Apprentissage, réseaux de neurones et modèles graphiques (RCP209)
Neural Networks and Deep Learning

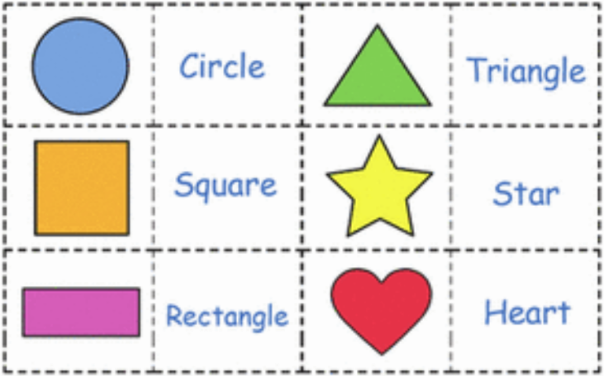
Nicolas Thome
Prenom.Nom@cnam.fr
`http://cedric.cnam.fr/vertigo/Cours/ml2/`

Département Informatique
Conservatoire Nationnal des Arts et Métiers (Cnam)

Outline

Fully Connected Networks: Limitations

- Fully connected networks: no assumption on data structure
  - Structure can be learned but need lots of annoatated data
  - Prior knowlege on data structure ⇒ useful
- Example: MLP training for shape recognition from color images

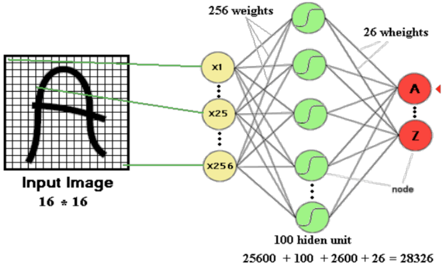


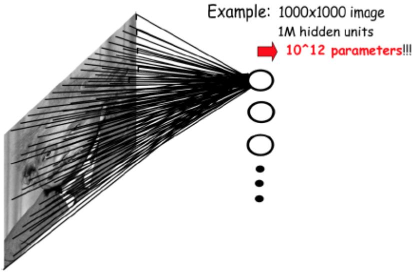
Input image encoding:

- Color (RBG) ?
- Grayscale $L = \frac{R+B+G}{3}$?

Fully Connected Networks: Limitations

- Scalability issue with Fully Connected Networks (MLP)

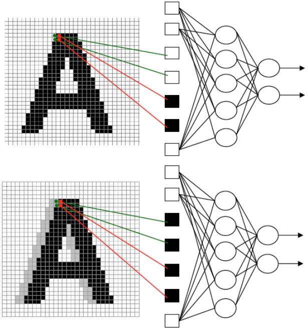


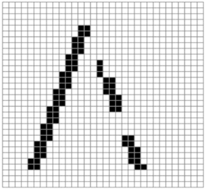


⇒ # Parameter explosion even for a single hidden layer !

Fully Connected Networks: Limitations

- Invariance and robustness to deformation (stability)
- What we expect:
    - Small deformation in input space ⇒ similar representations
    - Large transfo in input space ⇒ very dissimilar representations
- Example (image): impact of a 2 pixel shift





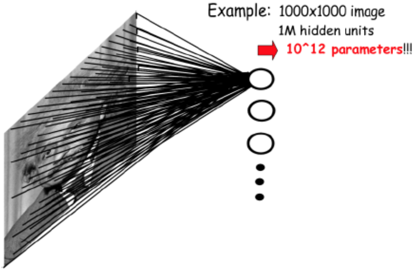
154 input change
from 2 shift left
77 : black to white
77 : white to black

@LeCun

Fully Connected Networks: Limitations

Conclusion of MLP on raw data

- Brute force connection of images as input of MLP NOT a good idea
  - No Invariance/Robustness of the representation because topology of the input data completely ignored
    $\Rightarrow$ *e.g.* indifferent to permutations of input pixel
  - Nb of weights grows largely with the size of the input image

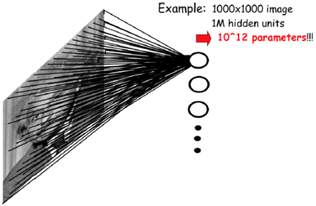


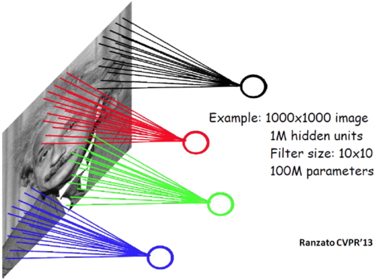
$\Rightarrow$ **How keep spatial topology?**
$\Rightarrow$ **How to limit the number of parameters?**

Taking advantage of structure: Convolution

How to limit the number of parameters?

1. Sparse connectivity: hidden unit only connected to a local patch
   - Weights connected to the patch: **filter** or **kernel**
   - Inspired by biological systems: cell only sensitive to a small sub-region of the input space (receptive field). Many cells tiled to cover the entire visual field

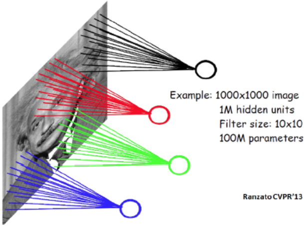


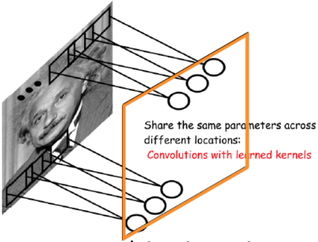


Taking advantage of structure: Convolution

How to limit the number of parameters?

2. Shared Weights
   - Hidden nodes at different locations share the same weights
     - Substantially reduces the number of parameters to learn
   - Keep spatial information in a 2D feature map (hidden layer map)





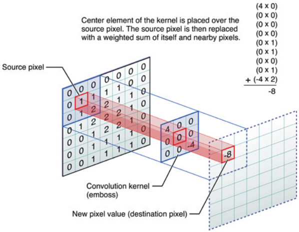
⇒ Computing responses at hidden nodes equivalent to convolving input image with a linear filter (learned)
⇒ A learned filter as a feature detector

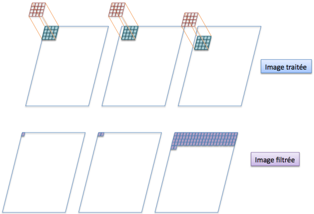
Convolution: Scalar Images

- 2D convolution with a Finite Impulse Response (FIR) h of size d (odd):

$$f'(i,j) = (f \star h)(i,j) = \sum_{n=-\frac{d-1}{2}}^{\frac{d-1}{2}} \sum_{m=-\frac{d-1}{2}}^{\frac{d-1}{2}} f(i-n, m-j) h(n,m)$$

- Simply centering filer h in pixel (x, y) $\Rightarrow$ weighted sum

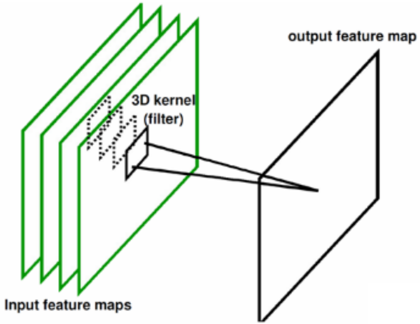


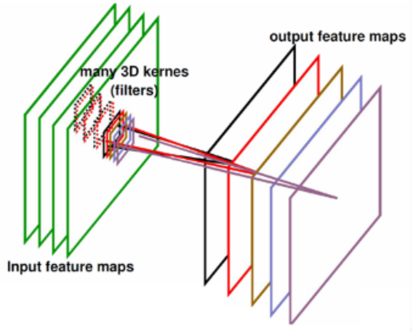


- Output for 1 filer (resp. K filters): 1 2D map (resp. K 2D maps)

Convolution: Vectorial Images (depth M)

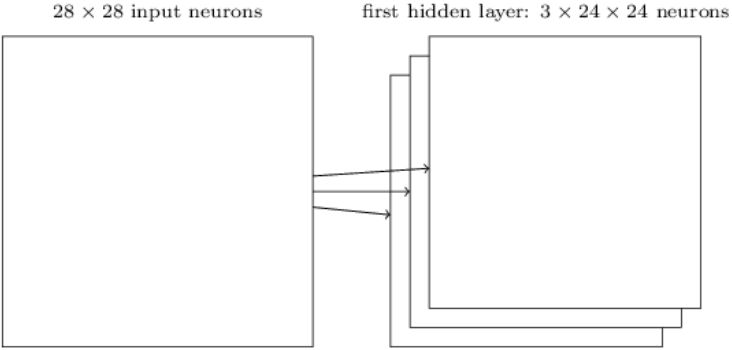
- Each filter has size $dxdxM$
- Example with $M = 3$, *e.g.* color images:





Convolutionnal Layers

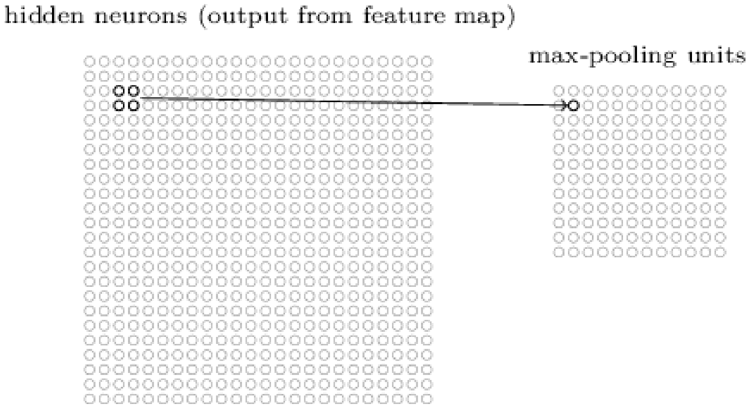
- Convolution layer $\Leftarrow$ local feature from previous layers
- Feature maps are equivariant to translation
- Followed by non-linearity (activation function)



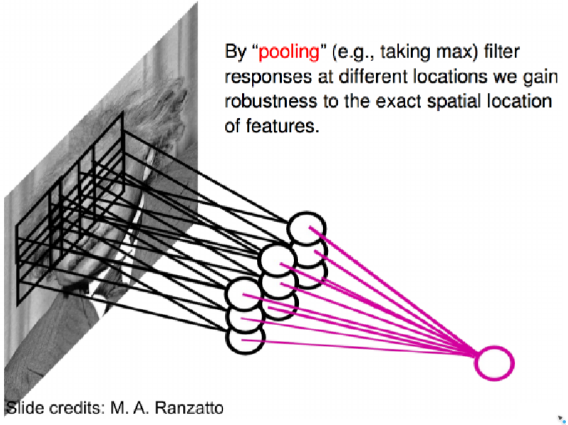
$\Rightarrow$ **How to gain (local) shift invariance ?**

Pooling Layers

- Spatial agregation for each layer
- If stride $s > 1$, sptial resolution decreases (subsampling) $\Rightarrow$ gaining invariace to local translations



Pooling Layers



Pooling Layers: Examples

Max-pooling:

$$h_j^n(x,y) = max_{\bar{x} \in N(x), \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})$$

Average-pooling:

$$h_j^n(x,y) = 1/K \sum_{\bar{x} \in N(x), \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})$$

L2-pooling:

$$h_j^n(x,y) = \sqrt{\sum_{\bar{x} \in N(x), \bar{y} \in N(y)} h_j^{n-1}(\bar{x}, \bar{y})^2}$$

L2-pooling over features:

$$h_j^n(x,y) = \sqrt{\sum_{k \in N(j)} h_k^{n-1}(x, y)^2}$$

Slide credits: M. A. Ranzatto

Convolutional Neural Networks (ConvNets)

- An elementary block: Convolution + Non linearity + pooling
- Stack several blocks: Convolutional Neural Networks (ConvNets)

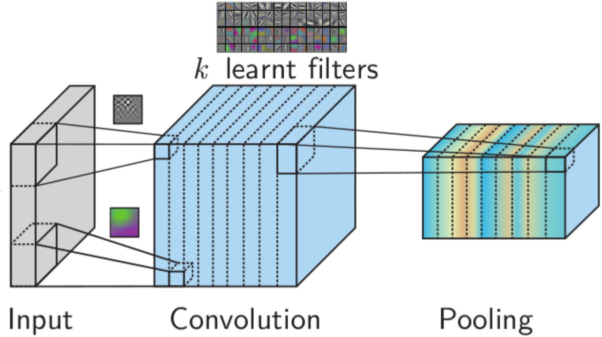


Figure : Important building blocks in CNN

Convolutional Neural Networks (ConvNets)

- Generally, Feature maps stacked together at one point ⇒ fully connected layers

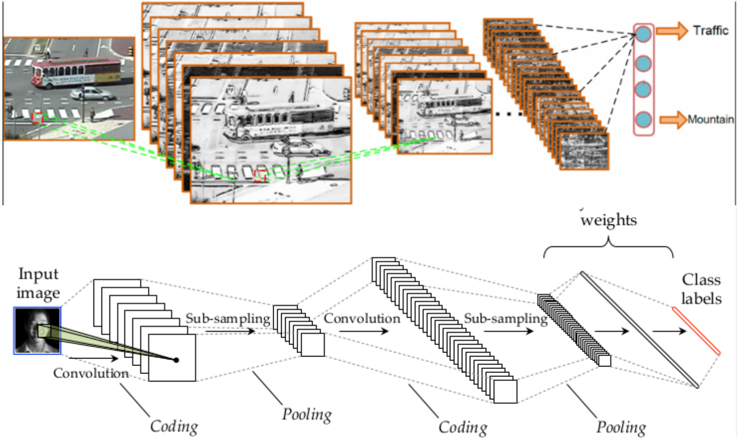
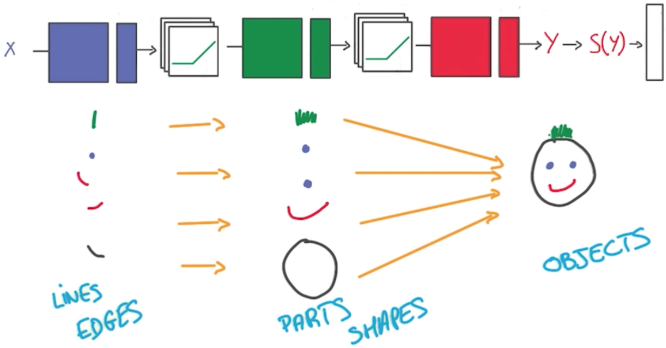


Figure : Important building blocks in CNN

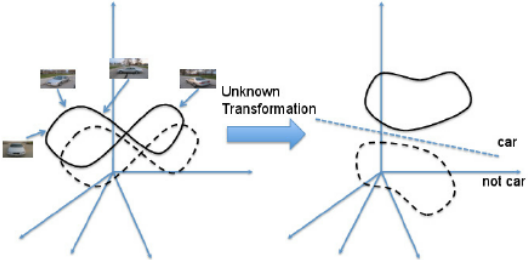
ConvNets: Conclusion

- Crucial step for tacking advantage of structure ⇒ local processing
- Useful for many data types and applications:
  - Low-level signal, *e.g.* image, audio (speech, music)
  - More semantic data, *e.g.* modern text embedding (`word2vec`) or RNN
- Block [Convolution + Non linearity + pooling] intuitive for modeling hierarchical information extraction

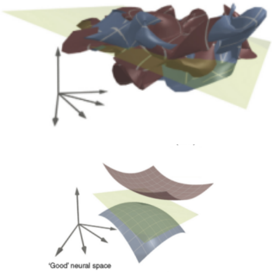


ConvNets and Manifold Untangling

Manifold Untangling

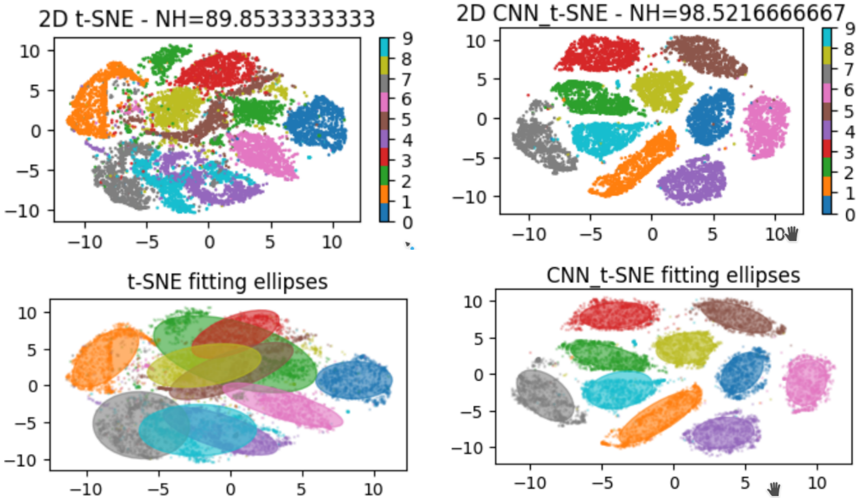


Credit: DiCarlo



ConvNets and Manifold Untangling

Convnet: 2 conv and 1 FC layer: latent space *vs* input space visu



Input space

Latent space

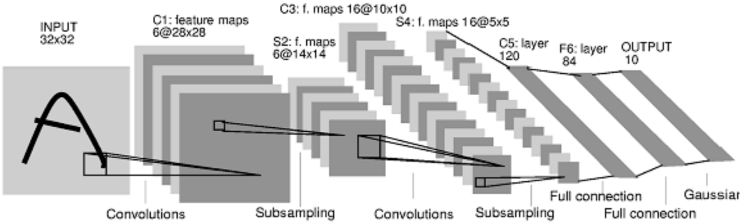
Outline

1. Deep Convolutionnal Neural Nets
2. Case Study: LeNet 5 Model

Case Study: LeNet 5 Model

80's: 1st Convolutionnal Neural Networks

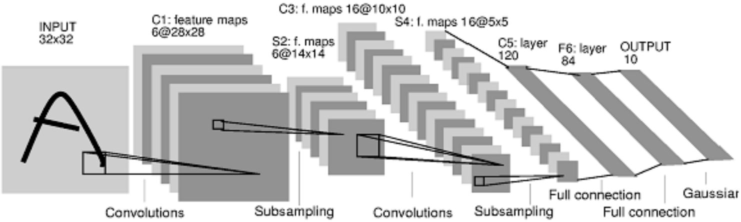
- LeNet 5 Model [LBD+89], trained using back-prop



- Input: 32x32 pixel image. Largest character is 20x20
- 2 successive blocks [Convolution + Sigmoid + Pooling (+sigmoid)]
  Cx: Convolutional layer, Sx: Subsampling layer
- C5: convolution layer ~ fully connected
- 2 Fully connected layers Fx

Case Study: LeNet 5 Model

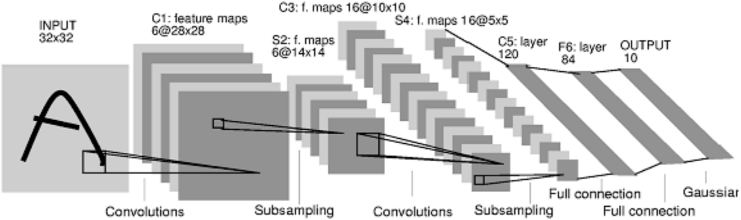
C1 Layer



- Convolutional layer with 6 5x5 filters $\Rightarrow$ 6 feature maps of size 28x28 (no padding).
- # Parameters: 5^2 per filer + bias $\Rightarrow (5*5+1)*6 = 156$
  - If it was fully connected: (32*32+1)*(28*28)*6 parameters $5 \sim 10^6$!

Case Study: LeNet 5 Model

S2 Layer

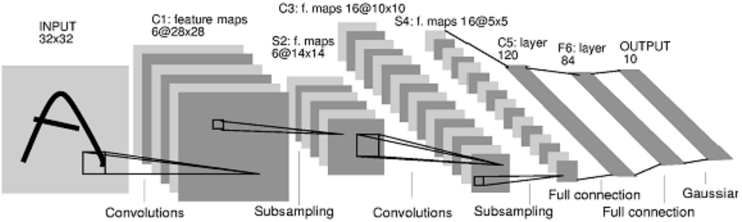


- Subsampling layer = pooling layer
- Pooling area : 2x2 in C1
- Pooling stride: 2 $\Rightarrow$ 6 features maps of size 14x14
- Pooling type : sum, multiplied by a trainable param + bias
  $\Rightarrow$ 2 parameters per channel
- Total # Parameters: $2 * 6 = 12$

Case Study: LeNet 5 Model

C3 Layer: Convolutional

- C3: 16 filters ⇒ 16 feature maps of size 10x10 (no padding)



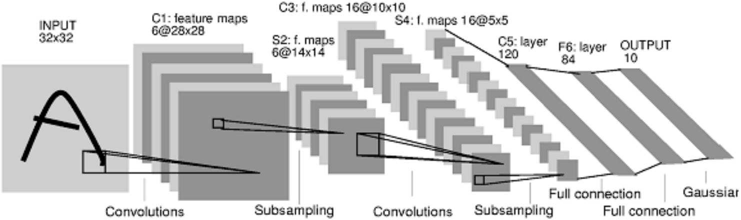
- 5x5 filters connected to a subset of S2 maps
  ⇒ 0-5 connected to 3, 6-14 to 4, 15 connected to 6

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

- # Parameters: 1516
  $(5*5*3+1)*6 + (5*5*4+1)*9 + (5*5*6+1) = 456+909+151$

Case Study: LeNet 5 Model

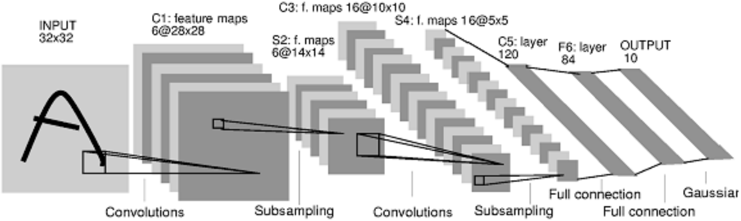
S4 Layer



- Subsampling layer = pooling layer
- Pooling area : 2x2 in C3
- Pooling stride: 2 ⇒ 16 features maps of size 5x5
- Pooling type : sum, multiplied by a trainable param + bias
  ⇒ 2 parameters per channel
- Total # Parameters: $2 * 6 = 12$

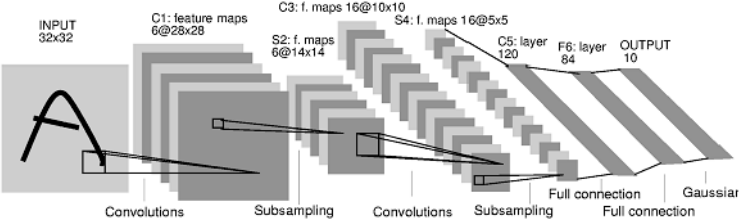
Case Study: LeNet 5 Model

C5 Layer: Convolutionnal layer



- 120 5x5x16 filters $\Rightarrow$ whole depth of $S4$ ($\neq C3$)
- Each maps in $S4$ is 5x5 $\Rightarrow$ single value for each $C5$ maps
- $C5$ 120 features map of size 1x1 (vector of size 120)
  $\Rightarrow$ spatial information lost, $\sim$ to a fully connected layer
- Total # Parameters: $(5 * 5 * 16 + 1) * 120 = 48210$

Case Study: LeNet 5 Model



F6 Layer: Fully Connected layer

- 84 fully connected units.
- # Parameters: 84*(120+1)=10164

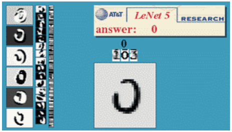
F7 Layer (output): Fully Connected layer

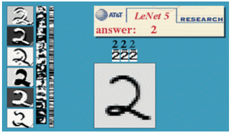
- 10 (# classes) fully connected units.
- # Parameters: 10*(84+1)=850

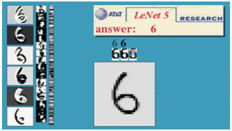
Case Study: LeNet 5 Model

- Evaluation on MNIST
- Total # parameters ~ 60000
  - 60,000 original datasets: test error: 0.95%
  - 540,000 artificial distortions + 60,000 original: Test error: 0.8%

- **Successful deployment for postal code reading in the US**







References I

Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel, *Backpropagation applied to handwritten zip code recognition*, Neural computation 1 (1989), no. 4, 541–551.