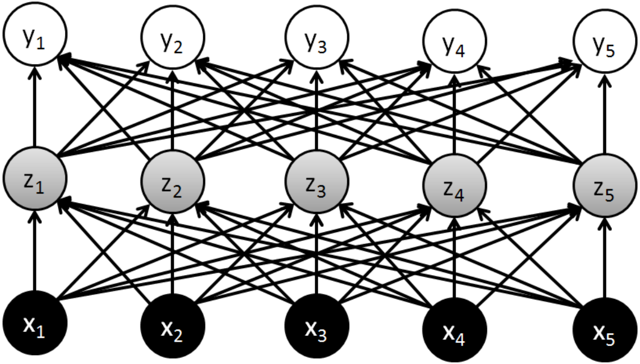
- Conditional random fields model dependencies in the y_j.



- Dependence is capture by a UGM on the y_j.
  - Looks weird to have "directed" parents, but *observed* so don't induce dependencies.

Neural Networks

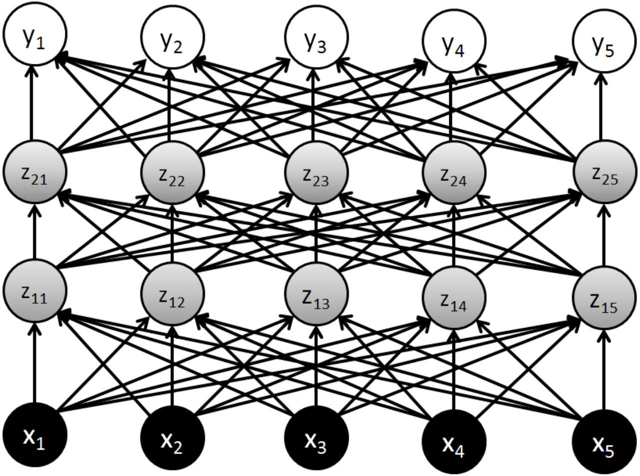
- Alternative to *fixed* features is to learn the features using neural networks.



- I'm using gray nodes because they're not observed but not random.
- This multi-output network shares features z_c across "tasks".
  - It's learning features that are good for predicting multiple labels.

Deep Neural Networks

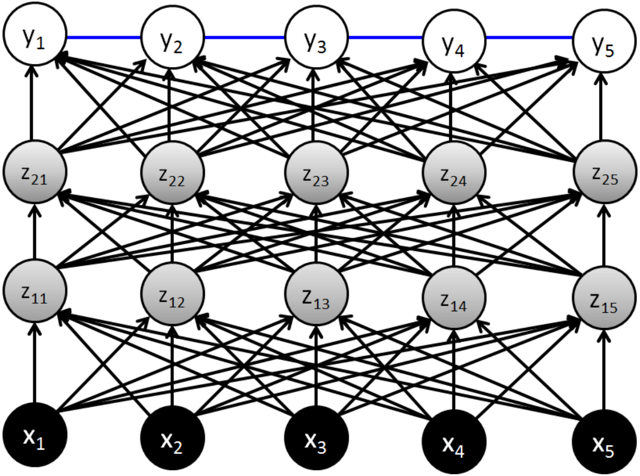
- Adding more layers increases our expressive power.



- More powerful than any fixed feature set, but doesn't model y_j dependence.

Conditional Neural Fields

- Conditional neural fields learn a neural network and CRF simultaneously:



- Because the $z_{cc'}$ are deterministic, does not increase cost of inference.

Conditional Neural Fields

- We can think of conditional neural fields as conditional log-linear models,

$$p(y|x,v) = \frac{1}{Z}\exp(v^T F(y,x)),$$

  but where F is the output of a neural network with its own parameters.
  - We learn v and the neural net parameters jointly.

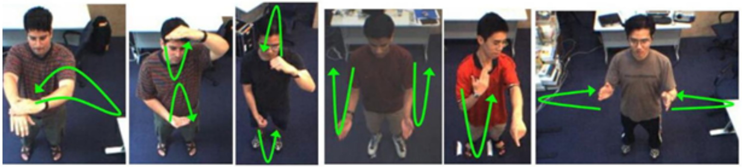
- Computing the gradient:
  1. Forward pass through neural network to get $z_{cc'}$ values (gives $F(y,x)$).
  2. Forward-backward algorithm to get marginals of y_j values.
  3. Backwards pass through neural network to get all gradients.

Beyond Combining CRFs and Neural Nets

- Conditional random fields combine UGMs with supervised learning.
- Conditioanl neural fields add deep learning to the mix.
- But we said that UGMs are more powerful when combined with other tricks:
  - Mixture models, latent factors, approximate inference.

Motivation: Gesture Recognition

- Want to recognize gestures from video:

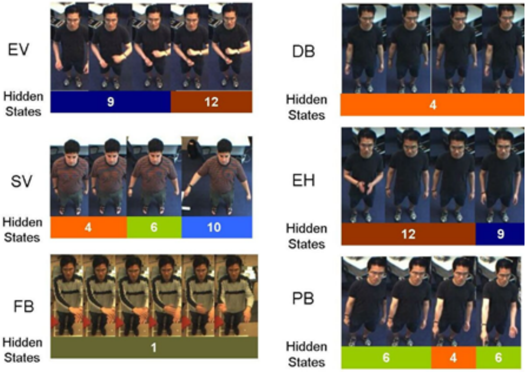


http://groups.csail.mit.edu/vision/vip/papers/wang06cvpr.pdf

- A gesture is composed of a sequence of parts:
  - And some parts appear in different gestures.

Motivation: Gesture Recognition

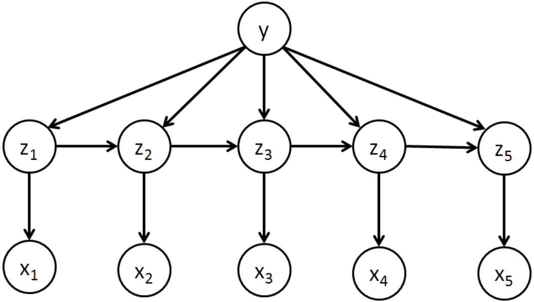
- We have a label for the whole sequence ("gesture") but no part labels.
  - We don't even know the set of possible parts.



http://groups.csail.mit.edu/vision/vip/papers/wang06cvpr.pdf

Generative Classifier based on an HMM

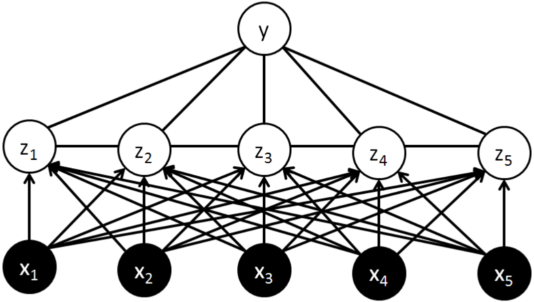
- We could address this scenario using a generative HMM model.



- Observed variable x_j in the image at time j (in this case x_j is a video frame).
- The gesture y is defined by sequence of parts z_j.
  - And we're learning what the parts should be.
- But modelling $p(x_j|z_j)$ is hard (probability of video frame given the hidden part).

Hidden Conditional Random Field

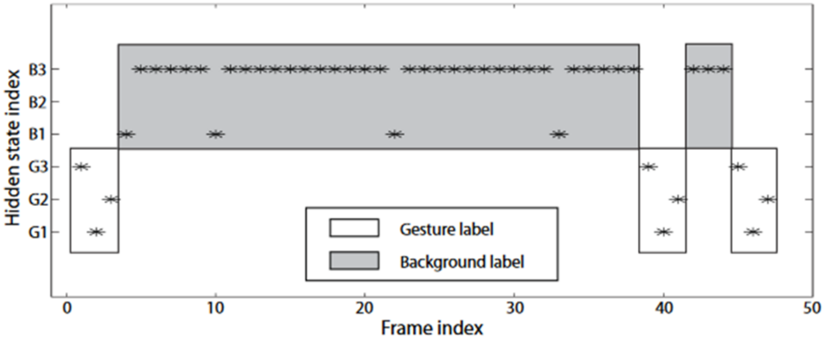
- A discriminative alternative is a hidden conditional random field.



- The label y is based on a "hidden" CRF on the z_j values.
  - Again learns the parts as well as their temporal dependence.
- Treats the x_j as fixed so we don't need to model the video.

Motivation: Gesture Recognition

- What if we want to label video with multiple potential gestures?
  - We're given a labeled video sequence.

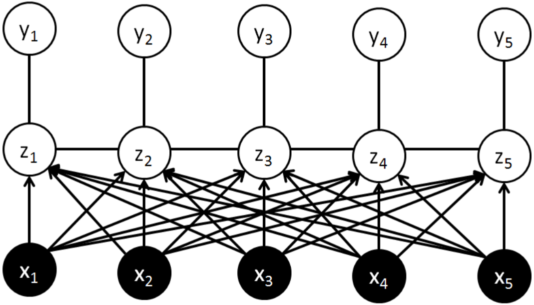


http://www.lsi.upc.edu/~aquattoni/AllMyPapers/cvpr_07_L.pdf

- Our videos are labeled with "gesture" and "background" frames,
  - But we again don't know the parts (G1, G2, G3, B1, B2, B3) that define the labels.

Latent-Dynamic Conditional Random Field

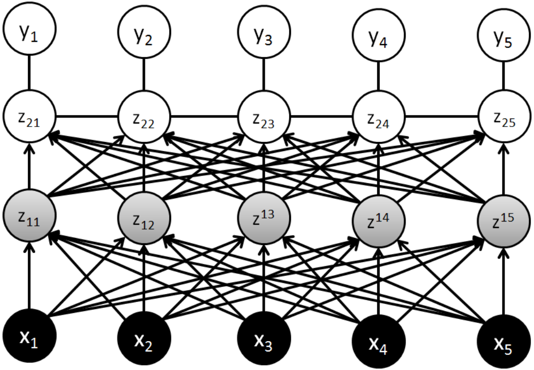
- Here we could use a latent-dynamic conditional random field



- The z_j still capture "latent dynamics", but we have a label y_j for each time.
- Notice in the above case that the conditional UGM is a tree.

Latent-Dynamic Conditional Neural Field

- Latent dynamic conditional neural field also learn features with a neural network..

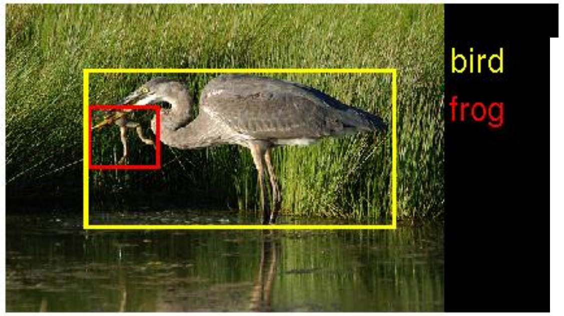


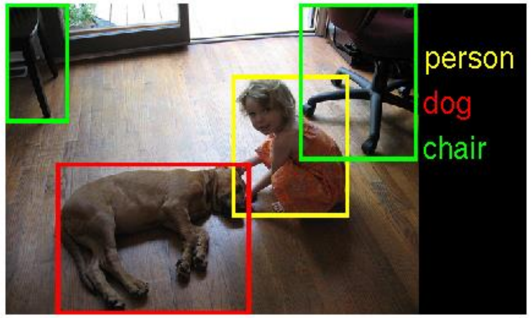
- Combines deep learning, mixture models, and garphical models.
    - This type of model is among state of the art in several applications.

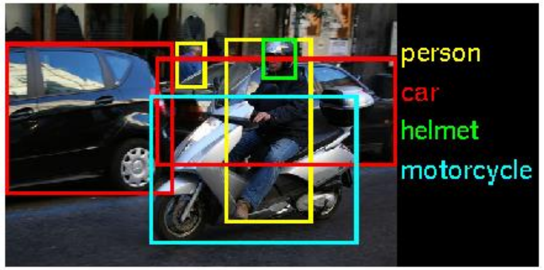
Outline

ImageNet Challenge

- Millions of labeled images, 1000 object classes.







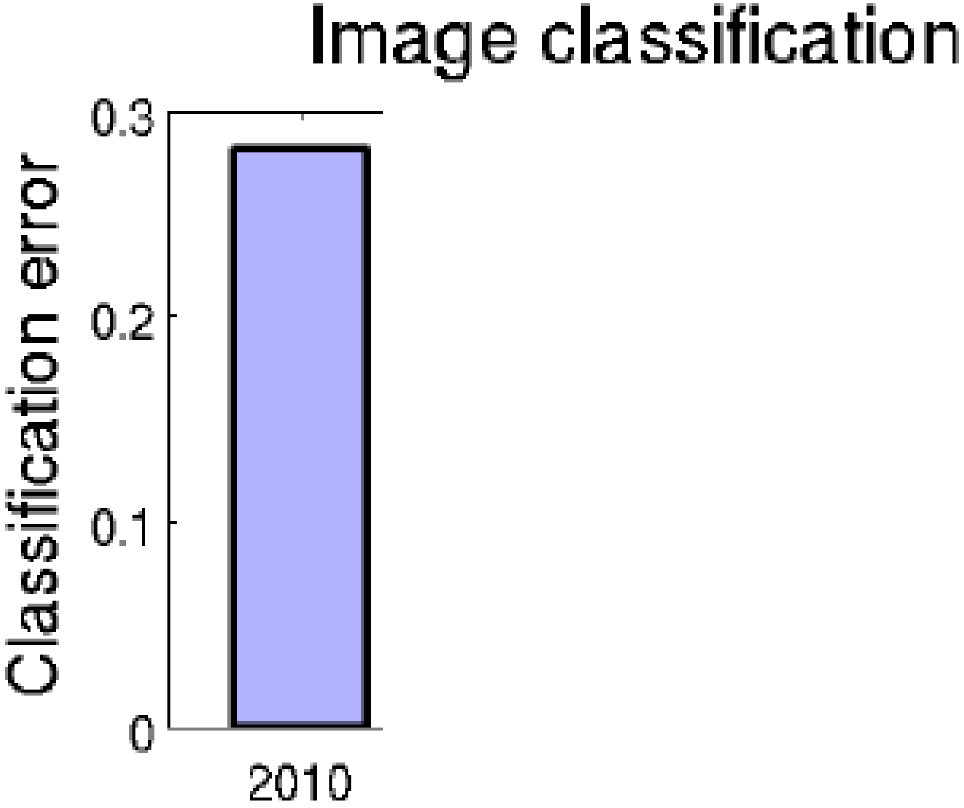
Easy for humans but hard for computers.

http://www.image-net.org/challenges/LSVRC/2014/

ImageNet Challenge

- Object detection task:
  - Single label per image.
  - Humans: ~5% error.

(a) Siberian husky (b) Eskimo dog

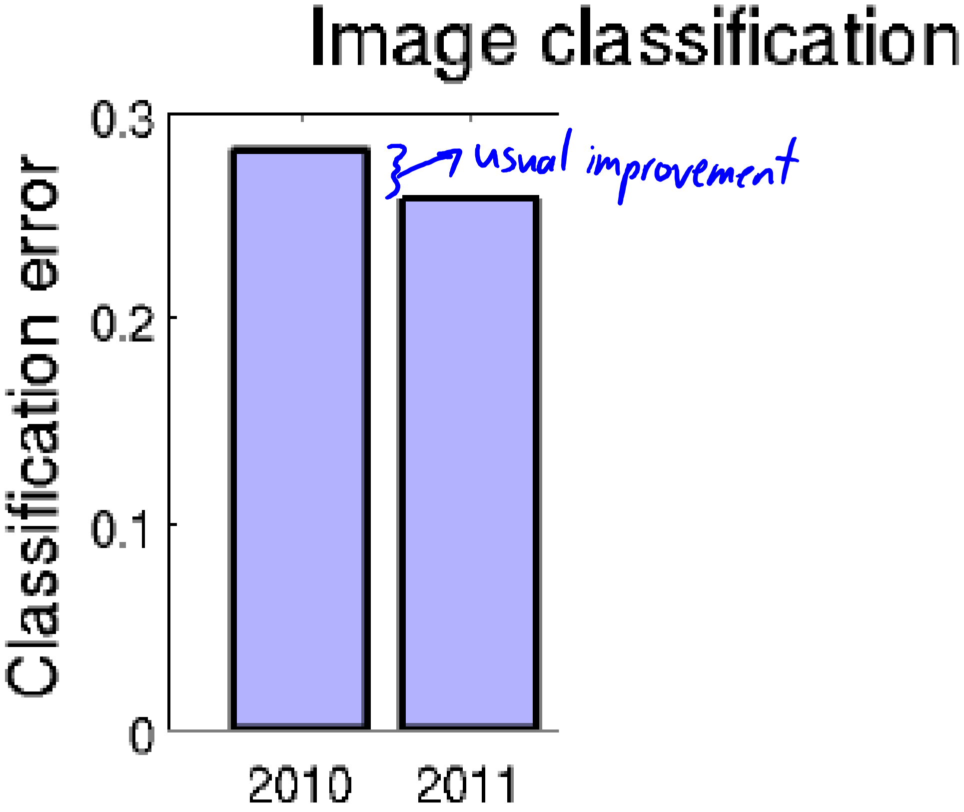


https://ischlag.github.io/2016/04/05/important-ILSVRC-achievements/
http://arxiv.org/pdf/1409.0575v3.pdf
http://arxiv.org/pdf/1409.4842v1.pdf

ImageNet Challenge

- Object detection task:
  - Single label per image.
  - Humans: ~5% error.

(a) Siberian husky (b) Eskimo dog



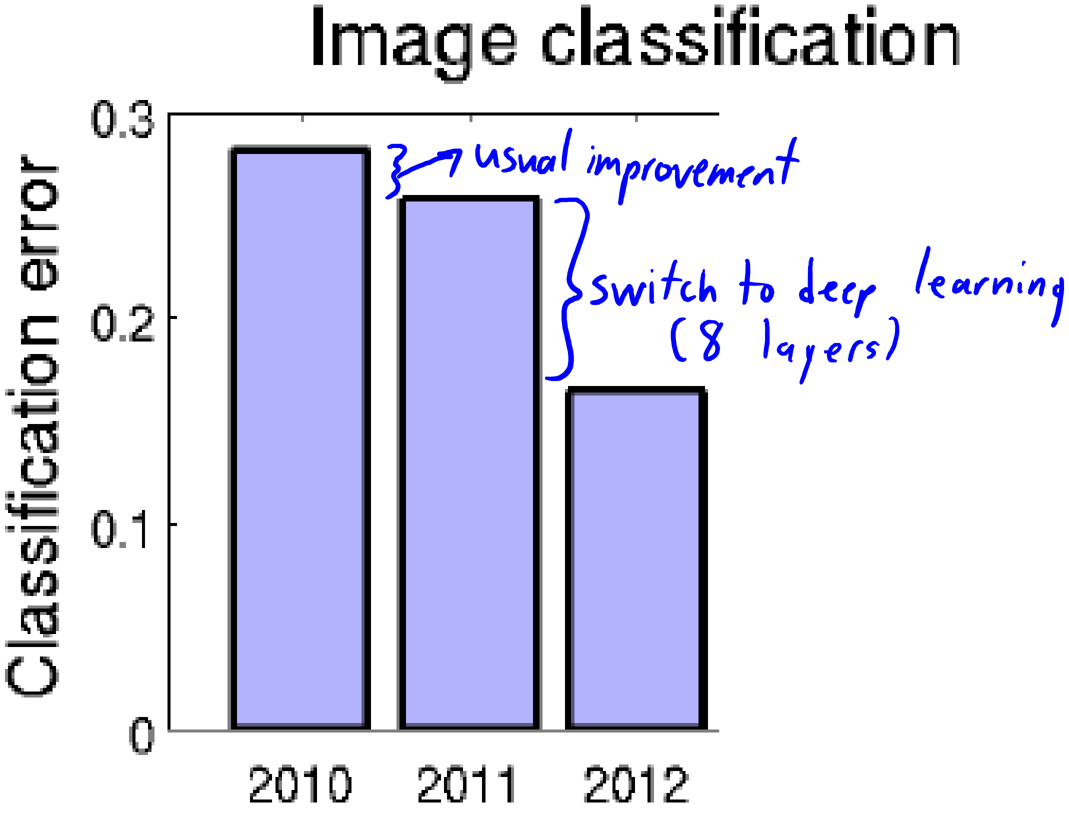
https://ischlag.github.io/2016/04/05/important-ILSVRC-achievements/
http://arxiv.org/pdf/1409.0575v3.pdf
http://arxiv.org/pdf/1409.4842v1.pdf

ImageNet Challenge

- Object detection task:
  - Single label per image.
  - Humans: ~5% error.

(a) Siberian husky

(b) Eskimo dog

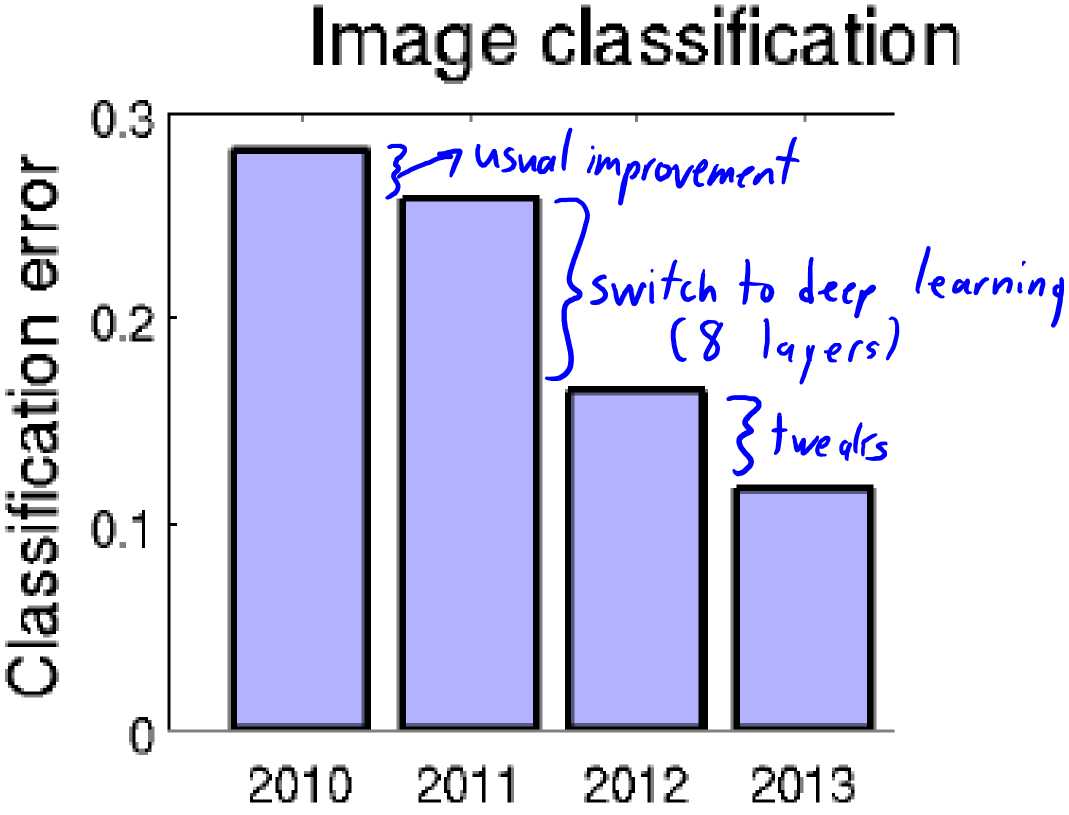


https://ischlag.github.io/2016/04/05/important-ILSVRC-achievements/
http://arxiv.org/pdf/1409.0575v3.pdf
http://arxiv.org/pdf/1409.4842v1.pdf

ImageNet Challenge

- Object detection task:
  - Single label per image.
  - Humans: ~5% error.

(a) Siberian husky (b) Eskimo dog

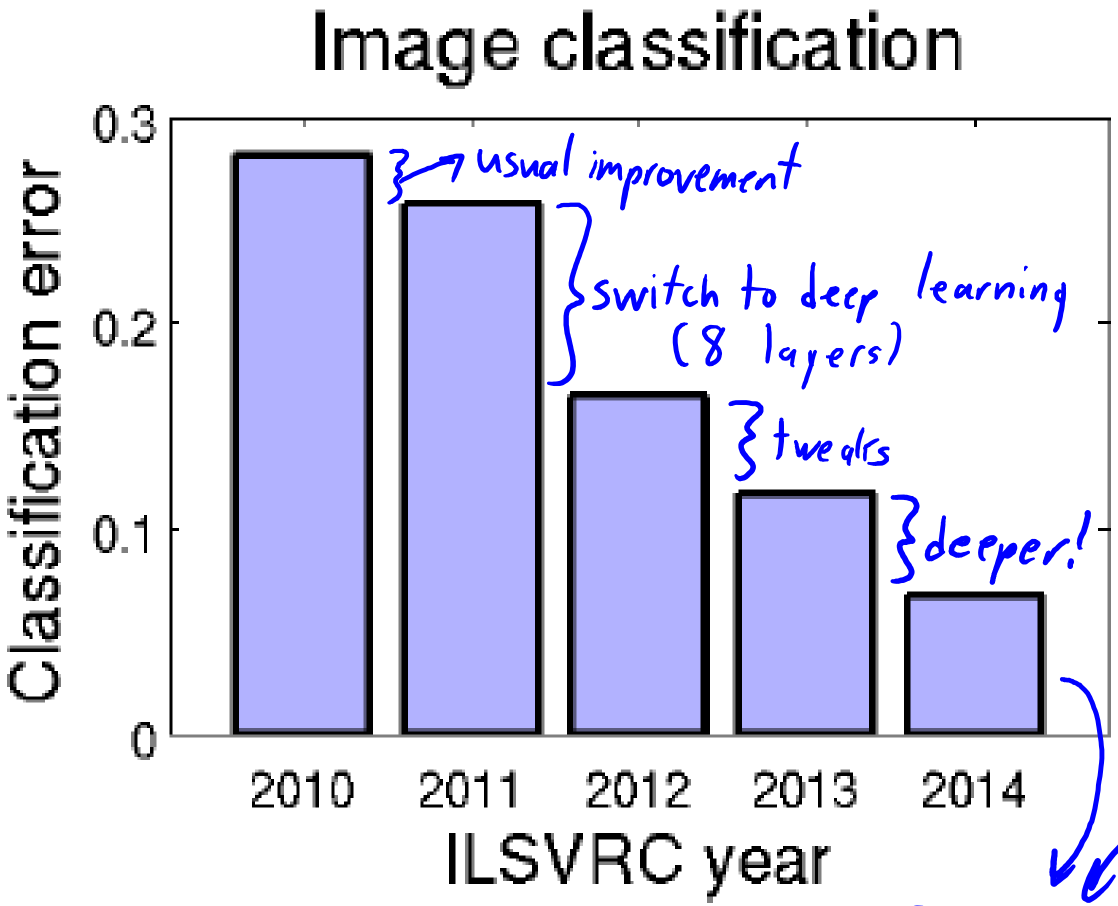


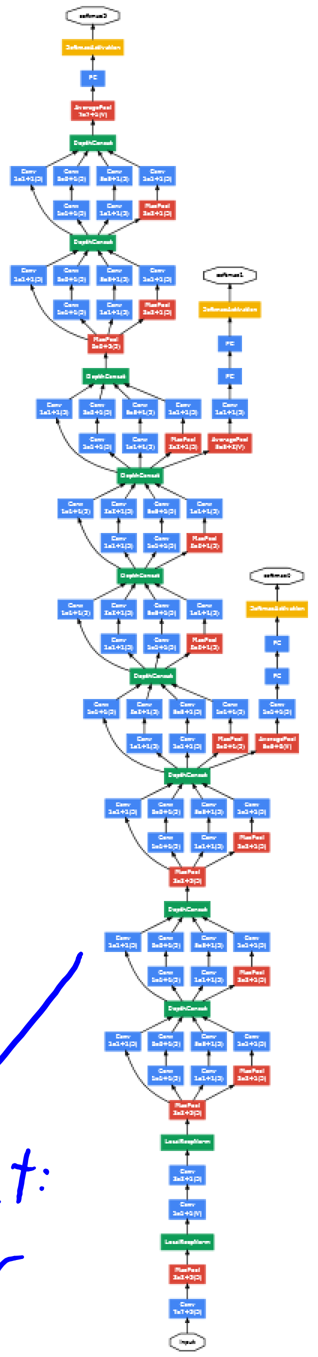
https://ischlag.github.io/2016/04/05/important-ILSVRC-achievements/
http://arxiv.org/pdf/1409.0575v3.pdf
http://arxiv.org/pdf/1409.4842v1.pdf

ImageNet Challenge

- Object detection task:
  - Single label per image.
  - Humans: ~5% error.

(a) Siberian husky (b) Eskimo dog





GoogLeNet:
6.7% error
22 layers

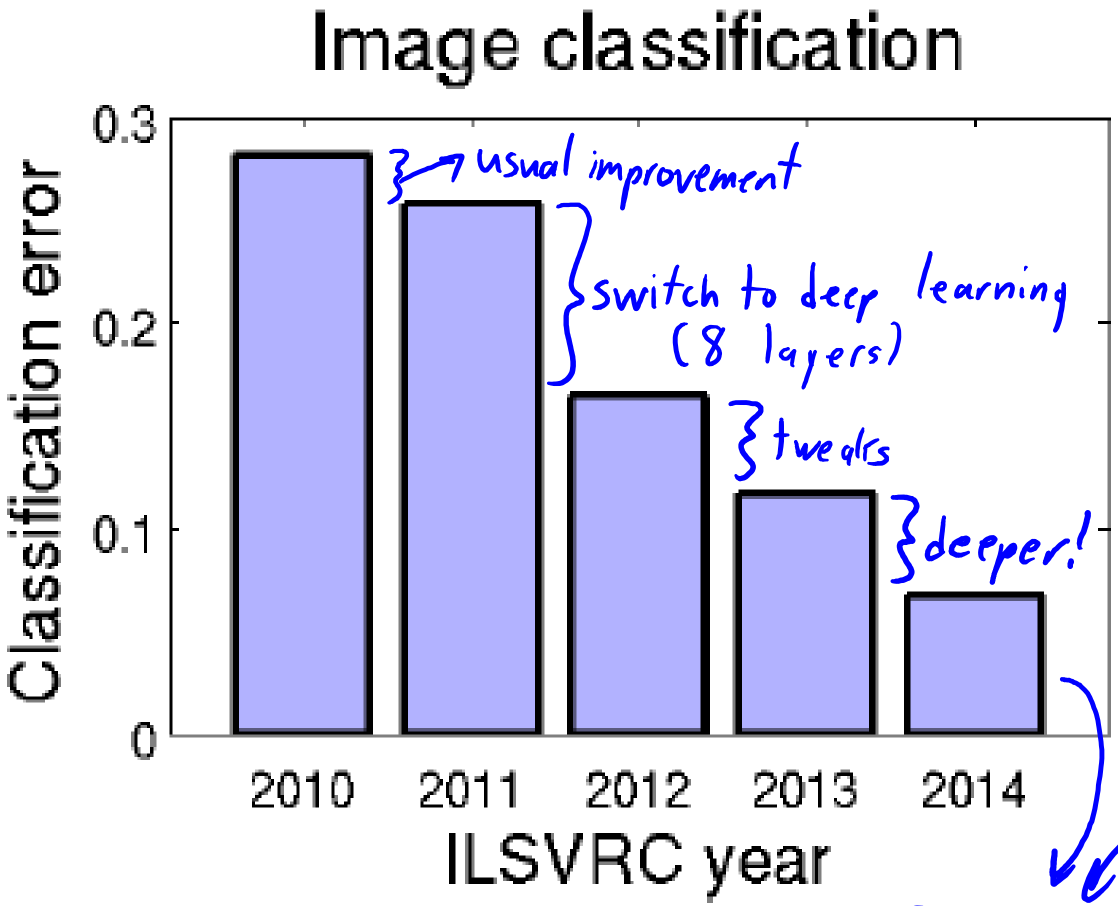
https://ischlag.github.io/2016/04/05/important-ILSVRC-achievements/
http://arxiv.org/pdf/1409.0575v3.pdf
http://arxiv.org/pdf/1409.4842v1.pdf

ImageNet Challenge

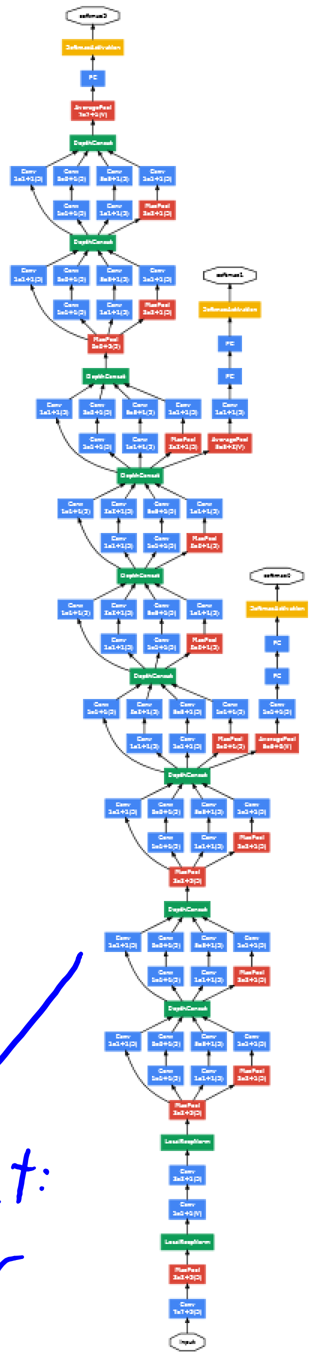
- Object detection task:
  - Single label per image.
  - Humans: ~5% error.

(a) Siberian husky (b) Eskimo dog

- 2015 winner: Microsoft
  - 3.6% error.
  - 152 layers.



GoogLeNet:
6.7% error
22 layers



https://ischlag.github.io/2016/04/05/important-ILSVRC-achievements/
http://arxiv.org/pdf/1409.0575v3.pdf
http://arxiv.org/pdf/1409.4842v1.pdf

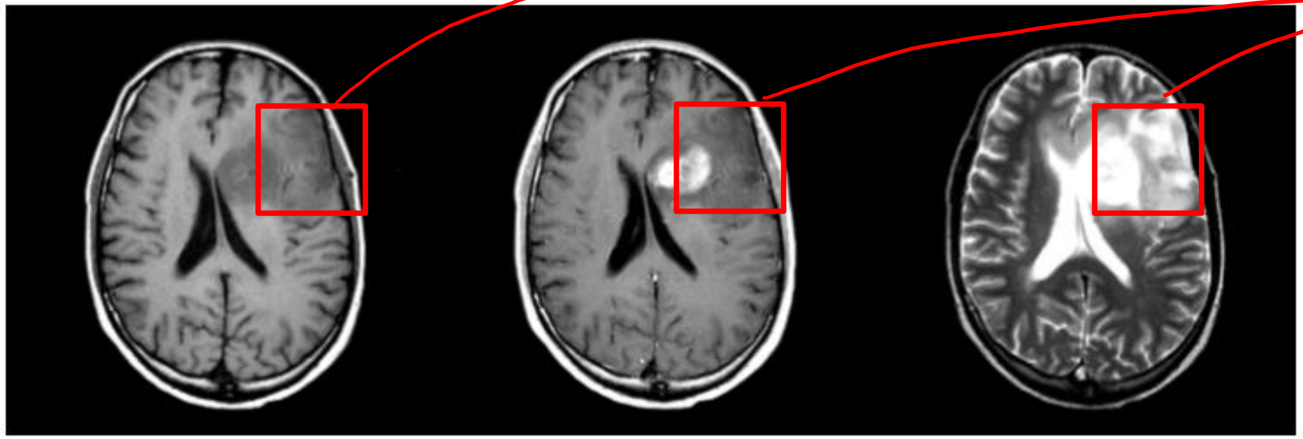
Convolutional Neural Networks

- There are many heuristics to make deep learning work:
  - Parameter initialization and data transformations.
  - Setting the step size(s) in stochastic gradient.
  - Alternative non-linear functions like ReLU.
  - L2-regularization.
  - Early stopping.
  - Dropout.
- Often, still not enough to get deep models working.

- Winning ImageNet models are all convolutional neural networks:
  - The $W^{(m)}$ are very sparse and have repeated parameters ("tied weights").
  - Drastically reduces number of parameters (and speeds up training).

Need for Context

- Want to represent spatial “context” of a pixel neighborhood.



$x_i = (\ldots , \ldots , \ldots)$

  - Standard approach uses convolutions to represent neighbourhood.
  - Usually use multiple convolutions at multiple scales.

Image Convolution Examples

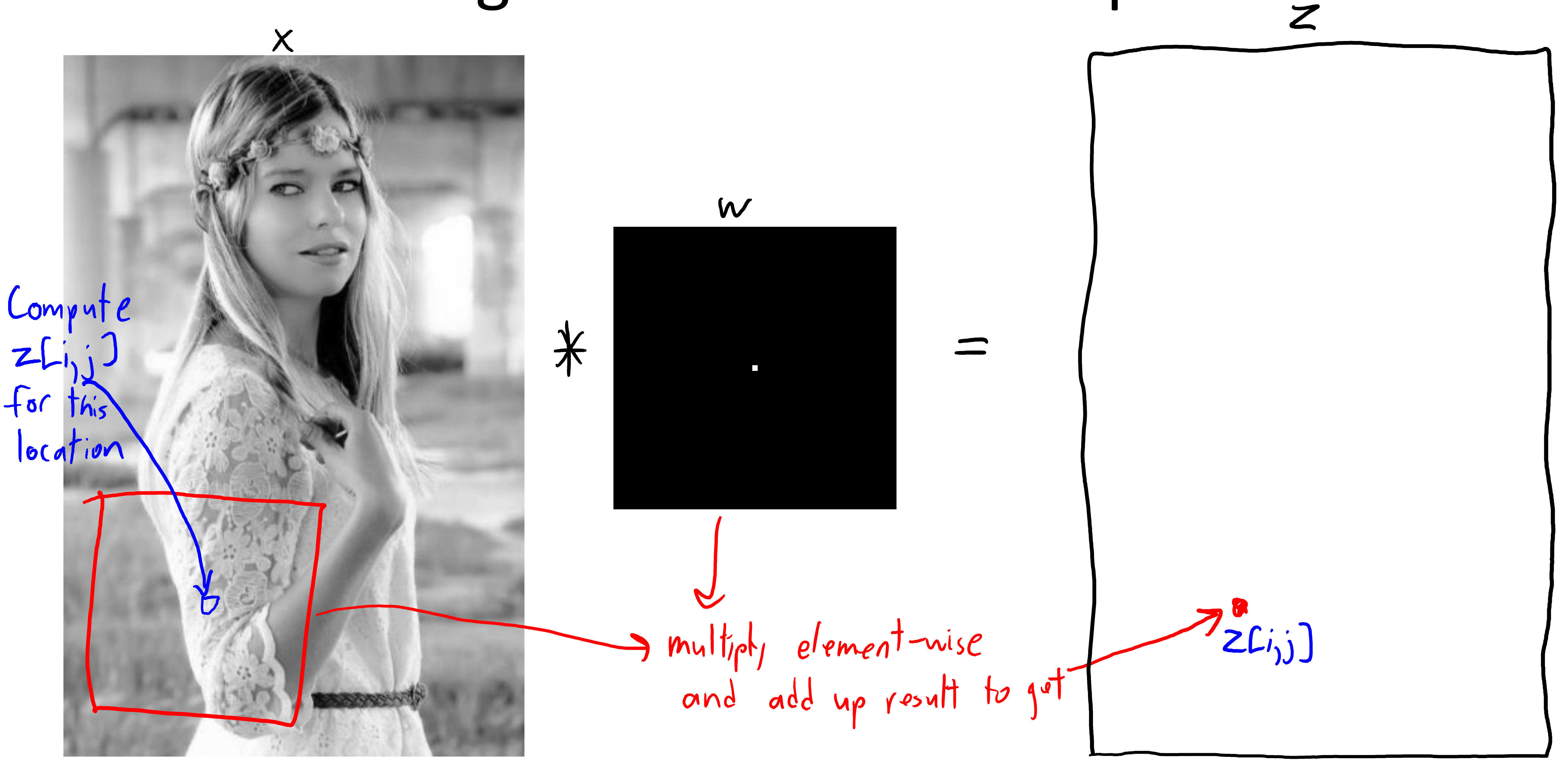
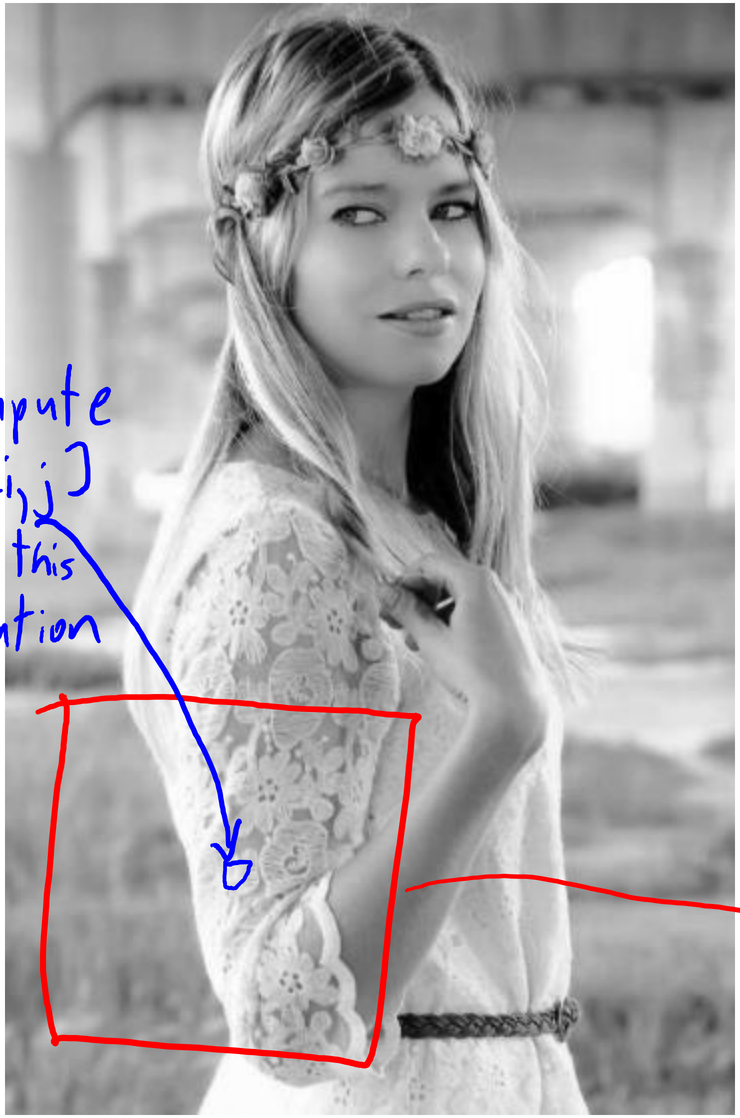


Image Convolution Examples

Identity convolution:
(zeroes with a '1' at $w_{0,0}$)

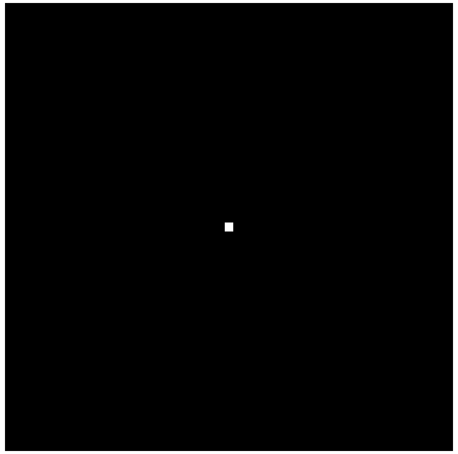
x



Compute $z[i,j]$ for this location

$*$

w



$=$

z

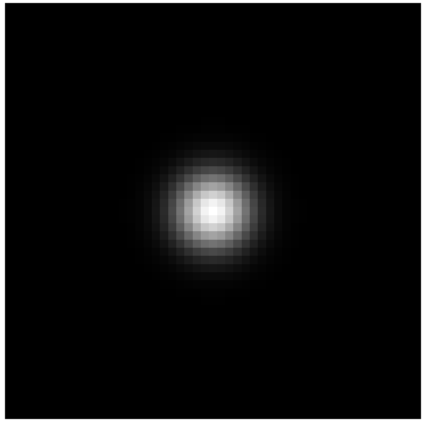
$z[i,j]$

multiply element-wise and add up result to get

Image Convolution Examples

Gaussian Convolution:

$*$



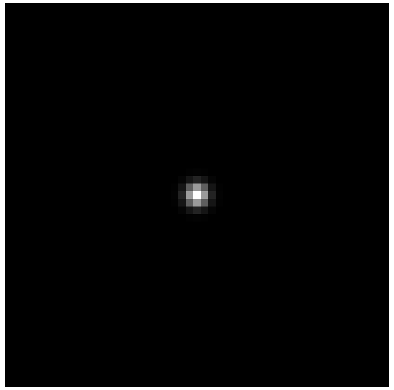
$=$

(smooths image)

Image Convolution Examples

Gaussian Convolution:

$*$



$=$

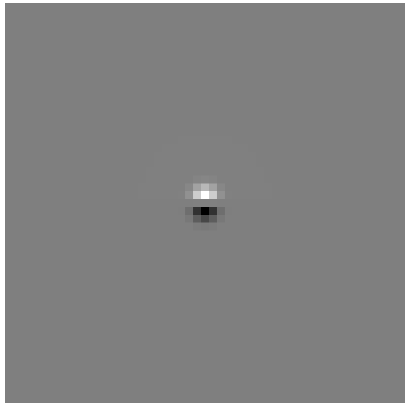
(smaller variance)

(smooths image)

Image Convolution Examples

Gabor Filter
(Gaussian multiplied by sine or cosine)

$*$



(smaller variance)

$=$

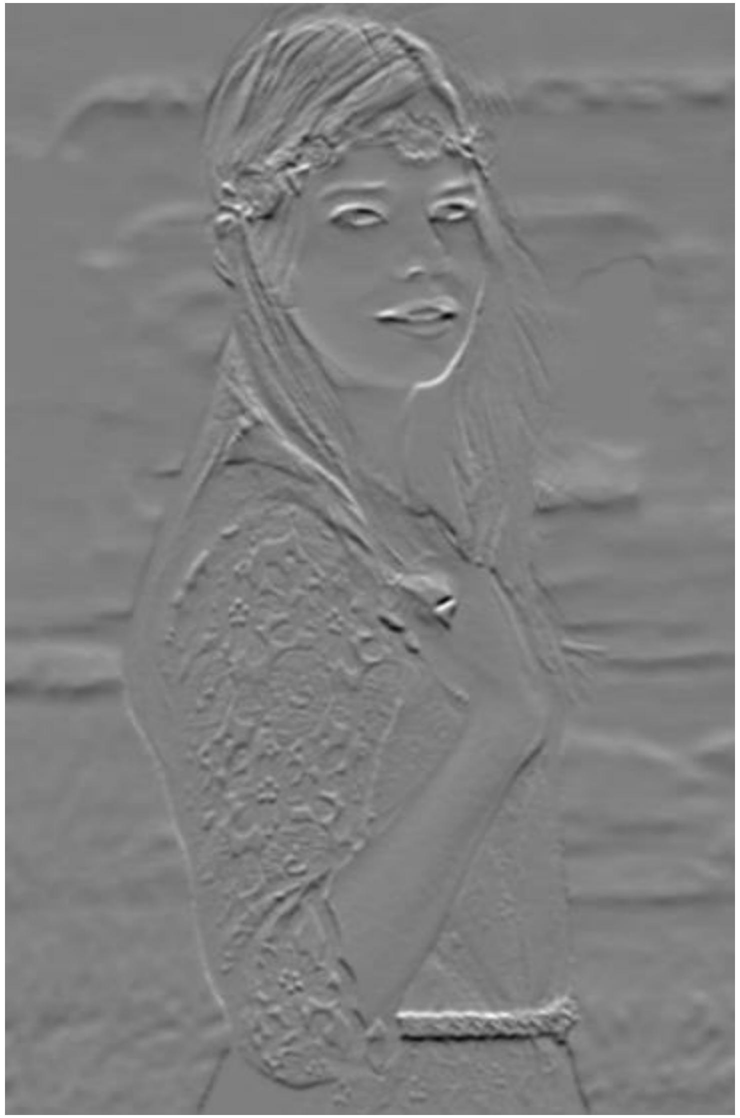
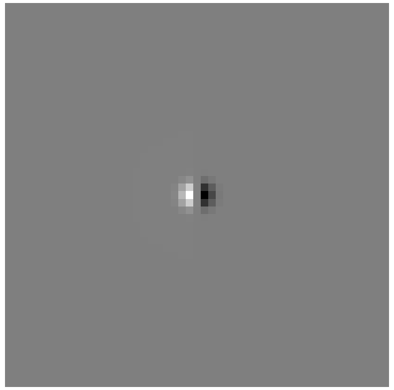


Image Convolution Examples

Gabor Filter
(Gaussian multiplied by sine or cosine)

$*$



$=$

(smaller variance)

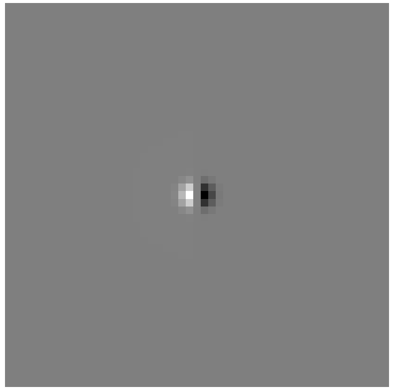
Vertical orientation

– Can obtain other orientations by rotating.

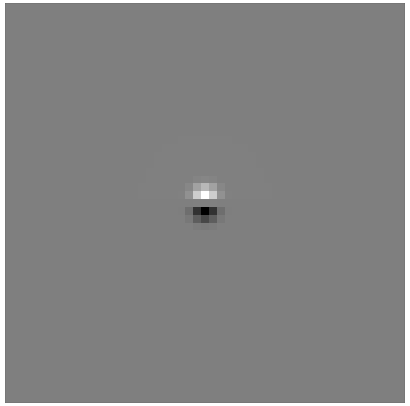
Image Convolution Examples

Max absolute value between horizontal and vertical Gabor:

$*$



$*$



maximum absolute value

"Horizontal/vertical edge detector"

3D Convolution

Represent as RGB

Can apply 3D convolutions

3D Convolution

Sharpen the blue channel.

3D Convolution

Gabor filter on each channel.

Motivation for Convolutional Neural Networks

- Consider training neural networks on 256 by 256 images.
- Each z_i in first layer has 65536 parameters (and 3x this for colour).
  - We want to avoid this huge number (due to storage and overfitting).

- Key idea: make Wx_i act like convolutions (to make it smaller):
  - Each row of W only applies to part of x_i.
  - Use the same parameters between rows.

$$W_1 = [0 \quad 0 \quad 0 \; \text{——} \; w \; \text{——} \; 0 \quad 0 \quad 0]$$

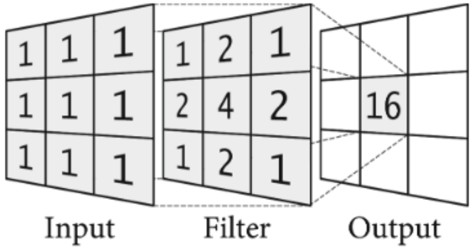
$$W_2 = [0 \; \text{——} \; w \; \text{——} \; 0 \quad 0 \quad 0 \quad 0 \quad 0]$$

2D Convolution as Matrix Multiplication

- 2D convolution:
  - Signal ‘x’, filter ‘w’, and output ‘z’ are now all images/matrices:

$$x \quad * \quad w \quad = \quad z$$

$$\begin{bmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$

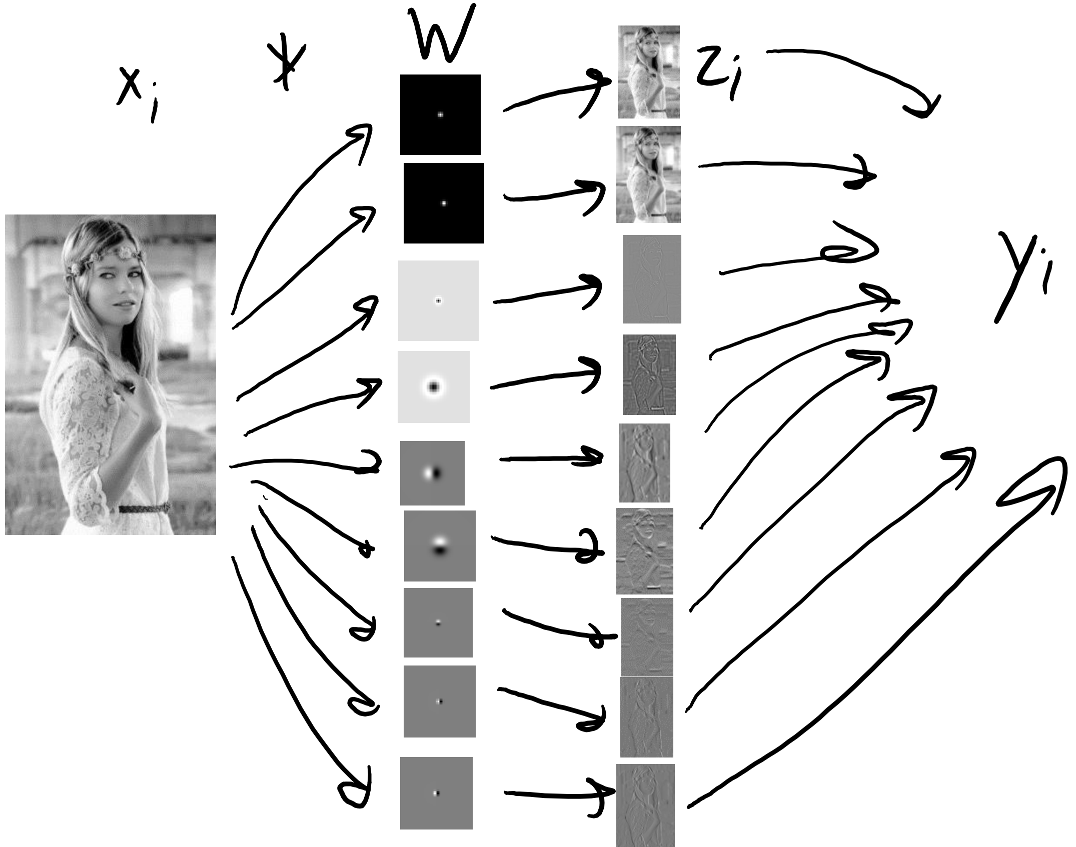


  - Vectorized ‘z’ can be written as a matrix multiplication with vectorized ‘x’:

$$W = \left[\begin{array}{cccccccc|cccccccc|cccccccc|c}
-2 & -1 & 0 & 0 & 0 & 0 & \cdots & 0 & -1 & 0 & 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 & 2 & 0 & 0 & 0 & \cdots & 0 & 0\,0\cdots \\
0 & -2 & -1 & 0 & 0 & 0 & \cdots & 0 & 0 & -1 & 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 & 1 & 2 & 0 & 0 & \cdots & 0 & 0\,0\cdots \\
 & & \vdots & & & \cdots & 2 & -1\;0 & 0 & 0 & 0 & \cdots & & & & -1\;0\;1 & 0 & 0 & 0 & \cdots & & & & 0\;1\;2 & 0\,0\cdots \\
0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 2 & -1 & 0 & 0 & 0 & 0 & \cdots & 0 & -1 & 0 & 1 & 0 & 0 & 0 & \cdots & 0 & 0\,1\,2 \\
0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 2 & -1 & 0 & 0 & 0 & \cdots & 0 & 0 & -1 & 0 & 1 & 0 & 0 & \cdots & 0 & 0\,0\,1 \\
 & & \vdots & & & & & & & & \vdots & & & & & & & & \vdots & & & & & &
\end{array}\right]$$

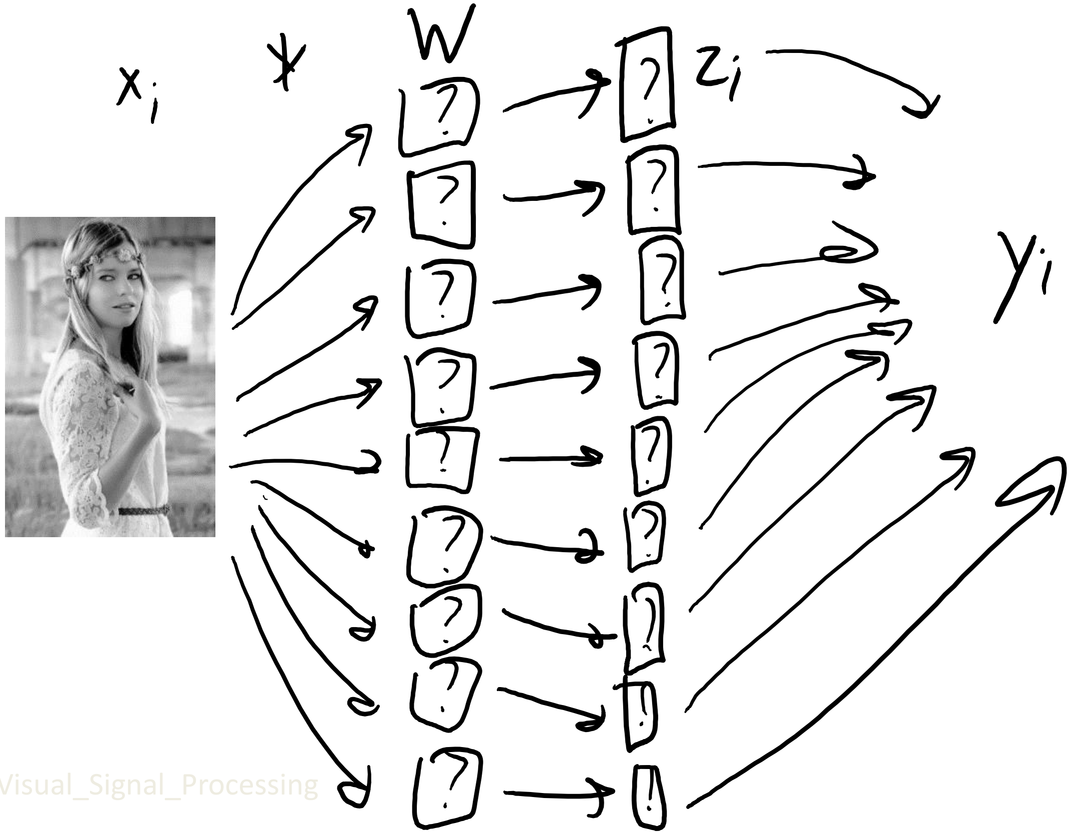
Using Fixed Convolutions

- Classic approach uses fixed convolutions as features:
  - Usually have different types/variances/orientations.
  - Can do subsampling or taking maxes across locations/orientations/scales.

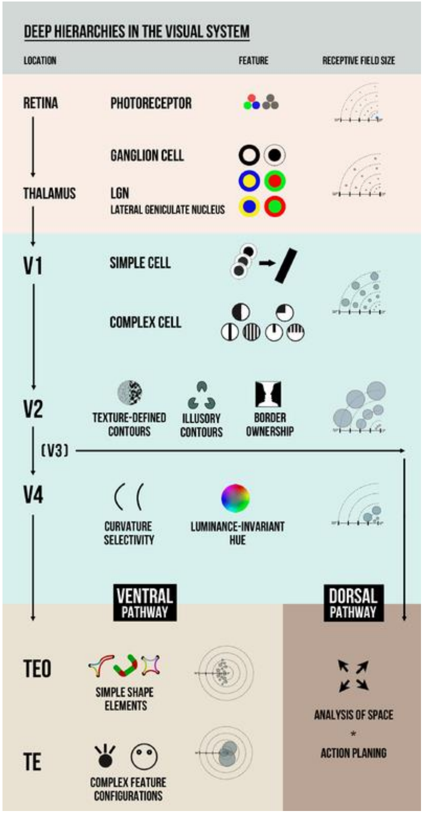


Learning Convolutions

- Convolutional neural networks learn the features:
  - Learning ‘W’ and ‘w’ automatically chooses types/variances/orientations.
  - Can do multiple layers of convolution to get deep hierarchical features.



https://en.wikibooks.org/wiki/Sensory_Systems/Visual_Signal_Processing



Convolutional Neural Networks

- Convolutional neural networks classically have 3 layer "types":
  - Fully connected layer: usual neural network layer with unrestricted W.

$$W^{(m)} = \begin{bmatrix} \text{———} & w_1^{(m)} & \text{———} \\ \text{———} & w_2^{(m)} & \text{———} \\ & \vdots & \\ \text{———} & w_k^{(m)} & \text{———} \end{bmatrix}$$

Convolutional Neural Networks

- Convolutional neural networks classically have 3 layer "types":
  - Fully connected layer: usual neural network layer with unrestricted W.
  - Convolutional layer: restrict W to results of several convolutions.

1D example

distance between centers of convolution is called "stride"

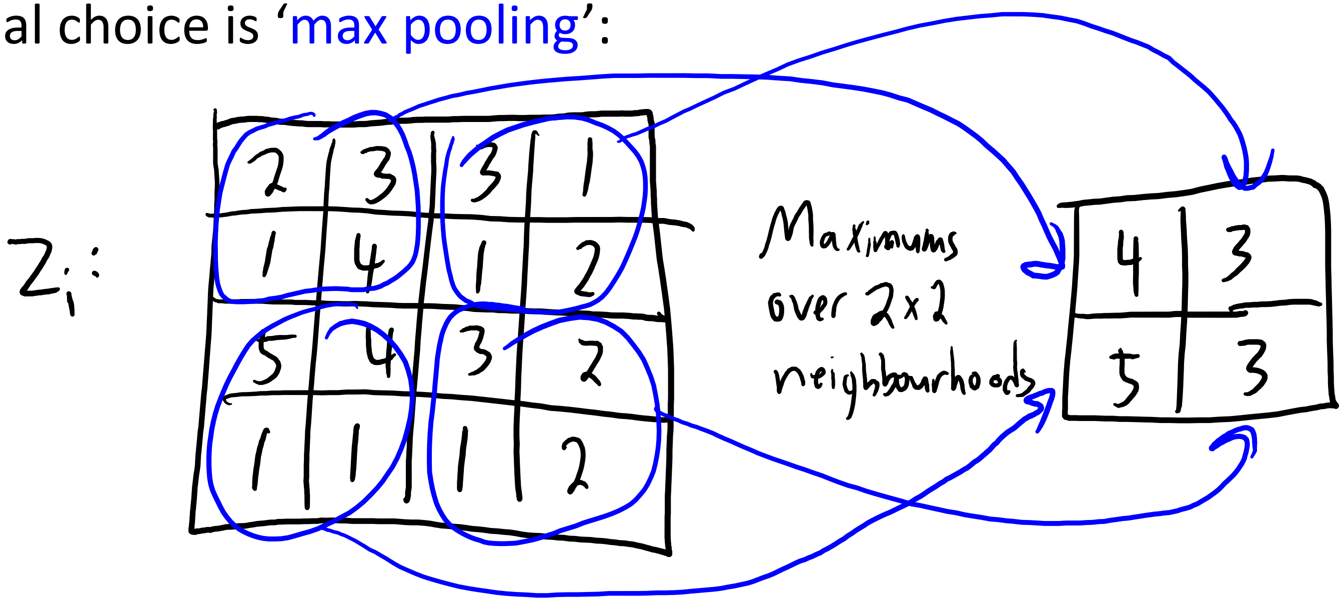
$$W^{(m)} = \begin{bmatrix} \text{——} & W_1^{(m)} & \text{——} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \text{——} & W_1^{(m)} & \text{——} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \text{——} & W_1^{(m)} & \text{——} \\ \text{——} & W_2^{(m)} & \text{——} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \text{——} & W_2^{(m)} & \text{——} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \text{——} & W_2^{(m)} & \text{——} \end{bmatrix}$$

Same $W_1^{(m)}$ used across multiple rows.

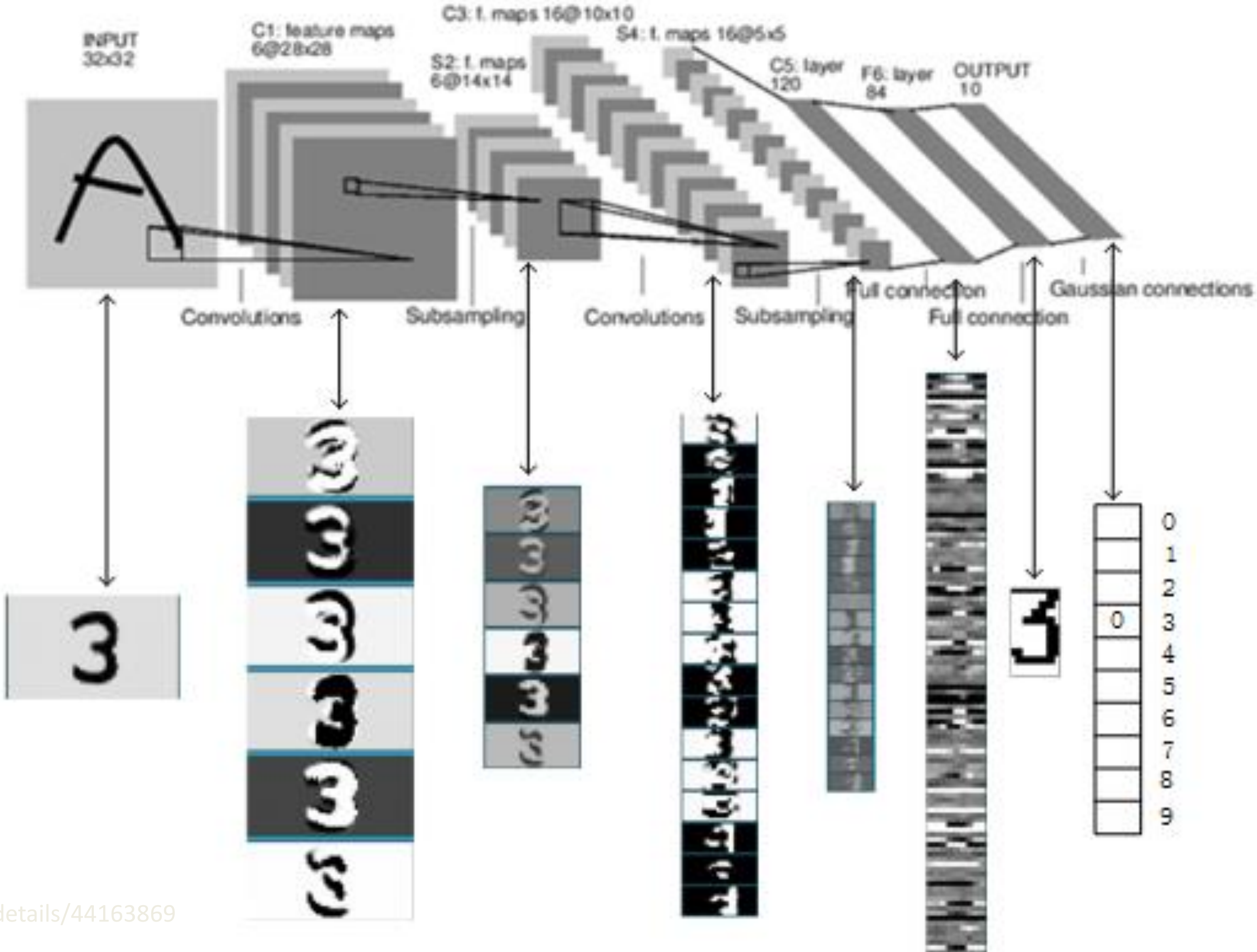
Sparse and small number of parameters

Convolutional Neural Networks

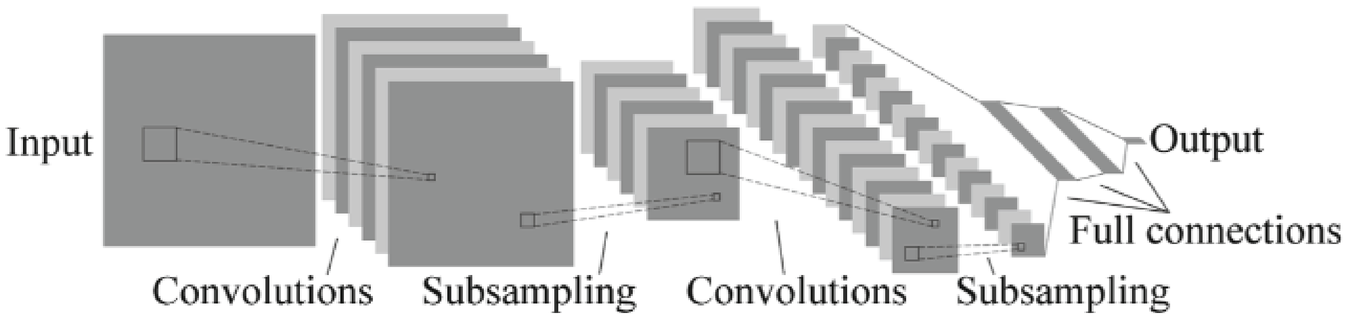
- Convolutional neural networks classically have 3 layer "types":
  - Fully connected layer: usual neural network layer with unrestricted W.
  - Convolutional layer: restrict W to results of several convolutions.
  - Pooling layer: downsamples result of convolution.
    - Can add invariances or just make the number of parameters smaller.
    - Usual choice is 'max pooling':



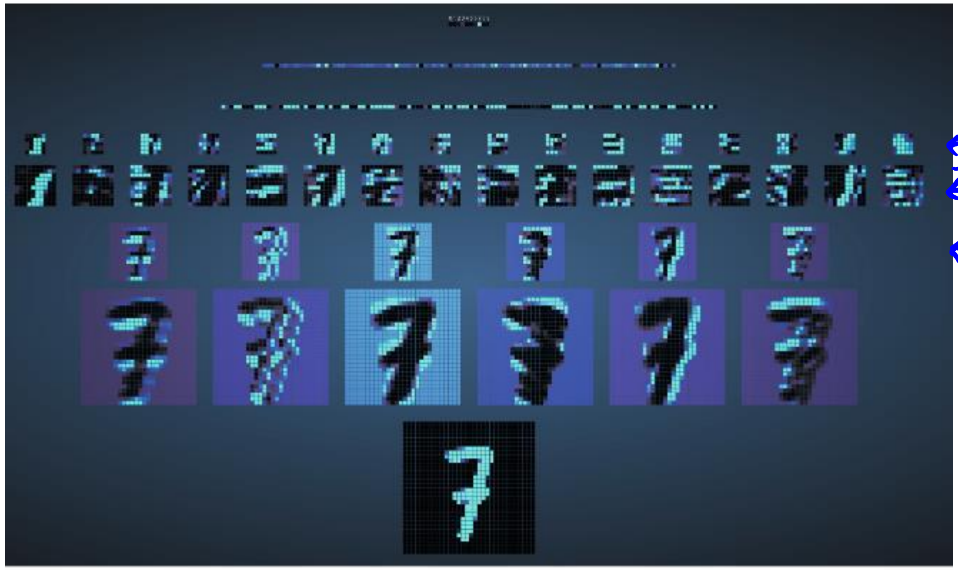
LeNet for Optical Character Recognition (1998)



LeNet for Optical Character Recognition (1998)



- Visualizing the “activations” of the layers:
  - http://scs.ryerson.ca/~aharley/vis/conv



softmax
2 "fully-connected"
max pooling
3D convolutions
max pooling
2D convolutions

http://scs.ryerson.ca/~aharley/vis/harley_vis_isvc15.pdf

Summary

- Neural networks learn features for supervised learning.
- Backpropagation: message-passing algorithm to compute gradient.
- Conditional neural fields combine CRFs with deep learning.
  - Latent dynamic variants use a “hidden” CRF.
- Convolutional neural networks:
  - Restrict W(m) matrices to represent sets of convolutions.
  - Often combined with max (pooling).

- Next time: modern convolutional neural networks and applications.
  - Image segmentation, depth estimation, image colorization, artistic style.