Lecture 8
Convolutional Neural Networks II
CMSC 35246: Deep Learning

Shubhendu Trivedi
&
Risi Kondor

University of Chicago

April 19, 2017

- Things we will look at today
  - Methods for Visualizing Convolutional Neural Networks
  - Motivations for Convolutions and Pooling
  - Variations
  - Dilated Convolutions
  - Idea genealogy for Convolutional Neural Networks

Housekeeping

- Quiz scores will be uploaded in a few hours
- Project proposals due tonight
- Mid term - 8 May (Just like quizzes, with some derivations)
- Late policy

Convolutional Neural Networks

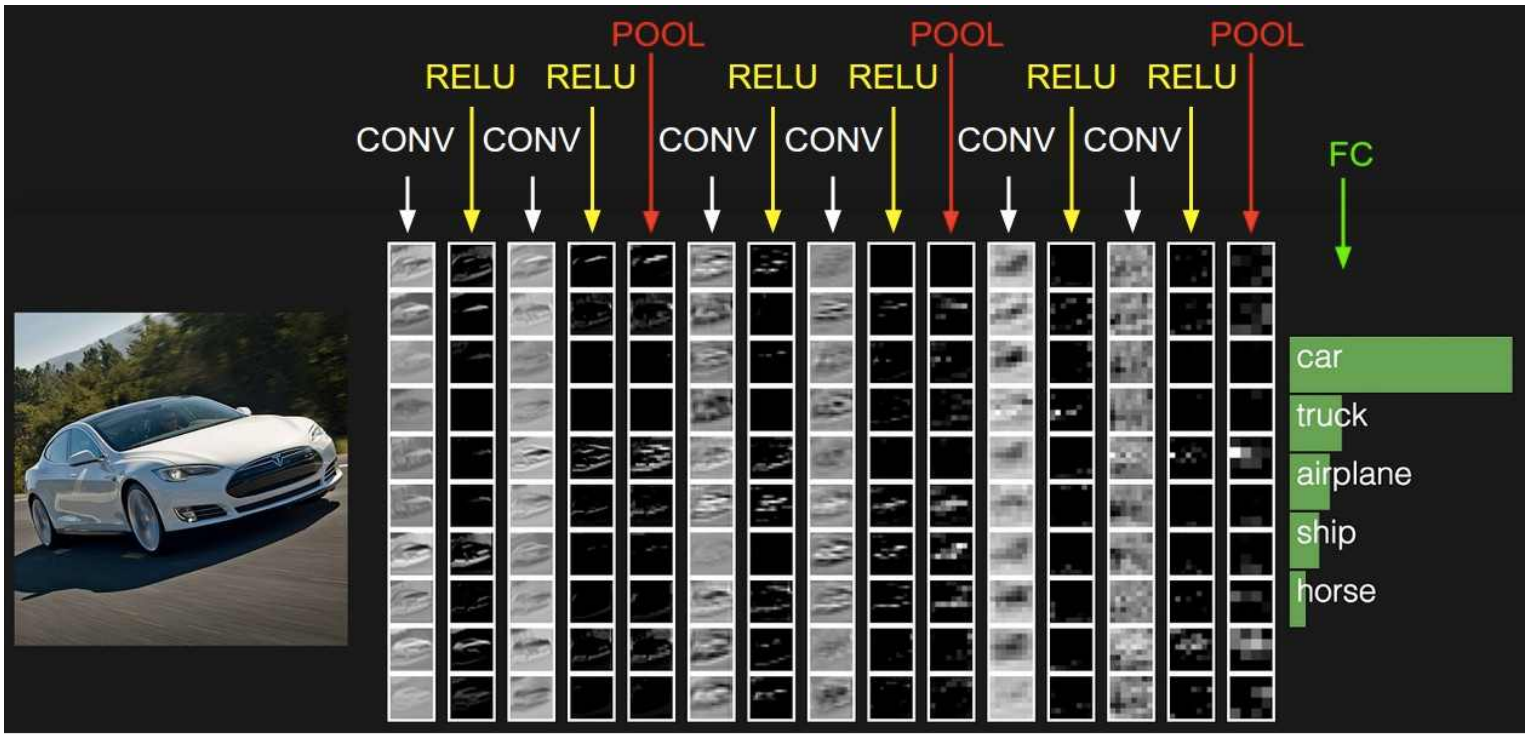


Figure: Andrej Karpathy

Last Time

- Saw convolutional networks last time
- Worked great (saw all these ImageNet results)
- How do we probe what they actually learn, or do?

A Global View: t-SNE

Stochastic Neighborhood Embedding

- For point $\mathbf{x}_j$, one measure of its similarity to another point $\mathbf{x}_i$:

$$p_{j|i} = \frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|_2^2/2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|\mathbf{x}_i - \mathbf{x}_k\|_2^2/2\sigma_i^2)}$$

- $\implies$ The conditional probability that $\mathbf{x}_i$ would pick $\mathbf{x}_j$ as its neighbor if neighbors were picked in proportion to their probability density under a Gaussian centered at $\mathbf{x}_i$
- For nearby points $p_{j|i}$ will be high
- Suppose we had low dimensional maps $\mathbf{x}_i \mapsto \mathbf{y}_i$ and $\mathbf{x}_j \mapsto \mathbf{y}_j$

Stochastic Neighborhood Embedding

- Similarity of mapped point $\mathbf{y}_j$ to $\mathbf{y}_i$

$$q_{j|i} = \frac{\exp(-\|\mathbf{y}_i - \mathbf{y}_j\|_2^2/2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|\mathbf{y}_i - \mathbf{y}_k\|_2^2/2\sigma_i^2)}$$

- SNE aims to find a lower dimensional embedding such that the discrepancy between $p_{j|i}$ and $q_{j|i}$ is minimized
- Obvious cost function:

$$J = \sum_i KL(P_i||Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}$$

t-SNE

- A variant of SNE that is better amenable to visualizations
- Avoids a *crowding problem* that SNE suffers from
- Modifies the cost function and uses a Student t-distribution to compute similarities in the low-dimensional space
- Takeaway: t-SNE embeds high dimensional points into a lower dimensional space so as to preserve pairwise distances
- In other words: Similar objects get embedded nearby

t-SNE on MNIST

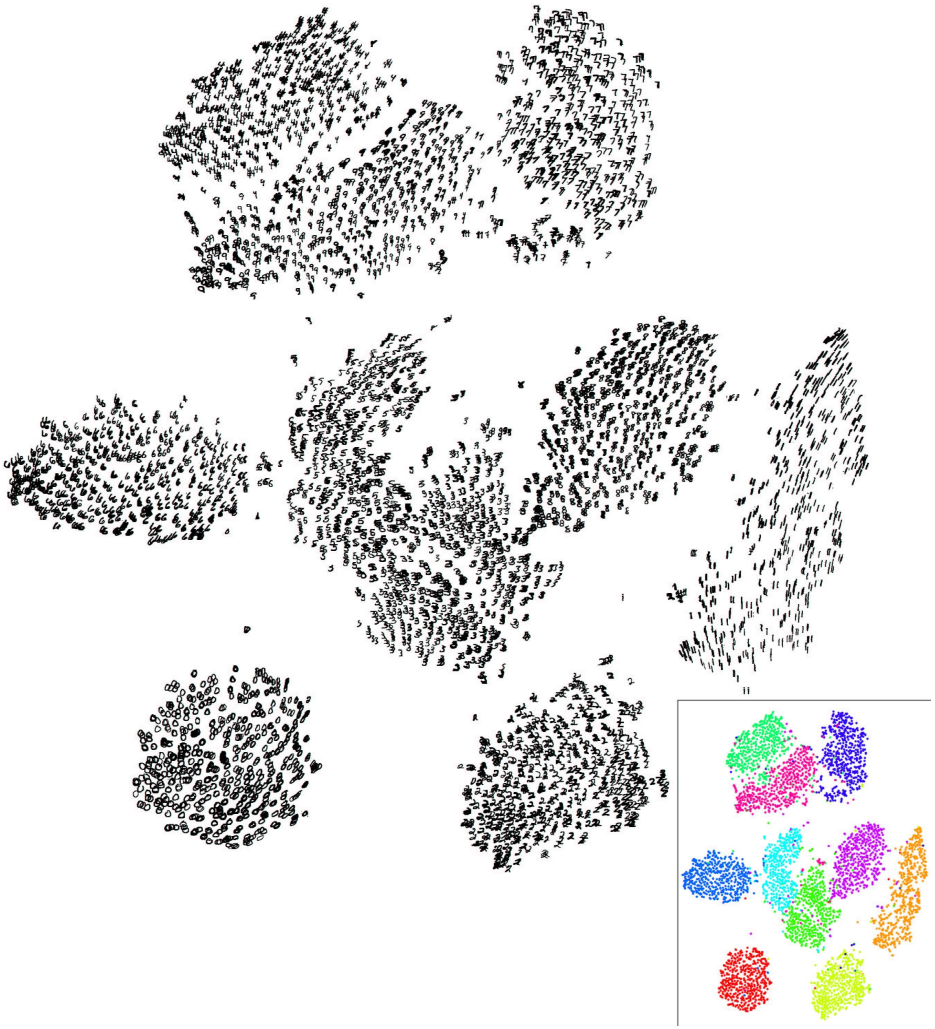
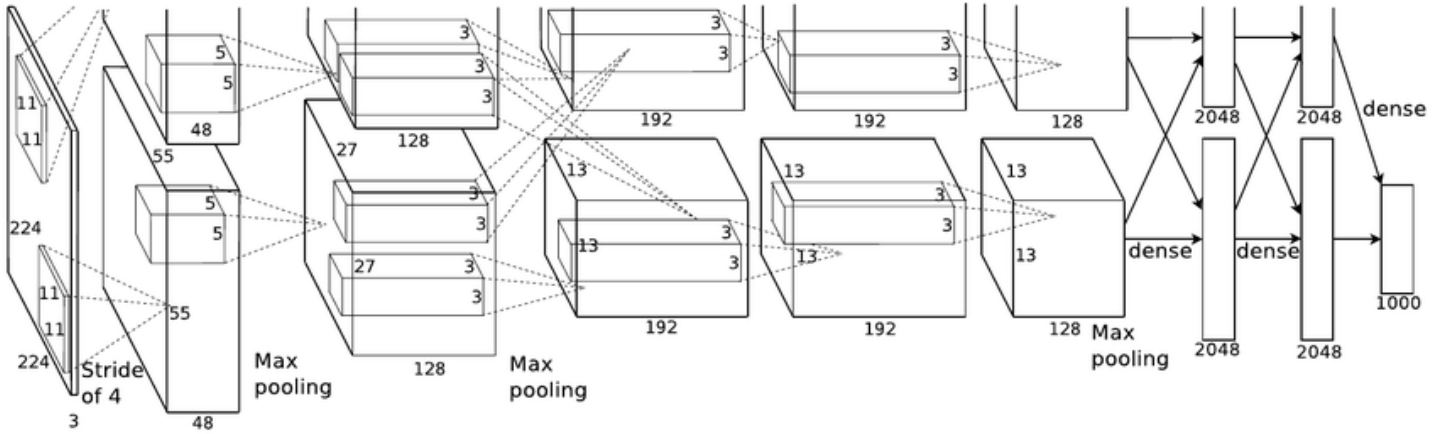


Figure: van der Maaten and Hinton

AlexNet



- AlexNet gives 4096 dimensional codes for each image
- t-SNE: place two codes close in 2D if they are close in 4096D

t-SNE on ImageNet

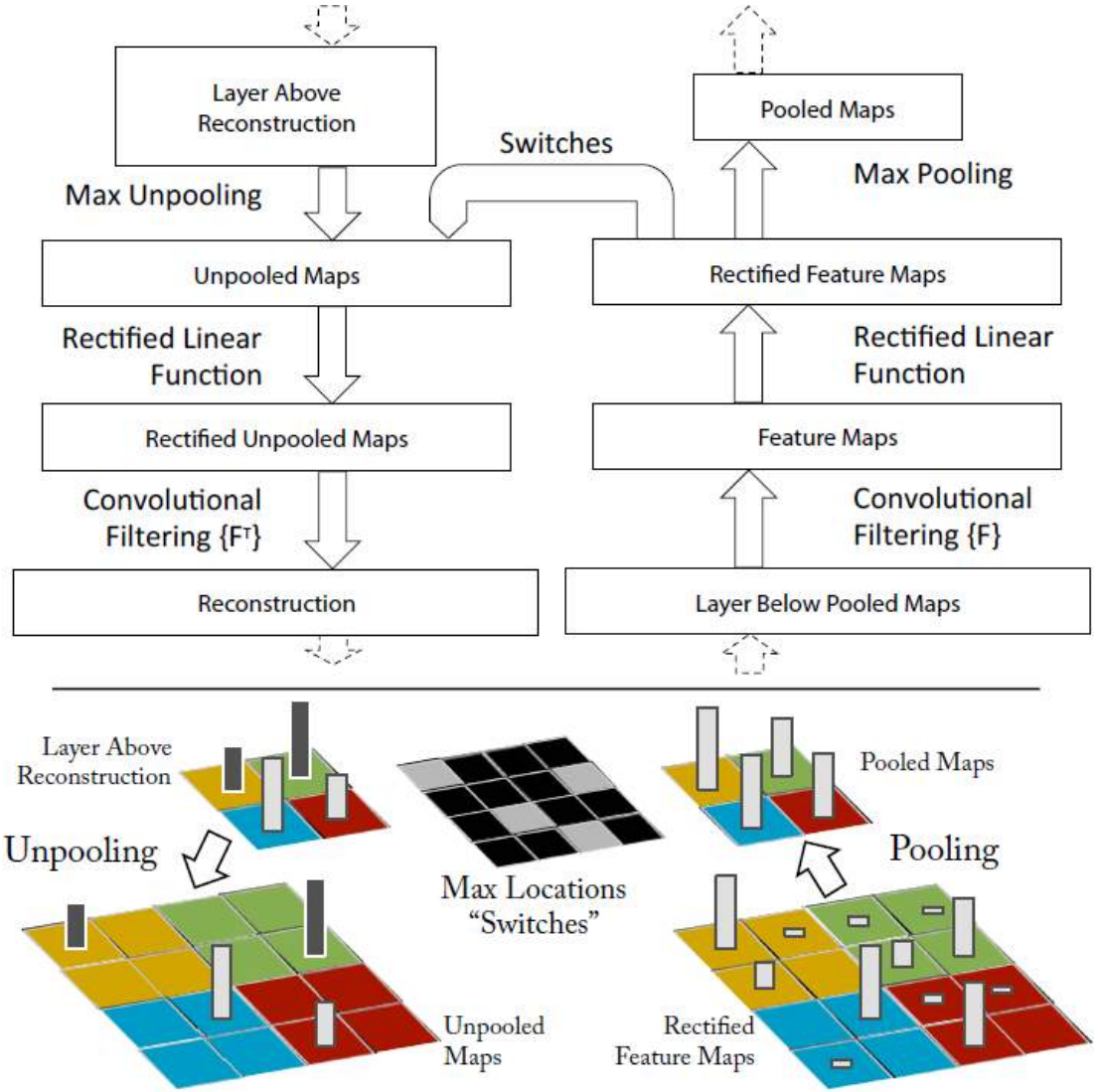
http://cs.stanford.edu/people/karpathy/cnnembed/

Visualizing Activations: DeConvolutional Approach

Zeiler and Fergus, ICML 2013

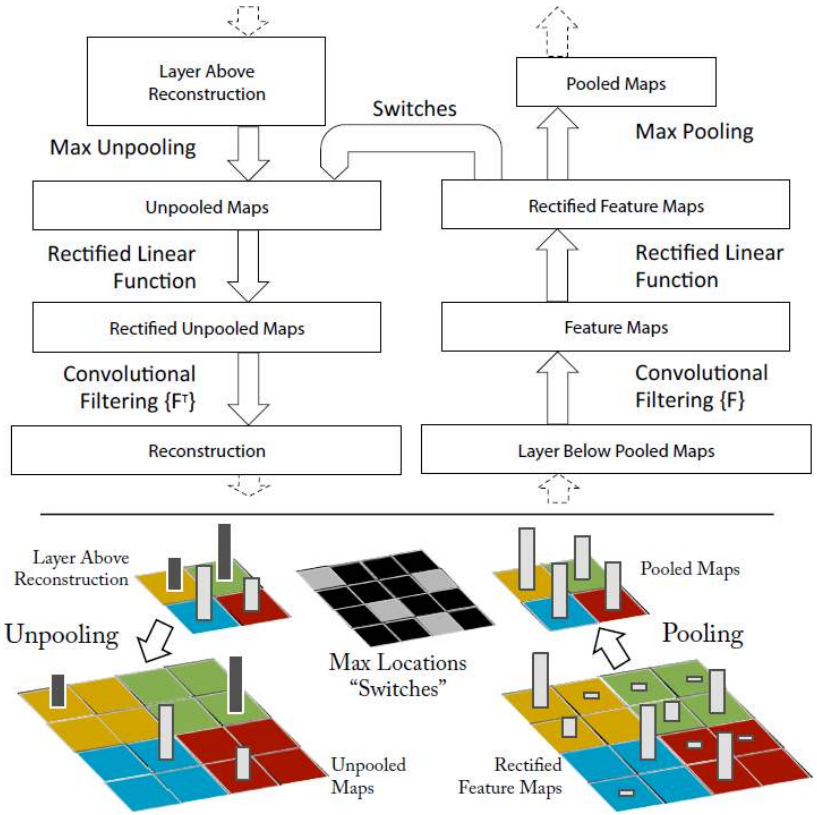
- Want to interpret activity in intermediate layers
- Idea: Map activations back to the pixel space
- Note: Visualizing kernel weights beyond the first layer is not useful
- Approach: Use a Deconvolutional Network to map back to pixel space
- A Deconvolutional Network is a convnet model run in reverse (runs all the same operations)
- PS: There are many later papers that improve the approach of Zeiler and Fergus (say using guided backprop), but the basic idea is similar

Deconvolutional Network



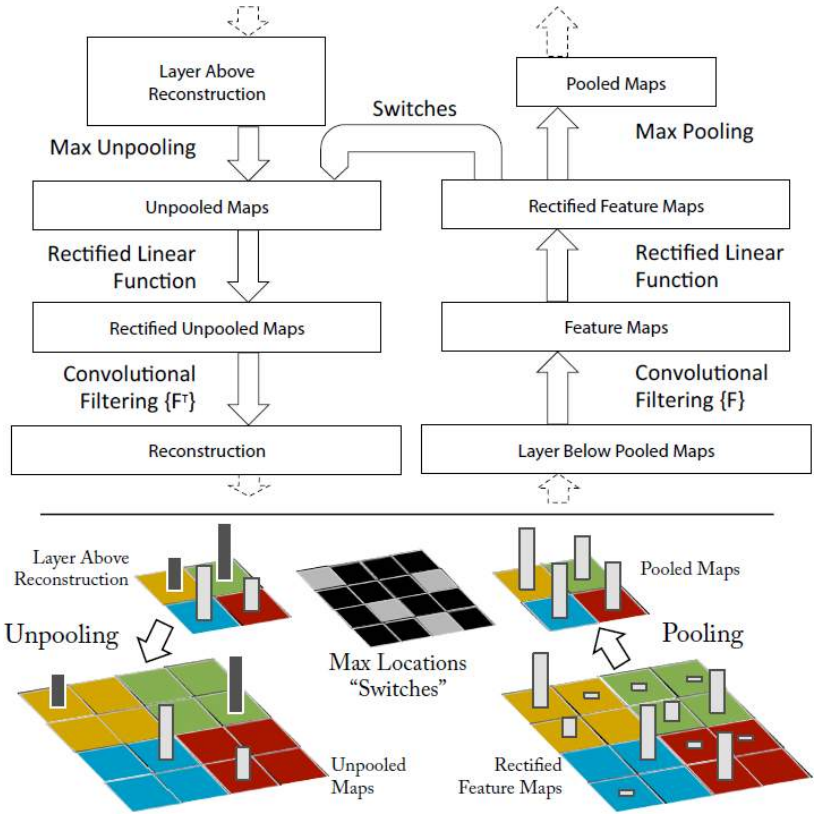
Approach

- Attach a DeConv Net to a layer of the convnet (to be examined)
- Pass input image through CNN and obtain activations
- For a given neuron, set all activations to zero and backprop from there



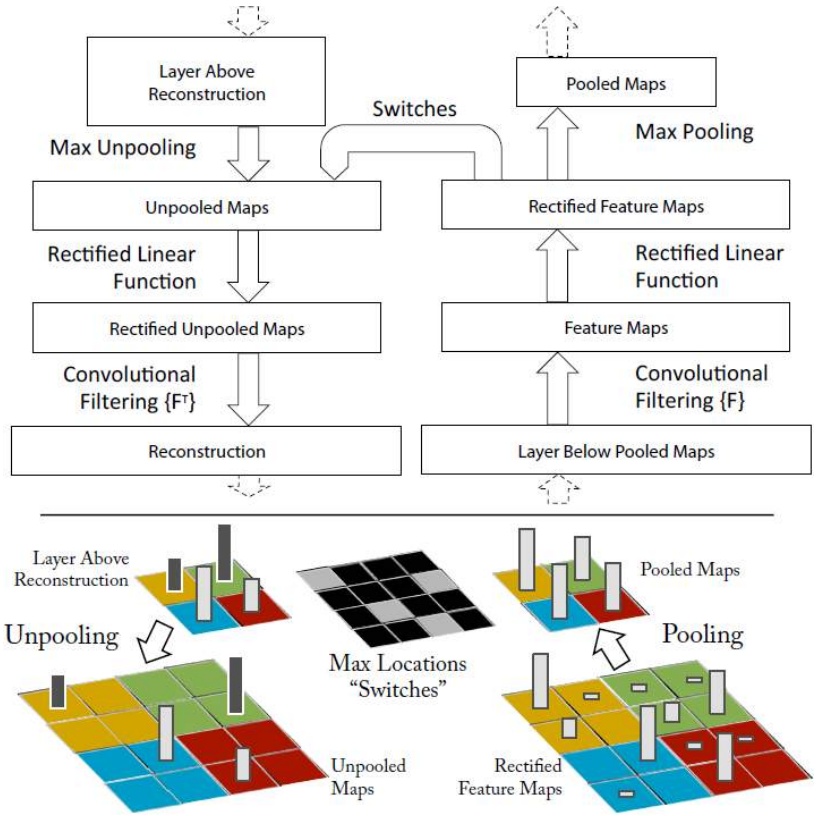
Approach

- Successively unpool, rectify and filter till pixel space is reached
- Max is not invertible: Keep switch variables to keep track of locations of max in each pooling



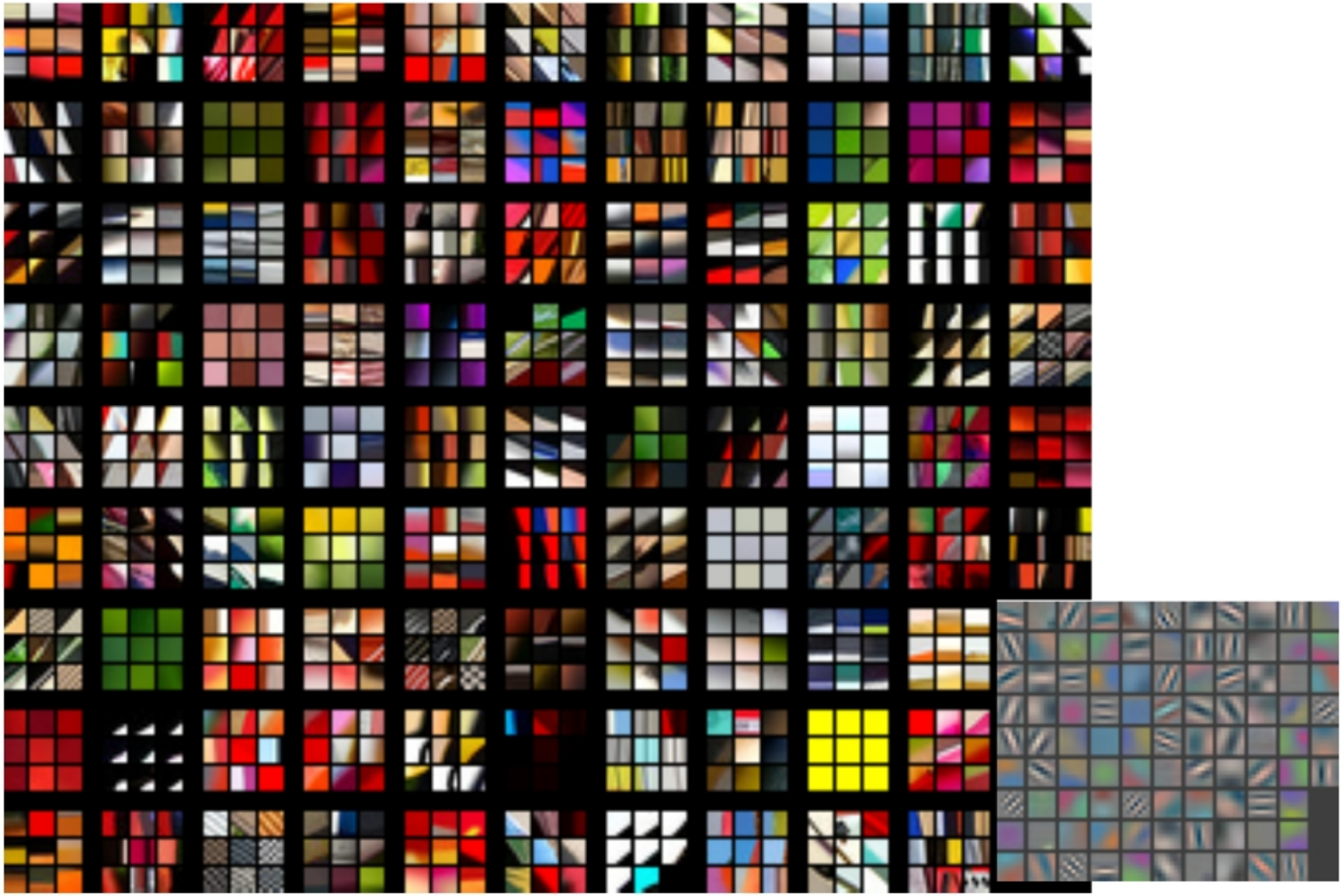
Approach

- ReLU: Pass the signal through a ReLU non-linearity
- Filtering: Use learned filters to convolve backward signal, but use transposed versions of filters and applied to ReLU activations
- Reconstructions show which parts of input image are discriminative



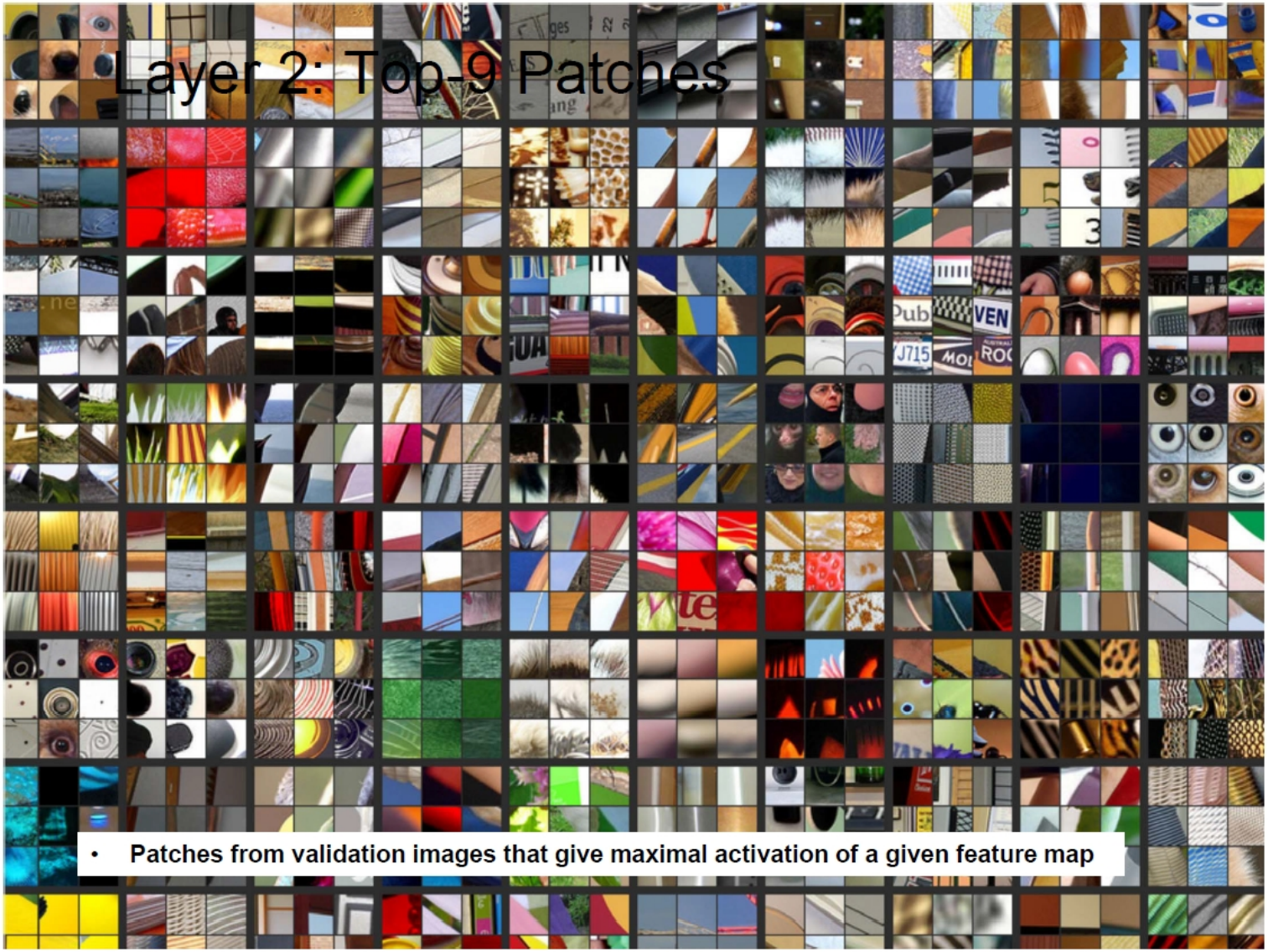
Feature Visualizations from last time were generated by this approach

Layer 1 filters

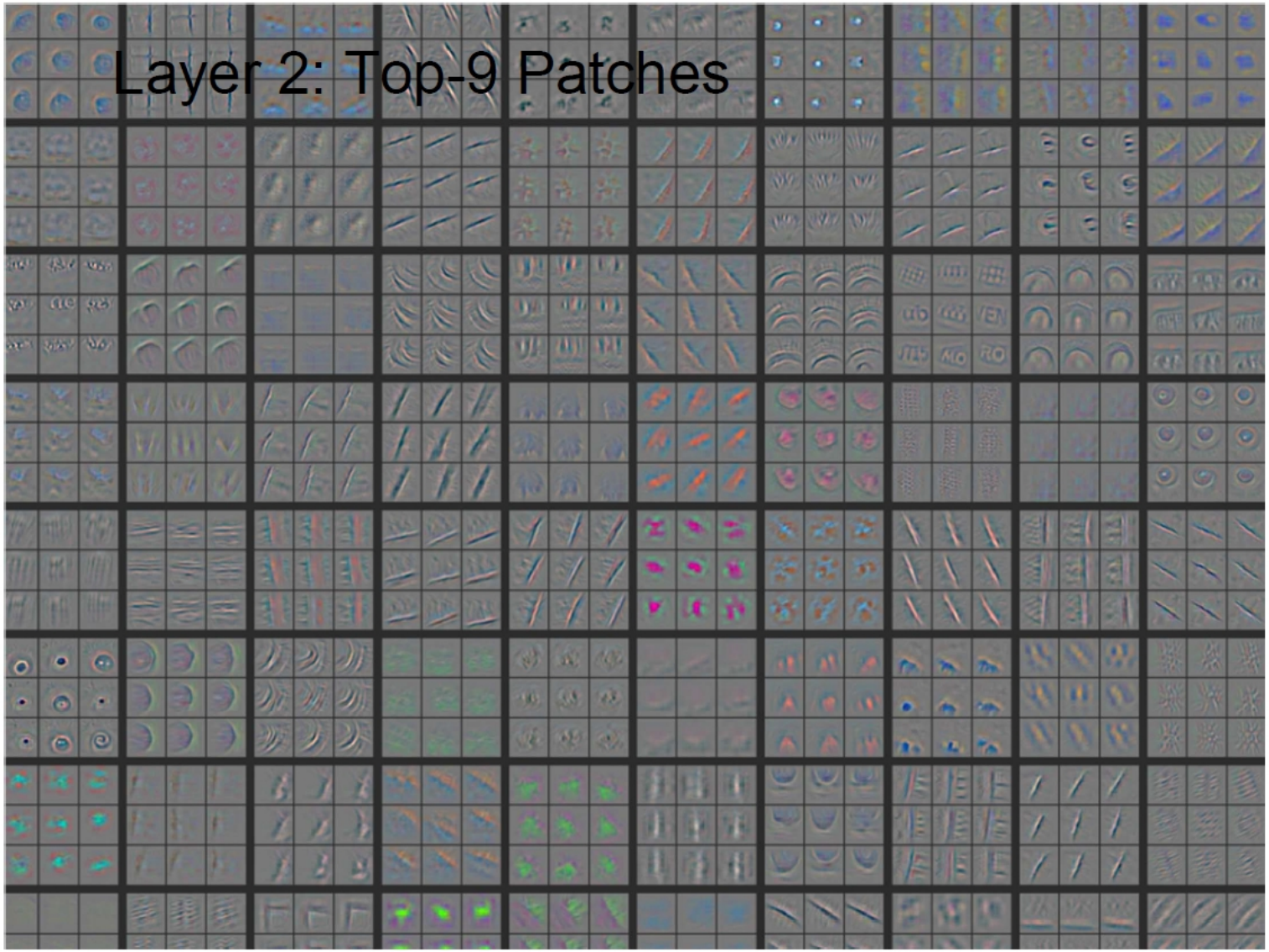


Matthew Zeiler and Rob Fergus

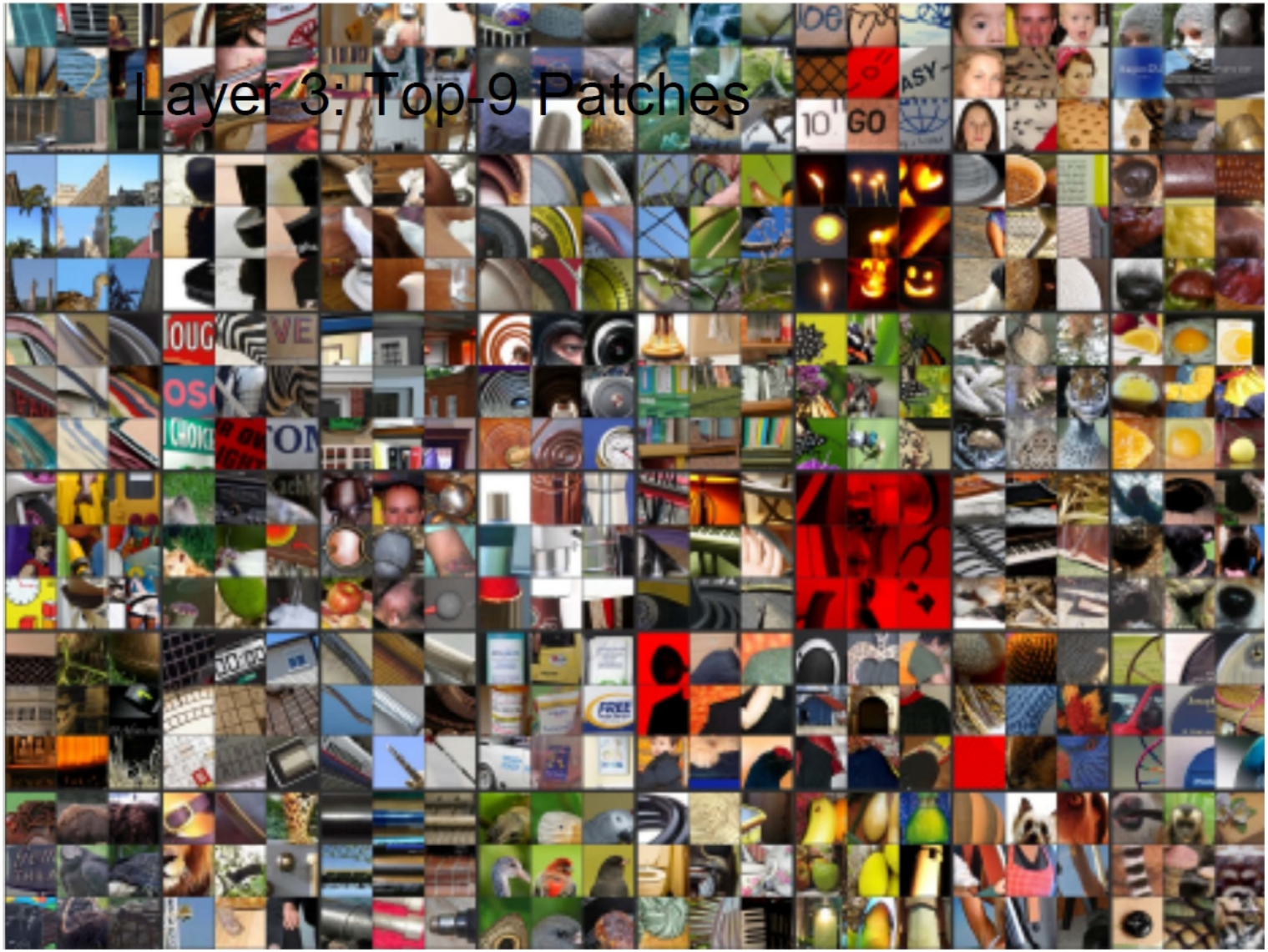
Layer 2 Patches



Layer 2 Patches

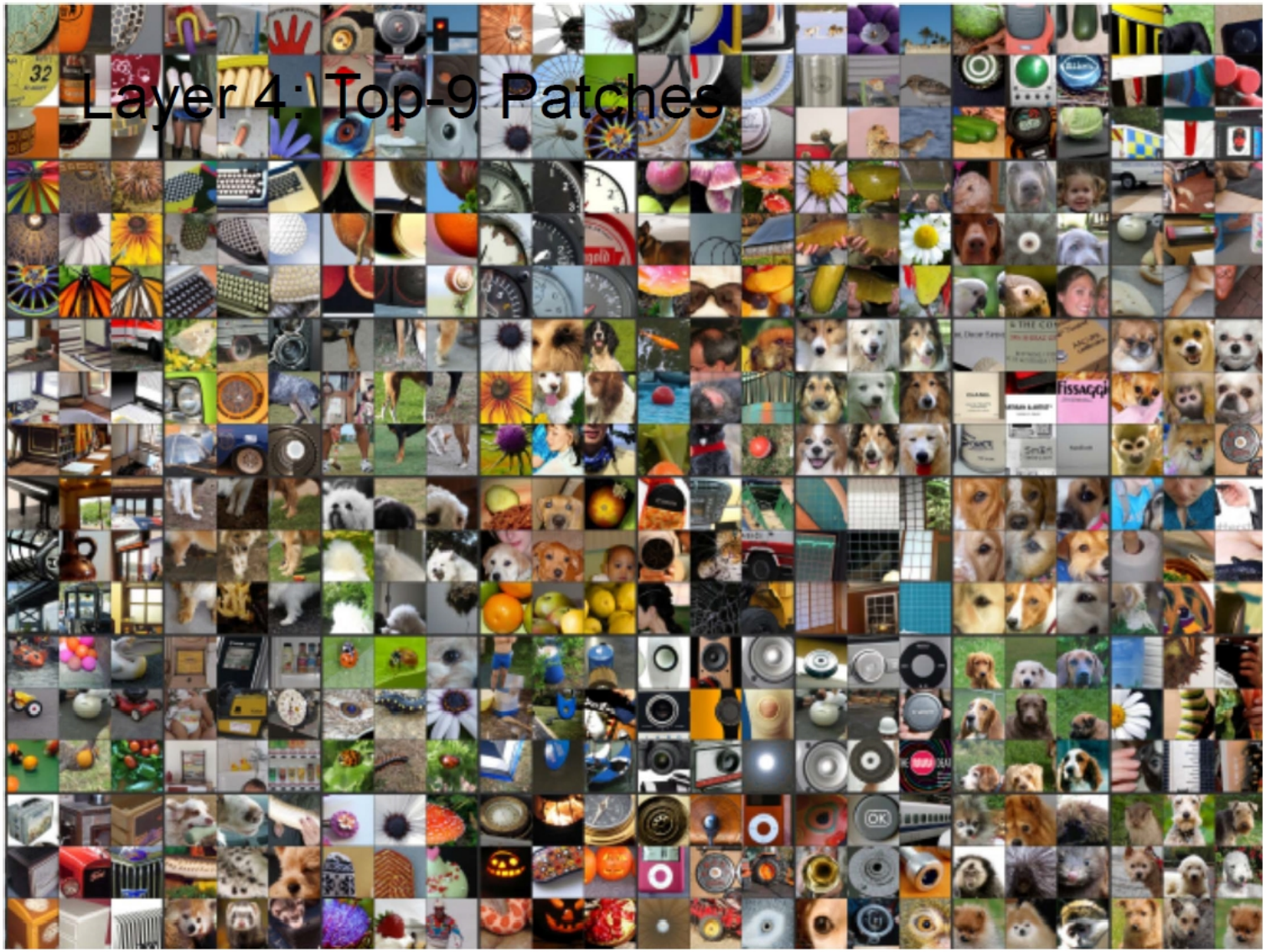


Layer 3 Patches



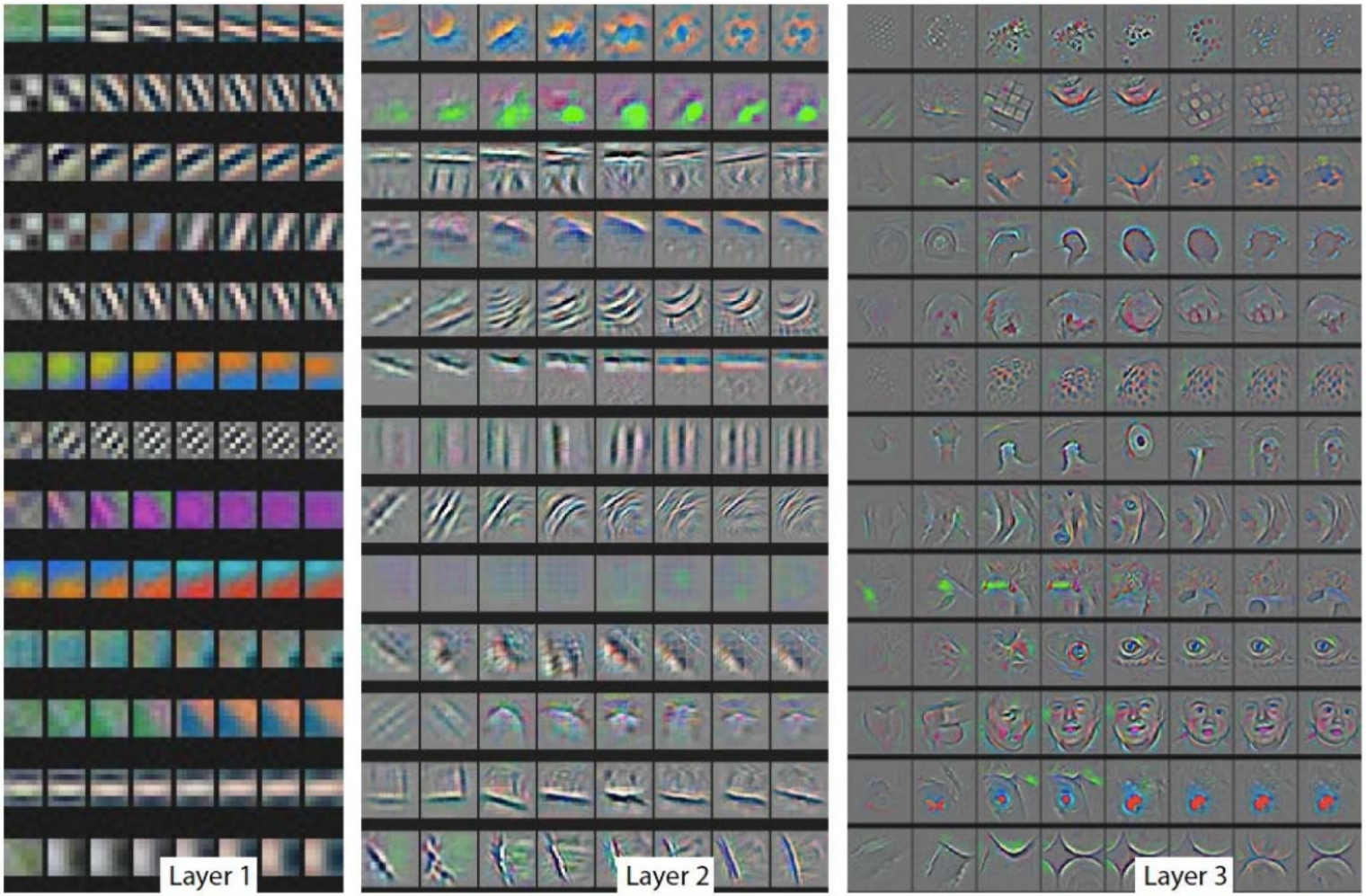
Layer 3 Patches

Layer 4 Patches

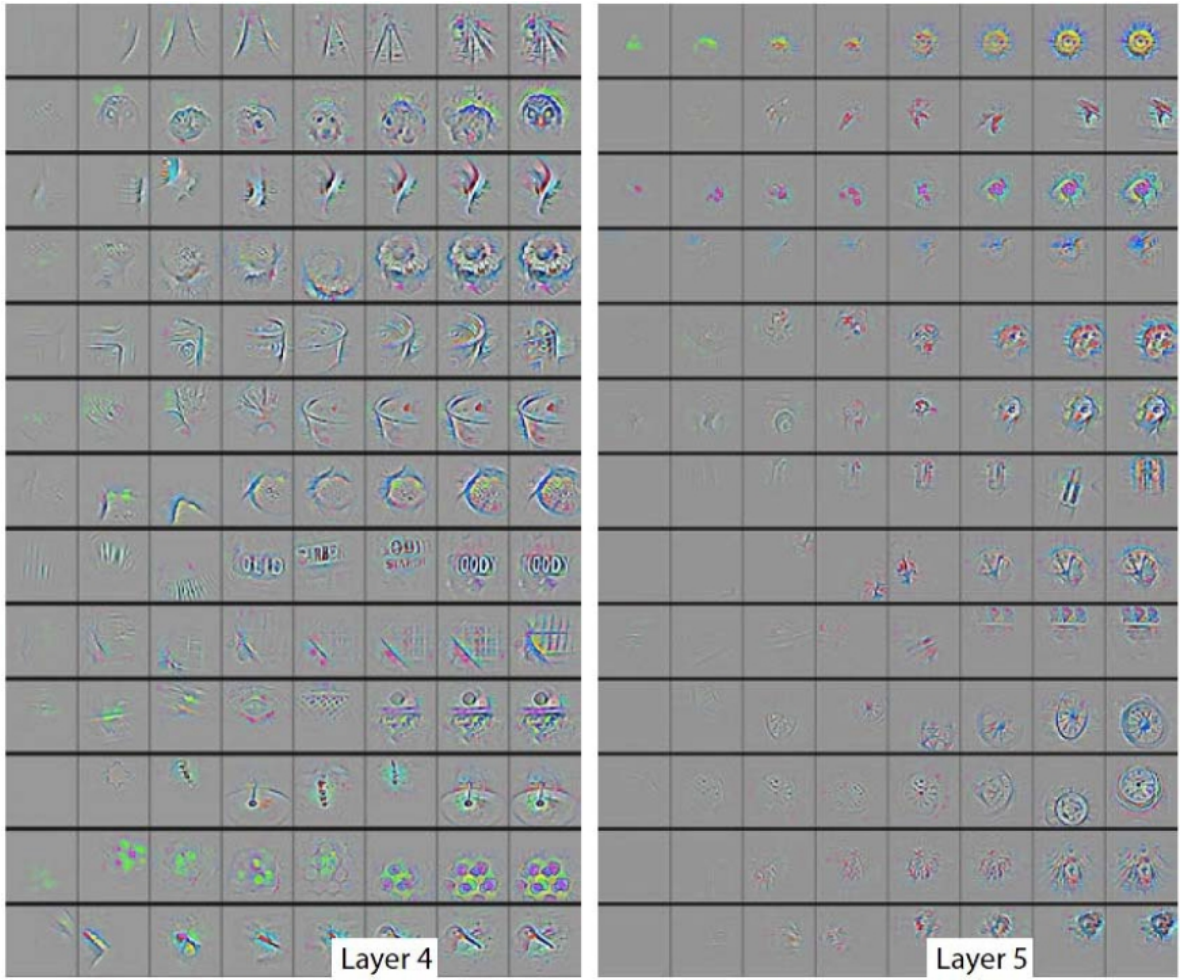


Layer 4 Patches

Evolution of Filters



Evolution of Filters

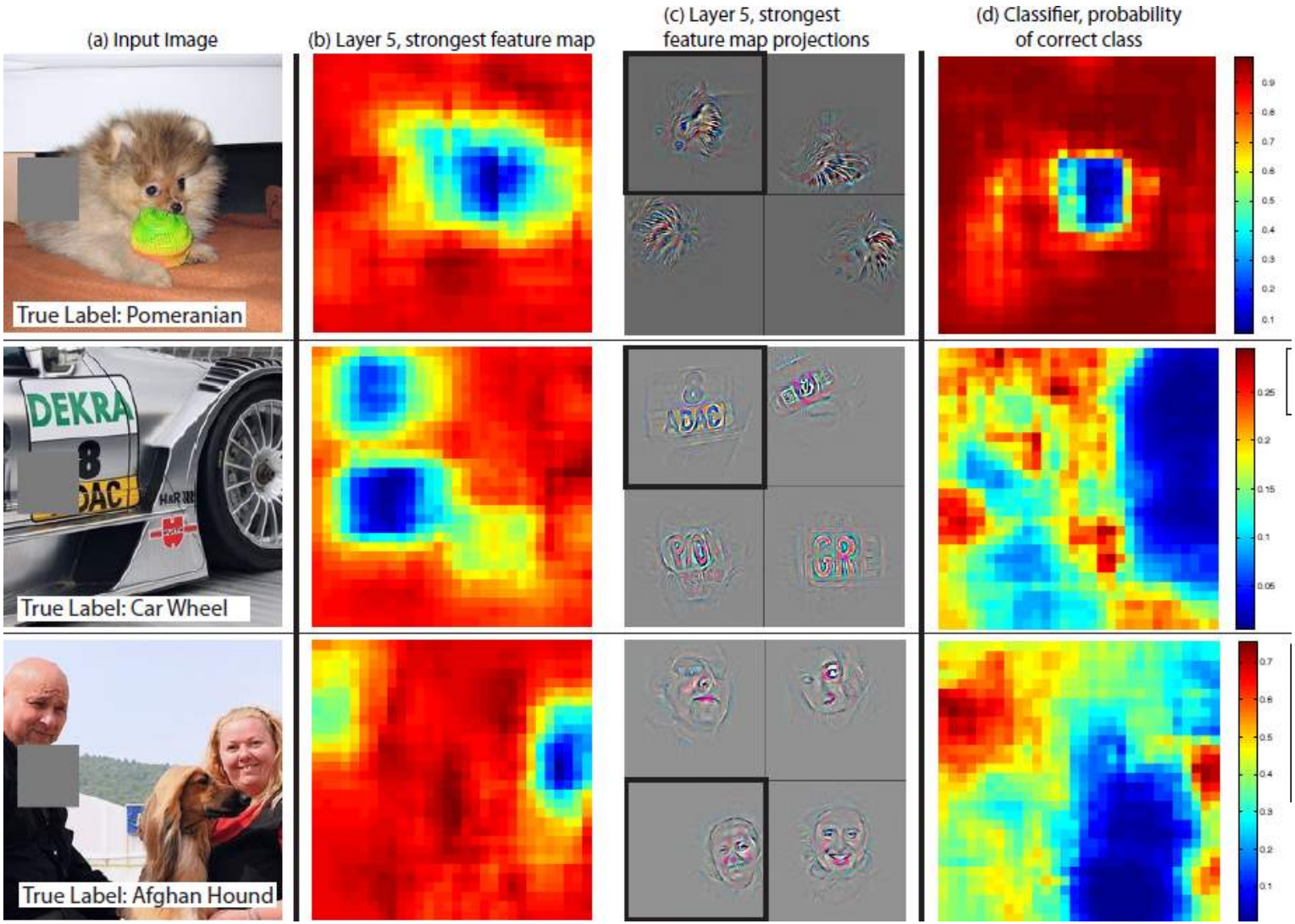


Caveat?

Occlusion Experiments

- Zeiler and Fergus also used feature visualizations to see if network really identified the object or depended on context
- Approach:Occlude images at different locations and visualize feature activations and classifier confidence

Occlusion Experiments



Class Saliency Visualization

Image Specific Class Saliency

- We want to visualize the spatial support of a particular class in an image
  - Given: Image I_0, a class c, a convnet score $S_c(I)$ function (un-normalized log probabilities)
  - Goal: Rank pixels of I_0 according to their influence on score $S_c(I_0)$
- Motivating case: Linear model for class c

$$S_c(I) = \mathbf{w}_c^T I + b_c$$

- Simple: Magnitude of w_i determines influence of pixel i

Image Specific Class Saliency

- In convnets $S_c(I)$ is a highly non-linear function of I
- How do we determine influence in this case?
- Useful hack: Just work with the first order approximation:

$$S_c(I) \approx \frac{\partial S_c}{\partial I}\Big|_{I_0} I + b$$

- $\mathbf{w} = \frac{\partial S_c}{\partial I}\Big|_{I_0}$ is the gradient evaluated at I_0
- Magnitude of w_i determines pixel influence (take max if multiple color channels)

Visualizing Gradients

- If we occlude pixels denoted by black pixels in original image, we won't mess up the network's prediction

Visualizing Gradients

- If we occlude pixels that represent the class spatial support, we will!

Generating an Image

Class Model Visualization

- Another approach: Numerically generate an image for a class
- Let $S_c(I)$ be the score function for class c:
  - $S_c(I)$: Forward pass through convnet and output before softmax
- Problem: Find a $L2$ regularize image I:

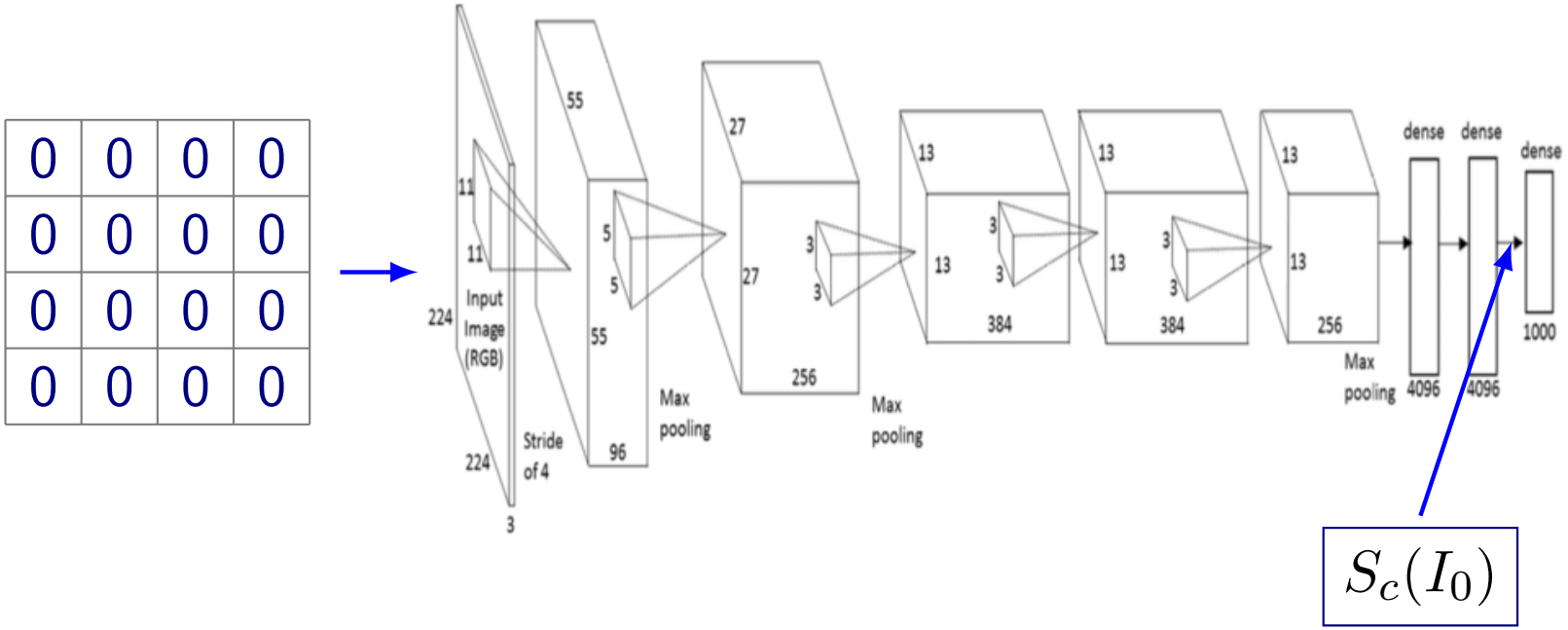
$$\arg\max_I S_c(I) - \lambda \|I\|_2^2$$

- Question: How do we find an *image*?

Class Model Visualization

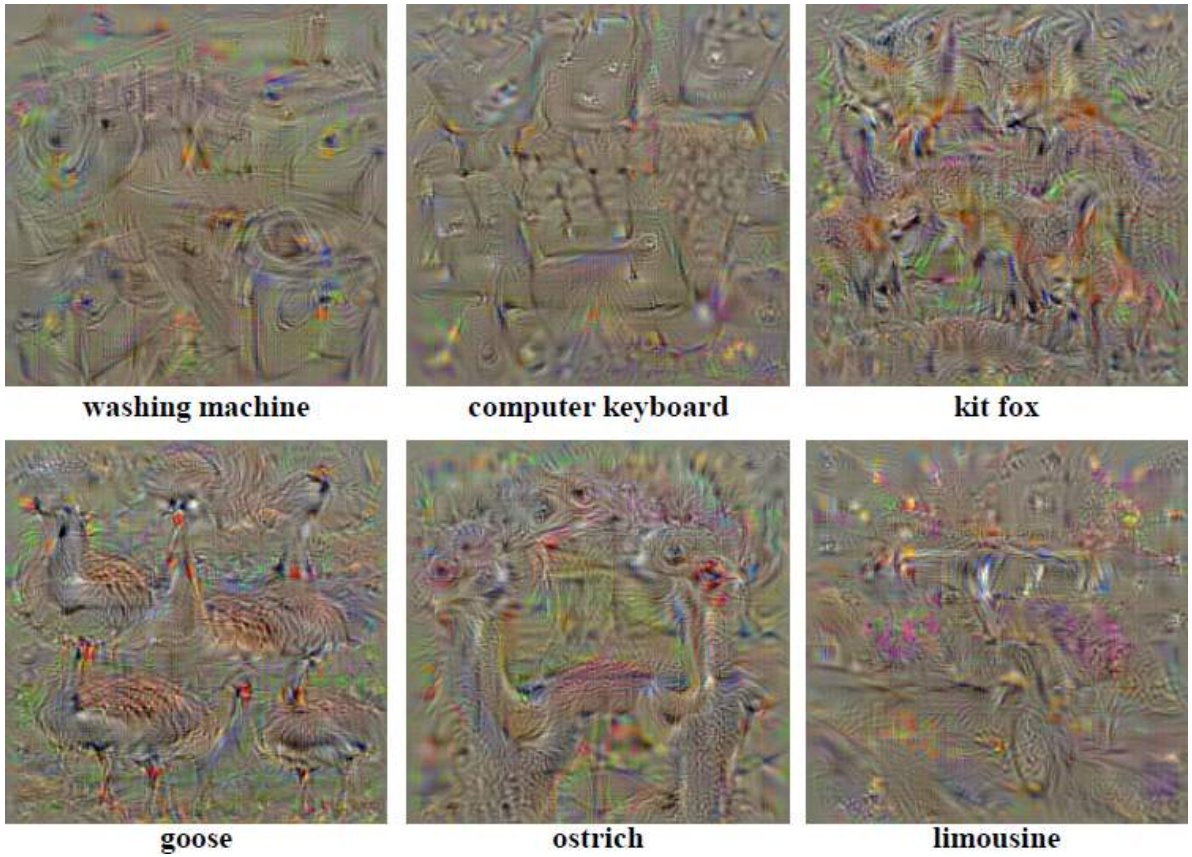
- Just do backpropagation!
  - Earlier: Used backpropagation to update weights
  - Now: Fix the network, pass image through network, obtain $S_c(I)$, go back and update pixels
- What should be the initial image?

Class Model Visualization



1. Fix network, input image I_0
2. Obtain $S_c(I_0)$ (Reminder: Unnormalized log probability for c)
3. Update image by backpropagation
4. Repeat till convergence

Class Model Visualization:Examples



Changing the Regularizer

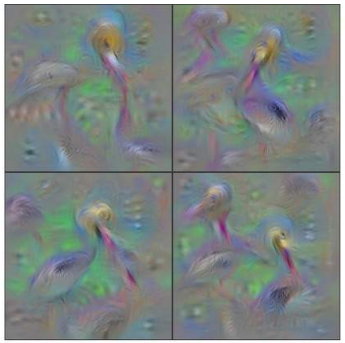
- We had the optimization problem:

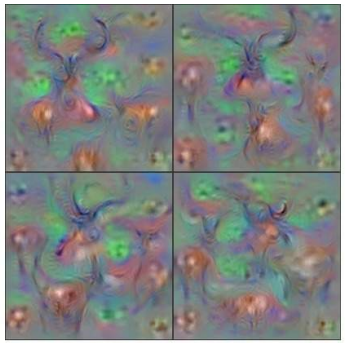
$$\arg\max_I S_c(I) - \lambda \|I\|_2^2$$

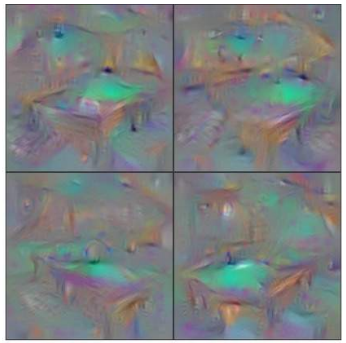
- Problem:Introduces high frequency artifacts
- Another regularizer: Blur I at each update
- Also clip pixel values with small contributions

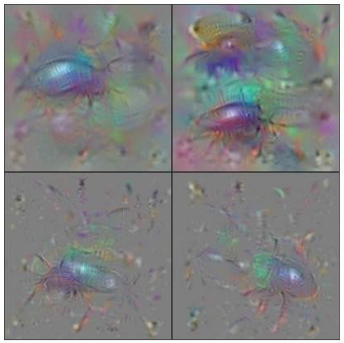
Class Model Visualization with Blur

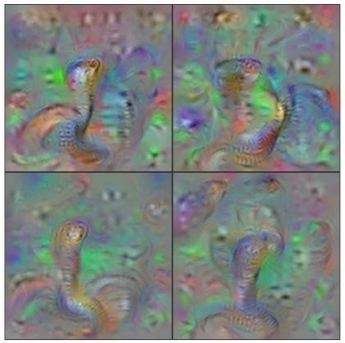

Flamingo

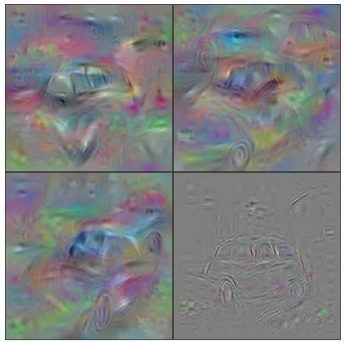

Pelican

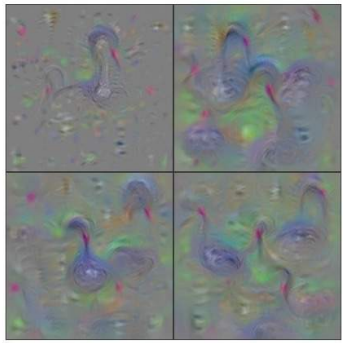

Hartebeest


Billiard Table


Ground Beetle


Indian Cobra


Station Wagon


Black Swan

Works for one neuron!

- We had the optimization problem:

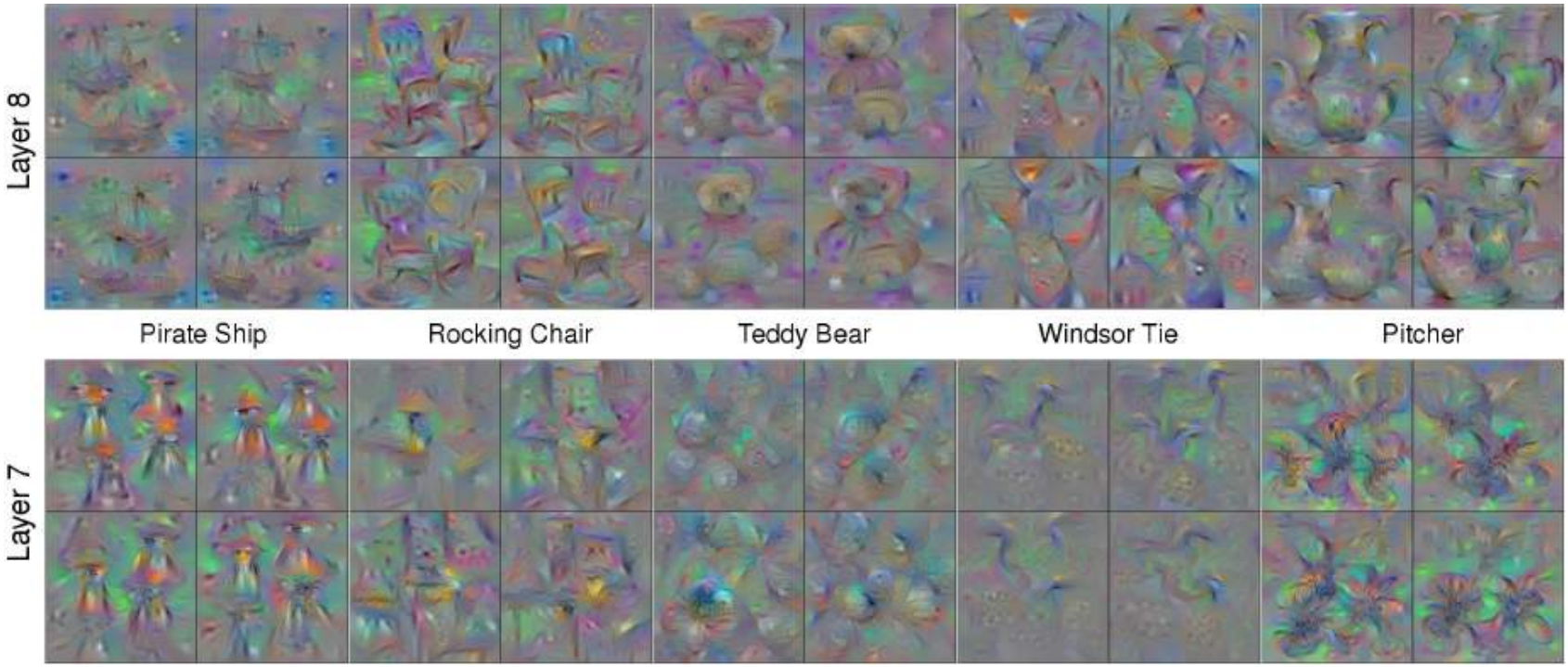
$$\arg\max_I S_c(I) - \lambda \|I\|_2^2$$

- Can do this for any neuronal activation

$$\arg\max_I a_i(I) - \lambda R(I)$$

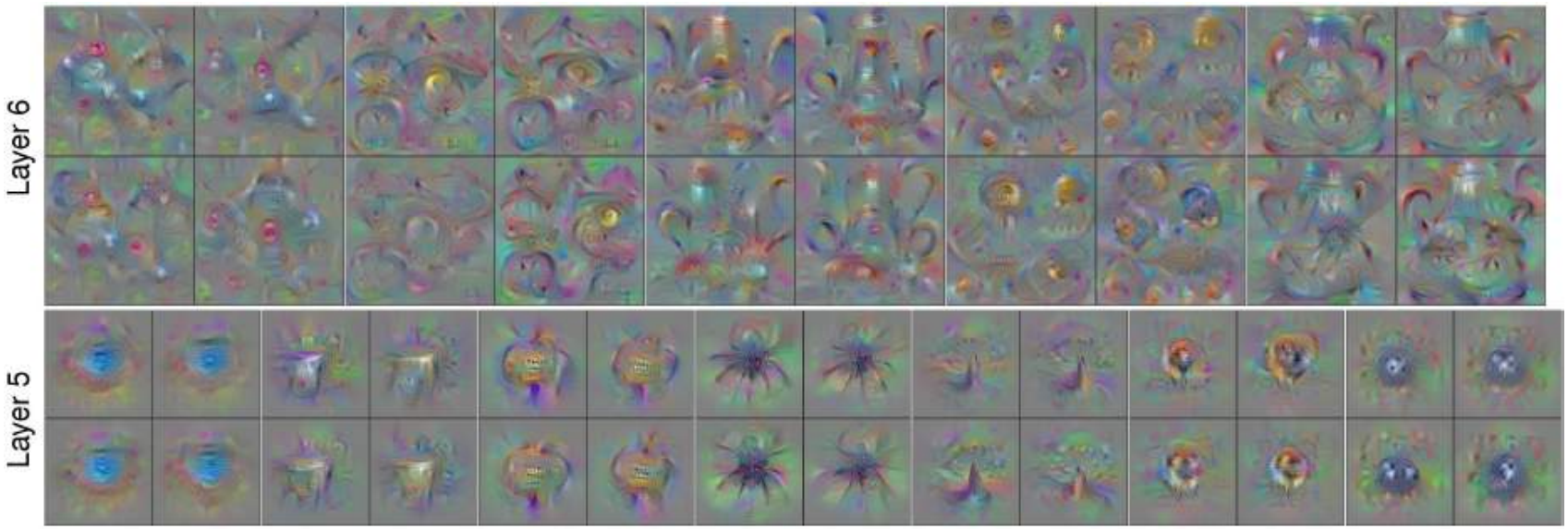
- Process remains the same
- Can use this to probe what each neuron likes!

Example Features

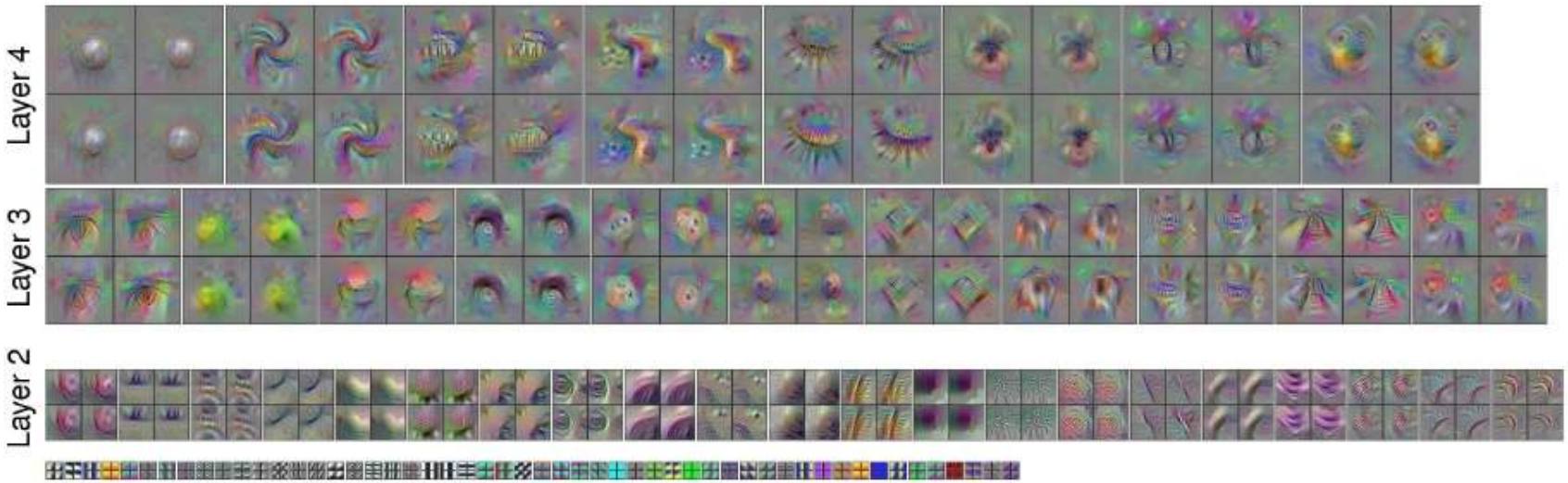


- Layer 8 visualization is a class model (same as before)

Example Features



Example Features



https://www.youtube.com/watch?v=AgkfIQ4IGaM

A Short Digression

Yet Another Image Optimization

- First we saw:

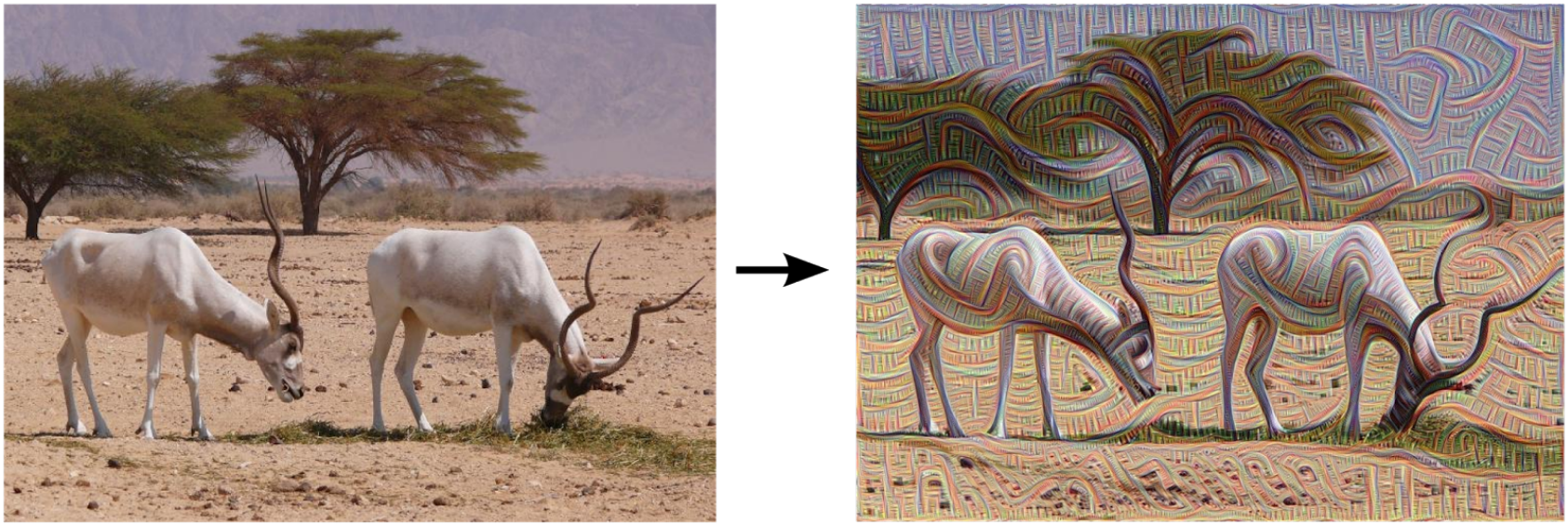
$$\arg\max_I S_c(I) - \lambda \|I\|_2^2$$

- Next for any internal neuron:

$$\arg\max_I a_i(I) - \lambda R(I)$$

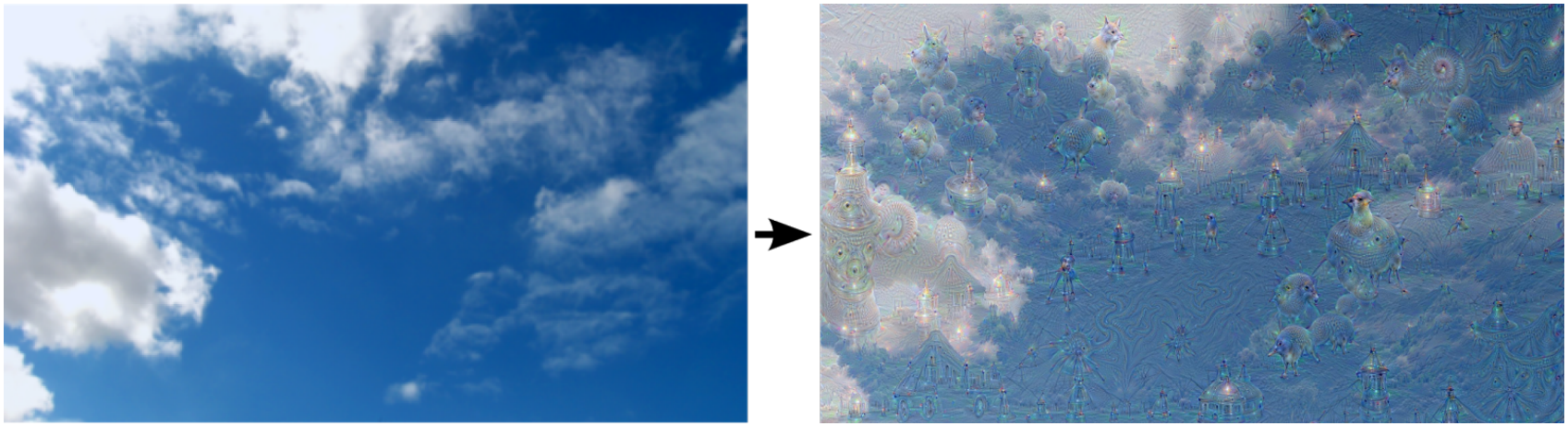
- We input a zero image, and optimized to generate an image that maximized for class score, or neuronal activation
- We could input a real image, and optimize it over activations in an entire layer!
- What would happen?

Deep Dream



Lower layers detect edges etc., on optimization such features will get boosted up

Deep Dream



"Admiral Dog!" "The Pig-Snail" "The Camel-Bird" "The Dog-Fish"

Slightly higher layers start to overinterpret shapes in images

Deep Dream

Slightly higher layers start to overinterpret shapes in images

Iterating

The input for these were noise images!

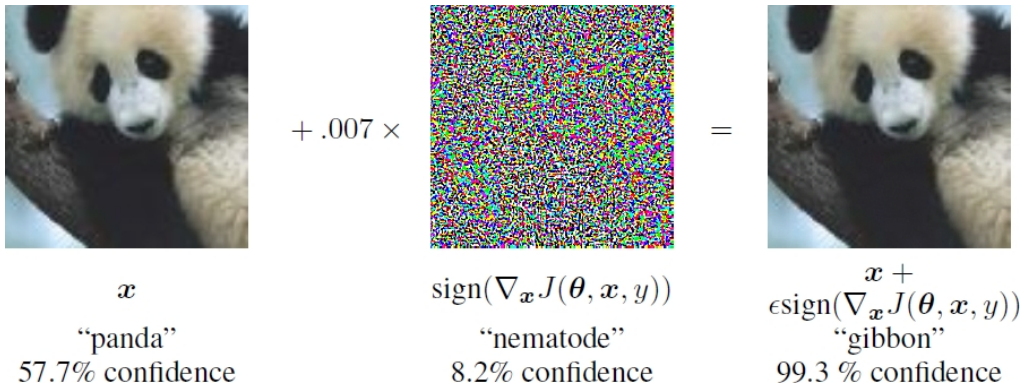
Video: Grocery store trip:
https://www.youtube.com/watch?v=DgPaCWJL7XI

Yet Another Image Optimization

- We have already seen an example of optimizing an image

$$\arg\min_{\Delta \mathbf{x}} \|\Delta \mathbf{x}\| \textbf{ s.t. } f(\mathbf{x} + \Delta \mathbf{x}; \theta) = y_g$$

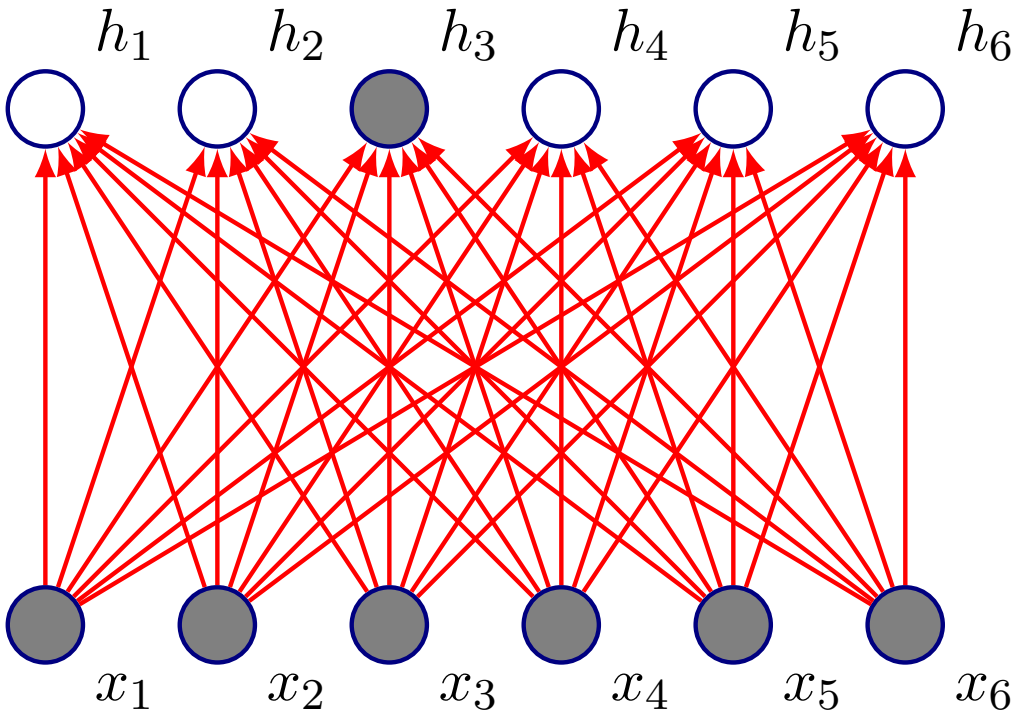
- Adversarial Examples! ($\Delta \mathbf{x}$ is an image)



Convolutions: Motivation

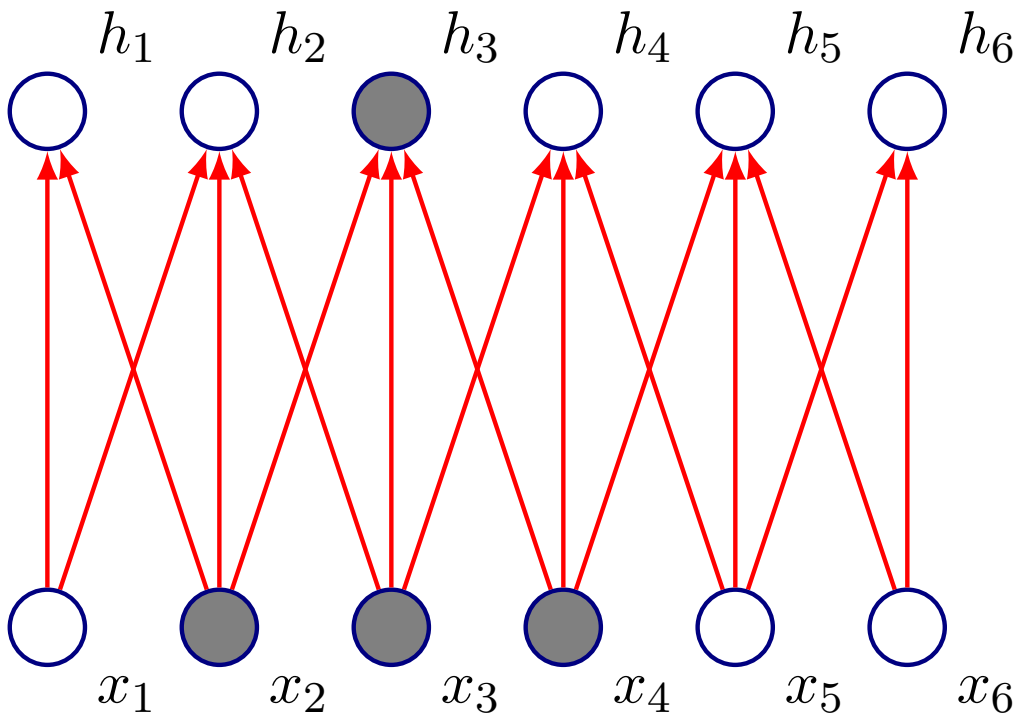
- Convolution leverages four ideas that can help ML systems:
  - Sparse interactions
  - Parameter sharing
  - Equivariant representations
  - Ability to work with inputs of variable size
- **Sparse Interactions**
  - Plain Vanilla NN ($y \in \mathbb{R}^n, x \in \mathbb{R}^m$): Need matrix multiplication $y = \mathbf{W}x$ to compute activations for each layer (every output interacts with every input)
  - Convolutional networks have *sparse interactions* by making kernel smaller than input
  - $\implies$ need to store fewer parameters, computing output needs fewer operations ($O(m \times n)$ versus $O(k \times n)$)

Motivation: Sparse Connectivity



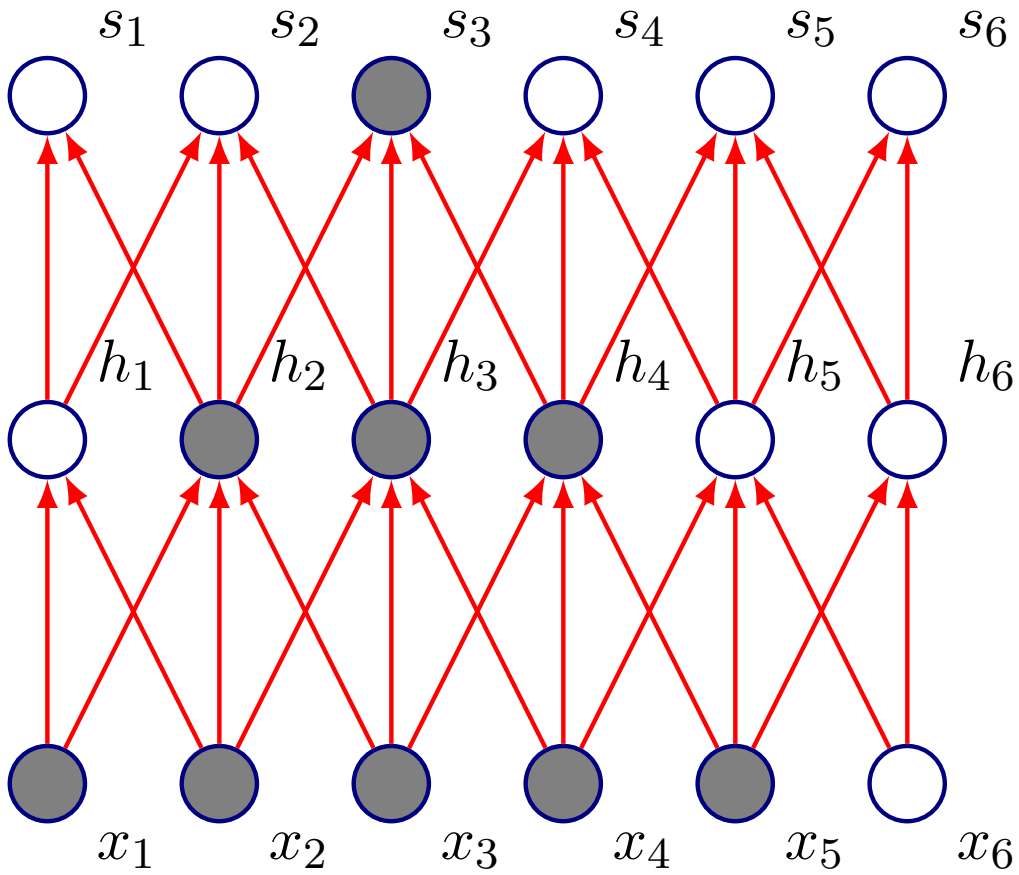
- Fully connected network: h_3 is computed by full matrix multiplication with no sparse connectivity

Motivation: Sparse Connectivity



- Kernel of size 3, moved with stride of 1
- h_3 only depends on x_2, x_3, x_4

Motivation: Sparse Connectivity



- Connections in CNNs are sparse, but units in deeper layers are connected to all of the input (larger receptive field sizes)

Motivation: Parameter Sharing

- Plain vanilla NN: Each element of $\mathbf{W}$ is used exactly once to compute output of a layer
- In convolutional networks, parameters are *tied*: weight applied to one input is tied to value of a weight applied elsewhere
- Same kernel is used throughout the image, so instead learning a parameter for each location, only a set of parameters is learnt
- Forward propagation remains unchanged $O(k \times n)$
- Storage improves dramatically as $k \ll m, n$

Motivation: Equivariance

- Let's first formally define convolution:

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

- In Convolutional Network terminology x is referred to as **input**, w as the **kernel** and s as the **feature map**
- Discrete Convolution:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$$

- Convolution is commutative, thus:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n)$$

Aside

- The latter is usually more straightforward to implement in ML libraries (less variation in range of valid values of m and n)
- Neither are usually used in practice in Neural Networks
- Libraries implement *Cross Correlation*, same as convolution, but without flipping the kernel

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n)$$

Motivation: Equivariance

- Equivariance: f is equivariant to g if $f(g(\mathbf{x})) = g(f(\mathbf{x}))$
- The form of parameter sharing used by CNNs causes each layer to be equivariant to translation
- That is, if g is any function that translates the input, the convolution function is equivariant to g

Motivation: Equivariance

- Implication: While processing time series data, convolution produces a timeline that shows when different features appeared (if an event is shifted in time in the input, the same representation will appear in the output)
- Images: If we move an object in the image, its representation will move the same amount in the output
- This property is useful when we know some local function is useful everywhere (e.g. edge detectors)
- Convolution is not equivariant to other operations such as change in scale or rotation

Pooling: Motivation

- Pooling helps the representation become slightly *invariant* to small translations of the input
- Reminder: Invariance: $f(g(\mathbf{x})) = f(\mathbf{x})$
- If input is translated by small amount: values of most pooled outputs don't change

Pooling: Invariance

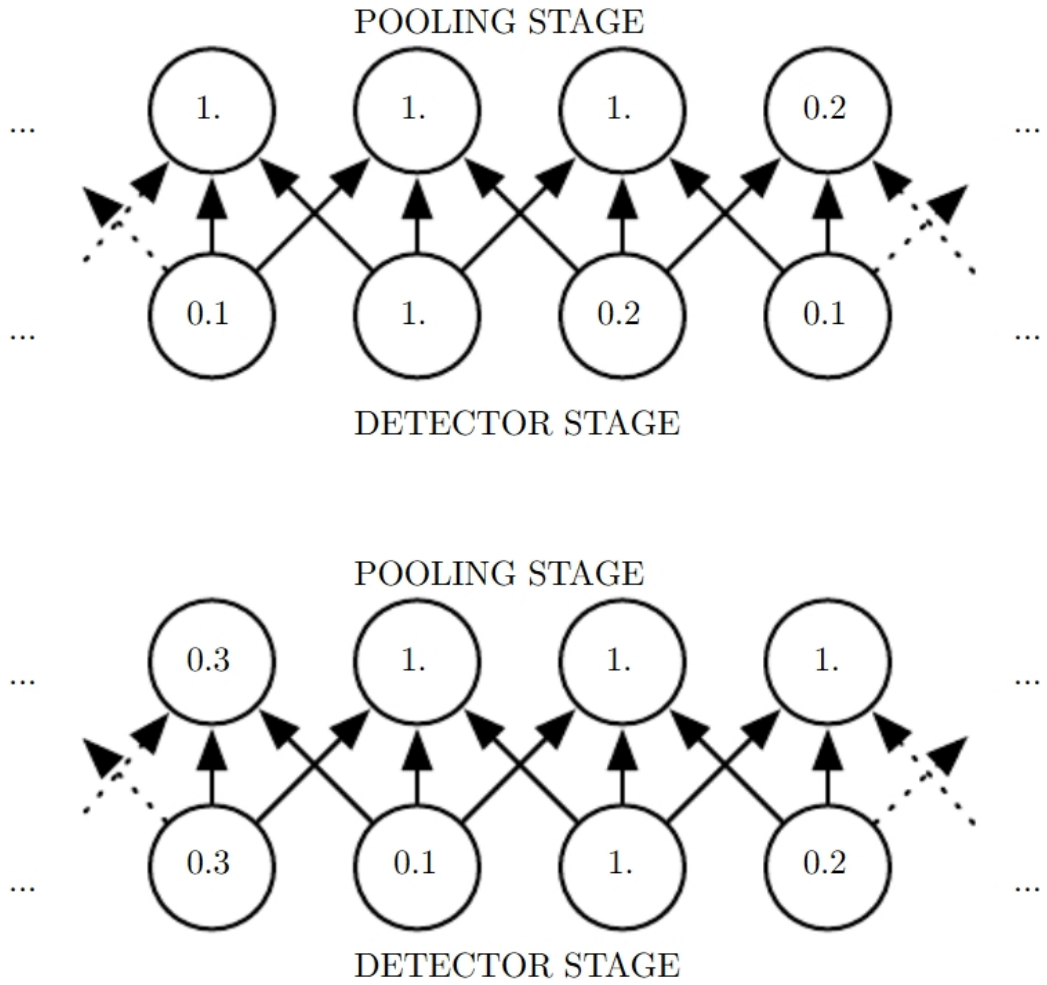


Figure: Goodfellow *et al.*

Pooling

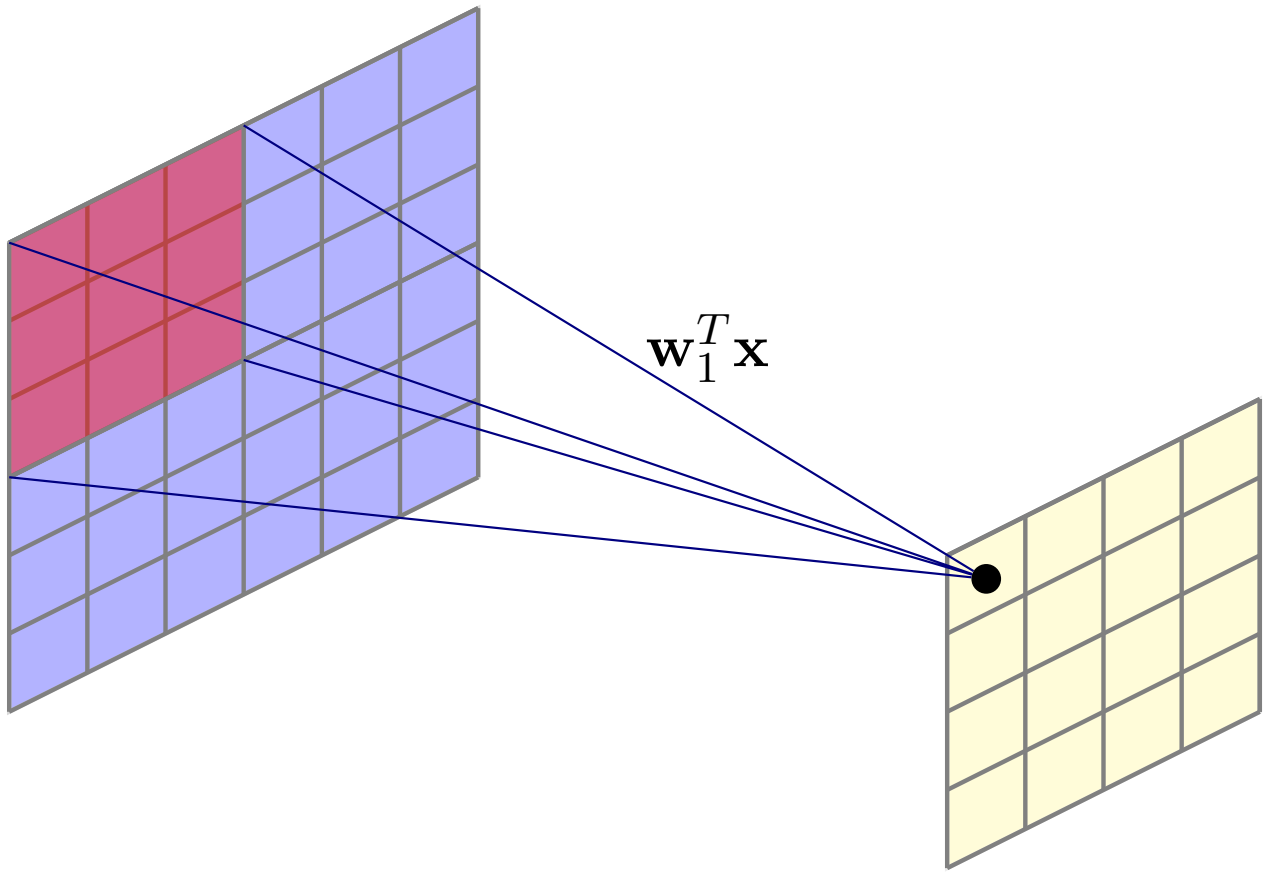
- Invariance to local translation can be useful if we care more about whether a certain feature is present rather than exactly where it is
- Pooling over spatial regions produces invariance to translation, what if we pool over separately parameterized convolutions?
- Features can learn which transformations to become invariant to (Example: Maxout Networks, Goodfellow *et al* 2013)
- One more advantage: Since pooling is used for downsampling, it can be used to handle inputs of varying sizes

Variations

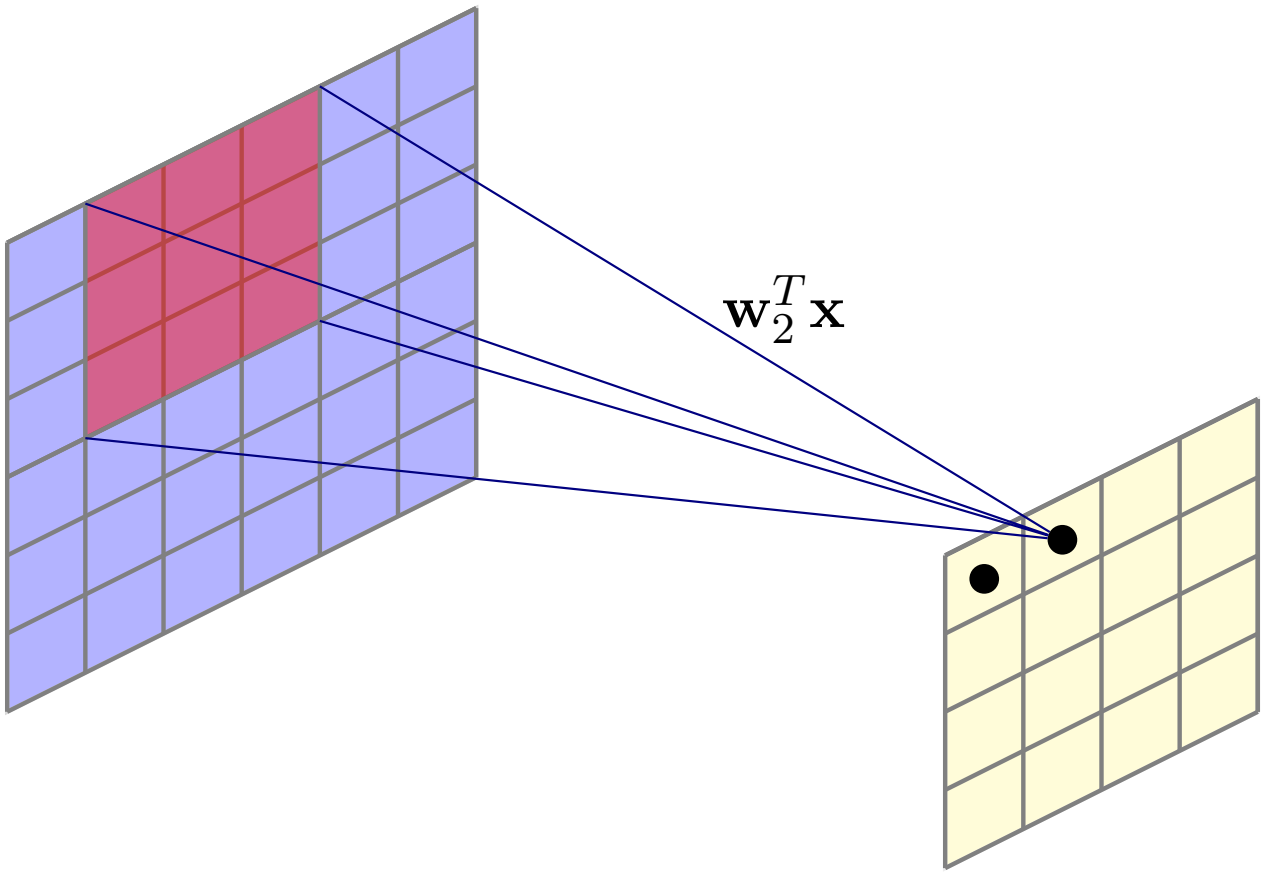
Locally Connected Layers

- In some applications: Feature should still be function of a small part of space, but might not occur throughout it
- Convolution: Have one kernel that we move across the grid to generate a feature map
- Unshared Convolution: Kernel is different at every location
- No parameter sharing!

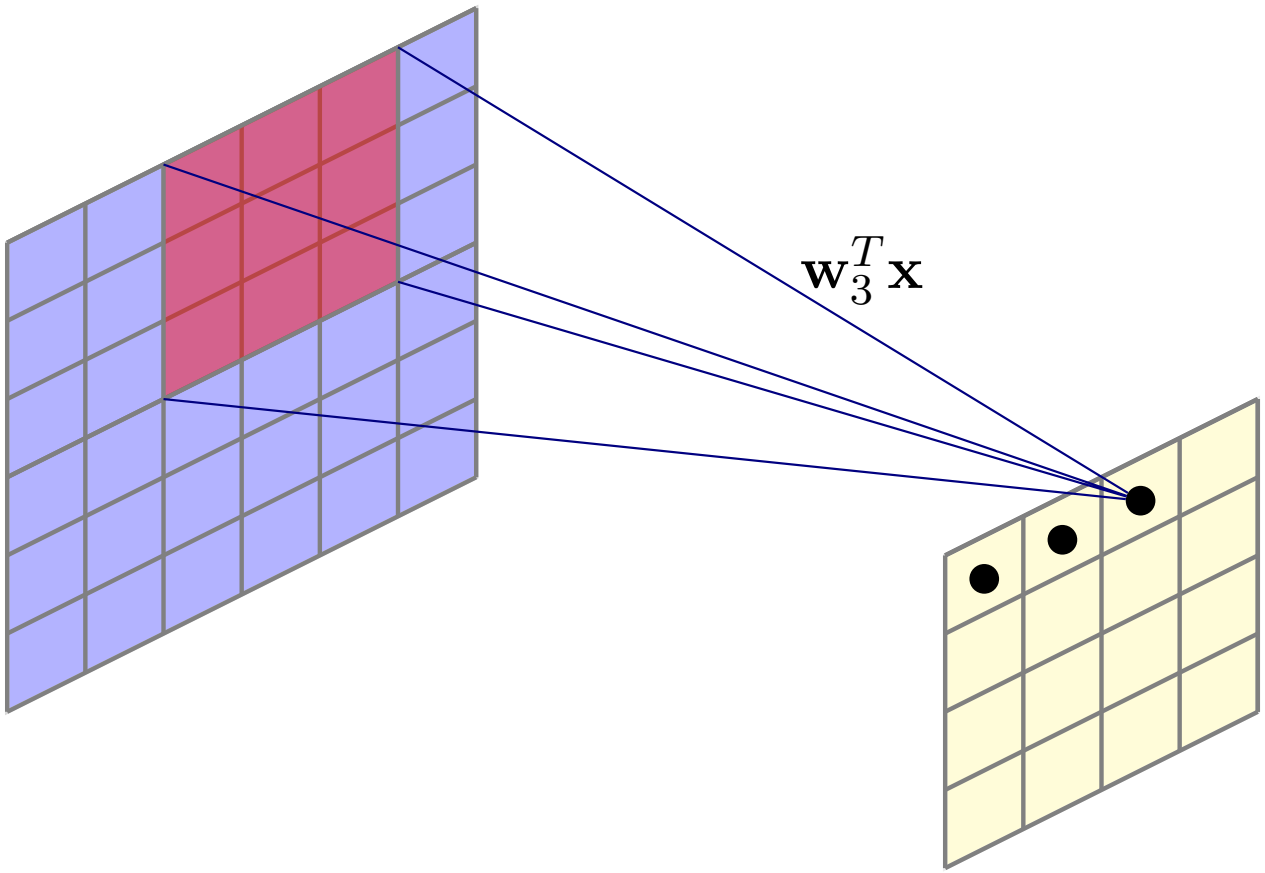
Locally Connected Layer



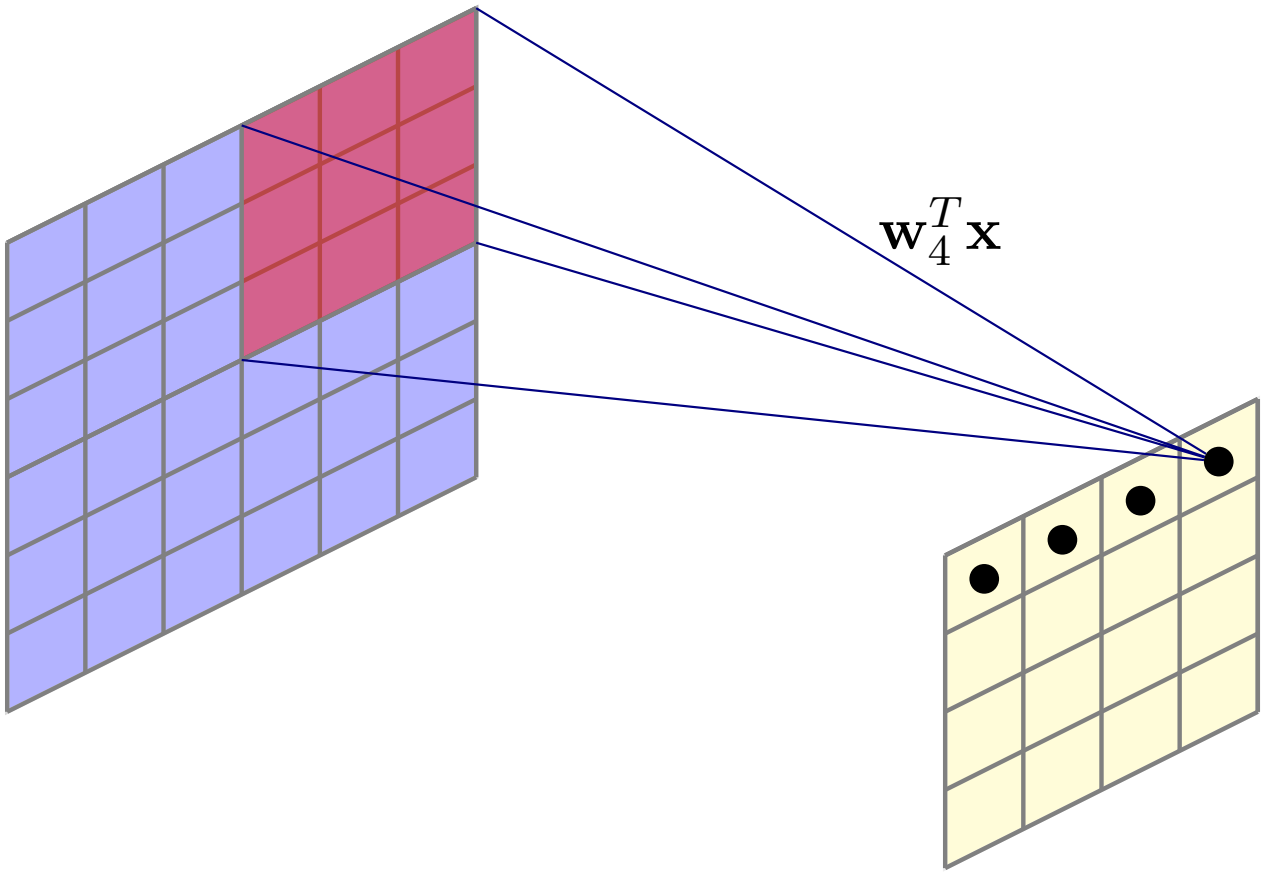
Locally Connected Layer



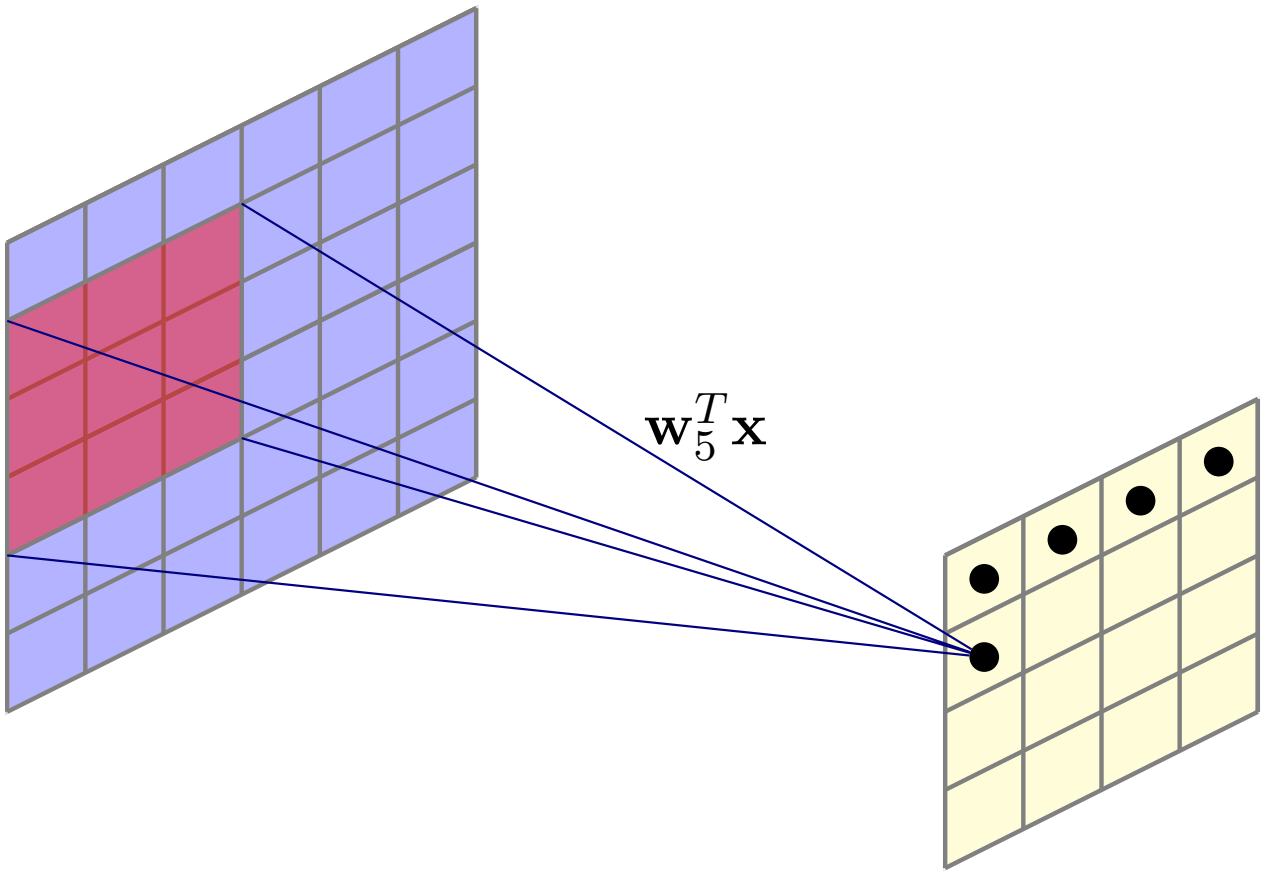
Locally Connected Layer



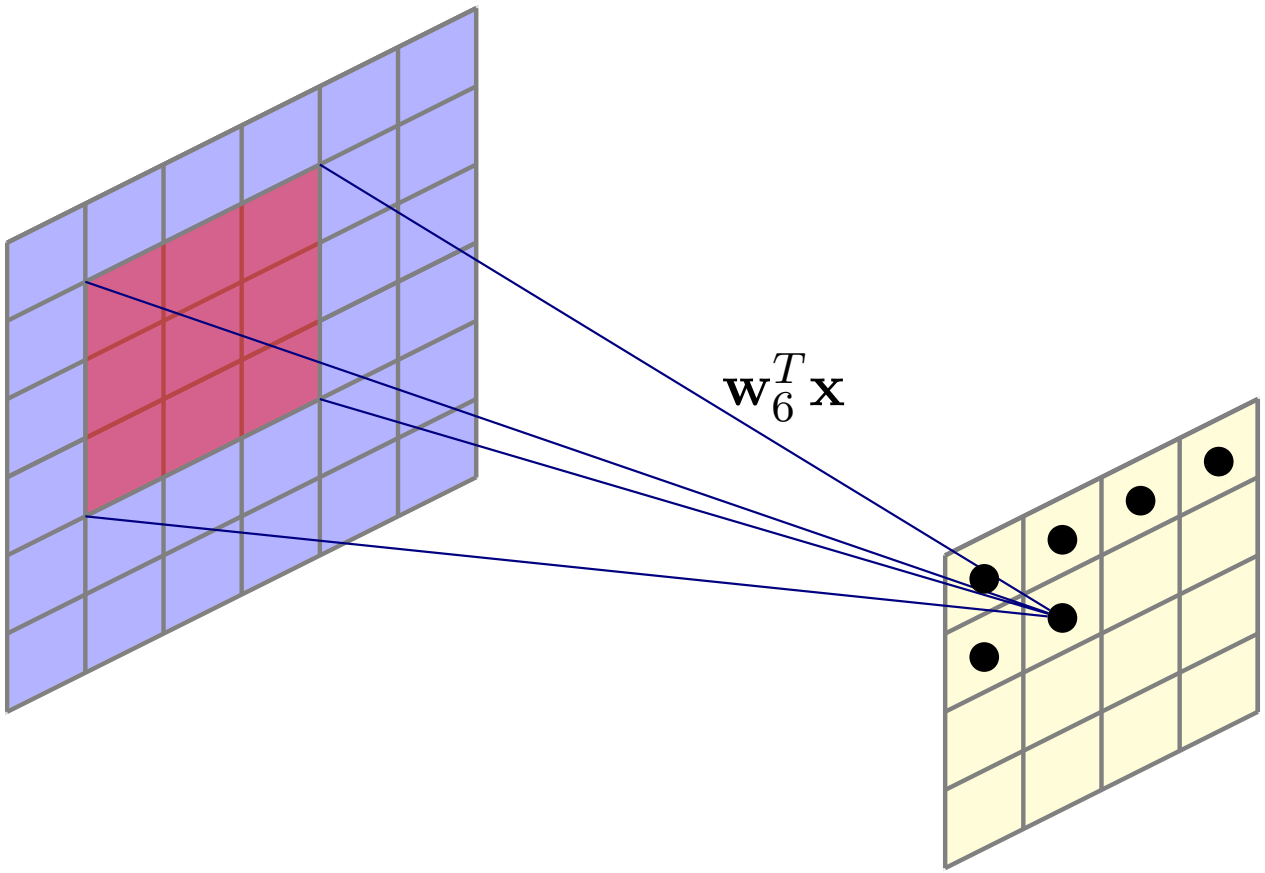
Locally Connected Layer



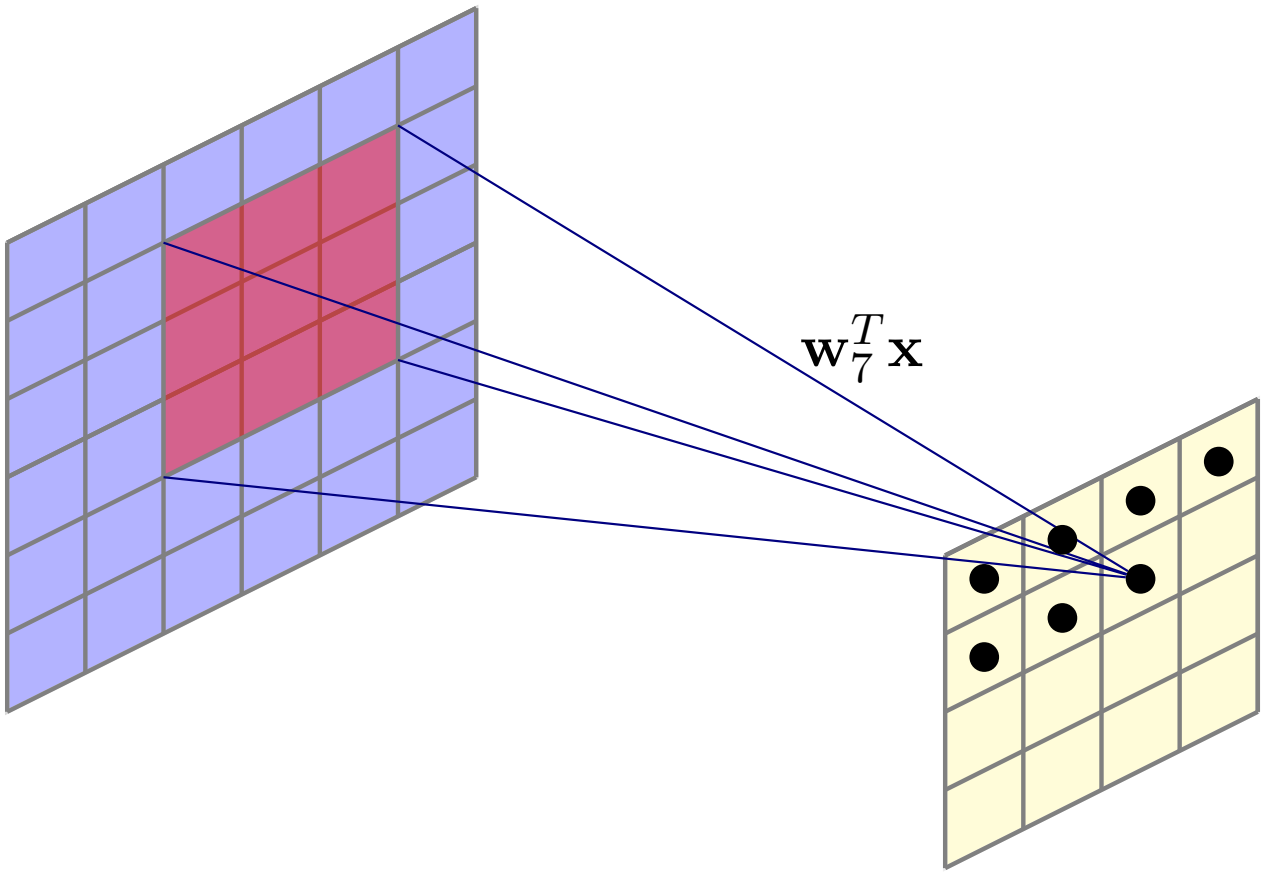
Locally Connected Layer



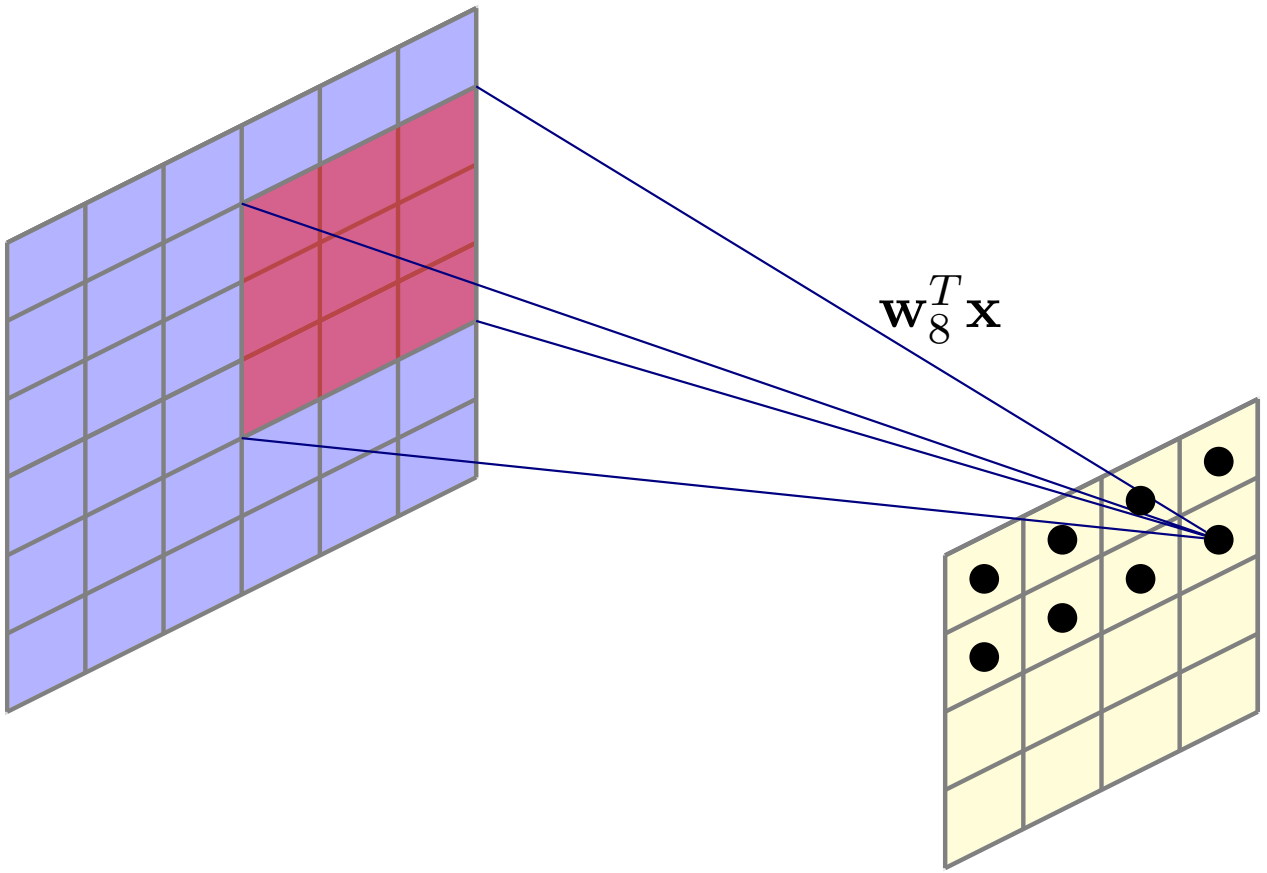
Locally Connected Layer



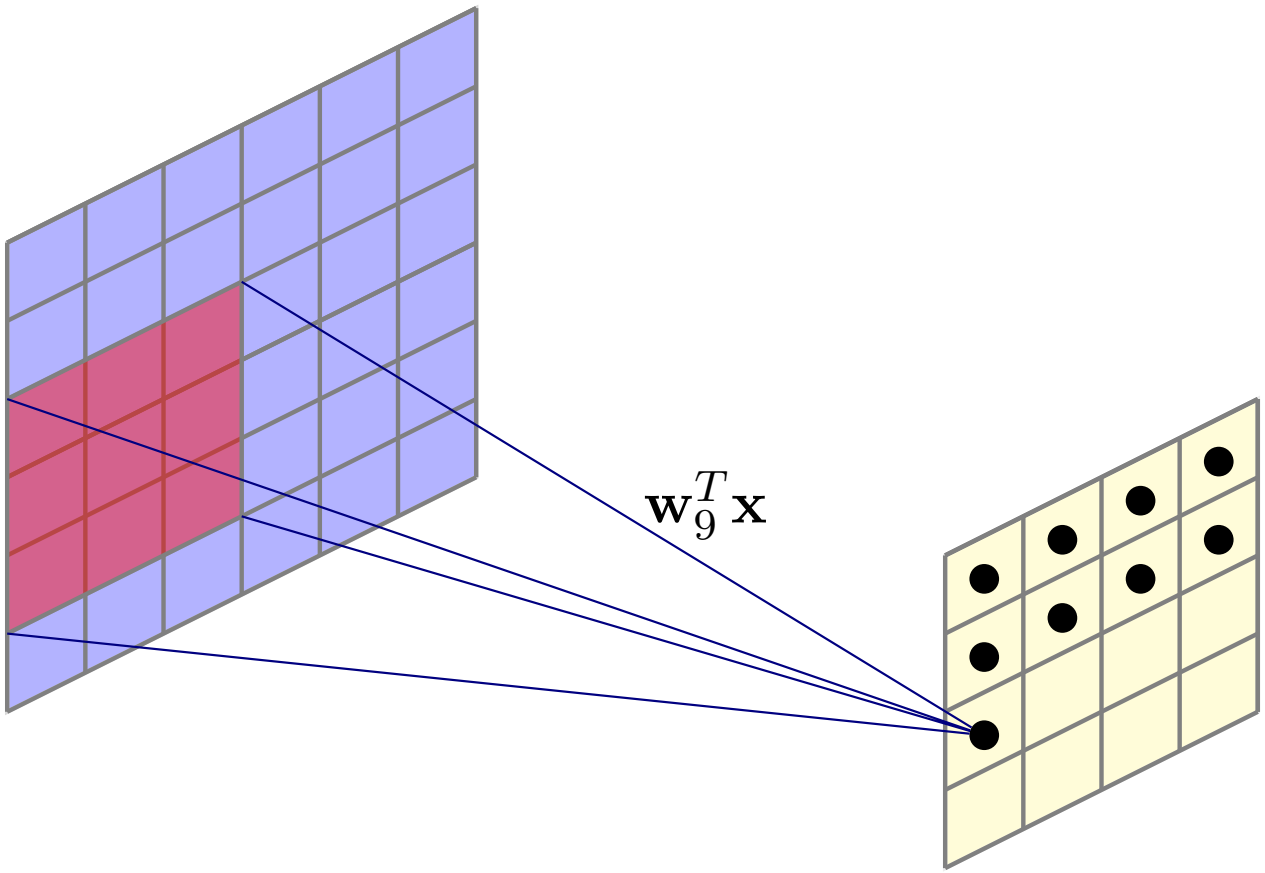
Locally Connected Layer



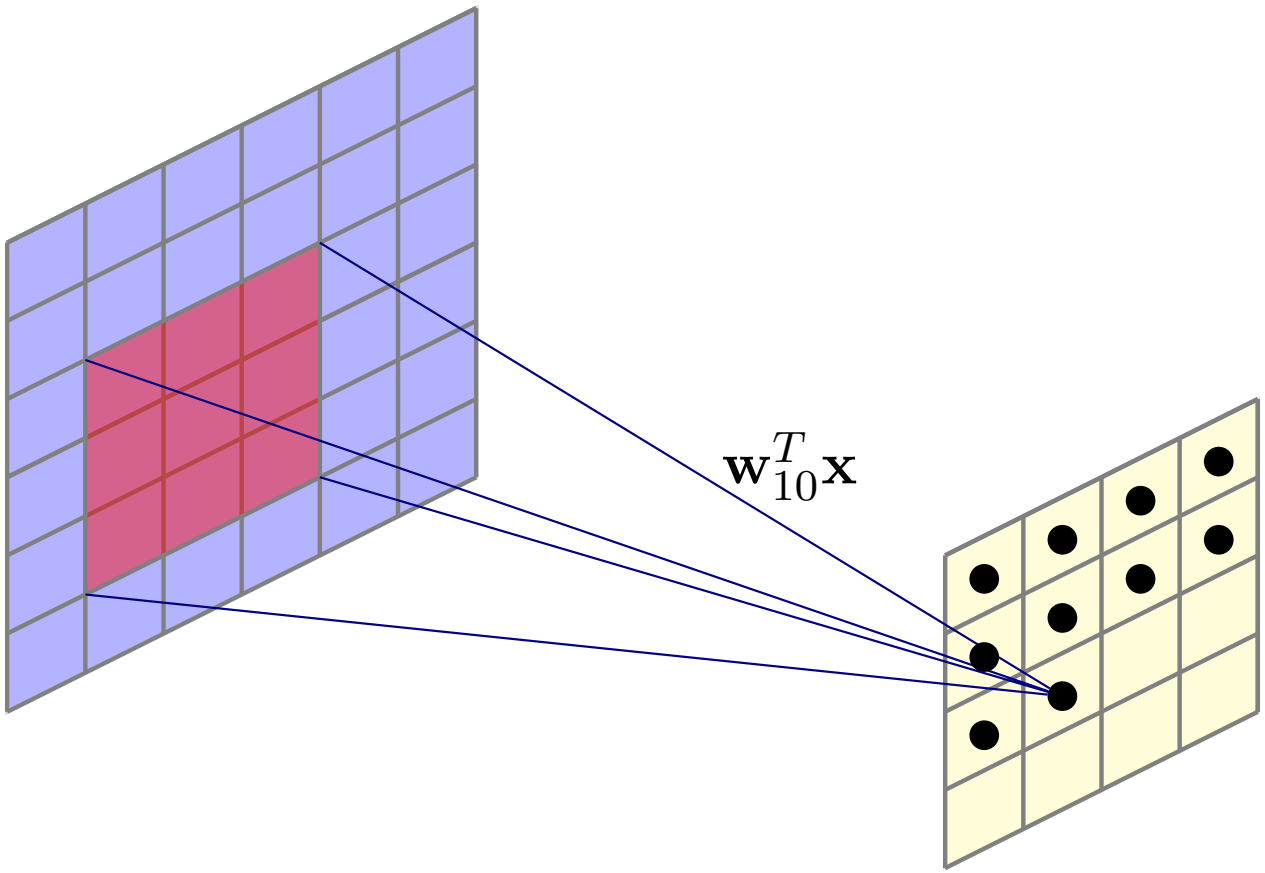
Locally Connected Layer



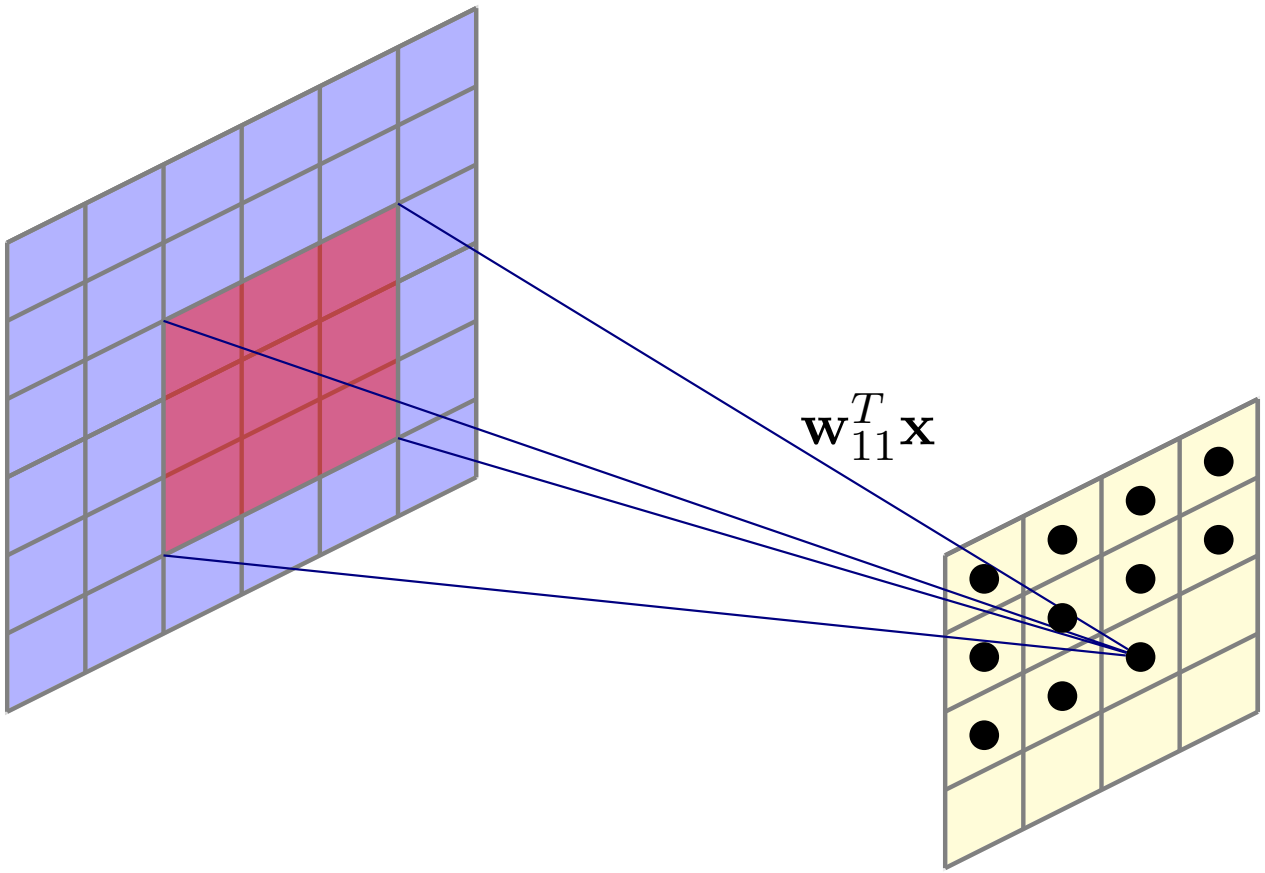
Locally Connected Layer



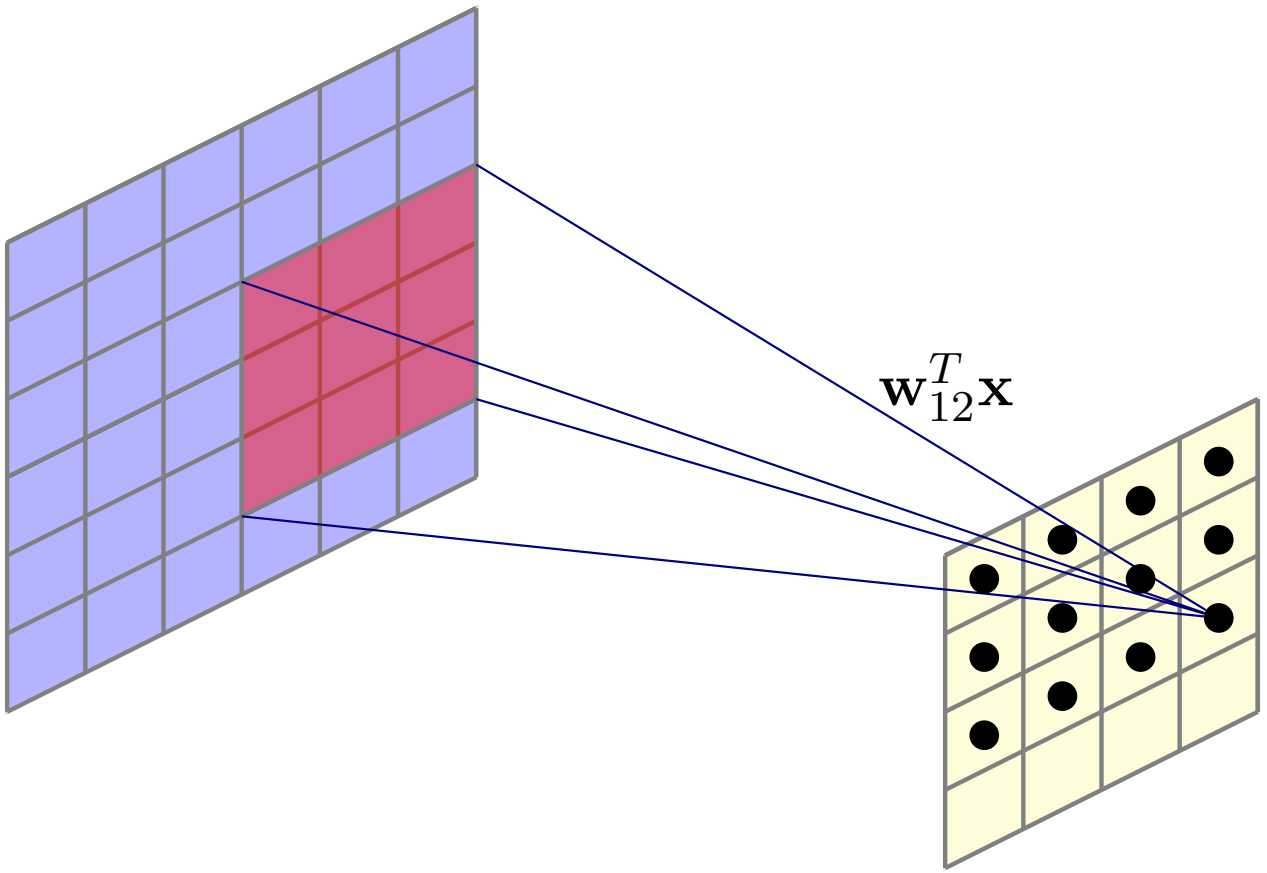
Locally Connected Layer



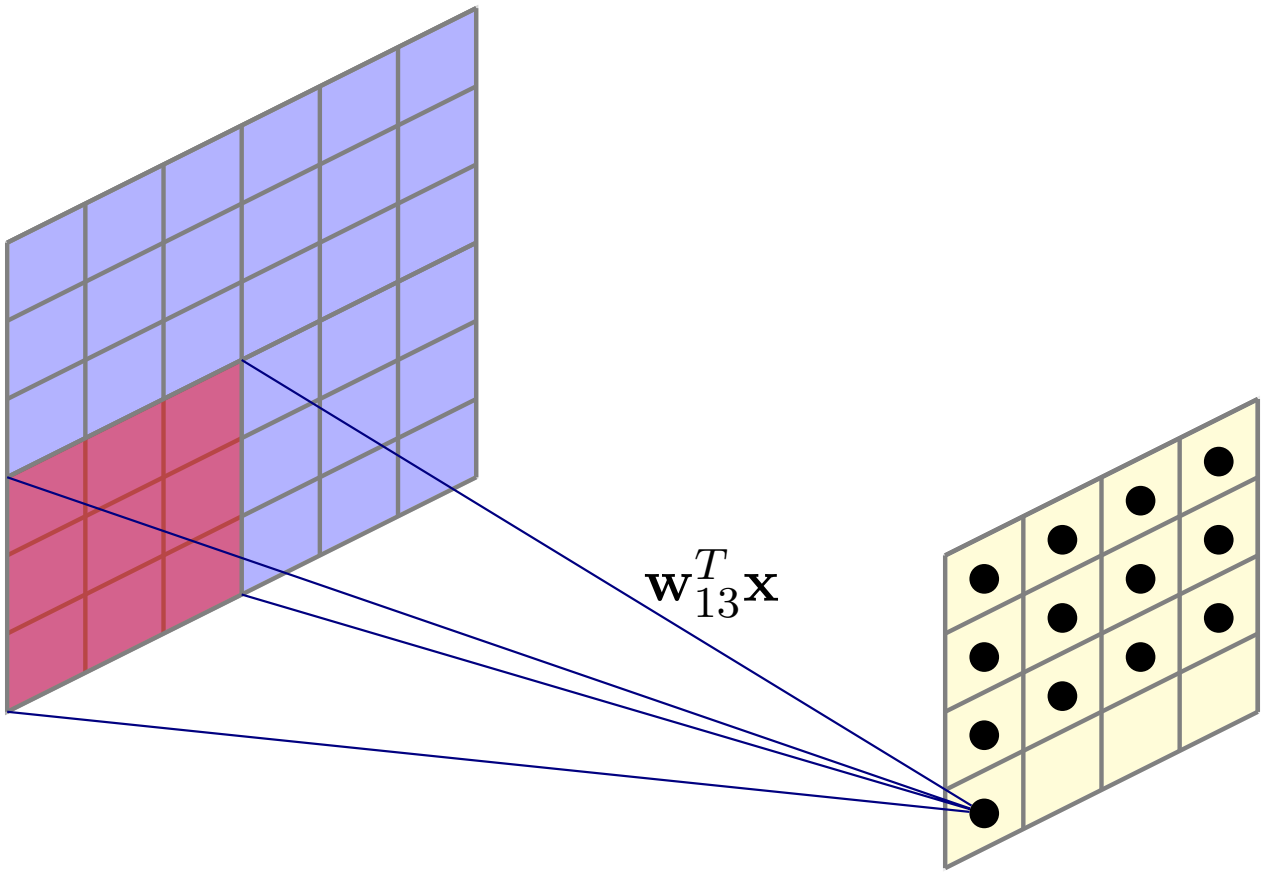
Locally Connected Layer



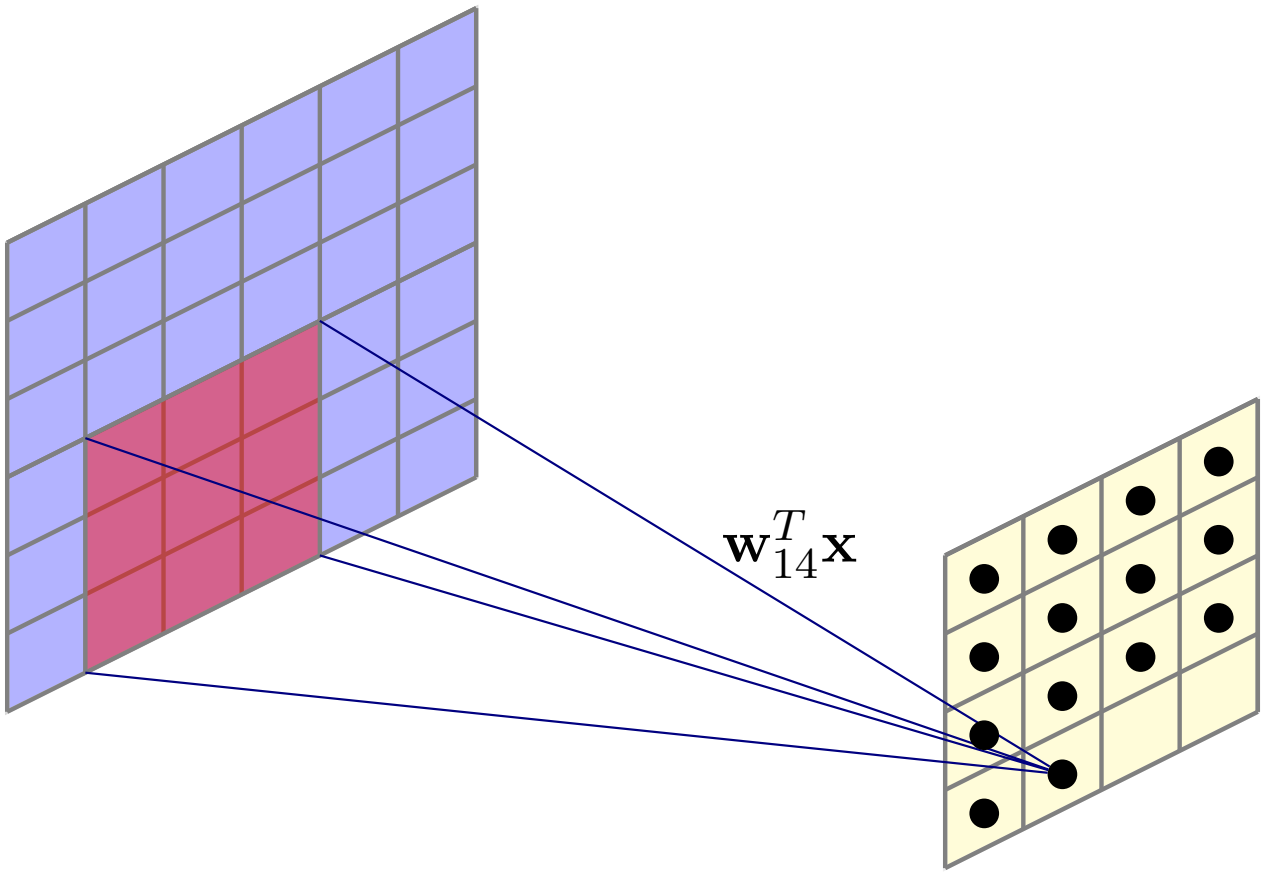
Locally Connected Layer



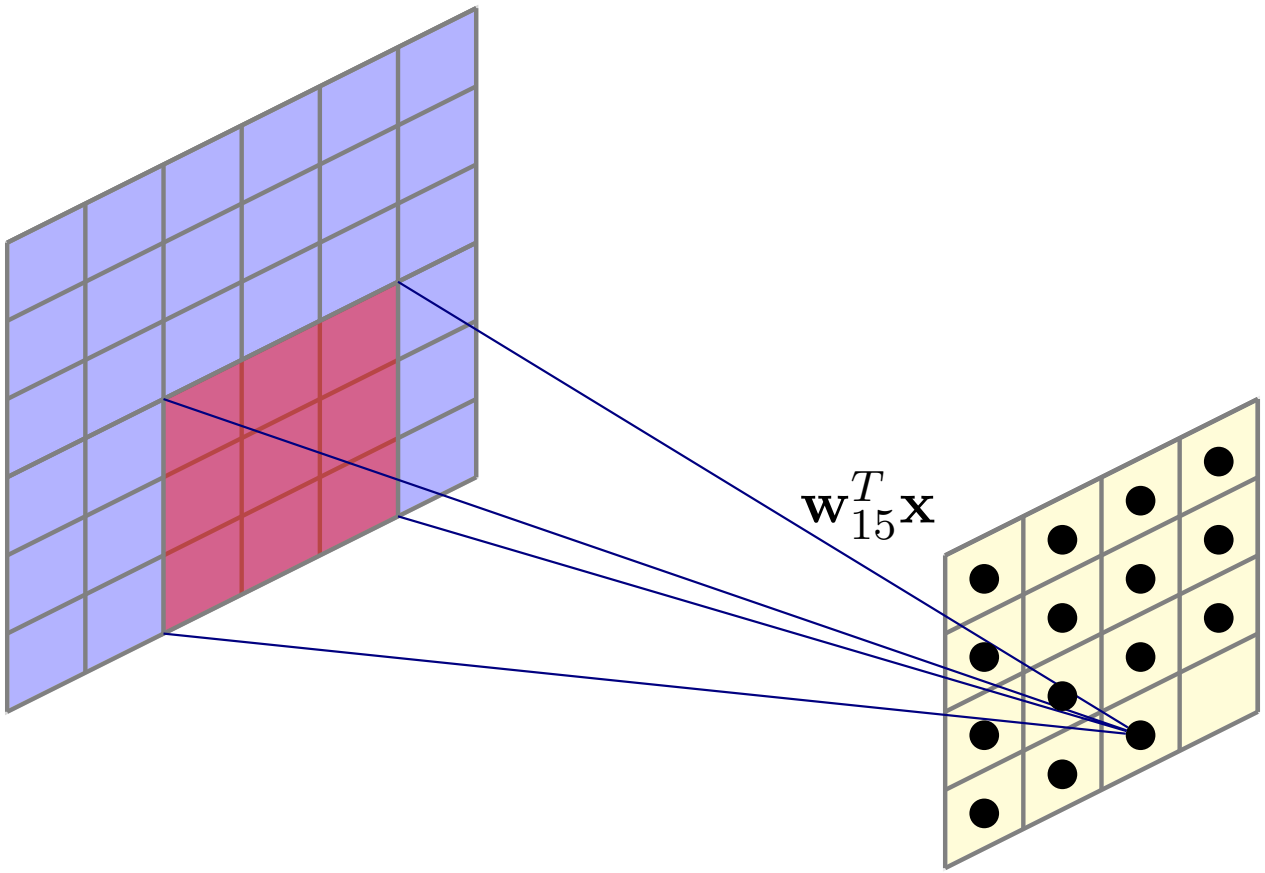
Locally Connected Layer



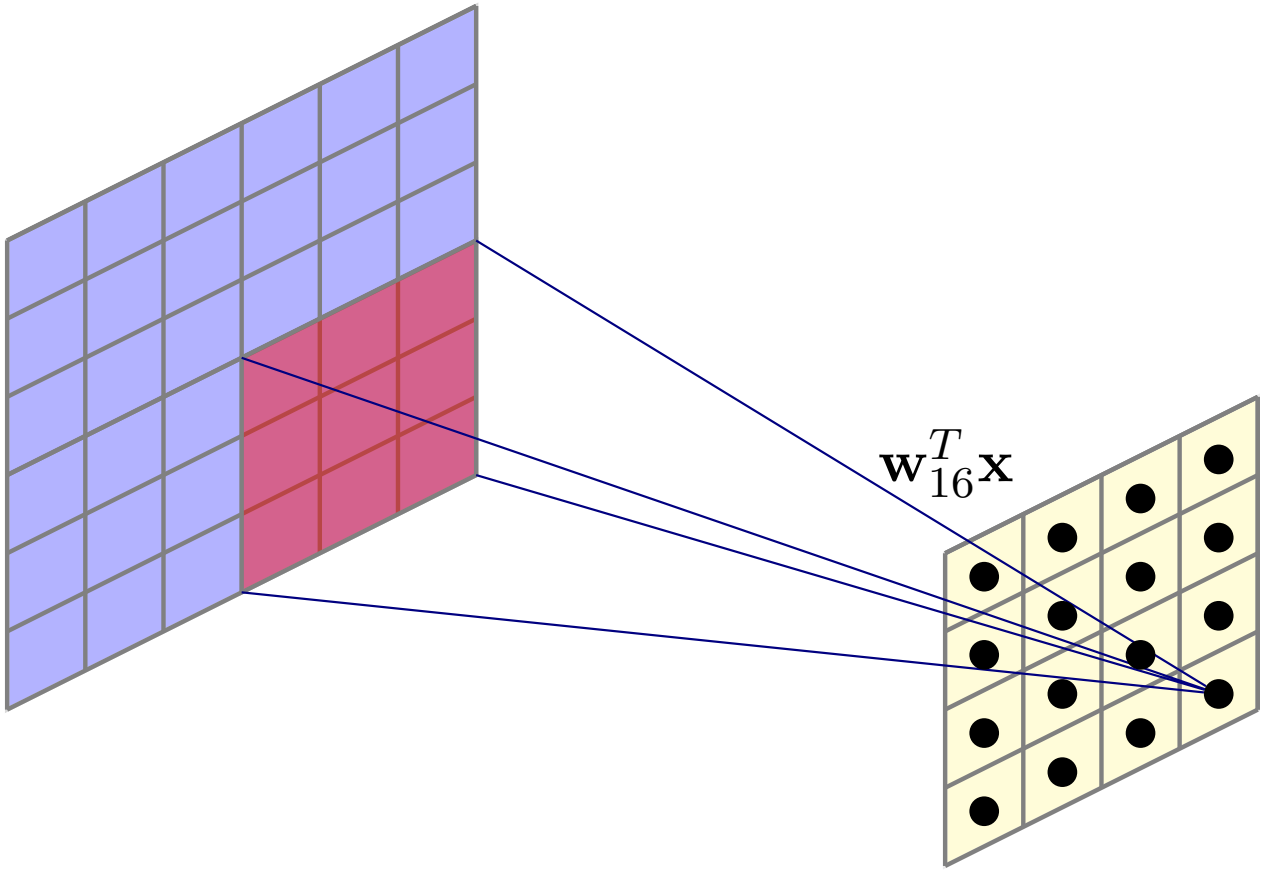
Locally Connected Layer



Locally Connected Layer



Locally Connected Layer



- What is the number of parameters?

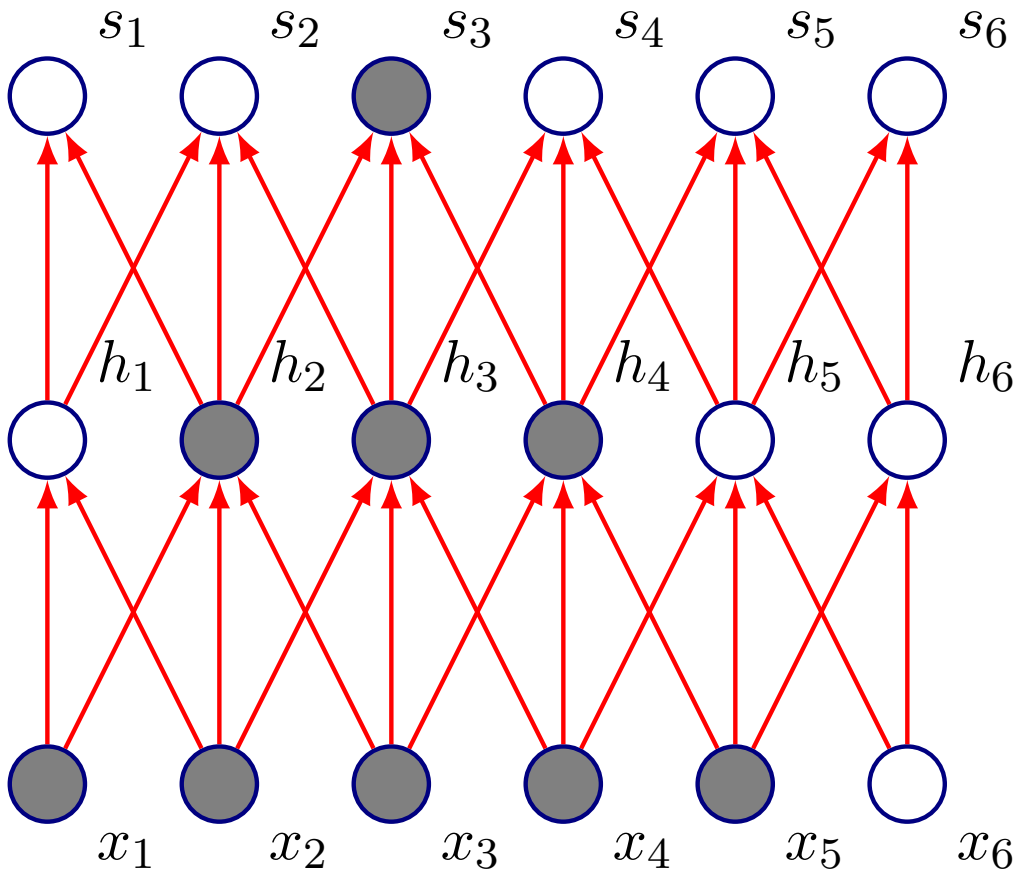
Tiled Convolution

- A compromise between locally connected layers and convolution
- Idea: Have a set of kernels and rotate them while traversal
- Ensures that immediate neighbors have different kernels
- Some parameters sharing (for 5 kernels in the previous example, what is the number of parameters?)

Dilated Convolutions

à trous: Convolutions with Holes

A Problem with regular convolutions

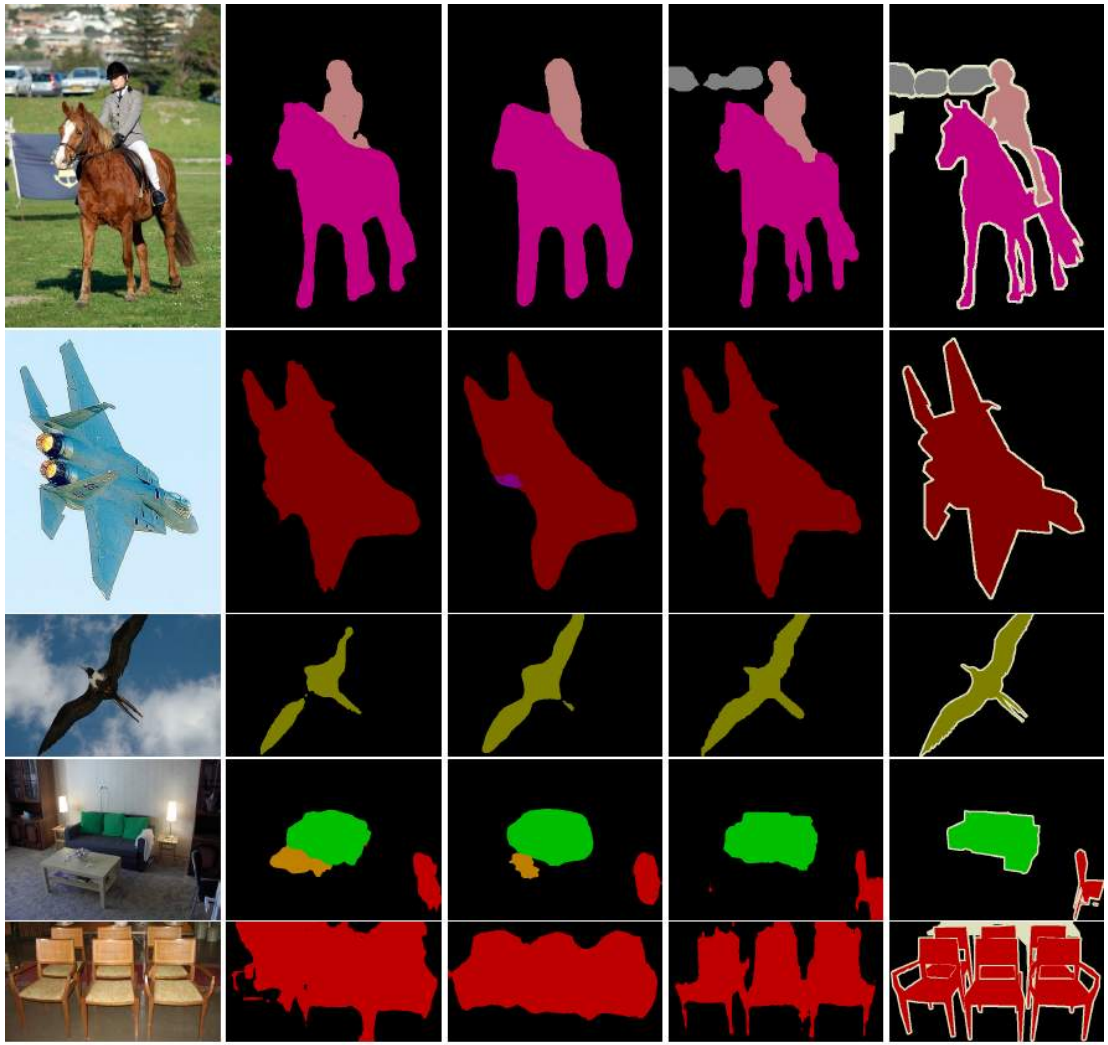


- Connections in CNNs are sparse, but units in deeper layers are connected to more of the input. At what rate does the effective receptive field size increase with depth?

A Problem with regular convolutions

- Convolutional networks repeat CONV-POOL-CONV-POOL to aggregate multiscale information until a global prediction is obtained
- In some applications we require dense prediction: Need multiscale reasoning as well as full-resolution output
- The global context of convolutional neural networks grows too slow for such applications
- Some examples?

Semantic Segmentation



(a) Image (b) FCN-8s (c) DeepLab (d) Our front end (e) Ground truth

Image: *Multiscale Context Aggregation by Dilated Convolutions*, Yu and Koltun, ICLR 2016

WaveNet: Causal Convolutions

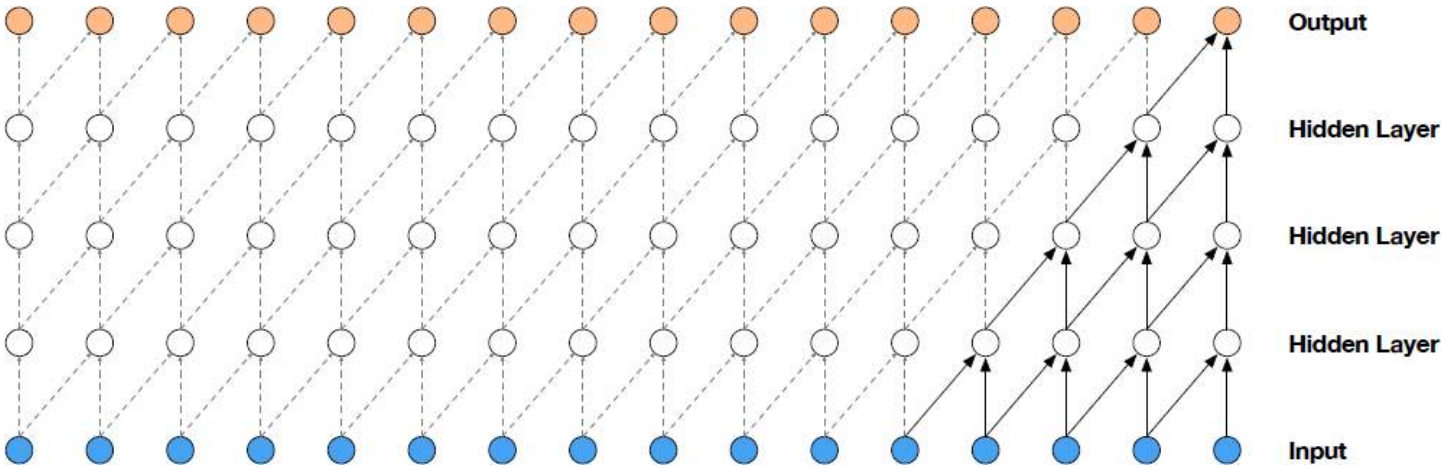


Image: *WaveNet: A Generative Model for Raw Audio*, Oord *et al.*, ICLR 2016

A Solution: Dilated Convolutions

- Recall discrete convolution:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$$

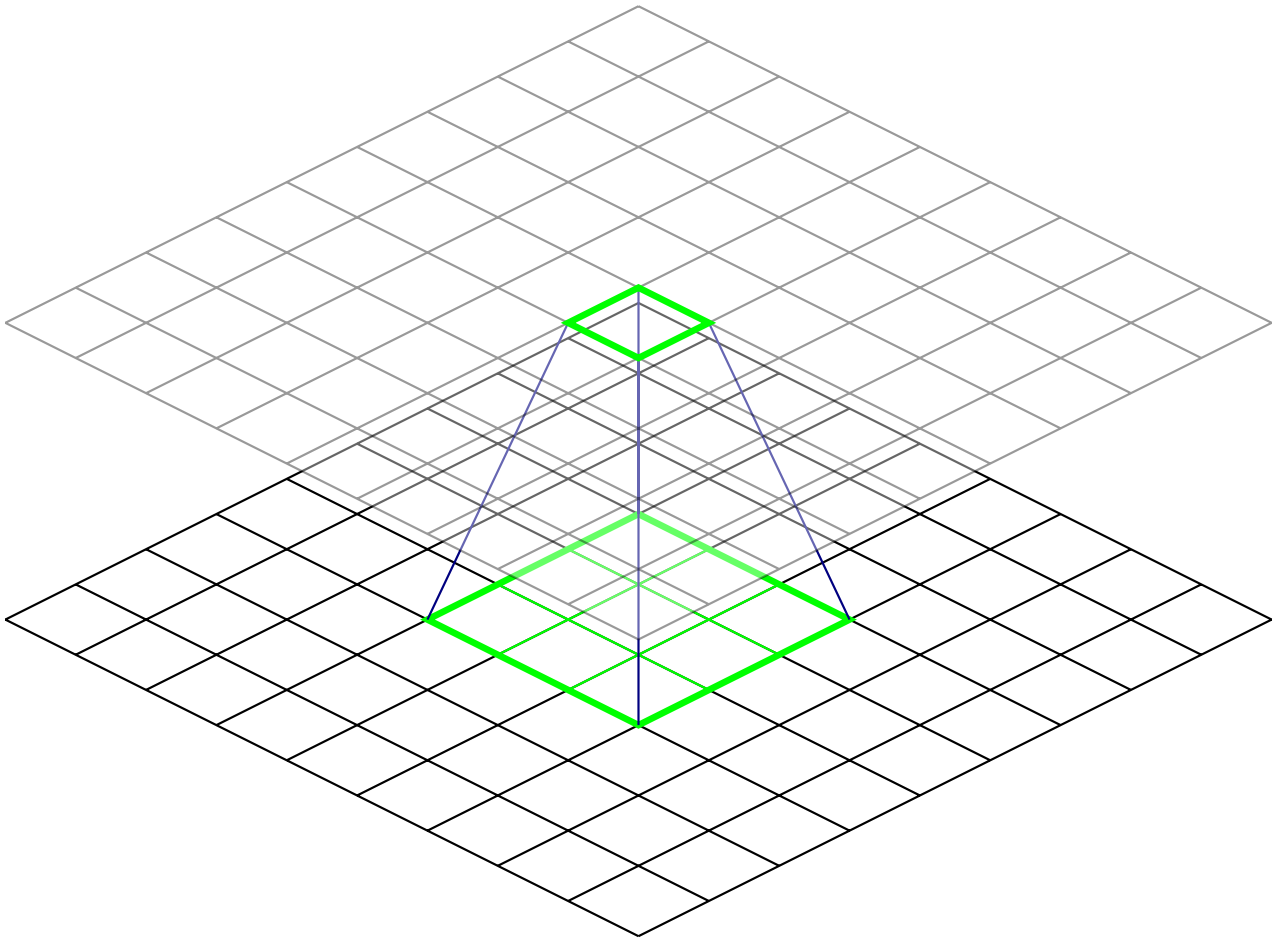
- Dilated Convolution:

$$S(i,j) = (I *_l K)(i,j) = \sum_m \sum_n I(m,n)K(i-lm, j-ln)$$

  l is a dilation factor

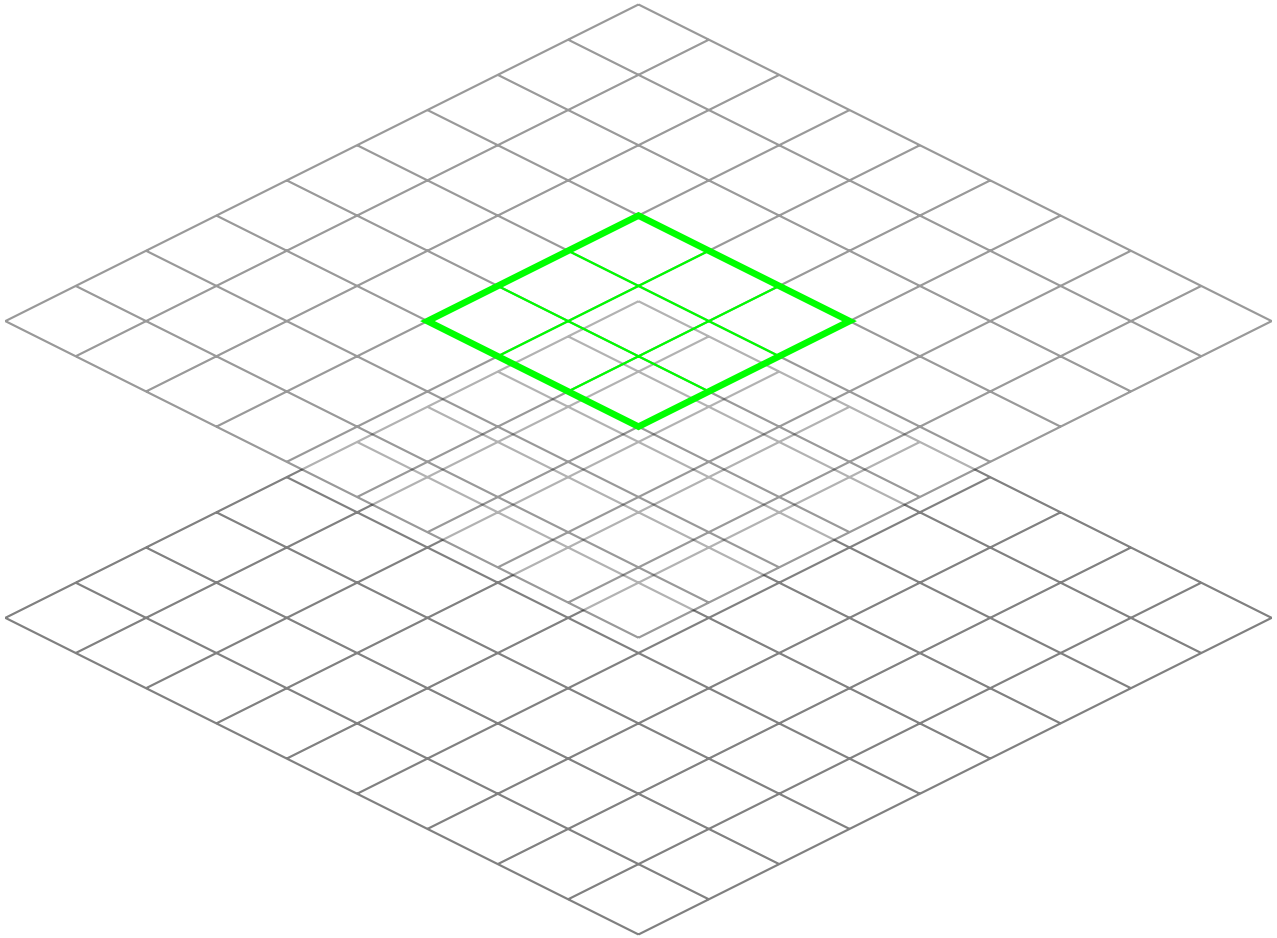
- Very old idea going to the 80s wavelet theory literature

Regular Convolution

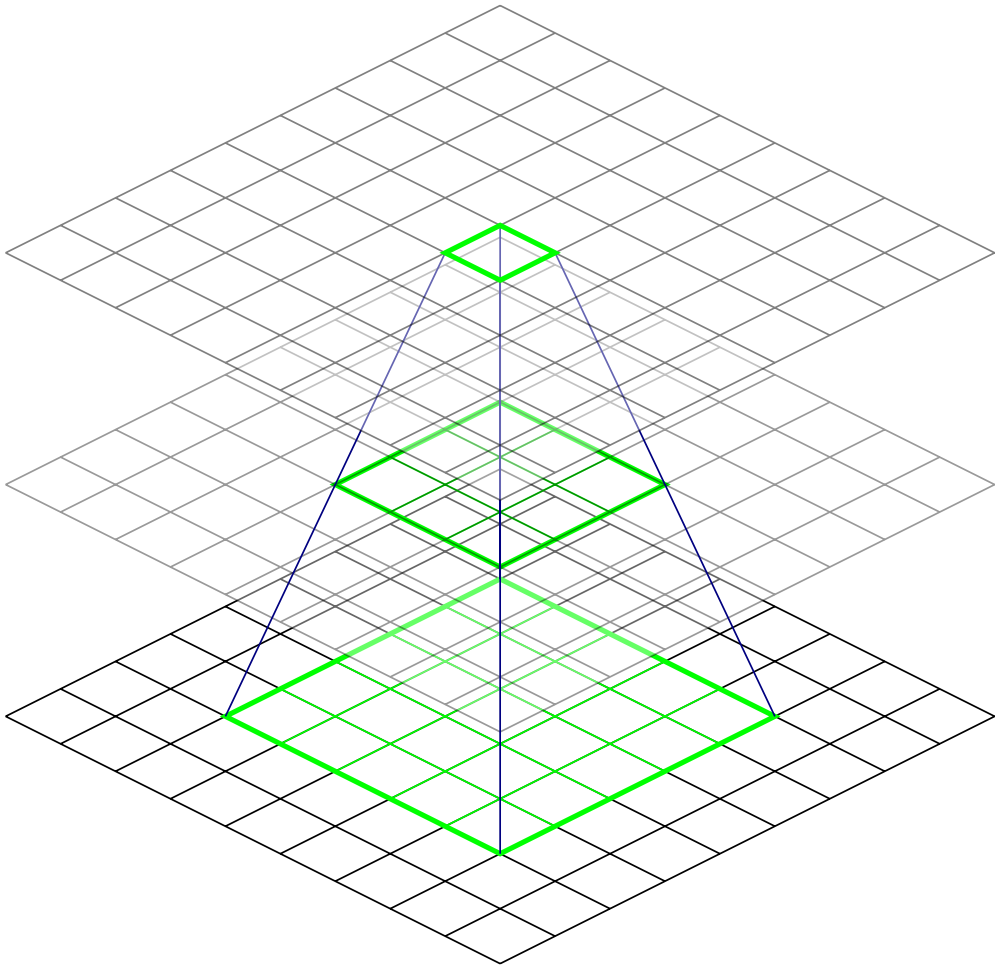


- The unit on the second layer has a receptive field of size 3×3

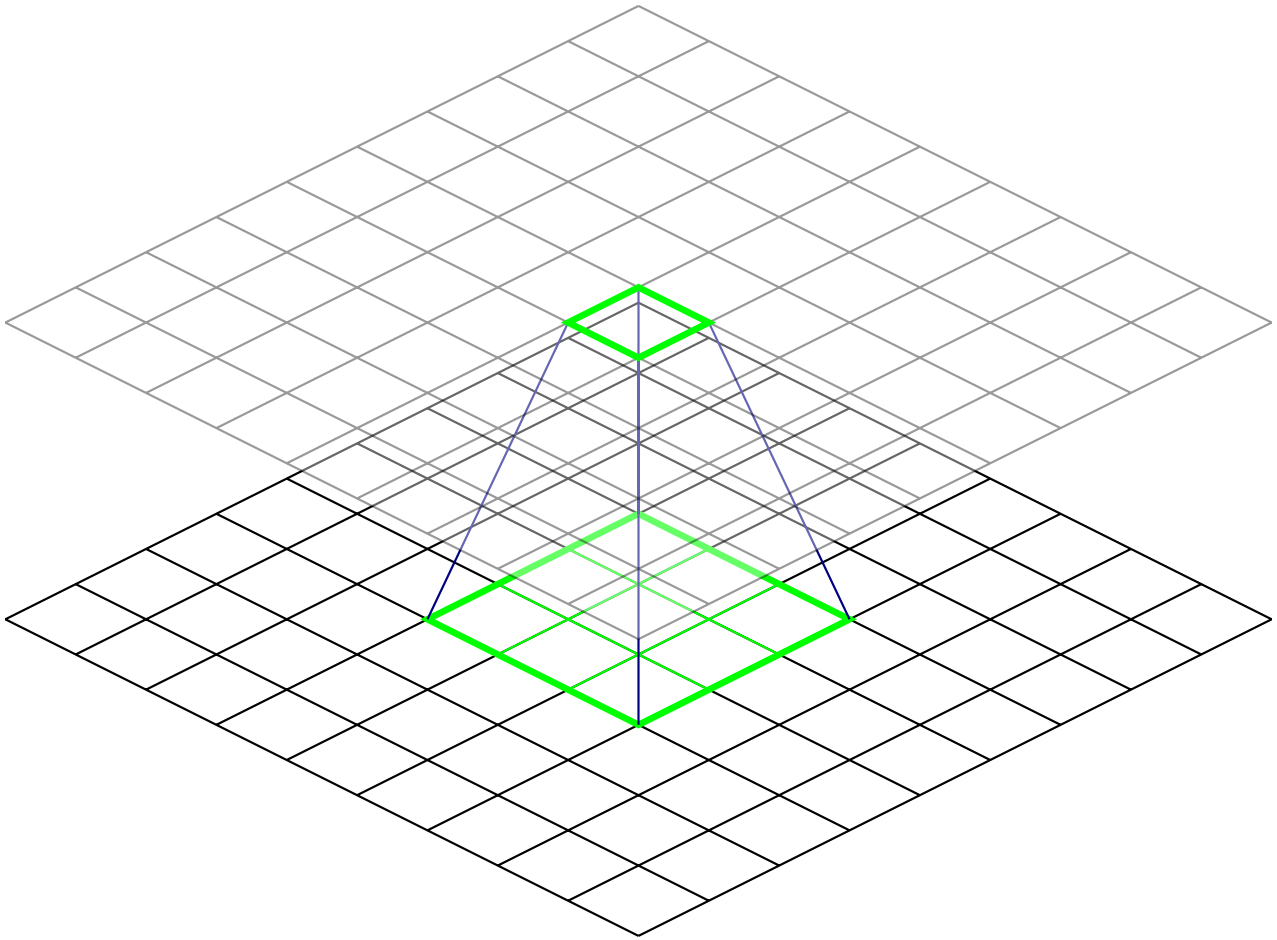
Regular Convolution



Regular Convolution

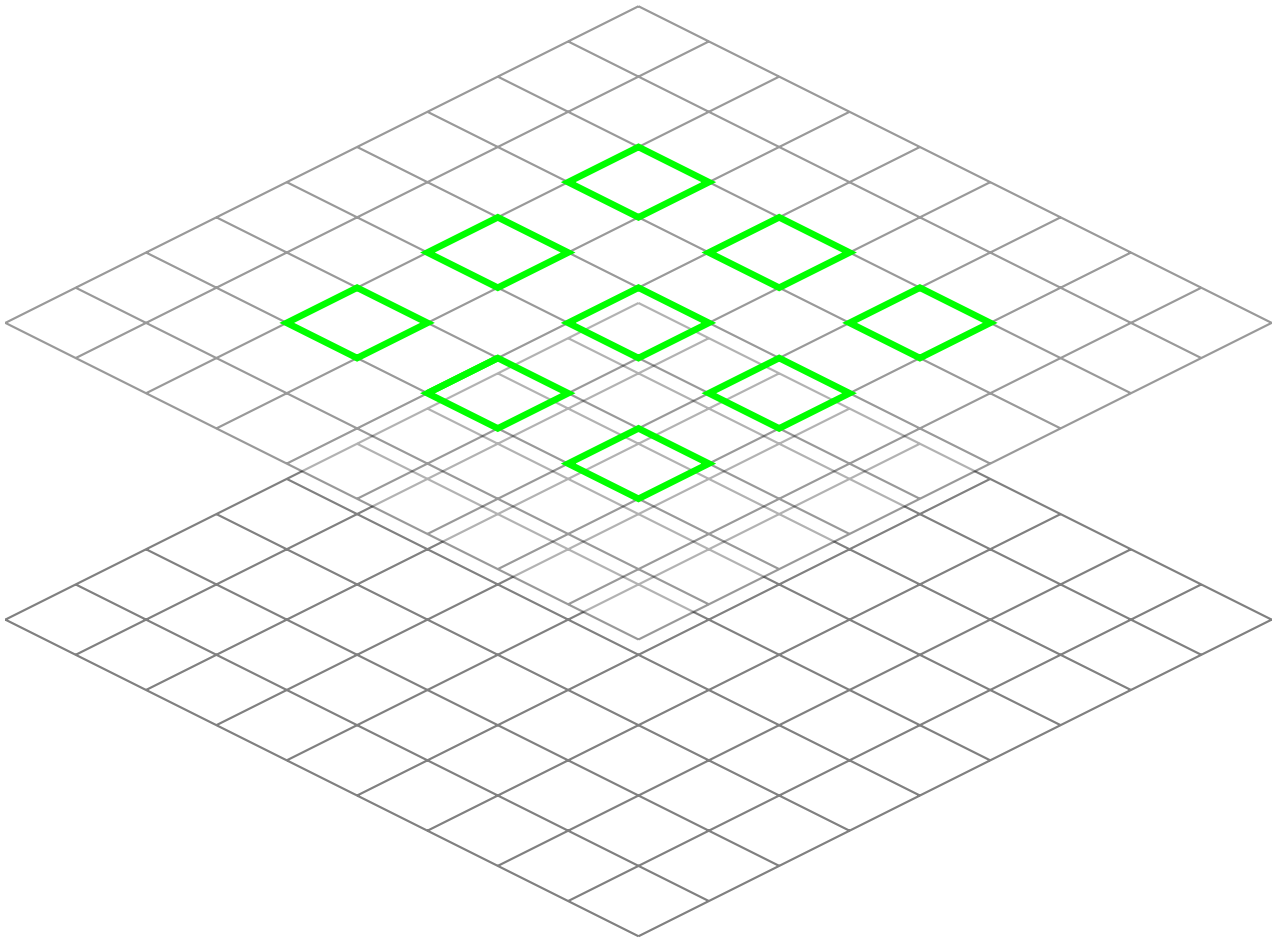


- The unit on the third layer has an effective receptive field of size 5×5

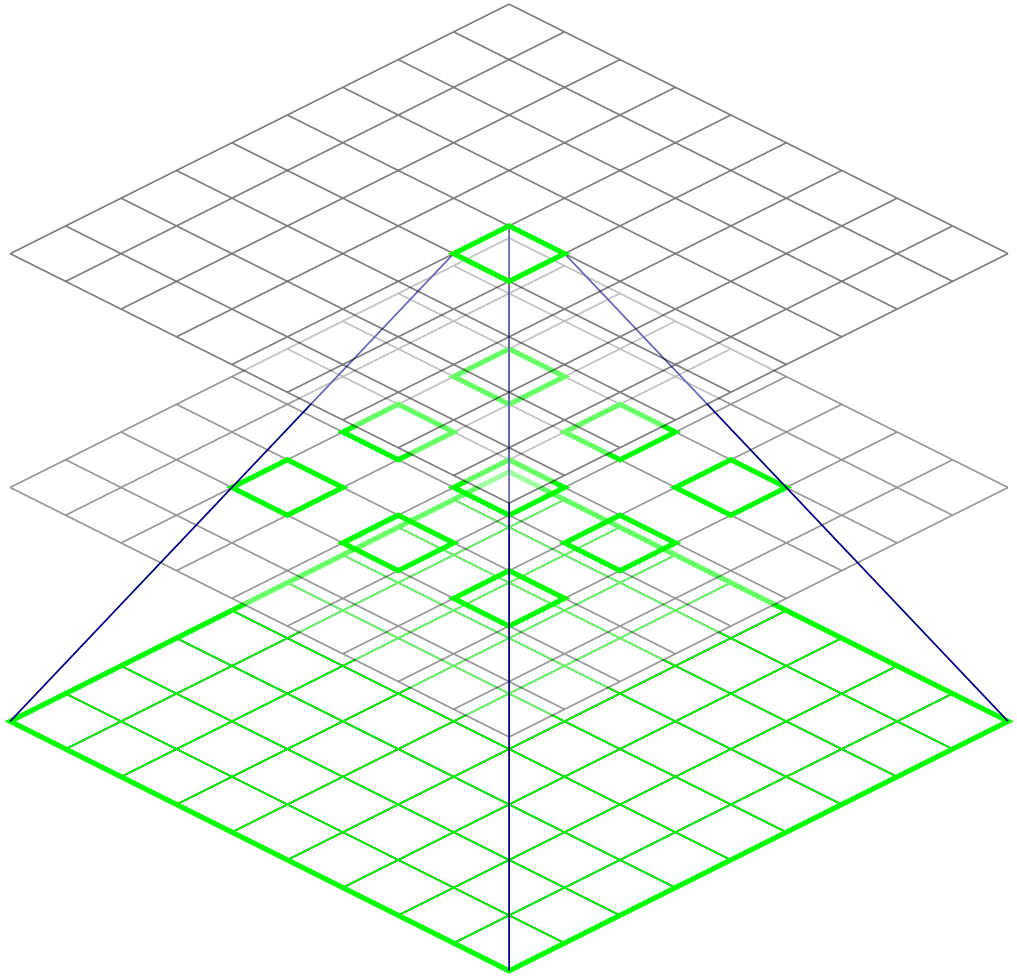


- The unit on the second layer has a receptive field of size 3×3

Dilated Convolution: Dilation of 1



Dilated Convolution



- The unit on the second layer has a receptive field of size 9×9

WaveNet: Dilated Causal Convolutions

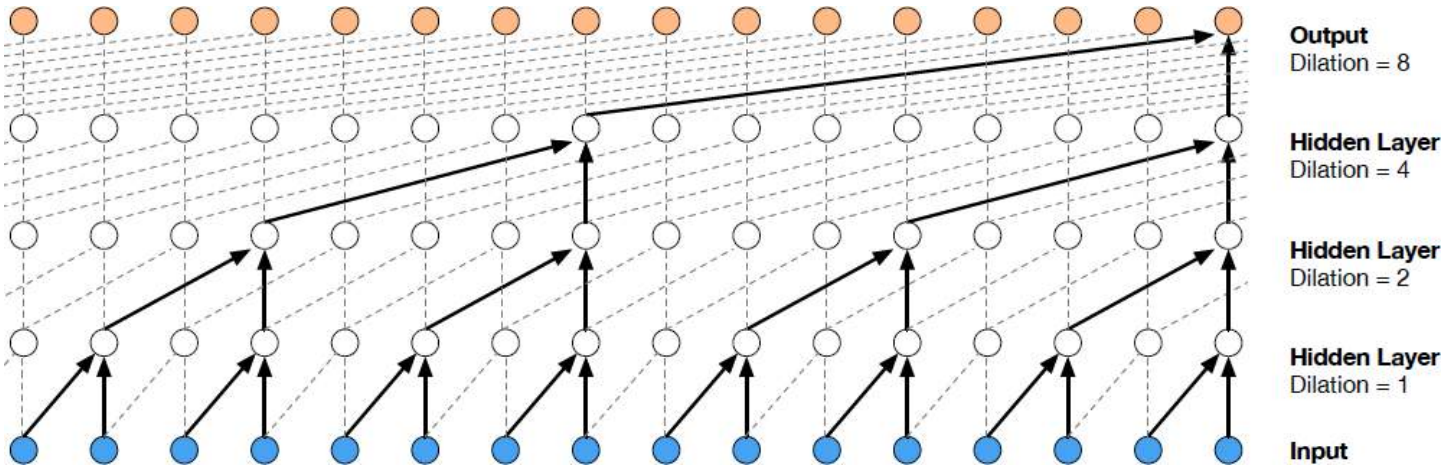


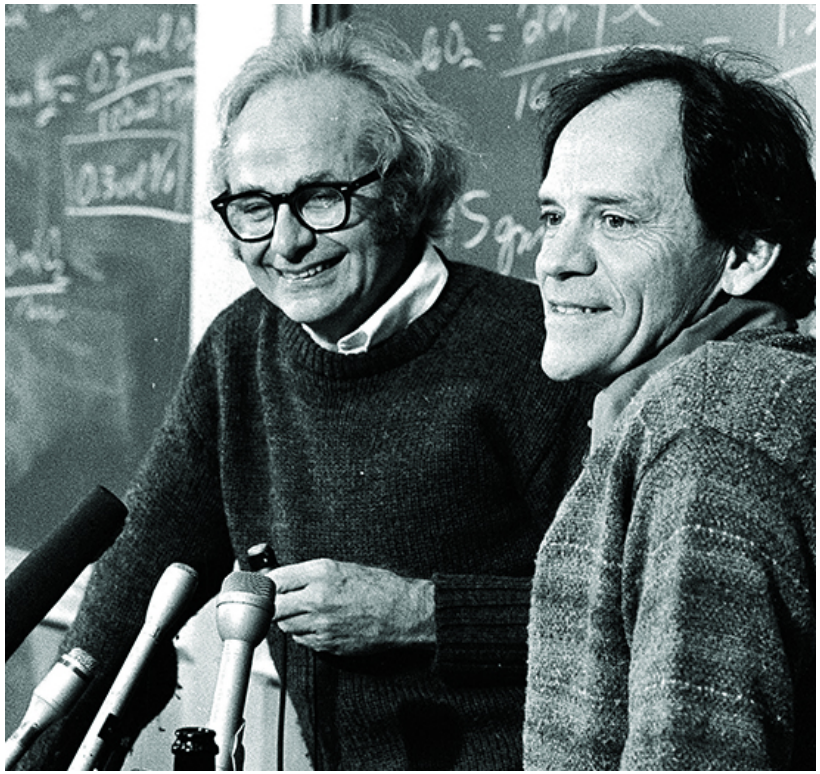
Image: *WaveNet: A Generative Model for Raw Audio*, Oord *et al.*, ICLR 2016

- We will see this in detail a few classes later

The Neuroscientific Motivation for Convolutional Networks

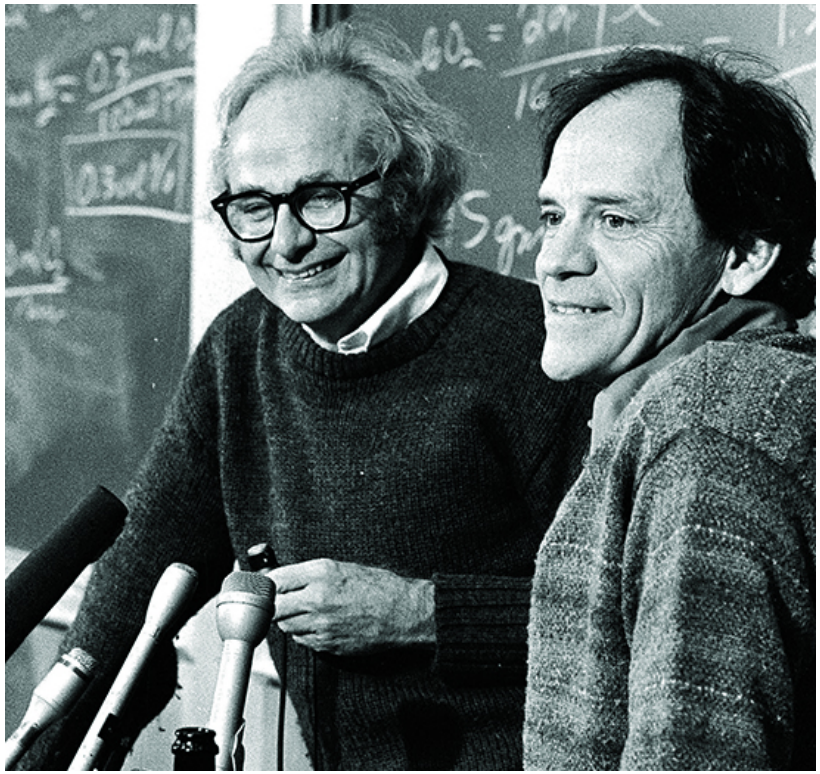
Idea Genealogy for CNNs

Hubel-Wiesel Experiments, 1959



- David Hubel and Torsten Wiesel did a set of famous experiments to determine basic facts about mammalian vision
- Example: Recorded activity of individual neurons and observed responses to images projected in precise locations on a screen in front of the cat

Hubel-Wiesel Experiments, 1959



- Neurons in the cat's early visual system responded very strongly to specific patterns of light, such as oriented bars and almost not at all to other patterns
- Neurons in the later visual system responded to more complex stimuli and responses also exhibited invariance to translations etc

A Simplified View of Brain Function

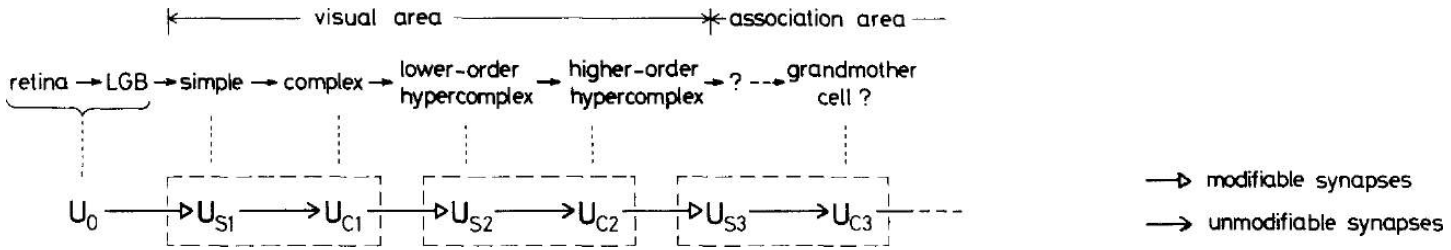


Image Source: Fukushima, 1980

- Images are projected onto the retina, neurons in retina do some simple preprocessing but do not substantially alter the representation
- The signal channels into the area LGN (through the optic nerve)
- Let's assume these regions simply carry the signal from eye to area V1

A Simplified View of Brain Function

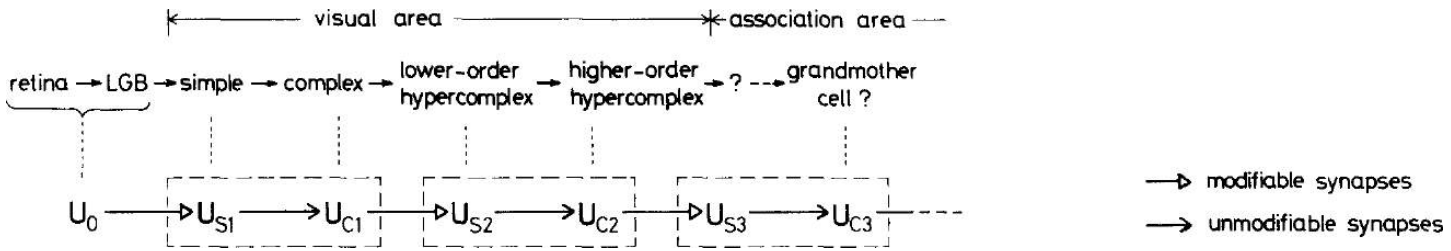


Image Source: Fukushima, 1980

- **V1 is arranged in a spatial map:**
  - 2D structure that mirrors structure of image in the retina
  - Light incident in the lower half of the retina only affects the lower half of V1

A Simplified View of Brain Function

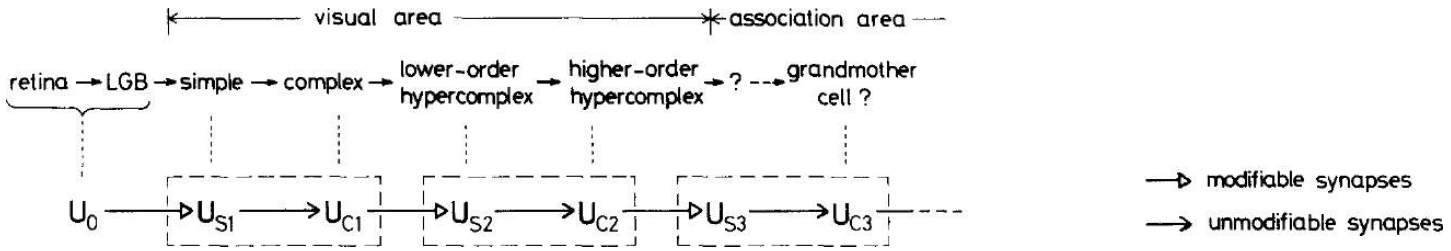


Image Source: Fukushima, 1980

- **V1 has many simple cells:** Roughly characterized by a linear function of image in small, spatially localized receptive fields (detection)
- **V1 has many complex cells:** Features detected similar to simple cells, but invariant to small shifts in position of feature (pooling)
- Also invariant to some changes in lighting

A Simplified View of Brain Function

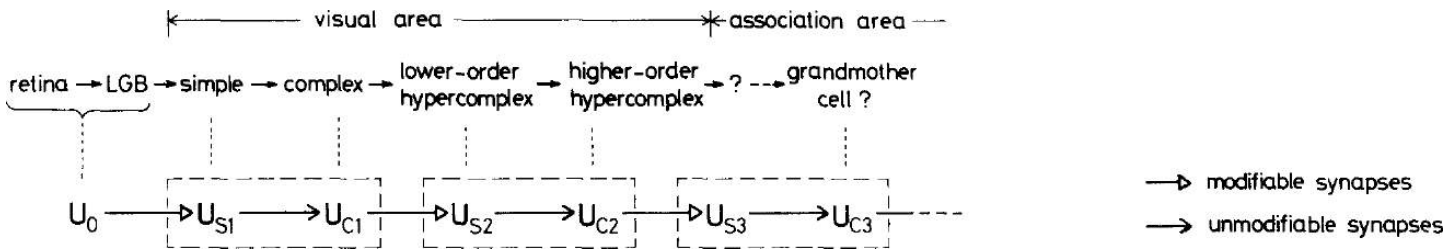


Image Source: Fukushima, 1980

- In the simplified view, this basic strategy is repeated many times
- After multiple layers, we find cells that respond to only specific concepts and are invariant to many transformations of the input (grandmother cells in the medial temporal lobe)

Simple and Complex Cells

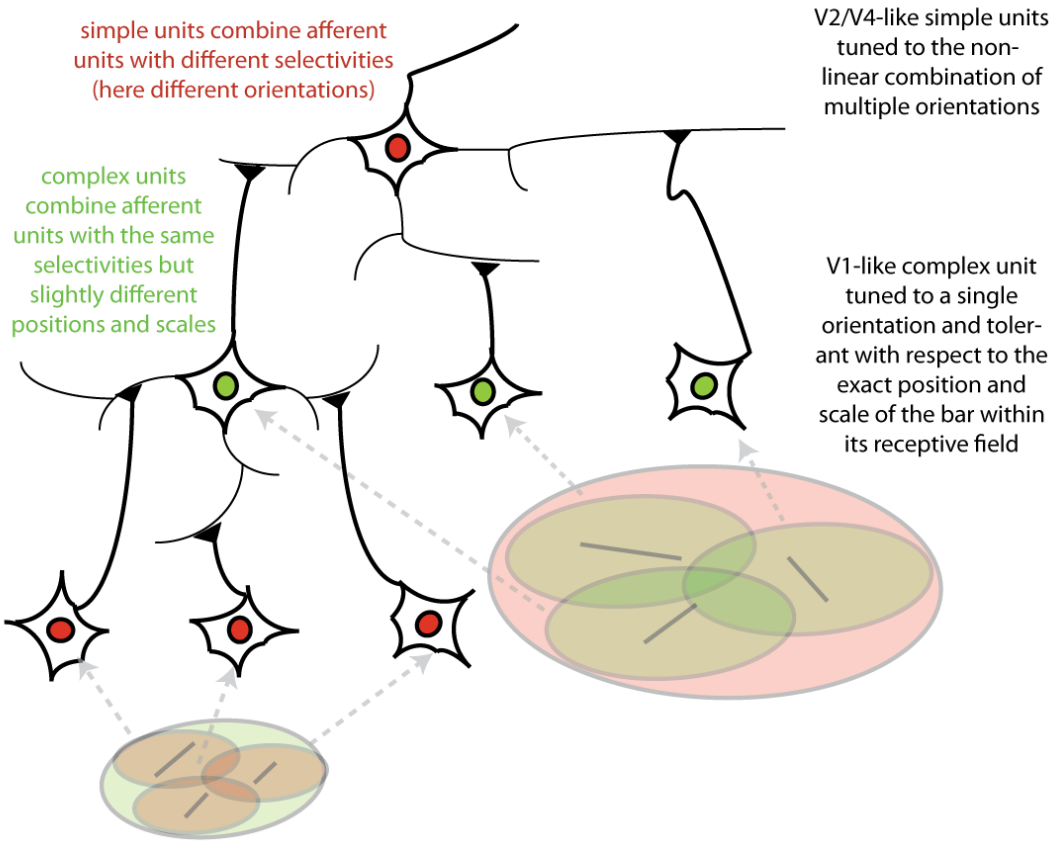
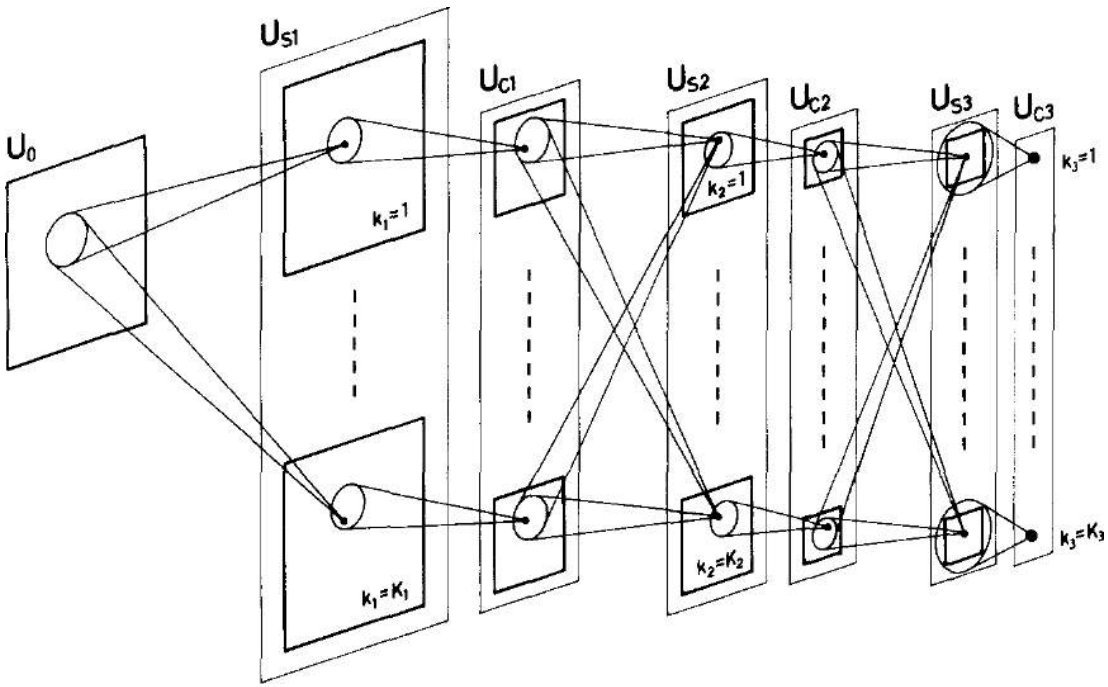


Image Source: Scholarpedia

Neocognitron (Fukushima, 1980)



- Fukushima used this simplified view of brain function to build a neural network which had detector units and pooling units one after the other
- Was trained by an unsupervised procedure

TDNNs and CNNs

- Waibel and Hinton introduced a 1-D Convolutional Network and trained it by backpropagation
- Convolutional Networks topology was directly inspired by the Neocognitron which was directly inspired by the Hubel-Weisel model
- TDNNs inspired the use of backpropagation for training for 2D CNNs (Yann LeCun, 1989)

Next time

- More on Equivariance
- Group Equivariant CNNs
- Spatial Transformers and related ideas
- Back to Architectures: Ultra Deep Models
- Begin: CNNs on Graphs and Combinatorial Data