COMP9444
Neural Networks and Deep Learning

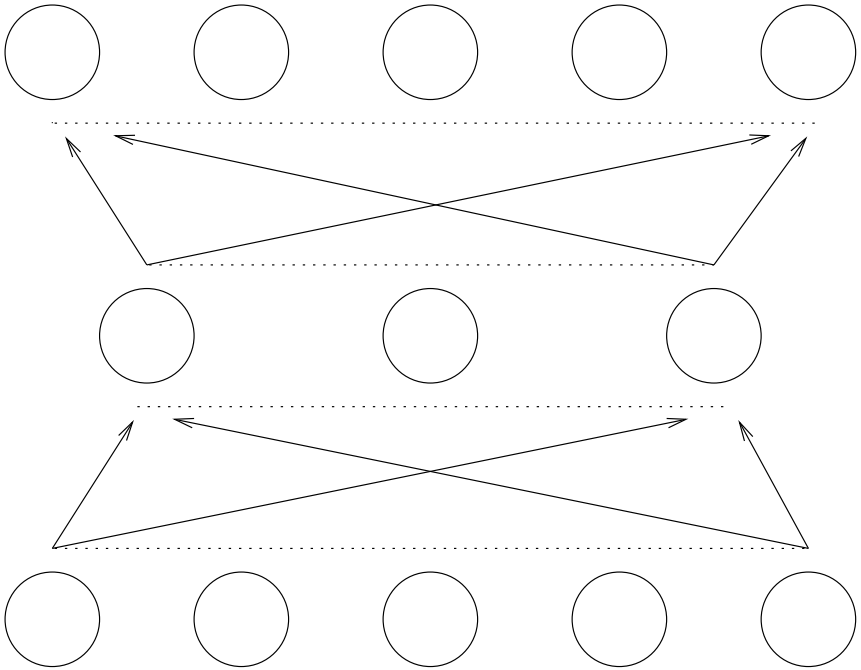
5. Convolutional Networks

Textbook, Sections 6.2.2, 6.3, 7.9, 7.11-7.13, 9.1-9.5

Outline

- Geometry of Hidden Unit Activations
- Limitations of 2-layer networks
- Alternative transfer functions (6.3)
- Convolutional Networks (9.1-9.5)
- Dropout

Encoder Networks

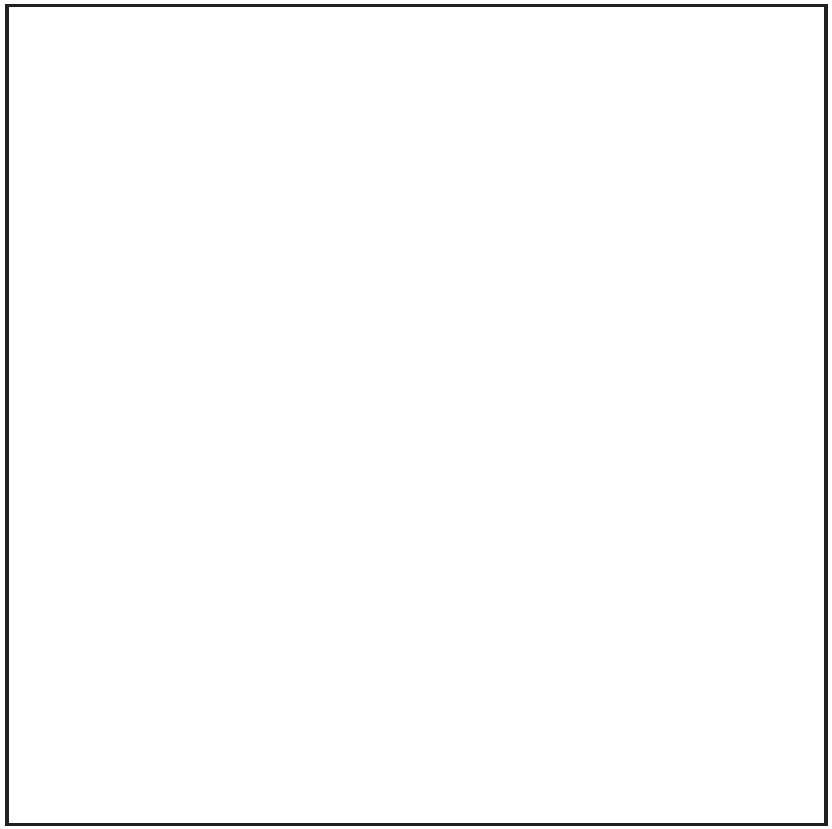


Inputs	Outputs
10000	10000
01000	01000
00100	00100
00010	00010
00001	00001

- identity mapping through a bottleneck
- also called N–M–N task
- used to investigate hidden unit representations

N–2–N Encoder

HU Space:



8–3–8 Encoder

Exercise:

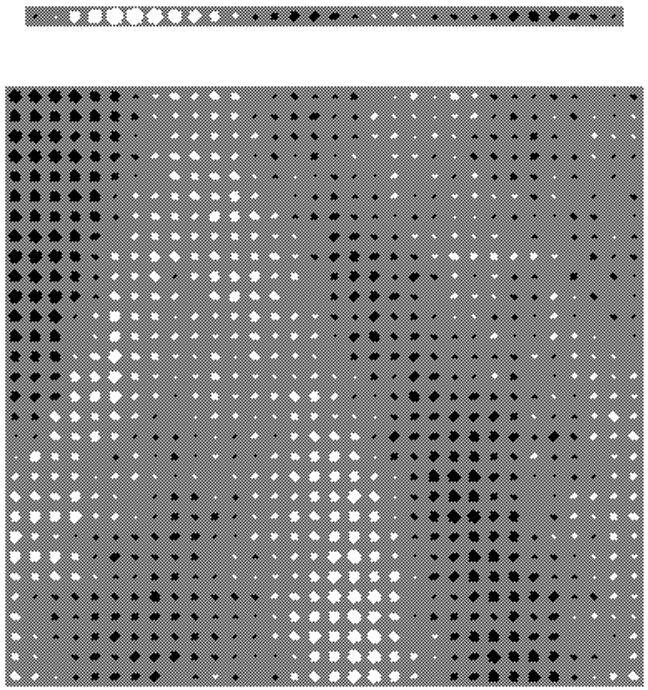
Draw the hidden unit space for 2-2-2, 3-2-3, 4-2-4 and 5-2-5 encoders.

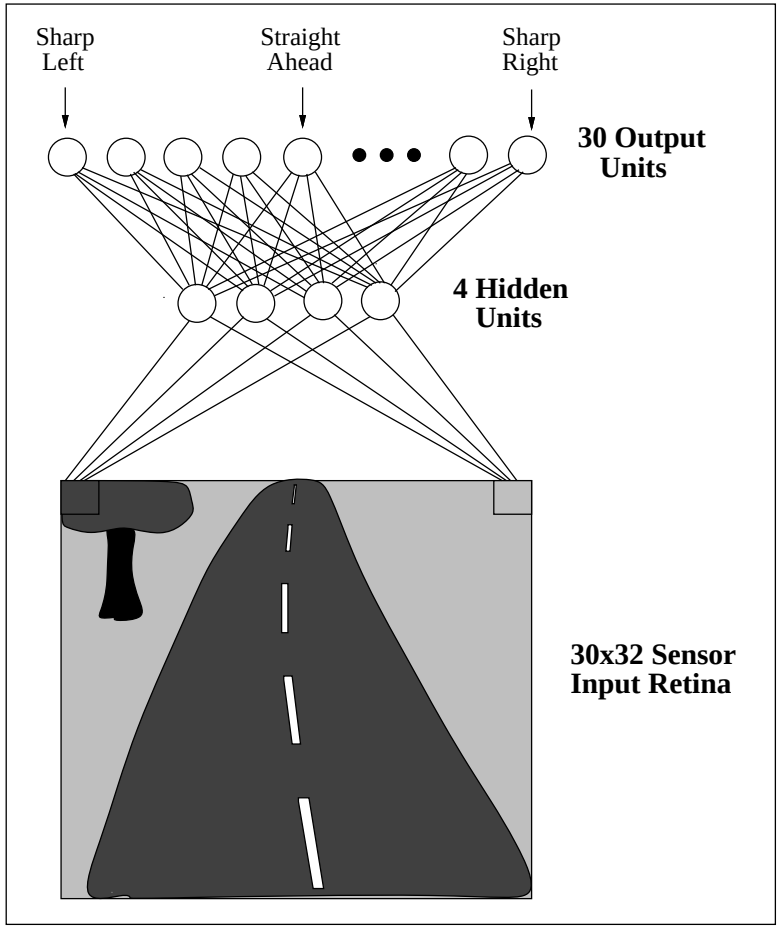
Represent the input-to-hidden weights for each input unit by a point, and the hidden-to-output weights for each output unit by a line.

Now consider the 8-3-8 encoder with its 3-dimensional hidden unit space. What shape would be formed by the 8 points representing the input-to-hidden weights for the 8 input units? What shape would be formed by the planes representing the hidden-to-output weights for each output unit?

Hint: think of two platonic solids, which are “dual” to each other.

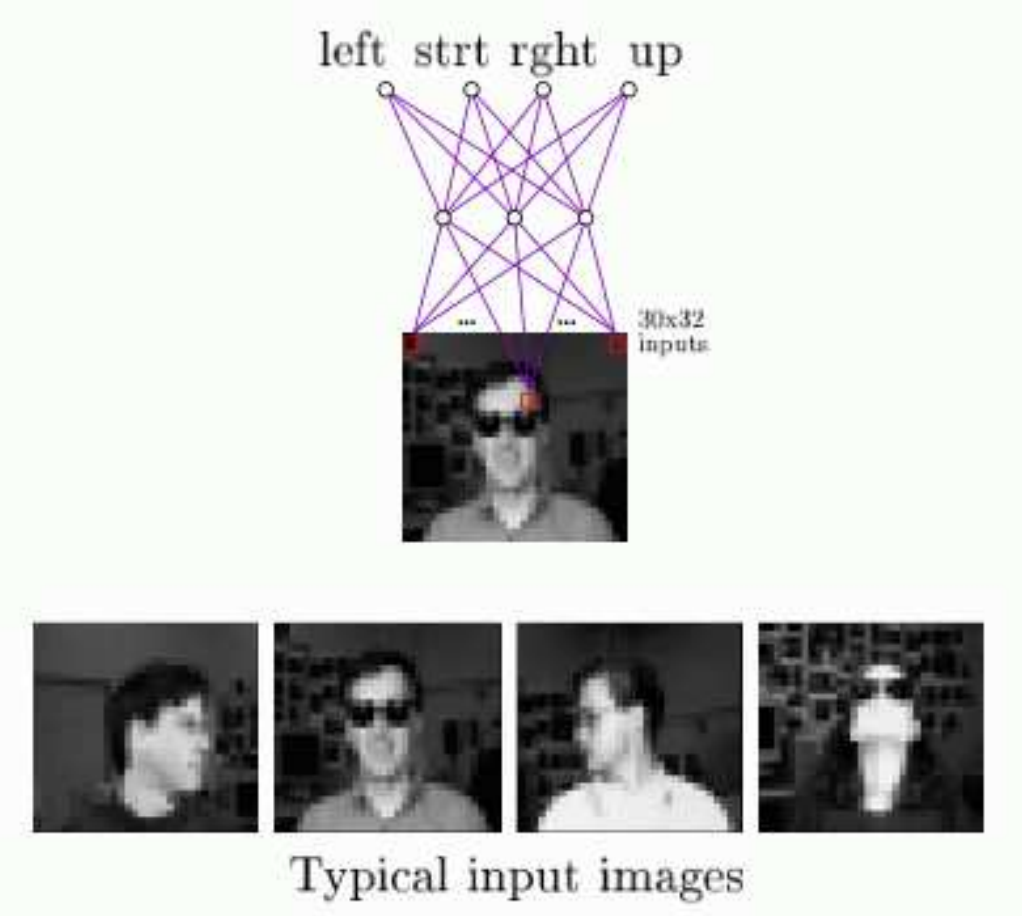
Hinton Diagrams



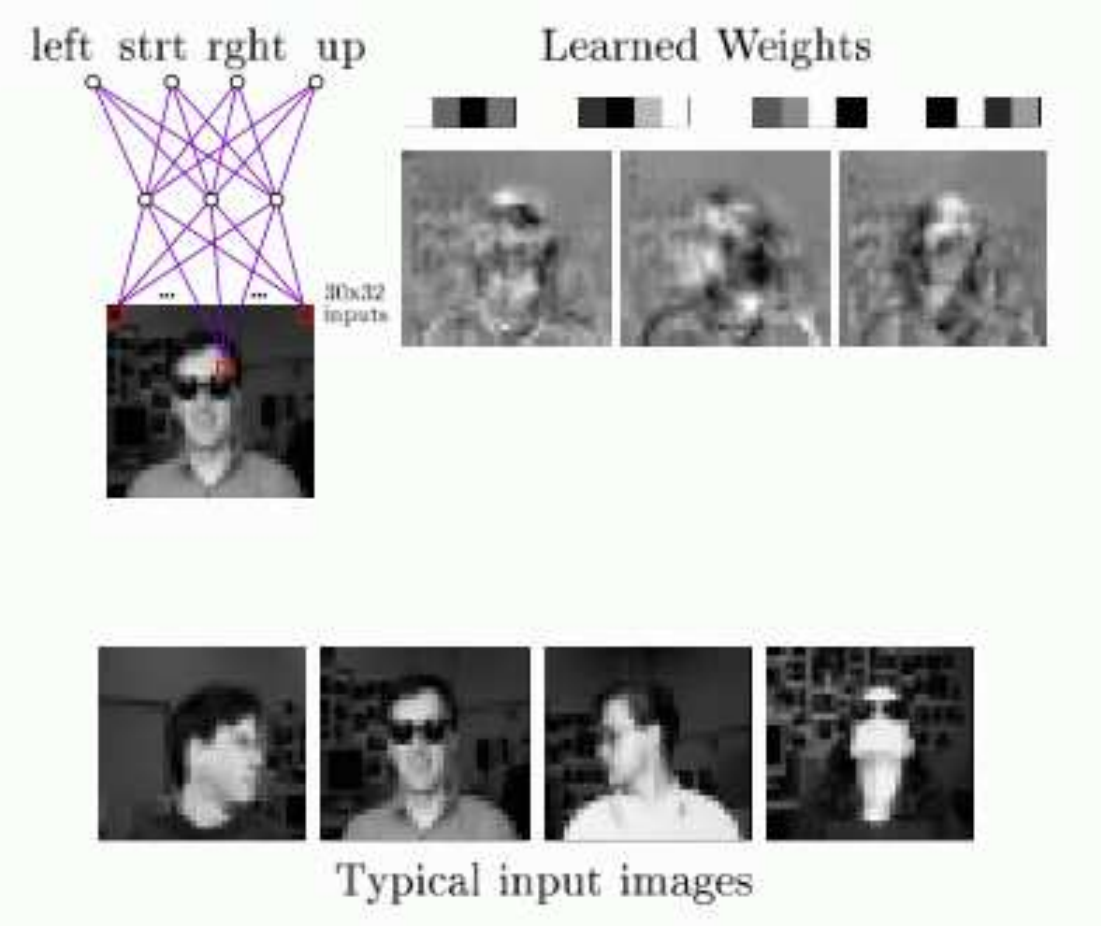


- used to visualize higher dimensions
- white = positive, black = negative

Learning Face Direction



Learning Face Direction



Symmetries

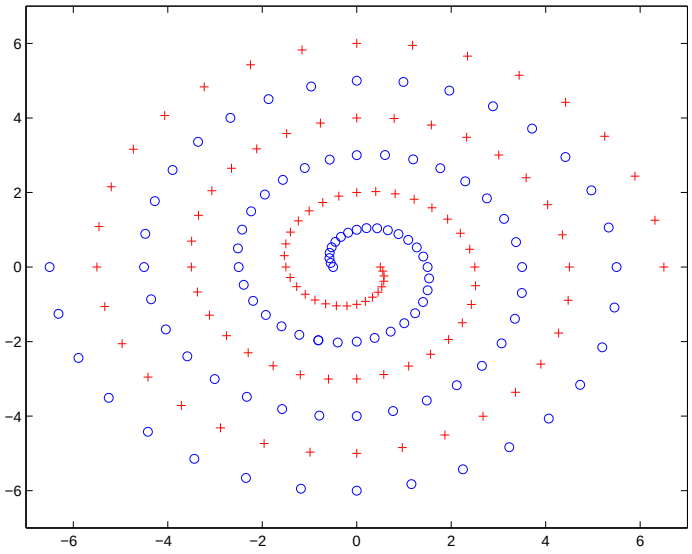
- swap any pair of hidden nodes, overall function will be the same
- on any hidden node, reverse the sign of all incoming and outgoing weights (assuming symmetric transfer function)
- hidden nodes with identical input-to-hidden weights in theory would never separate; so, they all have to begin with different (small) random weights
- in practice, all hidden nodes try to do similar job at first, then gradually specialize.

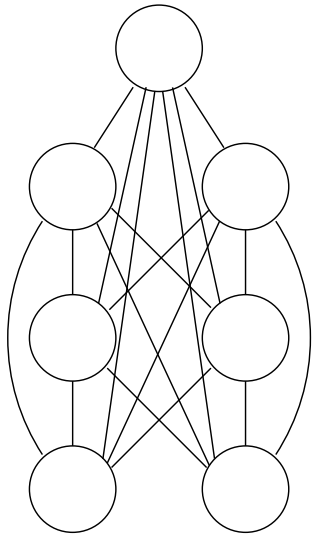
Controlled Nonlinearity

- for small weights, each layer implements an approximately linear function, so multiple layers also implement an approximately linear function.
- for large weights, transfer function approximates a step function, so computation becomes digital and learning becomes very slow.
- with typical weight values, two-layer neural network implements a function which is close to linear, but takes advantage of a limited degree of nonlinearity.

Limitations of Two-Layer Neural Networks

Some functions cannot be learned with a 2-layer sigmoidal network.





For example, this Twin Spirals problem cannot be learned with a 2-layer network, but it can be learned using a 3-layer network if we include shortcut connections between non-consecutive layers.

Adding Hidden Layers

- Twin Spirals can be learned by 3-layer network with shortcut connections
- first hidden layer learns linearly separable features
- second hidden layer learns “convex” features
- output layer combines these to produce “concave” features
- training the 3-layer network is delicate
- learning rate and initial weight values must be very small
- otherwise, the network will converge to a local optimum

Vanishing / Exploding Gradients

Training by backpropagation in networks with many layers is difficult.

When the weights are small, the differentials become smaller and smaller as we backpropagate through the layers, and end up having no effect.

When the weights are large, the activations in the higher layers will saturate to extreme values. As a result, the gradients at those layers will become very small, and will not be propagated to the earlier layers.

When the weights have intermediate values, the differentials will sometimes get multiplied many times is places where the transfer function is steep, causing them to blow up to large values.

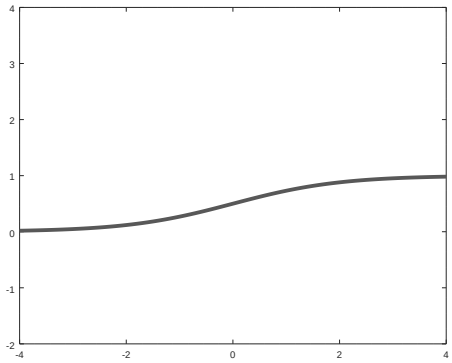
Solutions to Vanishing Gradients

- layerwise unsupervised pre-training
- long short term memory (LSTM)
- new activations functions

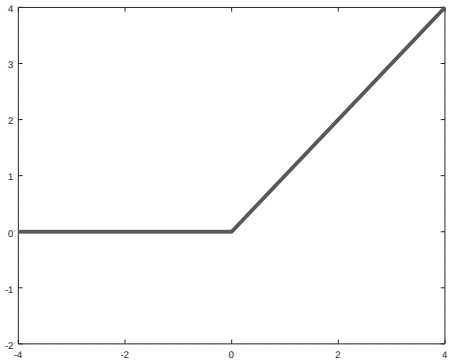
LSTM is specifically for recurrent neural networks.

We will discuss unsupervised pre-training and LSTM later in the course.

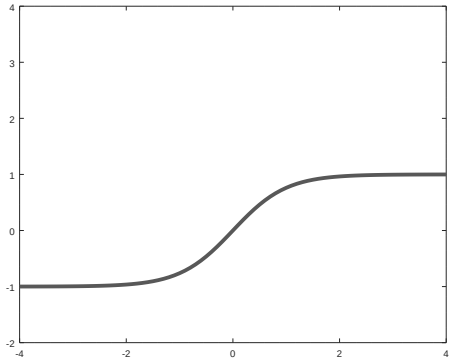
Activation Functions (6.3)



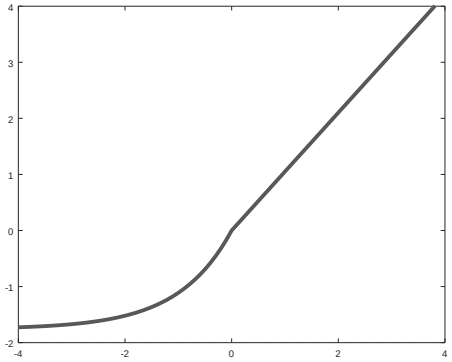
Sigmoid



Rectified Linear Unit (ReLU)



Hyperbolic Tangent

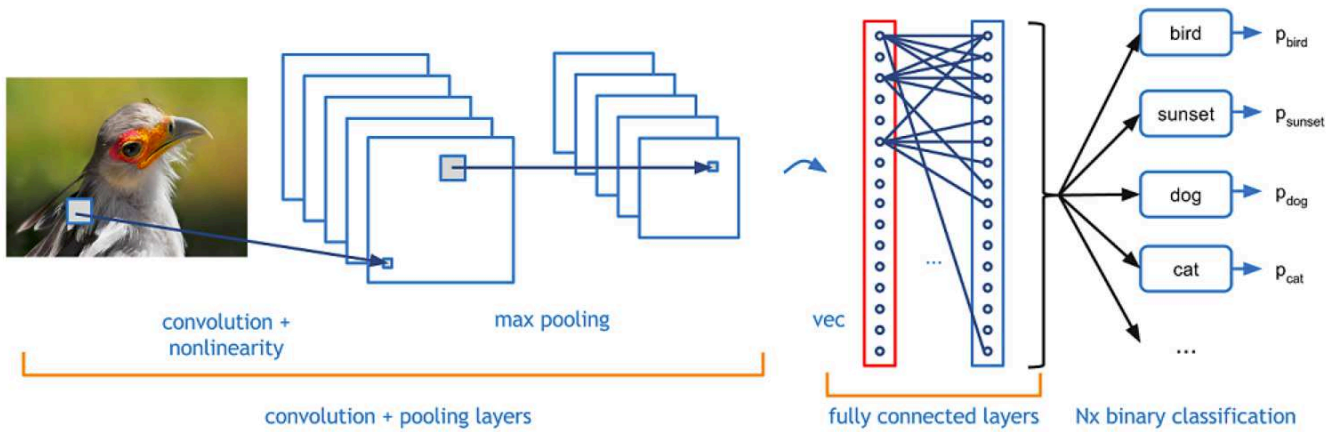


Scaled Exponential Linear Unit (SELU)

Activation Functions (6.3)

- Sigmoid and hyperbolic tangent traditionally used for 2-layer networks, but suffer from vanishing gradient problem in deeper networks.
- Rectified Linear Units (ReLUs) are popular for deep networks, including convolutional networks. Gradients don't vanish.
  But, their highly linear nature may cause other problems.
- Scaled Exponential Linear Units (SELUs) are a recent innovation which seems to work well for very deep networks.

Convolutional Networks

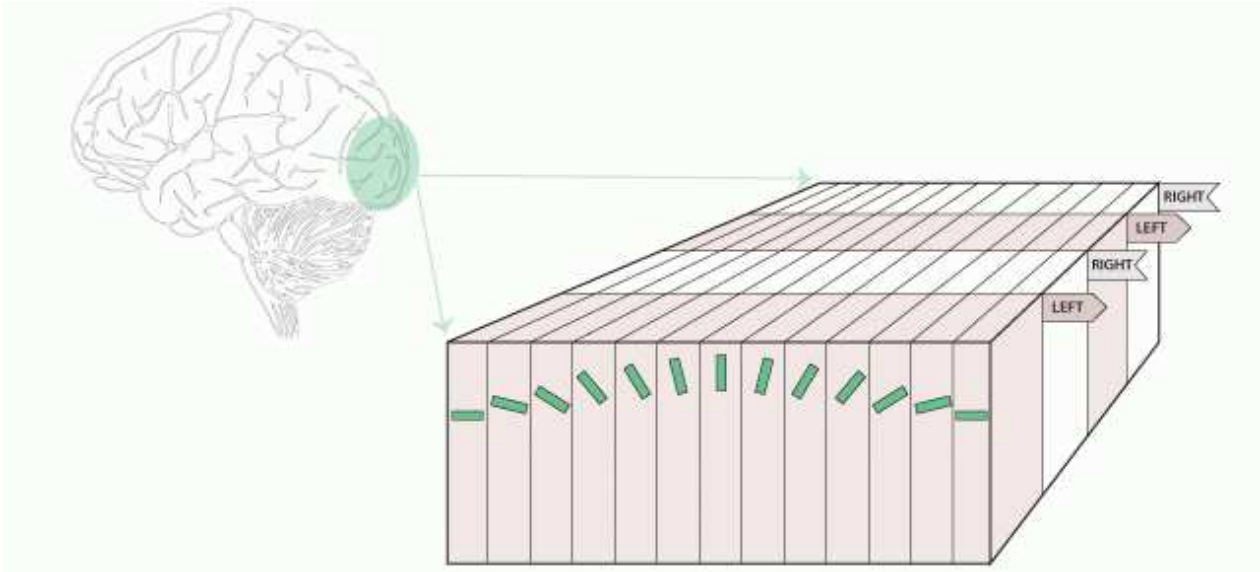


Suppose we want to classify an image as a bird, sunset, dog, cat, etc.

If we can identify features such as feather, eye, or beak which provide useful information in one part of the image, then those features are likely to also be relevant in another part of the image.

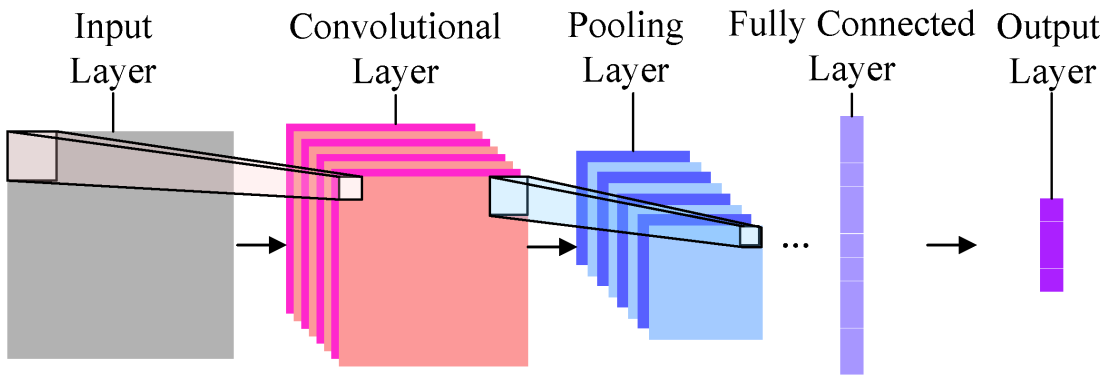
We can exploit this regularity by using a convolution layer which applies the same weights to different parts of the image.

Hubel and Weisel – Visual Cortex



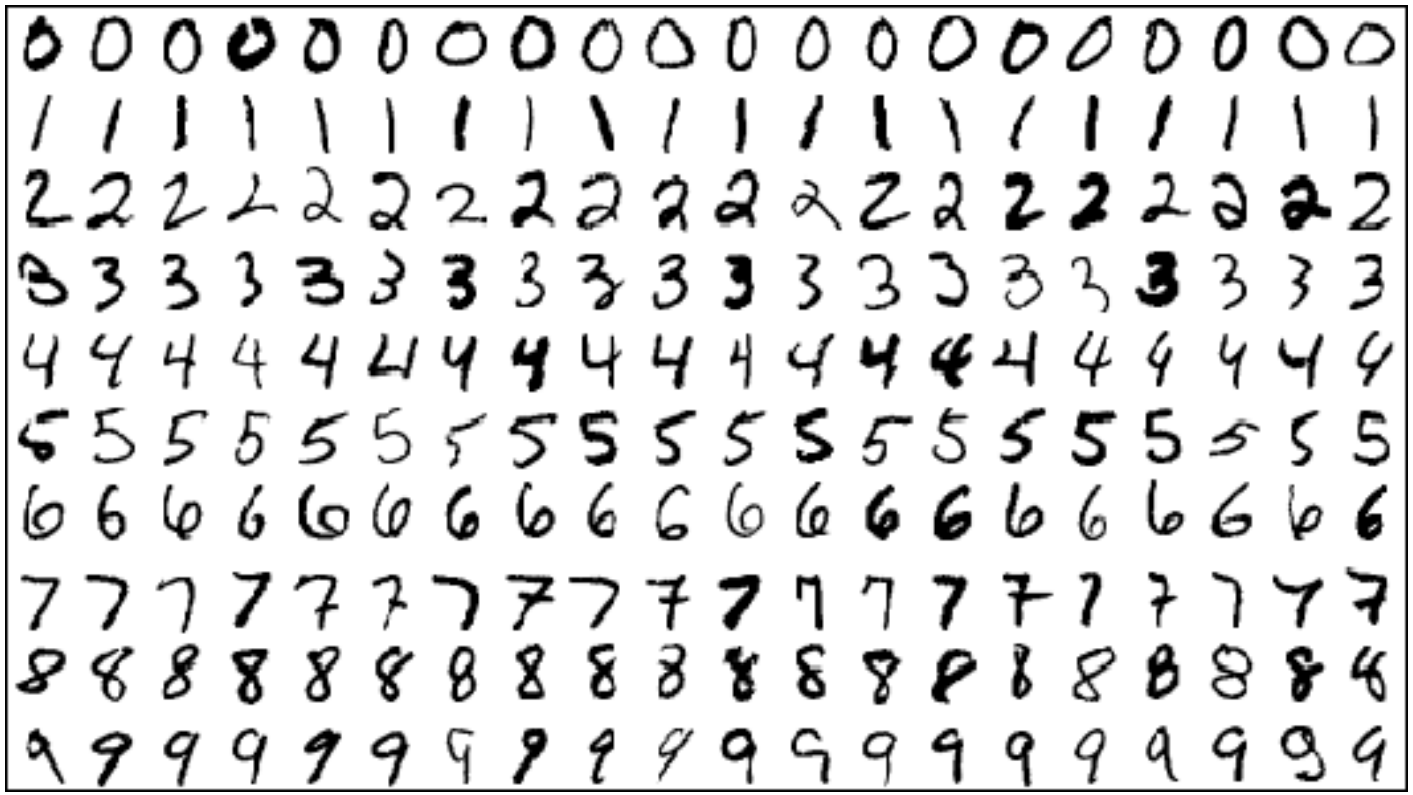
- cells in the visual cortex respond to lines at different angles
- cells in V2 respond to more sophisticated visual features
- Convolutional Neural Networks are inspired by this neuroanatomy
- CNN's can now be simulated with massive parallelism, using GPU's

Convolutional Network Components

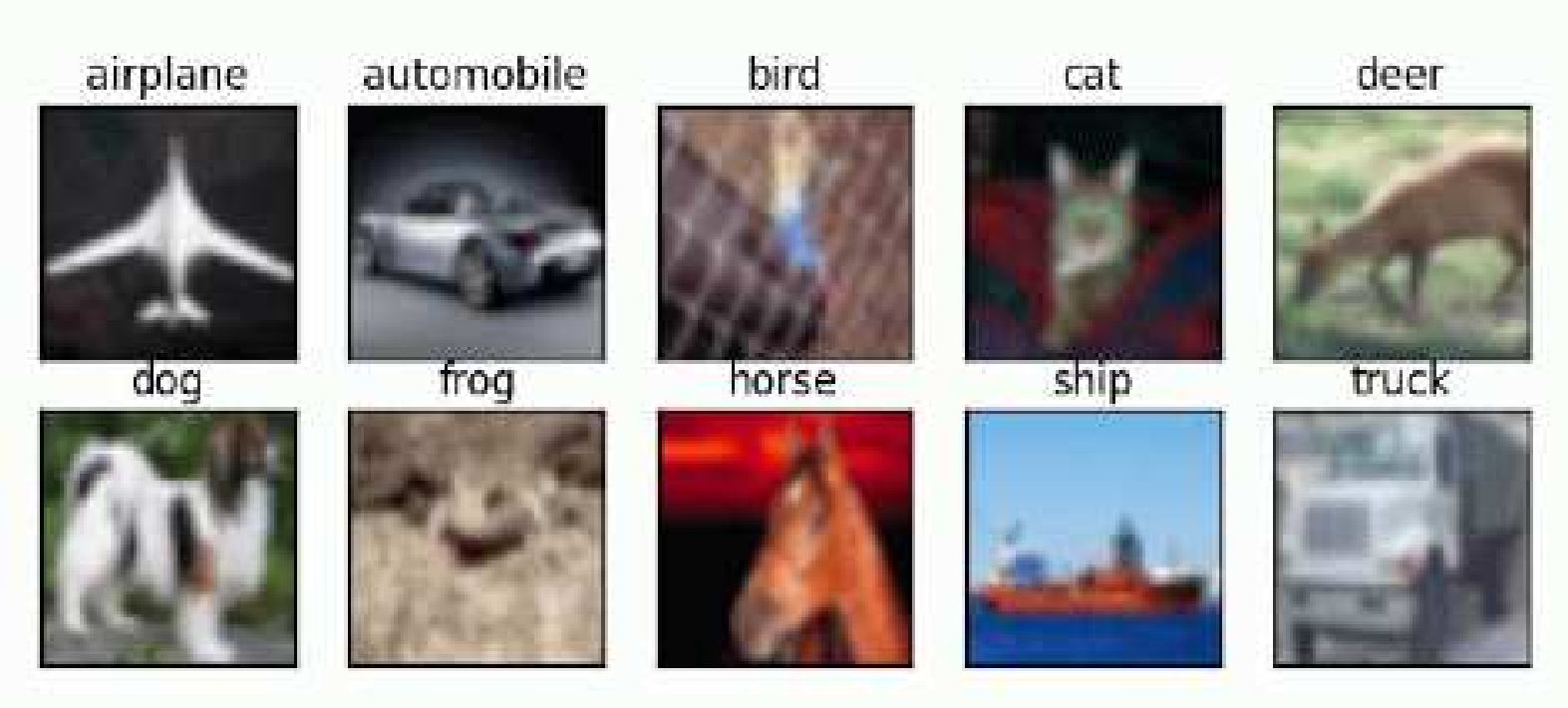


- **convolution layers**: extract shift-invariant features from the previous layer
- **subsampling or pooling layers**: combine the activations of multiple units from the previous layer into one unit
- **fully connected layers**: collect spatially diffuse information
- **output layer**: choose between classes

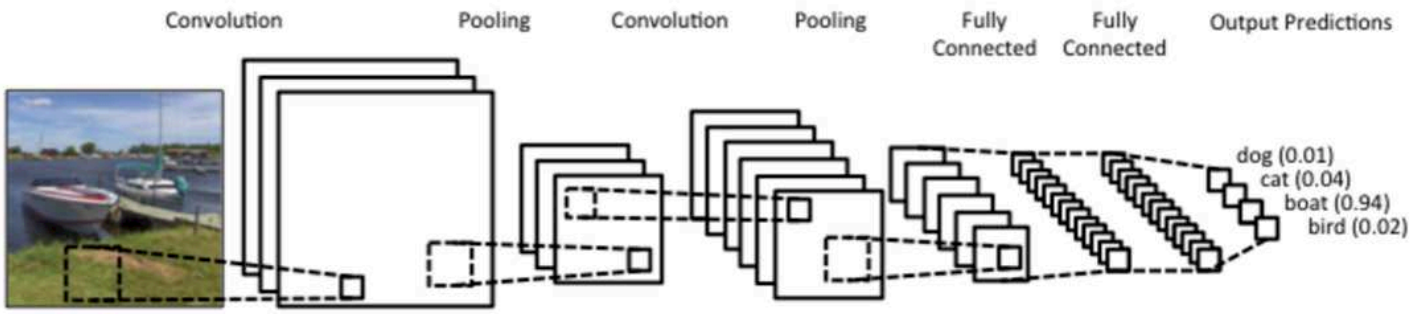
MNIST Handwritten Digit Examples



CIFAR Image Examples



Convolutional Network Architecture



There can be multiple steps of convolution followed by pooling, before reaching the fully connected layers.

Note how pooling reduces the size of the feature map (usually, by half in each direction).

Softmax (6.2.2)

Consider a classification task with N classes, and assume z_j is the output of the unit corresponding to class j.

We assume the network's estimate of the probability of each class j is proportional to $\exp(z_j)$. Because the probabilites must add up to 1, we need to normalize by dividing by their sum:

$$\text{Prob}(i) = \frac{\exp(z_i)}{\sum_{j=1}^{N} \exp(z_j)}$$

$$\log \text{Prob}(i) = z_i - \log \sum_{j=1}^{N} \exp(z_j)$$

If the correct class is i, we can treat $-\log \text{Prob}(i)$ as our cost function. The first term pushes up the correct class i, while the second term mainly pushes down the incorrect class j with the highest activation (if $j \neq i$).

Convolution Operator

Continuous convolution

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

Discrete convolution

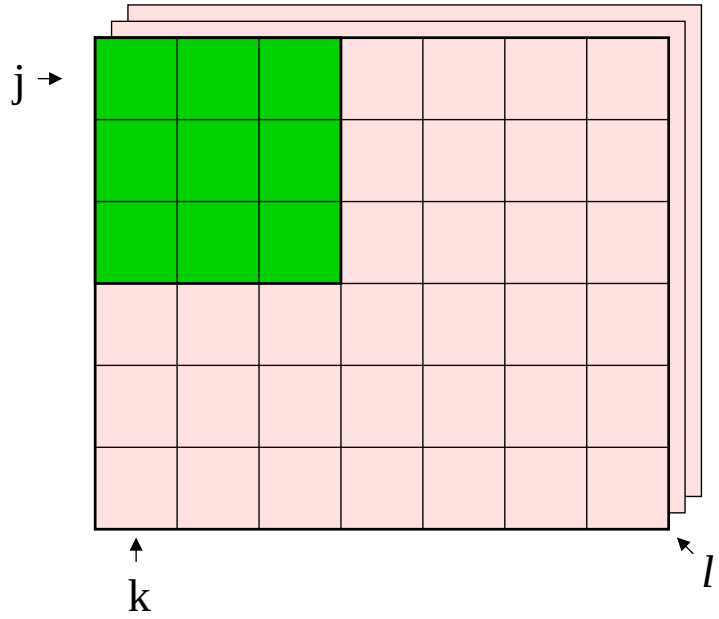
$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$$

Two-dimensional convolution

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

Note: Theoreticians sometimes write $I(j-m,k-n)$ so that the operator is commutative. But, computationally, it is easier to write it with a plus sign.

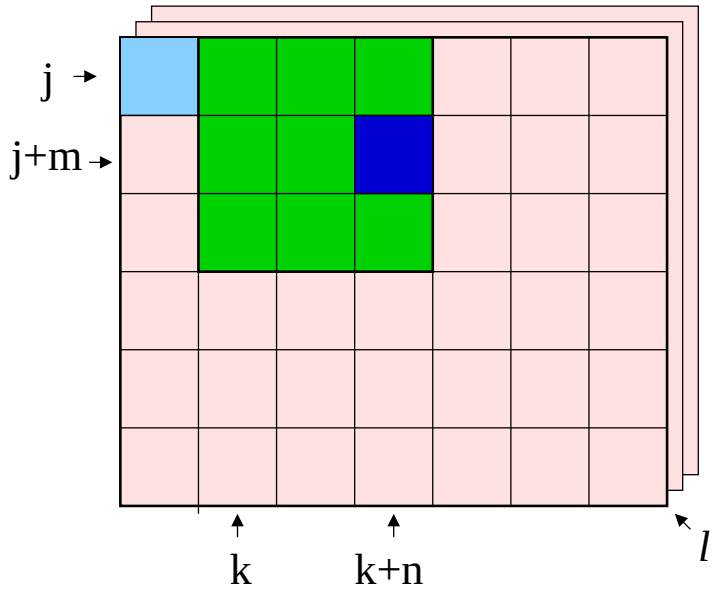
Convolutional Neural Networks



Assume the original image is $J \times K$, with L channels.

We apply an $M \times N$ "filter" to these inputs to compute one hidden unit in the convolution layer. In this example $J = 6, K = 7, L = 3, M = 3, N = 3$.

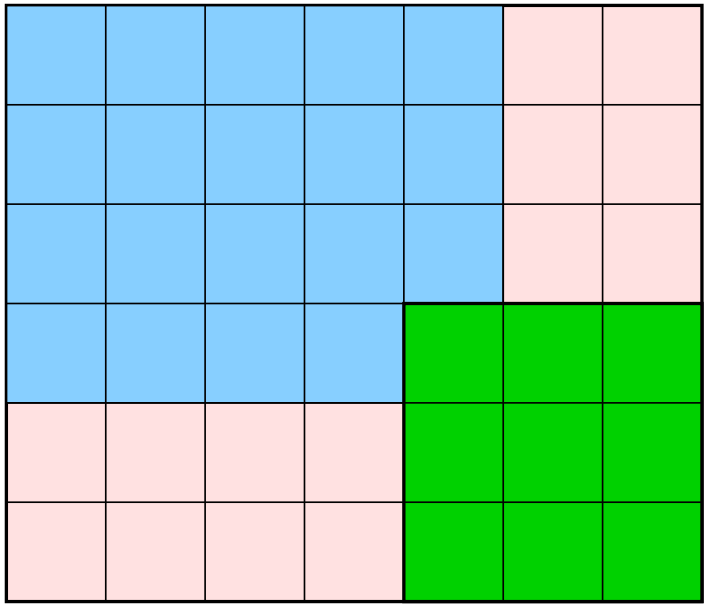
Convolutional Neural Networks



$$Z^i_{j,k} = g\left(b^i + \sum_l \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} K^i_{l,m,n} V^l_{j+m,k+n}\right)$$

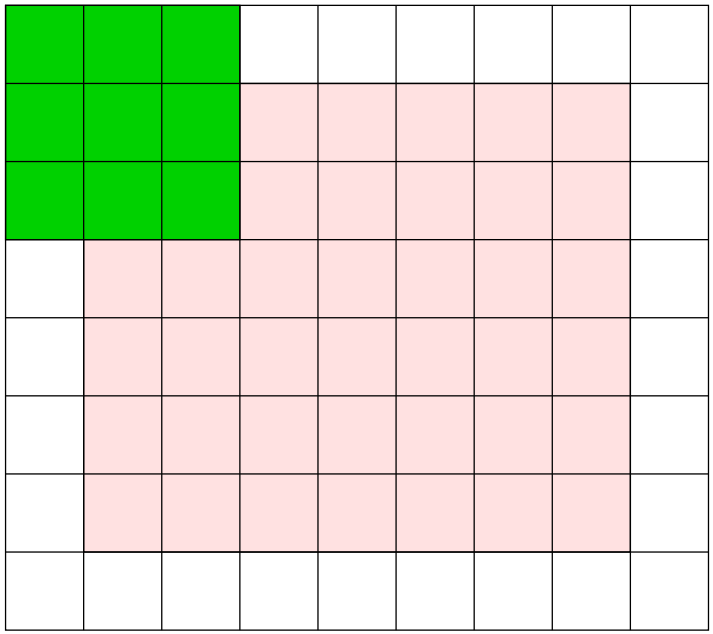
The same weights are applied to the next $M \times N$ block of inputs, to compute the next hidden unit in the convolution layer ("weight sharing").

Convolutional Neural Networks



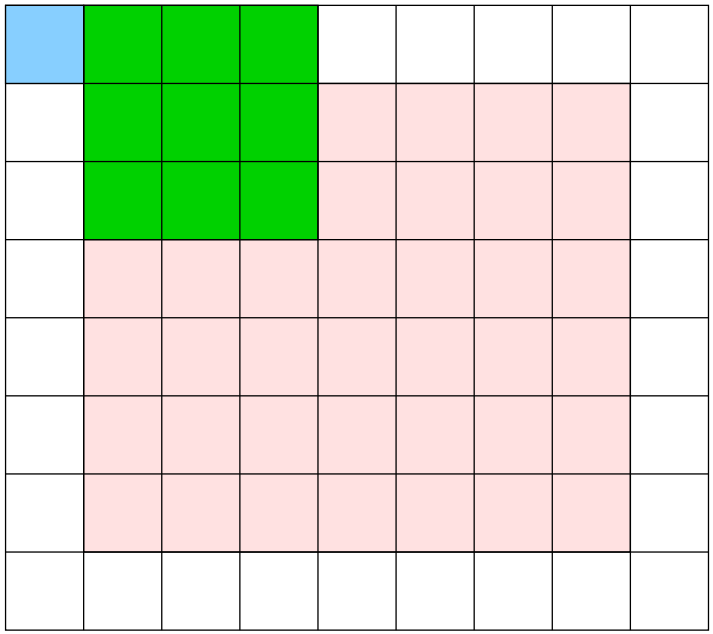
If the original image size is $J \times K$ and the filter is size $M \times N$, the convolution layer will be $(J+1-M) \times (K+1-N)$

Convolution with Zero Padding



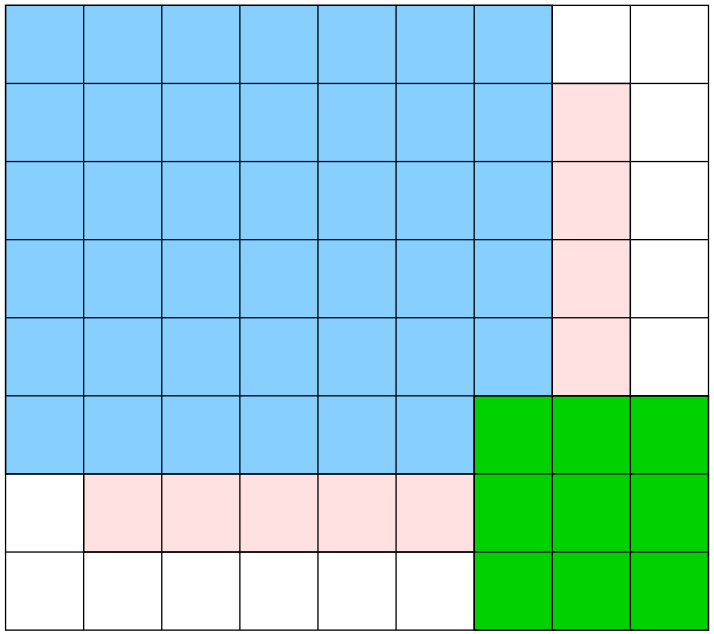
Sometimes, we treat the off-edge inputs as zero (or some other value).

Convolution with Zero Padding



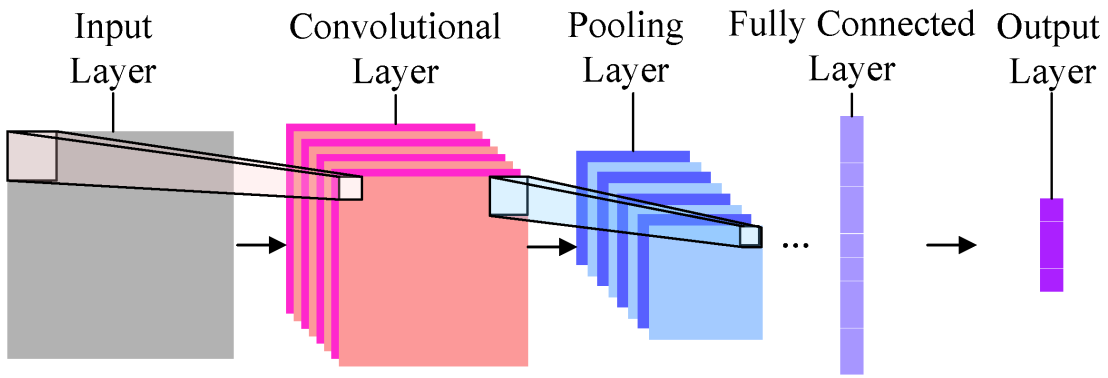
This is known as “Zero-Padding”.

Convolution with Zero Padding



With Zero Padding, the convolution layer is the same size as the original image (or the previous layer).

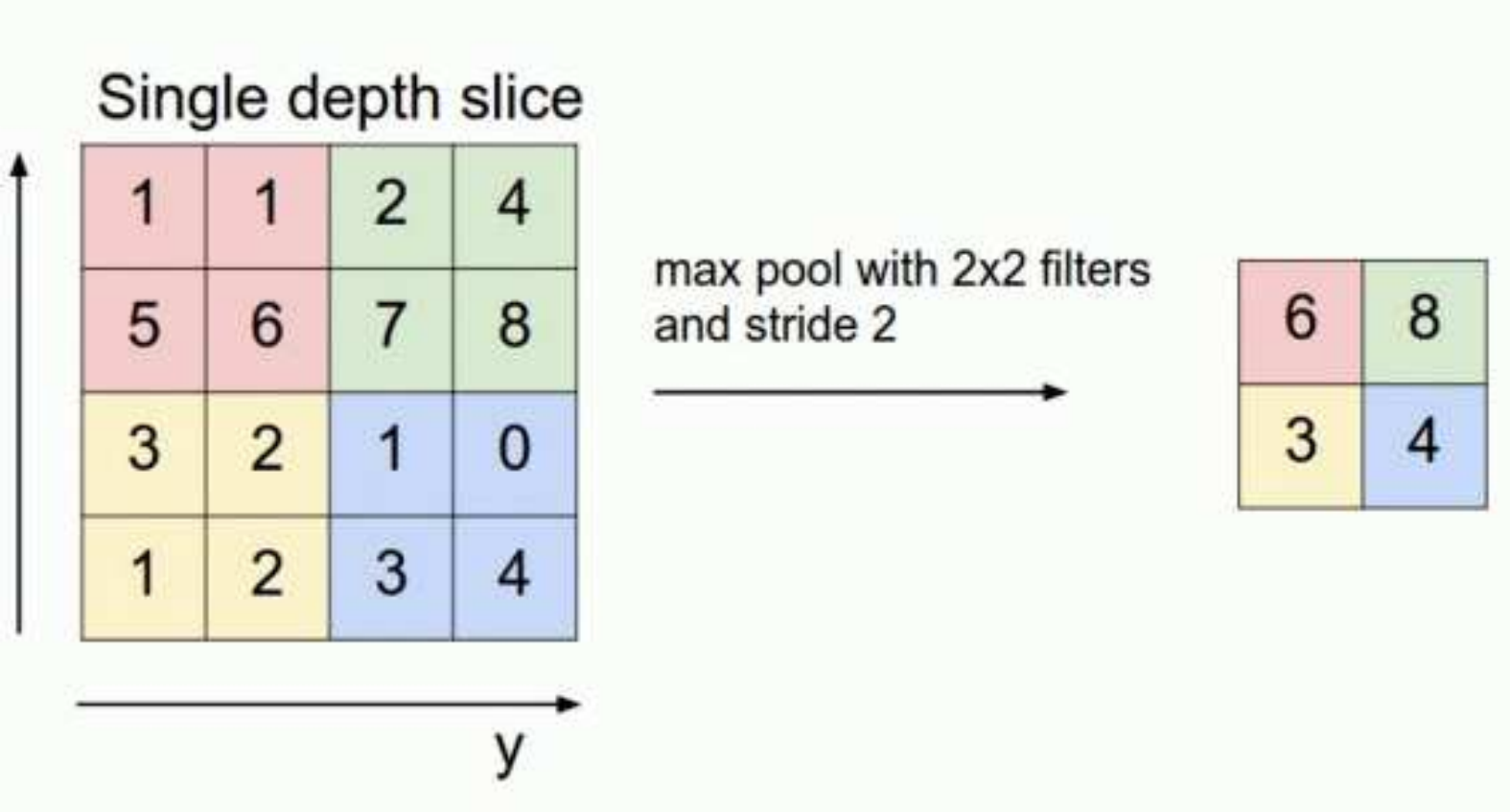
Pooling Layers



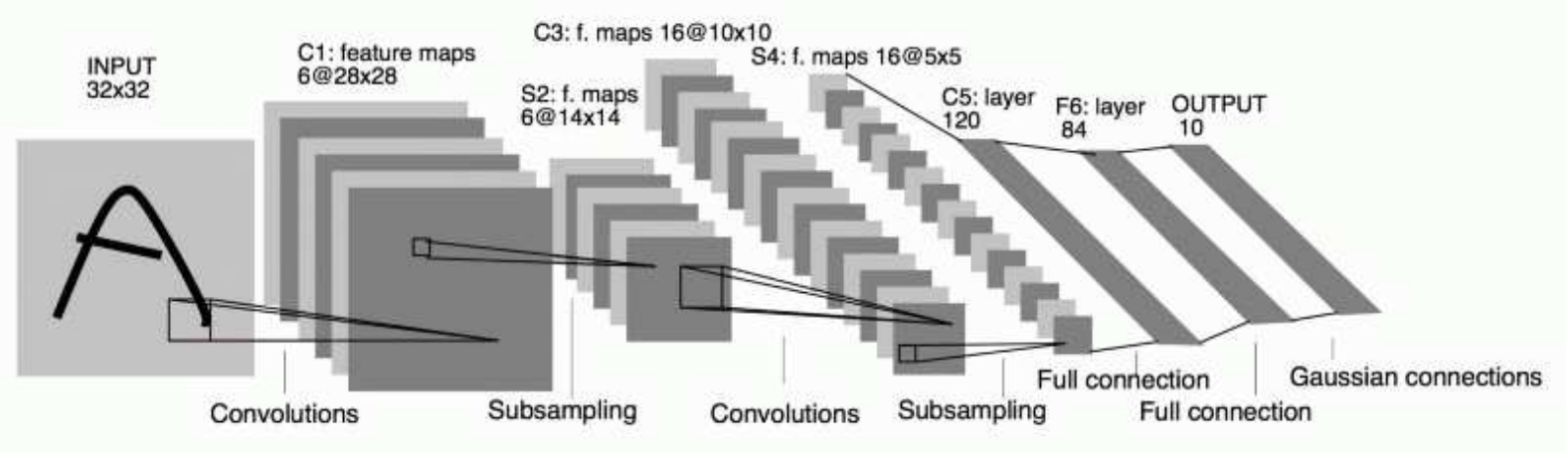
Pooling layers compute either the average, median or, more commonly, the maximum of the values in a neighborhood.

If the convolution layer detects a particular feature, then the pooling layer will indicate whether that feature is present in a slightly larger neigborhood.

Max Pooling

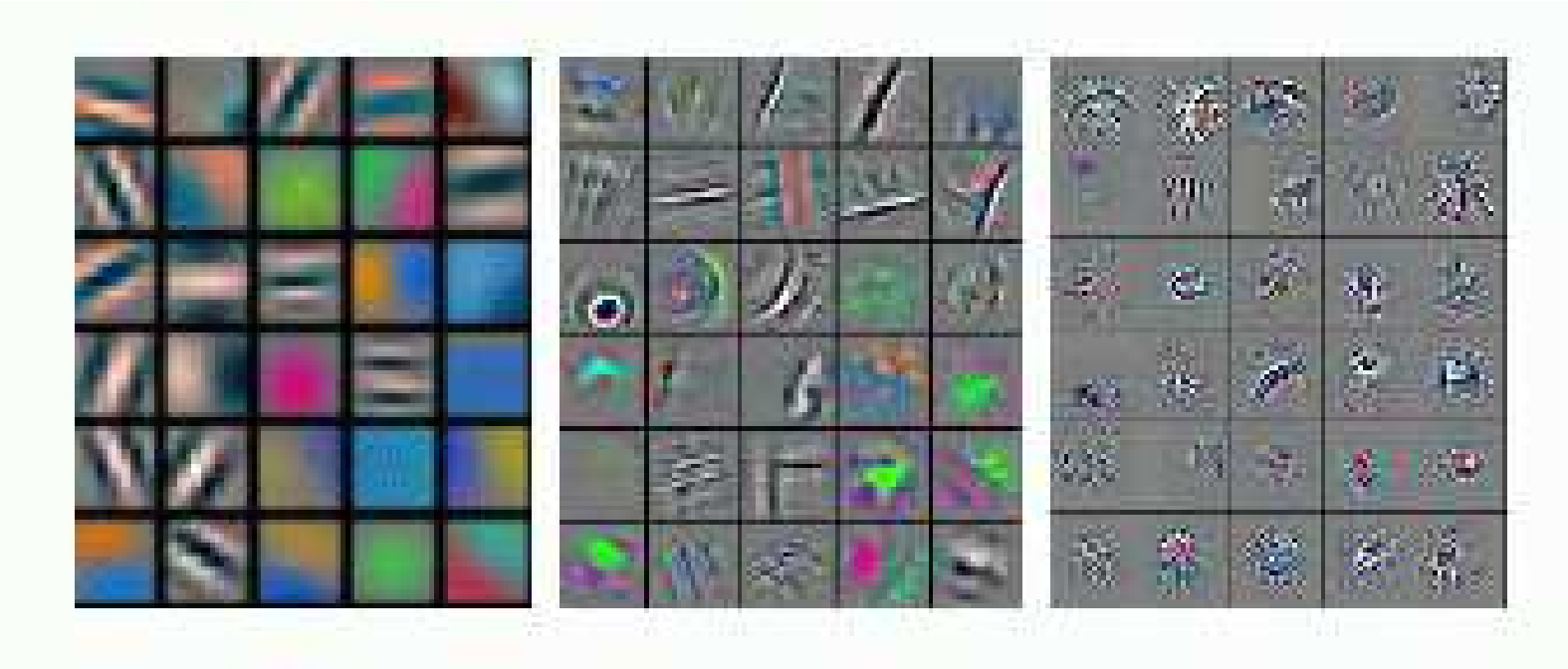


Example: LeNet trained on MNIST



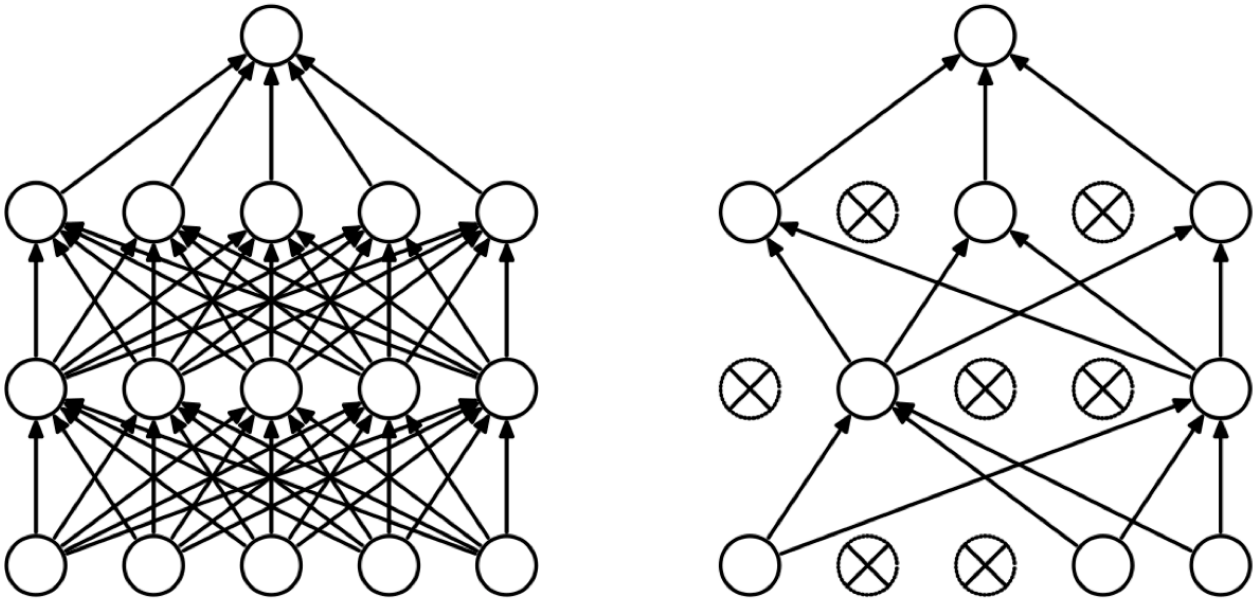
The 5×5 window of the first convolution layer extracts from the original 32×32 image a 28×28 array of features. Subsampling then halves this size to 14×14. The second Convolution layer uses another 5×5 window to extract a 10×10 array of features, which the second subsampling layer reduces to 5×5. These activations then pass through two fully connected layers into the 10 output units corresponding to the digits '0' to '9'.

Convolutional Filters



First Layer Second Layer Third Layer

Dropout (7.12)



(a) Standard Neural Net (b) After applying dropout.

Nodes are randomly chosen to not be used, with some fixed probability (usually, one half).

Dropout (7.12)

When training is finished and the network is deployed, all nodes are used, but their activations are multiplied by the same probability that was used in the dropout.

Thus, the activation received by each unit is the average value of what it would have received during training.

Dropout forces the network to achieve redundancy because it must deal with situations where some features are missing.

Another way to view dropout is that it implicitly (and efficiently) simulates an ensemble of different architectures.

Ensembling

Ensembling is a method where a number of different classifiers are trained on the same task, and the final class is decided by “voting” among them.

In order to benefit from ensembling, we need to have diversity in the different classifiers.

For example, we could train three neural networks with different architectures, three Support Vector Machines with different dimensions and kernels, as well as two other classifiers, and ensemble all of them to produce a final result.
(Kaggle Competition entries are often done in this way).

Bagging

Diversity can also be achieved by training on different subsets of data.

Suppose we are given N training items.

Each time we train a new classifier, we choose N items from the training set with replacement. This means that some items will not be chosen, while others are chosen two or three times.

There will be diversity among the resulting classifiers because they have each been trained on a different subset of data. They can be ensembled to produce a more accurate result than a single classifier.

Dropout as an Implicit Ensemble

In the case of dropout, the same data are used each time but a different architecture is created by removing the nodes that are dropped.

The trick of multiplying the output of each node by the probability of dropout implicitly averages the output over all of these different models.