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

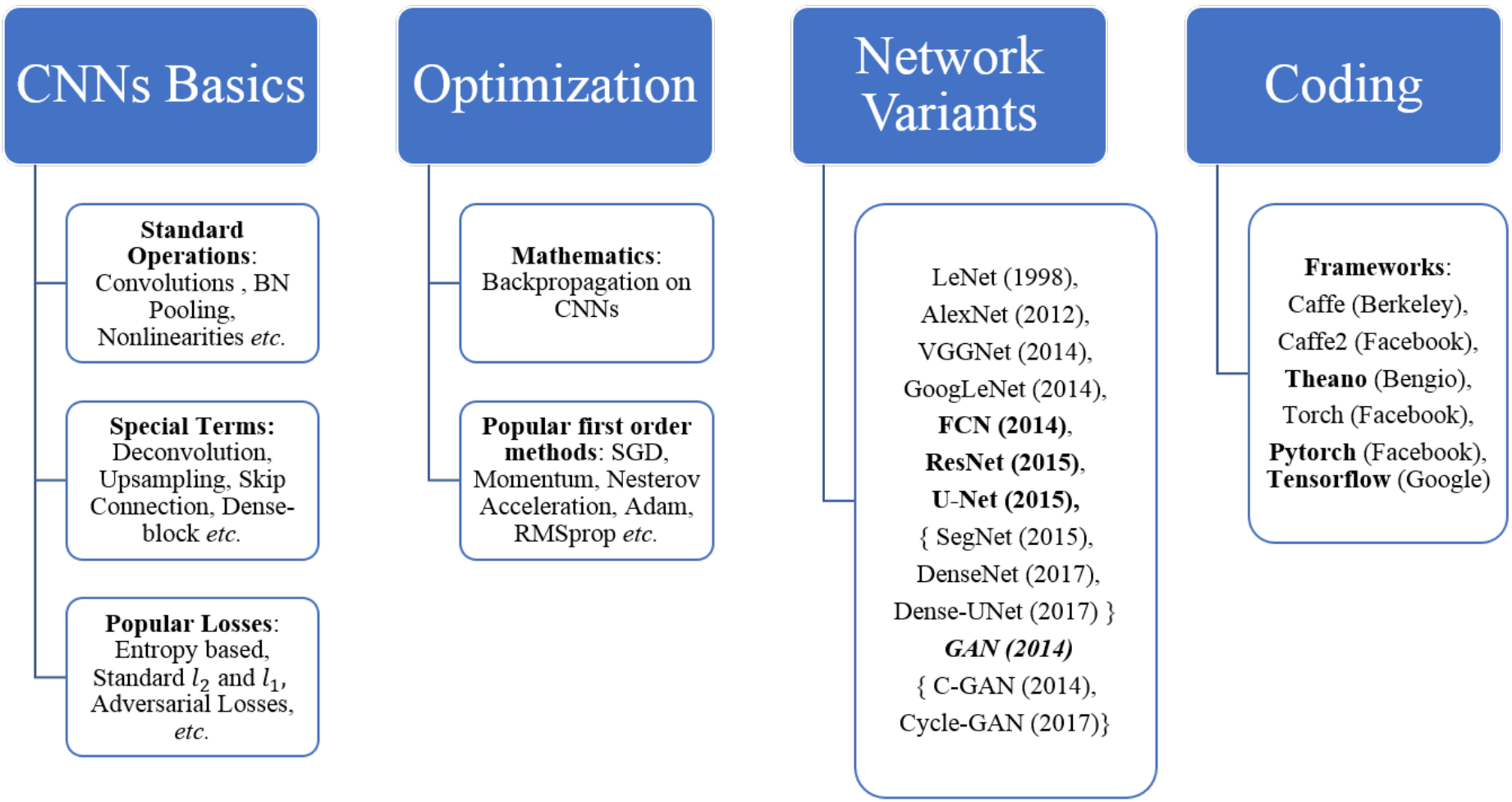
Yue Zhang

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

April 16, 2018

Outline

Overview of the Journey



Outline

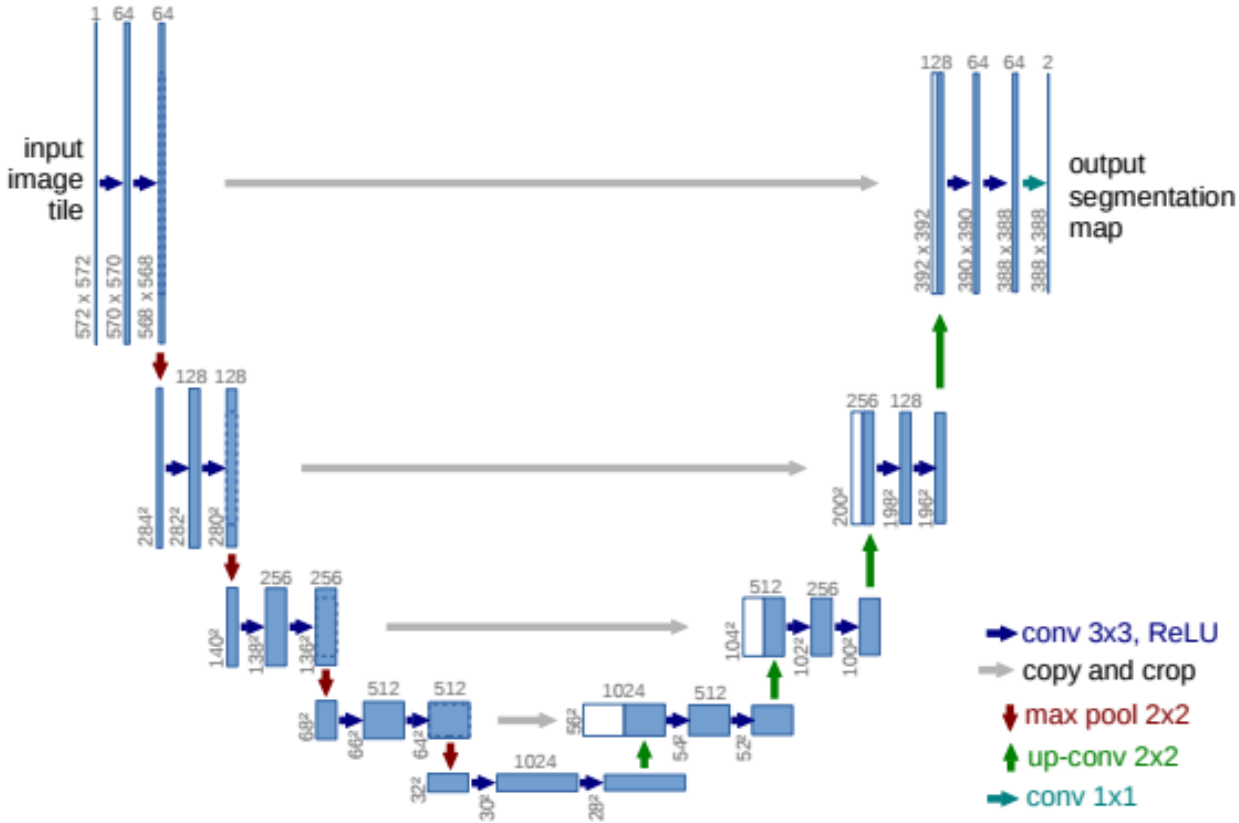
Basic Operations in CNNs

Standard operations in a Convolutional Neural Network:

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

A Quick Example: UNet

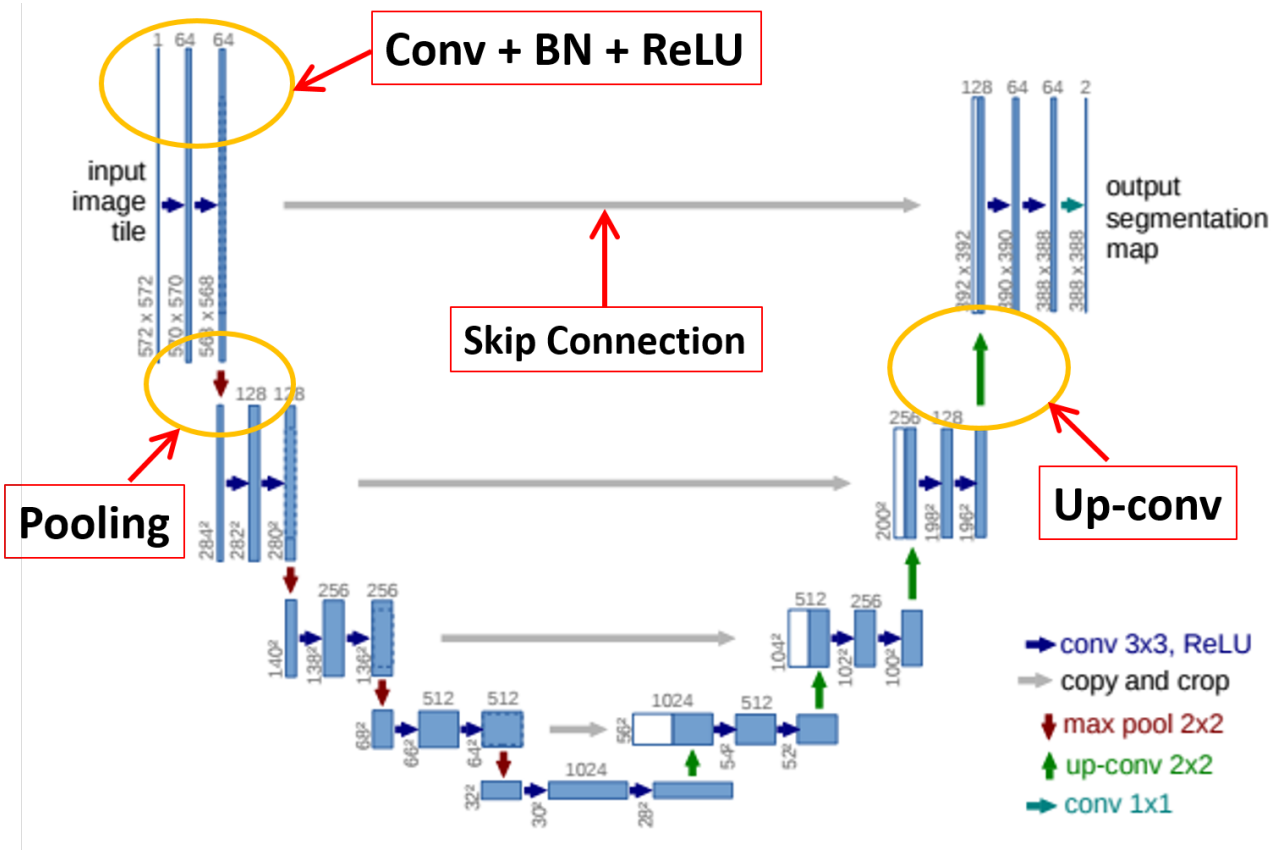
One of the most popular networks in semantic segmentation.



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**

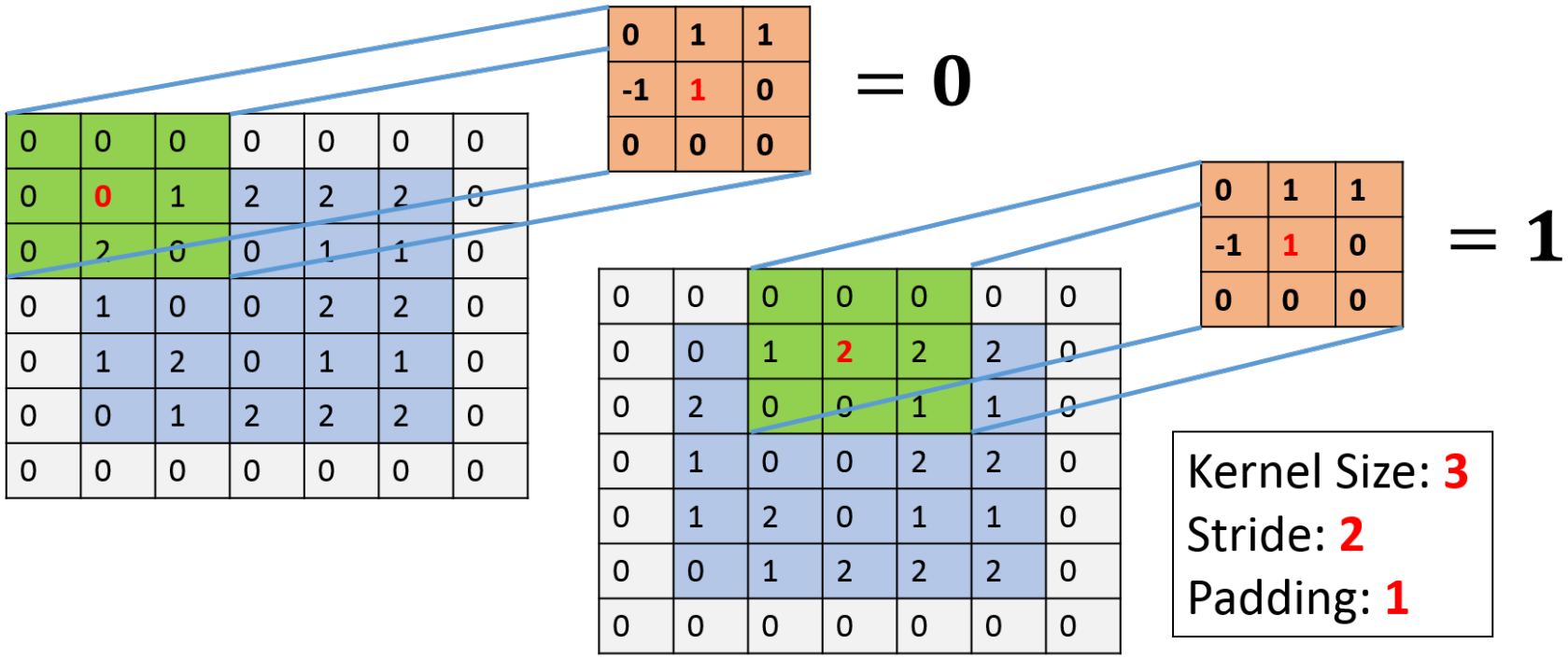


Figure: Illustration of Convolution[1].

[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)

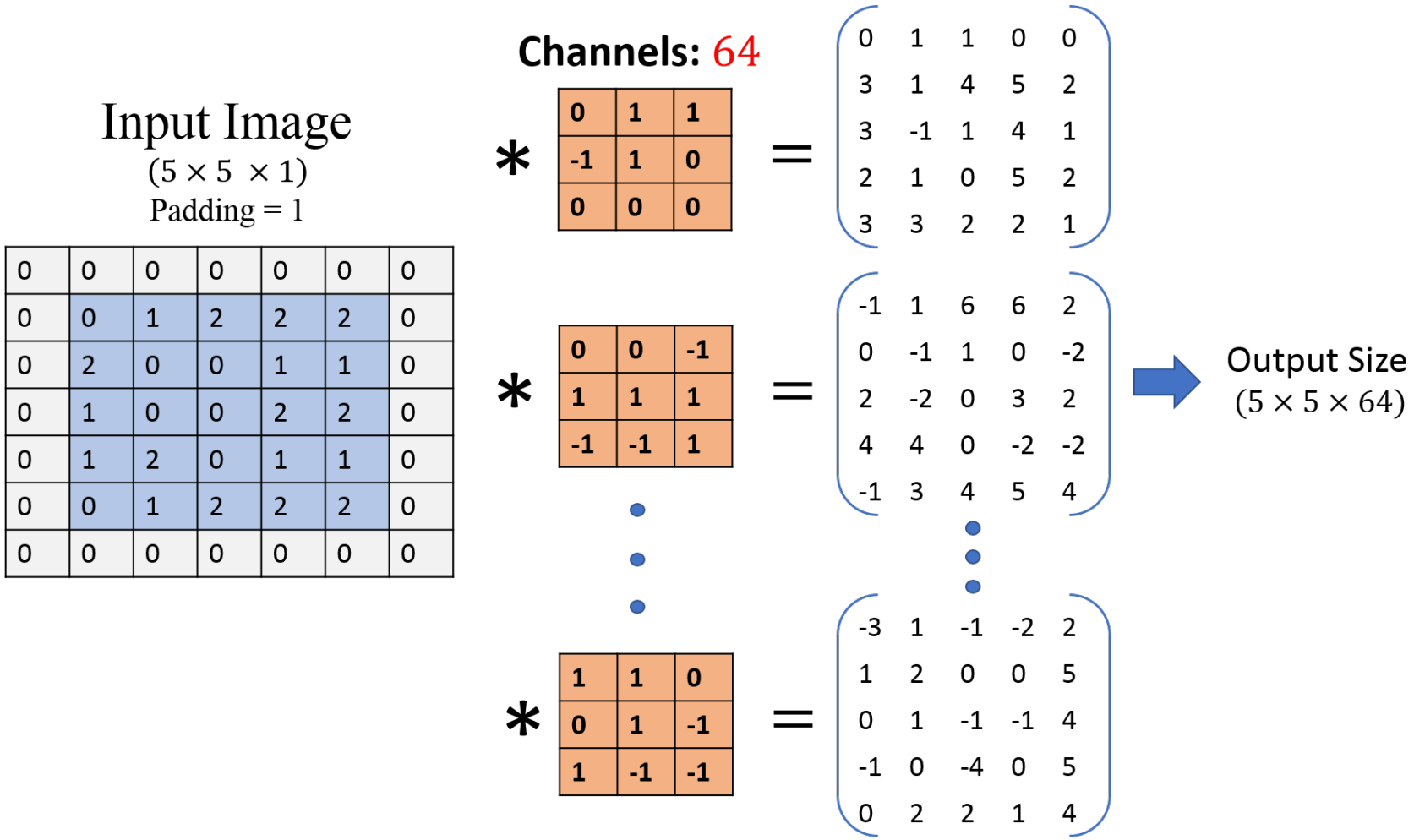


Figure: First Layer of CNN[2].

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

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

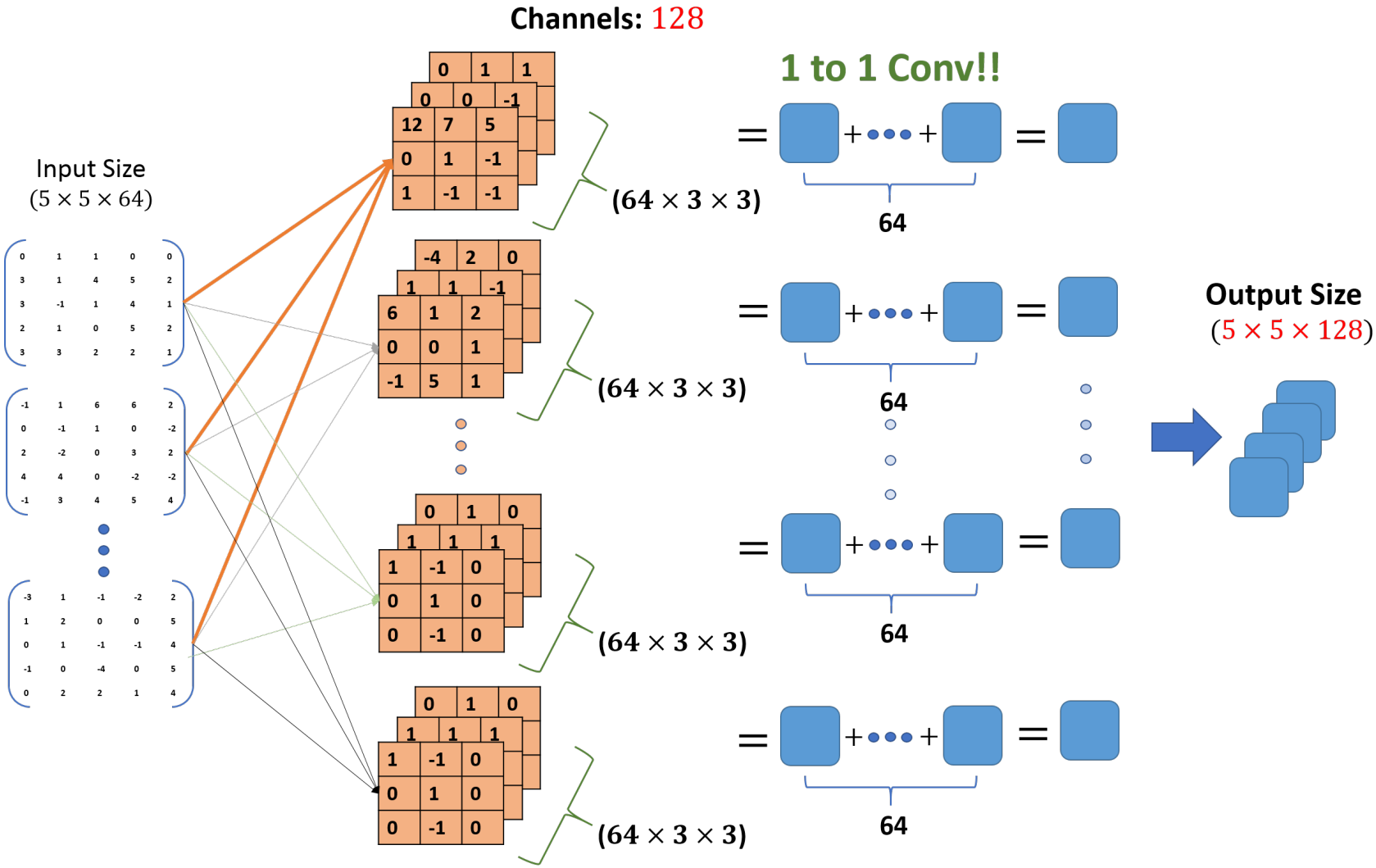


Figure: Each channel has 64 different 3×3 filters. Each filter convolves with only *one* channel of the input feature map!

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,

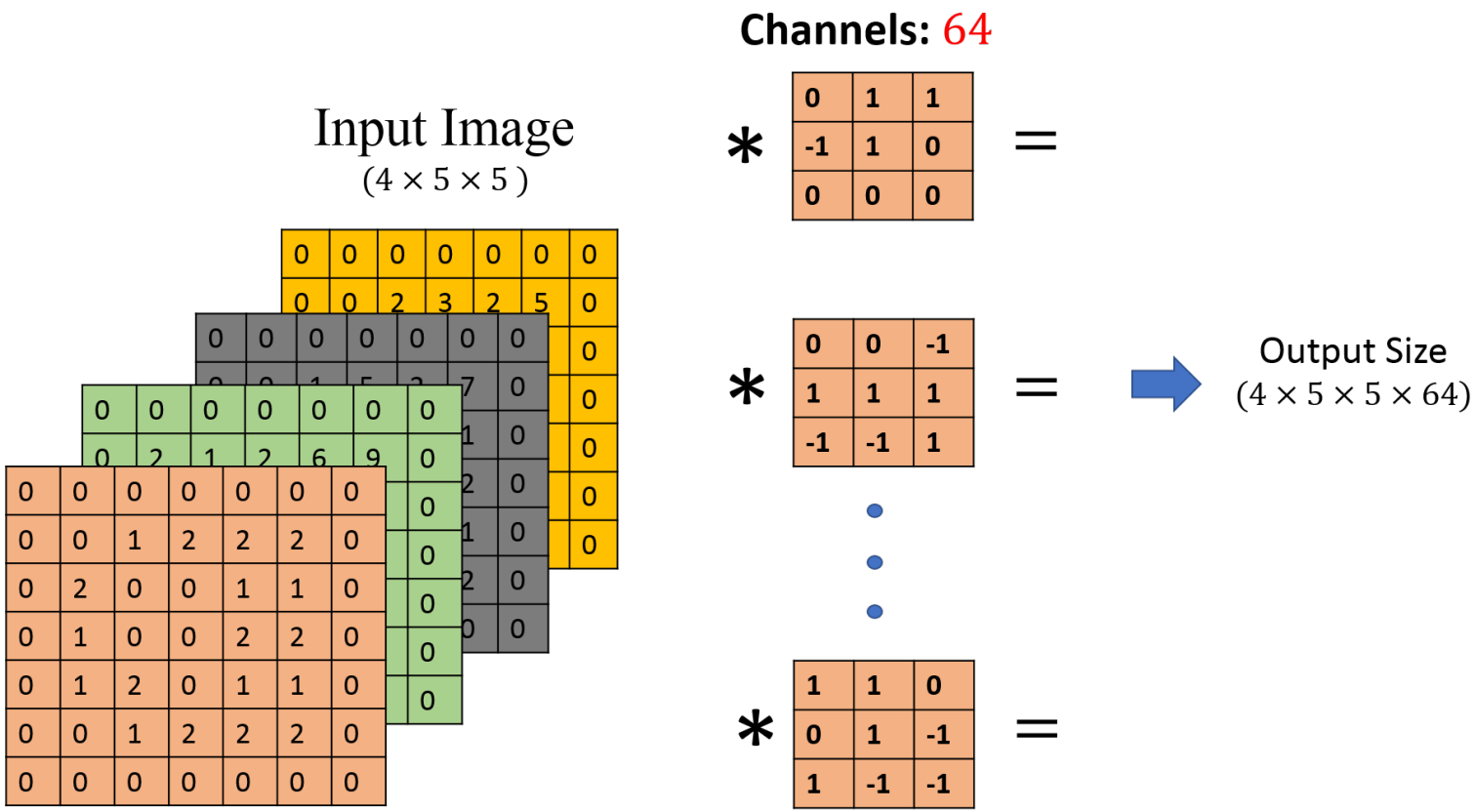


Figure: Batch Size is 4. Each Image is independently processed.

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: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

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

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

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

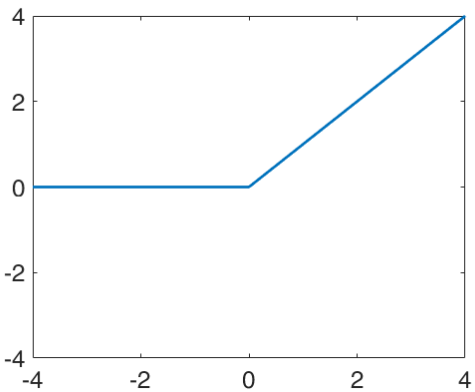
An effective way to resolve ***vanishing gradient*** problem!

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

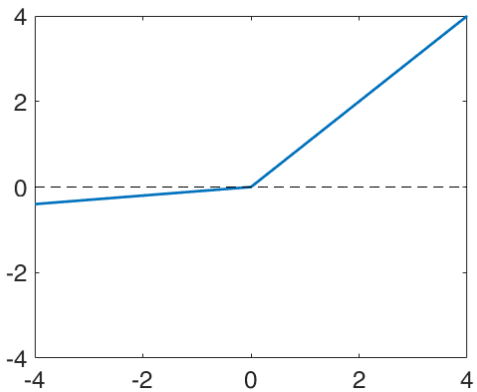
Nonlinear Activations: (Conv + BN + ReLU)

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

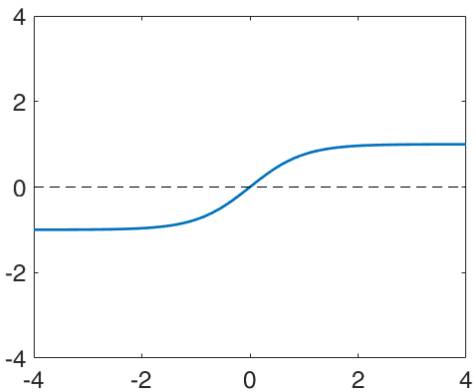
- ReLU: $\max(0, x)$.
- Leaky ReLU: $\max(0, x) + \gamma^2 \min(0, x)$
- Tanh: $\frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$
- Others [3]: ELU, SELU, PRELU, Threshold *etc.*



(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

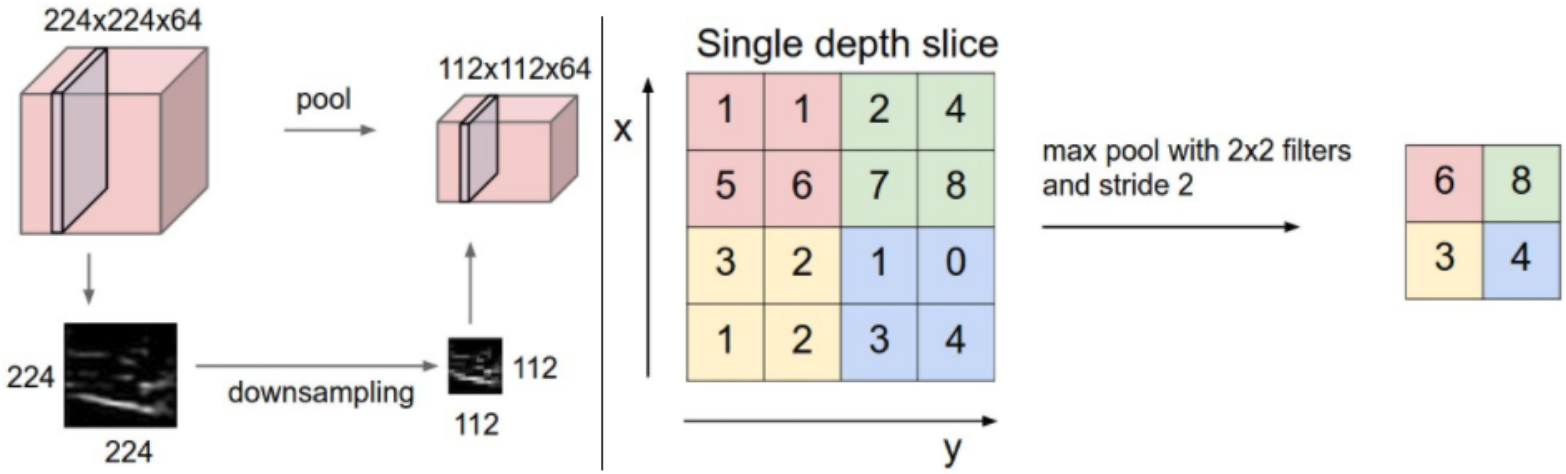


Figure: Illustration of *Max Pooling*. Here batch size 1 is omitted. [4]

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

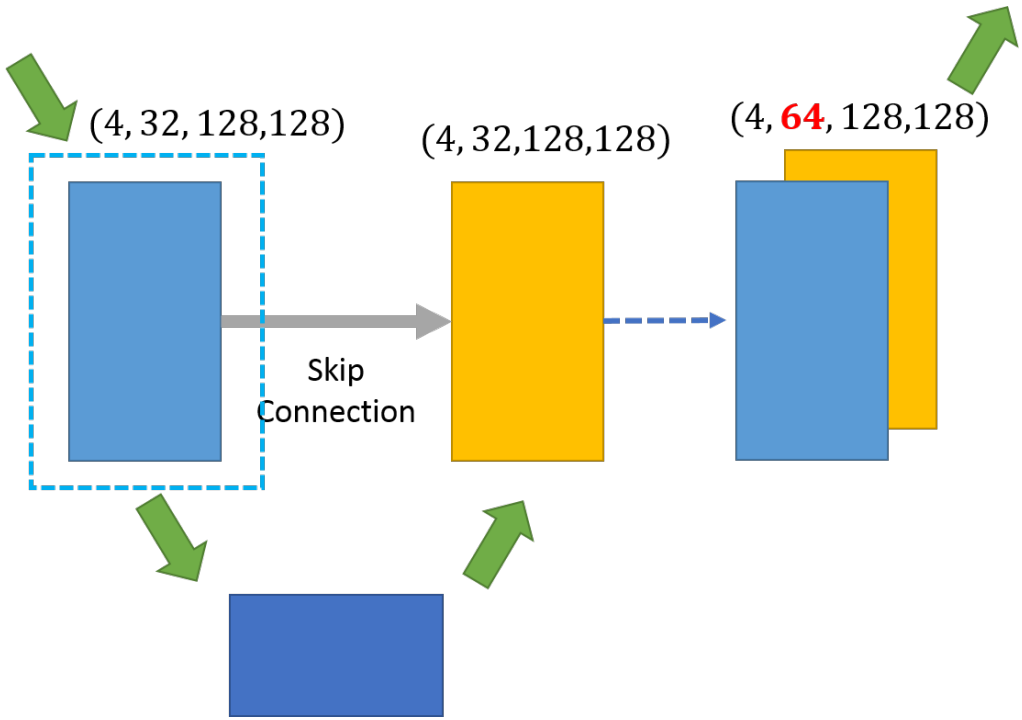


Figure: Skip connection is simply concatenating two feature maps (along channel dimension). Central crop is performed is there is a miss-match of the dimension.

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

Dense Layer

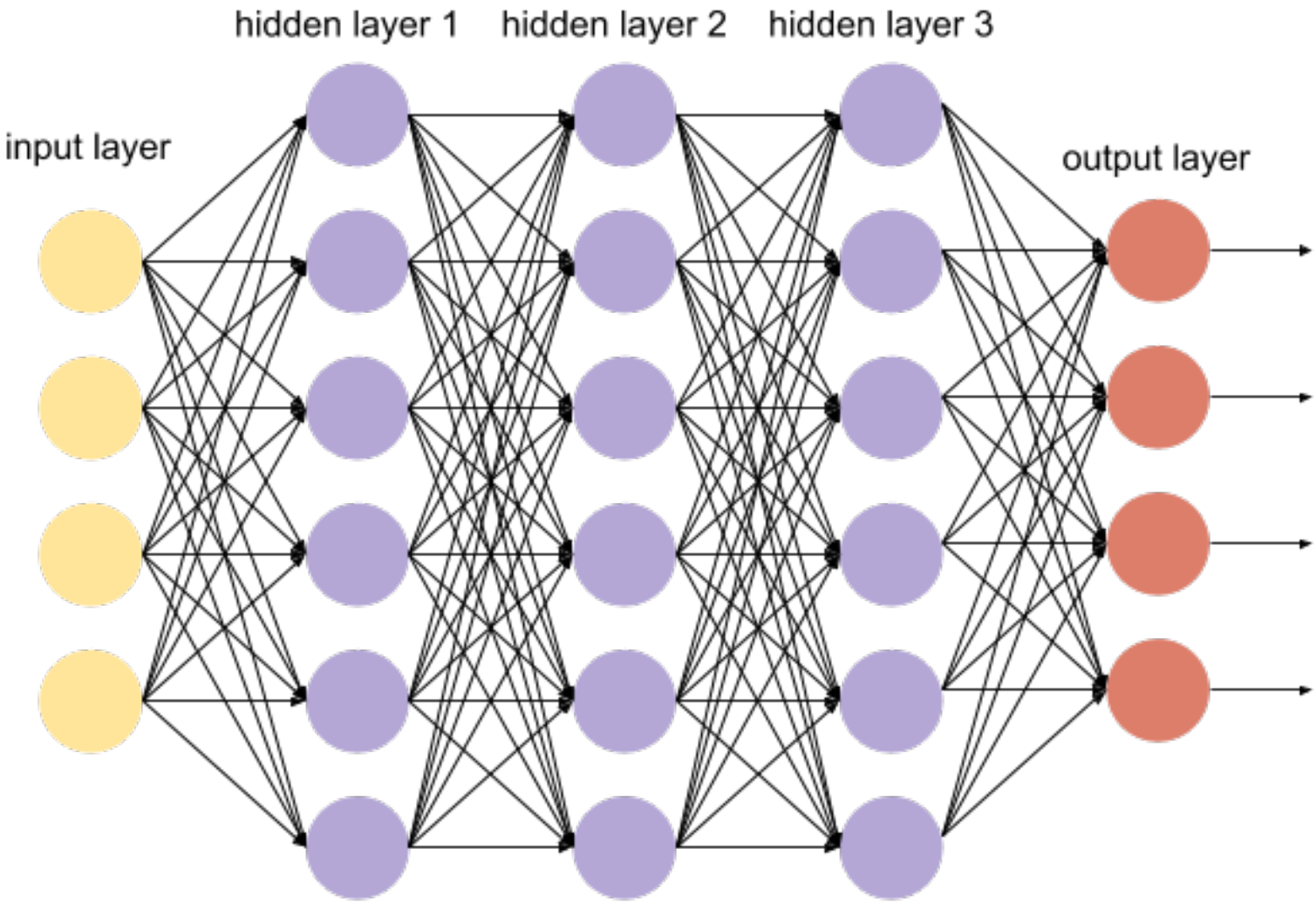


Figure: Dense Layers

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

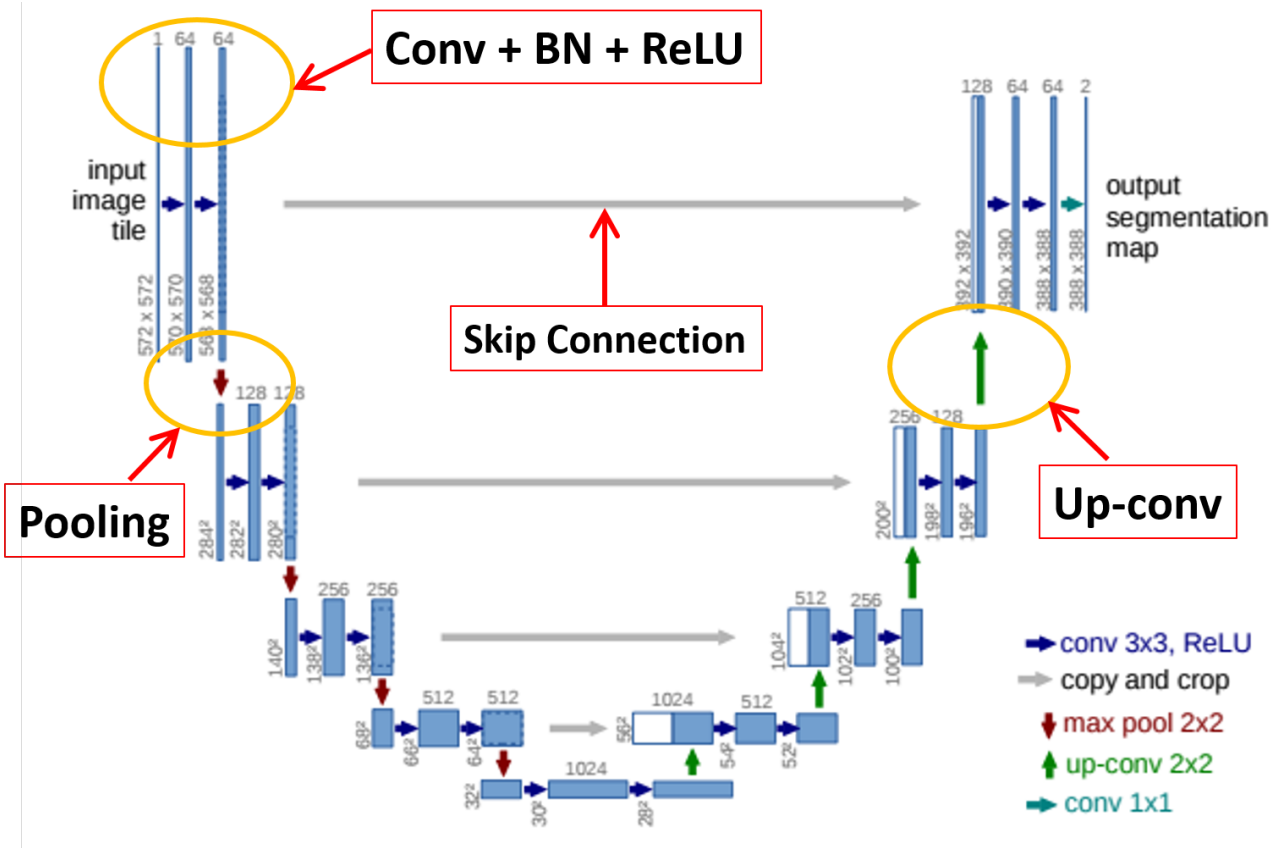
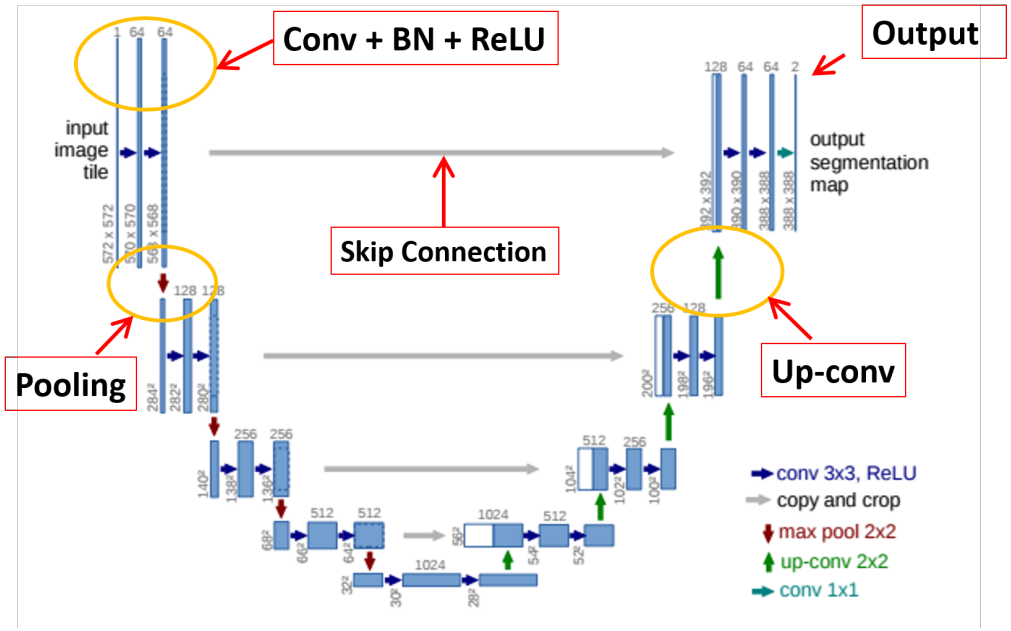


Figure: Networks are just special ways to stack all these operations.

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)

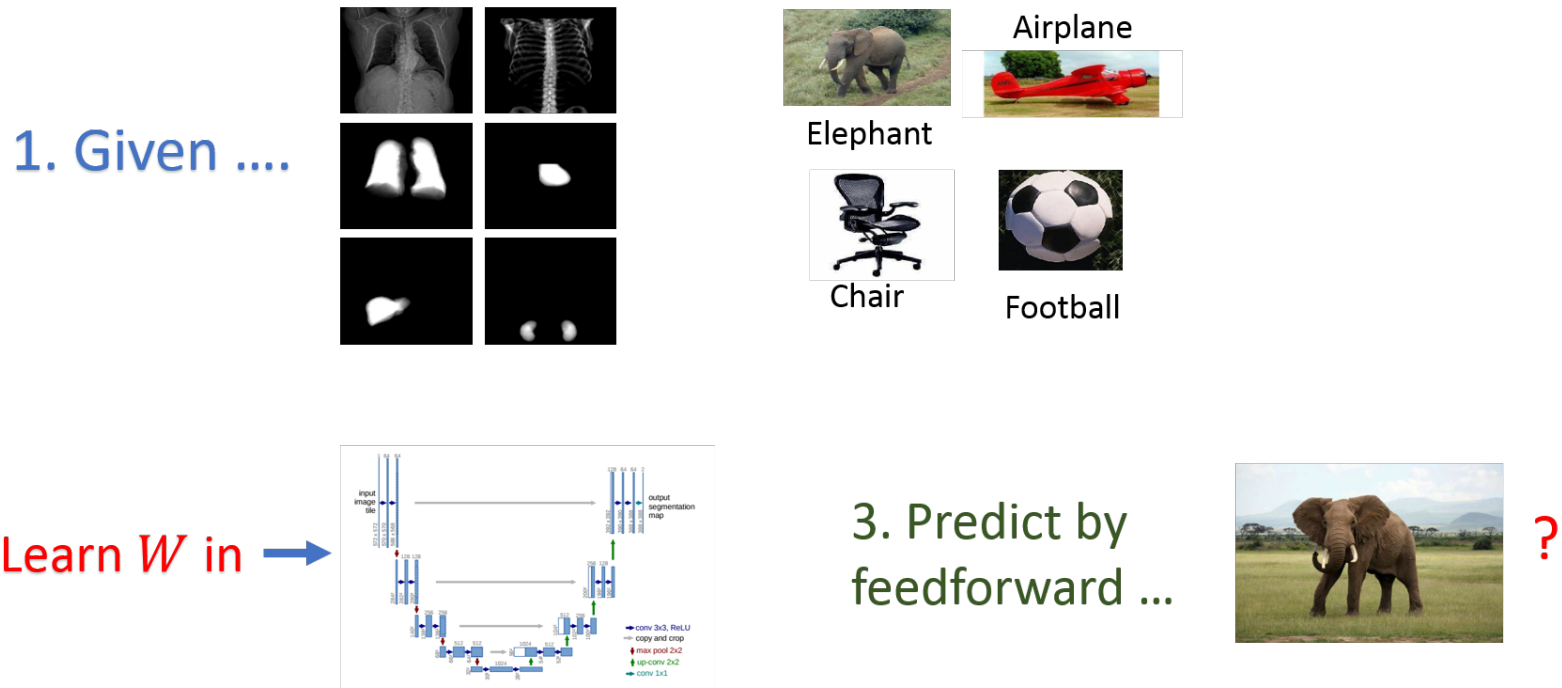


Figure: Working Logics of Supervised Deep Learning. (Training vs. Testing)

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

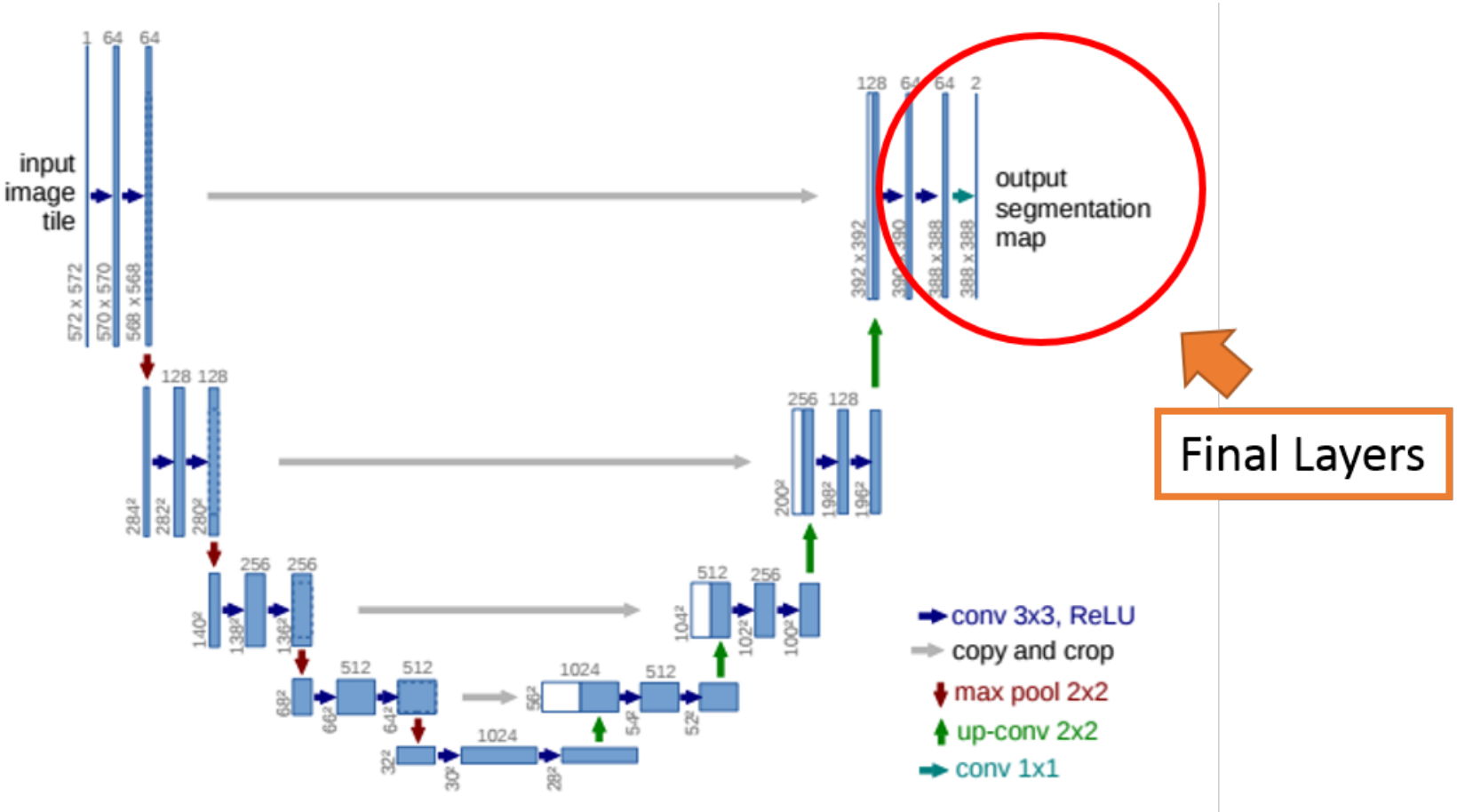


Figure: Let's look at what happens at the last layer (for single-object segmentation).

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.

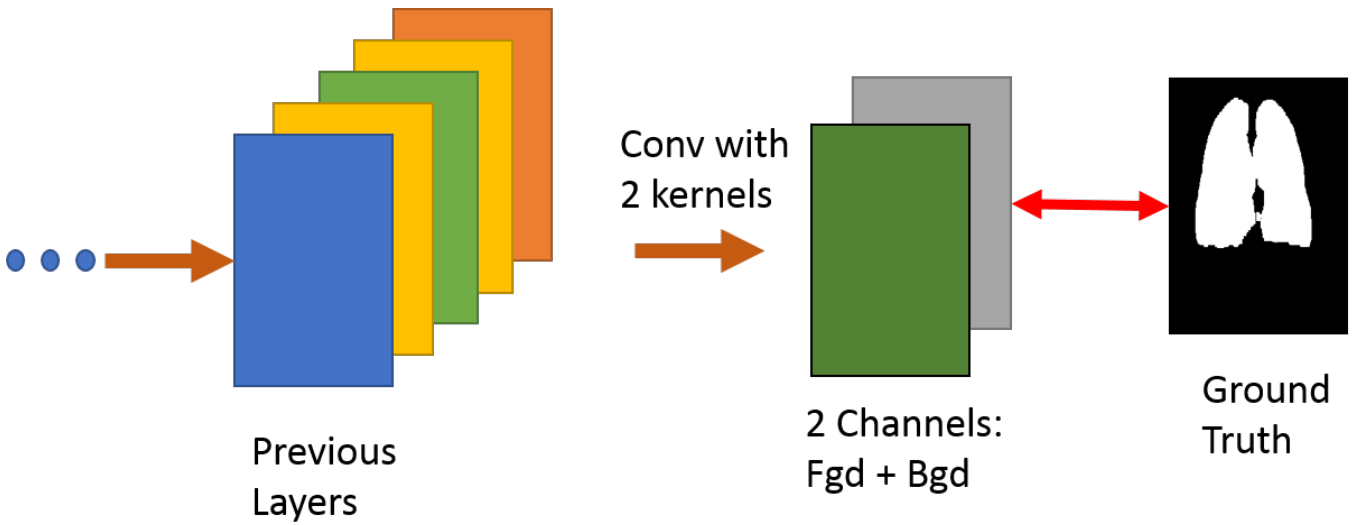


Figure: **Loss = Softmax + Cross-entropy**

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: $W_k = W_{k-1} - \alpha \nabla_W L(I; W_{k-1})$.
   2. if $mod(k, 10K) == 0$: $\alpha \leftarrow 0.8 * \alpha$.
3. Select the best iterations by cross-validation.

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

Backpropagation In Convolutional Neural Networks

Function $f(I;W)$ can be written as a series composition of operations, that is

$$f(\cdot;W) = C_n \circ \cdots C_2 \circ P_1 \circ R_1 \circ B_1 \circ C_1 \circ I_d$$

where I_d is identity, $C - B - R$ is conv-bn-relu, P is pooling. (Recall transposed convolution is also convolution, it therefore belongs to this general setting). For simplicity, consider the l_2 loss,

$$L = \frac{1}{2}\|f(I;W) - g\|_2^2$$

The gradient descent method requires the access to $\frac{\partial L}{\partial W}$, which is,

$$\begin{aligned}\frac{\partial L}{\partial W} &= \frac{\partial f}{\partial W}(f-g) \\ &= \frac{\partial}{\partial W}(C_n \circ \cdots \circ B_1 \circ C_1 \circ I_d)(f-g)\end{aligned}$$

Chain Rule

Recall the chain rule:

$$(f \circ g)' = (f' \circ g) \cdot g' \quad \text{for } f(x), g(x)$$

Or (with variable replacing trick).

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx} \quad \text{for } z = f(y), y = g(x)$$

And a straightforward consequence,

$$\frac{df}{dW} = \frac{df}{dC_n} \cdot \frac{dC_n}{dP_{n-1}} \cdot \frac{dP_{n-1}}{dR_{n-1}} \cdots \frac{dC_1}{dI} \quad \text{for } f = f(I; W)$$

We now study each of the factors, and a final output will be the products.

Backpropagation on Convolutions: $\frac{\partial f}{\partial C}$

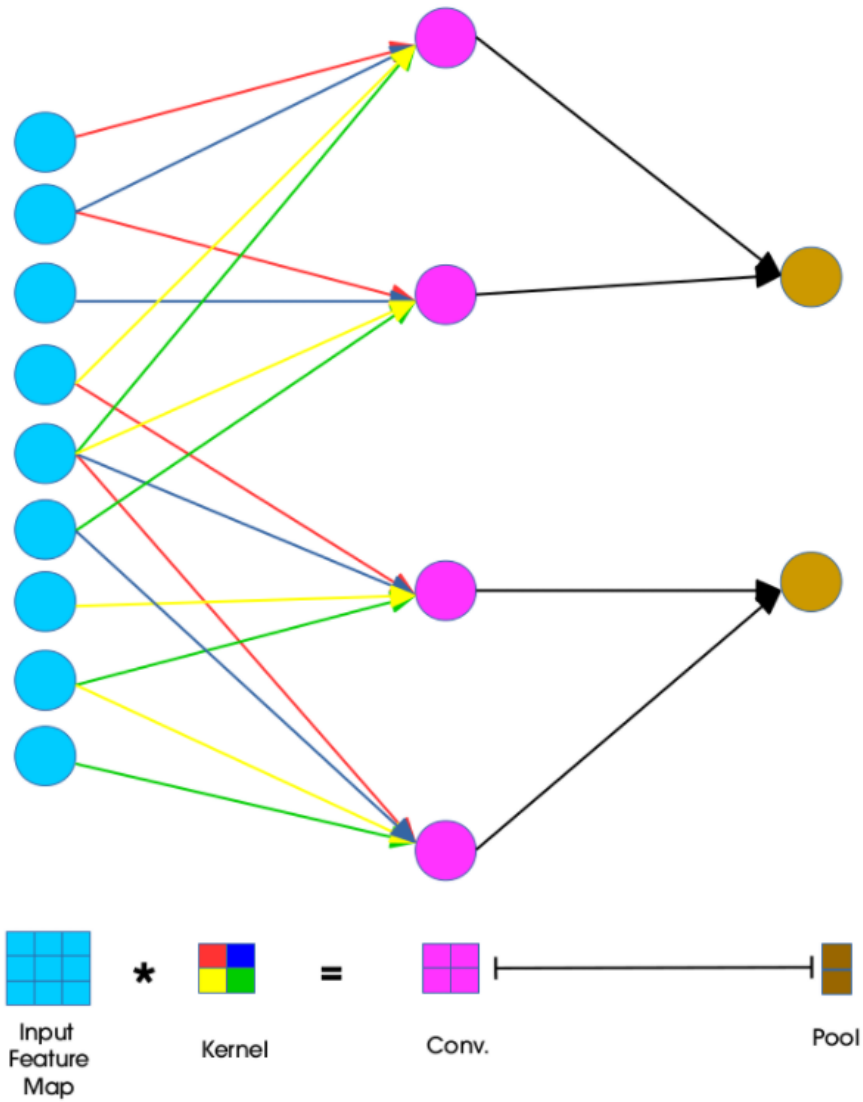


Figure: Convolution layer can be written as 'sparse' dense layers.

Backpropagation on Convolutions: $\frac{\partial f}{\partial C}$ 5

Derivatives of Convolution

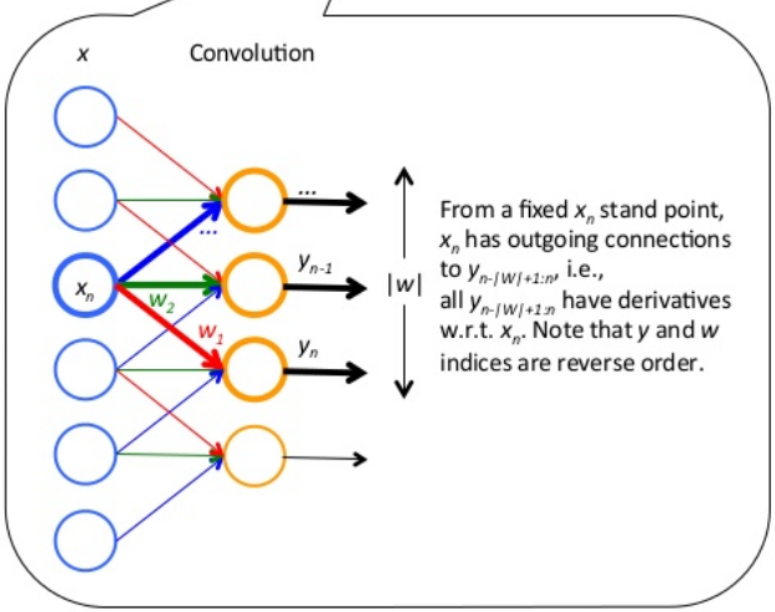
9 /14

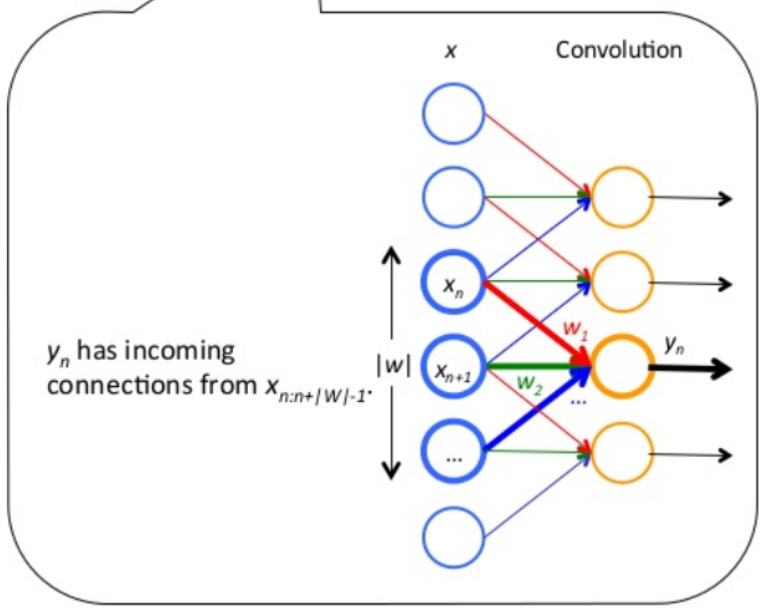
- Discrete convolution parameterized by a feature w and its derivatives

Let x be the input, and y be the output of convolution layer. Here we focus on only one feature vector w, although a convolution layer usually has multiple features $W = [w_1\ w_2\ \dots\ w_n]$. n indexes x and y where $1 \le n \le |x|$ for x_n, $1 \le n \le |y| = |x| - |w| + 1$ for y_n. i indexes w where $1 \le i \le |w|$. $(f*g)[n]$ denotes the n-th element of $f*g$.

$$y = x * w = [y_n], y_n = (x * w)[n] = \sum_{i=1}^{|w|} x_{n+i-1} w_i = w^T x_{n:n+|w|-1}$$

$$\frac{\partial y_{n-i+1}}{\partial x_n} = w_i, \quad \frac{\partial y_n}{\partial w_i} = x_{n+i-1} \quad \text{for } 1 \le i \le |w|$$





Backpropagation on Convolutions: $\frac{\partial f}{\partial C}$ 6

Backpropagation in Convolution Layer

10 /14

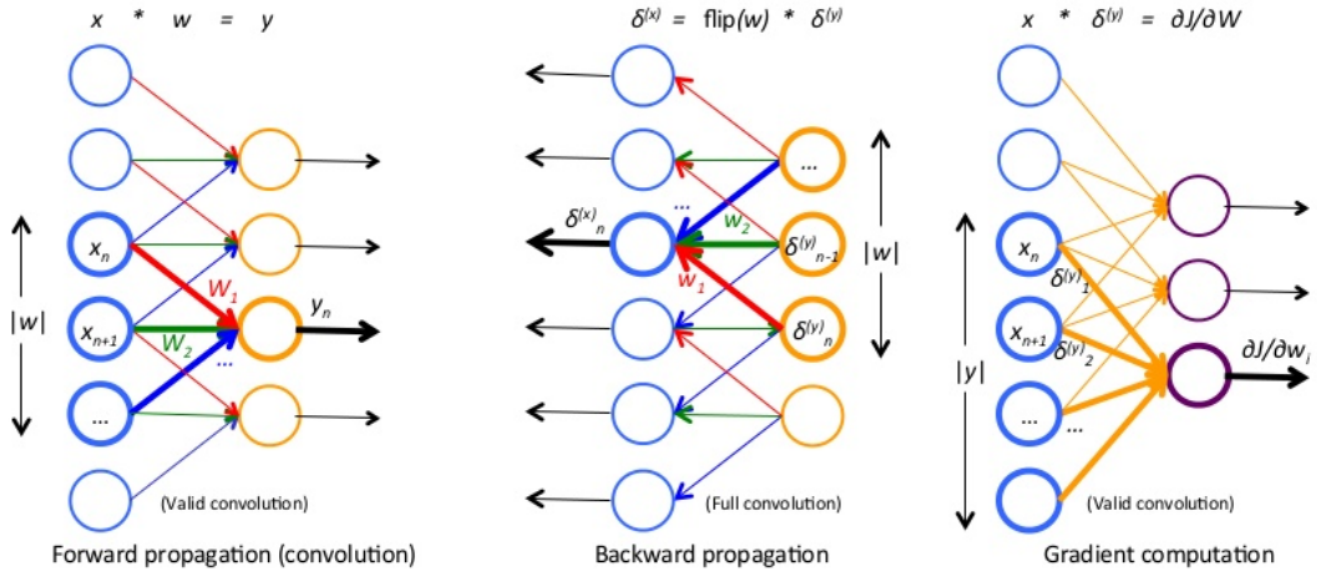
Error signals and gradient for each example are computed by convolution using the commutativity property of convolution and the multivariable chain rule of derivative.

Let's focus on single elements of error signals and a gradient w.r.t. w.

$$\delta_n^{(x)} = \frac{\partial J}{\partial x_n} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial x_n} = \sum_{i=1}^{|w|}\frac{\partial J}{\partial y_{n-i+1}}\frac{\partial y_{n-i+1}}{\partial x_n} = \sum_{i=1}^{|w|}\delta_{n-i+1}^{(y)} w_i = \left(\delta^{(y)} * \mathrm{flip}(w)\right)[n],\ \delta^{(x)} = \left[\delta_n^{(x)}\right] = \delta^{(y)} * \mathrm{flip}(w)$$

↑ Reverse order linear combination

$$\frac{\partial J}{\partial w_i} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial w_i} = \sum_{n=1}^{|x|-|w|+1}\frac{\partial J}{\partial y_n}\frac{\partial y_n}{\partial w_i} = \sum_{n=1}^{|x|-|w|+1}\delta_n^{(y)} x_{n+i-1} = \left(\delta^{(y)} * x\right)[i],\ \frac{\partial J}{\partial w} = \left[\frac{\partial J}{\partial w_i}\right] = \delta^{(y)} * x = x * \delta^{(y)}$$



Backpropagation on ReLU and Max-Pooling

- Since $R = \max(0, B)$, the derivative is defined piece-wisely as an indicator function,

$$\frac{\partial R}{\partial B} = \begin{cases} 0 & \text{if } B \leq 0 \\ \frac{\partial B}{\partial B} = 1 & \text{if } B > 0 \end{cases}$$

- Recall max-pooling is taking maximum of small neighborhood around each pixel, the derivate of it w.r.t. the previous layer will also be δ functions!

Backpropagation on BN [7]

Notation

Let's start with some notation.

- **BN** will stand for Batch Norm.
- f represents a layer upwards of the BN one.
- y is the linear transformation which scales x by γ and adds β.
- $\hat{x}$ is the normalized inputs.
- μ is the batch mean.
- σ^2 is the batch variance.

The below table shows you the inputs to each function and will help with the future derivation.

$f(y)$	$y(\hat{x}, \gamma, \beta)$	$\hat{x}(\mu, \sigma^2, x)$

Goal: Find the partial derivatives with respect to the inputs, that is $\frac{\partial f}{\partial \gamma}$, $\frac{\partial f}{\partial \beta}$ and $\frac{\partial f}{\partial x_i}$.

[7] https://kevinzakka.github.io/2016/09/14/batch_normalization/

Backpropagation on BN [8]

$$\frac{\partial f}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial f}{\partial y_i}$$

$$\frac{\partial f}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial f}{\partial y_i} \cdot \hat{x}_i$$

$$\frac{\partial f}{\partial x_i} = \frac{m\frac{\partial f}{\partial \hat{x}_i} - \sum_{j=1}^{m} \frac{\partial f}{\partial \hat{x}_j} - \hat{x}_i \sum_{j=1}^{m} \frac{\partial f}{\partial \hat{x}_j} \cdot \hat{x}_j}{m\sqrt{\sigma^2 + \epsilon}}$$

Figure: Details can be found from the link.

[8] https://kevinzakka.github.io/2016/09/14/batch_normalization/

First Order Methods in Deep Learning Optimization

Since we can obtain the gradient of $f(I; W)$ from the back-prop, we can minimize the loss using first order methods:

- SGD,
- Momentum,
- Nesterov accelerated gradient,
- Adam,
- RMSprop,
- Adagrad, ...

See `http://ruder.io/optimizing-gradient-descent/index.html`

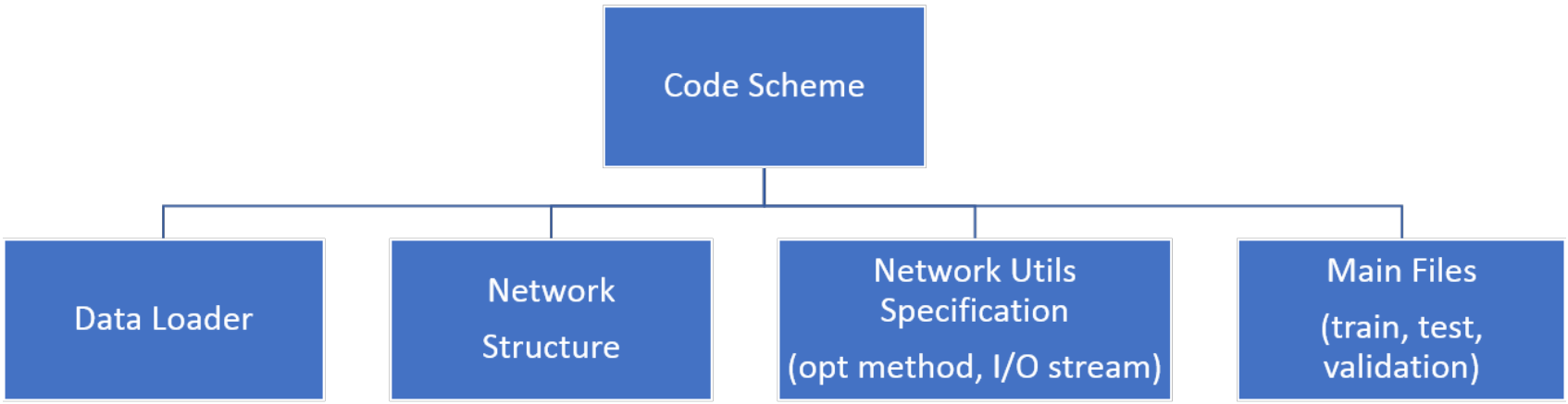
Outline

System/Software Requirements

- Python (preferred 3.0 and later).
- Theano + Lasagne:
  1. `http://deeplearning.net/software/theano/install.html`
  2. `https://github.com/Lasagne/Lasagne`

  Or Pytorch:
  1. OSX + Linux: `http://pytorch.org/`
  2. Windows: `https://github.com/peterjc123/pytorch-scripts`
- A GPU.

Popular Code Schemes



Here,

- **Data Loader:** a function file, loads data to (pytorch array);
- **Network Structure:** a function file, specifies network structure (UNet, Densenet *etc.*);
- **Network Utils Specification:** a function file, specifies optimization methods (adam, sgd, *etc.*) and input/output stream of the network;
- **Main Files:** the 3 running files: train, test, validation.

Data Loader (Pytorch)

- Load data → **augmentation** → return augmented images + labels

- It **requires** 3 functions:
  1. **init**: initialization
  2. **getitem**: load data and augmentation
  3. **len**: size of the data

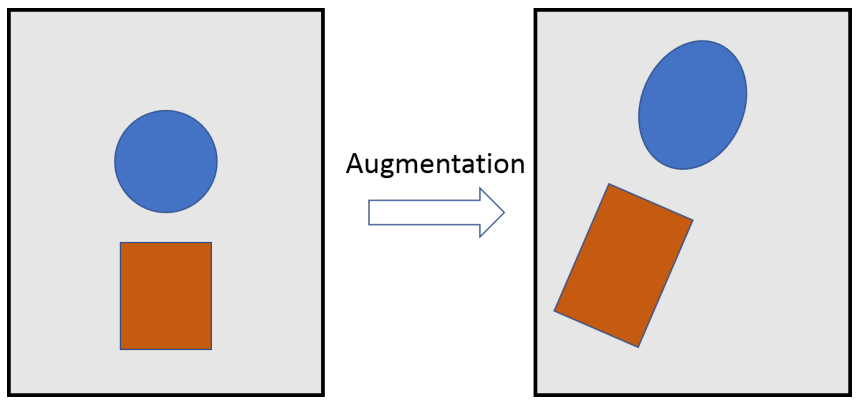


Figure: Illustration of Data Augmentation: translation+rotation+scaling

Network Structure (Lasagne)

This file contains the structure of the network:

- different layers: conv/deconv, pooling layer
- padding, stride, weight initialization
- batch normalization, nonlinear activation.

For Pytorch, a **forward** function has to be specified.

Networkio (Pytorch)

This file contains the necessary utility functions:

- specifies optimization method (learning rate, parameters),
- specifies the loss function and how it is computed,
- save the learned parameters to files,
- load the learned parameter from files.

Main Files (Pytorch)

Ideally, a program should include 3 main files:

- **train**: load training data, start training, save weights of each **epoch**.
- **validation**: load validation data, load the trained weights of each epoch, find the best one.
- **test**: load test data, load the best weights, report the testing accuracy.

Outline

Networks

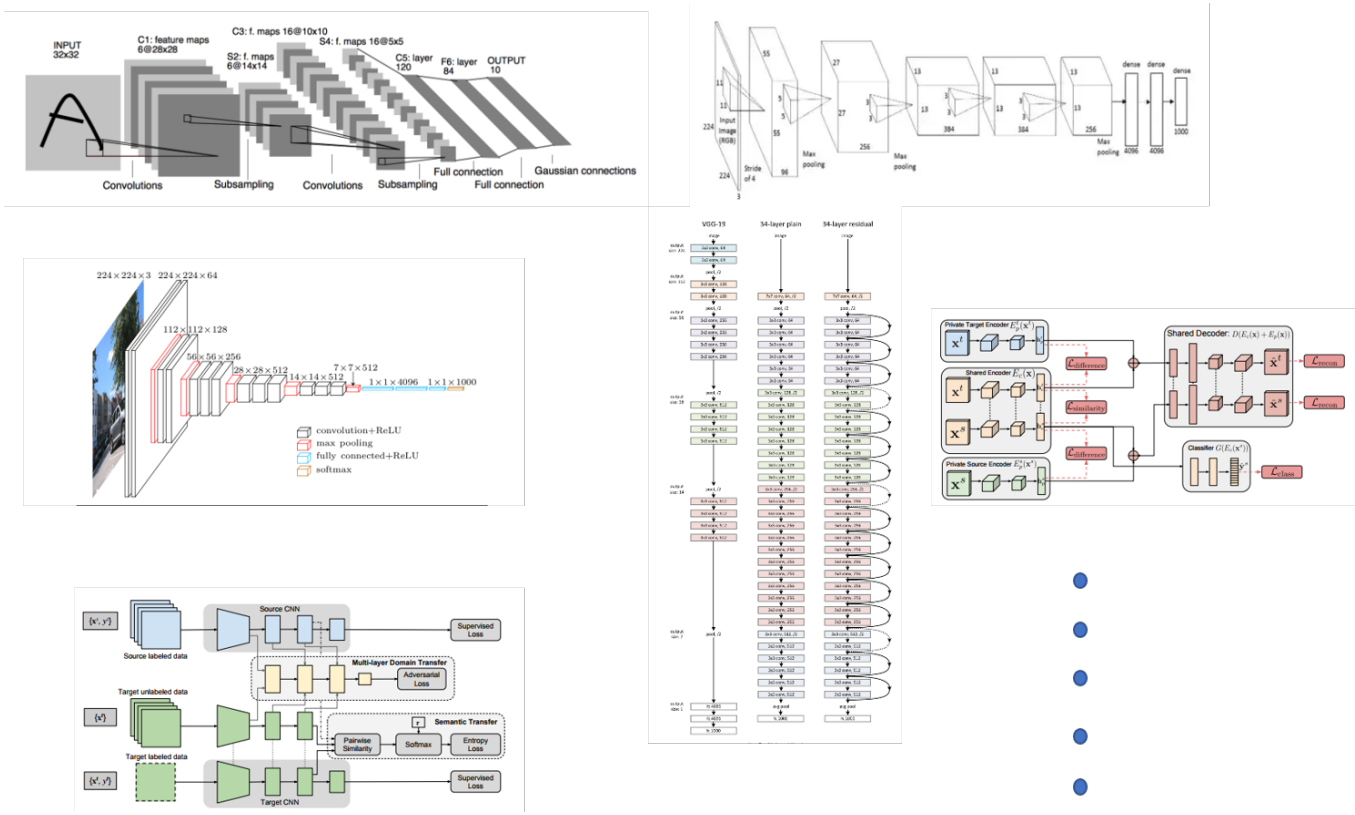


Figure: Networks are just special ways to stack all these operations.

Thank you!