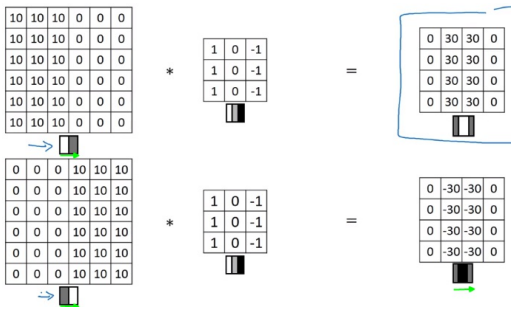
IV - Foundations of CNN

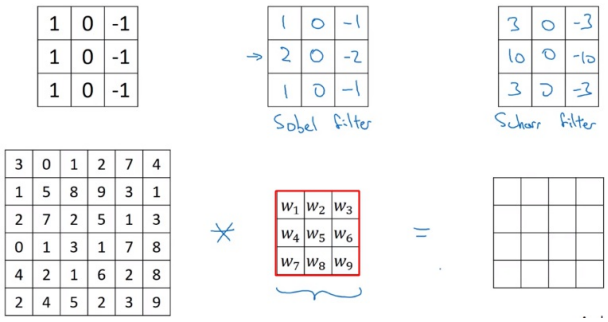
2017/11/27 18:52

- Sum

- Computer Vision
  - two reasons for the excitment for Deep Learning in Computer Vision
    - First, rapid advances in computer vision are enabling brand new applications to be able
    - the CV research community has been so creative and so inventive in coming up with new neural network architectures and algorithms, which creates a lot of cross-fertilization into other areas as well.
      - borrow ideas from them
  - One problem
    - the input for the NN can be very big
    - e.g. 1000 x 1000 x 3 = 3m input
      - 1000 neuron for the first layer
      - W1.shape = (1000, 3m) -- 3 billion parameters
    - problems followed
      - overfitting
      - high computation/memory requirement
- Edge Detection Example
  - to detect/classify the obj in the image, the first thing might be to detect the edges. Here, convolution helps.
  - the conv function is implemented in many program languages or packages
    - tt.nn.cov2d -- tensorflow
    - Conv2D -- keras
- More Edge Detection
  - the difference between positive and negative edges
    - light to dark, versus dark to light, edge transitions



    - to make them same, take the absolute value of the second output.
  - an algorithm to learn to detect the edges
    - CV specialists are competing on which is the best edge detector but there're certain constraints for those. **A more flexible and powerful way is let the NN to learn the filter for the specific edge detection.**



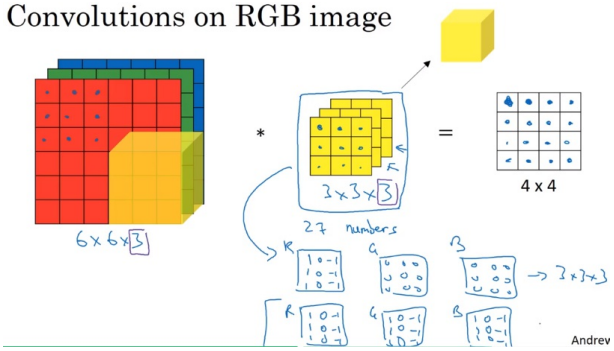
- Padding
  - In order to build deep neural networks, one modification to the basic convolutional operation that you need to really use is padding.
  - Motivation - downside of the above conv
    - the output image size is shrinked every time applied conv. which influence the later steps
    - the edge pixels are used much less then the middle so that much info. in the edges is lost
  - the solution is padding
    - extend the image size before applying conv.
      - p = (f-1)/2
    - normally, the size of the filter is odd.
  - how to pad
    - dark: 0s for the padding pixels
    - other ways
- Strided Convolutions
  - stride: the step the filter takes to appy conv.
  - the pixel change formula

$$n \times n \text{ image} \qquad f \times f \text{ filter}$$

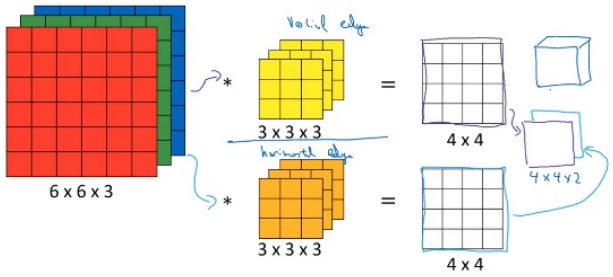
$$\text{padding } p \qquad \text{stride } s$$

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

  - by convention in ML, it's actually cross-correlation but most of the deep learning literature just calls this the convolution operator. i.e. we usually do not filp the filter ahead
    - for simplify the coding and doesn't influence the performance of NN.
- Convolutions Over Volume
  - conv. over volume
    - the channel dim must be the same for both the image and filter
    - the output is a 2D matrix
    - to detect different feature in different color, you can implement different patterns in the filter accordingly.

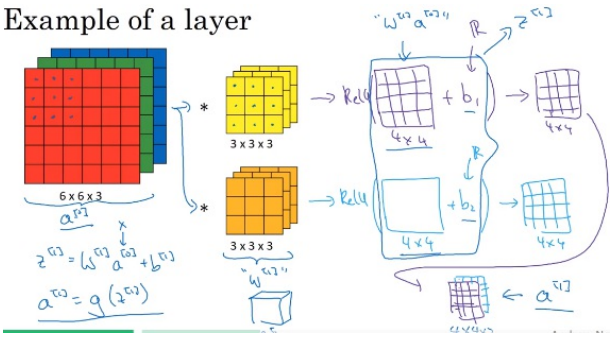


- Multiple filters
  - apply several filters to one RGB image and stack the their outputs together

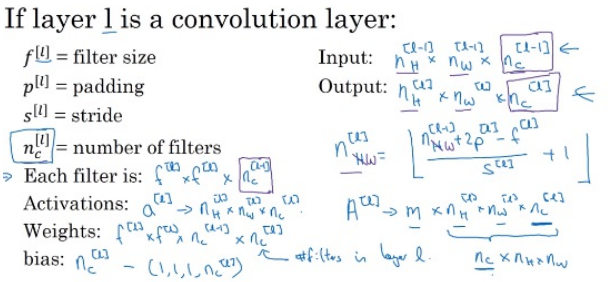


    - the #channels (depth) of the filters and images should be same.

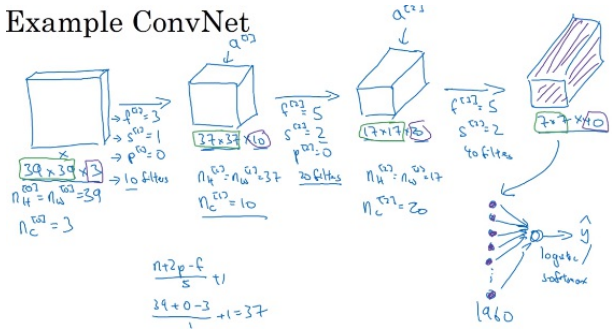
- One Layer of a Convolutional Network
  - one layer is defined from the input to the output
  - analogous to the basic NN forward prop.



  - the conv. is linear
  - One great advantage of CNN
    - the #para is determined by the filters instead of input image which is very big
      - so the #para can be much smaller => learn faster & less prone to overfit
    - notations: para and their relation



- Simple Convolutional Network Example



    - 3 conv layers the final NN layer takes the unwrapped elements as input
    - normal achitecture
        - the size of the image becomes smaller and smaller
        - while the #channels becomes bigger and bigger
    - types of layer in CNN
        - conv layer
        - pooling layer
        - fully connected layer
- Pooling Layers
    - functions
        - to reduce the size of their representation
        - to speed up computation
        - to make some of the features it detects a bit more robust
    - max pooling
        - para: f, stride --- no need to learn, but to tune
        - intuition
            - the bigger values in the feature map may indicate the existence of the featuer
            - if the feature is detected anywhere in this filter, then keep a high number. But if this feature is not detected, then the max of all those numbers is still itself quite small.
                - this is the reason specialist cited a lot, but it might just be a guess
                - the true reason is that in experiment, max pooling helps a lot.
        - implement
            - the #channels is same for the input and output
    - average pooling
        - So these days max pooling is used much more often than average pooling, with one exception,
            - in very deep in a neural network, you might **use average pooling to collapse your implementation from**, say, 7x7x1000. And average over all the spacial extents to get 1x1x1000.
    - sum

        - Hyperparameters
            - f: filter size
            - s: stride
            - max or average
        - normal choice
            - f = 2, s = 2 very often
            - f = 3, s = 2 sometimes
            - no padding usually
        - no parameters to learn
- CNN Example

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

    - by convention, conv layer = CONV1 + POOL1, only count the layer with parameters
    - Activation size gradually decrease as the net goes deeper and deeper
    - conv layers have much less parameters than the FC layers
    - how to decide the architecture/hyperparameters
        - one way is to look at others' NN to get an inspiration and borrow from the others
        - the other way is see a many examples so that you get an intuition
- Why Convolutions?
    - why CNN works greater than only FC -- reduce the parameters
        - parameter sharing
            - a feature 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
        - so that
            - fast to train
            - less prone to overfit
            - translation invariance

- Ex1 - convolution model
    - packages
        - numpy is the fundamental package for scientific computing with Python.

        - matplotlib is a library to plot graphs in Python.
        - np.random.seed(1) is used to keep all the random function calls consistent. It will help us grade your work.
    - **Note** that for every forward function, there is its corresponding backward equivalent. Hence, at every step of your forward module you will store some parameters in a cache. These parameters are used to compute gradients during backpropagation.
    - **3.1 - Zero-Padding**
        - 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.
        - Use np.pad.
    - **3.2 - Single step of convolution**
        - you will implement a single step of convolution, corresponding to applying a filter to just one of the positions to get a single real-valued output
    - **3.3 - Convolutional Neural Networks - Forward pass**
        - Use the output volume as a guide to compute each element respectively, i.e. iterate ove the ith training example, output high, output width, output channel.
    - **4.1 - Forward Pooling**
        - Same as Conv layer.
    - **5 - Backpropagation in convolutional neural networks**
        - **OPTIONAL**
            - In modern deep learning frameworks, you only have to

implement the forward pass, and the framework takes care of the backward pass, so most deep learning engineers don't need to bother with the details of the backward pass.

- **5.1 - Convolutional layer backward pass**
  - **看完 CH2再回来细细研究**

- Ex1 - convolution model - Application v1 - `tensorflow-based`
- **1.1 - Create placeholders**
  - `placeholder`
    - `tf.placeholder(dtype, shape=None, name=None)`
      - `dtype: tf.float16/32/64, tf.int8/16/32/64, tf.uint8/16, tf.bool, tf.string`
      - `#The shape of the tensor to be fed (optional). If the shape is not specified, you can feed a tensor of any shape.`
      - `var = None # optional`
    - This tensor will produce an error if evaluated. Its value must be fed using the `feed_dict` optional argument to `Session.run()`, `Tensor.eval()`, or `Operation.run()`.
      - e.g. `sess.run(y, feed_dict={x: rand_array})`
    - vs. `tf.Variable`
      - `tf.Variable` for trainable variables such as weights (W) and biases (B) for your model.
        - have to provide an initial value when you declare it
      - `tf.placeholder` is used to feed actual training examples.
        - don't have to provide an initial value

and you can specify it at run time with the `feed_dict` argument inside `Session.run`

- **1.2 - Initialize parameters**
  - ATT
    - You will initialize weights/filters W1 and W2 using `tf.contrib.layers.xavier_initializer(seed = 0)`. You don't need to worry about bias variables as you will soon see that **TensorFlow functions take care of the bias**. Note also that you will **only initialize the weights/filters for the conv2d functions**. **TensorFlow initializes the layers for the fully connected part automatically**.
  - tf.get_variable(`name, shape=None, dtype=None, initializer=None,...`)
    - Gets an existing variable with these parameters or create a new one.
    - same name can't be assigned twice
  - `contrib.layers`
    - Ops for building neural network layers, regularizers, summaries, etc.
  - `initializer -- used in tf.get_variable()`
    - `xavier_initializer -- Returns an initializer performing "Xavier" initialization for weights.`
      - `tf.contrib.layers.xavier_initializer`(uniform=True, seed=None, dtype=tf.float32)
      - `tf.contrib.layers.xavier_initializer_conv2d`(uniform=True, seed=None, dtype=tf.float32)
        - **`uniform`**: Whether to use **uniform** or normal distributed random initialization.
        - **`seed`**: A Python integer. Used to

create random seeds. See `set_random_seed` for behavior.

- normal initializers
  - `tf.constant_initializer(value, dtype, verify_shape)`
    - **`verify_shape`,** If `True`, the initializer will throw an error if the shape of `value` is not compatible with the shape of the initialized tensor.
  - tf.zeros_initializer()
  - tf.ones_initializer()