CNN review

Adapted from CS231N and CS231A
Outline

Common CNN layers

Conv / deconv

Pooling

Batch normalization

Some CNN architectures

Tensorflow/Pytorch implementations

Resources
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1

input  
1  
3072

$Wx$
10 x 3072 weights

activation
1
10

1 number: the result of taking a dot product between a row of $W$ and the input (a 3072-dimensional dot product)
Convolution Layer

32x32x3 image

5x5x3 filter

Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”
**Convolution Layer**

32x32x3 image

5x5x3 filter $\mathbf{w}$

1 number: the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. $5 \times 5 \times 3 = 75$-dimensional dot product + bias)

$$\mathbf{w}^T \mathbf{x} + b$$
Convolution Layer

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

activation map
Consider a second, green filter.

32x32x3 image
5x5x3 filter

Convolve (slide) over all spatial locations.

activation maps
Tensor dimensions
consider a second, green filter

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

activation maps
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied with stride 2
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied *with stride 2* => 3x3 output!
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied with stride 3?

doesn’t fit!
In practice: Common to zero pad the border

e.g. input 7x7
3x3 filter, applied with **stride 1**
**pad with 1 pixel** border => what is the output?

For **SAME** padding:

$$\text{output height} = \text{ceil} \left( \frac{H}{S_h} \right)$$
$$\text{output width} = \text{ceil} \left( \frac{W}{S_w} \right)$$

For **VALID** padding:

$$\text{output height} = \text{ceil} \left( \frac{H - F_h + 1}{S_h} \right)$$
$$\text{output width} = \text{ceil} \left( \frac{W - F_w + 1}{S_w} \right)$$
Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters $K$,
  - their spatial extent $F$,
  - the stride $S$,
  - the amount of zero padding $P$.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P) / S + 1$
  - $H_2 = (H_1 - F + 2P) / S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and $K$ biases.
- In the output volume, the $d$-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the $d$-th filter over the input volume with a stride of $S$, and then offset by $d$-th bias.

Common settings:

$K =$ (powers of 2, e.g. 32, 64, 128, 512)
- $F = 3, S = 1, P = 1$
- $F = 5, S = 1, P = 2$
- $F = 5, S = 2, P = ?$ (whatever fits)
- $F = 1, S = 1, P = 0$
Deconv

“Taking gradient of conv”

“Switch the input/output of conv”

Usually used with stride $> 1$

Padding $\rightarrow$ cropping
MAX POOLING

Single depth slice

max pool with 2x2 filters and stride 2
Global pooling

Useful if input image size is not uniform
Batch Normalization

- Makes networks robust to bad initialization of weights
- Usually inserted right before activation layers
- Reduce covariance shift by normalizing and scaling inputs
- The scale and shift parameters are trainable to avoid losing stability of the network

Input: Values of $x$ over a mini-batch: $B = \{x_1, \ldots, x_m\}$; Parameters to be learned: $\gamma, \beta$
Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$\mu_B \leftarrow \frac{1}{m} \sum_{i=1}^{m} x_i$  // mini-batch mean

$\sigma^2_B \leftarrow \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_B)^2$  // mini-batch variance

$\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma^2_B + \epsilon}}$  // normalize

$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i)$  // scale and shift

Algorithm 1: Batch Normalizing Transform, applied to activation $x$ over a mini-batch.
Activation Layer

- Used to increase non-linearity of the network without affecting receptive fields of conv layers
- Prefer ReLU, results in faster training
- LeakyReLU addresses the vanishing gradient problem

Other types:
Leaky ReLU, Randomized Leaky ReLU, Parameterized ReLU
Exponential Linear Units (ELU), Scaled Exponential Linear Units
Tanh, hardtanh, softtanh, softsign, softmax, softplus...
Softmax

- A special kind of activation layer, usually at the end of FC layer outputs
- Can be viewed as a fancy normalizer (a.k.a. Normalized exponential function)
- Produce a discrete probability distribution vector
- Very convenient when combined with cross-entropy loss

\[ P(y = j \mid x) = \frac{e^{x^T w_j}}{\sum_{k=1}^{K} e^{x^T w_k}} \]

Given sample vector input \( x \) and weight vectors \( \{w_j\} \), the predicted probability of \( y = j \).
Optional: Correlation layer for feature matching

Think of convolutional layer

Instead of learning filters that will be applied on all images,

What if the filters come from input images’ feature maps?
Optional: spatial transformer network

Help you achieve scale and rotation invariance

Focus on one area of the image

E.g. used in R-CNN for object detection
Constructing networks: Image classification
Constructing networks: Segmentation

Input: RGB Image

Convolutional Encoder-Decoder

Pooling Indices

Output: Segmentation

- Blue: Conv + Batch Normalisation + ReLU
- Green: Pooling
- Red: Upsampling
- Yellow: Softmax

Skip-links
Tensorflow vs Pytorch?

Static graph vs. dynamic graph

Tensorflow Eager

Many ways of coding in tensorflow

   Native TF (nn, layers), Keras, Estimator, etc.

Reusing existing code
Runnable tensorflow example

https://colab.research.google.com/drive/1bYbDgDuFwsSH4CGF4qExg7ZjPgbr8oh
Runnable example of pytorch

https://colab.research.google.com/drive/1jVG4WxTfYSI__McP4IkjhdamEisUCOTc
Resources

Tensorflow documentation

Tensorflow official models

Tensorflow seedbank

Pytorch documentation

Pytorch official models

Pytorch interactive tutorial

Gin-config