Introduction to Deep Learning in Vision: Basics, Optimization, Networks and Coding

Yue Zhang

Department of Mathematics, Applied Mathematics and Statistics
Case Western Reserve University

February 19, 2018

Outline

Overview of the Journey

CNNs Basics

- **Standard Operations**: Convolutions , BN Pooling, Nonlinearities *etc.*
- **Special Terms:** Deconvolution, Upsampling, Skip Connection, Dense-block *etc.*
- **Popular Losses**: Entropy based, Standard l_2 and l_1, Adversarial Losses, *etc.*

Optimization

- **Mathematics**: Backpropagation on CNNs
- **Popular first order methods**: SGD, Momentum, Nesterov Acceleration, Adam, RMSprop *etc.*

Network Variants

- LeNet (1998), AlexNet (2012), VGGNet (2014), GoogLeNet (2014), **FCN (2014)**, **ResNet (2015)**, **U-Net (2015),** { SegNet (2015), DenseNet (2017), Dense-UNet (2017) } ***GAN (2014)*** { C-GAN (2014), Cycle-GAN (2017)}

Coding

- **Frameworks**: Caffe (Berkeley), Caffe2 (Facebook), **Theano** (Bengio), Torch (Facebook), **Pytorch** (Facebook), **Tensorflow** (Google)

Outline

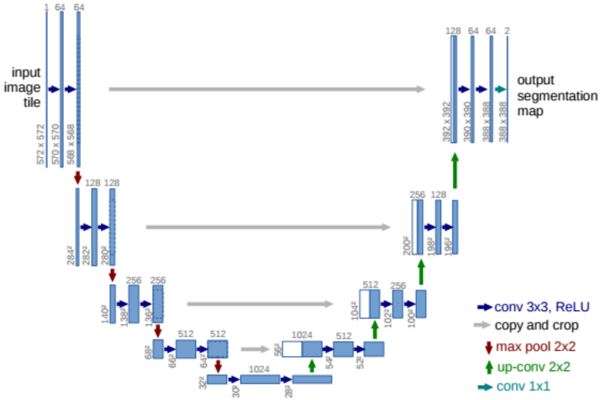
Basic Operations in CNNs

Standard operations in a Convolutional Neural Network:

- Convolution
- Pooling
- Batch Normalization
- Nonlinear Activation
- Others: Deconvolution, Upsampling, Skip Connections *etc.*

A Quick Example: UNet

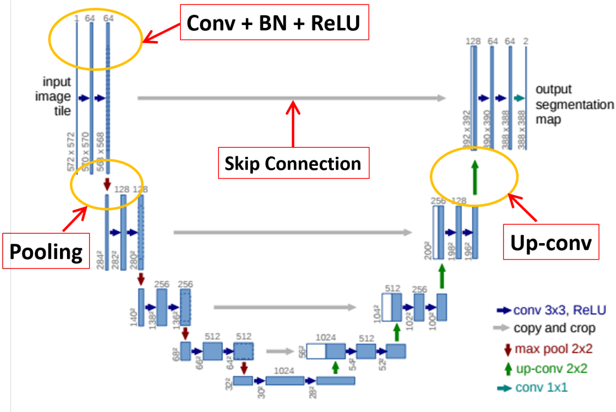
One of the most popular networks in semantic segmentation.



Olaf Ronneberger, Philipp Fischer, Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*, MICCAI 2015.

A Quick Example: UNet

One of the most popular networks in semantic segmentation.



Olaf Ronneberger, Philipp Fischer, Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*, MICCAI 2015.

Convolution and Convolution Layer (Conv + BN + ReLU)

Keywords in Convolution: • **Kernel Size** • **Stride** • **Padding**

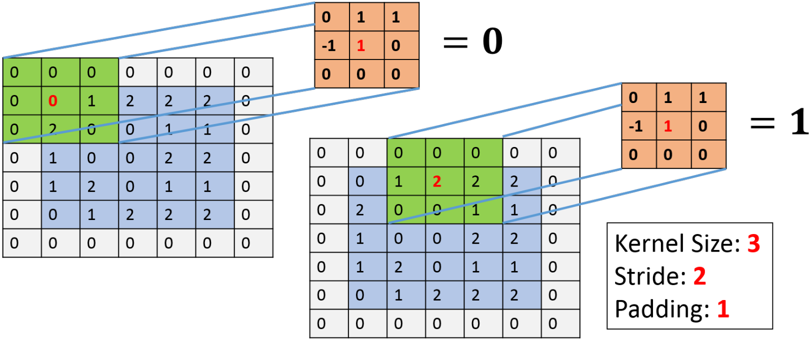


Figure: Illustration of Convolution[1].

[1] To make it simple, the kernel is already **rotated**. Only point-wise product and summation is needed

Convolutions in CNNs: (Conv + BN + ReLU)

Different types of convolutions.

- **Convolution**: (with/without) padding, $(1/>1)$ stride.
- **Transposed Convolution (Deconvolution)**: (with/without) padding, $(1/>1)$ stride.
- **Dilated Convolution**.

See the attached HTML file.

Helpful reading: Vincent Dumoulin, Francesco Visin. *A guide to convolution arithmetic for deep learning.*

Convolutional Layer (1st): (Conv + BN + ReLU)

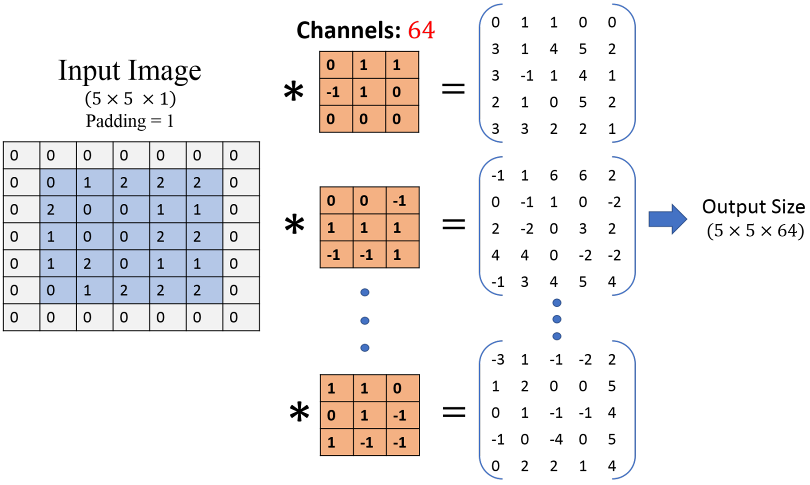


Figure: First Layer of CNN[2].

[2] Again, all the kernels are already **rotated**. Only point-wise product and summation is needed.

Convolutional Layer (2nd): (Conv + BN + ReLU)

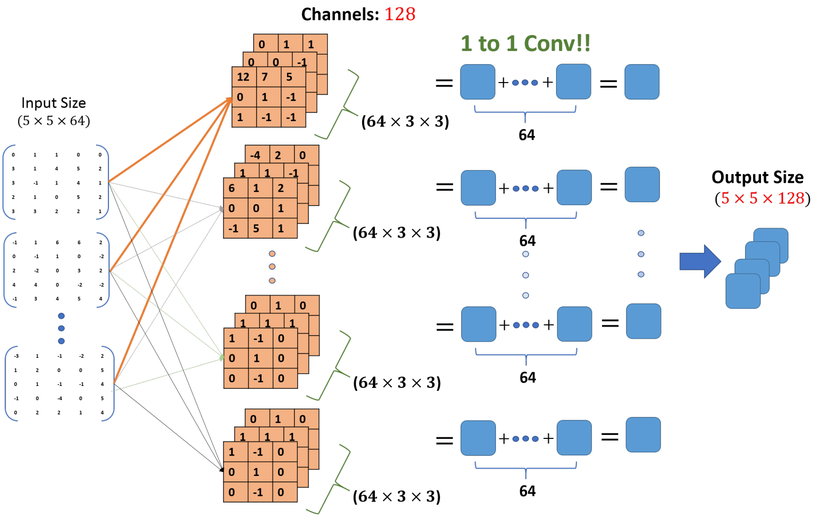


Figure: Each channel has 64 different 3×3 filters. Each filter convolves with only *one* channel of the input feature map!

A Short Break: A few questions ...

Suppose we have $4 \times 512 \times 1$ image as network input. That is, (batch size) 4 images where each of them is $512 \times 512 \times 1$ (gray images). Then,

- how many parameters (numbers in filters) do we have so far for the first two layers?

A Short Break: A few questions ...

Suppose we have $4 \times 512 \times 1$ image as network input. That is, (batch size) 4 images where each of them is $512 \times 512 \times 1$ (gray images). Then,

- how many parameters (numbers in filters) do we have so far for the first two layers?
  - $64 \times 3 \times 3 + 128 \times 64 \times 3 \times 3 = 576 + 73728 = 74,304$

A Short Break: A few questions ...

Suppose we have $4 \times 512 \times 1$ image as network input. That is, (batch size) 4 images where each of them is $512 \times 512 \times 1$ (gray images). Then,

- how many parameters (numbers in filters) do we have so far for the first two layers?
  - $64 \times 3 \times 3 + 128 \times 64 \times 3 \times 3 = 576 + 73728 = 74,304$
- since both the batches, feature maps are stored in memory, how much memory do we need? (suppose padding = 1, stride = 1)

$$
\begin{aligned}
&4 \times 512 \times 512 \times 1 + 4 \times 512 \times 512 \times 64 + 4 \times 512 \times 512 \times 128 \\
&\approx 1M + 67M + 134M = 202M \\
&= 202M \times 4 \text{ bytes} \approx 770\text{MB} \quad (202M \times 4/1024^2)
\end{aligned}
$$

Some background. Almost all deep learning models are trained on GPUs. A typical GPU now has $6 \sim 8$ GB memory. Advance GPUs has 12 GB memory (*e.g.* Nvidia GeForce GTX TITAN Z $\sim \$1.5K$ on amazon).

Batch Normalization (Conv + BN + ReLU)

In practice, to increase the training as well as testing speed, we usually feed **multiple** images to the network. The following figure shows a training batch of 4 images,

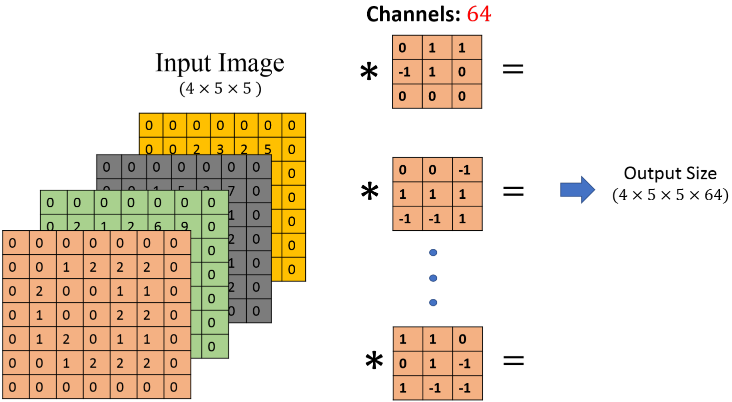


Figure: Batch Size is 4. Each Image is independently processed.

Batch Normalization : (Conv + BN + ReLU)

For each ***channel***, normalize the layers. Mean bad variance are computed across all the values in each channel.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

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

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

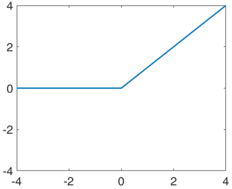
An effective way to resolve ***vanishing gradient*** problem!

Sergey Ioffe, Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*, NIPs 2015.

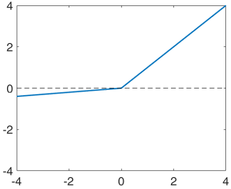
Nonlinear Activations: (Conv + BN + ReLU)

Popular nonlinearities used through all **but** last layer:

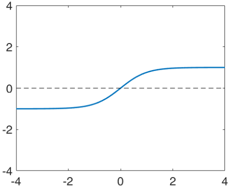
- ReLU: $\max(0, x)$.
- Leaky ReLU: $\max(0, x) + \gamma^2 \min(0, x)$
- Tanh: $\frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$
- Others [3]: ELU, SELU, PRELU, Threshold *etc.*



(a) ReLU



(b) Leaky ReLU



(c) Tanh

[3] A good place to find all these is the document of deep learning software frameworks, eg. `http://pytorch.org/docs/master/nn.html`

Pooling

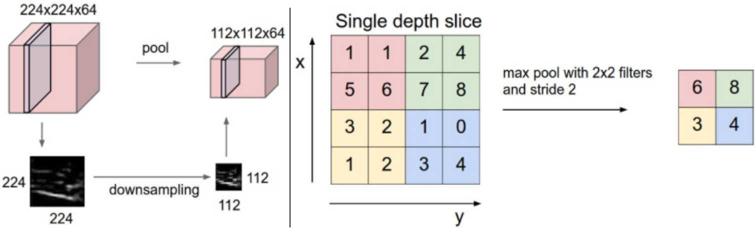


Figure: Illustration of *Max Pooling*. Here batch size 1 is omitted. [4]

There is also average pooling which takes average rather than maximum.

[4] Image courtesy of CS231n: Convolutional Neural Networks for Visual Recognition. Stanford.

Skip Connection

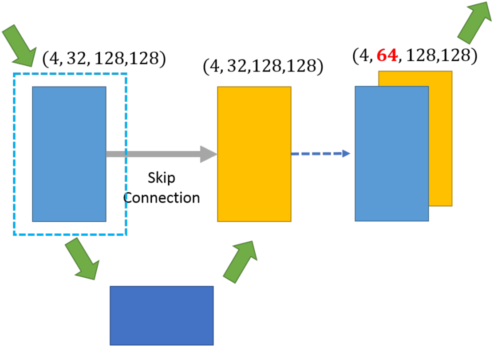


Figure: Skip connection is simply concatenating two feature maps (along channel dimension). Central crop is performed is there is a miss-match of the dimension.

Comment. Adding skip connections can usually increase the performance (at least) in segmentation.

Dense Layer

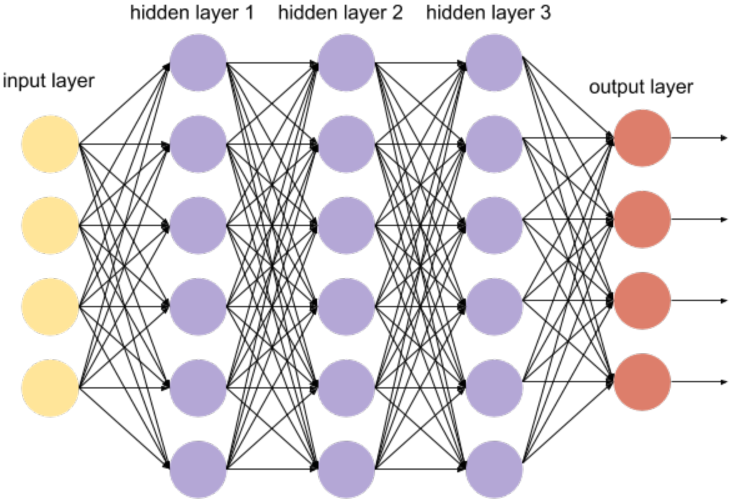


Figure: Dense Layers

Now We Know $(+ - \times \div)$ ...

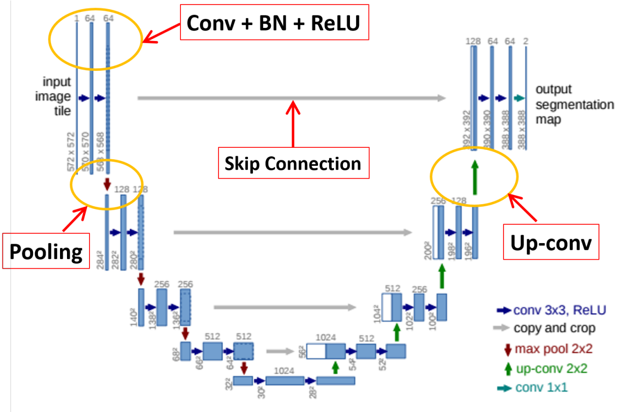
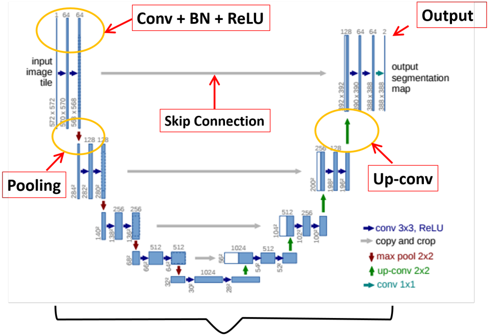


Figure: Networks are just special ways to stack all these operations.

Function Representation



$$f(I;W)$$

Figure: The entire network is just another representation of $f(I;W)$, where I is input image(s), W are the parameters in the network.

The Logics of Deep Learning (Supervised Training)

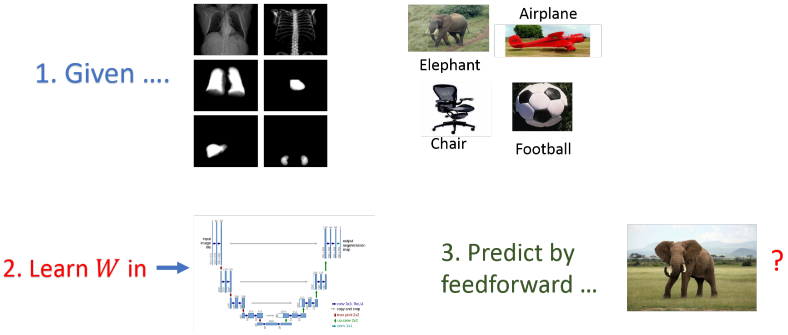


Figure: Working Logics of Supervised Deep Learning. (Training vs. Testing)

A standard workflow includes training → validation → testing.

Optimization Problem

In training state, we are given ground truth images and its labels for recognition/classification, or masks for segmentation, high-resolution image for super-resolution, clear image for de-noising *etc.*.

Let I be the truth images and g be their labels, the optimization problem is

$$\min_{W} \quad Loss\{f(I;W) - g\}$$

Popular losses,

- l_2 **distance**: image denoising, super-resolution, recognition.
- **Entropy**: binary/categorical cross-entropy, KL-divergence for segmentation.
- Many other **customized losses**, *e.g.* l_1 for GAN, smoothed dice loss for segmentation.

Loss Function Example: Cross Entropy

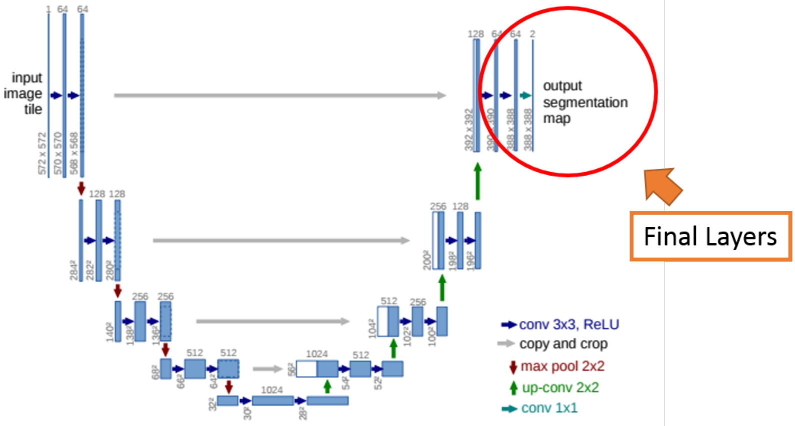


Figure: Let's look at what happens at the last layer (for single-object segmentation).

Loss Function Example: Cross Entropy

Suppose we are doing a single object segmentation, *e.g.* lung. The last convolutional channel will have 2 channels corresponding foreground and background.

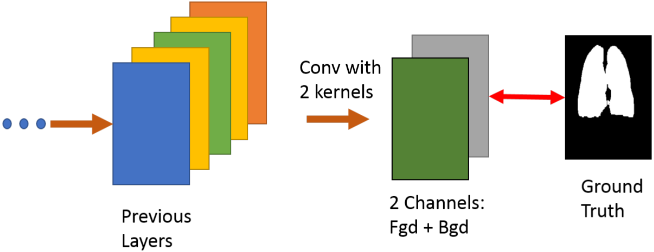


Figure: **Loss = Softmax + Cross-entropy**

Loss Function Example: Cross Entropy

Let $A = (F, B) \in \mathbb{R}^{256\times256\times2}$ be the output map with $F \in \mathbb{R}^{256\times256}$ and $B \in \mathbb{R}^{256\times256}$ are the foreground and background channels.

- Softmax:

$$S := Softmax(A)_{ij} = \frac{\exp(F_{ij})}{\exp(F_{ij}) + \exp(B_{ij})}$$

- Final Cross Entropy Loss:

$$\mathcal{L} = -\left(\frac{1}{256^2}\sum_{i=1}^{256}\sum_{j=1}^{256} y_{ij}\log(S_{ij}) + (1 - y_{ij})\log(1 - S_{ij})\right)$$

where y_{ij} is 1 if the pixel belongs to lung and 0 otherwise.

In **testing** phase, the prediction mask will be a simple comparison between foreground and background.

Code Samples

Code samples of Conv-BN-ReLU.

```
net['conv0_1'] = batch_norm(
Conv2DDNNLayer(net['input'],
num_filters=64, filter_size=3, pad='same',
W=HeNormal(gain='relu'), nonlinearity=rectify))

net['conv0_2'] = batch_norm(
Conv2DDNNLayer(net['conv0_1'], num_filters=64,
filter_size=3, pad='same',
W=HeNormal(gain='relu'), nonlinearity=rectify))
```

Outline

Optimization Routine of Deep Networks

The optimization of W is based on gradient descent algorithm.

1. Initialize the weights of the entire network as W_0, a learning rate α.
2. For $k = 1, 2, ...$
   1. Update W_k: $W_k = W_{k-1} - \alpha \nabla_W f(I; W_{k-1})$.
   2. if $mod(k, 10K) == 0$: $\alpha \leftarrow 0.8 * \alpha$.
3. Select the best iterations by cross-validation.

So, how can we calculate $\nabla_W f(I; W_{k-1})$?

Outline

Networks

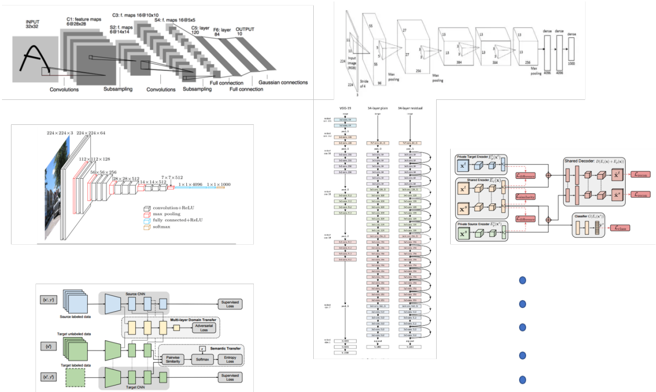


Figure: Networks are just special ways to stack all these operations.

Outline

Frameworks

Thank you!