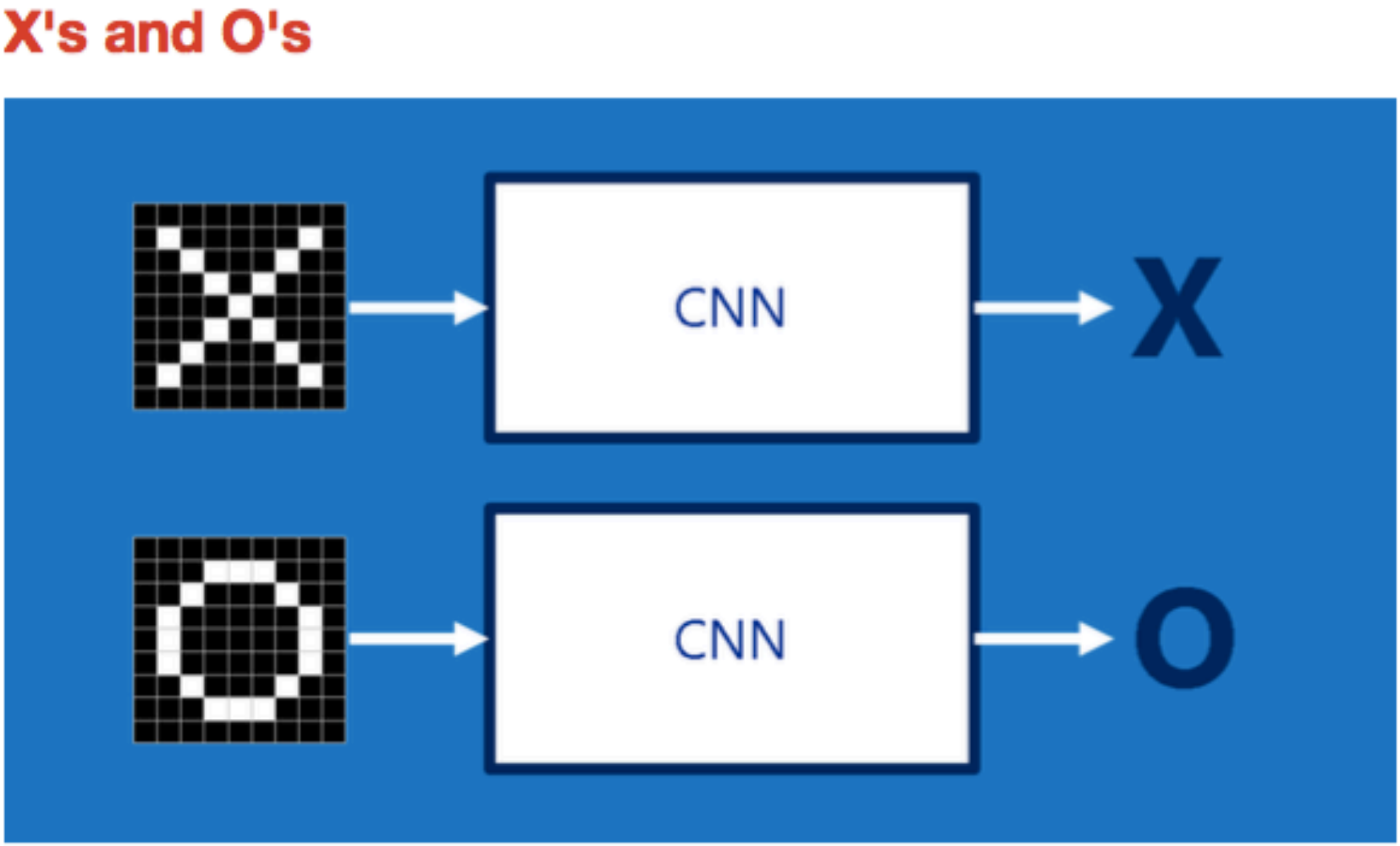
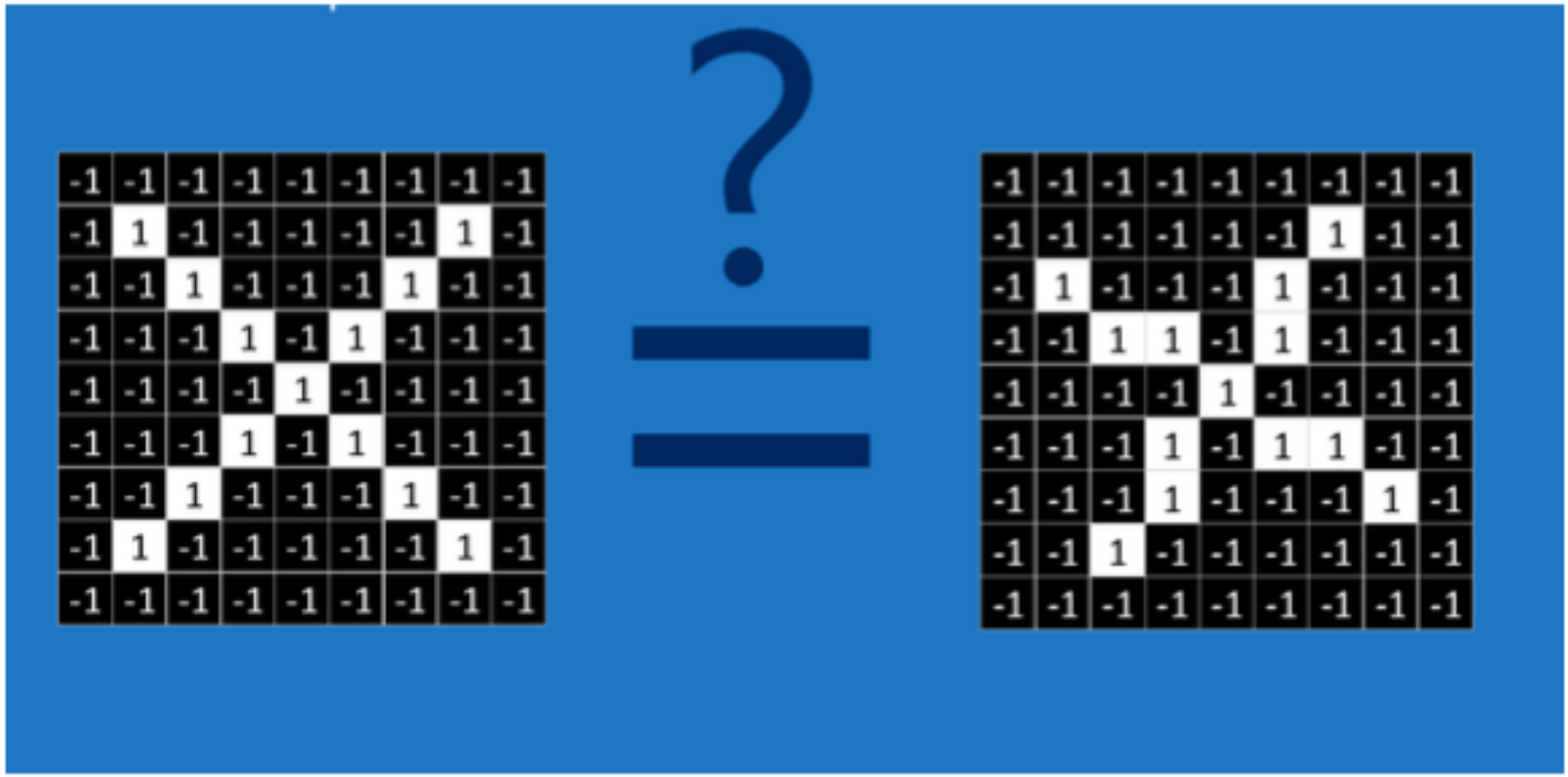
Convolutional Neural Networks – use in Vision and NLP

(Based on
http://brohrer.github.io/how_convolutional_neural_networks_work.html
and the survey paper by Yoav Goldberg)

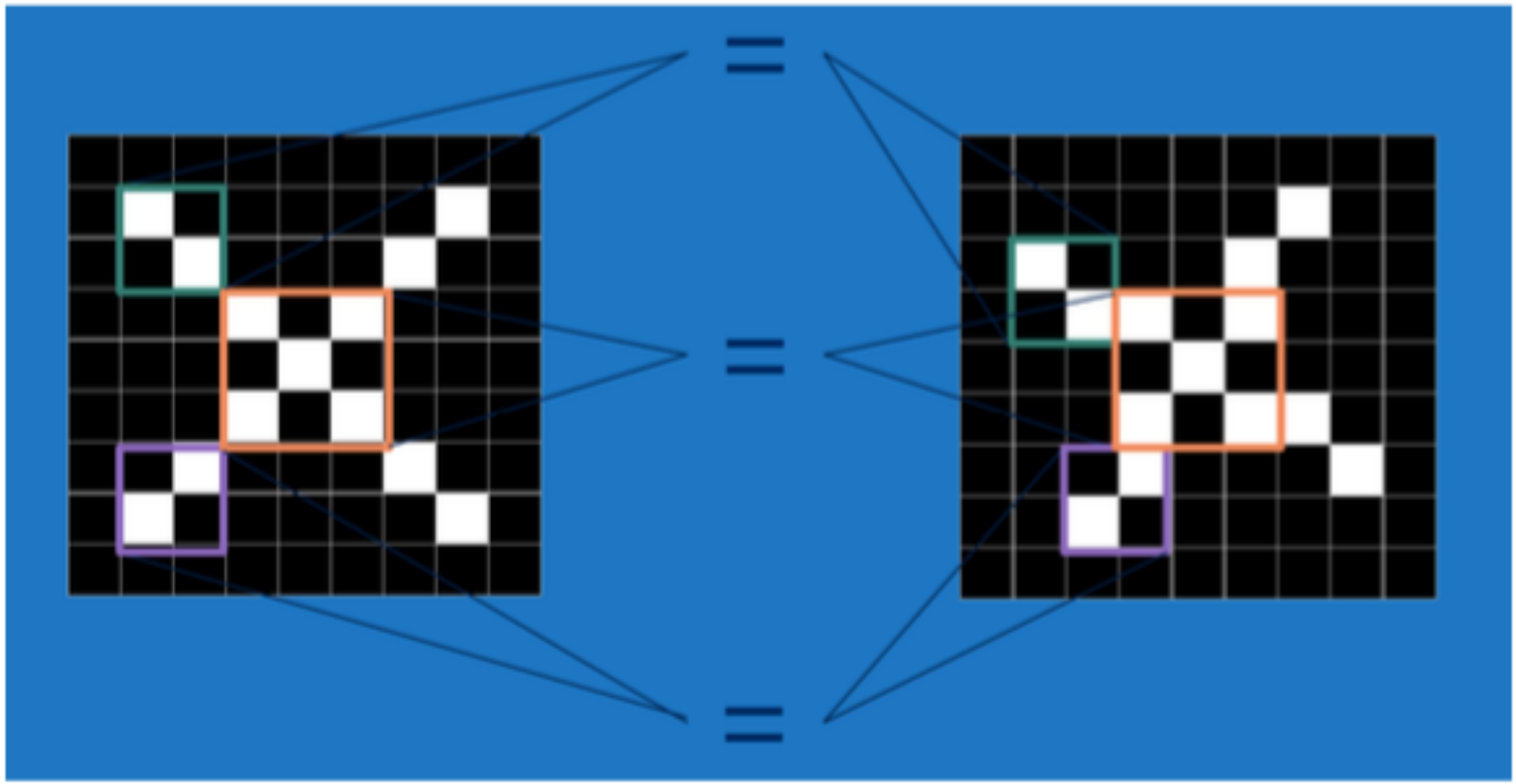
Simple Example:
Whether an image is an X or an O?



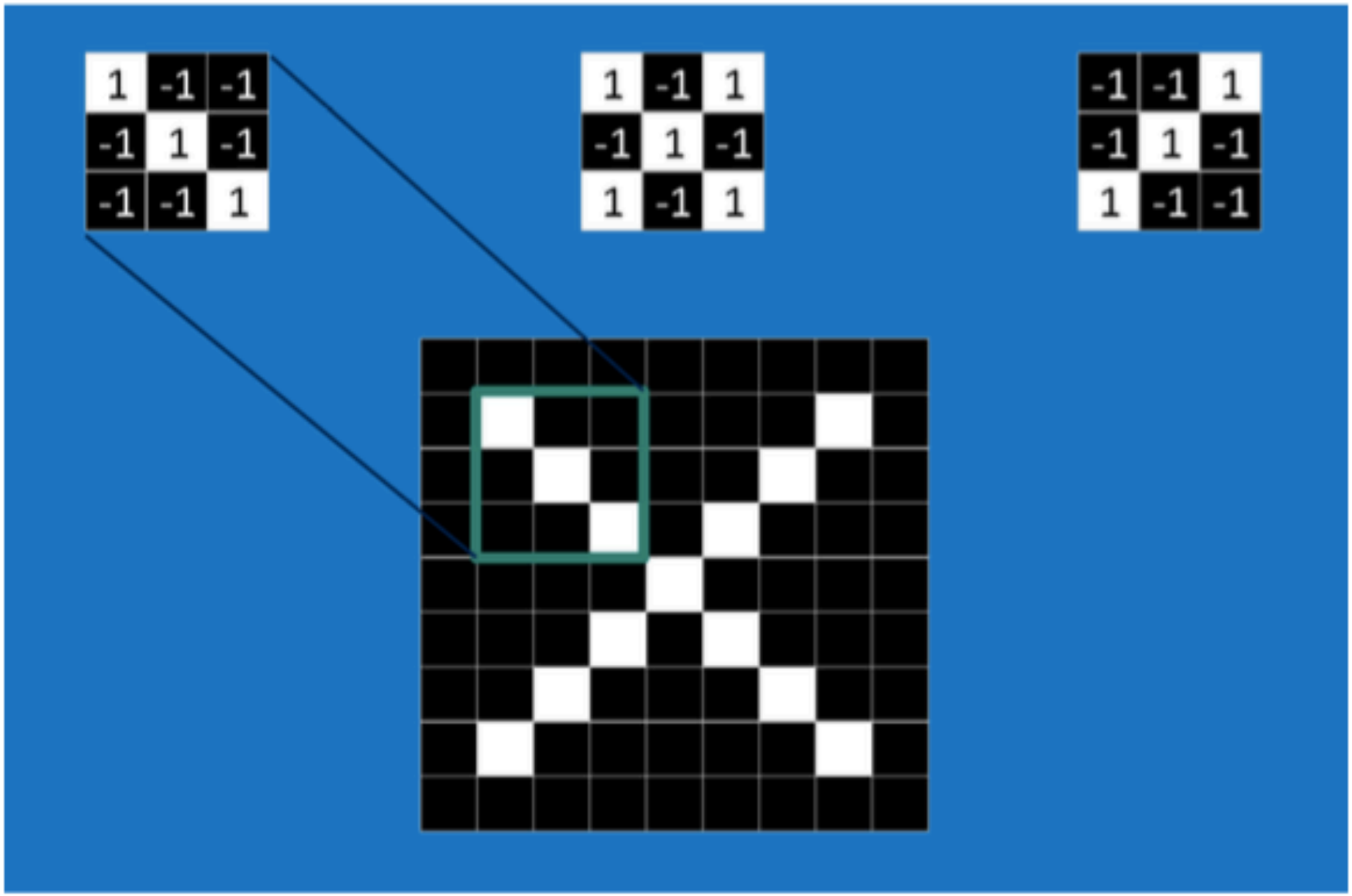
Challenge: We would like to be able to see X's and O's even if they're shifted, shrunken, slightly rotated or deformed.



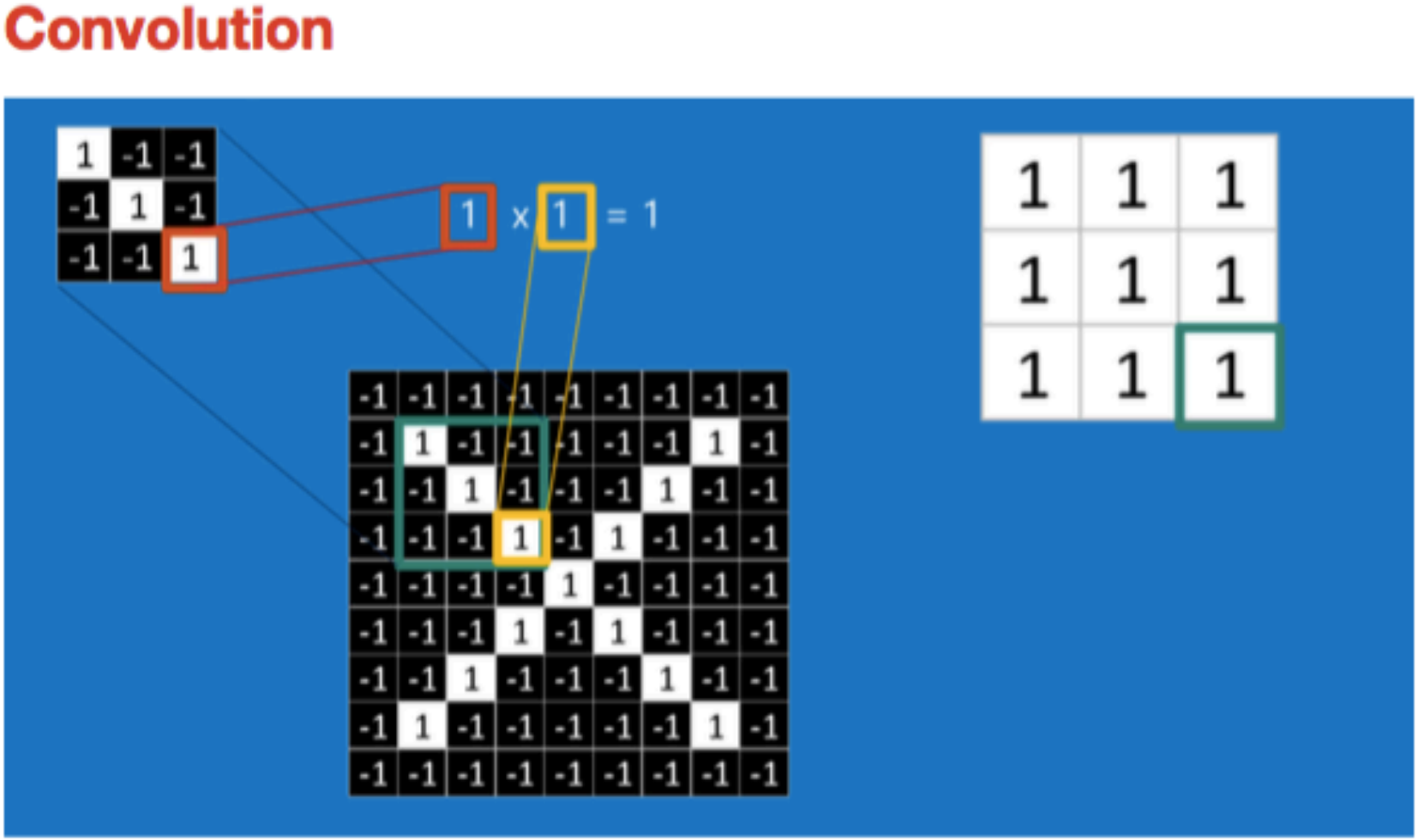
CNNs compare images piece by piece. The pieces that it looks for are called features.



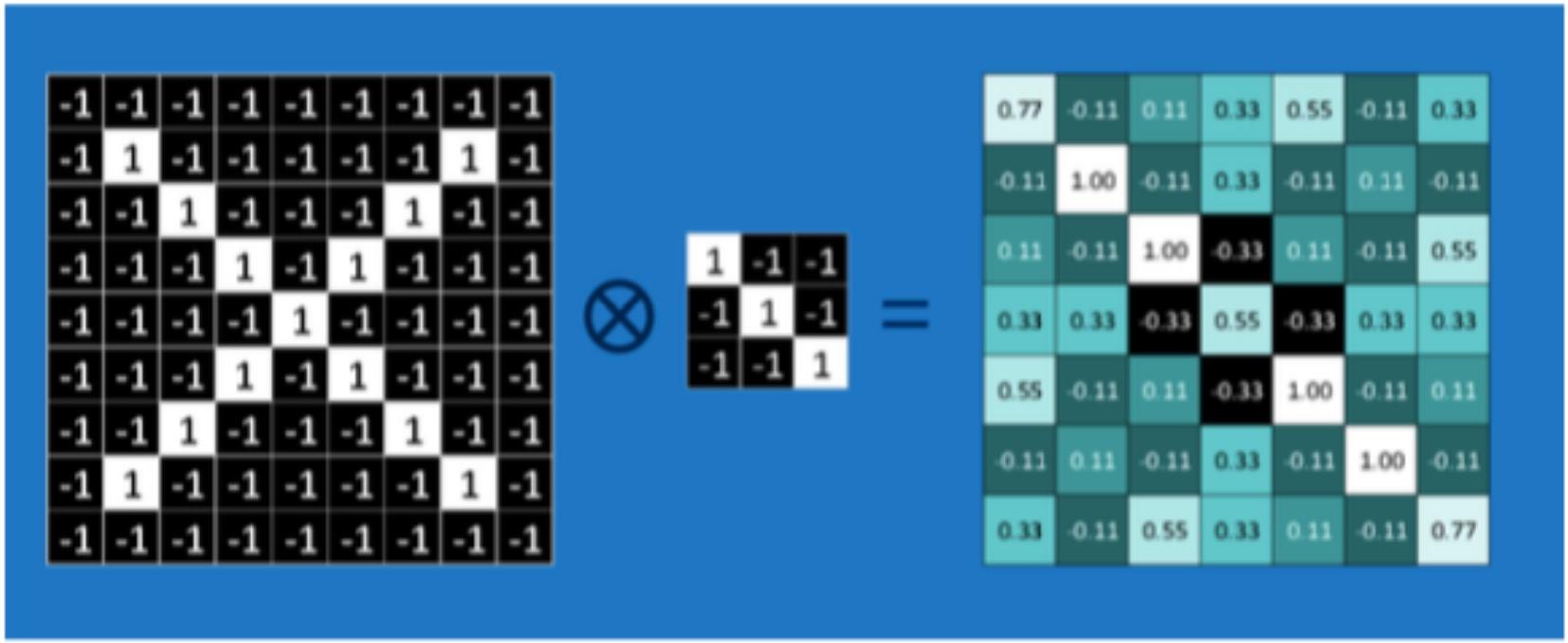
Each feature is like a mini-image—a small two-dimensional array of values. Features match common aspects of the images. In the case of X images, features consisting of diagonal lines and a crossing capture all the important characteristics of most X's.



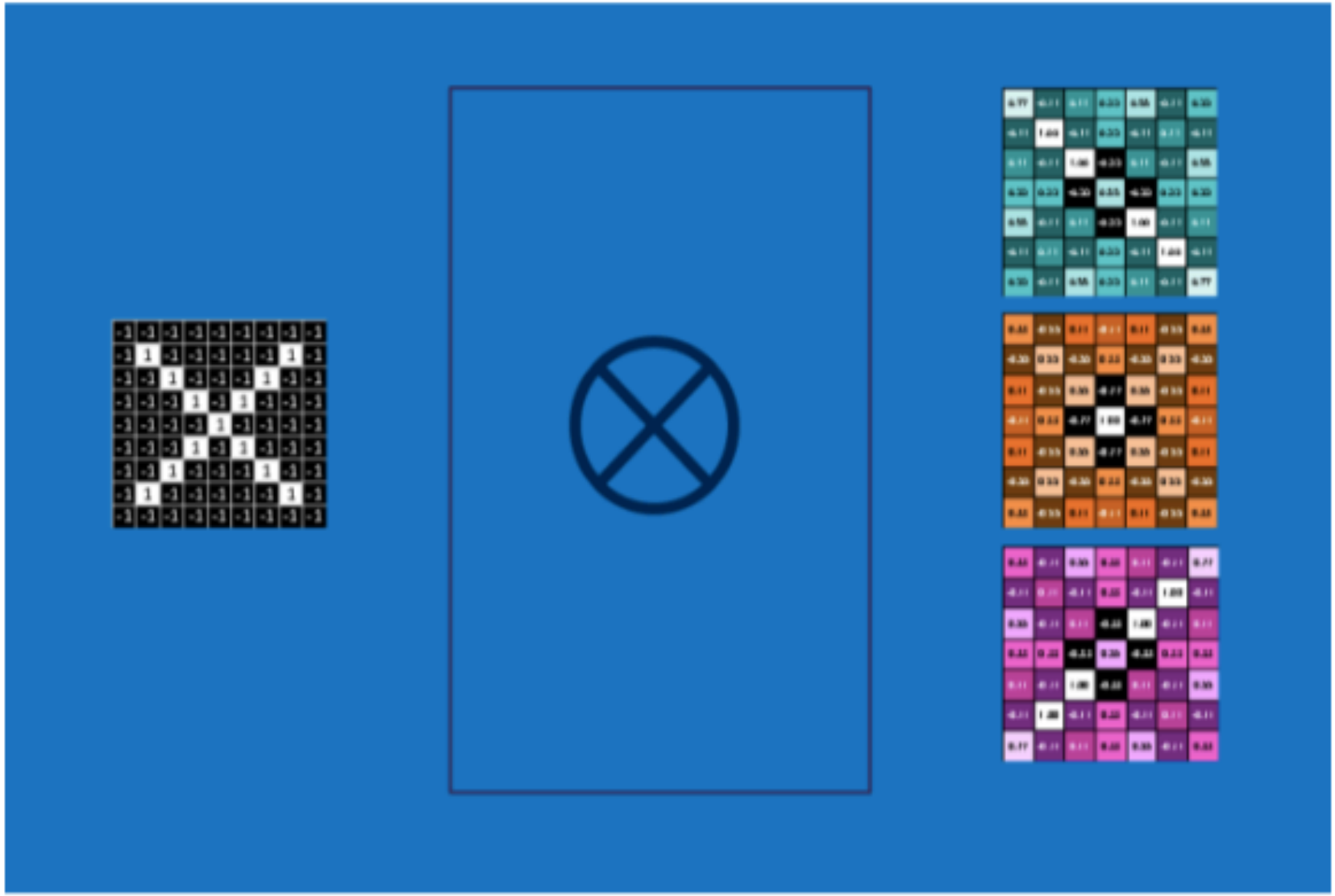
Convolution in one location



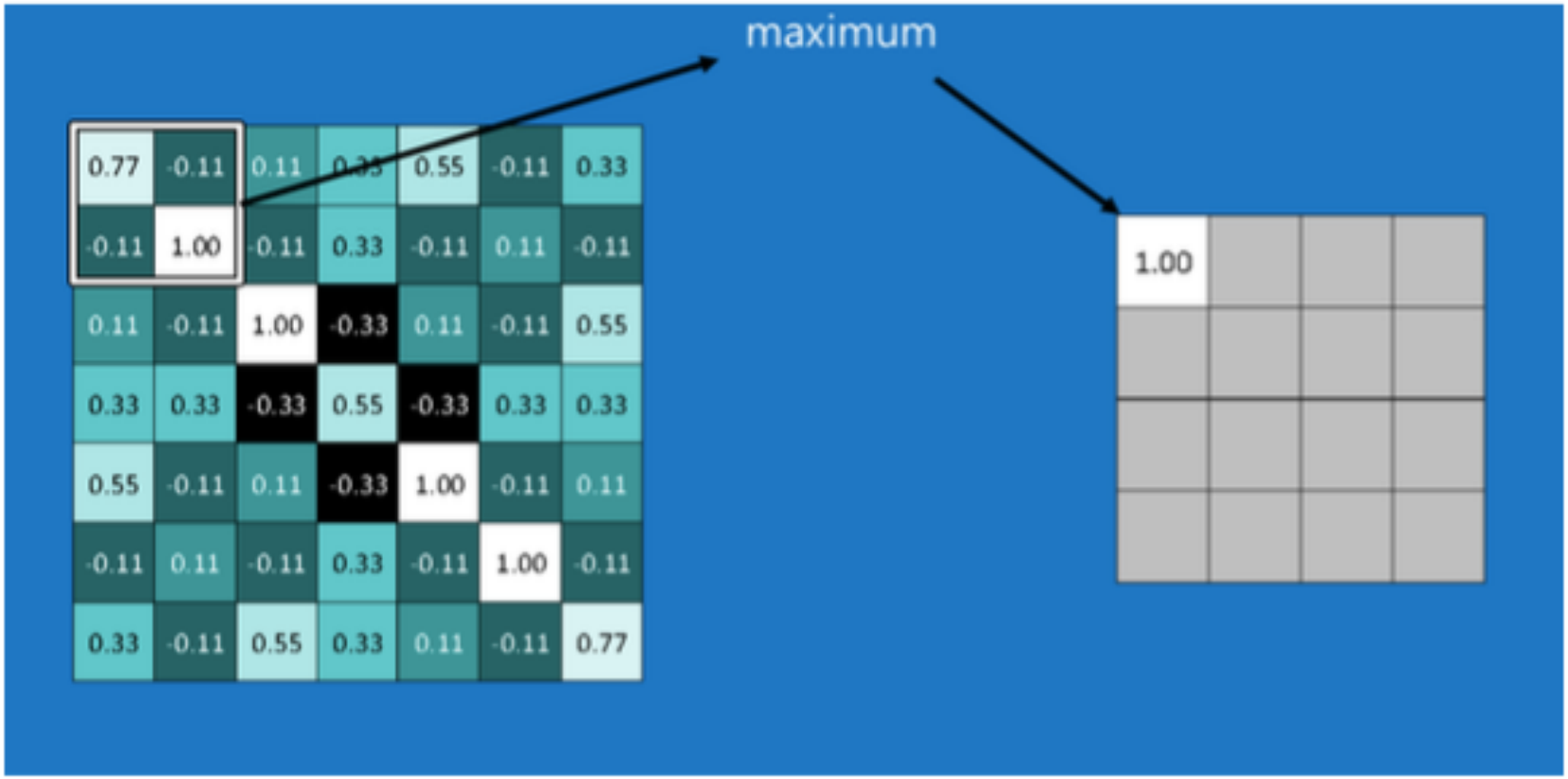
Convolution over the image



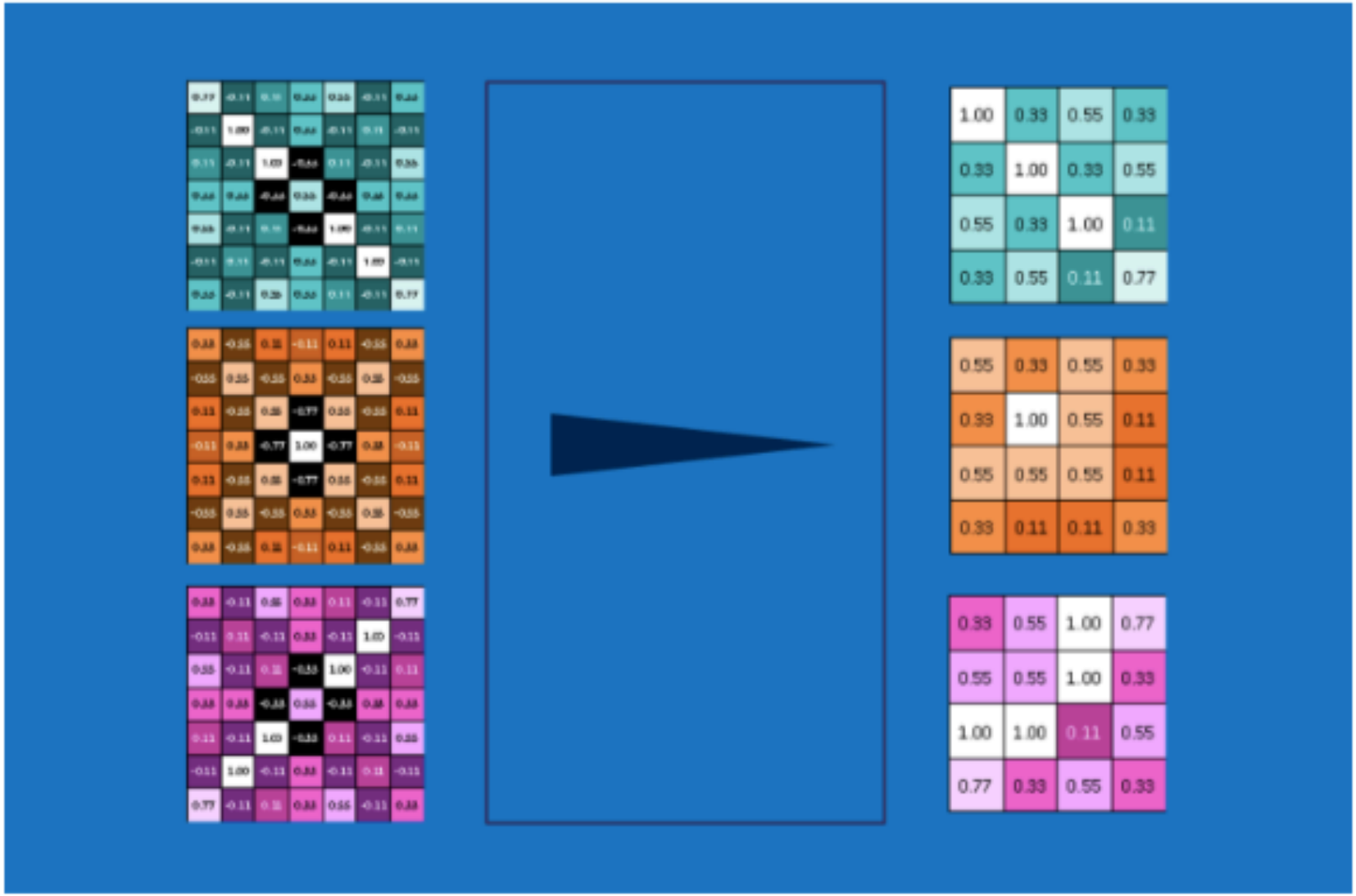
Convolution with multiple filters produces a stack of images



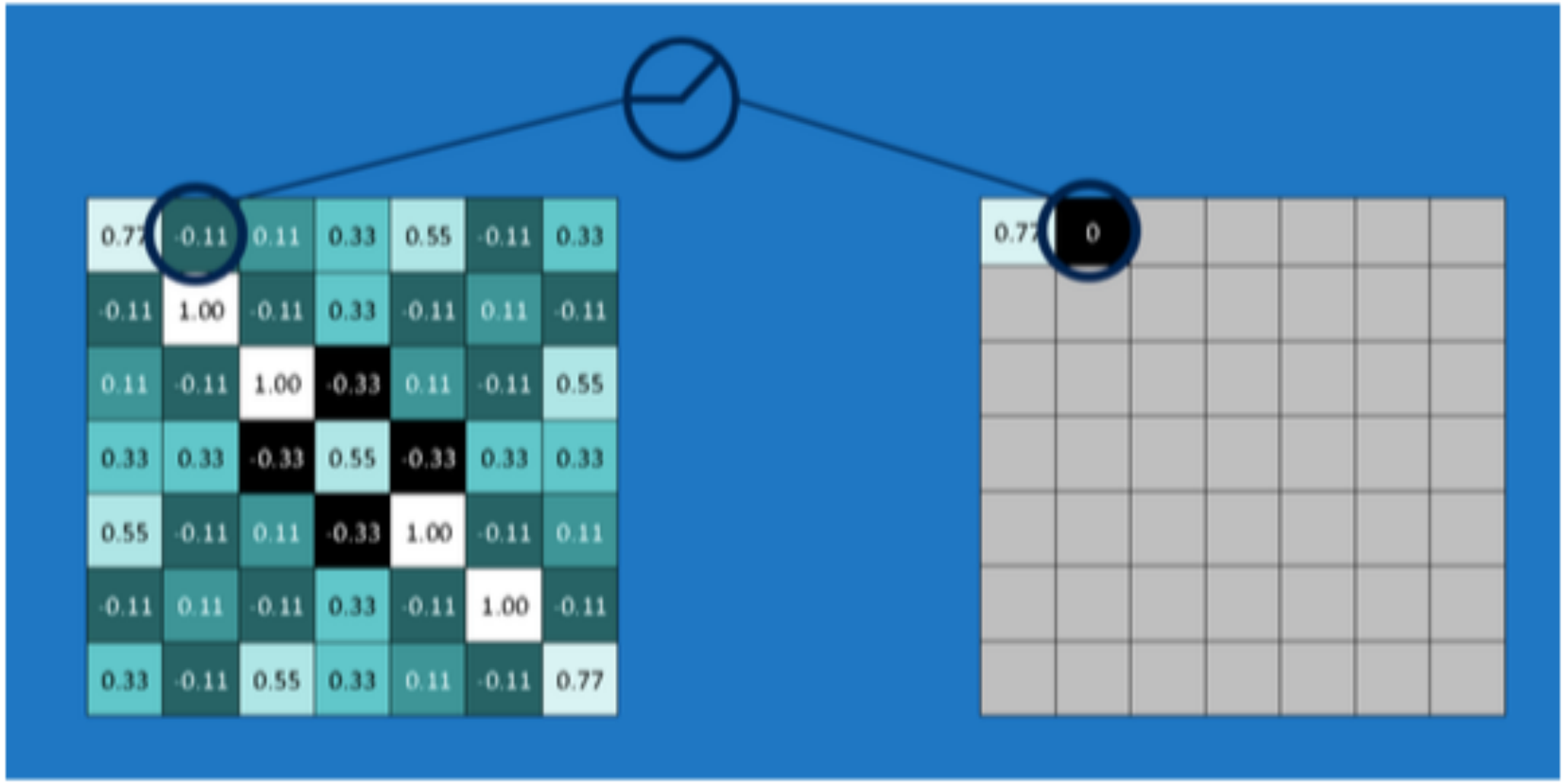
Pooling: take large images and shrink them down while preserving the most important information in them.
(Parameters: Window Size and stride)



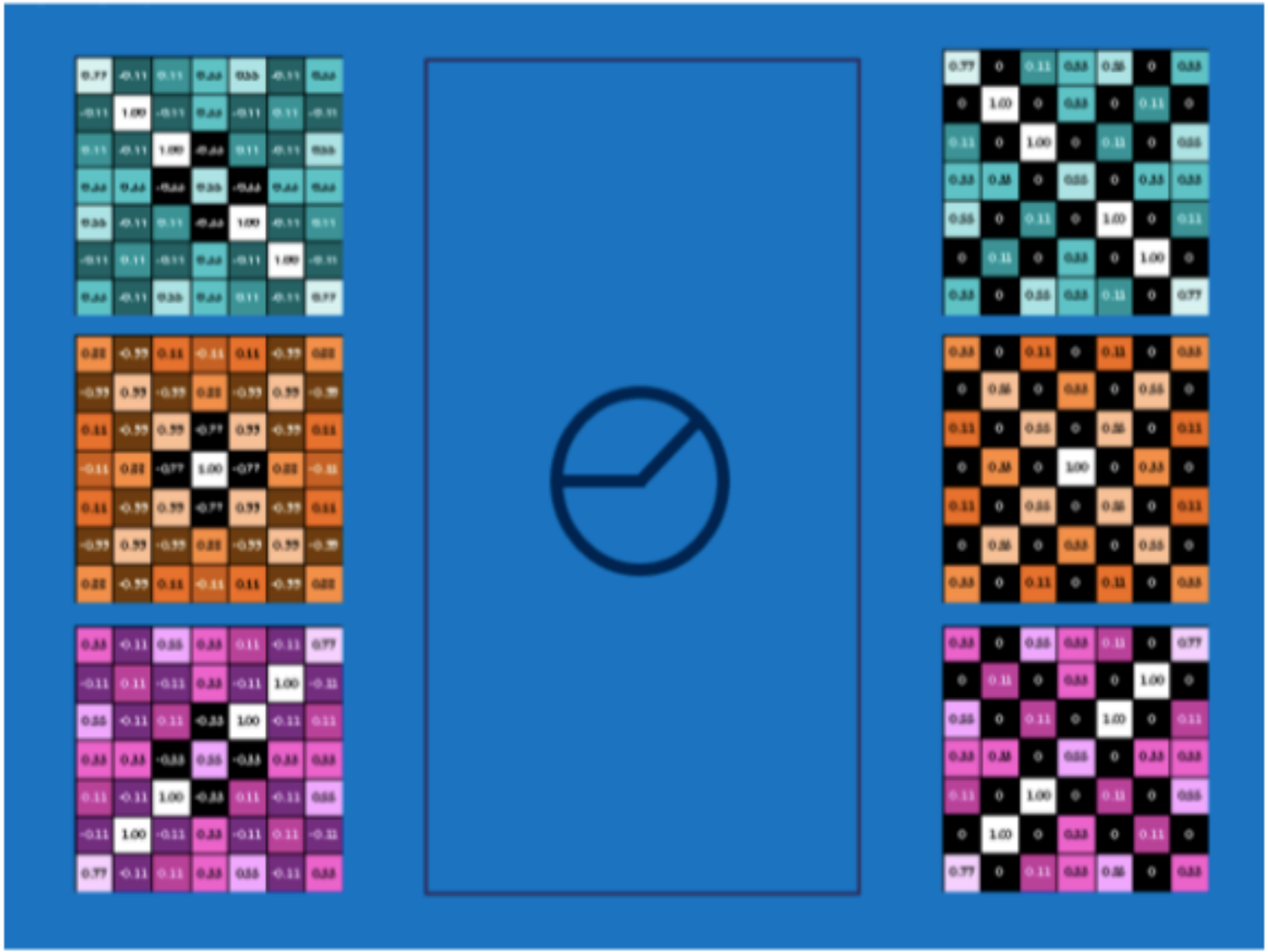
Maxpooling on the stack of images



Normalization: RELU layer

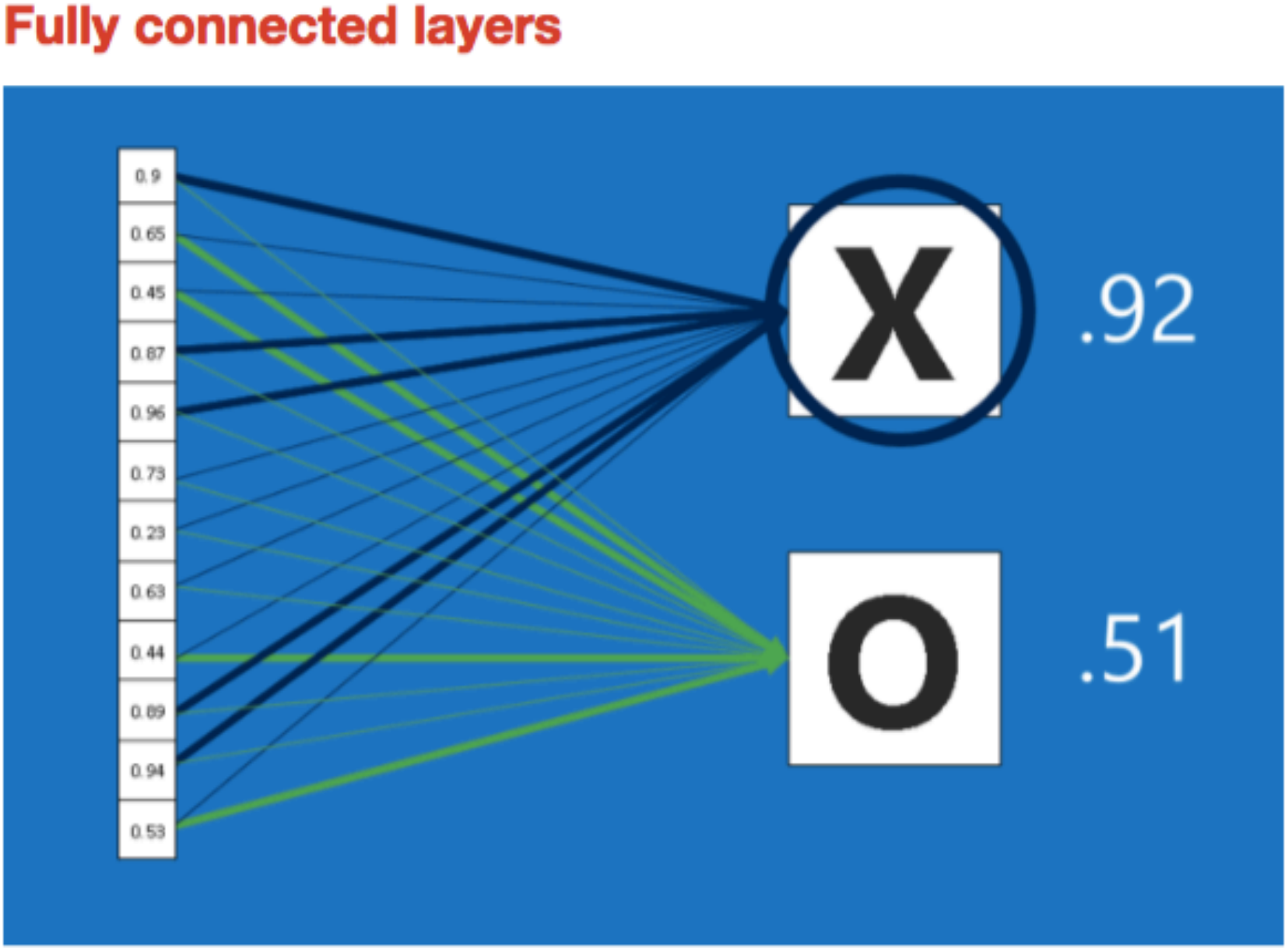


RELU applied to a stack of images



Stacking of modules

Fully connected layer



Multiple layers of different kinds of modules

Hyperparamaters

Hyperparameters

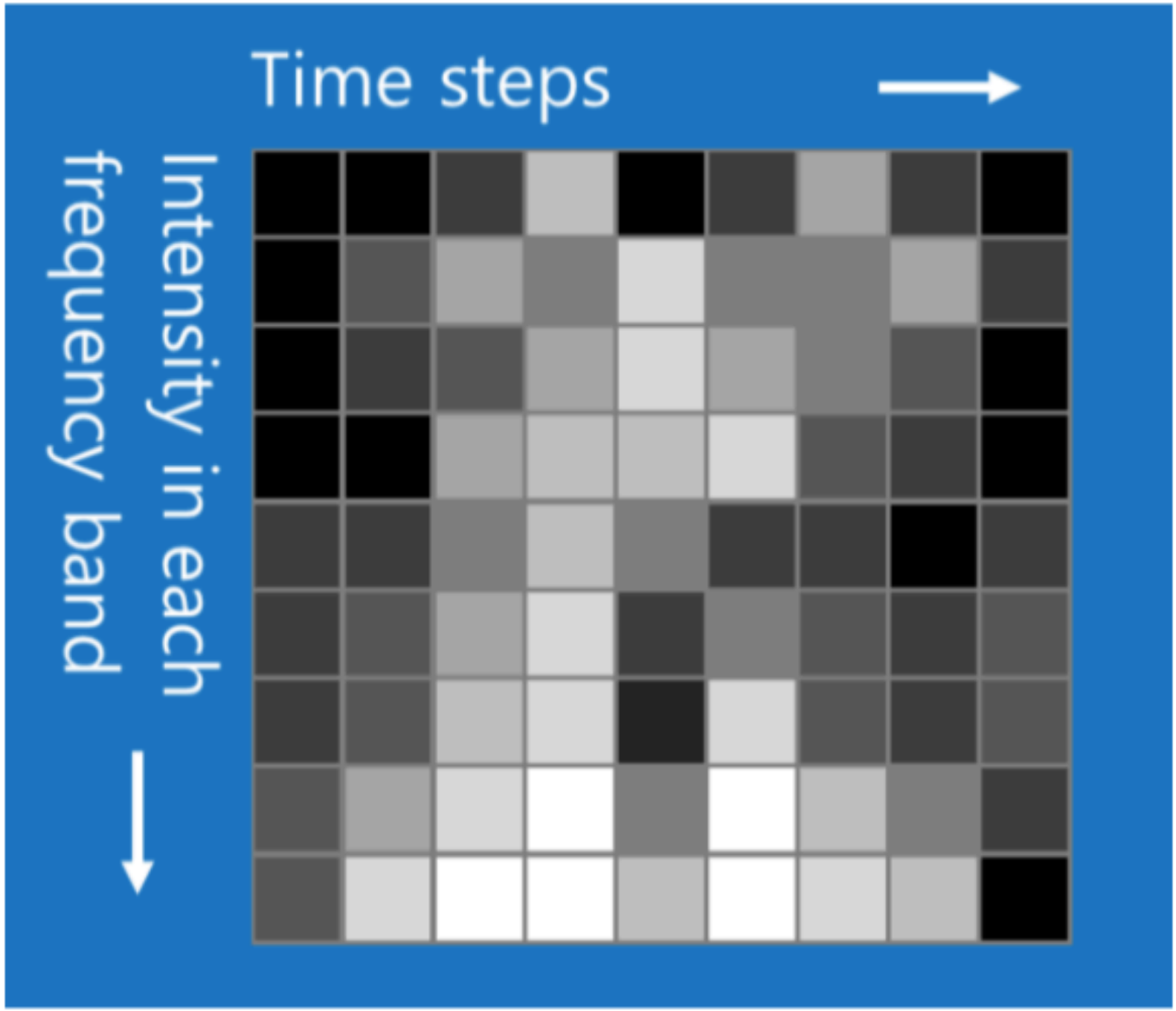
Unfortunately, not every aspect of CNNs can be learned in so straightforward a manner. There is still a long list of decisions that a CNN designer must make.

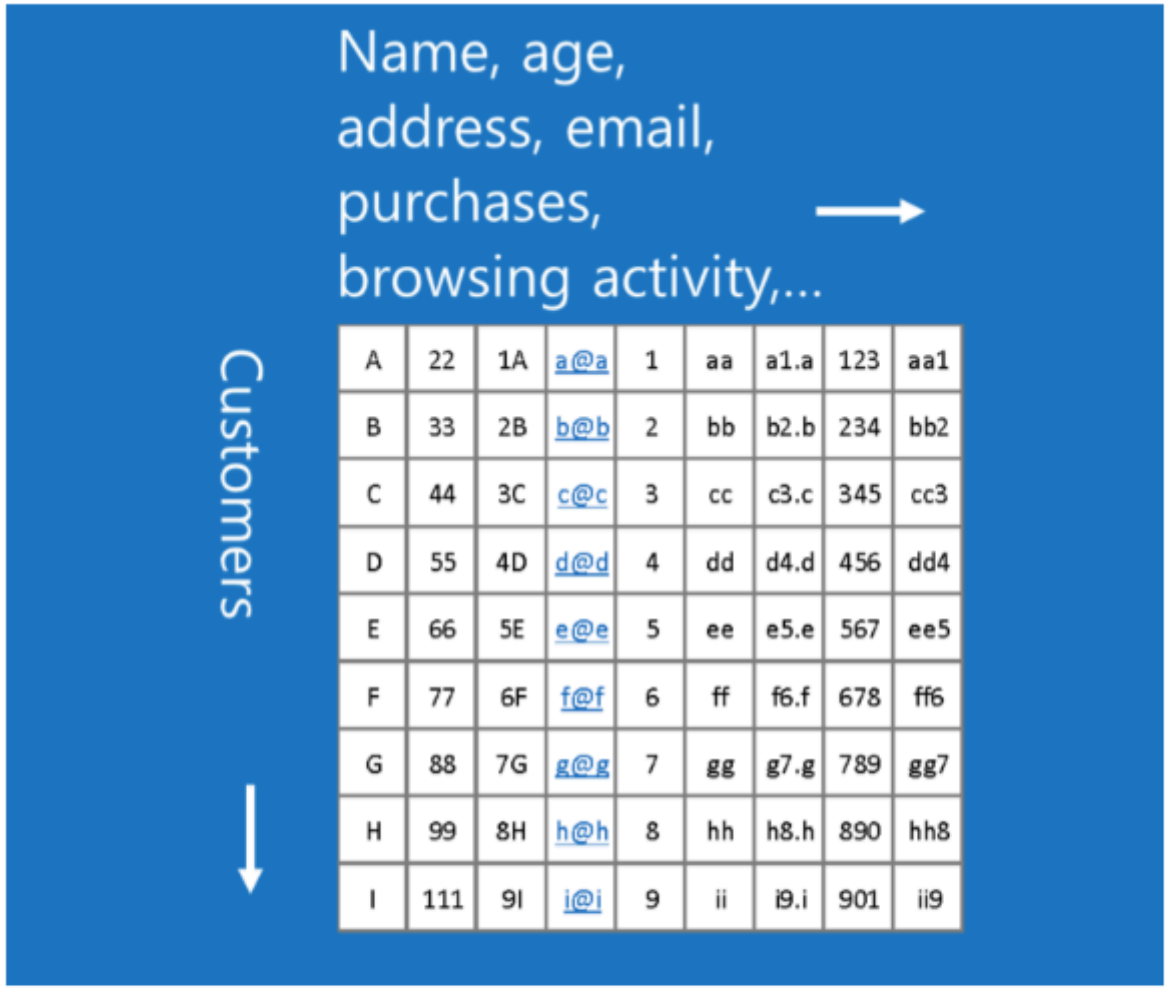
For each convolution layer, How many features? How many pixels in each feature?
For each pooling layer, What window size? What stride?
For each extra fully connected layer, How many hidden neurons?

In addition to these there are also higher level architectural decisions to make: How many of each layer to include? In what order? Some deep neural networks can have over a thousand layers, which opens up a lot of possibilities.

With so many combinations and permutations, only a small fraction of the possible CNN configurations have been tested. CNN designs tend to be driven by accumulated community knowledge, with occasional deviations showing surprising jumps in performance. And while we've covered the building blocks of vanilla CNNs, there are lots of other tweaks that have been tried and found effective, such as new layer types and more complex ways to connect layers with each other.

Beyond images





A	22	1A	a@a	1	aa	a1.a	123	aa1
B	33	2B	b@b	2	bb	b2.b	234	bb2
C	44	3C	c@c	3	cc	c3.c	345	cc3
D	55	4D	d@d	4	dd	d4.d	456	dd4
E	66	5E	e@e	5	ee	e5.e	567	ee5
F	77	6F	f@f	6	ff	f6.f	678	ff6
G	88	7G	g@g	7	gg	g7.g	789	gg7
H	99	8H	h@h	8	hh	h8.h	890	hh8
I	111	9I	i@i	9	ii	i9.i	901	ii9

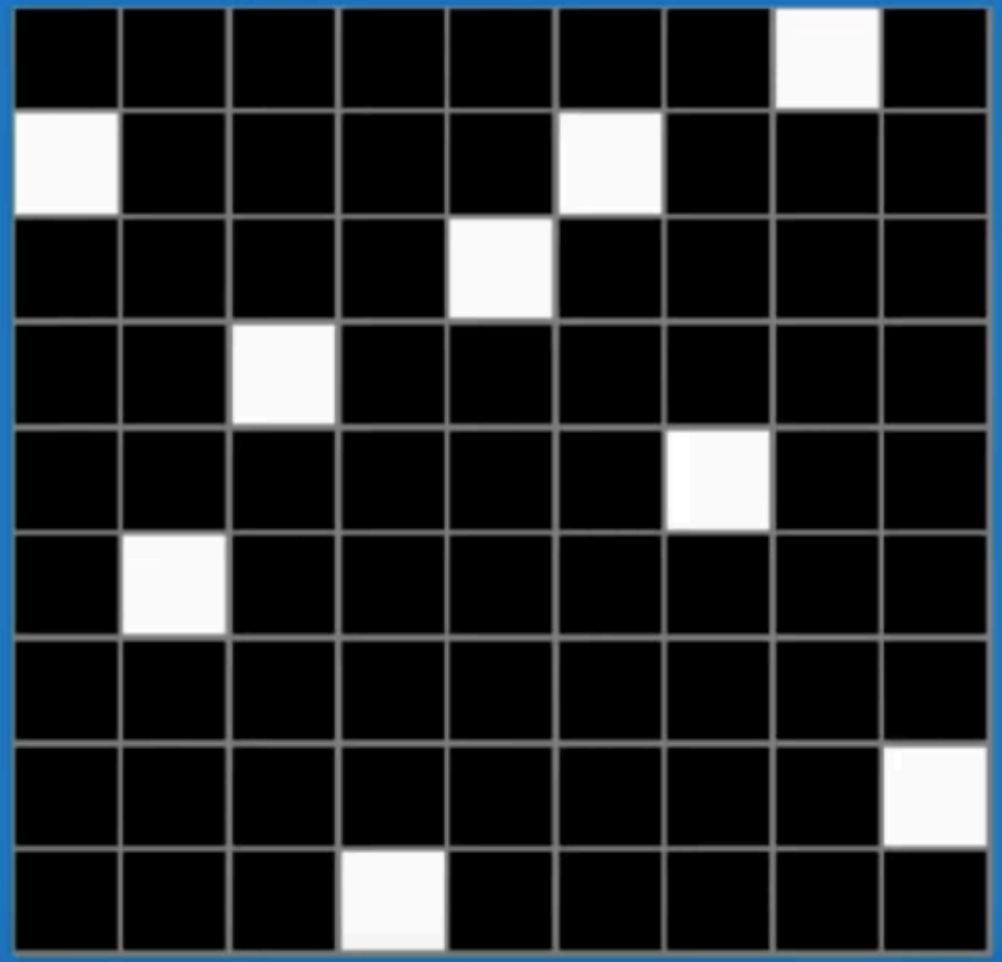
An example of data that doesn't fit this format is customer data, where each row in a table represents a customer, and each column represents information about them, such as name, address, email, purchases and browsing history. In this case, the location of rows and columns doesn't really matter. Rows can be rearranged and columns can be re-ordered without losing any of the usefulness of the data. In contrast, rearranging the rows and columns of an image makes it largely useless.

A rule of thumb: If your data is just as useful after swapping any of your columns with each other, then you can't use Convolutional Neural Networks.

Text

Position in sentence

Words in dictionary



9. Convolutional Layers

Sometimes we are interested in making predictions based on ordered sets of items (e.g. the sequence of words in a sentence, the sequence of sentences in a document and so on). Consider for example predicting the sentiment (positive, negative or neutral) of a sentence. Some of the sentence words are very informative of the sentiment, other words are less informative, and to a good approximation, an informative clue is informative regardless of its position in the sentence. We would like to feed all of the sentence words into a learner, and let the training process figure out the important clues. One possible solution is feeding a CBOW representation into a fully connected network such as an MLP. However, a downside of the CBOW approach is that it ignores the ordering information completely, assigning the sentences "it was not good, it was actually quite bad" and "it was not bad, it was actually quite good" the exact same representation. While the global position of the indicators "not good" and "not bad" does not matter for the classification task, the local ordering of the words (that the word "not" appears right before the word "bad") is very important. A naive approach would suggest embedding word-pairs (bi-grams) rather than words, and building a CBOW over the embedded bigrams. While such an architecture could be effective, it will result in huge embedding matrices, will not scale for longer n-grams, and will suffer from data sparsity problems as it does not share statistical strength between different n-grams (the embedding of "quite good" and "very good" are completely independent of one another, so if the learner saw only one of them during training, it will not be able to deduce anything about the other based on its component words). The convolution-and-pooling (also called convolutional neural networks, or CNNs) architecture is an elegant and robust solution to this modeling problem. A convolutional neural network is designed to identify indicative local predictors in a large structure, and combine them

is an elegant and robust solution to this modeling problem. A convolutional neural network is designed to identify indicative local predictors in a large structure, and combine them

to produce a fixed size vector representation of the structure, capturing these local aspects that are most informative for the prediction task at hand.

Convolution-and-pooling architectures (LeCun & Bengio, 1995) evolved in the neural networks vision community, where they showed great success as object detectors – recognizing an object from a predefined category ("cat", "bicycles") regardless of its position in the image (Krizhevsky et al., 2012). When applied to images, the architecture is using 2-dimensional (grid) convolutions. When applied to text, we are mainly concerned with 1-d (sequence) convolutions. Convolutional networks were introduced to the NLP community in the pioneering work of Collobert, Weston and colleagues (2011) who used them for semantic-role labeling, and later by Kalchbrenner et al. (2014) and Kim (2014) who used them for sentiment and question-type classification.

Convolution and Pooling in NLP

9.1 Basic Convolution + Pooling

The main idea behind a convolution and pooling architecture for language tasks is to apply a non-linear (learned) function over each instantiation of a k-word sliding window over the sentence. This function (also called "filter") transforms a window of k words into a d dimensional vector that captures important properties of the words in the window (each dimension is sometimes referred to in the literature as a "channel"). Then, a "pooling" operation is used to combine the vectors resulting from the different windows into a single d-dimensional vector, by taking the max or the average value observed in each of the d channels over the different windows. The intention is to focus on the most important "features" in the sentence, regardless of their location. The d-dimensional vector is then fed further into a network that is used for prediction. The gradients that are propagated back from the network's loss during the training process are used to tune the parameters of the filter function to highlight the aspects of the data that are important for the task the network is trained for. Intuitively, when the sliding window is run over a sequence, the filter function learns to identify informative k-grams.

More formally, consider a sequence of words $\mathbf{x} = x_1, \ldots, x_n$, each with their corresponding d_{emb} dimensional word embedding $v(x_i)$. A 1d convolution layer[61] of width k works by moving a sliding window of size k over the sentence, and applying the same "filter" to each window in the sequence $[v(x_i); v(x_{i+1}); \ldots; v(x_{i+k-1})]$. The filter function is usually a linear transformation followed by a non-linear activation function.

Let the concatenated vector of the ith window be $\mathbf{w_i} = [v(x_i); v(x_{i+1}); \ldots; v(x_{i+k-1})]$, $\mathbf{w_i} \in \mathbb{R}^{k \cdot d_{\text{emb}}}$. Depending on whether we pad the sentence with $k-1$ words to each side, we may get either $m = n - k + 1$ (*narrow convolution*) or $m = n + k + 1$ windows (*wide convolution*) (Kalchbrenner et al., 2014). The result of the convolution layer is m vectors $\mathbf{p_1}, \ldots, \mathbf{p_m}$, $\mathbf{p_i} \in \mathbb{R}^{d_{\text{conv}}}$ where:

$$\mathbf{p_i} = g(\mathbf{w_i}\mathbf{W} + \mathbf{b}) \tag{34}$$

g is a non-linear activation function that is applied element-wise, $\mathbf{W} \in \mathbb{R}^{k \cdot d_{\text{emb}} \times d_{\text{conv}}}$ and $\mathbf{b} \in \mathbb{R}^{d_{\text{conv}}}$ are parameters of the network. Each $\mathbf{p_i}$ is a d_{conv} dimensional vector, encoding the information in $\mathbf{w_i}$. Ideally, each dimension captures a different kind of indicative information. The m vectors are then combined using a *max pooling layer*, resulting in a single d_{conv} dimensional vector $\mathbf{c}$.

$$c_j = \max_{1 < i \leq m} \mathbf{p_i}[j] \tag{35}$$

$\mathbf{p_i}[j]$ denotes the jth component of $\mathbf{p_i}$. The effect of the max-pooling operation is to get the most salient information across window positions. Ideally, each dimension will "specialize" in a particular sort of predictors, and max operation will pick on the most important predictor of each type.

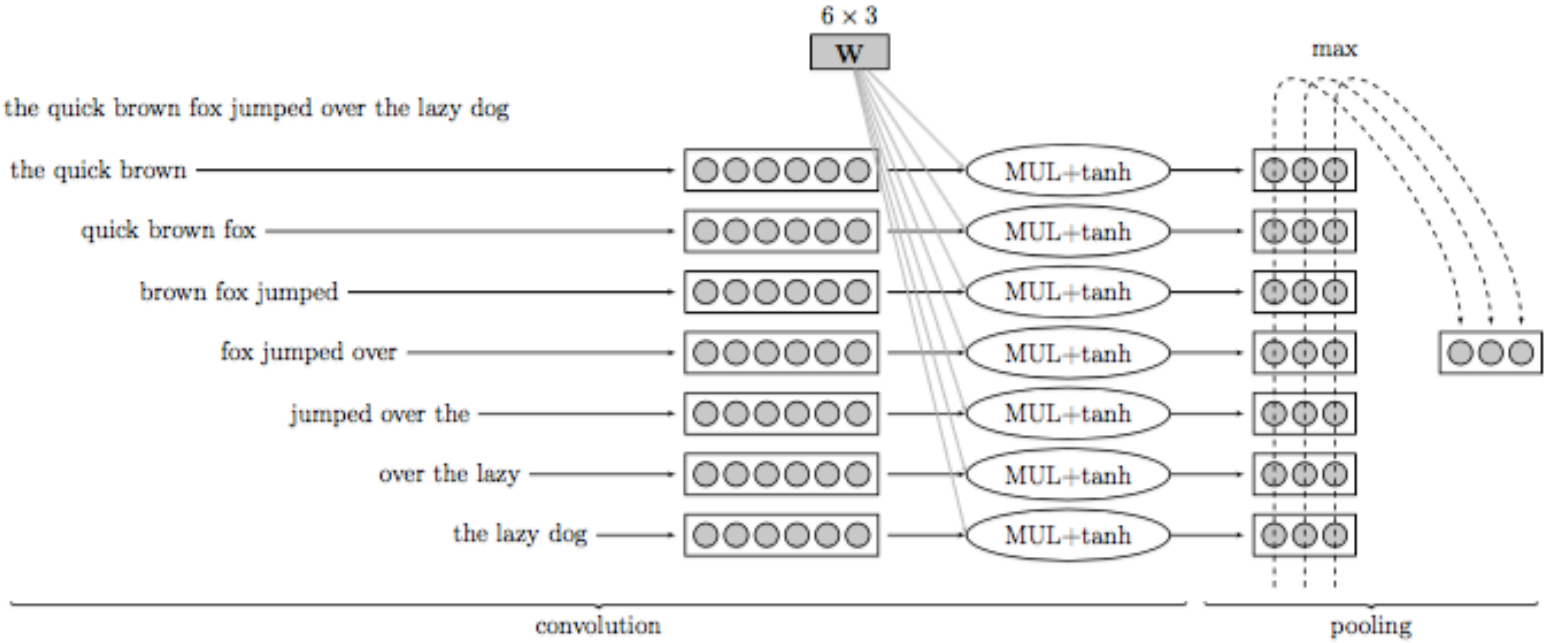


Figure 4: 1d convolution+pooling over the sentence "the quick brown fox jumped over the lazy dog". This is a narrow convolution (no padding is added to the sentence) with a window size of 3. Each word is translated to a 2-dim embedding vector (not shown). The embedding vectors are then concatenated, resulting in 6-dim window representations. Each of the seven windows is transfered through a 6×3 filter (linear transformation followed by element-wise tanh), resulting in seven 3-dimensional filtered representations. Then, a max-pooling operation is applied, taking the max over each dimension, resulting in a final 3-dimensional pooled vector.