Computer vision

Computer vision is a field of study focused on the problem of helping computers to see.

At an abstract level, the goal of computer vision problems is to use the observed image data to infer something about the world.

— Page 83, Computer Vision: Models, Learning, and Inference, 2012.

Computer vision is the automated extraction of information from images. Information can mean anything from 3D models, camera position, object detection and recognition to grouping and searching image content.

— Page ix, Programming Computer Vision with Python, 2012.

The 2010 textbook on computer vision titled "Computer Vision: Algorithms and Applications" provides a list of some high-level problems where we have seen success with computer vision.

- *Optical character recognition (OCR)*
- *Machine inspection*
- *Retail (e.g. automated checkouts)*
- *3D model building (photogrammetry)*
- *Medical imaging*
- *Automotive safety*
- *Match move (e.g. merging CGI with live actors in movies)*
- *Motion capture (mocap)*
- *Surveillance*
- *Fingerprint recognition and biometrics*

Examples of a computer vision problems includes:

- Image classification.
- Object detection.
  - Detect object and localize them.
- Neural style transfer
  - Changes the style of an image using another image.

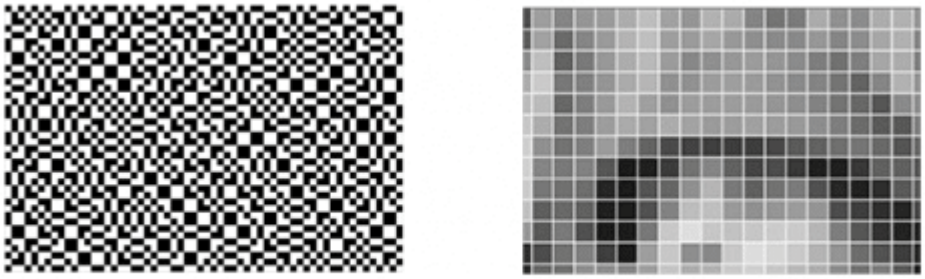
Many popular computer vision applications involve trying to recognize things in photographs; for example:

- **Object Classification**: What broad category of object is in this photograph?
- **Object Identification**: Which type of a given object is in this photograph?
- **Object Verification**: Is the object in the photograph?
- **Object Detection**: Where are the objects in the photograph?
- **Object Landmark Detection**: What are the key points for the object in the photograph?
- **Object Segmentation**: What pixels belong to the object in the image?
- **Object Recognition**: What objects are in this photograph and where are they?

Understanding image structure -

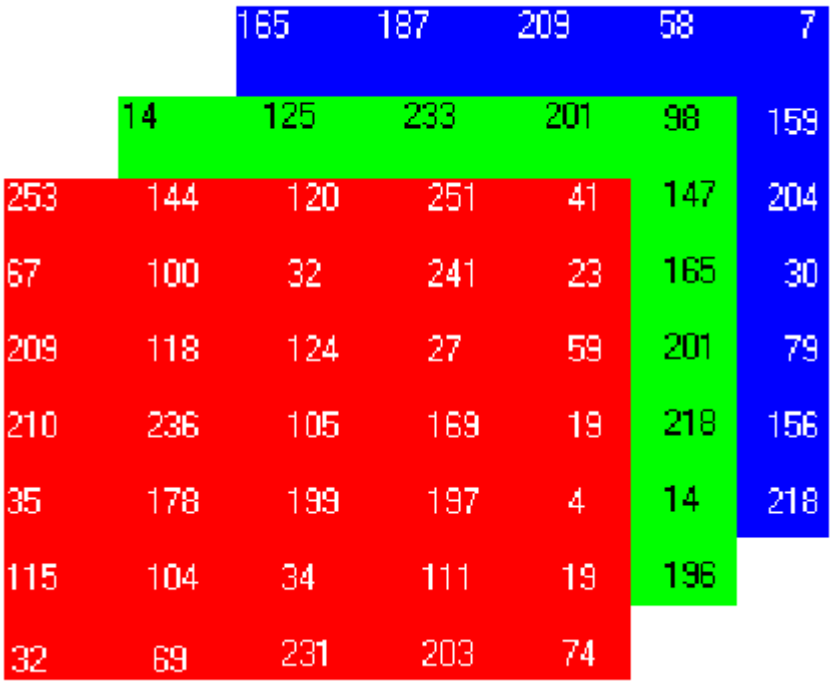
To understand the architecture of CNN, we must first know about the input which it takes. So, what is an image?

An image is a collection of a large number of squares called pixels. Pixels are the building blocks of image and are the one to decide color, contrast, brightness, and sharpness of the image. If we talk about black and white images, then we can take a pixel with value “1” for black color and a pixel with value “0” for white color. So ,in the image with these 2 types of a pixel, we will have an image like P(1). And if we will talk about a grayscale image, then there are not just two colors (0 and 1), but a range from 0-255, which is shown in P(2).



P(1) P(2)

But when we talk about coloured images, we basically have 3 layers of red, green and blue color, and in each layer, there is a range 0-255 for a pixel, where “0” is white and “255” is the base color and these three layers together form a colored image where pixel can be defined in rgb terms. Three layers with pixels ranging from 0-255 is shown below:



Why we use CNN over normal NN

One of the challenges of computer vision problem that images can be so large and we want a fast and accurate algorithm to work with that.

- For example, a 1000x1000 image will represent 3 million feature/input to the full connected neural network. If the following hidden layer contains 1000, then we will want to learn weights of the shape [1000, 3 million] which is 3 billion parameters only in the first layer and that is so computationally expensive!

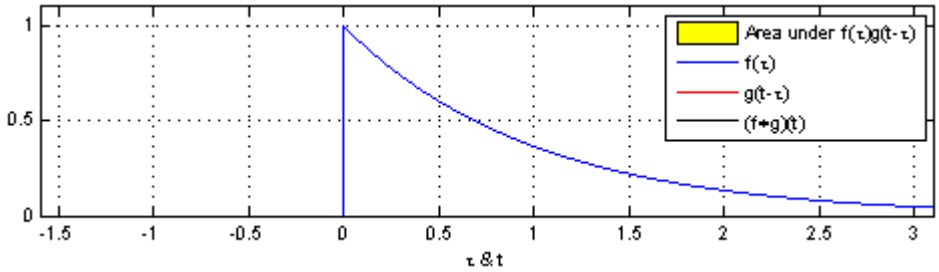
One of the solutions is to build this using convolution layers instead of the fully connected layers.

Types of layer in a convolutional network:

- Convolution. #Conv
- Pooling #Pool
- Fully connected #FC

What is a Convolution?

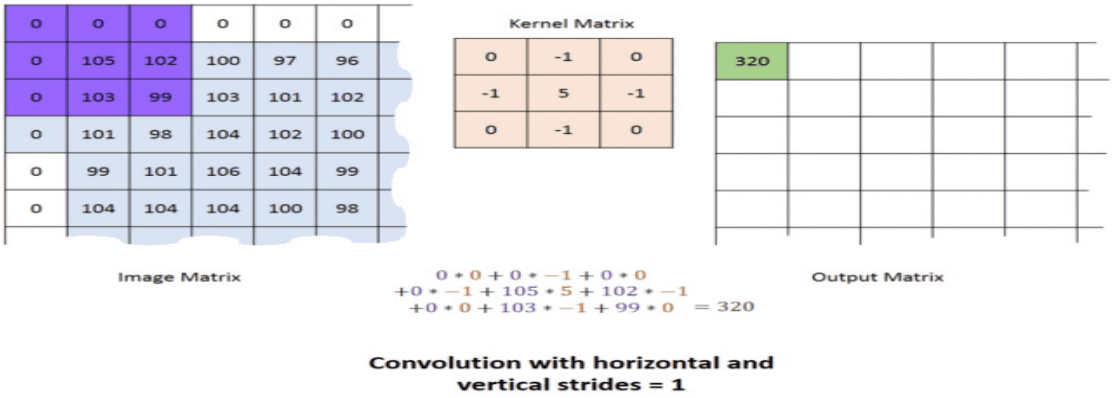
A convolution is a mathematical operation that combines two functions into a third one. For instance, let's say that we have two given functions, ***f(t)*** and ***g(t)***, and we are interested in applying one on top of the other one and calculate the area of intersection: ***f(t) * g(t) = (f*g)(t)***.



Source: Brian Amberg

In this case, we are applying ***g(t)*** (called kernel) over ***f(t)*** and changing the response ***(f*g)(t)*** according to the intersection over the area of both functions. This concept of convolution is the most used technique in Signal Processing, thus it is also applied in Computer Vision, which can be seen as processing the signal of multiple RGB sensors. The intuition behind the effectiveness of convolutions is their ability to filter given input signals into a combined, more useful result.

In the particular case of images, the signal is better understood in terms of matrices rather than waveforms. Therefore, our functions ***f(t)*** and **g(t)** will now become ***image(matrix)*** and ***kernel(matrix)***, respectively. Once again, we can apply convolution by sliding one of the matrices on top of the other one:



Source: Machine Learning Guru

Convolutional Layer

In a convolutional neural network, a convolutional layer is responsible for the systematic application of one or more filters to an input.

The multiplication of the filter to the input image results in a single output. The input is typically three-dimensional images (e.g. rows, columns and channels), and in turn, the filters are also three-dimensional with the same number of channels and fewer rows and columns than the input image. As such, the filter is repeatedly applied to each part of the input image, resulting in a two-dimensional output map of activations, called a feature map.

Keras provides an implementation of the convolutional layer called a Conv2D.

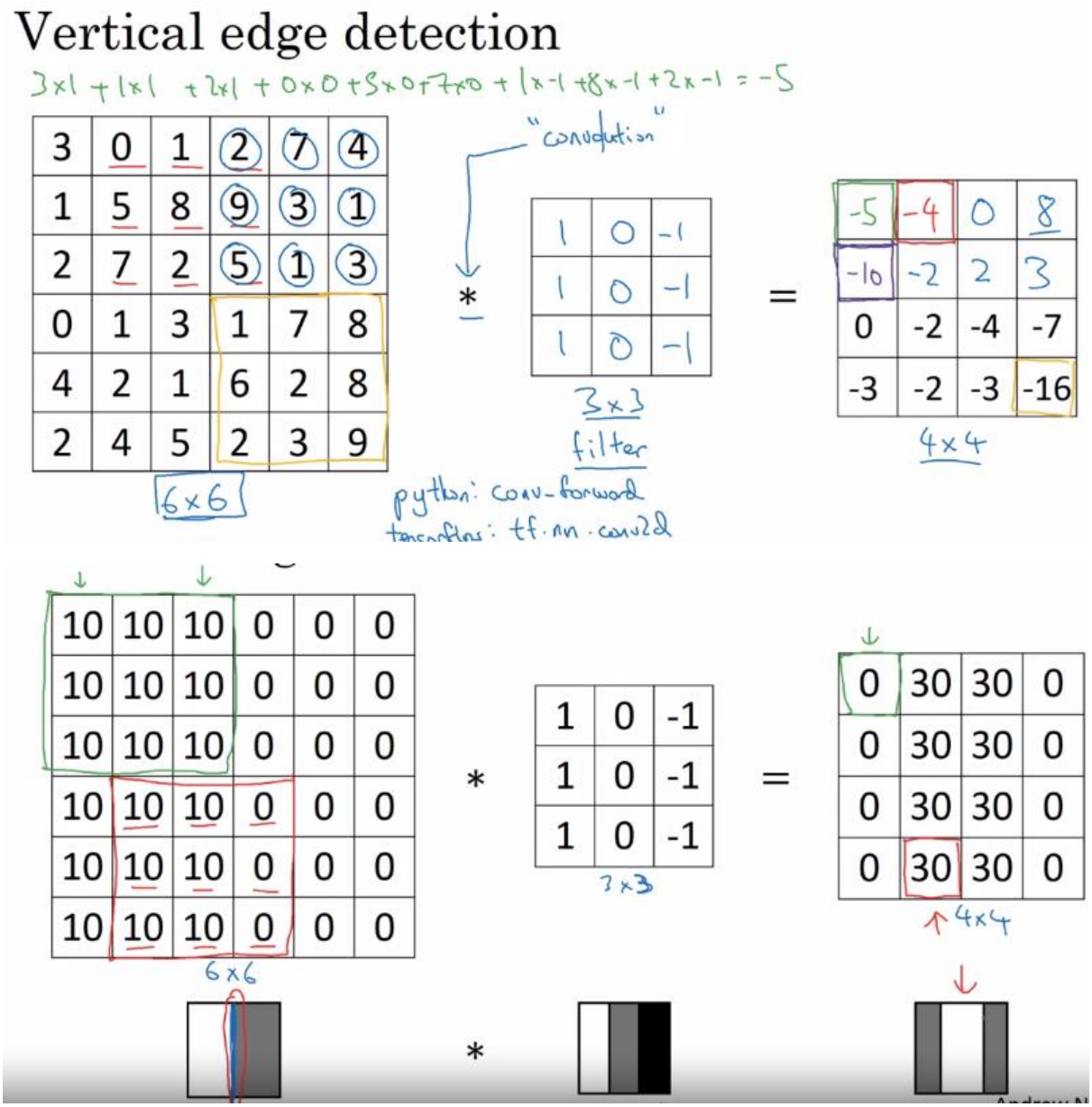
It requires that you specify the expected shape of the input images in terms of rows (height), columns (width), and channels (depth) or *[rows, columns, channels]*.

Edge detection –

The convolution operation is one of the fundamentals blocks of a CNN. One of the examples about convolution is the image edge detection operation.

We use a normal grey scale image(n*m*1) and apply filter on that using convolution operation. The filter moves over whole input image and when on a particular block it computes element-wise multiplication and addition over pixel values.

Vertical edge detection:



As per image we take a 6*6 image and convolve with a 3*3 filter which has pixels showing transition from brighter(0) to darker pixels(-1). The result is 4*4 image with highlighting egde.

In above image, the last 4*4 matrix shows white vertical edge detected but since the image is input 6*6 it comes a bit wider.

The dimension 4*4 came out from simple rule –

N - f + 1 where N = 6, f(filter) = 3

More Edge Detection

The type of filter that we choose helps to detect the vertical or horizontal edges. We can use the following filters to detect different edges:

1	0	-1
1	0	-1
1	0	-1

Vertical

1	1	1
0	0	0
-1	-1	-1

Horizontal

Some of the commonly used filters are:

1	0	-1
2	0	-2
1	0	-1

Sobel filter

3	0	-3
10	0	-10
3	0	-3

Scharr filter

The Sobel filter puts a little bit more weight on the central pixels. Instead of using these filters, we can create our own as well and treat them as a parameter which the model will learn using backpropagation.

Padding

Padding add another layer of pixels to original image thus the original image size remains

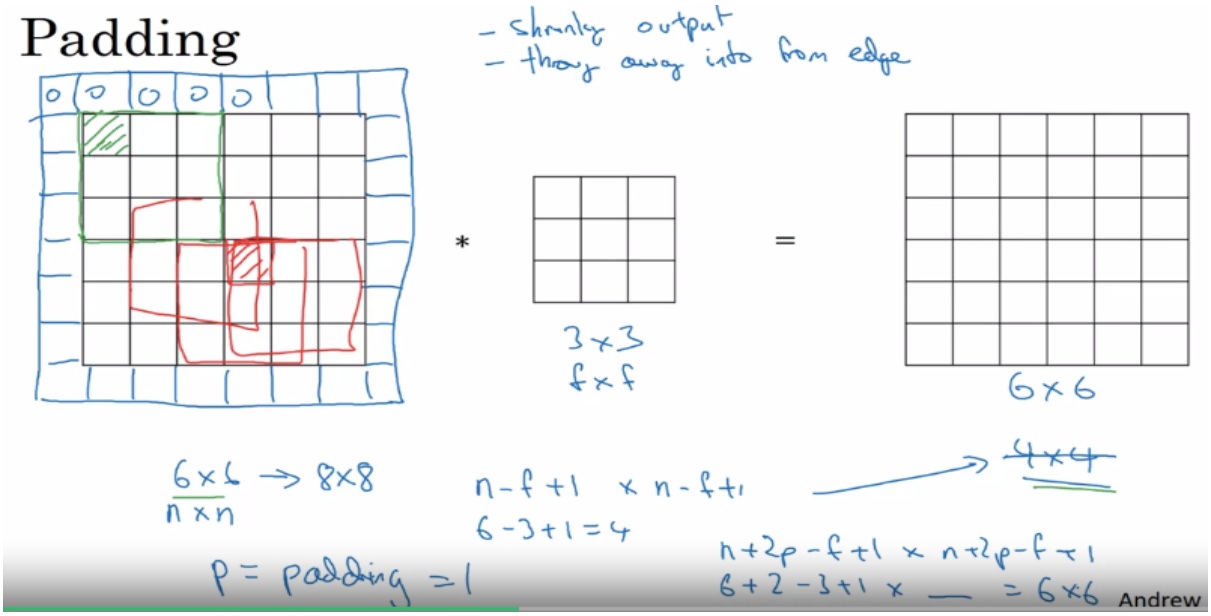
The main benefits of padding are the following:

- It allows you to use a CONV layer without necessarily shrinking the height and width of the volumes. This is important for building deeper networks, since otherwise the height/width would shrink as you go to deeper layers. An important special case is the "same" convolution, in which the height/width is exactly preserved after one layer.
- It helps us keep more of the information at the border of an image. Without padding, very few values at the next layer would be affected by pixels as the edges of an image.

We can pad the input image before convolution by adding some rows and columns to it. We will call the padding amount P the number of row/columns that we will insert in top, bottom, left and right of the image. In almost all the cases the padding values are zeros.

The general rule now, if a matrix nxn is convolved with fxf filter/kernel and padding p give us

n+2p-f+1,n+2p-f+1matrix.



P is padding amount and now the formula for output image dim becomes

n(input image)+2p(padding) – f(filter) + 1

here f is usually odd. Filter dimensions are always odd.

Same convolutions is a convolution with a pad so that output size is the same as the input size. It's given by the equation: PADDING

p = f – 1 /2

In **Keras**, this is specified via the "*padding*" argument on the Conv2D layer, which has the default value of '*valid*' (no padding). This means that the filter is applied only to valid ways to the input.

The '*padding*' value of '*same*' calculates and adds the padding required to the input image (or feature map) to ensure that the output has the same shape as the input.

The example below adds padding to the convolutional layer in our worked example.

```
# example a convolutional layer with padding
from keras.models import Sequential
from keras.layers import Conv2D
# create model
model = Sequential()
model.add(Conv2D(1, (3,3), padding='same', input_shape=(8, 8, 1)))
# summarize model
model.summary()
```

padding: One of `"valid"`, `"causal"` or `"same"` (case-insensitive). `"valid"` means "no padding". `"same"` results in padding the input such that the output has the same length as

the original input. `"causal"` results in causal (dilated) convolutions, e.g. `output[t]` does not depend on `input[t + 1:]`. A zero padding is used such that the output has the same length as the original input. Useful when modeling temporal data where the model should not violate the temporal order

Look for other parameters in keras doc below

https://keras.io/layers/convolutional/

Strides

Stride specifies how much we move the convolution filter at each step. By default the value is 1

- Now the general rule is :
    - if a matrix n x n is convolved with f x f filter/kernel and padding p and strides it give us (n+2p-f)/s + 1,(n+2p-f)/s + 1 matrix.
- In case (n+2p-f)/s + 1 is fraction we can take floor of this value.

.

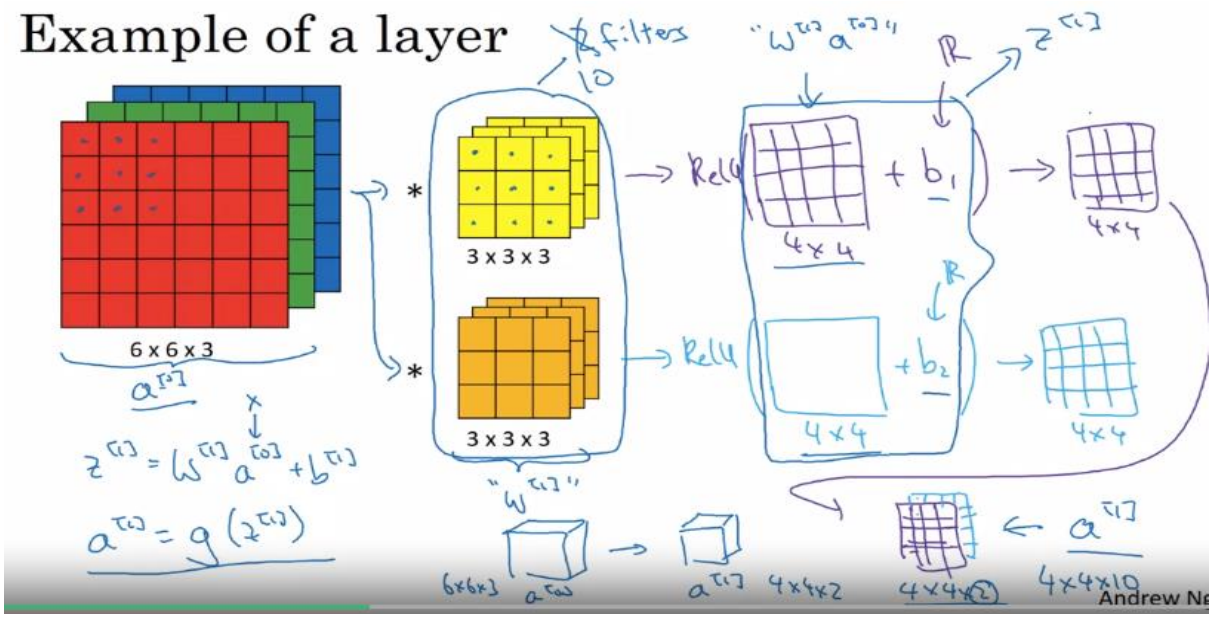
Summary of convolutions

$n \times n$ image $\quad f \times f$ filter

padding p $\quad$ stride s

$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$$

One Layer CNN



Summary of notation

If layer l is a convolution layer:

$f^{[l]}$ = filter size

$p^{[l]}$ = padding

$s^{[l]}$ = stride

$n_c^{[l]}$ = number of filters

→ Each filter is: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$

Activations: $a^{[l]} \rightarrow n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

Weights: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$

bias: $n_c^{[l]} - (1,1,1,n_c^{[l]})$ ← #filters in layer l.

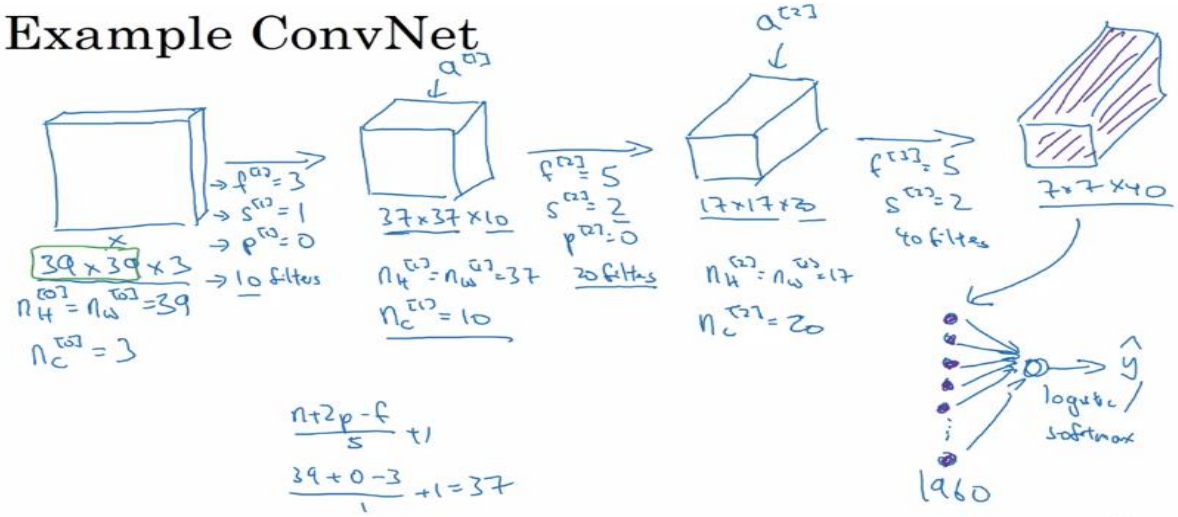
Input: $n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]}$

Output: $n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

$$n_{H,W}^{[l]} = \left\lfloor \frac{n_{H,W}^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$$

$A^{[l]} \rightarrow m \times n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

Example ConvNet



One Layer of a Convolutional Network

- First we convolve some filters to a given input and then add a bias to each filter output and then get RELU of the result. Example:
    - Input image: 6x6x3 # a0
    - 10 Filters: 3x3x3 #W1
    - Result image: 4x4x10 #W1a0
    - Add b (bias) with 10x1 will get us : 4x4x10 image #W1a0 + b
    - Apply RELU will get us: 4x4x10 image #A1 = RELU(W1a0 + b)
    - In the last result p=0, s=1
    - Hint number of parameters here are: (3x3x3x10) + 10 = 280
- The last example forms a layer in the CNN.
- Hint: no matter the size of the input, the number of the parameters is same if filter size is same. That makes it less prone to overfitting.
- Here are some notations we will use. If layer l is a conv layer:

```
Hyperparameters
f[l] = filter size
p[l] = padding      # Default is zero
s[l] = stride
nc[l] = number of filters

Input:  n[l-1] x n[l-1] x nc[l-1]        Or          nH[l-1] x nW[l-1] x nc[l-1]
Output: n[l] x n[l] x nc[l]   Or          nH[l] x nW[l] x nc[l]
Where n[l] = (n[l-1] + 2p[l] - f[l] / s[l]) + 1

Each filter is: f[l] x f[l] x nc[l-1]

Activations: a[l] is nH[l] x nW[l] x nc[l]
                A[l] is m x nH[l] x nW[l] x nc[l]    # In batch or minbatch training

Weights: f[l] * f[l] * nc[l-1] * nc[l]
    bias:  (1, 1, 1, nc[l])
```

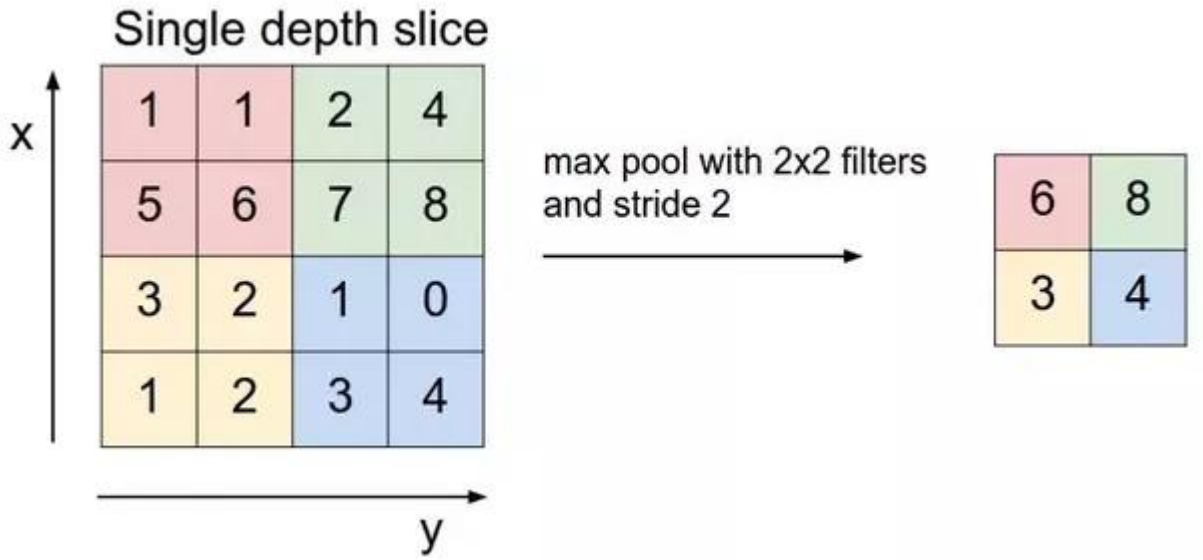
For further info refer https://github.com/mbadry1/DeepLearning.ai-Summary/tree/master/4-%20Convolutional%20Neural%20Networks#computer-vision

Pooling Layer

Pooling layers section would reduce the number of parameters when the images are too large. Spatial pooling also called subsampling or downsampling which reduces the dimensionality of each map but retains the important information. Spatial pooling can be of different types:

- Max Pooling
- Average Pooling
- Sum Pooling

Max pooling take the largest element from the rectified feature map. Taking the largest element could also take the average pooling. Sum of all elements in the feature map call as sum pooling.



Summary of pooling

Hyperparameters:

f : filter size

s : stride

Max or average pooling

p: padding

No parameters to learn!

f=2, s=2
f=3, s=2

$n_H \times n_w \times n_c$

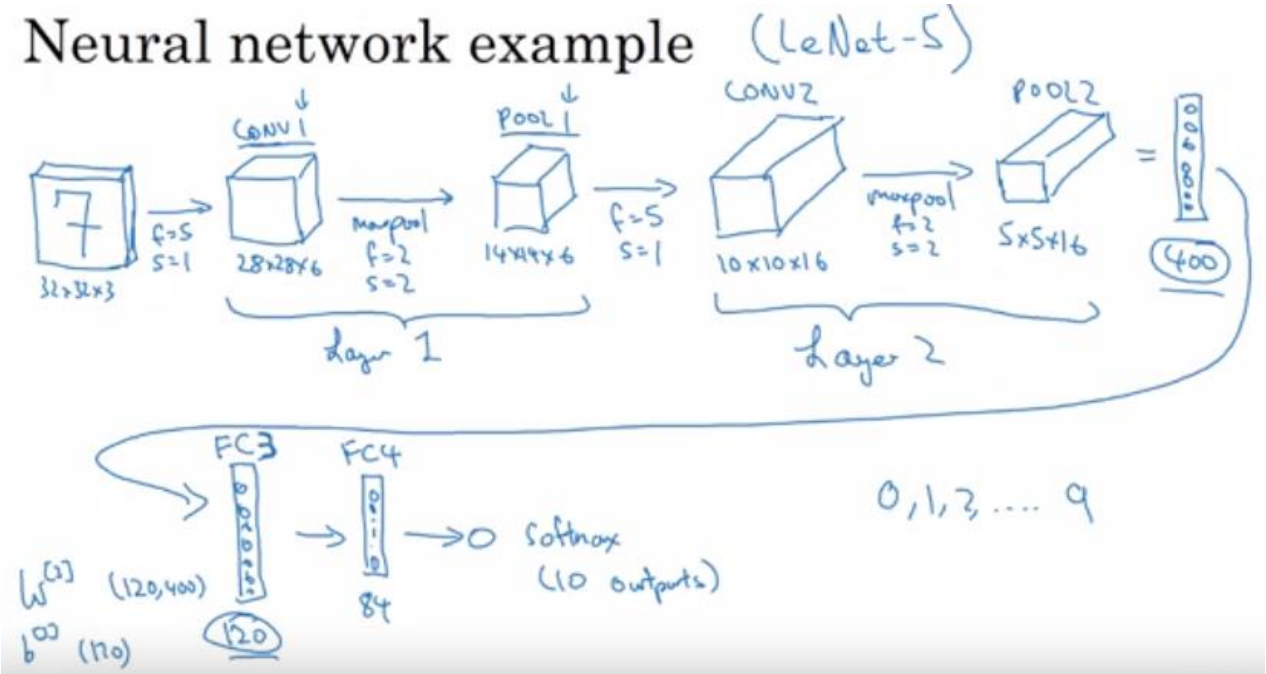
$\downarrow$

$\left\lfloor \frac{n_H - f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n_w - f}{s} + 1 \right\rfloor \times n_c$

The function of Pooling is to progressively reduce the spatial size of the input representation [4]. In particular, pooling

- makes the input representations (feature dimension) smaller and more manageable
- reduces the number of parameters and computations in the network, therefore, controlling overfitting
- makes the network invariant to small transformations, distortions and translations in the input image (a small distortion in input will not change the output of Pooling – since we take the maximum / average value in a local neighborhood).
- helps us arrive at an almost scale invariant representation of our image (the exact term is “equivariant”). This is very powerful since we can detect objects in an image no matter where they are located.

NN Example from deeplearning.ai



Things learnt till now -

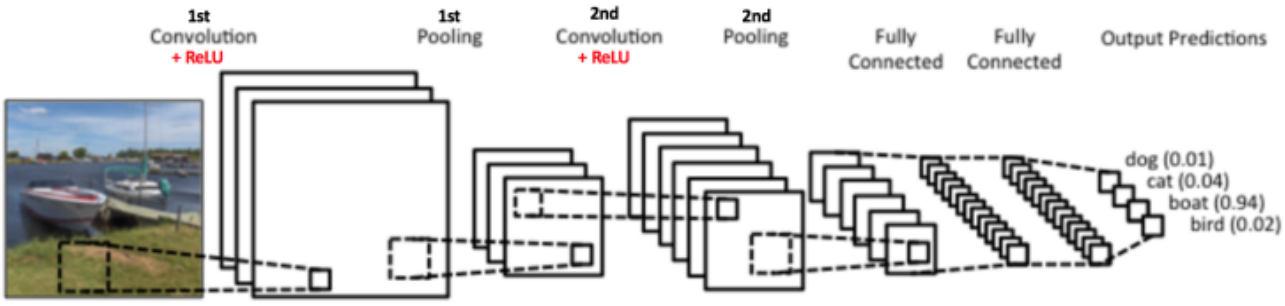


Figure 13

So far we have seen how Convolution, ReLU and Pooling work. It is important to understand that these layers are the basic building blocks of any CNN. As shown in **Figure 13**, we have two sets of Convolution, ReLU & Pooling layers – the 2nd Convolution layer performs convolution on the output of the first Pooling Layer using six filters to produce a total of six feature maps. ReLU is then applied individually on all of these six feature maps. We then perform Max Pooling operation separately on each of the six rectified feature maps.

Together these layers extract the useful features from the images, introduce non-linearity in our network and reduce feature dimension while aiming to make the features somewhat equivariant to scale and translation

Summary

- Provide input image into convolution layer
- Choose parameters, apply filters with strides, padding if requires. Perform convolution on the image and apply ReLU activation to the matrix.
- Perform pooling to reduce dimensionality size
- Add as many convolutional layers until satisfied
- Flatten the output and feed into a fully connected layer (FC Layer)
- Output the class using an activation function (Logistic Regression with cost functions) and classifies images.

Ref - https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

References

https://towardsdatascience.com/computers-that-learn-to-see-a2b783576137

https://www.analyticsvidhya.com/blog/2018/12/guide-convolutional-neural-network-cnn/

https://machinelearningmastery.com/padding-and-stride-for-convolutional-neural-networks/

https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/ (IMP)