Implementing Deep Learning using cuDNN

DEVIEW
2015

이예하
VUNO Inc.

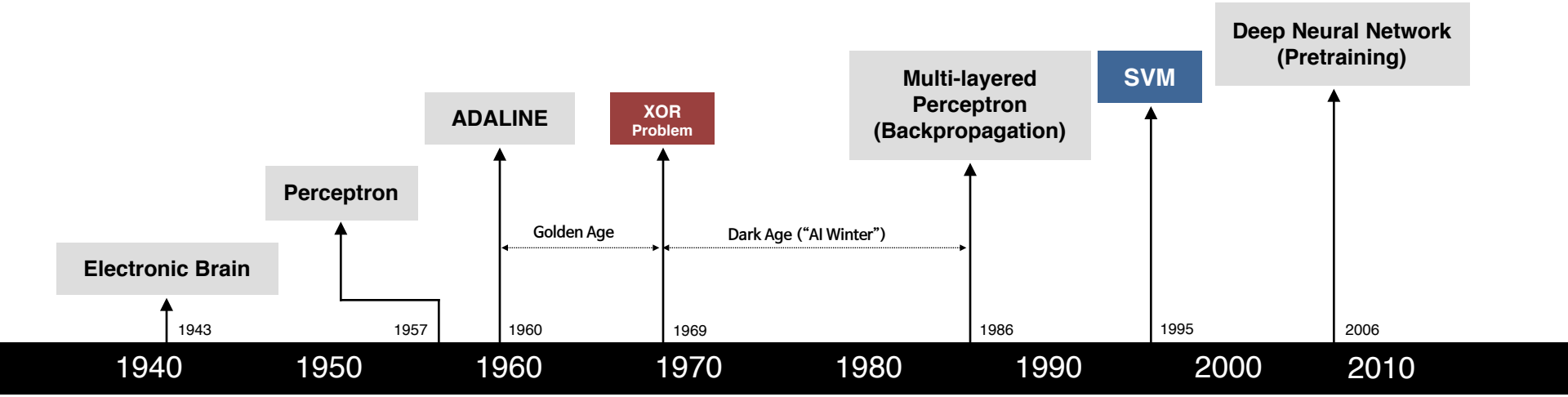
Contents

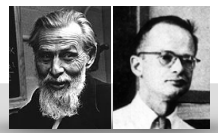
1. Deep Learning Review
2. Implementation on GPU using cuDNN
3. Optimization Issues
4. Introduction to VUNO-Net

1.
Deep Learning Review

Brief History of Neural Network

DEVIEW 2015





S. McCulloch - W. Pitts

F. Rosenblatt

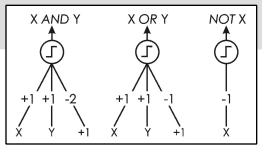
B. Widrow - M. Hoff

M. Minsky - S. Papert

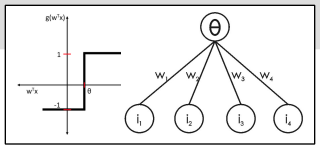
D. Rumelhart - G. Hinton - R. Wiliams

V. Vapnik - C. Cortes

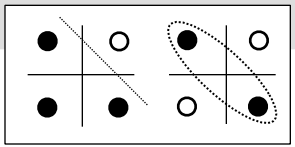
G. Hinton - S. Ruslan



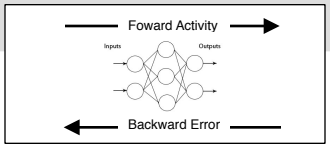
- Adjustable Weights
- Weights are not Learned



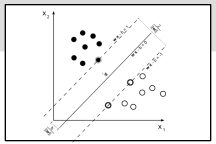
- Learnable Weights and Threshold



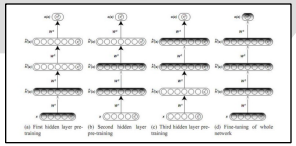
- XOR Problem



- Solution to nonlinearly separable problems
- Big computation, local optima and overfitting

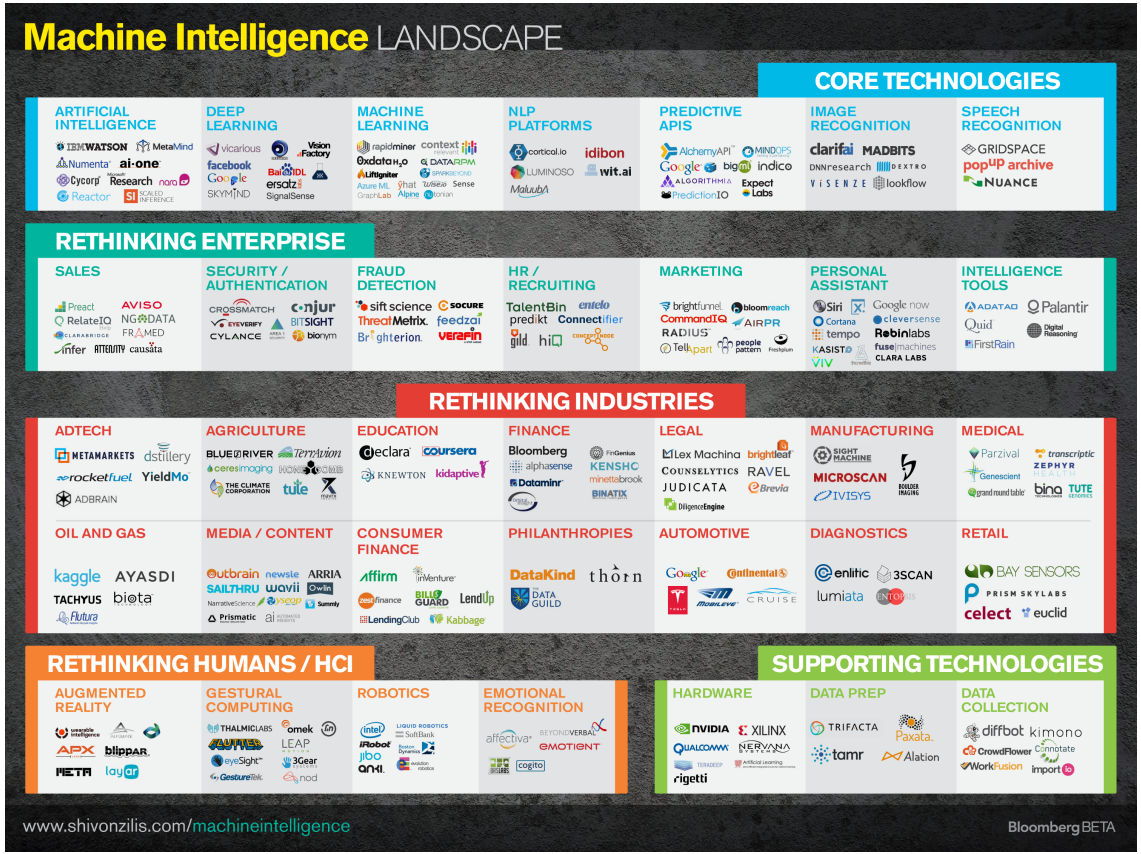


- Limitations of learning prior knowledge
- Kernel function: Human Intervention



- Hierarchical feature Learning

Machine/Deep Learning is eating the world

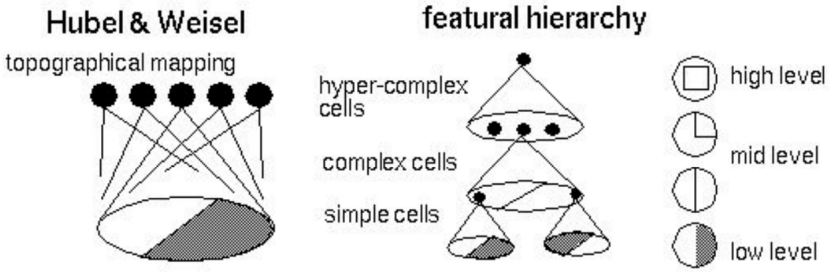


Building Blocks

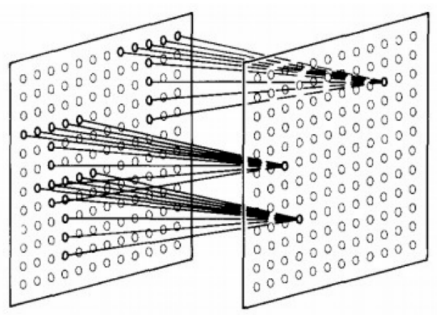
- Restricted Boltzmann machine
- Auto-encoder
- Deep belief network
- Deep Boltzmann machines
- Generative stochastic networks
- **Convolutional neural networks**
- Recurrent neural networks

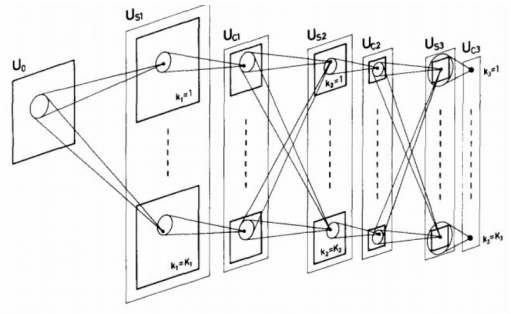
Convolutional Neural Networks

- Receptive Field (Huber & Wiesel, 1959)



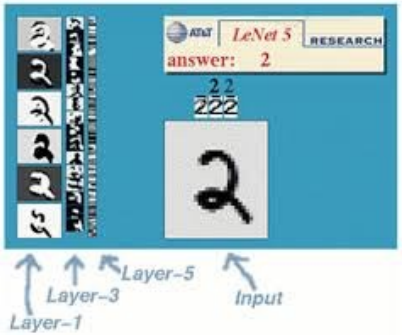
- Neocognitron (Fukushima, 1980)

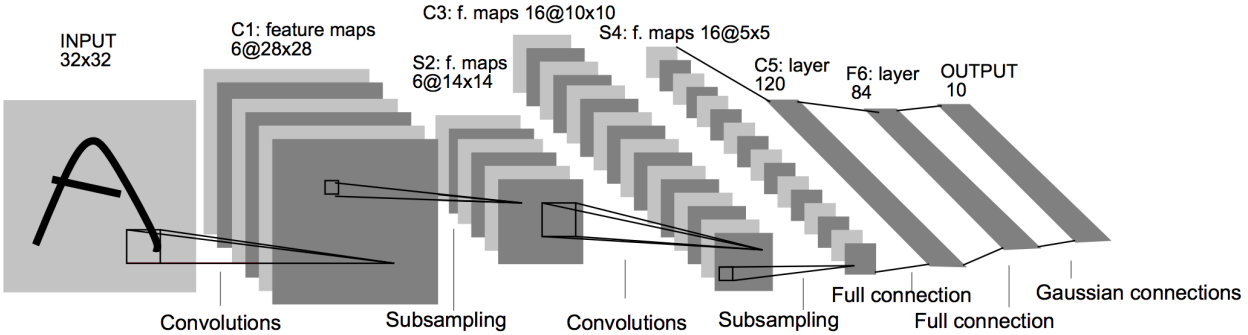




Convolutional Neural Networks

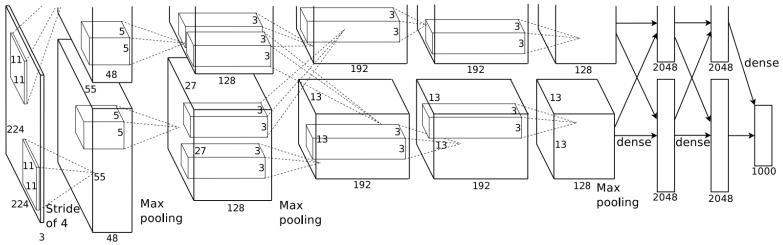
- LeNet-5 (Yann LeCun, 1998)



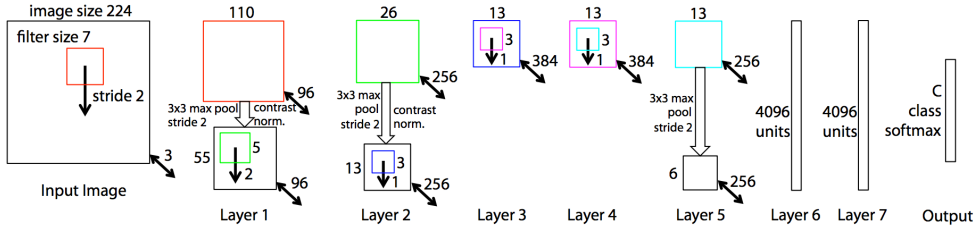


Convolutional Neural Networks

- Alex Net (Alex Krizhevsky et. al., 2012)

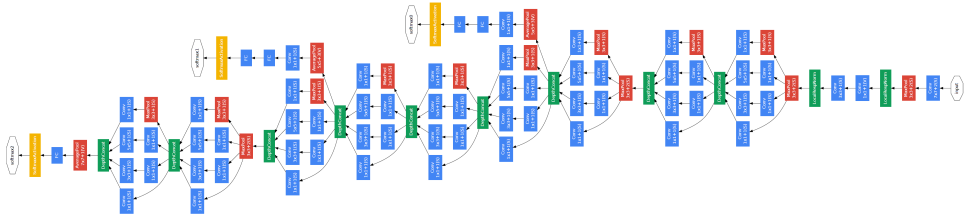


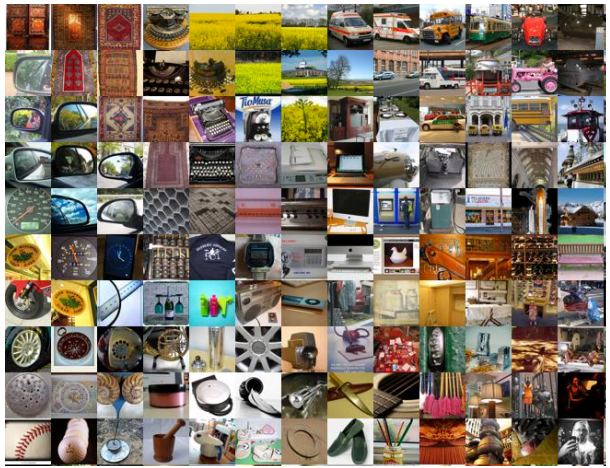
- Clarifai (Matt Zeiler and Rob Fergus, 2013)



ImageNet Challenge

- Detection/Classification and Localization
- 1.2M for Training/50K for Validation
- 1000 Classes





	Error rate	Remark
Human	5.1%	Karpathy
google (2014/8)	6.67%	Inception
baidu (2015/1)	5.98%	Data Augmentation
ms (2015/2)	4.94%	SPP, PReLU, Weight Initialization
google (2015/2)	4.82%	Batch Normalization
baidu (2015/5)	4.58%	Cheating

Convolutional Neural Networks

DEVIEW 2015

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

VGG (K. Simonyan and A. Zisserman, 2015)

Network

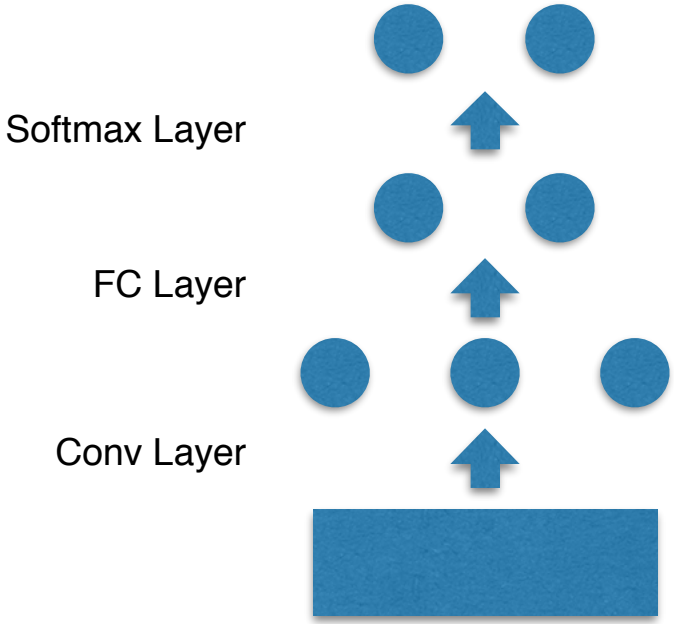
- Softmax Layer (Output)
- Fully Connected Layer
- Pooling Layer
- Convolution Layer

Layer

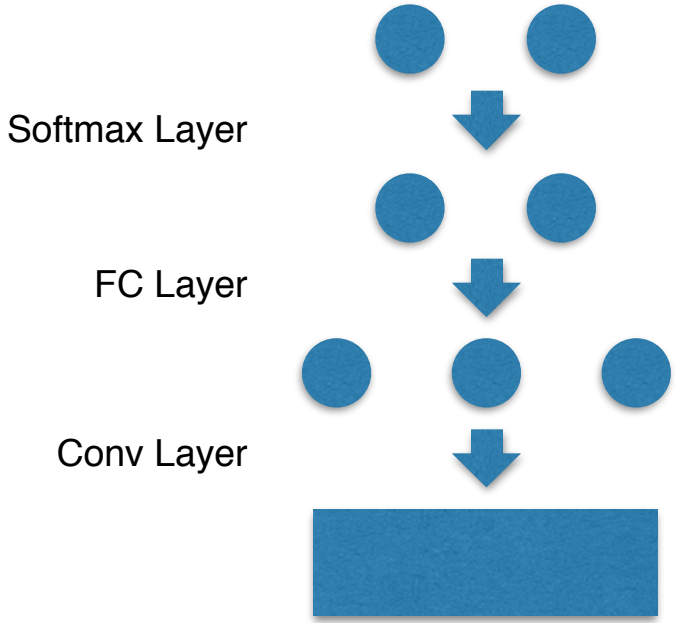
- Input / Output
- Weights
- Neuron activation

Neural Network

Forward Pass

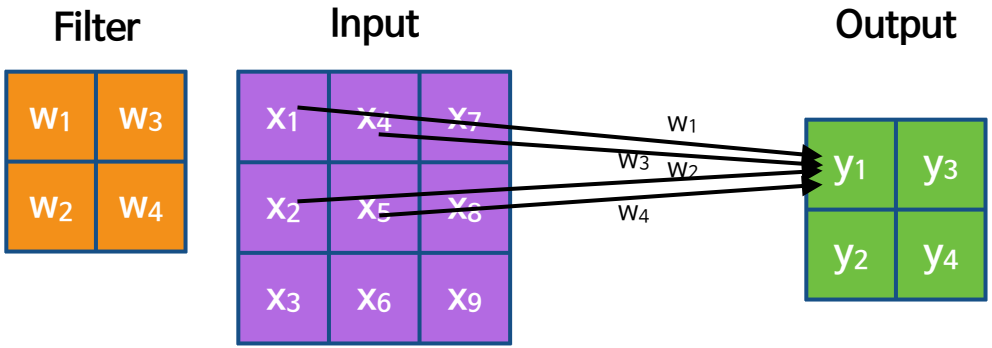


Backward Pass



Convolution Layer – Forward

$$Y_i^{(l)} = B_i^{(l)} + \sum_{j=1}^{m_1^{(l-1)}} K_{i,j}^{(l)} * Y_j^{(l-1)}$$



$$y_1 = f(a_1) \qquad a_1 = w_1x_1 + w_2x_2 + w_3x_4 + w_4x_5$$

$$y_2 = f(a_2) \qquad a_2 = w_1x_2 + w_2x_3 + w_3x_5 + w_4x_6$$

$$y_3 = f(a_3) \qquad a_3 = w_1x_4 + w_2x_5 + w_3x_7 + w_4x_8$$

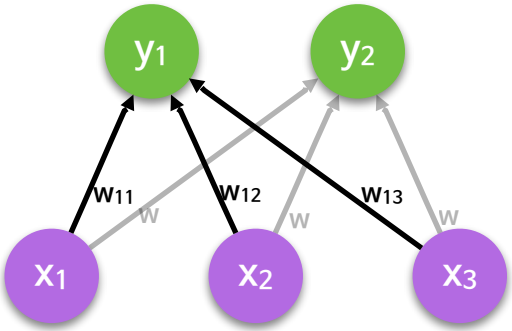
$$y_4 = f(a_4) \qquad a_4 = w_1x_5 + w_2x_6 + w_3x_8 + w_4x_9$$

Convolution Layer - Forward

How to evaluate the convolution layer efficiently?

1. Lower the convolutions into a matrix multiplication (cuDNN)
   - There are several ways to implement convolutions efficiently
2. Fast Fourier Transform to compute the convolution (cuDNN_v3)
3. Computing the convolutions directly (cuda-convnet)

Fully Connected Layer – Forward



$$a_1 = w_{11}x_1 + w_{12}x_2 + w_{13}x_3$$

$$y_1 = f(a_1)$$

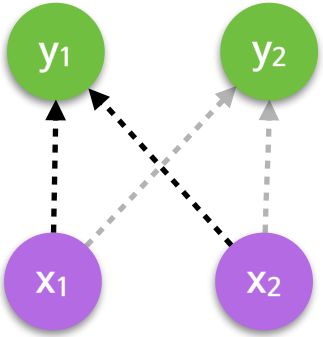
$$a_2 = w_{21}x_1 + w_{22}x_2 + w_{23}x_3$$

$$y_2 = f(a_2)$$

- Matrix calculation is very fast on GPU
  - cuBLAS library

$$\begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Softmax Layer – Forward



$$y_i = \frac{e^{x_i}}{\sum_{k=1}^{K} e^{x_k}}$$

Issues

1. How to efficiently evaluate denominator?
2. Numerical instability due to the too large or too small x_k

Fortunately, softmax Layer is supported by cuDNN

Learning on Neural Network

Update network (i.e. weights) to minimize loss function

- Popular loss function for classification

Sum-of-squared	Cross-entropy
$L=\frac{1}{2}\sum_{n=1}^{N}(y_n-t_n)^2$	$L=-\sum_{n=1}^{N}\sum_{k=1}^{K}t_{n,k}lny_{n,k}$

- Necessary criterion

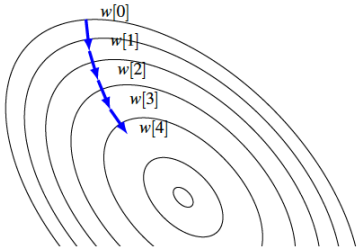
$$\frac{\partial E_n}{\partial w}=\nabla E_n(w)\stackrel{!}{=}0$$

Learning on Neural Network

- Due to the complexity of the loss, a closed-form solution is usually not possible
  - Using iterative approach

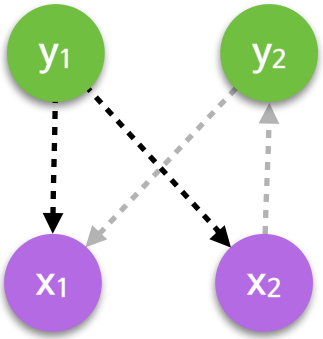
$$w[t+1] = w[t] + \Delta w[t]$$

Gradient descent	Newton's method
$\Delta w[t] = -\gamma \frac{\partial L}{\partial w[t]}$	$\Delta w[t] = -\gamma \left(\frac{\partial^2 L}{\partial w[t]^2} \right)^{-1} \frac{\partial L}{\partial w[t]}$



- How to evaluate the gradient $\frac{\partial L}{\partial w[t]}$
  - Error backpropagation

Softmax Layer – Backward



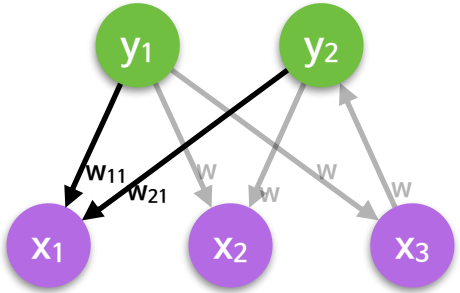
- Errors back-propagated from softmax Layer can be calculated
- Subtract 1 from the output value with target index
- This can be done efficiently using GPU threads

Loss function: $L=-\sum_{k=1}^{K} t_k ln y_k$

$$\frac{\partial L}{\partial a_i}=\sum_{k=1}^{K}\frac{\partial L}{\partial y_k}\frac{\partial y_k}{\partial a_i}=\sum_{k=1}^{K}-\frac{t_k}{y_k}(\delta_{i,k}y_i-y_i y_k)$$

$$=y_i-t_i$$

Fully Connected Layer – Backward



- Matrix calculation is very fast on GPU
- Element-wise multiplication can be done efficiently using GPU thread

Forward pass:

$$a_1^{l+1} = w_{11} f(a_1^l) + w_{12} f(a_2^l) + w_{13} f(a_3^l)$$
$$a_2^{l+1} = w_{21} f(a_1^l) + w_{22} f(a_2^l) + w_{23} f(a_3^l)$$

Error:

$$\frac{\partial L}{\partial a_i^l} = \sum_{j=1}^{H} \frac{\partial L}{\partial a_j^{l+1}} \frac{\partial a_j^{l+1}}{\partial a_i^l} = \frac{\partial f(a_i^l)}{\partial a_i^l} \sum_{j=1}^{H} \frac{\partial L}{\partial a_j^{l+1}} w_{ji}$$

Gradient:

$$\frac{\partial L}{\partial w_{ji}} = \frac{\partial L}{\partial a_j^{l+1}} \frac{\partial a_j^{l+1}}{\partial w_{ji}} = \frac{\partial L}{\partial a_j^{l+1}} f(a_i^l)$$

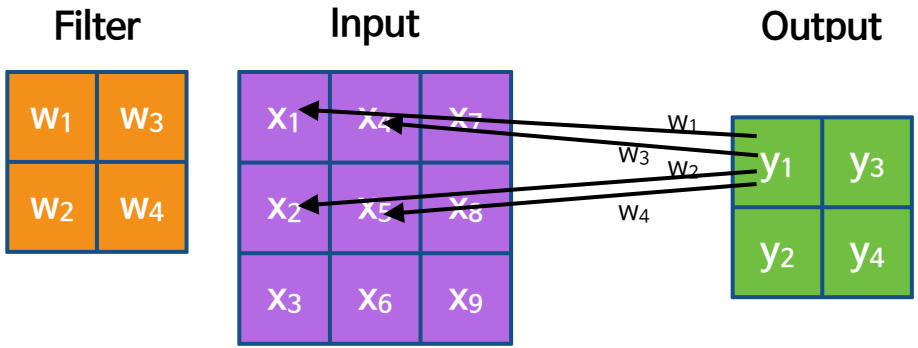
Error:

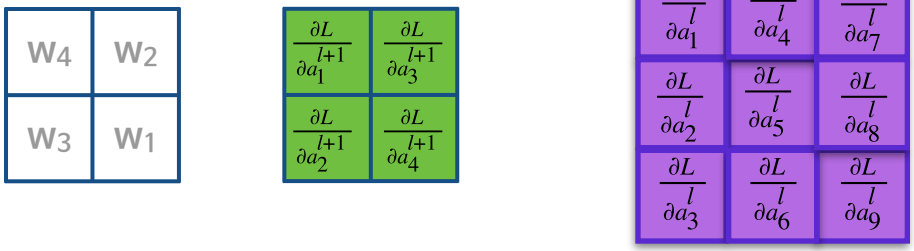
$$\begin{bmatrix} \frac{\partial L}{\partial a_1^l} \\ \frac{\partial L}{\partial a_2^l} \\ \frac{\partial L}{\partial a_3^l} \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}^T \begin{bmatrix} \frac{\partial L}{\partial a_1^{l+1}} \\ \frac{\partial L}{\partial a_2^{l+1}} \end{bmatrix}$$

Gradient:

$$\begin{bmatrix} \frac{\partial L}{\partial w_{11}} & \frac{\partial L}{\partial w_{12}} & \frac{\partial L}{\partial w_{13}} \\ \frac{\partial L}{\partial w_{23}} & \frac{\partial L}{\partial w_{23}} & \frac{\partial L}{\partial w_{23}} \end{bmatrix} = \begin{bmatrix} \frac{\partial L}{\partial a_1^{l+1}} \\ \frac{\partial L}{\partial a_2^{l+1}} \end{bmatrix} \begin{bmatrix} f(a_1^l) \\ f(a_2^l) \\ f(a_3^l) \end{bmatrix}^T$$

Convolution Layer – Backward





Forward

$$a_1^{l+1} = w_1 f(a_1^l) + w_2 f(a_2^l) + w_3 f(a_4^l) + w_4 f(a_5^l)$$

$$a_2^{l+1} = w_1 f(a_2^l) + w_2 f(a_3^l) + w_3 f(a_5^l) + w_4 f(a_6^l)$$

$$a_3^{l+1} = w_1 f(a_4^l) + w_2 f(a_5^l) + w_3 f(a_7^l) + w_4 f(a_8^l)$$

$$a_4^{l+1} = w_1 f(a_5^l) + w_2 f(a_6^l) + w_3 f(a_8^l) + w_4 f(a_9^l)$$

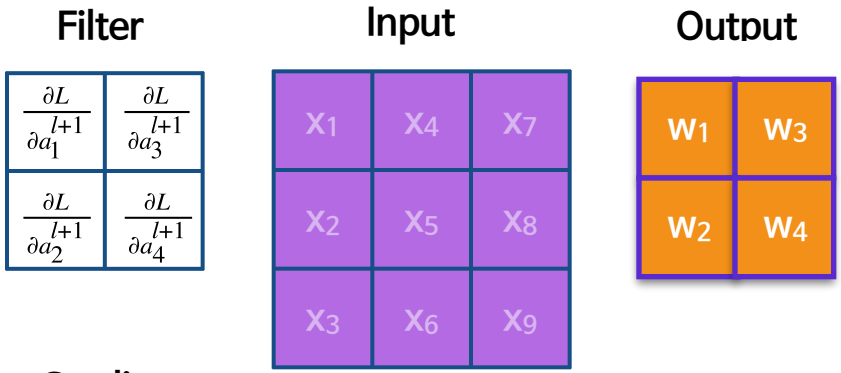
Error

$$\frac{\partial L}{\partial a_1^l} = \frac{\partial f(a_1^l)}{\partial a_1^l}\left(\frac{\partial L}{\partial a_1^{l+1}} w_1\right)$$

$$\frac{\partial L}{\partial a_5^l} = \frac{\partial f(a_5^l)}{\partial a_5^l}\left(\frac{\partial L}{\partial a_1^{l+1}} w_4 + \frac{\partial L}{\partial a_2^{l+1}} w_3 + \frac{\partial L}{\partial a_3^{l+1}} w_2 + \frac{\partial L}{\partial a_4^{l+1}} w_1\right)$$

$$\frac{\partial L}{\partial a_9^l} = \frac{\partial f(a_9^l)}{\partial a_9^l}\left(\frac{\partial L}{\partial a_4^{l+1}} w_4\right)$$

Convolution Layer – Backward



Gradient

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial a_1^{l+1}} f(a_1^l) + \frac{\partial L}{\partial a_2^{l+1}} f(a_2^l) + \frac{\partial L}{\partial a_3^{l+1}} f(a_4^l) + \frac{\partial L}{\partial a_4^{l+1}} f(a_5^l)$$

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial a_1^{l+1}} f(a_2^l) + \frac{\partial L}{\partial a_2^{l+1}} f(a_3^l) + \frac{\partial L}{\partial a_3^{l+1}} f(a_5^l) + \frac{\partial L}{\partial a_4^{l+1}} f(a_6^l)$$

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial a_1^{l+1}} f(a_4^l) + \frac{\partial L}{\partial a_2^{l+1}} f(a_5^l) + \frac{\partial L}{\partial a_3^{l+1}} f(a_7^l) + \frac{\partial L}{\partial a_4^{l+1}} f(a_8^l)$$

$$\frac{\partial L}{\partial w_4} = \frac{\partial L}{\partial a_1^{l+1}} f(a_5^l) + \frac{\partial L}{\partial a_2^{l+1}} f(a_6^l) + \frac{\partial L}{\partial a_3^{l+1}} f(a_8^l) + \frac{\partial L}{\partial a_4^{l+1}} f(a_9^l)$$

- The error and gradient can be computed with convolution scheme
- cuDNN supports this operation

2.
Implementation on GPU using cuDNN

Introduction to cuDNN

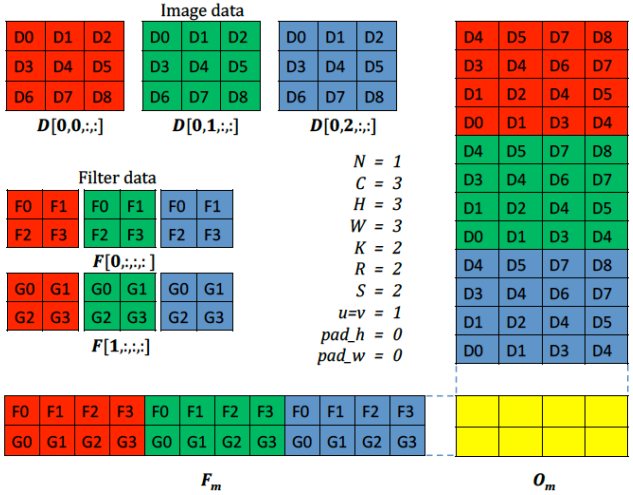
cuDNN is a GPU-accelerated library of primitives for deep neural networks

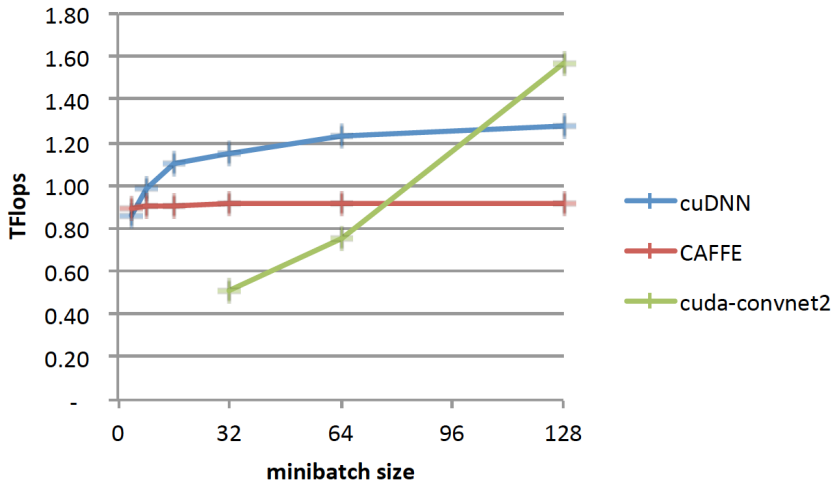
- Convolution forward and backward
- Pooling forward and backward
- Softmax forward and backward
- Neuron activations forward and backward:
  - Rectified linear (ReLU)
  - Sigmoid
  - Hyperbolic tangent (TANH)
- Tensor transformation functions

Introduction to cuDNN (version 2)

cuDNN's convolution routines aim for performance competitive with the fastest GEMM

- Lowering the convolutions into a matrix multiplication





(Sharan Chetlur et. al., 2015)

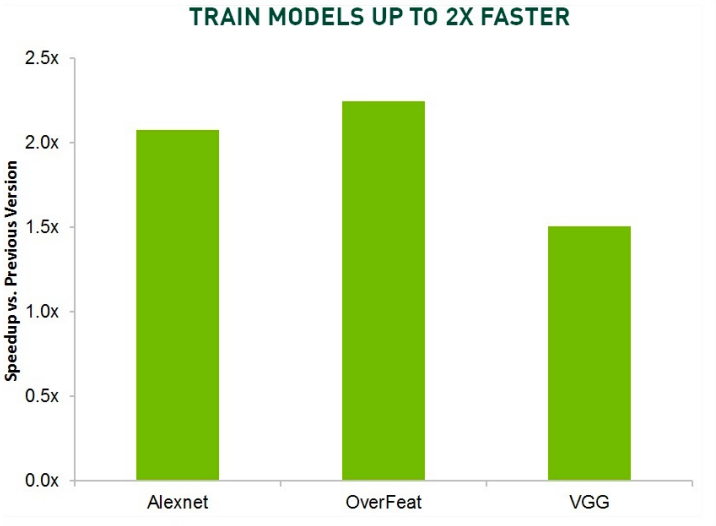
Introduction to cuDNN

Benchmarks

Overfeat [fast] - Input 128x3x231x231

Library	Class	Time (ms)	forward (ms)	backward (ms)
CuDNN[R3]-fp16	cudnn.SpatialConvolution	**313**	**107**	**206**
CuDNN[R3]-fp32	cudnn.SpatialConvolution	326	113	213
fbfft	SpatialConvolutionCuFFT	342	114	227
Nervana-fp16	ConvLayer	355	112	242
Nervana-fp32	ConvLayer	398	124	273
cudaconvnet2*	ConvLayer	723	176	547
CuDNN[R2] *	cudnn.SpatialConvolution	810	234	576
Caffe	ConvolutionLayer	823	355	468
Torch-7 (native)	SpatialConvolutionMM	878	379	499
CL-nn (Torch)	SpatialConvolutionMM	963	388	574
Caffe-CLGreenTea	ConvolutionLayer	2857	616	2240

https://github.com/soumith/convnet-benchmarks



https://developer.nvidia.com/cudnn

Goal

Learning VGG model using cuDNN

- Data Layer
- Convolution Layer
- Pooling Layer
- Fully Connected Layer
- Softmax Layer

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Preliminary

Common data structure for Layer

- Device memory & tensor description for input/output data & error
  - Tensor Description defines dimensions of data

```
float                   *d_input, *d_output, *d_inputDelta, *d_outputDelta
cudnnTensorDescriptor_t inputDesc;
cudnnTensorDescriptor_t outputDesc;
```

cuDNN initialize & release

```
cudaSetDevice( deviceID );
cudnnCreate( &cudnn_handle );
```

```
cudnnDestroy( cudnn_handle );
cudaDeviceReset();
```

Data Layer

create & set Tensor Descriptor

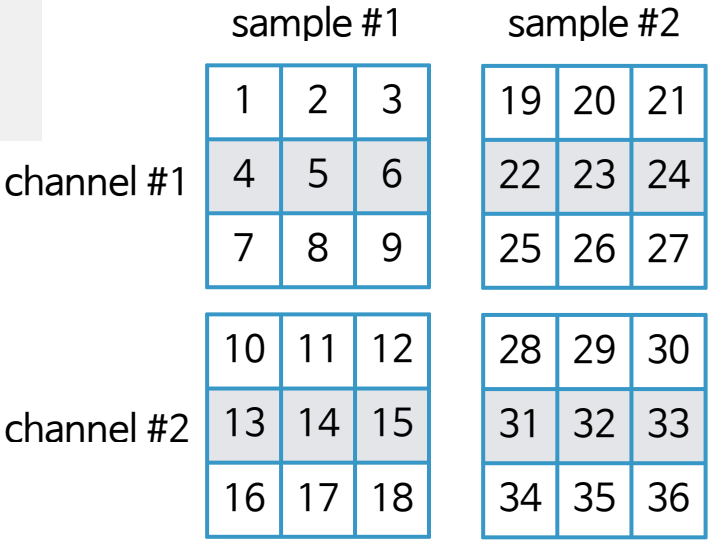
cudnnCreateTensorDescriptor();

cudnnSetTensor4dDescriptor();

```
cudnnStatus_t
cudnnSetTensor4dDescriptor( cudnnTensorDescriptor_t   tensorDesc,
                            cudnnTensorFormat_t format,
                            cudnnDataType_t dataType,
                            int n,
                            int c,
                            int h,
                            int w )
```

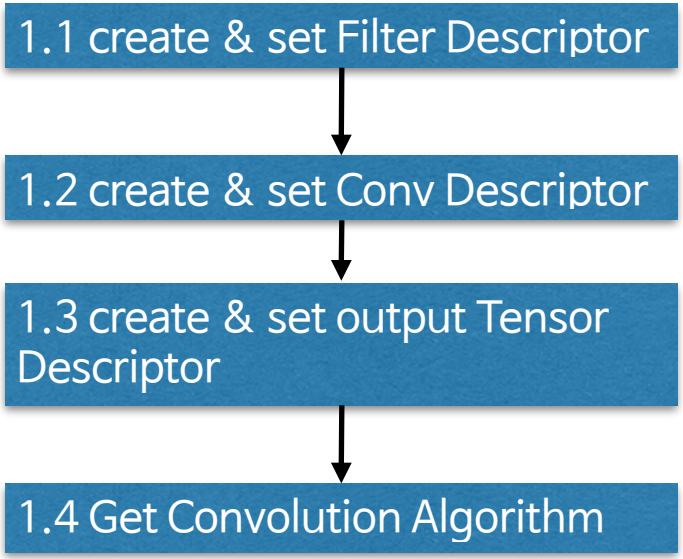
```
cudnnSetTensor4dDescriptor(
                            outputDesc,
                            CUDNN_TENSOR_NCHW,
                            CUDNN_FLOAT,
                            sampleCnt,
                            channels,
                            height,
                            width
);
```

Example: 2 images (3x3x2)



Convolution Layer

1. Initialization




```
cudnnCreateFilterDescriptor(&filterDesc);
cudnnSetFilter4dDescriptor(…);

cudnnCreateConvolutionDescriptor(&convDesc);
cudnnSetConvolution2dDescriptor(…);

cudnnGetConvolution2dForwardOutputDim(…);
cudnnCreateTensorDescriptor(&dstTensorDesc);
cudnnSetTensor4dDescriptor();

cudnnGetConvolutionForwardAlgorithm(…);
cudnnGetConvolutionForwardWorkspaceSize(…);
```

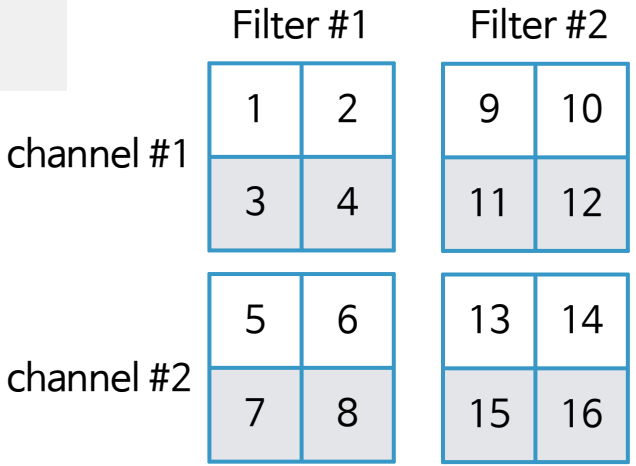
Convolution Layer

1.1 Set Filter Descriptor

```
cudnnStatus_t
cudnnSetFilter4dDescriptor( cudnnFilterDescriptor_t filterDesc,
                            cudnnDataType_t dataType,
                            int k,
                            int c,
                            int h,
                            int w )
```

```
cudnnSetFilter4dDescriptor(
                            filterDesc,
                            CUDNN_FLOAT,
                            filterCnt,
                            input_channelCnt,
                            filter_height,
                            filter_width
);
```

Example: 2 Filters (2x2x2)



Convolution Layer

1.2 Set Convolution Descriptor

```
cudnnStatus_t
cudnnSetConvolution2dDescriptor( cudnnConvolutionDescriptor_t convDesc,
                                 int pad_h,
                                 int pad_w,
                                 int u,
                                 int v,
                                 int upscalex,
                                 int upscaley,
                                 cudnnConvolutionMode_t mode )
```

CUDNN_CROSS_CORRELATION

Convolution Layer

1.3 Set output Tensor Descriptor

```
cudnnStatus_t
cudnnGetConvolution2dForwardOutputDim( const cudnnConvolutionDescriptor_t
 convDesc,
                                       const cudnnTensorDescriptor_t
 inputTensorDesc,
                                       const cudnnFilterDescriptor_t filterDesc,
                                       int *n,
                                       int *c,
                                       int *h,
                                       int *w )
```

- n, c, h and w indicate output dimension
  - Tensor Description defines dimensions of data

Convolution Layer

1.4 Get Convolution Algorithm

```
cudnnStatus_t
cudnnGetConvolutionForwardAlgorithm( cudnnHandle_t                       handle,
                                     const cudnnTensorDescriptor_t       srcDesc,
                                     const cudnnFilterDescriptor_t
 filterDesc,
                                     const cudnnConvolutionDescriptor_t
 convDesc,
                                     const cudnnTensorDescriptor_t
 destDesc,
                                     cudnnConvolutionFwdPreference_t
 preference,
                                     size_t
 memoryLimitInbytes,
                                     cudnnConvolutionFwdAlgo_t           *algo
                                   )
```

inputDesc

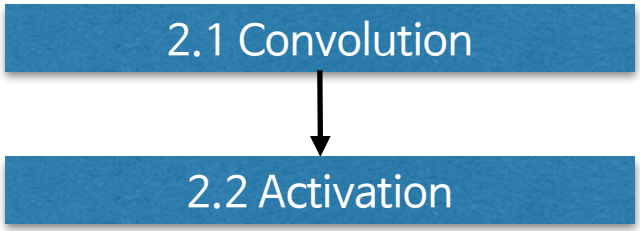
outputDesc

CUDNN_CONVOLUTION_FWD_PREFER_FASTEST

```
cudnnStatus_t
cudnnGetConvolutionForwardWorkspaceSize( cudnnHandle_t   handle,
                                         const   cudnnTensorDescriptor_t
 srcDesc,
                                         const   cudnnFilterDescriptor_t
 filterDesc,
                                         const   cudnnConvolutionDescriptor_t
 convDesc,
                                         const   cudnnTensor4dDescriptor_t
 destDesc,
                                         cudnnConvolutionFwdAlgo_t
 algo,
                                         size_t
 *sizeInBytes
                                   )
```

Convolution Layer

2. Forward Pass



cudnnConvolutionForward(…);

cudnnActivationForward(…);

Convolution Layer

2.1 Convolution

```
cudnnStatus_t
cudnnConvolutionForward( cudnnHandle_t                    handle,
                         const void                      *alpha,
                         const cudnnTensorDescriptor_t    srcDesc,
                         const void                      *srcData,
                         const cudnnFilterDescriptor_t    filterDesc,
                         const void                      *filterData,
                         const cudnnConvolutionDescriptor_t  convDesc,
                         cudnnConvolutionFwdAlgo_t        algo,
                         void                            *workSpace,
                         size_t
 workSpaceSizeInBytes,
                         const void                      *beta,

                         const cudnnTensorDescriptor_t    destDesc,
                         void                            *destData )
```

d_input
inputDesc

d_output
outputDesc

Convolution Layer

2.2 Activation

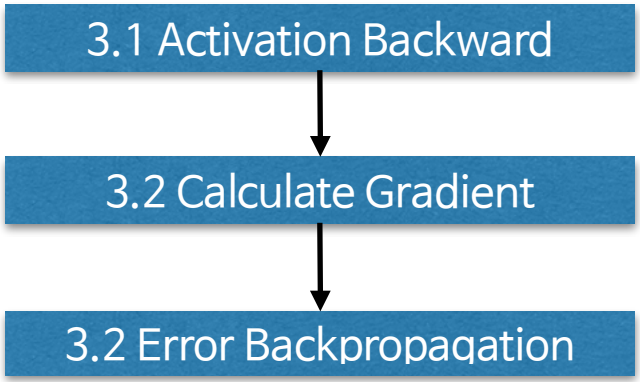
```
cudnnStatus_t
cudnnActivationForward( cudnnHandle_t                    handle,
                        cudnnActivationMode_t            mode,
                        const void                      *alpha,
                        const cudnnTensorDescriptor_t    srcDesc,
                        const void                      *srcData,
                        const void                      *beta,
                        const cudnnTensorDescriptor_t    destDesc,
                        void                            *destData )
```

sigmoid
tanh
ReLU

outputDesc
d_output

Convolution Layer

3. Backward Pass



cudnnActivationBackward(…);

cudnnConvolutionBackwardFilter(…);

cudnnConvolutionBackwardData(…);

Convolution Layer

3.1 Activation Backward

```
cudnnStatus_t
cudnnActivationBackward( cudnnHandle_t                    handle,
                         cudnnActivationMode_t            mode,
                         const void                      *alpha,
                         const cudnnTensorDescriptor_t    srcDesc,
                         const void                      *srcData,
                         const cudnnTensorDescriptor_t    srcDiffDesc,
                         const void                       *srcDiffData,
                         const cudnnTensorDescriptor_t    destDesc,
                         const void                      *destData,
                         const void                      *beta,
                         const cudnnTensorDescriptor_t    destDiffDesc,
                         void                            *destDiffData );
```

outputDesc
d_output

outputDesc
d_outputDelta

outputDesc
d_output

outputDesc
d_outputDelta

- Errors back-propagated from l+1 layer (d_outputDelta) is multiplied by $\frac{\partial f(a_i)}{\partial a_i}$
- See 22 slide (Convolution Layer – Backward)

Convolution Layer

3.2 Calculate Gradient

```
cudnnStatus_t
cudnnConvolutionBackwardFilter( cudnnHandle_t                    handle,
                                const void                      *alpha,
                                const cudnnTensorDescriptor_t    srcDesc,
                                const void                      *srcData,
                                const cudnnTensorDescriptor_t    diffDesc,
                                const void                      *diffData,
                                const cudnnConvolutionDescriptor_t  convDesc,
                                const void                      *beta,

                                const cudnnFilterDescriptor_t    gradDesc,
                                void                            *gradData );
```

inputDesc
d_input

outputDesc
d_outputDelta

filterDesc
d_filterGradient

Convolution Layer

3.2 Error Backpropagation

```
cudnnStatus_t
cudnnConvolutionBackwardData( cudnnHandle_t                       handle,
                              const void                         *alpha,
                              const cudnnFilterDescriptor_t       filterDesc,
                              const void                         *filterData,
                              const cudnnTensorDescriptor_t       diffDesc,
                              const void                         *diffData,
                              const cudnnConvolutionDescriptor_t  convDesc,
                              const void                         *beta,
                              const cudnnTensorDescriptor_t       gradDesc,
                              void                               *gradData

                            );
```

outputDesc
d_outputDelta

inputDesc
d_inputDelta

Pooling Layer / Softmax Layer

Pooling Layer

1. Initilization

```
cudnnCreatePoolingDescriptor(&poolingDesc);
cudnnSetPooling2dDescriptor(…);
```

2. Forward Pass

```
cudnnPoolingForward(…);
```

3. Backward Pass

```
cudnnPoolingBackward(…);
```

Softmax Layer

Forward Pass

```
cudnnSoftmaxForward(…);
```

3.
Optimization Issues

Optimization

Learning Very Deep ConvNet

We know the Deep ConvNet can be trained without pre-training

- weight sharing
- sparsity
- Recifier unit

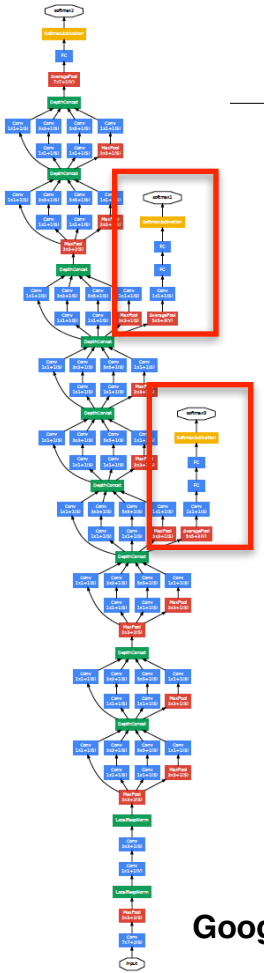
But, **"With fixed standard deviations very deep models have difficulties to converges"** (Kaiming He et. al., 2015)

- e.g. random initialization from Gaussian dist. with 0.01 std
- 〉8 convolution layers

Optimization

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

VGG (K. Simonyan and A. Zisserman, 2015)



GoogleNet (C. Szegedy et. al., 2014)

Optimization

Initialization of Weights for Rectifier (Kaiming He et. al., 2015)

- The variance of the response in each layer

$$Var[\Delta x_2] = Var[\Delta x_{L+1}] \left(\prod_{l=2}^{L} \frac{1}{2} \hat{n}_l Var[w_l] \right)$$

- Sufficient condition that the gradient is not exponentially large/small

$$\frac{1}{2} \hat{n}_l Var[w_l] = 1, \quad \forall l$$

- **Standard deviation for initialization**

$$\sqrt{2/\hat{n}_l} \qquad \hat{n}_l = k_l^2 d_l$$

(spatial filter size)^2 x (filter Cnt)

Optimization

Case study

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

3x3 filter

The filter number	$\sqrt{2/\hat{n}_l}$
64	0.059
128	0.042
256	0.029
512	0.021

- When using 0.01, the std of the gradient propagated from conv10 to convey

$$1/(5.9 \times 4.2^2 \times 2.9^2 \times 2.1^4) = 1/(1.7 \times 10^4)$$

Error vanishing

Speed

Data loading & model learning

- Reducing data loading and augmentation time
  - Data provider thread (dp_thread)
  - Model learning thread (worker_thread)

```
readData();
for(…) {
    readData();
    pthread_create(worker_thread)
    …
    pthread_join(worker_thread)
}
```

```
readData()
{
    if(is_dp_thread_running) pthread_join(dp_thread)
    …
    if(is_data_remain) pthread_create(dp_thread)
}
```

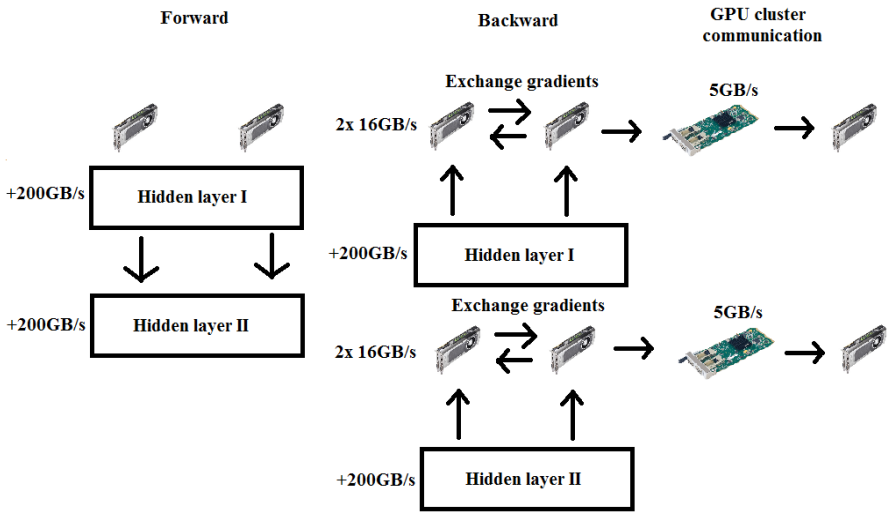
Multi-GPU

- Data parallelization v.s. Model parallelization
  - Distribute the model, use the same data : Model Parallelism
  - Distribute the data, use the same model : Data Parallelism

- Data parallelization & Gradient Average
  - One of the easiest way to use Multi-GPU
  - The result is same with using single GPU

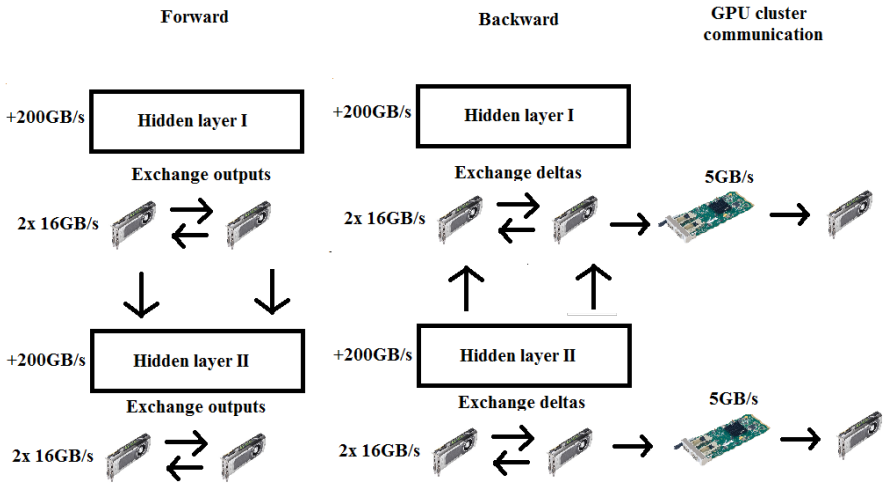
Parallelism

Data Parallelization



The good : Easy to implement
The bad : Cost of sync increases with the number of GPU

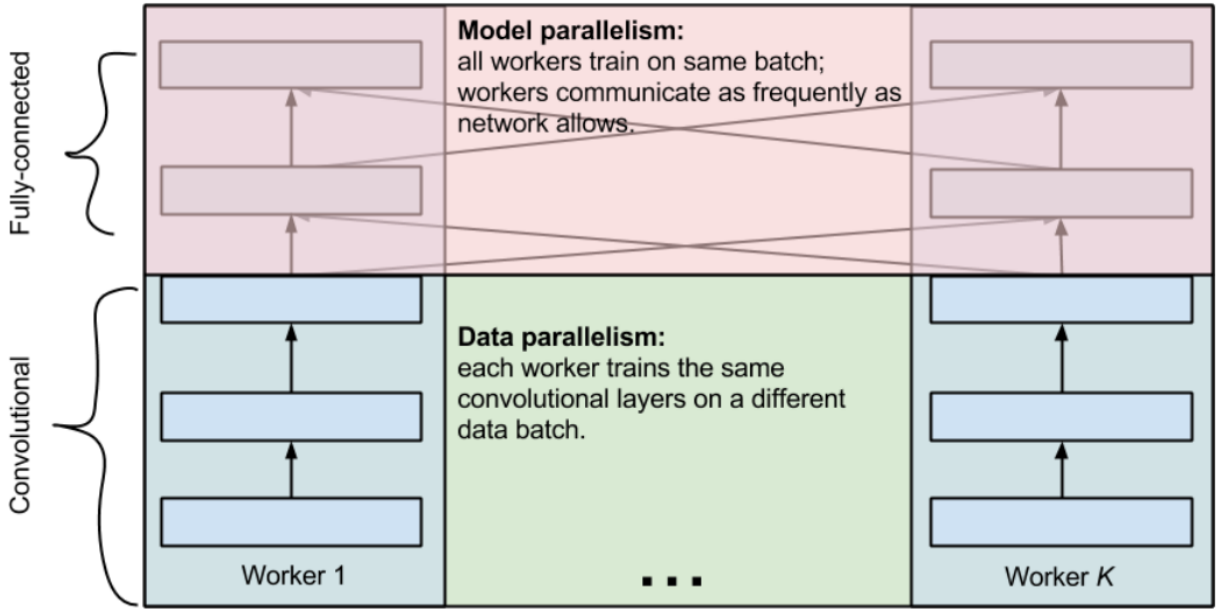
Model Parallelization



The good : Larger network can be trained
The bad : Sync is necessary in all layers

Parallelism

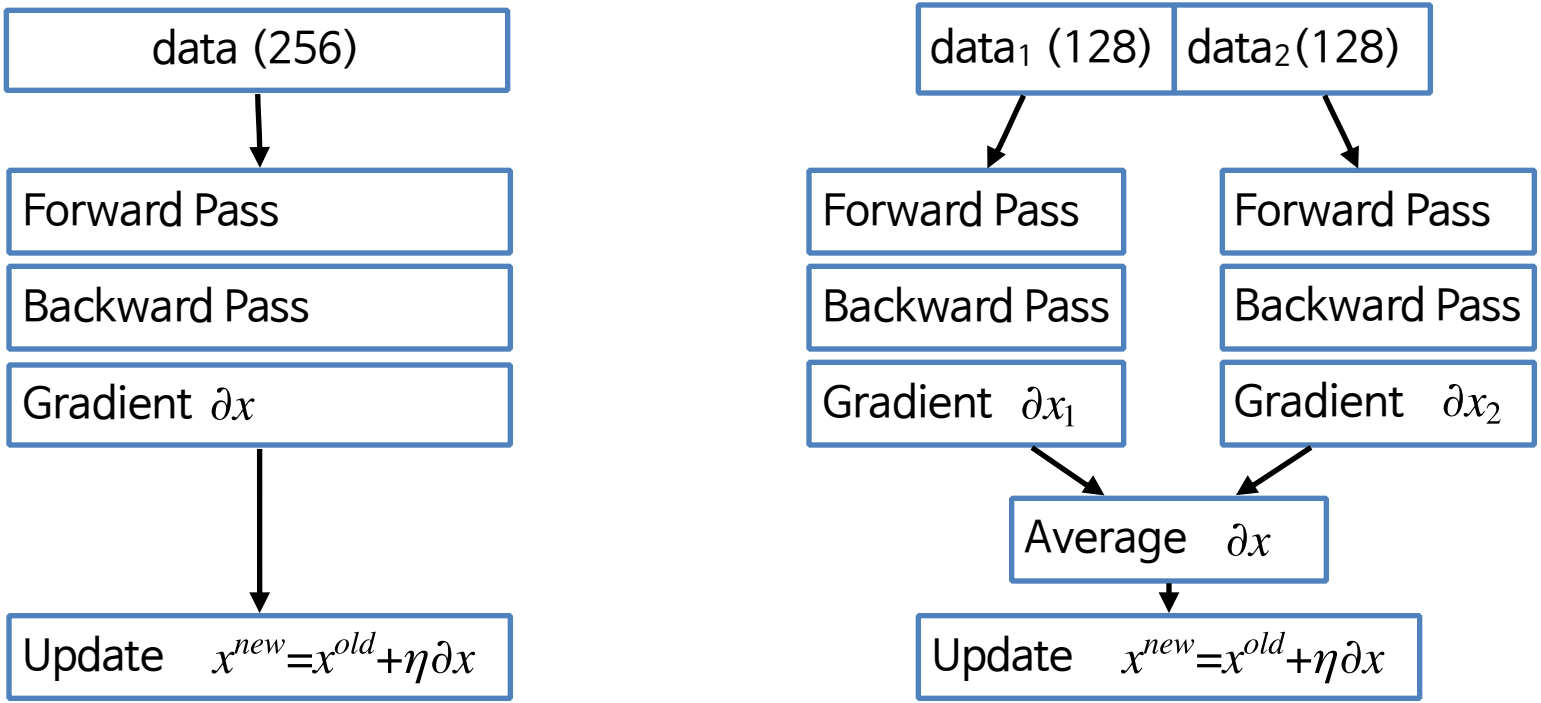
Mixing Data Parallelization and Model Parallelization



(Krizhevsky, 2014)

Speed

Data Parallelization & Gradient Average



4. Introduction to VUNO-Net

The Team

DEVIEW
2015

정규환(CTO)

포항공대 산업공학 박사
[최적화/기계학습]

SAMSUNG
삼성종합기술원

이예하(CEO)

포항공대 컴퓨터공학 박사
[정보검색/기계학습]

SAMSUNG
삼성종합기술원

김현준(CSO)

인하대 컴퓨터공학 박사
[이미지처리/기계학습]

SAMSUNG
삼성종합기술원

김상기(Researcher)

포항공대 컴퓨터공학 박사
[시계열데이터분석/기계학습]

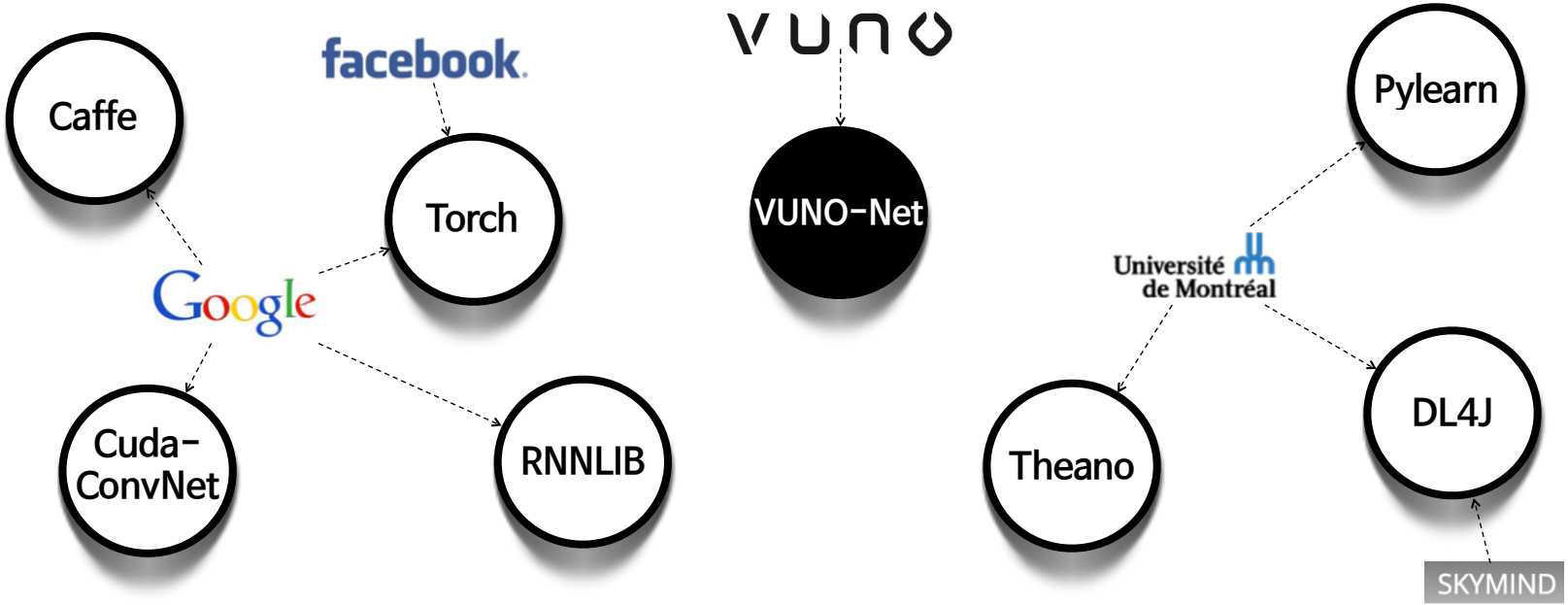
LG전자
미래IT융합연구소

최윤섭(Advisor)

포항공대 생명과학 박사
[전산생물학/디지털 헬스케어]

서울대학교병원
SEOUL NATIONAL UNIVERSITY HOSPITAL

VUNO-Net


Caffe
facebook
Torch
Google
Cuda-
ConvNet
RNNLIB
VUNO
VUNO-Net
Pylearn
Université
de Montréal
Theano
DL4J
SKYMIND

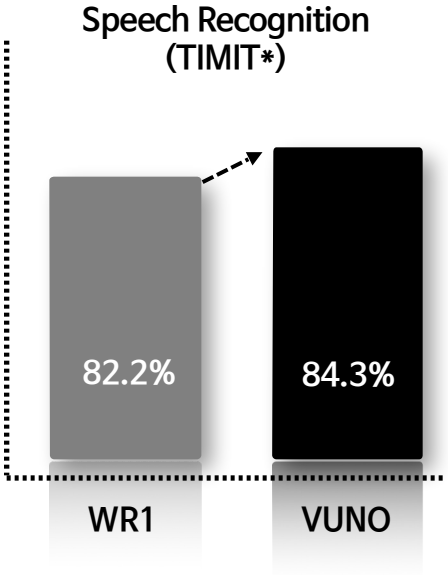
VUNO-Net

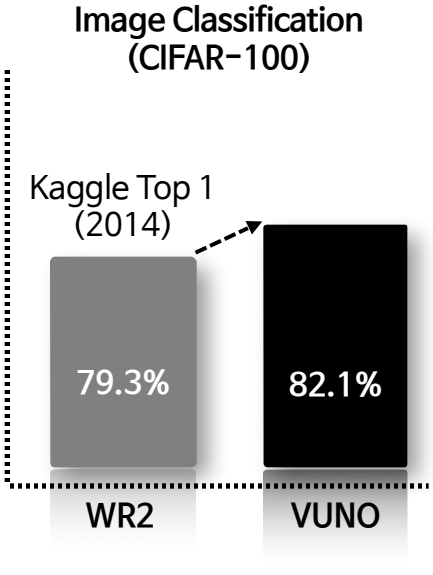
Features

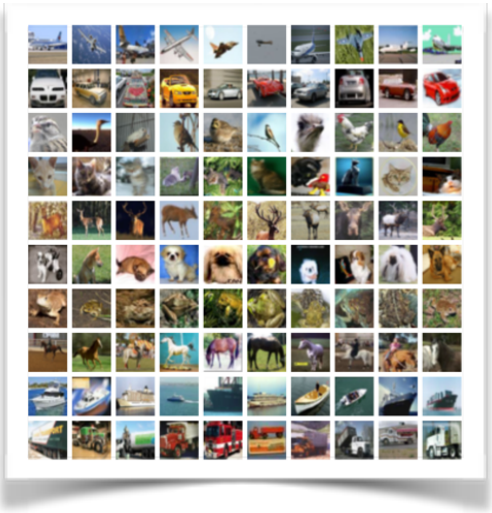
Structure	Output	Learning
Convolution	Softmax	Multi-GPU Support
LSTM	Regression	Batch Normalization
MD-LSTM(2D)	Connectionist Temporal Classification	Parametric Rectifier
Pooling		Initialization for Rectifier
Spatial Pyramid Pooling		Stochastic Gradient Descent
Fully Connection		Dropout
Concatenation		Data Augmentation

Performance

The state-of-the-art performance at image & speech



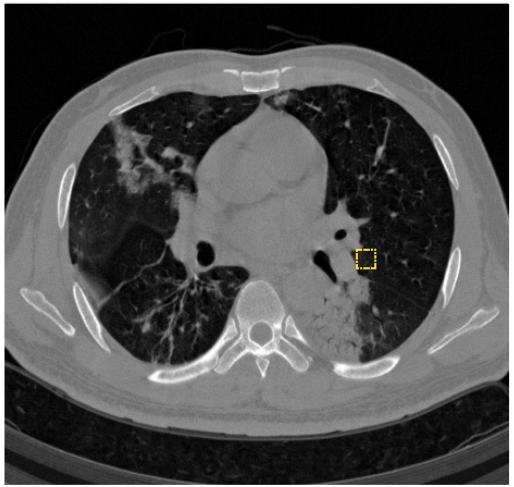


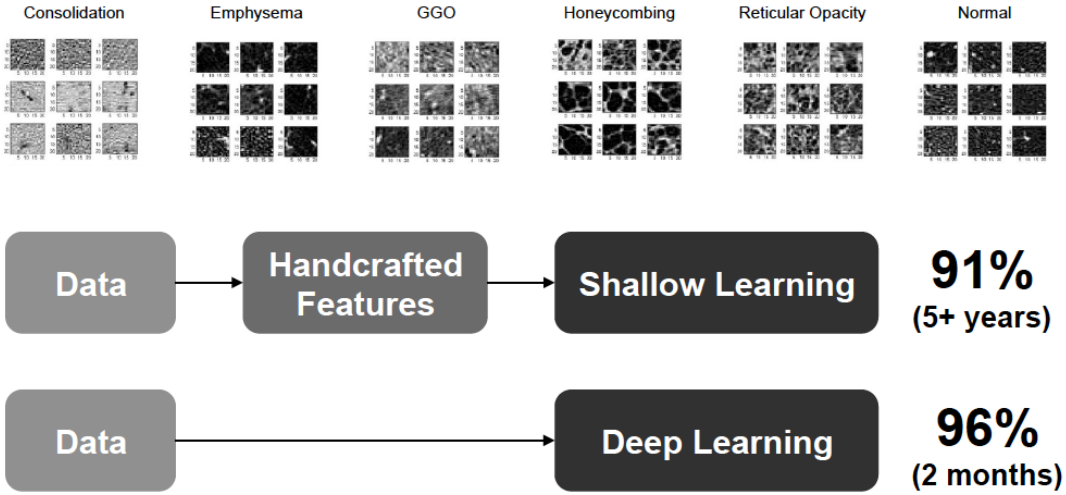


* TIMIT is a one of most popular benchmark dataset for speech recognition task (Texas Instrument - MIT)
WR1 (World Record) - "Speech Recognitino with Deep Recurrent Neural Networks", Alex Graves, ICCASP(2013)
WR2 (World Record) - "kaggle competition: https://www.kaggle.com/c/cifar-10"

Application

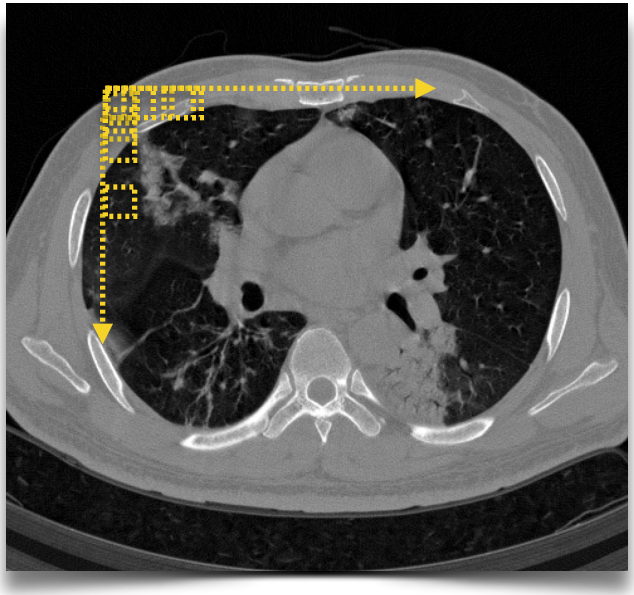
We've achieved record breaking performance on medical image analysis.





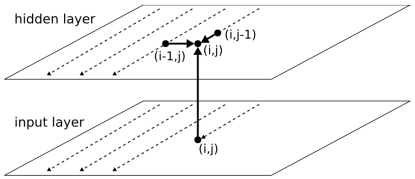
Application

Whole Lung Quantification

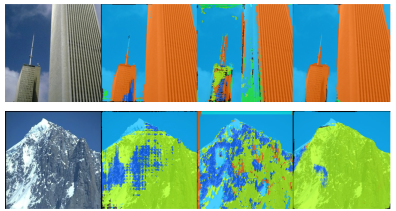


- Sliding window: Pixel level classification
- But, context information is more important

- Ongoing works



MD-LSTM

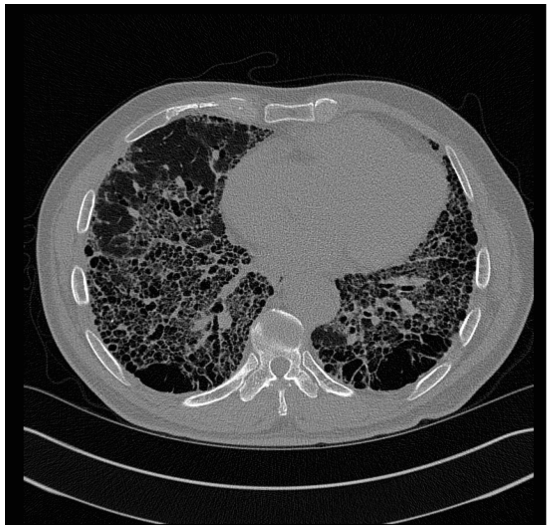


Recurrent CNN

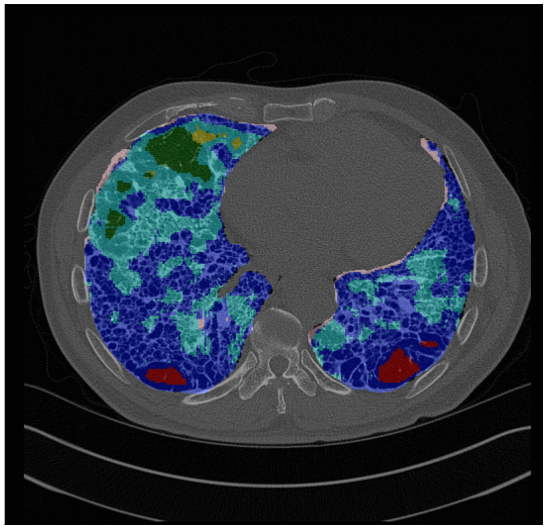
Application

Whole Lung Quantification

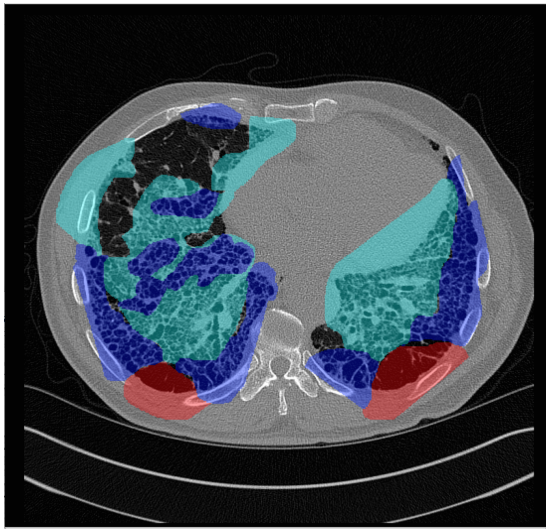
Example #1



Original Image (CT)



VUNO

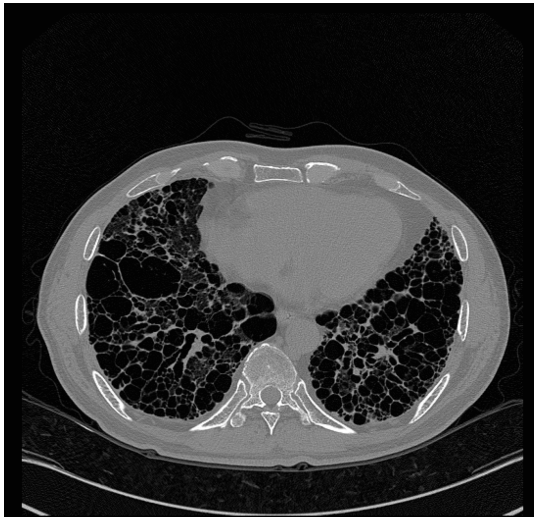


Golden Standard

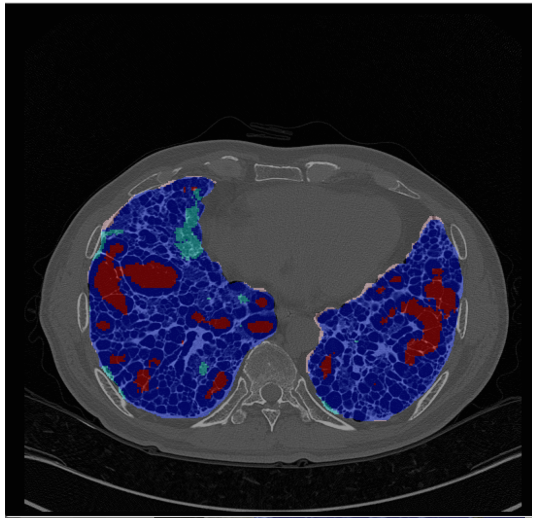
Application

Whole Lung Quantification

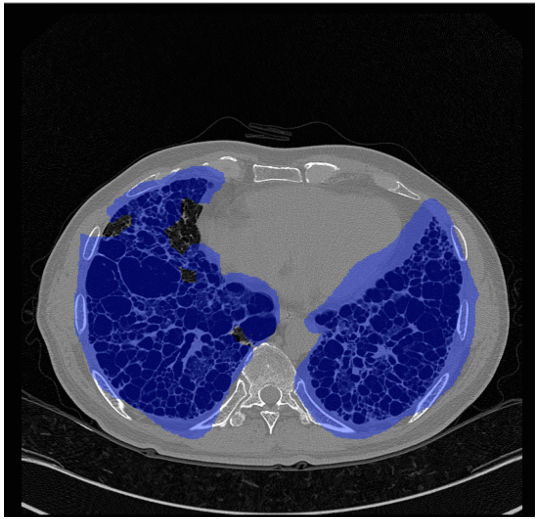
Example #2



Original Image (CT)



VUNO

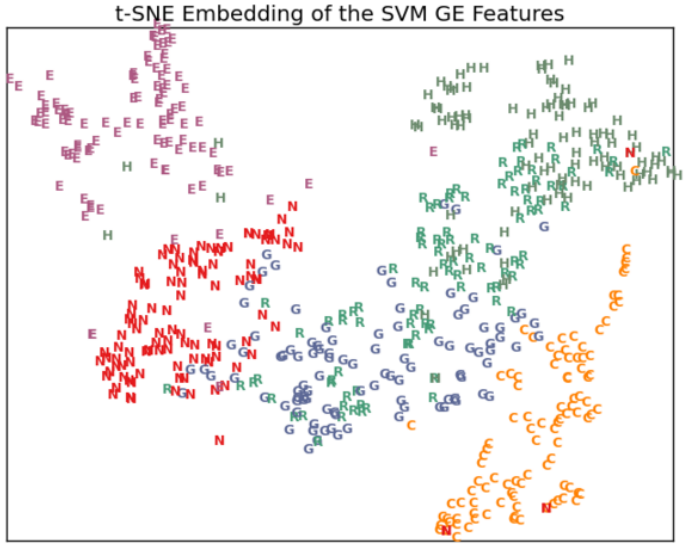


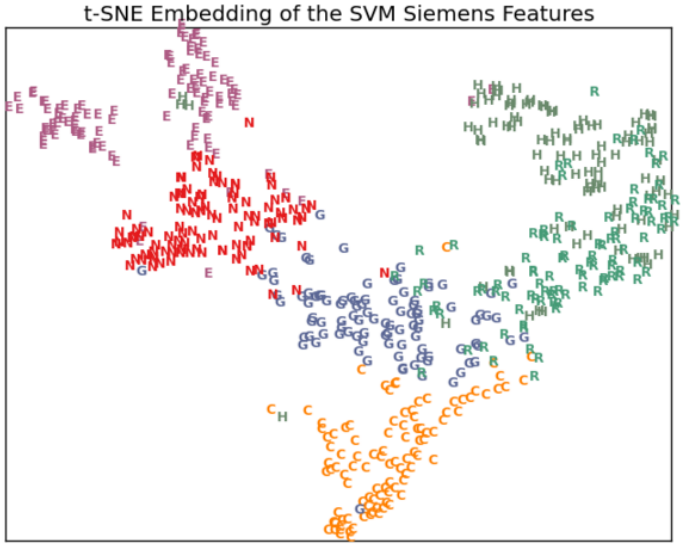
Golden Standard

Visualization

SVM Features (22 features)

Histogram, Gradient, Run-length, Co-occurrence matrix, Cluster analysis, Top-hat transformation

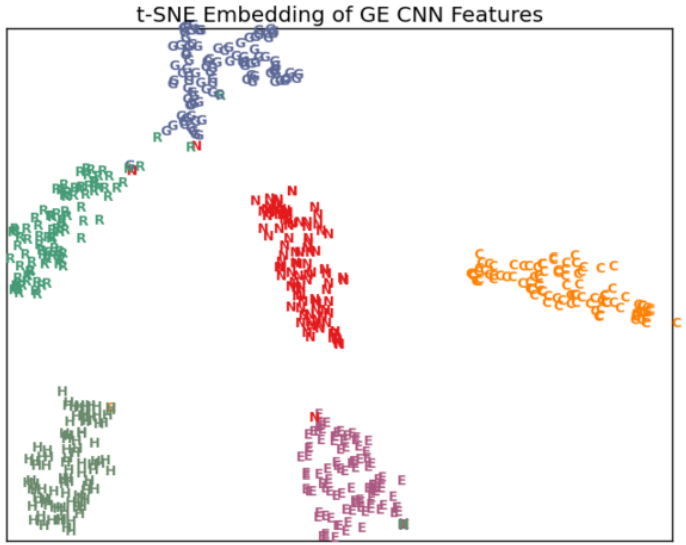


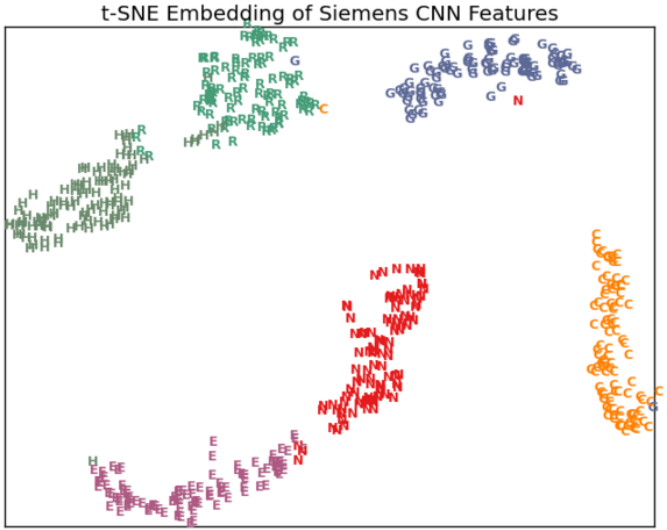


Visualization

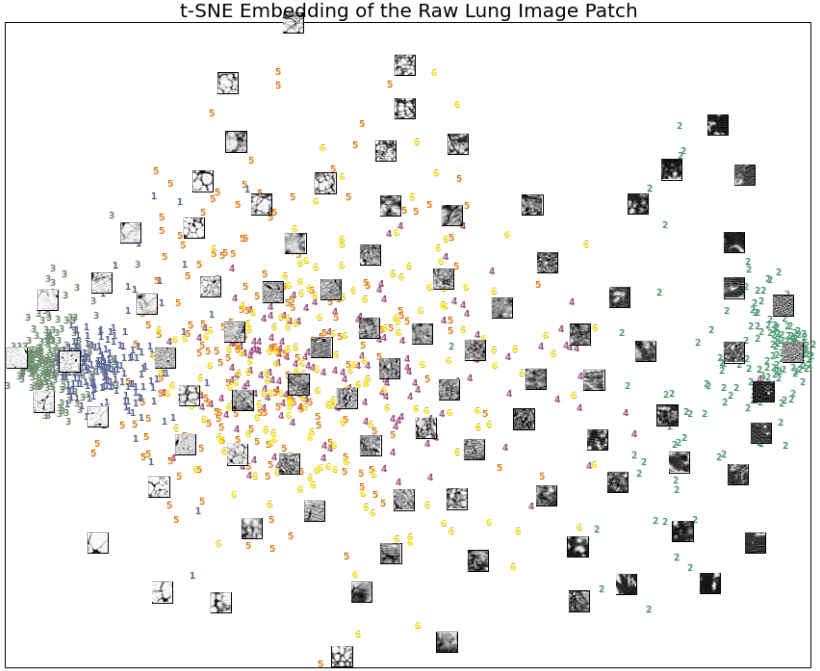
Activation of top hidden layer

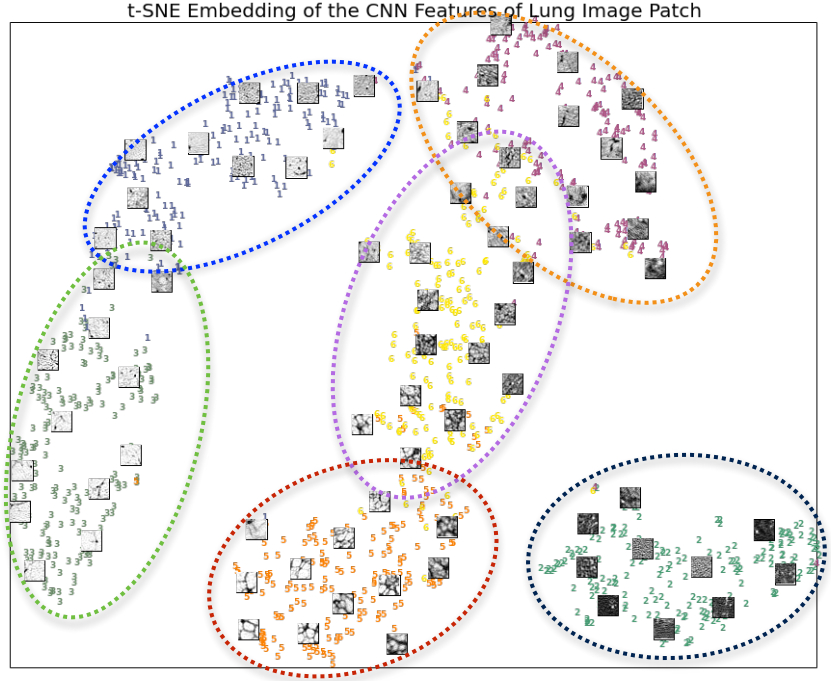
200 hidden nodes





Visualization






We Are Hiring!!
Algorithm Engineer
CUDA Programmer
Application Developer
Business Developer
Staff Member

Q&A

Thank You