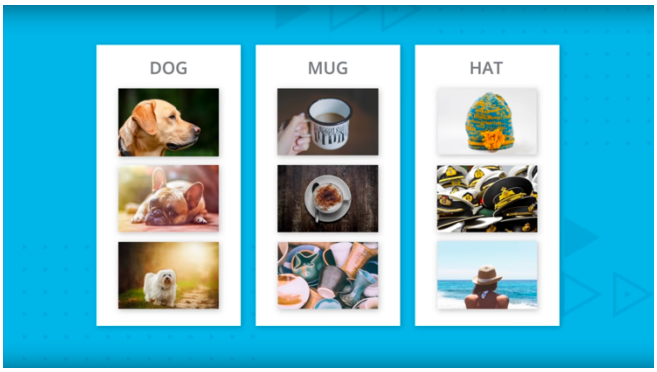
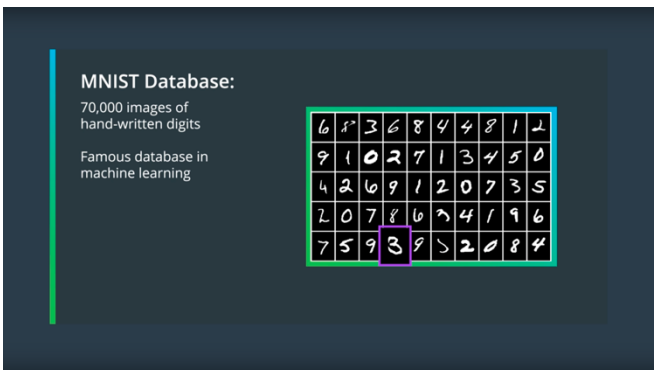
Chapter 1. Image classification

Computer vision *is an important subdomain of AI that seeks to mimic human visual system. The problem of* ***image classification*** *is one of the key tasks in respect to that:*



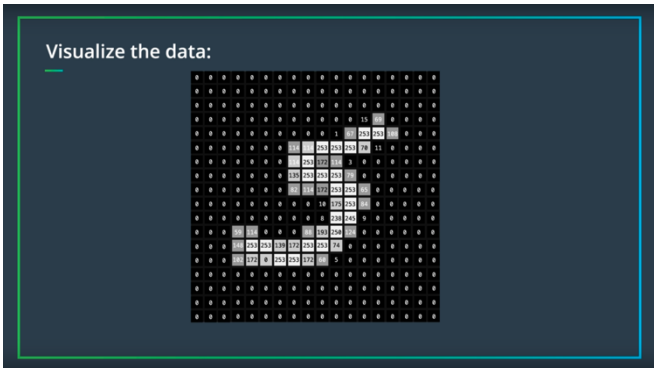
Chapter 2. MNIST database

MNIST database *is a 'hello world' of computer vision and general machine learning. It is a large database of images containing handwritten digits, and it was used to develop and demonstrate a lot of AI algorithms. All images are centered and fixed-size, therefore, the database can be used with minimal preprocessing (which is not usually the case in the real-world applications). However, some images (like number three in the picture) are ambiguous:*



Chapter 3. How computers interpret images

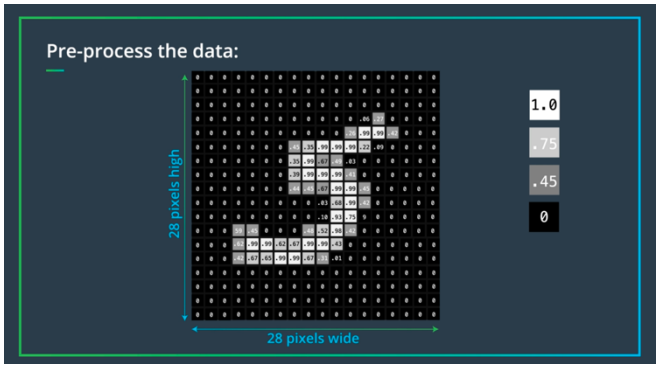
For a computer an image is just a bunch of numbers (pixel values), therefore, it is crucially important to think about what exactly will be fed into a neural network. Some ***visualization*** *is a good start:*



Chapter 4. Normalization

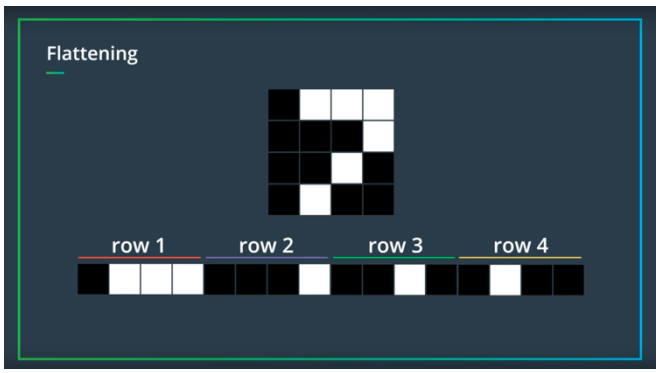
Normalization *is a preprocessing step that ensures that the data comes from the standard distribution: the range of pixel values in one input image are the same as the range in another image. Normalization is essential for fast and efficient training.*

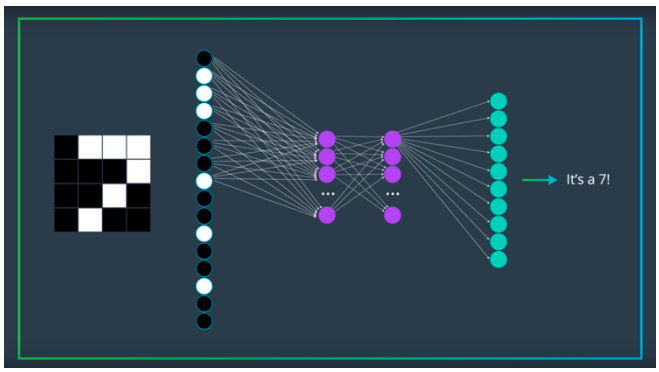
In the case of gray-scale images, every pixel is a number in the $[0, 256)$ *interval. Color images have three channels – red, green and blue, so every colored pixel is defined by three numbers in this interval, one for each channel. Before using these numbers in the training, we need to normalize them, so that every number is mapped onto the* $[0,1]$ *interval.*



Chapter 5. Flattening

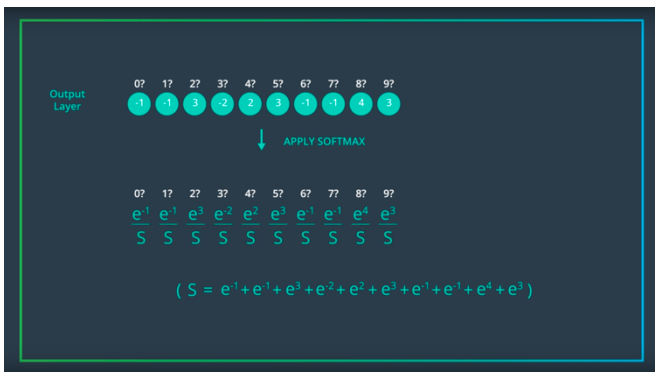
Flattening *is a way to represent image data so that we are able to actually feed this data into a simple neural network. The process involves taking a 2-d array (in the case of a gray-scale image) of pixel values, and flattening it into 1-d array by concatenating all the rows together. In the case of a MNIST image, which is 28 by 28, we get a vector of 784 elements.*

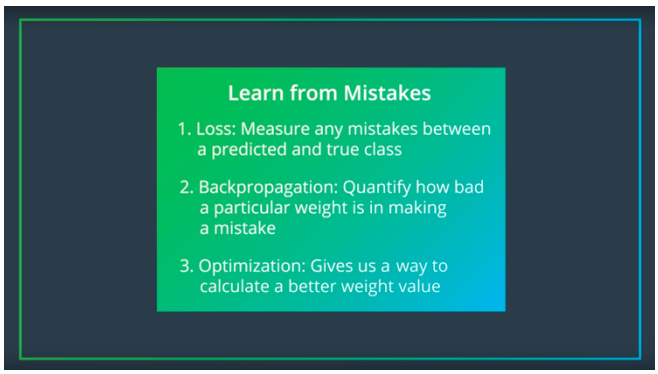


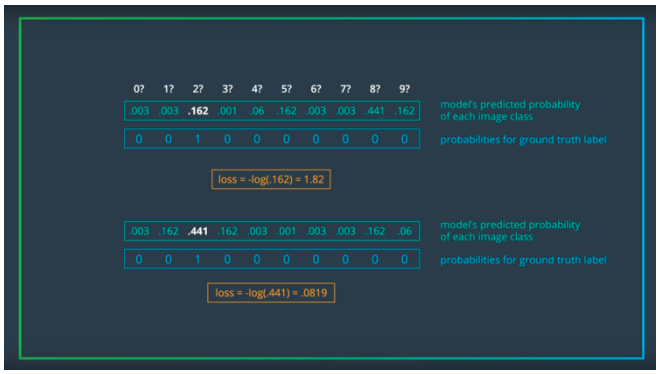


Chapter 6. MLP for image classification

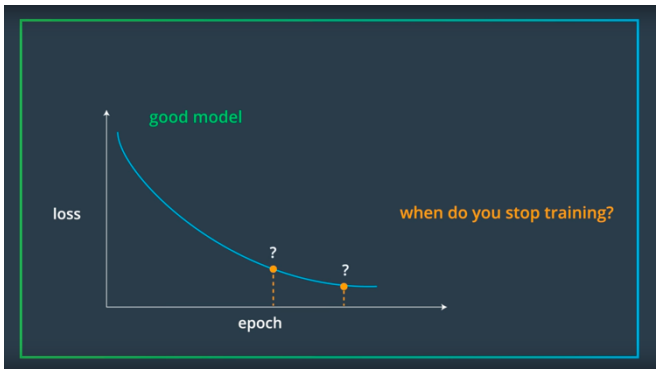
*Simple problems do not require complex solutions: MNIST classification can be tackled with just a simple **multilayer perceptron** (MLP). As an input it takes a flattened array of normalized pixel values, and then passes this input through a couple of fully-connected layers with the softmax function at the end. The last layer should have 10 neurons - one for every digit. This network has all the usual components: linear and nonlinear transformations, gradient descent, cross-entropy loss and softmax at the end.*



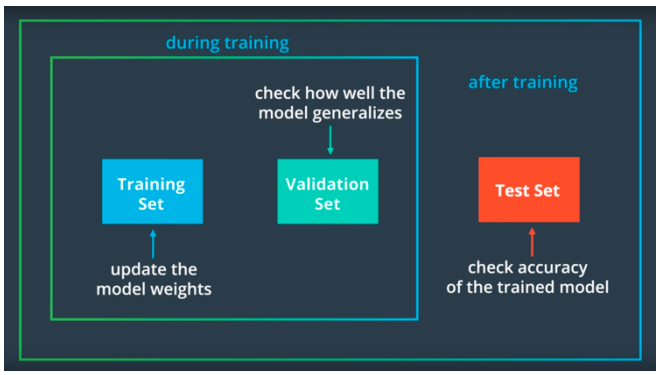




Chapter 7. Model validation

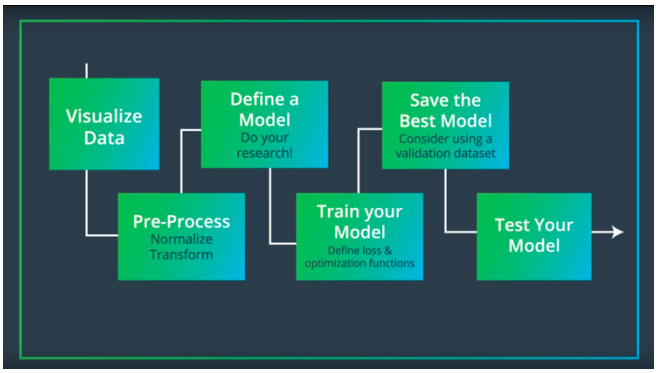


*In order to understand when is the best moment to stop the training is, we need to use two datasets: **training set** and **validation set**. Both are used in the actual training process, but the validation set is used only to see how well the model performs, the results of validation are not used to update the weights. Additional test set is required to estimate the quality of the model on completely new and unfamiliar data, because, even though the model cannot learn from the validation set, it still sees it during the training (and sometimes it is possible to overfit validation set). Therefore, it is important to have a separate **test set** which contains the truly unseen data.*



The best moment to stop the training is when both training and validation errors are small:

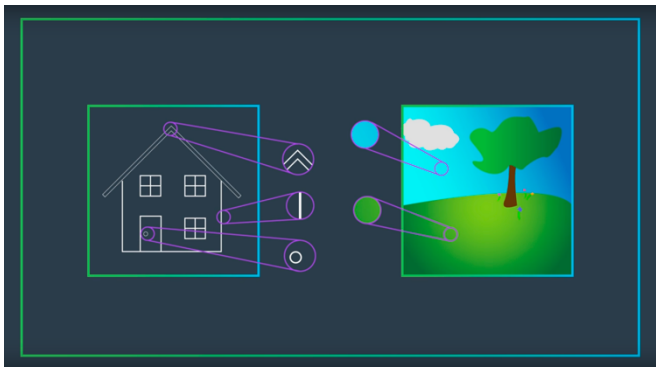
Chapter 8. General pipeline



Chapter 9. Convolutional neural networks

Flattening destroys all spatial information present in a picture. It is clear that pixels in one region of a picture (let's say upper left corner) are more closely related to each other than to the pixels in a different region (let's say upper right corner). However, flattened array does not reflect this at all, so one major drawback of using regular neural networks with input flattening is that this approach loses essential (to a human eye) spatial information and introduces unnecessary relations (every entry in the flattened vector is connected to every neuron in the next layer).

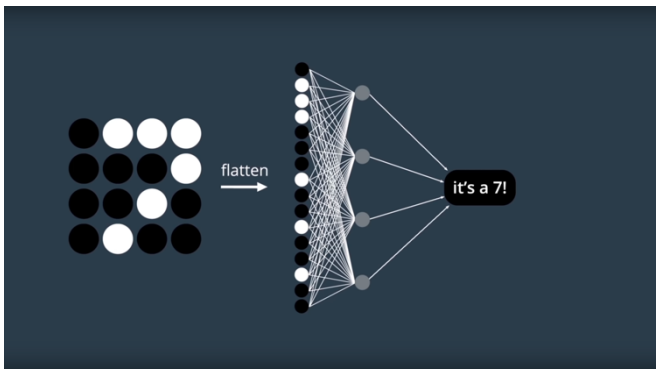
***Convolutional neural networks (CNN)** were designed to overcome exactly this. CNN look at the image as the whole and learn to identify spatial patterns such as prominent colors and shapes.*



The data in the MNIST database is relativity easy to handle, all images are centered and clean, but the real problems will not have such properties. Even though, MLP is able to classify MNIST pictures, it does not guarantee high performance in other tasks. CNN can deal with noisier images and solve more complex problems:

Chapter 10. Local connectivity

*The distinctive characteristic of MLP is **global connectivity**, which means that every neuron of the previous layer is connected to every neuron in the next layer (fully connected layers). However, it is not the only option.*

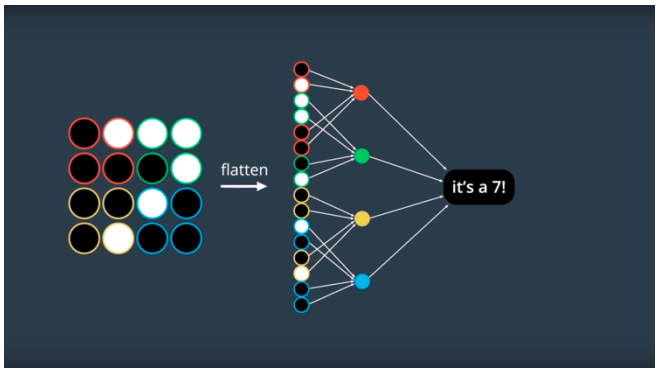


*In case of image processing, **local connectivity** makes more sense because of two reasons:*

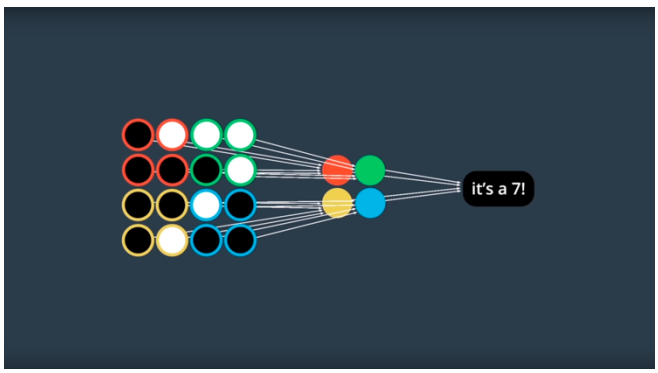
1. *An image is spatially organized, and it is important to consider where pixels are located in relation to each other.*

2. *Fully connected layers have a lot of parameters. In the case of 4×4 image flattened to a vector with 16 elements, and a hidden layer of 4 neurons (this configuration we see in the previous picture), we have 64 parameters to train. The number of parameters grows quickly with the size of the input and the number of hidden layers. Therefore, global connectivity introduces unnecessary computational complexity which can be (and should be) avoided.*

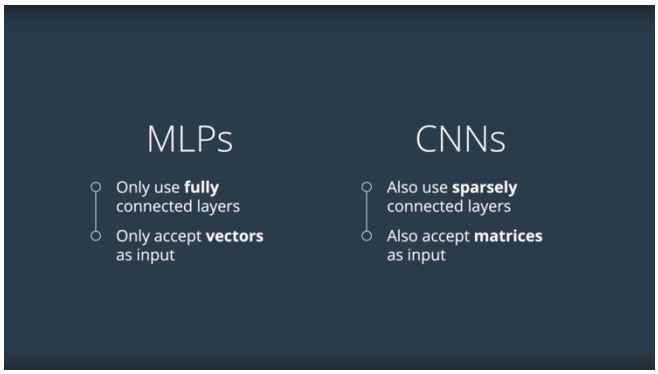
Using the previous example, we can split the image into four regions of four pixels each. In the picture below these regions are colored in red, green, yellow and blue. Each region is connected to only one neuron in the hidden layer, and, therefore, the total number of parameters is 16 instead of 64:



The same example with small rearrangements:

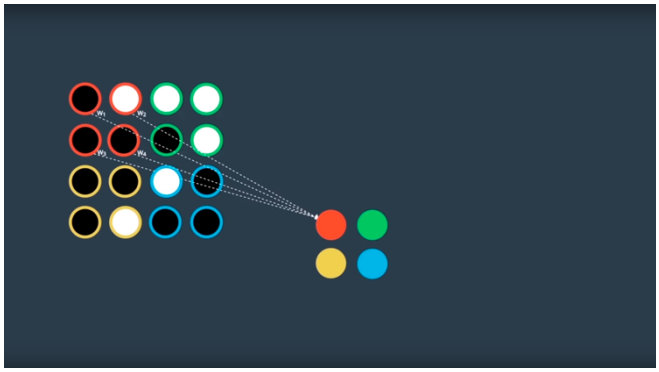


MLPs vs CNNs:

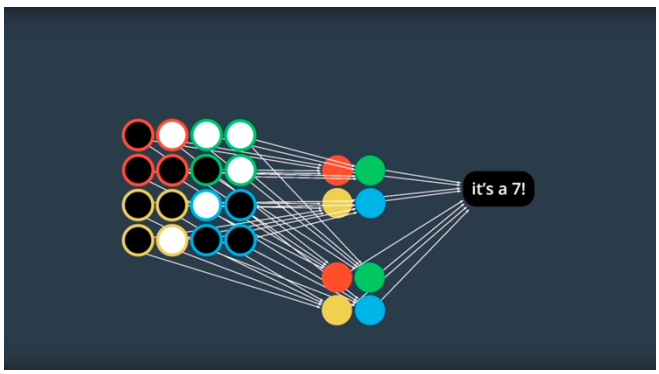


Chapter 11. Shared weights

*The concept of **shared weights** is crucial to CNNs. The network below has 4 nodes in the hidden layer, and each hidden node is connected to 4 input nodes via 4 weights. However, in an actual convolution layer those 4 weights will be shared between 4 hidden nodes: $[w_1, w_2, w_3, w_4]$ connecting the red node to the red region, the green node to the green region, etc. The total number of parameters in this convolutional layer is only 4 (a huge win comparing to initial 64). These weights $[w_1, w_2, w_3, w_4]$ constitute something known as **filter** (or **kernel**), which is an important concept in CNN theory.*

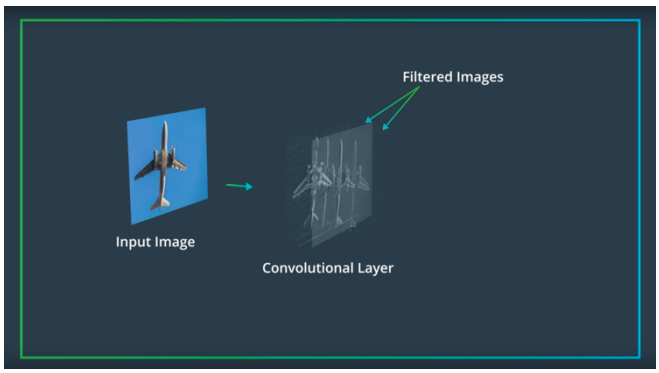


Usually there are multiple filters in one convolutional layer:

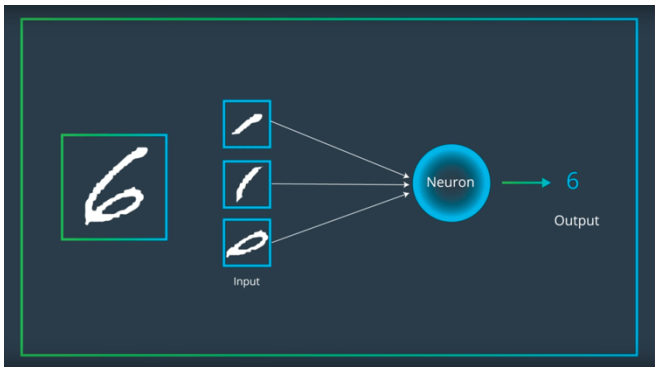


Chapter 12. Filters

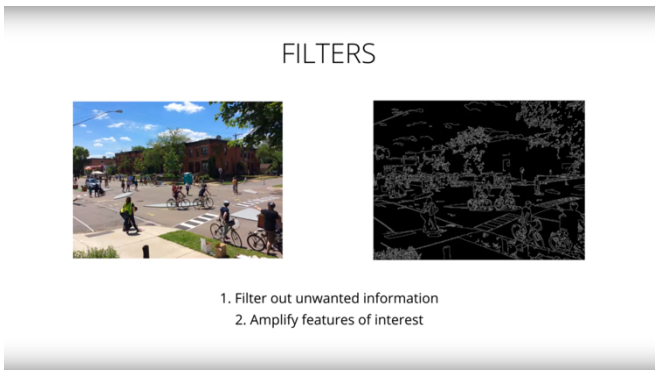
Convolutional layers preserve spatial information, they apply a series of filters (kernels) that extract different image features (for example, shape or color):



In the case of MNIST, CNN learns to identify different shapes that constitute a digit:



In image processing, filters are used to remove unwanted information, or amplify and extract specific features:



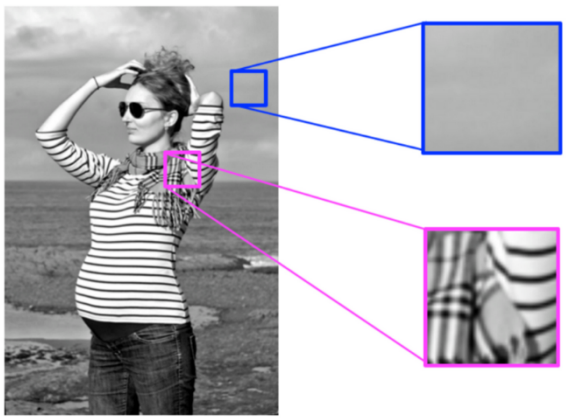
Different filters are designed for different tasks, for example, ***high-pass filters*** *notice regions where intensity rapidly changes (from bright to dark and vice versa) and sharpen the image.*

Naturally, high-pass filters are good for edge detection.

Chapter 13. Image frequency and edge detection

Shape identification is a key to classify an image which is, basically, to understand what kind of object is present in the picture. How do we recognize shapes? The first step is to clearly identify all edges which constitute the shape of an object. This is where the notion of image ***frequency*** *comes in handy.*

UDACITY: Frequency is a rate of change. A high frequency image is one where the intensity changes a lot. And the level of brightness changes quickly from one pixel to the next. A low frequency image may be one that is relatively uniform in brightness or changes very slowly.



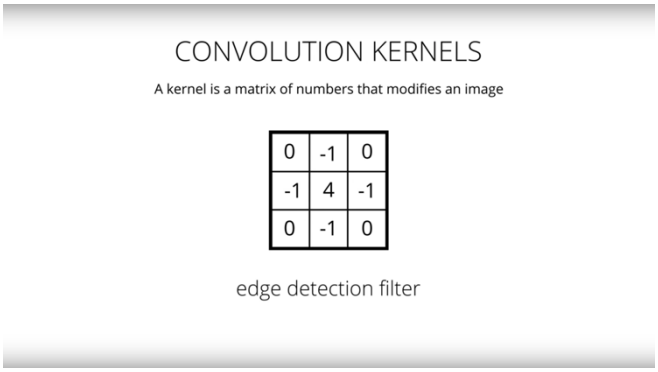
High and low frequency image patterns.

UDACITY: Most images have both high-frequency and low-frequency components. In the image above, on the scarf and striped shirt, we have a high-frequency image pattern; this part changes very rapidly from one brightness to another. Higher up in this same image, we see parts of the sky and background that change very gradually, which is considered a smooth, low-frequency pattern.

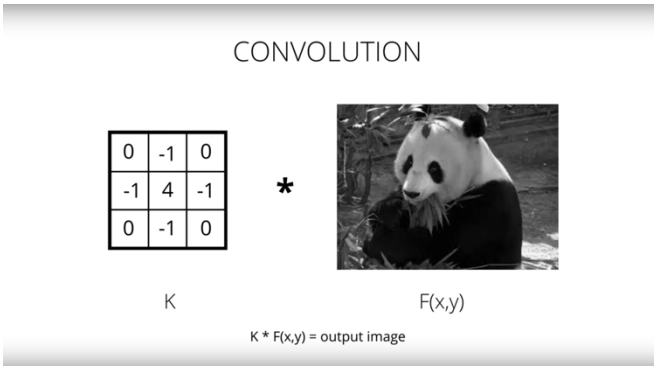
UDACITY: High-frequency components also correspond to the edges of objects in images, which can help us classify those objects.

Chapter 14. Convolutions

How do filters modify an image? With the help of convolution kernels.



An image is convolved with this kernel in order to get a new image with detected edges.

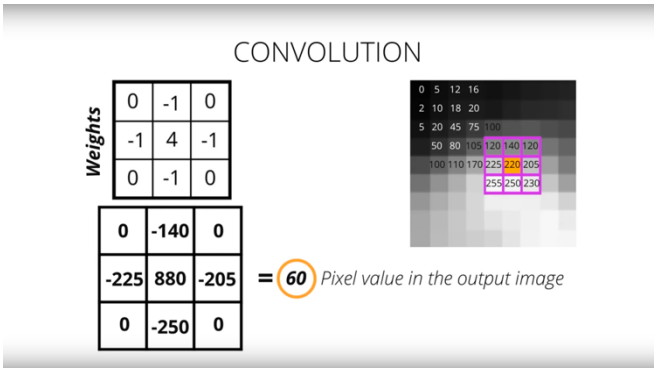


The kernel slides over an image pixel by pixel producing new pixel values. For example, in the picture below we want to compute a new value for the pixel whose current value is 220 (highlighted in orange). In order to do this, we need to multiply surrounding pixels with the kernel and take the sum:

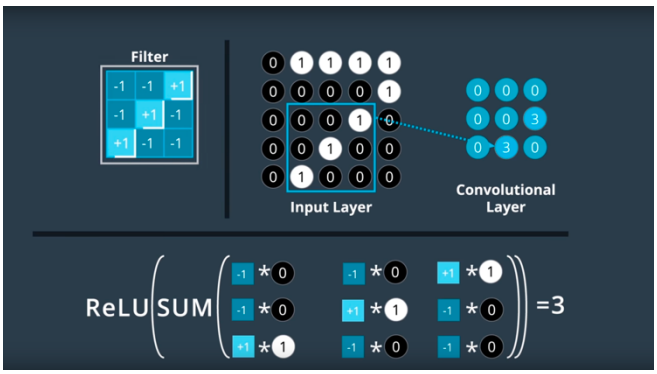
$$0 * 120 + (-1) * 140 + 0 * 120$$

$$+ (-1) * 225 + 4 * 220 + (-1) * 205$$

$$+ 0 * 255 + (-1) * 250 + 0 * 230 = 60$$

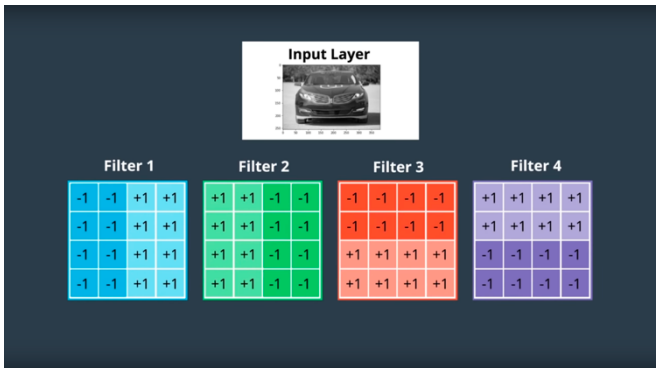


An actual CNN example:

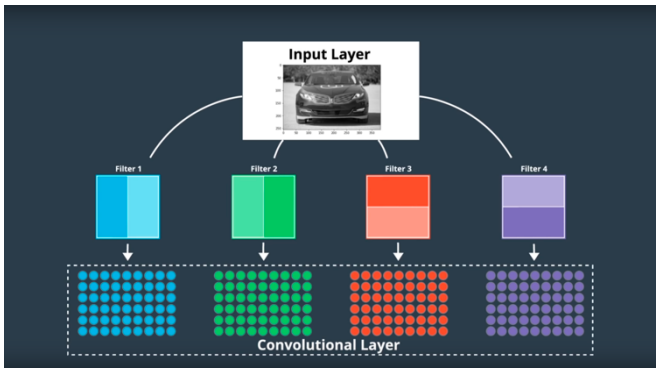


Chapter 15. Convolutional layer

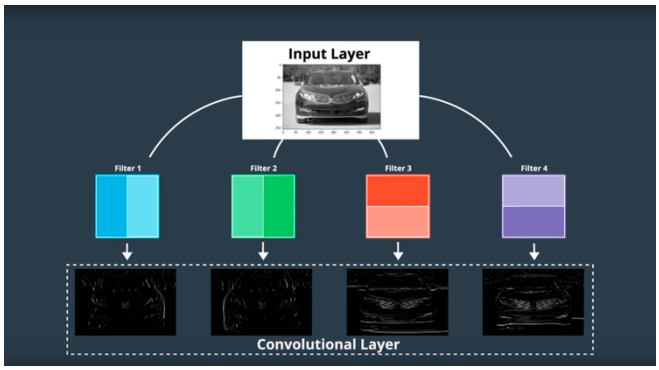
A convolutional layer consists of such kernels. In the example above all filters have been known beforehand, but the goal of CNN is to learn the best kernels (filters) for a given task. In the picture below there are four kernels (filters):

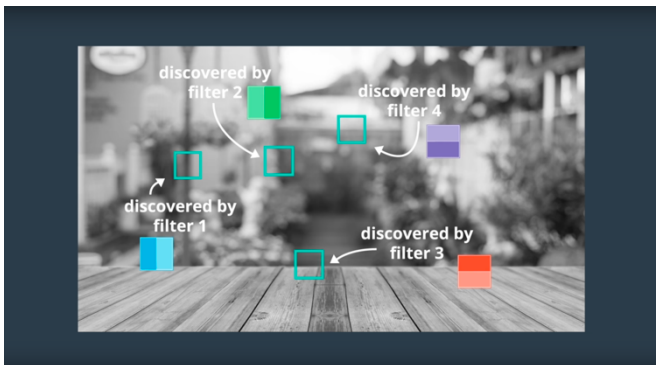


*Each filter is convolved with the original image of a car producing a **feature map** (activation map).*



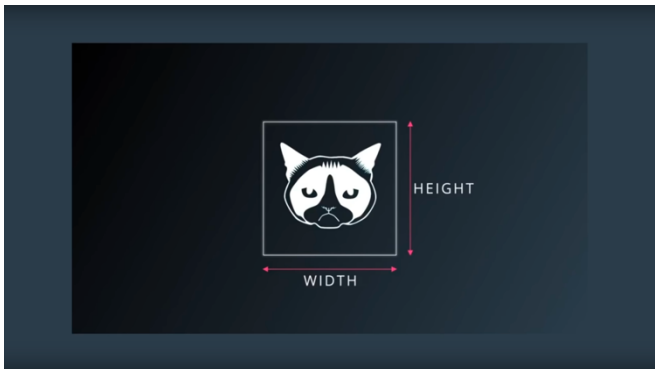
When visualized those feature maps look like filtered images:



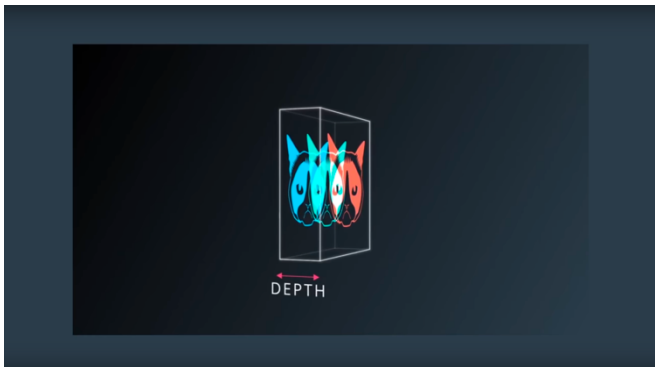


Chapter 16. CNN and colored images

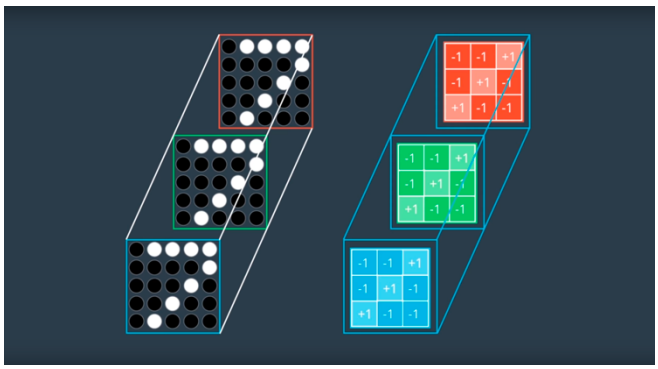
Gray-scale images can be represented as a 2-d array with width and height:



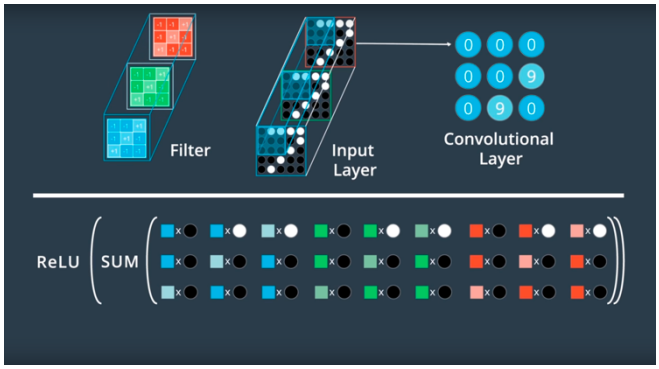
Color adds one more dimension – depth. In the case of RGB format the depths is equal to 3:



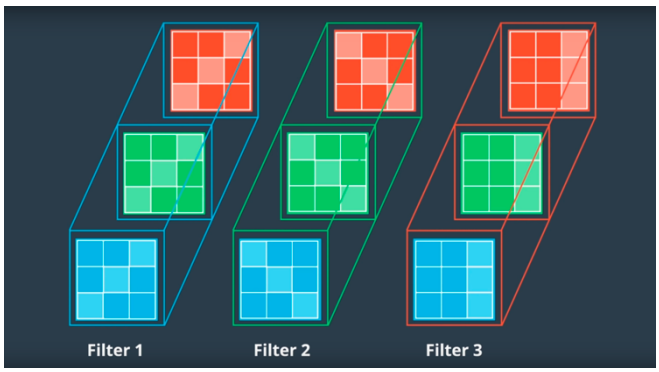
RGB color images can be visualized as three 2-d matrices stacked together. The filters, in this case, itself have 3 dimensions:



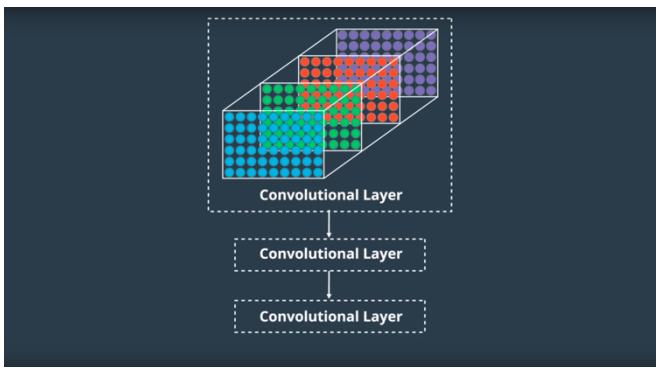
3-d convolutions are performed as usual. However, the final sum has three times more product terms:



Usually, there are multiple 3-d filters applied to an image:

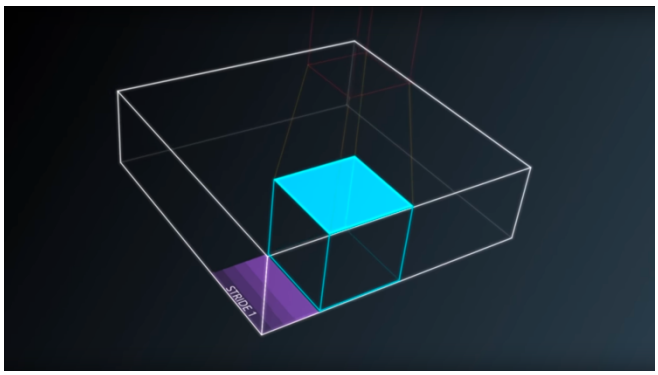


The feature maps produced by different filters are stacked together. The feature maps consist of just transformed pixel values and, therefore, they can be seen as images as well. Those new images can be fed into another convolutional layer to detect patterns within patterns:

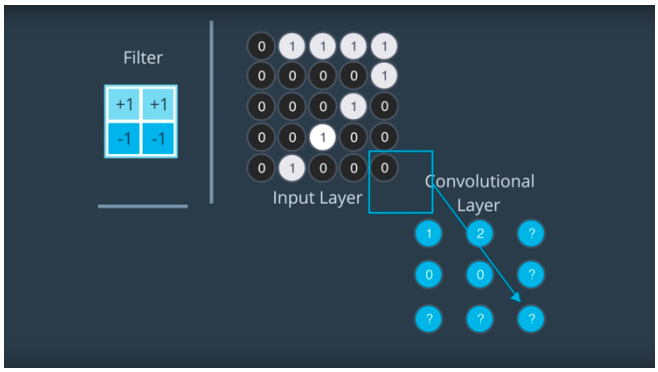


Chapter 17. Stride and padding

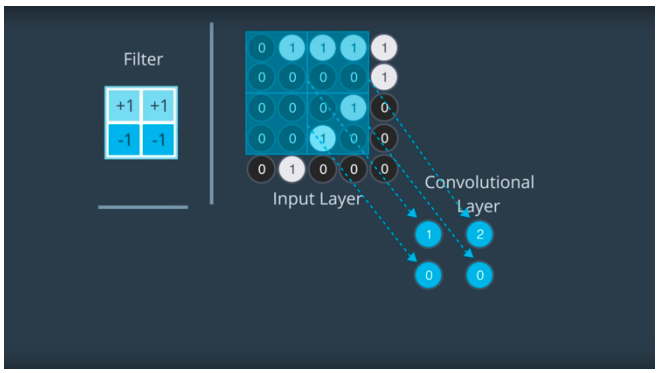
Stride *is an amount by which a filter slides over the image. In the examples above the stride was equal to 1, this means that the filter moved horizontally and vertically one pixel at a time. If stride is 1, the size of produced feature map is roughly the same as the size of the original image.*



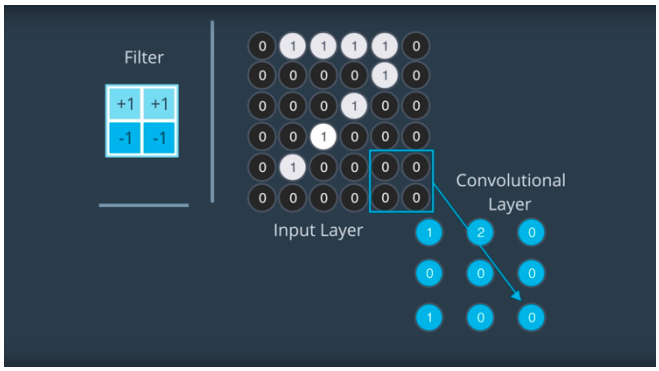
An important question is how to treat edges. Let's consider the example below with the stride equal to 2. It is unclear what to do with the edges, when there are not enough numbers to perform the full convolution, and the filter extends outside the image:



One option is to just get rid of the edges. In this case, the convolutional layer will lose some information about the picture:

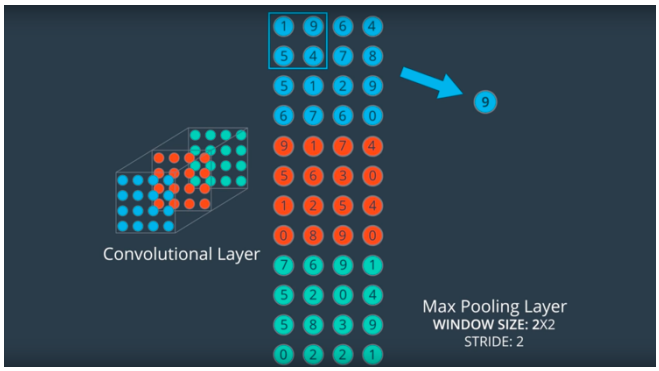


Another option is to do ***padding*** *with zeros, so there is enough space for the filter to slide around:*

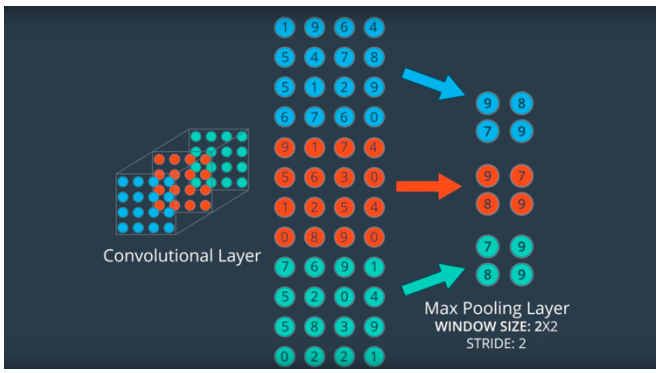


Chapter 18. Pooling

Pooling *is a technique used for dimensionality reduction. In order to perform the pooling operation a* ***stride*** *and a* ***kernel (window) size*** *need to be specified. There multiple variations, including average pooling and max pooling. In the case of the max pooling operation, there is a window sliding over an image (or a feature map) and outputting maximum value in that window:*

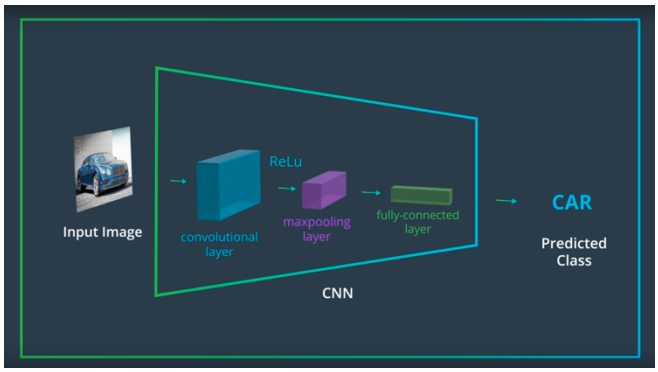


The configuration below allows to reduce the size of the feature map by a factor of 2:

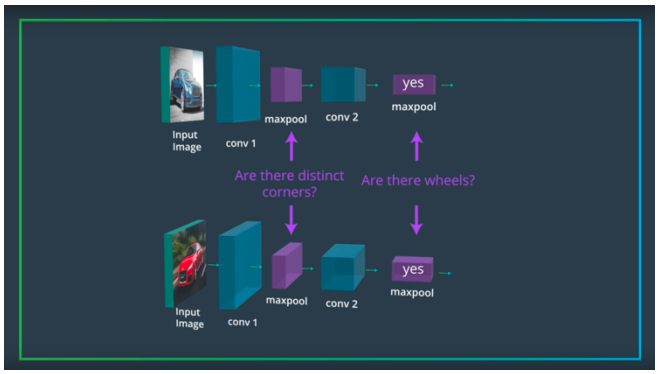


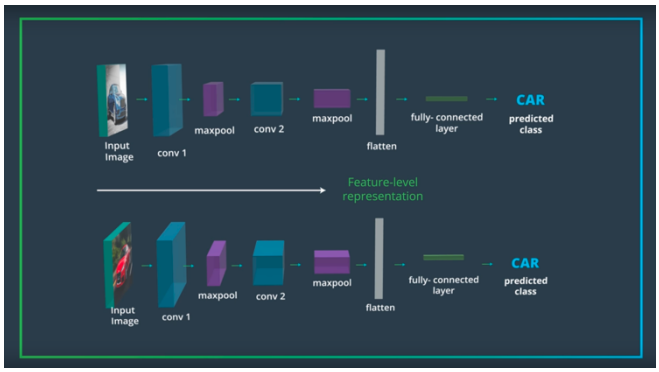
Chapter 19. CNN architecture and dataflow

*There are 3 major building blocks of CNN: **convolutional layer** with kernel, stride and padding specified, **pooling layer** with stride and kernel specified and **fully-connected layer**. The fully connected layer usually comes at the very end and takes as an input the last flattened feature map produced by the last convolutional layer.*



In the case of classification, we do not care about the image style. We want to learn the invariant representation of different objects. In the picture below, we see two different images of a car, and the desired output is just an indicator of whether there is a car in an image. Those two images have quite different styles; however, CNN will learn invariant representation of a car. Ideally the last convolutional layer will produce similar feature maps for both of the images.

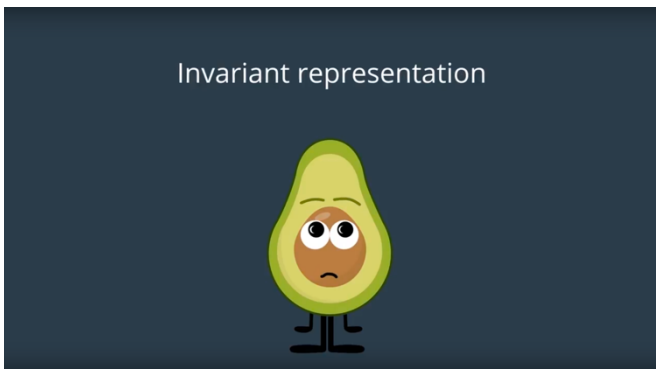


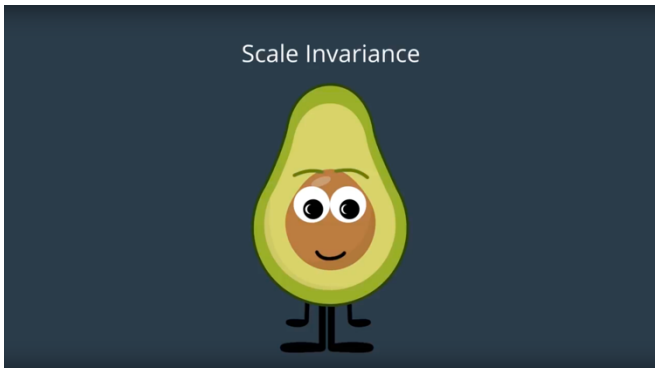


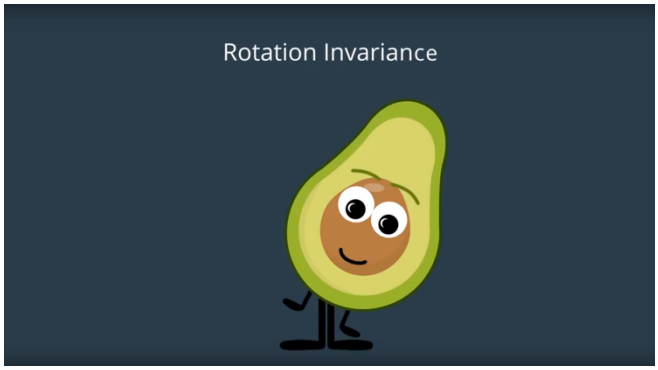
Chapter 20. Data augmentation

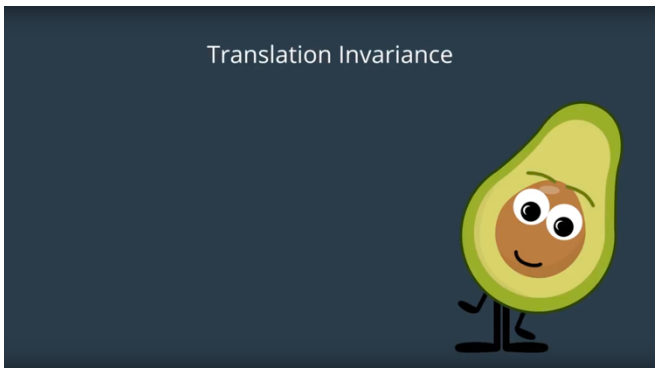
The goal of image classification is to learn an invariant representation of an object. Let's say, we want our CNN to tell us whether there is a cat in an image. It is not important whether this cat is small or big (scale invariance), whether the cat is in the upper corner of the picture or in the center of it (translation invariance), and whether the cat is upside-down (rotation invariance). In order to reflect this, we can augment the database with some rotated, scaled and translated images.

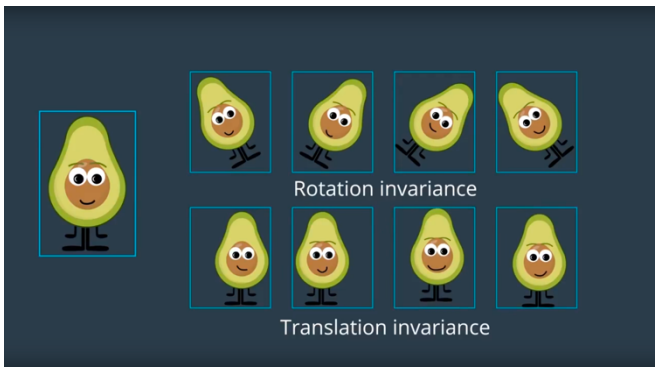
It is worth to mention that CNNs have some built-in translation invariance because of the shared weights concept. One feature map is produced by the same filter, and, therefore, if there are some target features available anywhere is the image, the filter will notice them.





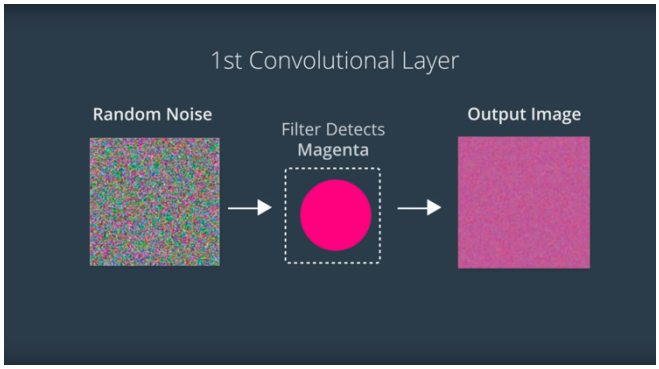


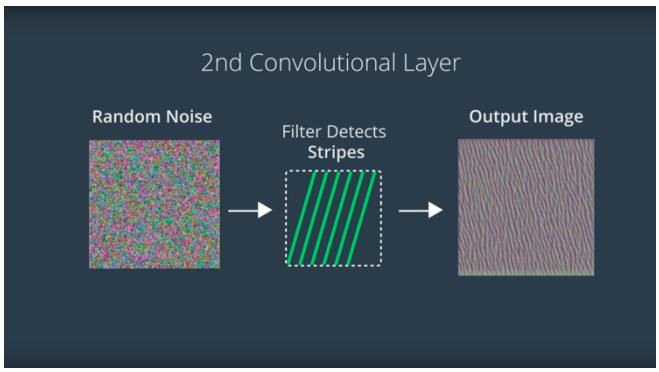


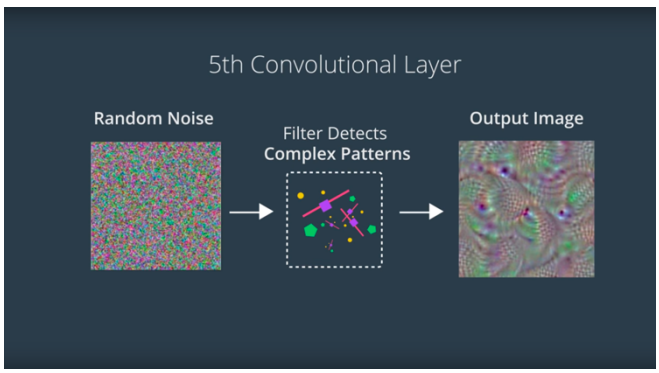


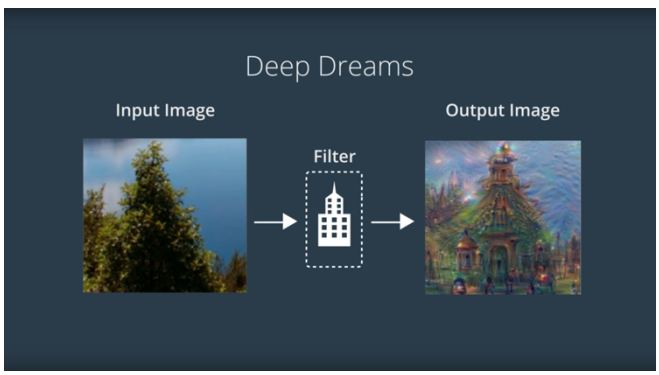
Chapter 21. Visualization

In order to understand what exactly a trained CNN does, it might be useful to take a picture with random noise and put it through network's convolutional layers visualizing the resulting feature maps:



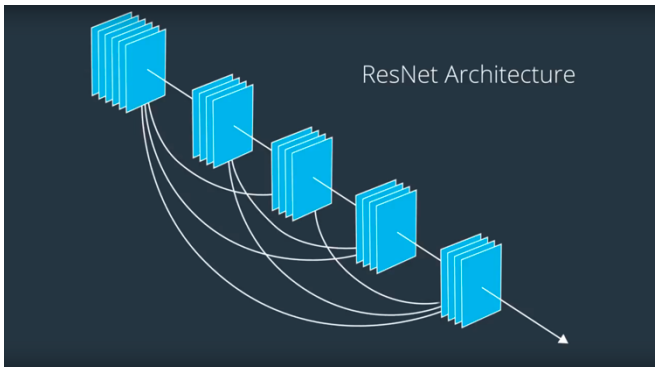


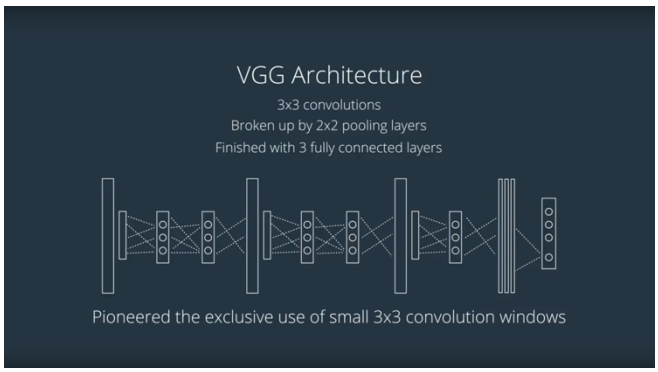


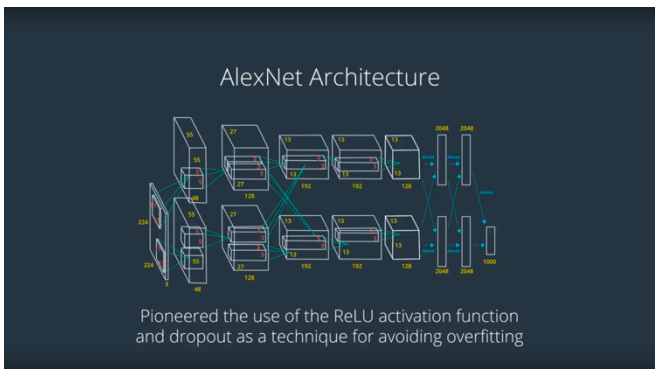


Chapter 22. Existing CNN architectures

There are also some pretrained CNNs available which can be used in transfer learning or as-is:







Author. *Tatiana Gaponova, tatiana.gaponova@gmail.com*

Course. *Intro to DL with PyTorch (Udacity)*