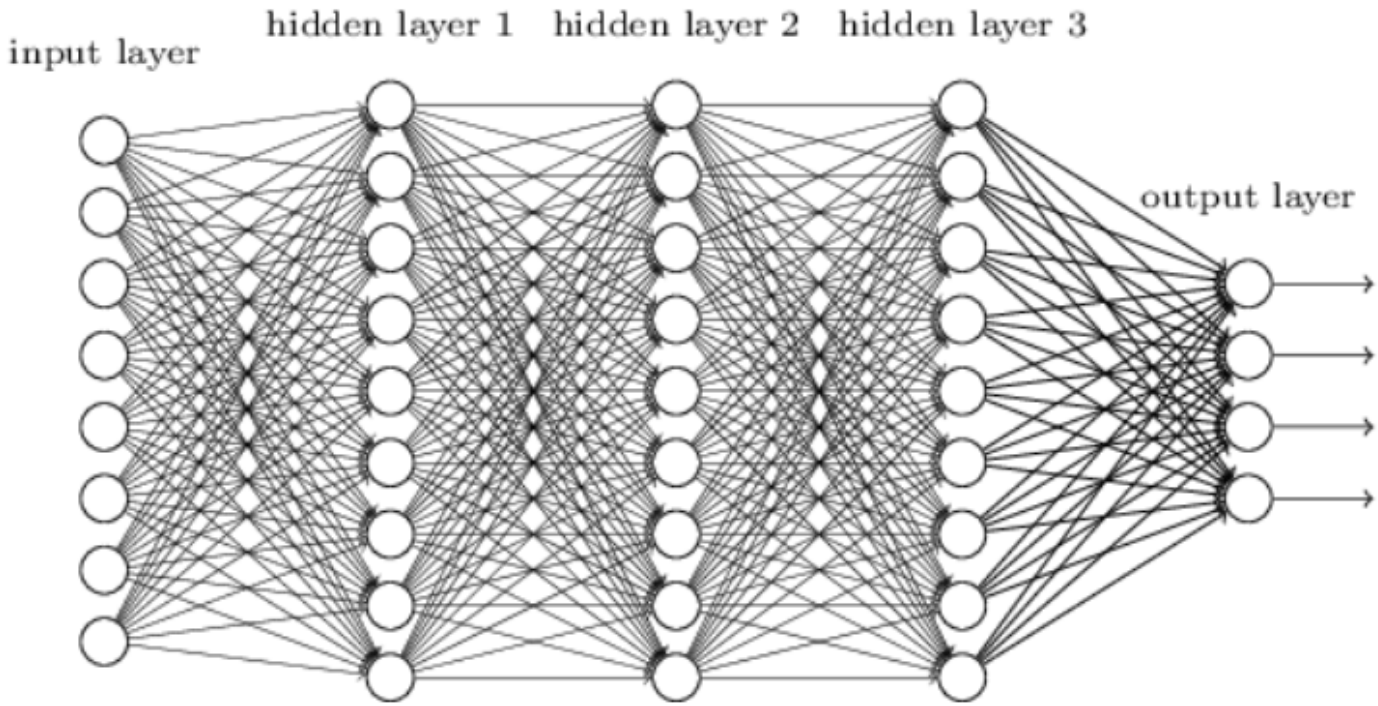
CS 5600/6600: F20: Intelligent Systems

Deep Learning and Convolutional Neural Nets: Part 1

Vladimir Kulyukin
Department of Computer Science
Utah State University

Deep Learning

Given the ANN Universality Theorem, why do we need to have multiple layers?



Theory of Deep Learning

Deeper nets use intermediate layers to develop multiple layers of abstraction.

Example: In deeper nets that recognize images, the neurons in the 1st layer may learn to recognize edges; the neurons in the 2nd layer may learn to recognize complex shapes (e.g., circles, rectangles, triangles) built from the edges recognized in the 1st layer; the 3rd layer may learn to use more complex shapes to identify objects such as cars or elephants, etc.

Should We Use Fully Connected Nets?

Does it make sense to use fully connected layers to classify images?

Some researchers believe that full connectivity doesn't take into account the spatial structure of the images in that it treats pixels that are far apart in the same way as pixels that are close to each other.

Convolutional neural networks (ConvNets) are based on the above intuition, which makes them well-adapted to classifying images.

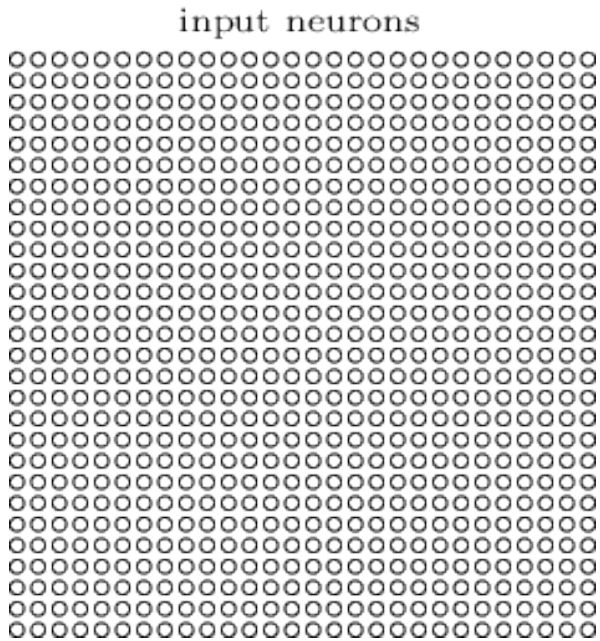
Theory of ConvNets

ConvNets use three basic ideas:

1. Local Receptive Fields;
2. Shared Weights;
3. Pooling.

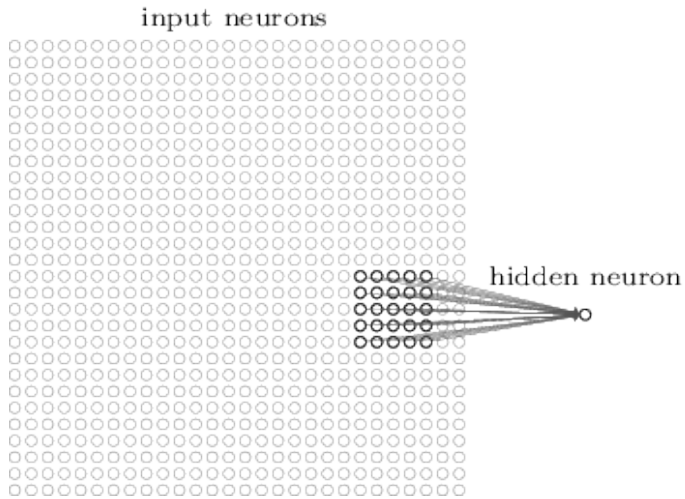
Local Receptive Fields

Let's suppose that we are classifying 28 x 28 MNIST images. Below is a sample input image of 28 x 28 input neurons. We'll connect these neurons to the hidden layer.



Local Receptive Fields

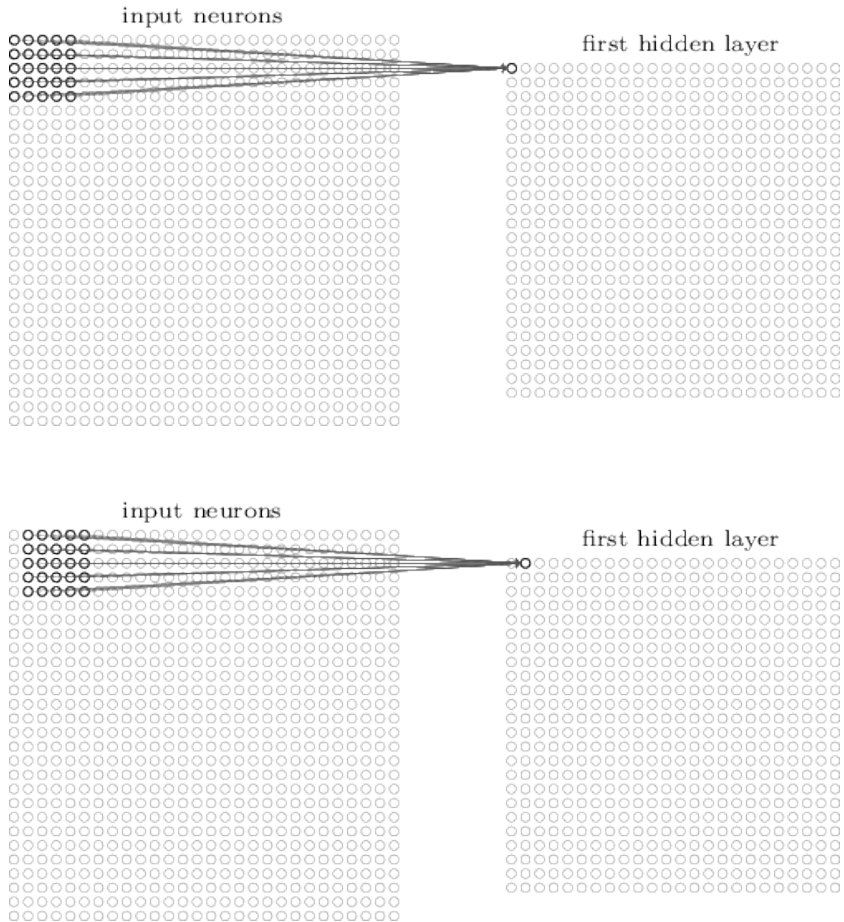
Each neuron in the 1st hidden layer is connected to a small region of the input neurons, e.g., 5 x 5 region that corresponds to 25 pixels. This region is called *local receptive field*.



Each connection learns a weight. The hidden neuron learns an overall bias as well. The neuron learns to analyze its local receptive field.

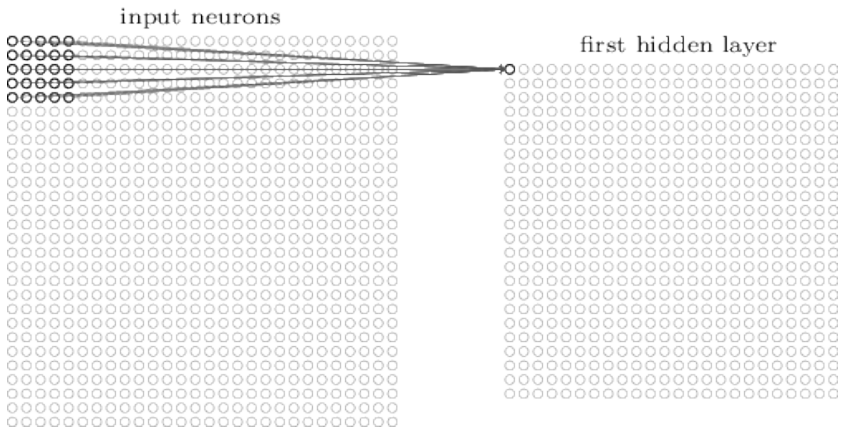
Sliding Local Receptive Field

The local receptive field is slided across the input image. For each local receptive field, there is a different hidden neuron in the 1st hidden layer. Here is a concrete example of shifting a 5x5 local receptive field one pixel at a time.



Sliding Local Receptive Field

If we have a 5x5 local receptive field and slide is one pixel at a time (this is called *stride length*), we can only slide it 23 neurons across and 23 neurons down. Hence, the hidden layer has 24x24 neurons.



Shared Weights and Biases

The same weights and bias are used for each of the 24x24 hidden neurons. If we have a 5x5 local receptive field, for hidden neuron (j, k), the output is

$$\sigma\left(b + \sum_{l=0}^{4}\sum_{m=0}^{4} w_{l,m} a_{j+l,k+m}\right)$$

For example, for hidden neuron $(11, 7)$, the activation is

$$\sigma\left(z_{11,7}\right),$$

where

$z_{11,7} = b + w_{0,0}a_{11,7} + w_{0,1}a_{11,8} + w_{0,2}a_{11,9} + w_{0,3}a_{11,10} + w_{0,4}a_{11,11} +$
$\ldots + w_{4,0}a_{15,7} + w_{4,1}a_{15,8} + w_{4,2}a_{15,9} + w_{4,3}a_{15,10} + w_{4,4}a_{15,11}$.
and $a_{x,y}$ is the activation of the hidden neuron at (x, y).

A Fundamental Implication of Shared Weights and Biases

If all hidden neurons share the same weights and bias, all neurons detect exactly the same feature at different locations in the input image.

For example, if the shared weights and bias detect a specific feature (e.g., a horizontal edge or a vertical edge), they can detect it anywhere in the image.

The shared weights and bias makes ConvNets more robust to the translation invariance of images: if an edge is moved to a different location, it will be detected.

Feature Maps

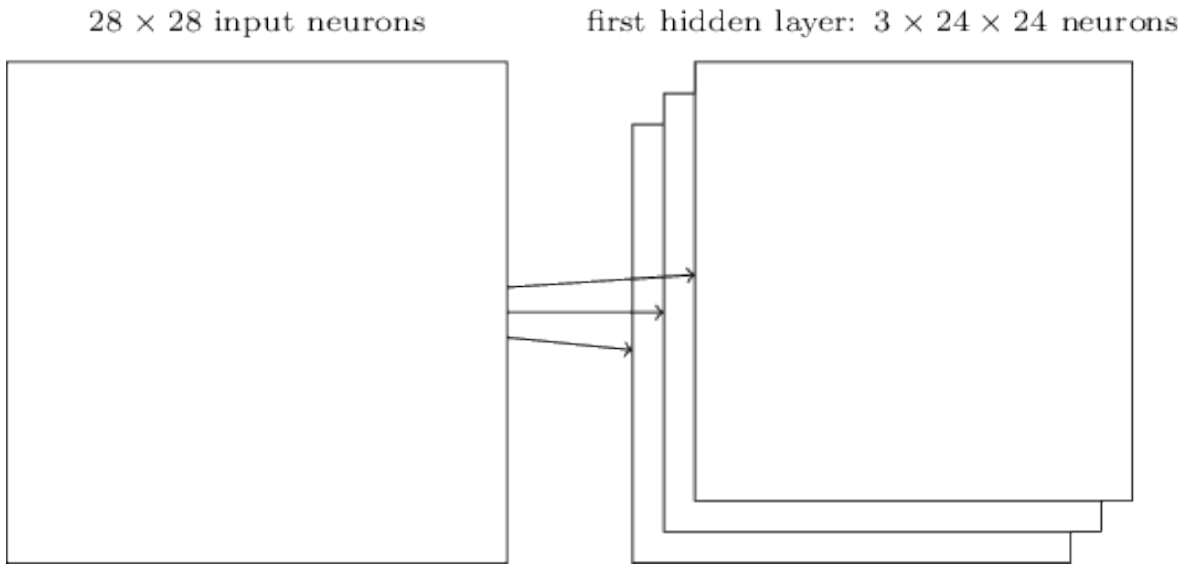
The map from the input layer to the hidden layer is called a *feature map*.

A feature map is defined by the shared weights and bias.

In the ConvNet literature, the shared weights and bias are said to define a *filter* or a *kernel*.

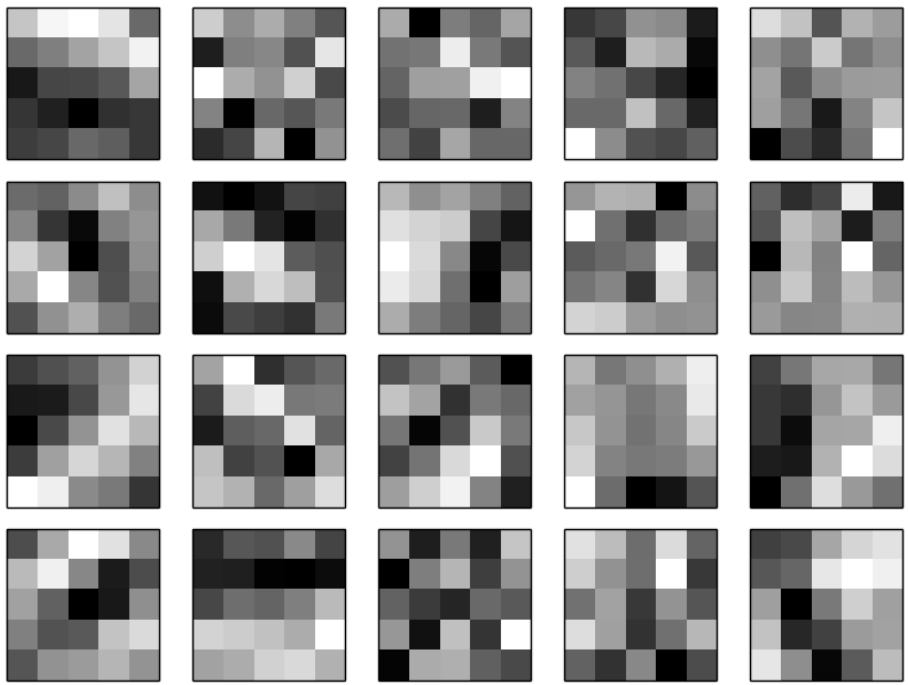
Multiple Feature Maps

Of course, if we want to recognize images with complex shapes, we need multiple features. A *convolutional layer* typically consists of several different maps. Here is an example of a convolutional layer that learns three features from its input layer. Each feature map is defined by a set of 5x5 shared weights and a single shared bias.



Multiple Feature Maps

This is an example of a trained MNIST ConvNet with 20 5x5 feature maps (filters/kernels). Each map is represented by a 5x5 image where whiter cells mean smaller (sometimes negative) weights and darker cells mean larger weights. This is a visualization of the types of features learned by a trained ConvNet.



Visualization of ConvNets

If you are interested in visualization of ConvNets, you may want to read M. Zeiler and R. Fergus. "Visualizing and Understanding Convolutional Networks." The paper can be dowloaded from `https://arxiv.org/abs/1311.2901`.

Computational Advantage of Sharing Weights and Biases

A big advantage of feature maps is that it greatly reduces the number of parameters in a ConvNet.

If we have a convolutional layer with a local receptive field of 5x5, then for each feature map we need 5x5 = 25 shared weights plus 1 bias, i.e., 26 parameters per feature map. If we have 20 feature maps, we need a total of 20x26 = 520 parameters for this convolutional layer.

If we have a 28x28 input layer fully connected to a hidden layer of 20 neurons, we need 784x20 weights plus 20 biases, which equals 15,700 parameters. This is 30 times more parameters than for the convolutional layer.

Terminological Note

The term *convolutional* comes from the fact that the activation operation used in NNs is known in mathematics as *convolution*. The operation is expressed as

$$a^1 = \sigma(b + w * a^0),$$

where a^1 is the set of output activations from one feature map and a^0 is the set of input operation and $*$ is the convolution operation.

Pooling

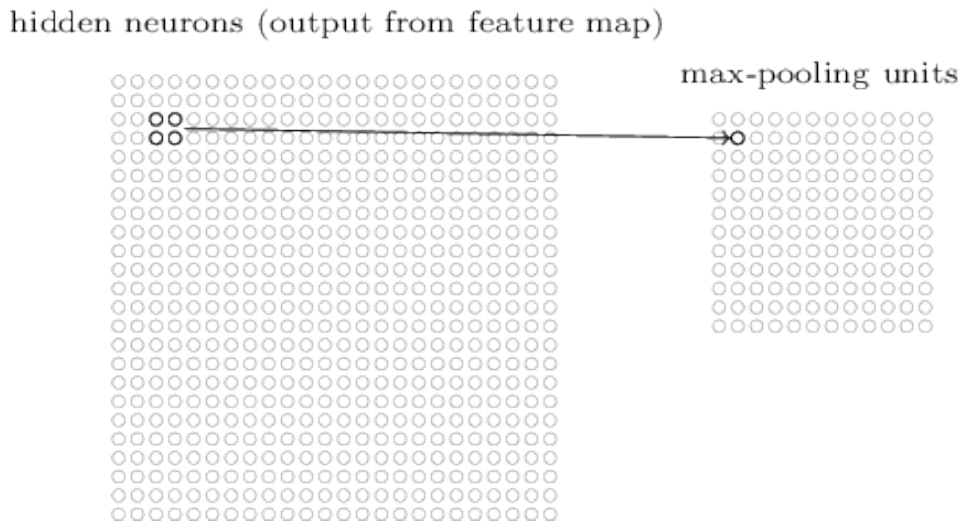
Pooling layers are used for downsampling and typically follow convolutional layers.

A pooling layer takes a feature map from a convolutional layer and outputs a condensed feature map.

A common procedure for pooling is *max-pooling* where a pooling window outputs the maximum value.

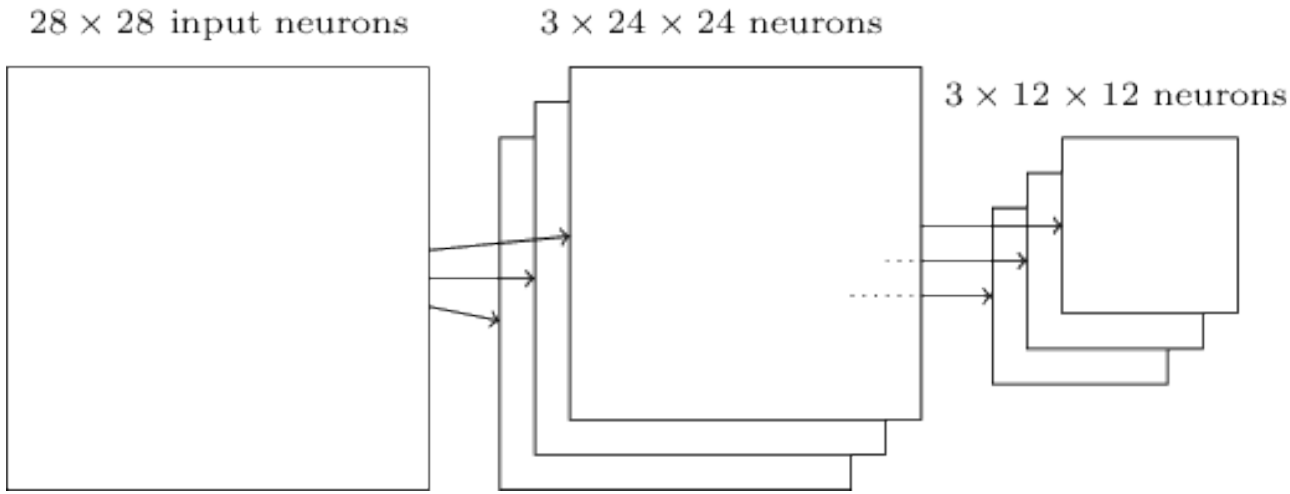
Pooling

If a max-pooling window is 2x2, a maximum value is taken from each 2x2 window applied to a feature map's output. If we have a 24x24 output from a feature map, and a 2x2 max-pooling window, we have a 12x12 neurons after max-pooling.



Pooling

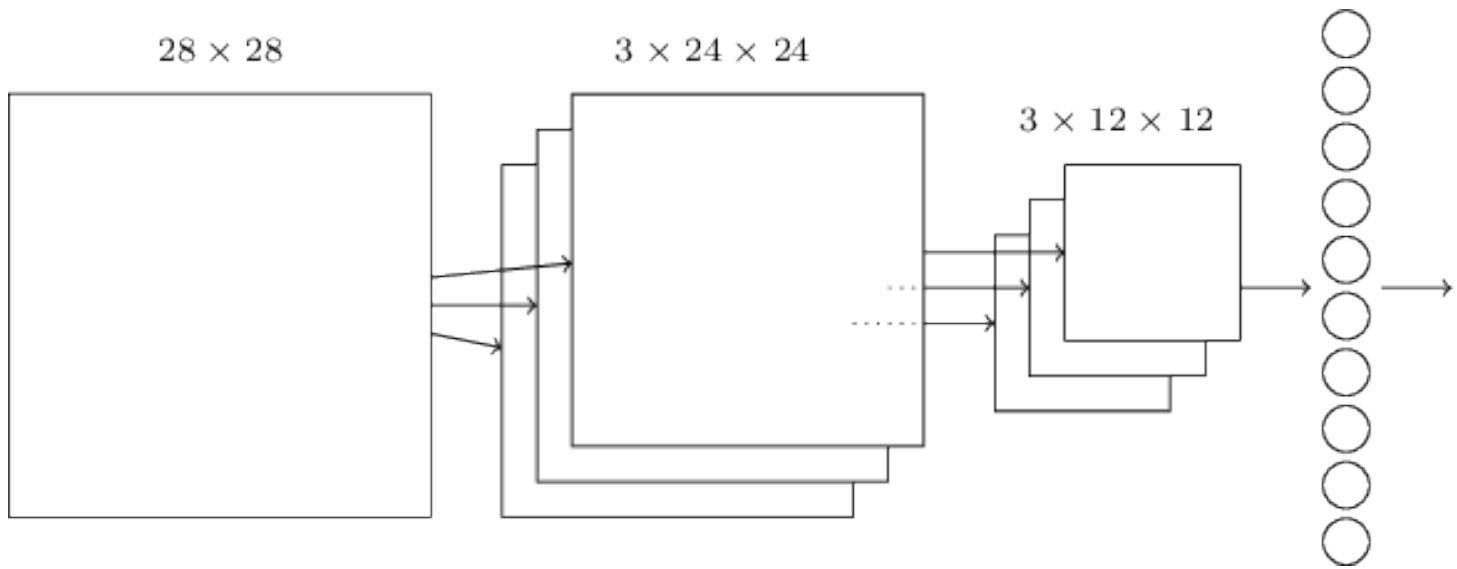
Pooling is applied to each feature map's output. Pooling does not reduce the number of feature maps' outputs, it simply downsamples each output.



Pooling is a method for the network to ask if a given feature is presented anywhere in a region of the image and to throw away the exact positional information.

Fully Connected Layer

The final type of layer is fully connected. This layer connects every neuron in the previous layer to every one of its neurons. For example, if we are training a ConvNet MNIST network, we can add a fully connected layer of 10 neurons. Here is a simple ConvNet with one convolutional layer, one max-pooling layer, and one fully connected layer.



References

1. T.M. Mitchell. *Machine Learning*.
2. M. Neilsen. *Neural Networks and Deep Learning*.