Can we design a network that takes account of the image structure?

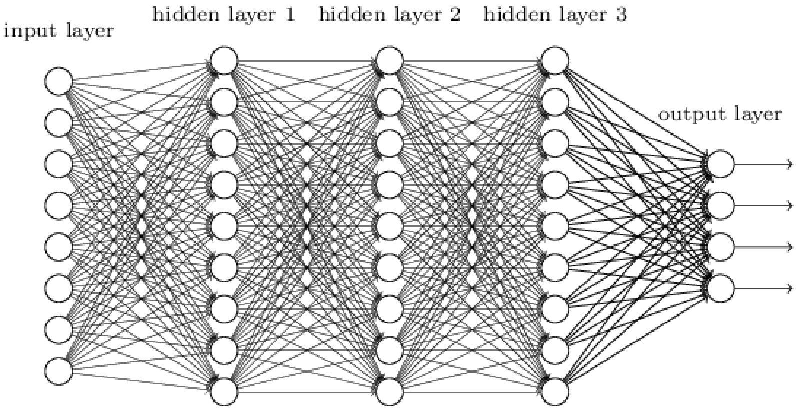
(And learns invariances...)

Convolutional Neural Networks

Steve Renals

Machine Learning Practical — MLP Lecture 7
4 November 2015

Recap: Multi-layer network for MNIST



(image from: Michael Nielsen, *Neural Networks and Deep Learning*, http://neuralnetworksanddeeplearning.com/chap6.html)

How can we make this better?

On MNIST, we can get about 2% error (or even better) using these kind of networks, but

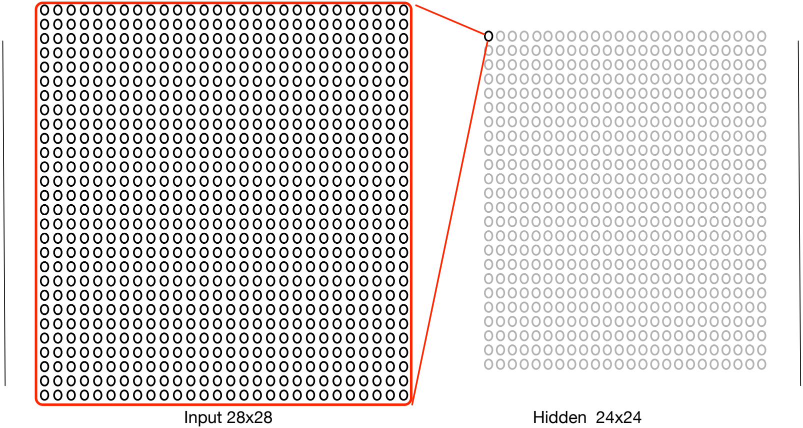
- They ignore the spatial (2-D) structure of the input images – unroll each 28x28 image into a 784-D vector
- Each hidden unit looks at the units in the layer below, so pixels that are spatially separate are treated the same way as pixels that are adjacent
- There is no obvious way for networks to learn the same features (e.g. edges) at different places in the input image

Convolutional networks

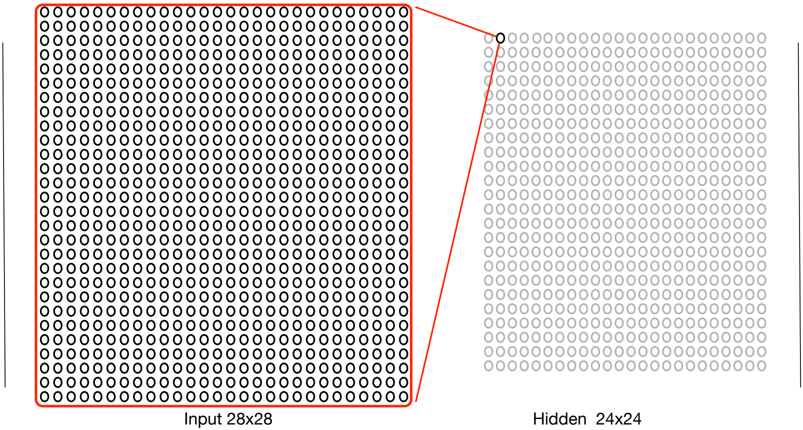
Convolutional networks address these issues through

- **Local receptive fields** in which hidden units are connected to local patches of the layer below,
- **Weight sharing** which enables the construction of feature maps,
- **Pooling** which condenses information from the previous layer.

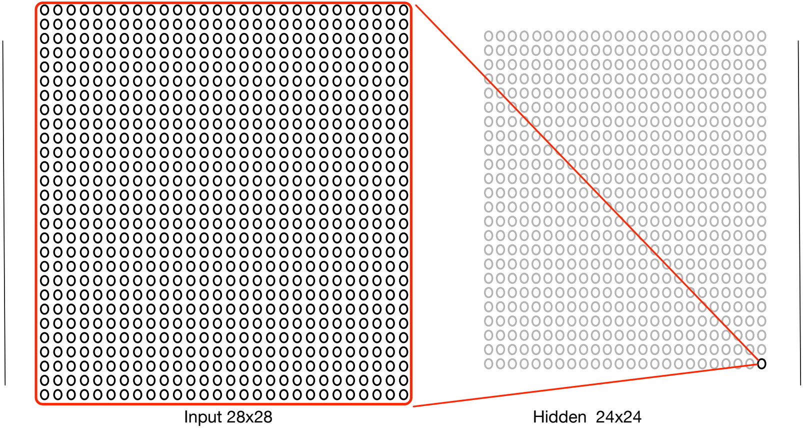
Fully connected hidden layer – 576 hidden units



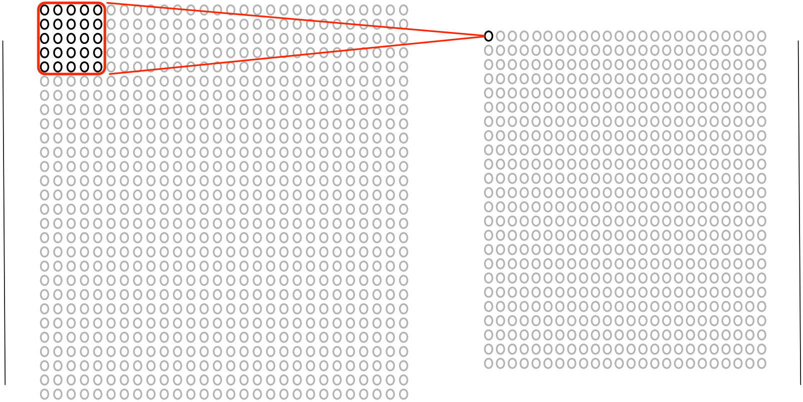
Fully connected hidden layer – 576 hidden units



Fully connected hidden layer – 576 hidden units



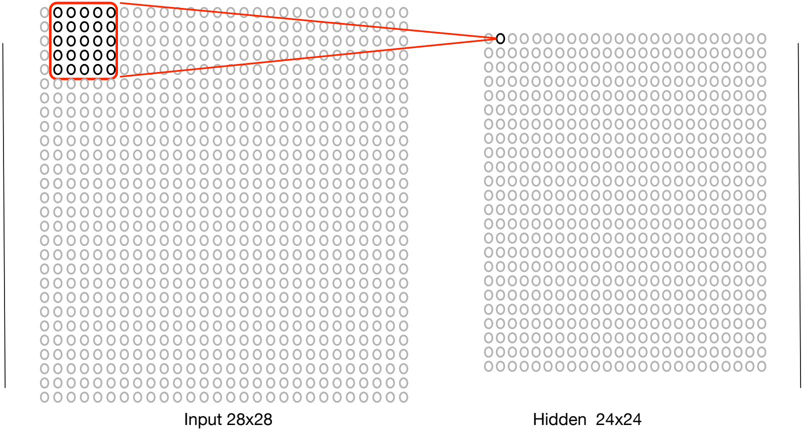
Local receptive fields – 24x24 hidden units



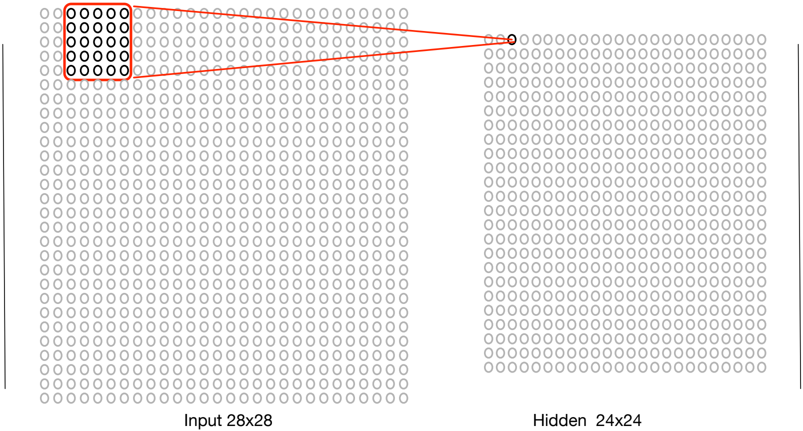
Input 28x28

Hidden 24x24

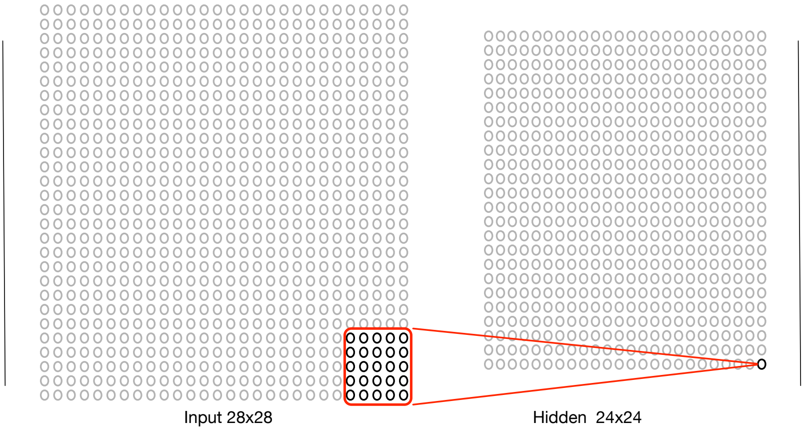
Local receptive fields – 24x24 hidden units



Local receptive fields – 24x24 hidden units



Local receptive fields – 24x24 hidden units



Local receptive fields

- Each hidden unit is connected to a small $(m \times m)$ region of the input space – the *local receptive field*
- If we have a $d \times d$ input space, then we have $(d - m + 1) \times (d - m + 1)$ hidden unit space
- Each hidden unit extracts a feature from "its" region of input space
- Here the receptive field "stride length" is 1, it could be larger

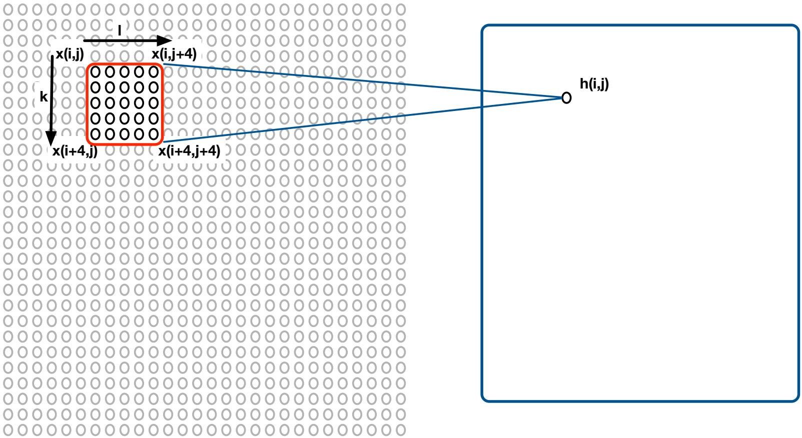
Shared weights

- Constrain each hidden unit $h_{i,j}$ to extract the same feature by sharing weights across the receptive fields
- For hidden unit $h(i,j)$

$$h_{i,j} = \text{sigmoid}(\sum_{k=0}^{m-1} \sum_{\ell=0}^{m-1} w_{k,\ell} x_{i+k,j+\ell} + b)$$

  where $w_{k,\ell}$ are elements of the shared $m \times m$ weight matrix $\mathbf{w}$, b is the shared bias, and $x_{i+k,j+\ell}$ is the input at $i+k, j+\ell$
- We use k and l to index into the receptive field, whose top left corner is at $x_{i,j}$

Shared weights & Receptive Fields



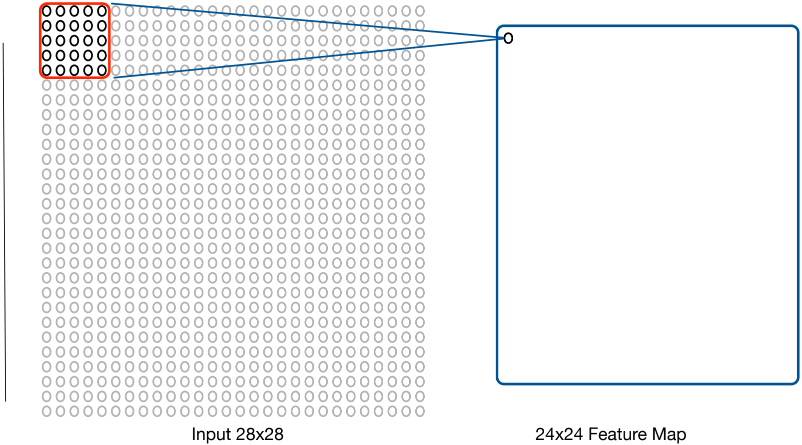
Input 28x28

24x24 Feature Map

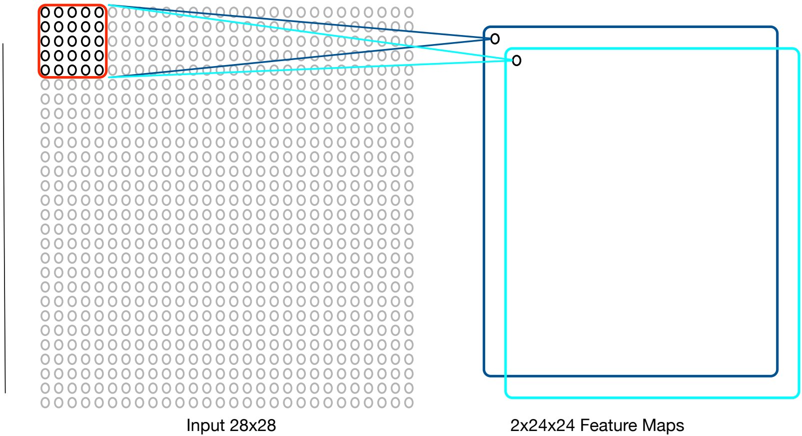
Feature Maps

- We scan the $m \times m$ feature detector across the image, so we obtain a map of where the feature corresponding to the shared weight matrix (**kernel**) occurs in the image - **Feature map**
- The feature map encodes **translation invariance** – it doesn't matter where an digit image is in the input we can extract the same features
- **Multiple feature maps** – a hidden layer can consist of F different feature maps – in this case $F \times 24 * 24$ units in total

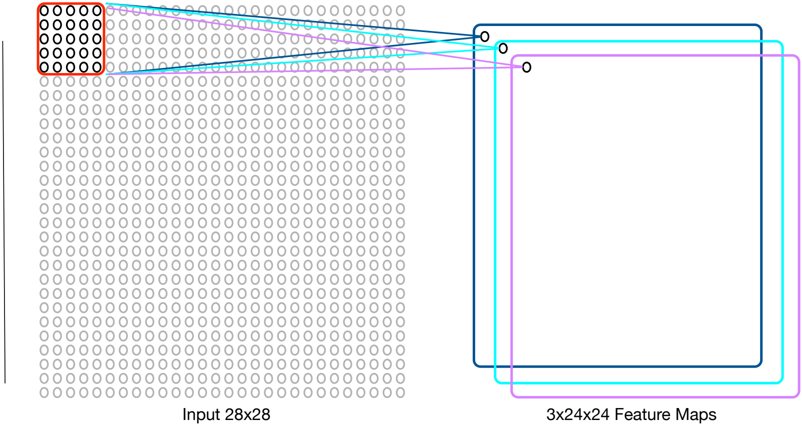
Feature Maps



Feature Maps



Feature Maps



Weights and Connections

Consider an MNIST hidden layer with a single feature maps, using a 5x5 kernels (so a 24x24 feature map):

- Number of connections per feature map:
  $24 \times 24 \times 5 \times 5 = 14,400$ connections
  $24 \times 24 = 576$ biases
- But since weights are shared, we have
  $5 \times 5 = 25$ weights
  1 bias

Now consider the case where we have 40 feature maps. We will have 1,000 weights (and 40 biases), but 576,000 (+ 23,040) connections!

In comparison the 800 hidden unit MLP from the coursework 1 has $784 \times 800 + 800 = 628,000$ input-hidden weights

Learning image kernels

Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	

https://en.wikipedia.org/wiki/Kernel_(image_processing)

- Image kernels have been designed and used for feature extraction in image processing (e.g. edge detection)
- However, we can learn multiple kernel functions (feature maps) by optimising the network cost function
- Automating feature engineering

Convolutional Layer

- This type of feature map is often called a **Convolutional layer**
- We can write the feature map hidden unit equation:

$$h_{i,j} = \text{sigmoid}\Big(\sum_{k=1}^{m}\sum_{\ell=1}^{m} w_{k,\ell}\, x_{i+k,j+\ell} + b\Big)$$

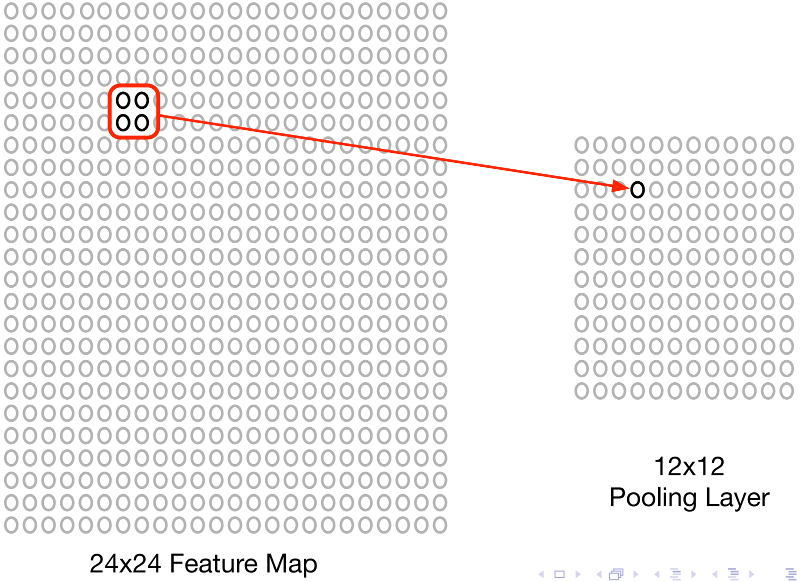
as

$$h = \text{sigmoid}(w * x + b)$$

$*$ is called a convolution in signal processing

(*Note for signal processing experts*: The way a 2D convolution is defined in signal and image processing, we would need "flip" the $m \times m$ weight matrix (reflect horizontally and vertically). We have been using a cross-correlation (i.e. "unflipped"). In common with most of the Conv Nets literature we shall use convolution to describe both cases. As long as you are consistent it is not important which you apply, for our purposes.)

Pooling (subsampling)



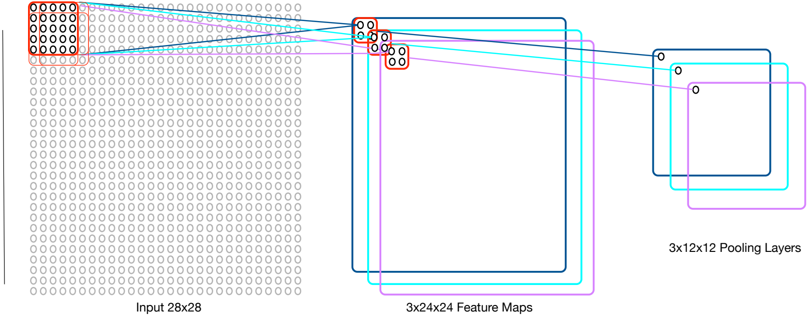
Pooling

- Pooling or subsampling takes a feature map and reduces it in size – e.g. by transforming a set of 2x2 regions to a single unit
- Pooling functions
  - Max-pooling – takes the maximum value of the units in the region (c.f. maxout)
  - L_p-pooling – take the L_p norm of the units in the region:

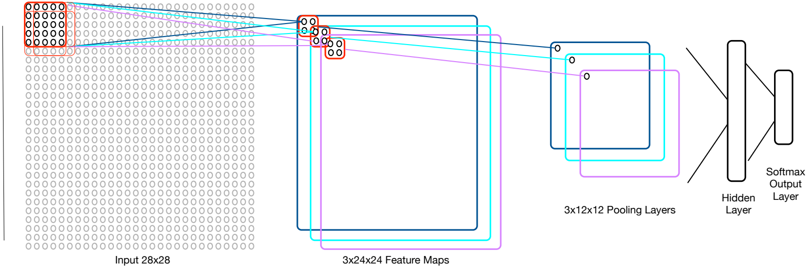
$$h' = \left(\sum_{i \in \text{region}} h_i^p \right)^{1/p}$$

  - Average- / Sum-pooling – takes the average / sum value of the pool
- Information reduction – removes precise location information for a feature
- Apply pooling to each feature map separately

Putting it together – convolutional+pooling layer



ConvNet – Convolutional Network



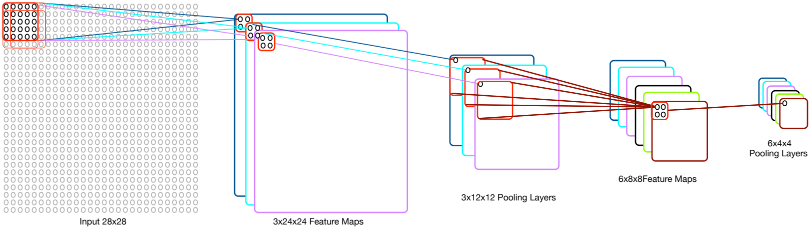
Simple ConvNet:

- Convolutional layer with max-pooling
- Final fully connected hidden layer (no sharing weight)
- Softmax output layer
- With 20 feature maps and a final hidden layer of 100 hidden unit:
  $20 \times (5 \times 5 + 1) + 20 \times 12 \times 12 \times 100 + 100 + 100 \times 10 + 10 = 289,630$ weights

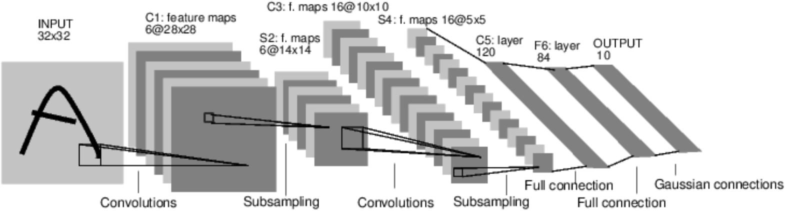
Multiple input images

- If we have a colour image, each pixel is defined by 3 RGB values – so our input is in fact 3 images (one R, one G, and one B)
- If we want stack convolutional layers, then the second layer needs to take input from all the feature maps in the first layer
- **Local receptive fields across multiple input images**
- In a second convolutional layer (C2) on top of 20 12×12 feature maps, each unit will look at $20 \times 5 \times 5$ input units(combining 20 receptive fields each in the same spatial location)
- Typically do not tie weights across feature maps, so each unit in C2 has $20 \times 5 \times 5 = 500$ weights, plus a bias. (Assuming a 5×5 kernel size)

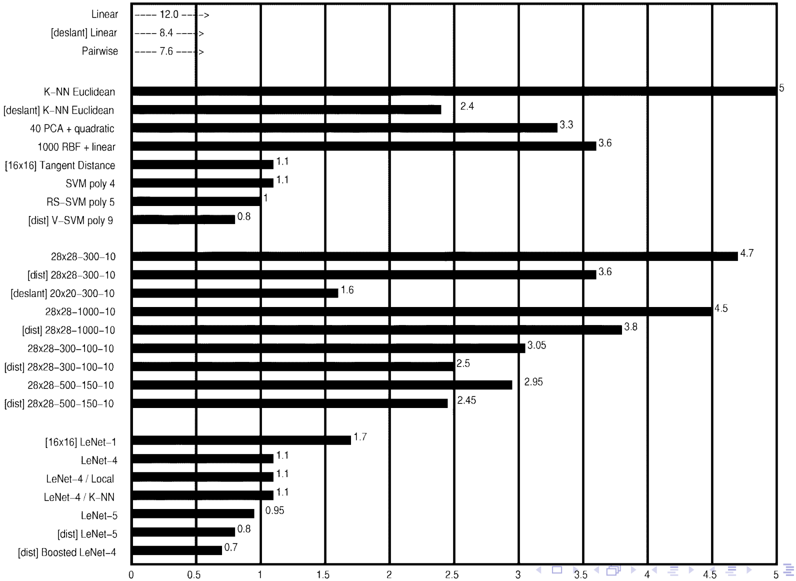
Stacking convolutional layers



Example: LeNet5 (LeCun et al, 1997)



MNIST Results (1997)



Training Convolutional Networks

- Train convolutional networks with a straightforward but careful application of backprop / SGD
- Exercise: prior to the next lecture, write down the gradients for the weights and biases of the feature maps in a convolutional network. Remember to take account of weight sharing.
- Next lecture: implementing convolutional networks: how to deal with local receptive fields and tied weights, computing the required gradients...
- Coursework 2 will involve implementing and testing convolutional networks

Summary

- Convolutional networks include local receptive fields, weight sharing, and pooling leading to:
  - Modelling the spatial structure
  - Translation invariance
  - Local feature detection
- Reading:

  Michael Nielsen, *Neural Networks and Deep Learning* (ch 6)

  http://neuralnetworksanddeeplearning.com/chap6.html

  Yann LeCun et al, "Gradient-Based Learning Applied to Document Recognition", *Proc IEEE*, 1998.

  http://dx.doi.org/10.1109/5.726791

  Yoshua Bengio et al, *Deep Learning* (ch 9)

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