Deep Networks as Feature Extractors

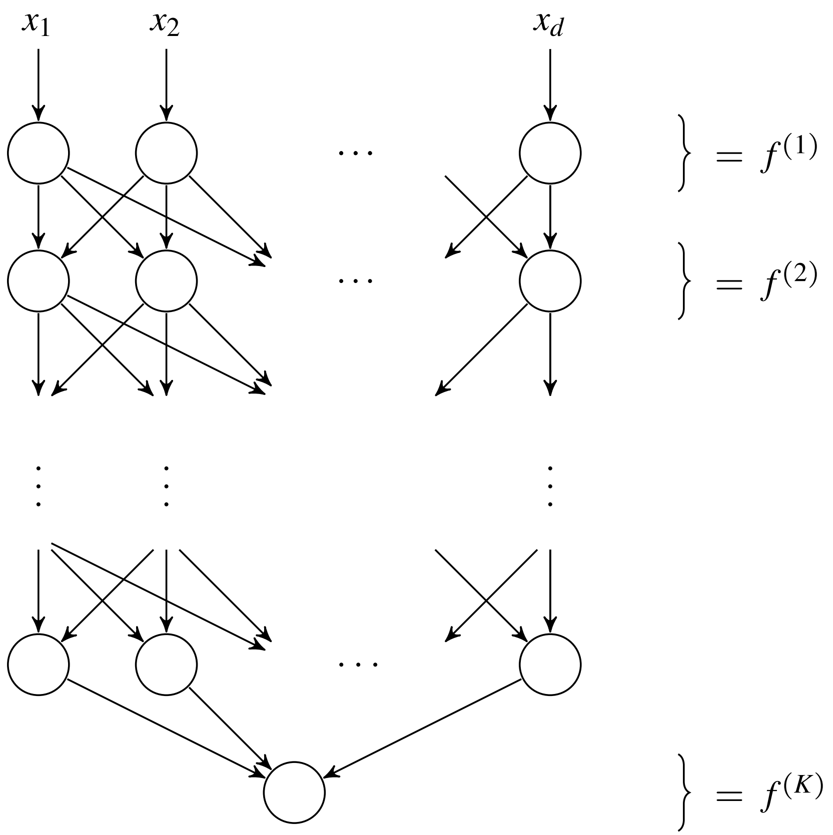
- The network on the right is a classifier $f : \mathbf{R}^d \to \{0, 1\}$.
- Suppose we subdivide the network into the first $K-1$ layer and the final layer, by defining

$$\mathbf{F}(\mathbf{x}) := f^{(K-1)} \circ \ldots \circ f^{(1)}(\mathbf{x})$$

- The entire network is then

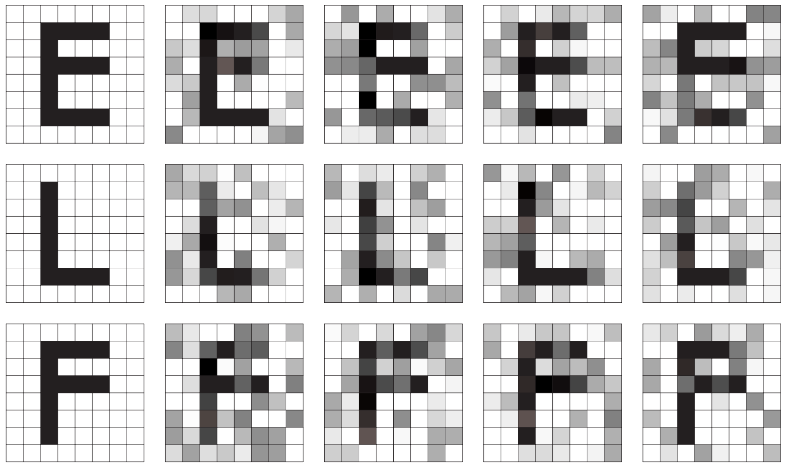
$$f(\mathbf{x}) = f^{(K)} \circ \mathbf{F}(\mathbf{x})$$

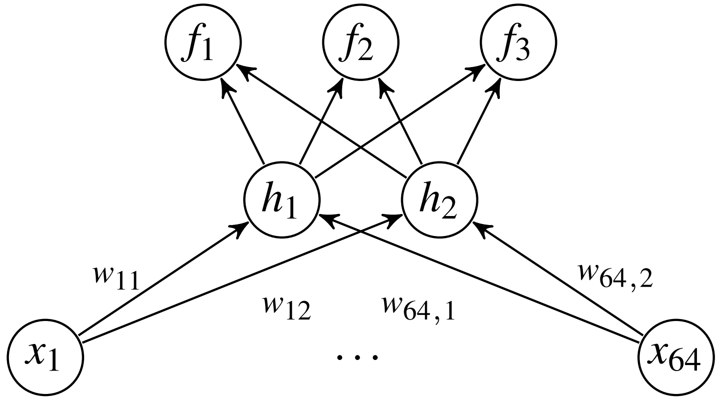
- The function $f^{(K)}$ is a two-class logistic regression classifier.
- We can hence think of f as a feature extraction $\mathbf{F}$ followed by linear classification $f^{(K)}$.



$$f^{(K)}(\,\bullet\,) = \sigma(\langle \mathbf{w}^{(K)}, \bullet \rangle)$$

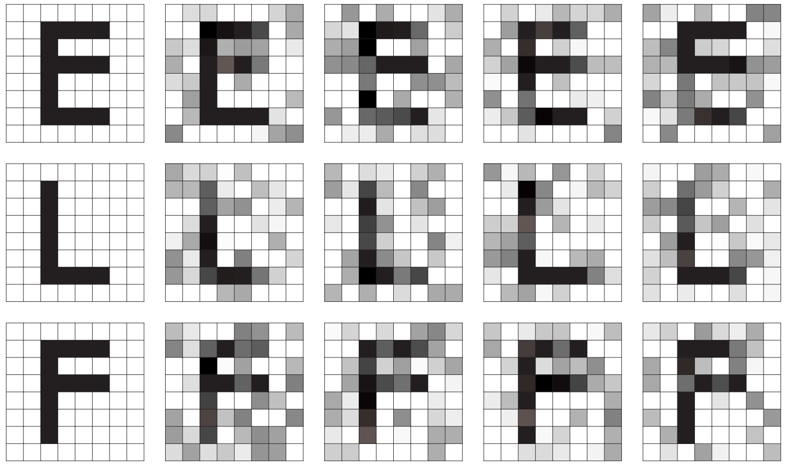
A SIMPLE EXAMPLE



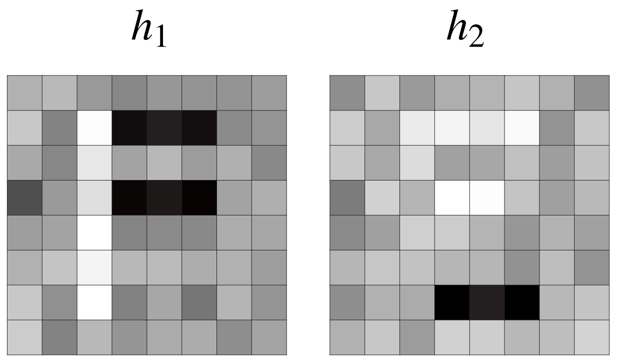


- Problem: Classify characters into three classes (E, F and L).
- Each digit given as a $8 \times 8 = 64$ pixel image
- Neural network: 64 input units (=pixels)
- 2 hidden units
- 3 binary output units, where $f_i(\mathbf{x}) = 1$ means image is in class i.
- Each hidden unit has 64 input weights, one per pixel. The weight values can be plottes as 8×8 images.

A Simple Example



training data (with random noise)



weight values of h_1 and h_2 plotted as images

- Dark regions = large weight values.
- Note the weights emphasize regions that distinguish characters.
- We can think of weight (= each pixel) as a feature.
- The features with large weights for h_1 distinguish $\{$E,F$\}$ from L.
- The features for h_2 distinguish $\{$E,L$\}$ from F.

Illustration: R.O. Duda, P.E. Hart, D.G. Stork, *Pattern Classification*, Wiley 2001

EXAMPLE: AUTOENCODERS

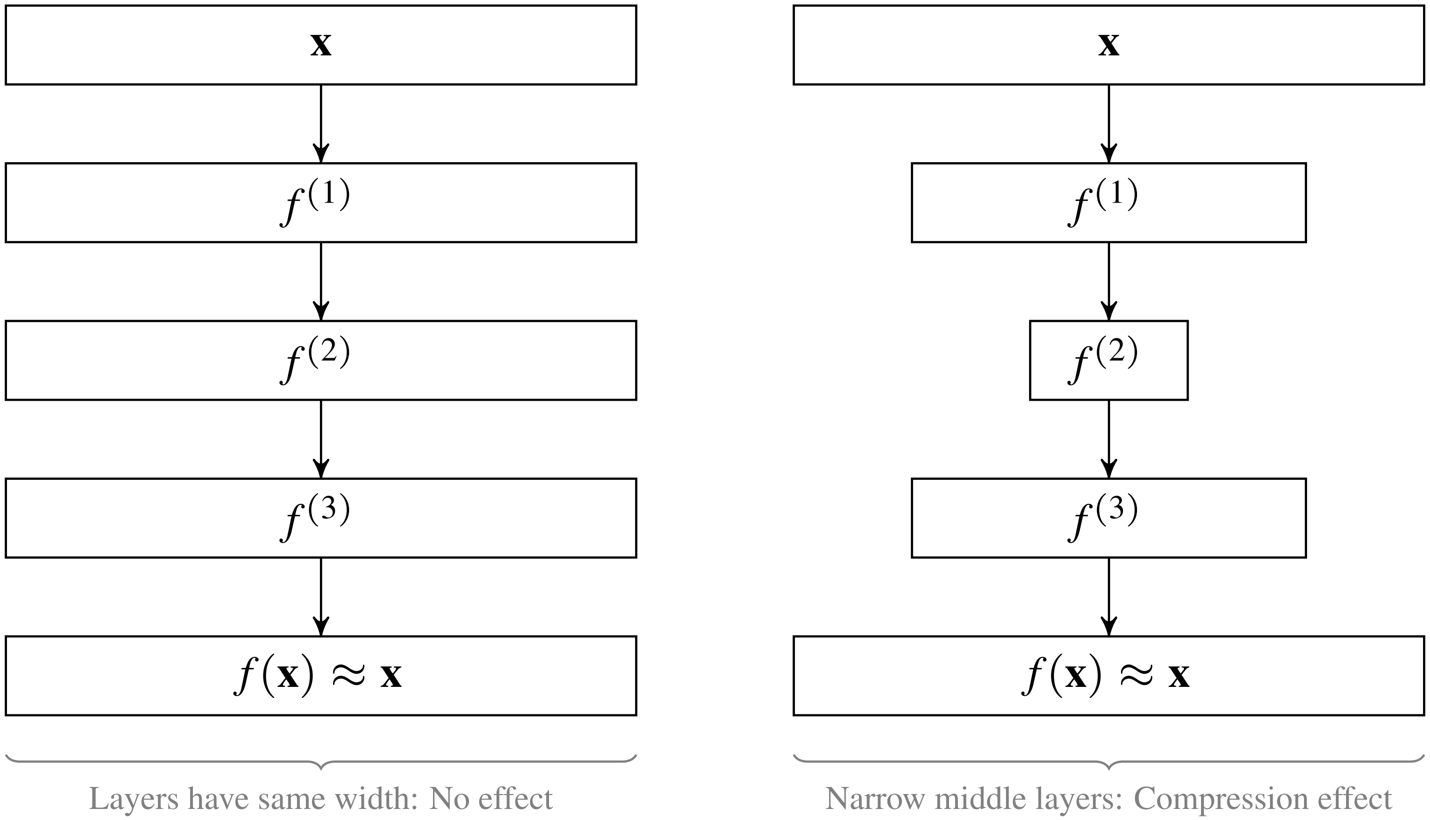
An example for the effect of layer are autoencoders.

- An **autoencoder** is a neural network that is trained on its own input: If the network has weights $\mathbf{W}$ and represents a function $f_{\mathbf{W}}$, training solves the optimization problem

$$\min_{\mathbf{W}} \|\mathbf{x} - f_{\mathbf{W}}(\mathbf{x})\|^2$$

  or something similar for a different norm.
- That seems pointless at first glance: The network tries to approximate the identity function using its (possibly nonlinear) component functions.
- However: If the layers in the middle have much fewer nodes that those at the top and bottom, the network learns to *compress the input*.

AUTOENCODERS


$\mathbf{x}$
$f^{(1)}$
$f^{(2)}$
$f^{(3)}$
$f(\mathbf{x}) \approx \mathbf{x}$
Layers have same width: No effect
$\mathbf{x}$
$f^{(1)}$
$f^{(2)}$
$f^{(3)}$
$f(\mathbf{x}) \approx \mathbf{x}$
Narrow middle layers: Compression effect

- Train network on many images.
- Once trained: Input an image $\mathbf{x}$.
- Store $\mathbf{x}' := f^{(2)}(\mathbf{x})$. Note $\mathbf{x}'$ has fewer dimensions than $\mathbf{x} \rightarrow$ compression.
- To decompress $\mathbf{x}'$: Input it into $f^{(3)}$ and apply the remaining layers of the network $\rightarrow$ reconstruction $f(\mathbf{x}) \approx \mathbf{x}$ of $\mathbf{x}$.

AUTOENCODERS

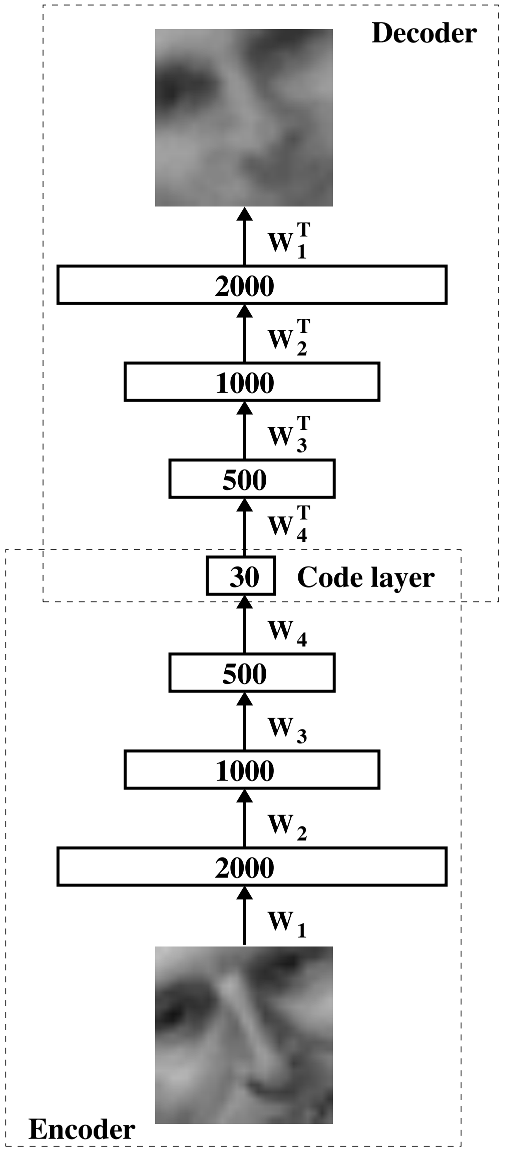


Illustration: K. Murphy, *Machine Learning: A Bayesian perspective*, MIT Press 2012

Cross-correlation Operations

Definition

Suppose we define a small (here: 3×3) matrix

$$K = \begin{pmatrix} k_{-1,-1} & k_{-1,0} & k_{-1,1} \\ k_{0,-1} & k_{0,0} & k_{0,1} \\ k_{1,-1} & k_{1,0} & k_{1,1} \end{pmatrix}$$

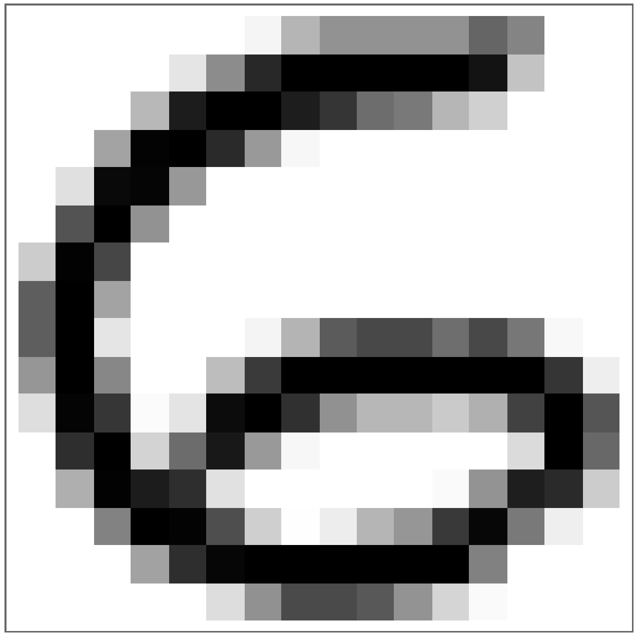
For a large matrix A, we define the **cross-correlation** of A and K as the matrix $A \odot K$ with entries

$$(A \odot K)_{ij} := a_{ij}k_{0,0} + a_{i-1,j-1}k_{-1,-1} + \ldots = \sum_{m,n=-1}^{1} a_{i+m,j+n}k_{m,n}$$

Remarks

- K is sometimes called a **kernel**. Caution: The term kernel is used for several, different concepts in both mathematics and machine learning.
- We can similarly define the cross-correlation if K is of size 5×5 etc. The numbers of rows and columns should be odd, so that k_{00} is at the center of K.

CROSS-CORRELATION FOR IMAGES

$A =$  $=$

$$\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 10 & 74 & 109 & 109 & 109 & 109 & 154 & 123 & 0 & 0 \\
0 & 0 & 0 & 0 & 26 & 115 & 215 & 255 & 255 & 255 & 255 & 255 & 236 & 60 & 0 & 0 \\
0 & 0 & 0 & 71 & 227 & 255 & 255 & 226 & 202 & 146 & 134 & 73 & 47 & 0 & 0 & 0 \\
0 & 0 & 92 & 252 & 255 & 213 & 102 & 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 31 & 246 & 250 & 103 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 172 & 255 & 109 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
51 & 253 & 185 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
161 & 255 & 92 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
161 & 255 & 26 & 0 & 0 & 0 & 11 & 75 & 164 & 183 & 183 & 145 & 183 & 136 & 7 & 0 \\
105 & 255 & 120 & 0 & 0 & 66 & 197 & 255 & 255 & 255 & 255 & 255 & 255 & 255 & 202 & 17 \\
33 & 251 & 201 & 4 & 27 & 243 & 255 & 207 & 110 & 72 & 72 & 53 & 79 & 190 & 255 & 170 \\
0 & 209 & 255 & 44 & 147 & 231 & 102 & 8 & 0 & 0 & 0 & 0 & 0 & 36 & 255 & 151 \\
0 & 80 & 253 & 227 & 209 & 30 & 0 & 0 & 0 & 0 & 0 & 5 & 108 & 225 & 213 & 51 \\
0 & 0 & 125 & 255 & 252 & 177 & 48 & 1 & 18 & 74 & 105 & 198 & 248 & 134 & 16 & 0 \\
0 & 0 & 0 & 93 & 209 & 249 & 255 & 255 & 255 & 255 & 255 & 255 & 126 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 34 & 110 & 181 & 181 & 167 & 108 & 42 & 5 & 0 & 0 & 0
\end{pmatrix}$$

- Recall that we can represent a grayscale image as a matrix A.
- We can then define a kernel matrix K and compute the cross-correlation $A \odot K$.

Effect of cross-correlation on images

- Consider again a 3×3 kernel

$$K = \begin{pmatrix} k_{-1,-1} & k_{-1,0} & k_{-1,1} \\ k_{0,-1} & k_{0,0} & k_{0,1} \\ k_{1,-1} & k_{1,0} & k_{1,1} \end{pmatrix} \quad \text{with} \quad (A \odot K)_{ij} = \sum_{m,n=-1}^{1} a_{i+m,\,j+n} k_{m,n}$$

- Consider the pixel value a_{ij} at location i, j in A. In the new image $A \odot K$, a_{ij} is the sum of element-wise producs of K and the direct neighborhood of a_{ij}:

$$(A \odot K)_{ij} = \text{ sum of entries of } \begin{pmatrix} k_{-1,-1}a_{i-1,j-1} & k_{-1,0}a_{i-1,j} & k_{-1,1}a_{i-1,j+1} \\ k_{0,-1}a_{i,j-1} & k_{0,0}a_{ij} & k_{0,1}a_{i,j+1} \\ k_{1,-1}a_{i+1,j-1} & k_{1,0}a_{i+1,j} & k_{1,1}a_{i+1,j+1} \end{pmatrix}$$

- In other words, $(A \odot K)_{ij}$ is a weighted average of a_{ij} and its neighbors.
- The next few slides illustrate the effect of different choices of K.

EXAMPLES

For the identity kernel, nothing happens:

$A =$

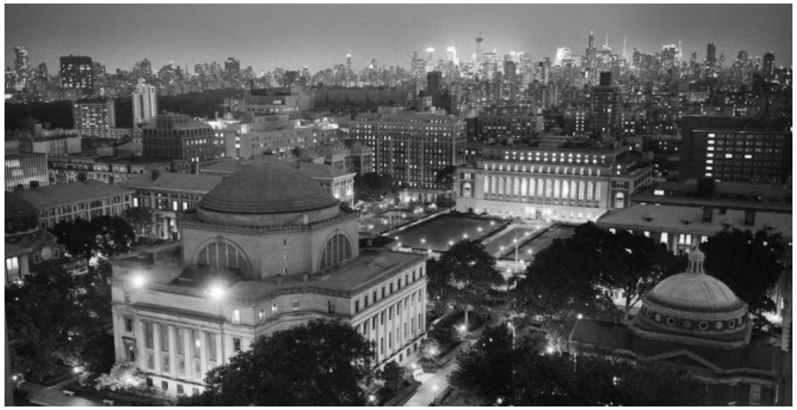
$$K = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$A \odot K =$

Examples

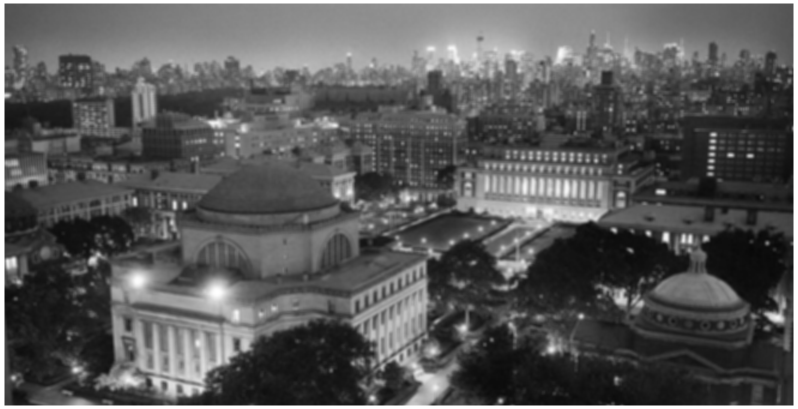
If all entries of K are identical, each pixel in the image is "averaged together" with its neighbors. That results in blurring:

$A =$



$$K = \begin{pmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{pmatrix}$$

$A \odot K =$



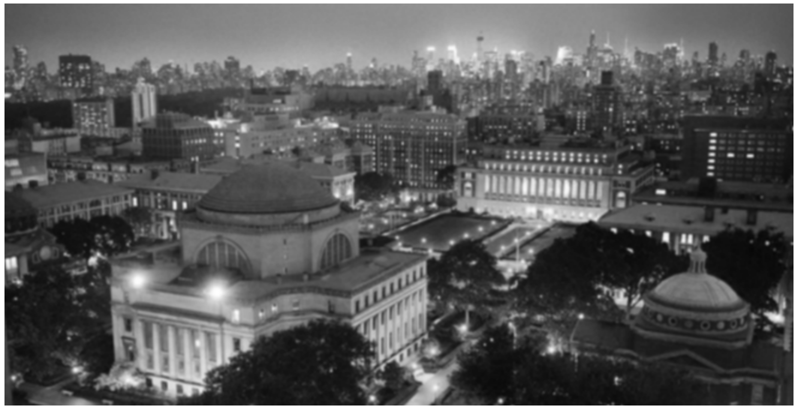
EXAMPLES

Since diagonal neighbors are further away than horizontal/vertical ones, we can give them smaller weights. This is also called a "Gaussian blur":

$A =$

$$K = \frac{1}{16}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$$

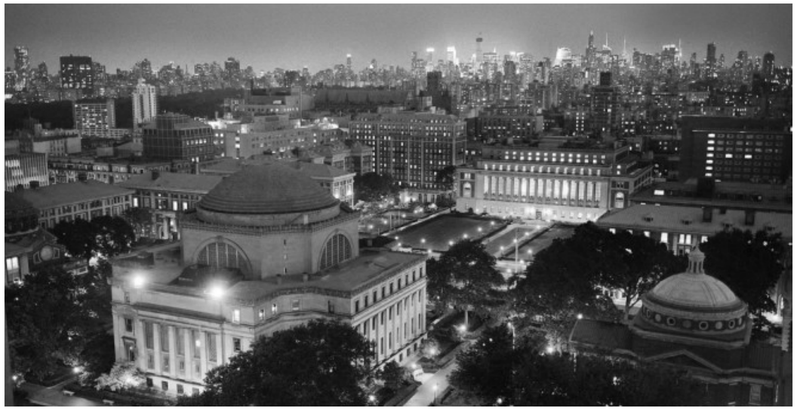
$A \odot K =$



EXAMPLES

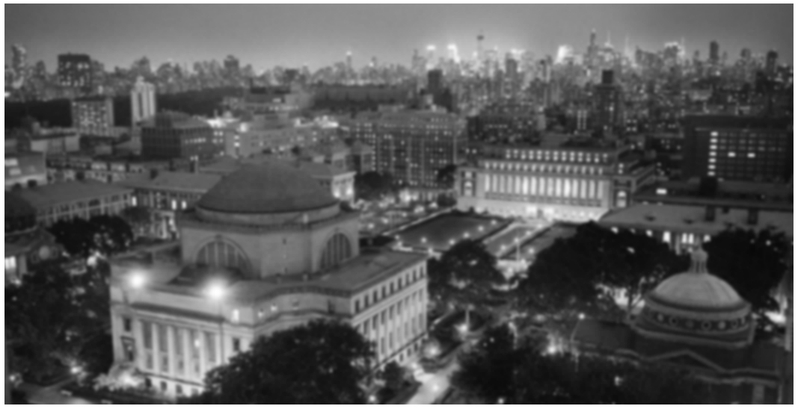
We can increase the size of K, which means we are mixing a_{ij} with more neighbors. Here is a 5×5 Gaussian blur:

$A =$



$$K = \frac{1}{256}\begin{pmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{pmatrix}$$

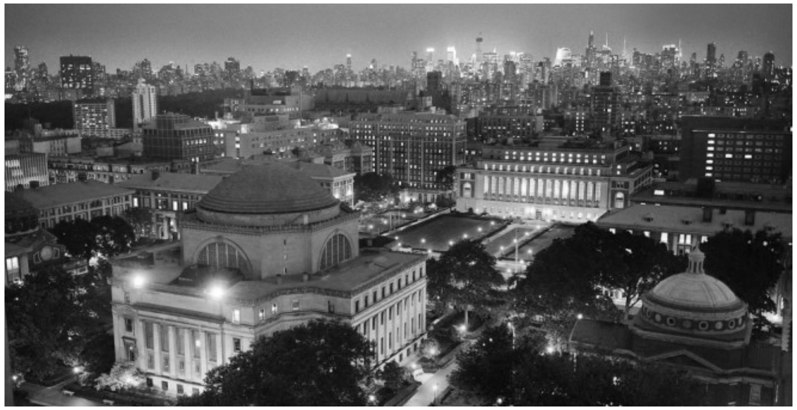
$A \odot K =$



EXAMPLES

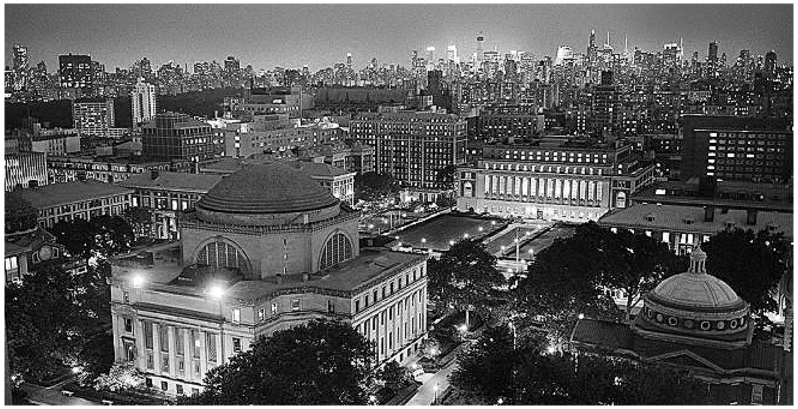
The opposite effect is sharpening: We give the neighbors negative weights. If two adjacent points look different, $A \odot K$ substracts them from each other, so they look even more different:

$A =$



$$K = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

$A \odot K =$

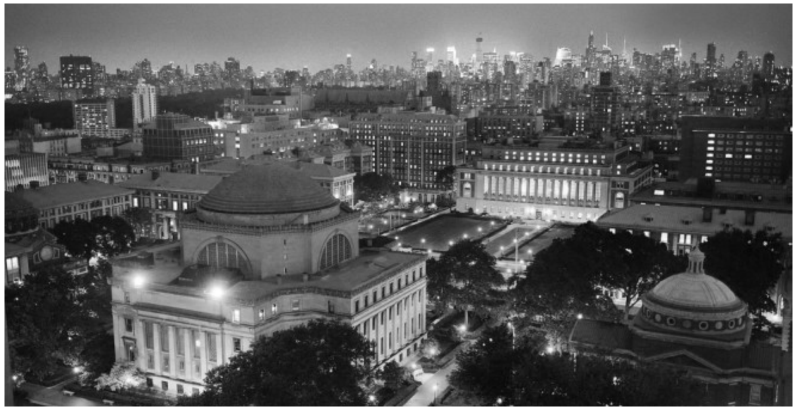


Note the entries of K add up to 1.

Examples

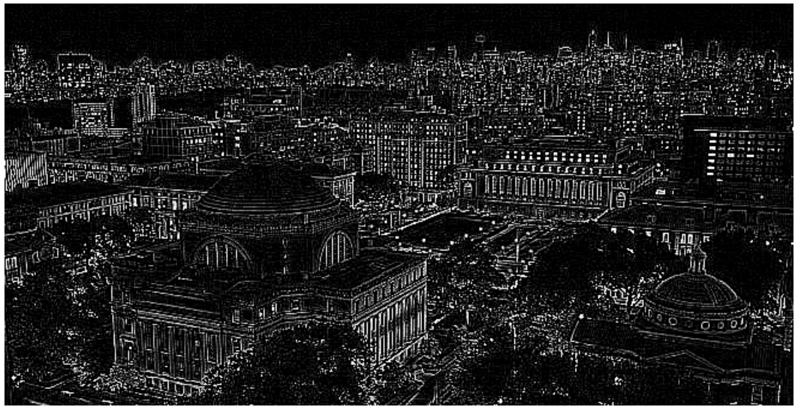
A more drastic form of sharpening is edge detection:

$A =$



$$K = \begin{pmatrix} -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} \\ -\frac{1}{8} & 1 & -\frac{1}{8} \\ -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} \end{pmatrix}$$

$A \odot K =$

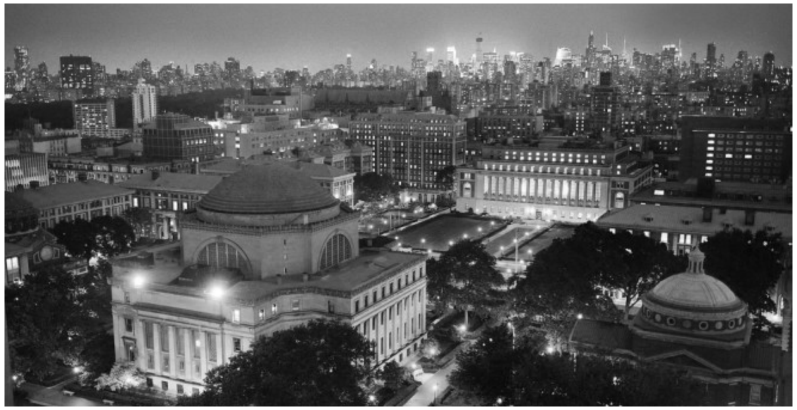


Here, the entries of K add up to 0, so $(A \odot K)_{ij}$ is visible only if a_{ij} is very different from its neighbors.

Examples

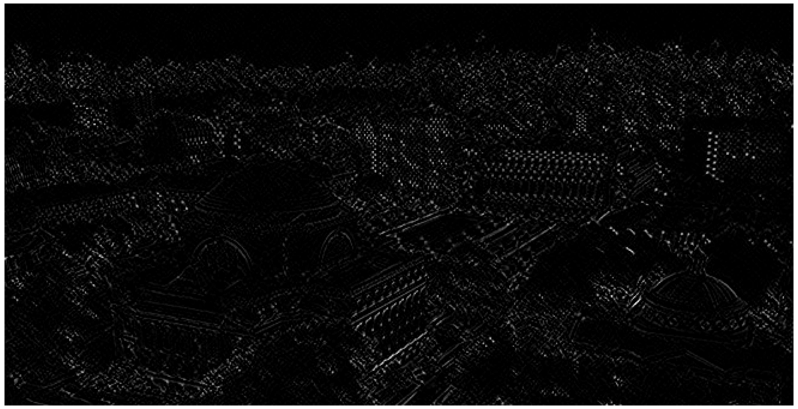
This kernel find points that are similar to their lower left and upper right neighbor, and different from their upper left and lower right one. That means it detects diagonal edges:

$A =$

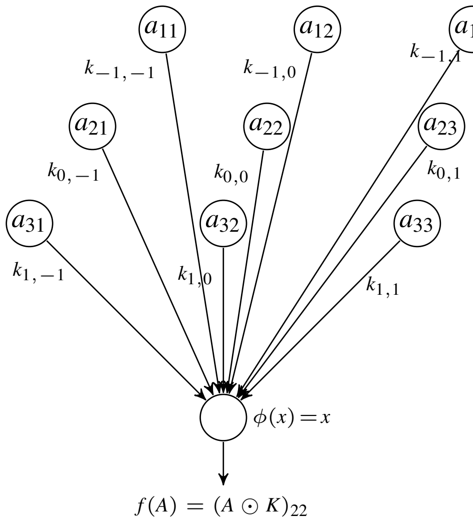


$$K = \begin{pmatrix} -1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & -1 \end{pmatrix}$$

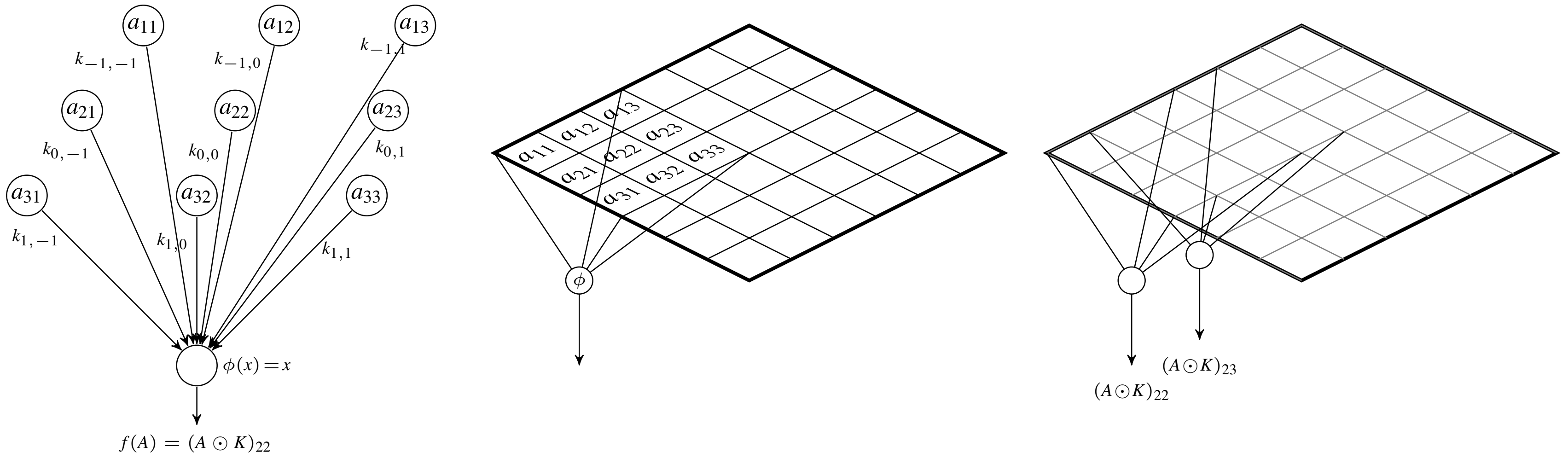
$A \odot K =$



Cross-correlation as a Neural Network



- Suppose we build a neural network one input unit for each entry of $\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$.
- We use the entries of K as weights and connect everything to a single linear unit ("linear unit" means $\phi(x) = x$).
- The network then computes the sum of the weighted inputs, which by definition of $A \odot K$ is just $(A \odot K)_{22}$.
- We can obtain another entry $(A \odot K)_{ij}$ by replacing the input values with another submatrix of A.



(i) (ii) (iii)

- Neural network layers whose units are arranged in a two-dimensional grid are often visualized as “sheets” as in (ii) and (iii).
- The network (i) collects information from a small portion of the input layer, as visualized in (ii).
- We can use a similar network (with different input values but identical weights) to similarly compute $(A \odot K)_{23}$, $(A \odot K)_{24}$, etc as in (iii).
- In that manner, we can compute every entry of $(A \odot K)$ and arrange these entries on another grid of units as the next layer.
- In other words: We attach a network of the form (i) to every 3×3 patch of input values. *All these networks use the same weights, given by the matrix K.* The two-layer network so obtained computes $(A \odot K)$. If we changed the weights to some other matrix K', it would compute $(A \odot K')$.