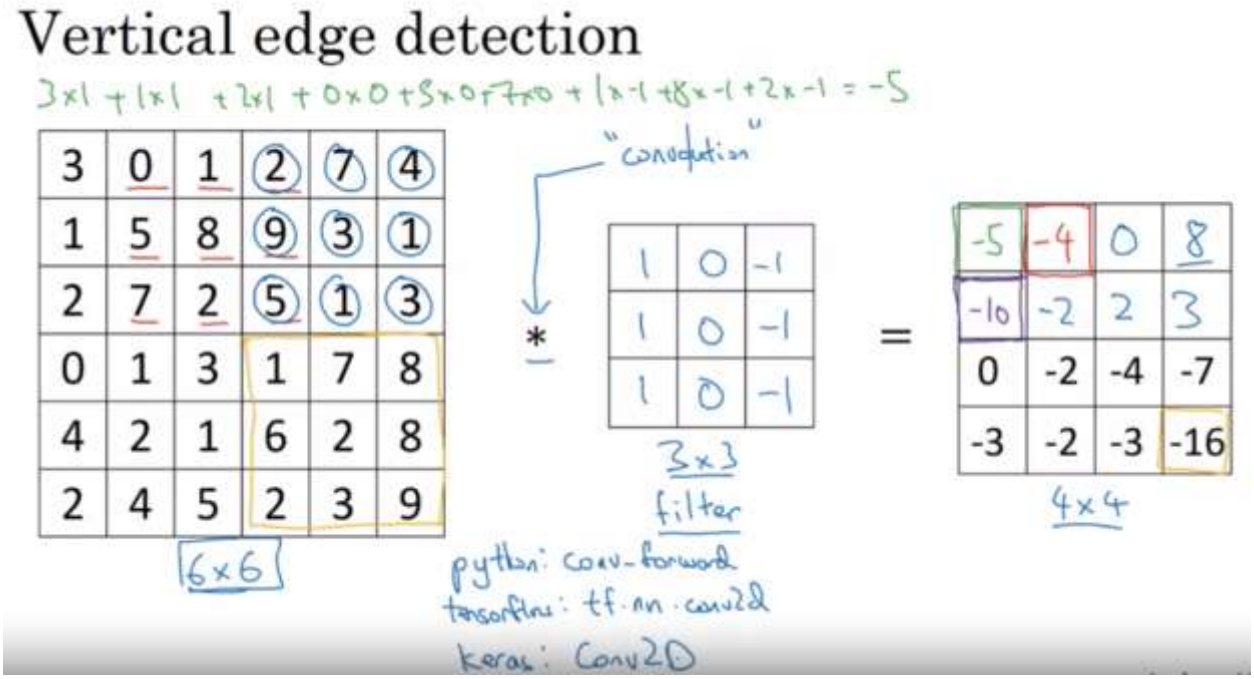
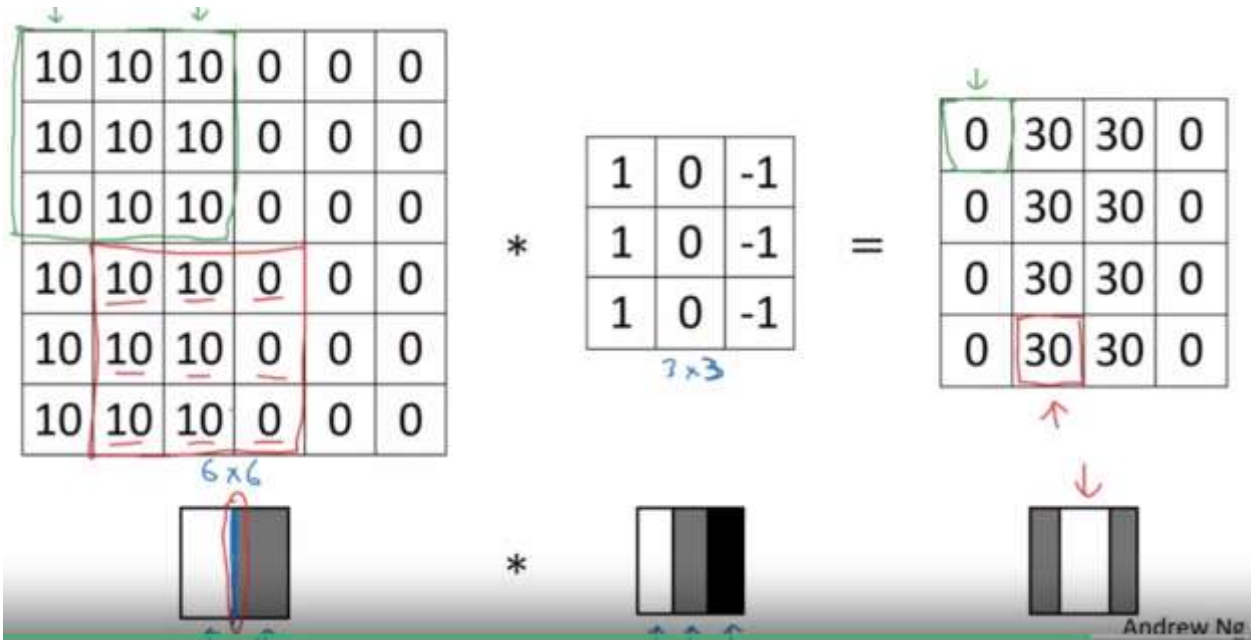
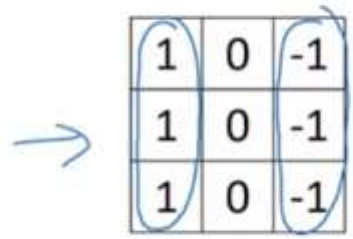
Convolutional Neural Networks



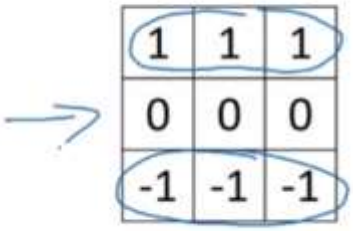


Vertical and Horizontal Edge Detection



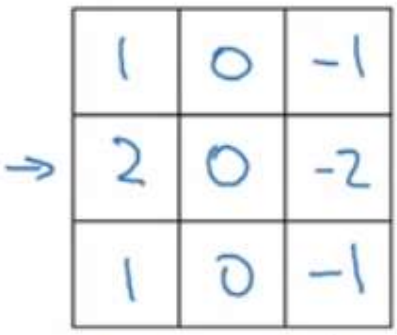
1	0	-1
1	0	-1
1	0	-1

Vertical



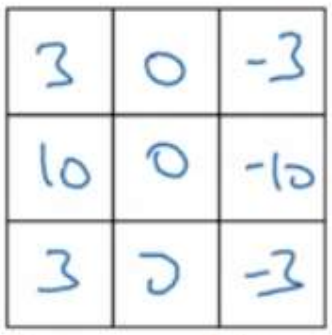
1	1	1
0	0	0
-1	-1	-1

Horizontal



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

Sobel filter



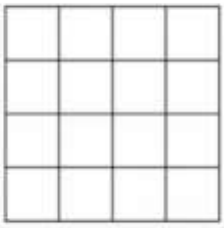
3	0	-3
10	0	-10
3	0	-3

Scharr filter

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

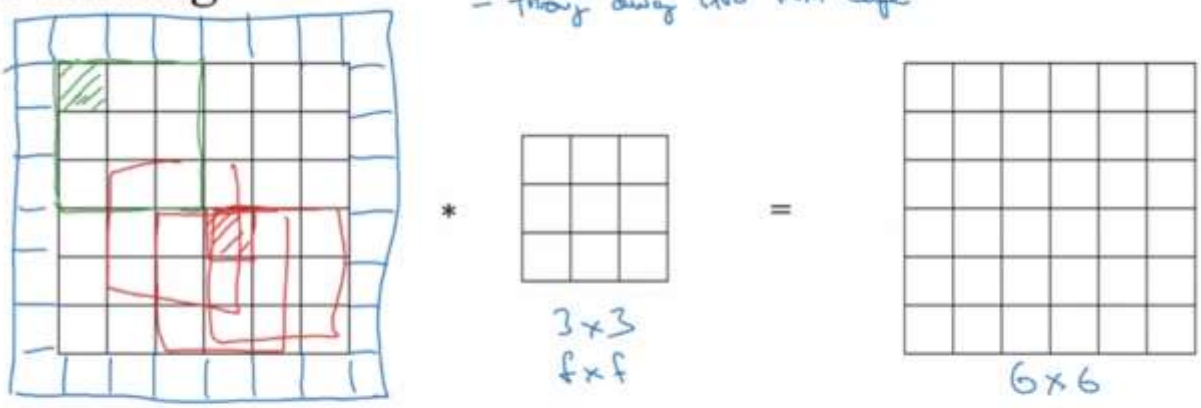
w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

45°
70°
73°



Padding

- shrinky output
- throw away into from edge



6×6 → 8×8
n×n

n-f+1 × n-f+1
6-3+1=4

Valid and Same convolutions

↳ no padding

"Valid": $n \times n \quad * \quad f \times f \quad \rightarrow \quad n-f+1 \times n-f+1$

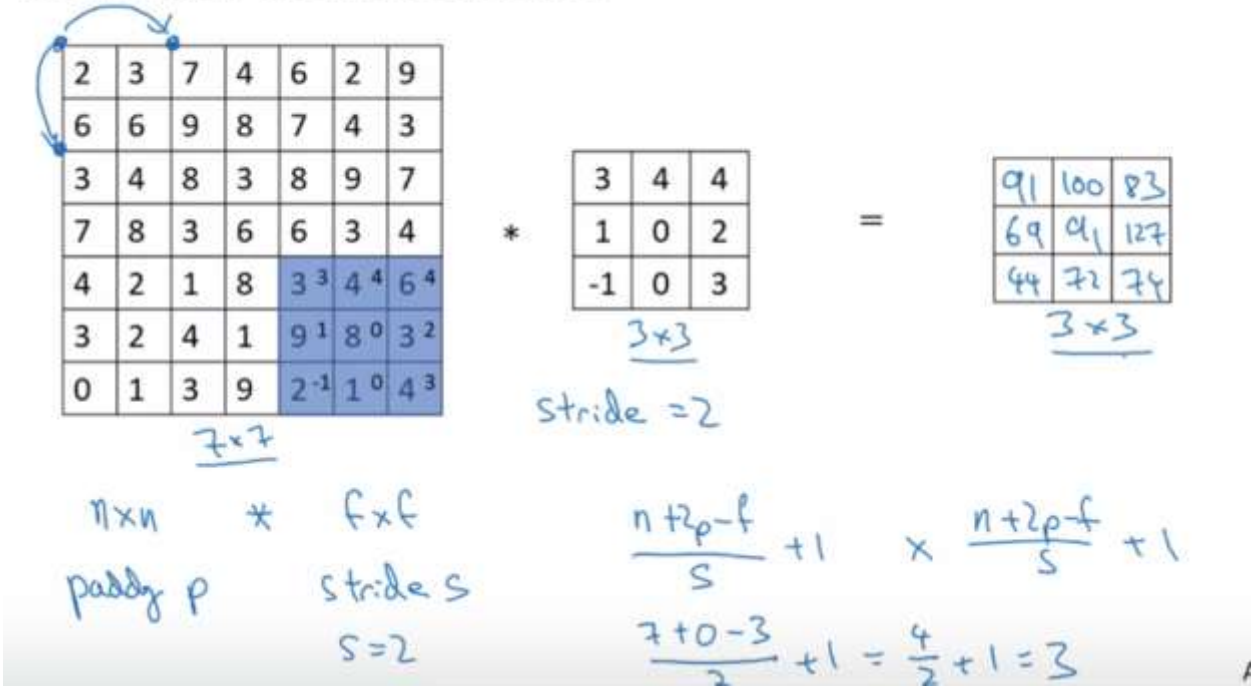
$6 \times 6 \quad * \quad 3 \times 3 \quad \rightarrow \quad 4 \times 4$

"Same": Pad so that output size is the same as the input size.

$n+2p-f+1 \times n+2p-f+1$

$n+2p$

Strided convolution



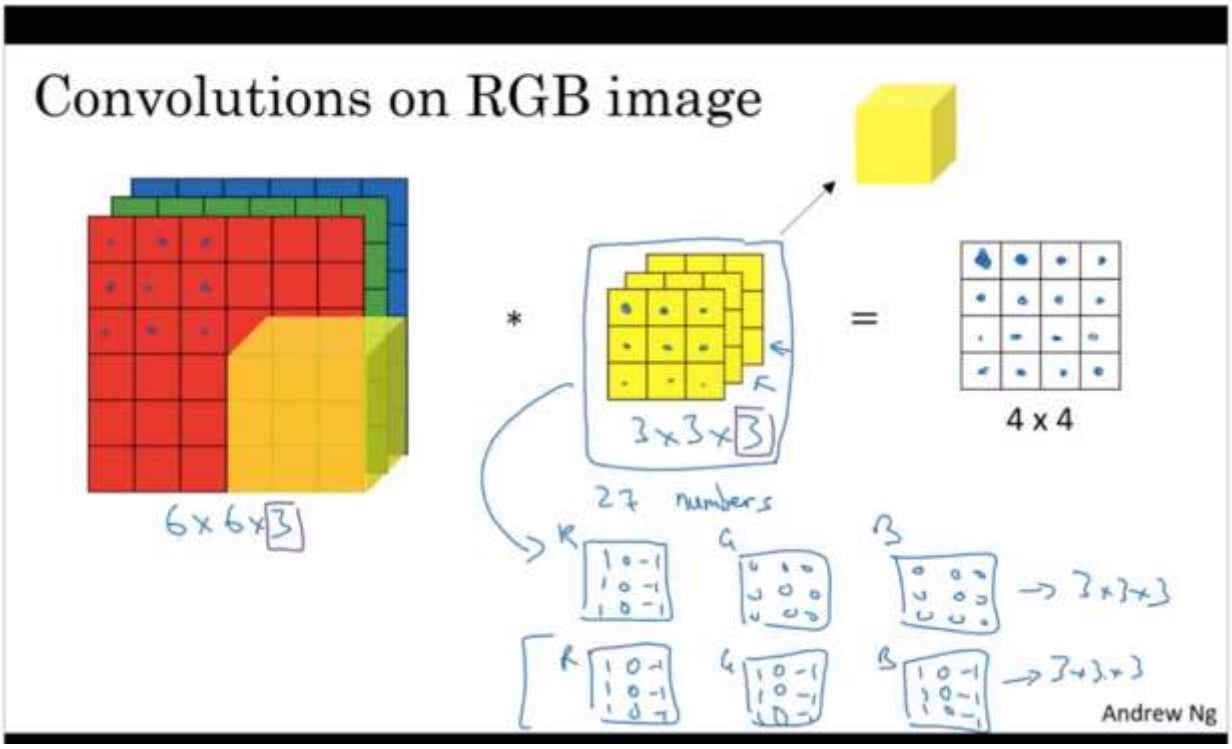
$n \times n$ image $\quad f \times f$ filter

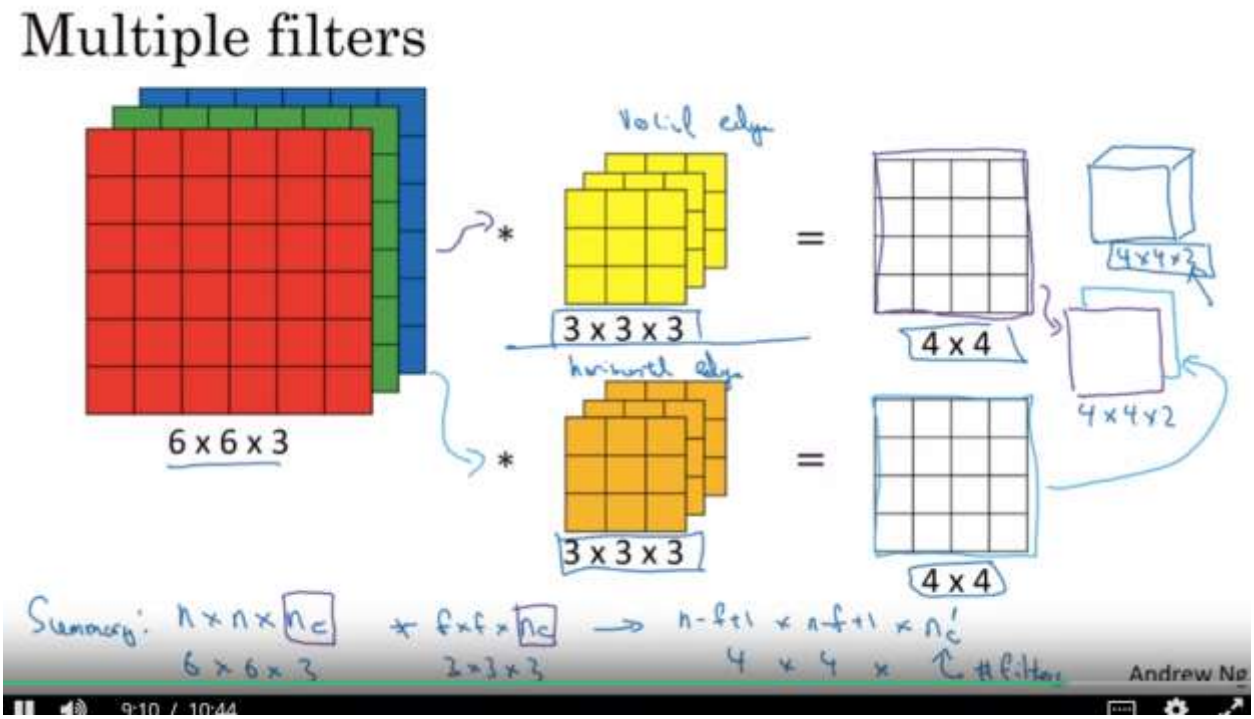
padding p $\quad$ stride s

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

The convolution operation used in Deep Learning is actually called cross-correlation (in mathematics).

Convolutions Over Volume





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.

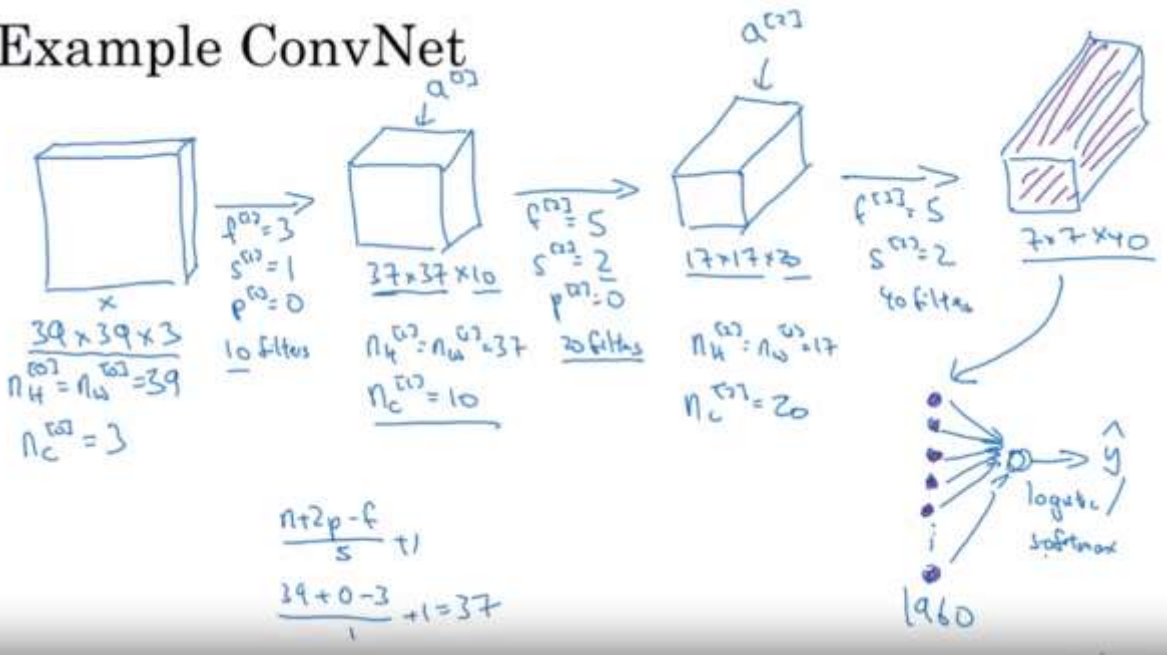
Input: $n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]}$ ←

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

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

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

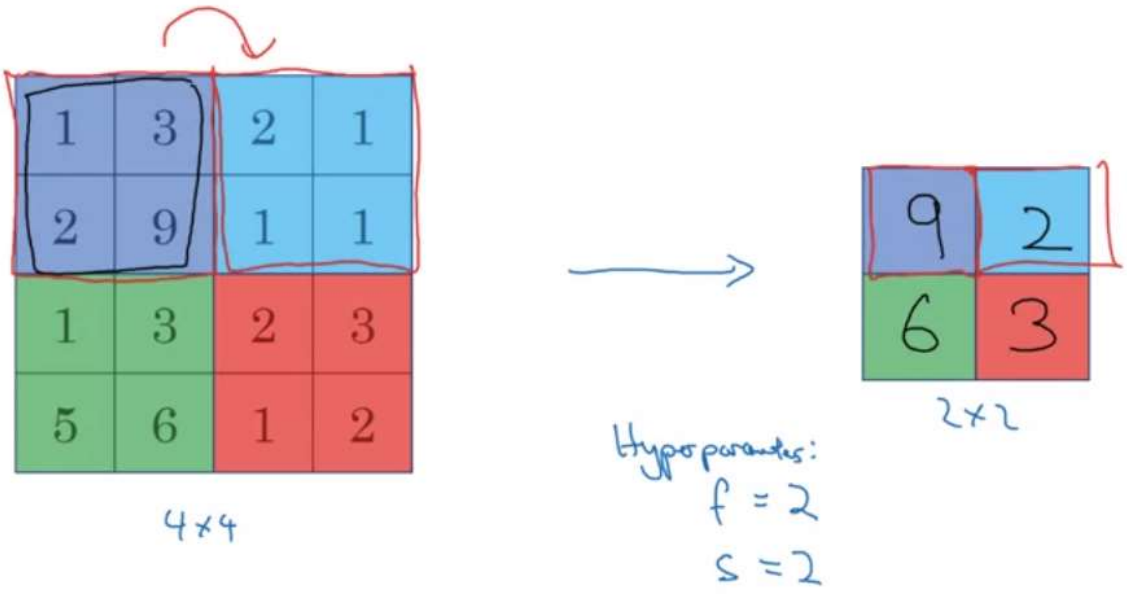
Example ConvNet



Types of layer in a convolutional network:

- Convolution (CONV) ←
- Pooling (POOL) ←
- Fully connected (FC) ←

Pooling layer: Max pooling



There is also Average pooling.

Summary of pooling

Hyperparameters:

f : filter size (f=2, s=2; f=3, s=2)

s : stride

Max or average pooling

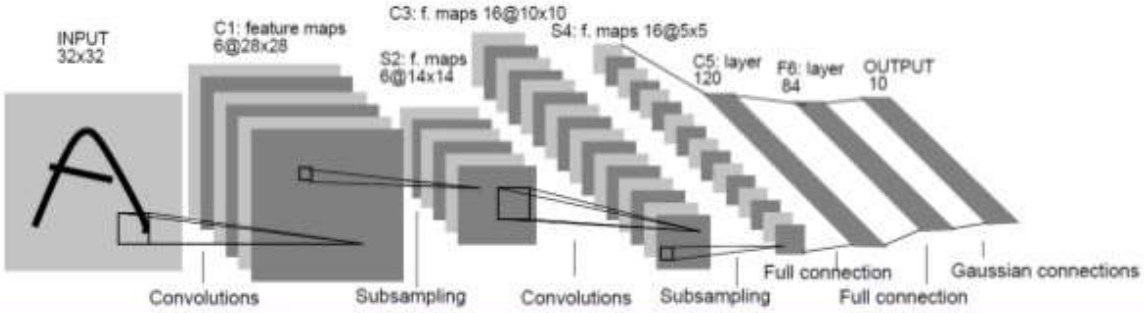
LeNet (LeNet-5)

A layered model composed of convolution and subsampling operations followed by a holistic representation and ultimately a classifier for handwritten digits;

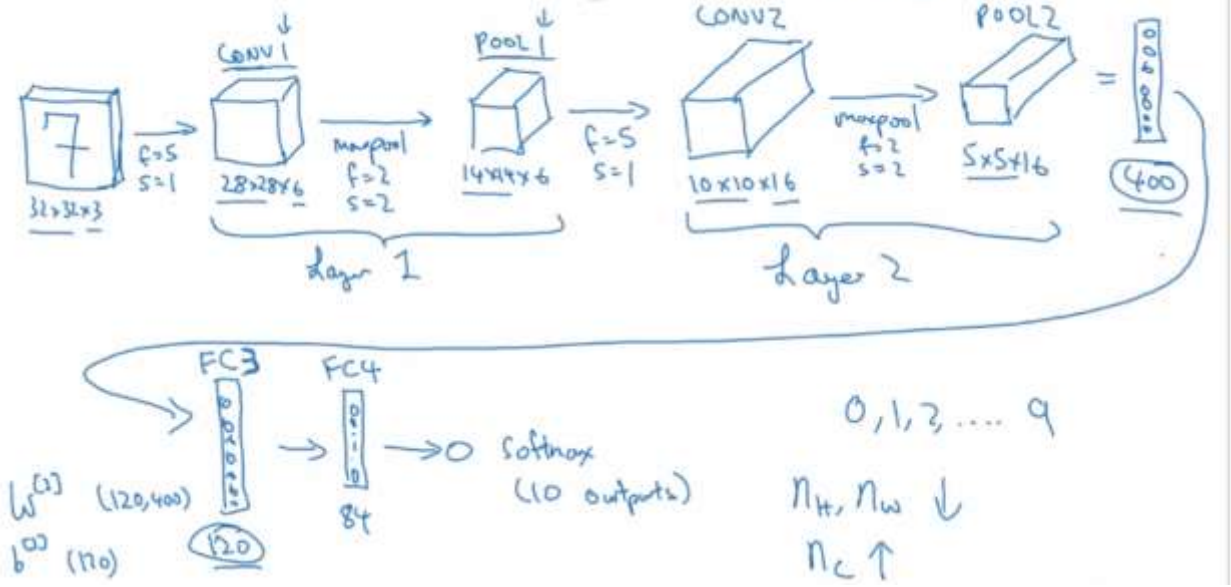
Local receptive fields (5x5) with local connections;

Output via a RBF function, one for each class, with 84 inputs each;

Learning by Graph Transformer Networks (GTN);



Neural network example (LeNet-5)

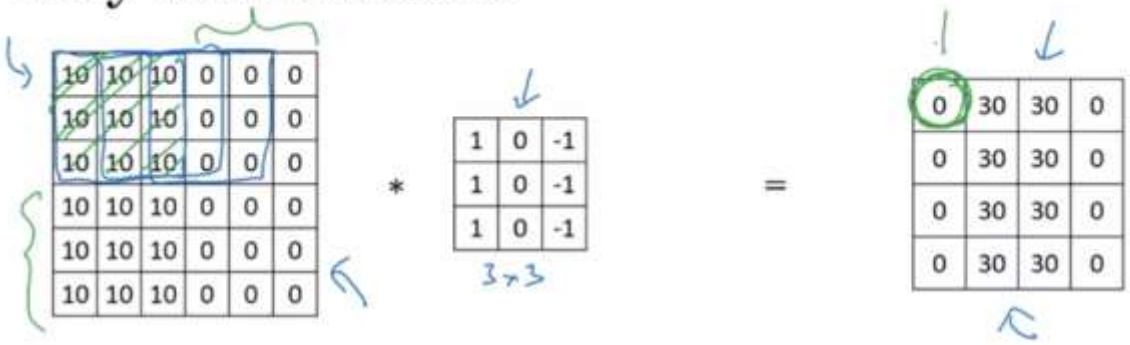


Neural network example

	Activation shape	Activation Size	# parameters
Input:	(32,32,3)	3,072	0
CONV1 (f=5, s=1)	(28,28,8)	6,272	208
POOL1	(14,14,8)	1,568	0
CONV2 (f=5, s=1)	(10,10,16)	1,600	416
POOL2	(5,5,16)	400	0
FC3	(120,1)	120	48,001
FC4	(84,1)	84	10,081
Softmax	(10,1)	10	841

There are some typos at the computations above.

Why convolutions

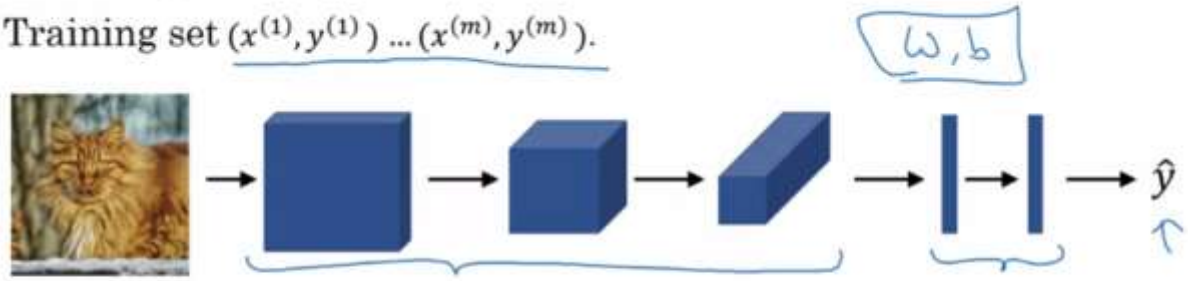


Parameter sharing: A feature detector (such as a vertical edge detector) that's useful in one part of the image is probably useful in another part of the image.

Sparsity of connections: In each layer, each output value depends only on a small number of inputs.

Putting it together

Training set $(x^{(1)}, y^{(1)}) \ldots (x^{(m)}, y^{(m)})$.



Cost $J = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$

Use gradient descent to optimize parameters to reduce J

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.

Exercise: Implement the following function, which pads all the images of a batch of examples X with zeros. Use np.pad. Note if you want to pad the array "a" of shape $(5, 5, 5, 5, 5)$ with `pad = 1` for the 2nd dimension, `pad = 3` for the 4th dimension and `pad = 0` for the rest, you would do:

1.3 - Forward propagation

In TensorFlow, there are built-in functions that implement the convolution steps for you.

- **tf.nn.conv2d(X,W, strides = [1,s,s,1], padding = 'SAME'):** given an input X and a group of filters W, this function convolves W's filters on X. The third parameter ([1,s,s,1]) represents the strides for each dimension of the input (m, n_H_prev, n_W_prev, n_C_prev). Normally, you'll choose a stride of 1 for the number of examples (the first value) and for the channels (the fourth value), which is why we wrote the value as `[1,s,s,1]`. You can read the full documentation on conv2d.
- **tf.nn.max_pool(A, ksize = [1,f,f,1], strides = [1,s,s,1], padding = 'SAME'):** given an input A, this function uses a window of size (f, f) and strides of size (s, s) to carry out max pooling over each window. For max pooling, we usually operate on a single example at a time and a single channel at a time. So the first and fourth value in `[1,f,f,1]` are both 1. You can read the full documentation on max_pool.
- **tf.nn.relu(Z):** computes the elementwise ReLU of Z (which can be any shape). You can read the full documentation on relu.
- **tf.contrib.layers.flatten(P)**: given a tensor "P", this function takes each training (or test) example in the batch and flattens it into a 1D vector.
    - If a tensor P has the shape (m,h,w,c), where m is the number of examples (the batch size), it returns a flattened tensor with shape (batch_size, k), where $k = h \times w \times c$. "k" equals the product of all the dimension sizes other than the first dimension.
    - For example, given a tensor with dimensions [100,2,3,4], it flattens the tensor to be of shape [100, 24], where 24 = 2 *3* 4. You can read the full documentation on flatten.
- **tf.contrib.layers.fully_connected(F, num_outputs):** given the flattened input F, it returns the output computed using a fully connected layer. You can read the full documentation on full_connected.

In the last function above (`tf.contrib.layers.fully_connected`), the fully connected layer automatically initializes weights in the graph and keeps on training them as you train the model. Hence, you did not need to initialize those weights when initializing the parameters.

Window, kernel, filter

The words "window", "kernel", and "filter" are used to refer to the same thing. This is why the parameter `ksize` refers to "kernel size", and we use `(f,f)` to refer to the filter size. Both "kernel" and "filter" refer to the "window."

You might find these two functions helpful:

- **tf.nn.softmax_cross_entropy_with_logits(logits = Z, labels = Y):** computes the softmax entropy loss. This function both computes the softmax activation function as well as the resulting loss. You can check the full documentation softmax_cross_entropy_with_logits.
- **tf.reduce_mean:** computes the mean of elements across dimensions of a tensor. Use this to calculate the sum of the losses over all the examples to get the overall cost. You can check the full documentation reduce_mean.

Details on softmax_cross_entropy_with_logits (optional reading)

- Softmax is used to format outputs so that they can be used for classification. It assigns a value between 0 and 1 for each category, where the sum of all prediction values (across all possible categories) equals 1.
- Cross Entropy is compares the model's predicted classifications with the actual labels and results in a numerical value representing the "loss" of the model's predictions.
- "Logits" are the result of multiplying the weights and adding the biases. Logits are passed through an activation function (such as a relu), and the result is called the "activation."
- The function is named `softmax_cross_entropy_with_logits` takes logits as input (and not activations); then uses the model to predict using softmax, and then compares the predictions with the true labels using cross entropy. These are done with a single function to optimize the calculations.

```
output_for_var1, output_for_var2 = sess.run(
                                           fetches=[var1, var2],
                                           feed_dict={var_inputs: the_batch_of_inputs,
                                                      var_labels: the_batch_of_labels}
                                           )
```