深蓝学院
shenlanxueyuan.com

3D Point Clouds

Lecture 5 – Deep Learning with Point Clouds

主讲人 黎嘉信

Aptiv 自动驾驶
新加坡国立大学 博士
清华大学 本科

1. Introduction to Deep Learning
2. PointNet
3. PointNet++

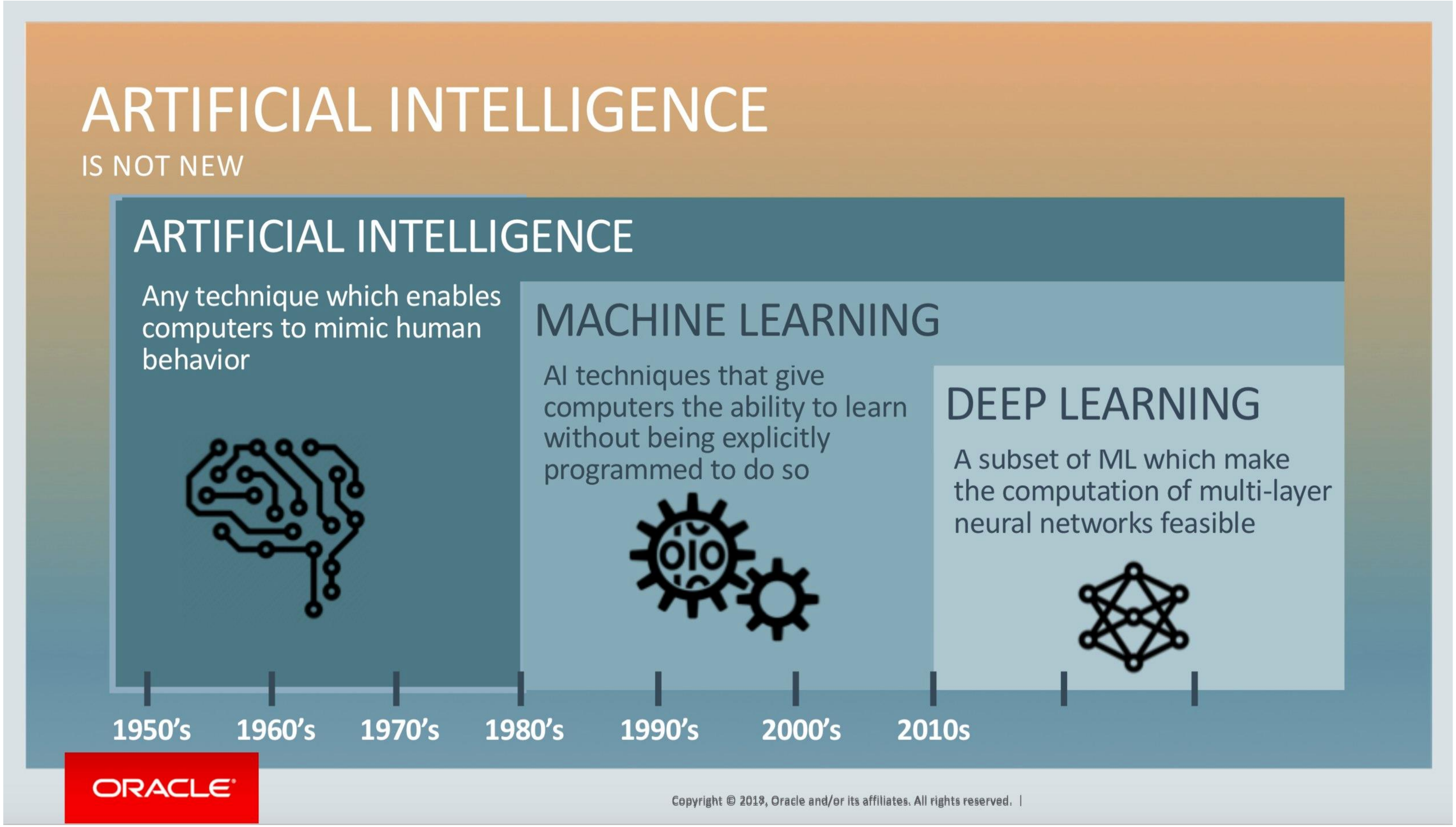
Intended Learning Outcomes

1. Explain the principles of the operations of different layers and training algorithms

2. Compare different neural network architectures

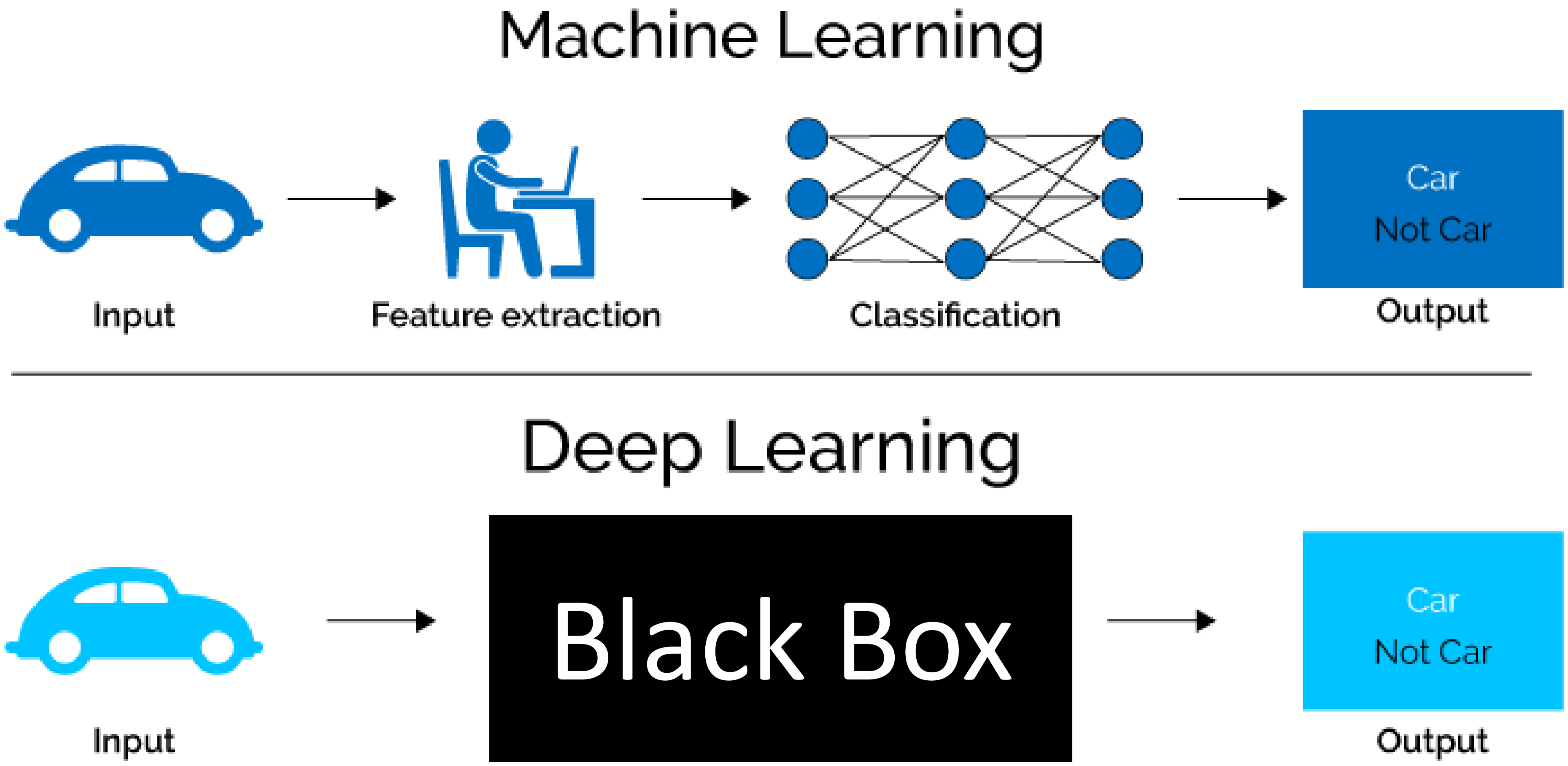
3. Implement popular neural networks

4. Solve problems using neural networks and deep learning techniques



https://blogs.oracle.com/bigdata/difference-ai-machine-learning-deep-learning

Machine Learning vs Deep Learning



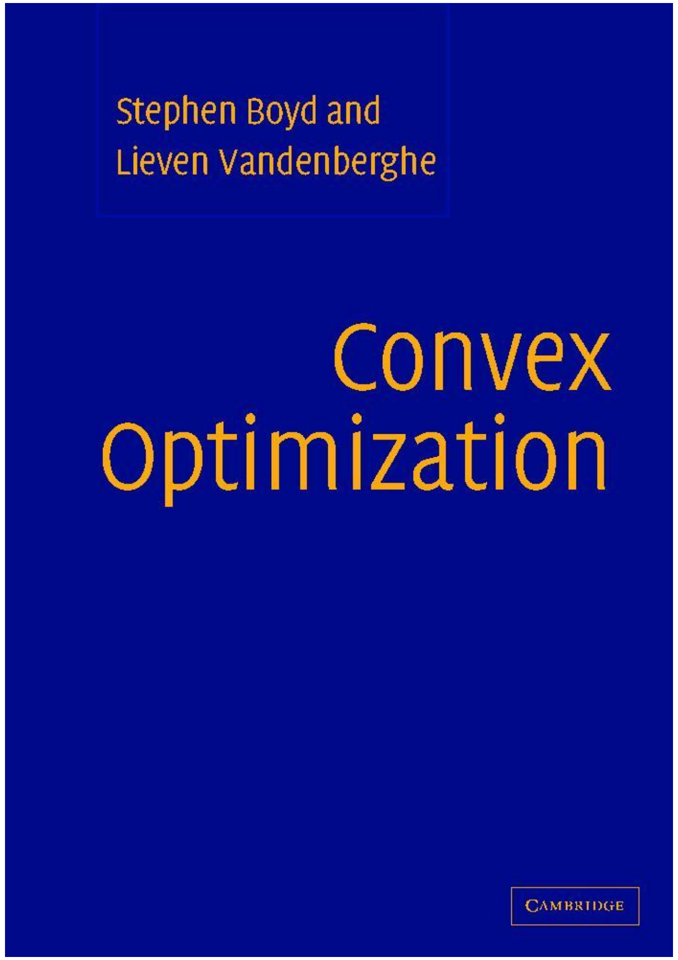
https://towardsdatascience.com/cnn-application-on-structured-data-automated-feature-extraction-8f2cd28d9a7e

- Everything is an optimization problem.

  – Stephen Boyd

- We can’t really solve most optimization problems.

- Deep Learning saves the world because it is a magical optimization solver.



Another explain of Deep Learning

- Deep learning is trying to solve one problem

$$\min_x f(x)$$

- Let's start from linear function

$$y = f(x) = w^T x + b$$

  - Given a set of $\{x_i, \in \mathbb{R}^n\ y_i \in \mathbb{R}\}$
  - Solve $w \in \mathbb{R}^n, b \in \mathbb{R}$

- Consider the problem of house price prediction
- A dataset consist of M instance
  - Features $x_i \in \mathbb{R}^n$
  - Ground truth (price) $y_i \in \mathbb{R}$

$$x_i = \begin{bmatrix} x_{i1} \\ x_{i2} \\ x_{i3} \\ \vdots \\ x_{in} \end{bmatrix} = \begin{bmatrix} size \\ floor \\ age \\ \vdots \\ *** \end{bmatrix}$$

- Unknown parameter $w \in \mathbb{R}^n, b \in \mathbb{R}$
- Denote prediction of one instance

$$\hat{y}_i = w^T x_i + b = \begin{bmatrix} w \\ b \end{bmatrix} \begin{bmatrix} x_i & 1 \end{bmatrix}$$

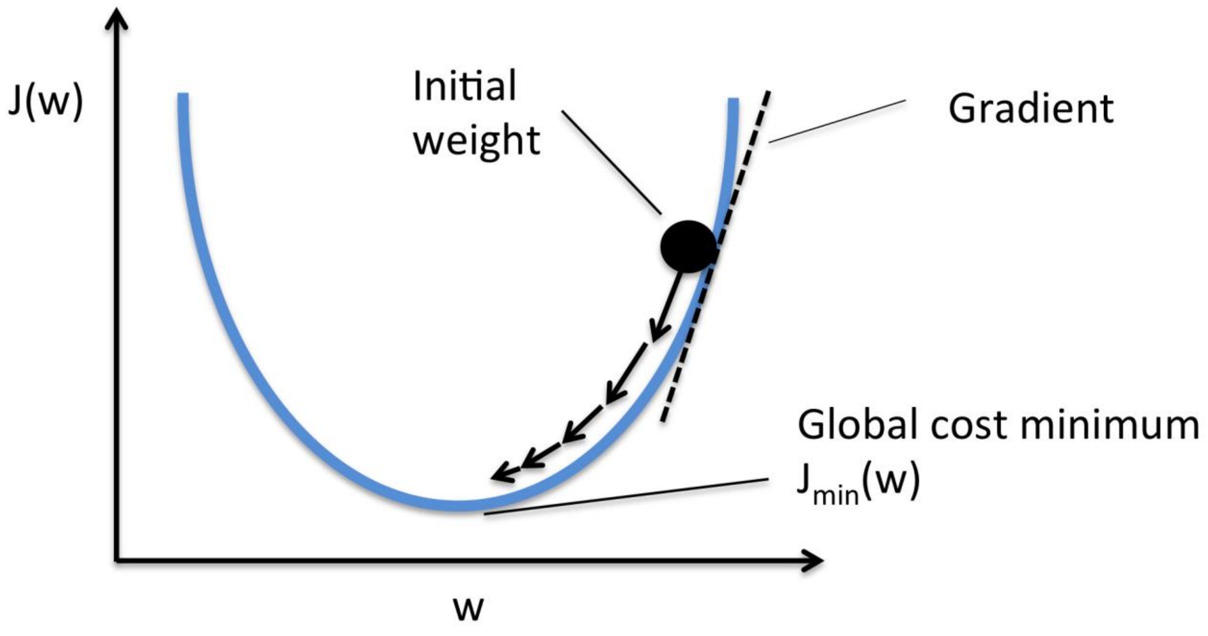
- A bit abuse of notation with *homogeneous coordinate*

$$\hat{y}_i = w^T x_i$$

- It is a linear regression problem

$$w = \arg\min_{w} \sum_{i=1}^{M} \frac{1}{2}(w^T x_i - y_i)^2$$

- Typical $Ax = b$ problem, but can we solve it by *Gradient Descent*?



Gradient Descent

- For any differentiable function $F(x): \mathbb{R}^n \to \mathbb{R}^m$
- The objective is to minimize $F(x)$

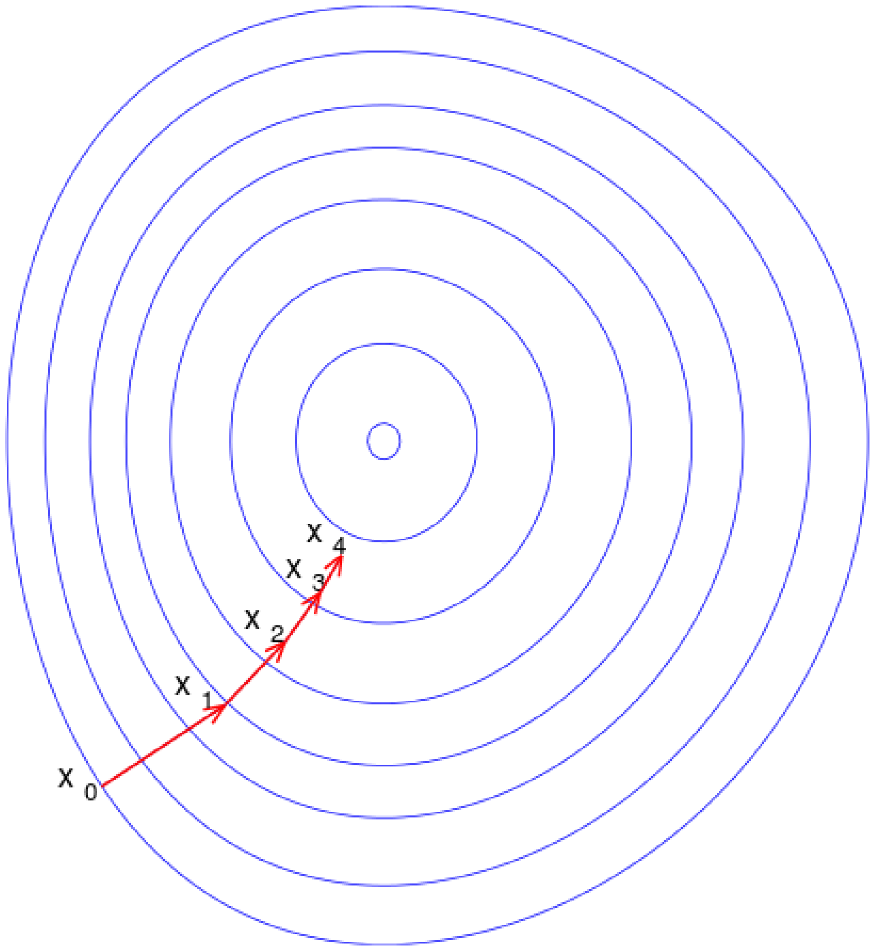
$$x^* = \underset{x}{\operatorname{argmin}}\, F(x)$$

- At position x_n, $F(x_{n+1}) \leq F(x_n)$ is guaranteed if,

$$x_{n+1} = x_n - \gamma \nabla F(x_n)$$

where $\nabla F(x_n)$ is the gradient of F at x_n

- γ is the *step size*



- Objective

$$w = \arg\min_{w} \sum_{i=1}^{M} \frac{1}{2}(w^T x_i - y_i)^2$$

- Consider a single instance

$$J(w) = \frac{1}{2}(w^T x - y)^2 = \frac{1}{2} z^2, \qquad z = w^T x - y$$

- Gradient descent requires

$$\frac{\partial J(w)}{\partial w} = \frac{\partial J}{\partial z} \frac{\partial z}{\partial w} = zx = (w^T x - y)x$$

where $J(w) \in \mathbb{R}, z \in \mathbb{R}, w \in \mathbb{R}^n$

Univariate Derivative

- Constant Rule $f(x) = c \;\rightarrow\; f'(x) = 0$
- Constant Multiple Rule $f(x) = c \cdot g(x) \;\rightarrow\; f'(x) = cg'(x)$
- Power Rule $f(x) = x^n \;\rightarrow\; f'(x) = nx^{n-1}$
- Sum Rule $f(x) = g(x) + h(x) \;\rightarrow\; f'(x) = g'(x) + h'(x)$
- Product Rule $f(x) = g(x)h(x) \;\rightarrow\; f'(x) = g'(x)h(x) + g(x)h'(x)$
- Quotient Rule $f(x) = \frac{g(x)}{h(x)} \;\rightarrow\; f'(x) = \frac{g'(x)h(x) - g(x)h'(x)}{h(x)^2}$
- Chain Rule $f(x) = g(u), u = h(x) \;\rightarrow\; f'(x) = g'(u)h'(x)$

- Given a multivariable function $f(x, y)$, and two univariate functions $x(t), y(t)$, the multivariable chain rule is:

$$\frac{df\big(x(t), y(t)\big)}{dt} = \frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial y}\frac{dy}{dt}$$

- How to compute $\frac{\partial f(x)}{\partial x}$ if $f(x)$ and/or x is scalar, vector, matrix?

$$\mathbf{x} \in \mathbb{R}^n, \mathbf{y} \in \mathbb{R}^m, \mathbf{X} \in \mathbb{R}^{n\times m}, \mathbf{Y} \in \mathbb{R}^{m\times n}$$

what are $\frac{\partial \mathbf{y}}{\partial x}, \frac{\partial y}{\partial \mathbf{x}}, \frac{\partial \mathbf{y}}{\partial \mathbf{x}}, \frac{\partial y}{\partial \mathbf{X}}, \frac{\partial \mathbf{Y}}{\partial x}$

Types	Scalar	Vector	Matrix
Scalar	$\frac{\partial y}{\partial x}$	$\frac{\partial \mathbf{y}}{\partial x}$	$\frac{\partial \mathbf{Y}}{\partial x}$
Vector	$\frac{\partial y}{\partial \mathbf{x}}$	$\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$	
Matrix	$\frac{\partial y}{\partial \mathbf{X}}$		

Denominator layout

$$\frac{\partial \mathbf{y}}{\partial x} \in \mathbb{R}^{1\times m}, \frac{\partial y}{\partial \mathbf{x}} = \mathbb{R}^{\boldsymbol{n}}$$

Numerator layout

$$\frac{\partial \mathbf{y}}{\partial x} \in \mathbb{R}^{m}, \frac{\partial y}{\partial \mathbf{x}} = \mathbb{R}^{\mathbf{1}\times \boldsymbol{n}}$$

$$\mathbf{x} \in \mathbb{R}^n, \mathbf{y} \in \mathbb{R}^m, \mathbf{X} \in \mathbb{R}^{n\times m}, \mathbf{Y} \in \mathbb{R}^{m\times n}$$

$$\frac{\partial y}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \\ \vdots \\ \frac{\partial y}{\partial x_n} \end{bmatrix} \qquad \frac{\partial \mathbf{y}}{\partial x} = \begin{bmatrix} \frac{\partial y_1}{\partial x} & \frac{\partial y_2}{\partial x} & \cdots & \frac{\partial y_m}{\partial x} \end{bmatrix} \qquad \frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_2}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_1} \\ \frac{\partial y_1}{\partial x_2} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_1}{\partial x_n} & \frac{\partial y_2}{\partial x_n} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}$$

$$\frac{\partial y}{\partial \mathbf{X}} = \begin{bmatrix} \frac{\partial y}{\partial x_{11}} & \frac{\partial y}{\partial x_{12}} & \cdots & \frac{\partial y}{\partial x_{1m}} \\ \frac{\partial y}{\partial x_{21}} & \frac{\partial y}{\partial x_{22}} & \cdots & \frac{\partial y}{\partial x_{2m}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y}{\partial x_{n1}} & \frac{\partial y}{\partial x_{n2}} & \cdots & \frac{\partial y}{\partial x_{nm}} \end{bmatrix} \qquad \frac{\partial \mathbf{Y}}{\partial x} = \begin{bmatrix} \frac{\partial y_{11}}{\partial x} & \frac{\partial y_{21}}{\partial x} & \cdots & \frac{\partial y_{m1}}{\partial x} \\ \frac{\partial y_{12}}{\partial x} & \frac{\partial y_{22}}{\partial x} & \cdots & \frac{\partial y_{m2}}{\partial x} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_{1n}}{\partial x} & \frac{\partial y_{2n}}{\partial x} & \cdots & \frac{\partial y_{mn}}{\partial x} \end{bmatrix}$$

$\mathbf{y} =$	$\mathbf{a}$	$\mathbf{x}$	$\boldsymbol{A}\mathbf{x}$	$\mathbf{x}^T\boldsymbol{A}$	
$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} =$	$\mathbf{0}$	$\boldsymbol{I}$	$\boldsymbol{A}^{\mathrm{T}}$	$\boldsymbol{A}$	
$\mathbf{y} =$	$a\mathbf{u}$ $\mathbf{u} = \mathbf{u}(\mathbf{x})$	$v\mathbf{u}$ $v = v(\mathbf{x}), \mathbf{u} = \mathbf{u}(\mathbf{x})$	$\mathbf{v} + \mathbf{u}$ $\mathbf{v} = \mathbf{v}(\mathbf{x}), \mathbf{u} = \mathbf{u}(\mathbf{x})$	$\boldsymbol{A}\mathbf{u}$ $\mathbf{u} = \mathbf{u}(\mathbf{x})$	$g(\mathbf{u})$ $\mathbf{u} = \mathbf{u}(\mathrm{x})$
$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} =$	$a\frac{\partial \mathbf{u}}{\partial \mathbf{x}}$	$v\frac{\partial \mathbf{u}}{\partial \mathbf{x}} + \frac{\partial v}{\partial \mathbf{x}}\mathbf{u}^T$	$\frac{\partial \mathbf{u}}{\partial \mathbf{x}} + \frac{\partial \mathbf{v}}{\partial \mathbf{x}}$	$\frac{\partial \mathbf{u}}{\partial \mathbf{x}}\boldsymbol{A}^T$	$\frac{\partial g(\mathbf{u})}{\partial \mathbf{u}}\frac{\partial \mathbf{u}}{\partial \mathbf{x}}$
$y =$	a	$\mathbf{u}^{\mathrm{T}}\mathbf{v}$ $\mathbf{v} = \mathbf{v}(\boldsymbol{x}), \mathbf{u} = \mathbf{u}(\boldsymbol{x})$	$g(u)$ $u = u(\mathbf{x})$		
$\frac{\partial y}{\partial \mathbf{x}} =$	$\mathbf{0}$	$\frac{\partial \mathbf{u}}{\partial \mathbf{x}}\mathbf{v} + \frac{\partial \mathbf{v}}{\partial \mathbf{x}}\mathbf{u}$	$\frac{\partial g(u)}{\partial u}\frac{\partial u}{\partial \mathbf{x}}$		

Multivariate Linear Regression

- Objective

$$w = \arg\min_{w} \sum_{i=1}^{M} \frac{1}{2}(w^T x_i - y_i)^2$$

- Consider a single instance

$$J(w) = \frac{1}{2}(w^T x - y)^2 = \frac{1}{2} z^2, \qquad z = w^T x - y$$

- Gradient descent requires

$$\frac{\partial J(w)}{\partial w} = \frac{\partial J}{\partial z}\frac{\partial z}{\partial w} = zx = (w^T x - y)x$$

where $J(w) \in \mathbb{R}, z \in \mathbb{R}, w \in \mathbb{R}^n$

- Update w

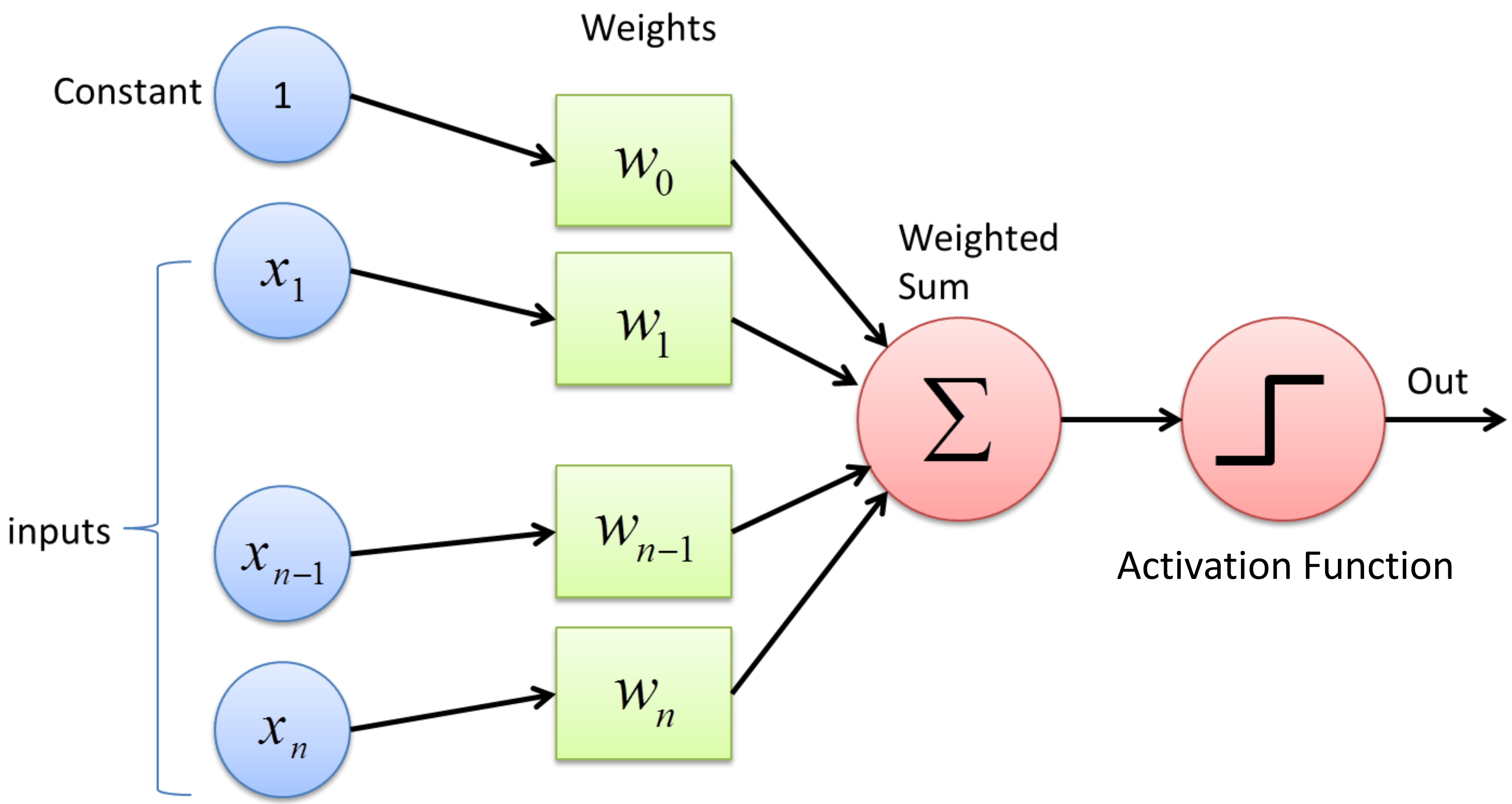
$$w_{n+1} = w_n - \lambda \frac{\partial J(w)}{\partial w}\bigg|_{w=w_n} = w_n - \lambda(w_n^T x - y)x$$

- What we solve just now, is

$$w = \arg\min_{w} \sum_{i=1}^{M} \frac{1}{2}(w^T x_i - y_i)^2$$

- The is the very basic Neural Network – Perceptron
  - w is the trainable parameter
  - x_i is the data point, y_i is the label
  - Invented in 1968
  - Frank Rosenblatt, Cornel Aeronautical Laboratory

Perceptron



The most simple Neural Network (NN): A Perceptron: $y = w^T x$

Train a Perceptron

Given several examples

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

and a perceptron

$$y = w^T x$$

Modify weight w such that $\hat{y}$ gets **'closer'** to y

perceptron parameter

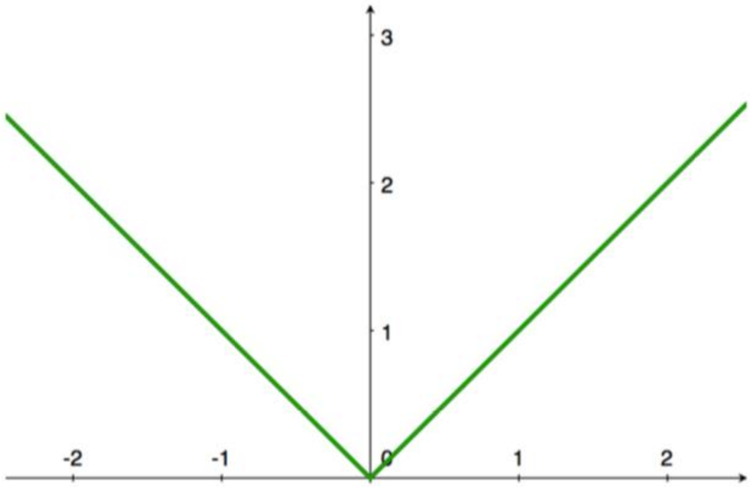
perceptron output

what does this mean?

true label

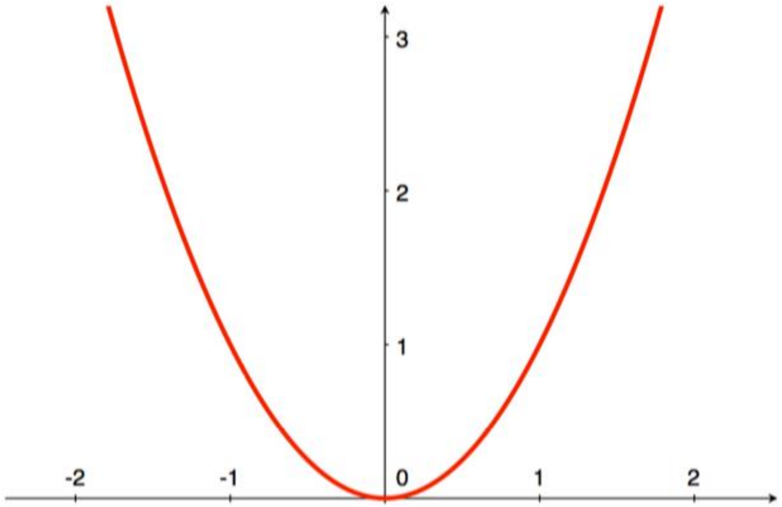
L1 Loss

$$\ell(\hat{y}, y) = |\hat{y} - y|$$

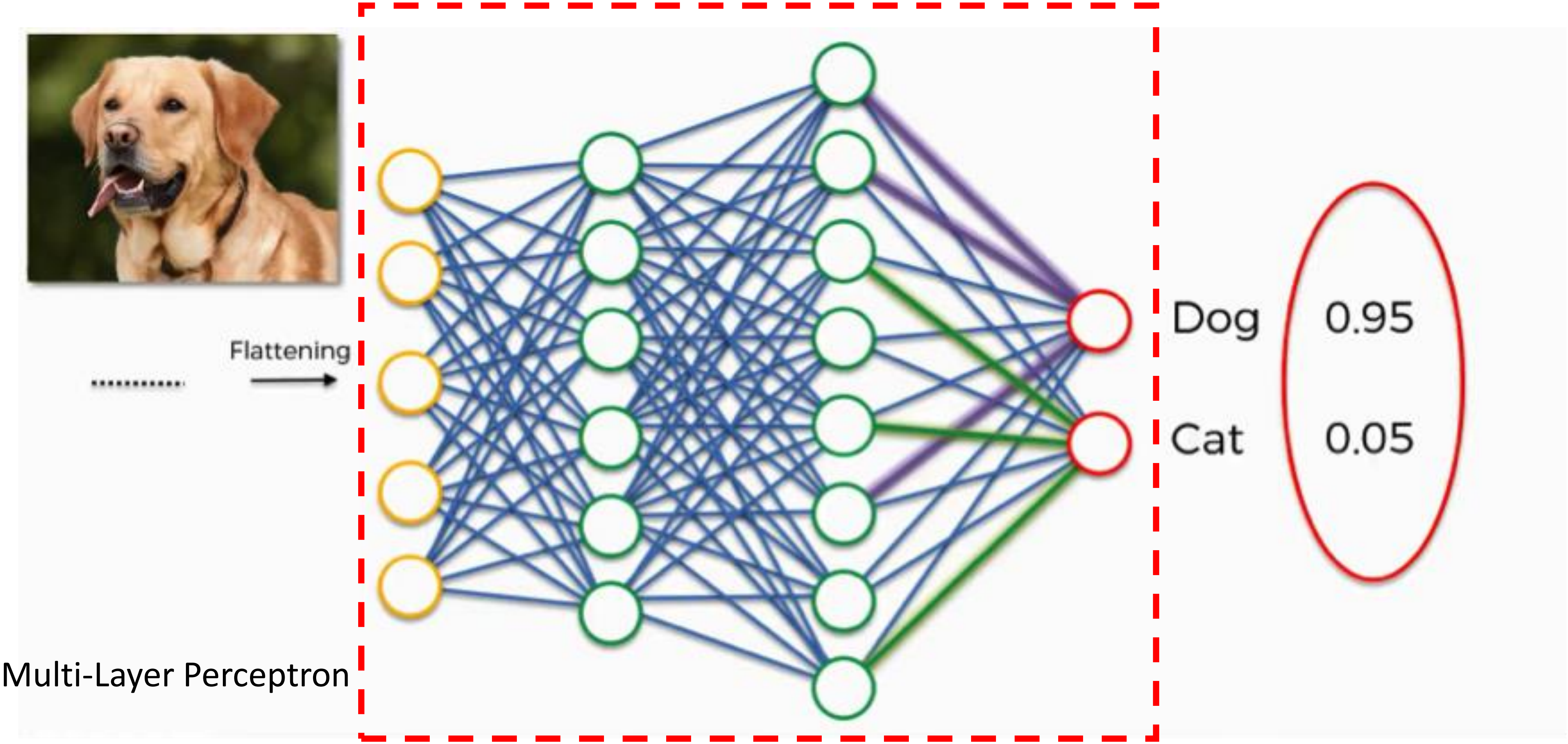


L2 Loss

$$\ell(\hat{y}, y) = (\hat{y} - y)^2$$



Loss – Classification – Cross Entropy Loss



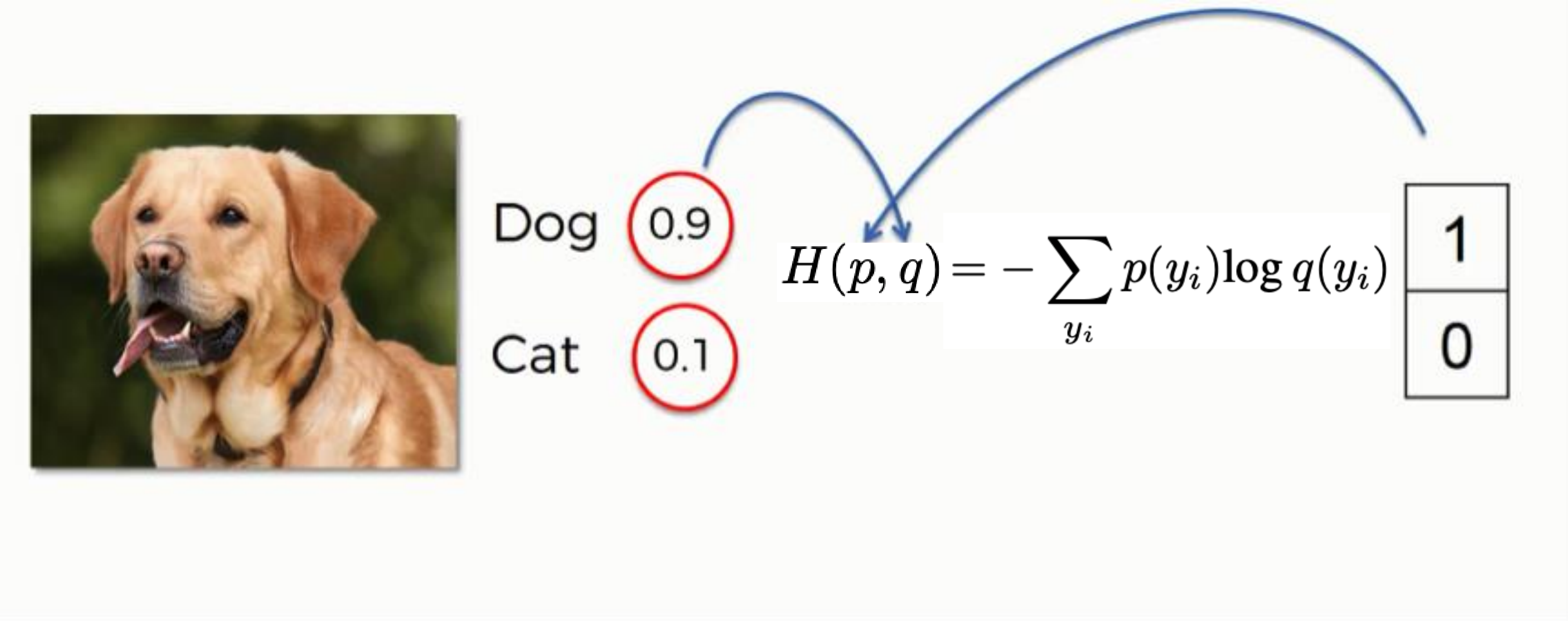
Multi-Layer Perceptron

Image source: https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-softmax-crossentropy

- Cross Entropy Loss

$$H(p,q) = -\sum_{y_i} p(y_i)\log q(y_i)\,, \qquad \sum_{y_i} p(y_i) = 1\,, \sum_{y_i} q(y_i) = 1$$

- $p(y_i = 1)$ is the ground-truth probability of category i
- $q(y_i = 1)$ is the predicted probability of category i



Cross Entropy Loss → Negative Log Softmax

$$H(p,q) = -\sum_{y_i} p(y_i) \log q(y_i) \Leftrightarrow L = H(p,q) = -\log \frac{e^{s_{i^*}}}{\sum_j e^{s_j}}, \text{ where } p(y_{i^*}) = 1$$

Unnormalized probabilities

dog	3.2		24.5		0.13
cat	5.1	exp	164.0	normalize	0.87
frog	-1.7		0.18		0.00

Network output **score**:
Unnormalized log probabilities

Probabilities

$$L = -\log \frac{e^{3.2}}{e^{3.2} + e^{5.1} + e^{-1.7}}$$
$$= -\log \frac{24.5}{24.5 + 164.0 + 0.18}$$
$$= -\log 0.13$$
$$= 0.89$$

Multi-Layer Perceptron (MLP)

- How many Perceptrons?

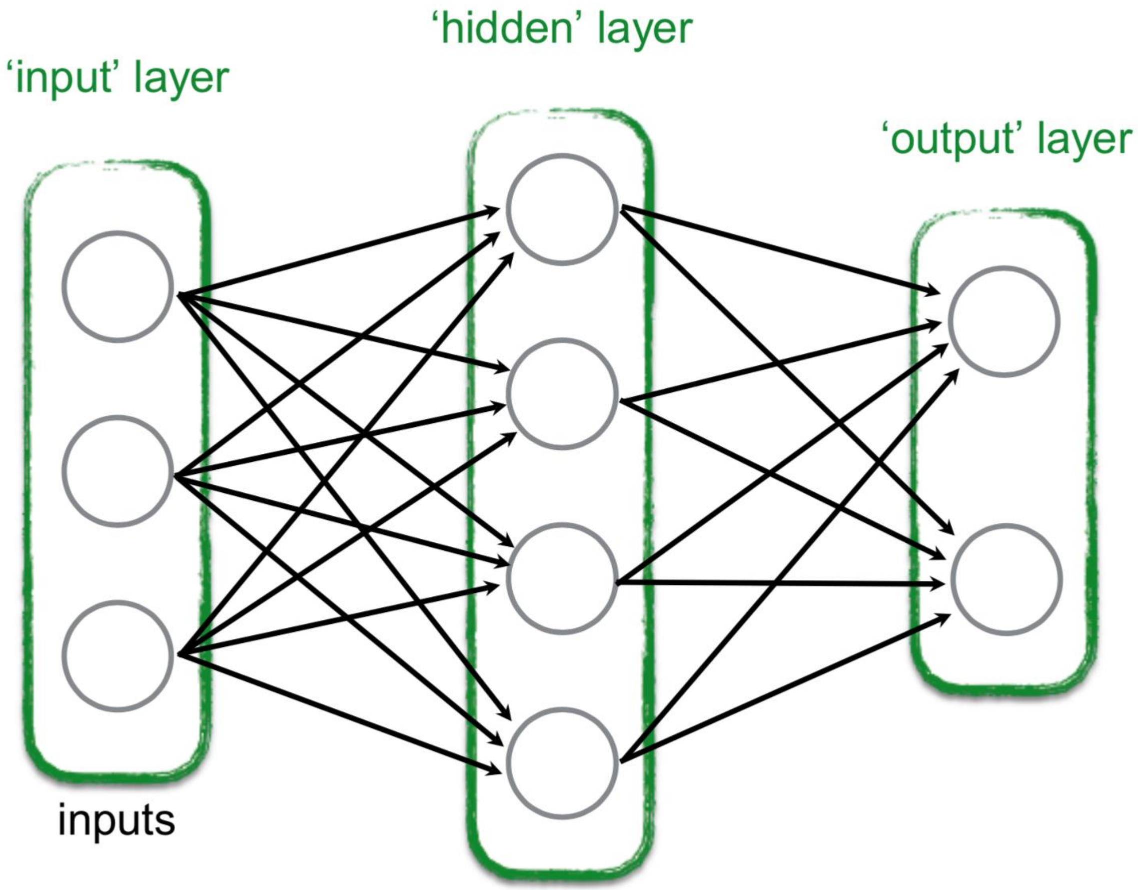
$$4 + 2 = 6$$

- How many trainable parameters?

$$3 \times 4 + 4 \times 2 = 20$$

- Output $y = w_2^T(w_1^T x) = w'^T x$

- It is linear no matter how many layers are in the MLP!



$$XOR(0,0) = XOR(1,1) = 0$$
$$XOR(0,1) = XOR(1,0) = 1$$

Train a linear MLP $y = w^T[x_1, x_2]$ so that:

$$w^T[0,0] = w^T[1,1] < 0$$
$$w^T[0,1] = w^T[1,0] > 0$$

NOT Possible, because $w^T x = 0$ is a hyperplane

How? Make it non-linear

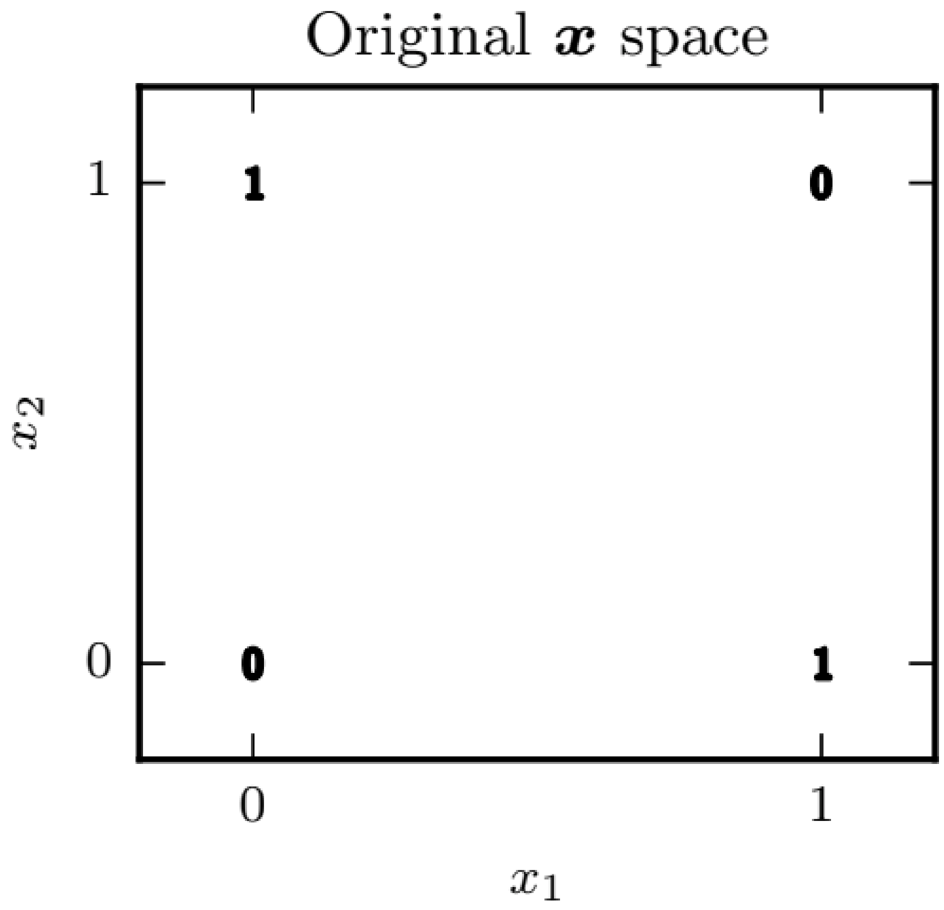


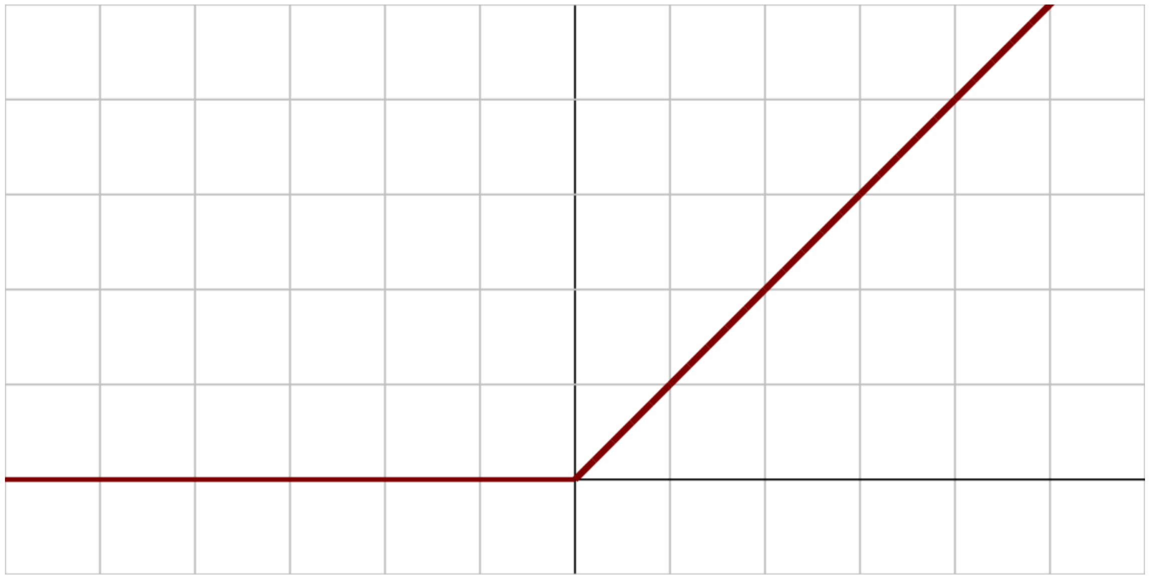
Image source: Deep Learning, Goodfellow, et. al. 2017

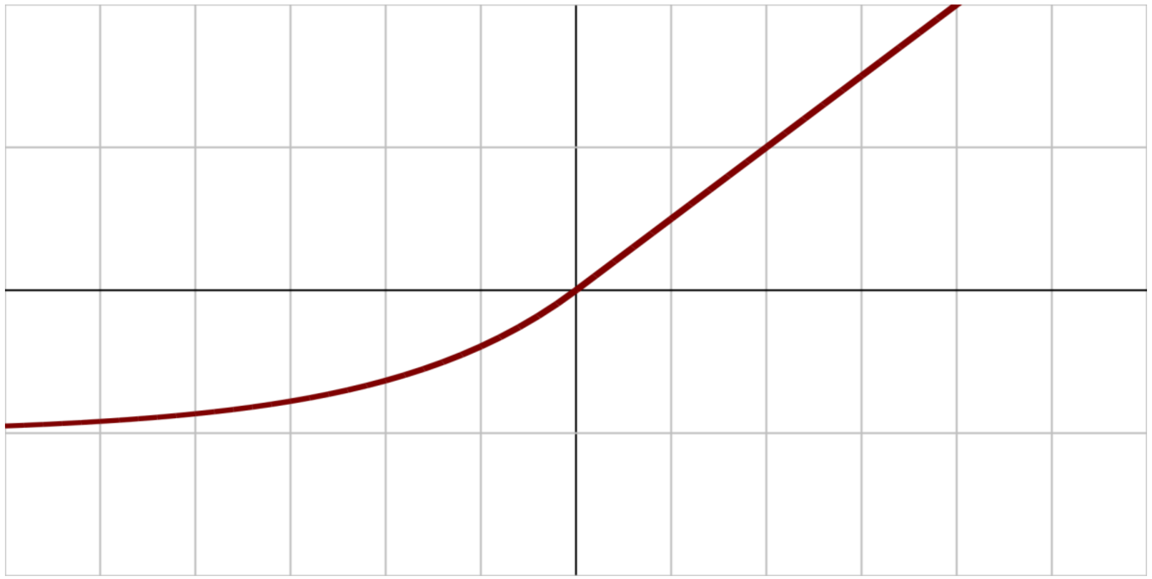
MLP – Activation Function

- Add non-linearity by activation function $g(\cdot)$
- A perception becomes $y = g\big(f(x)\big) = g(w^T x)$

Rectified Linear Unit (ReLU) $f(x) = \begin{cases} 0, & for\ x \leq 0 \\ x, & for\ x > 0 \end{cases}$

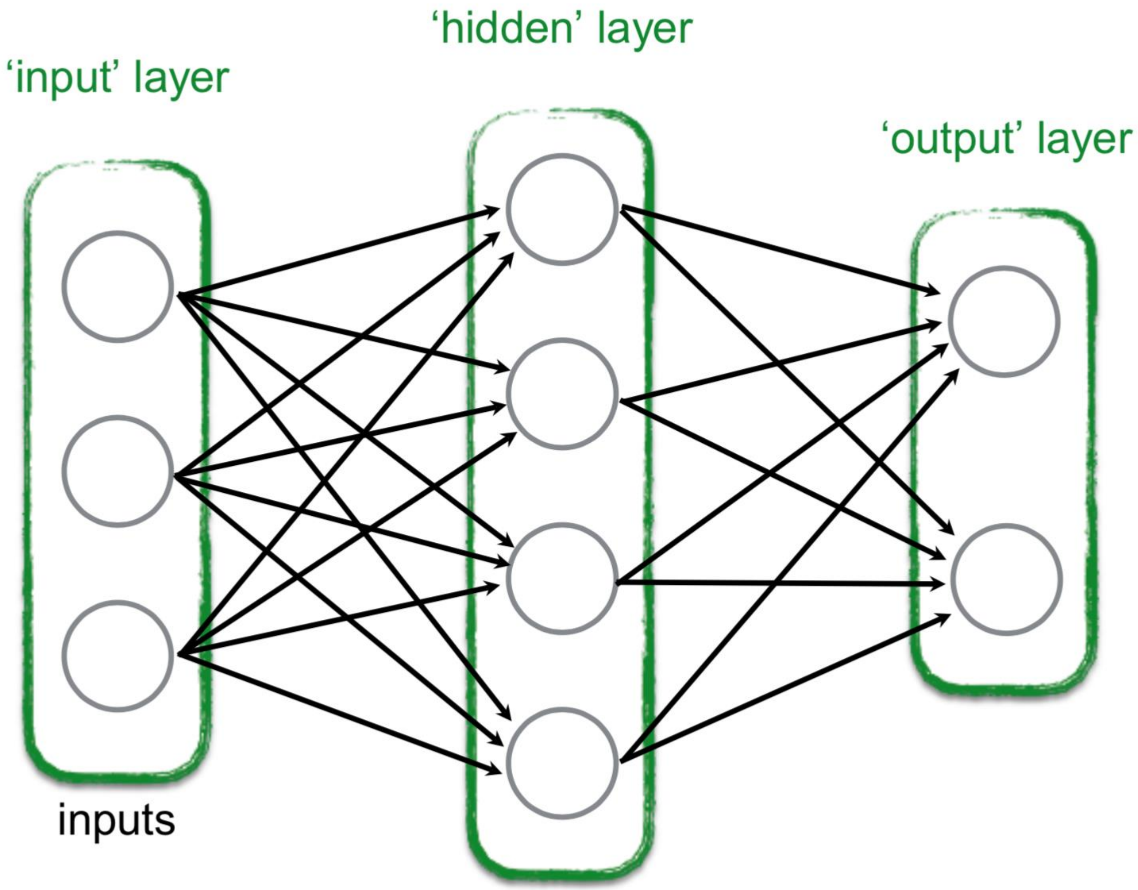
Exponential Linear Unit (ELU) $f(x) = \begin{cases} \alpha(e^x) - 1, & for\ x \leq 0 \\ x, & for\ x > 0 \end{cases}$





Multi-Layer Perceptron (MLP)
Fully Connected Network (FC)

- A MLP with activation & ≥ 1 hidden layers
- Is able to simulate ANY function $f(x)$, where x is the input
- There is proof for this, not shown in this lecture.



How to Train a MLP?

- Same, gradient descent.

$$w_{n+1} = w_n - \lambda \frac{\partial L}{\partial w}\bigg|_{w=w_n}, \forall w$$

- What is Back Propagation?
- An efficient way of gradient descent.

- What are optimizers like SGD (Stochastic Gradient Descent), Adam?
- Improvements over gradient descent, still gradient descent.

Gradient Descent Example

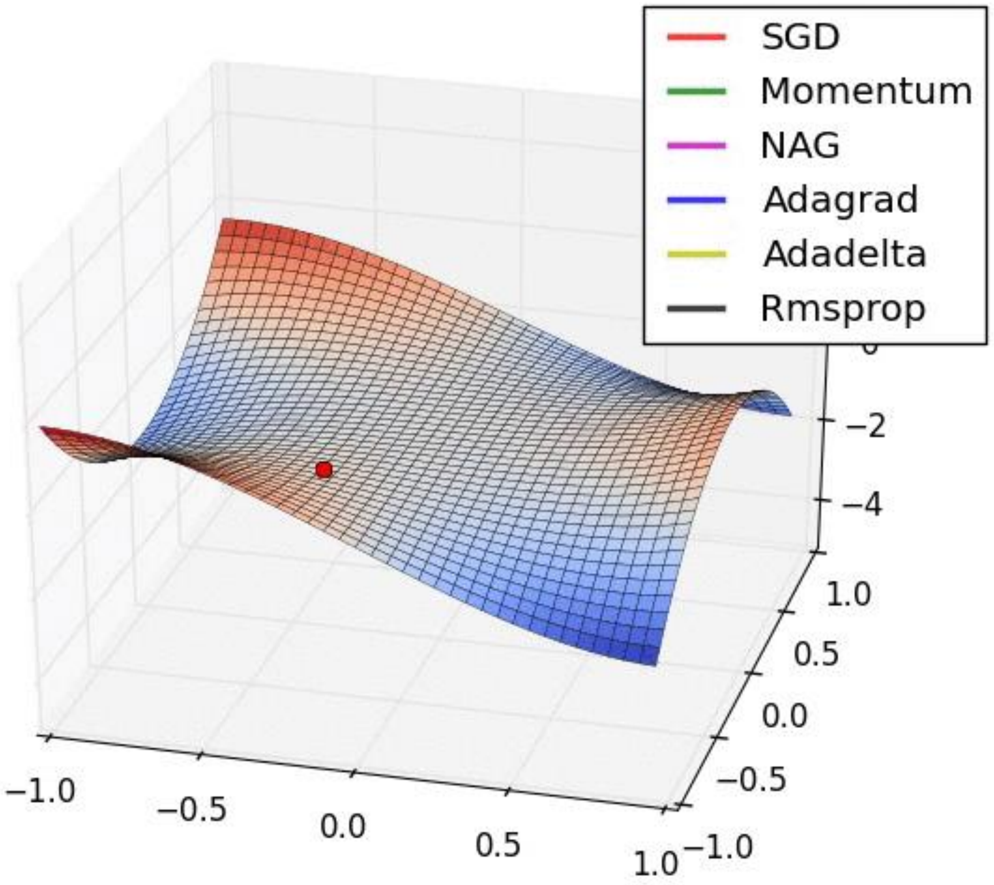


Image: Alec Radford

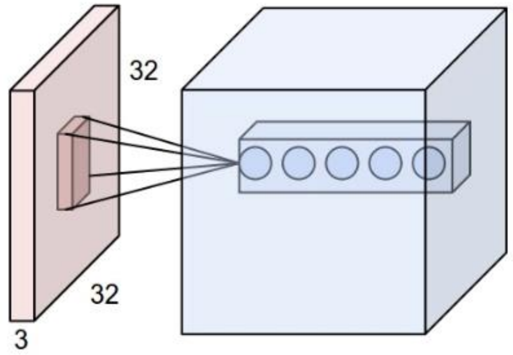
What is CNN?

- Apply convolution on 1D/2D/3D

Why CNN?

- Features can be extracted in a local neighborhood.

(We will give detailed answers later, after showing what is CNN)



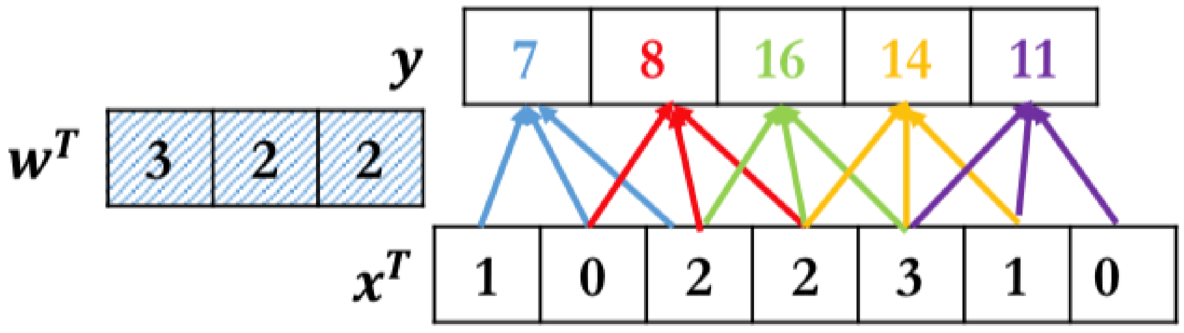
Input image

Convolution Kernel

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

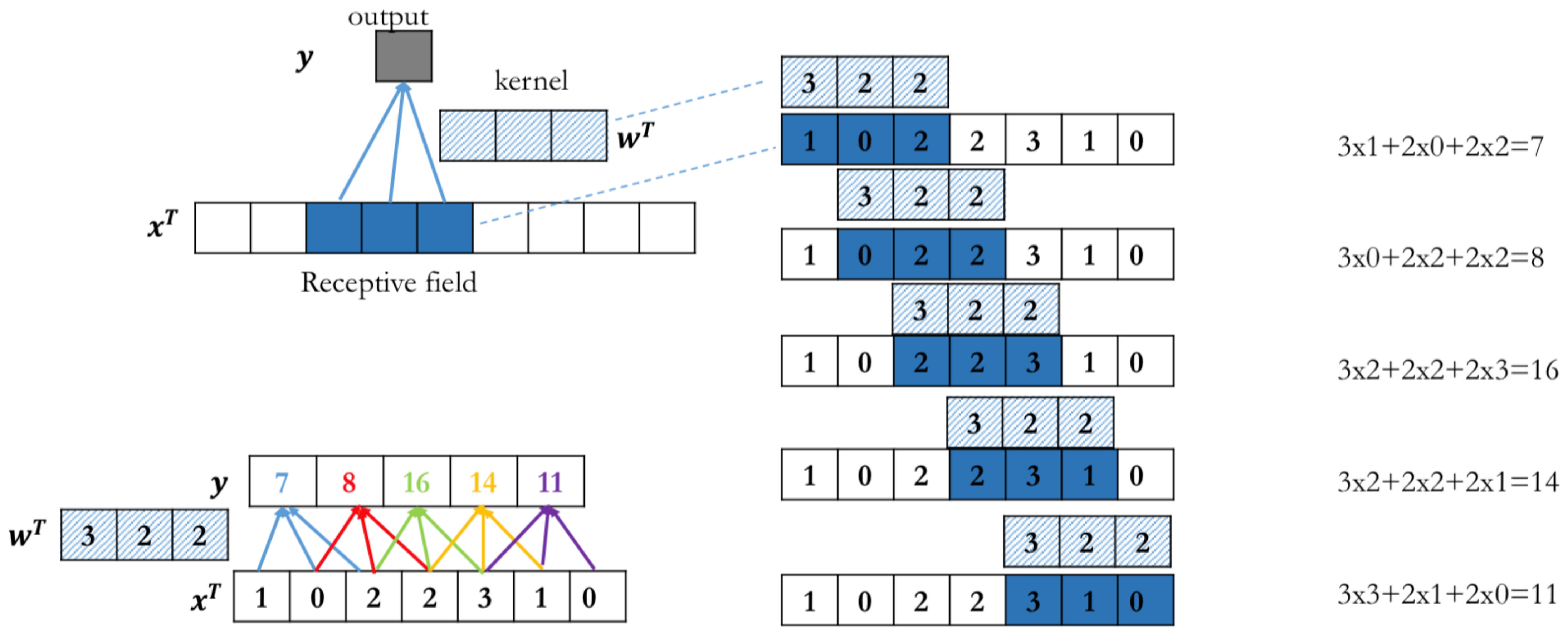
Feature map

$$y_t = \sum_{i=0}^{k-1} w_i \cdot x_{t+i}$$



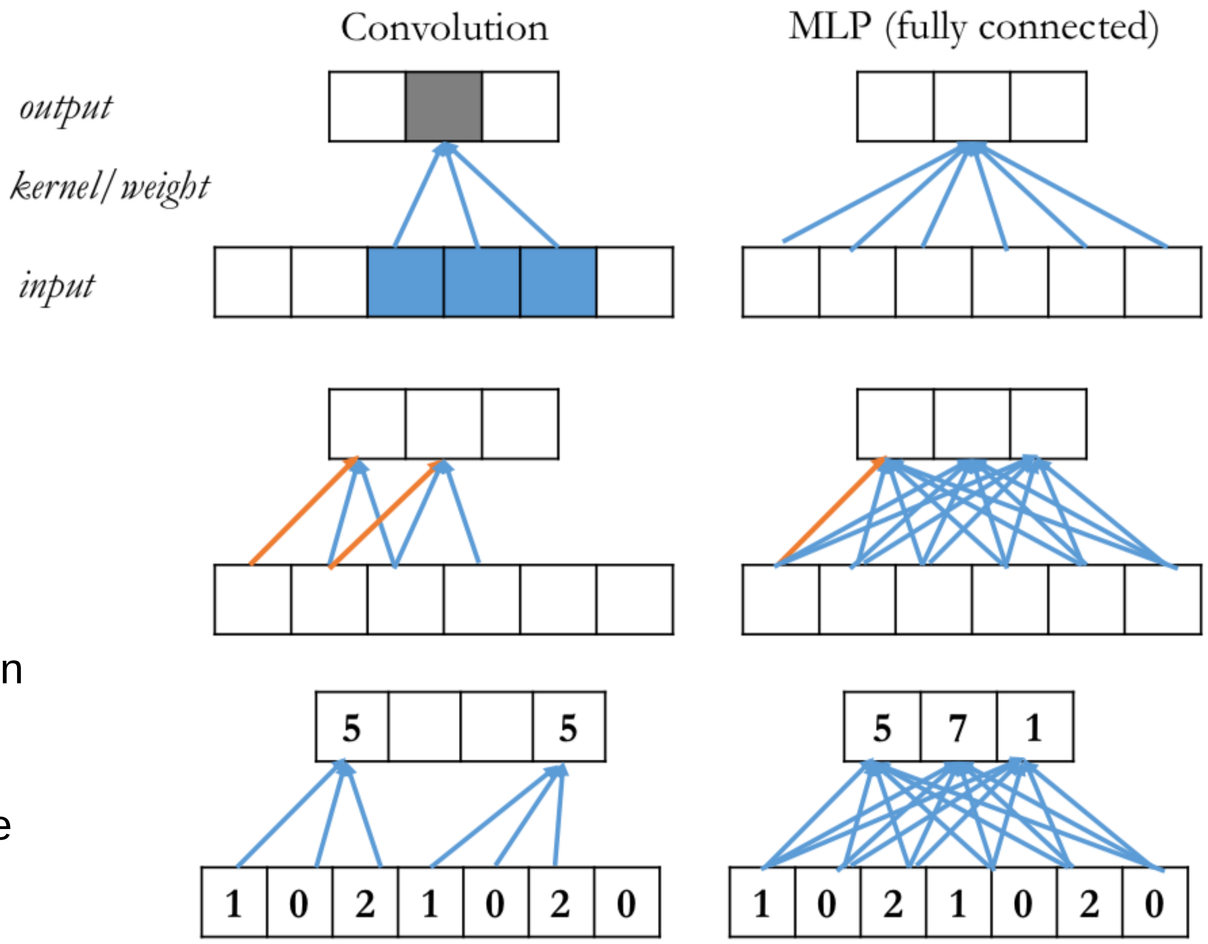
- w is the kernel / filter
  - Length k
  - They are the unknown/trainable parameters
- x is the input
  - Length n
  - Receptive field: $[x_t, x_{t+1}, \cdots, x_{t+k-1}]$
  - Each receptive field generates one output value y_t
- y is the output
  - Length o

1D Convolution



Why CNN ?

1. Sparse connection vs. Dense
   - Fewer parameters. 3 vs. 18
   - Less overfitting
2. Weight sharing vs. Unique weights
   - Less overfitting
3. Local invariant vs. Local variant
   - Features should not depend on the location within the image
   - Make the same prediction no matter where is the object in the image



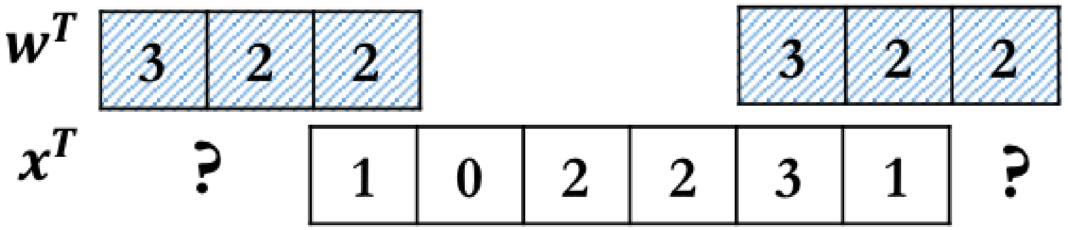
- How to determine edge values?
- Padding!

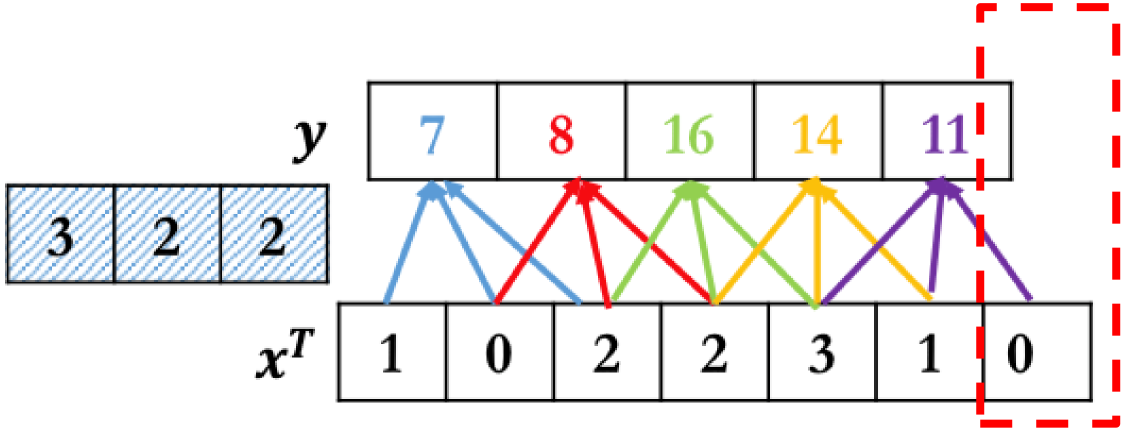
- Given padding p, what is the output length o?
  - kernel size k
  - Input length n
- Without padding:

$$o = n - k + 1$$

- With padding:

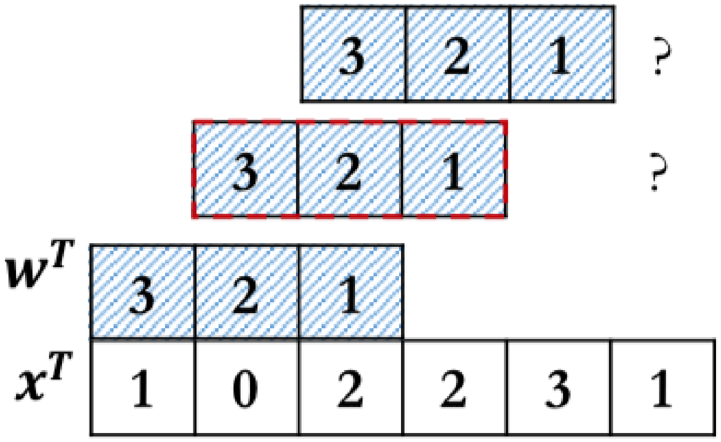
$$o = (n + p) - k + 1$$

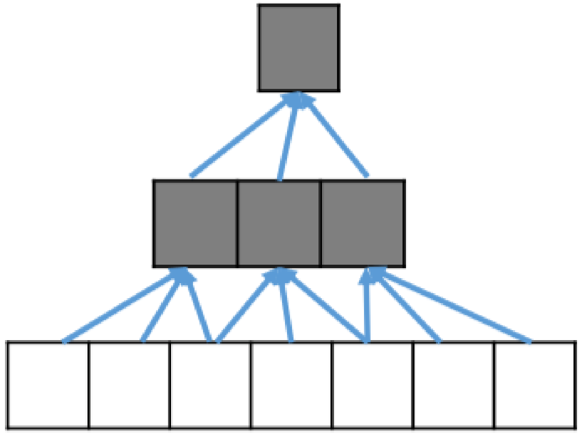




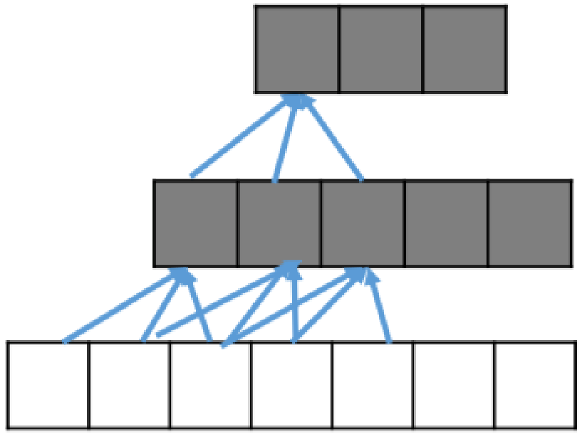
1D Convolution - Stride

- How many steps to move for the next receptive field?
  - Stride! Skip some elements when $s > 1$
- Why?
  - Less compute
  - Increase receptive field





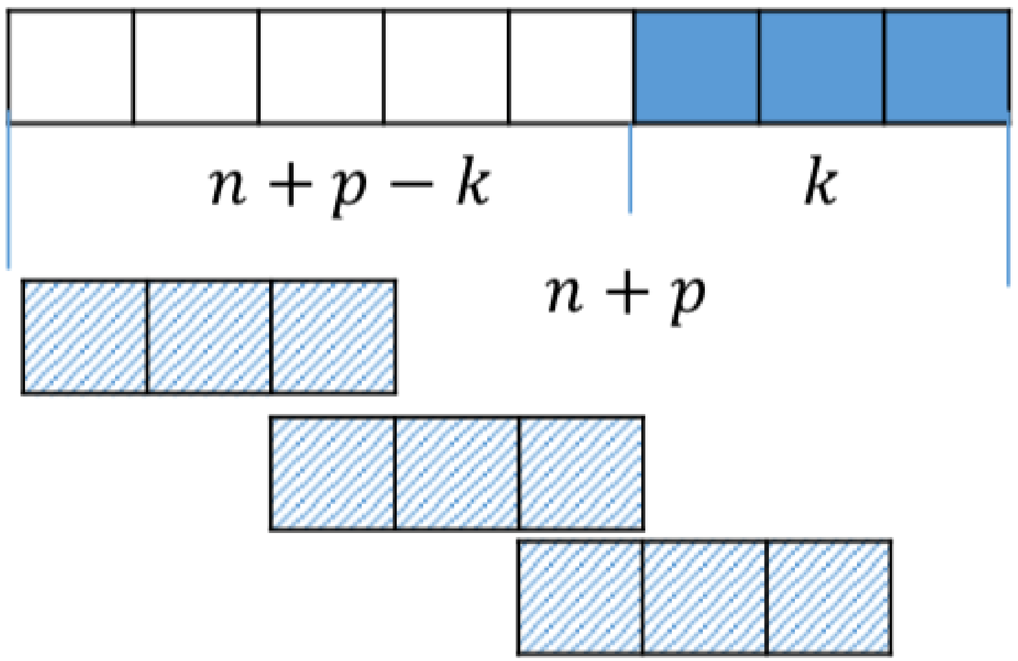
$s = 2$



$s = 1$

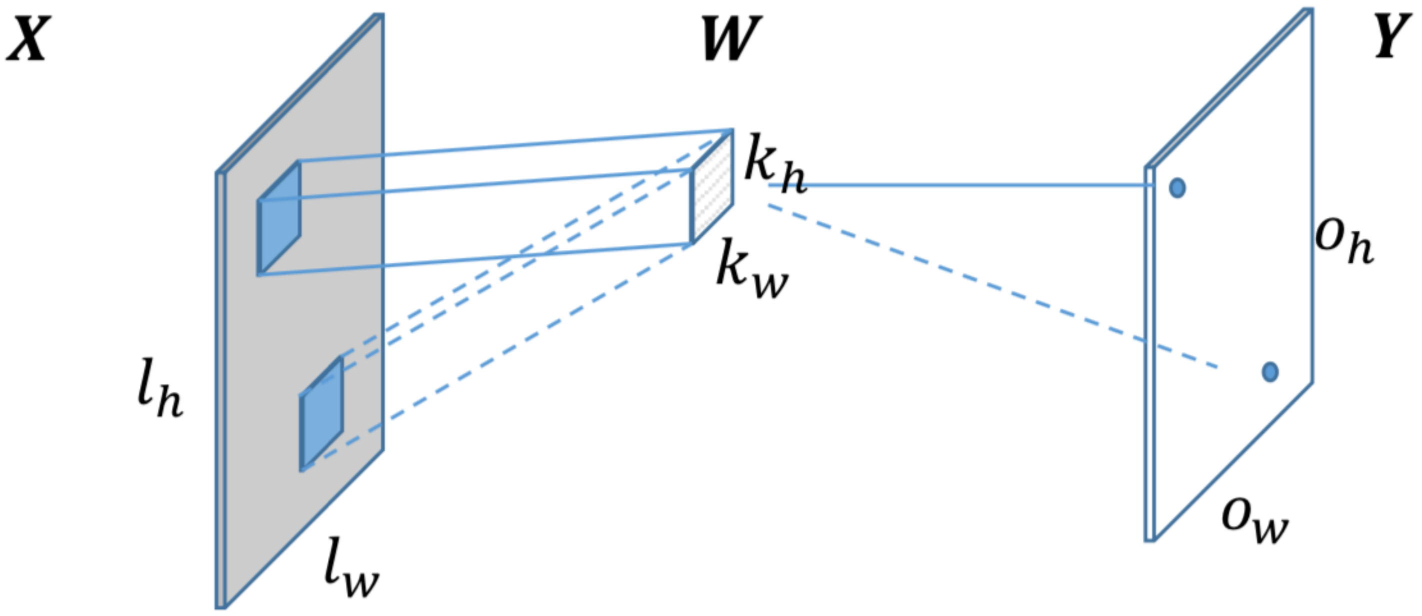
What is the output length o?

- kernel size k
- Input length n
- Padding p
- Stride s



- $o = \left\lfloor \frac{n+p-k}{s} \right\rfloor + 1$

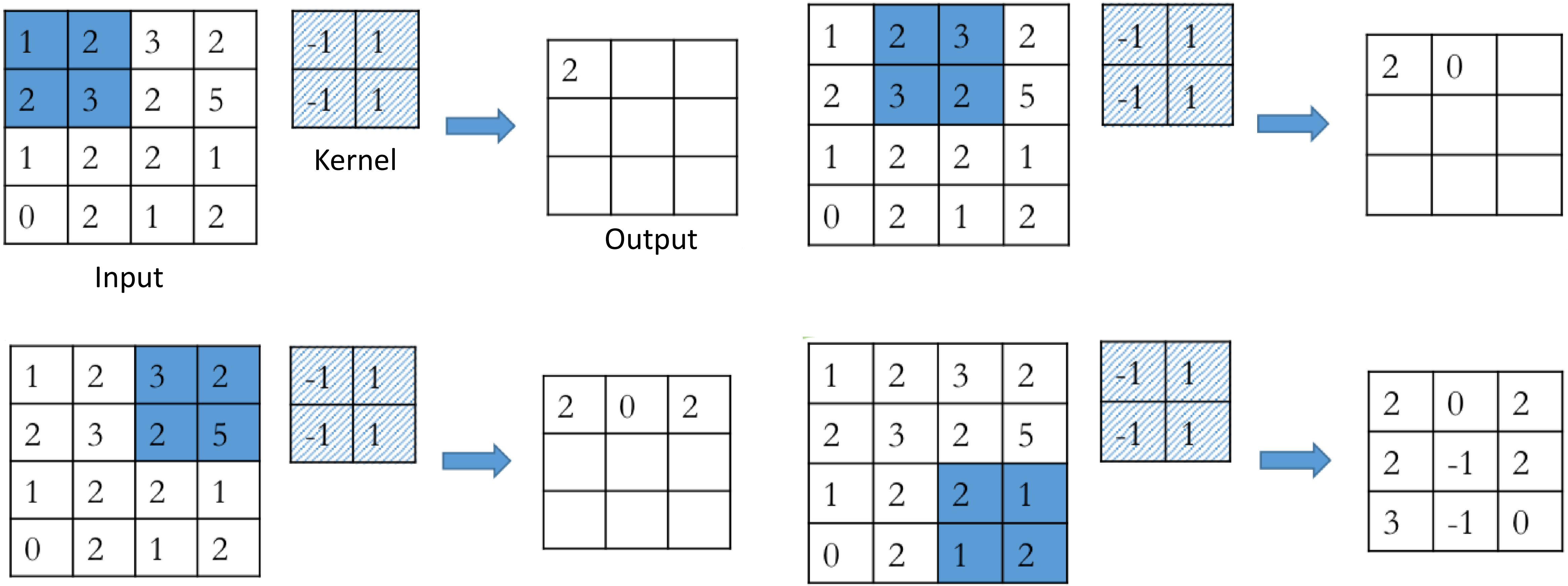
- Same but everything 2D



$$\boldsymbol{Y}_{i,j} = \sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1} \boldsymbol{X}_{i*s_h+a,j*s_w+b} \times \boldsymbol{W}_{a,b}$$

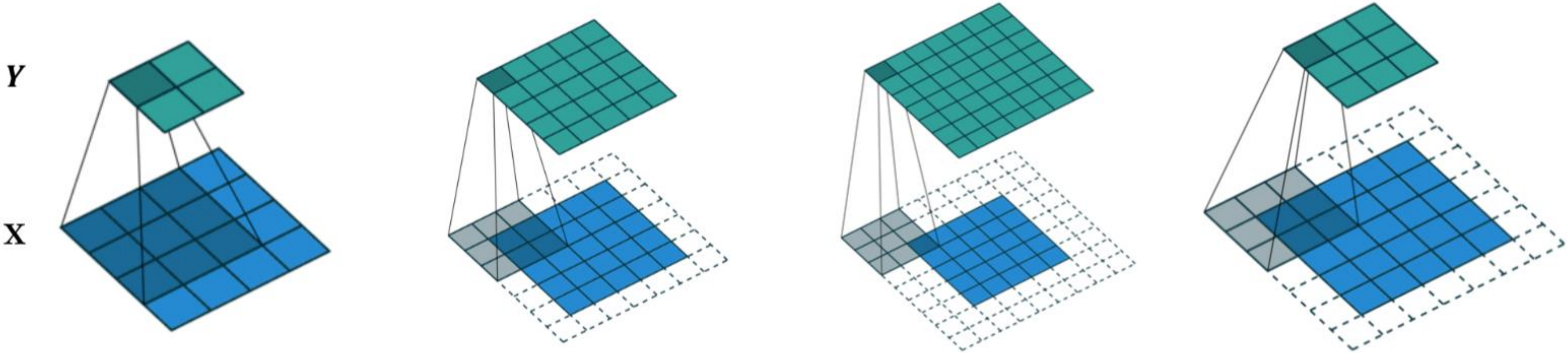
2D Convolution

$$\boldsymbol{Y}_{i,j} = \sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1} \boldsymbol{X}_{i*s_h+a,\, j*s_w+b} \times \boldsymbol{W}_{a,b}$$



2D Convolution

kernel size k
Padding p
Stride s

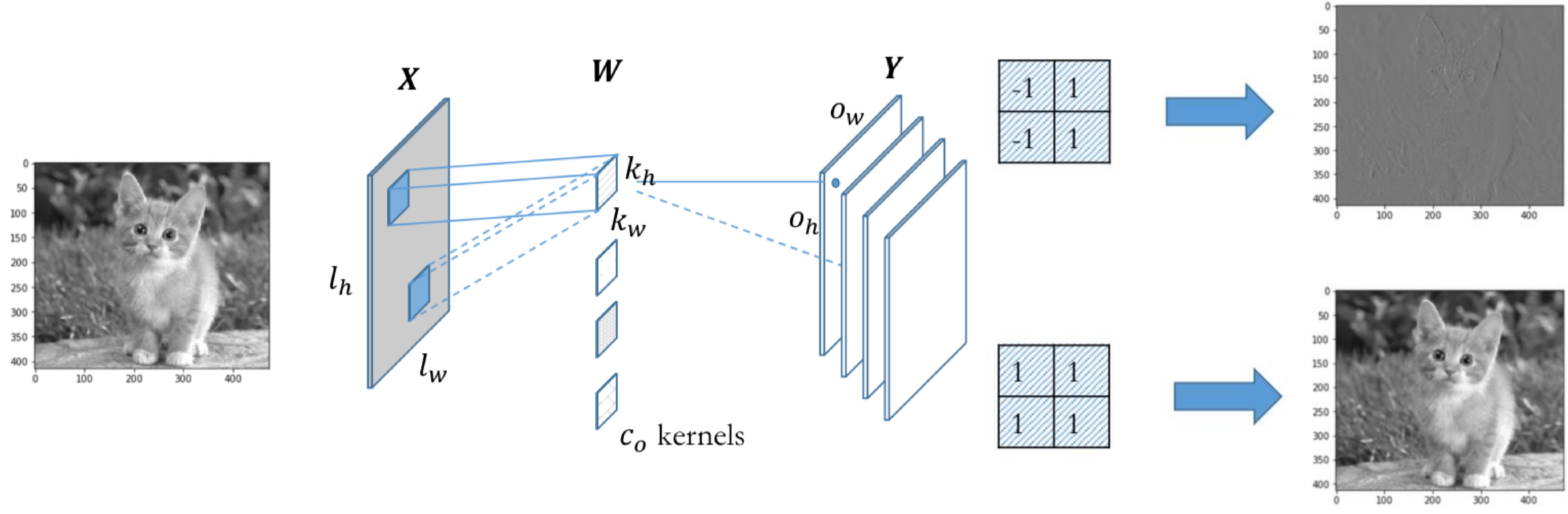


k=3, p=0, s=1 (Valid)　k=3, p=2, s=1 (Same)　k=3, p=4, s=1 (Full)　k=3, p=2, s=2

Valid: No padding

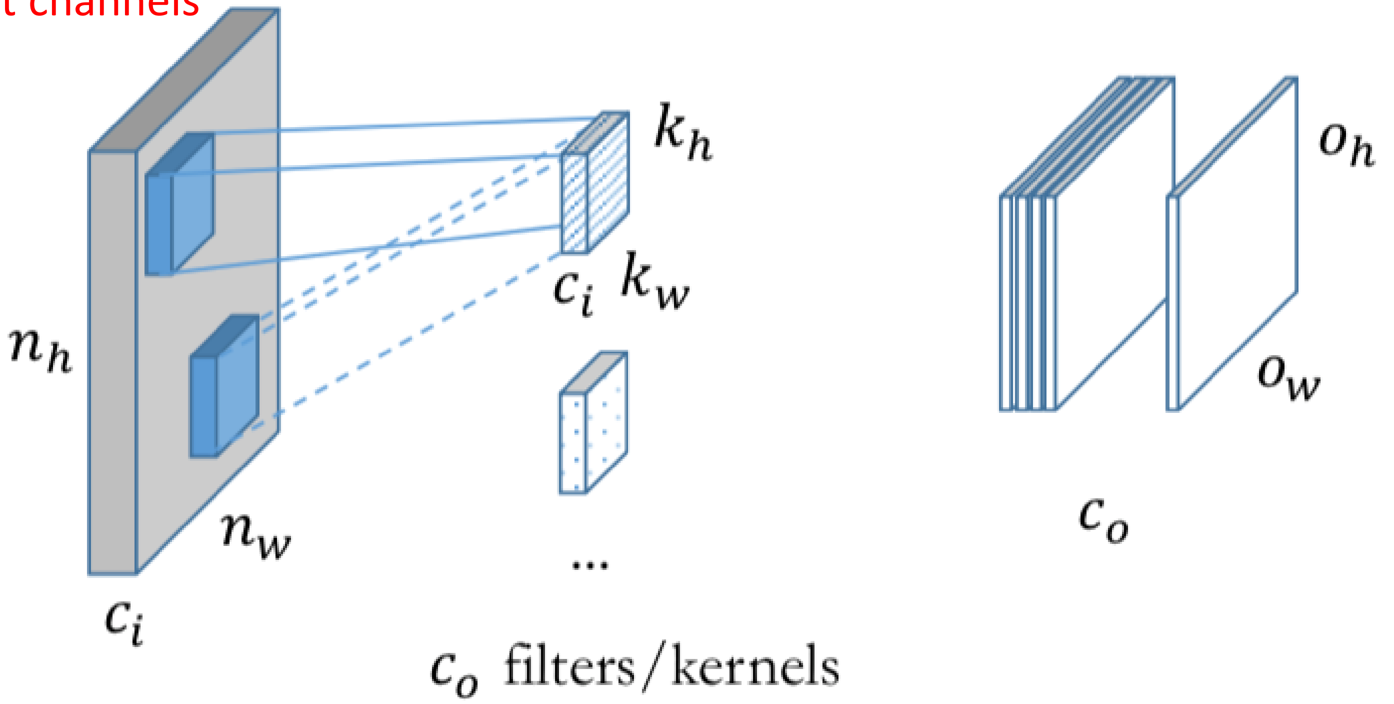
Same: Add padding shat
input size = output size

- Multiple features → Multiple kernels



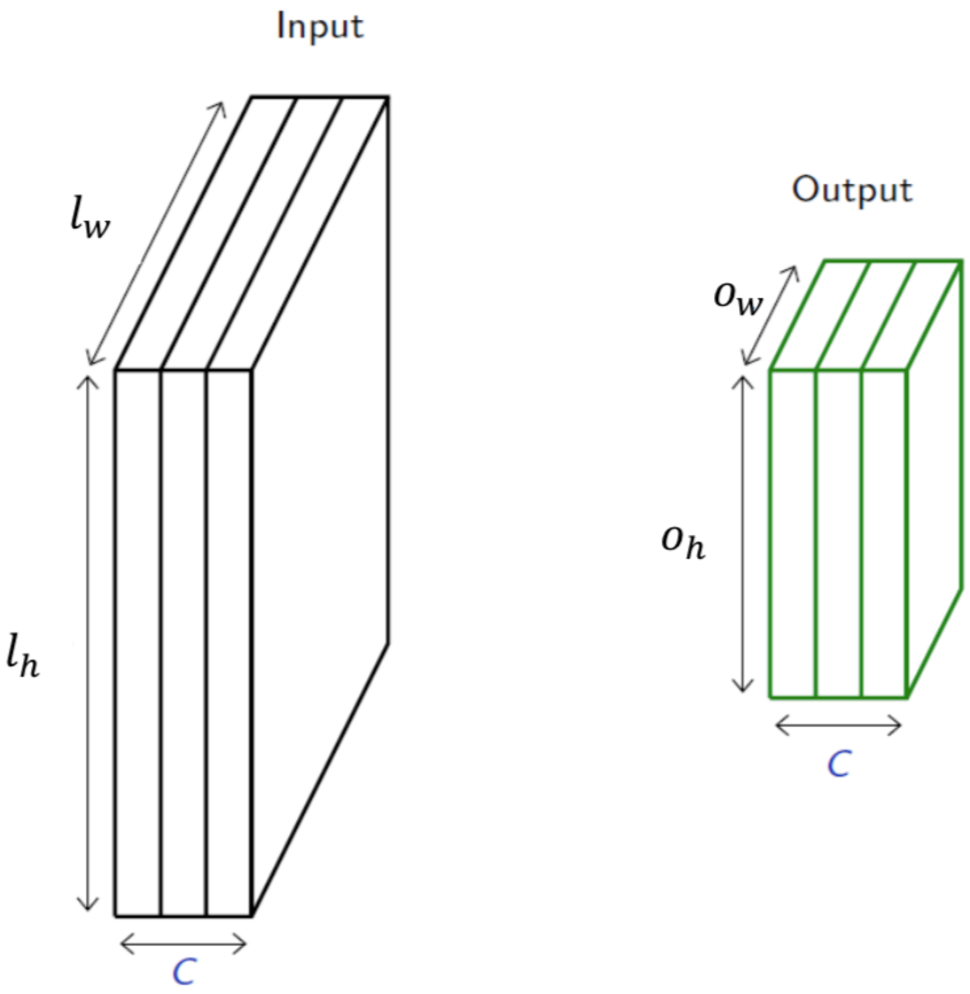
2D Convolution – Multiple Inputs & Kernels

$$Y_{l,i,j} = \underbrace{\sum_{d=0}^{c_i-1}}_{\text{Each kernel covers all input channels (indexed by d)}} \underbrace{\sum_{a=0}^{k_h-1} \sum_{b=0}^{k_w-1}}_{\text{2D convolution across the plane (indexed by a, b)}} X_{d,i+a,j+b} \times W_{l,d,a,b} + b_l \,, l \in [0, c_o)$$



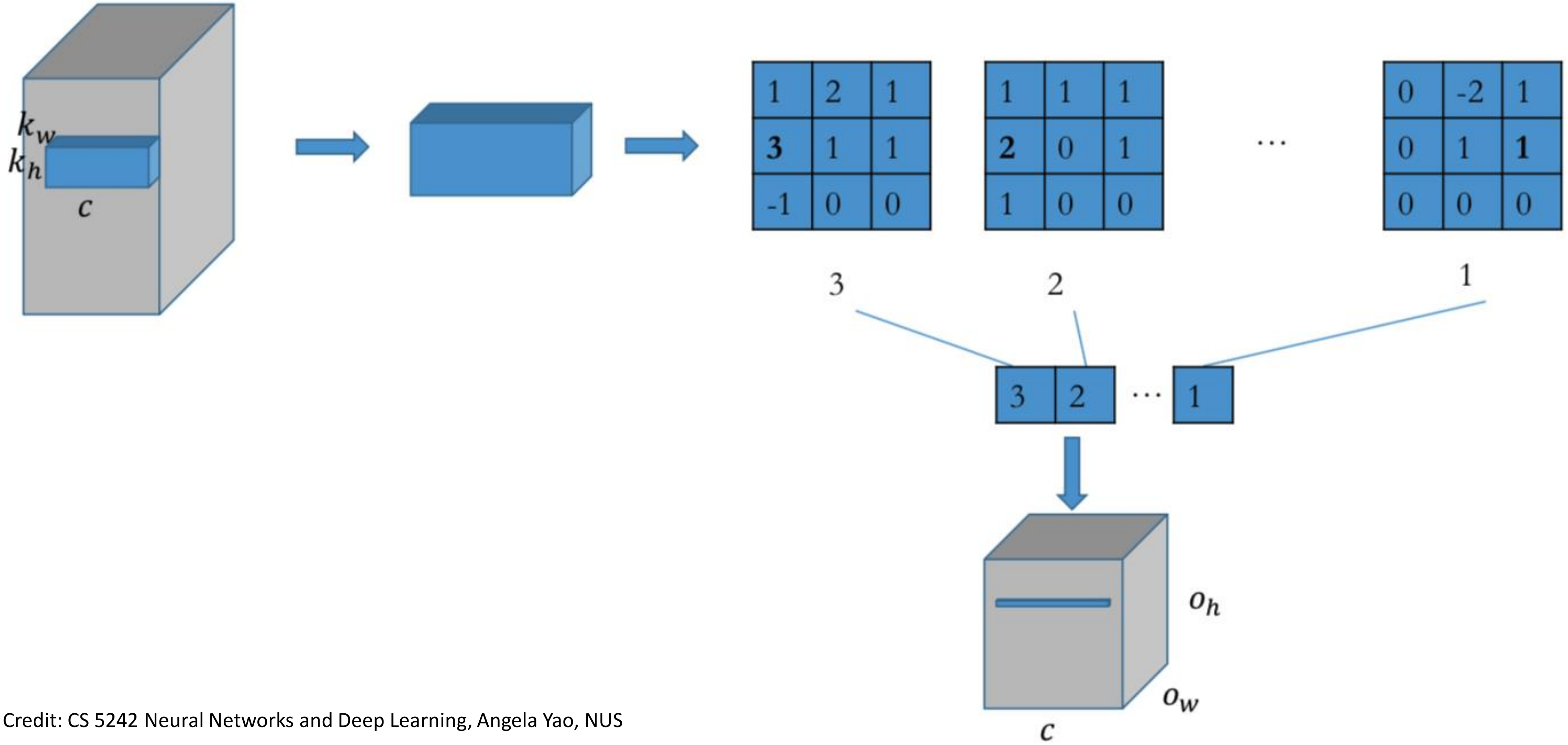
2D Convolution - Pooling

- Aggregate information in each receptive field
  - Max
  - Average
- No trainable parameter
- # input channels = # output channels
  - $c_i = c_o$
- Same padding/stride method



Credit: CS 5242 Neural Networks and Deep Learning, Angela Yao, NUS

2D Convolution – Max Pooling



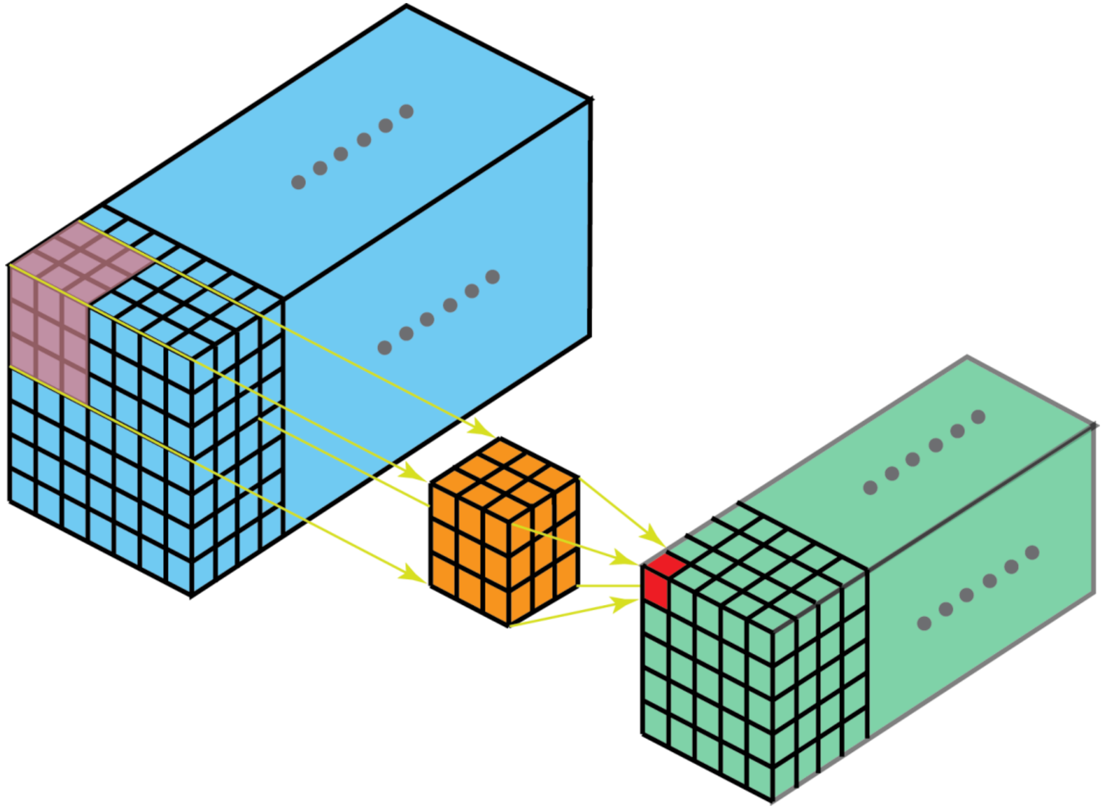
Credit: CS 5242 Neural Networks and Deep Learning, Angela Yao, NUS

2D Convolution - Summary

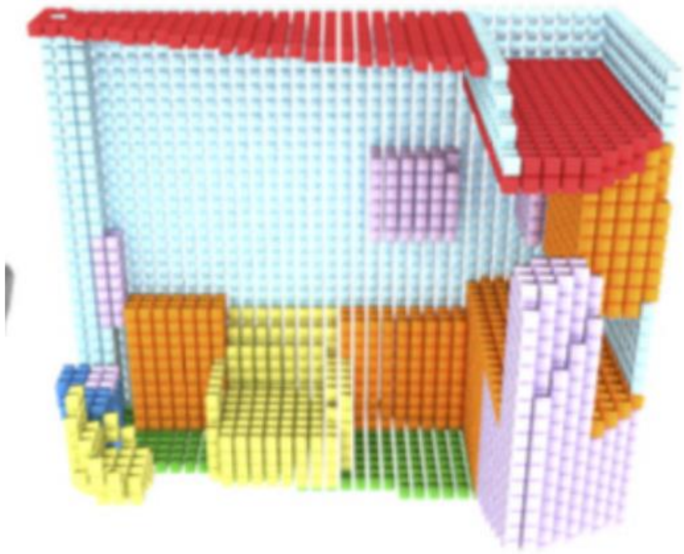
- There are c_o kernels, they share same stride and padding, NOT parameter
- Parameter size $c_o \times k_h \times k_w$
- Output shape
  - $(c_o, o_h, o_w) = \left(c_o, \left\lfloor \frac{n_h + p_h - k_h}{s_h} \right\rfloor + 1, \left\lfloor \frac{n_w + p_w - k_w}{s_w} \right\rfloor + 1\right)$
- Computation cost
  - $O((c_o \times k_h \times k_w) \times (o_h \times o_w))$
- 1D Convolution Multiple inputs/kernels
  - Same as 2D
  - Set k_h or k_w to be 1.

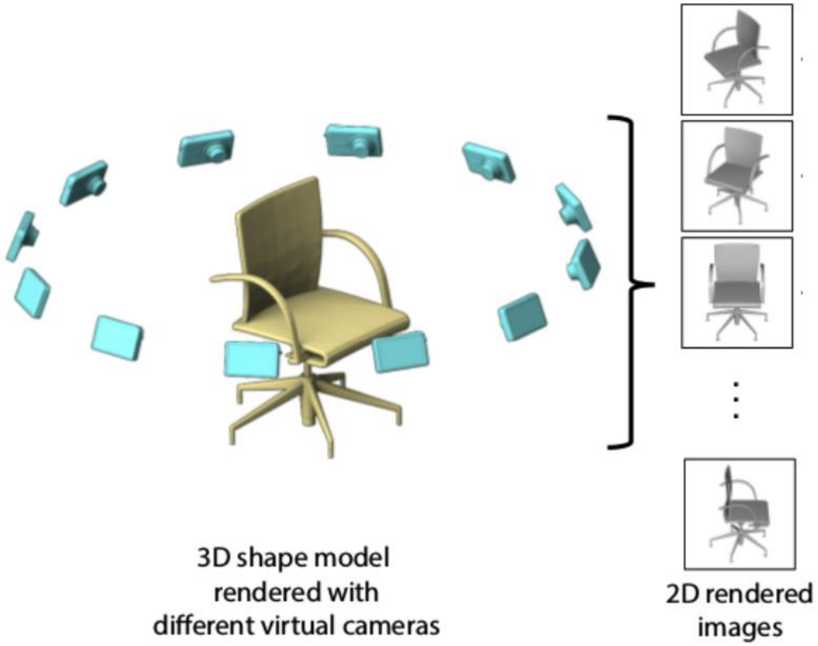
3D Convolution

- Natural extension of 2D Convolution
- Blue: input
  - Each small cube contains d features/channels
- Orange: kernel
  - Each small cube contains d weights
- Green: output
  - Each small cube is a scalar.
  - There are o green blocks



- 3D convolution
- Multi-view projection onto images + 2D convolution
- Simply run 1D/2D convolution or even MLP on point cloud





x1	y1	z1
x2	y2	z2
x3	y3	z3
x4	y4	z4

VoxNet, IROS 2015

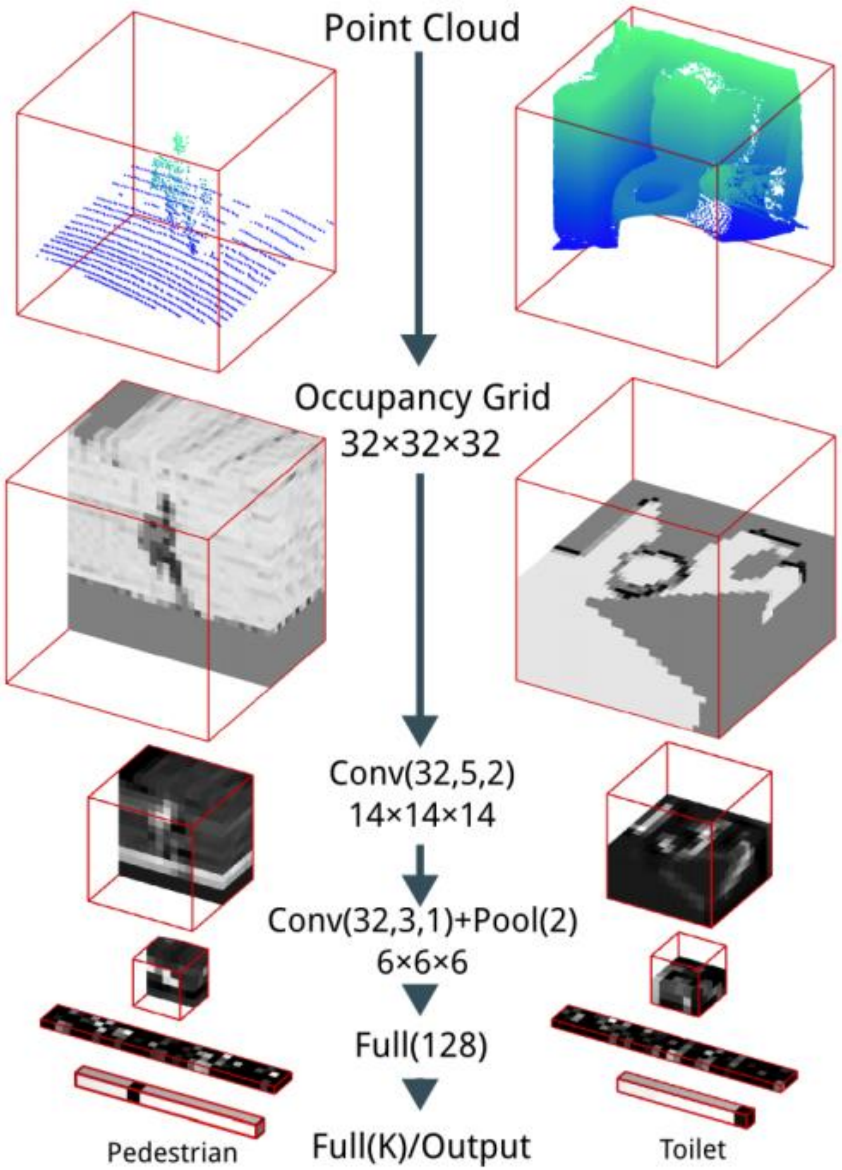
Content of each grid cell

- Binary
- Number of points
- Probability
- etc.

Accuracy on ModelNet40: 83%

Conv(o, k, s)

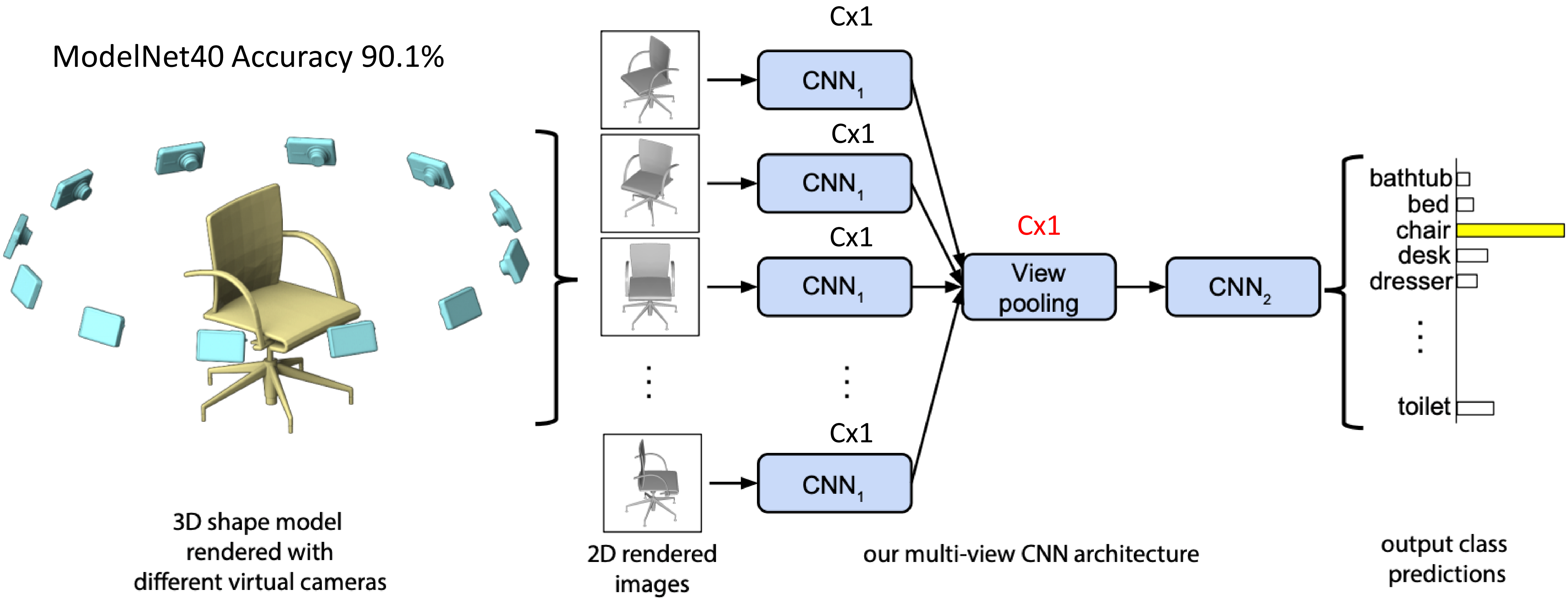
- o: number of kernels
- k: size of kernel (same for x/y/z)
- s: stride (same for x/y/z)



VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition, D Maturana, et.al.

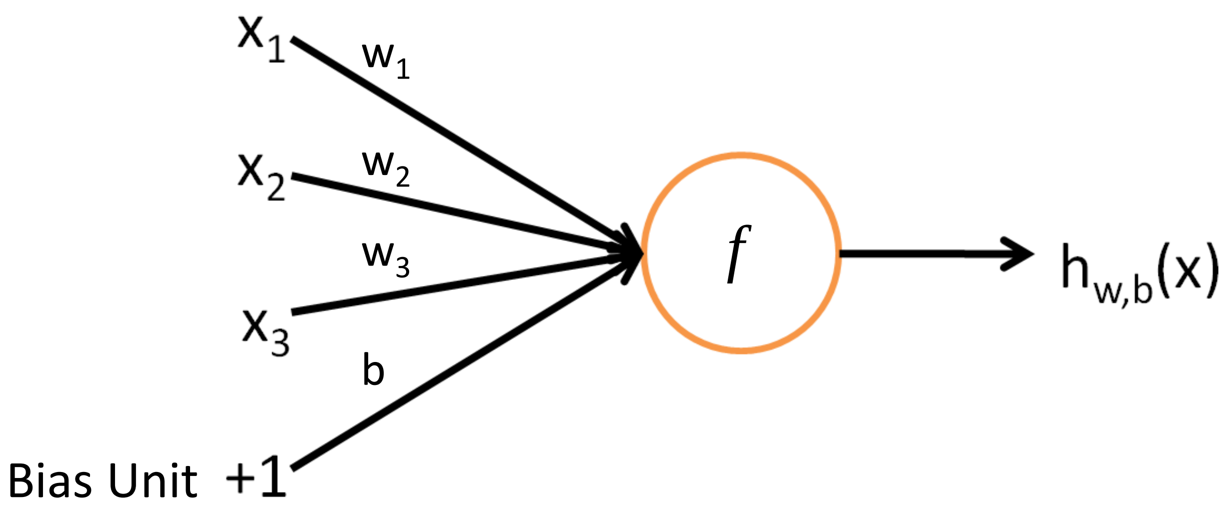
MVCNN – ICCV 2015

ModelNet40 Accuracy 90.1%



Multi-view Convolutional Neural Networks for 3D Shape Recognition, Hang Su et.al.

MLP for Point Cloud?



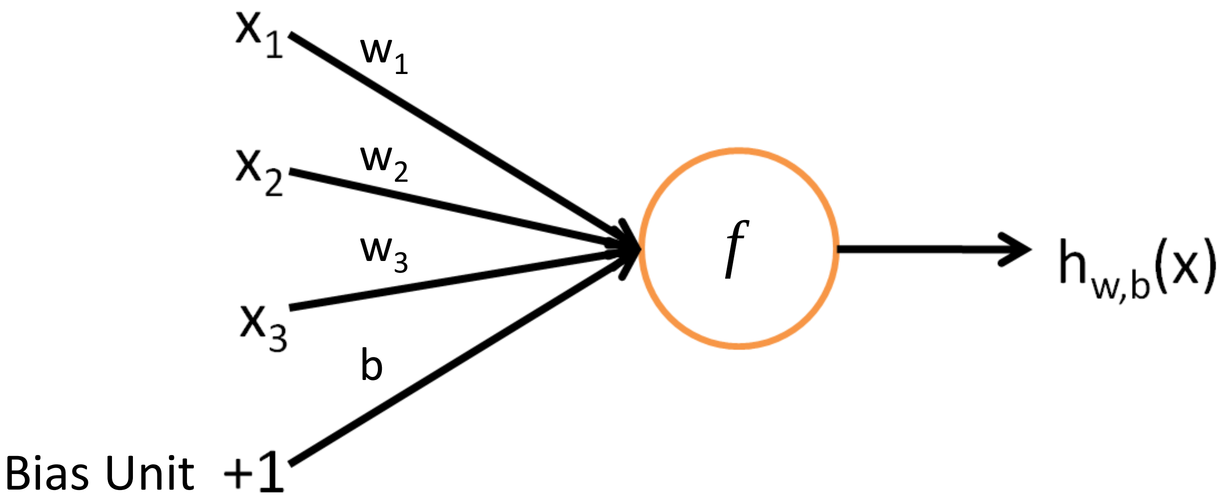
Activation function

$$h_{W,b}(x) = f(W^T x) = f\left(\sum_{i=1}^{3} w_i x_i + b\right)$$

$$= f(w_1 x_1 + w_2 x_2 + w_3 x_3 + b)$$

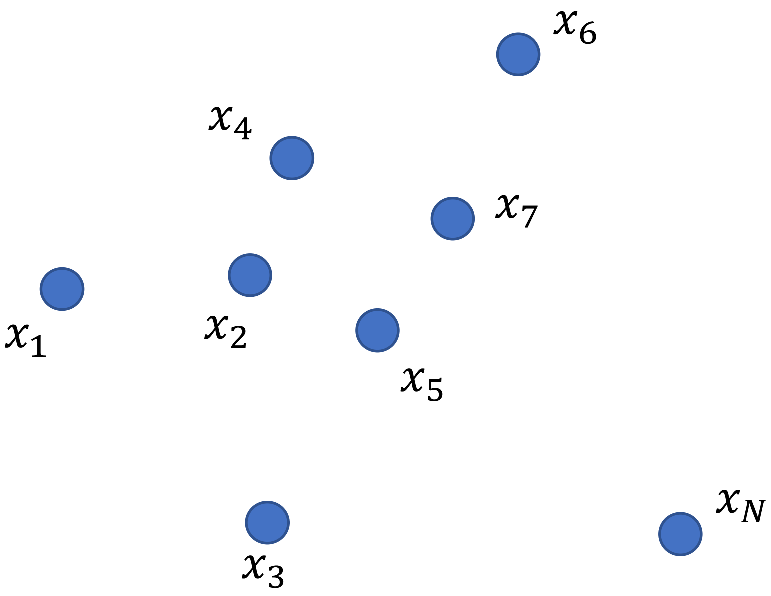
MLP for Point Cloud?

Not Permutation Invariant!



$$f(w_1 x_3 + w_2 x_1 + w_3 x_2 + b) \neq h_{W,b}(x)$$

Enumerate All Permutation?



$$X_1 = [x_1^T, x_2^T, \dots, x_N^T]^T$$

$$X_2 = [x_2^T, x_1^T, \dots, x_N^T]^T$$

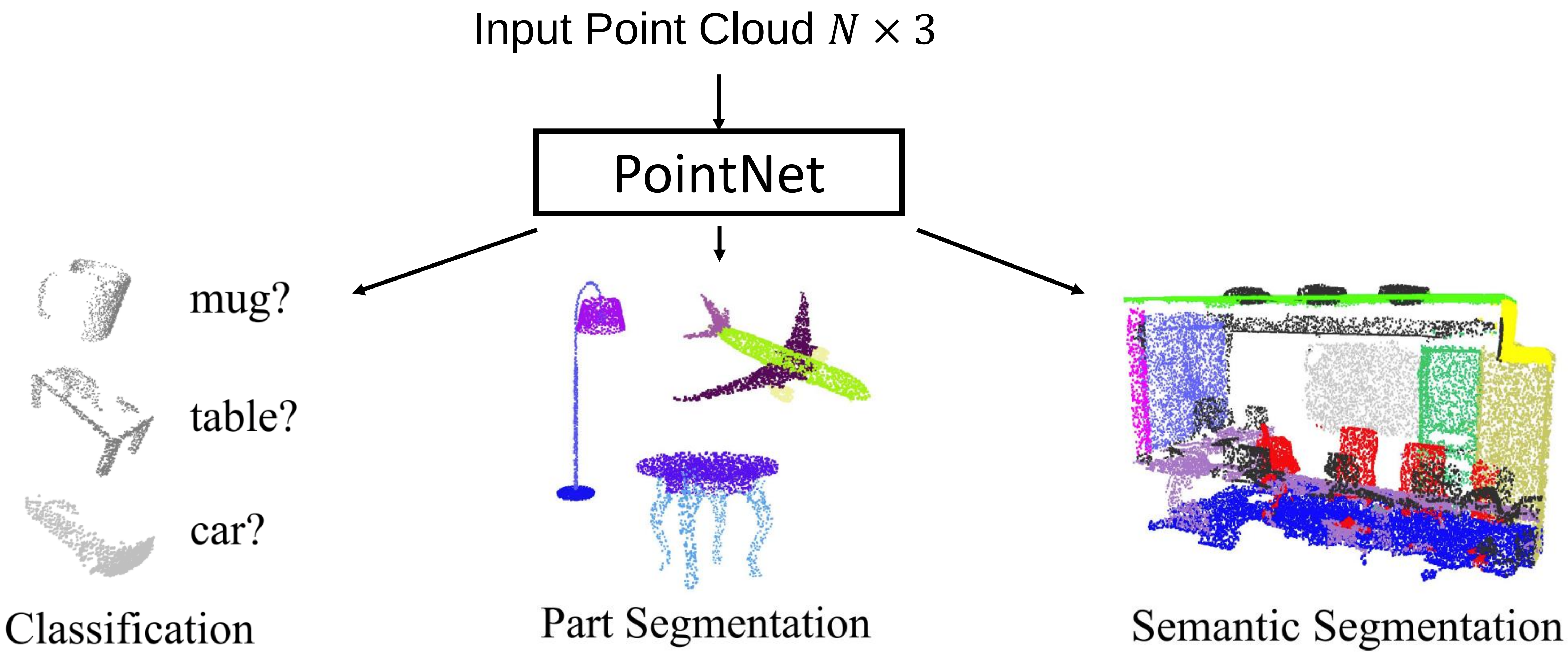
$$\vdots$$

$$\vdots$$

$$X_{N!} = \left[x_i^T, x_j^T, \dots, x_n^T\right]^T$$

$N!$ possibilities

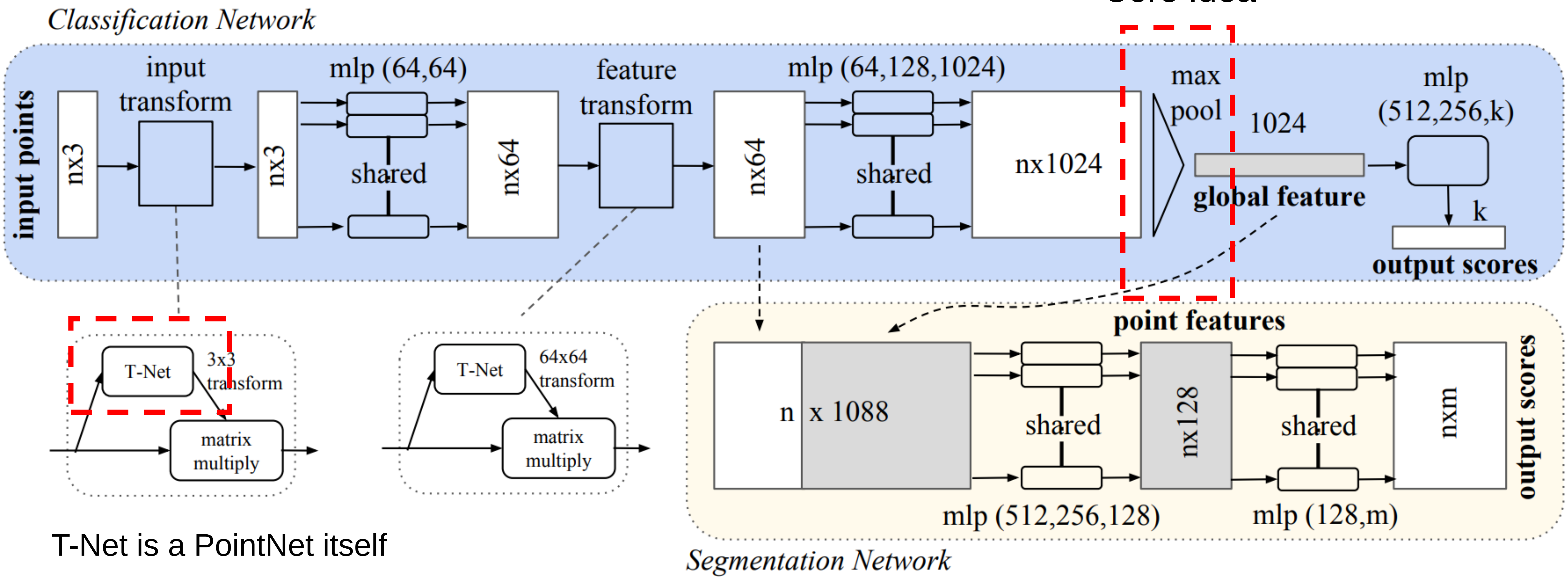
PointNet (PN) – Charles Qi, et. al., CVPR 2017,



PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. CR Qi et.al.

PointNet Architecture

Core Idea



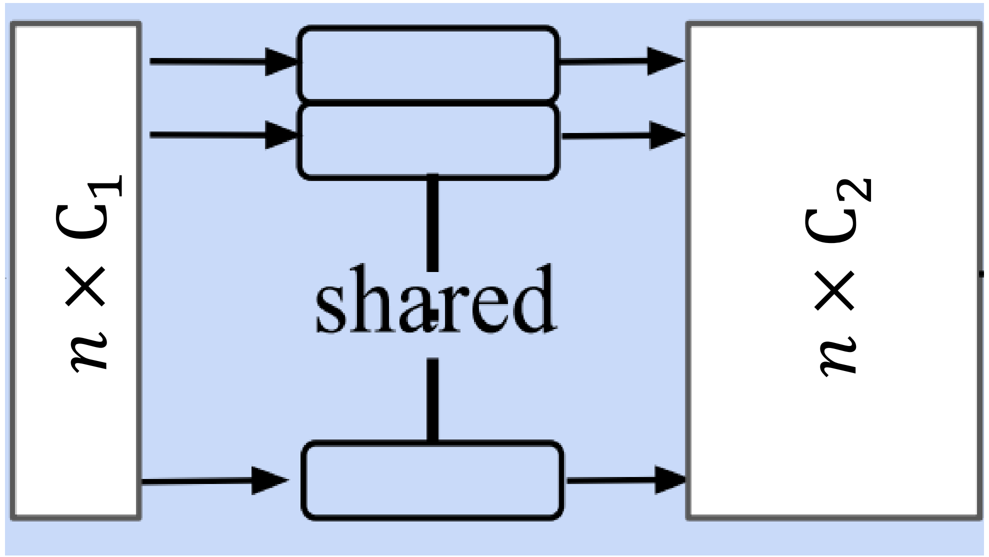
T-Net is a PointNet itself

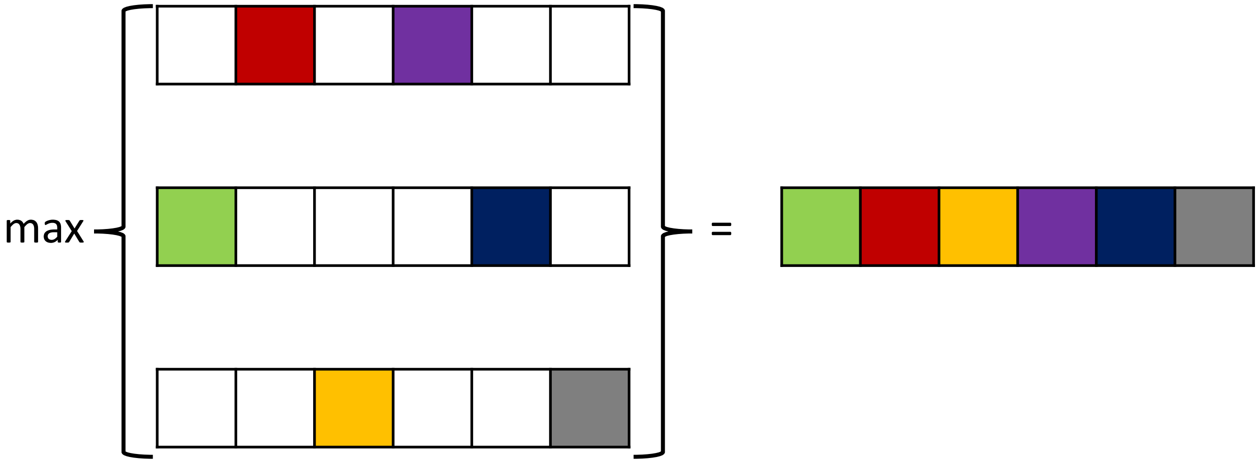
PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. CR Qi et.al.

PointNet

PointNet流程总计两步：

- Process each point (feature) independently $n \times C_1 \rightarrow n \times C_2$
- Use Max/Average to pool the features $n \times C \rightarrow 1 \times C$





Proof – PointNet is able to simulate any function on the point cloud

Some concepts:

- Input points $S = \{x_1, \cdots, x_n\}, x_i \in \mathbb{R}^m, x_i \in [0, 1]$.
  - Denote the space of S as χ, i.e., $S \in \chi$
- A continuous function $f : \chi \rightarrow \mathbb{R}$
  - This is the "any function" we want to simulate
- MAX function: takes n vectors, give element-wise maximum

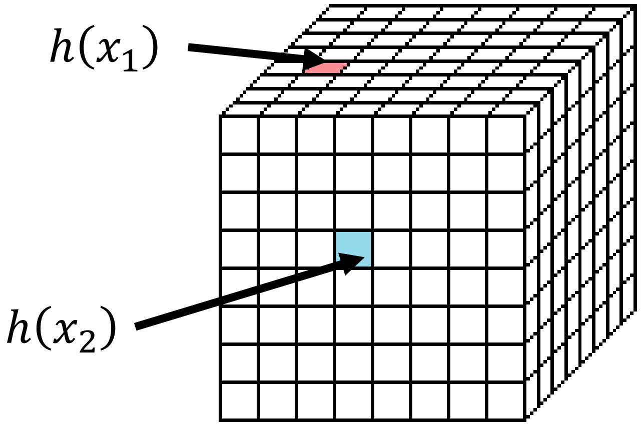
- Given continuous $f: \chi \rightarrow \mathbb{R}$
- $\forall \epsilon > 0, \exists h: \mathbb{R}^m \rightarrow \mathbb{R}^{m'}, and\ \gamma: \mathbb{R}^n \rightarrow \mathbb{R}$
- s.t. $\forall S \in \chi, e.g. S = \{x_1, \cdots, x_n\}, x_i \in \mathbb{R}^m$

$$\left| f(S) - \underbrace{\gamma}_{\text{MLP for global feature}} \left(MAX\left(h(x_1), \cdots, \underbrace{h}_{\text{Shared MLP}}(x_n) \right) \right) \right| < \epsilon$$

PointNet – Proof

$$\left|f(S) - \gamma\left(MAX\left(h(x_1), \cdots, h(x_n)\right)\right)\right| < \epsilon$$

- $h(\cdot)$ maps x_i to the some deterministic position of a huge vector
  - By Voxel Grid Downsampling
- $MAX\left(h(x_1), \cdots, h(x_n)\right)$ simply builds a voxel grid representation.
  - There will be lots of 0 elements because of empty cells in voxel grid.
- $\gamma(\cdot) = reconstruct\ the\ points + f(\cdot)$
- Done

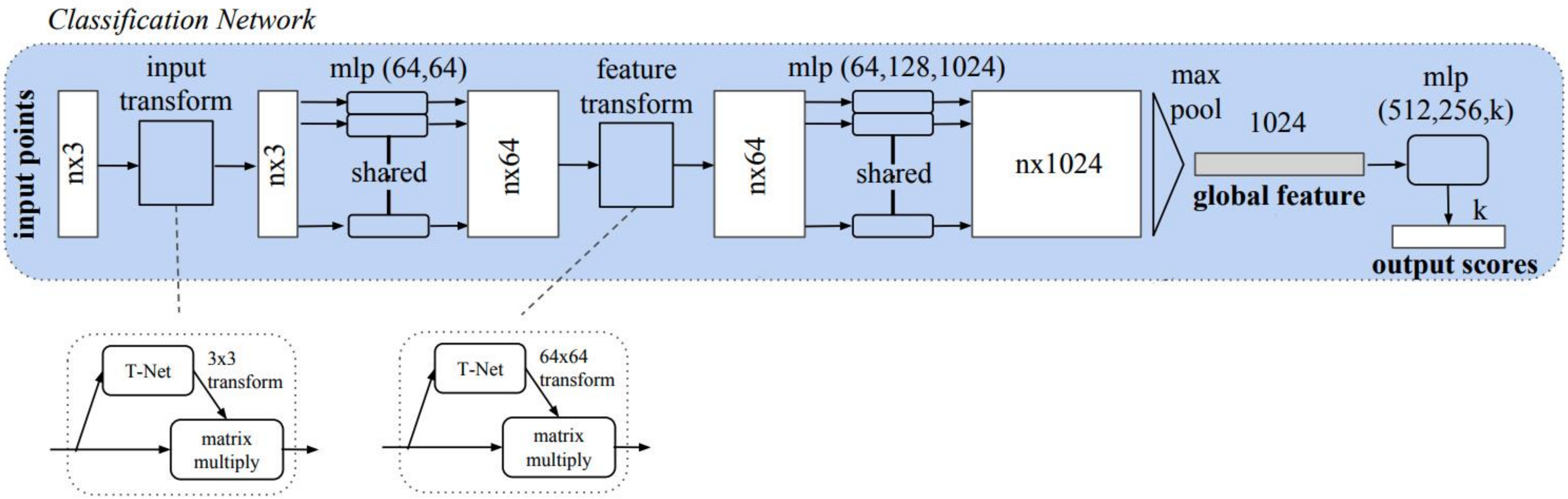


$$MAX\left\{\begin{array}{ll} h(x_1) & \begin{array}{|c|c|c|c|c|c|c|}\hline \cdots & 1 & 0 & 0 & \cdots & 0 & \cdots \\ \hline\end{array} \\ h(x_2) & \begin{array}{|c|c|c|c|c|c|c|}\hline \cdots & 0 & 0 & 1 & \cdots & 0 & \cdots \\ \hline\end{array} \\ & \\ h(x_n) & \begin{array}{|c|c|c|c|c|c|c|}\hline \cdots & 0 & 0 & 0 & \cdots & 1 & \cdots \\ \hline\end{array} \end{array}\right\} = \begin{array}{|c|c|c|c|c|c|c|}\hline \cdots & 1 & \cdots & 0 & \cdots & 1 & \cdots \\ \hline\end{array}$$

$$\downarrow \gamma$$

$$\begin{array}{|c|c|c|c|c|c|}\hline x_1' & x_2' & x_3' & \cdots & \cdots & x_n' \\ \hline\end{array}$$

PointNet – Classification



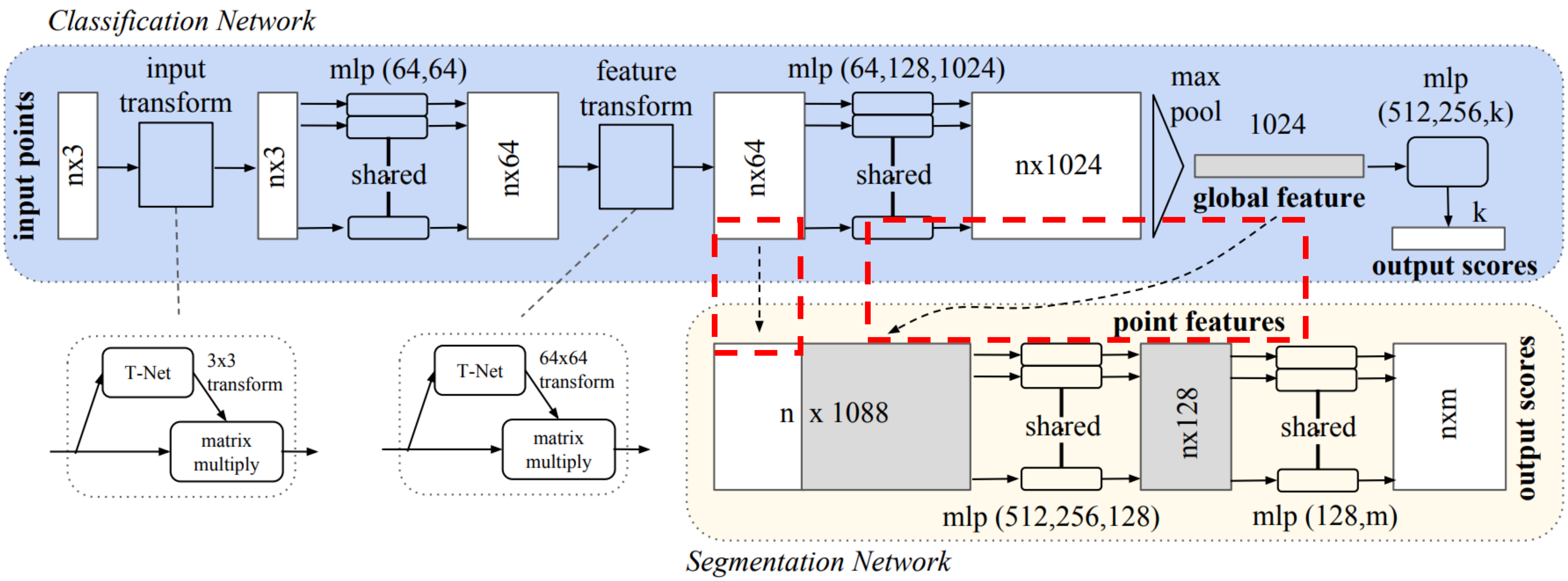
PointNet – Classification

	input	#views	accuracy avg. class	accuracy overall
SPH [11]	mesh	-	68.2	-
3DShapeNets [28]	volume	1	77.3	84.7
VoxNet [17]	volume	12	83.0	85.9
Subvolume [18]	volume	20	86.0	**89.2**
LFD [28]	image	10	75.5	-
MVCNN [23]	image	80	**90.1**	-
Ours baseline	point	-	72.6	77.4
Ours PointNet	point	1	86.2	**89.2**

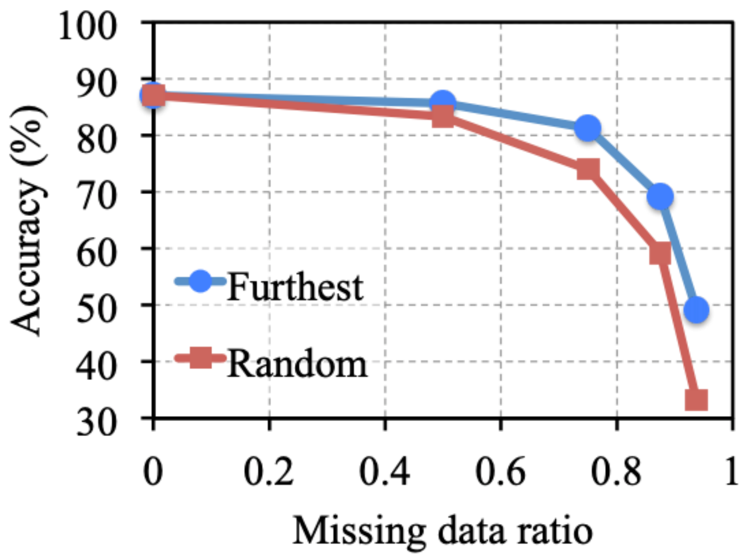
Table 1. **Classification results on ModelNet40.** Our net achieves state-of-the-art among deep nets on 3D input.

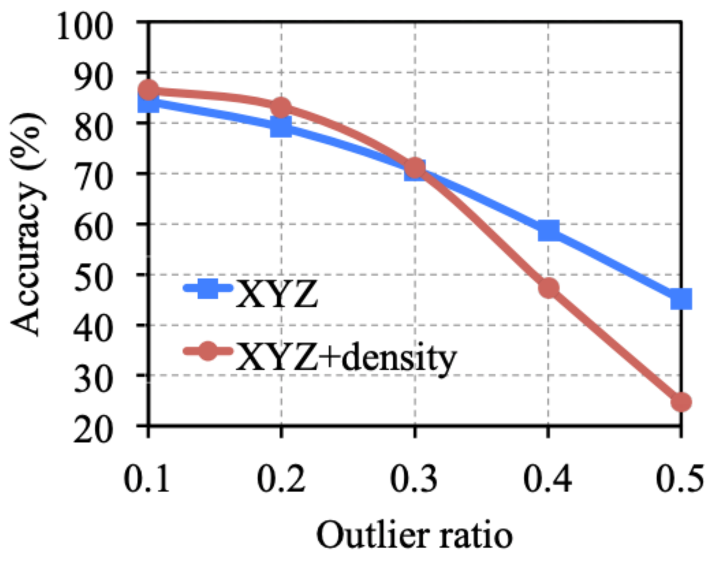
PointNet – Segmentation

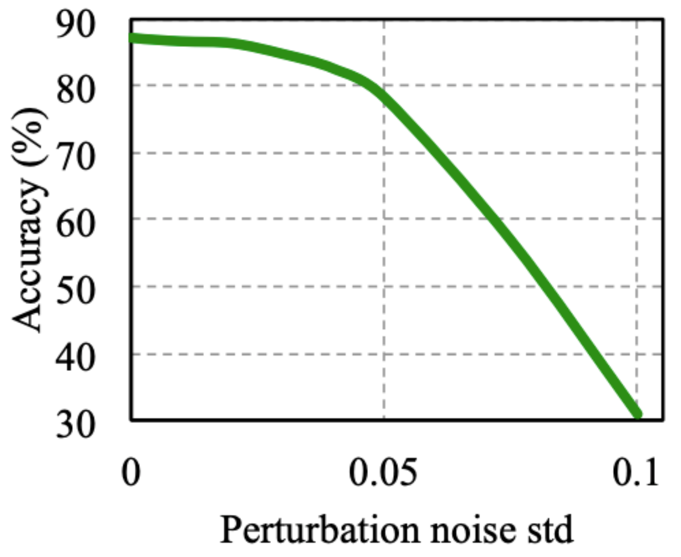
- Segmentation is per-point classification
  - MLP on per-point feature, instead of global feature
  - How to get per-point feature?



PointNet – Robustness







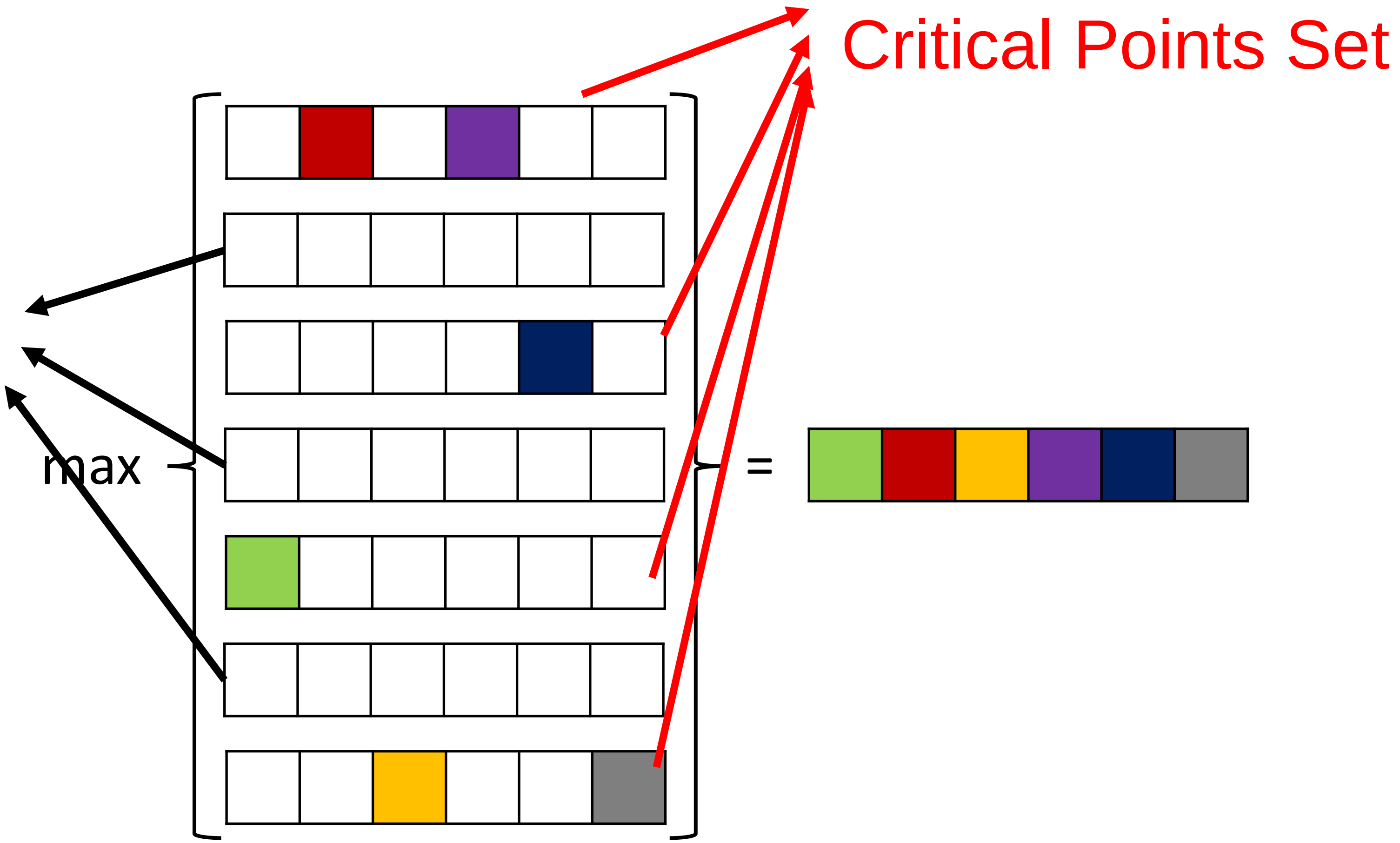
Left: Downsample points by random sampling / FPS (farthest point sampling)

Middle: Insert uniformally sampled points.

Right: Add Gaussian noise to input points

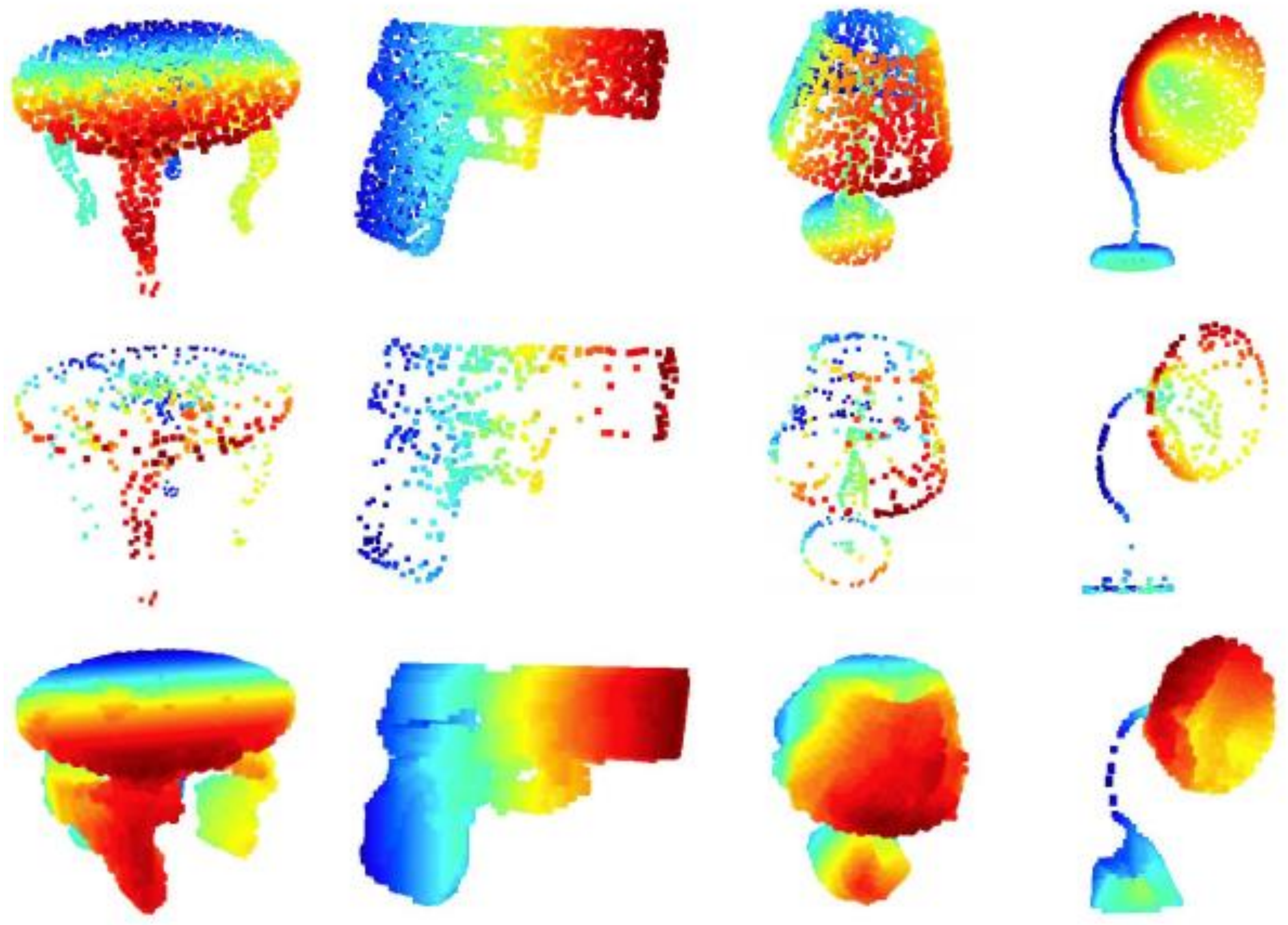
Upper Bound Shape

- Add those “useless” points



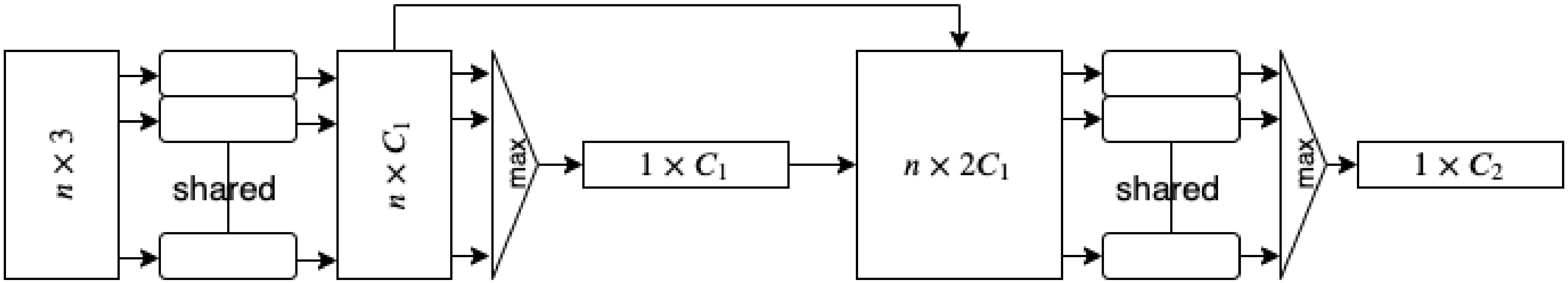
Critical Points Set & Upper Bound Shape

- Original Shape
- Critical Point Sets
- Upper Bound Shape



Voxel Feature Encoding (VFE)

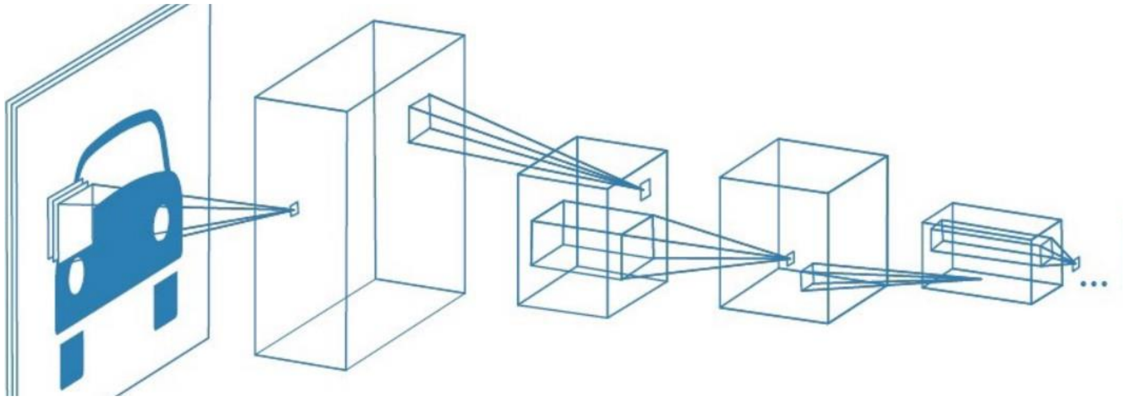
- Proposed in VoxelNet, a 3D object detection in point cloud, Lecture 6
- Two PN in a row
- Performs better than PN



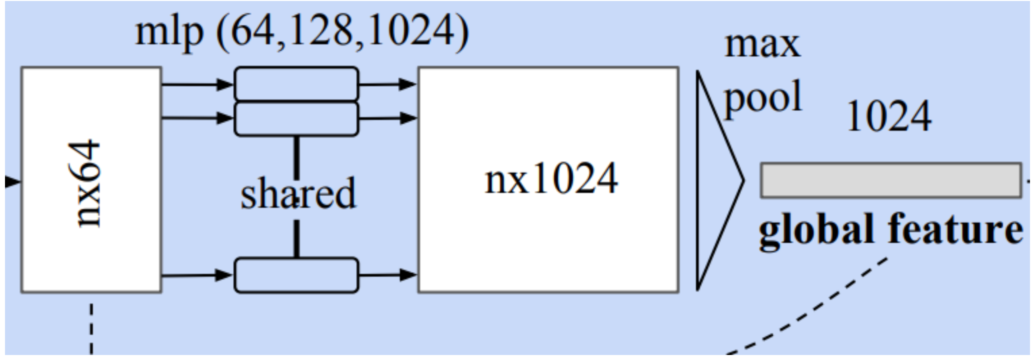
Lack of hierarchical feature aggregation

- CNN has multiple, increasing receptive field
- PointNet has one receptive field – all points

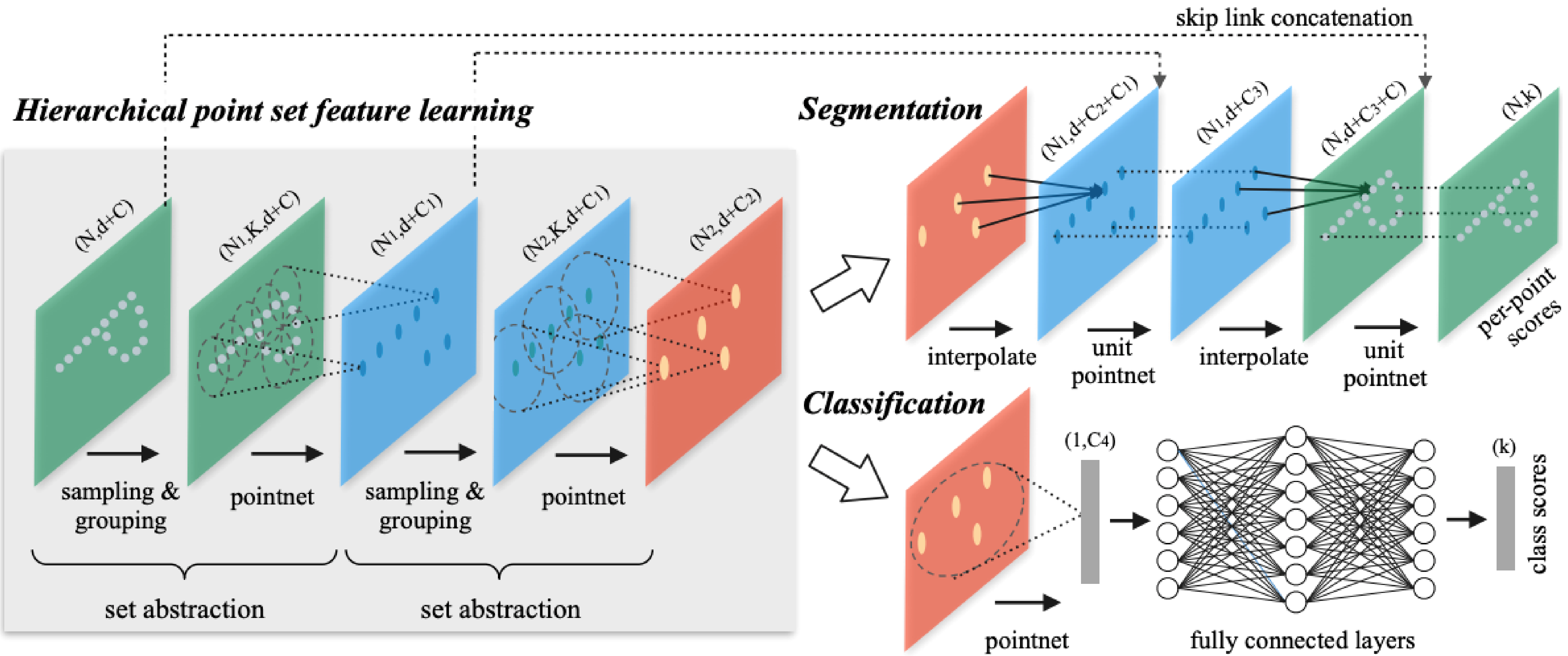
CNN



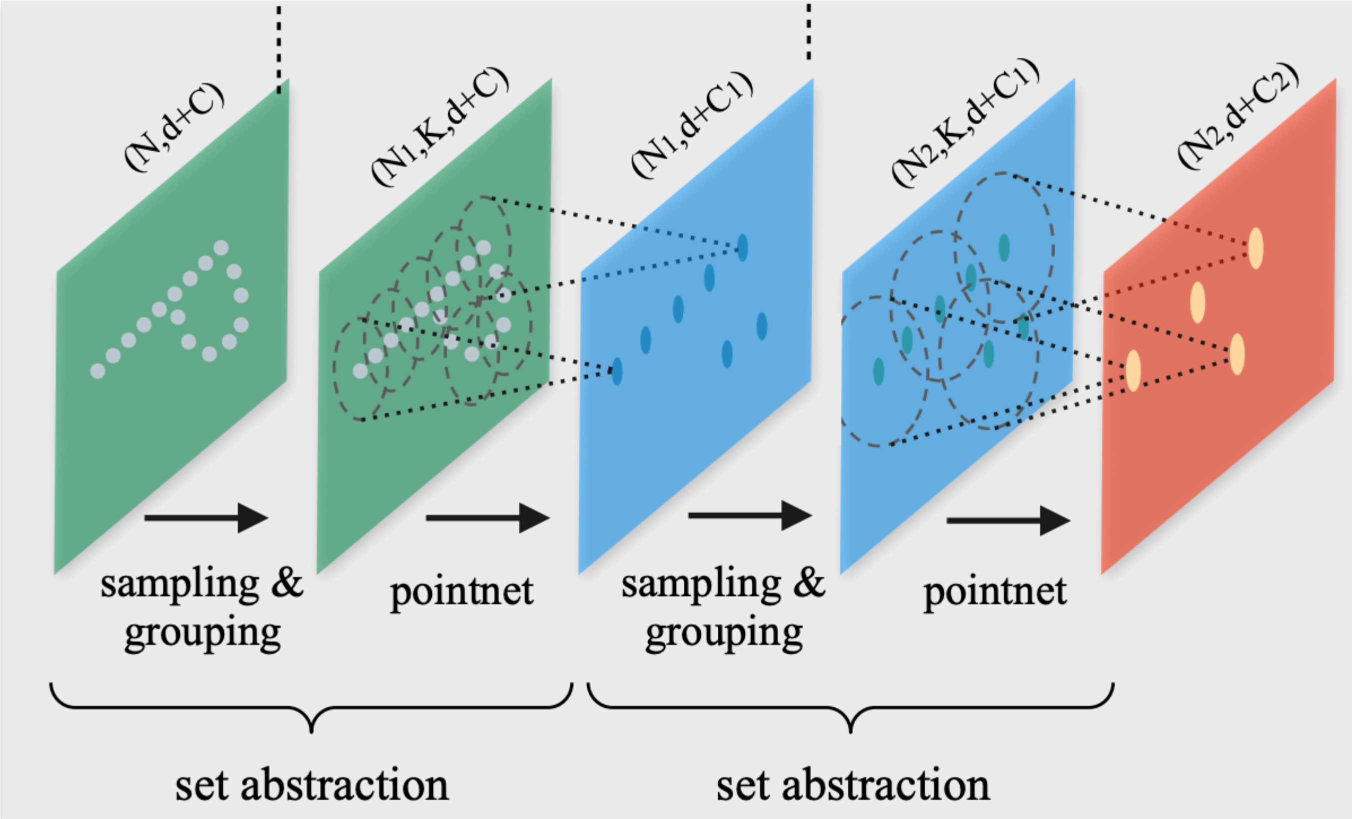
PointNet



PointNet++ - Looks complicated?



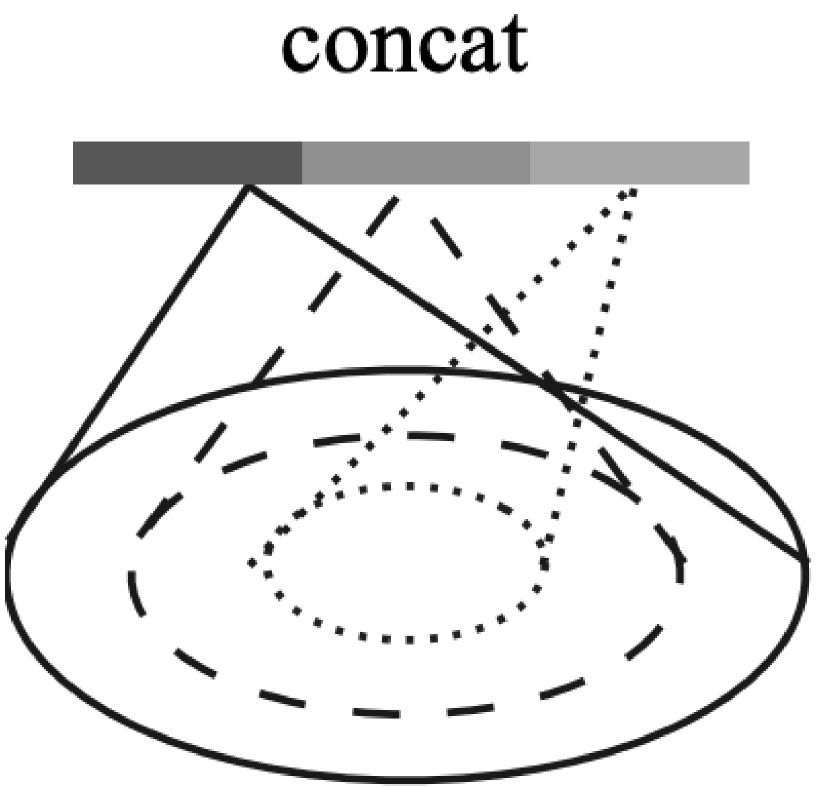
PointNet++ - Hierarchical Features



- In each set abstraction:
- Sampling: FPS
  - Point #: $N_{i-1} \to N_i$
- Grouping:
  - Radius Neighbors + random sampling
  - K Nearest Neighbors
- PointNet
  - Point #: N_i
  - Channel #: $C_{i-1} \to C_i$
  - Concatenate with coordinates so $d + C_{i-1} \to C_i$
  - *Normalize point coordinate in the group*
  - *Centered with the Node*

Multi-scale grouping (MSG)

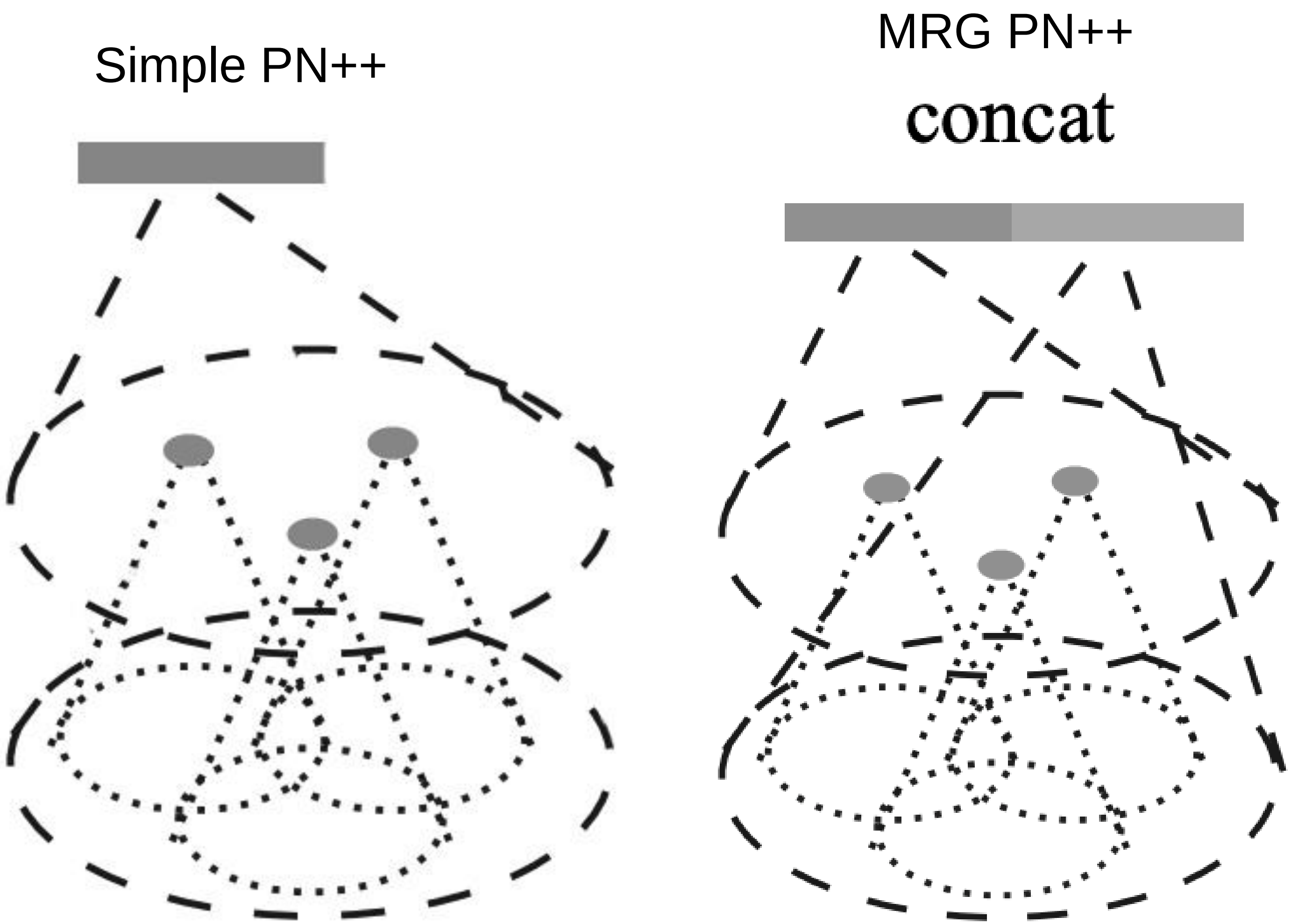
- One Sampling
- Multiple Grouping & PointNet
  - $r = 0.1$ grouping + PN
  - $r = 0.2$ grouping + PN
  - $r = 0.4$ grouping + PN
  - This is compute intensive
- Concatenate the multi-scale feature vectors



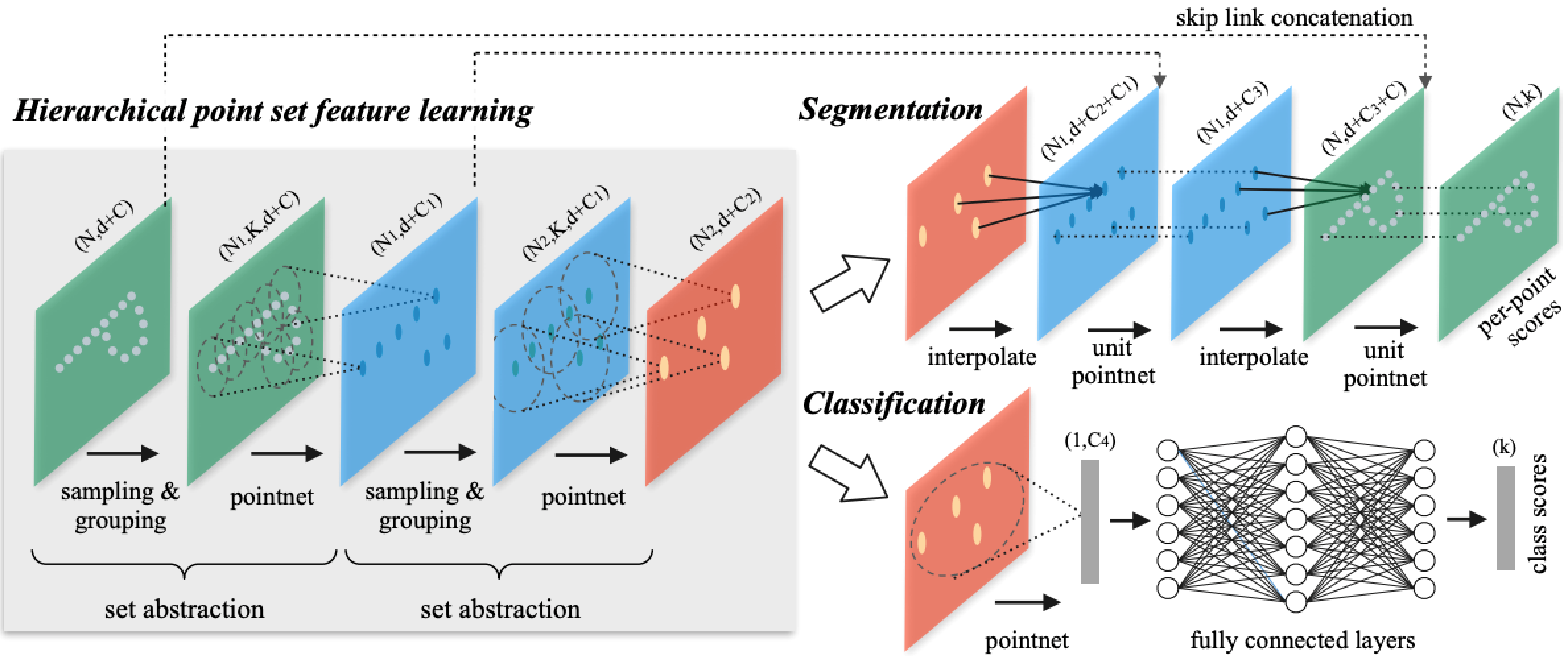
Multi-resolution grouping (MRG)

Get features from

- previous level
- previous previous level
- Still increases compute

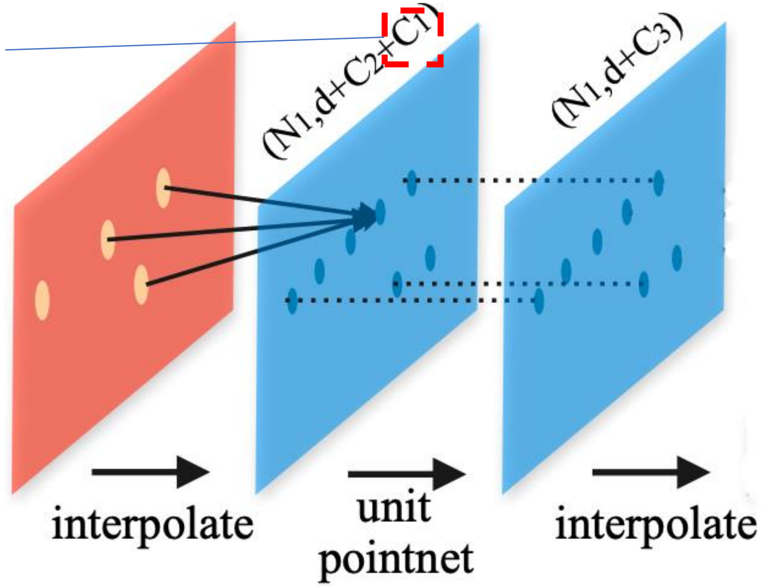


PointNet++ - Segmentation



PointNet++ - Segmentation

From encoding stage



- Interpolation
  - Upsample the features from previous layer
- $x \in \mathbb{R}^3$: point coordinates at the upsampled level, $\# = N_1$
- $f \in \mathbb{R}^{C_2}$: interpolated features
- $x_i \in \mathbb{R}^3$: point coordinates at the previous level (N_2 points)
- $w_i \in \mathbb{R}$: reciprocal of distance $d(x, x_i)$
- $f_i \in R^{C_2}$: point features at the previous level

$$f^{(j)}(x) = \frac{\sum_{i=1}^{k} w_i(x) f_i^{(j)}}{\sum_{i=1}^{k} w_i(x)} \quad \text{where} \quad w_i(x) = \frac{1}{d(x, x_i)^p},\ j = 1, \ldots, C$$

- PointNet vanilla: without T-Net
  - Same as a single layer in PN++

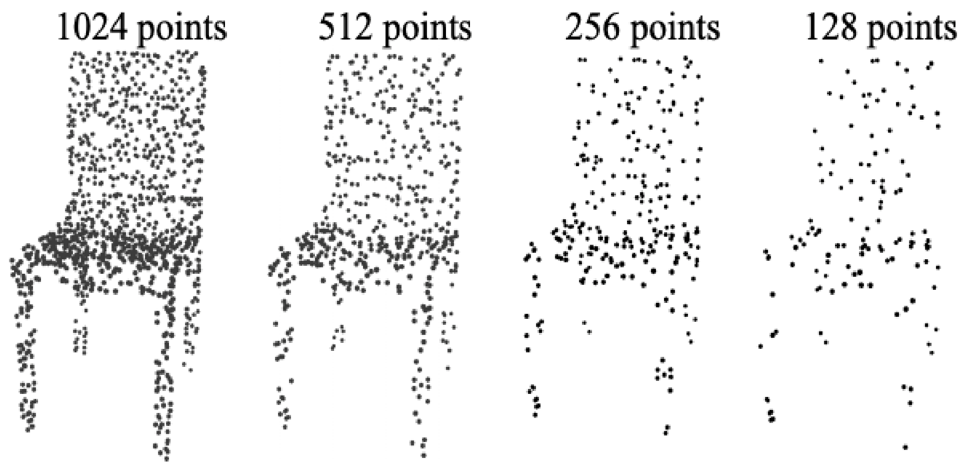
- There isn't T-Net in PN++

Method	Input	Accuracy (%)
Subvolume [21]	vox	89.2
MVCNN [26]	img	90.1
PointNet (vanilla) [20]	pc	87.2
PointNet [20]	pc	89.2
Ours	pc	90.7
Ours (with normal)	pc	**91.9**

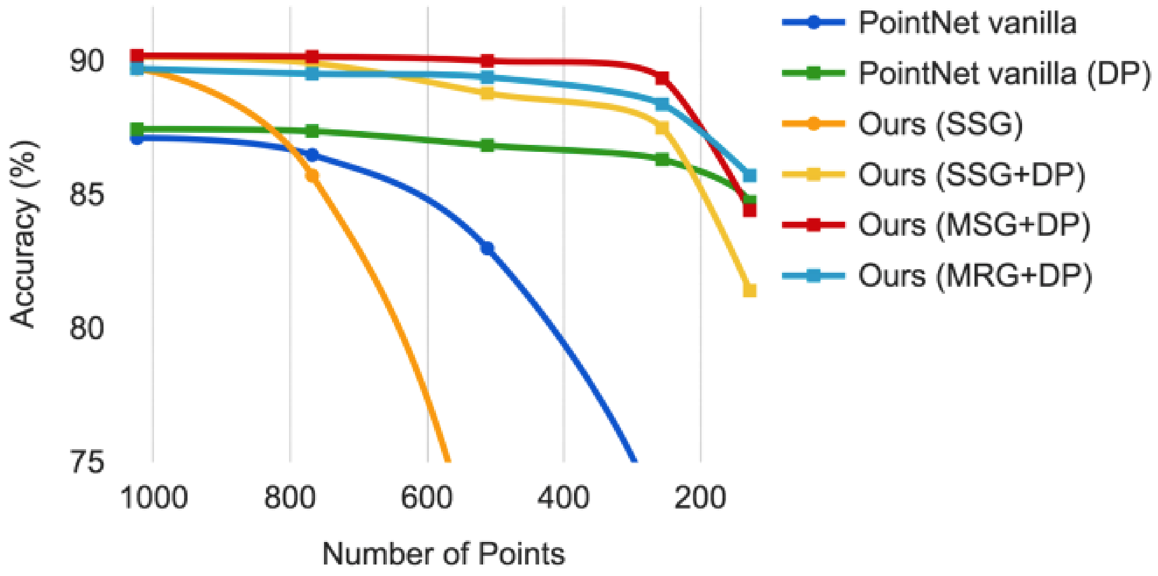
Table 2: ModelNet40 shape classification.

- MSG / MRG collects features from multiple scale / resolution
- Performs better with low intensity

Random Downsample on ModelNet40



Classification on Downsampled Objects



PointNet

- First method to process point cloud with deep learning
- Lack of hierarchical feature aggregation

PointNet++

- Hierarchical feature aggregation by repeating *sampling-grouping-PointNet*
- Significantly better performance
- Requires slightly more compute

Tricks – Data Augmentation

Normalization

- PointNet:
  - Normalized input to zero-mean
- PointNet++:
  - centered-with-node
  - zero-mean

Input Point Dropout

- E.g. Max input point number 5000, randomly dropout to [100, 5000] in each batch

Gaussian Noise

Rotation ???

- Less overfitting
- Worse performance
- Rotation equivariance / invariance is a research topic

Homework - Classification

- Classification over ModelNet40
  - Build your own network with pytorch
    - PointNet example: https://github.com/fxia22/pointnet.pytorch
  - ModelNet40 Dataset given by PointNet++:
    https://shapenet.cs.stanford.edu/media/modelnet40_normal_resampled.zip
  - Follow the training/testing split
  - Remember to add random rotation over z-axis
  - Report testing accuracy
- If you are not familiar with pytorch / deep learning
  - Simply run the open-source code to train / test on the ModelNet40 dataset.
  - Report the testing accuracy.
- Please write a report with screenshots to show the training/testing loss/accuracy curve.

Object detection pipeline for lidar

- Use KITTI 3D object detection dataset
- Step 1. Remove the ground from the lidar points
  - Any method you want – LSQ, Hough, RANSAC
- Step 2. Clustering over the remaining points
  - Any method you want
- Step 3. Classification over the clusters
- Step 4. Report the detection precision-recall for three categories: vehicle, pedestrian, cyclist (Next Lecture)

Step 3. Classification over the clusters

– vehicle / pedestrian / cyclist / other

- Step 3.1. Build dataset from KITTI 3D object detection dataset
  - Extract objects in the box for vehicle / pedestrian / cyclist
  - Other objects? Clustering!
- Step 3.2. Build deep network to classify them.
- Step 3.3. Report classification accuracy

Step 4. Report the detection Precision-Recall for three categories: vehicle, pedestrian, cyclist

- Fit a cuboid over object detected as vehicle / pedestrian / cyclist
- Generate object detection results for KITTI
- Evaluation code provided next Lecture.