Chapter 18
Convolutional Neural Networks

Advances in deep learning were primarily catapulted by image data analysis. For this reason, *Convolutional Neural Networks* play a dominant role in the developments of machine learning research.

Recall that a Neural Network has its **input, hidden, and output** layers. This is because learning in layers promotes the learning of individual **feature representations.** How well different features within a data are learned relies on our ability as ML developers to optimize hyperparameters. In this chapter we will transition into a more specific architecture type that optimizes learning on image data.

18.1 Images as Data

We know that input data for our machine learning models use vectors. How can we represent image data as vectors? If we were to flatten a two-dimensional image into a single stream of numbers such that we could capture it as a vector, we lose some key relationships between the pixels, such as the distance between pixels representing the same object. In addition to that, if we are working with colored images, a pixel in an image is generally defined by three values: Red, Green, and Blue. So, if a 200 by 200 colored image has 3 colored channels, we are dealing with a vector of 200*200*3=120 thousand features. That is quite a hefty bite to chew on for our model especially for such a small image, so we introduce the concept of **convolutions**.

18.2 Convolutions

Definition 18.1: Convolution

A **convolution** in neural network training is a reduction of the number of weights that the model needs to learn by summarizing the image into fewer pixels.

A convolution combines information about local pixels such that pixels close to each other in an image are "summarized" by a smaller set of pixels.

This "summarization" is done by "sliding" a **kernel** across an image to output a final product for each position of the kernel.

Definition 18.2: Kernel

A **kernel** in a convolution is an n x n matrix of numbers. In Figure 18.1, the kernal is the dark blue shaded region. Since the image is a 4x4 image and the kernel is a 3x3 matrix, the kernel can have 4 unique positions on the image. As a result, the output is a 4-pixel, or 2x2 image.

Each number inside the kernel matrix is multiplied with the value for each pixel it lines up with, and then all the elements of the kernel are summed to output one single value. Then, the kernel slides over to a new position in the image and the process is repeated.

The optimal numbers for a kernel matrix are learned by the model. To have the highest success rate, a model will learn values for a kernel that capture some sort of key information within an image, though that information is not directly interpretable by humans. Some examples of things that CNN developers have discovered kernels can extract are specific objects within an image, structural patterns, or dominant outlines in the image.

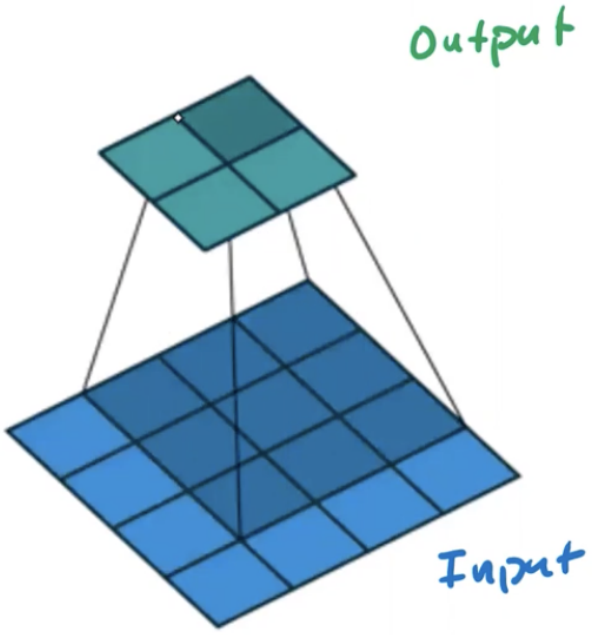


Figure 18.94: In the convolution above, the 4x4 image is summarized into a 2x2 image.

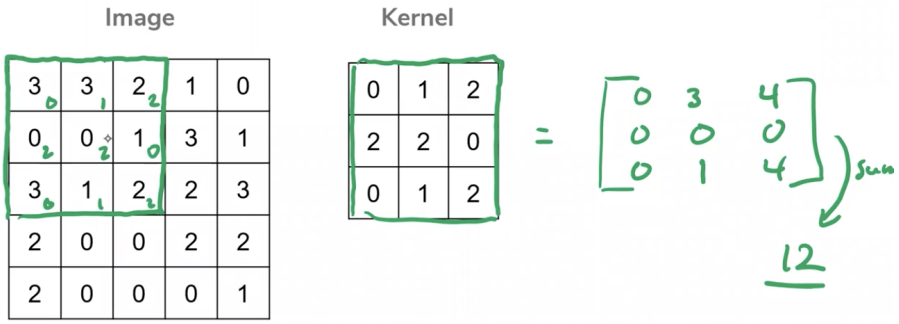


Figure 18.95: Calculations for applying a kernel to an image for one position.

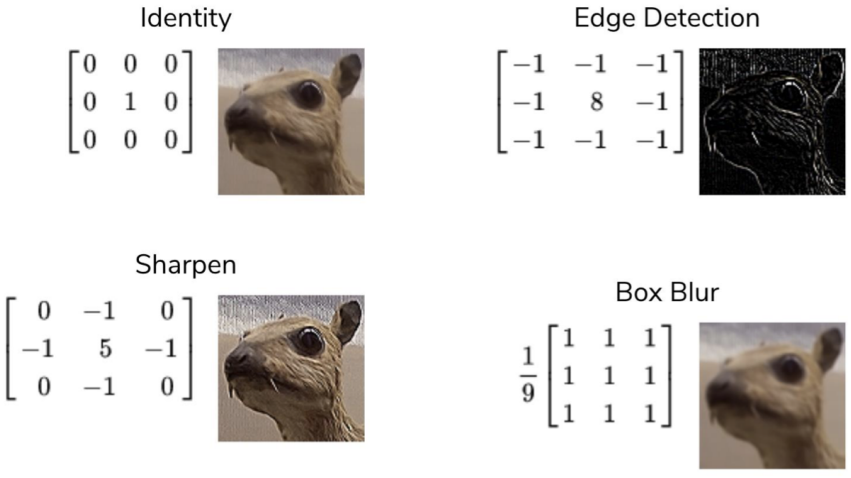
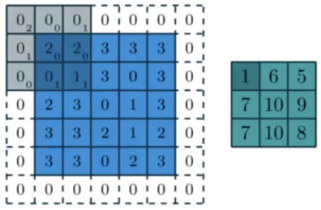
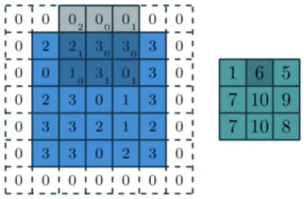


Figure 18.96: Special kernels which have specific properties such as maintaining the same image, detecting edges, sharpening the image, and applying a box blur



[First iteration]



[Second iteration]

Figure 18.97: The above shows a kernel of size 3x3 applied to a 5x5 image. The image has 1x1 padding. Notice that the second number in the output, 6, is the result of the kernel slidden over 2 pixels to the right across the image, so we know the stride is 2x2.

We can tune some specifications for our kernel such as **padding**: a layer of pixels of some static (unchanging) value wrapped around the image so that the kernel can align with the image more conveniently, **dimension**: how large the kernel should be, **stride**: how far the kernel should slide across the image each iteration.

18.3 Pooling

Pooling is another similar operation to convolutions done on an image. Sometimes, applying a convolution to an image doesn't reduce an image enough, so we can further utilizing pooling operations. Pooling is a lot more simple than applying a convolution in that we do not have a kernel full of elements. Instead, we have a filter that we slide across the image, and for every position of this filter, we simply take the minimum, maximum, mean, or median, of the set of pixels inside the image that fall underneath this filter. It is most typical to use a **max pool** with a 2x2 filter and a 2x2 stride, such that there is no overlap between filters. Using max pool has seemed to work better than the average pool.

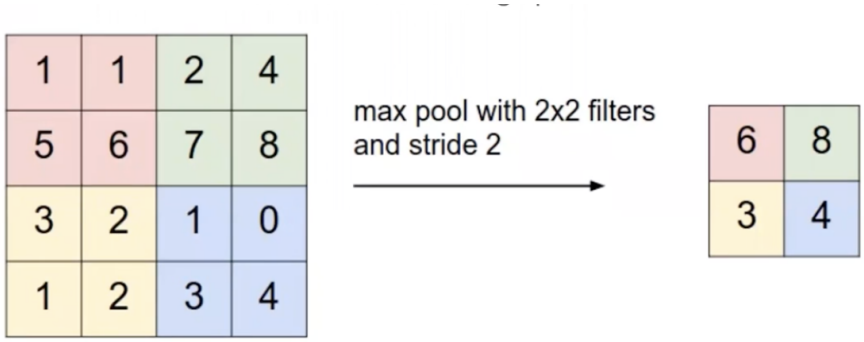


Figure 18.98: A maxpooling in action. You can think of each colored region as a position of the 2x2 filter.

18.4 Summary

All in all, we can think of a Convolutional Neural Network as a layered combination of these operations we have just learned. Take a look at this example CNN diagram below. Notice how after each operation, we end up with some dimensionality reduction. This dimensionality reduction helps us learn fewer weights since we have less data now, and the data is a holistic "summary" of the original input. Why? Because with convolutions and pools, we apply kernels and filters that slide across the entire image. So even though we have less data, the data is extracted from dispersed pieces of the image. So, the data, 'represents' the image

pretty well (even if we don't know exactly how it has chosen to represent it)–hence why machine learning developers will often use the terminology "representations".

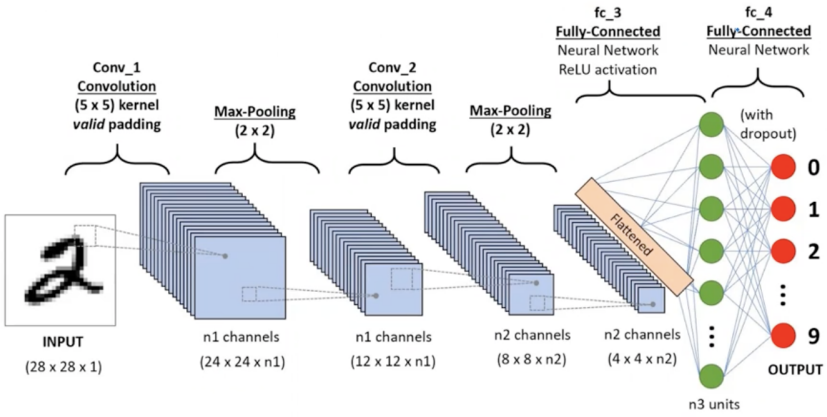


Figure 18.99: A high-level overview of a sample CNN.

This method is an extremely clever tactic that not only optimizes our deep learning algorithms **(efficiency)**, but also teaches the model fundamental groupings within a a two-dimensional image **(shift invariance)**. Outlines within an image are learned through these layers. Or, perhaps, the presence of two eyeballs of some animal. Or, it could be learning how to associate a bright yellow ball in the sky with the behavior of casting shadows on other objects. These are all relationships within a two-dimensional space that can be captured by convolutions since convolutions make use of distance within 2D space, but those same relationships might not be captured if we were to flatten the image into a vector. I mention these examples so that you may understand how effective convolutions and pools are at extracting **syntactical** meaning from an image. The sun, as an object, and the way it affects other objects in an image, should be learned. In other words, whether the sun is on the left side or the right side of the image shouldn't make our model think it's a totally different input. It should be able to recognize objects within a 2-dimensional space in a consistent manner even if those objects shift around spatially. This is what we mean by shift-invariance.

Due to the multitude of unknowns in our CNNs (and really, for all Neural Networks for that matter) deep learning is a very experimental field. We just can't really know how one model architecture will perform against another, so researchers test out many different architectures and combinations of hyperparameters when desigining their models.

18.4.1 A Kernel as a Neuron

An alternative way to think about convolutions in a CNN is by treating each kernel as a neuron. We can think of each kernel or each neuron responsible for learning some thing about the image. Depending on our task, we will have a specific amount of output neurons. Our ultimate goal is the optimize the weights for these output neurons. So, each of these output neurons will have learned a specific set weights pertaining to a specific attribute about the image (sharp edges, the sun, an eyeball) by having scanned the entire image.

18.4.2 General CNN Architecture

In general, the start of the CNN contains a series of Convolutions, Activation Functions, and Pooling layers. It's very common to do a pool after each convolution. After this series, the lower-dimension representation of the image is flattened to work with the final neural network.

18.4.3 Impacts

CNNs have proven to be extremely successful when it comes to image classification. The ImageNet challenge, an image classification competition, has seen slow and steady improvements in the task with standard neural networks up until 2012, when SuperVision, a convolutional network, superceded its competitors with a historical 17% error rate. Google's GoogLeNet achieved a 6.66% error rate in 2014. Since then, convolutional networks have been paving the way for unprecedented successes in image data analysis.

CNNs are important for many different industries in our society. For example, recognizing roads and buildings are important for self-driving car vision. CNN's aren't limited to image data. They have also proven to be groundbreaking tools in the fields of Natural Language Processing and Speech Recognition.

Of course, with all Machine Learning algorithms, CNNs have their limitations too. For one, they require an overwhelming amount of data. The ImageNet challenge provides competitors with 1.2 million images, but this is a relatively small dataset for neural network standards. They are also computationally expensive and even more expensive environmentally. It is hard to tune hyper-parameters as there are numerous choices to make such as the size of kernels, stride, 0 padding, number of convolutional layers, and the depth of outputs of convolutional layers. Finally, neural networks are extremely uninterpretable, which makes them an ethically unpopular choice for tools that directly impact humanity.

18.4.4 Transfer Learning

Transfer learning is a useful technique for neural networks. It is the act of using a pre-trained model for a new task. Since the early stages of a neural network learns high-level features that is usually not sensitive to the task at hand, one can chop off the first part of a neural network and tack on a new end to it such that those learned high level features are preserved.

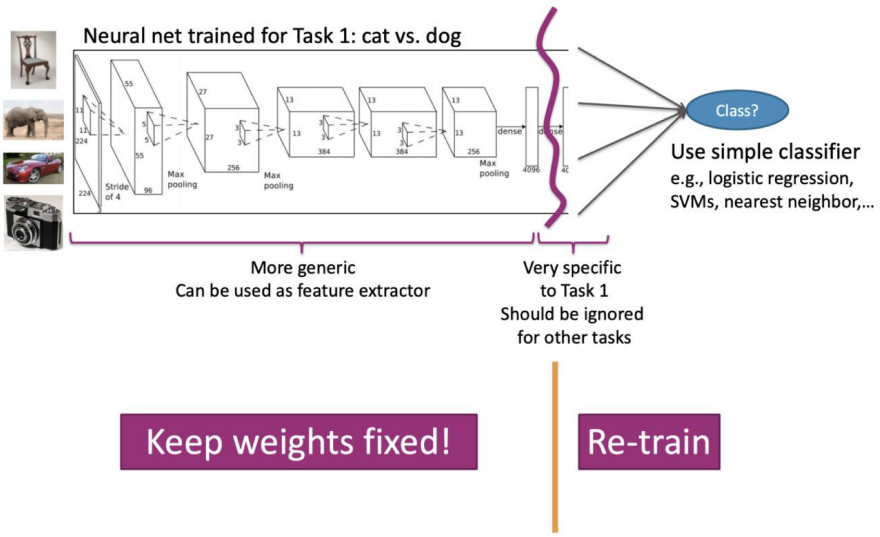


Figure 18.100: A transfer learning example using a neural network trained for differentiating between cats and dogs to help classify numerous other animals

18.4.5 Limitations

Neural Networks are easily decievable due to dataset bias. Previously, we discussed the idea of "shift invariance". Unfortunately for many neural networks, this is still a challenge. Simply moving an object's

position can easily confused a neural network that trained on recognizing an object in a specific position.