Short Introduction to Convolutions and Pooling: Deep Learning 101!

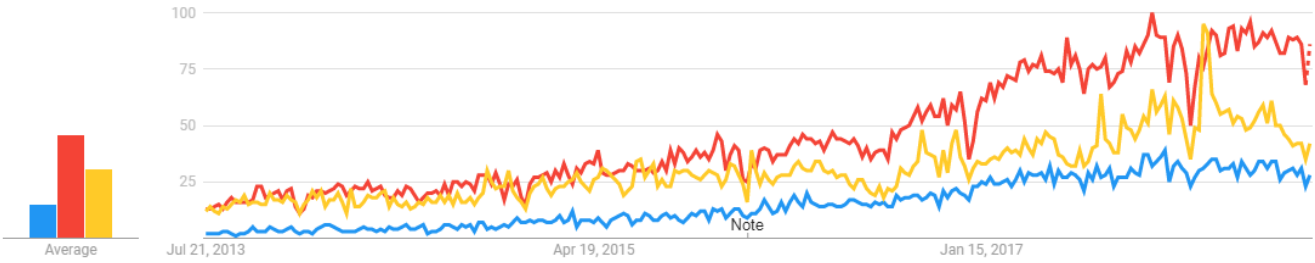
Packt_Pub Follow
Sep 25, 2018 · 7 min read

Learn the concepts of convolutions and pooling in this tutorial by Joshua Eckroth, an assistant professor of computer science at Stetson University.

Deep learning is a vast field that's generating massive interest these days. It's popularly used in research but has slowly gained market penetration in the industry in the last few years. But what essentially is deep learning?

Deep learning refers to neural networks with lots of layers. It's still quite a buzzword, but the technology behind it is real and quite sophisticated. The term has been rising in

popularity, along with machine learning and artificial intelligence, as shown in the below Google trend chart:



The primary advantage of deep learning is that combining more data with computational power often produces more accurate results, without the significant effort required for engineering tasks.

In this article, we will take a quick look at the concepts of convolutions and pooling. *This article assumes you have a basic knowledge of basic deep learning terms.*

Deep learning methods

Deep learning refers to several methods which may be used in a particular application. These methods include convolutional layers and pooling. Simpler and faster activation functions, such as ReLU, return the neuron's weighted sum if it's positive, and zero if negative.

Regularization techniques, such as dropout, randomly ignore weights during the weight update base to prevent overfitting. GPUs are used for faster training with the order that is 50 times faster. This is because they're optimized for matrix calculations that are used extensively in neural networks and memory units for applications such as speech recognition.

Several factors have contributed to deep learning's dramatic growth in the last five years. Large public datasets, such as ImageNet, that holds millions of labeled images covering a thousand categories and Mozilla's Common Voice Project that contain speech samples are now available. Such datasets have satisfied the basic requirement for deep

learning-lot of training data. GPUs have transitioned to deep learning and clusters while also focusing on gaming. This helps make large-scale deep learning possible.

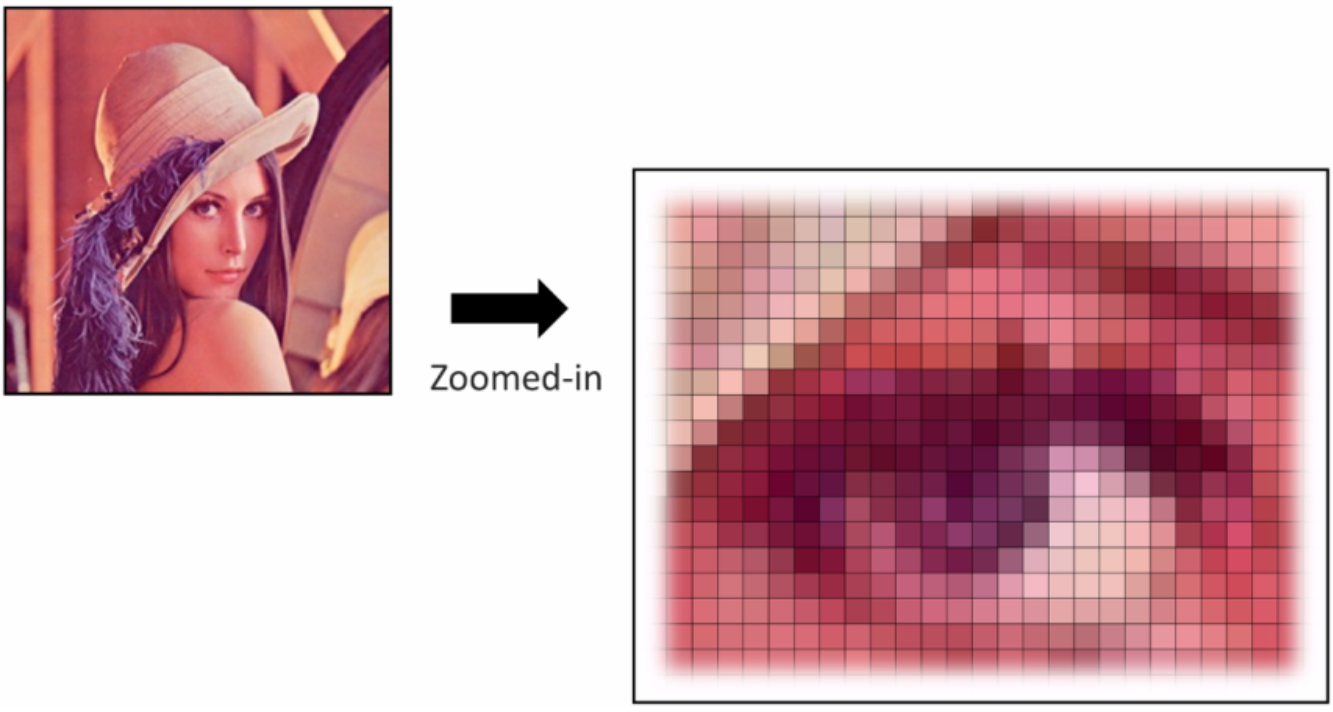
Advanced software frameworks that were released open source and are undergoing rapid improvement are also available to everyone. These include TensorFlow, Keras, Torch, and Caffe. Deep architectures that achieve state-of-the-art results, such as Inception-v3 are being used for the ImageNet dataset. This network actually has an approximate of 24 million parameters, and a large community of researchers and software engineers are quickly translating research prototypes into open source software that anyone can download, evaluate, and extend.

Convolutions and pooling

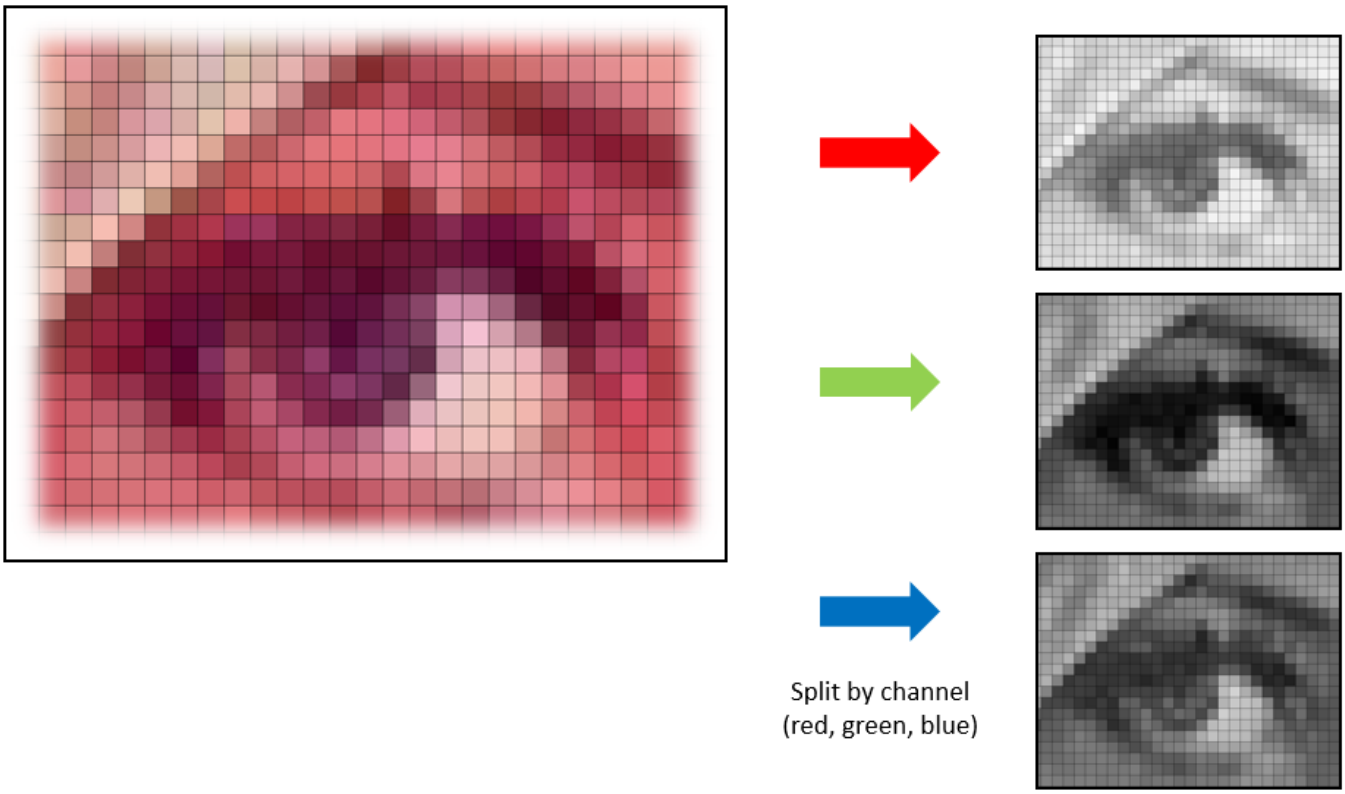
Take a closer look at two fundamental deep learning technologies, namely, convolution and pooling. Throughout this section, images have been used to understand these concepts. Nevertheless, what you'll be studying can also be applied to other data, such as, audio signals.

Convolution

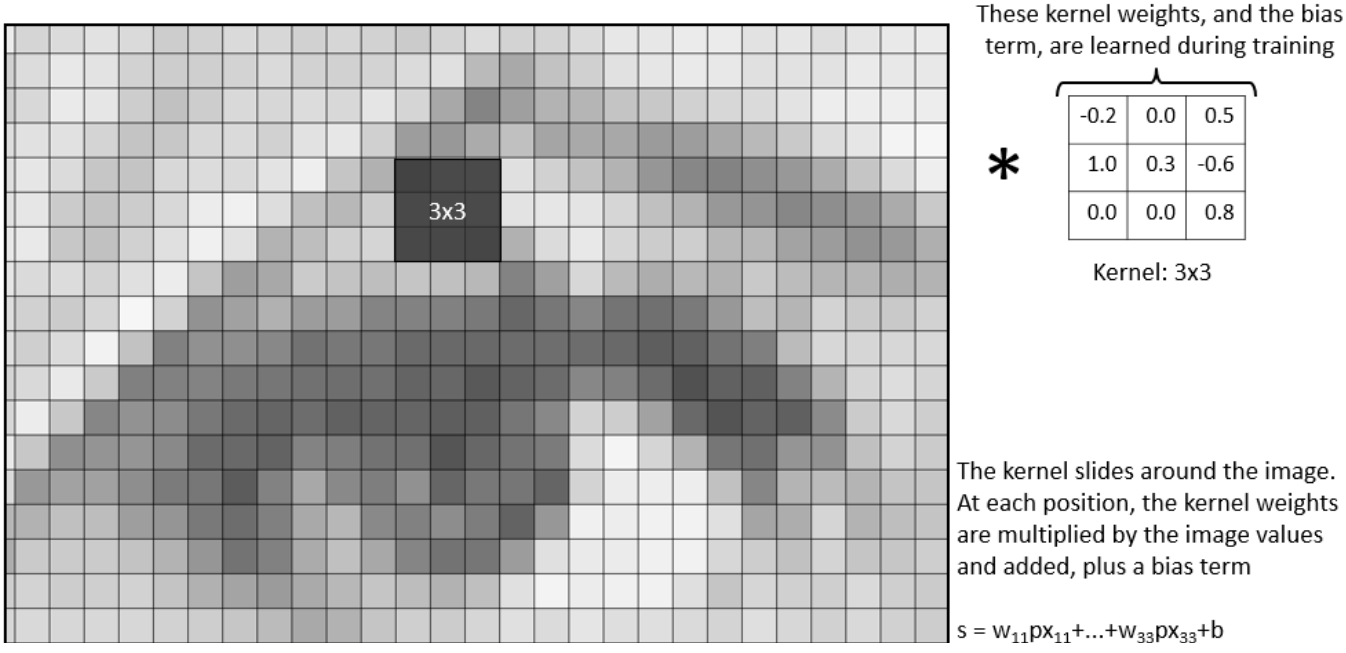
Take a look at the following photo and begin by zooming in to observe the pixels:



Convolutions occur per channel. An input image would generally consist of three channels; red, green, and blue. The next step would be to separate these three colors. The following diagram depicts this:



A convolution is a kernel. In this image, a 3 x 3 kernel is applied. Every kernel contains a number of weights. The kernel slides around the image and computes the weighted sum of the pixels on the kernel, each multiplied by their corresponding kernel weights:



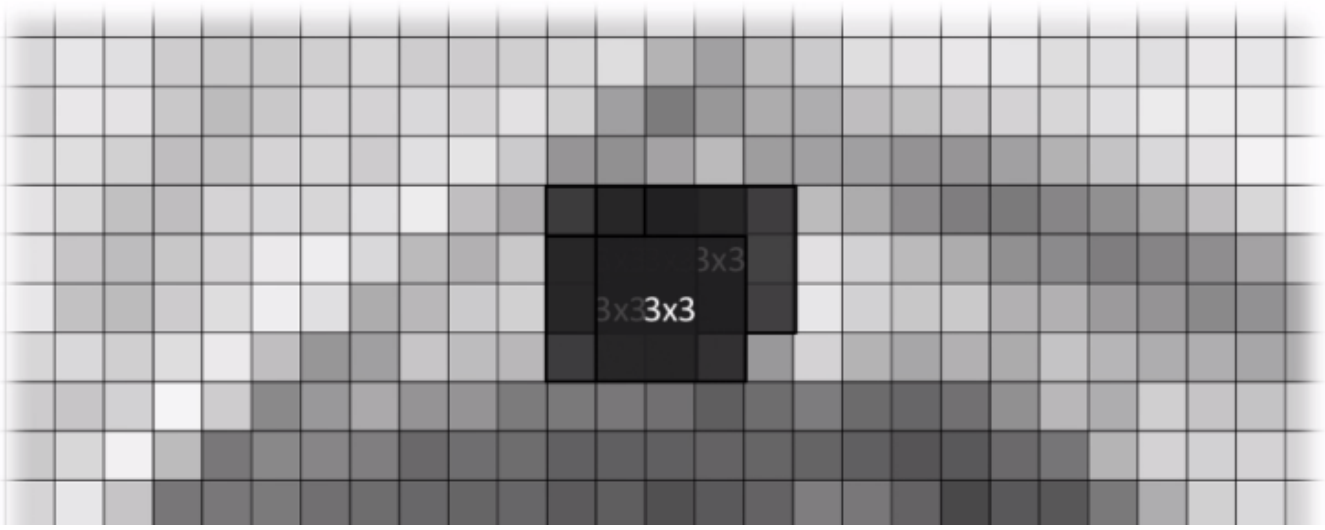
A bias term is also added. A single number, the weighted sum, is produced for each position that the kernel slides over. The kernel's weights start off with any random value and change during the training phase. The following diagram shows three examples of kernels with different weights:

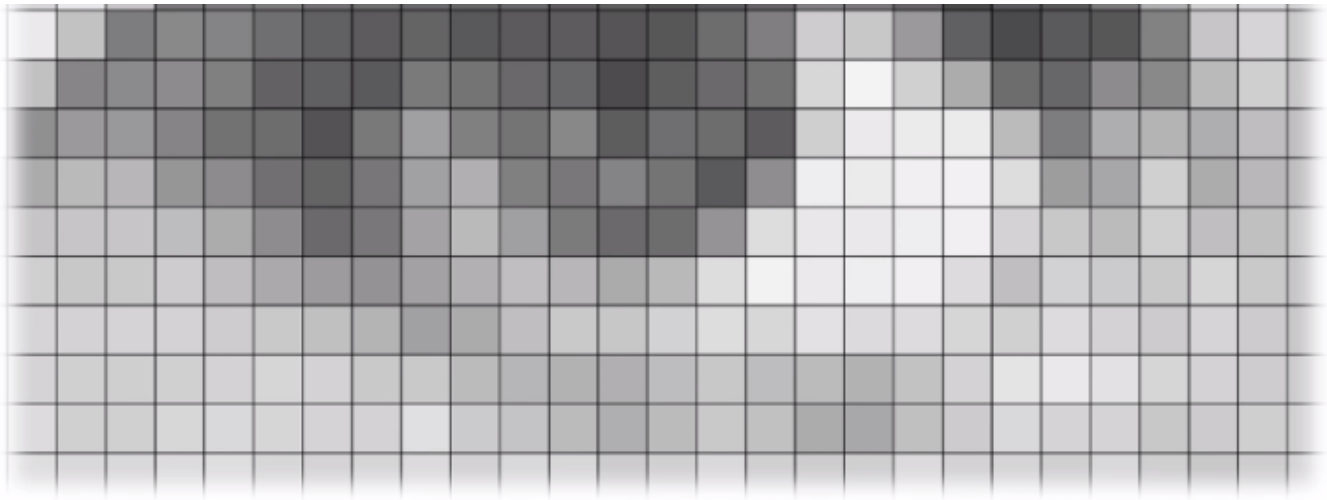
-0.2	0.0	0.5
1.0	0.3	-0.6
0.0	0.0	0.8

0.0	0.0	0.0
0.8	-0.5	0.8
0.0	-0.2	0.0

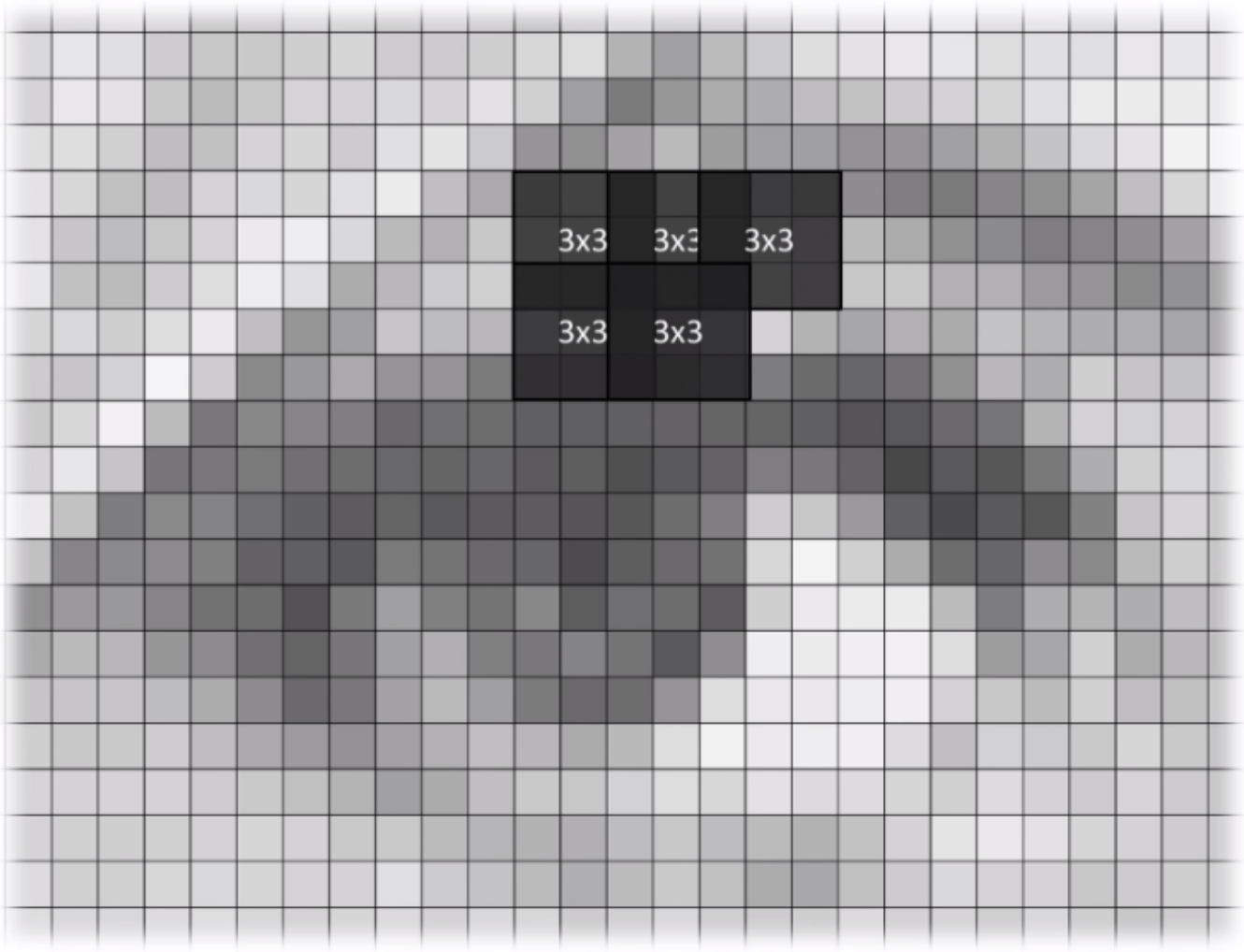
0.4	0.2	-0.2
-0.8	0.0	0.8
0.0	-0.5	0.2

You can see how the image transforms differently depending on the weights. The rightmost image highlights the edges, which is often useful for identifying objects. The stride helps you understand how the kernel slides across the image. The following diagram is an example of a 1 x 1 stride:

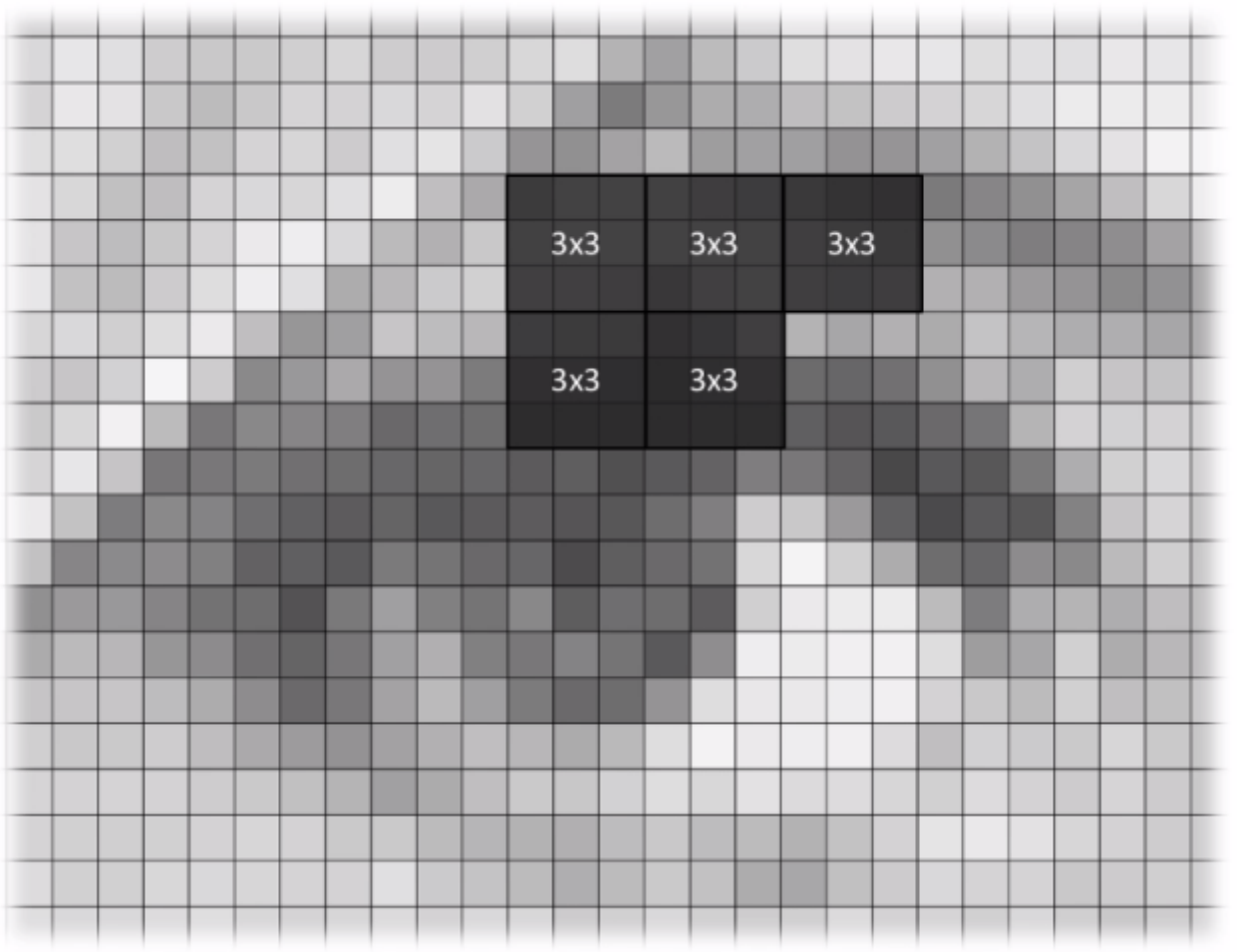




The kernel moves by one pixel to the right and then down. Throughout this process, the center of the kernel will hit every pixel of the image whilst overlapping the other kernels. It is also observed that some pixels are missed by the center of the kernel. The following image depicts a 2 x 2 stride:



In certain cases, it is observed that no overlapping takes place. To prove this, the following diagram contains a 3 x 3 stride:



In such cases, no overlap takes place because the kernel is the same size as the stride.

However, the borders of the image need to be handled differently. To effect this, you can use padding. This helps avoid extending the kernel across the border. Padding consists of extra pixels, which are always zero. They don't contribute to the weighted sum. The padding allows the kernel's weights to cover every region of the image while still letting the kernels assume that the stride is 1. The kernel produces one output for every region it covers.

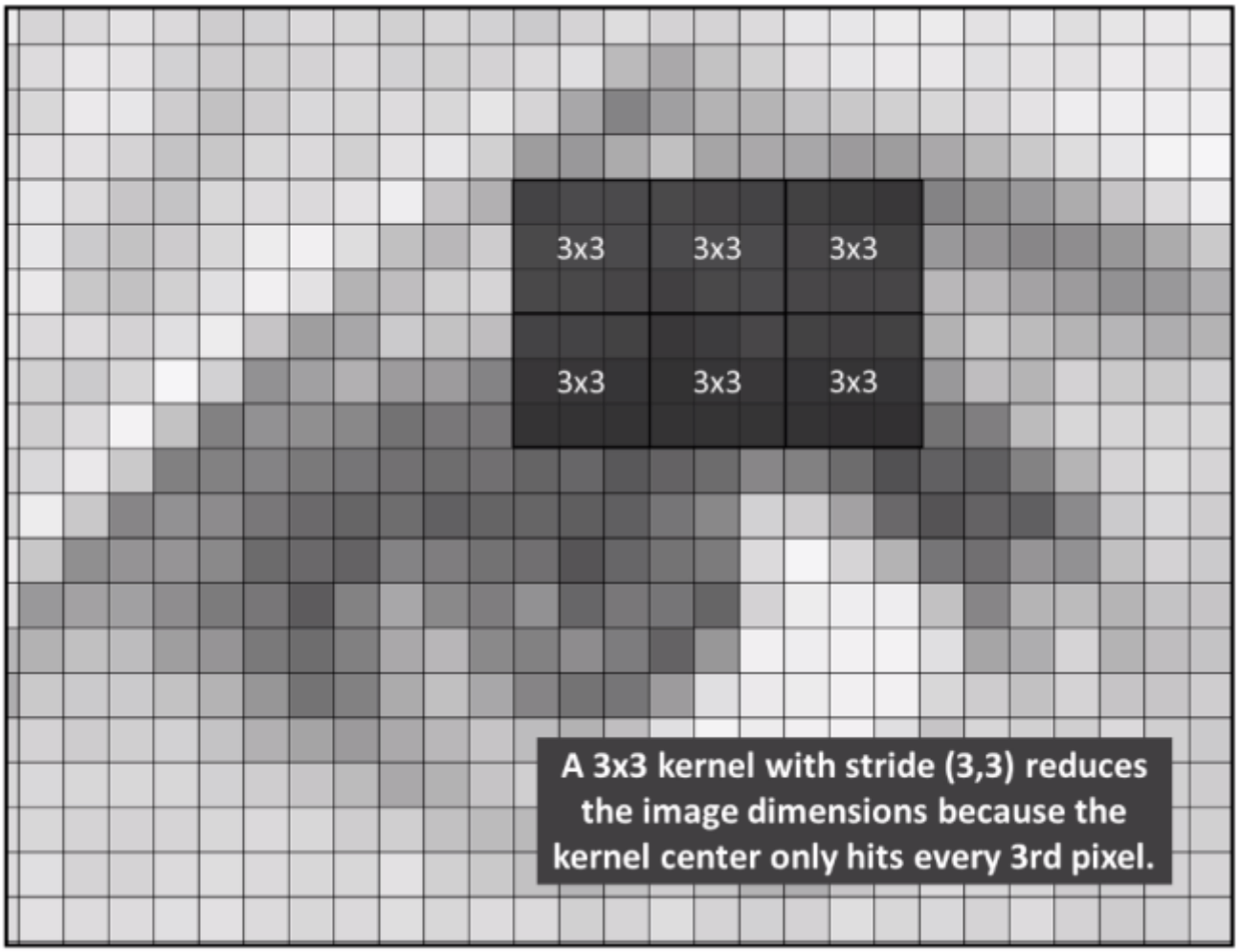
Hence, if you have a stride that is greater than 1, you'll have fewer outputs than there were original pixels. In other words, the convolution helped reduce the image's dimensions. The formula shown here tells us the dimensions of the output of a convolution:

$$D = 1 + (W - K + 2*P)/S,$$

W = input width, K = kernel size,
P = padding size, S = stride size

For example, if W = 256, K = 3,
P = 1, and S = 3, then the output
dimension is 85x85

It is a general practice to use square images. Kernels and strides are used for simplicity. This helps to focus on only one dimension, which will be the same for the width and height. In the following diagram, a 3 x 3 kernel with a (3, 3) stride is depicted:

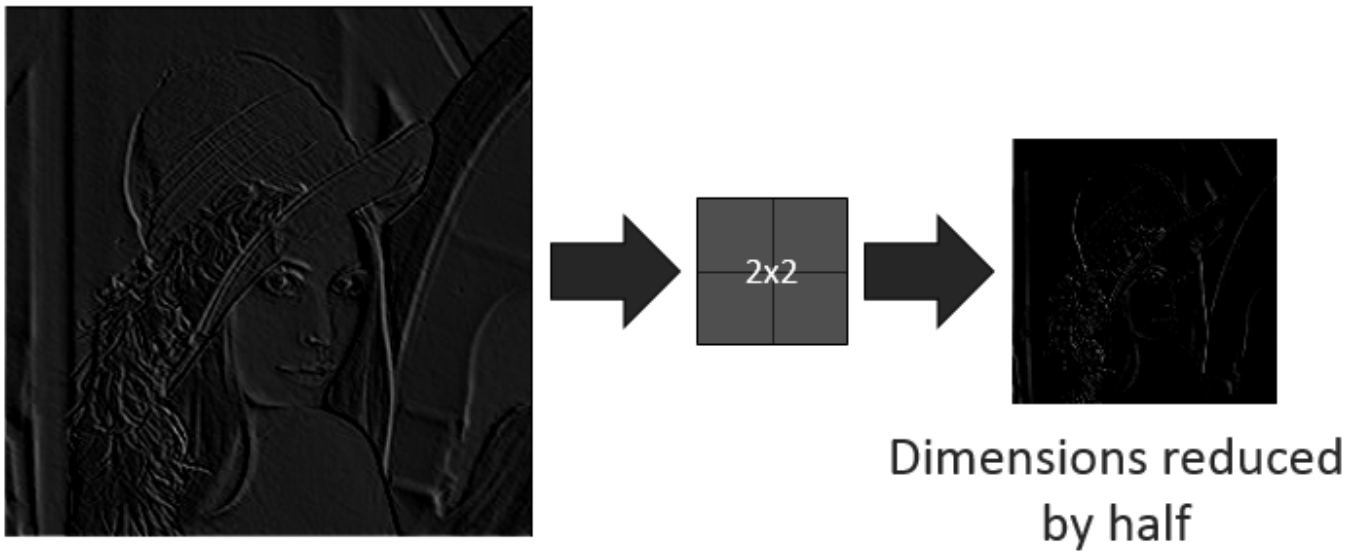


The preceding calculation gives the result of 85 width and 85 height. The image's width and height have effectively been reduced by a factor of three from the original 256. Rather than using a large stride, you can let the convolution hit every pixel using a stride of 1. This will help attain a more practical result. You also need to make sure that there is sufficient padding.

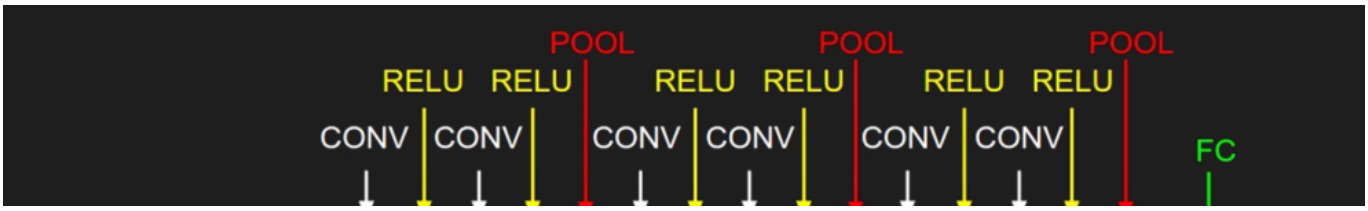
However, it is beneficial to reduce the image dimensions as you move through the network. This helps the network train faster as there will be fewer parameters. Fewer parameters imply a smaller chance of over-fitting.

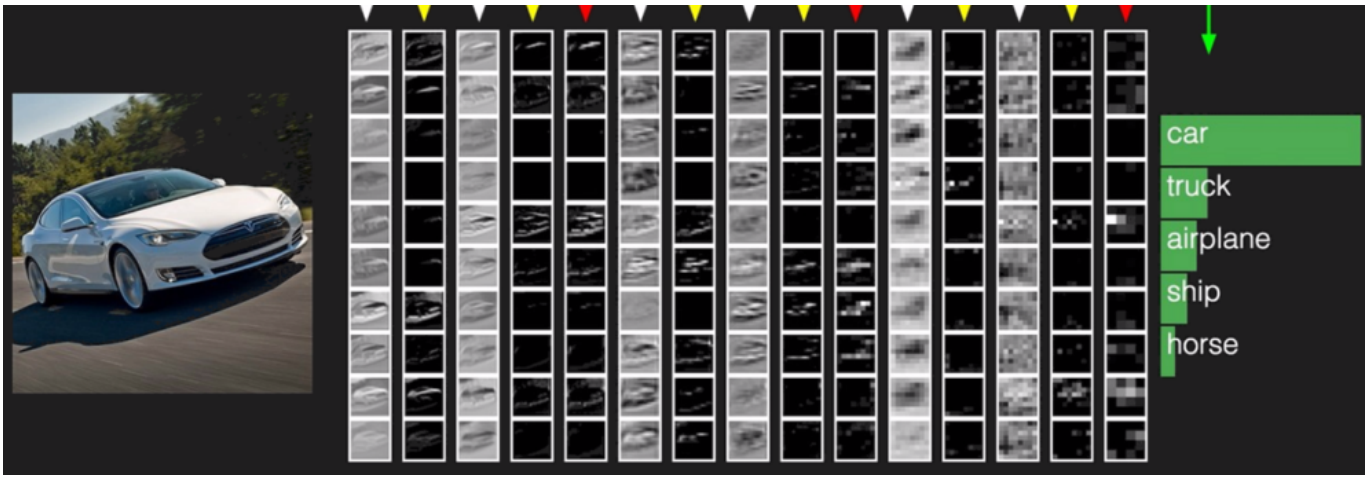
Pooling

You may often use max or average pooling between convolution dimensionality instead of varying the stride length. Pooling looks at a region, which lets you assume, is 2 x 2 and keeps only the largest or average value. The following image depicts a 2 x 2 matrix that depicts pooling:



A pooling region always has the same-sized stride as the pool size. This helps to avoid overlapping. Here's a relatively shallow **convolutional neural networks** (**CNN**) representation:





Source: cs231.github.io, MIT License

You can observe that the input image is subjected to various convolutions and pooling layers with ReLU activations between them before finally arriving at a traditionally fully connected network. The fully connected network, though not depicted in the diagram, is ultimately predicting the class.

In this example, as in most CNNs, you'll have multiple convolutions at each layer. Here, you'll observe 10, which are depicted as rows. Each of these 10 convolutions has their own kernels in each column so that different convolutions can be learned at each resolution. The fully connected layers on the right will determine which convolutions best identify the car or the truck, and so forth.

If you found this article, you can explore Dr. Joshua Eckroth's Python Artificial Intelligence Projects for Beginners to build smart applications by implementing real-world artificial intelligence projects. This book demonstrates AI projects in Python, covering modern techniques that make up the world of artificial intelligence.

Joshua Eckroth teaches big data mining and analytics, artificial intelligence (AI), and software engineering at Stetson University. He also has a PhD in AI and cognitive science, focusing on abductive reasoning and meta-reasoning.

Machine Learning Deep Learning Convolutional Network Neural Networks

Artificial Intelligence