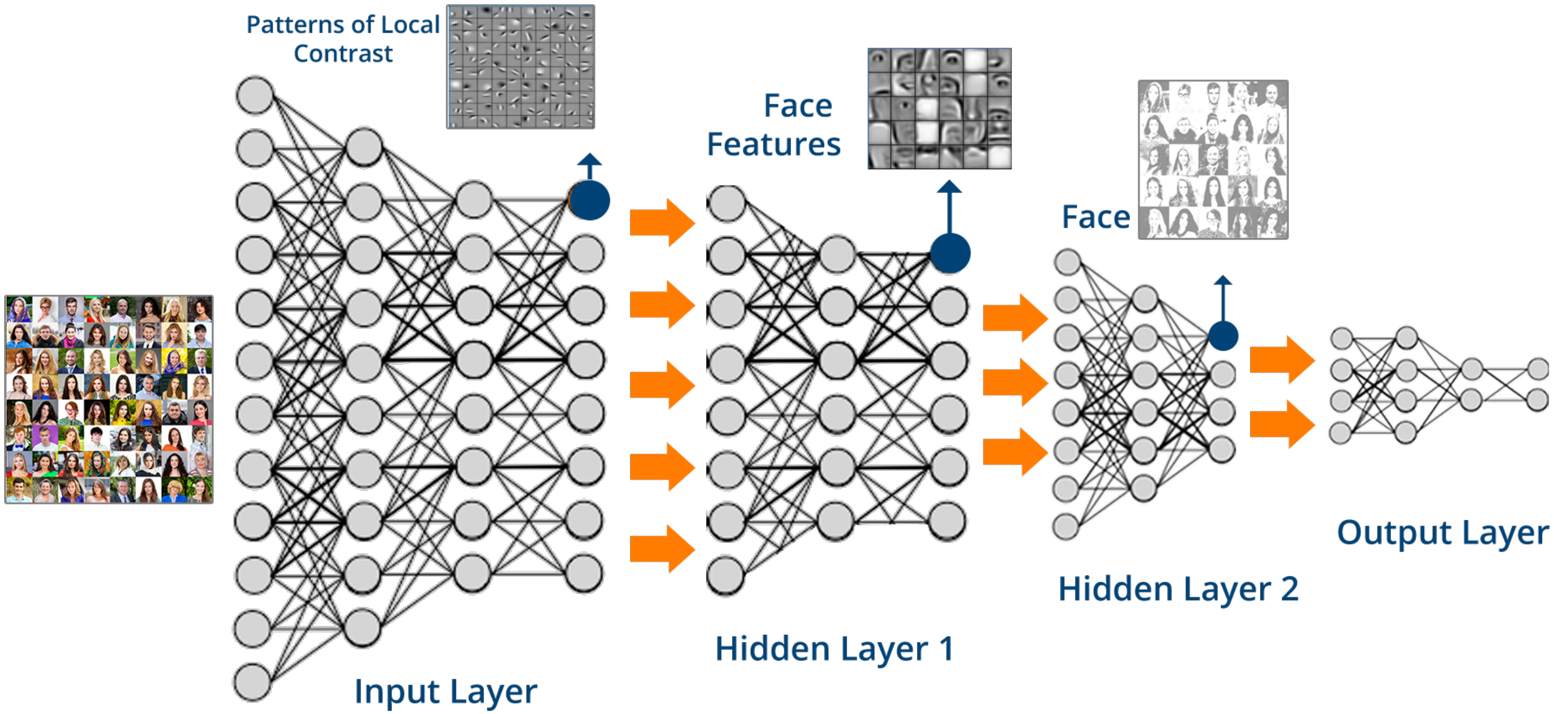
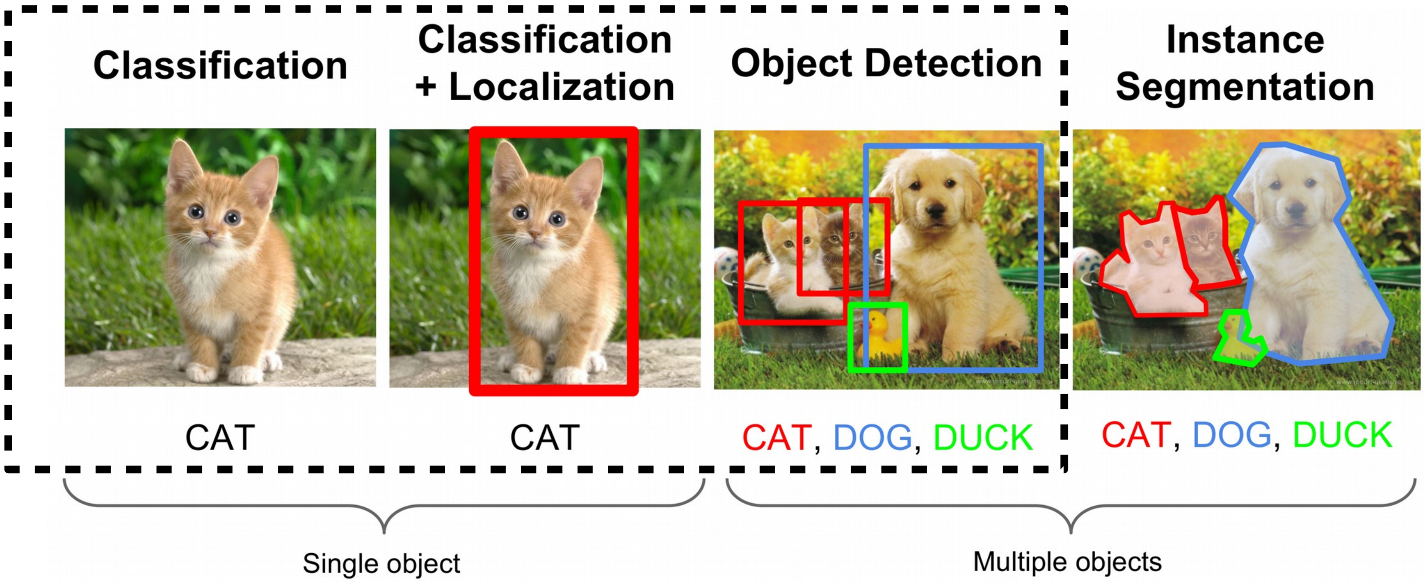
Neural Networks



What is this deep learning thing, anyway?

Problems we want to solve

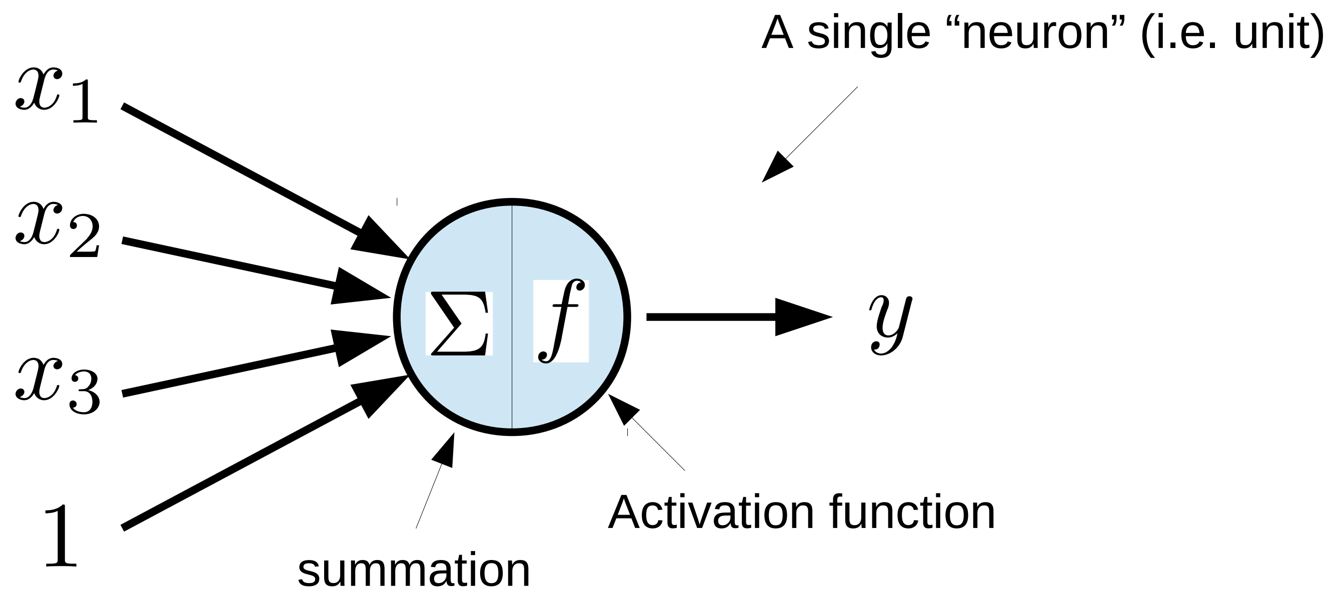
We will focus on these applications



<u>We will ignore these applications</u>
– image segmentation
– speech-to-text
– natural language processing
– …

.. but deep learning has been applied in lots of ways...

The multi-layer perceptron

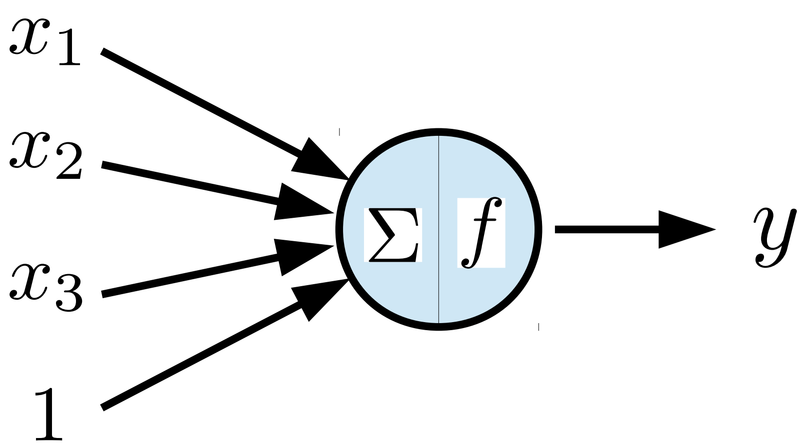


$$y = f(w_1 x_1 + w_2 x_2 + w_3 x_3 + \cdots + b)$$

$$= f(w^T x + b)$$

$$x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \qquad w = \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix}$$

The multi-layer perceptron

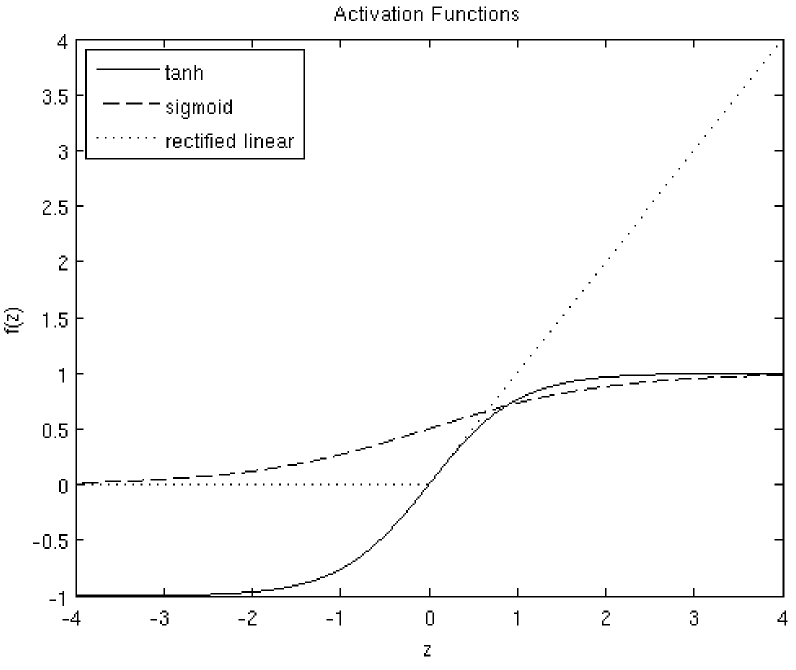


Different activation functions:

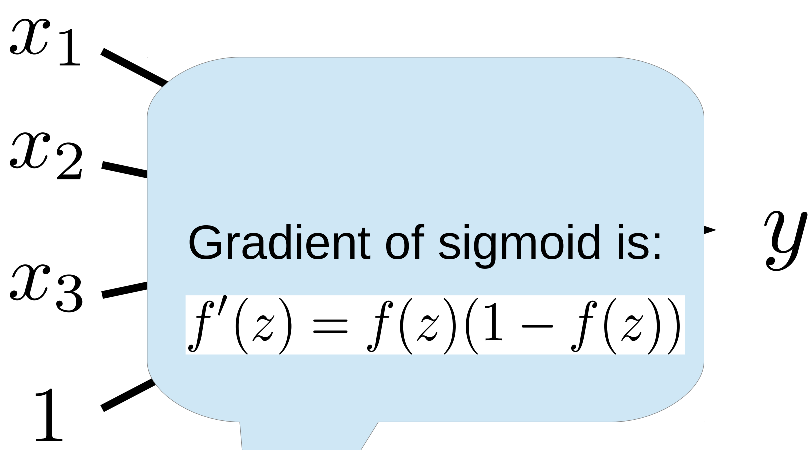
– sigmoid $f(z) = \dfrac{1}{1+e^{-z}}$

– tanh $f(z) = tanh(z) = \dfrac{e^z - e^{-z}}{e^z + e^{-z}}$

– rectified linear unit (ReLU) $f(z) = \max(0, z)$



The multi-layer perceptron



Gradient of sigmoid is:

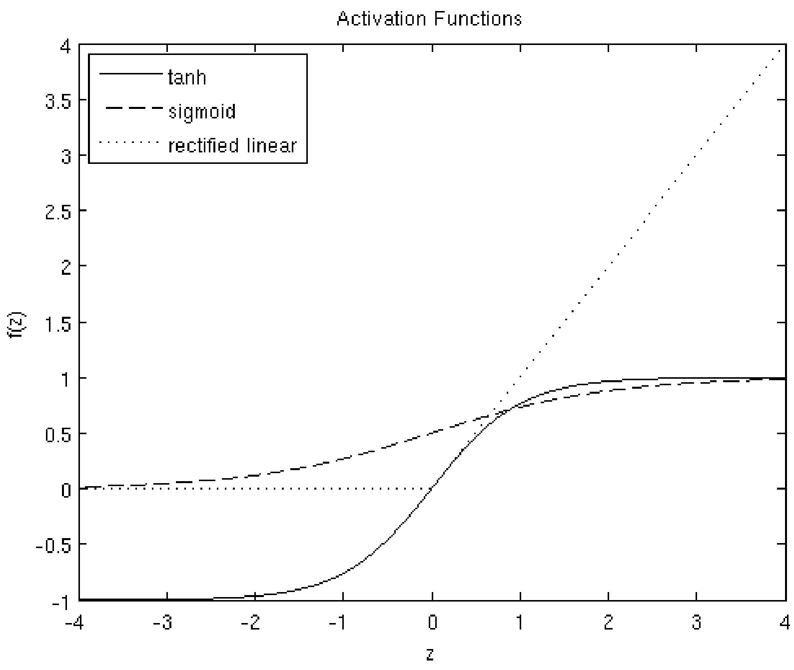
$$f'(z) = f(z)(1 - f(z))$$

Different activation functions:

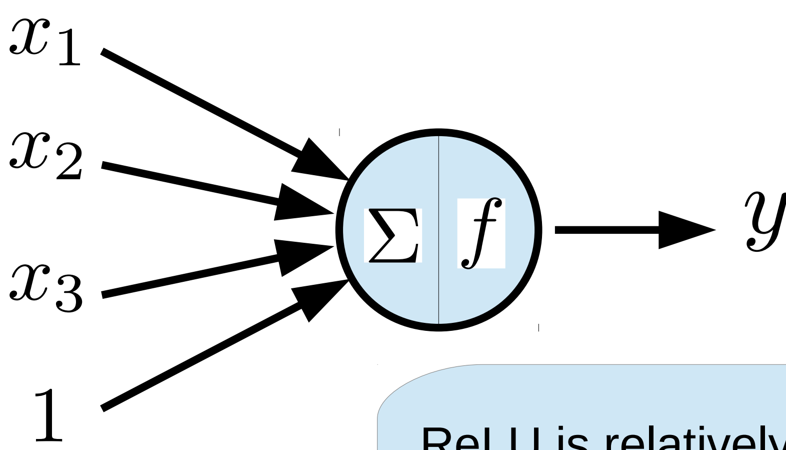
– sigmoid $$f(z) = \frac{1}{1+e^{-z}}$$

– tanh $$f(z) = tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

– rectified linear unit (ReLU) $$f(z) = \max(0, z)$$



The multi-layer perceptron



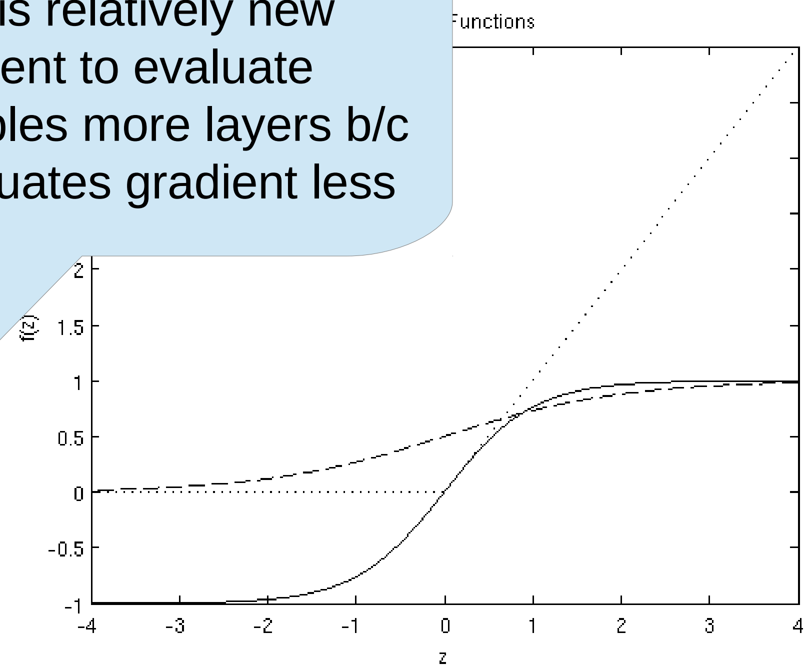
ReLU is relatively new
– efficient to evaluate
– enables more layers b/c attenuates gradient less

Different activation functions:

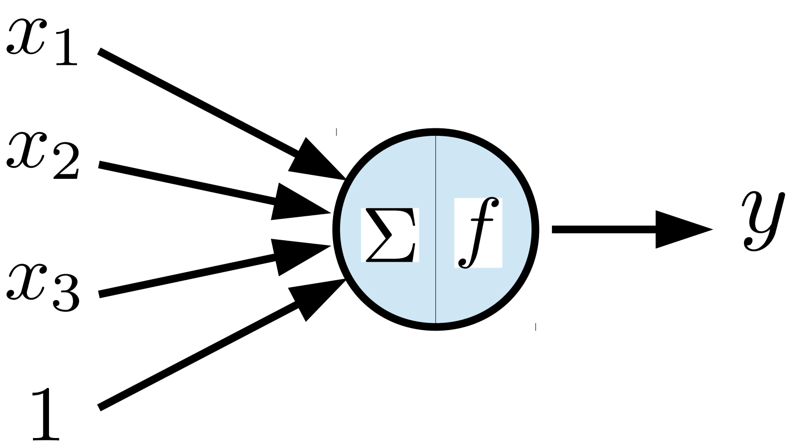
– sigmoid $f(z) = \dfrac{1}{1+e^{-z}}$

– tanh $f(z) = tanh(z) = \dfrac{e^z - e^{-z}}{e^z + e^{-z}}$

– rectified linear unit (ReLU) $f(z) = \max(0, z)$



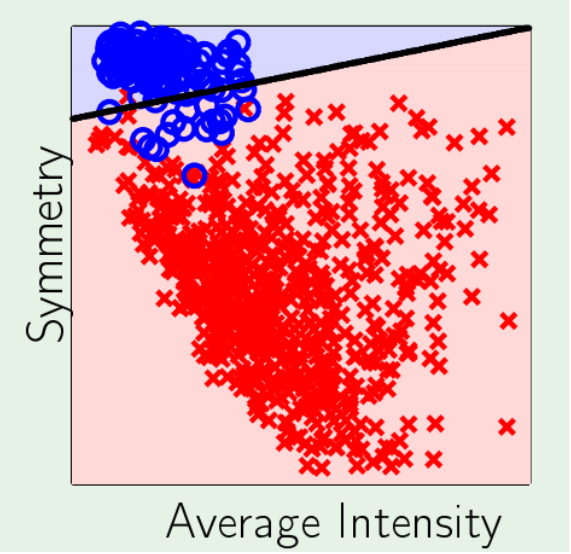
The multi-layer perceptron



One layer neural network has a simple interpretation: linear classification.

$$y = f(w^T x + b)$$

X_1 == symmetry
X_2 == avg intensity
Y == class label (binary)



What do *w* and *b* correspond to in this picture?

Training

Given a dataset: $D = \{(x^1, y^1), (x^2, y^2), \ldots, (x^n, y^n)\}$

Define loss function: $L(x^i, y^i; w, b) = (y^i - f(w^T x^i + b))^2$

Training

Given a dataset: $D = \{(x^1, y^1), (x^2, y^2), \ldots, (x^n, y^n)\}$

Define loss function: $L(x^i, y^i; w, b) = \frac{1}{2}(y^i - f(w^T x^i + b))^2$

Loss function tells us how
well the network classified x^i

Training

Given a dataset: $D = \{(x^1, y^1), (x^2, y^2), \dots, (x^n, y^n)\}$

Define loss function: $L(x^i, y^i; w, b) = \frac{1}{2}(y^i - f(w^T x^i + b))^2$

Loss function tells us how well the network classified x^i

Method of training: adjust *w, b* so as to minimize the net loss over the dataset

i.e.: adjust *w, b* so as to minimize: $\sum_{(x^i, y^i) \in D} L(x^i, y^i; w, b)$

If the sum of losses is zero, then the network has classified the dataset perfectly

Training

Method of training: adjust *w, b* so as to minimize the net loss over the dataset

i.e.: adjust *w, b* so as to minimize: $$\sum_{(x^i, y^i) \in D} L(x^i, y^i; w, b)$$

Training

Method of training: adjust *w, b* so as to minimize the net loss over the dataset

i.e.: adjust *w, b* so as to minimize: $$\sum_{(x^i, y^i) \in D} L(x^i, y^i; w, b)$$

How?

Training

Method of training: adjust *w, b* so as to minimize the net loss over the dataset

i.e.: adjust *w, b* so as to minimize: $\sum_{(x^i,y^i)\in D} L(x^i, y^i; w, b)$

<u>Do gradient descent on dataset:</u>

1. repeat

2. $$w \leftarrow w - \alpha \frac{1}{n} \sum_{(x^i,y^i)\in D} \nabla_w L(x^i, y^i; w, b)$$

3. $$b \leftarrow b - \alpha \frac{1}{n} \sum_{(x^i,y^i)\in D} \nabla_b L(x^i, y^i; w, b)$$

4. until converged

Where:

$$\nabla_w L(x^i, y^i; w, b) = -(y^i - f(w^T x^i + b)) f'(w^T x^i + b) x^i$$

$$\nabla_b L(x^i, y^i; w, b) = -(y^i - f(w^T x^i + b)) f'(w^T x^i + b)$$

Training

Method of training: adjust *w, b* so

i.e.: adjust *w, b* so as to minimize:

This is the similar to logistic regression
– logistic regression uses a cross entropy loss
– we are using a quadratic loss

<u>Do gradient descent on dataset:</u>

1. repeat

2. $$w \leftarrow w - \alpha \frac{1}{n} \sum_{(x^i, y^i) \in D} \nabla_w L(x^i, y^i; w, b)$$

3. $$b \leftarrow b - \alpha \frac{1}{n} \sum_{(x^i, y^i) \in D} \nabla_b L(x^i, y^i; w, b)$$

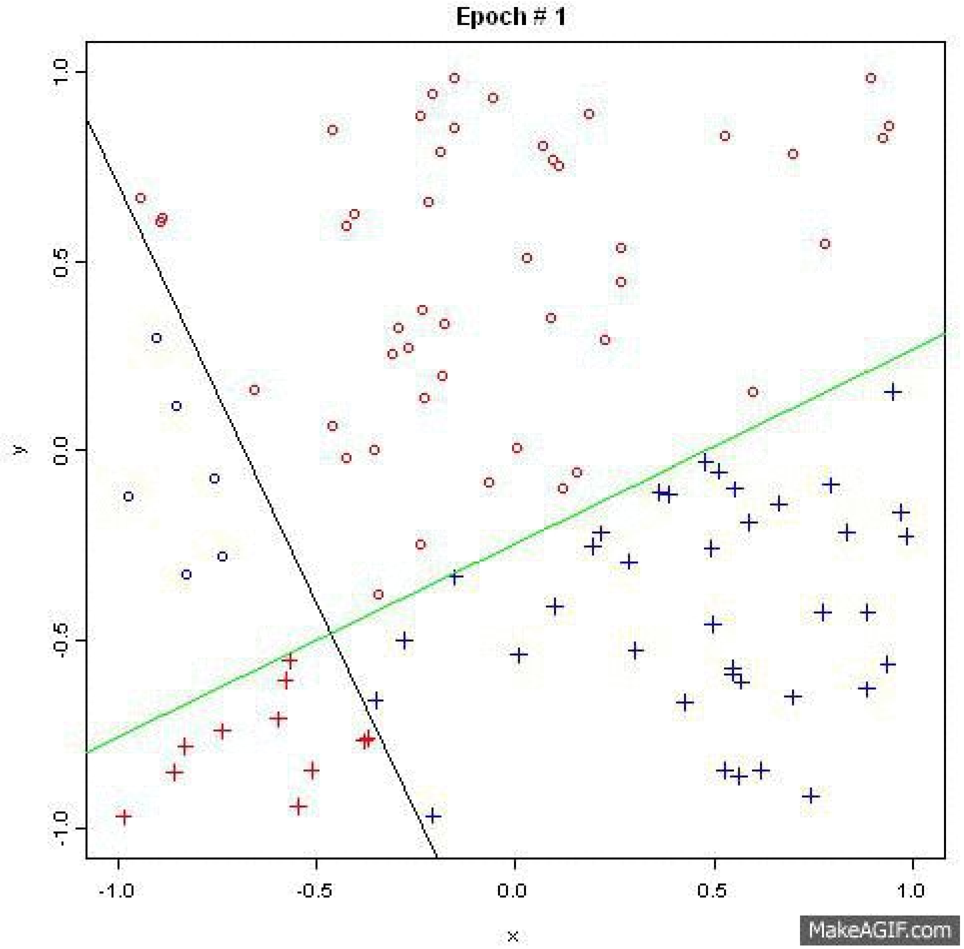
4. until converged

Where:

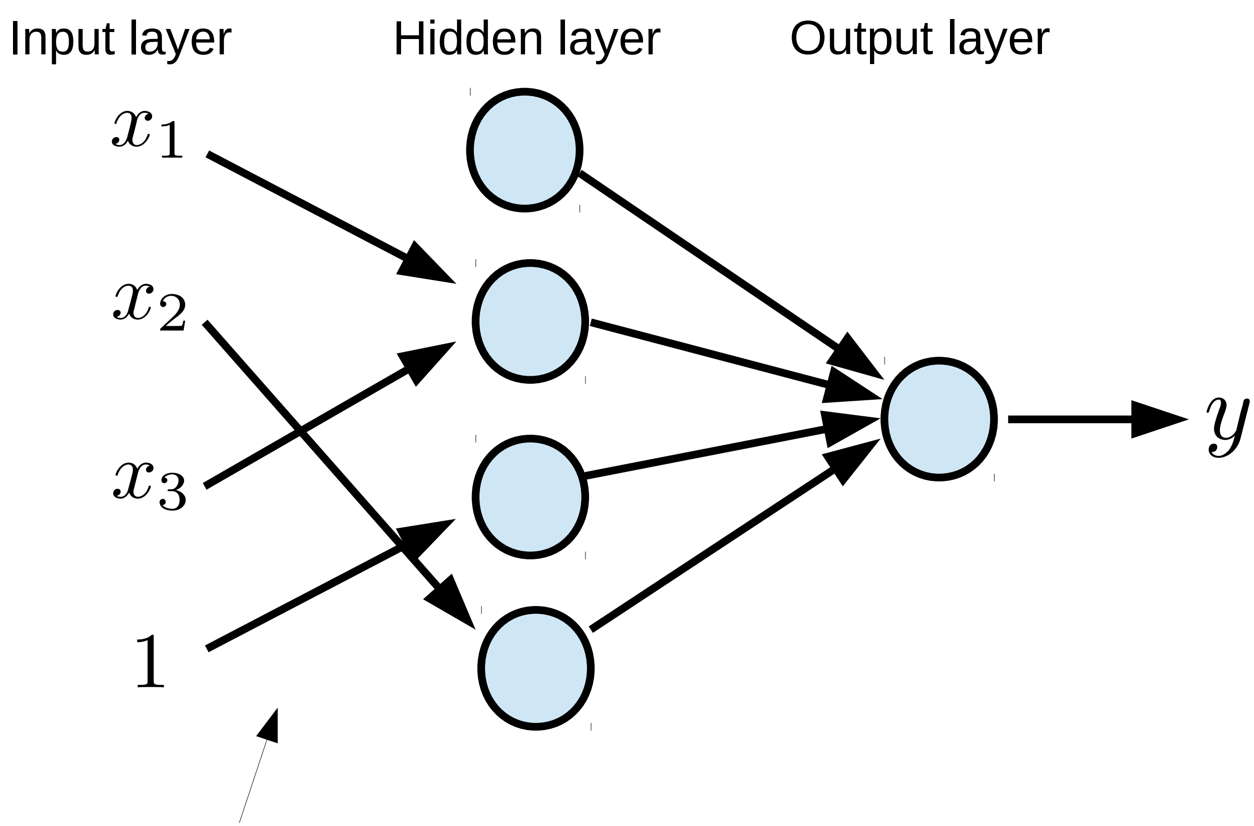
$$\nabla_w L(x^i, y^i; w, b) = -(y^i - f(w^T x^i + b)) f'(w^T x^i + b) x^i$$

$$\nabla_b L(x^i, y^i; w, b) = -(y^i - f(w^T x^i + b)) f'(w^T x^i + b)$$

Training example

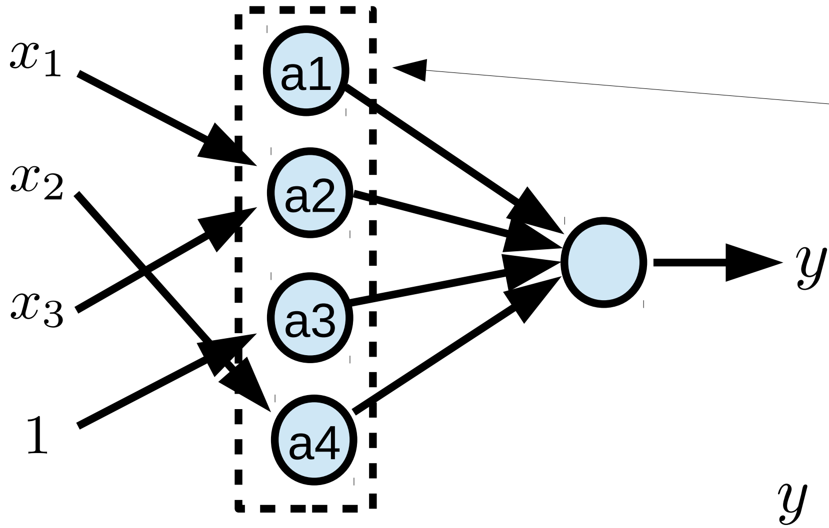


Going deeper: a one layer network



Each hidden node is
connected to every input

Multi-layer evaluation works similarly



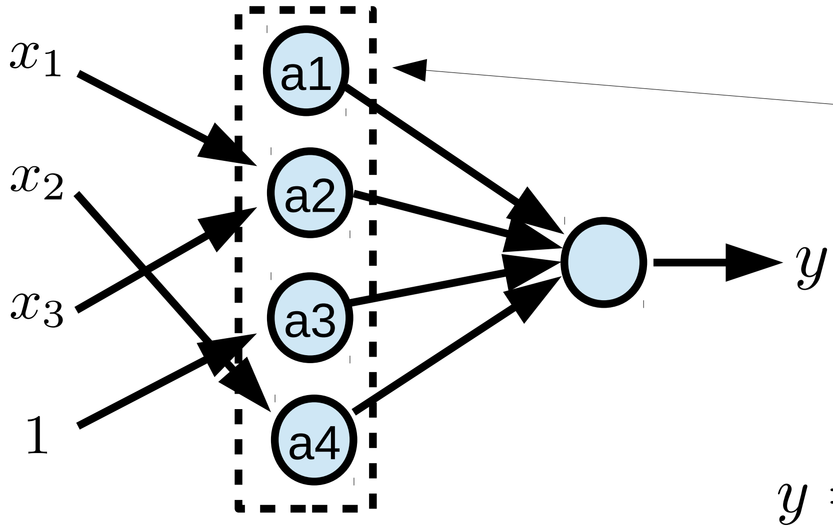
$$a = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix}$$

Vector of hidden layer activations

$$y = f({w^{(2)}}^T a + b)$$

Single activation: $a_i = f({w_i^{(1)}}^T x + b_i^{(1)})$

Multi-layer evaluation works similarly



$$a = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix}$$

Vector of hidden layer activations

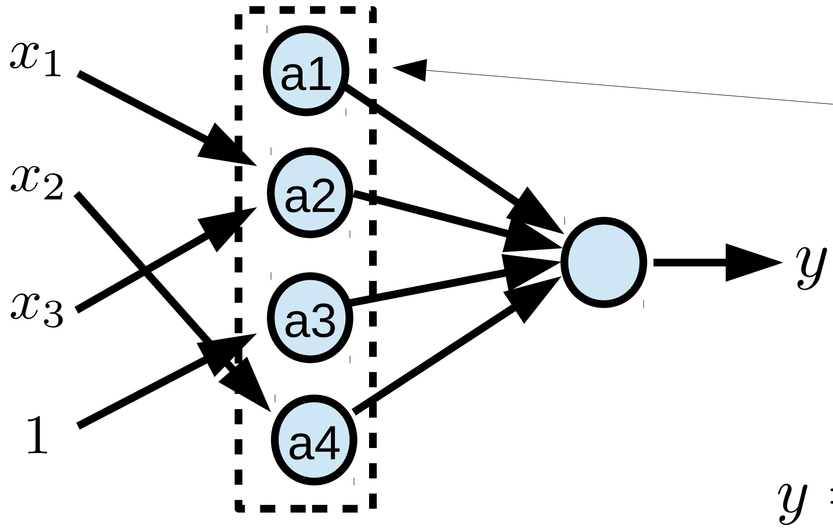
$$y = f({w^{(2)}}^T a + b)$$

Single activation: $a_i = f({w_i^{(1)}}^T x + b_i^{(1)})$

Vector of activations: $a = f(W^{(1)} x + b^{(1)})$

where $W^{(1)} = \begin{pmatrix} {w_1^{(1)}}^T \\ \vdots \\ {w_4^{(1)}}^T \end{pmatrix}$ $b^{(1)} = \begin{pmatrix} b_1 \\ \vdots \\ b_4 \end{pmatrix}$

Multi-layer evaluation works similarly



$$a = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix}$$

Vector of hidden layer activations

$$y = f({w^{(2)}}^T a + b)$$

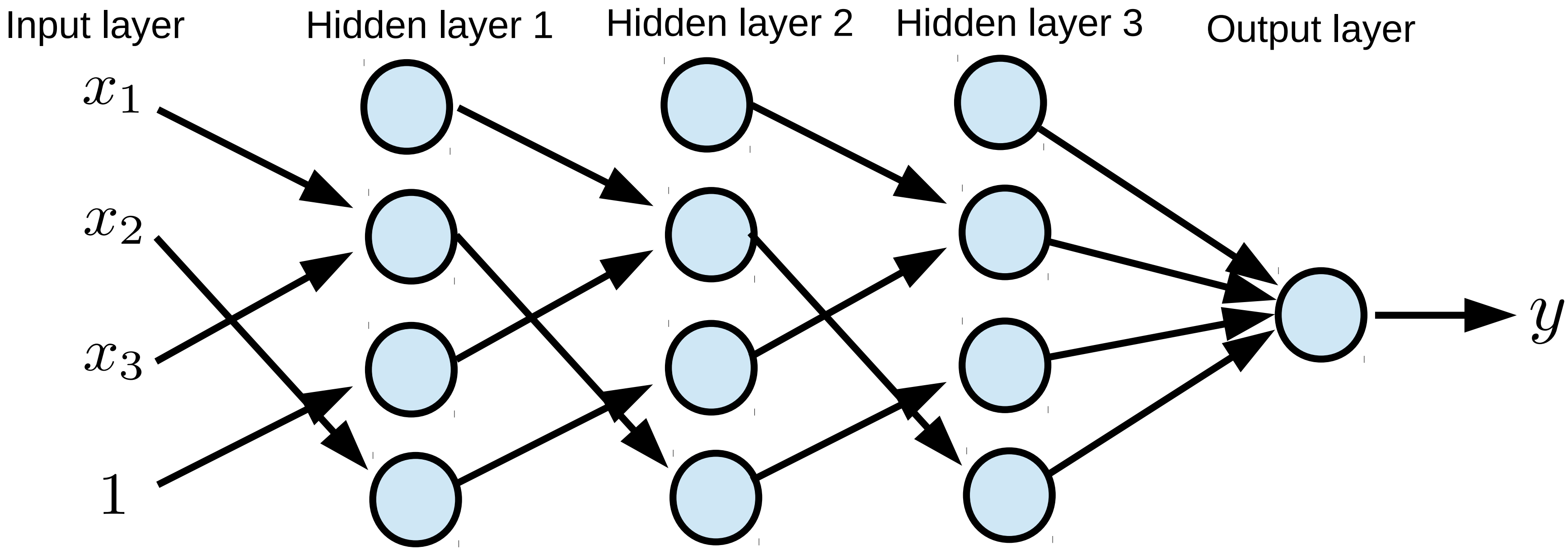
Called “forward propagation” – b/c the activations are propogated forward...

Single activation: $a_i = f({w_i^{(1)}}^T x + b_i^{(1)})$

Vector of activations: $a = f(W^{(1)} x + b^{(1)})$

where $W^{(1)} = \begin{pmatrix} {w_1^{(1)}}^T \\ \vdots \\ {w_4^{(1)}}^T \end{pmatrix}$ $b^{(1)} = \begin{pmatrix} b_1 \\ \vdots \\ b_4 \end{pmatrix}$

Can create networks of arbitrary depth...



– Forward propagation works the same for any depth network.

– Whereas a single output node corresponds to linear classification, adding hidden nodes makes classification non-linear

How do we train multi-layer networks?

Almost the same as in the single-node case...

Do gradient descent on dataset:

1. repeat

2. $$w \leftarrow w - \alpha \frac{1}{n} \sum_{(x^i, y^i) \in D} \nabla_w L(x^i, y^i; w, b)$$

3. $$b \leftarrow b - \alpha \frac{1}{n} \sum_{(x^i, y^i) \in D} \nabla_b L(x^i, y^i; w, b)$$

4. until converged

Now, we're doing gradient descent on all weights/biases in the network – not just a single layer

– this is called *backpropagation*

Backpropagation

1. Perform a feedforward pass, computing the activations for layers L_2, L_3, and so on up to the output layer L_{n_l}.

2. For each output unit i in layer n_l (the output layer), set

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2}\|y - h_{W,b}(x)\|^2 = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})$$

3. For $l = n_l - 1, n_l - 2, n_l - 3, \ldots, 2$

   For each node i in layer l, set

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)}\right) f'(z_i^{(l)})$$

4. Compute the desired partial derivatives, which are given as:

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W, b; x, y) = a_j^{(l)} \delta_i^{(l+1)}$$

$$\frac{\partial}{\partial b_i^{(l)}} J(W, b; x, y) = \delta_i^{(l+1)}.$$

http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/

Training in mini-batches

A batch is typically between 32 and 128 samples

1. repeat

2. randomly sample a mini-batch: $B \subset D$

3. $$w \leftarrow w - \alpha \frac{1}{n} \sum_{(x^i, y^i) \in B} \nabla_w L(x^i, y^i; w, b)$$

3. $$b \leftarrow b - \alpha \frac{1}{n} \sum_{(x^i, y^i) \in B} \nabla_b L(x^i, y^i; w, b)$$

4. until converged

Training in mini-batches helps b/c:
– don't have to load the entire dataset into memory
– training is still relatively stable
– random sampling of batches helps avoid local minima

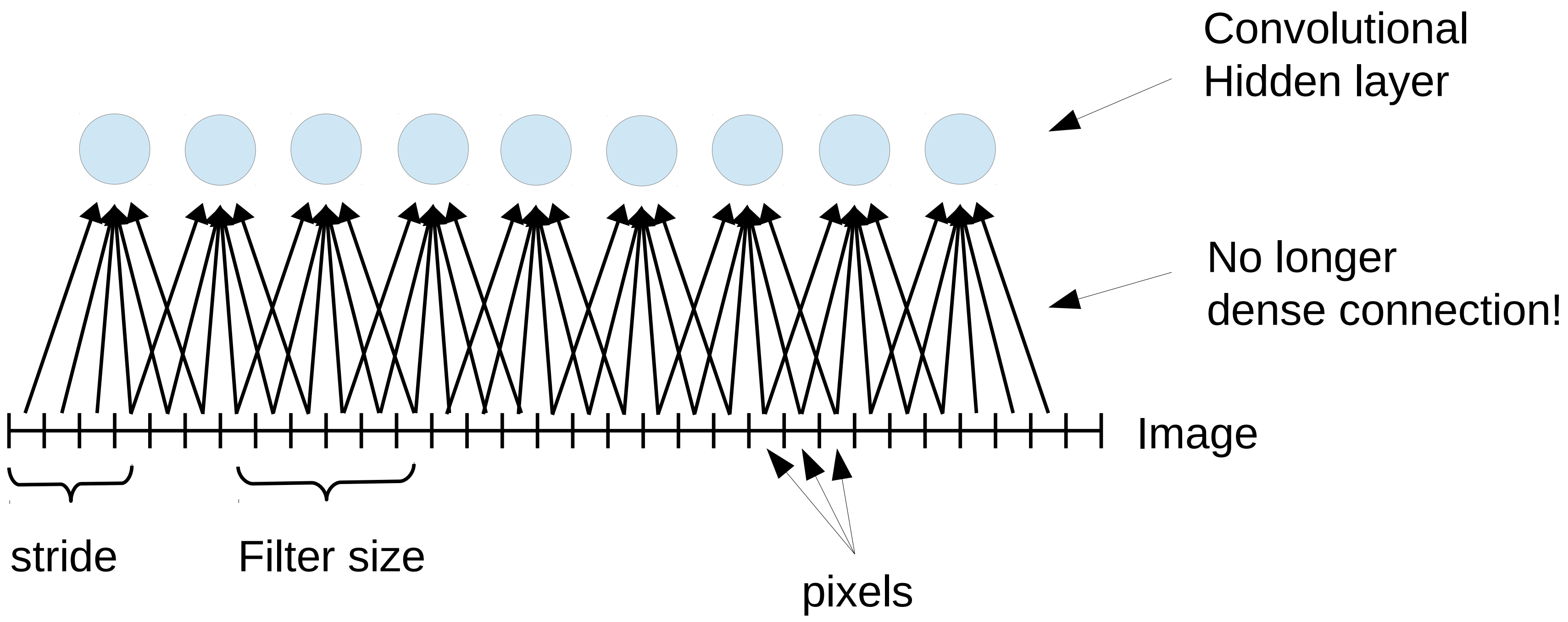
Convolutional layers

Deep multi-layer perceptron networks
– general purpose
– involve huge numbers of weights

We want:
– special purpose network for image and NLP data
– fewer parameters
– fewer local minima

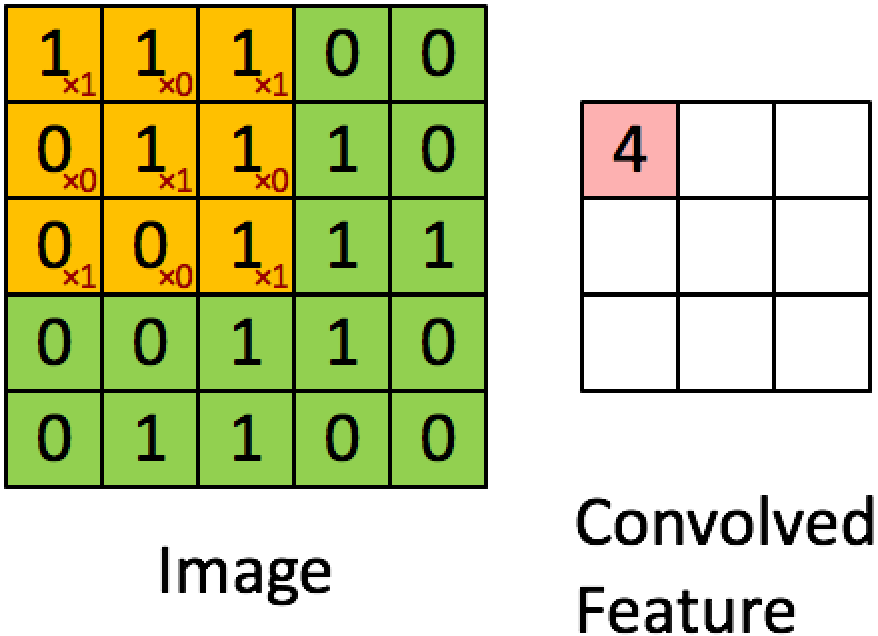
Answer: convolutional layers!

Convolutional layers



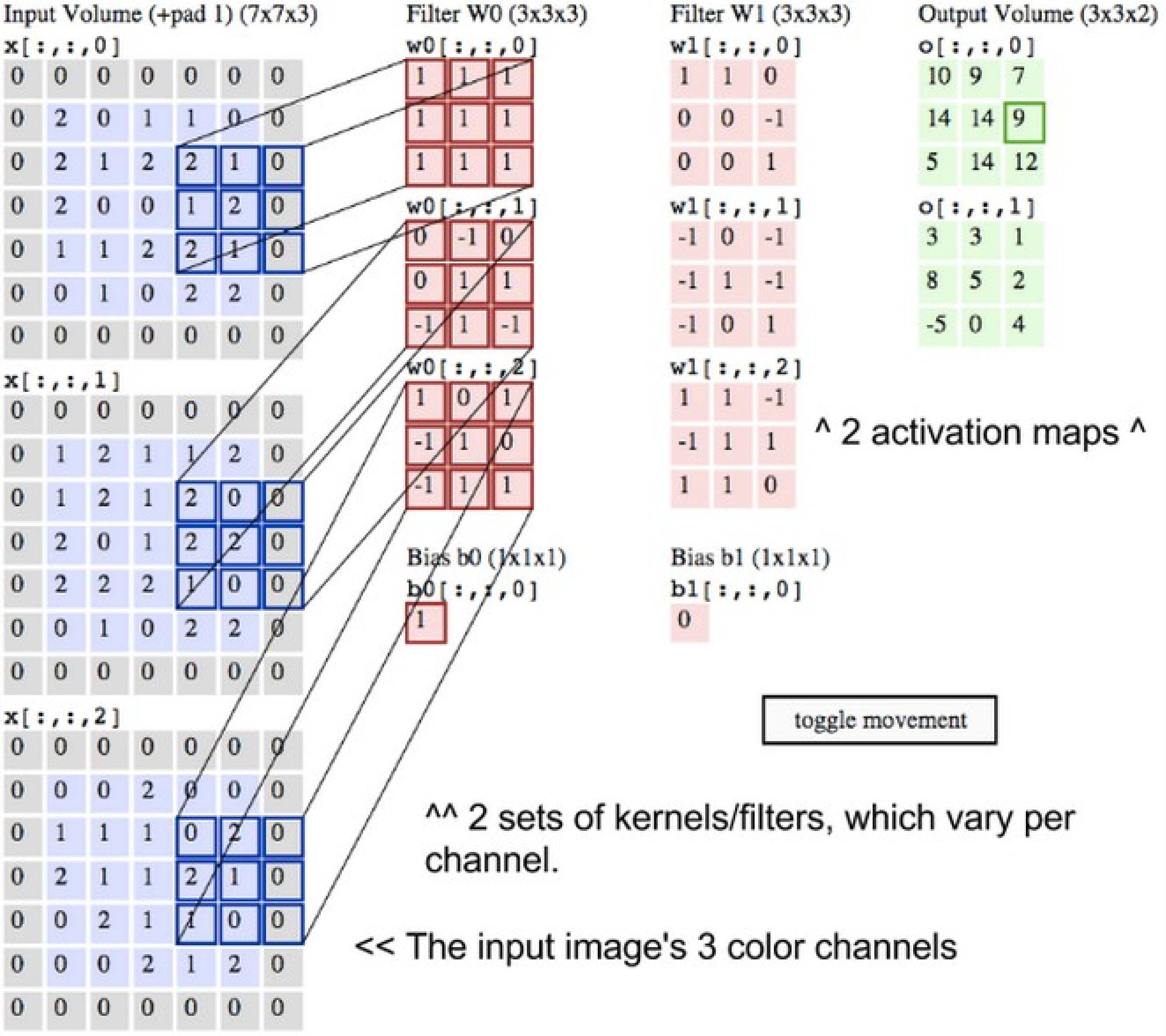
Convolutional layers

Two dimensional example:



Why do you think they call this “convolution”?

Convolutional layers

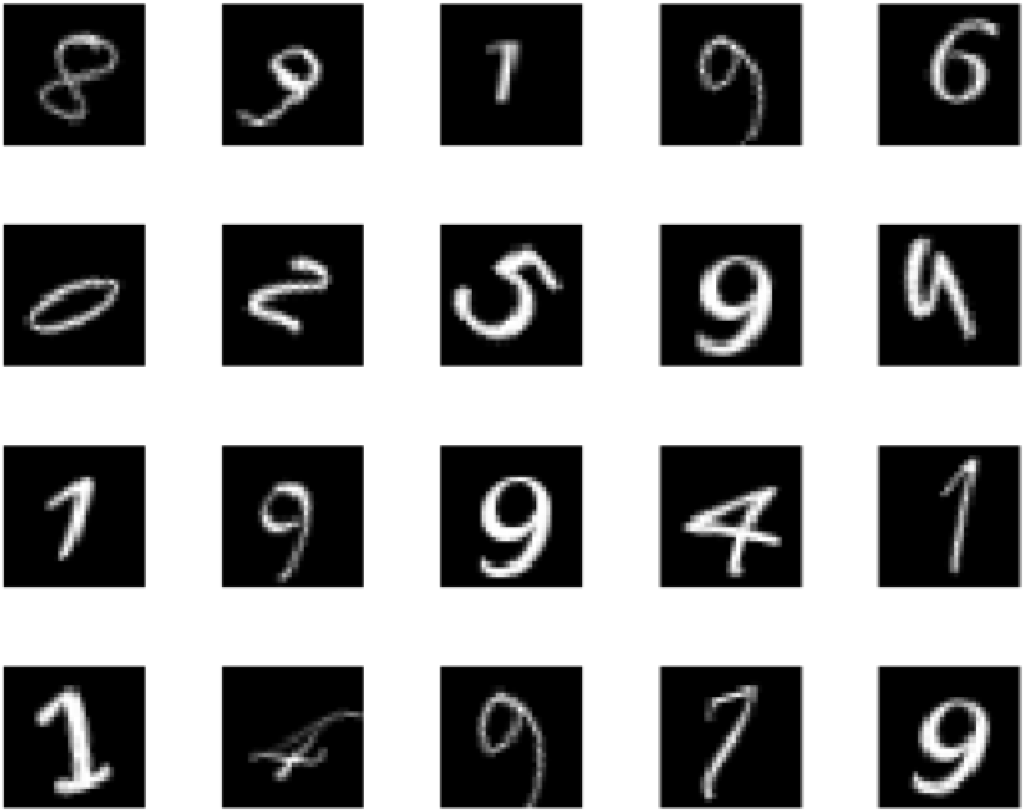


^ 2 activation maps ^

^^ 2 sets of kernels/filters, which vary per channel.

<< The input image's 3 color channels

Example: MNIST digit classification with LeNet

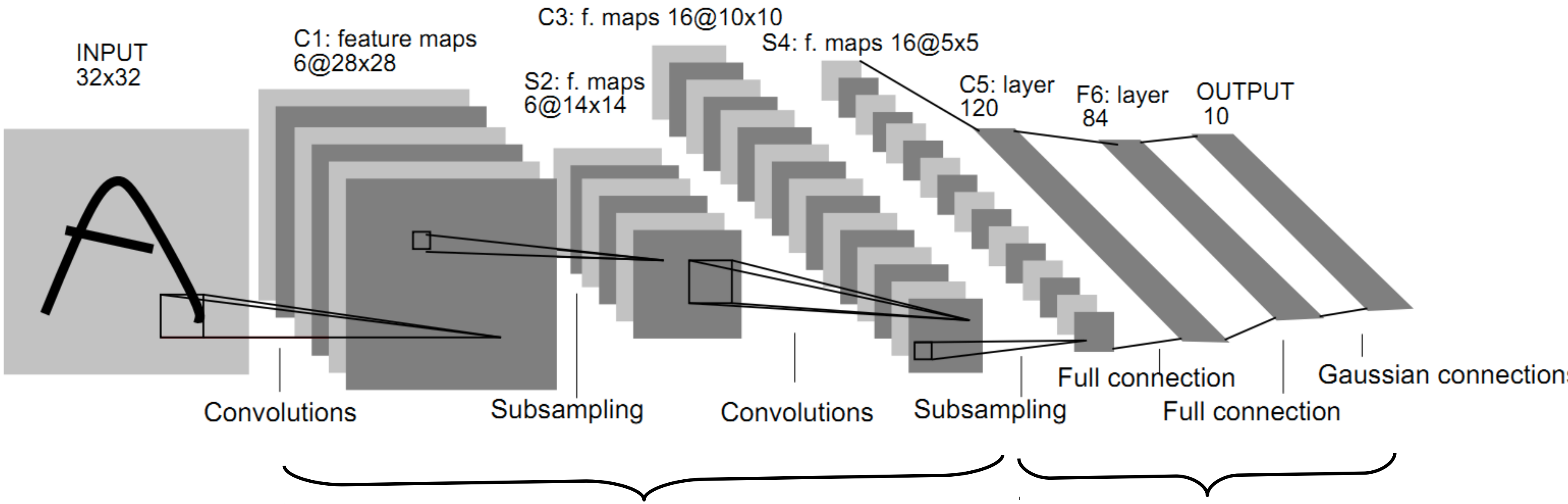


MNIST dataset: images of 10,000 handwritten digits

Objective: classify each image as the corresponding digit

Example: MNIST digit classification with LeNet

LeNet:



two convolutional layers
– conv, relu, pooling

two fully connected layers
– relu
– last layer has logistic activation function

Example: MNIST digit classification with LeNet

Load dataset, create train/test splits

```
digitDatasetPath = fullfile(matlabroot,'toolbox','nnet','nndemos',...
    'nndatasets','DigitDataset');
digitData = imageDatastore(digitDatasetPath,...
    'IncludeSubfolders',true,'LabelSource','foldernames');

trainNumFiles = 750;
[trainDigitData,valDigitData] = splitEachLabel(digitData,trainNumFiles,'randomize');
```

Example: MNIST digit classification with LeNet

Define the neural network structure:

```
layers = [
    imageInputLayer([28 28 1])
    convolution2dLayer(3,16,'Padding',1)
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2,'Stride',2)
    convolution2dLayer(3,32,'Padding',1)
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2,'Stride',2)
    convolution2dLayer(3,64,'Padding',1)
    batchNormalizationLayer
    reluLayer
    fullyConnectedLayer(50)
    reluLayer
    fullyConnectedLayer(10)
    softmaxLayer
    classificationLayer];

options = trainingOptions('sgdm',...
'MaxEpochs',3, ...
'ValidationData',valDigitData,...
'ValidationFrequency',30,...
'Verbose',true,...
'ExecutionEnvironment','gpu',...
'Plots','training-progress');
```

Input

Conv1

Conv2

FC1

FC2

Example: MNIST digit classification with LeNet

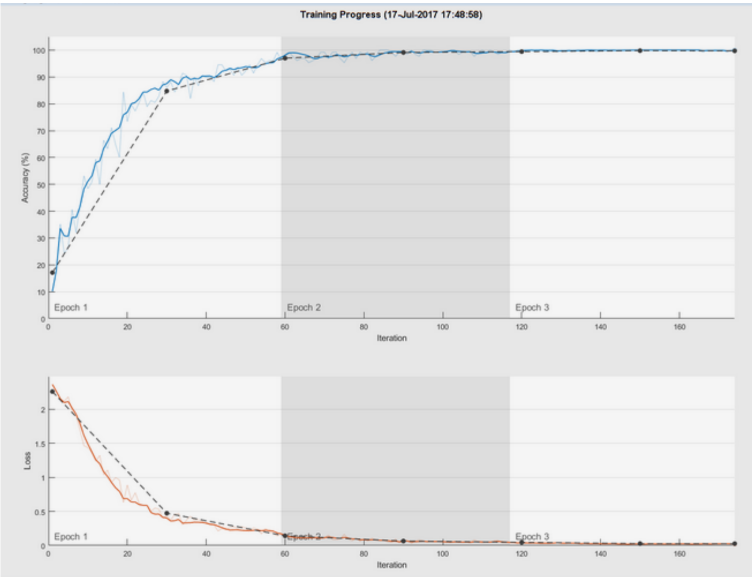
<u>Train network, classify test set, measure accuracy</u>

– notice we test on a different set (a holdout set) than we trained on

```
net = trainNetwork(trainDigitData,layers,options);

predictedLabels = classify(net,valDigitData);
valLabels = valDigitData.Labels;

accuracy = sum(predictedLabels == valLabels)/numel(valLabels);
```



Using the GPU makes
a huge differece...

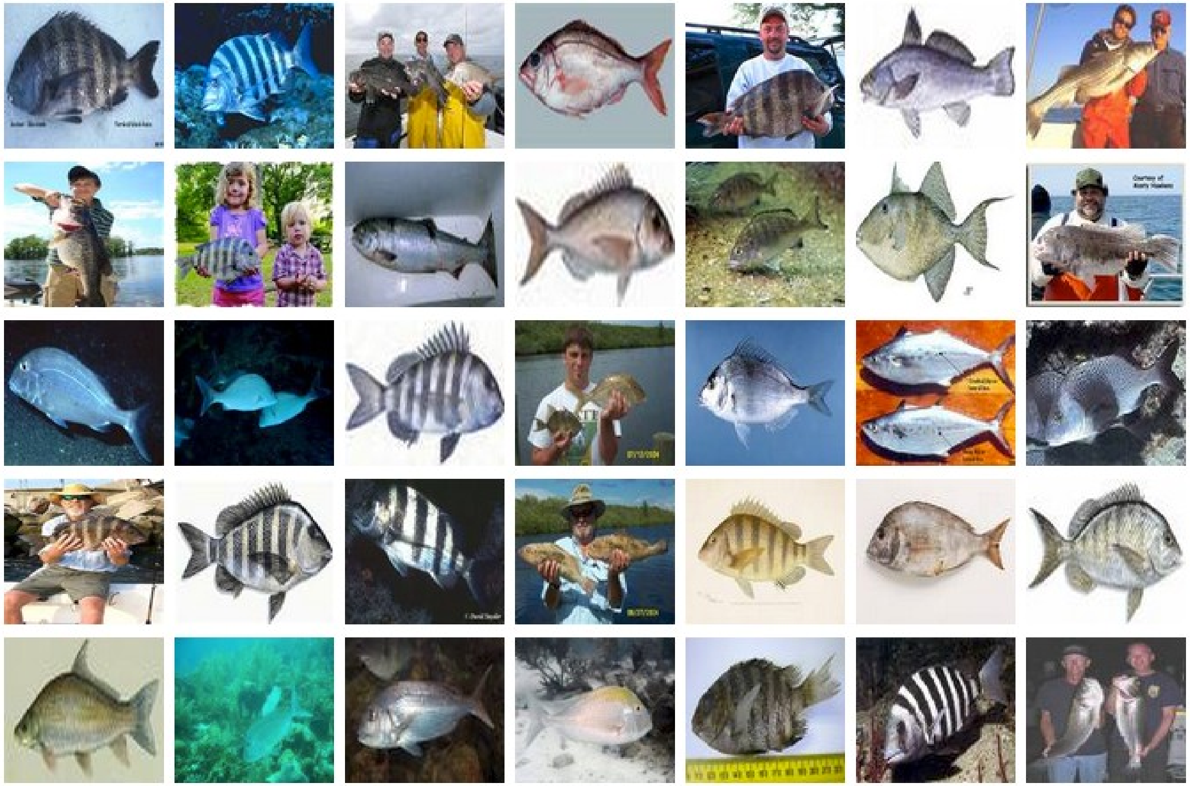
Deep learning packages

You don't need to use Matlab (obviously)

Tensorflow is probably the most popular platform

Caffe and Theano are also big

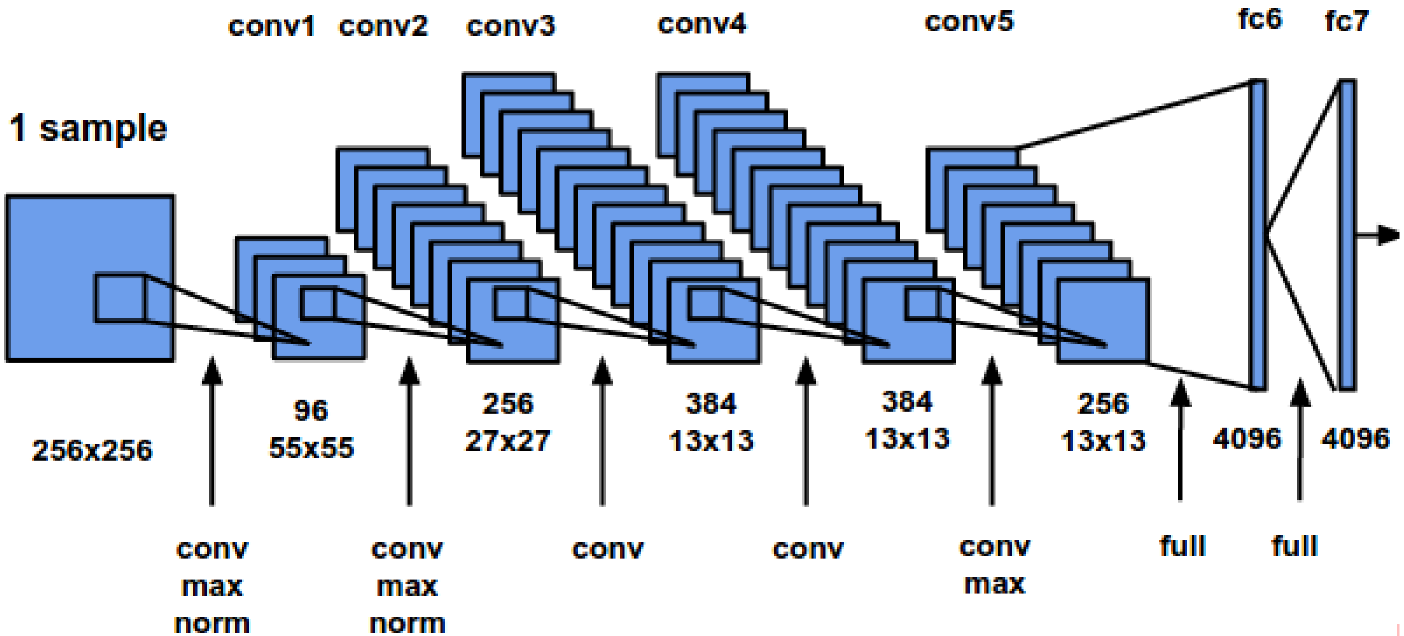
Another example: image classification w/ AlexNet



ImageNet dataset: millions of images of objects

Objective: classify each image as the corresponding object (1k categories in ILSVRC)

Another example: image classification w/ AlexNet



AlexNet has 8 layers: five conv followed by three fully connected

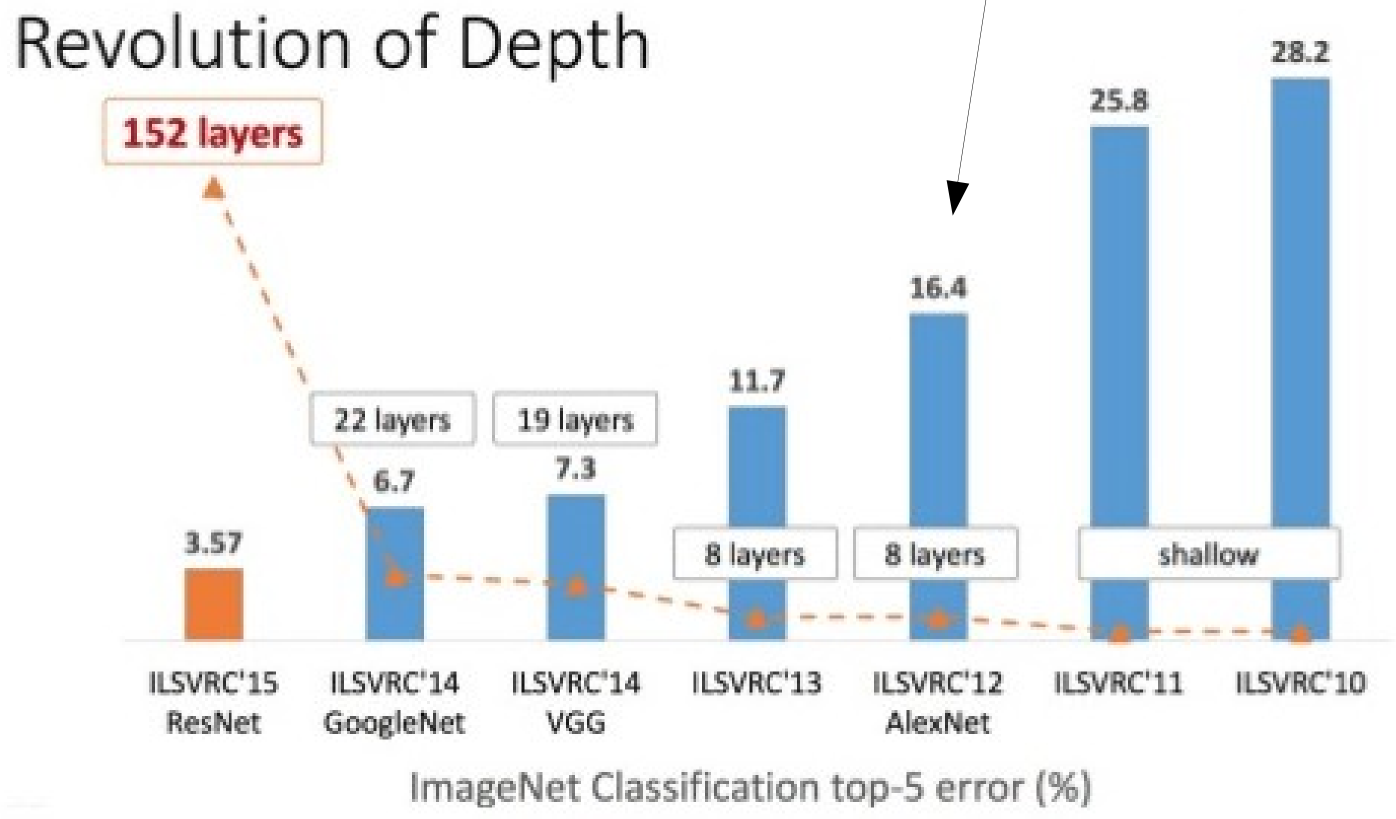
Another example: image classification w/ AlexNet

1	'data'	Image Input	227x227x3 images with 'zerocenter' normalization
2	'conv1'	Convolution	96 11x11x3 convolutions with stride [4 4] and padding [0 0]
3	'relu1'	ReLU	ReLU
4	'norm1'	Cross Channel Normalization	cross channel normalization with 5 channels per element
5	'pool1'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0]
6	'conv2'	Convolution	256 5x5x48 convolutions with stride [1 1] and padding [2 2]
7	'relu2'	ReLU	ReLU
8	'norm2'	Cross Channel Normalization	cross channel normalization with 5 channels per element
9	'pool2'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0]
10	'conv3'	Convolution	384 3x3x256 convolutions with stride [1 1] and padding [1 1]
11	'relu3'	ReLU	ReLU
12	'conv4'	Convolution	384 3x3x192 convolutions with stride [1 1] and padding [1 1]
13	'relu4'	ReLU	ReLU
14	'conv5'	Convolution	256 3x3x192 convolutions with stride [1 1] and padding [1 1]
15	'relu5'	ReLU	ReLU
16	'pool5'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0]
17	'fc6'	Fully Connected	4096 fully connected layer
18	'relu6'	ReLU	ReLU
19	'drop6'	Dropout	50% dropout
20	'fc7'	Fully Connected	4096 fully connected layer
21	'relu7'	ReLU	ReLU
22	'drop7'	Dropout	50% dropout
23	'fc8'	Fully Connected	1000 fully connected layer
24	'prob'	Softmax	softmax
25	'output'	Classification Output	crossentropyex with 'tench', 'goldfish', and 998 other classes

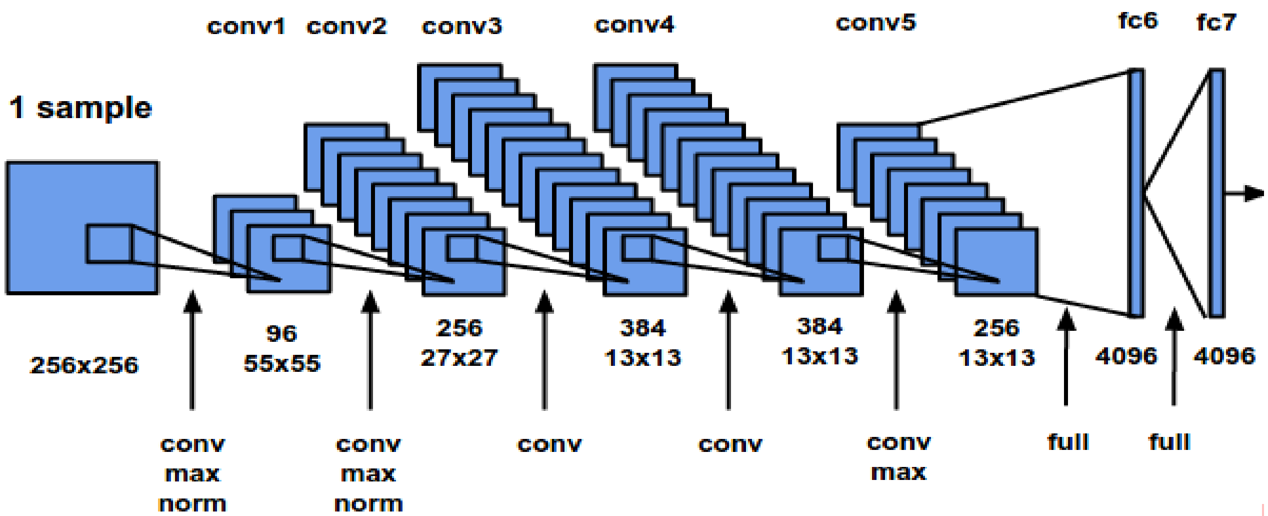
AlexNet has 8 layers: five conv followed by three fully connected

Another example: image classification w/ AlexNet

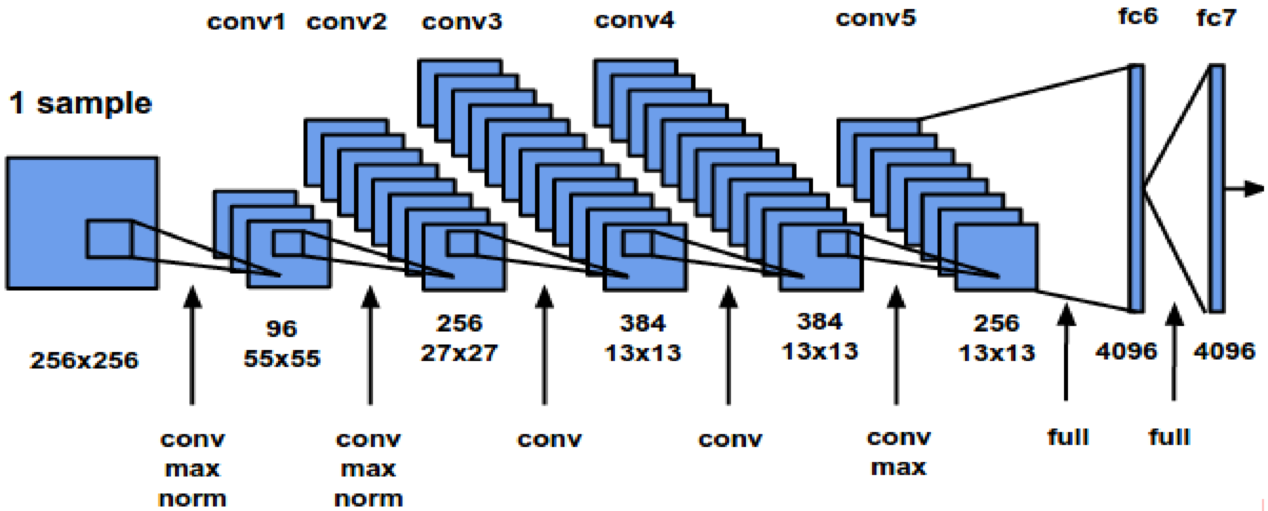
AlexNet won the 2012 ILSVRC challenge
– sparked the deep learning craze



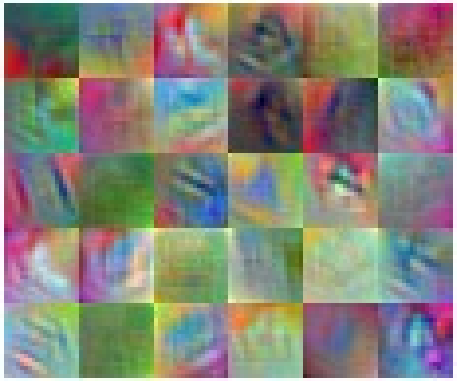
What exactly are deep conv networks learning?



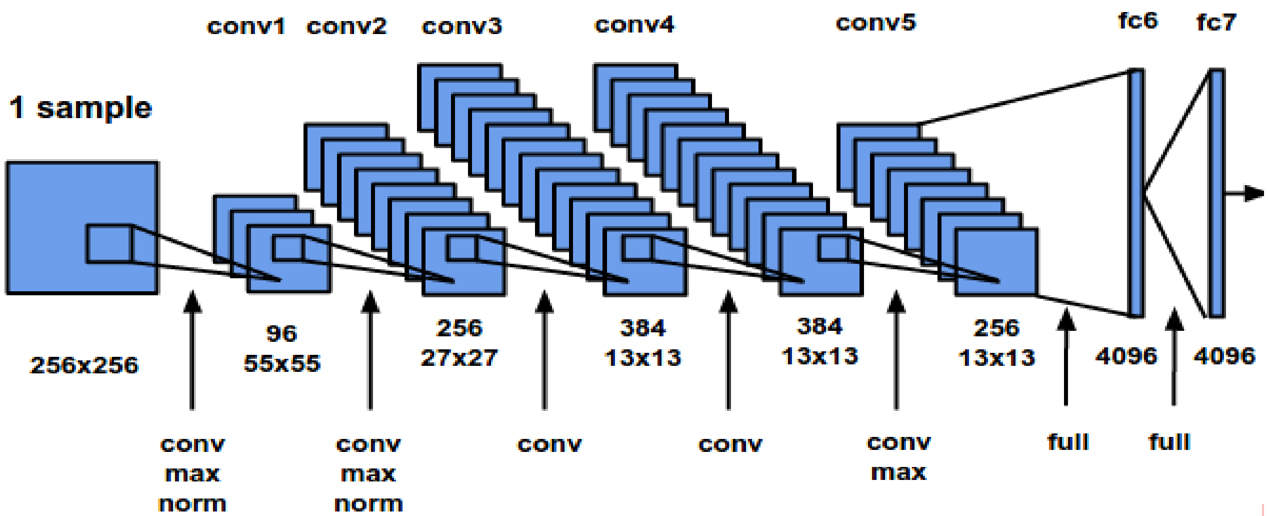
What exactly are deep conv networks learning?



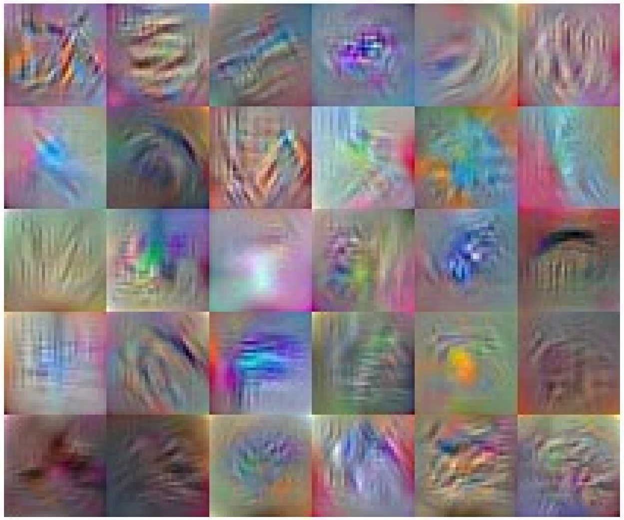
Layer conv2 Features



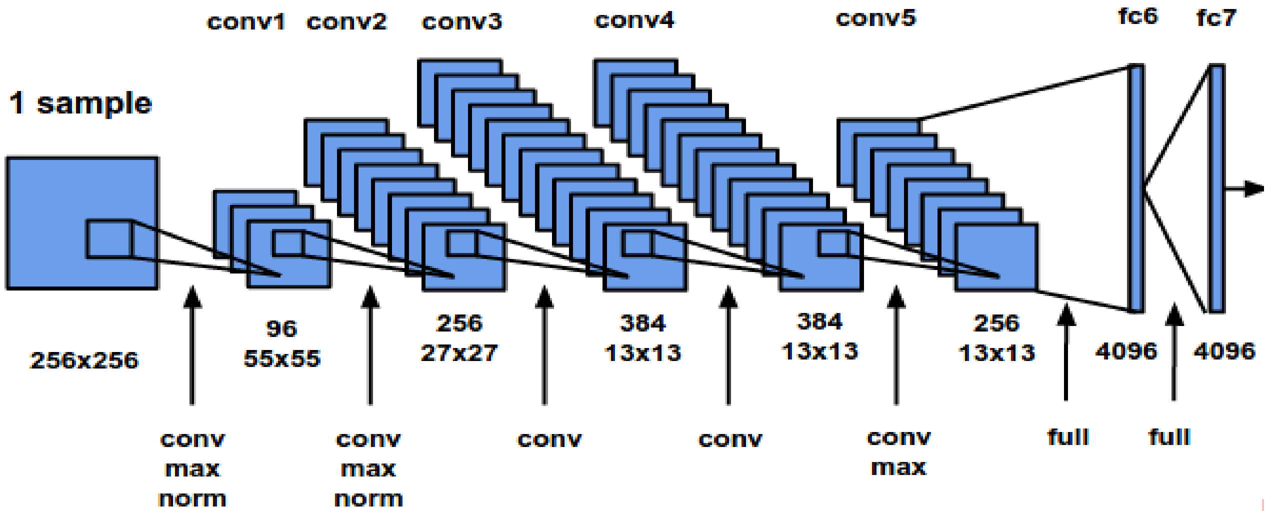
What exactly are deep conv networks learning?



Layer conv3 Features

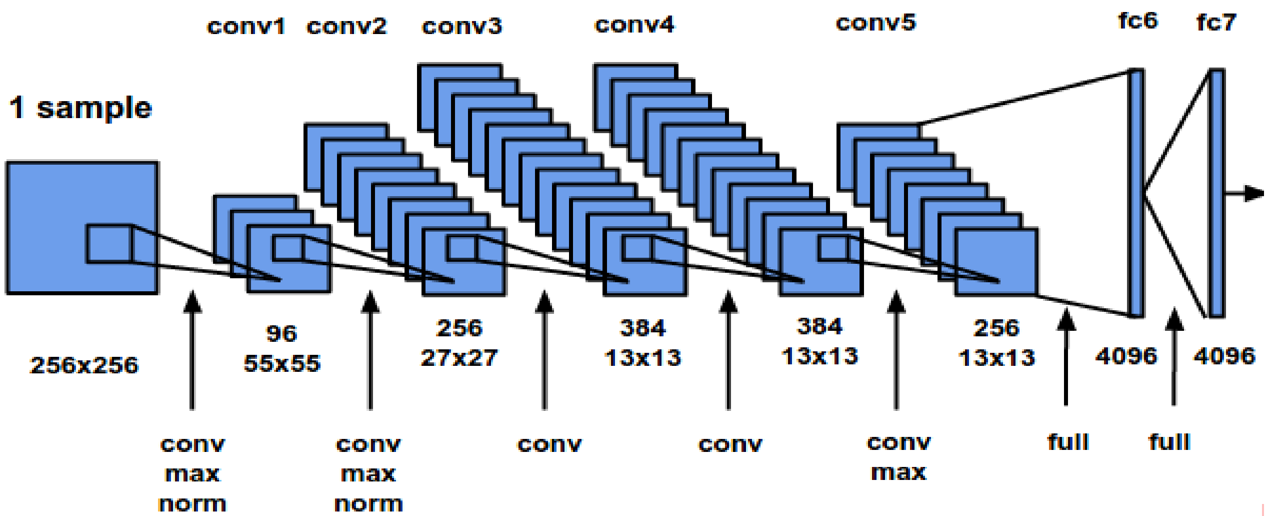


What exactly are deep conv networks learning?

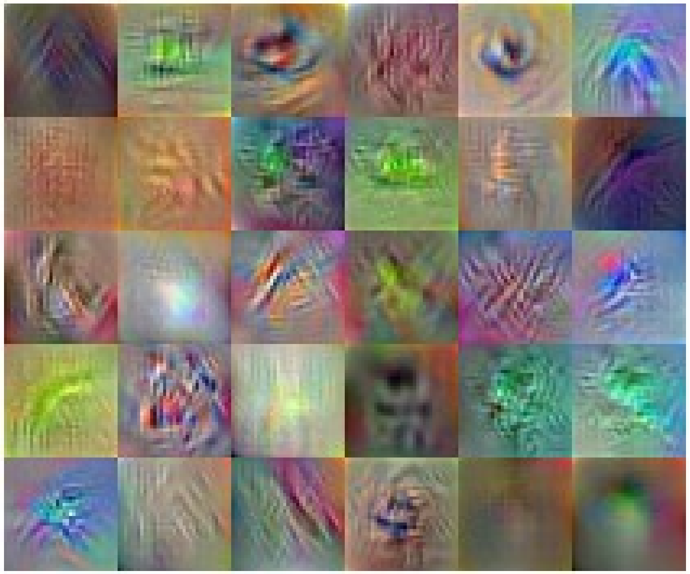


Layer conv4 Features

What exactly are deep conv networks learning?

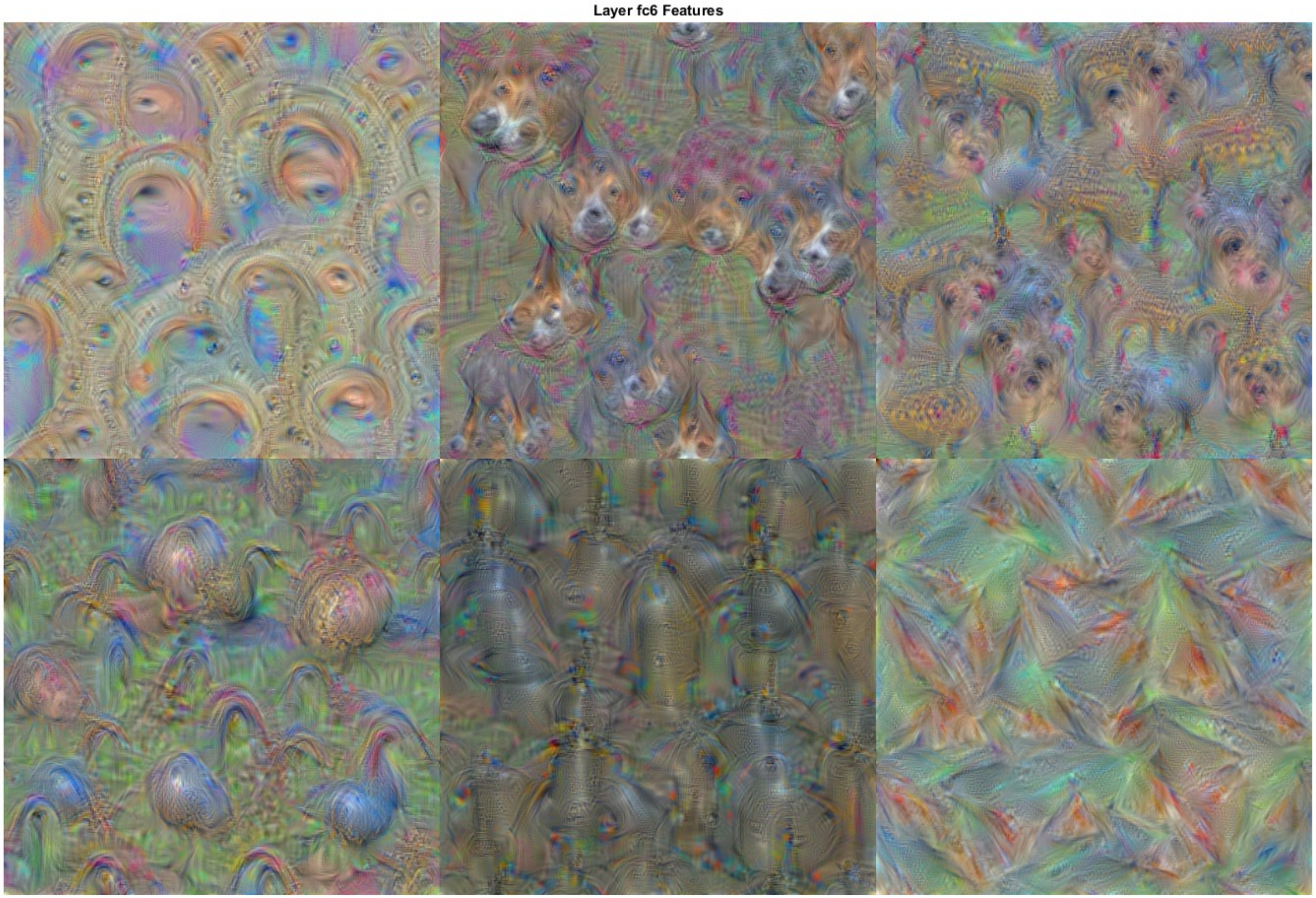


Layer conv5 Features



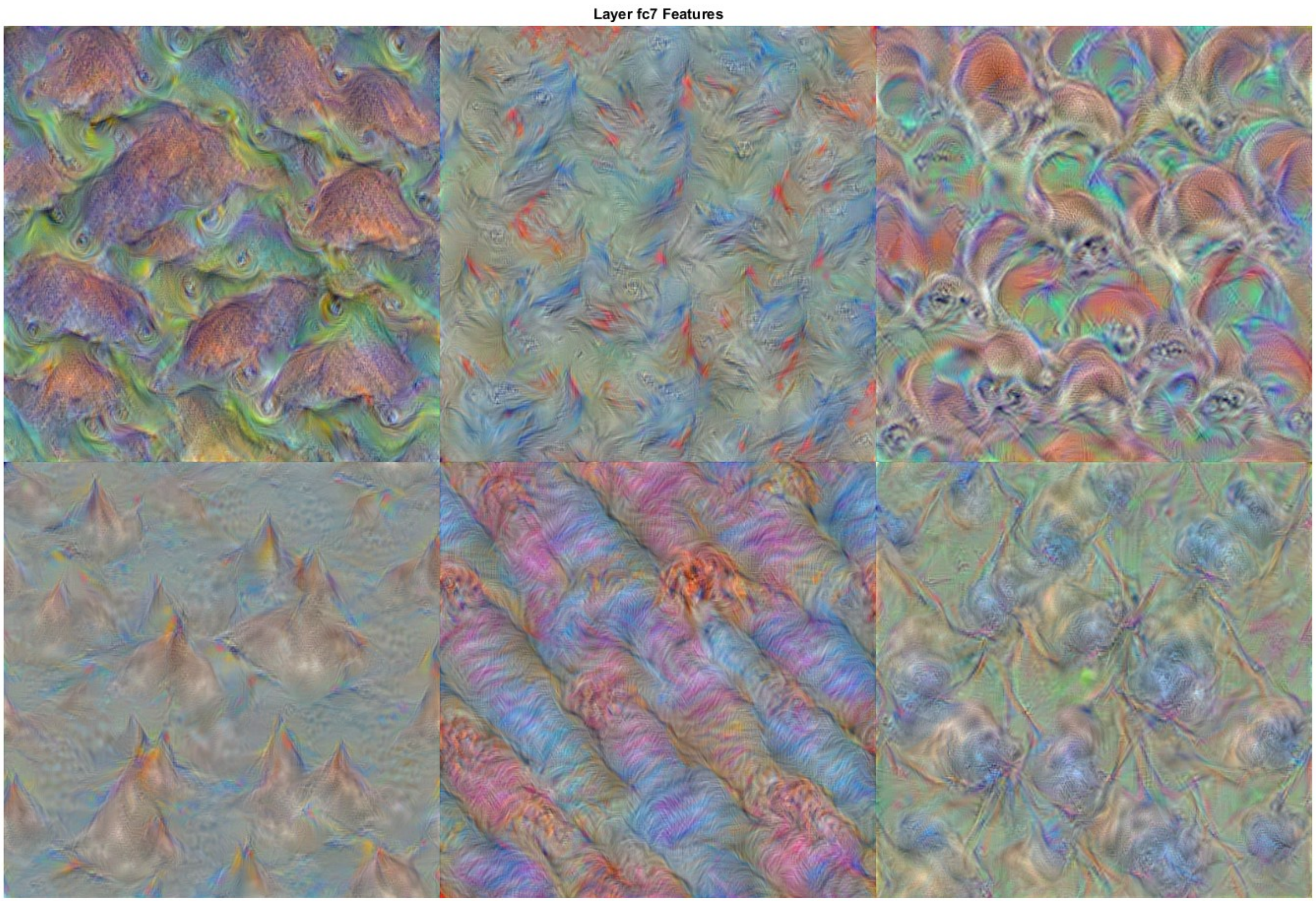
What exactly are deep conv networks learning?

FC layer 6



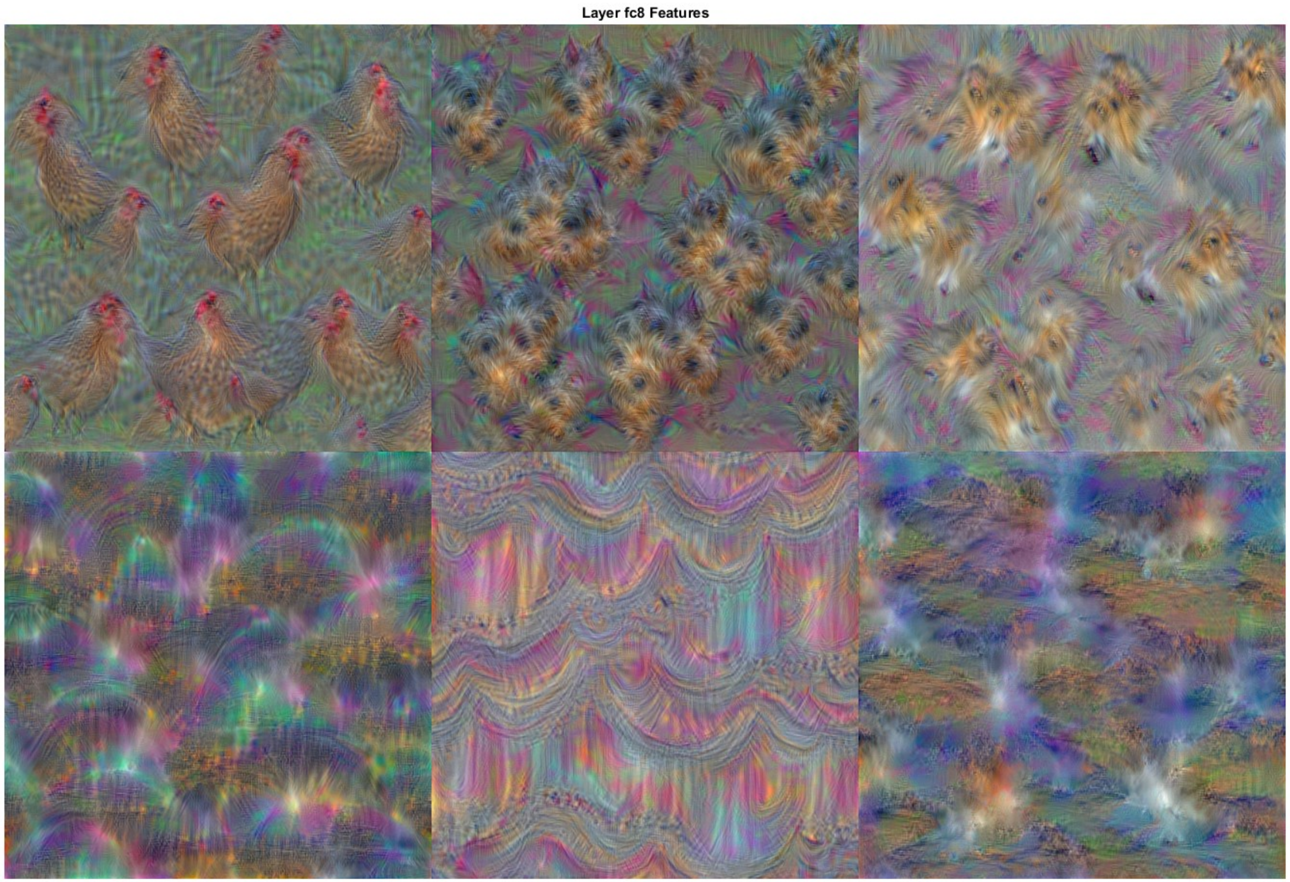
What exactly are deep conv networks learning?

FC layer 7



What exactly are deep conv networks learning?

Output layer



Finetuning

AlexNet has 60M parameters
– therefore, you need a very large training set (like imagenet)

Suppose we want to train on our own images, but we only have a few hundred?
– AlexNet will drastically overfit such a small dataset… (won't generalize at all)

Finetuning

<u>Idea:</u>
1. pretrain on imagenet
2. finetune on your own dataset

AlexNet has 60M parameters
– therefore, you need a very large training set (like imagenet)

Suppose we want to train on our own images, but we only have a few hundred?
– AlexNet will drastically overfit such a small dataset… (won't generalize at all)