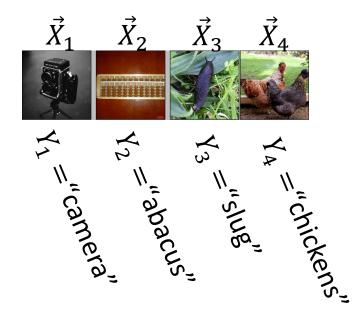


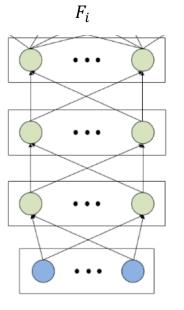
Deep Learning

- Notation
- Forward Propagation (Using the neural net)
- Loss functions (Testing the neural net)
- Back-Propagation (Training the neural net)
- Convolutional Neural Net
- Singing-Voice Separation Using Deep Recurrent Network
- Semantic Image Inpainting with Deep Generative Models



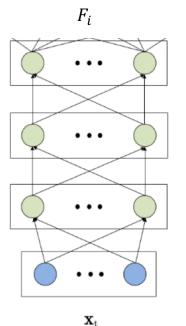
Usually we have two databases:

- A training database consists of Ndifferent training tokens (one token = one image, or sentence, or speech files, or whatever). We write them as vectors, $\vec{X_i} = [X_{i1}, \dots, X_{iM}]$, for $1 \le i \le N$. Each one has an associated reference (ground truth) label Y_i .
- A testing database contains only the test tokens $\vec{X_i}$, for N + 1 $\leq i$



For both training and testing, we have to present the token $\vec{X_i}$ to the input of the neural net, and then the neural net computes some output $\vec{F_i}$.

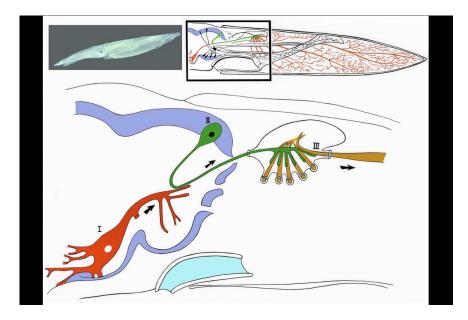




A deep neural net has thousands of neurons (also called nodes).

Each node computes two variables:

- The "affine", Z_{ij} , models the synapse of a biological neuron.
- The "activation," A_{ij} , models the axon of a biological neuron.



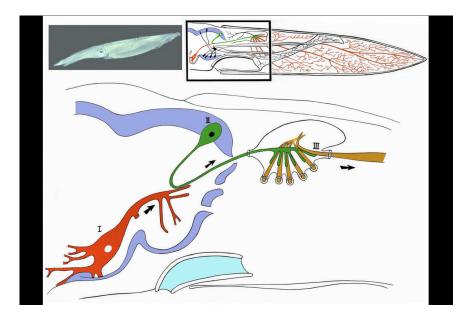
A deep neural net has thousands of neurons (nodes).

Each neuron (node) has two key variables:

 The "affine", Z_{ij}, models the synapse of a biological neuron, collecting information from a lot of other neurons:

$$Z_{ij} = \sum_{k} A_{ik} W_{kj}$$

• The "activation," A_{ij} , models the axon of a biological neuron.



A deep neural net has thousands of neurons (nodes).

Each neuron (node) has two key variables:

- The "affine", Z_{ij} , models the synapse of a biological neuron.
- The "activation," A_{ij}, models the axon of a biological neuron, i.e., it's zero when the input is negative, and nonzero when the input is positive:

$$A_{ij} = g(Z_{ij})$$

Notation for a Neural Net without Layers

- A_{ij} is the j^{th} activation for the i^{th} token:
 - Some of the activations are provided by the input, i.e., A_{ij} = X_{ij} for some of the j's.
 - Some of the activations are outputs, i.e., $F_{ij} = A_{ij}$ for some of the j's.
 - Some of the activations are neither inputs nor outputs. Those are called "hidden nodes."
 - Which ones are inputs, hidden, and outputs? Well, it depends on the particular neural network design, there's no way to know, in general.
- Z_{ij} is the j^{th} affine for the i^{th} token
- W_{kj} is the $(k, j)^{th}$ weight.

Notation for a Neural Net with Layers

- $A_{ij}^{(l)}$ is the j^{th} activation in the l^{th} layer for the i^{th} token:
 - The 0^{th} layer is the input, i.e., $A_{ij}^{(0)} = X_{ij}$.
 - The L^{th} layer is the output, i.e., $F_{ij} = A_{ij}^{(L)}$.
 - All other layers are "hidden layers."
- $Z_{ij}^{(l)}$ is the j^{th} affine **in the** l^{th} layer for the i^{th} token
- $W_{kj}^{(l)}$ is the $(k, j)^{th}$ weight in the l^{th} layer.

$$Z_{ij}^{(l)} = \sum_{k} A_{ik}^{(l-1)} W_{kj}^{(l)}$$

Deep Learning

- Notation
- Forward Propagation (Using the neural net)
- Loss functions (Testing the neural net)
- Back-Propagation (Training the neural net)
- Convolutional Neural Net
- Singing-Voice Separation Using Deep Recurrent Network
- Semantic Image Inpainting with Deep Generative Models

Forward Propagation (Using the Neural Net)

- We use a neural net by presenting a token $\vec{X_i}$, and computing the output $\vec{F_i}$.
- This is done by setting:

•
$$A_{ij}^{(0)} = X_{ij}$$

• For $1 \leq l \leq L$:

•
$$Z_{ij}^{(l)} = \sum_k A_{ik}^{(l-1)} W_{kj}^{(l)}$$

•
$$A_{ij}^{(l)} = g(Z_{ij}^{(l)})$$

•
$$F_{ij} = A_{ij}^{(L)}$$

• This algorithm is called "forward propagation," because information propagates forward through the network, from the 0^{th} layer to the L^{th} layer.

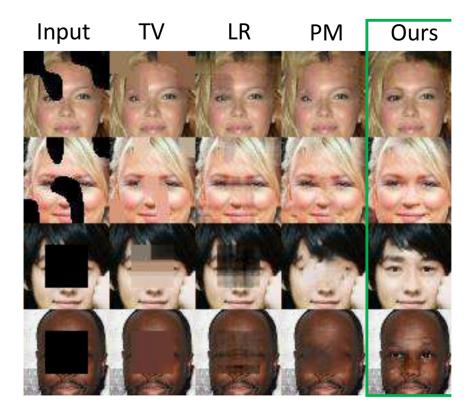
Deep Learning

- Notation
- Forward Propagation (Using the neural net)
- Loss functions (Testing the neural net)
- Back-Propagation (Training the neural net)
- Convolutional Neural Net
- Singing-Voice Separation Using Deep Recurrent Network
- Semantic Image Inpainting with Deep Generative Models

How well did it do?

- We test a neural net by computing \vec{F}_i from \vec{X}_i , for each of the tokens $1 \le i \le N$, and then comparing the network output to the reference (ground truth) answer, Y_i .
 - During training: we measure error using training data, and try to train the network in order to reduce the error rate.
 - During "development test:" we compare different networks on the development test data.
 - During "evaluation test:" our customer tests our network with data it's never seen before.
- But... How do we compare $\vec{F_i}$ to Y_i ? i.e., how we define "error" or "loss"?

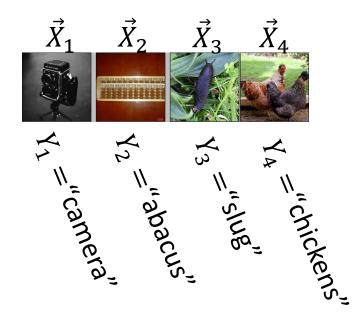
Regression problems: Sum-squared error



- For example, suppose that the network output is an image.
- An image is a vector, $\vec{F}_i = [F_{i1}, ..., F_{iM}]$
- The "right answer" is the image we were trying to reconstruct, $\vec{Y}_i = [Y_{i1}, \dots, Y_{iM}]$.
- Then a reasonable loss function is sum-squared error (SSE):

$$L_{SSE} = \sum_{i=1}^{N} \sum_{j=1}^{M} (Y_{ij} - F_{ij})^{2}$$

Classifier problems: Cross-entropy



- On the other hand, for this course, we usually want Y_i to be some category label, for example, $Y_i = "chickens"$.
- In that case, we can use a special kind of nonlinearity at the output of our neural network, called a softmax, that gives a probabilistic interpretation to the network outputs:

 $F_{ij} = P(Y_i = j^{th} \text{ type of category})$

 Then a reasonable loss function is the log probability of the correct class:

$$L_{CE} = -\sum_{i=1}^{N} \ln F_{i,Y_i}$$

 This error criterion is called "cross entropy" for reasons that are fascinating but way beyond the scope of this course.

Classifier output: Softmax

- We want Y_i to be some category label, for example, $Y_i = "boots"$.
- In that case, we want F_{ij} to meet the criteria for a probability, i.e., we need $F_{ij} \ge 0$ and $\sum_j F_{ij} = 1$.
- In order to do that, we use a special kind of nonlinearity in the last layer of the neural net, called a softmax:

$$F_{ij} = \frac{e^{Z_{ij}^{(L)}}}{\sum_k e^{Z_{ik}^{(L)}}}$$

Deep Learning

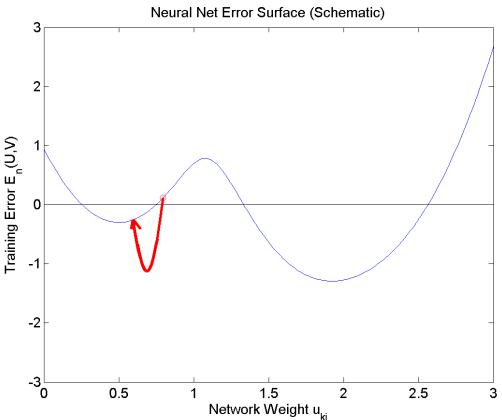
- Notation
- Forward Propagation (Using the neural net)
- Loss functions (Testing the neural net)
- Back-Propagation (Training the neural net)
- Convolutional Neural Net
- Singing-Voice Separation Using Deep Recurrent Network
- Semantic Image Inpainting with Deep Generative Models

Training the Neural Net

A neural net is trained according to gradient descent:

$$W_{jk}^{(l)} = W_{jk}^{(l)} - \eta \frac{\partial L}{\partial W_{jk}^{(l)}}$$

So that the loss function, L, gradually approaches a local minimum.



Training the Neural Net: Notation

• Let's use the following shorthand:

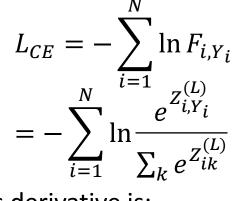
$$\delta(\text{Variable}) = \frac{\partial L}{\partial(\text{Variable})}$$

For example:

$$\delta W_{kj}^{(l)} = \frac{\partial L}{\partial W_{kj}^{(l)}}$$

Training the Neural Net: Last Layer

The cross entropy loss is:



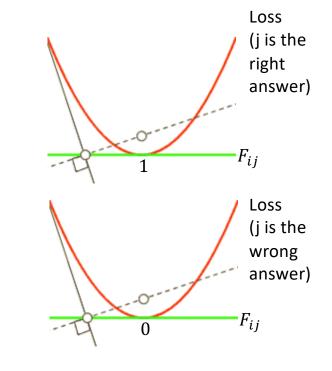
Its derivative is:

$$\delta Z_{ij}^{(L)} = \begin{cases} F_{ij} - 1 & j = Y_i \\ F_{ij} & j \neq Y_i \end{cases}$$

Here's how to remember that:

• If j is the right answer, then error is minimized $(\delta Z_{ij}^{(L)} = 0)$ when $F_{ij} = 1$.

• If j is the wrong answer, then error is minimized $(\delta Z_{ij}^{(L)} = 0)$ when $F_{ij} = 0$.



Credit: Tosha, distributed under CC-BY 1.0, https://commons.wikimedia.org/wiki/File:Parabola-antipodera.gif

Training the Neural Net: Other Layers

Every other layer is given by:

$$A_{ij}^{(l)} = g\left(Z_{ij}^{(l)}\right), \qquad Z_{ij}^{(l)} = \sum_{k} A_{ik}^{(l-1)} W_{kj}^{(l)},$$

Given any "forward equation" like the ones above, and given the derivative of the loss w.r.t. the OUTPUT, we can work our way backward through the equation to find the derivative w.r.t. the INPUT. For example, given $\delta A_{ij}^{(l)}$, and if we know g'(x) = dg/dx, we can find $\delta Z_{ij}^{(l)}$ as:

$$\delta Z_{ij}^{(l)} = g' \left(Z_{ij}^{(l)} \right) \delta A_{ij}^{(l)}$$

Training the Neural Net: Other Layers

$$A_{ij}^{(l)} = g\left(Z_{ij}^{(l)}\right), \qquad Z_{ij}^{(l)} = \sum_{k} A_{ik}^{(l-1)} W_{kj}^{(l)},$$

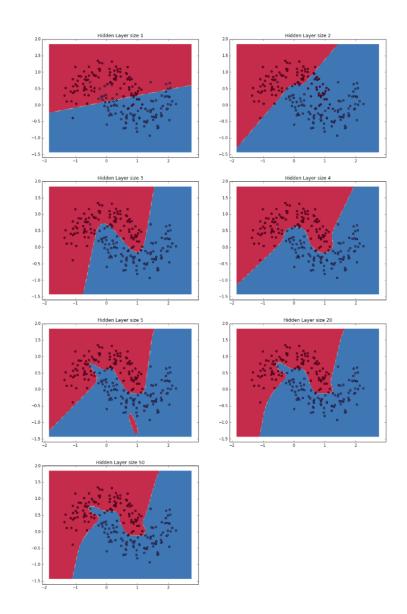
Likewise, given $\delta Z_{ij}^{(l)}$, we can find $\delta A_{ik}^{(l-1)}$ as:

$$\delta A_{ik}^{(l-1)} = \sum_{j} W_{kj}^{(l)} \delta Z_{ij}^{(l)}$$

Example from a blog

Here's an example of training a two-layer network in python. It has a good display of the different decision boundaries you get with different #s of hidden nodes.

https://medium.com/mlalgorithms/neural-networks-fordecision-boundary-in-pythonb243440fb7d1

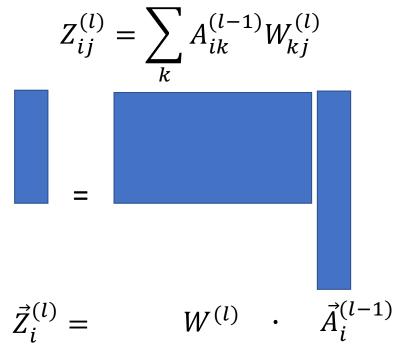


Deep Learning

- Differentiable Perceptron/One-Layer Neural Net
- Two-Layer Neural Net
- Loss functions (Testing the neural net)
- Back-Propagation (Training the neural net)
- Convolutional Neural Net
- Singing-Voice Separation Using Deep Recurrent Network
- Semantic Image Inpainting with Deep Generative Models

Convolution versus Matrix Multiplication

A regular neural net uses a matrix multiplication in each layer:

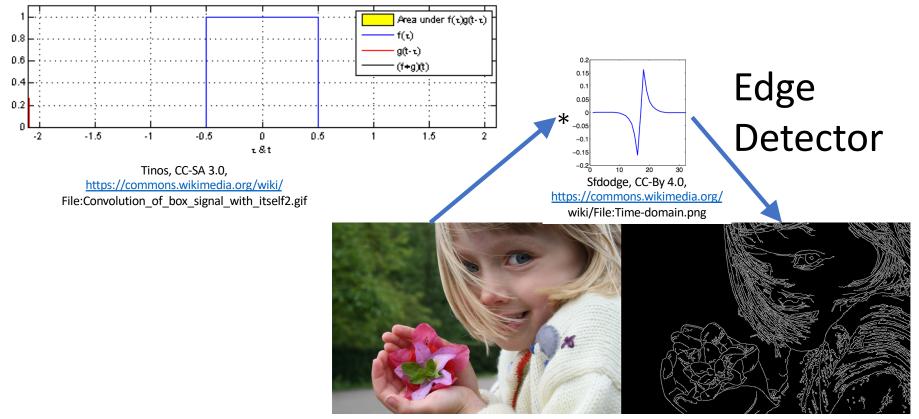


A convolutional neural net uses a convolution at each layer:

$$Z_{ij}^{(l)} = \sum_{k} A_{i,k}^{(l-1)} W_{j-k}^{(l)}$$
$$=$$
$$\vec{Z}_{i}^{(l)} = W^{(l)} * \vec{A}_{i}^{(l-1)}$$

Example Convolutions: Moving Average, Edge Detector

Moving Average



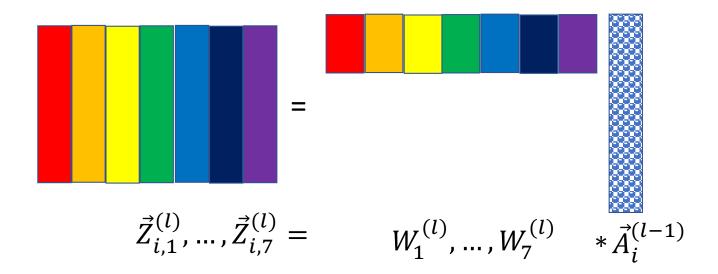
Jon McLoone, CC-SA 3.0, https://commons.wikimedia.org/wiki/File:%C3%84%C3%A4retuvastuse_n%C3%A4ide.png

Convolution with Many Channels

Usually, we want the convolutional network to compute many different channels, c:

$$Z_{ij,c}^{(l)} = \sum_{k} A_{i,k}^{(l-1)} W_{j-k,c}^{(l)}$$

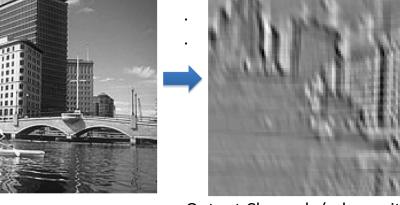
Each of the channels is computing a different type of feature (average, edge, etc.). Each pixel, in each output channel, tells the degree to which that channel exists at that location in the image.



What is a convolution?

- Weighted moving average
- All positive weights: average
- Some weights negative: finds edges, corners, etc.



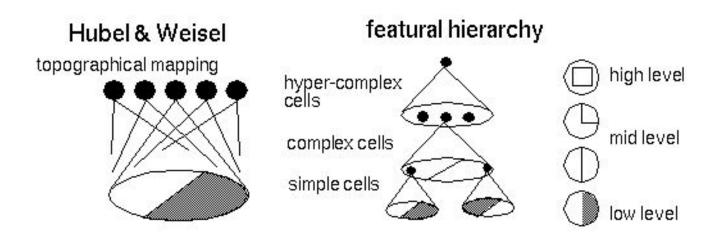


Output Channels (edges with different orientations)

Input

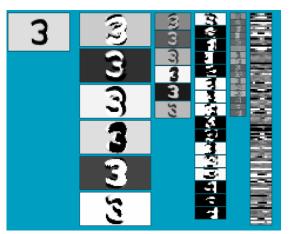
Biological inspiration

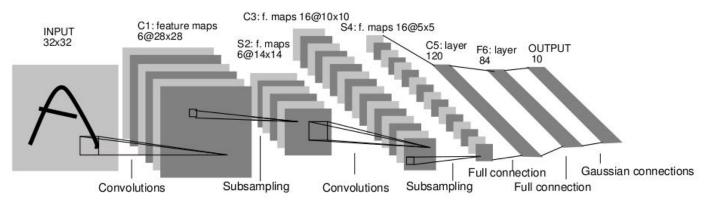
- D. Hubel and T. Wiesel (1959, 1962, Nobel Prize 1981)
 - Visual cortex consists of a hierarchy of *simple, complex,* and *hyper-complex* cells



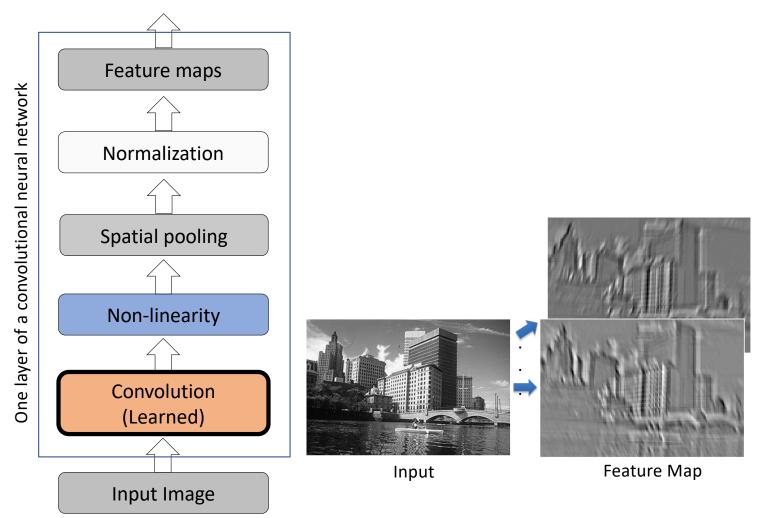
<u>Source</u>

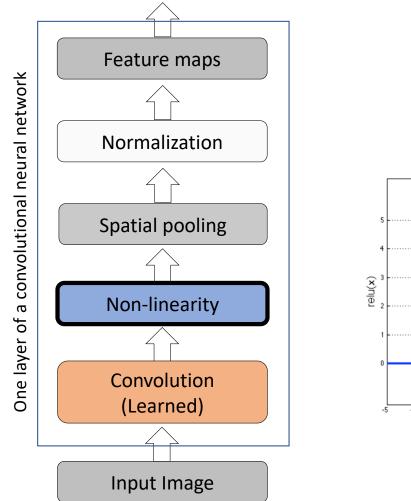
- Neural network with specialized connectivity structure
- Stack multiple stages of feature extractors
- Higher stages compute more global, more invariant features
- Classification layer at the end

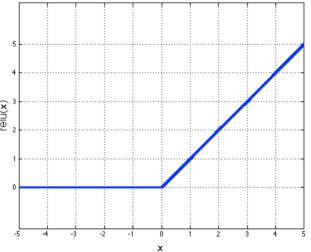


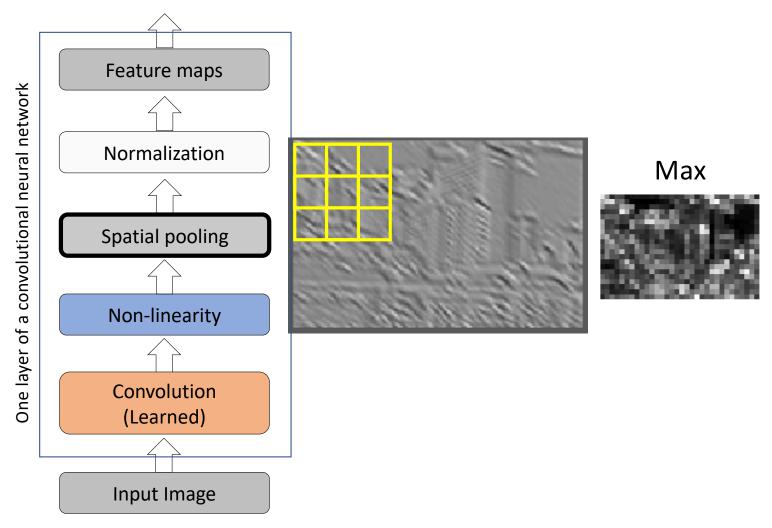


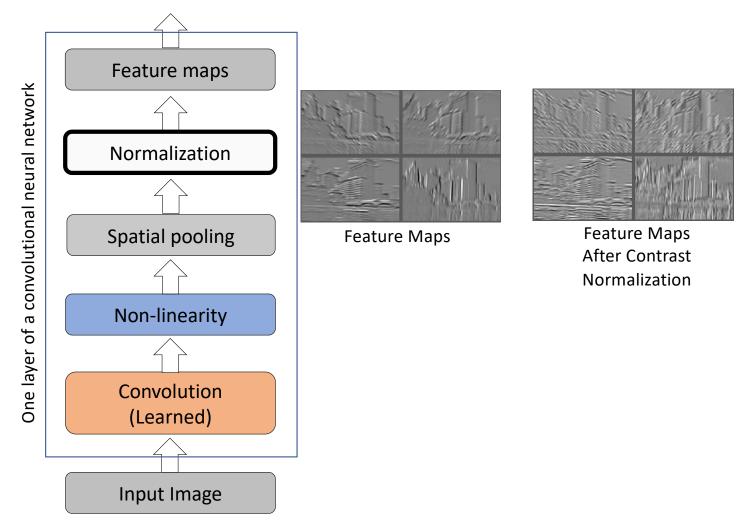
Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, <u>Gradient-based learning applied to document recognition</u>, Proc. IEEE 86(11): 2278–2324, 1998.



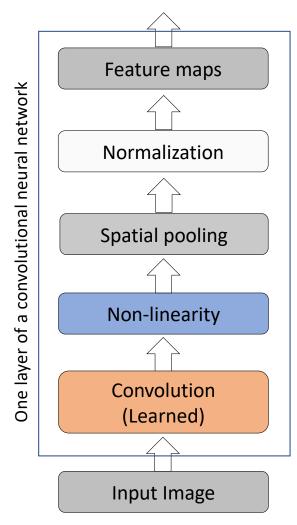








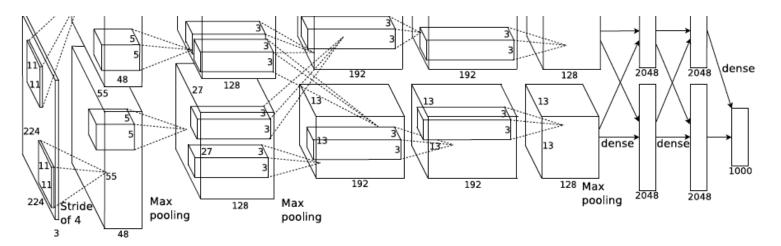
•



- Convolutional filters are trained in a supervised manner by back-propagating classification error
- Basically, you can think of the top layer as a "linear classifier," and the layer below it learns features. And its features are computed from the outputs of the layer below that, and so on.

AlexNet

- Similar framework to LeCun'98 but:
 - Bigger model (7 hidden layers, 650,000 units, 60,000,000 params)
 - More data (10⁶ vs. 10³ images)
 - GPU implementation (50x speedup over CPU)
 - Trained on two GPUs for a week



A. Krizhevsky, I. Sutskever, and G. Hinton, <u>ImageNet Classification with Deep Convolutional</u> <u>Neural Networks</u>, NIPS 2012

ImageNet Challenge

IM GENET

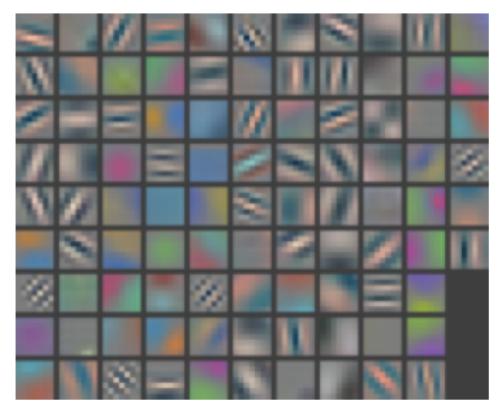


[Deng et al. CVPR 2009]

- ~14 million labeled images, 20k classes
- Images gathered from Internet
- Human labels via Amazon MTurk
- Challenge: 1.2 million training images, 1000 classes

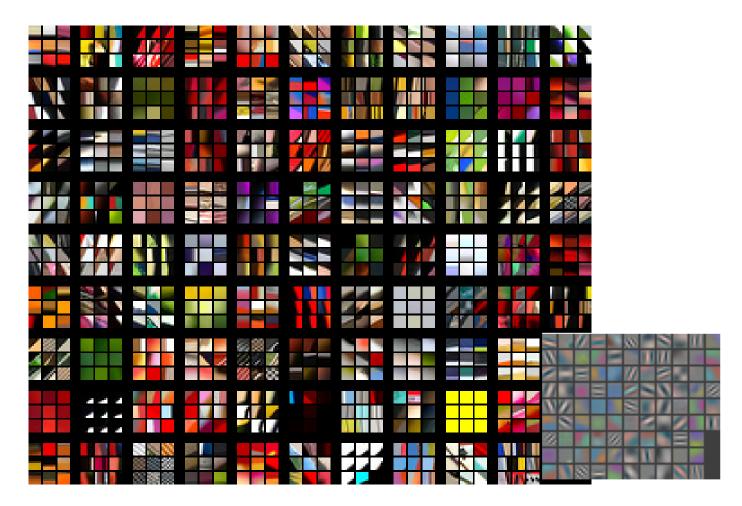
A. Krizhevsky, I. Sutskever, and G. Hinton, <u>ImageNet Classification with Deep Convolutional</u> <u>Neural Networks</u>, NIPS 2012

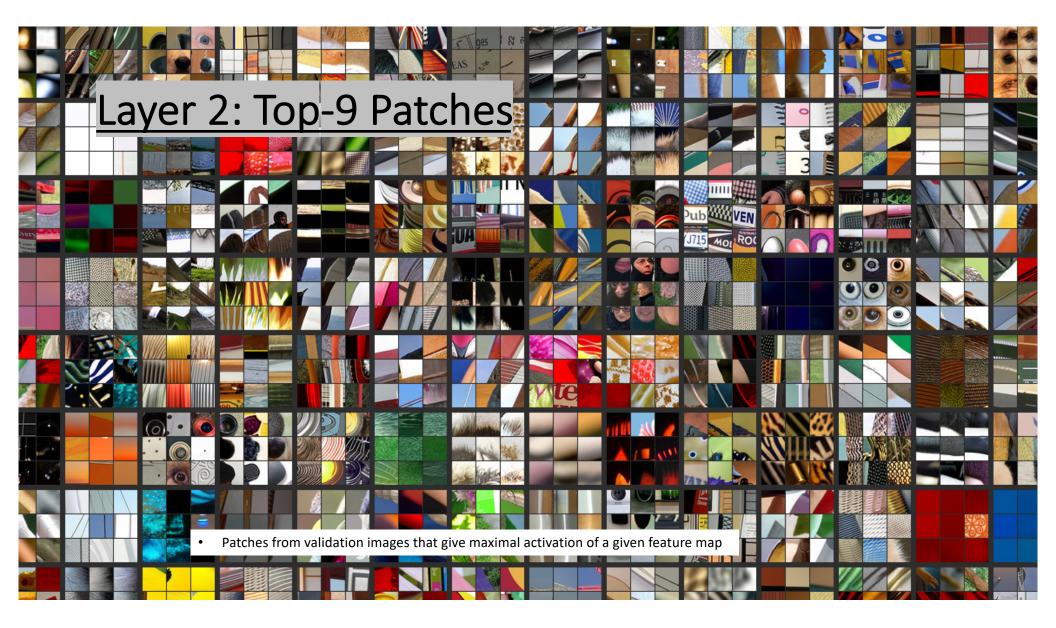
Layer 1 Filters



M. Zeiler and R. Fergus, <u>Visualizing and Understanding Convolutional Networks</u>, arXiv preprint, 2013

Layer 1: Top-9 Patches









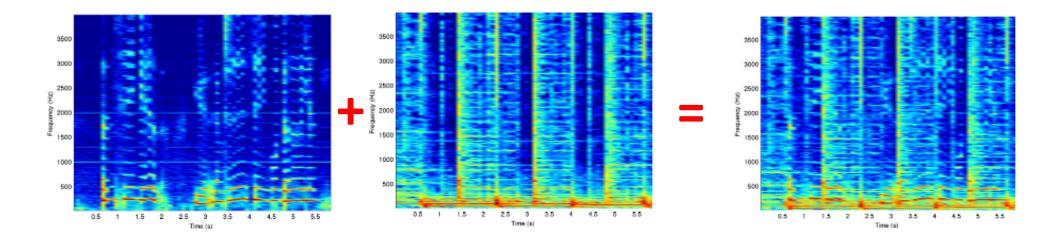


Deep Learning

- Differentiable Perceptron/One-Layer Neural Net
- Two-Layer Neural Net
- Loss functions (Testing the neural net)
- Back-Propagation (Training the neural net)
- Convolutional Neural Net
- Singing-Voice Separation Using Deep Recurrent Network
- Semantic Image Inpainting with Deep Generative Models

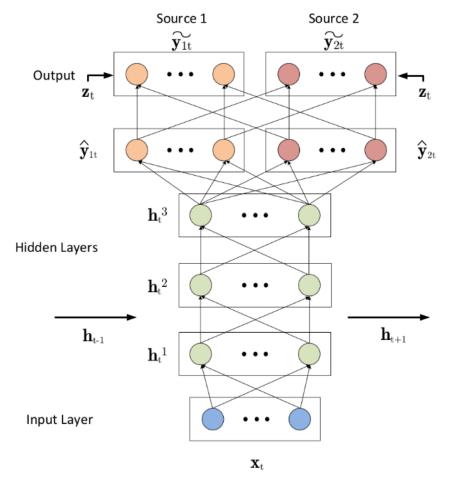
Singing-Voice Separation from Monaural Recordings Using Deep Recurrent Neural Networks Po-Sen Huang, Minje Kim, Mark Hasegawa-Johnson and Paris Smaragdis, ISMIR 2014

The problem:



Singing-Voice Separation

The solution is to train this:



To minimize this:

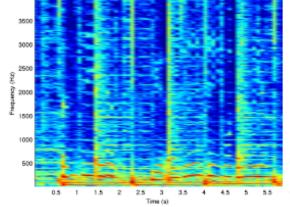
$$J_{MSE} = ||\mathbf{\hat{y}}_{1_t} - \mathbf{y}_{1_t}||_2^2 + ||\mathbf{\hat{y}}_{2_t} - \mathbf{y}_{2_t}||_2^2$$

Using these specialized output nodes:

$$\begin{split} \tilde{\mathbf{y}}_{\mathbf{1}_{t}} &= \frac{|\hat{\mathbf{y}}_{\mathbf{1}_{t}}|}{|\hat{\mathbf{y}}_{\mathbf{1}_{t}}| + |\hat{\mathbf{y}}_{\mathbf{2}_{t}}|} \odot \mathbf{z}_{t} \\ \tilde{\mathbf{y}}_{\mathbf{2}_{t}} &= \frac{|\hat{\mathbf{y}}_{\mathbf{1}_{t}}| + |\hat{\mathbf{y}}_{\mathbf{2}_{t}}|}{|\hat{\mathbf{y}}_{\mathbf{1}_{t}}| + |\hat{\mathbf{y}}_{\mathbf{2}_{t}}|} \odot \mathbf{z}_{t}, \end{split}$$

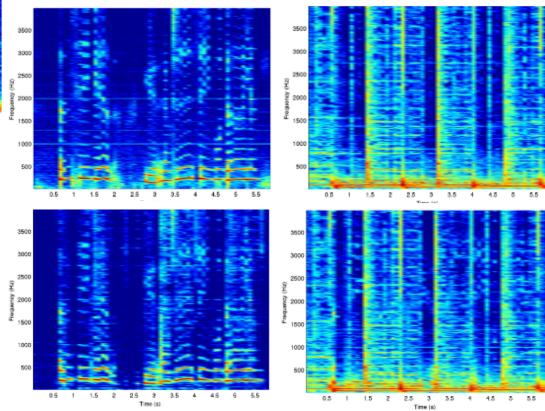
Singing-Voice Separation Results Example

• Input:





ActualNetworkOutputs:



Deep Learning

- Differentiable Perceptron/One-Layer Neural Net
- Two-Layer Neural Net
- Loss functions (Testing the neural net)
- Back-Propagation (Training the neural net)
- Convolutional Neural Net
- Singing-Voice Separation Using Deep Recurrent Network
- Semantic Image Inpainting with Deep Generative Models

Semantic Image Inpainting with Deep Generative Models

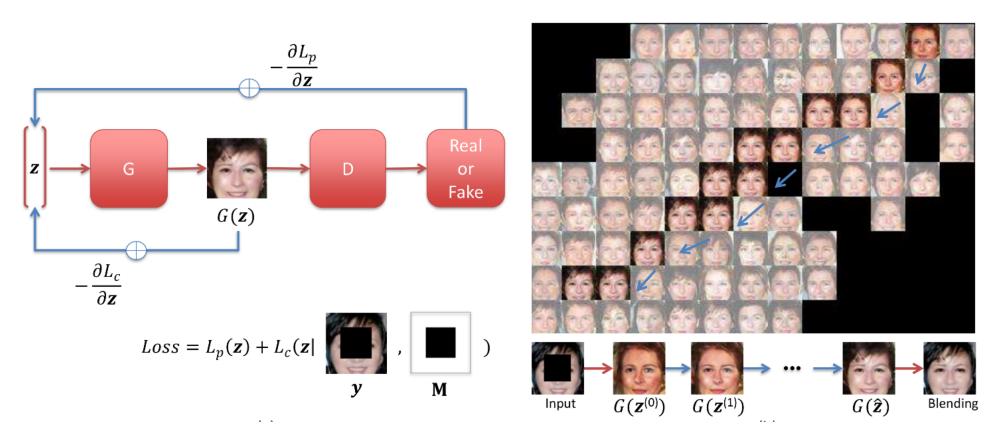
Raymond Yeh, Chen Chen, Teck Yian Lim, Alexander G. Schwing, Mark Hasegawa-Johnson and Minh Do

The problem:



Semantic Image Inpainting

The solution:



Semantic Image Inpainting

The results: Input TV LR PM Ours

