SPATIAL REASONING (3)

OBJECT AND SCENE RECOGNITION

Dr TIAN Jing

tianjing@nus.edu.sg

Module objective

Knowledge and understanding

- Understand the fundamentals of spatial recognition: 3D data representation model, 3D scene classification and object segmentation in 3D data.

Key skills

- Perform RGB-D object detection
- Perform RGB-D scene segmentation
- Perform point cloud data classification

Module reference

- [Advanced] CVPR 2017 tutorial 3D deep learning, http://3ddl.stanford.edu/
- [Comprehensive] A collection on 3D machine learning, https://github.com/timzhang642/3D-Machine-Learning
- [Advanced] CS231A: Computer Vision, From 3D Reconstruction to Recognition, http://web.stanford.edu/class/cs231a/
- [Advanced] 3D vision, https://cvg.ethz.ch/teaching/3dvision/courseSchedule.php

Topics

- 3D data representation
- 3D data machine learning
  - RGB-D object detection
  - RGB-D semantic segmentation
  - Point cloud object classification

Introduction

- AR / VR: For sensing real 3D environments and reconstructing them in the virtual world. (Project Tango)
- Robotics: For navigation, localization, mapping, and avoiding collision.
- Facial recognition: For improving convenience while preventing fraud.
- Gesture and proximity detection: For gaming, security. (Kinect, RealSense)

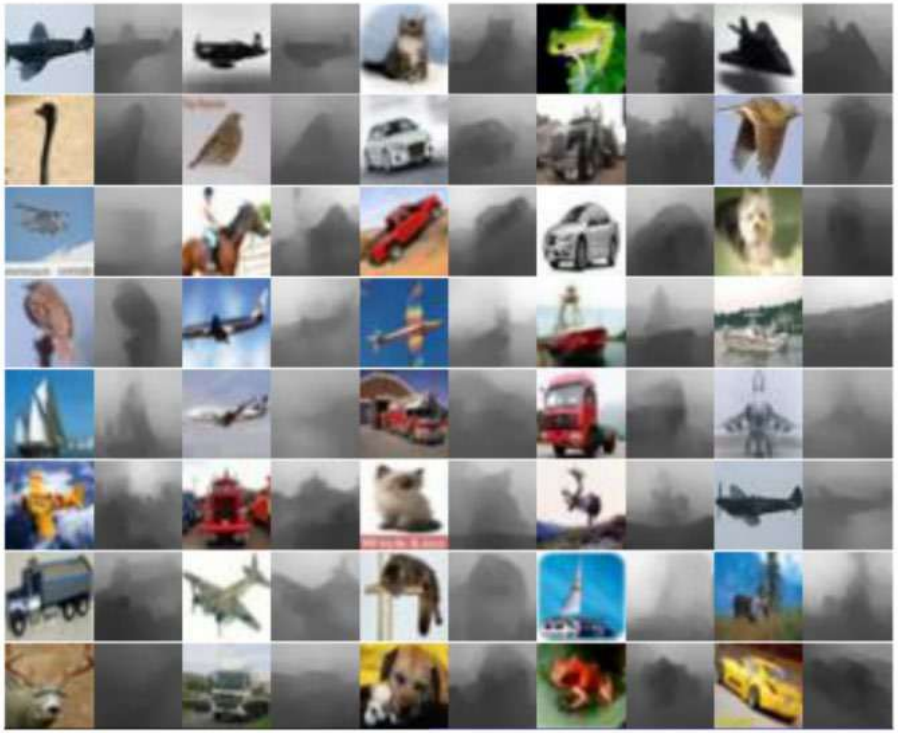
Reference: https://blog.cometlabs.io/depth-sensors-are-the-key-to-unlocking-next-level-computer-vision-applications-3499533d3246

Comparison of 3D imaging technologies

	Time of Flight (ToF)	Stereoscopic vision	Fixed structured light
Operational principle	**IR pulse, measure light transit time**	**Two 2D sensors emulate human eyes**	**Single pattern visible or IR illumination, detects distortion**
Point cloud generation	Direct out of chipset	High SW Processing	Medium SW processing
Latency	Low	Medium	Medium
Active illumination	Yes	No	Yes
Low light performance	Good	Weak	Good
Bright light performance	Medium	Good	Medium / weak *Depends on illumination power*
Power consumption	Medium/high *Scales w/ distance*	Low	Medium
Range	Short to long range *Depends on laser power & modulation*	Mid range *Depends on spacing between cameras*	Very short to mid range *Depends on illumination power*
Resolution	QQVGA, QVGA -> Roadmap to VGA	Camera Dependent	Projected pattern dependent
Depth accuracy	mm to cm *Depends on resolution of sensor*	mm to cm *Difficulty with smooth surface*	mm to cm
Scanning speed	Fast *Limited by sensor speed*	Medium *Limited by software complexity*	Medium *Limited by SW complexity*

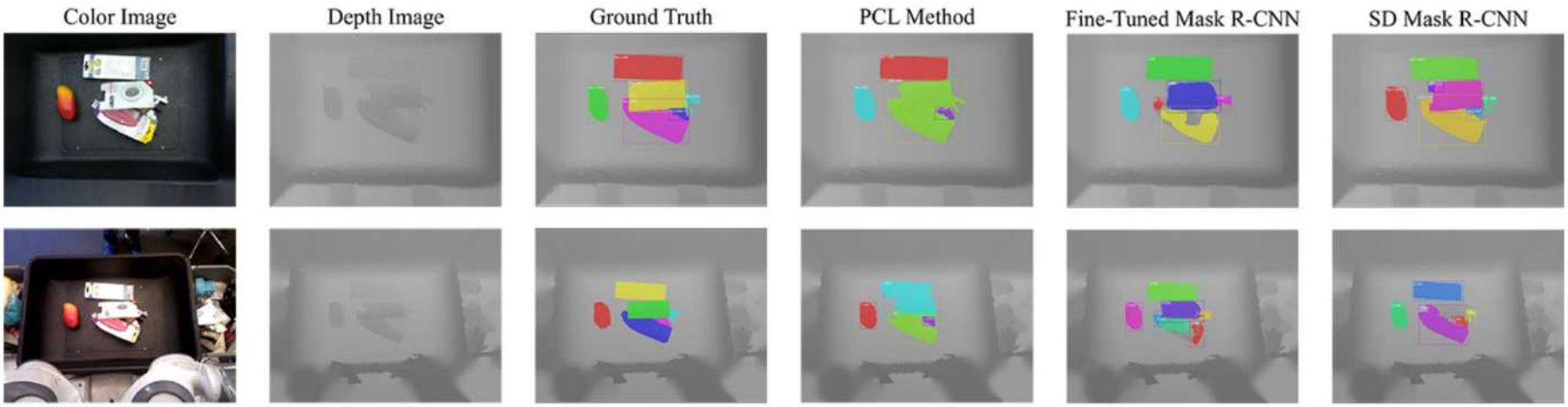
Depth can help us

RGB-D CIFAR10 dataset



- Detection/Segmentation: Depth information encodes the geometric cues necessary to separate object instances.
- Assist RGB image classification using RGB-D dataset.

Reference:
- https://bair.berkeley.edu/blog/2018/10/23/depth-sensing/
- Estimated depth map helps image classification, https://arxiv.org/abs/1709.07077



Introduction

Indoors, e.g., low-cost sensors such as the Microsoft Kinect

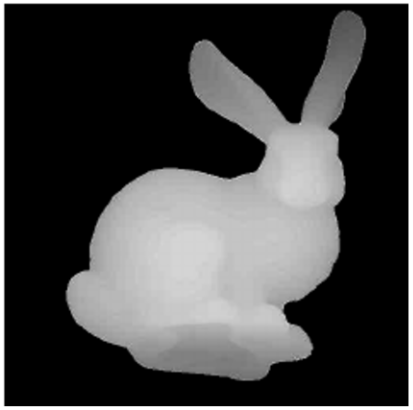
- Short range sensors for indoor scenes
  - Real-time 30 FPS depth maps based on structured light or time of flight in infrared
  - Integrated RGB camera is better aligned and provides better quality under indoor conditions than LiDAR systems
  - RGB-D image grid makes it well suited for traditional 2D computer vision techniques on image frames from a single view

Outdoors, e.g., LiDAR(Light Detection and Ranging)

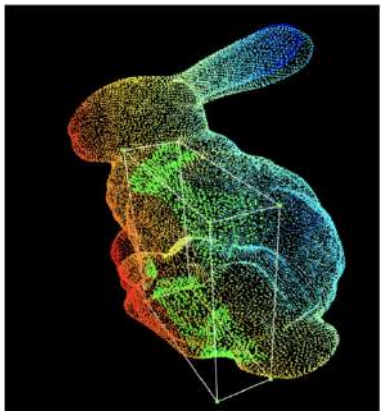
- Long range sensors for outdoor scenes
- Fast scans at low resolution or slow scans at high resolution, depending on number of individual sensors
- Moving sensors and registration from multiple scans result in unstructured point cloud data with no adjacency grid
- RGB imagery tends to be low quality, challenging to align, or simply unavailable

3D data representation

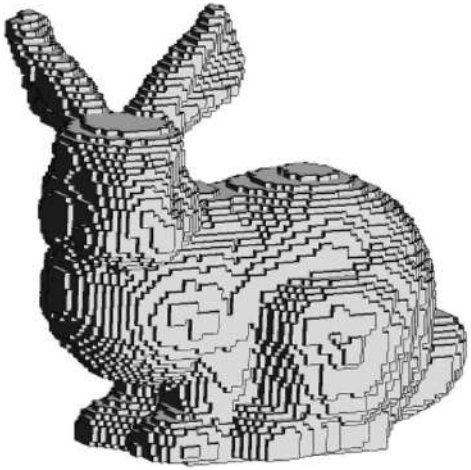
Rasterized form (regular grids)	Multi-view images	A 2D projected view of mesh
	Depth image	A 2.5D format
	Volumetric	A discrete version of point cloud
Geometric form (irregular)	Polygonal mesh	A surface version of point cloud
	Point cloud	3D points



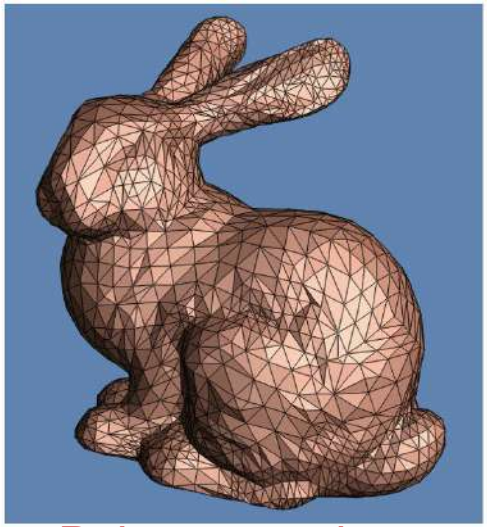
Depth map



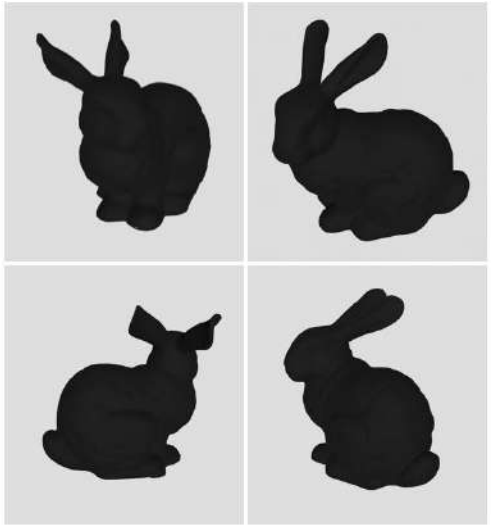
Point cloud



Volumetric



Polygon meshes



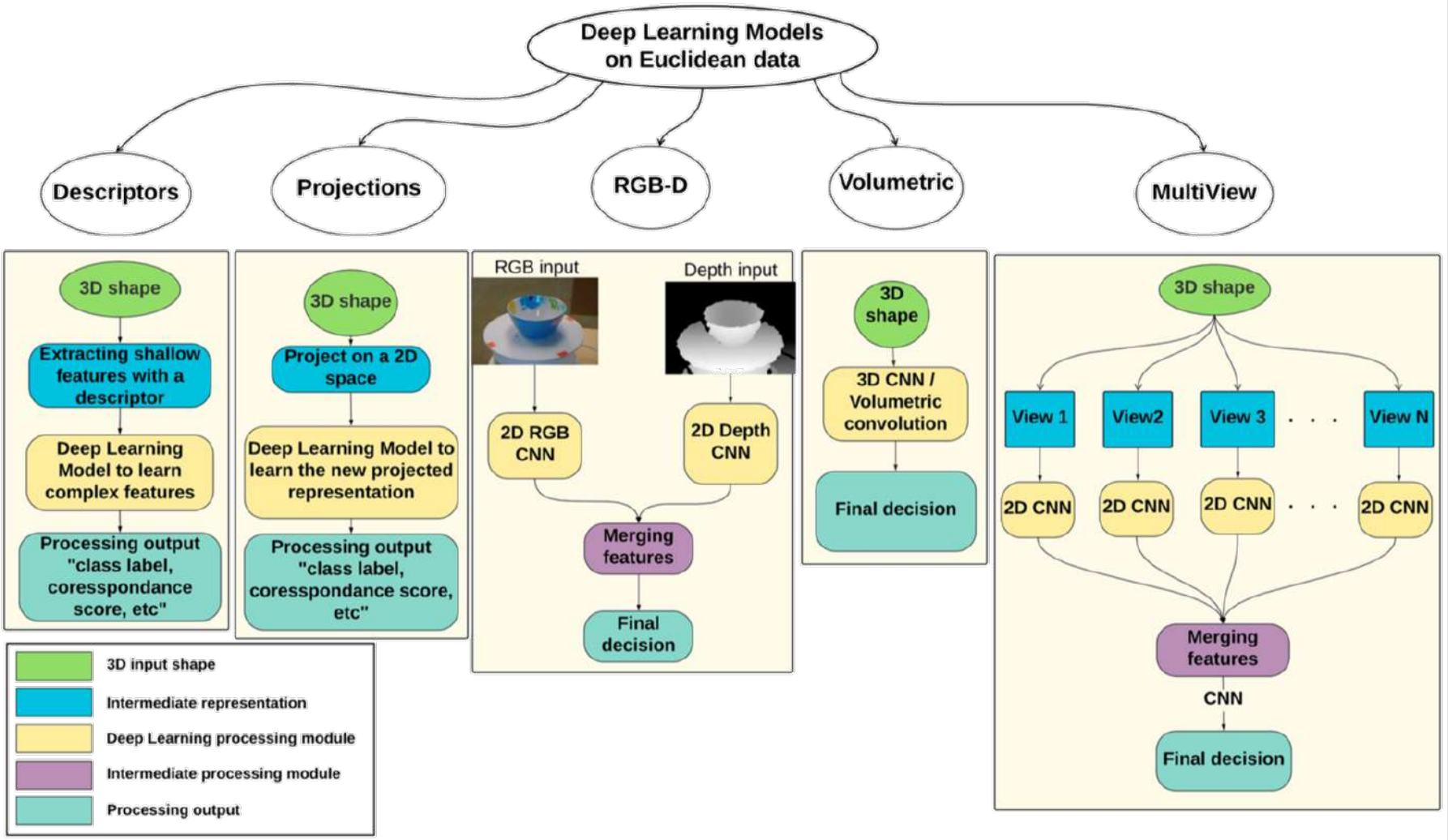
Multi-view images

Reference: 3D deep learning in geometric forms, https://cseweb.ucsd.edu/~haosu/slides/NIPS16_3DDL.pptx

Topics

- 3D data representation
- 3D data machine learning
  - RGB-D object detection
  - RGB-D semantic segmentation
  - Point cloud object classification

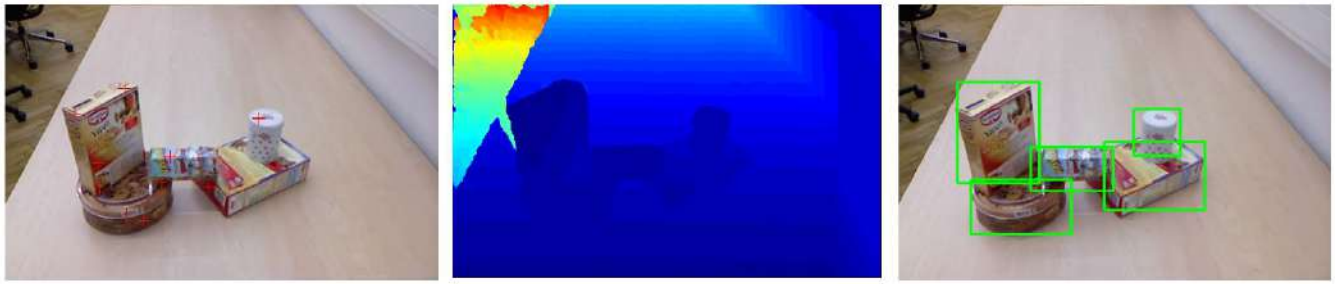
3D data machine learning



Reference: https://github.com/philastotle/pointnet-tutorial

Object detection

Objective: Perform object detection on a pair of RGB image and depth image,



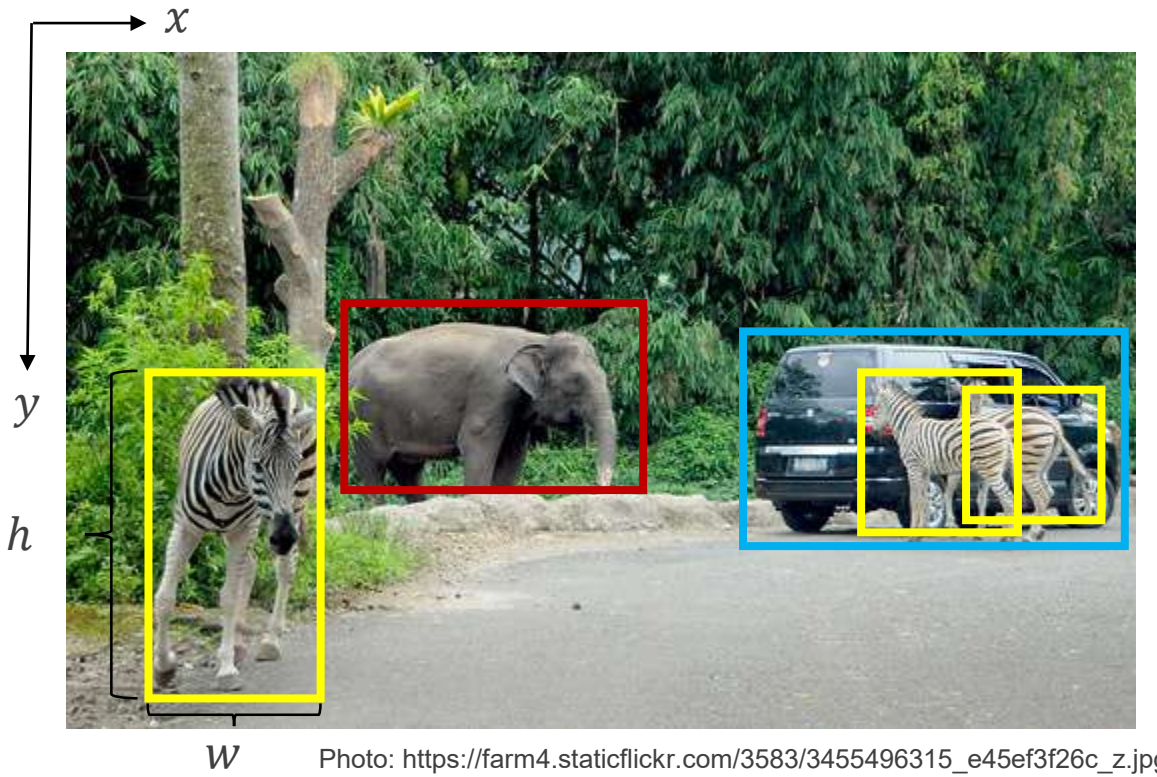


Photo: https://farm4.staticflickr.com/3583/3455496315_e45ef3f26c_z.jpg

We need to build a model that can output

- Box label (one-hot vector, a classification problem)
- Box coordinates (x, y, w, h) (a regression problem)
  - Column index of top-left corner x
  - Row index of top-left corner y
  - Width of box w
  - Height of box h

Sliding-window based RGB object detection

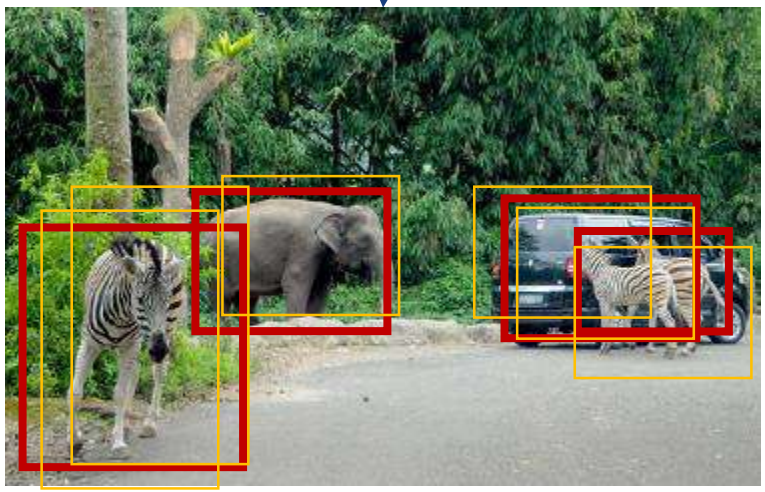
Select sliding window, crop & resize

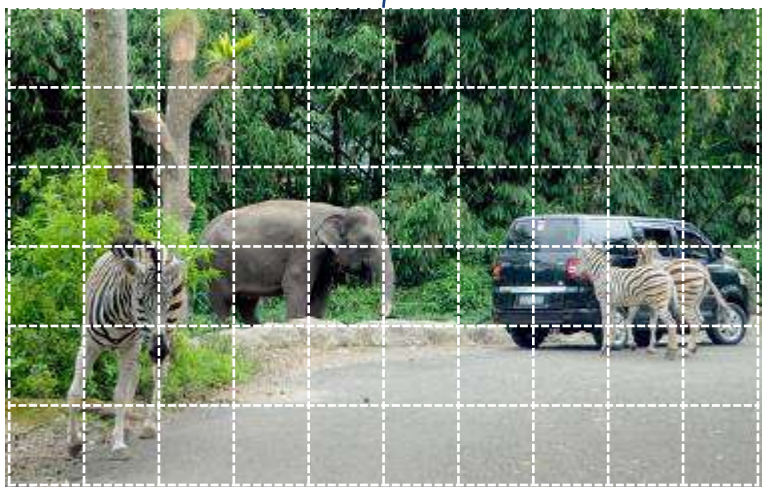
Apply the classifier on each sliding window

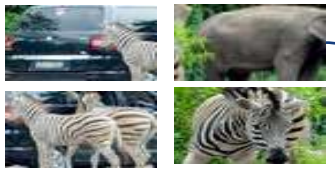
Output: A set of
- Box label and score
- Box coordinates (x, y, w, h)

Apply *Non-Maximum Suppression* (NMS) to select best (red) boxes

See next slide

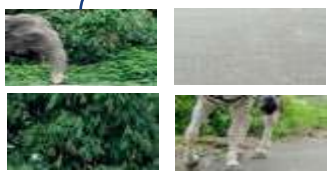






Train an object classifier

See next slide



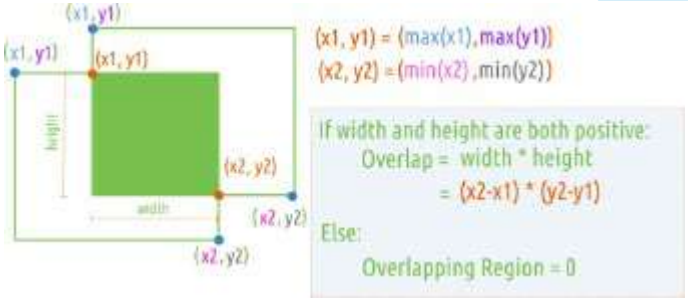
Object classifier

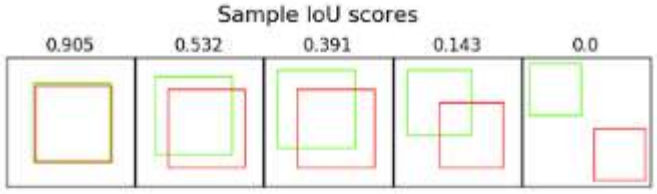
An object classifier

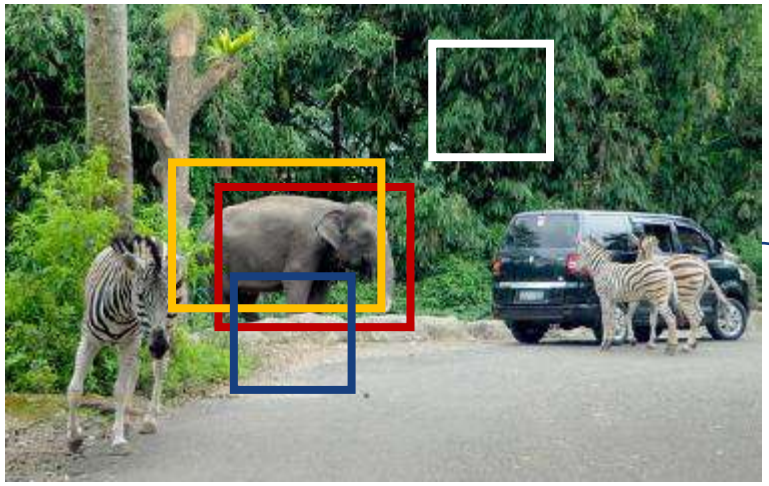
- Input: Image patches with the same resolution
- Output: Object/background labels with scores
- Training samples used for object categories: image patches with big overlapping with labelled ground-truth box.
- Training samples used for background category: image patches with small/no overlapping with labelled ground-truth box.

How to evaluate overlapping:
Intersection of Union (IOU)

$$IOU = \frac{Area\ of\ Overlap}{Area\ of\ Union}$$





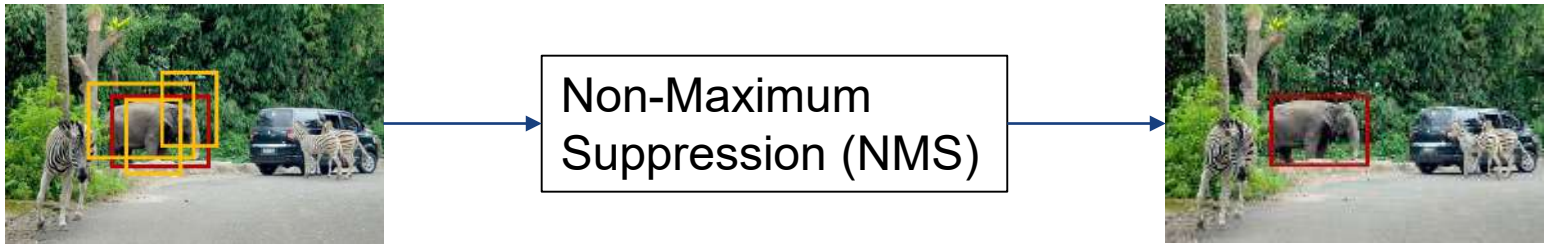


- Red box: labelled ground-truth box
- Yellow box: Elephant sample
- Blue box: Background sample
- White box: Background sample

Reference: http://ronny.rest/tutorials/module/localization_001/iou/

Non-maximum suppression (NMS)

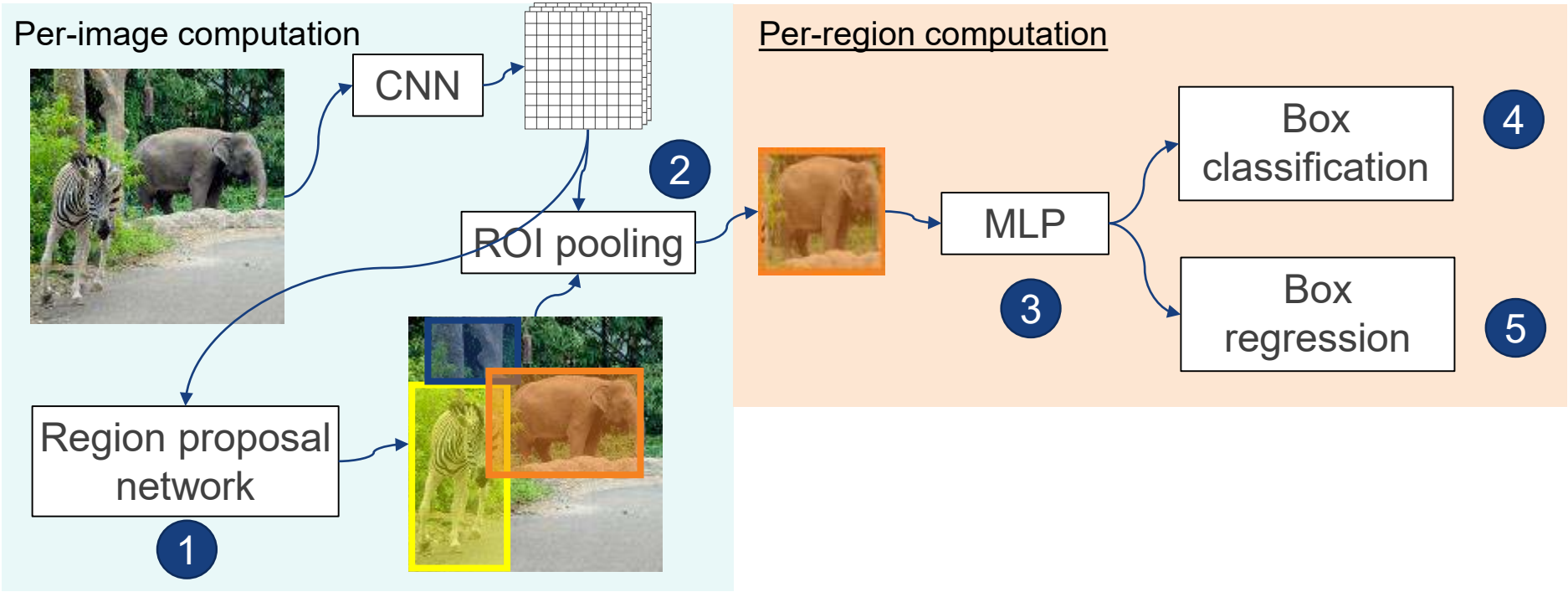
Non-Maximum Suppression (NMS): Select one entity (e.g. bounding boxes) out of many overlapping entities. The selection criteria is the overlap measure (e.g. IOU).



Input: A list of boxes B, corresponding scores S (from object classifier).
Output: A list of filtered boxes D.
Algorithm:

1. Select the box with highest confidence score, remove it from B and add it to the final proposal list D. (Initially D is empty).
2. Now compare this box with each of remaining boxes in B, calculate their IOU. If the IOU is greater than the (user-defined) threshold N, remove that box from B.
3. Again take the box with the highest confidence from the remaining boxes in B and remove it from B and add it to D.
4. Once again calculate the IOU of this box with all the remaining boxes in B and eliminate the boxes which have high IOU than threshold.
5. This process is repeated until there are no more boxes left in B.

Two-stage RGB object detection



1. Train a region proposal network to generate sliding window (proposals).
2. Select, crop each proposal window to obtain a fixed-size network input.
3. Forward propagate the fixed-size network input to get a feature representation.
4. Object classification.
5. Refine proposal localization with bounding-box regression

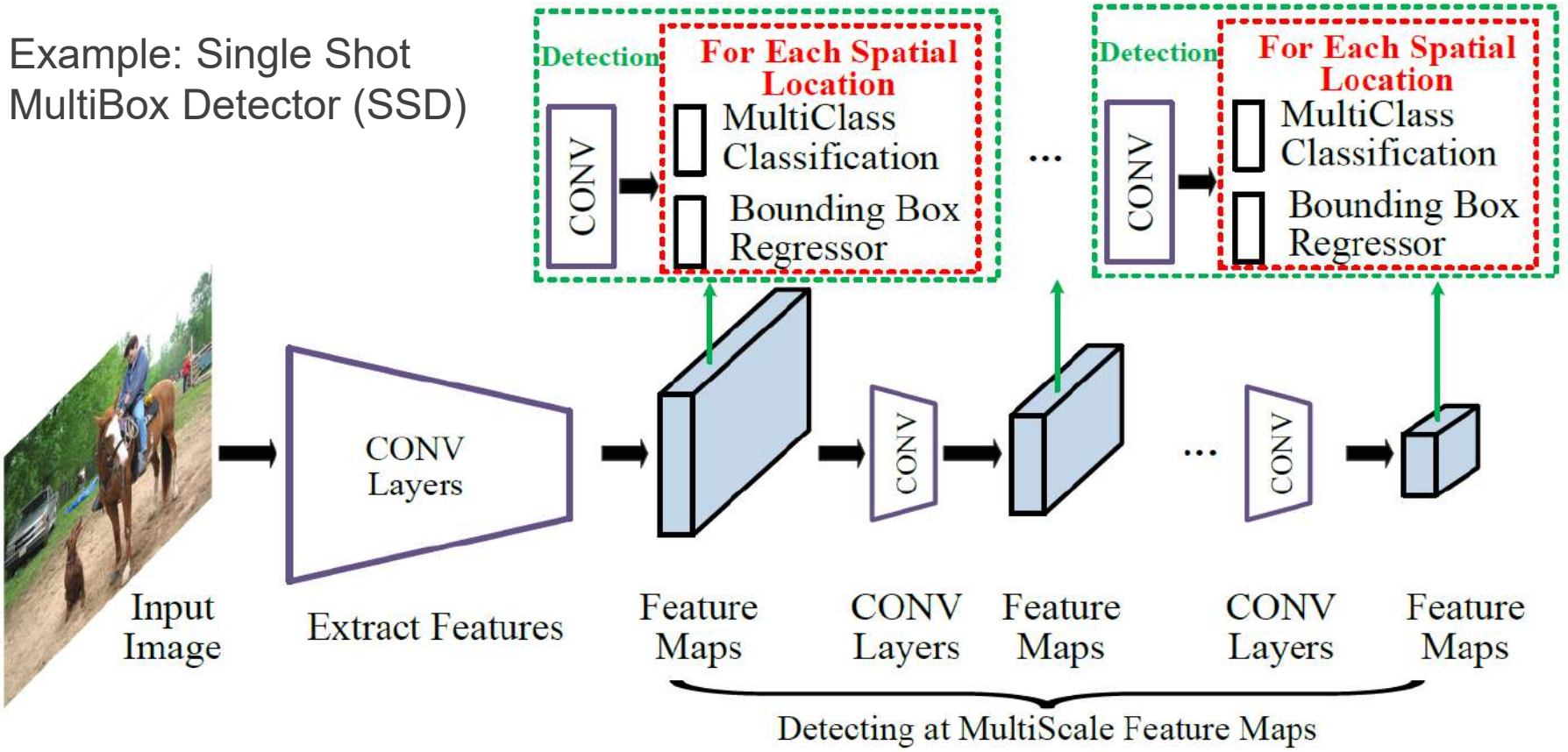
Reference: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, NIPS 2015, https://arxiv.org/abs/1506.01497

One-stage RGB object detection

Perform detection on the feature map

1. How to get feature map? → CNN backbone
2. Where to check on feature map? → Anchor box
3. Multi-scale object in image? → Anchor box at different (scaled) feature map

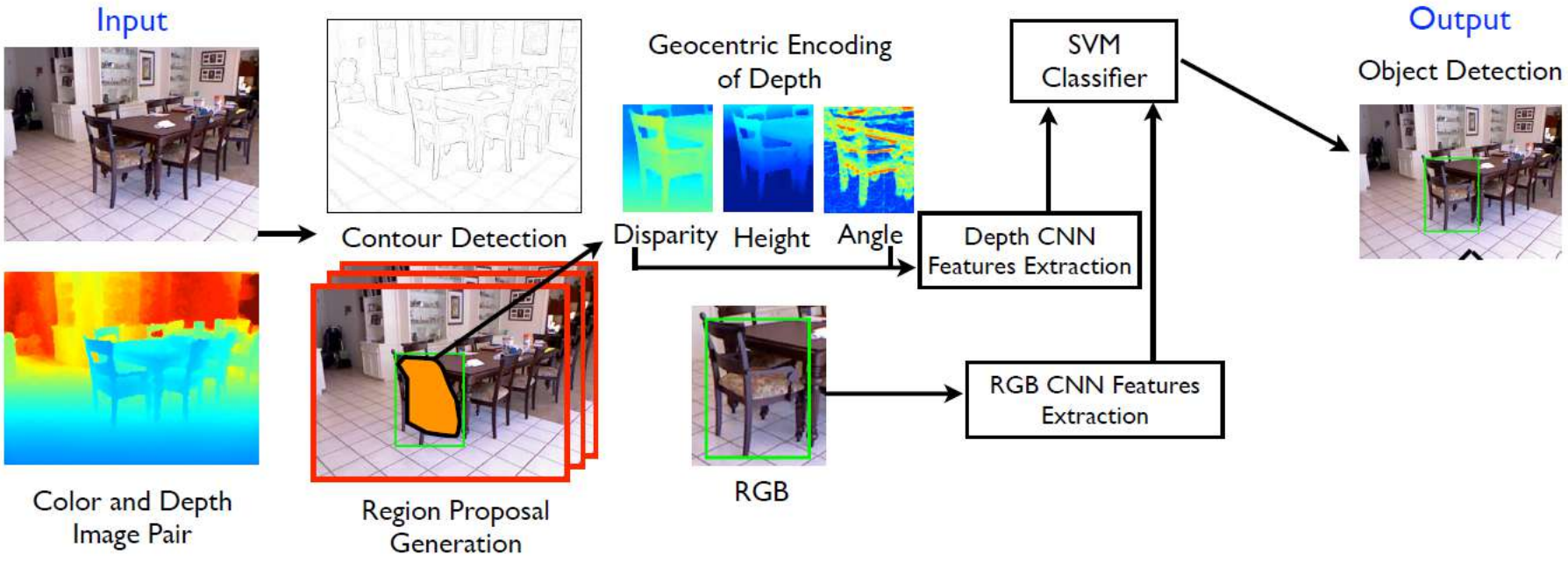
Example: Single Shot MultiBox Detector (SSD)



Reference: SSD: Single Shot MultiBox Detector, ECCV 2016, https://arxiv.org/abs/1512.02325

RGB-D data: Object detection

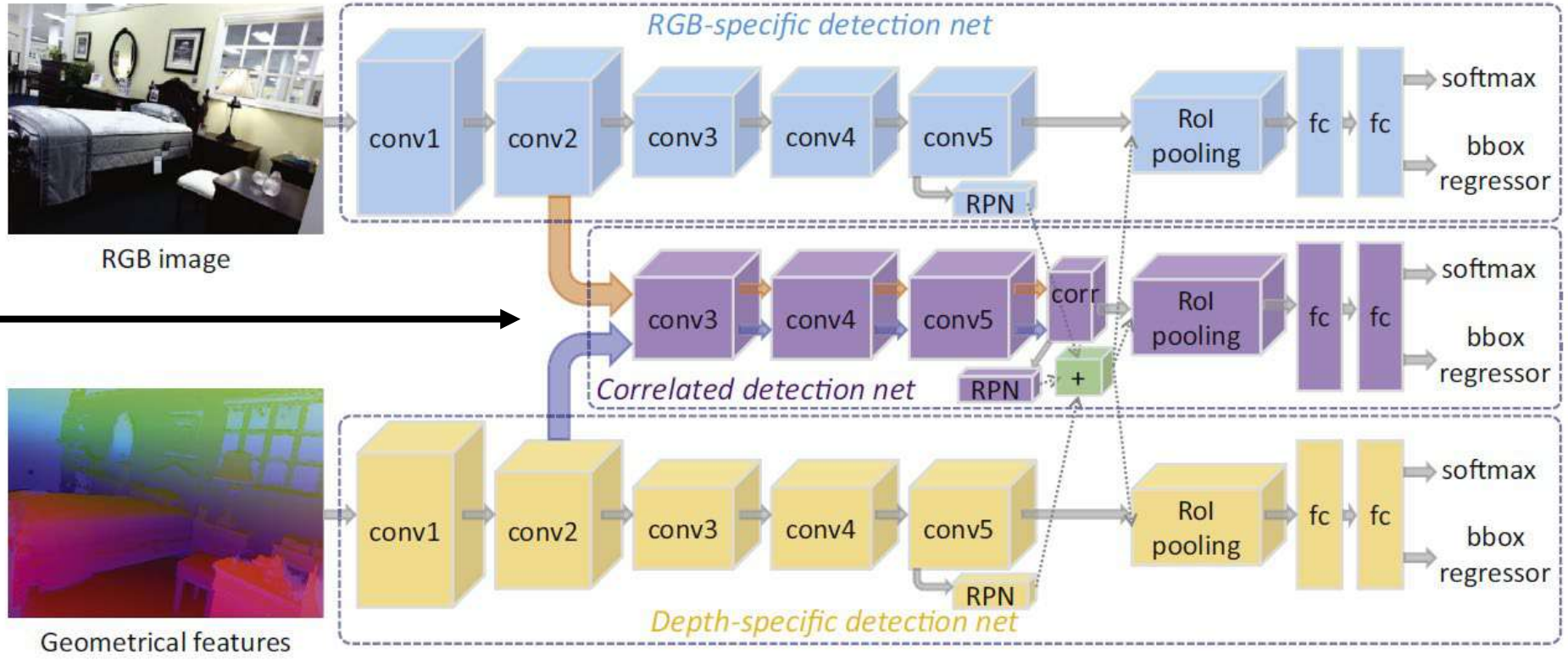
Conventional approach
- Sliding window on input pair of RGB-D images
- Region proposal (optional)
- Feature extraction and classification



Reference: Learning rich features from RGB-D images for object detection and segmentation, ECCV 2014, https://arxiv.org/abs/1407.5736

RGB-D data: Object detection

Objective: How to add depth image as input with RGB input image together?

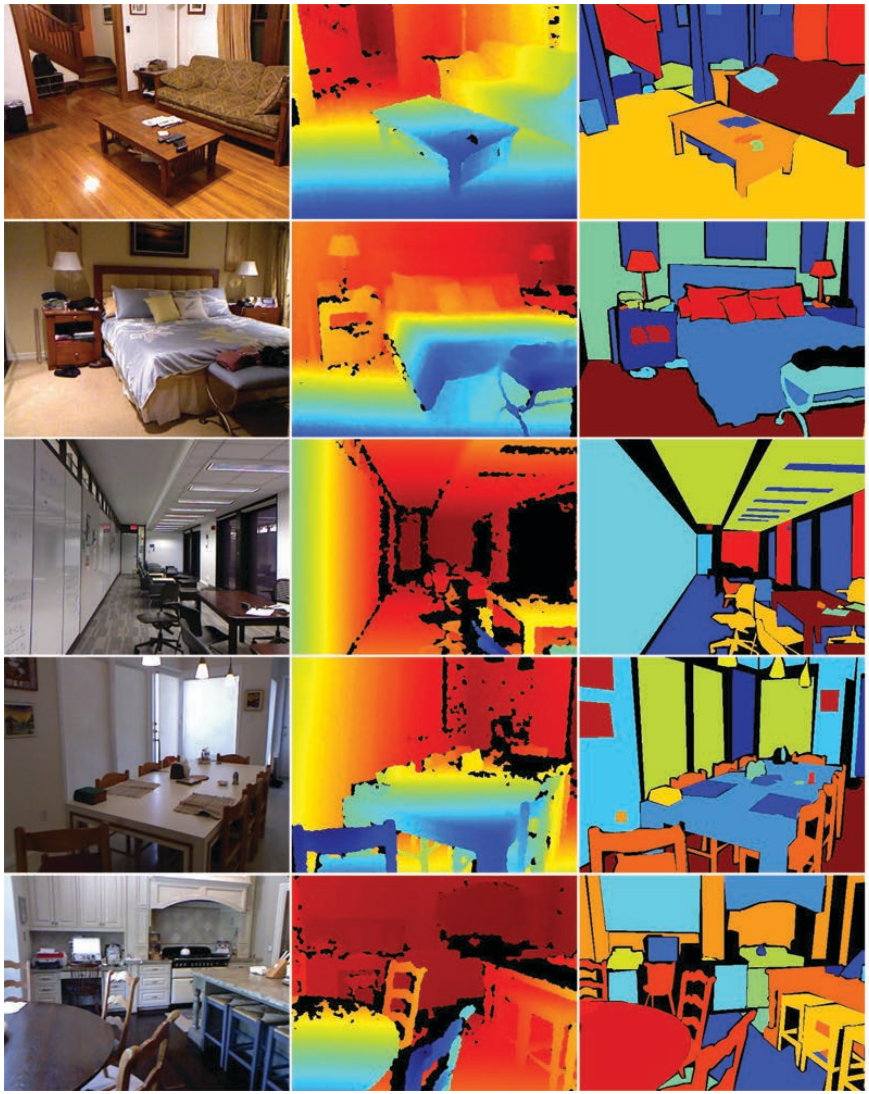


- Shared convolution layers ($conv3$, $conv4$, $conv5$)
- $corr$ means a parameter-free correlation layer, which performs multiplicative comparisons (element-wise product, Hadamard product) between feature maps of two modalities.

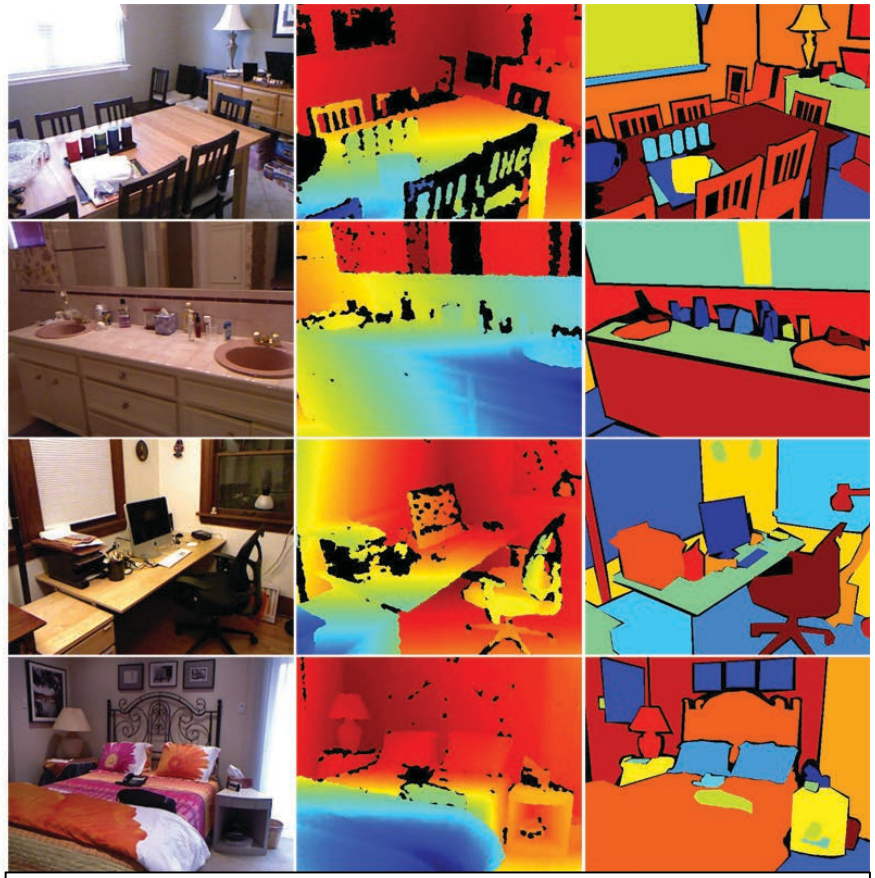
Reference: Multi-modal deep feature learning for RGB-D object detection, https://par.nsf.gov/servlets/purl/10073960

RGB-D data: Semantic segmentation

RGB Depth Ground truth



RGB Depth Ground truth



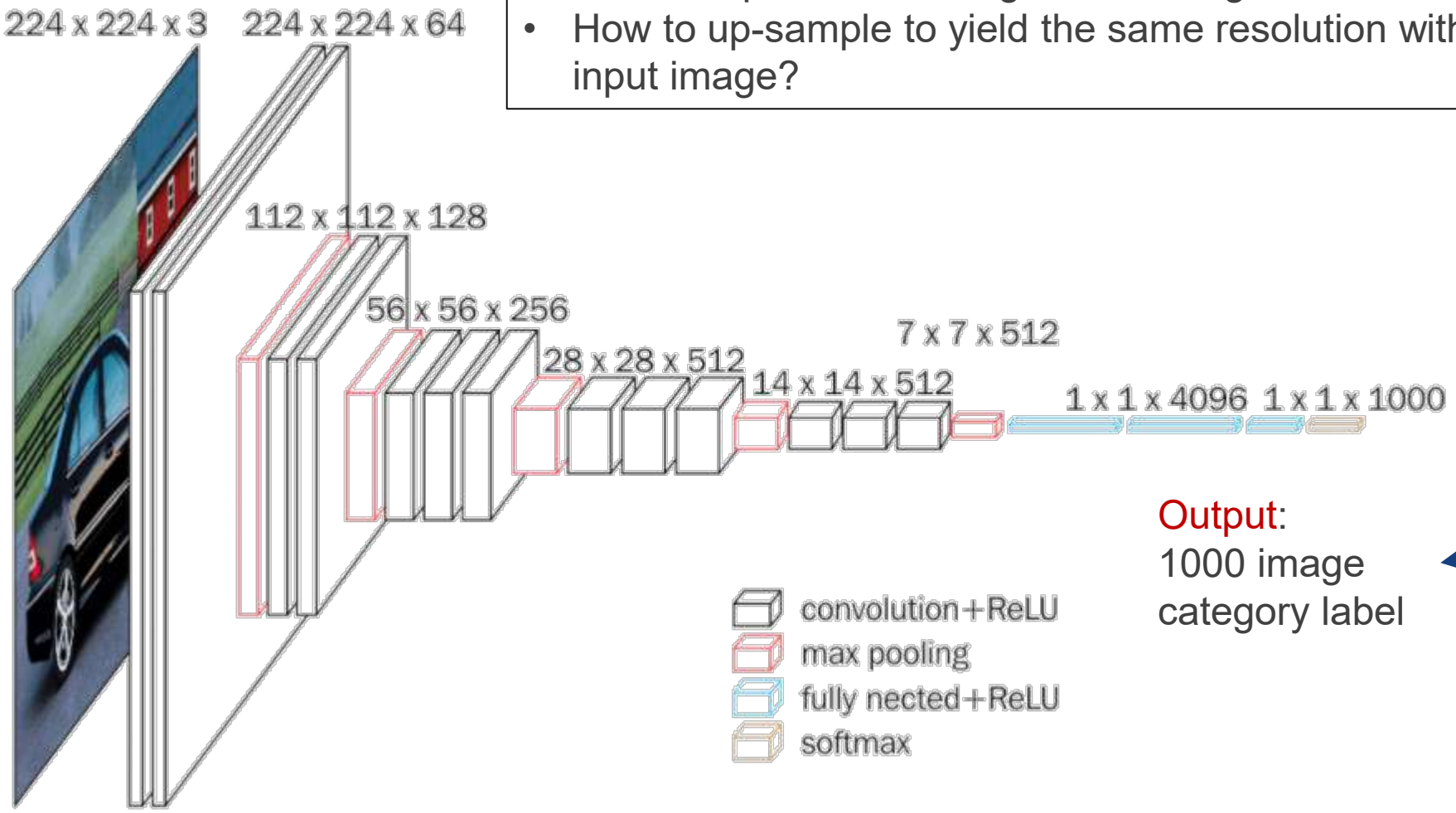
- Label each pixel in the image with a category label
- Don't differentiate instances, only care about pixels

Reference: https://cs.nyu.edu/~silberman/datasets/nyu_depth_v2.html

Recall: Classification network

Input: A color image

Challenges to achieve prediction at each pixel.
- How to represent the target label image?
- How to up-sample to yield the same resolution with input image?



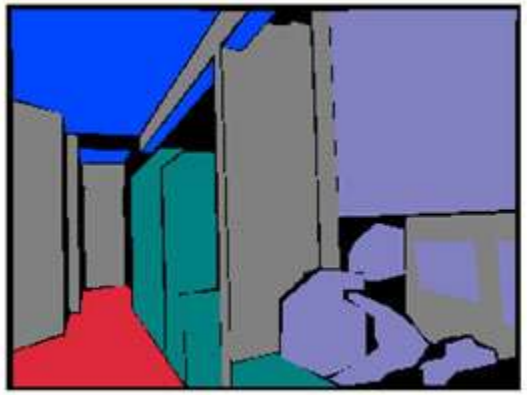
Output:
1000 image category label

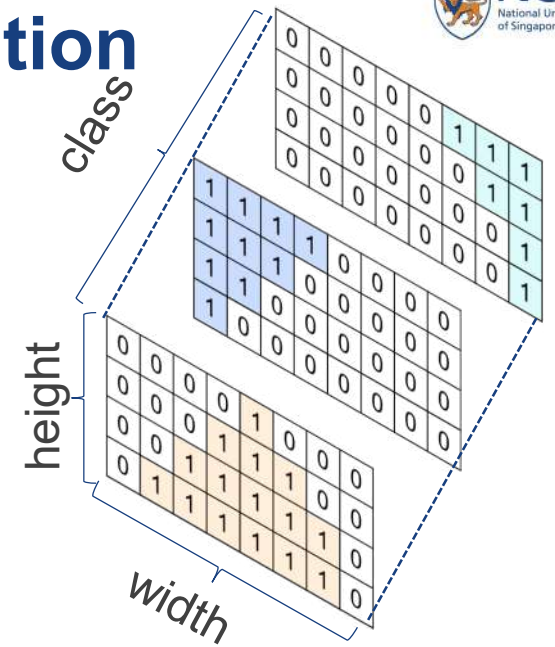
Note: A VGG model is used here for demonstration.

Image: https://neurohive.io/en/popular-networks/vgg16/

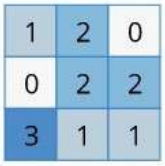
Recall: Semantic segmentation

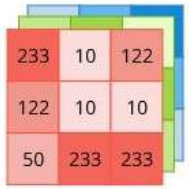
How to represent the target label image?

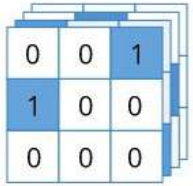




1	bed
2	books
3	ceiling
4	chair
5	floor
6	furniture
7	objects
8	picture
9	sofa
10	table
11	tv
12	wall
13	window





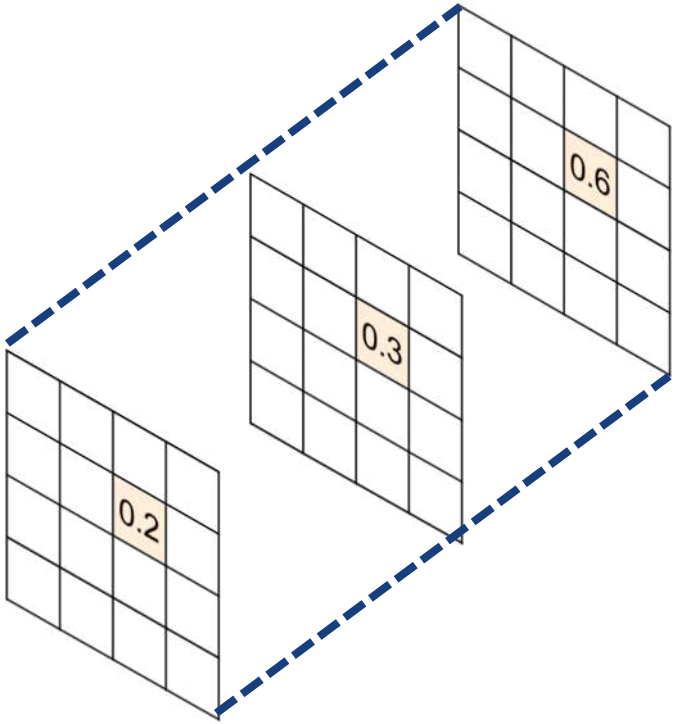


- Greyscale images: The pixel intensity represents the class id.
- Color images: Different classes of objects will have some arbitrary RGB color for visualization. It requires pre-processing to be converted into either class labels, or One Hot Vector representations, before it can be used in a deep learning pipeline.
- One hot vectors: Each pixel is encoded as a one-hot vector, with a value of 1 for the class it represents.

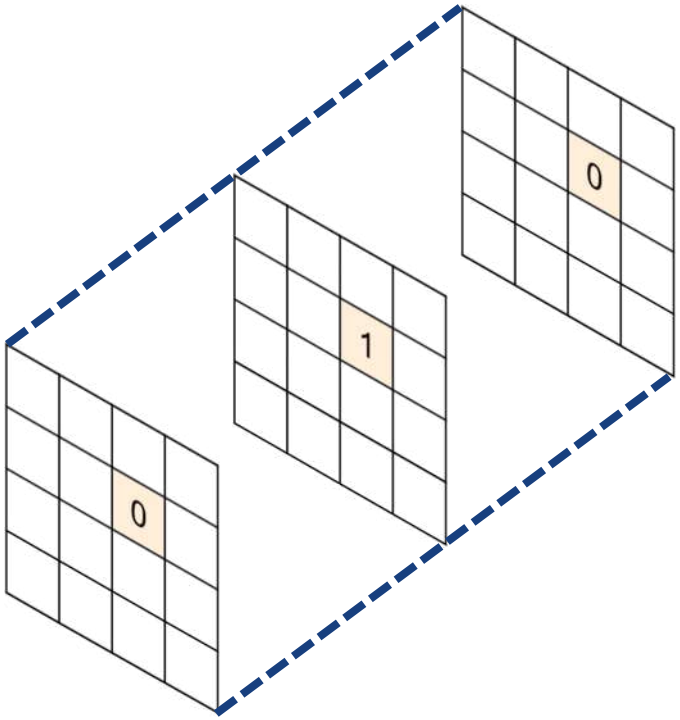
Reference: https://www.jeremyjordan.me/semantic-segmentation/

Performance metric

- Pixel-wise cross-entropy loss is calculated based on model prediction (e.g., $[0.2, 0.3, 0.6]$) and the ground truth one hot vector (e.g., $[0, 1, 0]$).
- This calculation is repeated over all pixels and averaged.



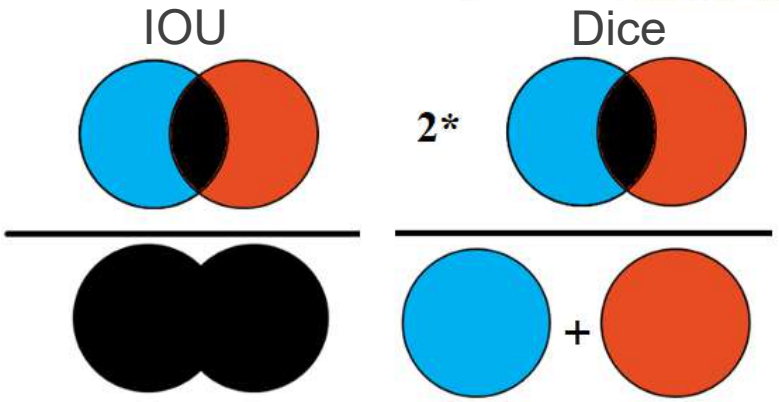
Prediction for a selected pixel from a machine learning model



Ground truth for the corresponding pixel (one hot vector)

Performance metric

- IOU (intersection over union): Overlapping region divided by combine region.
- Dice coefficient: $\frac{2|A \cap B|}{|A|+|B|}$, where $|A \cap B|$ represents the common elements between regions A and B, $|A|$ and $|B|$ represent the number of elements in set A and set B, respectively.



$$\text{Dice loss} = 1 - \frac{2\sum_{\text{pixels}}\left(y_{\text{true}} * y_{\text{pred}}\right)}{\sum_{\text{pixels}}(y_{\text{true}}) + \sum_{\text{pixels}}\left(y_{\text{pred}}\right)}$$

Example

$$A = \begin{bmatrix} 0.01 & 0.03 & 0.02 & 0.02 \\ 0.05 & 0.12 & 0.09 & 0.07 \\ 0.89 & 0.85 & 0.88 & 0.91 \\ 0.99 & 0.97 & 0.95 & 0.97 \end{bmatrix} \quad |A \cap B| = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.89 & 0.85 & 0.88 & 0.91 \\ 0.99 & 0.97 & 0.95 & 0.97 \end{bmatrix} \xrightarrow{\text{sum}} 7.41$$

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \quad |A| = \begin{bmatrix} 0.01 & 0.03 & 0.02 & 0.02 \\ 0.05 & 0.12 & 0.09 & 0.07 \\ 0.89 & 0.85 & 0.88 & 0.91 \\ 0.99 & 0.97 & 0.95 & 0.97 \end{bmatrix} \xrightarrow{\text{sum}} 7.82 \quad |B| = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \xrightarrow{\text{sum}} 8$$

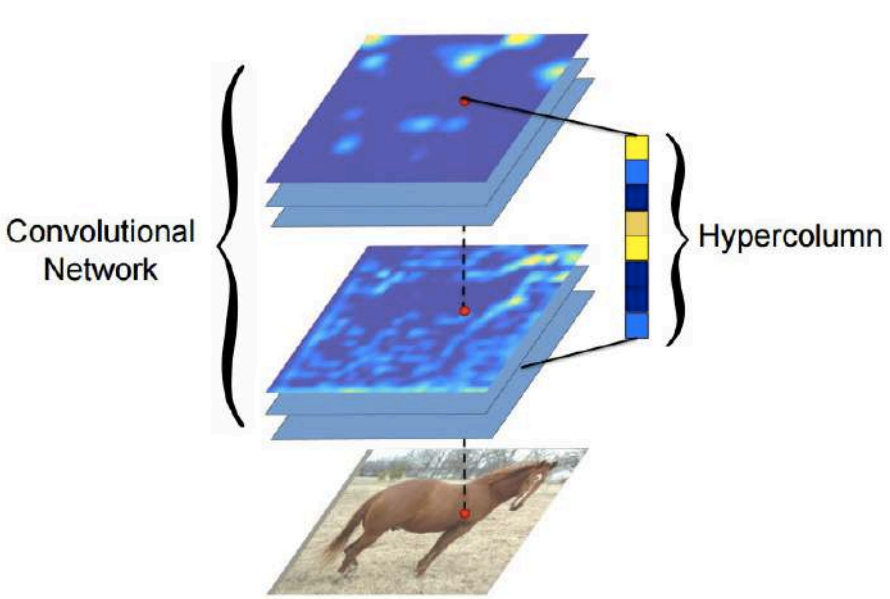
$$\text{Dice loss} = 1 - \frac{2|A \cap B|}{|A| + |B|} = 1 - \frac{2 * 7.41}{7.82 + 8} = 0.0632$$

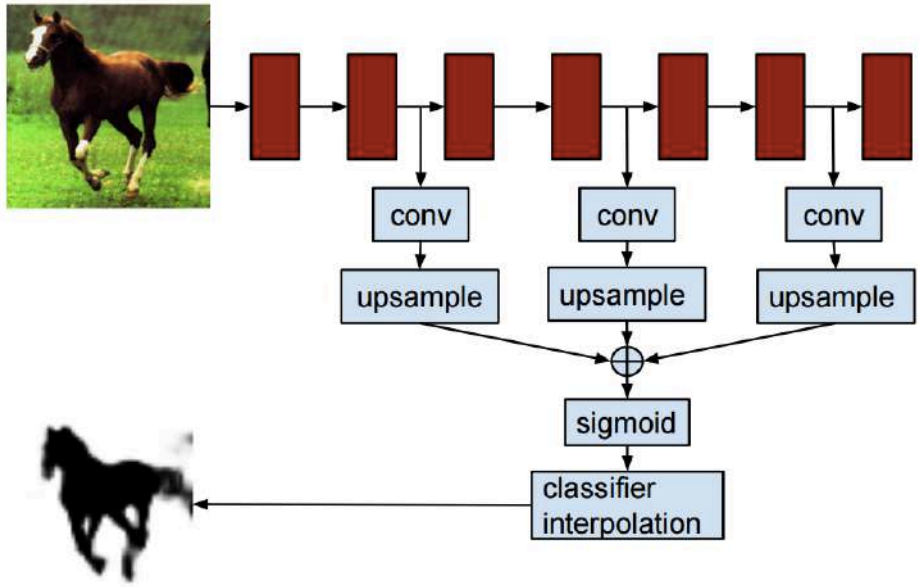
Note: For multiple classes, the Dice loss score is repeated over all pixels/classes and averaged.

Semantic segmentation

How to up-sample to yield the same resolution with input image?

- Idea: To obtain a feature representation for an individual pixel, upsample all feature maps to original image resolution and concatenate values from feature maps “above” that pixel.

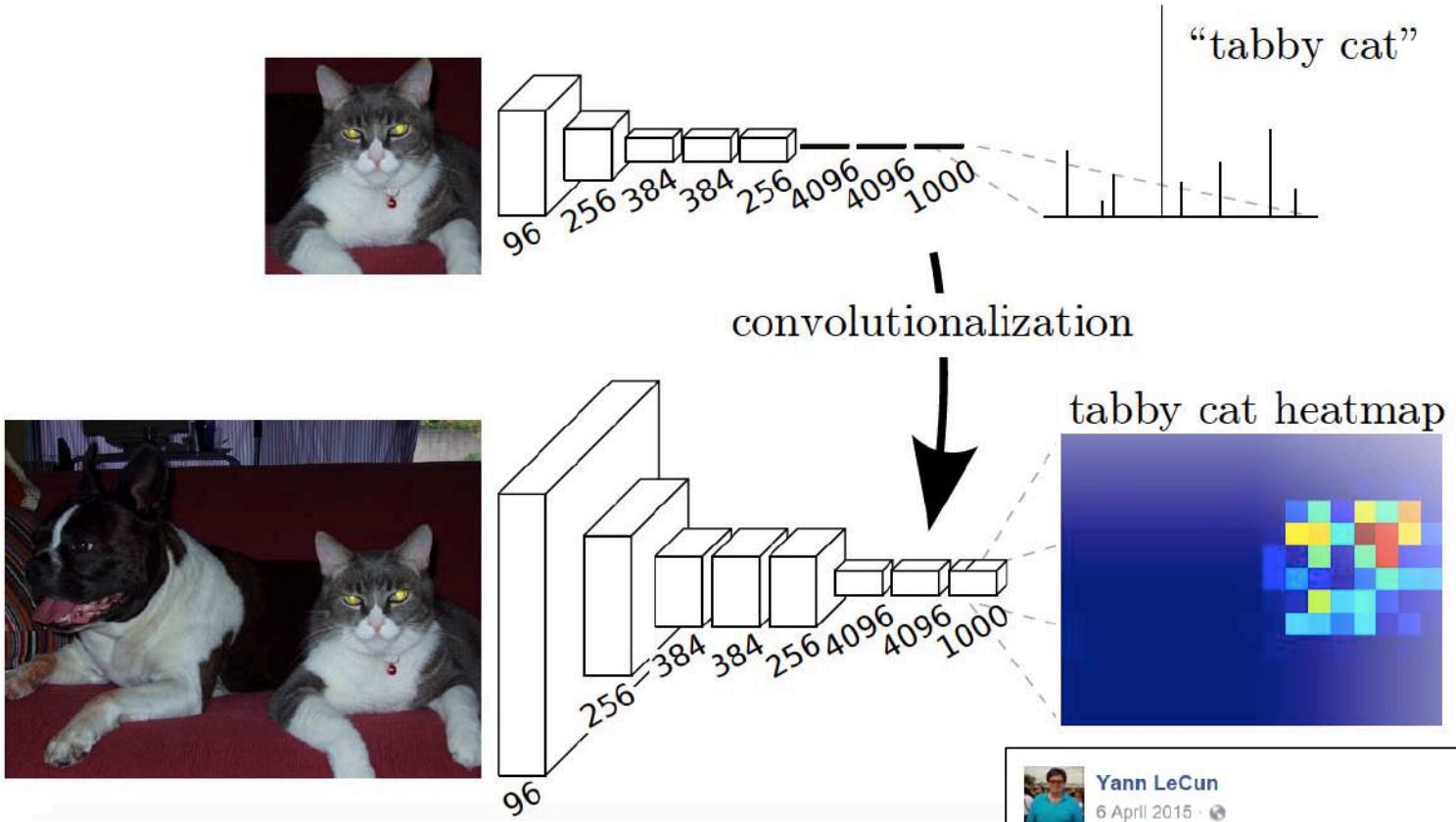




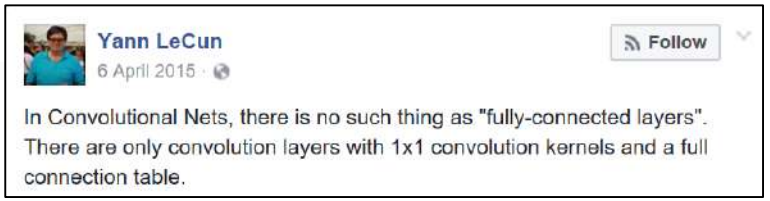
Reference: J. Long, E. Shelhamer, and T. Darrell, Fully Convolutional Networks for Semantic Segmentation, CVPR 2015, https://people.eecs.berkeley.edu/~jonlong/long_shelhamer_fcn.pdf

Fully convolutional networks

- Design a network with only convolutional layers, make predictions for all pixels at once



Take input of any size and output pixel classification maps



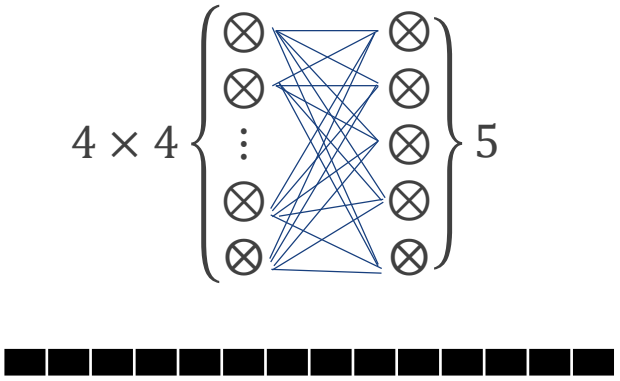
Reference: J. Long, E. Shelhamer, and T. Darrell, Fully Convolutional Networks for Semantic Segmentation, CVPR 2015, https://people.eecs.berkeley.edu/~jonlong/long_shelhamer_fcn.pdf

Fully convolutional networks

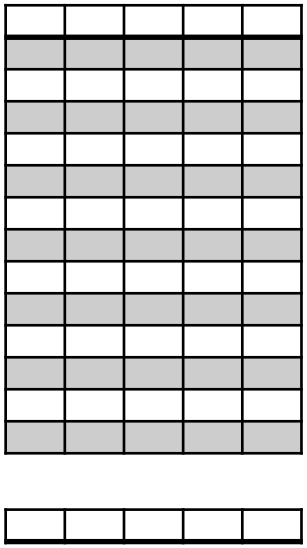
Converting FC layers to CONV layers

- The only difference between FC and CONV layers is that the neurons in the CONV layer are connected only to a local region in the input, and that many of the neurons in a CONV volume share parameters. However, the neurons in both layers still compute dot products, so their functional form is identical.
- Any FC layer can be converted to a CONV layer. For example, an FC layer with $K = 4096$ that is looking at some input volume of size $7 \times 7 \times 512$ can be equivalently expressed as a CONV layer with $F = 7,\ P = 0,\ S = 1,\ K = 4096$. In other words, we are setting the filter size to be exactly the size of the input volume, and hence the output will simply be $1 \times 1 \times 4096$ since only a single depth column "fits" across the input volume, giving identical result as the initial FC layer.

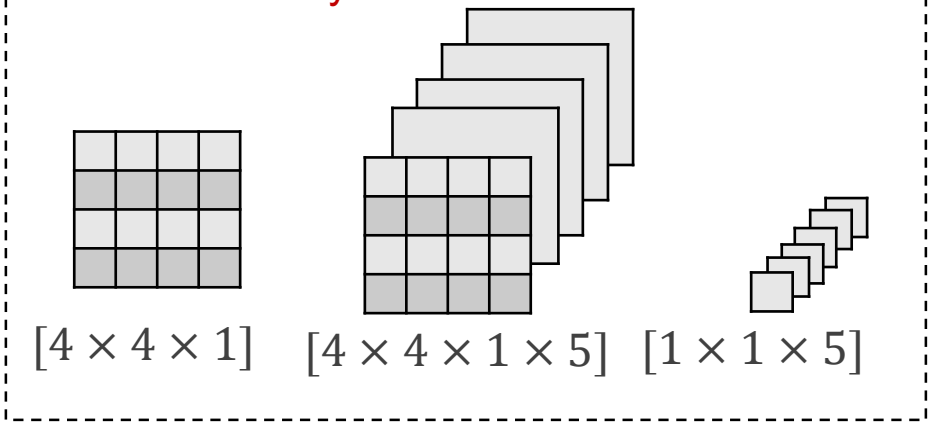
Example: Fully-connected layer



Mathematical calculations

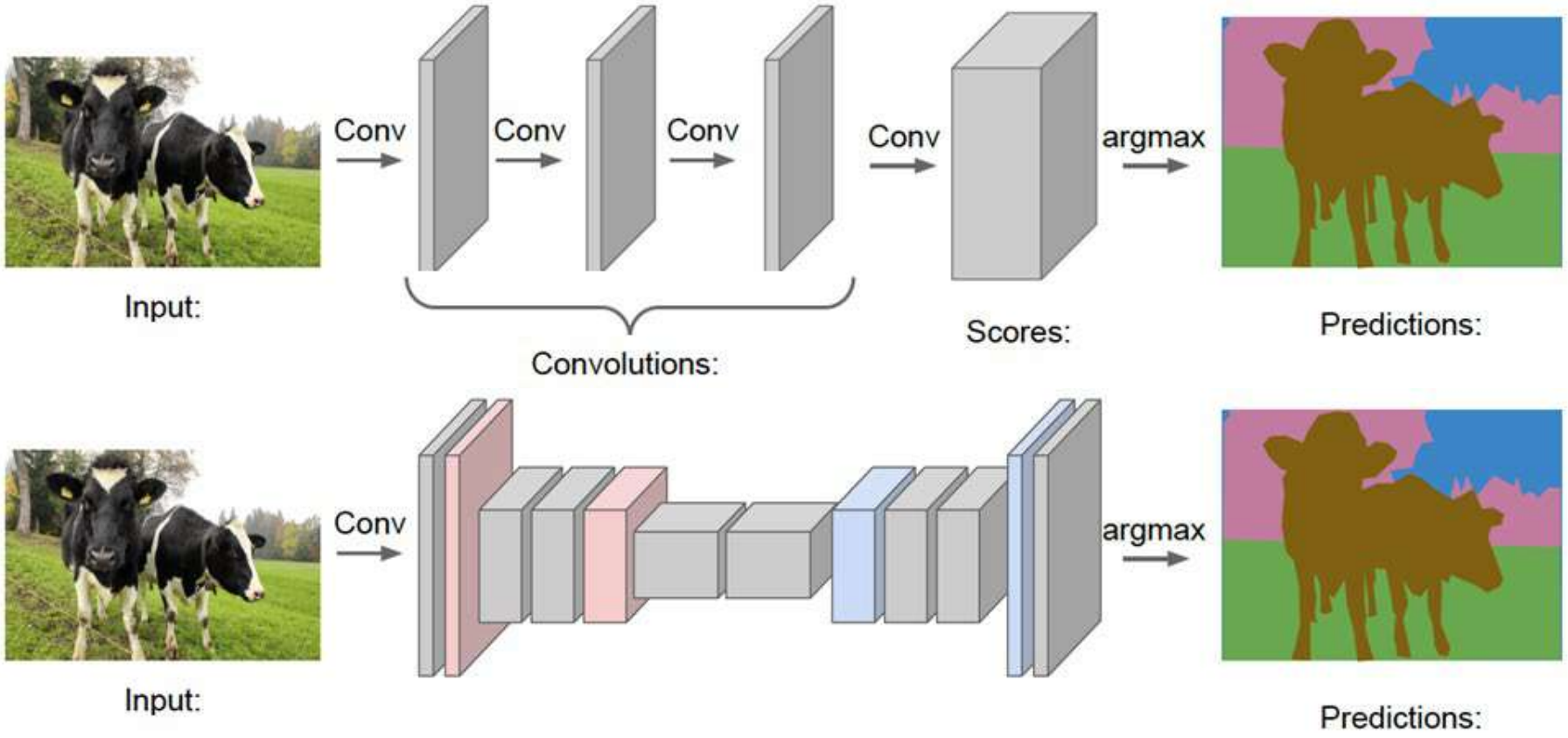


Convolutional layer



Fully convolutional networks

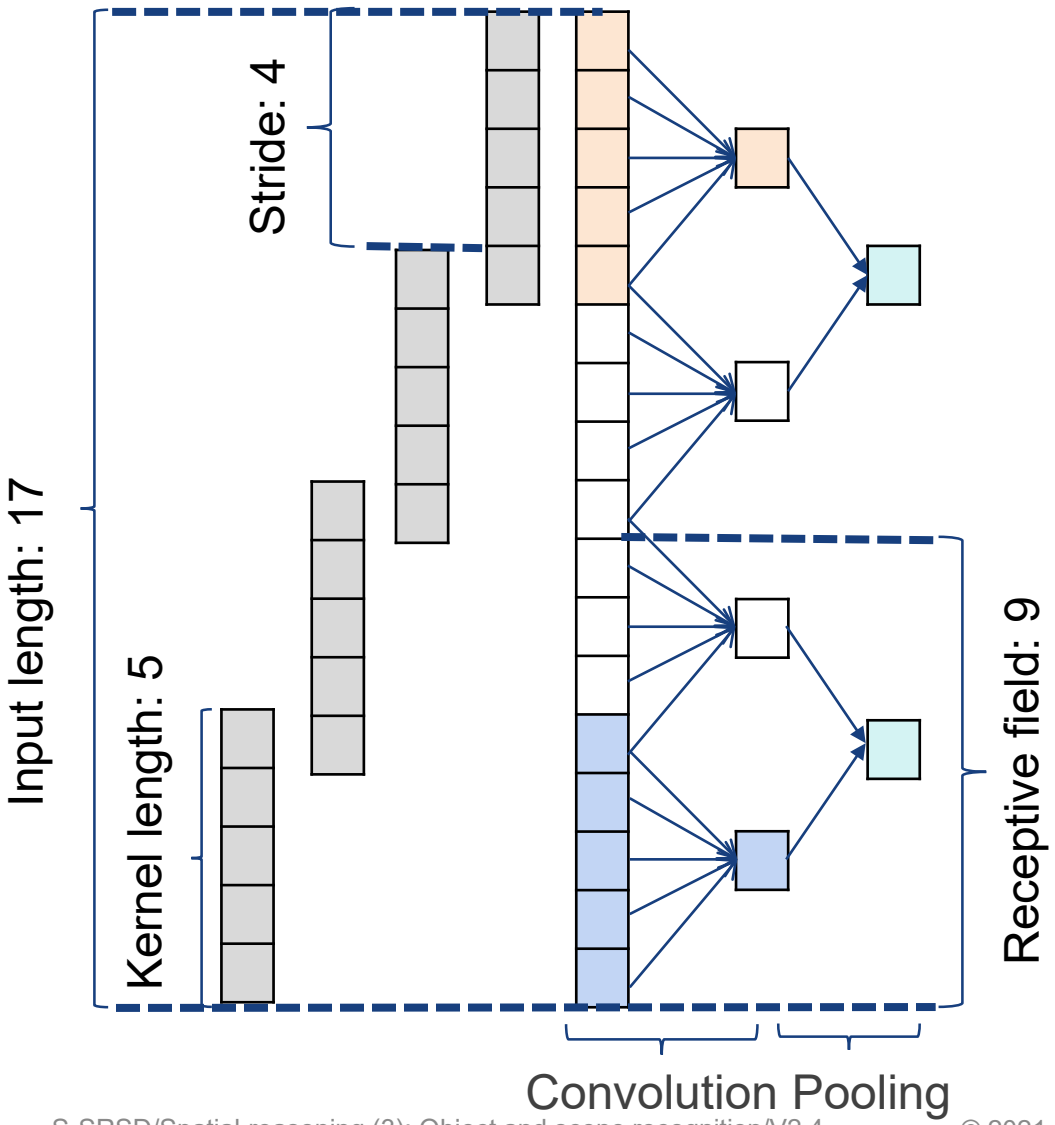
- To design a network with only convolutional layers, make predictions for all pixels at once. Ideally, we want convolutions at full image resolution.
- Idea 1: First down-sample, then up-sample. (Example: U-Net)



Reference: Lecture 11, segmentation, http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture11.pdf

Fully convolutional networks

Idea 2: Stride convolution + pooling



Stride convolution

- Output size: $\left(\frac{L-K+2P}{S}+1\right) \times N$
- Trainable parameters: $(K+1) \times N$

L	Input length	17
K	Kernel length	5
P	Padding	0
S	Stride (sliding kernel)	4
N	Number of kernels	1

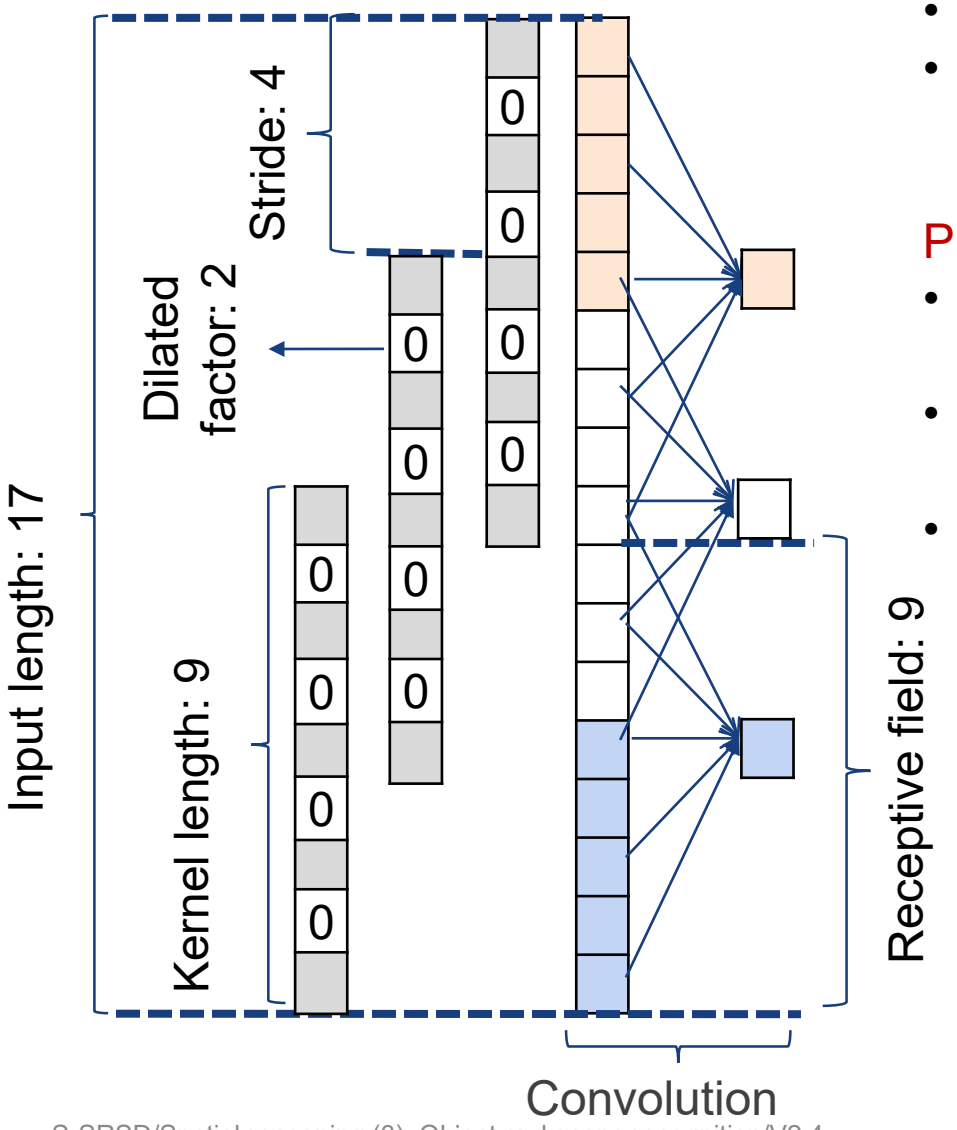
Pooling

- Output size: $\frac{L-D}{D}+1$
- There is no trainable parameter

L	Input length	4
D	Pooling (down-sampling) factor	2

Fully convolutional networks

Idea 3: Dilated convolution



Dilated convolution

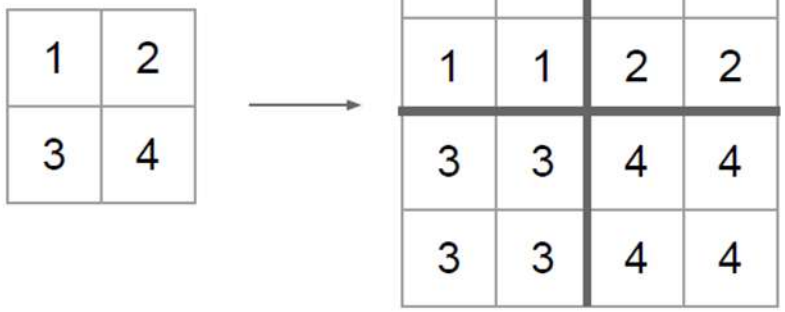
- Widen receptive field effectively
- Avoid spatial resolution coarsening (in stride convolution or conventional pooling)

Process

- First, it expands (dilates) the convolution filter according to the dilation rate.
- Second, it fills the empty spaces with zeros, creating a sparse like filter.
- Finally, it performs regular convolution using the dilated filter.

Up-sampling: Unpooling

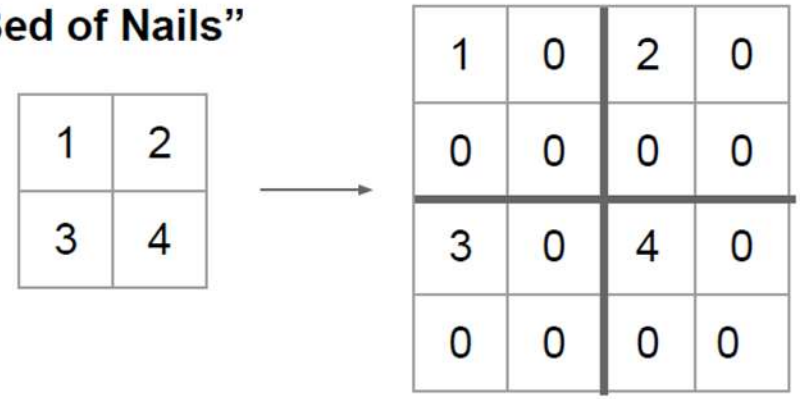
Nearest Neighbor



Input: 2 x 2

Output: 4 x 4

"Bed of Nails"



Input: 2 x 2

Output: 4 x 4

Max Unpooling

Use positions from pooling layer

Input: 2 x 2

Output: 4 x 4

Reference: Lecture 11, segmentation, http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture11.pdf

Up-sampling: Transpose convolution

- Recap: Regular convolution (stride 1, pad 0)

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ x_{31} & x_{32} & x_{33} & x_{34} \\ x_{41} & x_{42} & x_{43} & x_{44} \end{bmatrix} \underset{\text{Convolution}}{*} \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} = \begin{bmatrix} z_{11} & z_{12} \\ z_{21} & z_{22} \end{bmatrix}$$

- Matrix-vector form

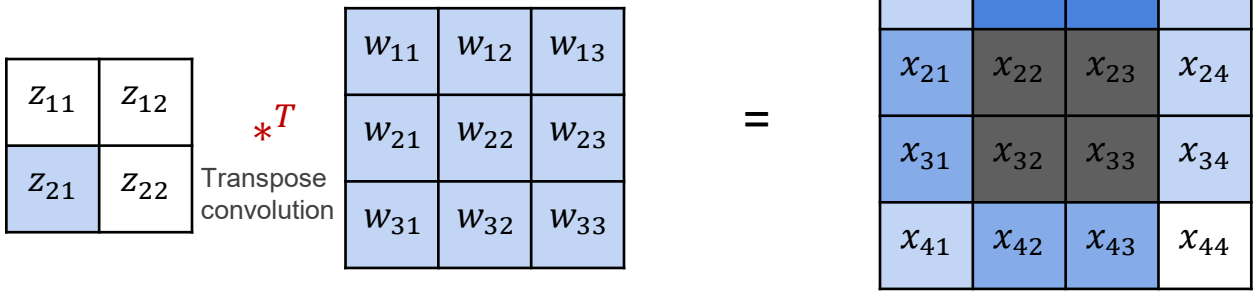
$$\begin{pmatrix} w_{11} & w_{12} & w_{13} & 0 & w_{21} & w_{22} & w_{23} & 0 & w_{31} & w_{32} & w_{33} & 0 & 0 & 0 & 0 & 0 \\ 0 & w_{11} & w_{12} & w_{13} & 0 & w_{21} & w_{22} & w_{23} & 0 & w_{31} & w_{32} & w_{33} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_{11} & w_{12} & w_{13} & 0 & w_{21} & w_{22} & w_{23} & 0 & w_{31} & w_{32} & w_{33} & 0 \\ 0 & 0 & 0 & 0 & 0 & w_{11} & w_{12} & w_{13} & 0 & w_{21} & w_{22} & w_{23} & 0 & w_{31} & w_{32} & w_{33} \end{pmatrix} \begin{pmatrix} x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ \vdots \\ x_{44} \end{pmatrix} = \begin{pmatrix} z_{11} \\ z_{12} \\ z_{21} \\ z_{22} \end{pmatrix}$$

4×4 input, 2×2 output

Reference: CS498, Lecture 10, Segmentation, Introduction to Deep Learning, http://slazebni.cs.illinois.edu/fall18/

Up-sampling: Transpose convolution

- Transposed convolution



$$\begin{pmatrix} w_{11} & 0 & 0 & 0 \\ w_{12} & w_{11} & 0 & 0 \\ w_{13} & w_{12} & 0 & 0 \\ 0 & w_{13} & 0 & 0 \\ w_{21} & 0 & w_{11} & 0 \\ w_{22} & w_{21} & w_{12} & w_{11} \\ w_{23} & w_{22} & w_{13} & w_{12} \\ 0 & w_{23} & 0 & w_{13} \\ w_{31} & 0 & w_{21} & 0 \\ w_{32} & w_{31} & w_{22} & w_{21} \\ w_{33} & w_{32} & w_{23} & w_{22} \\ 0 & w_{33} & 0 & w_{23} \\ 0 & 0 & w_{31} & 0 \\ 0 & 0 & w_{32} & w_{31} \\ 0 & 0 & w_{33} & w_{32} \\ 0 & 0 & 0 & w_{33} \end{pmatrix} \begin{pmatrix} z_{11} \\ z_{12} \\ x_{21} \\ z_{22} \end{pmatrix} = \begin{pmatrix} x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{21} \\ x_{22} \\ x_{23} \\ x_{24} \\ x_{31} \\ x_{32} \\ x_{33} \\ x_{34} \\ x_{41} \\ x_{42} \\ x_{43} \\ x_{44} \end{pmatrix}$$

Alternate view:

- Place copies of the filter on the output, weighted by entries of the input
- Sum where copies of the filter overlap

Reference: CS498, Lecture 10, Segmentation, Introduction to Deep Learning, http://slazebni.cs.illinois.edu/fall18/

Appendix: Transpose convolution

- Transpose convolution

$$\begin{bmatrix} z_{11} & z_{12} \\ z_{21} & z_{22} \end{bmatrix} *^{T} \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ x_{31} & x_{32} & x_{33} & x_{34} \\ x_{41} & x_{42} & x_{43} & x_{44} \end{bmatrix}$$

Transpose convolution

$$\begin{pmatrix} w_{11} & 0 & 0 & 0 \\ w_{12} & w_{11} & 0 & 0 \\ w_{13} & w_{12} & 0 & 0 \\ 0 & w_{13} & 0 & 0 \\ w_{21} & 0 & w_{11} & 0 \\ w_{22} & w_{21} & w_{12} & w_{11} \\ w_{23} & w_{22} & w_{13} & w_{12} \\ 0 & w_{23} & 0 & w_{13} \\ w_{31} & 0 & w_{21} & 0 \\ w_{32} & w_{31} & w_{22} & w_{21} \\ w_{33} & w_{32} & w_{23} & w_{22} \\ 0 & w_{33} & 0 & w_{23} \\ 0 & 0 & w_{31} & 0 \\ 0 & 0 & w_{32} & w_{31} \\ 0 & 0 & w_{33} & w_{32} \\ 0 & 0 & 0 & w_{33} \end{pmatrix} \begin{pmatrix} z_{11} \\ z_{12} \\ z_{21} \\ z_{22} \end{pmatrix} = \begin{pmatrix} x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{21} \\ x_{22} \\ x_{23} \\ x_{24} \\ x_{31} \\ x_{32} \\ x_{33} \\ x_{34} \\ x_{41} \\ x_{42} \\ x_{43} \\ x_{44} \end{pmatrix}$$

2×2 input, 4×4 output

Reference: CS498, Lecture 10, Segmentation, Introduction to Deep Learning, http://slazebni.cs.illinois.edu/fall18/

Appendix: Transpose convolution

- Transposed convolution

$$\begin{bmatrix} z_{11} & z_{12} \\ z_{21} & z_{22} \end{bmatrix} *^{T} \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ x_{31} & x_{32} & x_{33} & x_{34} \\ x_{41} & x_{42} & x_{43} & x_{44} \end{bmatrix}$$

Transpose convolution

$$\begin{pmatrix} w_{11} & 0 & 0 & 0 \\ w_{12} & w_{11} & 0 & 0 \\ w_{13} & w_{12} & 0 & 0 \\ 0 & w_{13} & 0 & 0 \\ w_{21} & 0 & w_{11} & 0 \\ w_{22} & w_{21} & w_{12} & w_{11} \\ w_{23} & w_{22} & w_{13} & w_{12} \\ 0 & w_{23} & 0 & w_{13} \\ w_{31} & 0 & w_{21} & 0 \\ w_{32} & w_{31} & w_{22} & w_{21} \\ w_{33} & w_{32} & w_{23} & w_{22} \\ 0 & w_{33} & 0 & w_{23} \\ 0 & 0 & w_{31} & 0 \\ 0 & 0 & w_{32} & w_{31} \\ 0 & 0 & w_{33} & w_{32} \\ 0 & 0 & 0 & w_{33} \end{pmatrix} \begin{pmatrix} z_{11} \\ z_{12} \\ z_{21} \\ z_{22} \end{pmatrix} = \begin{pmatrix} x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{21} \\ x_{22} \\ x_{23} \\ x_{24} \\ x_{31} \\ x_{32} \\ x_{33} \\ x_{34} \\ x_{41} \\ x_{42} \\ x_{43} \\ x_{44} \end{pmatrix}$$

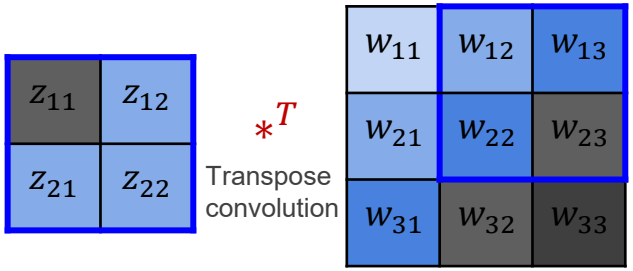
Convolve input with *flipped* filter

$$x_{12} = w_{12}z_{11} + w_{11}z_{12}$$

Reference: CS498, Lecture 10, Segmentation, Introduction to Deep Learning, http://slazebni.cs.illinois.edu/fall18/

Appendix: Transpose convolution

- Transposed convolution



$=$

x_{11}	x_{12}	x_{13}	x_{14}
x_{21}	x_{22}	x_{23}	x_{24}
x_{31}	x_{32}	x_{33}	x_{34}
x_{41}	x_{42}	x_{43}	x_{44}

Convolve input with *flipped* filter

$$\begin{pmatrix} w_{11} & 0 & 0 & 0 \\ w_{12} & w_{11} & 0 & 0 \\ w_{13} & w_{12} & 0 & 0 \\ 0 & w_{13} & 0 & 0 \\ w_{21} & 0 & w_{11} & 0 \\ w_{22} & w_{21} & w_{12} & w_{11} \\ w_{23} & w_{22} & w_{13} & w_{12} \\ 0 & w_{23} & 0 & w_{13} \\ w_{31} & 0 & w_{21} & 0 \\ w_{32} & w_{31} & w_{22} & w_{21} \\ w_{33} & w_{32} & w_{23} & w_{22} \\ 0 & w_{33} & 0 & w_{23} \\ 0 & 0 & w_{31} & 0 \\ 0 & 0 & w_{32} & w_{31} \\ 0 & 0 & w_{33} & w_{32} \\ 0 & 0 & 0 & w_{33} \end{pmatrix} \begin{pmatrix} z_{11} \\ z_{12} \\ z_{21} \\ z_{22} \end{pmatrix} = \begin{pmatrix} x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{21} \\ x_{22} \\ x_{23} \\ x_{24} \\ x_{31} \\ x_{32} \\ x_{33} \\ x_{34} \\ x_{41} \\ x_{42} \\ x_{43} \\ x_{44} \end{pmatrix}$$

$$x_{23} = w_{23}z_{11} + w_{22}z_{12} + w_{13}z_{21} + w_{12}z_{22}$$

Reference: CS498, Lecture 10, Segmentation, Introduction to Deep Learning, http://slazebni.cs.illinois.edu/fall18/

Appendix: Transpose convolution

- Transposed convolution

$$\begin{bmatrix} z_{11} & z_{12} \\ z_{21} & z_{22} \end{bmatrix} \; *^T \; \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ x_{31} & x_{32} & x_{33} & x_{34} \\ x_{41} & x_{42} & x_{43} & x_{44} \end{bmatrix}$$

Transpose convolution

$$\begin{bmatrix} w_{11} & 0 & 0 & 0 \\ w_{12} & w_{11} & 0 & 0 \\ w_{13} & w_{12} & 0 & 0 \\ 0 & w_{13} & 0 & 0 \\ w_{21} & 0 & w_{11} & 0 \\ w_{22} & w_{21} & w_{12} & w_{11} \\ w_{23} & w_{22} & w_{13} & w_{12} \\ 0 & w_{23} & 0 & w_{13} \\ w_{31} & 0 & w_{21} & 0 \\ w_{32} & w_{31} & w_{22} & w_{21} \\ w_{33} & w_{32} & w_{23} & w_{22} \\ 0 & w_{33} & 0 & w_{23} \\ 0 & 0 & w_{31} & 0 \\ 0 & 0 & w_{32} & w_{31} \\ 0 & 0 & w_{33} & w_{32} \\ 0 & 0 & 0 & w_{33} \end{bmatrix} \begin{bmatrix} z_{11} \\ z_{12} \\ x_{21} \\ z_{22} \end{bmatrix} = \begin{bmatrix} x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{21} \\ x_{22} \\ x_{23} \\ x_{24} \\ x_{31} \\ x_{32} \\ x_{33} \\ x_{34} \\ x_{41} \\ x_{42} \\ x_{43} \\ x_{44} \end{bmatrix}$$

Alternate view:

- Place copies of the filter on the output, weighted by entries of the input

Reference: CS498, Lecture 10, Segmentation, Introduction to Deep Learning, http://slazebni.cs.illinois.edu/fall18/

Appendix: Transpose convolution

- Transposed convolution

$$\begin{bmatrix} z_{11} & z_{12} \\ z_{21} & z_{22} \end{bmatrix} \overset{\text{Transpose convolution}}{*^T} \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ x_{31} & x_{32} & x_{33} & x_{34} \\ x_{41} & x_{42} & x_{43} & x_{44} \end{bmatrix}$$

$$\begin{bmatrix} w_{11} & 0 & 0 & 0 \\ w_{12} & w_{11} & 0 & 0 \\ w_{13} & w_{12} & 0 & 0 \\ 0 & w_{13} & 0 & 0 \\ w_{21} & 0 & w_{11} & 0 \\ w_{22} & w_{21} & w_{12} & w_{11} \\ w_{23} & w_{22} & w_{13} & w_{12} \\ 0 & w_{23} & 0 & w_{13} \\ w_{31} & 0 & w_{21} & 0 \\ w_{32} & w_{31} & w_{22} & w_{21} \\ w_{33} & w_{32} & w_{23} & w_{22} \\ 0 & w_{33} & 0 & w_{23} \\ 0 & 0 & w_{31} & 0 \\ 0 & 0 & w_{32} & w_{31} \\ 0 & 0 & w_{33} & w_{32} \\ 0 & 0 & 0 & w_{33} \end{bmatrix} \begin{bmatrix} z_{11} \\ z_{12} \\ x_{21} \\ z_{22} \end{bmatrix} = \begin{bmatrix} x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{21} \\ x_{22} \\ x_{23} \\ x_{24} \\ x_{31} \\ x_{32} \\ x_{33} \\ x_{34} \\ x_{41} \\ x_{42} \\ x_{43} \\ x_{44} \end{bmatrix}$$

Alternate view:

- Place copies of the filter on the output, weighted by entries of the input
- Sum where copies of the filter overlap

Reference: CS498, Lecture 10, Segmentation, Introduction to Deep Learning, http://slazebni.cs.illinois.edu/fall18/

FCN model

- Semantic segmentation faces an inherent tension between semantics and location: global information resolves “what”, while local information resolves “where”. Combining fine layers and coarse layers lets the model make local predictions that respect global structure.
- These skip connections from earlier layers in the network (prior to a down-sampling operation) should provide the necessary detail in order to reconstruct accurate shapes for segmentation boundaries.

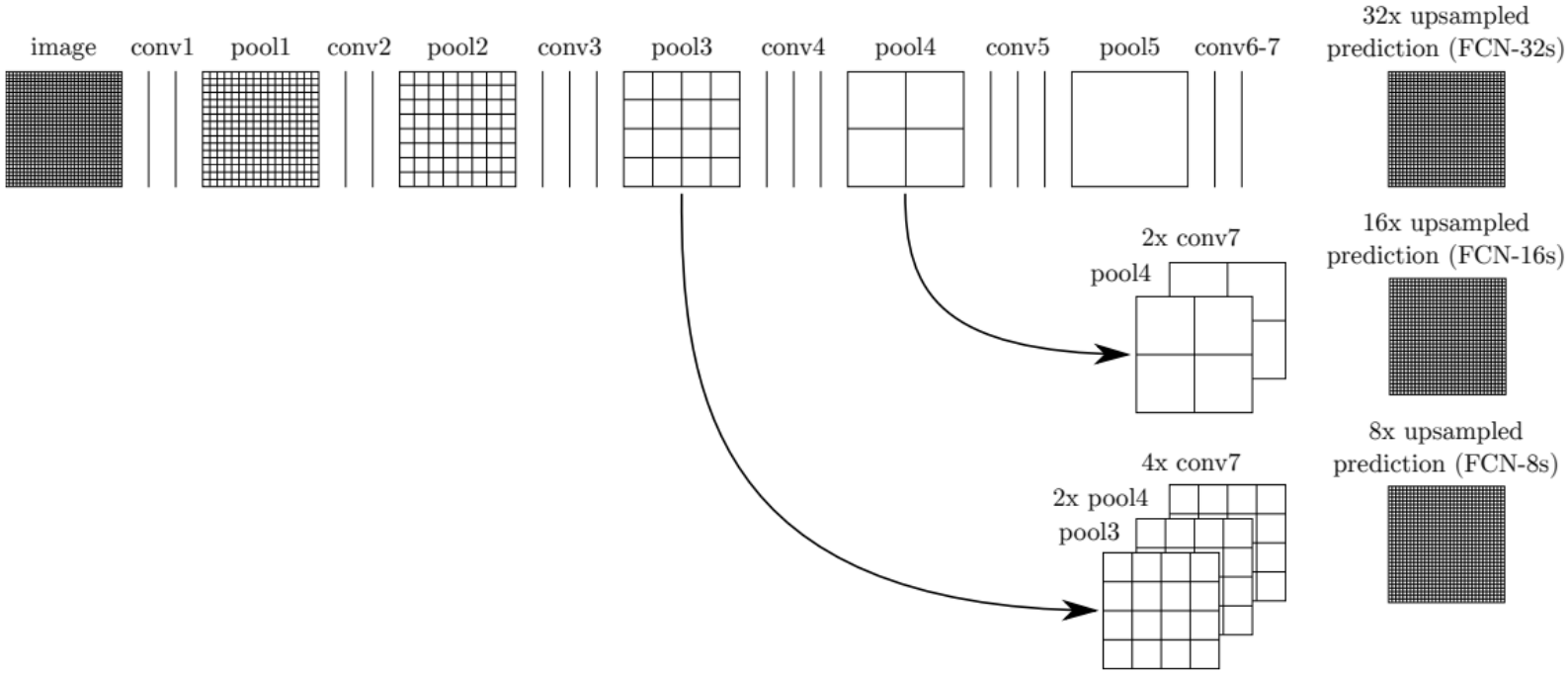


Figure 3. Our DAG nets learn to combine coarse, high layer information with fine, low layer information. Pooling and prediction layers are shown as grids that reveal relative spatial coarseness, while intermediate layers are shown as vertical lines. First row (FCN-32s): Our single-stream net, described in Section 4.1, upsamples stride 32 predictions back to pixels in a single step. Second row (FCN-16s): Combining predictions from both the final layer and the `pool4` layer, at stride 16, lets our net predict finer details, while retaining high-level semantic information. Third row (FCN-8s): Additional predictions from `pool3`, at stride 8, provide further precision.

Reference: Fully Convolutional Networks for Semantic Segmentation, CVPR 2015, https://people.eecs.berkeley.edu/~jonlong/long_shelhamer_fcn.pdf

Other semantic segmentation works

Semantic segmentation is fundamentally a pixel labelling task, can be used for other pixel-level labelling, such as colorization, depth estimation.

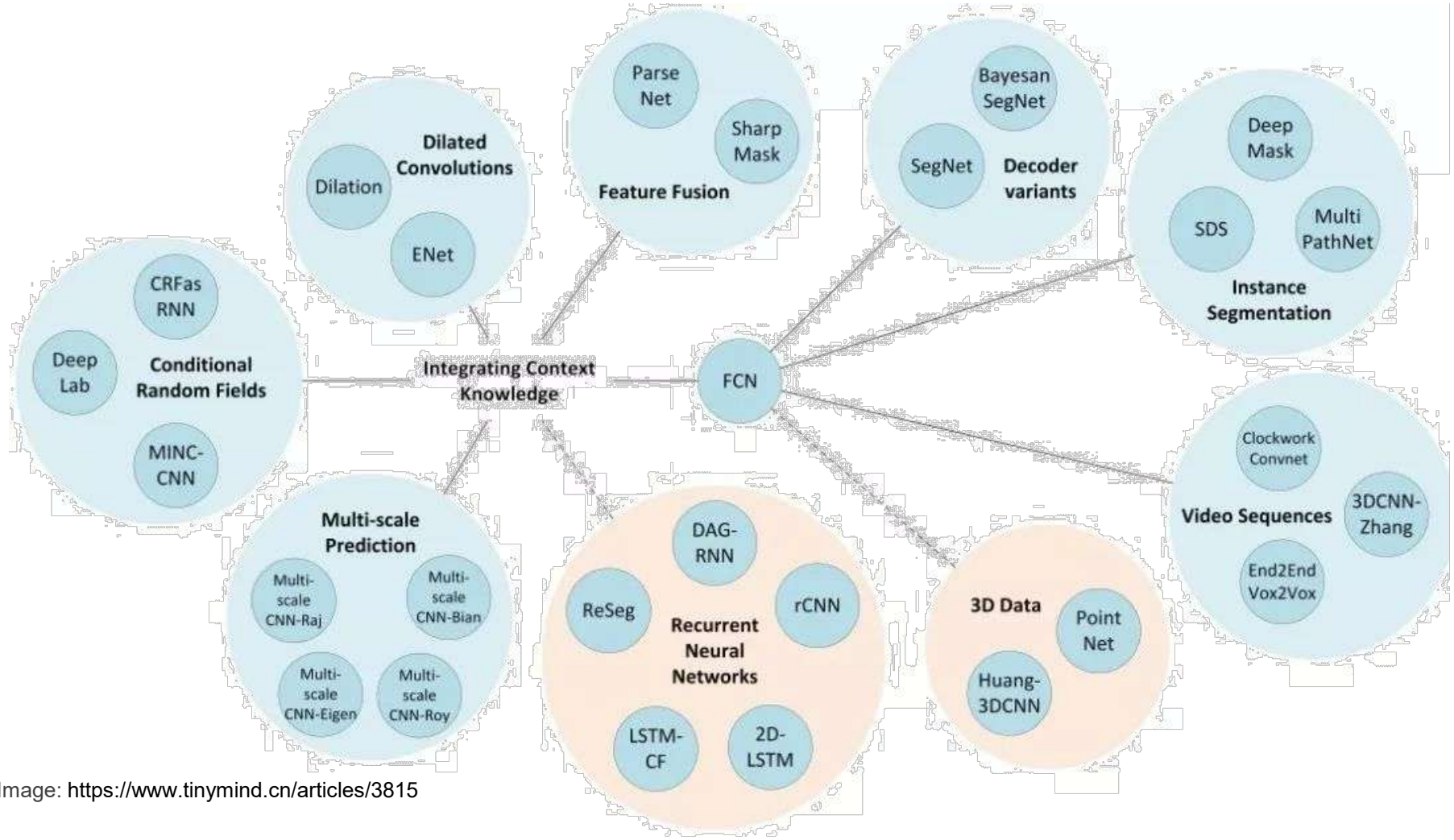
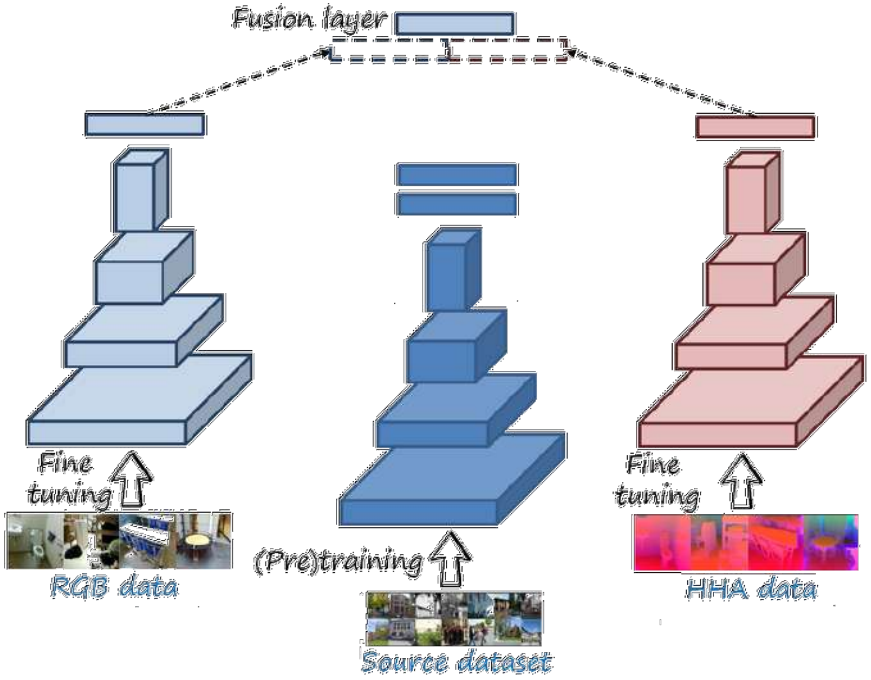


Image: https://www.tinymind.cn/articles/3815

RGB-D semantic segmentation

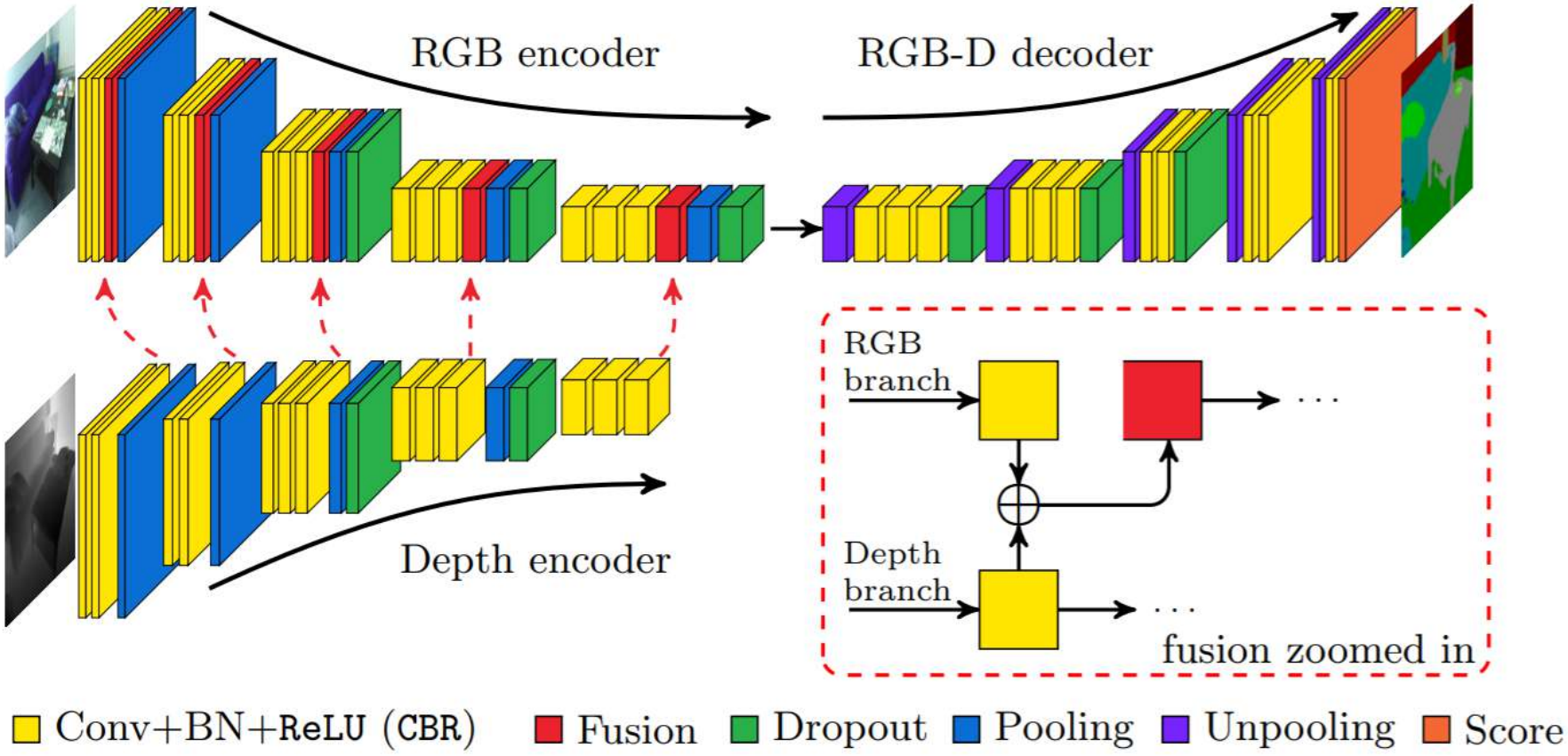
- Idea 1: Add depth as the 4th channel of the input image.
- Idea 2: Generate HHA image from the depth image, feed 3-channel RGB image and 3-channel HHA image into CNN framework, respectively.
- HHA: A geocentric embedding which converts a one channel-depth image into a three-channel image: Horizontal disparity, Height above ground and Angle with gravity.
- S. Gupta, Learning Rich Features from RGB-D Images for Object Detection and Segmentation, ECCV, 2014, http://saurabhg.web.illinois.edu/pdfs/gupta2014learning.pdf
- Python implementation: https://github.com/charlesCXK/Depth2HHA-python



Reference: http://www.lherranz.org/2018/10/17/rgbd-features/

RGB-D semantic segmentation

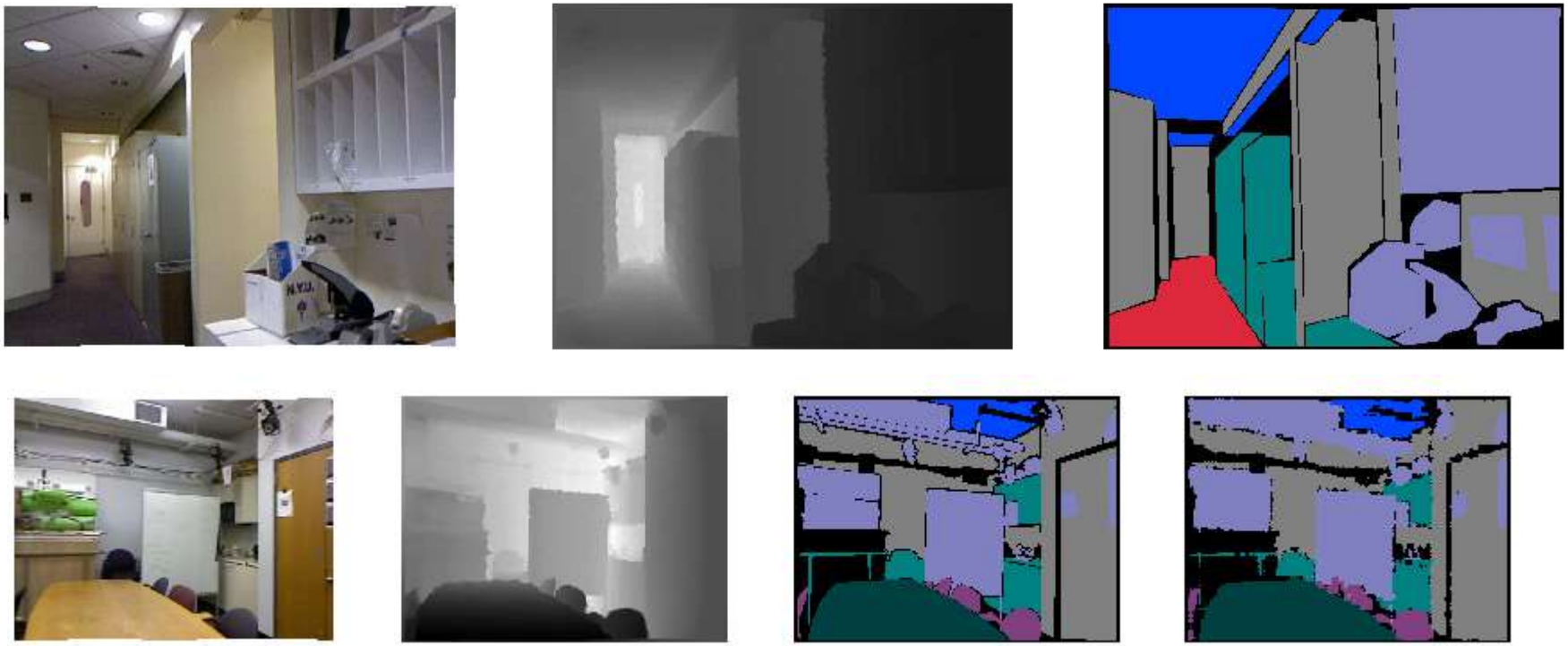
- Idea 3: Fuse two streams of RGB and depth image, respectively.



Reference: C. Hazirbas, L. Ma, C. Domokos, and D. Cremers, "FuseNet: Incorporating Depth into Semantic Segmentation via Fusion-Based CNN Architecture," ACCV 2016, https://vision.in.tum.de/_media/spezial/bib/hazirbasma2016fusenet.pdf

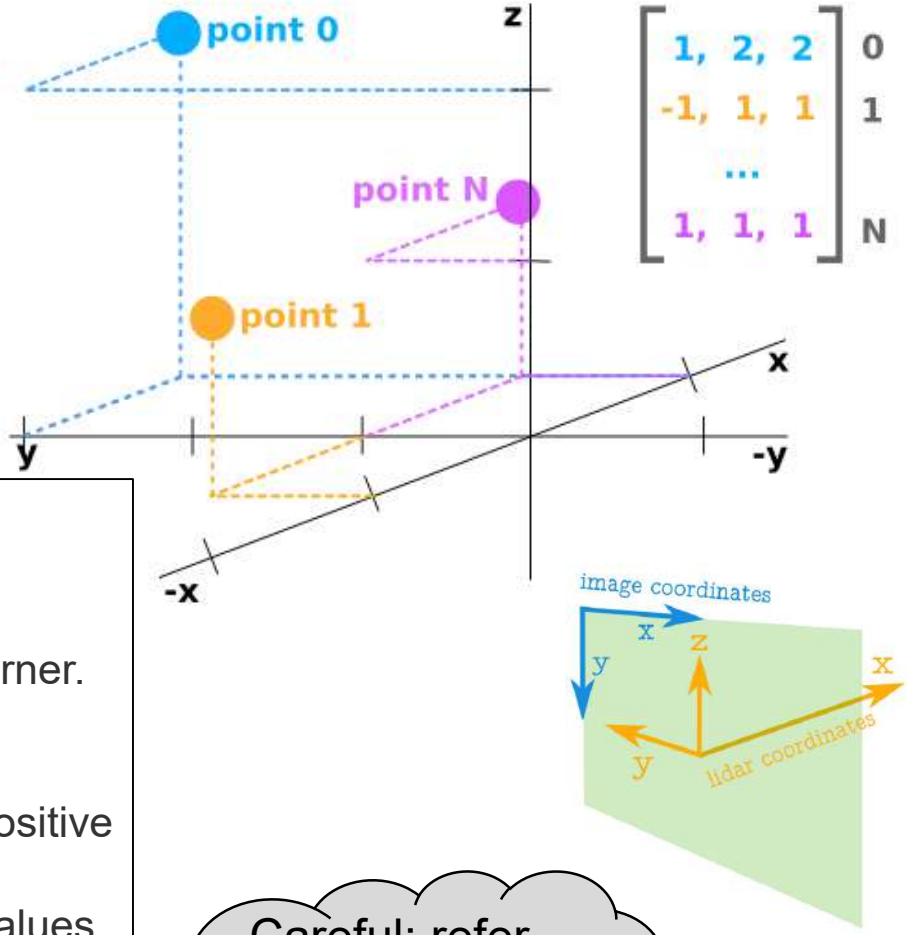
Example

- Objective: RGB-D semantic segmentation
- Dataset: https://cs.nyu.edu/~silberman/datasets/nyu_depth_v2.html
- **1449** densely labeled pairs of aligned RGB and depth images



Point cloud

- The point cloud data can be represented as a numpy array with N rows and 3 columns. Each row corresponds to a single point, which is represented using at least 3 values for its position in space (x, y, z).



Image

- The coordinate values in an image are always positive.
- The origin is located on the upper left hand corner.
- The coordinates are integer values.

Point cloud

- The coordinate values in point cloud can be positive or negative.
- The coordinates can take on real numbered values.
- The positive x axis represents forward.
- The positive y axis represents left.
- The positive z axis represents up.

Careful: refer specification of your LiDAR sensor

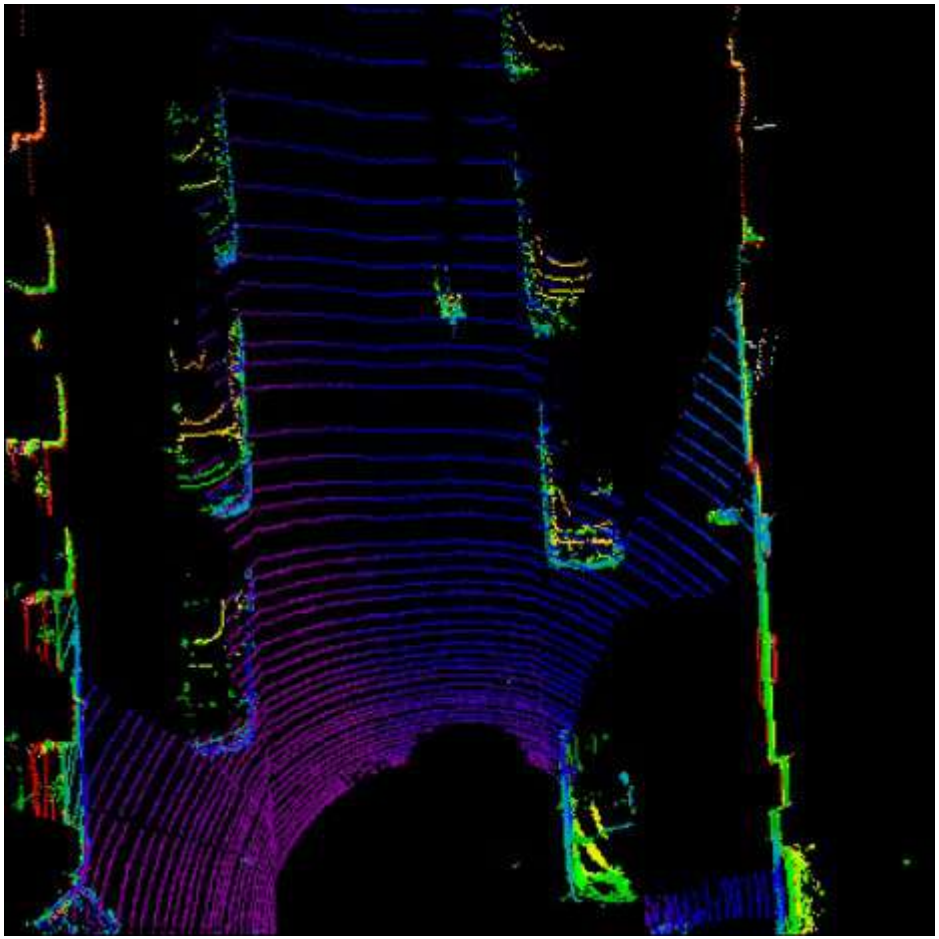
Reference: http://ronny.rest/tutorials/module/pointclouds_01/point_cloud_data/

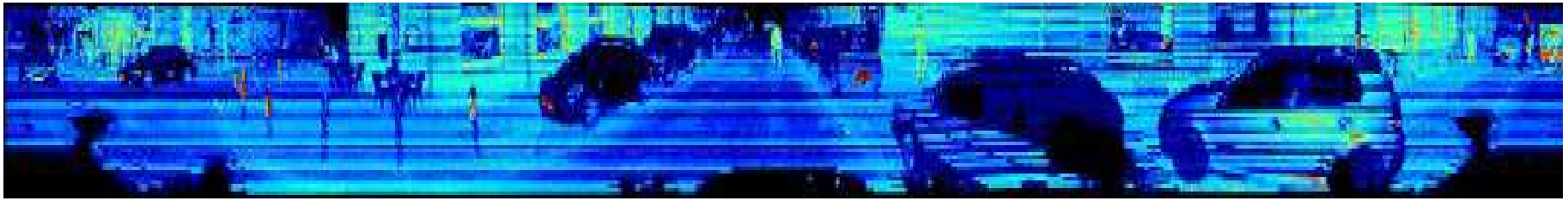
Point cloud

- Bird overview view
- Panoramic view

Demo: RPLIDAR A1

http://www.slamtec.com/en/lidar/a1

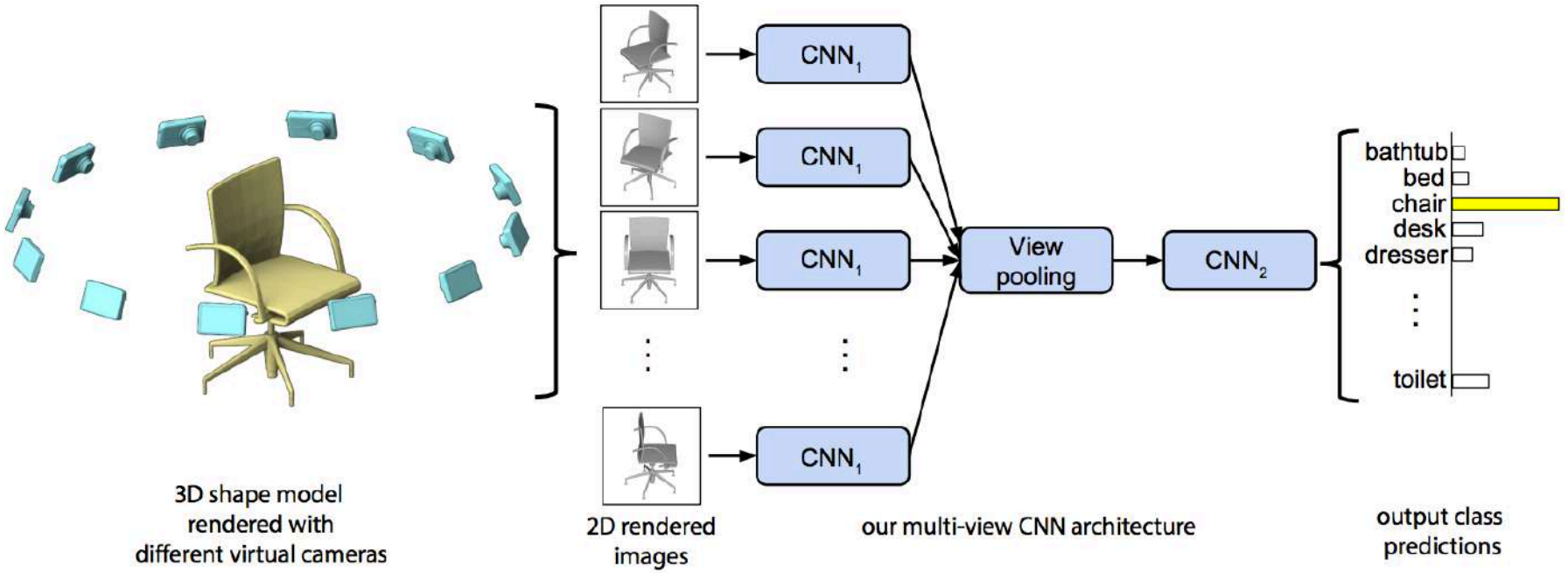




Reference: http://ronny.rest/tutorials/module/pointclouds_01/point_cloud_panoramic360/

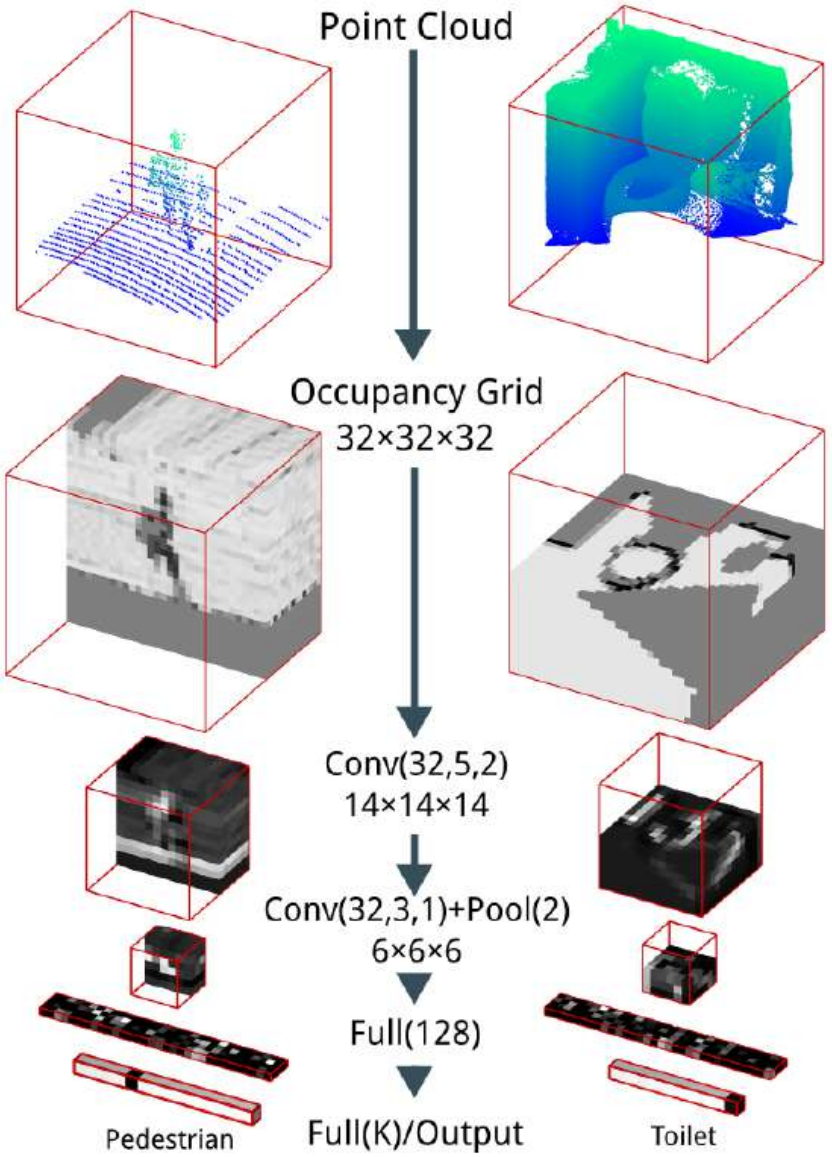
MVCNN

- First CNN: Extract image features from individual view
- View pooling: Element-wise max-pooling across all views.
- Second CNN: Extract shape features from pooled image



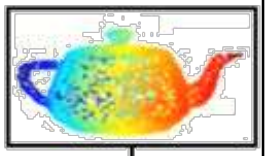
Reference: H. Su, S. Maji, E. Kalogerakis, E. Learned-Miller, "Multi-view Convolutional Neural Networks for 3D Shape Recognition," *IEEE Inter. Conf. on Computer Vision* (ICCV), Santiago, Chile, Dec. 2015, pp. 945-953, http://vis-www.cs.umass.edu/mvcnn/

VoxNet



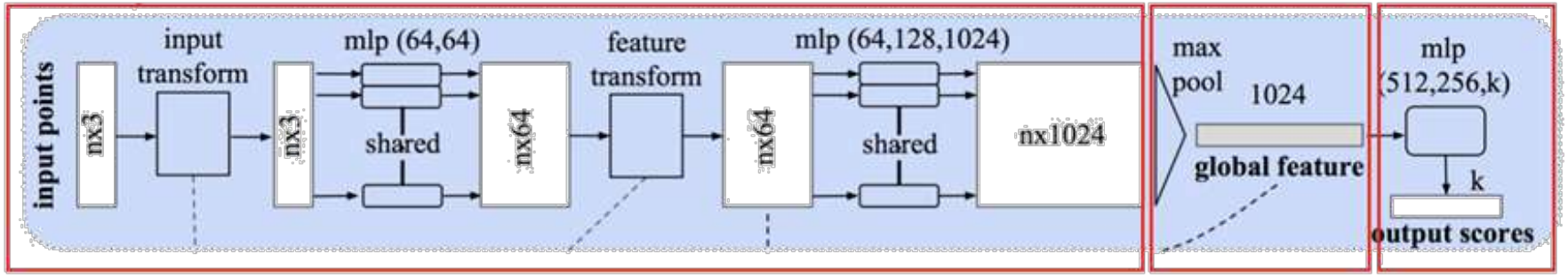
Reference: D. Maturana and S. Scherer, "VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition," IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), Hamburg, Germany, Oct. 2015, pp. 922-928.

PointNet



It uses a shared *multi-layer perceptron* (MLP) to map each of the n points from 3D (x, y, z) to 64 dimensions (i.e., mapping is identical on the n points). It is repeated to map the n points from 64 dimensions to 1024 dimensions. Max pooling is used to create a global feature vector. Finally, a three-layer fully-connected network is used to map the global feature vector to k output classification scores.

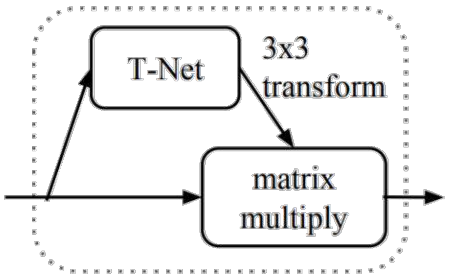
Teapot

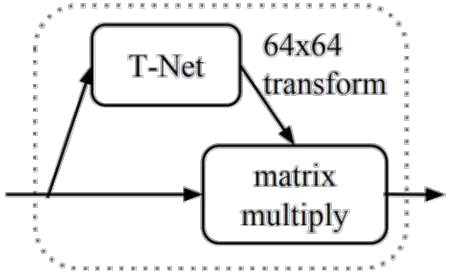


Non-linear transformations

Aggregates point features

Classification

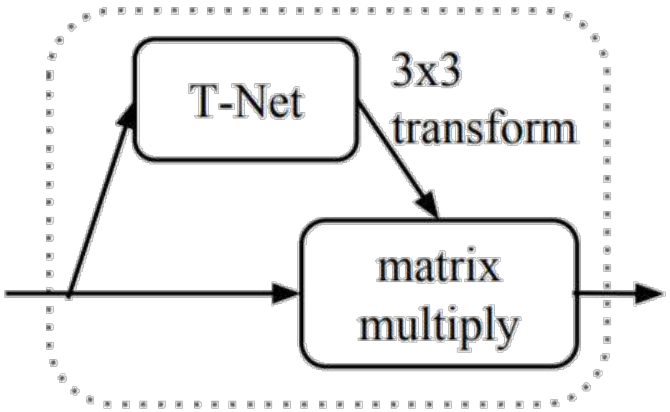


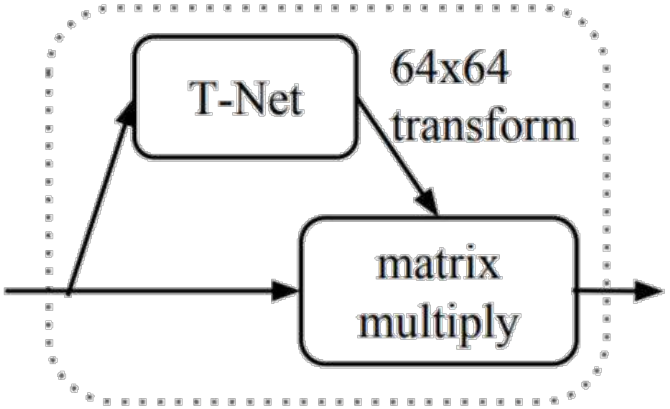


Reference: R. Qi, H. Su, M. Kaichun, L. J. Guibas, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," CVPR 2017, pp. 77-85.

PointNet

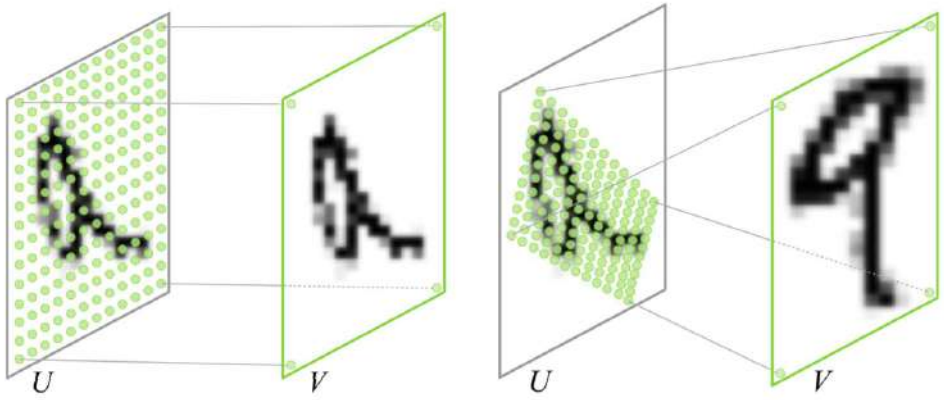
- For a given input point cloud, apply an appropriate rigid or affine transformation to achieve pose normalization. Because each of the n input points are represented as a vector and are mapped to the embedding spaces independently, applying a geometric transformation simply amounts to matrix multiplying each point with a transformation matrix. The T-Net is a regression network that is tasked with predicting an input-dependent 3-by-3 transformation matrix that is then matrix multiplied with the n-by-3 input.





Spatial transformer networks

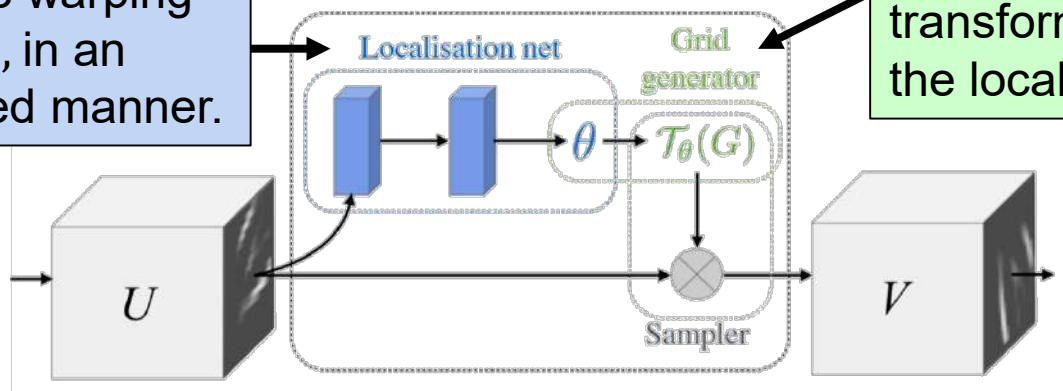
- Takes in feature map U and outputs parameters of the transformation θ, which is regressed using fully-connected or convolutional networks.
- Input: feature map U of shape (H, W, C)
- Output: transformation matrix θ of shape (6,)
- Architecture: fully-connected network or ConvNet



- U: Source transformed grid
- V: Target regular grid

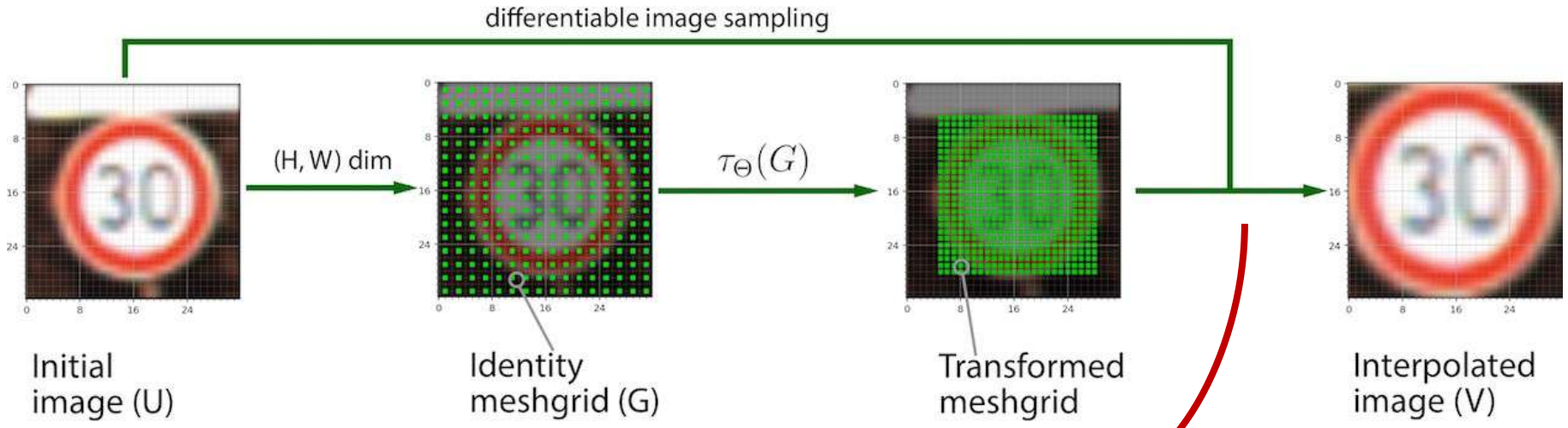
Regresses the warping parameters, θ, in an input-depended manner.

Generates sampling grid $T_\theta(G)$ by using the transformation predicted by the localization network.



Reference: Spatial transformer networks, https://arxiv.org/abs/1506.02025

Appendix: Spatial transformer networks



Differentiable image sampling

$$V_i^c = \sum_n^H \sum_m^W U_{nm}^c k(x_i^s - m; \Phi_x) k(y_i^s - n; \Phi_y) \quad \forall i \in [1 \ldots H'W'] \quad \forall c \in [1 \ldots C]$$

- x coordinate in $\tau_\Theta(G)$
- parameters of sampling kernel
- value at location (n, m) in channel c of input U
- interpolation kernel
- pixel in a channel c
- channels

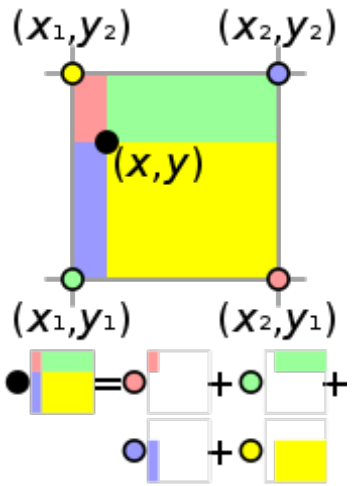
Where Φ_x and Φ_y are the parameters of a sampling kernel $k()$ which defines the image interpolation (e.g. bilinear), U_{nm}^c is the value at location (n, m) in channel c of the input, and V_i^c is the output value for pixel i at location (x_i^t, y_i^t) in channel c.

Photo: https://towardsdatascience.com/convnets-series-spatial-transformer-networks-cff47565ae81

Appendix: Spatial transformer networks

Bilinear resampling filter: If we choose a coordinate system, where the four points of f is known are $(0,0)$, $(0,1)$, $(1,0)$, and $(1,1)$, then the interpolation formula simplifies to be

$$f(x,y) = f(0,0)\,(1-x)(1-y) + f(1,0)\,x(1-y) + f(0,1)\,(1-x)y + f(1,1)\,xy$$



Differentiable image sampling (rewritten from the previous slide using bilinear filter)

$$V_i^c = \sum_n^H \sum_m^W U_{nm}^c \max(0, 1 - |x_i^s - m|) \max(0, 1 - |y_i^s - n|)$$

$$\frac{\partial V_i^c}{\partial U_{nm}^c} = \sum_n^H \sum_m^W \max(0, 1 - |x_i^s - m|) \max(0, 1 - |y_i^s - n|)$$

$$\frac{\partial V_i^c}{\partial x_i^s} = \sum_n^H \sum_m^W U_{nm}^c \max(0, 1 - |y_i^s - n|) \begin{cases} 0 & \text{if } |m - x_i^s| \geq 1 \\ 1 & \text{if } m \geq x_i^s \\ -1 & \text{if } m < x_i^s \end{cases}$$

Photo: https://towardsdatascience.com/convnets-series-spatial-transformer-networks-cff47565ae81

Workshop

- Objective: Point cloud classification
- Dataset: ModelNet10, http://modelnet.cs.princeton.edu/

PointNet, R. Qi, H. Su, M. Kaichun, L. J. Guibas, “PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation,” CVPR 2017, pp. 77-85.

What we have learnt

- 3D data representation including RGB-D and point cloud
- 3D data machine learning with various machine learning methods, including detection, segmentation, classification
- Case studies on RGB-D, point cloud machine learning and applications

Thank you!

Dr TIAN Jing
Email: tianjing@nus.edu.sg