GPU TECHNOLOGY CONFERENCE

IMPLEMENTING DEEP LEARNING USING CUDNN

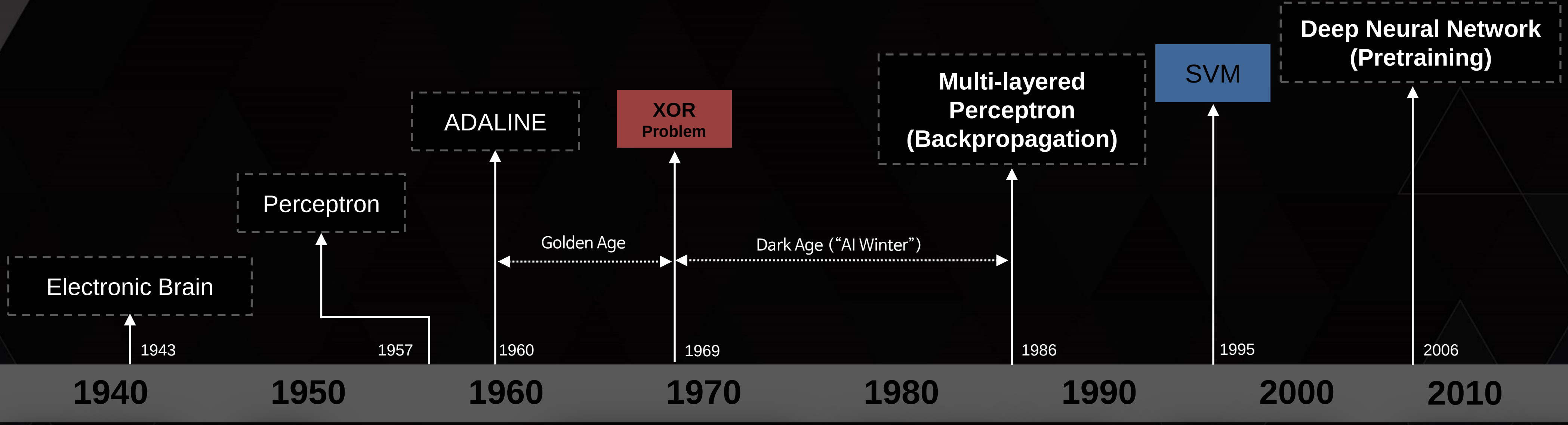
이예하 VUNO INC.

CONTENTS

- Deep Learning Review
- Implementation on GPU using cuDNN
- Optimization Issues
- Introduction to VUNO-Net

DEEP LEARNING REVIEW

BRIEF HISTORY OF NEURAL NETWORK


Electronic Brain — 1943

Perceptron — 1957

ADALINE — 1960

XOR Problem — 1969

Multi-layered Perceptron (Backpropagation) — 1986

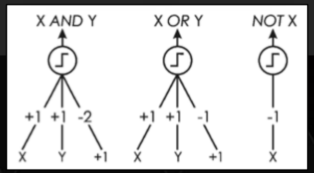
SVM — 1995

Deep Neural Network (Pretraining) — 2006

Golden Age

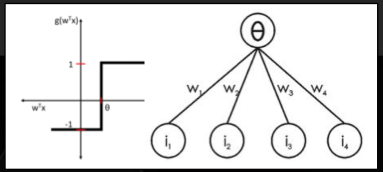
Dark Age ("AI Winter")

S. McCulloch - W. Pitts



- Adjustable Weights
- Weights are not Learned

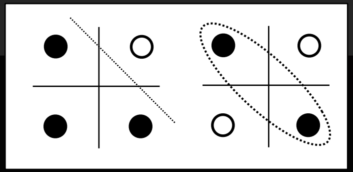
F. Rosenblatt



- Learnable Weights and Threshold

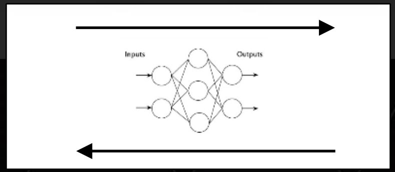
B. Widrow - M. Hoff

M. Minsky - S. Papert



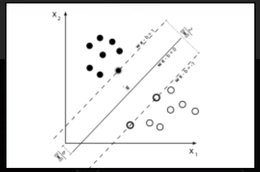
- XOR Problem

D. Rumelhart - G. Hinton - R. Wiliams



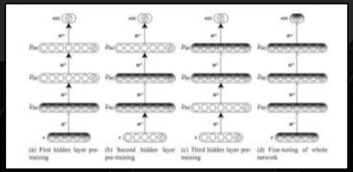
- Solution to nonlinearly separable problems
- Big computation, local optima and overfitting

V. Vapnik - C. Cortes



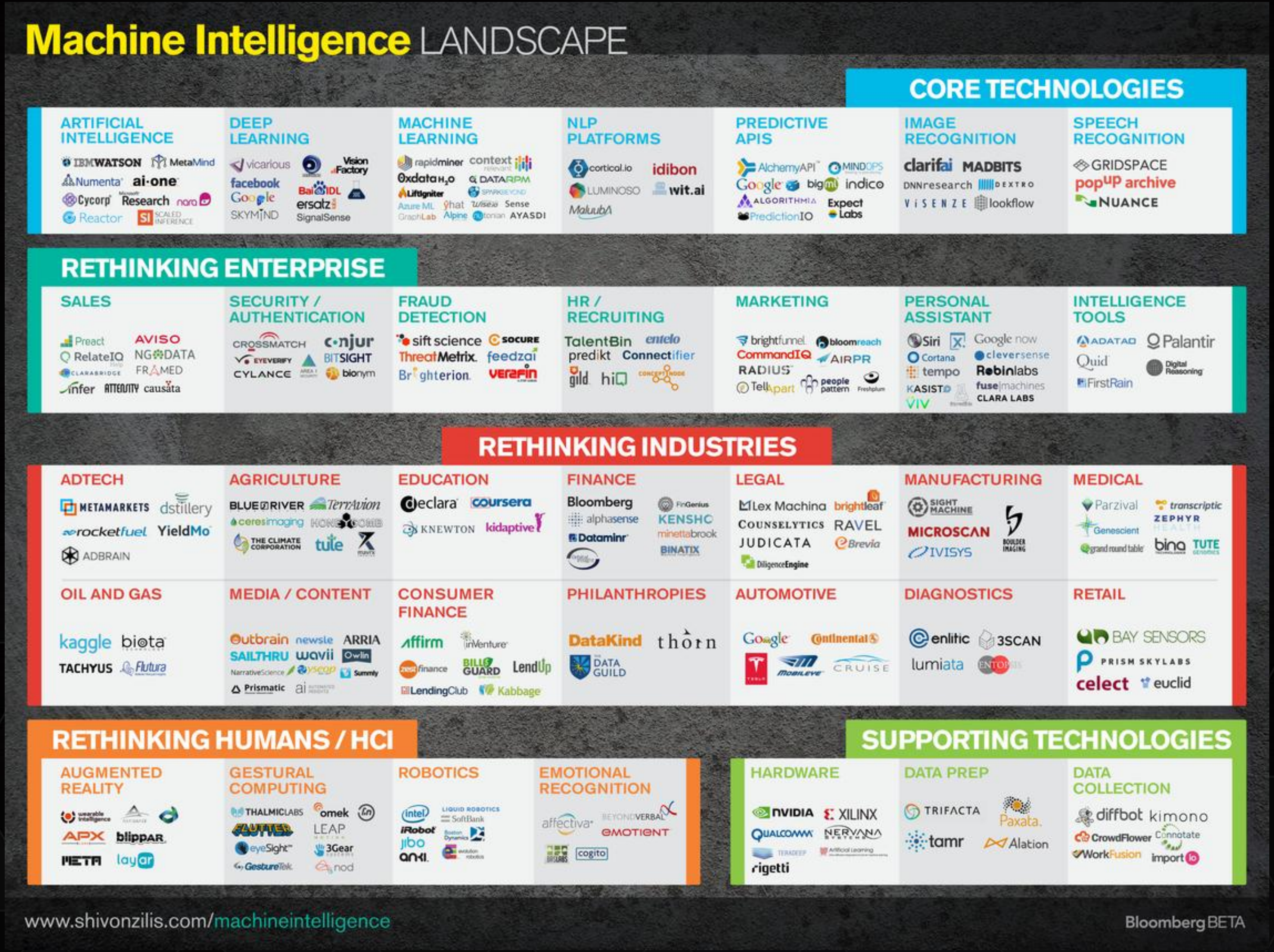
- Limitations of learning prior knowledge
- Kernel function: Human Intervention

G. Hinton - S. Ruslan



- Hierarchical feature Learning

MACHINE/DEEP LEARNING IS EATING THE WORLD!



Machine Intelligence LANDSCAPE

CORE TECHNOLOGIES

ARTIFICIAL INTELLIGENCE: IBM WATSON, MetaMind, Numenta, ai·one, Cycorp, Microsoft Research, nara, Reactor, SI Scaled Inference

DEEP LEARNING: vicarious, facebook, Google, SKYMIND, Vision Factory, Baidu IDL, ersatz, SignalSense

MACHINE LEARNING: rapidminer, context relevant, 0xdata H2O, DATARPM, Liftlgniter, SPARKBEYOND, Azure ML, yhat, Wise.io, Sense, GraphLab, Alpine, Nutonian, AYASDI

NLP PLATFORMS: cortical.io, idibon, LUMINOSO, wit.ai, Maluuba

PREDICTIVE APIS: AlchemyAPI, MINDOPS, Google, bigml, indico, ALGORITHMIA, Expect Labs, PredictionIO

IMAGE RECOGNITION: clarifai, MADBITS, DNNresearch, DEXTRO, VISENZE, lookflow

SPEECH RECOGNITION: GRIDSPACE, popup archive, NUANCE

RETHINKING ENTERPRISE

SALES: Preact, AVISO, RelateIQ, NGDATA, CLARABRIDGE, FRAMED, infer, ATTENTITY, causata

SECURITY / AUTHENTICATION: CROSSMATCH, conjur, EYEVERIFY, BITSIGHT, CYLANCE, AREA 1 SECURITY, bionym

FRAUD DETECTION: sift science, SOCURE, ThreatMetrix, feedzai, Brighterion, verafin

HR / RECRUITING: TalentBin, entelo, predikt, Connectifier, gild, hiQ, CONCEPT NODE

MARKETING: brightfunnel, bloomreach, CommandIQ, AIRPR, RADIUS, TellApart, people pattern, Freshplum

PERSONAL ASSISTANT: Siri, x.ai, Google now, Cortana, cleversense, tempo, Robinlabs, KASISTO, fuse machines, VIV, CLARA LABS

INTELLIGENCE TOOLS: ADATAO, Palantir, Quid, Digital Reasoning, FirstRain

RETHINKING INDUSTRIES

ADTECH: METAMARKETS, dstillery, rocketfuel, YieldMo, ADBRAIN

AGRICULTURE: BLUE RIVER, TerrAvion, ceresimaging, HONEYCOMB, THE CLIMATE CORPORATION, tule, mavrx

EDUCATION: declara, coursera, KNEWTON, kidaptive

FINANCE: Bloomberg, FinGenius, alphasense, KENSHO, Dataminr, minetta brook, BINATIX

LEGAL: Lex Machina, brightleaf, COUNSELYTICS, RAVEL, JUDICATA, Brevia, DiligenceEngine

MANUFACTURING: SIGHT MACHINE, MICROSCAN, BOULDER IMAGING, IVISYS

MEDICAL: Parzival, transcriptic, ZEPHYR HEALTH, Genescient, grand round table, bina, TUTE GENOMICS

OIL AND GAS: kaggle, biota, TACHYUS, Flutura

MEDIA / CONTENT: Outbrain, newsle, ARRIA, SAILTHRU, wavii, Owlin, NarrativeScience, Yseop, Summly, Prismatic, ai

CONSUMER FINANCE: affirm, inVenture, zestfinance, BILLGUARD, LendUp, LendingClub, Kabbage

PHILANTHROPIES: DataKind, thorn, DATA GUILD

AUTOMOTIVE: Google, Continental, TESLA, MOBILEYE, CRUISE

DIAGNOSTICS: enlitic, 3SCAN, lumiata, ENTOPSIS

RETAIL: BAY SENSORS, PRISM SKYLABS, celect, euclid

RETHINKING HUMANS / HCI

AUGMENTED REALITY: wearable intelligence, APX, blippar, META, layar

GESTURAL COMPUTING: THALMICLABS, omek, FLUTTER, LEAP MOTION, eyeSight, 3Gear, GestureTek, nod

ROBOTICS: intel, LIQUID ROBOTICS, SoftBank, iRobot, Boston Dynamics, jibo, anki, evolution robotics

EMOTIONAL RECOGNITION: affectiva, BEYOND VERBAL, EMOTIENT, cogito

SUPPORTING TECHNOLOGIES

HARDWARE: NVIDIA, XILINX, QUALCOMM, NERVANA, TERADEEP, Artificial Learning, rigetti

DATA PREP: TRIFACTA, Paxata, tamr, Alation

DATA COLLECTION: diffbot, kimono, CrowdFlower, Connotate, WorkFusion, import io

www.shivonzilis.com/machineintelligence

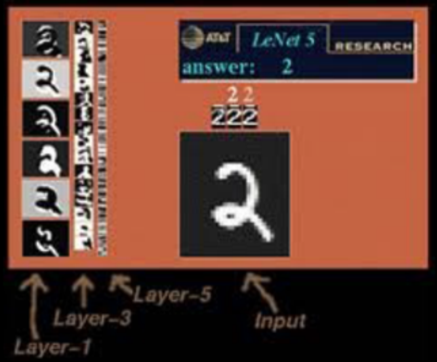
Bloomberg BETA

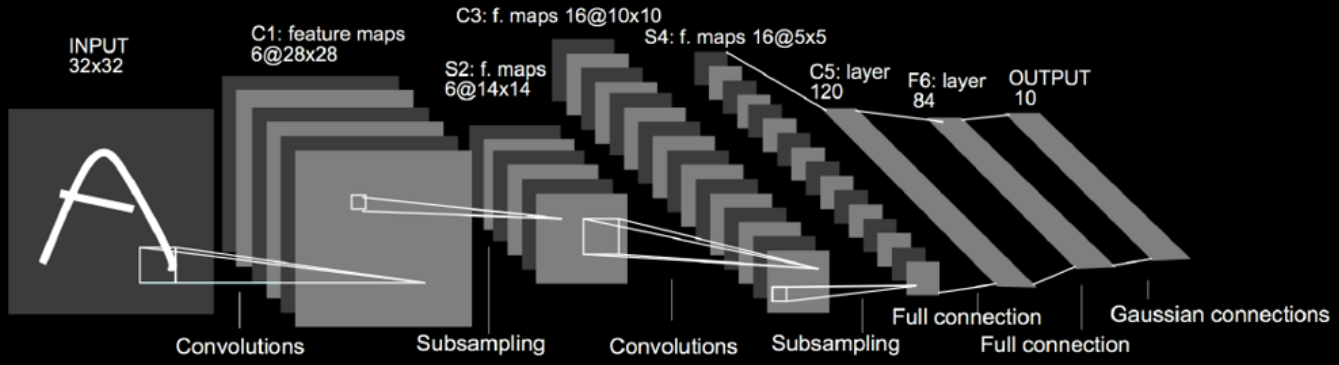
BUILDING BLOCKS

- Restricted Boltzmann machine
- Auto-encoder
- Deep belief Network
- Deep Boltzmann machine
- Generative stochastic networks
- Recurrent neural networks
- **Convolutional neural netwoks**

CONVOLUTIONAL NEURAL NETWORKS

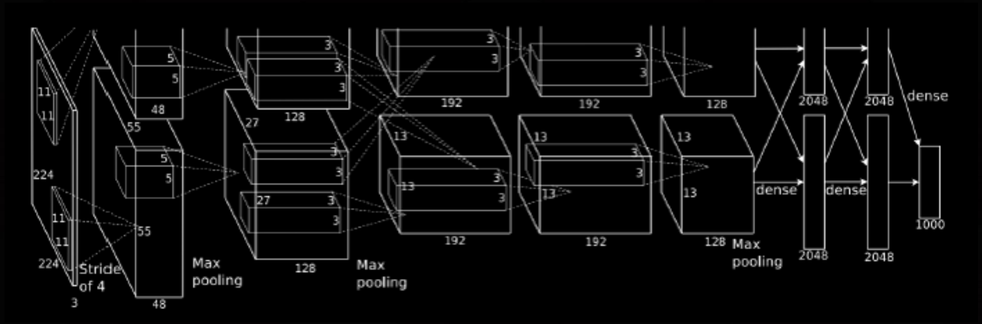
- LeNet-5 (Yann LeCun, 1998)



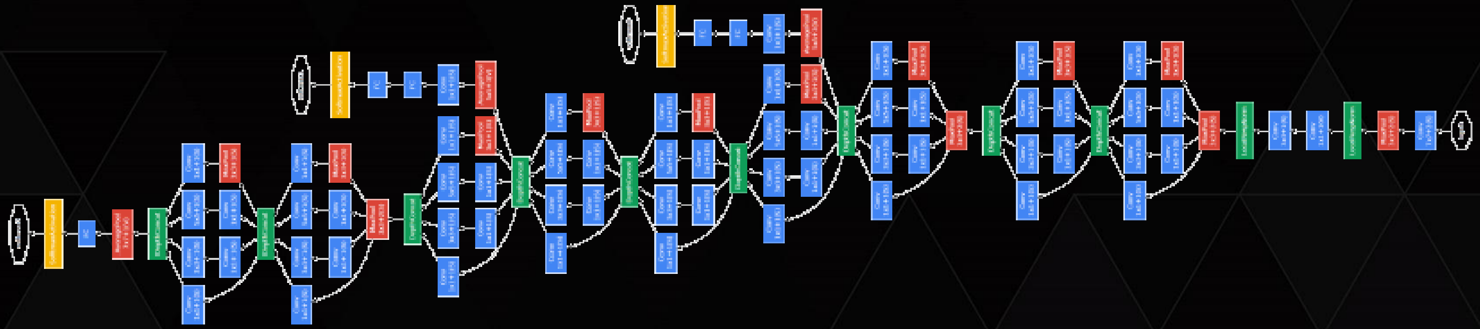


CONVOLUTIONAL NEURAL NETWORKS

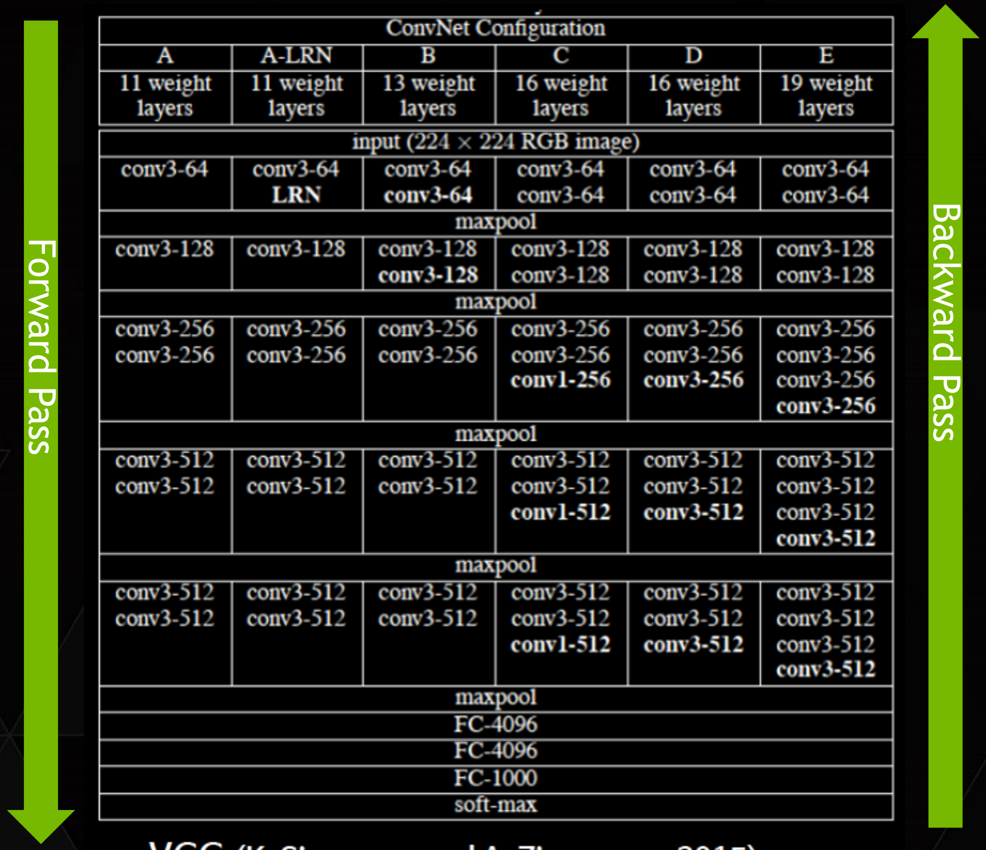
- Alex Net (Alex Krizhevsky et. al., 2012)



- GoogleNet (Szegedy et. Al., 2015)



CONVOLUTIONAL NEURAL NETWORKS



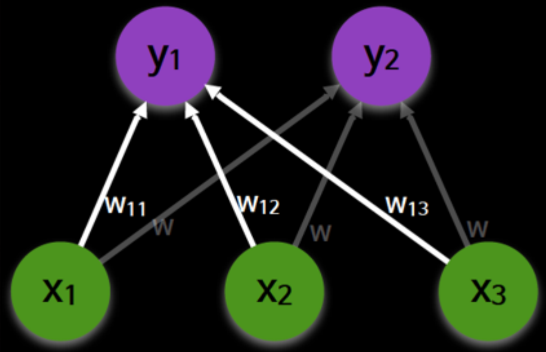
Forward Pass
Backward Pass

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

FULLY CONNECTED LAYER - FORWARD



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

$y_1 = f(a_1)$

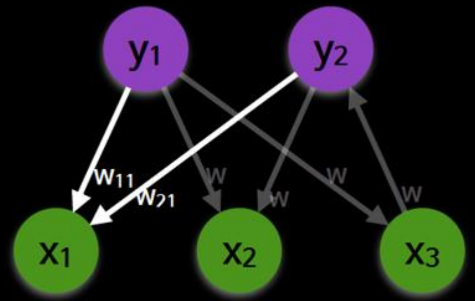
$a_2 = w_{21}x_1 + w_{22}x_2 + w_{23}x_3$

$y_2 = f(a_2)$

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

- Matrix calculation is very fast on GPU
  - cuBLAS library

FULLY CONNECTED LAYER - BACKWARD



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

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

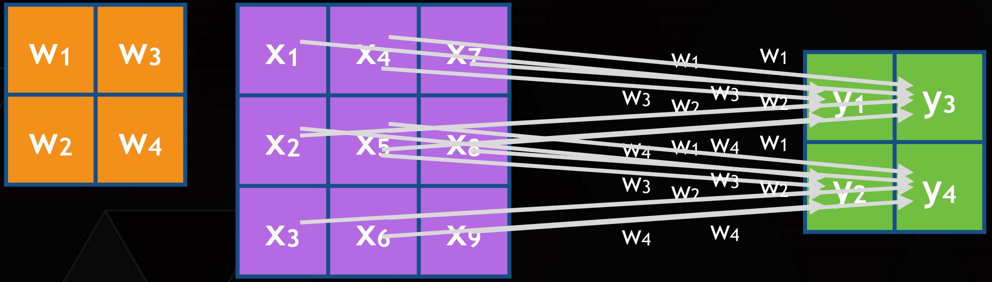
Error:
$$\begin{bmatrix}\frac{\partial L}{\partial a_1^l}\\ \frac{\partial L}{\partial a_2^l}\\ \frac{\partial L}{\partial a_3^l}\end{bmatrix}=\begin{bmatrix}w_{11} & w_{12} & w_{13}\\ w_{21} & w_{22} & w_{23}\end{bmatrix}^T\begin{bmatrix}\frac{\partial L}{\partial a_1^{l+1}}\\ \frac{\partial L}{\partial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

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

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

CONVOLUTION LAYER - FORWARD

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



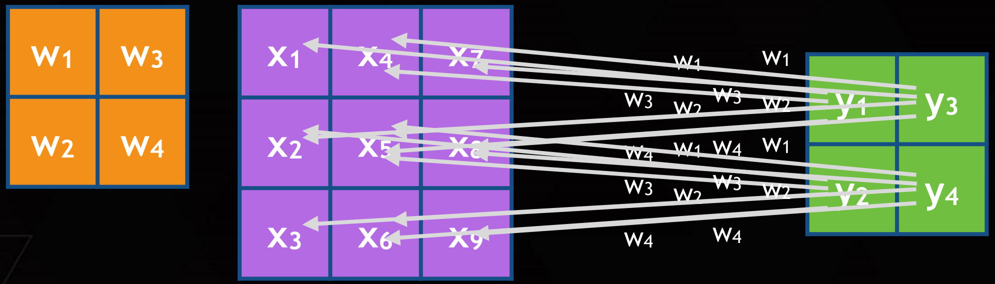
$y_1 = f(a_1)$ $a_1 = w_1 x_1 + w_2 x_2 + w_3 x_4 + w_4 x_5$

$y_2 = f(a_2)$ $a_2 = w_1 x_2 + w_2 x_3 + w_3 x_5 + w_4 x_6$

$y_3 = f(a_3)$ $a_3 = w_1 x_4 + w_2 x_5 + w_3 x_7 + w_4 x_8$

$y_4 = f(a_4)$ $a_4 = w_1 x_5 + w_2 x_6 + w_3 x_8 + w_4 x_9$

CONVOLUTION LAYER - BACKWARD



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

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

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

CONVOLUTION LAYER - BACKWARD

Error

W4	W2
W3	W1

W4	W4	W4	W2
W4	W4	W4	W2
W4	W4	W4	W2
W3	W3	W3	W1

$\frac{\partial L}{\partial a_1^l}$	$\frac{\partial L}{\partial a_4^l}$	$\frac{\partial L}{\partial a_7^l}$
$\frac{\partial L}{\partial a_2^l}$	$\frac{\partial L}{\partial a_5^l}$	$\frac{\partial L}{\partial a_8^l}$
$\frac{\partial L}{\partial a_3^l}$	$\frac{\partial L}{\partial a_6^l}$	$\frac{\partial L}{\partial a_9^l}$

Gradient

$\frac{\partial L}{\partial a_1^{l+1}}$	$\frac{\partial L}{\partial a_3^{l+1}}$
$\frac{\partial L}{\partial a_2^{l+1}}$	$\frac{\partial L}{\partial a_4^{l+1}}$

$\frac{\partial L}{\partial a_1^{l+1}}$	$\frac{\partial L}{\partial a_3^{l+1}}$	$\frac{\partial L}{\partial a_3^{l+1}}$
$\frac{\partial L}{\partial a_2^{l+1}}$	$\frac{\partial L}{\partial a_4^{l+1}}$	$\frac{\partial L}{\partial a_3^{l+1}}$
$\frac{\partial L}{\partial a_2^{l+1}}$	$\frac{\partial L}{\partial a_2^{l+1}}$	$\frac{\partial L}{\partial a_4^{l+1}}$

$\frac{\partial L}{\partial w_1}$	$\frac{\partial L}{\partial w_3}$
$\frac{\partial L}{\partial w_2}$	$\frac{\partial L}{\partial w_4}$

HOW TO EVALUATE THE CONVOLUTION LAYER EFFICIENTLY?

- Both Forward and Backward passes can be computed with convolution scheme

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

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

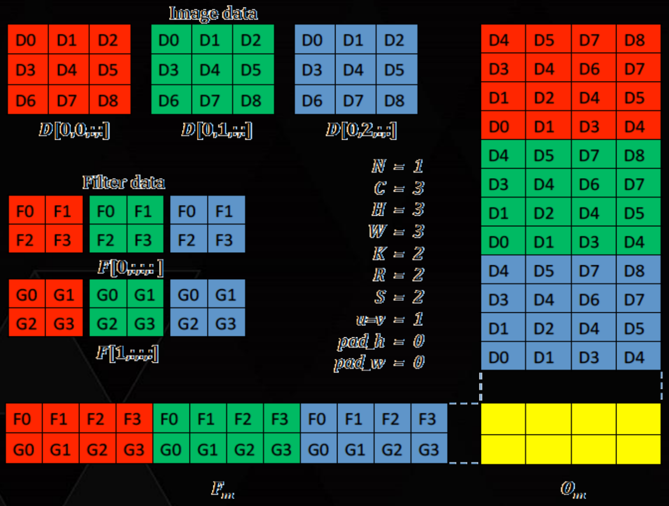


Image data

D0	D1	D2
D3	D4	D5
D6	D7	D8

$D[0,0,:,:]$ $D[0,1,:,:]$ $D[0,2,:,:]$

Filter data

F0	F1
F2	F3

$F[0,:,:,:]$

G0	G1
G2	G3

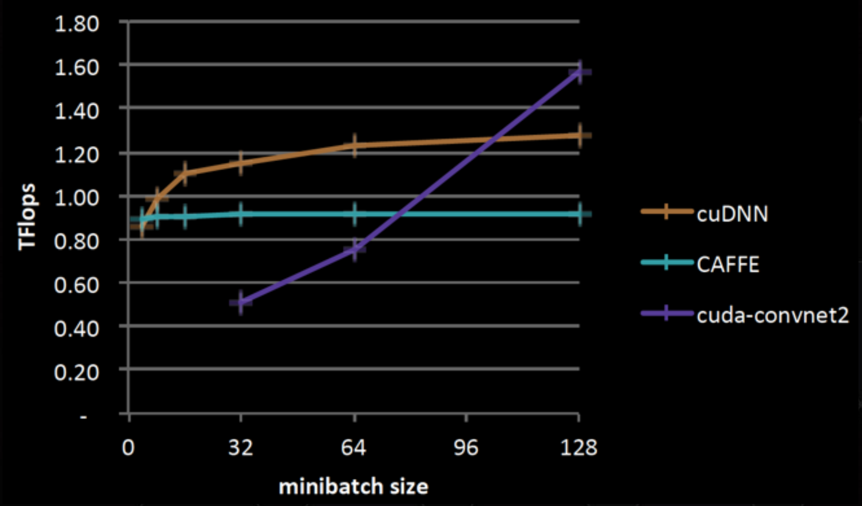
$F[1,:,:,:]$

$N = 1$
$C = 3$
$H = 3$
$W = 3$
$K = 2$
$R = 2$
$S = 2$
$u = v = 1$
$pad\_h = 0$
$pad\_w = 0$

D4	D5	D7	D8
D3	D4	D6	D7
D1	D2	D4	D5
D0	D1	D3	D4
D4	D5	D7	D8
D3	D4	D6	D7
D1	D2	D4	D5
D0	D1	D3	D4
D4	D5	D7	D8
D3	D4	D6	D7
D1	D2	D4	D5
D0	D1	D3	D4

F0	F1	F2	F3	F0	F1	F2	F3	F0	F1	F2	F3
G0	G1	G2	G3	G0	G1	G2	G3	G0	G1	G2	G3

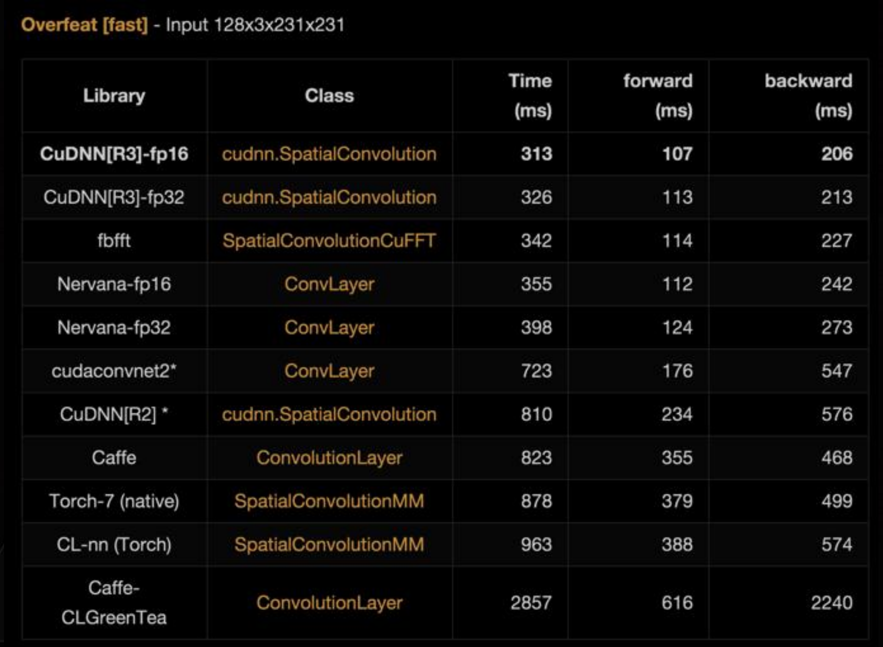
F_m O_m



(Sharan Chetlur et. al., 2015)

INTRODUCTION TO CUDNN

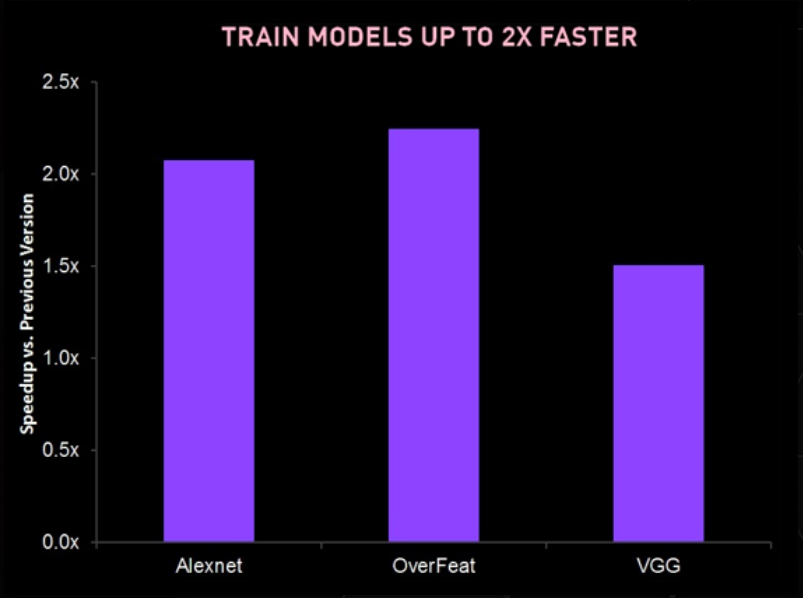
- Benchmarks



Overfeat [fast] - Input 128x3x231x231

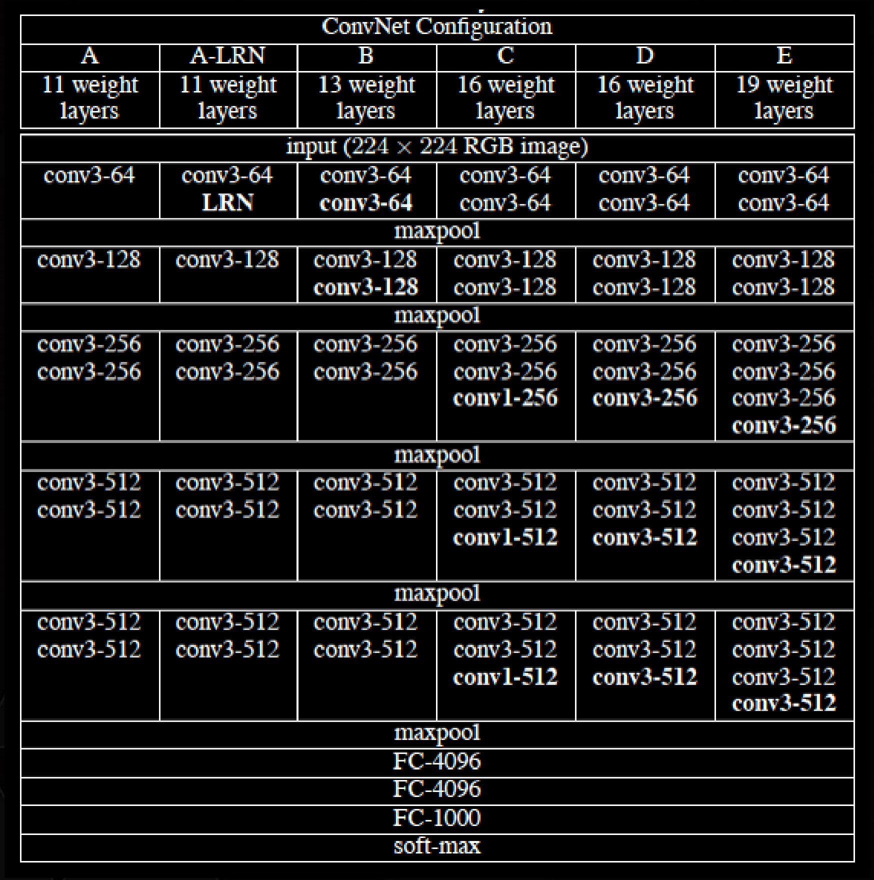
Library	Class	Time (ms)	forward (ms)	backward (ms)
CuDNN[R3]-fp16	cudnn.SpatialConvolution	**313**	**107**	**206**
CuDNN[R3]-fp32	cudnn.SpatialConvolution	326	113	213
fbfft	SpatialConvolutionCuFFT	342	114	227
Nervana-fp16	ConvLayer	355	112	242
Nervana-fp32	ConvLayer	398	124	273
cudaconvnet2*	ConvLayer	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

LEARNING VGG MODEL USING CUDNN



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					

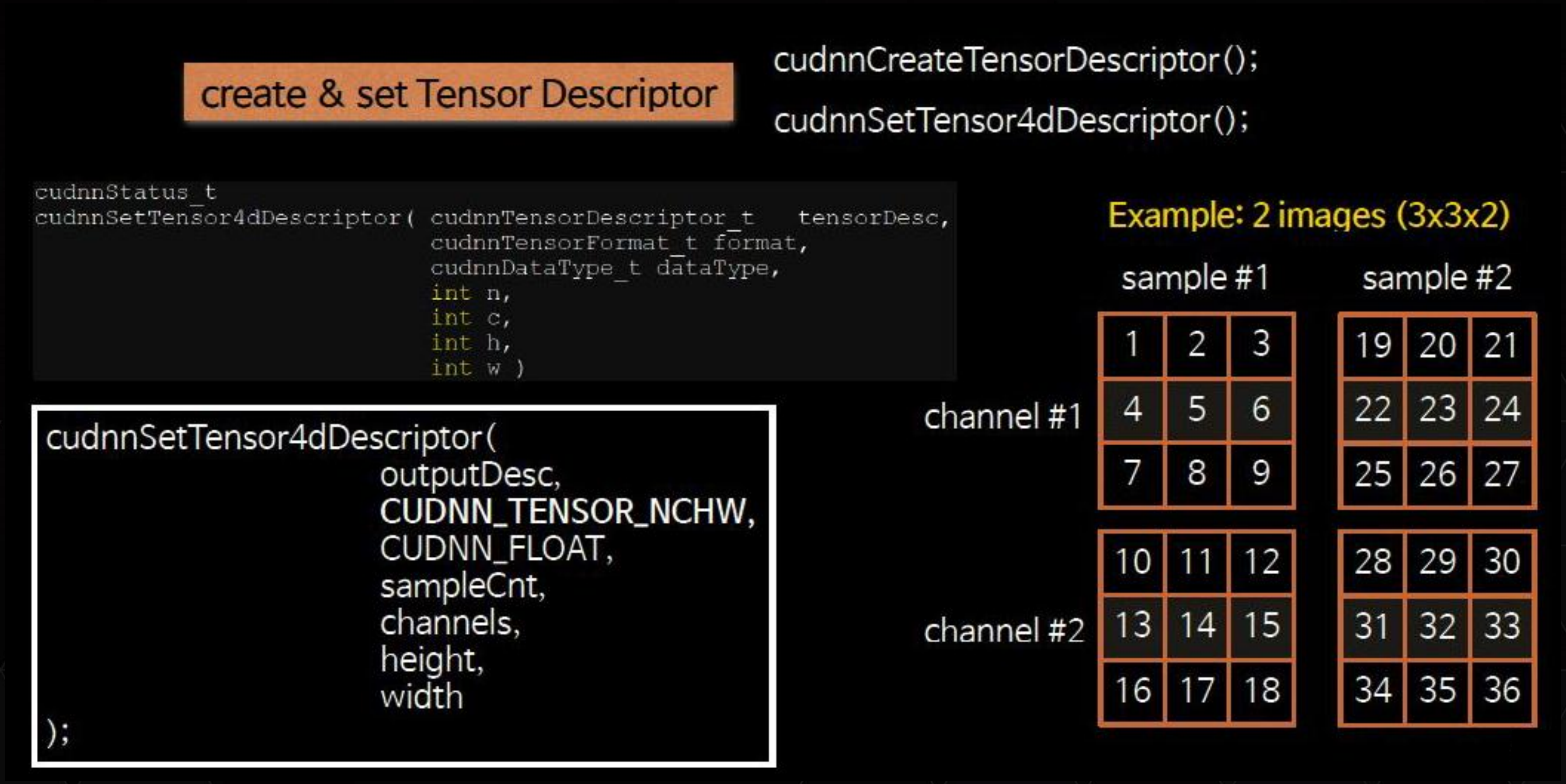
- Data Layer
- Convolution Layer
- Pooling Layer
- Fully Connected Layer
- Softmax Layer

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;
```

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 )
```

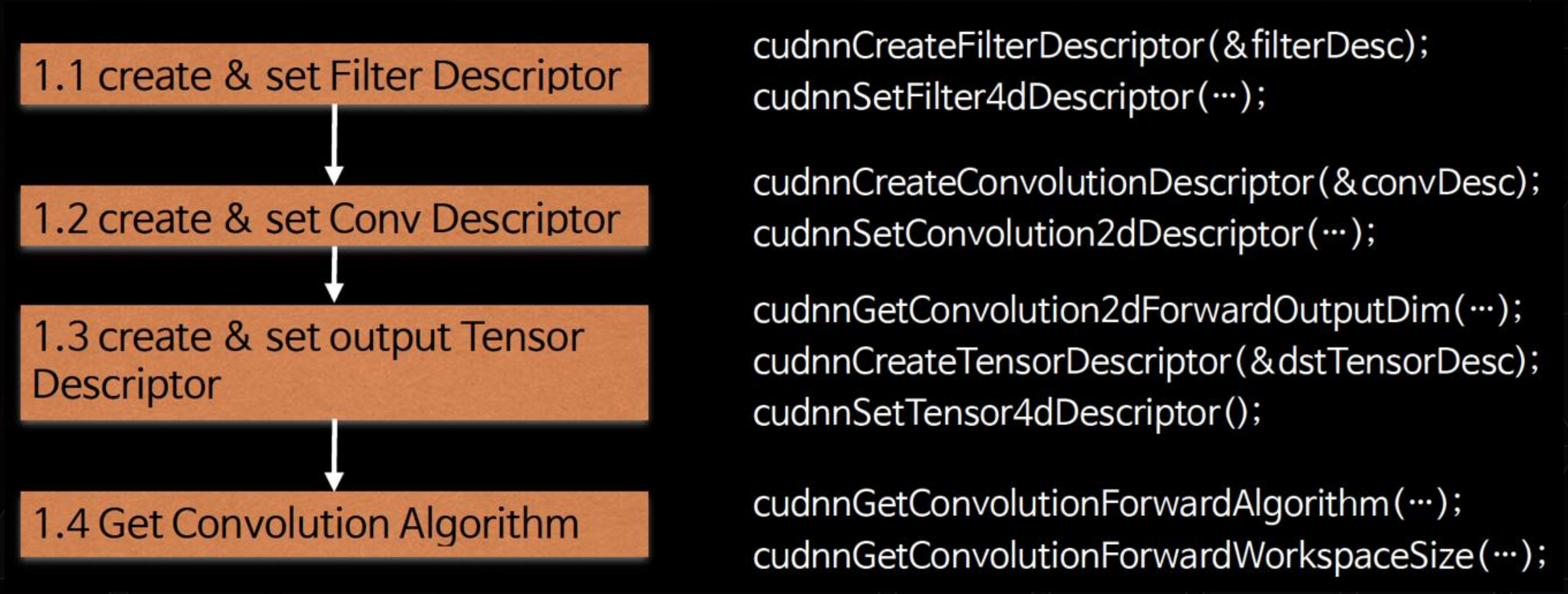
```
cudnnSetTensor4dDescriptor(
                    outputDesc,
                    CUDNN_TENSOR_NCHW,
                    CUDNN_FLOAT,
                    sampleCnt,
                    channels,
                    height,
                    width
);
```

Example: 2 images (3x3x2)

	sample #1			sample #2		
channel #1	1	2	3	19	20	21
	4	5	6	22	23	24
	7	8	9	25	26	27
channel #2	10	11	12	28	29	30
	13	14	15	31	32	33
	16	17	18	34	35	36

CONVOLUTION LAYER

- Initailization



1.1 create & set Filter Descriptor

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

1.2 create & set Conv Descriptor

```
cudnnCreateConvolutionDescriptor(&convDesc);
cudnnSetConvolution2dDescriptor(…);
```

1.3 create & set output Tensor Descriptor

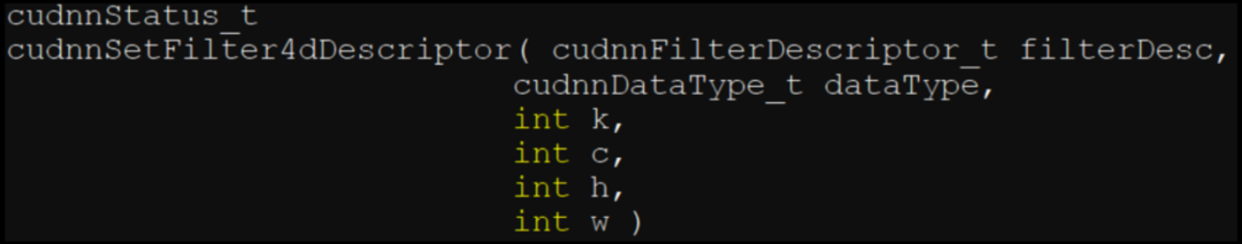
```
cudnnGetConvolution2dForwardOutputDim(…);
cudnnCreateTensorDescriptor(&dstTensorDesc);
cudnnSetTensor4dDescriptor();
```

1.4 Get Convolution Algorithm

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

CONVOLUTION LAYER

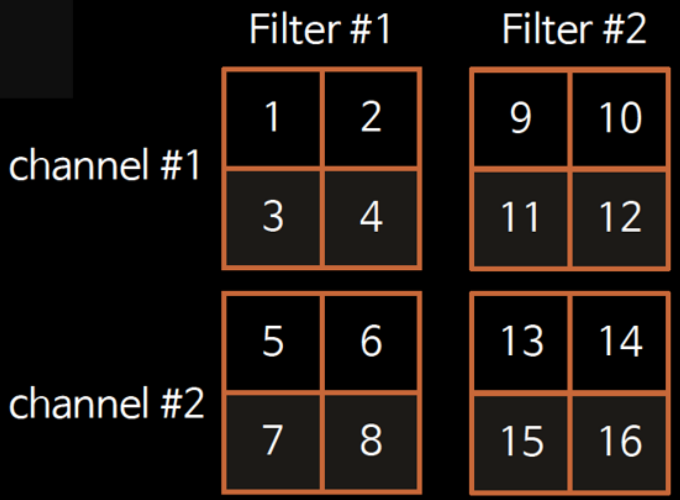
1.1 Set Filter Descriptor



```
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
);
```

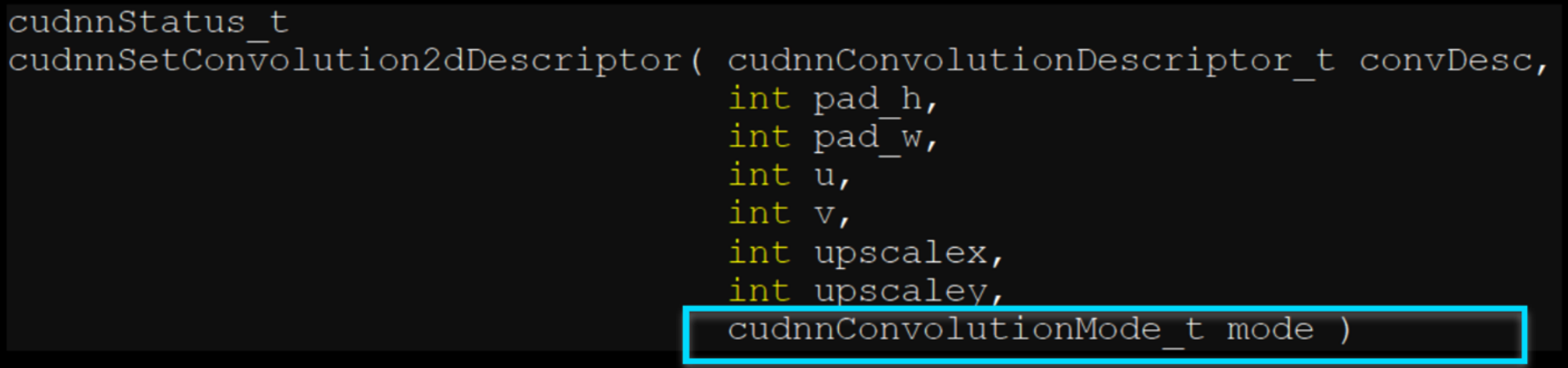
Example: 2 Filters (2x2x2)



	Filter #1		Filter #2	
channel #1	1	2	9	10
	3	4	11	12
channel #2	5	6	13	14
	7	8	15	16

CONVOLUTION LAYER

1.2 Set Convolution Descriptor

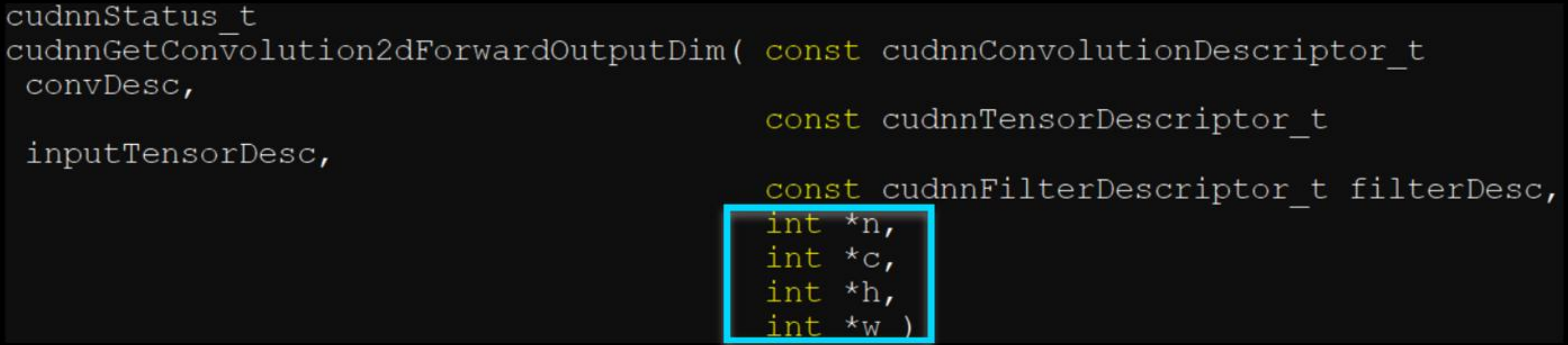


```
cudnnStatus_t
cudnnSetConvolution2dDescriptor( cudnnConvolutionDescriptor_t convDesc,
                                 int pad_h,
                                 int pad_w,
                                 int u,
                                 int v,
                                 int upscalex,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

CUDNN_CROSS_CORRELATION

CONVOLUTION LAYER

1.3 Set output Tensor Descriptor

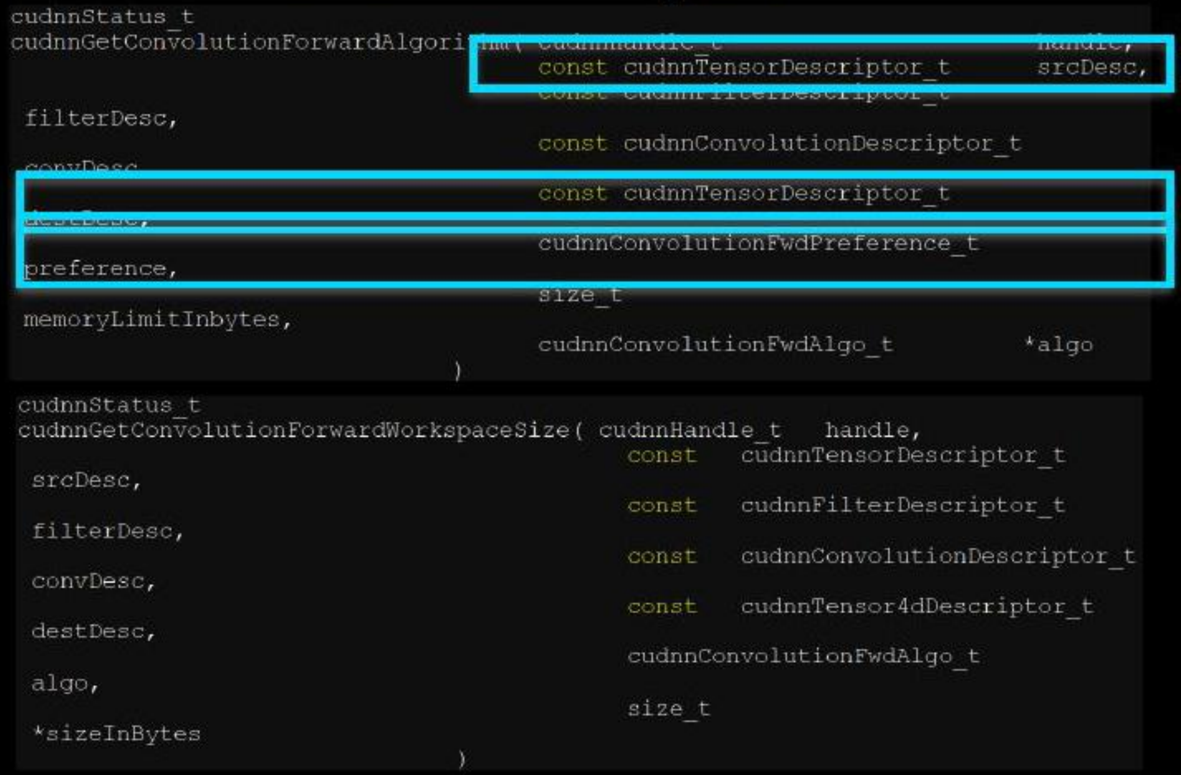


```
cudnnStatus_t
cudnnGetConvolution2dForwardOutputDim( const cudnnConvolutionDescriptor_t
 convDesc,
                                       const cudnnTensorDescriptor_t
 inputTensorDesc,
                                       const cudnnFilterDescriptor_t filterDesc,
                                       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
                                   )
```

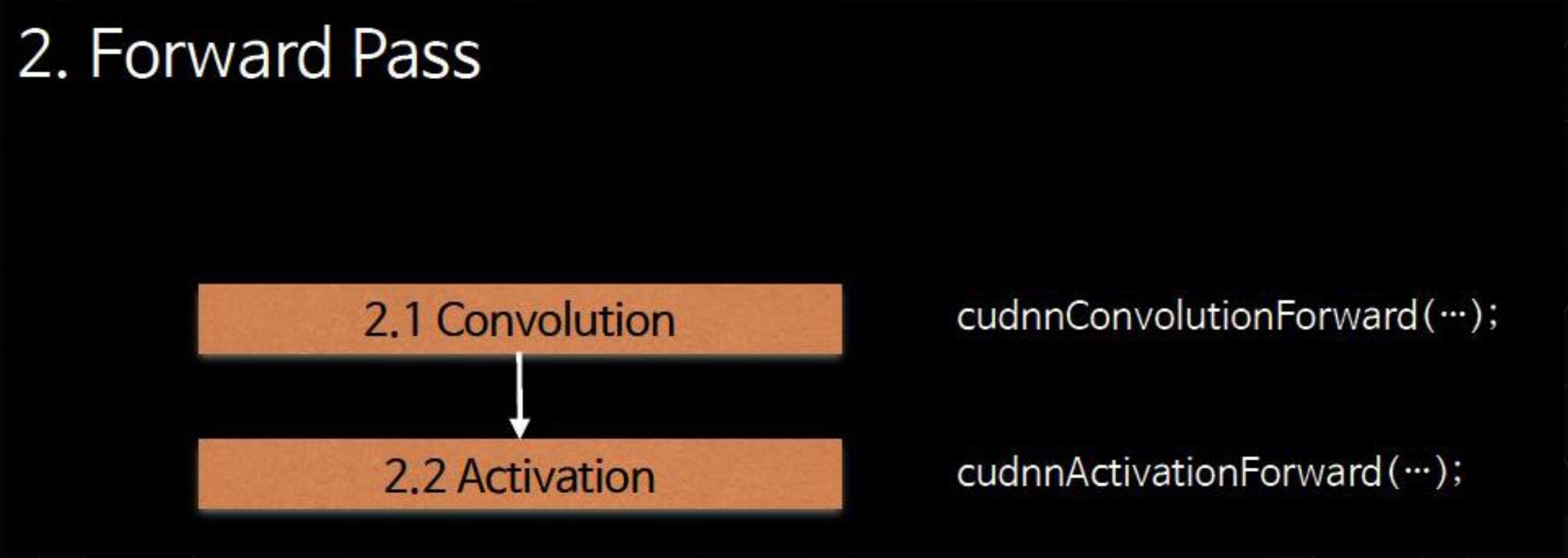
```
cudnnStatus_t
cudnnGetConvolutionForwardWorkspaceSize( cudnnHandle_t   handle,
                                         const   cudnnTensorDescriptor_t
 srcDesc,
                                         const   cudnnFilterDescriptor_t
 filterDesc,
                                         const  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

inputDesc

outputDesc

CUDNN_CONVOLUTION_FWD_PREFER_FASTEST

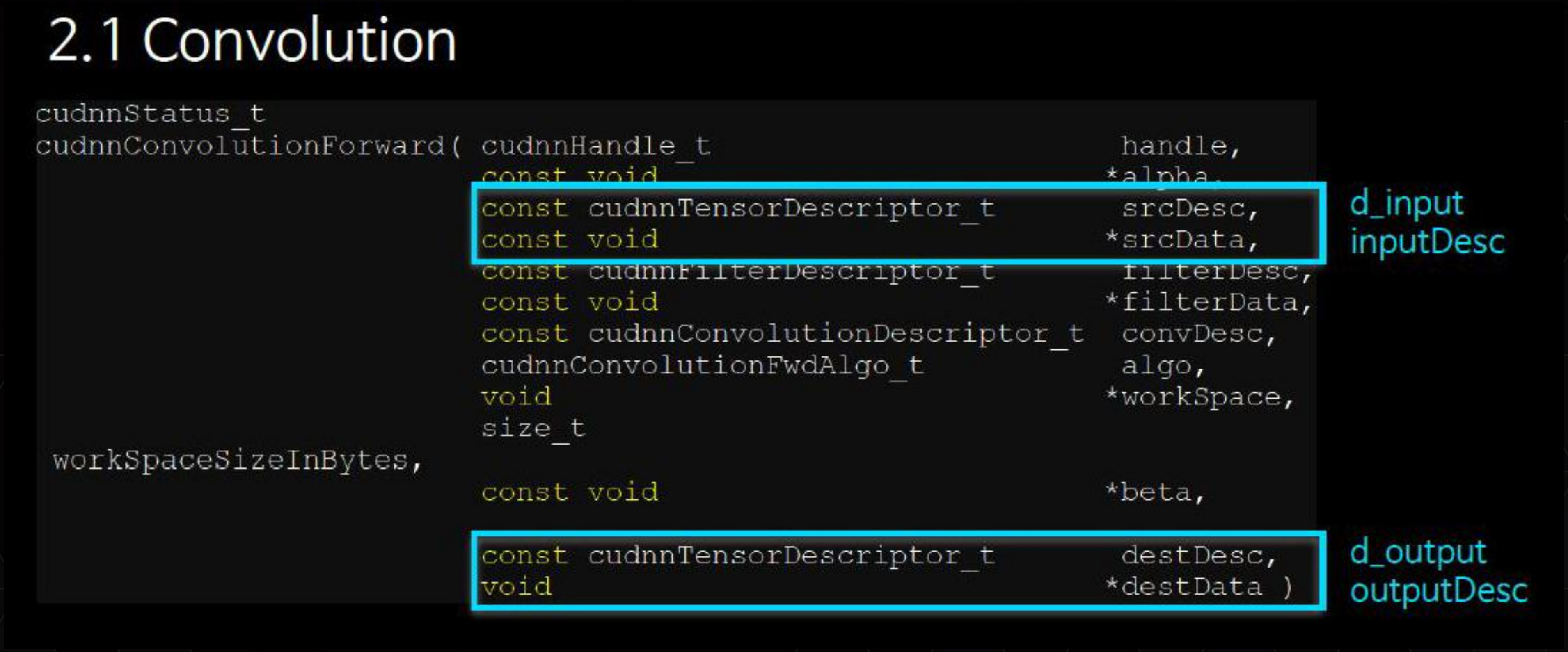
CONVOLUTION LAYER



2. Forward Pass

```
cudnnConvolutionForward(…);
cudnnActivationForward(…);
```

CONVOLUTION LAYER



2.1 Convolution

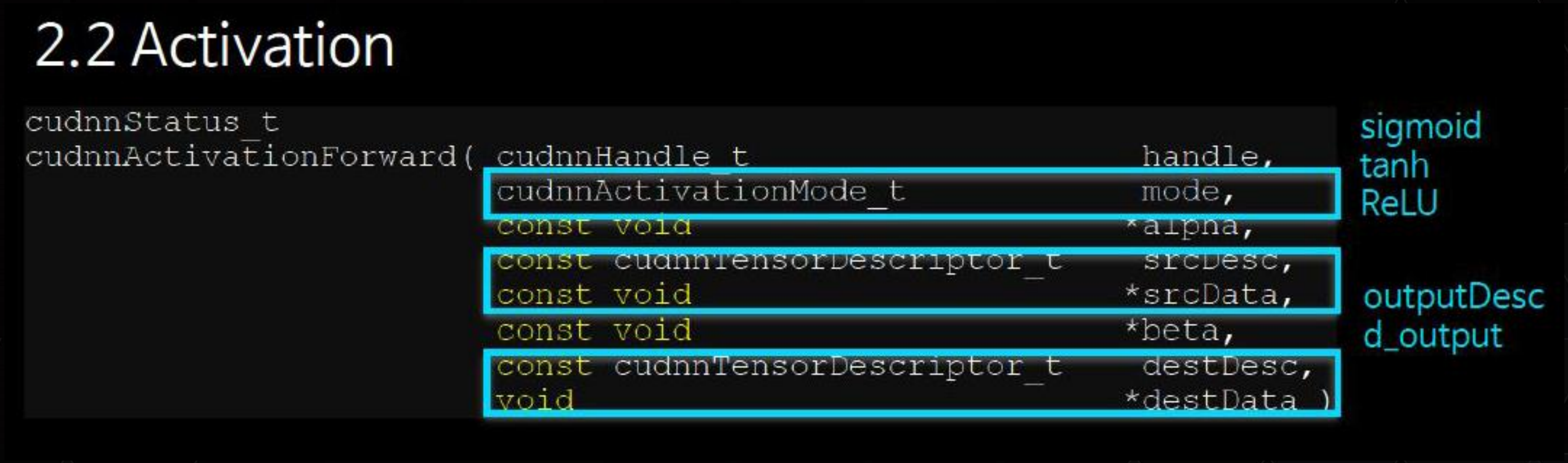
```
cudnnStatus_t
cudnnConvolutionForward( cudnnHandle_t                          handle,
                         const void                            *alpha,
                         const cudnnTensorDescriptor_t          srcDesc,
                         const void                            *srcData,
                         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

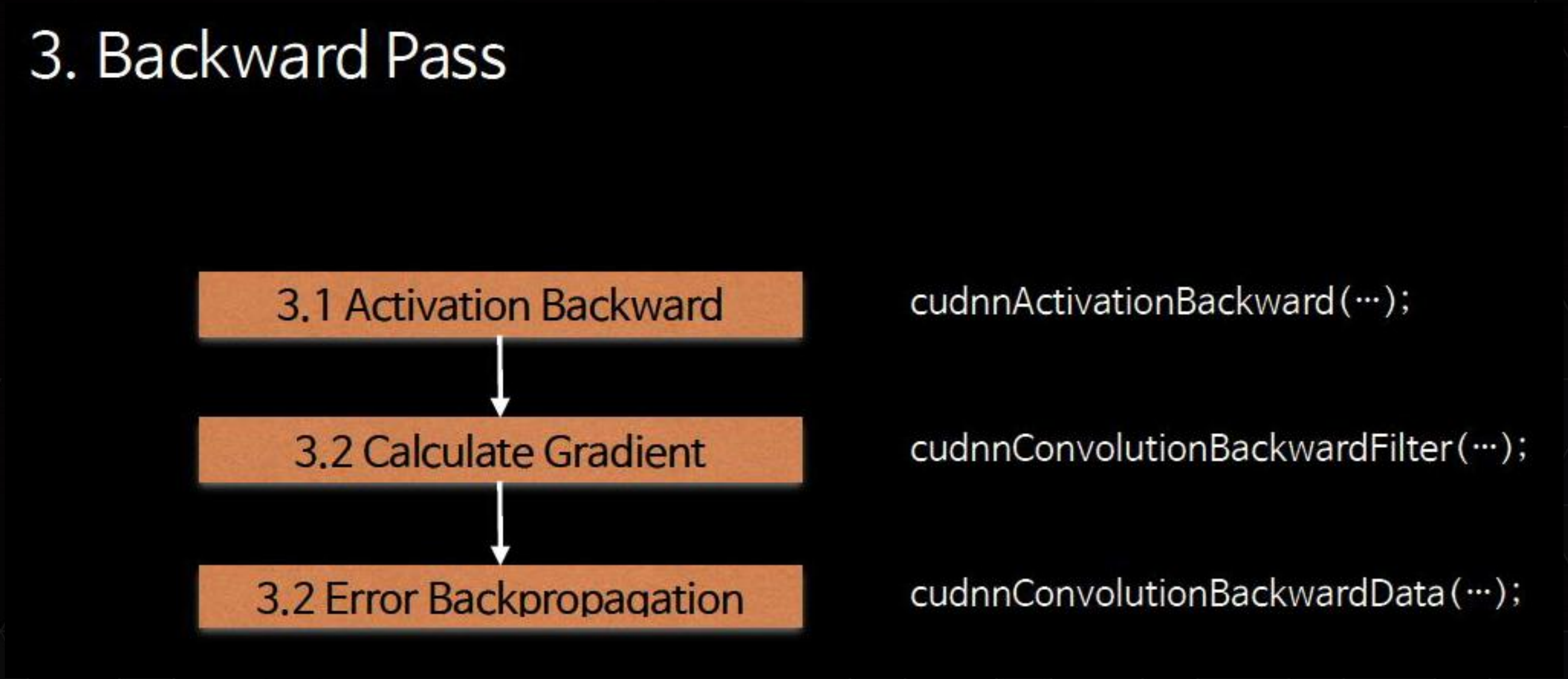
CONVOLUTION LAYER



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 )
```

CONVOLUTION LAYER



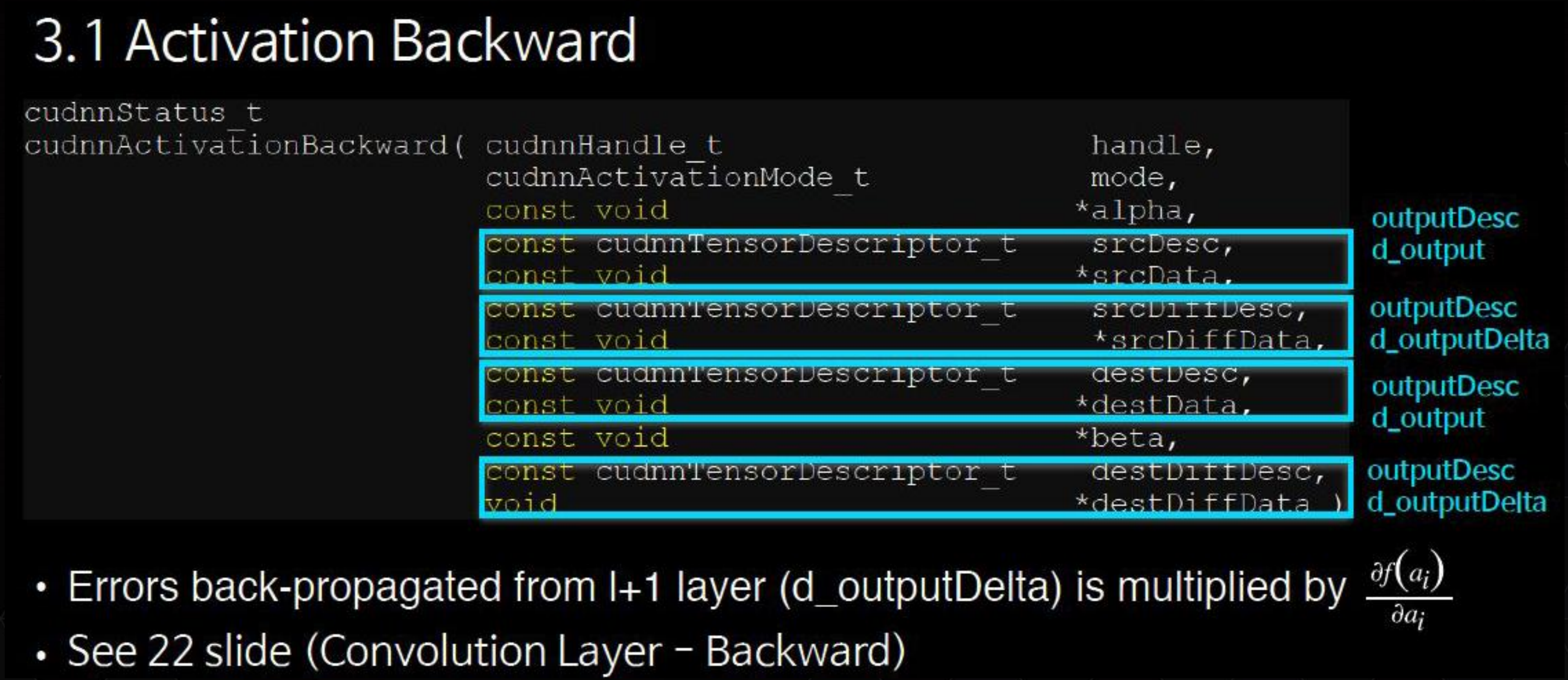
3. Backward Pass

3.1 Activation Backward — cudnnActivationBackward(…);

3.2 Calculate Gradient — cudnnConvolutionBackwardFilter(…);

3.2 Error Backpropagation — 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

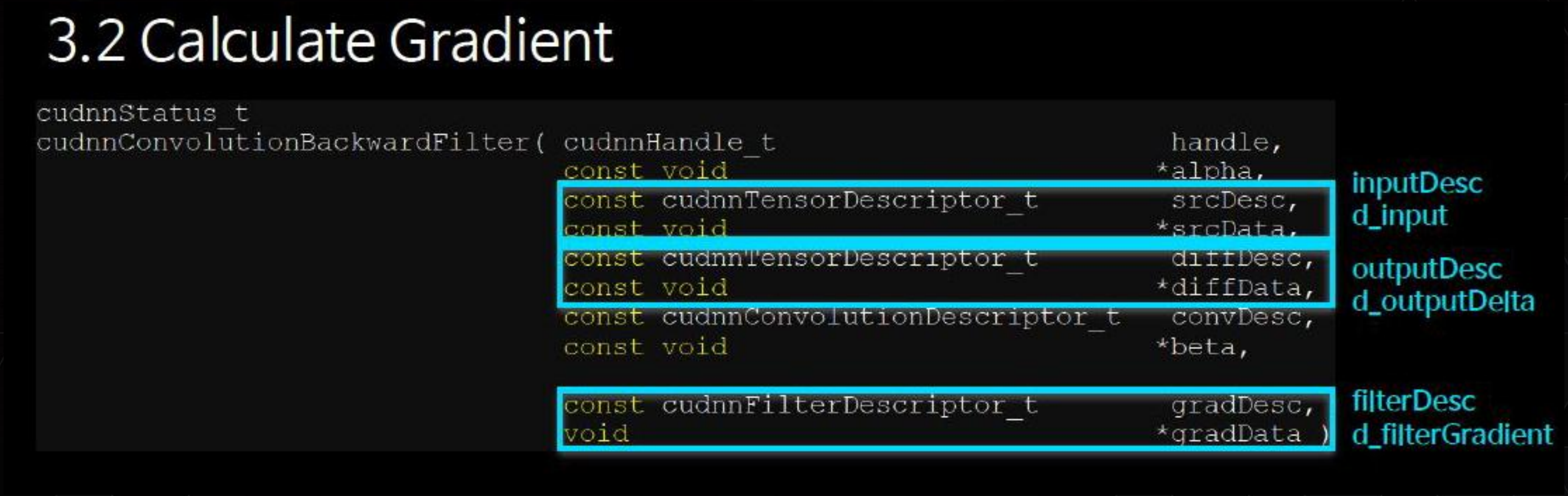
outputDesc
d_outputDelta

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

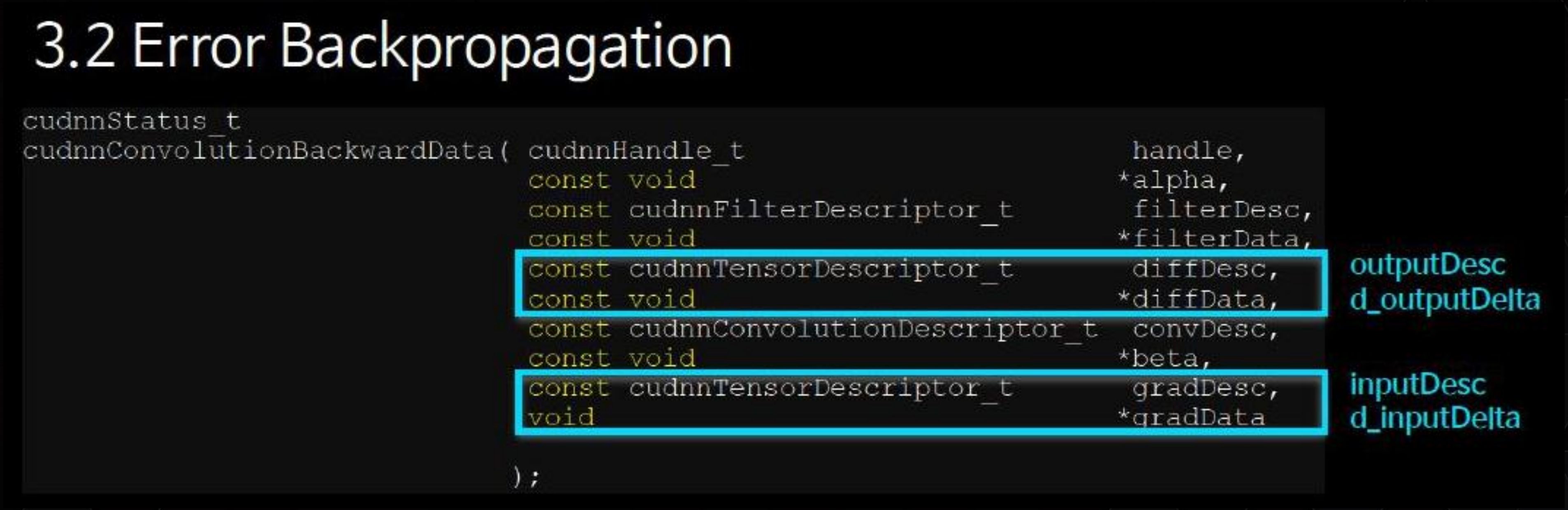


3.2 Calculate Gradient

```
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 )
```

CONVOLUTION LAYER



3.2 Error Backpropagation

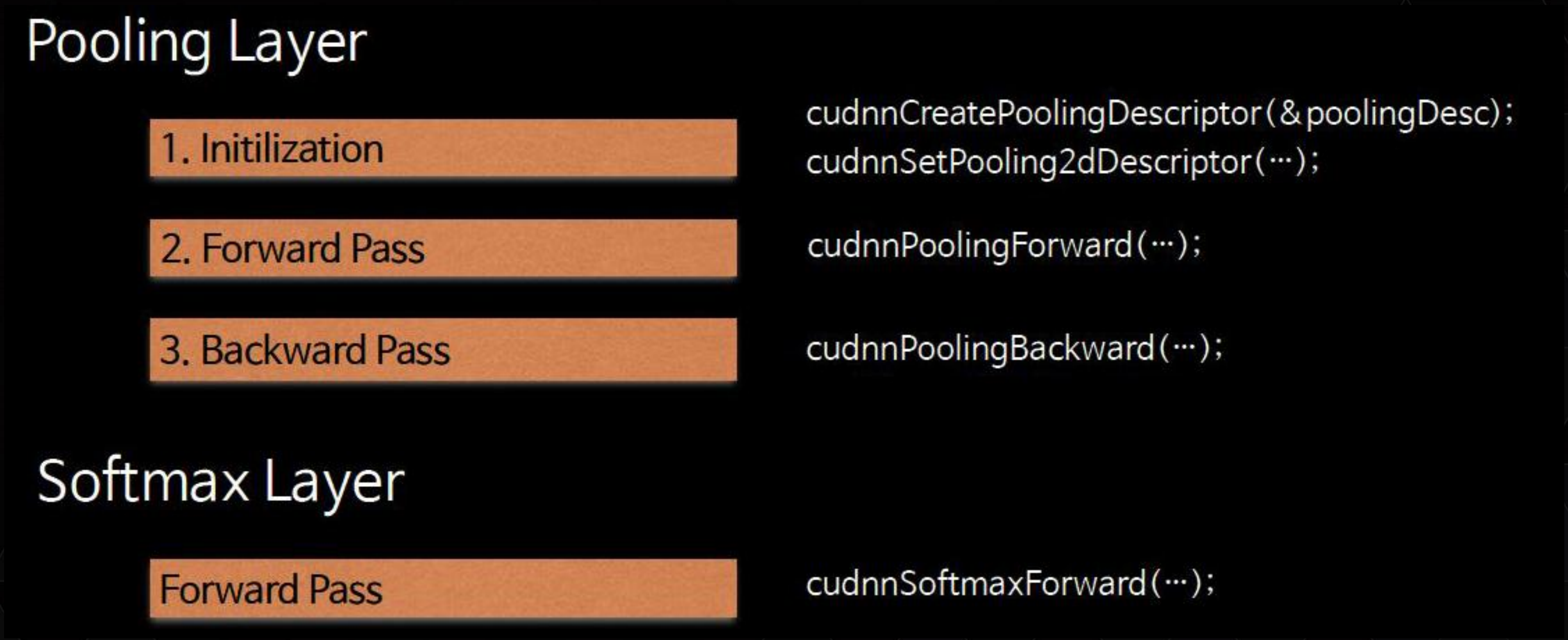
```
cudnnStatus_t
cudnnConvolutionBackwardData( cudnnHandle_t                       handle,
                              const void                         *alpha,
                              const cudnnFilterDescriptor_t       filterDesc,
                              const void                         *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(…);

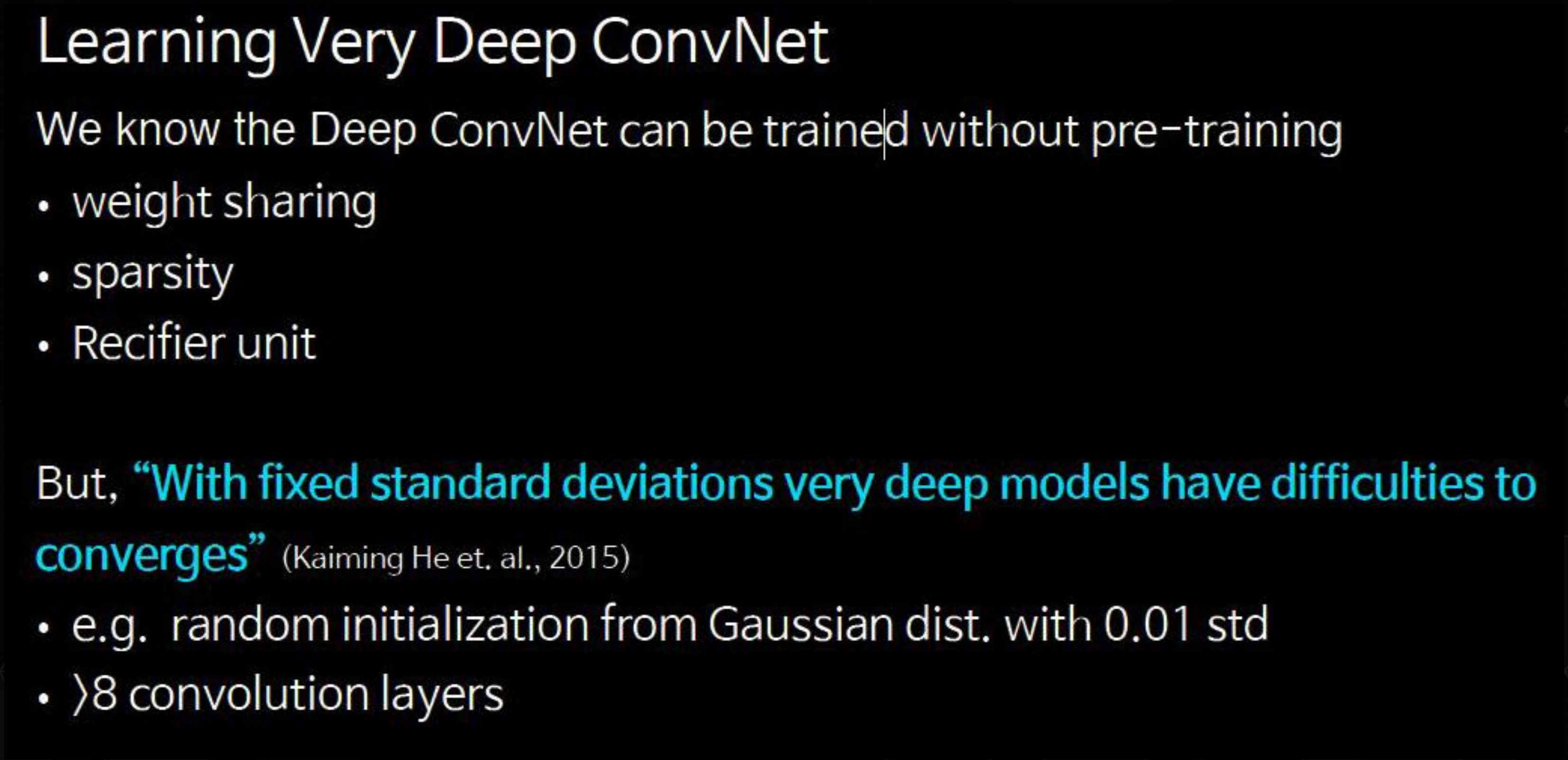
3. Backward Pass — cudnnPoolingBackward(…);

Softmax Layer

Forward Pass — cudnnSoftmaxForward(…);

OPTIMIZATION ISSUES

OPTIMIZATION



Learning Very Deep ConvNet

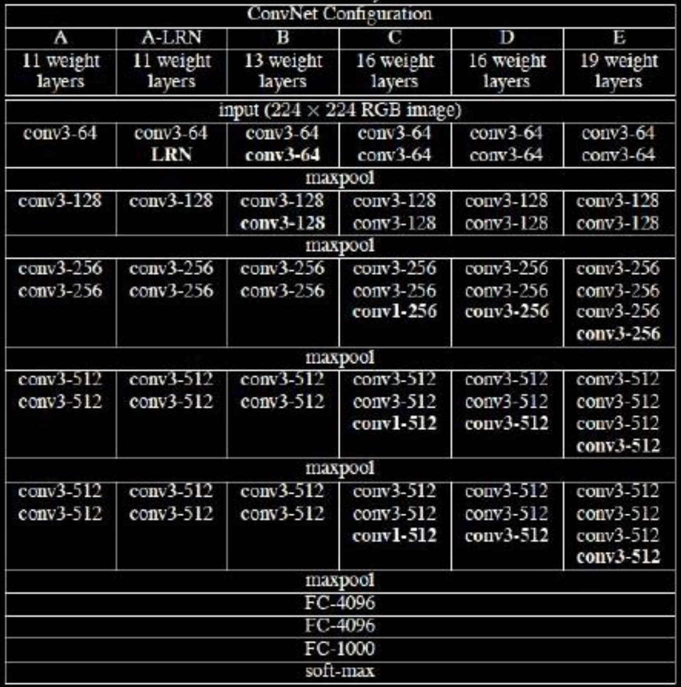
We know the Deep ConvNet can be trained without pre-training

- weight sharing
- sparsity
- Recifier unit

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

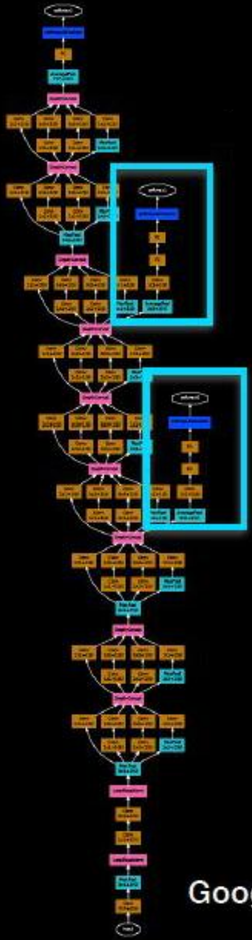
- e.g. random initialization from Gaussian dist. with 0.01 std
- 〉8 convolution layers

OPTIMIZATION



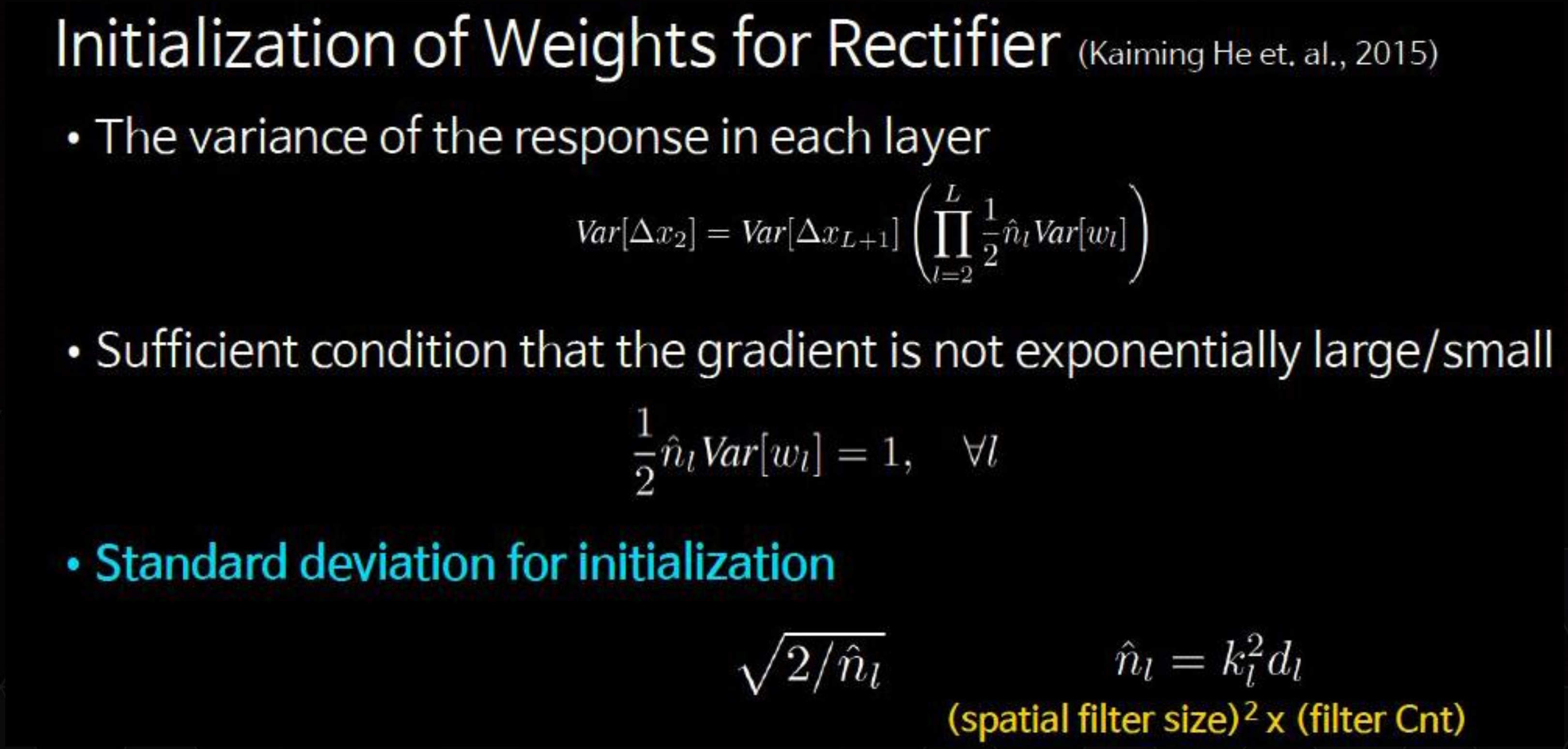
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	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

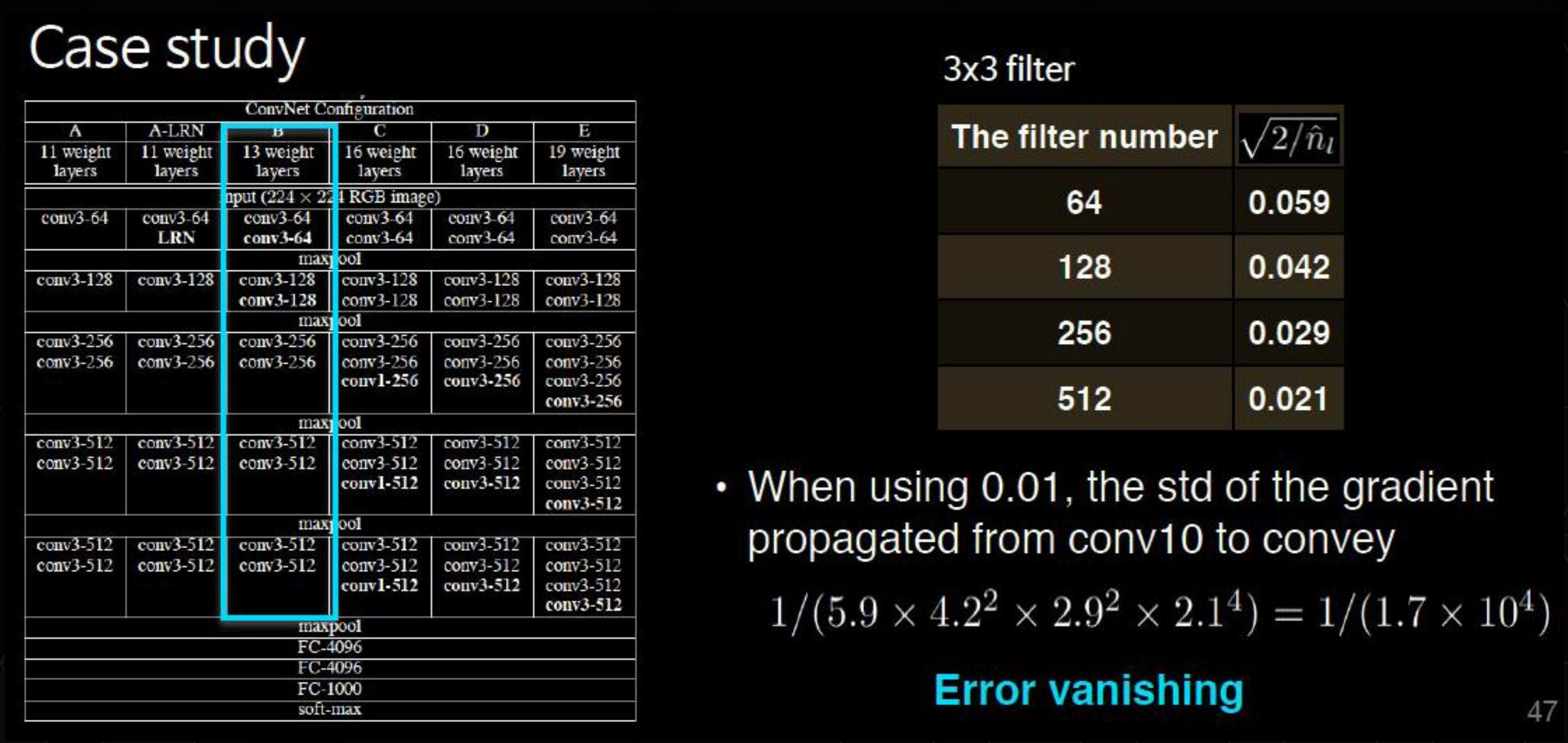
$$\frac{1}{2}\hat{n}_l Var[w_l] = 1, \quad \forall l$$

- Standard deviation for initialization

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

(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

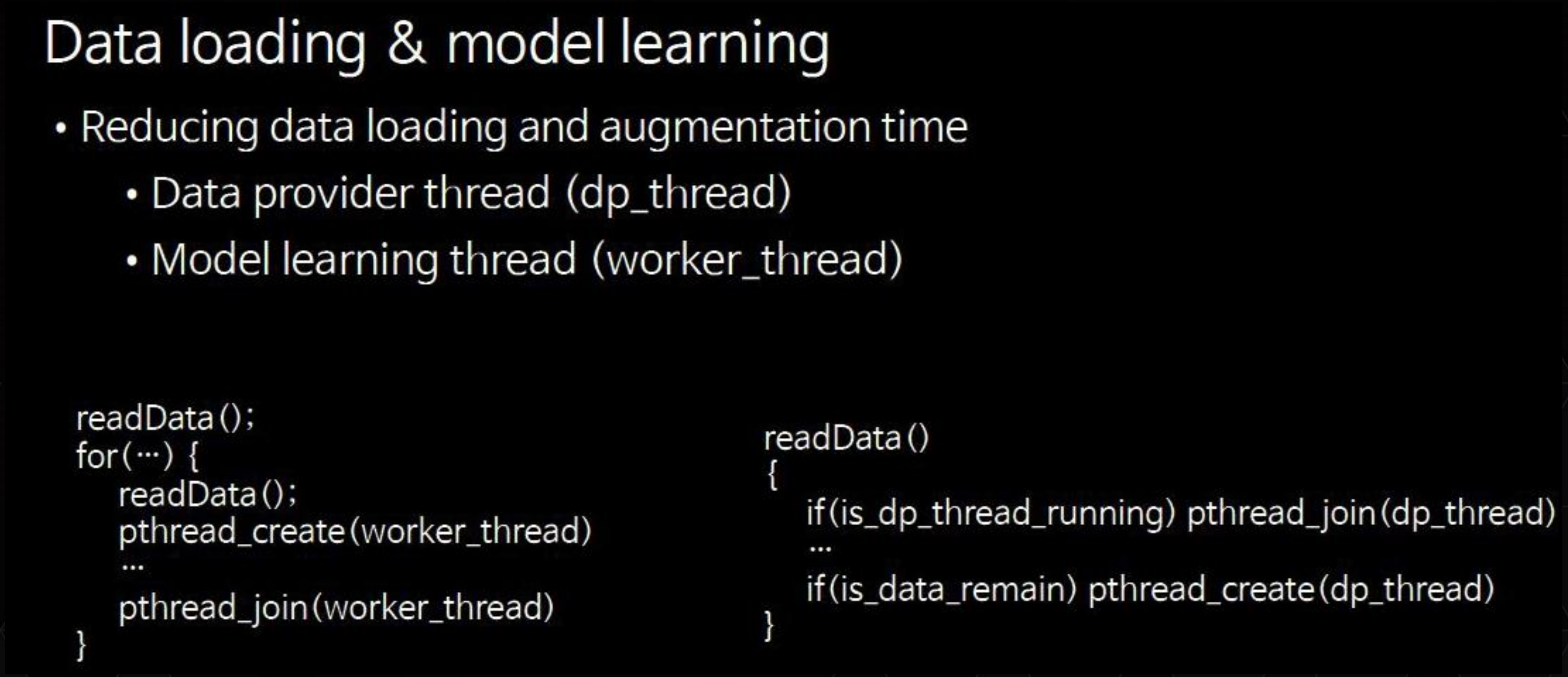
The filter number	$\sqrt{2/\hat{n}_l}$
64	0.059
128	0.042
256	0.029
512	0.021

- 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



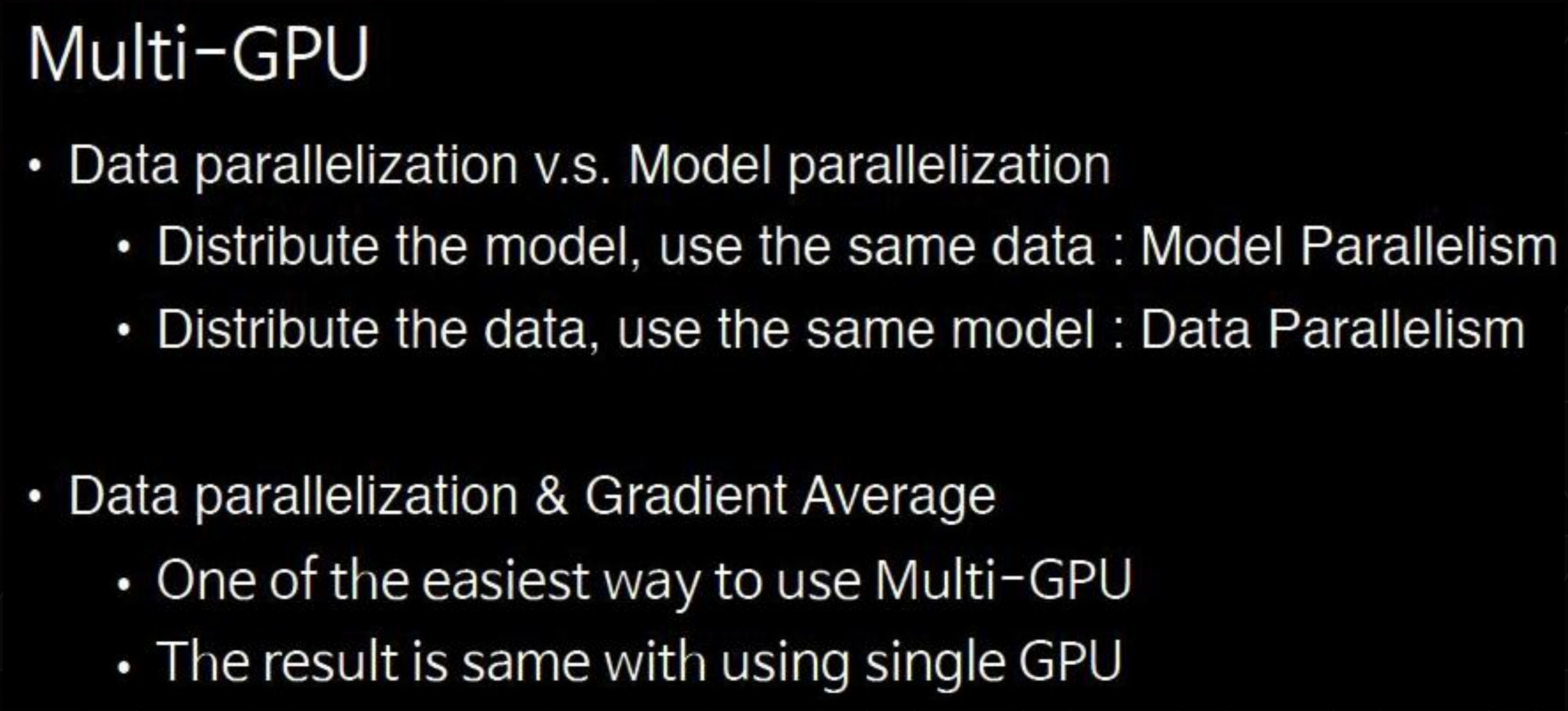
Data loading & model learning

- 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)
}
```

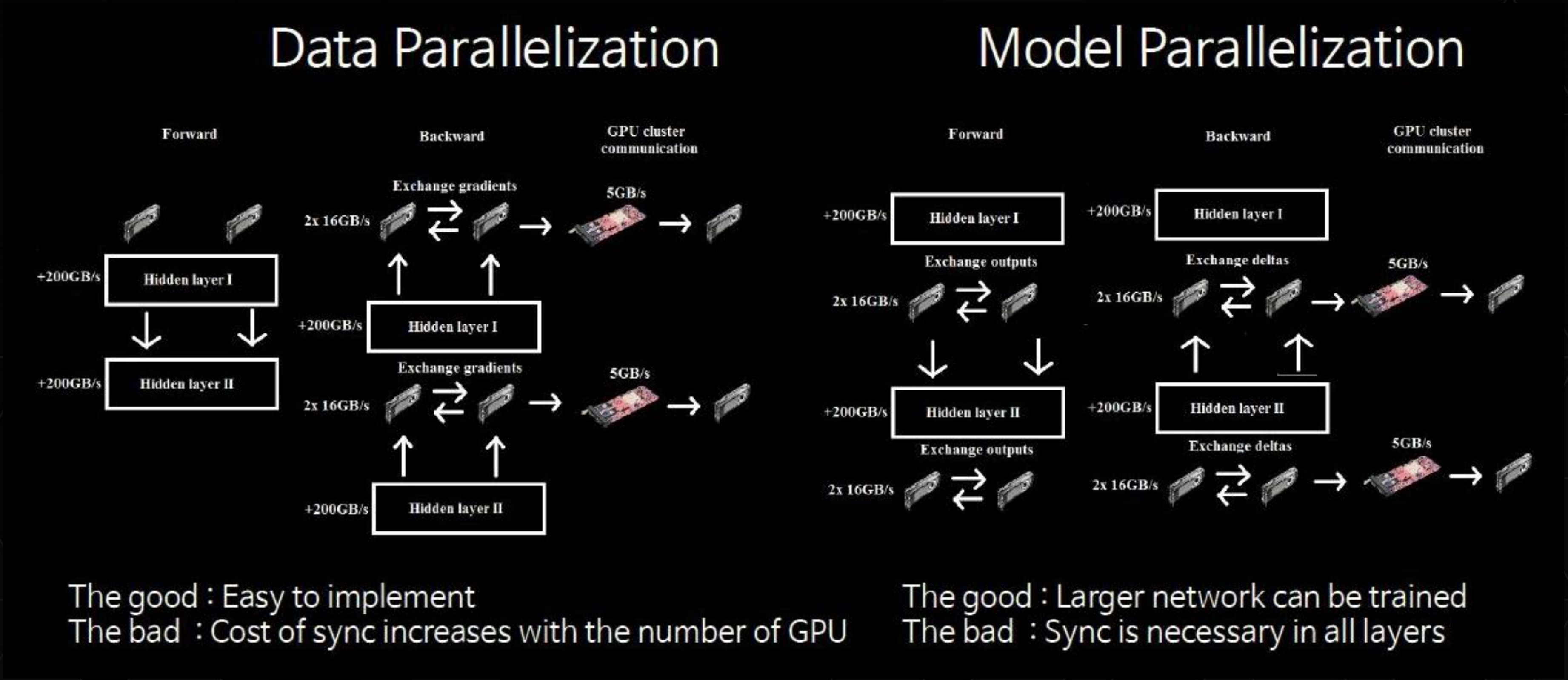
SPEED



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

Forward

+200GB/s Hidden layer I

+200GB/s Hidden layer II

Backward

Exchange gradients

2x 16GB/s

+200GB/s Hidden layer I

Exchange gradients

2x 16GB/s

+200GB/s Hidden layer II

GPU cluster communication

5GB/s

5GB/s

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

Model Parallelization

Forward

+200GB/s Hidden layer I

Exchange outputs

2x 16GB/s

+200GB/s Hidden layer II

Exchange outputs

2x 16GB/s

Backward

+200GB/s Hidden layer I

Exchange deltas

2x 16GB/s

+200GB/s Hidden layer II

Exchange deltas

2x 16GB/s

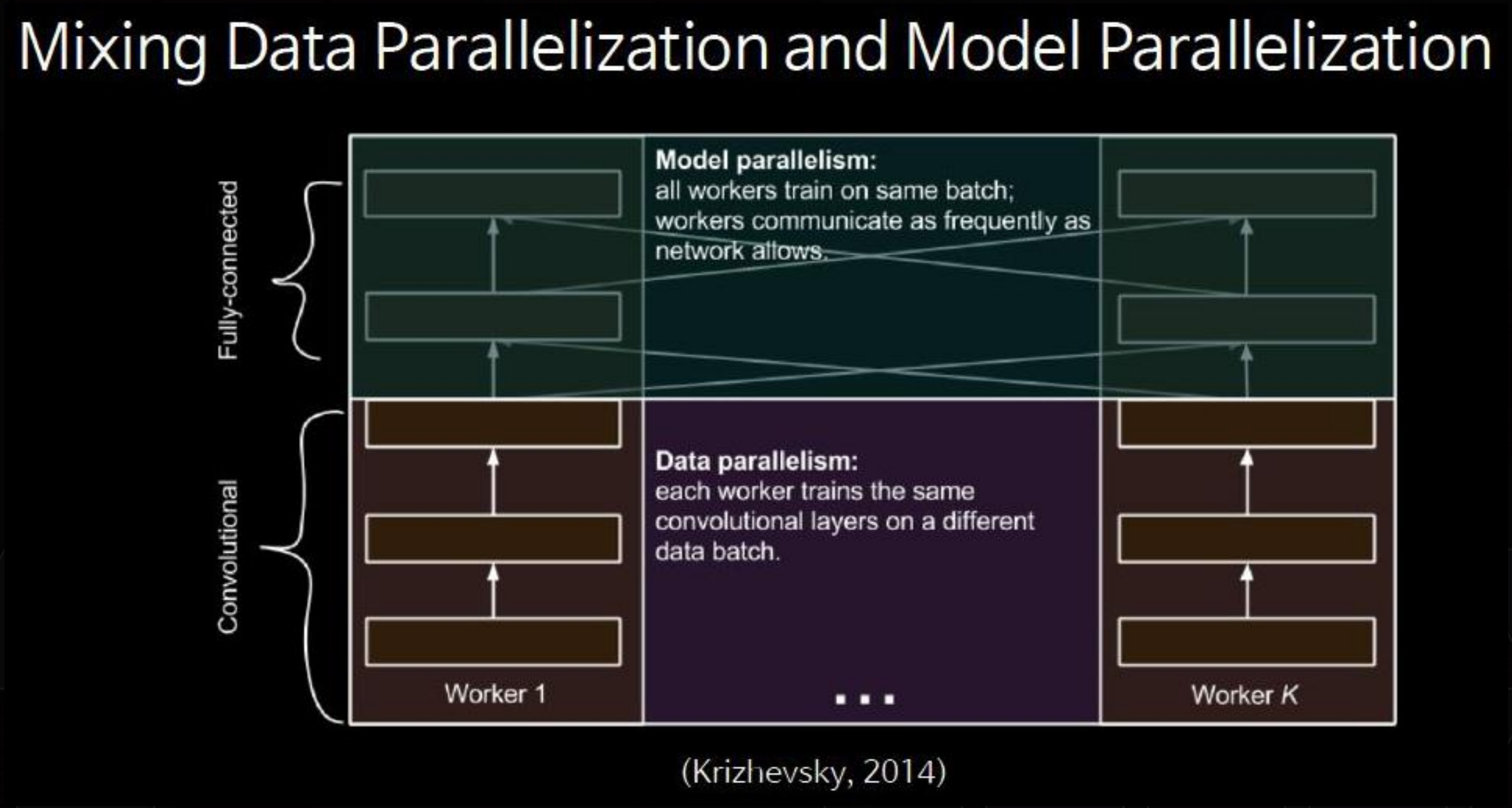
GPU cluster communication

5GB/s

5GB/s

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

PARALLELISM



Mixing Data Parallelization and Model Parallelization

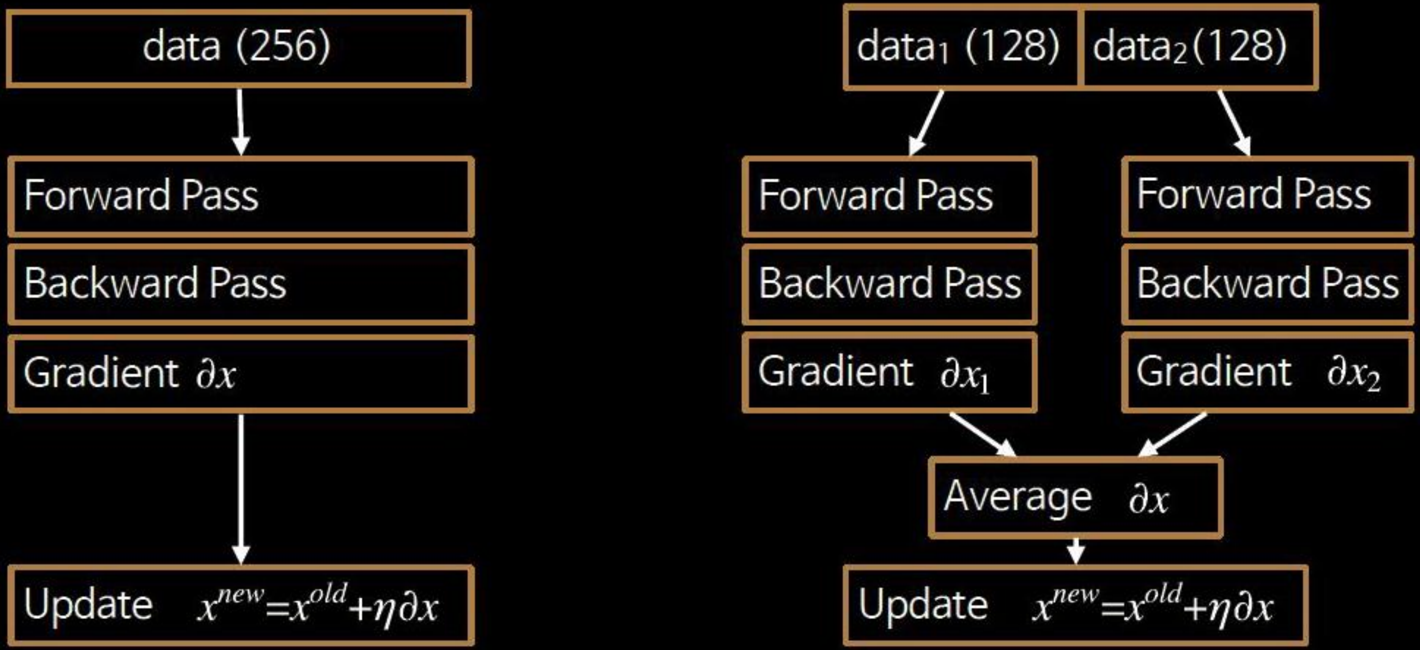
Model parallelism: all workers train on same batch; workers communicate as frequently as network allows.

Data parallelism: each worker trains the same convolutional layers on a different data batch.

(Krizhevsky, 2014)

PARALLELISM

Data Parallelization & Gradient Average



data (256)

Forward Pass

Backward Pass

Gradient ∂x

Update $x^{new}=x^{old}+\eta\partial x$

data$_1$ (128)

data$_2$ (128)

Forward Pass

Backward Pass

Gradient ∂x_1

Forward Pass

Backward Pass

Gradient ∂x_2

Average ∂x

Update $x^{new}=x^{old}+\eta\partial x$

INTRODUCING VUNO-NET

THE TEAM

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]

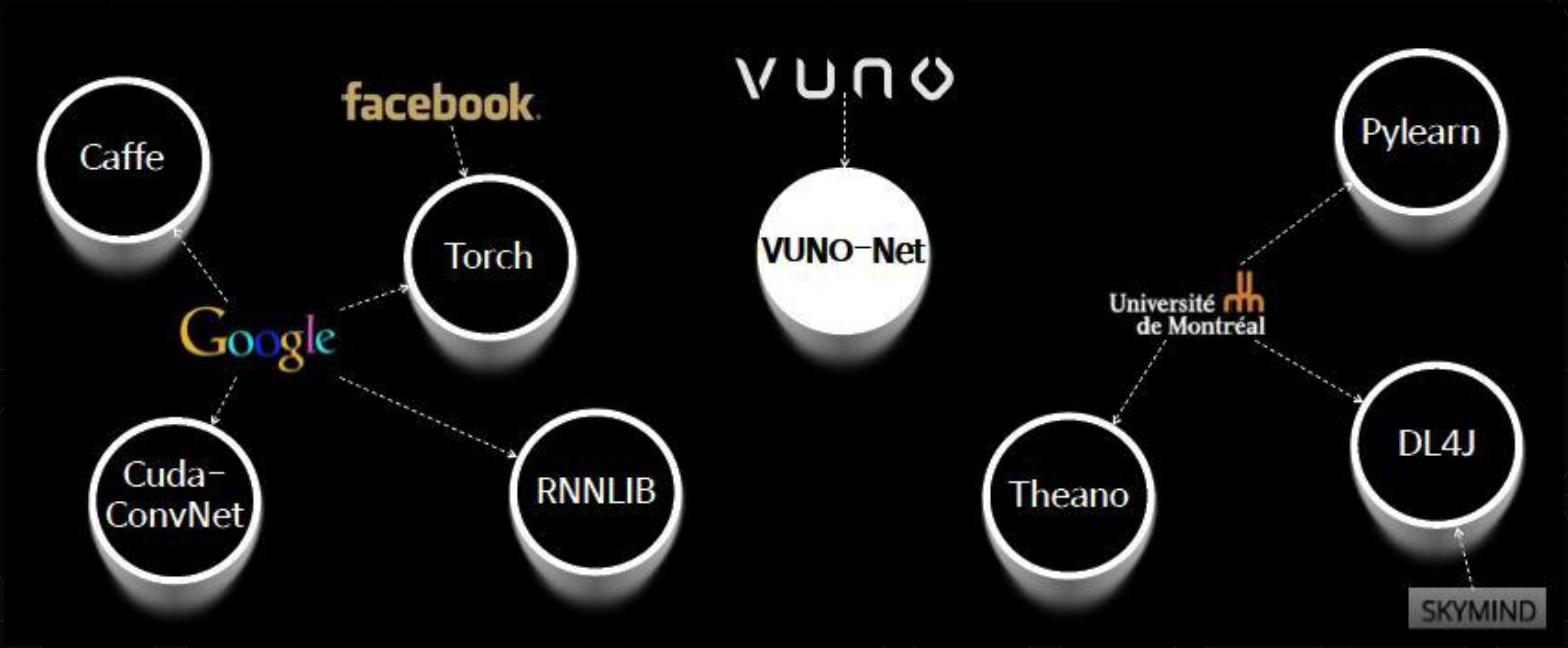
LG전자
미래IT융합연구소

최윤섭(Advisor)

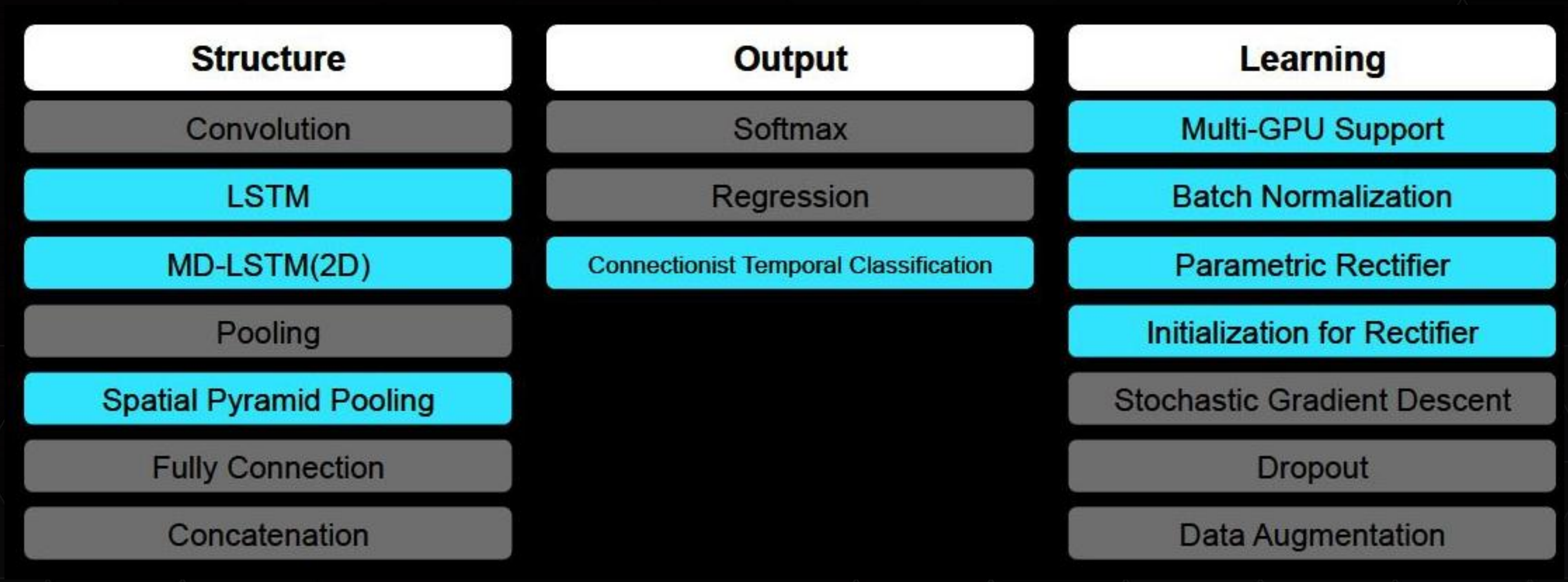
포항공대 생명과학 박사
[전산생물학/디지털 헬스케어]

서울대학교병원
SEOUL NATIONAL UNIVERSITY HOSPITAL

VUNO-NET

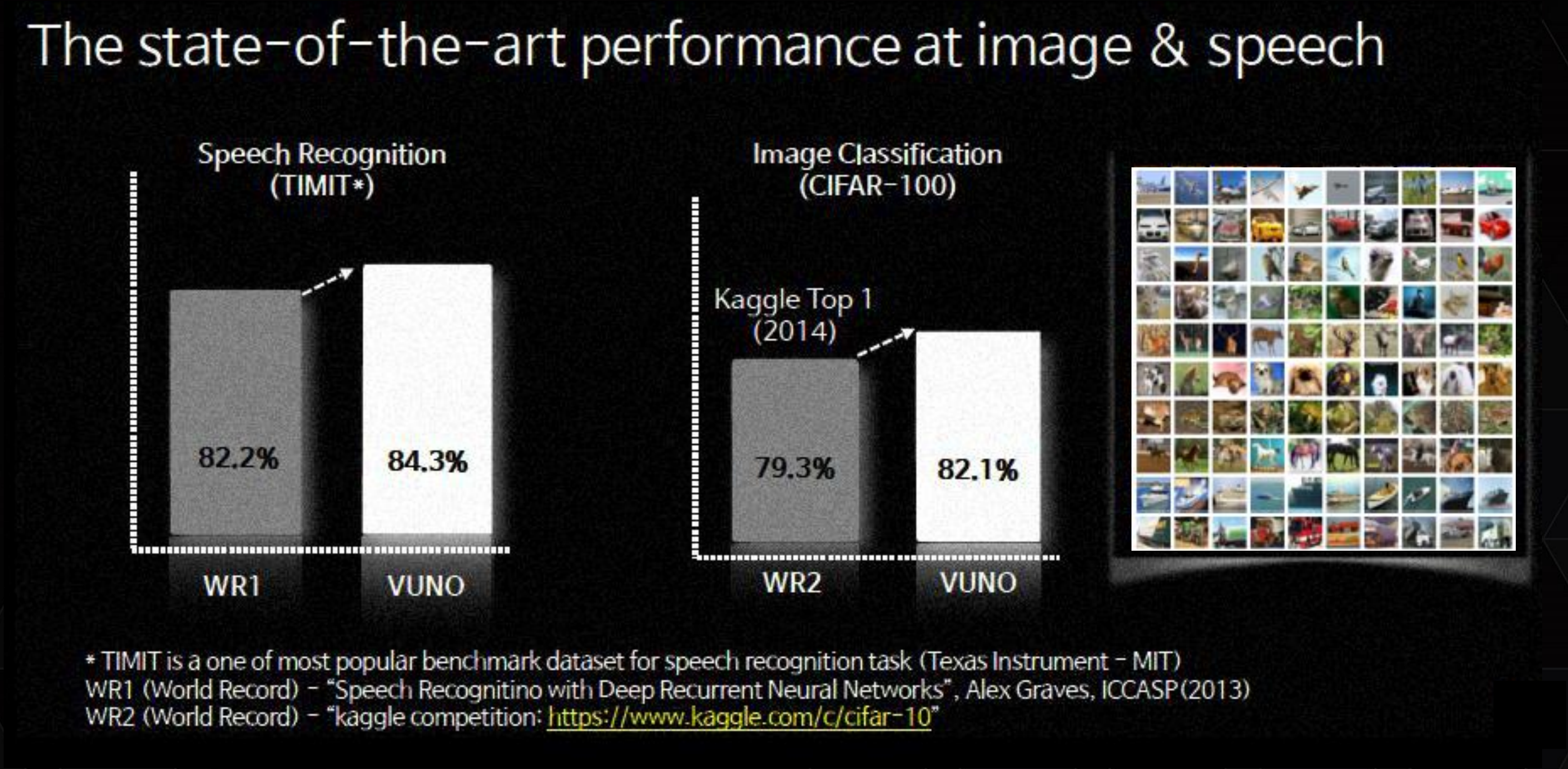

Caffe
facebook
Torch
Google
Cuda-ConvNet
RNNLIB
VUNO
VUNO-Net
Pylearn
Université de Montréal
Theano
DL4J
SKYMIND

VUNO-NET



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



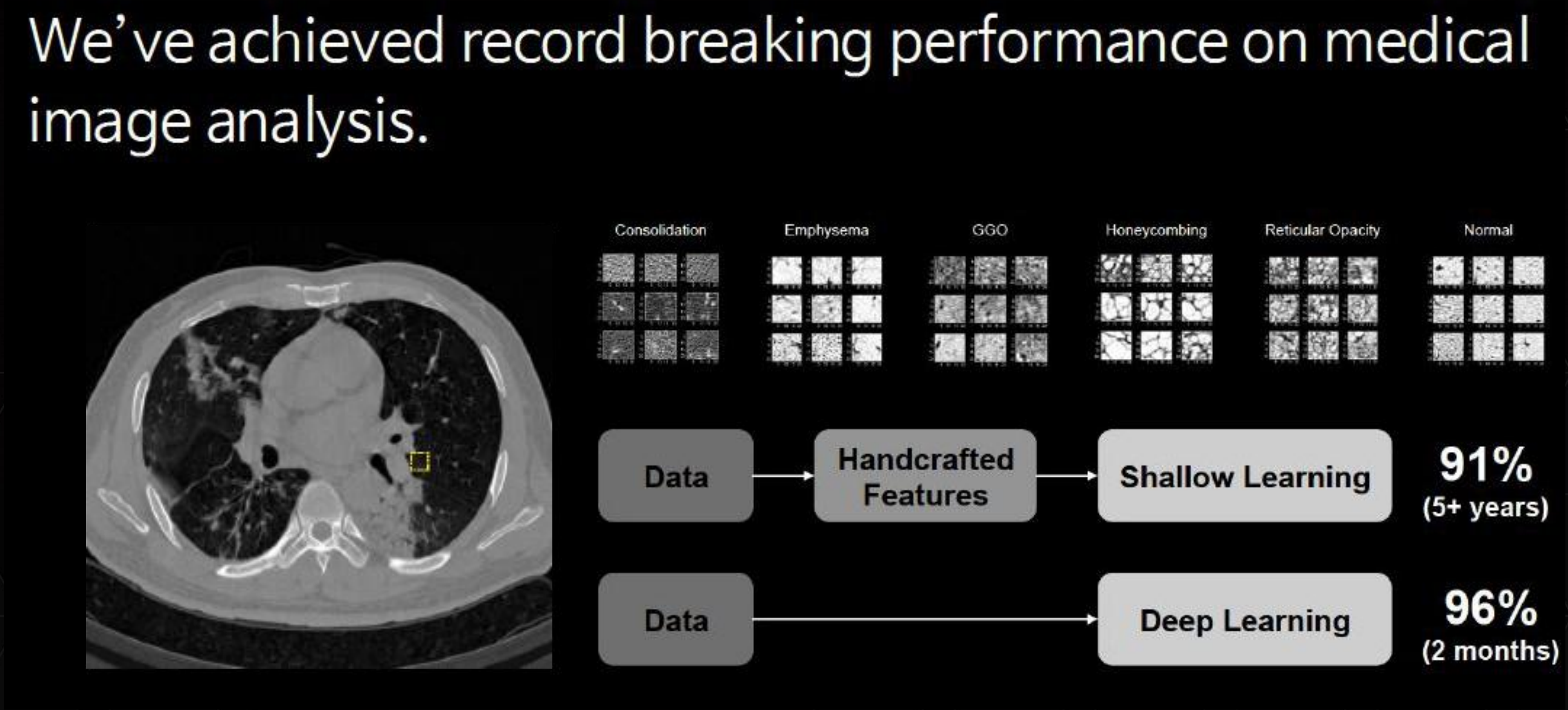
The state-of-the-art performance at image & speech

Speech Recognition (TIMIT*)

Image Classification (CIFAR-100)

* 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

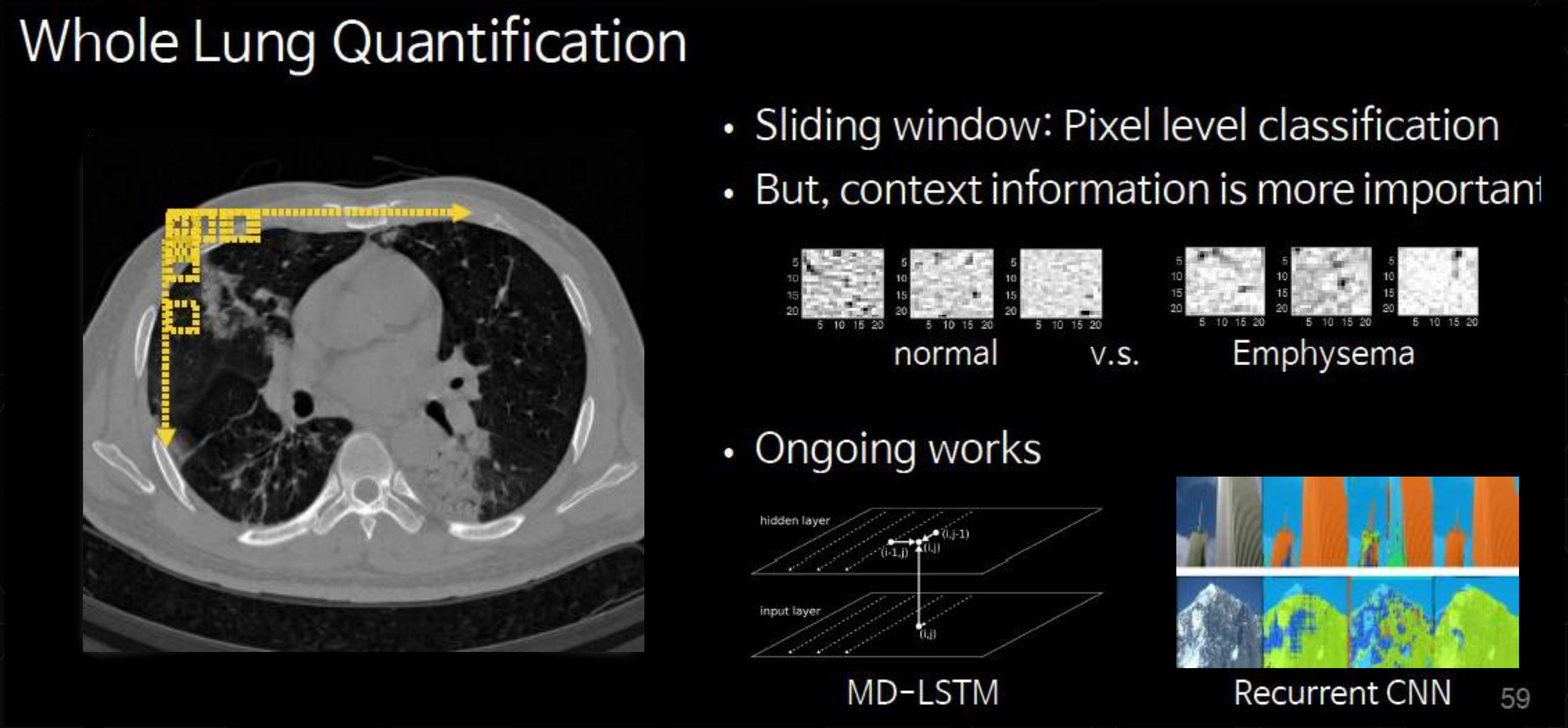


We've achieved record breaking performance on medical image analysis.

Data → Handcrafted Features → Shallow Learning — 91% (5+ years)

Data → Deep Learning — 96% (2 months)

APPLICATION



Whole Lung Quantification

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

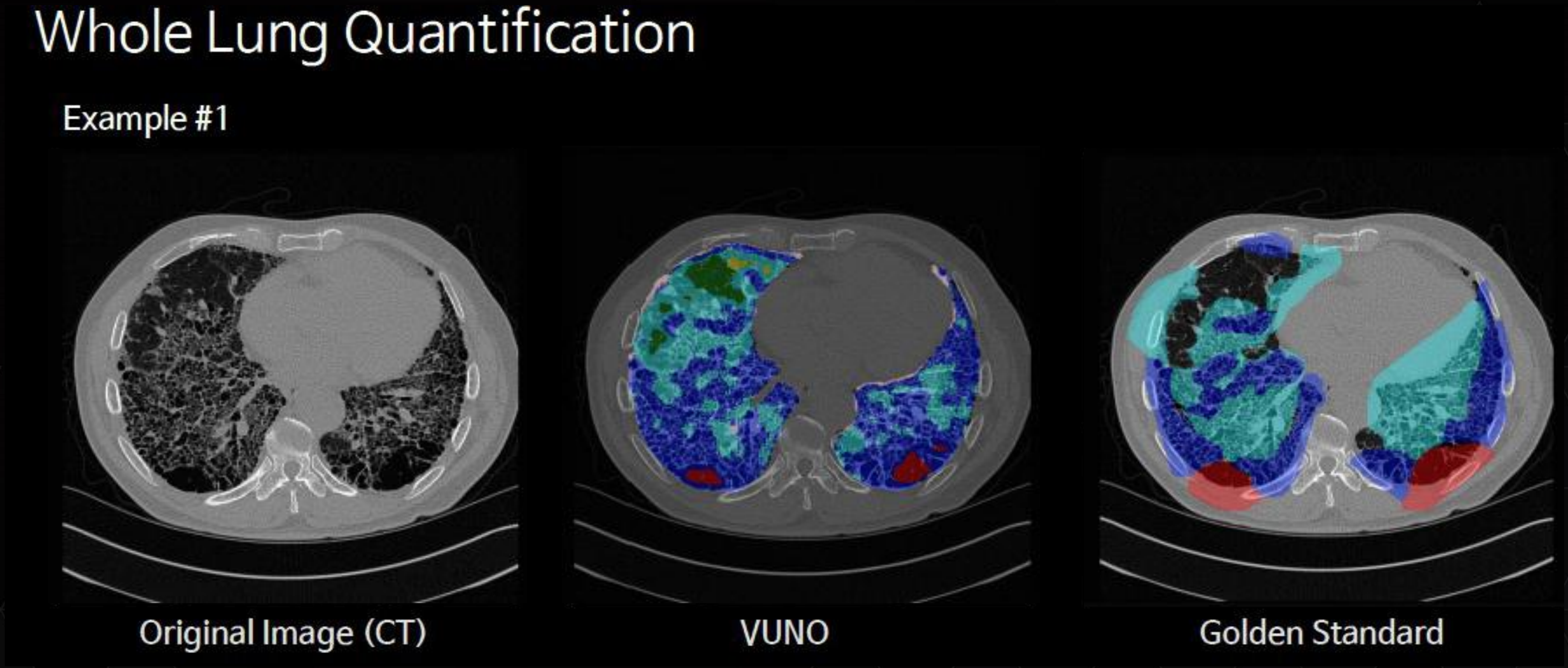
normal v.s. Emphysema

- Ongoing works

MD-LSTM

Recurrent CNN

APPLICATION



Whole Lung Quantification

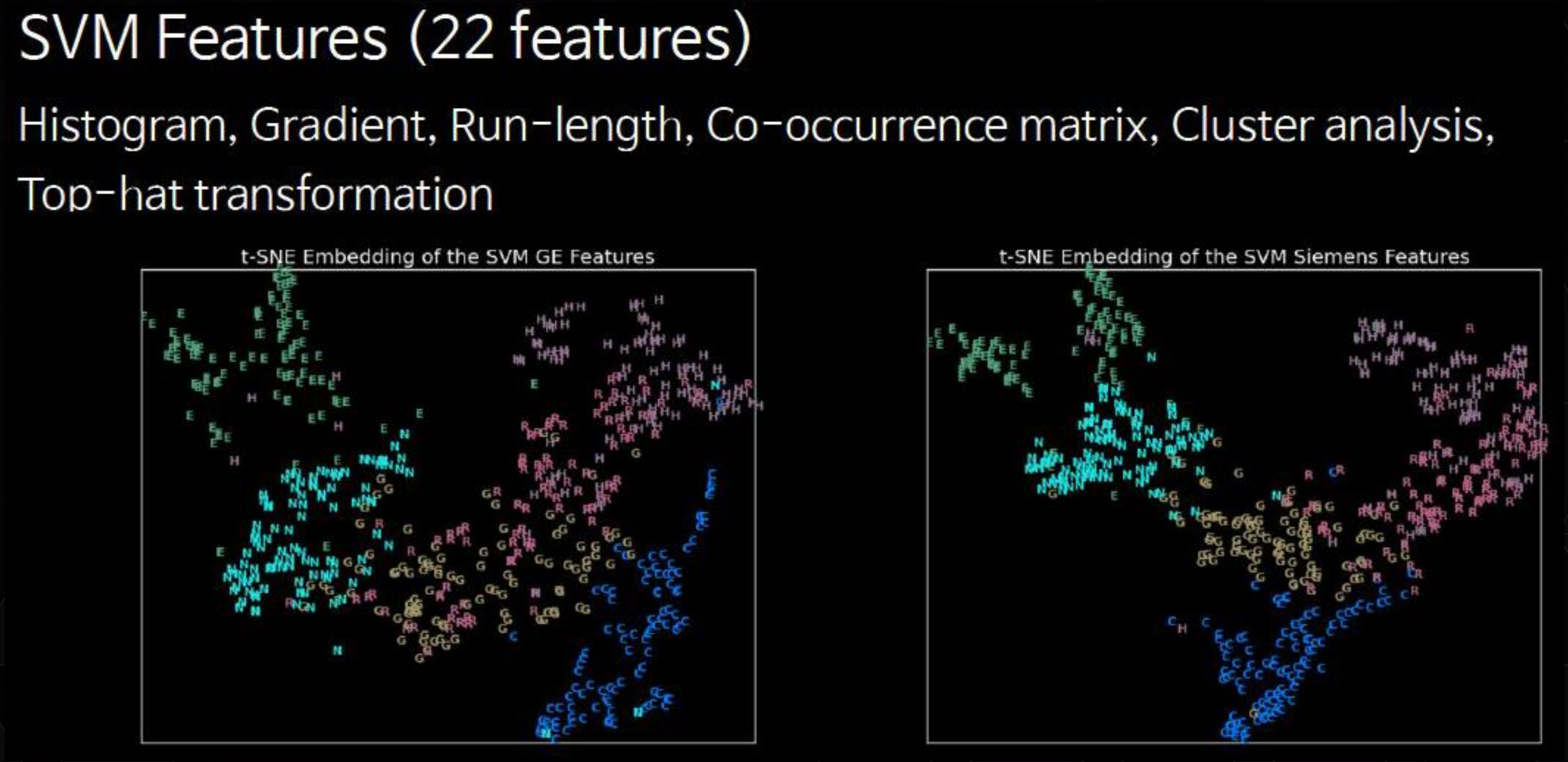
Example #1

Original Image (CT)

VUNO

Golden Standard

VISUALIZATION



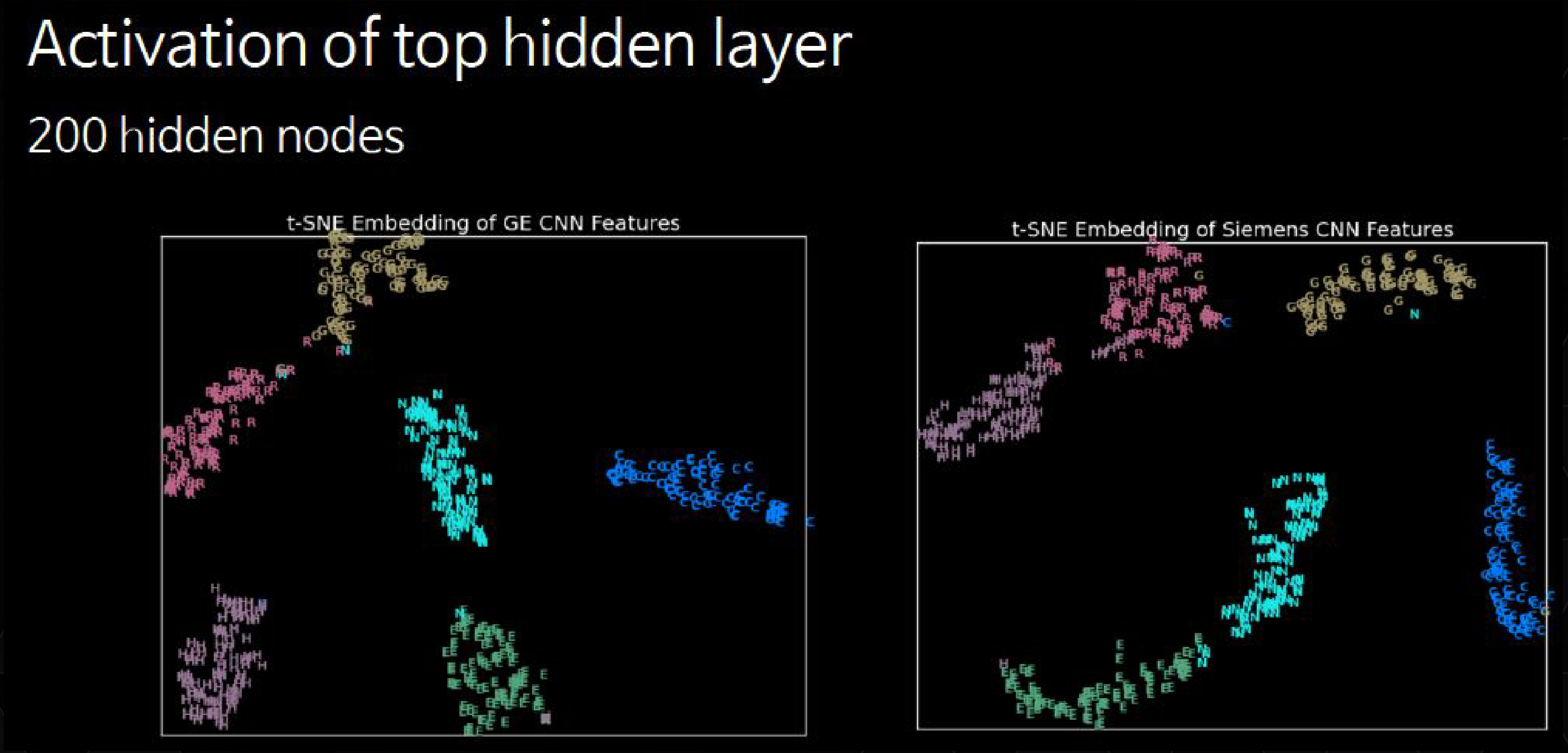
SVM Features (22 features)

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

t-SNE Embedding of the SVM GE Features

t-SNE Embedding of the SVM Siemens Features

VISUALIZATION



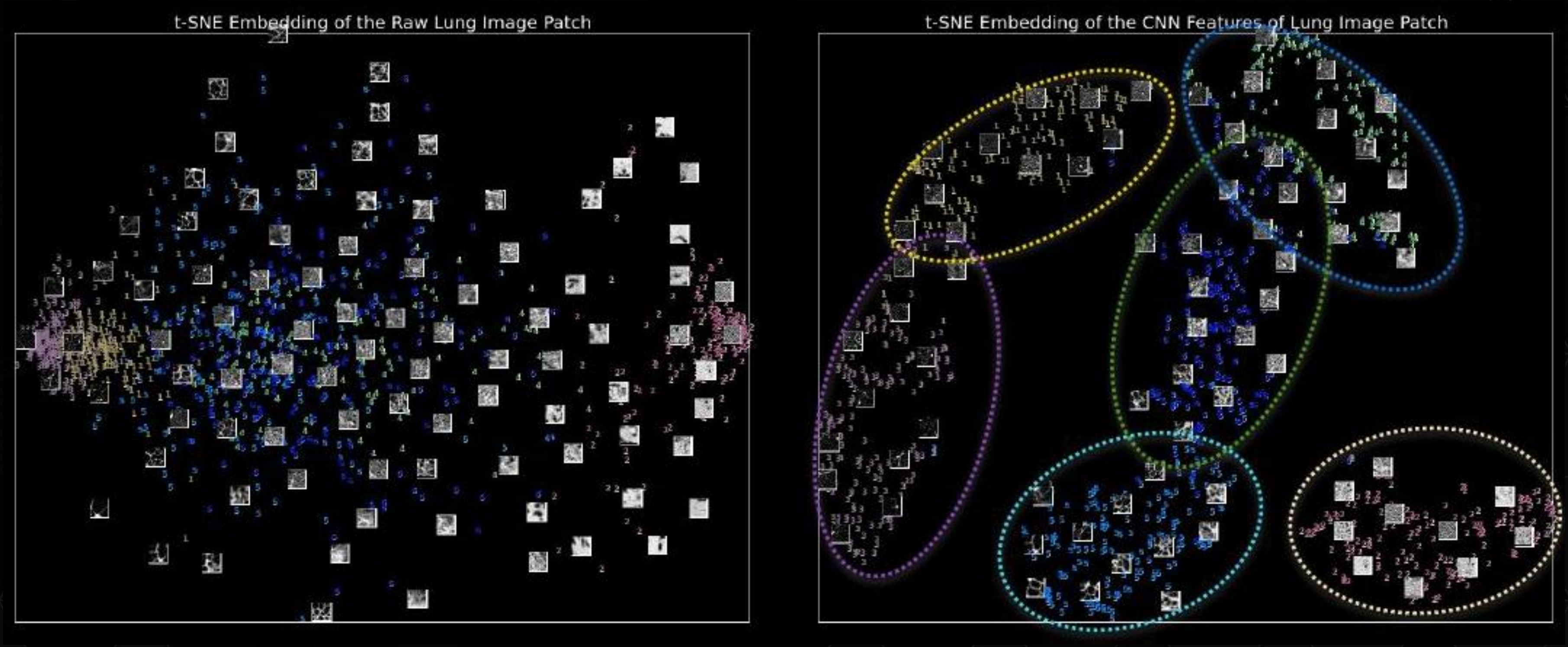
Activation of top hidden layer

200 hidden nodes

t-SNE Embedding of GE CNN Features

t-SNE Embedding of Siemens CNN Features

VISUALIZATION



t-SNE Embedding of the Raw Lung Image Patch

t-SNE Embedding of the CNN Features of Lung Image Patch

We Are Hiring!!

Algorithm Engineer

CUDA Programmer

Application Developer

Business Developer

Staff Member

GPU TECHNOLOGY CONFERENCE

THANK YOU

JOIN THE CONVERSATION

#GTC15