Lecture Two - Stanford CS231N

Name: Eli Andrew

- The sizes of images make very large input layers which makes traditional fully-connected layers not feasible for many images
- **Convolution**
  - Vertical edge detection filter

$$Filter = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

  - For an $n \times n$ image convolved with an $f \times f$ filter you get an output of size:

$$(n - f + 1) \times (n - f + 1)$$

  - Regular convolution has several issues:
    - Image shrinks on every convolution
    - Not using a lot of information at the edge of the image
  - If you pad your image with p pixels around the edge you get an output size of:

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

  - A **Valid** convolution uses no padding
  - A **Same** convolution uses enough padding to keep the same output size
  - **Strided** convolution uses a stride to control how many indices you move on each step of the convolution
  - The size of a strided convolution with stride s and padding p is equal to:

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

  - When convolving over three dimensions your filter should also have 3 dimensions and you should sum over all dimensions for each step of the convolution
  - So the third dimension of each filter needs to match the third dimension of the input
  - And the third dimension of the output will be equal to the number of filters you used
- **Convolutional Layer**

- Each filter of the layer can be thought of as a feature (ex: vertical edge filter, horizontal edge filter)
- If layer l is a convolutional layer:

$$f^{[l]} = size_{filter}$$
$$p^{[l]} = padding$$
$$s^{[l]} = stride$$
$$n_c^{[l]} = number_{filters}$$
$$Input = n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]}$$
$$Output = n_H^{[l]} \times n_W^{l} \times n_c^{[l]}$$
$$Filter_{dimensions} = f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$$
$$Activations = a^{[l]} = n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$$
$$Weights = f^{[l]} * f^{[l]} * n_c^{[l-1]} * n_c^{[l]}$$

- ConvNets tend to decrease in n_H and n_W and increase in n_c

- **Pooling Layers**
  - Filter and stride are the parameters
  - **Max Pooling**: on each step you take the max of all elements in the filter boundaries
  - Max pooling almost never uses any padding
  - Unlike convolutional layers, max pooling layers have no parameters to learn

- **ResNets**
  - Residual block: Adds the activtions from a previous layer to two layers ahead using a skip connection

$$a^{[l+2]} = g(z^{[l+1]} + a^{[l]})$$

  - Why ResNets work
    * Easy for block to learn the identity function by setting W and b to 0 so the added layer doesn't hurt performance since it can always just use identity function
    * $z^{[l+1]}$ and $a^{[l]}$ have to be the same dimension so people tend to use SAME convolution to ensure that the dimensions are the same
    * If dimensions aren't the same you can add a W matrix to multiple $a^{[l]}$ by to ensure the dimensions are the same (this W can be learned or could be constant that just implemetns zero padding)

- What do 1×1 convolutions do?
  - If you have a filter with size 1 on an $x \times y \times z$ sized input then each convolution is like a single neuron multipling the z weights together and putting it through an activation and outputing to the next layer
  - With multiple filters then it's like having multiple neurons that do this same operation
  - This produces an output volume of $x \times y \times num_{filters}$
  - This is also called Networks in Networks
  - **Example**: you could use this to shrink a $28 \times 28 \times 192$ input size to a $28 \times 28 \times 32$ output size by using 32 $1 \times 1 \times 192$ filters
  - This is useful because pooling layers only let you shrink the n_H and n_W
- **Inception Network**
  - Performs several convolutions, or pooling on a single layer (with different f) and stacks the outputs
  - To help with computation cost it uses 1×1 convolutions in a "bottleneck" layer where you shrink the number of channels down by around a factor of 10
  - It also adds softmax branches at various points within the network to make sure that even the intermediate weights are good at classifing output classes (this has a regularization effect)