

1.3 Convolutional Neural Networks

1.3.1 Basic ConvNets

- We use a filter to take a dot product with a portion of an image. So given some $n^{[l-1]} \times n^{[l-1]}$ input, using a $f \times f$ filter with padding p and stride s , our output is size

$$n^{[l]} = \frac{n^{[l-1]} - f + 2p}{s} + 1 . \quad (5)$$

We run many filters for each input image. Formally, given some input image of shape $(n_h^{[l-1]}, n_w^{[l-1]}, n_c^{[l-1]})$, we can use a filter of shape $(f^{[l]}, f^{[l]}, n_c^{[l-1]})$ with padding $p^{[l]}$ and stride $s^{[l]}$ where the weights are shape $(f^{[l]}, f^{[l]}, n_c^{[l-1]}, n_c^{[l]})$ and the bias is shape $(1, 1, 1, n_c^{[l]})$ to produce an output of shape $(n_h^{[l]}, n_w^{[l]}, n_c^{[l]})$.

- Typically, max pooling (or average pooling) layers are also used, with $f = 2$ and $s = 2$.
- Two reasons we use convolutions are (1) they share parameters and (2) the sparsity of connections (in each layer, each output value only depends on a small number of inputs).
- Three classic networks:
 - LeNet-5 (60k parameters): trained on MNIST with $2 \times (\text{conv} + \text{pool})$ layers with two FC layers.
 - AlexNet (60M parameters): worked on ImageNet, similar to LeNet but had 8 layers. Used ReLU function with multiple GPUs. Used local response normalization (normalize activations on the n_c axis, doesn't help much). Used data augmentation and dropout to reduce overfitting.
 - VGG-16 (138M parameters): simple because it used convolutions with 3×3 filters with $s = 1$ and max pooling with $f = 2$ and $s = 2$.
- Filters for edge detection:

$$\text{horizontal edge detector : } \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}, \text{ Sobel filter for vertical edge detection : } \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

1.3.2 ResNet

- Context: deep networks are harder to train because of the exploding/vanishing gradient problem. In theory, training error should go down for deeper networks, but in practice, training error went up after some point for deeper networks.
- ResNet solved this problem by introducing skip connections. We have

$$a^{[l+2]} = g(z^{[l+2]} + a^{[l]}) . \quad (6)$$

Because $z^{[l+2]} = w^{[l+2]}a^{[l+1]} + b^{[l+2]}$, when $w^{[l+2]}$ and $b^{[l+2]}$ are 0, then $a^{[l+2]} = a^{[l]}$, so it's easy to learn the identity function.

1.3.3 Inception Net

- 1×1 convolutions are very important. While for an input with only one channel, you're just multiplying by one number, for multiple channels, you're multiplying the number in each channel by some weight, summing it, and applying an activation function. You can think of it as a small FC layer.
- In an inception module, instead of picking the filter sizes, you can use 1×1 , 3×3 , and 5×5 , and maxpool layers, and stack their outputs together.

- The cost of computing some outputs, however, can be large. For instance, for an input of $(28, 28, 192)$ with a 5×5 convolutional layer with same padding and 32 channels, you have to do $28 \times 28 \times 32 \times 5 \times 5 \times 32 = 120\text{M}$ computations. You can reduce this by using a 1×1 convolution to decrease the number of channels in the input from size 192 to 16, for example.
- Yolo uses a 19 by 19 grid cell with 5 anchors, and then outputs a probability of object, 4 coordinates for bounding boxes, and then softmax for each class.
- Non-maximum suppression discards all boxes with a $p_c < \text{threshold}$, and while there are boxes remaining, pick the largest p_c , and discards boxes with a high IOU with that p_c . Do this for every object class.
- For face recognition, need to do one-shot learning.
 - So pass through a neural net, and then take one of the encodings of the last layer $f(x)$.
 - Then learn a similarity function $d(\text{img}_1, \text{img}_2) = \|f(x_1) - f(x_2)\|_2^2$ that is less than τ if different, more than τ if same.
 - Minimize the loss so that $d(x_1, x_2)$ small if same person, otherwise big.
 - So we use a triplet loss where given an anchor (A), a positive (P) and a negative (N)

$$L(A, P, N) = \max(d(A, P) - d(A, N) + \alpha, 0)$$

- And then train with triplets that hard, i.e., $D(A, P)$ is almost equal to $D(A, N)$.
- For Neural Style Transfer, want a generated image G with content C and style S .
 - Measure content with the similarity between the weights of hidden layers, or $J_{\text{content}} = \frac{1}{2} \|a_c - a_g\|_2^2$ for a hidden layer l .
 - Define style as the correlation between activations across channels, or $G = \sum_i \sum_j a_{ijk} a_{ijk'}$, and $J_{\text{style}} = \|G_S - G_G\|^2$ over many layers.