3D Vision and Videos

Jisang Han

onground@korea.ac.kr

Artificial Intelligence in KU (AIKU)

Department of Computer Science and Engineering, Korea University

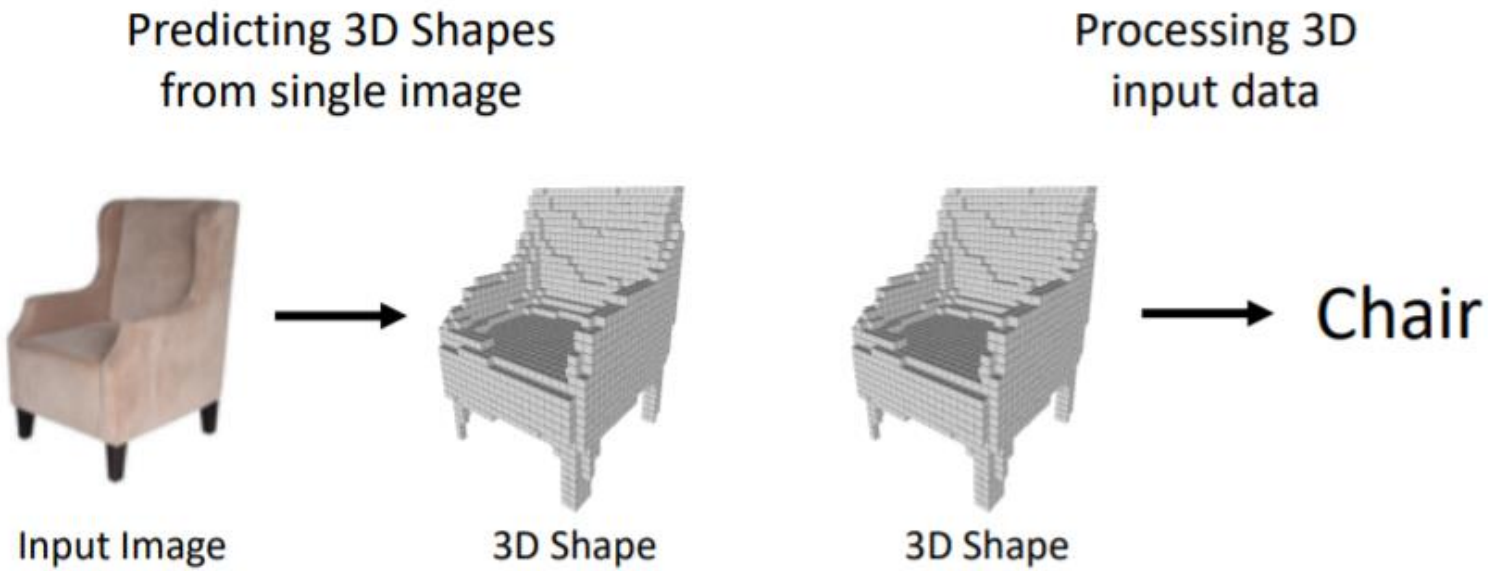
AIKU

3D Vision

AIKU

Table of Contents

3D Vision topics



Two Problems

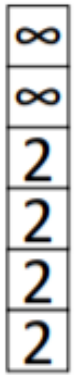
1. $2D\ Input \rightarrow 3D\ shape$
2. $3D\ Shape \rightarrow classification/segmentation$

3D Vision topics

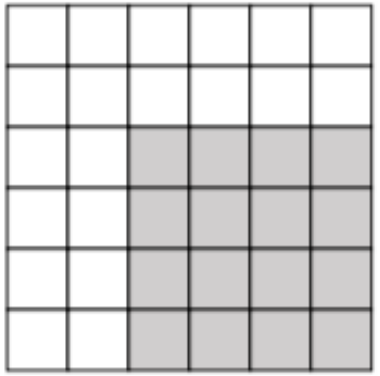
Computing correspondences
Multi-view stereo
Structure from Motion
Simultaneous Localization and Mapping (SLAM)
Self-supervised learning
View Synthesis
Differentiable graphics
3D Sensors

Many non-Deep Learning methods alive and well in 3D!

3D Shape Representations

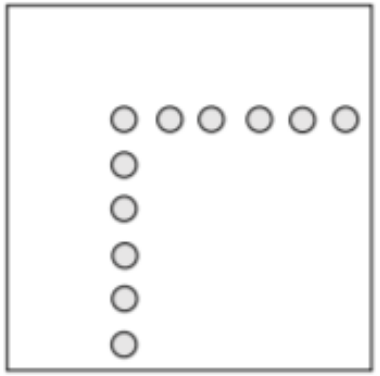


Depth Map

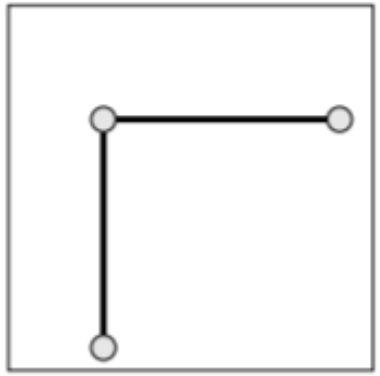


Voxel Grid

Implicit Surface



Pointcloud



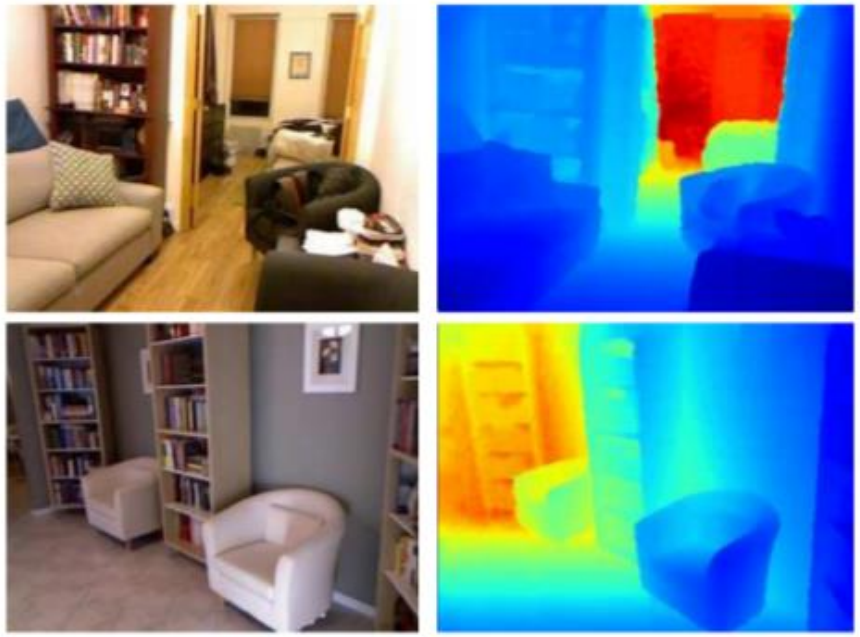
Mesh

3D Shape Representations: Depth Map

For each pixel, **depth map** gives distance from the camera to the object in the world at that pixel

RGB image + Depth image
= RGB-D Image (2.5D)

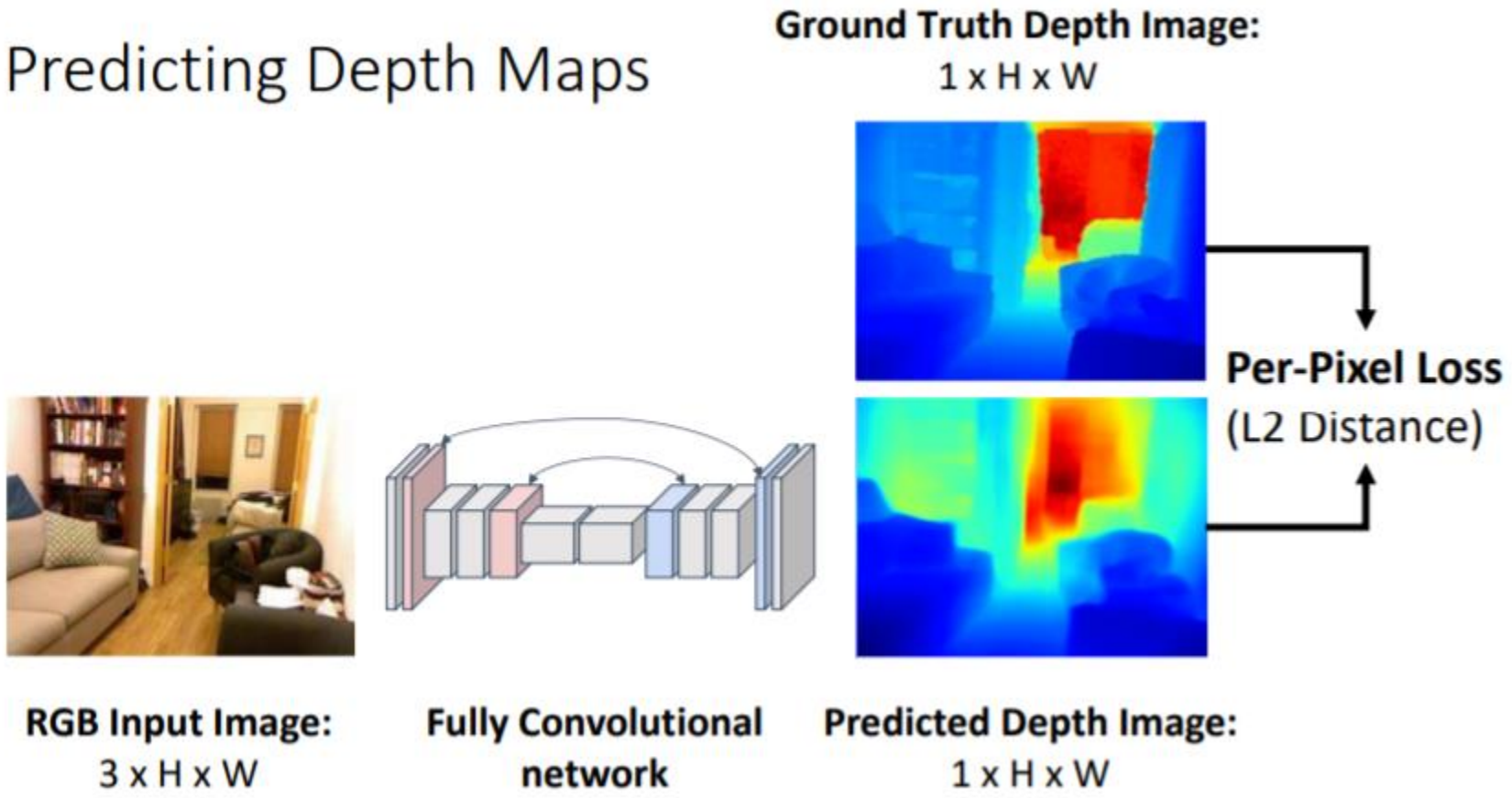
This type of data can be recorded directly for some types of 3D sensors (e.g. Microsoft Kinect)



RGB Image: 3 x H x W Depth Map: H x W

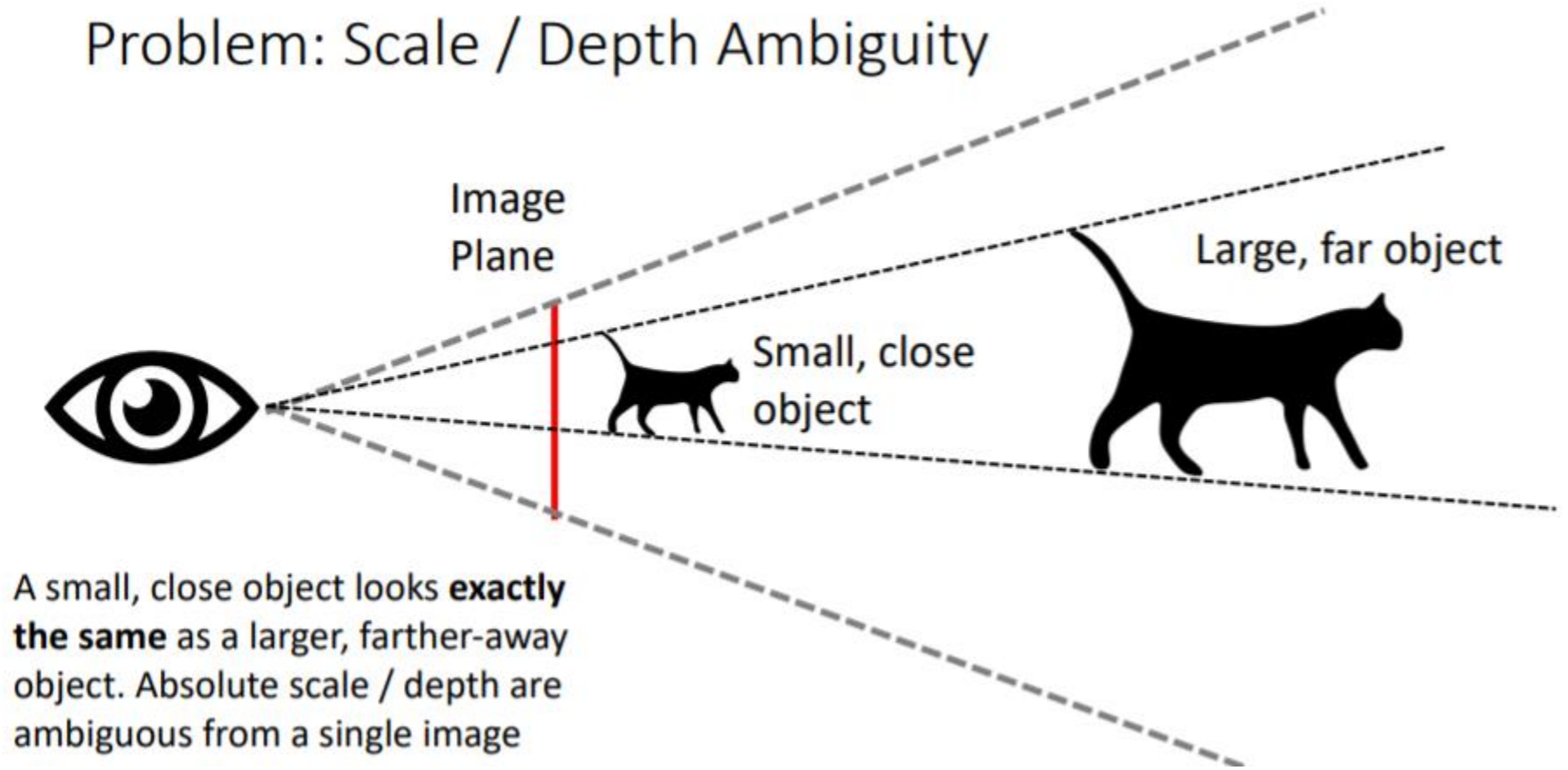
The Depth Map assigns **the distance between the camera and the pixel for each pixel**.

3D Shape Representations: Depth Map



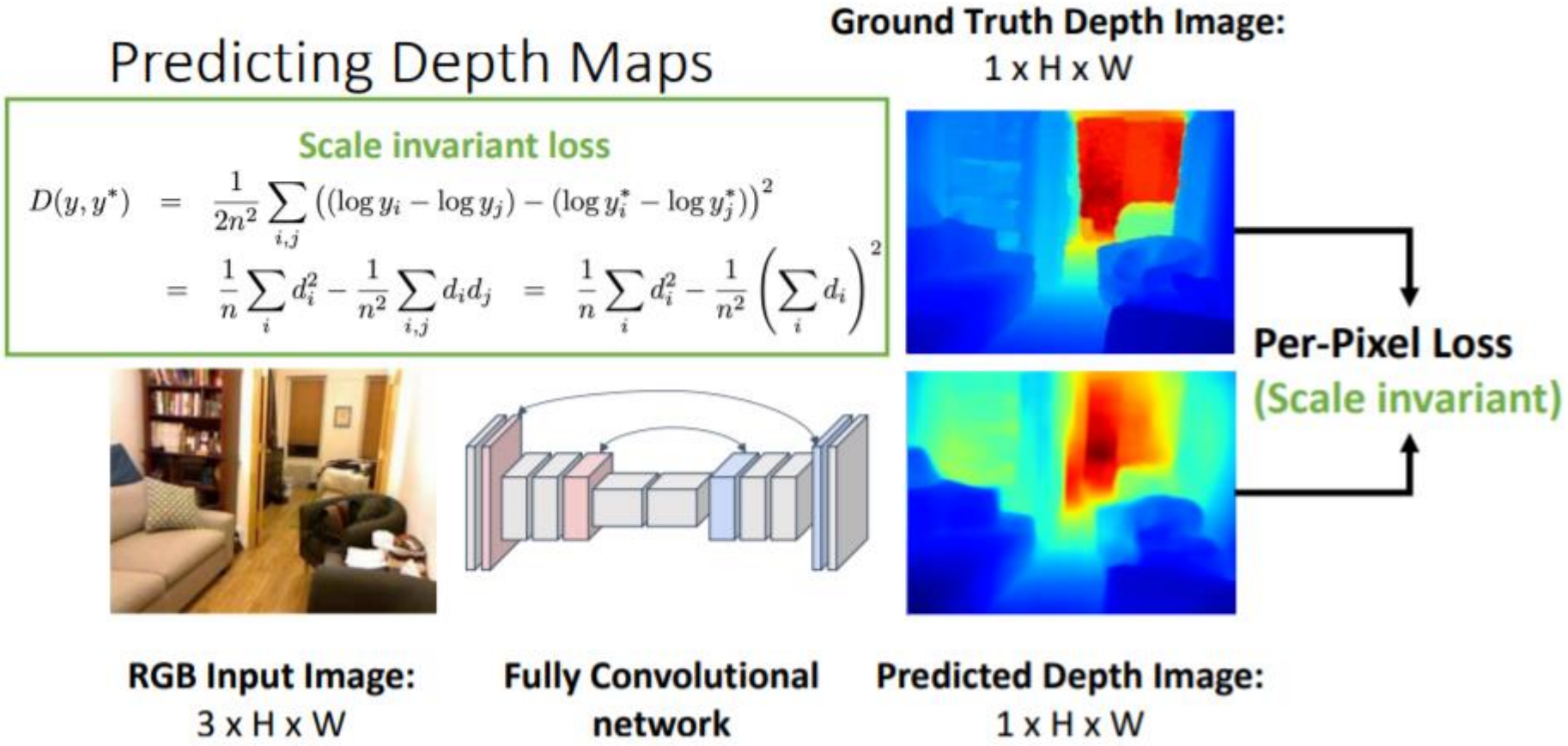
The Depth Map assigns **the distance between the camera and the pixel for each pixel**.

3D Shape Representations: Depth Map



Given **an image**, we cannot distinguish between small, close objects and large, far-off objects. For example, a cat of size 1 and a cat of size 2 look exactly the same.

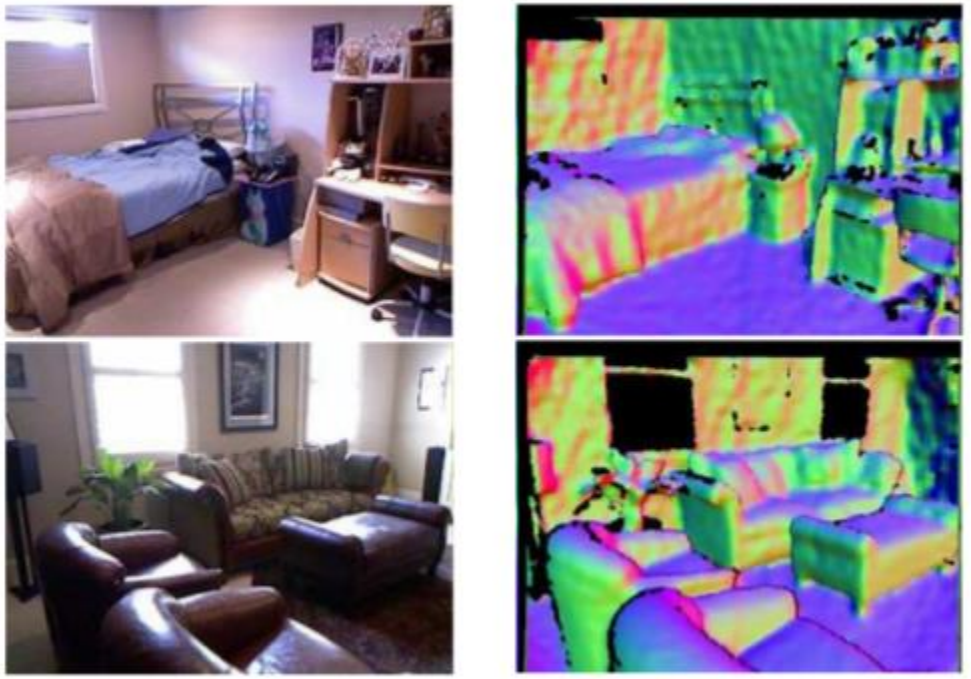
3D Shape Representations: Depth Map



https://deeesp.github.io/deep%20learning/SI-SE/

3D Shape Representations: Surface Normals

For each pixel, **surface normals** give a vector giving the normal vector to the object in the world for that pixel

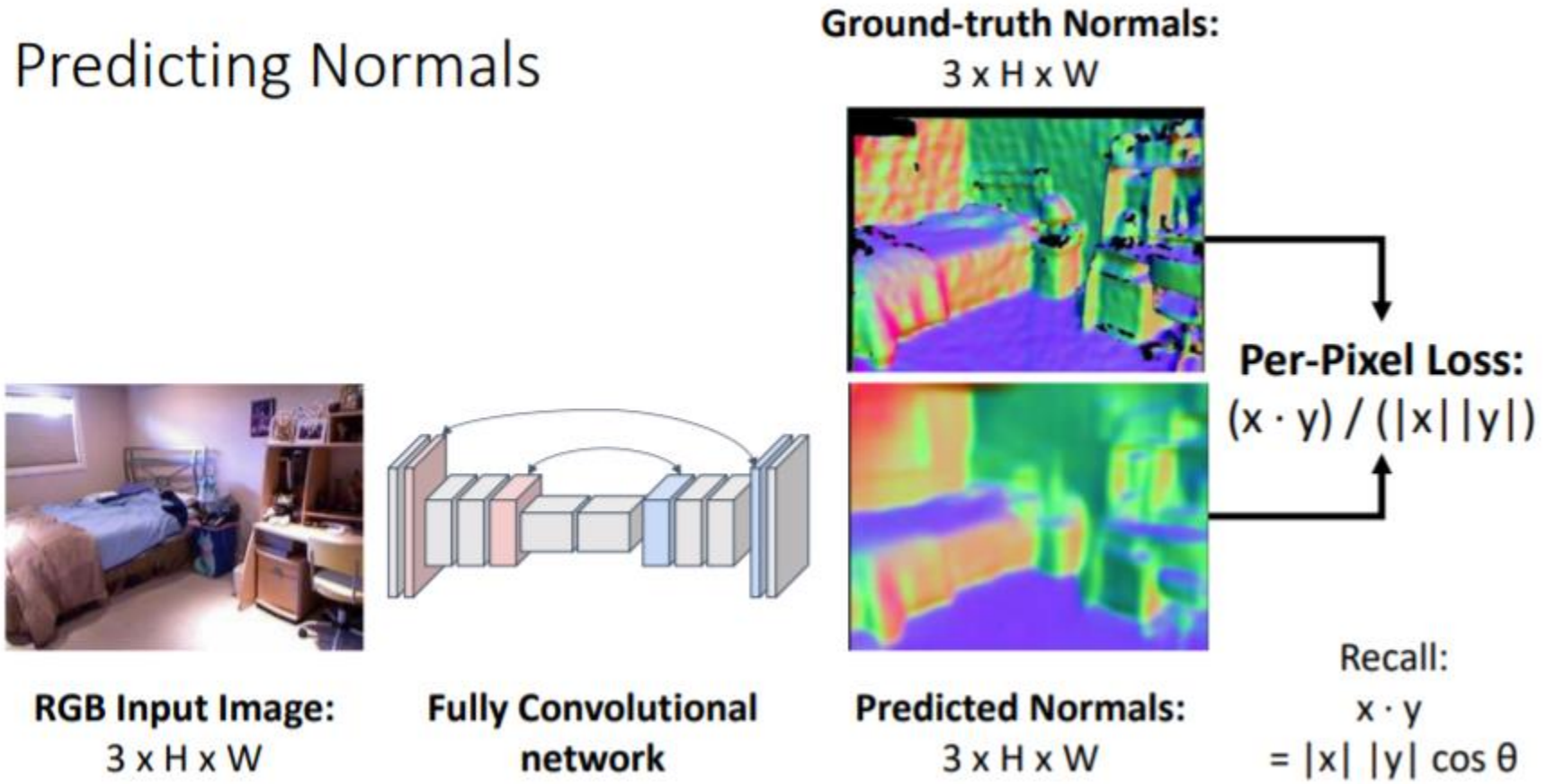


RGB Image: 3 x H x W

Normals: 3 x H x W

- Assign the orientation of the surface of that object for each pixel
- $\uparrow: blue, \downarrow: red, \leftarrow: green$

3D Shape Representations: Surface Normals



3D Shape Representations: Surface Normals

4.2. Surface Normals

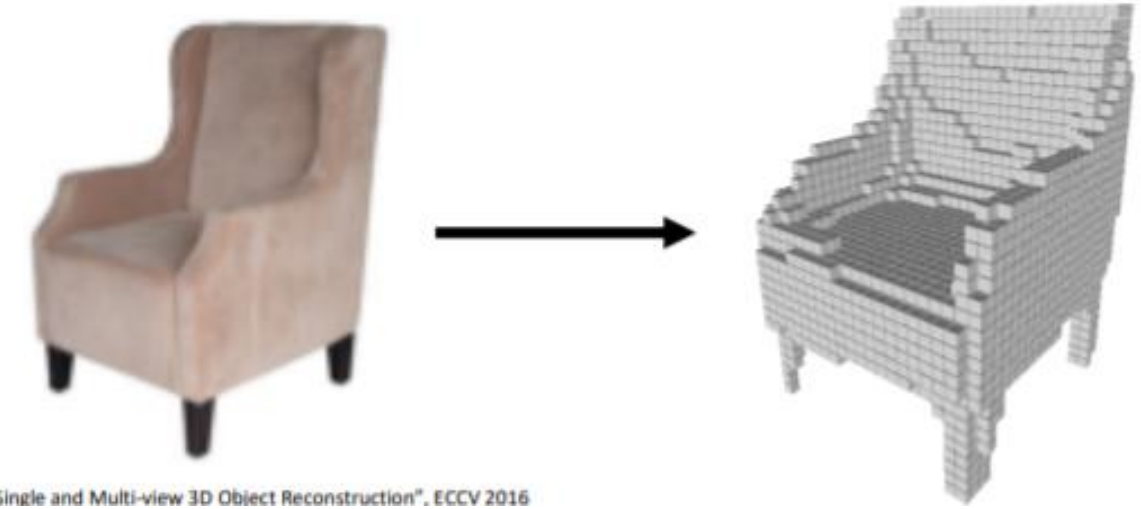
To predict surface normals, we change the output from one channel to three, and predict the x, y and z components of the normal at each pixel. We also normalize the vector at each pixel to unit l_2 norm, and backpropagate through this normalization. We then employ a simple elementwise loss comparing the predicted normal at each pixel to the ground truth, using a dot product:

$$L_{normals}(N, N^*) = -\frac{1}{n}\sum_i N_i \cdot N_i^* = -\frac{1}{n} N \cdot N^* \quad (2)$$

where N and N^* are predicted and ground truth normal vector maps, and the sums again run over valid pixels (*i.e.* those with a ground truth normal).

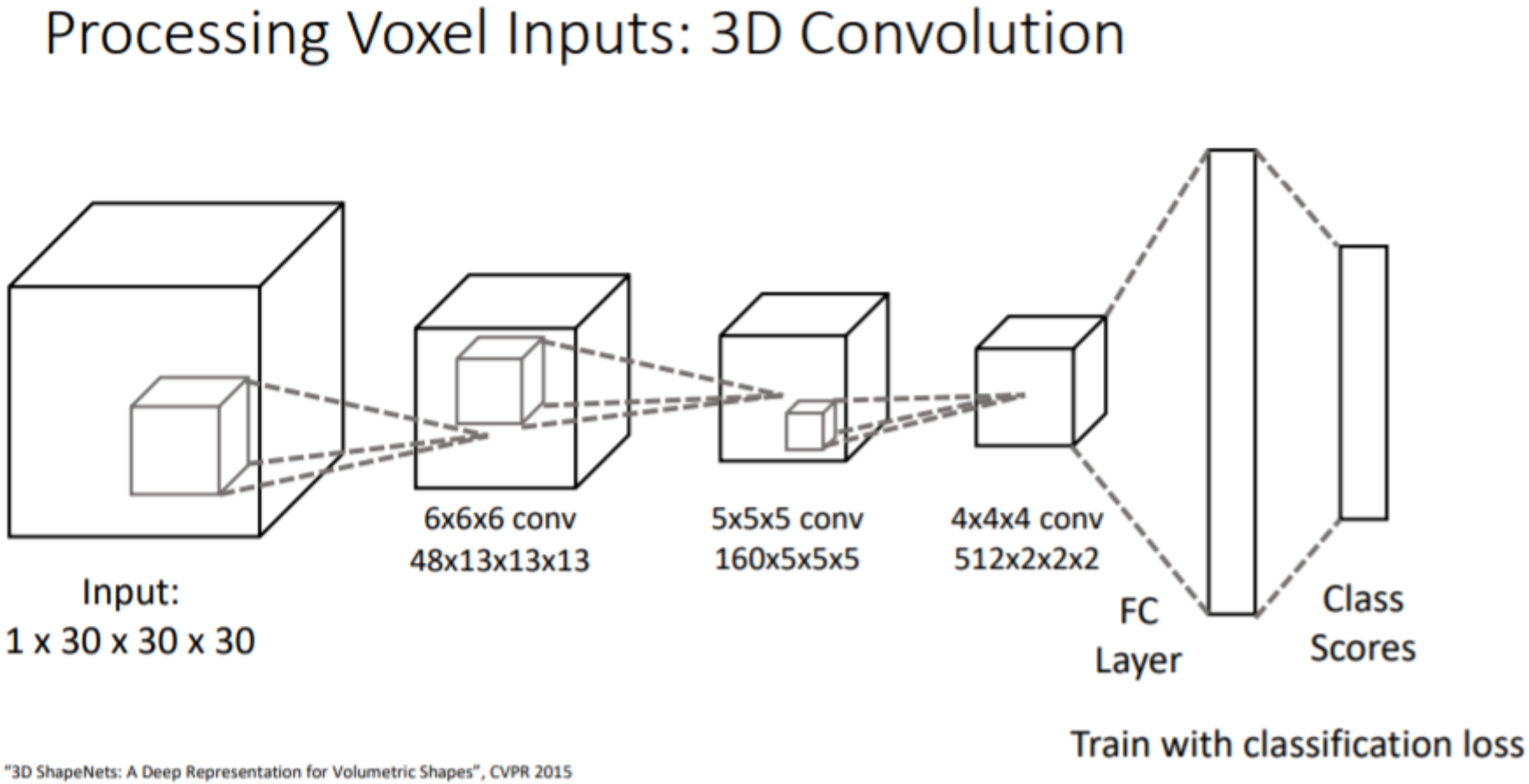
3D Shape Representations: Voxels

- Represent a shape with a V x V x V grid of occupancies
- Just like segmentation masks in Mask R-CNN, but in 3D!
- (+) Conceptually simple: just a 3D grid!
- (-) Need high spatial resolution to capture fine structures
- (-) Scaling to high resolutions is nontrivial!

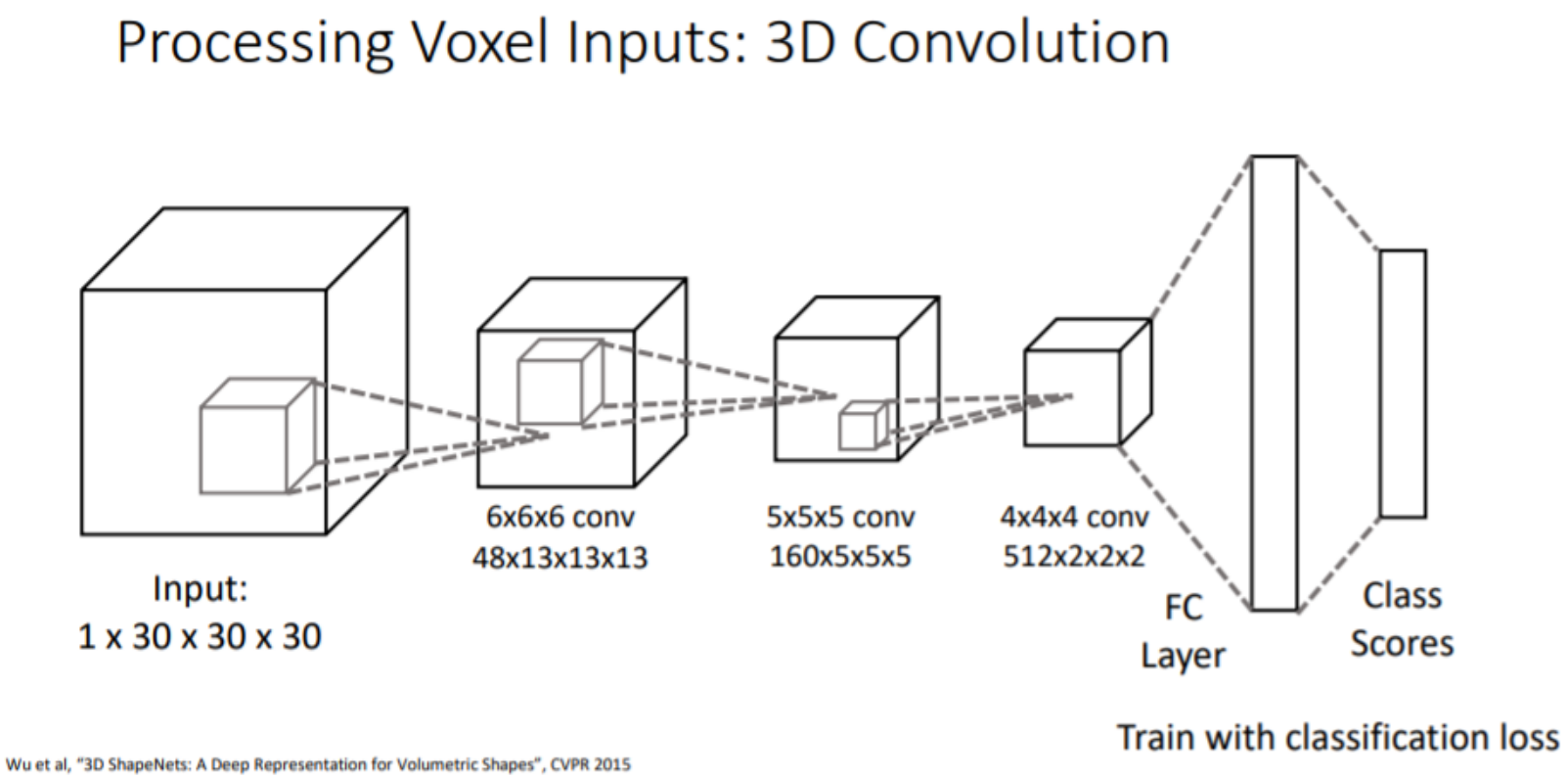


Choy et al. "3D-R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction". ECCV 2016

Processing Voxel Inputs: 3D Convolution



Processing Voxel Inputs: 3D Convolution

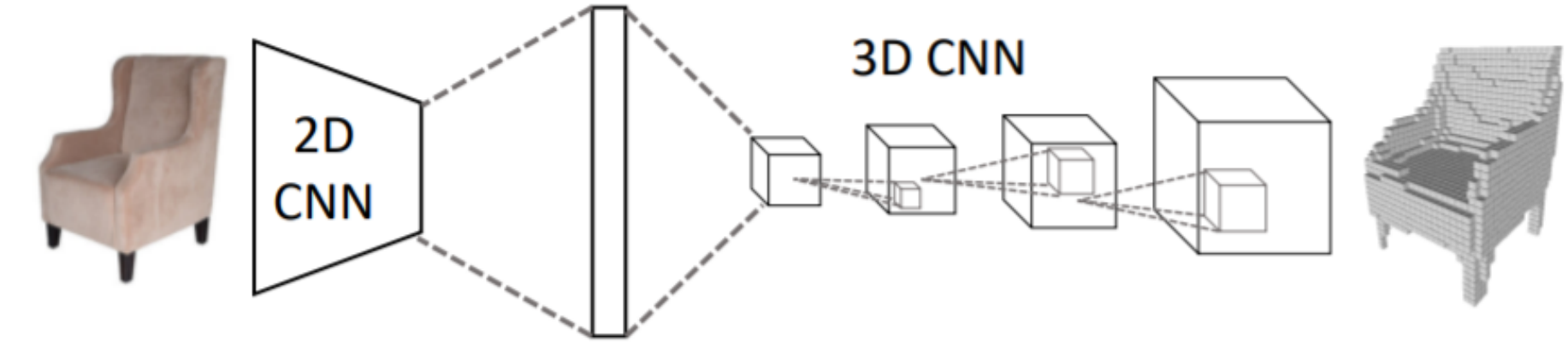


Input Tensor

- $1(channel) \times 30(x) \times 30(y) \times 30(z)$
- $channel$: 1 or 0 for 3D voxel grid (filled or not)

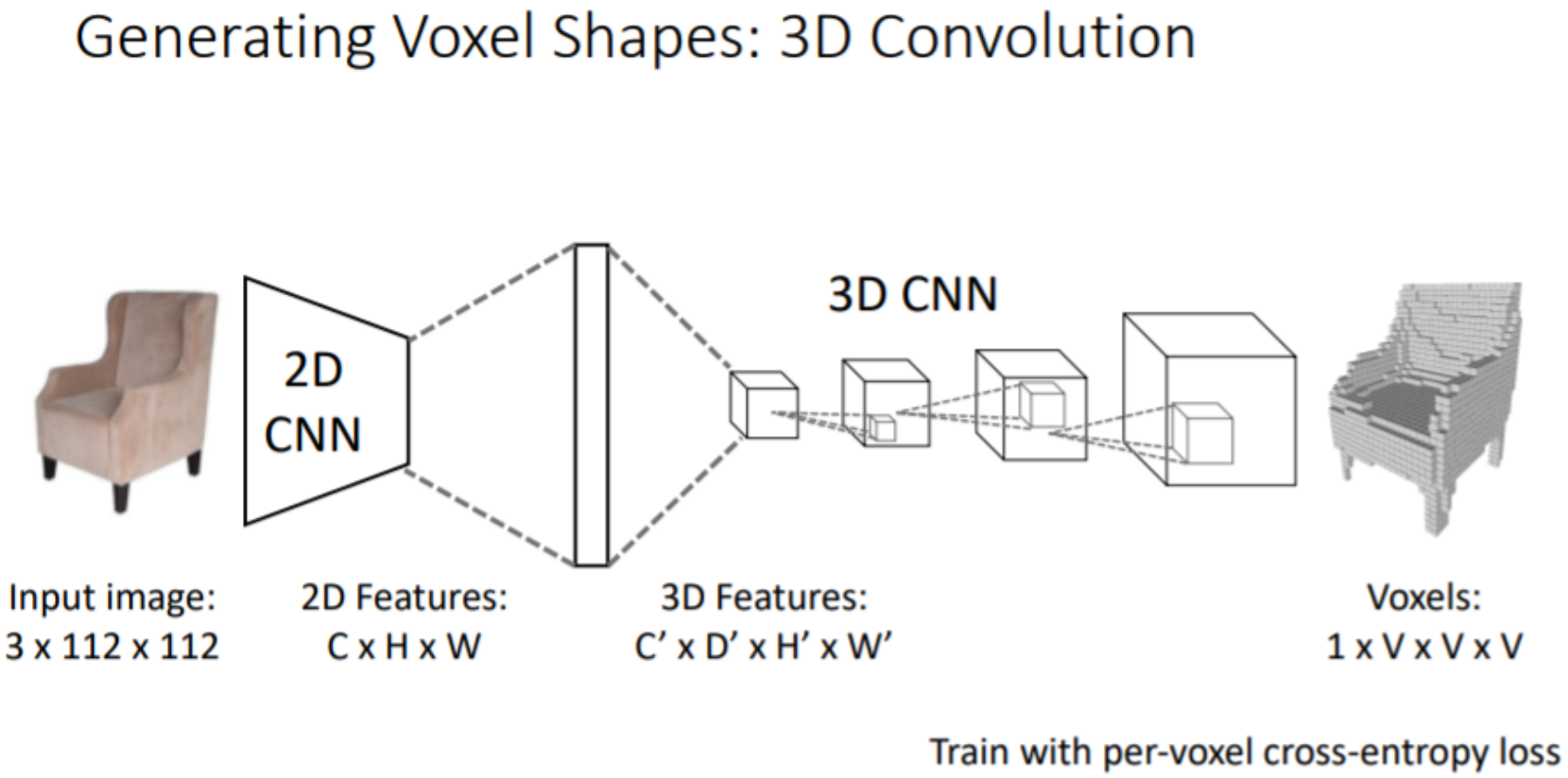
Generating Voxel Shapes: 3D Convolution

Generating Voxel Shapes: 3D Convolution



Train with per-voxel cross-entropy loss

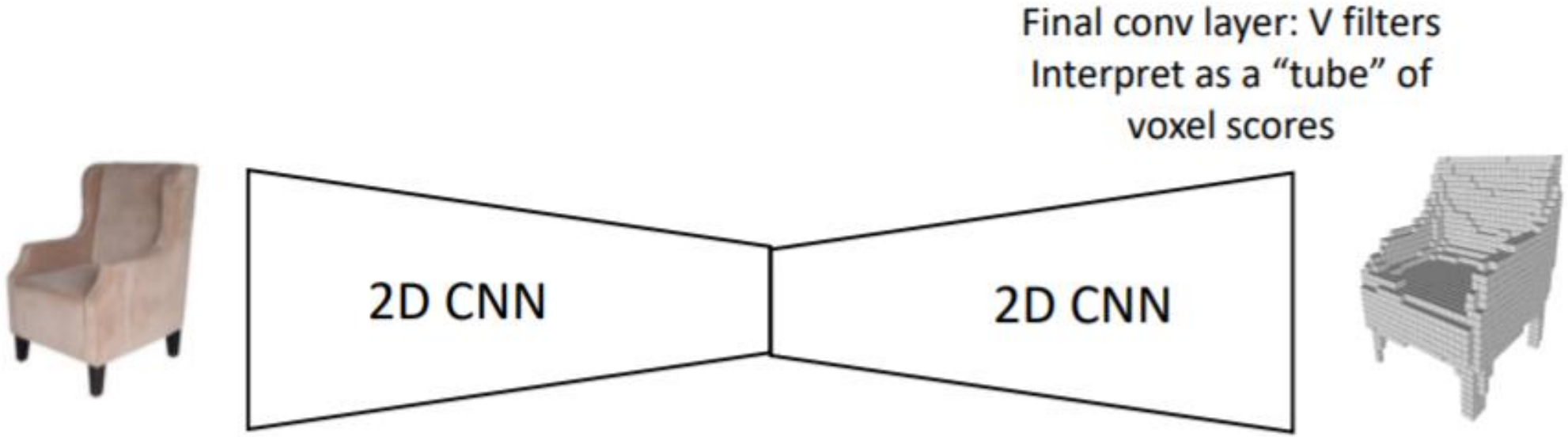
Generating Voxel Shapes: 3D Convolution



Input $3(RGB) \times (112 \times 112) \rightarrow$ **Output** $1(Voxel\ ON|OFF) \times (V \times V \times V)$

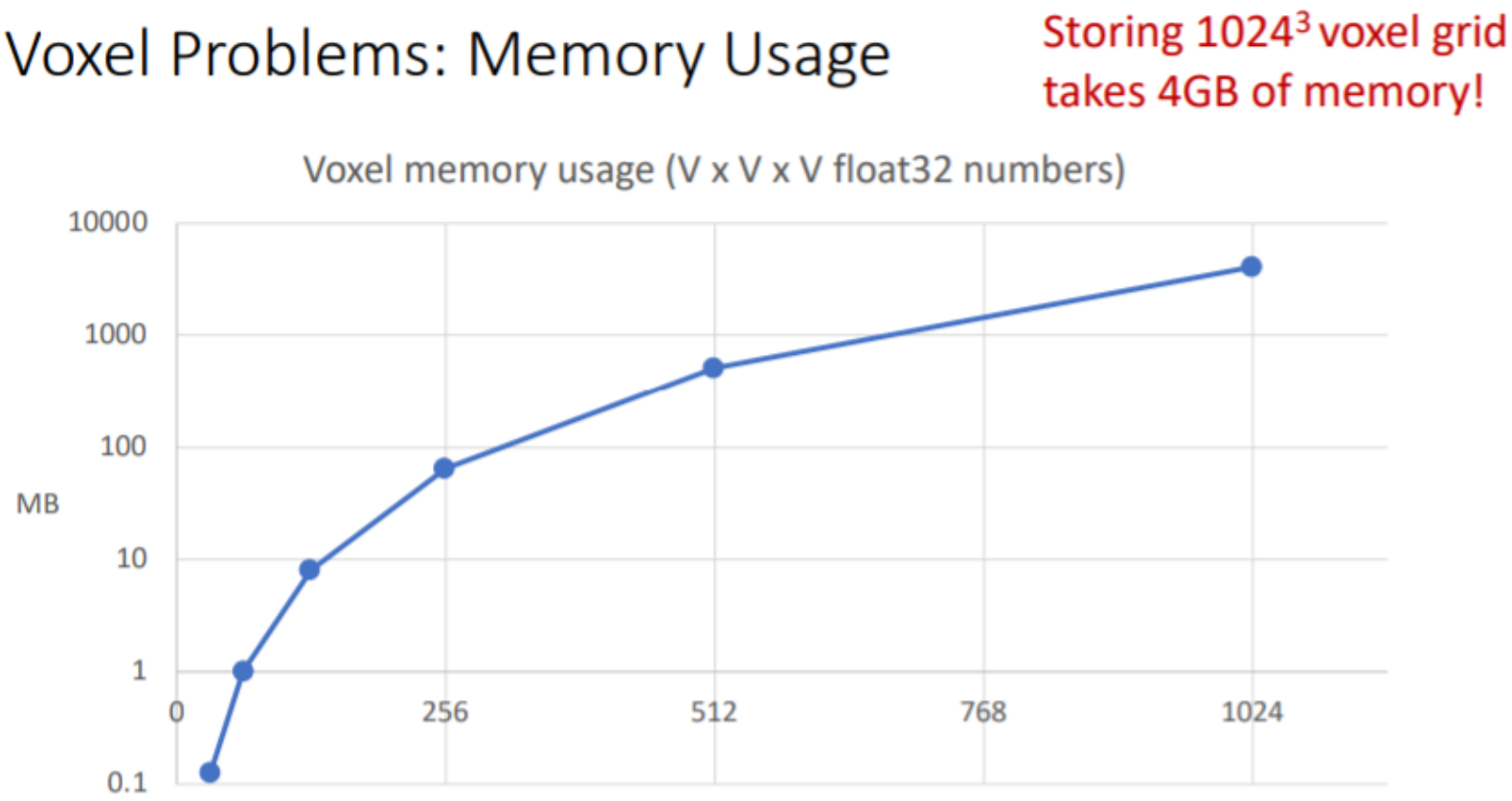
3D Convolution needs lots of computation

Generating Voxel Shapes: ”Voxel Tubes”



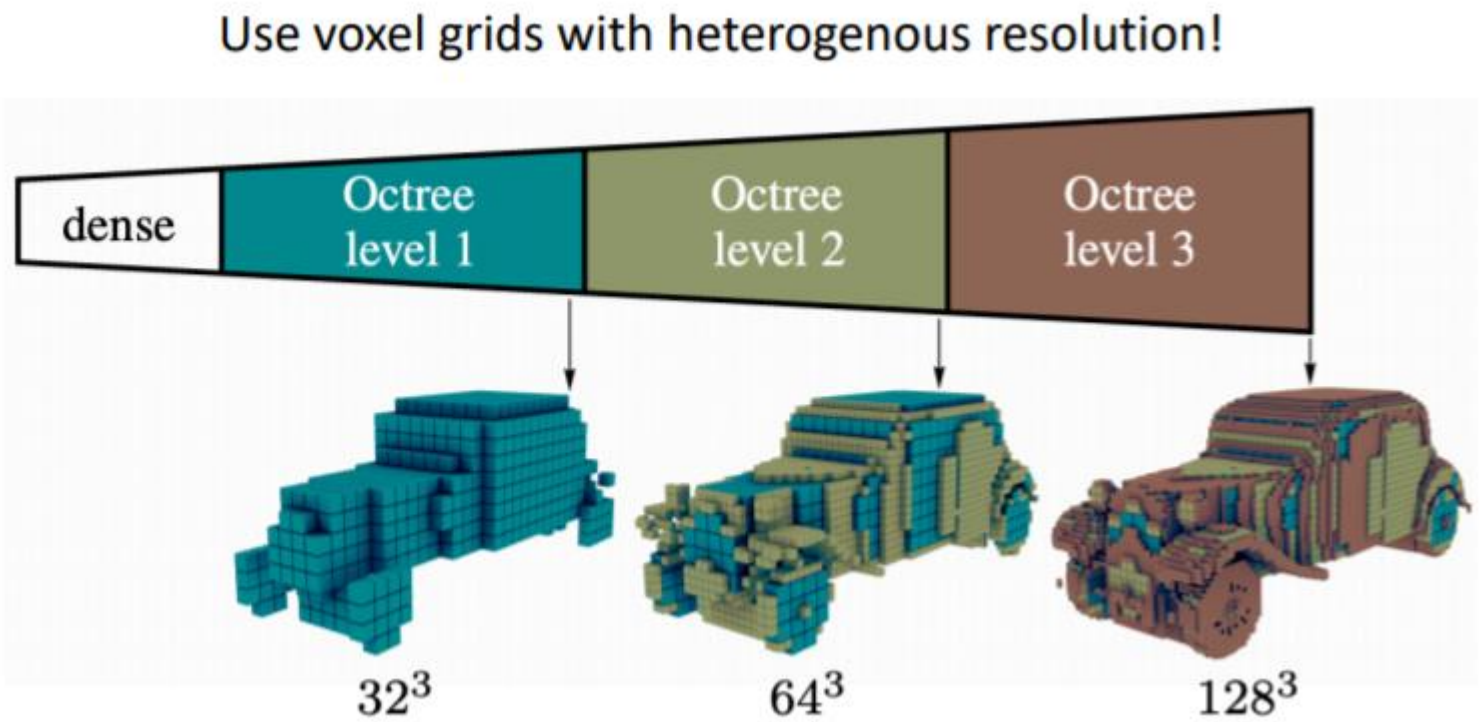
Train with per-voxel cross-entropy loss

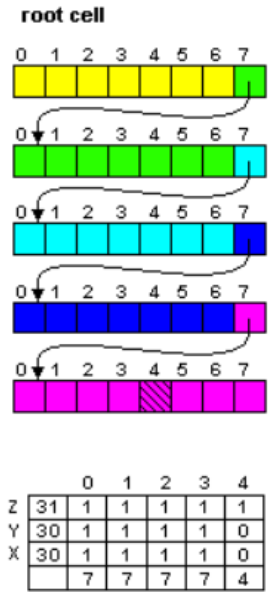
Voxel Problems: Memory Usage

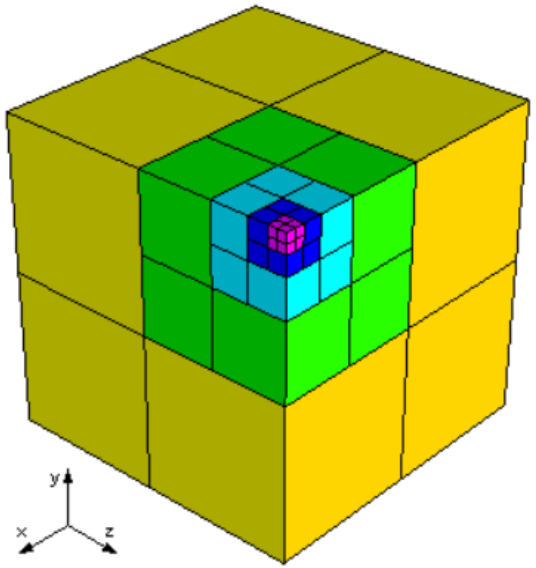


- Hard to use naïve voxel grids for high spatial resolutions
- There are some methods to scaling up the voxel grid

Scaling Voxels: Oct-Trees



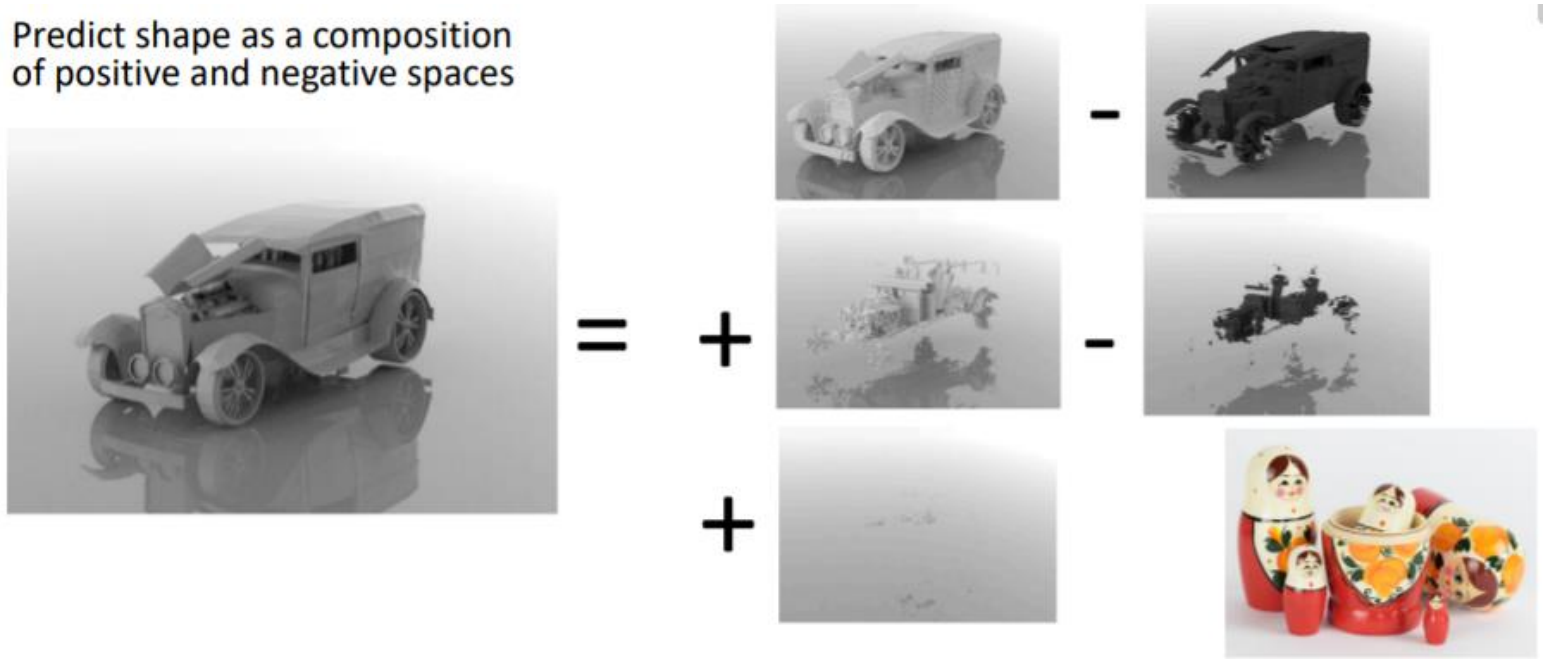




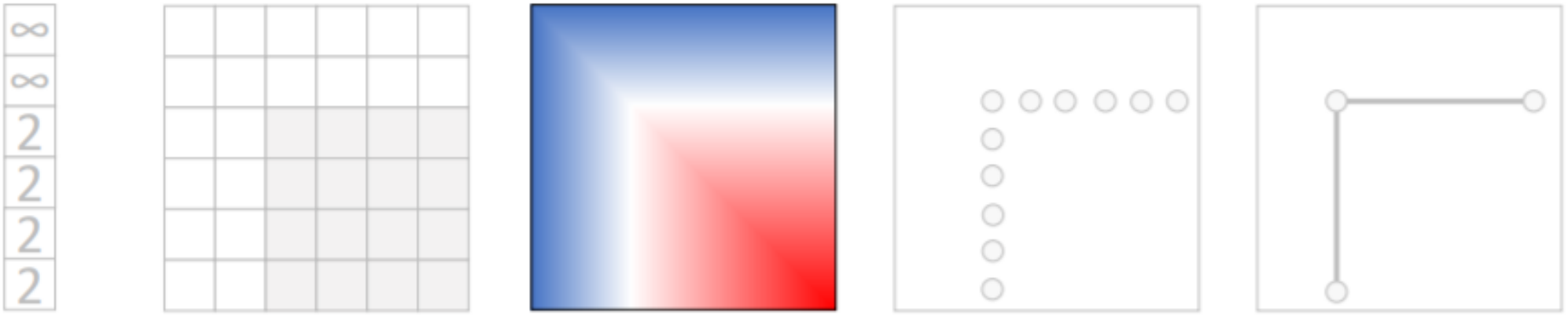
- Multiple resolution for voxel grid
- Generate coarse voxels, Fine voxels for fine details (higher resolution)

Scaling Voxels: Nested Shape Layers

Figure 4. **Composing shapes from nested shape layers.** The proposed method reconstructs a shape $S_{1:5}$ by iteratively adding (S_1,S_3,S_5) and subtracting (S_2,S_4) shape layers built from fused depth maps (a). This is akin to the layers of a Matryoshka doll (b).



3D Shape Representations: Implicit Functions



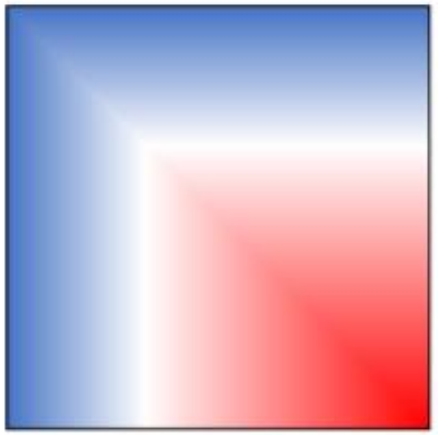
Depth Map | Voxel Grid | **Implicit Surface** | Pointcloud | Mesh

3D Shape Representations: Implicit Functions

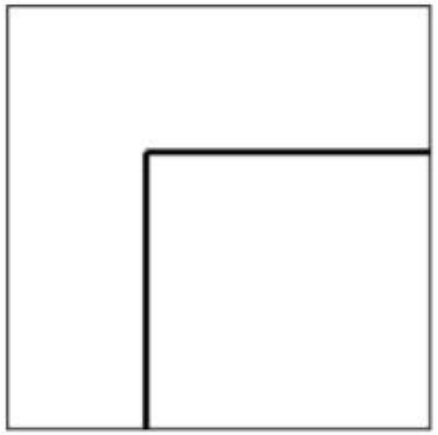
Learn a function to classify arbitrary 3D points as inside / outside the shape

$$o : \mathbb{R}^3 \rightarrow \{0, 1\}$$

The surface of the 3D object is the level set $\{x : o(x) = ½\}$



Implicit function

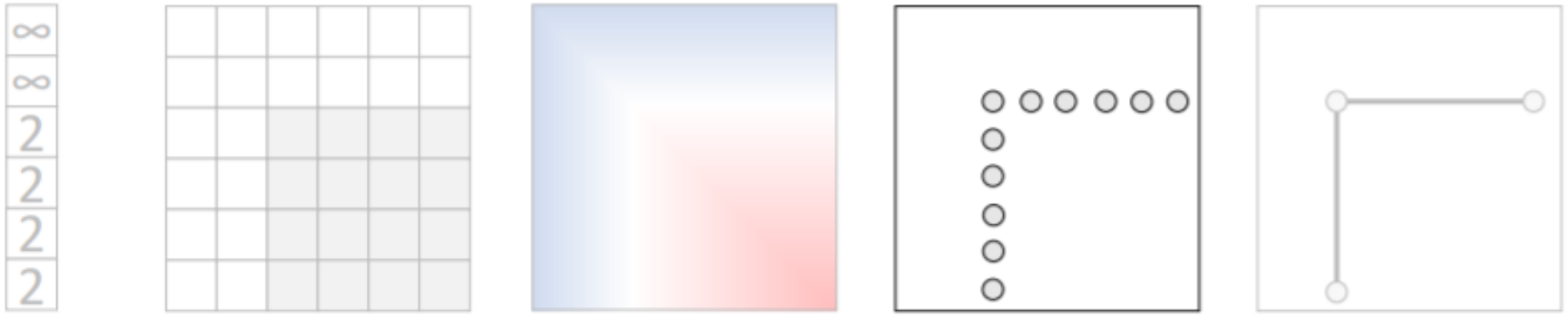


Explicit Shape

Same idea: **signed distance function (SDF)** gives the Euclidean distance to the surface of the shape; sign gives inside / outside

- $3D\ space\ coordinate \rightarrow$ probability that position that arbitrary position is either occupied or not occupied by the object
- $o(x) = \frac{1}{2} \rightarrow$ object surface

3D Shape Representations: Point Cloud

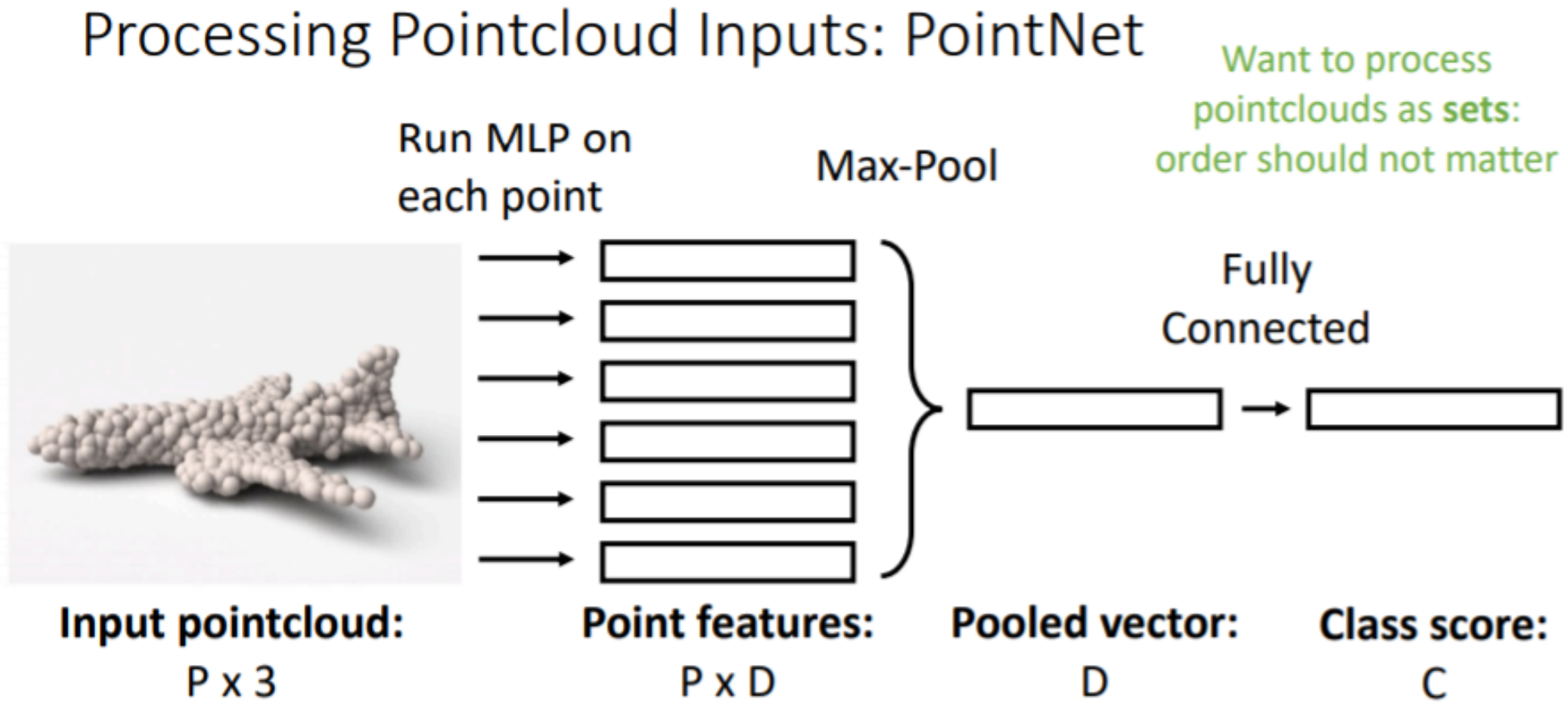


Depth Map | Voxel Grid | Implicit Surface | **Pointcloud** | Mesh

3D Shape Representations: Point Cloud

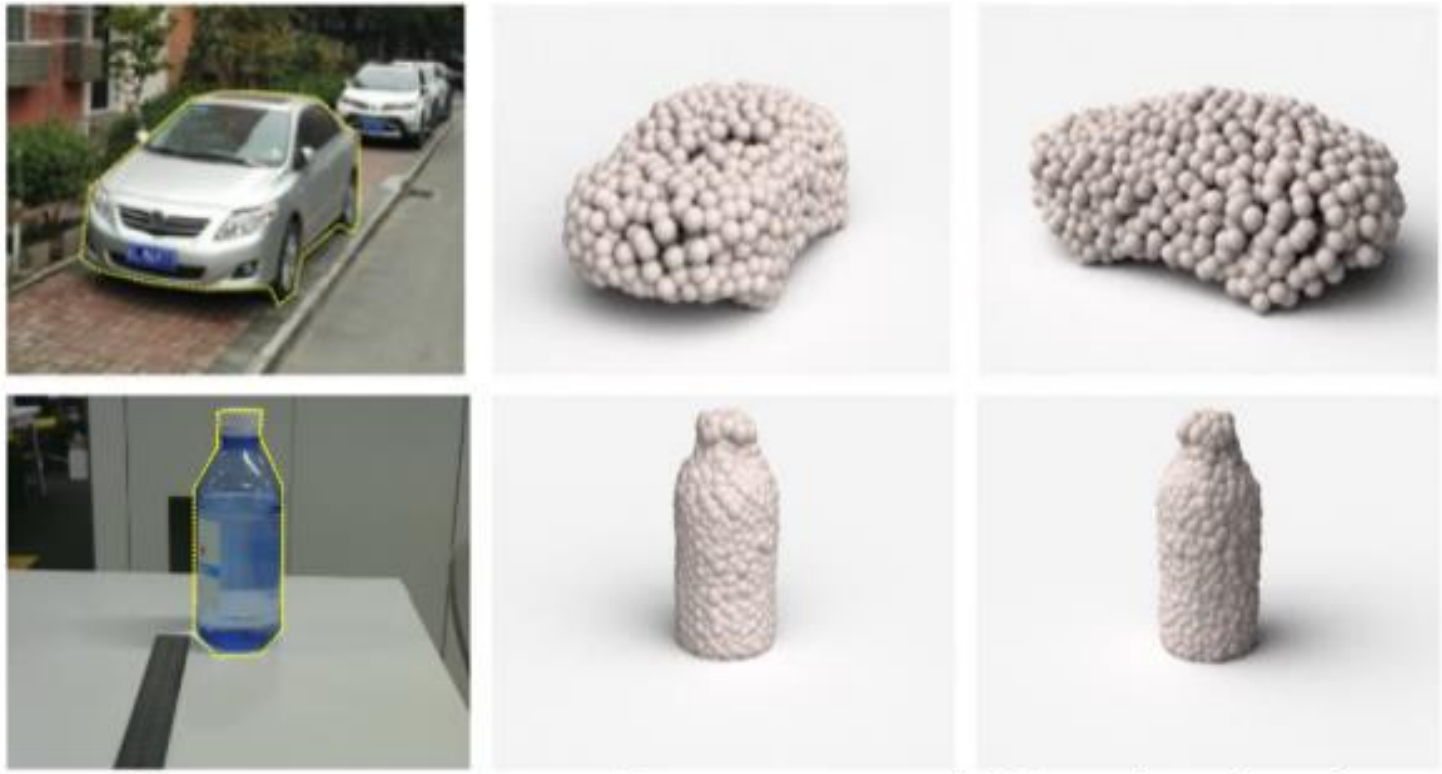
- Represent shape as a set of P points in 3D space
- (+) Can represent fine structures without huge numbers of points
- ( ) Requires new architecture, losses, etc
- (-) Doesn't explicitly represent the surface of the shape: extracting a mesh for rendering or other applications requires post-processing

Processing Pointcloud Inputs: PointNet

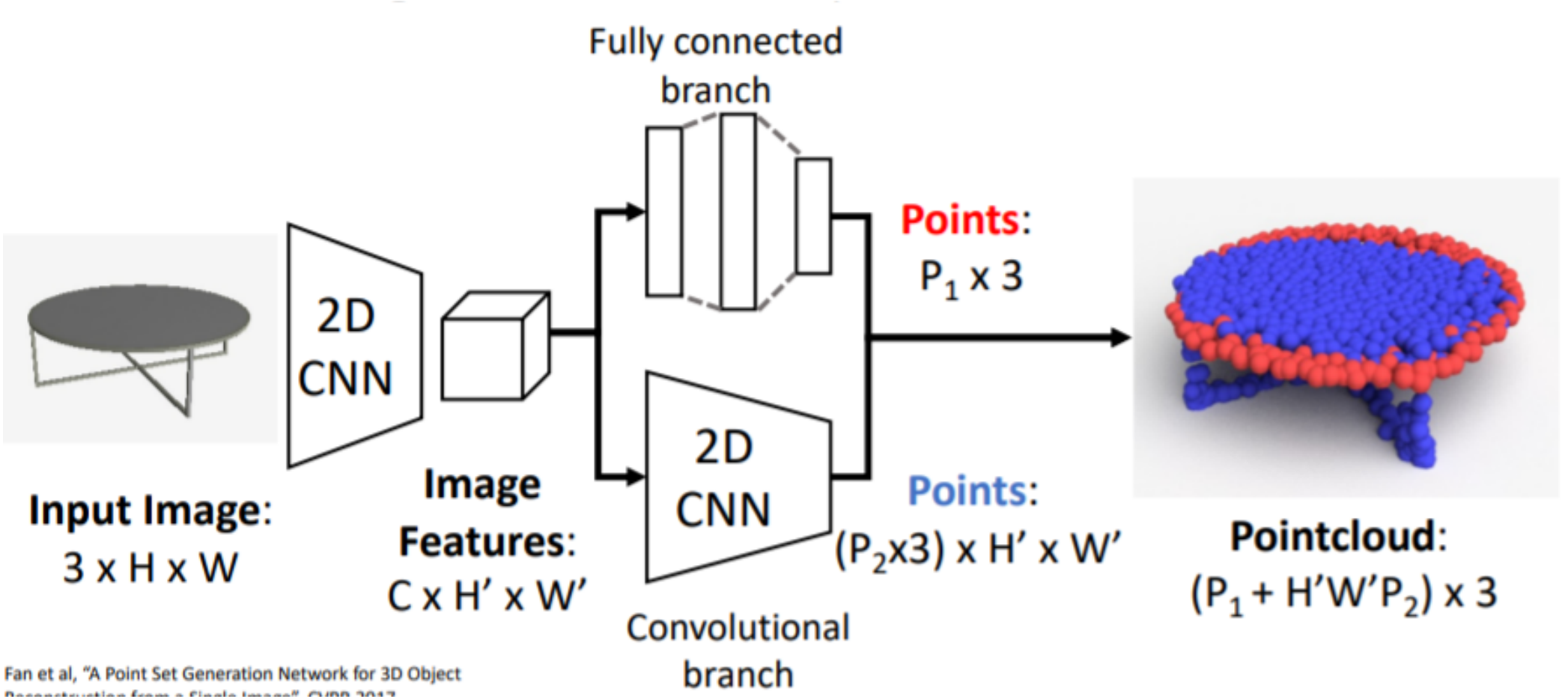


Input P $(points) \times 3$ $(x, y, z) \rightarrow$ **Output** $class\ score$

Generating Pointcloud Outputs



Generating Pointcloud Outputs



Fan et al, "A Point Set Generation Network for 3D Object

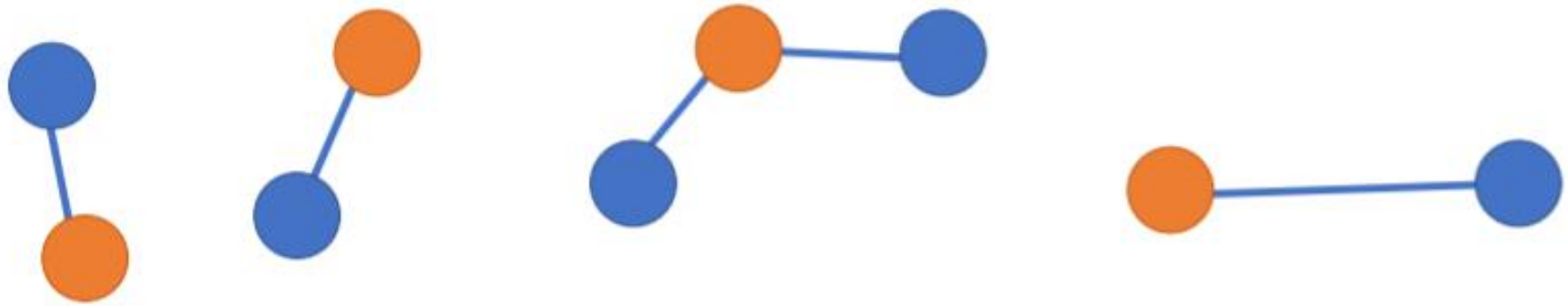
Predicting Point Clouds: Loss Function

Chamfer Distance

$$d_{CD}(S_1, S_2) = \sum_{x \in S_1} \min_{y \in S_2} \|x - y\|_2^2 + \sum_{y \in S_2} \min_{x \in S_1} \|x - y\|_2^2$$

Chamfer distance is the sum of L2 distance to each point's nearest neighbor in the other set

$$d_{CD}(S_1, S_2) = \sum_{x \in S_1} \min_{y \in S_2} \|x - y\|_2^2 + \sum_{y \in S_2} \min_{x \in S_1} \|x - y\|_2^2$$



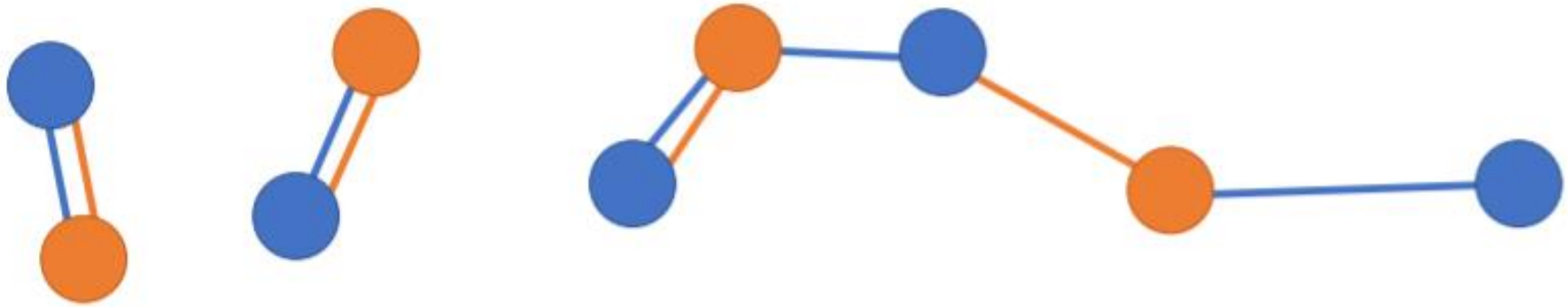
Predicting Point Clouds: Loss Function

Chamfer Distance

$$d_{CD}(S_1, S_2) = \sum_{x \in S_1} \min_{y \in S_2} \|x - y\|_2^2 + \sum_{y \in S_2} \min_{x \in S_1} \|x - y\|_2^2$$

Chamfer distance is the sum of L2 distance to each point's nearest neighbor in the other set

$$d_{CD}(S_1, S_2) = \sum_{x \in S_1} \min_{y \in S_2} \|x - y\|_2^2 + \sum_{y \in S_2} \min_{x \in S_1} \|x - y\|_2^2$$



3D Shape Representations: Triangle Mesh

Depth Map | Voxel Grid | Implicit Surface | Pointcloud | **Mesh**

3D Shape Representations: Triangle Mesh

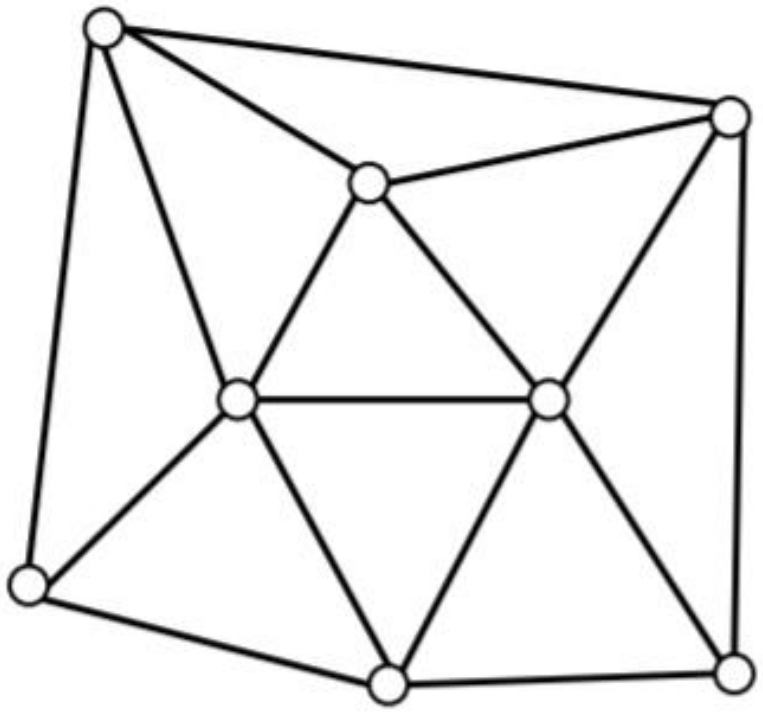
Represent a 3D shape as a set of triangles

Vertices: Set of V points in 3D space

Faces: Set of triangles over the vertices

(+) Standard representation for graphics

(+) Explicitly represents 3D shapes



3D Shape Representations: Triangle Mesh

Represent a 3D shape as a set of triangles

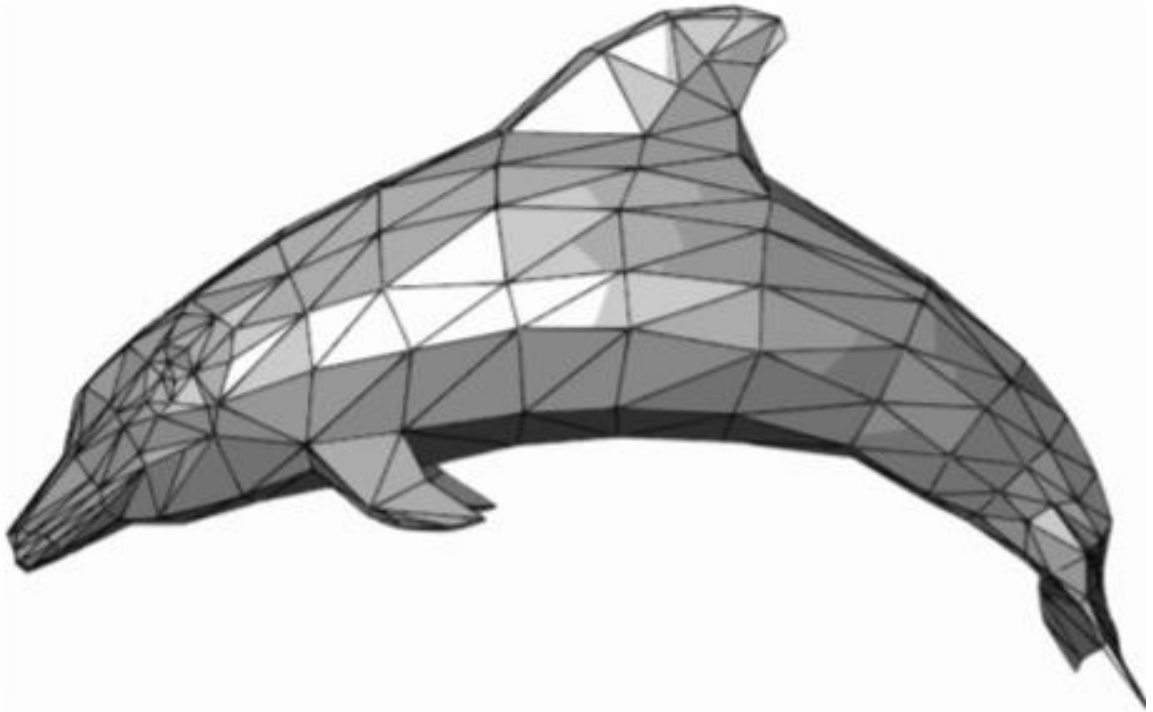
Vertices: Set of V points in 3D space

Faces: Set of triangles over the vertices

(+) Standard representation for graphics

(+) Explicitly represents 3D shapes

(+) Adaptive: Can represent flat surfaces very efficiently, can allocate more faces to areas with fine detail



3D Shape Representations: Triangle Mesh

Represent a 3D shape as a set of triangles

Vertices: Set of V points in 3D space

Faces: Set of triangles over the vertices

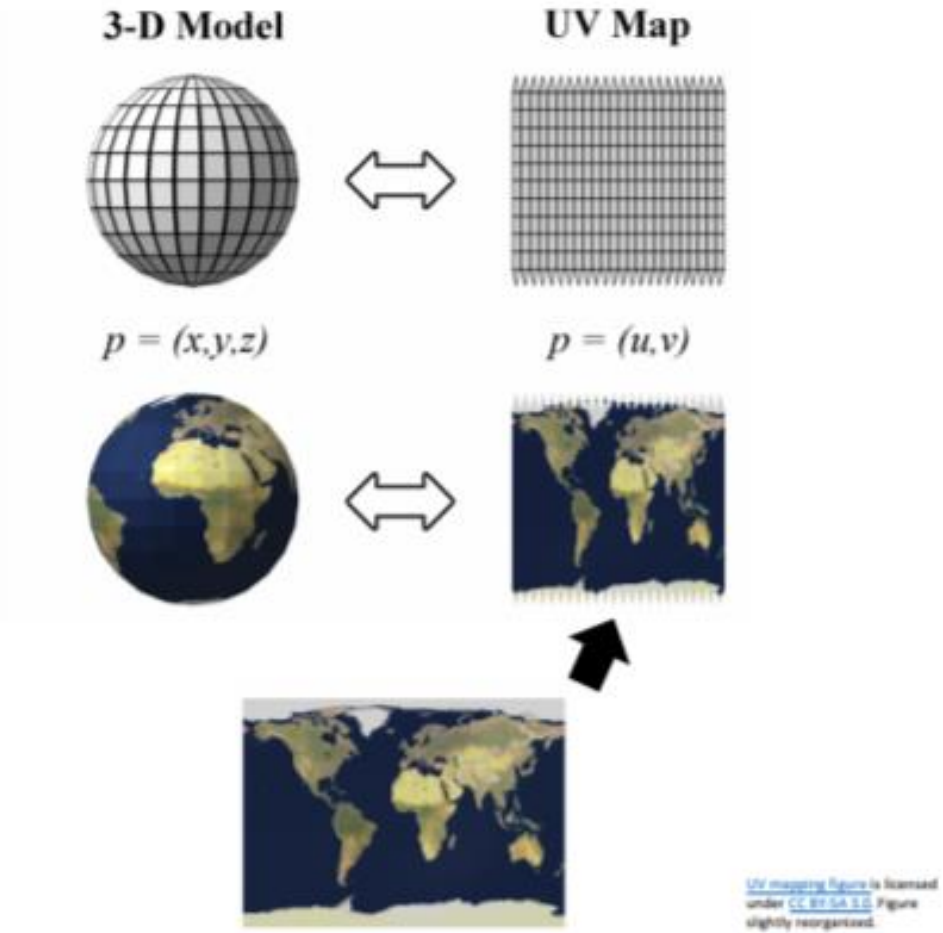
(+) Standard representation for graphics

(+) Explicitly represents 3D shapes

(+) Adaptive: Can represent flat surfaces very efficiently, can allocate more faces to areas with fine detail

(+) Can attach data on verts and interpolate over the whole surface: RGB colors, texture coordinates, normal vectors, etc.

(-) Nontrivial to process with neural nets!

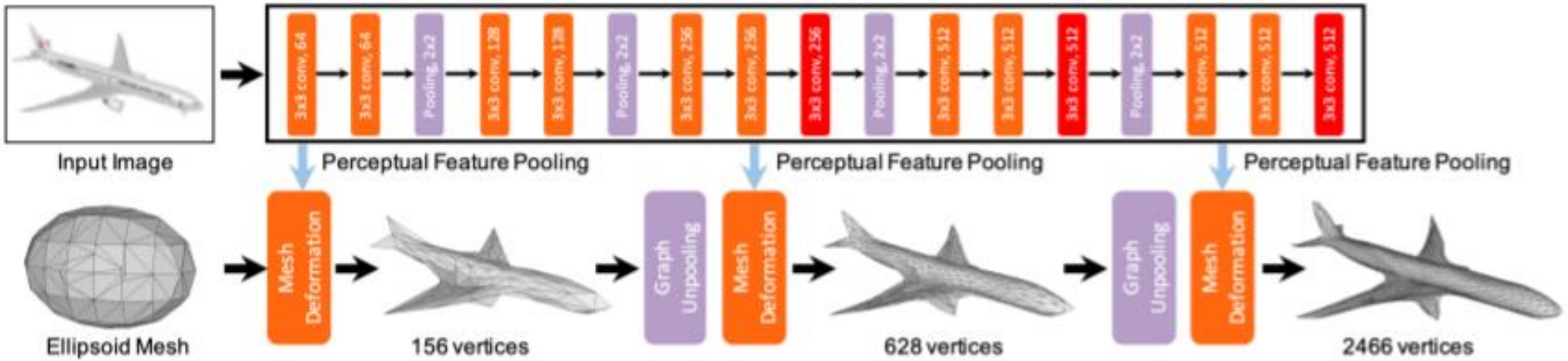


Predicting Meshes: Pixel2Mesh

Input: Single RGB Image of an object

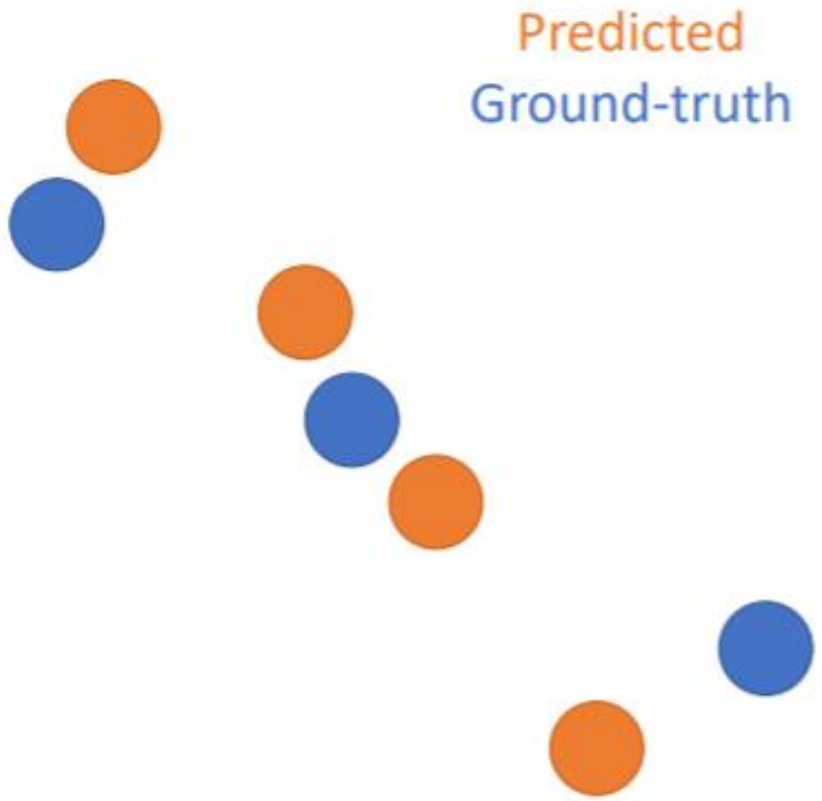
Key ideas:
Iterative Refinement
Graph Convolution
Vertex Aligned-Features
Chamfer Loss Function

Output: Triangle mesh for the object



Shape Comparison Metrics: F1 Score

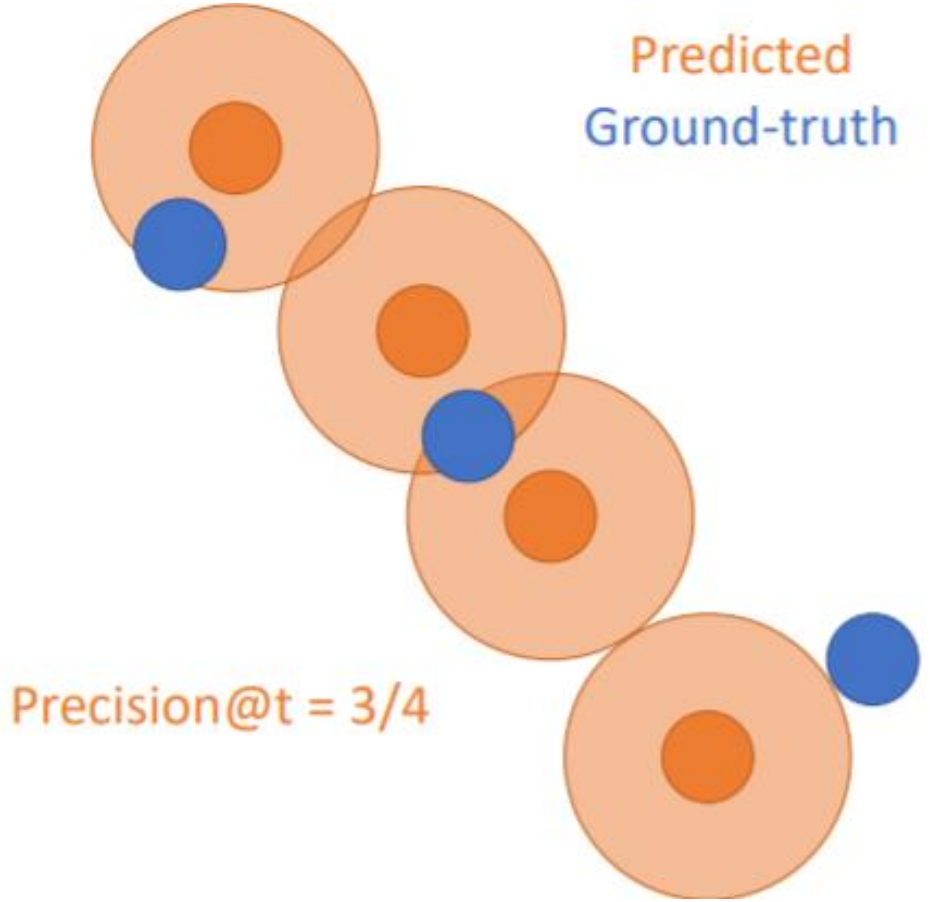
Similar to Chamfer, sample points from the surface of the prediction and the ground-truth



Shape Comparison Metrics: F1 Score

Similar to Chamfer, sample points from the surface of the prediction and the ground-truth

Precision@t = fraction of predicted points within t of some ground-truth point



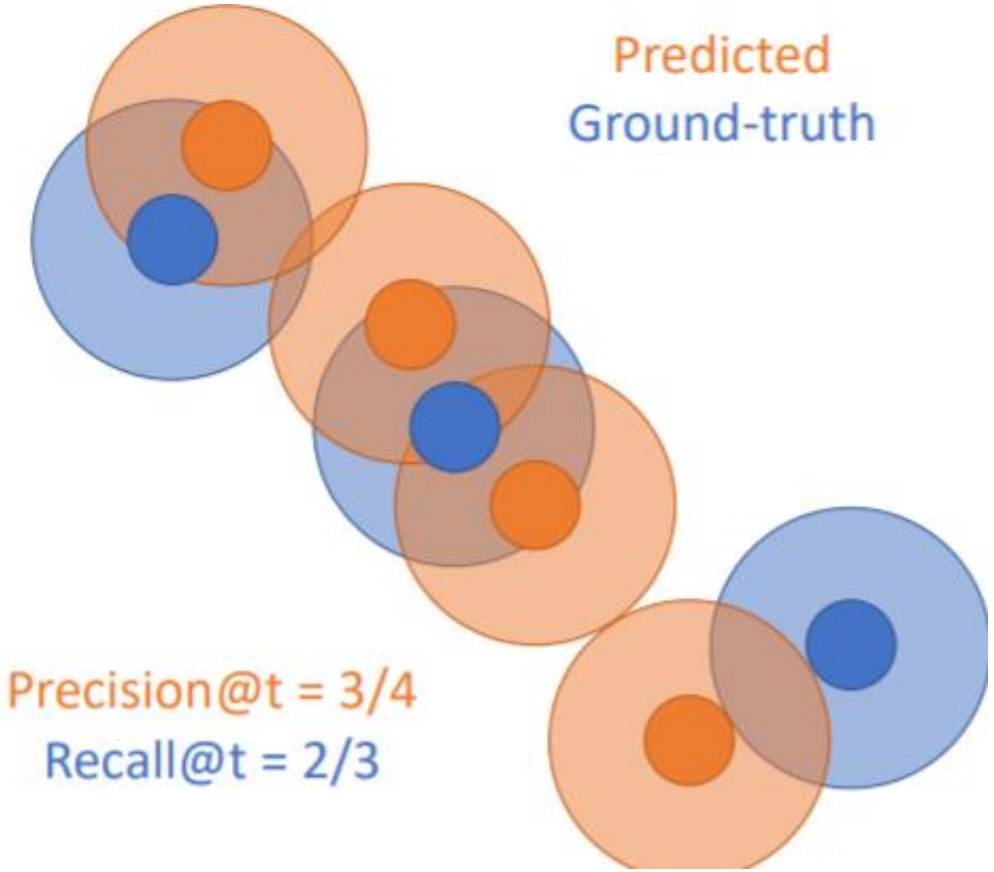
Assign a circle of a certain size to **predicated points** and set the point to true predefined points if a GT point enters here.

Shape Comparison Metrics: F1 Score

Similar to Chamfer, sample points from the surface of the prediction and the ground-truth

Precision@t = fraction of predicted points within t of some ground-truth point

Recall@t = fraction of ground-truth points within t of some predicted point



Assign a circle of a certain size to **GT points**, and if a predicated point comes in here, true.

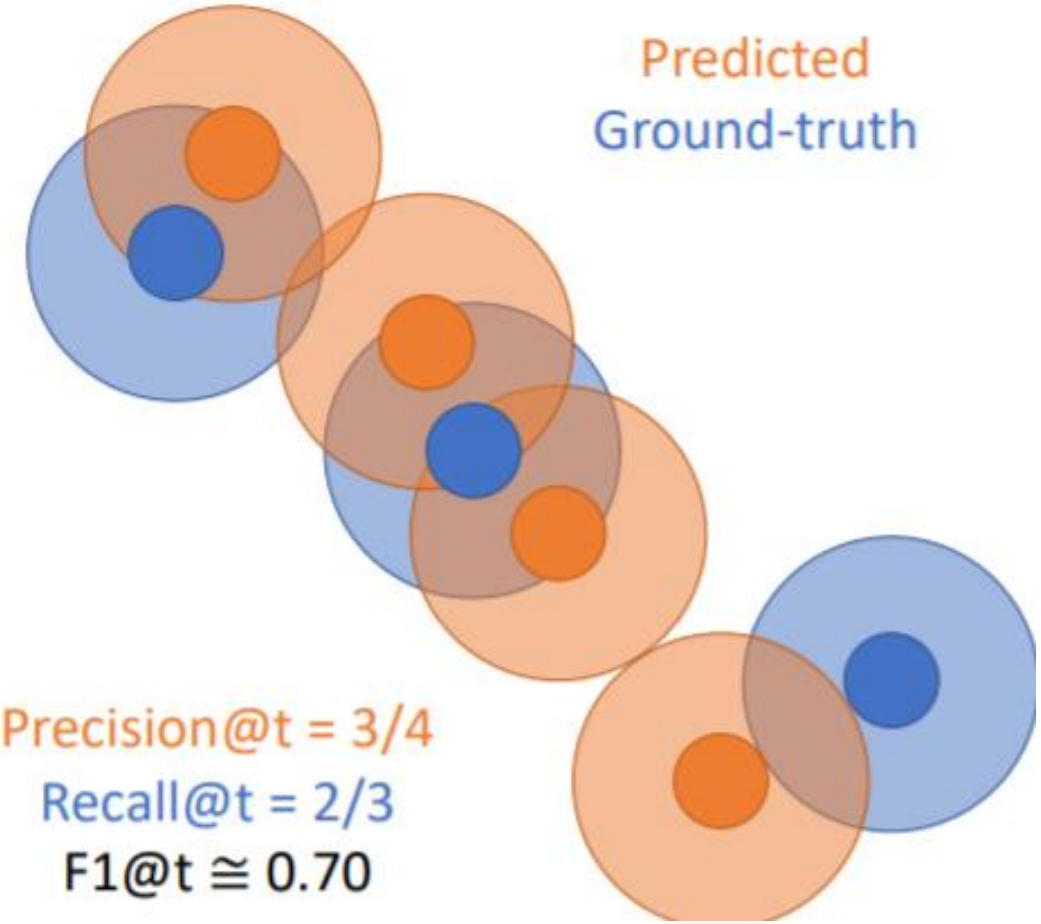
Shape Comparison Metrics: F1 Score

Similar to Chamfer, sample points from the surface of the prediction and the ground-truth

Precision@t = fraction of predicted points within t of some ground-truth point

Recall@t = fraction of ground-truth points within t of some predicted point

$$F1@t = 2 * \frac{Precision@t * Recall@t}{Precision@t + Recall@t}$$



Shape Comparison Metrics: F1 Score

Similar to Chamfer, sample points from the surface of the prediction and the ground-truth

Precision@t = fraction of predicted points within t of some ground-truth point

Recall@t = fraction of ground-truth points within t of some predicted point

$$F1@t = 2 * \frac{Precision@t * Recall@t}{Precision@t + Recall@t}$$

F1 score is robust to outliers!

Conclusion: F1 score is probably the best shape prediction metric in common use

Figure credit: Tatarchenko et al, "What Do Single-view 3D Reconstruction Networks Learn?", CVPR 2019

3D Datasets

ShapeNet

club chair | cantilever chair | armchair

~50 categories, ~50k 3D CAD models

Standard split has 13 categories, ~44k models, 25 rendered images per model

Many papers show results here

(-) Synthetic, isolated objects; no context

(-) Lots of chairs, cars, airplanes

Chang et al, "ShapeNet: An Information-Rich 3D Model Repository", arXiv 2015
Choy et al, "3D-R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction", ECCV 2016

Pix3D

9 categories, 219 3D models of IKEA furniture aligned to ~17k real images

Some papers train on ShapeNet and show qualitative results here, but use ground-truth segmentation masks

(+) Real images! Context!

(-) Small, partial annotations – only 1 obj/image

Sun et al, "Pix3D: Dataset and Methods for Single-Image 3D Shape Modeling", CVPR 2018

Mesh R-CNN

Input: Single RGB image

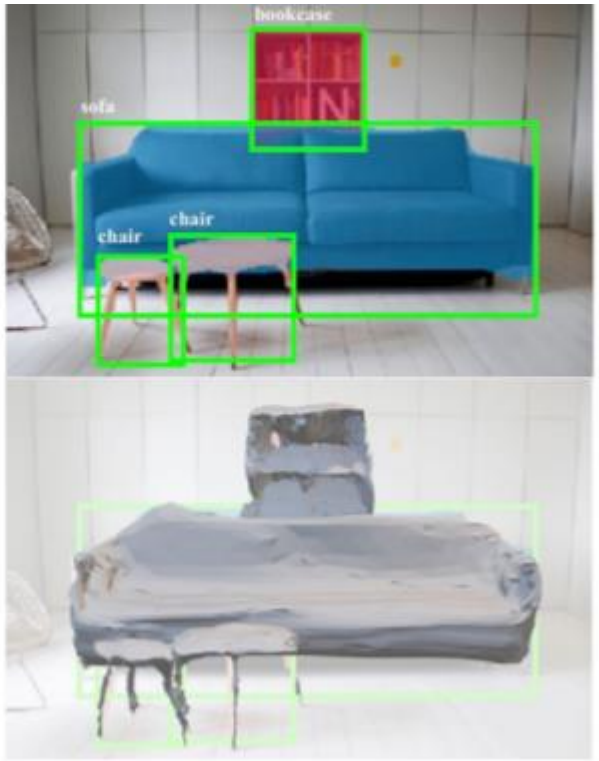
Output:

A set of detected objects

For each object:

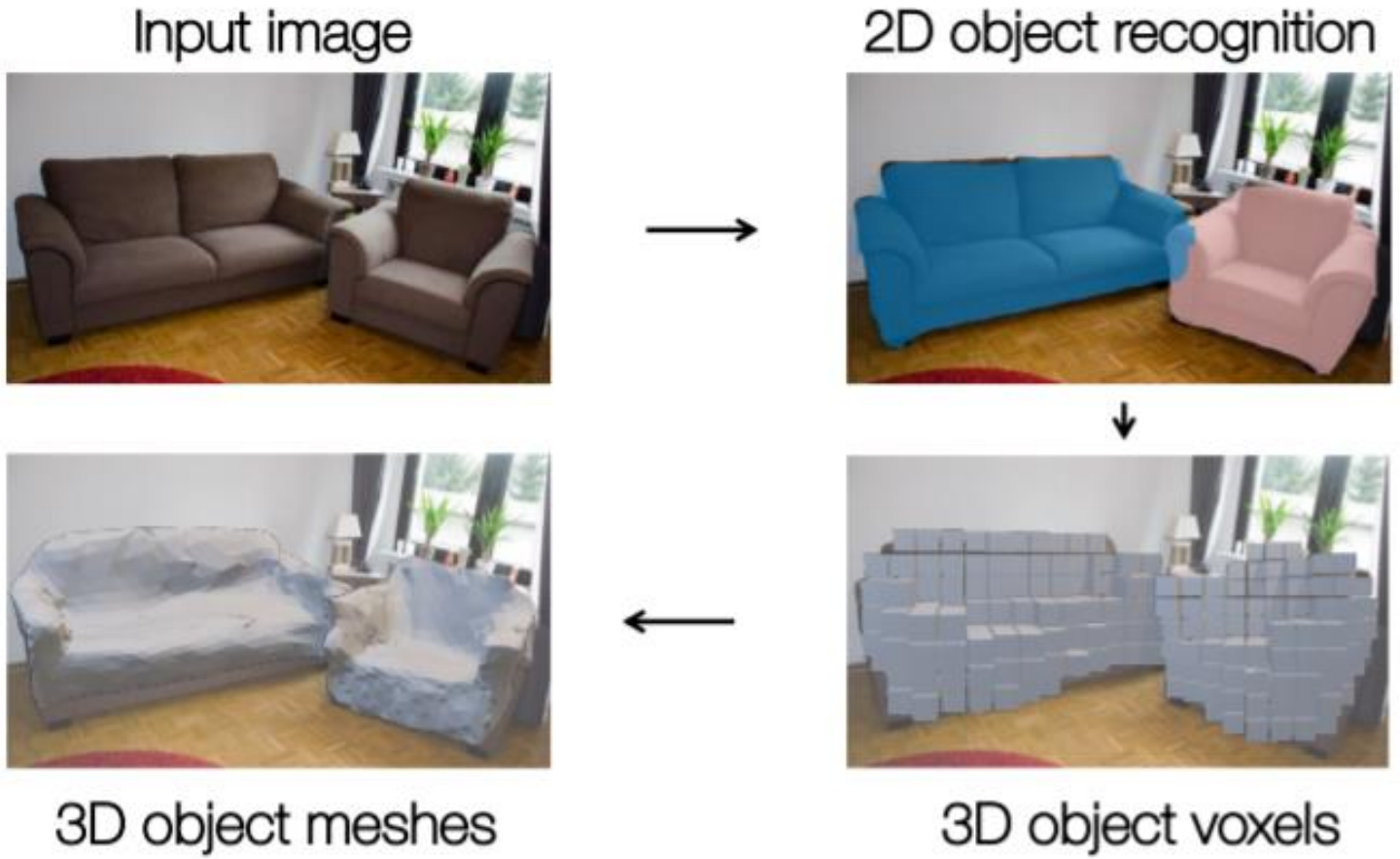
Mask R-CNN
- Bounding box
- Category label
- Instance segmentation

Mesh head
- 3D triangle mesh

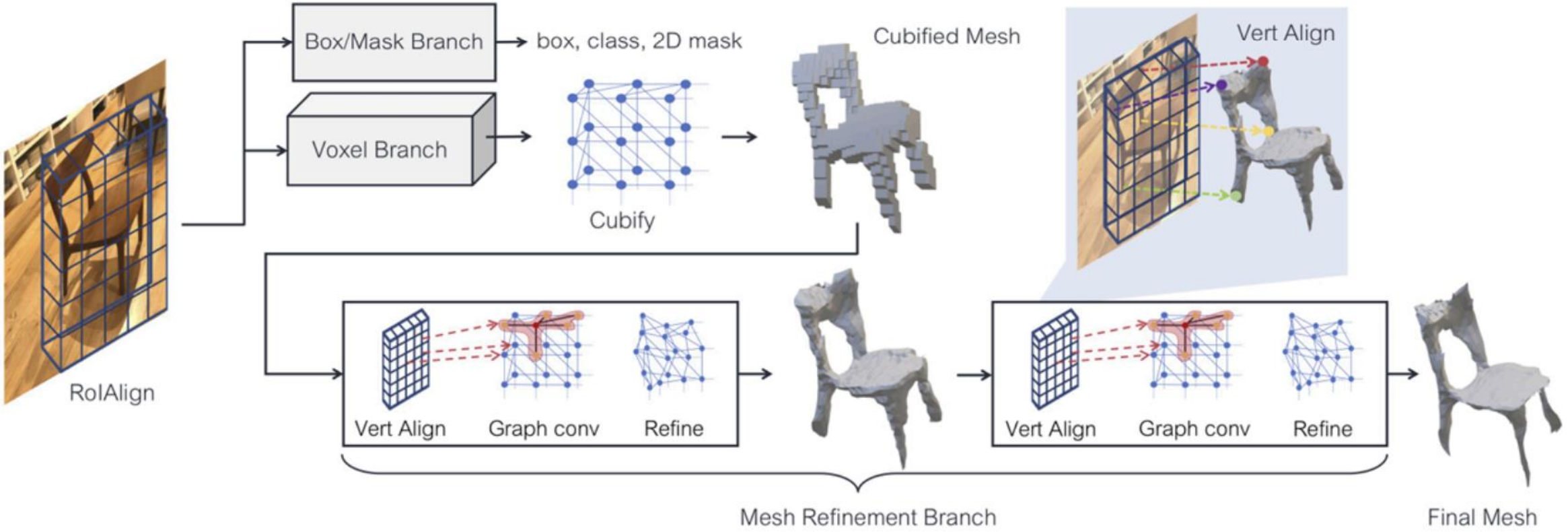


Mesh R-CNN

Mesh R-CNN

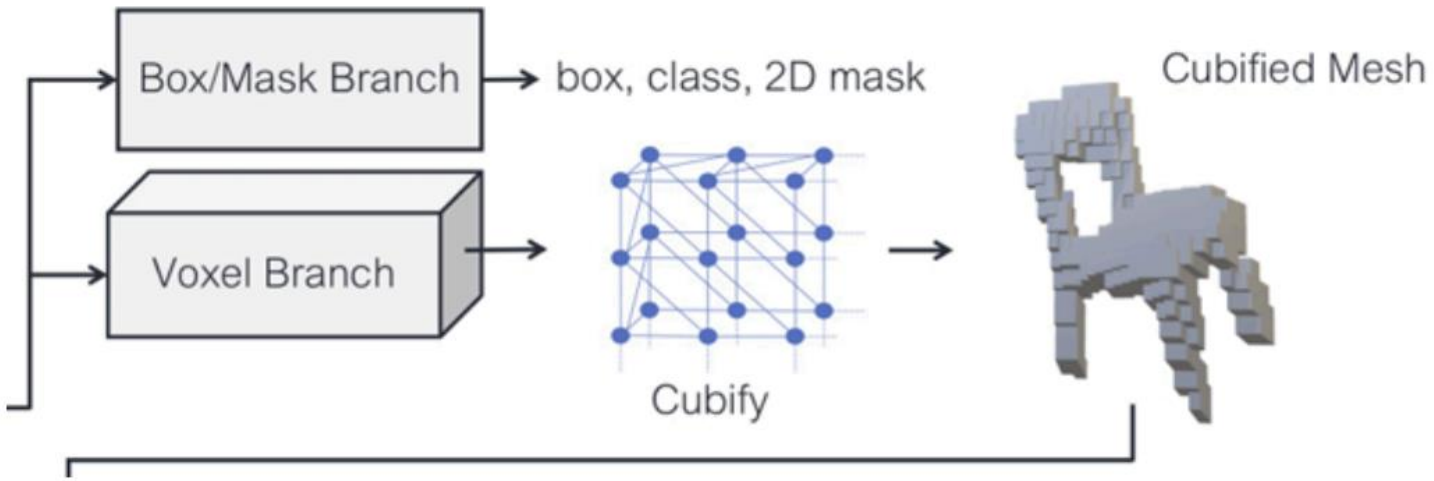

Input image
2D object recognition
3D object meshes
3D object voxels

Mesh R-CNN: Method



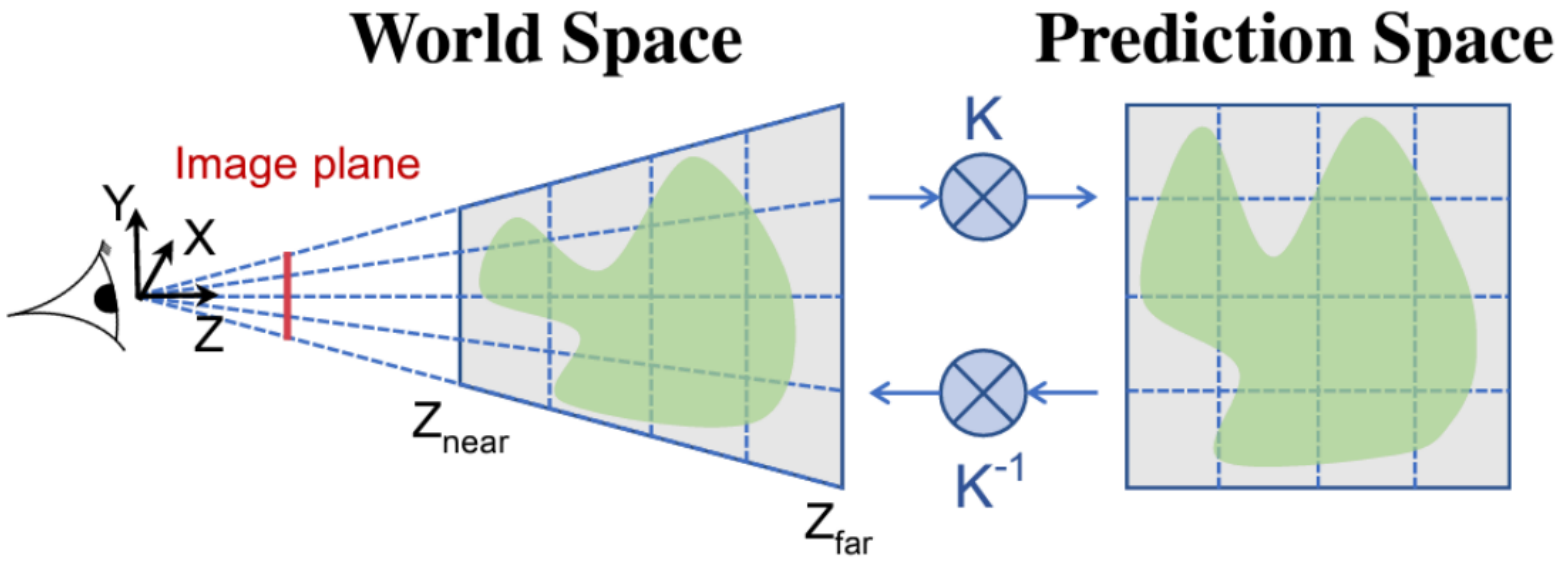
The **voxel branch** predicts a **coarse shape** for each detected object which is further deformed with a sequence of refinement stages in the **mesh refinement branch**

Mesh R-CNN: Voxel Branch



- The **voxel branch** predicts **a grid of voxel occupancy probabilities**.
- Rather than predicting a M×M grid giving the object's shape in the image plane, we instead **predict a G×G×G grid giving the object's full 3D shape.**
- Each occupied voxel is **replaced with a cubified triangle mesh** with 8 vertices, 18 edges, and 12 faces.

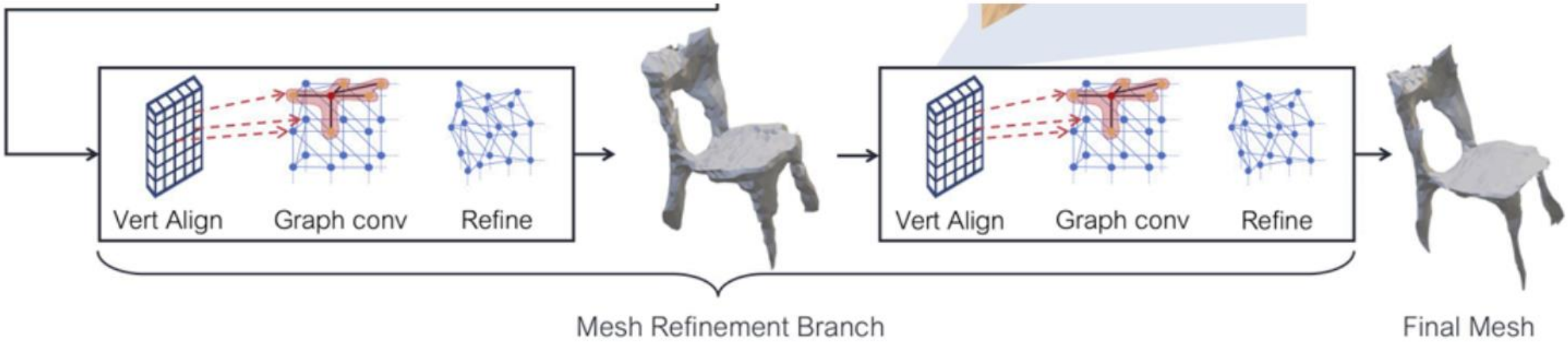
Mesh R-CNN: Voxel Branch



Objects become smaller as they recede from the camera.

Achieved this effect by **making voxel predictions in a space that is transformed by the camera's (known) intrinsic matrix K.**

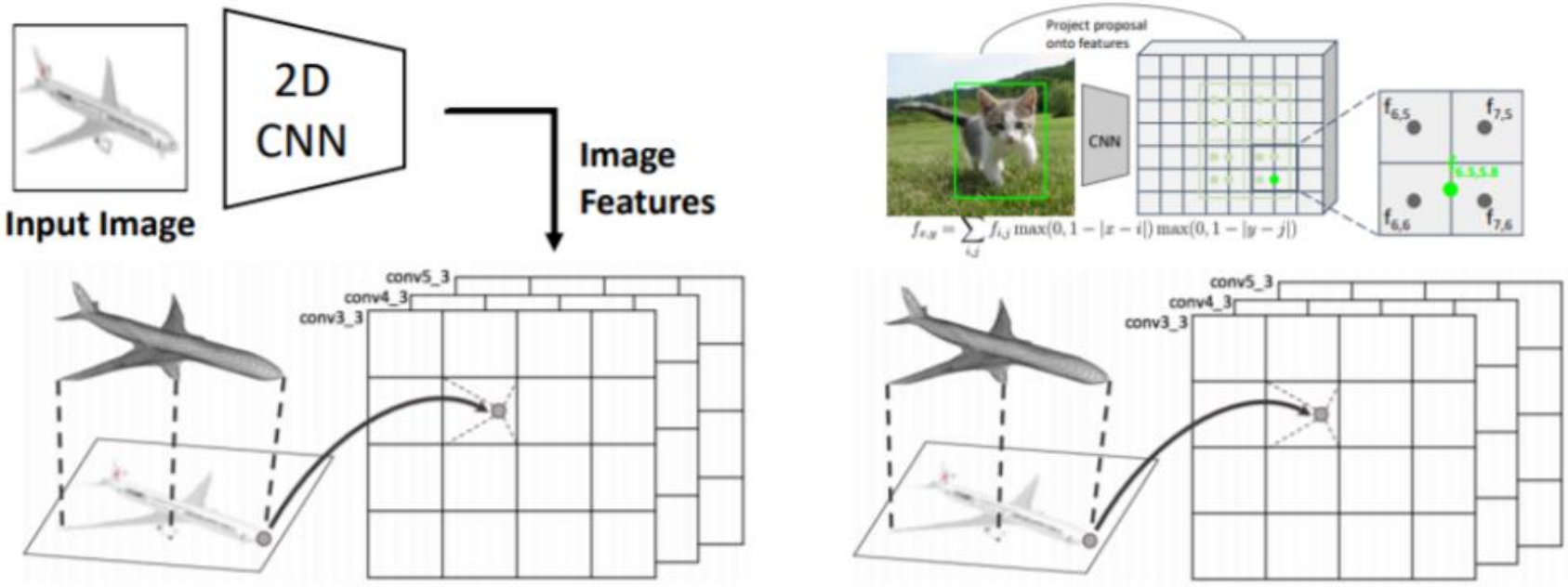
Mesh R-CNN: Mesh Refinement Branch



The mesh refinement branch **processes this initial cubified mesh**, **refining its vertex positions with a sequence of refinement stages**

- **Vertex alignment** : extracts image features for vertices
- **Graph convolution** : propagates information along mesh edges
- **Vertex refinement** : updates vertex positions

Mesh R-CNN: Vertex Alignment



- Project mesh's verticies to image plane (Image Feature map)
- Bilinear Interpolation -> exact image feature vector

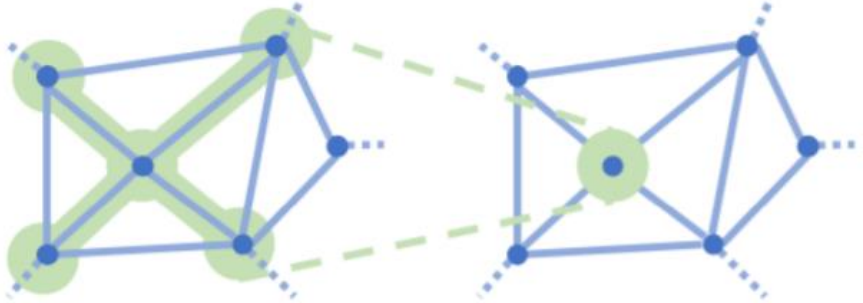
Mesh R-CNN: Vertex Graph Convolution

$$f'_i = W_0 f_i + \sum_{j \in \mathcal{N}(i)} W_1 f_j$$

Vertex v_i has feature f_i

New feature f'_i for vertex vi depends on feature of neighboring vertices N(i)

Use same weights W0 and W1 to compute all outputs



Input: Graph with a feature vector at each vertex

Output: New feature vector for each vertex

- **Input**: Graph that contains feature vector f_i for each vertex.
- **Output**: new feature vector f_i' for each vertex

Mesh R-CNN: Vertex Refinement

$$v_i' = v_i + \tanh(W_{vert}\,[\bar{f}_i; v_i])$$

Vertex Refinement **computes updated vertex positions** where W_vert is a learned weight matrix. This updates the mesh geometry, keeping its topology fixed.

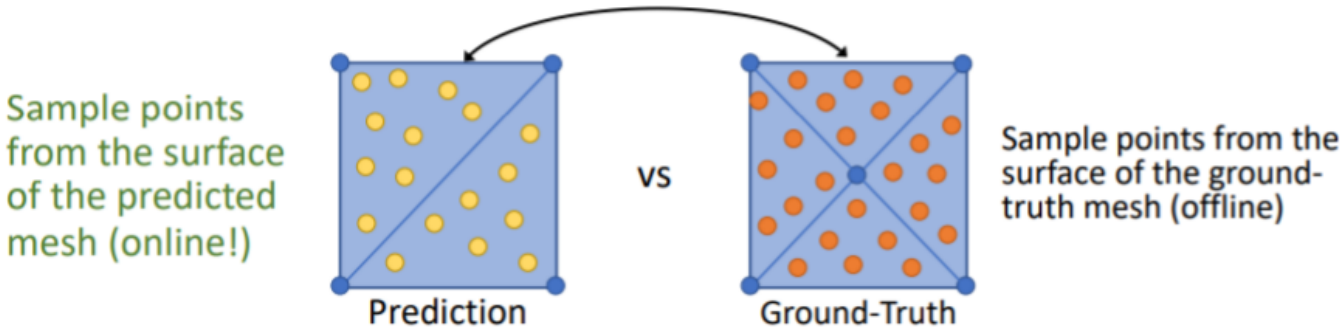
Mesh R-CNN: Loss

Method - Mesh Predictor

Mesh Losses

Defining losses that operate natively on triangle meshes is challenging, so we instead use **loss functions defined over a finite set of points**. We represent a **mesh with a pointcloud by densely sampling its surface**. Consequently, a **pointcloud loss approximates a loss over shapes**.

Loss = Chamfer distance between predicted samples and ground-truth samples



$$\mathcal{L}_{\text{cham}}(P,Q) = |P|^{-1} \sum_{(p,q)\in\Lambda_{P,Q}} \|p-q\|^2 + |Q|^{-1} \sum_{(q,p)\in\Lambda_{Q,P}} \|q-p\|^2$$

$$\mathcal{L}_{\text{norm}}(P,Q) = -|P|^{-1} \sum_{(p,q)\in\Lambda_{P,Q}} |u_p \cdot u_q| - |Q|^{-1} \sum_{(q,p)\in\Lambda_{Q,P}} |u_q \cdot u_p|$$

Mesh R-CNN: Loss

Method - Mesh Predictor

Edge loss

The **chamfer** and **normal distances** penalize mismatched positions and normals between two pointclouds, but minimizing these distances alone results in **degenerate meshes**.

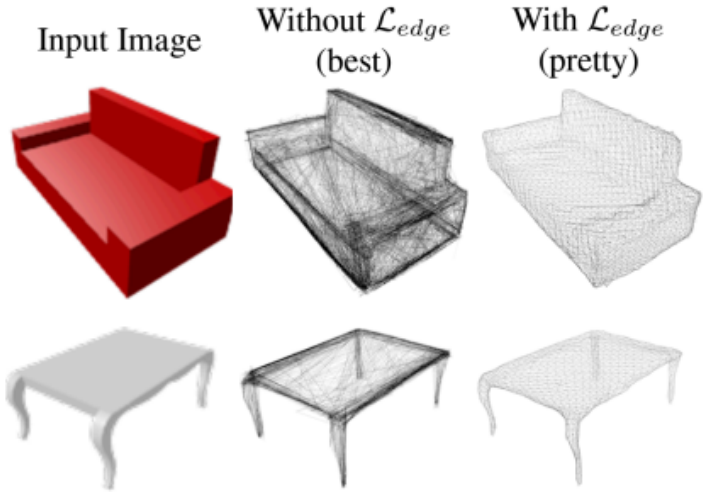


Figure 5. Training without the edge length regularizer $\mathcal{L}_{edge}$ results in degenerate predicted meshes that have many overlapping faces. Adding $\mathcal{L}_{edge}$ eliminates this degeneracy but results in worse agreement with the ground-truth as measured by standard metrics such as Chamfer distance.

High-quality mesh predictions require additional **shape regularizers**

$$\mathcal{L}_{\text{edge}}(V, E) = \frac{1}{|E|} \sum_{(v,v') \in E} \|v - v'\|^2$$

The mesh loss of the i-th stage is a weighted sum of $\mathcal{L}_{\text{cham}}(P^i, P^{gt})$, $\mathcal{L}_{\text{norm}}(P^i, P^{gt})$ and $\mathcal{L}_{\text{edge}}(V^i, E^i)$

Mesh R-CNN: Loss

Method - Mesh Predictor

Edge loss

The **chamfer** and **normal distances** penalize mismatched positions and normals between two pointclouds, but minimizing these distances alone results in **degenerate meshes**.

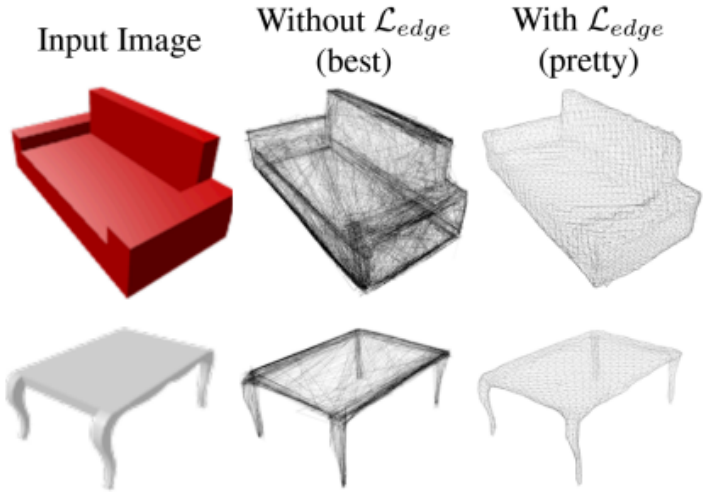


Figure 5. Training without the edge length regularizer $\mathcal{L}_{edge}$ results in degenerate predicted meshes that have many overlapping faces. Adding $\mathcal{L}_{edge}$ eliminates this degeneracy but results in worse agreement with the ground-truth as measured by standard metrics such as Chamfer distance.

High-quality mesh predictions require additional **shape regularizers**

$$\mathcal{L}_{\text{edge}}(V, E) = \frac{1}{|E|} \sum_{(v,v') \in E} \|v - v'\|^2$$

The mesh loss of the i-th stage is a weighted sum of $\mathcal{L}_{\text{cham}}(P^i, P^{gt})$, $\mathcal{L}_{\text{norm}}(P^i, P^{gt})$ and $\mathcal{L}_{\text{edge}}(V^i, E^i)$

Videos

AIKU

Table of Contents

Videos

Today: **Video** = 2D + Time

A video is a **sequence** of images
4D tensor: T x 3 x H x W
(or 3 x T x H x W)

Video Classification

Images: Recognize **objects**

Dog
Cat
Fish
Truck

Videos: Recognize **actions**

Swimming
Running
Jumping
Eating
Standing

Problem: Videos are big

Input video:
T x 3 x H x W

Videos are ~30 frames per second (fps)

Size of uncompressed video
(3 bytes per pixel):

SD (640 x 480): **~1.5 GB per minute**
HD (1920 x 1080): ~**10 GB per minute**

Solution: Train on short **clips:** low fps and low spatial resolution
e.g. T = 16, H=W=112
(3.2 seconds at 5 fps, 588 KB)

Training on Clips

Raw video: Long, high FPS

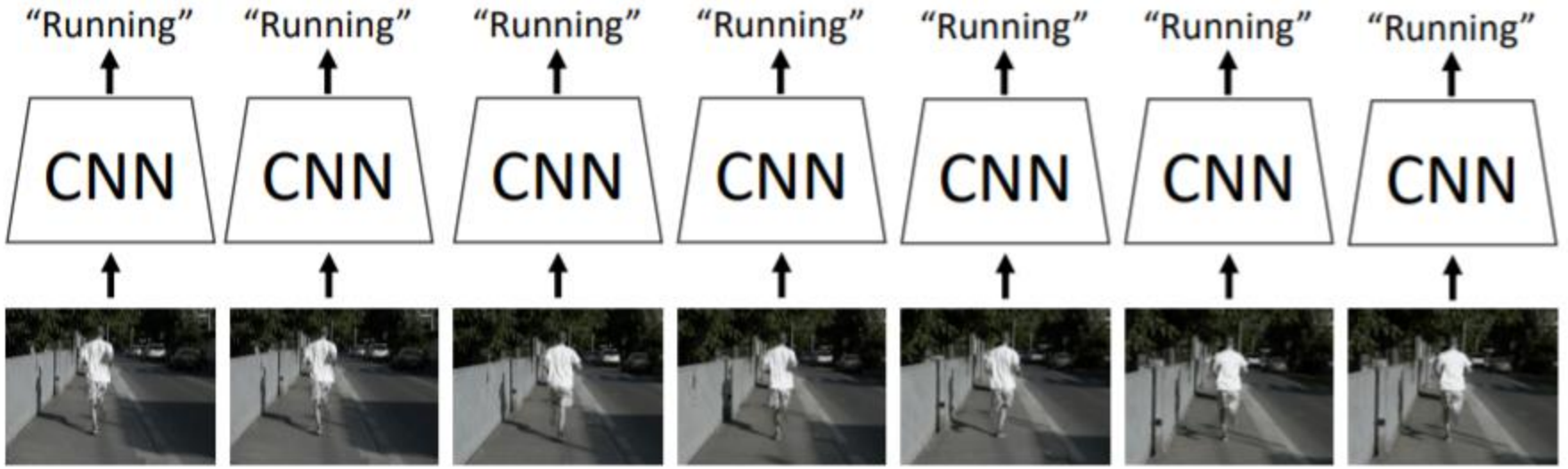
Training: Train model to classify short **clips** with low FPS

Testing: Run model on different clips, average predictions

- Raw video → clips (with low fps)
- Use same size of clips to classify in test time
- Classification for each clip, average prediction → Final output

Video Classification(1): Single-Frame CNN

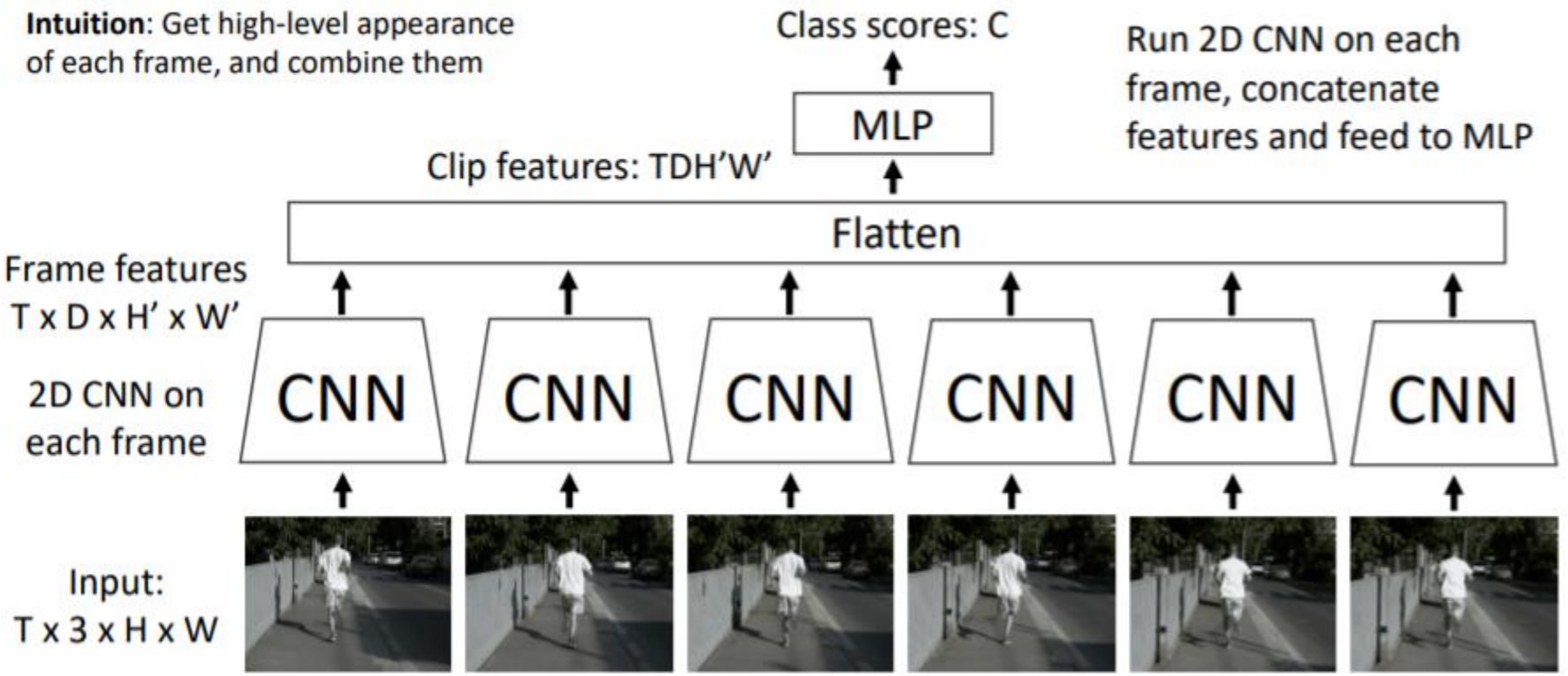
Simple idea: train normal 2D CNN to classify video frames independently!
(Average predicted probs at test-time)
Often a **very** strong baseline for video classification



Learn a basic 2D CNN that divides all frames in the video so that each 2D image can be classified independently.

- Abandoning the temporal nature of the video.
- Simple

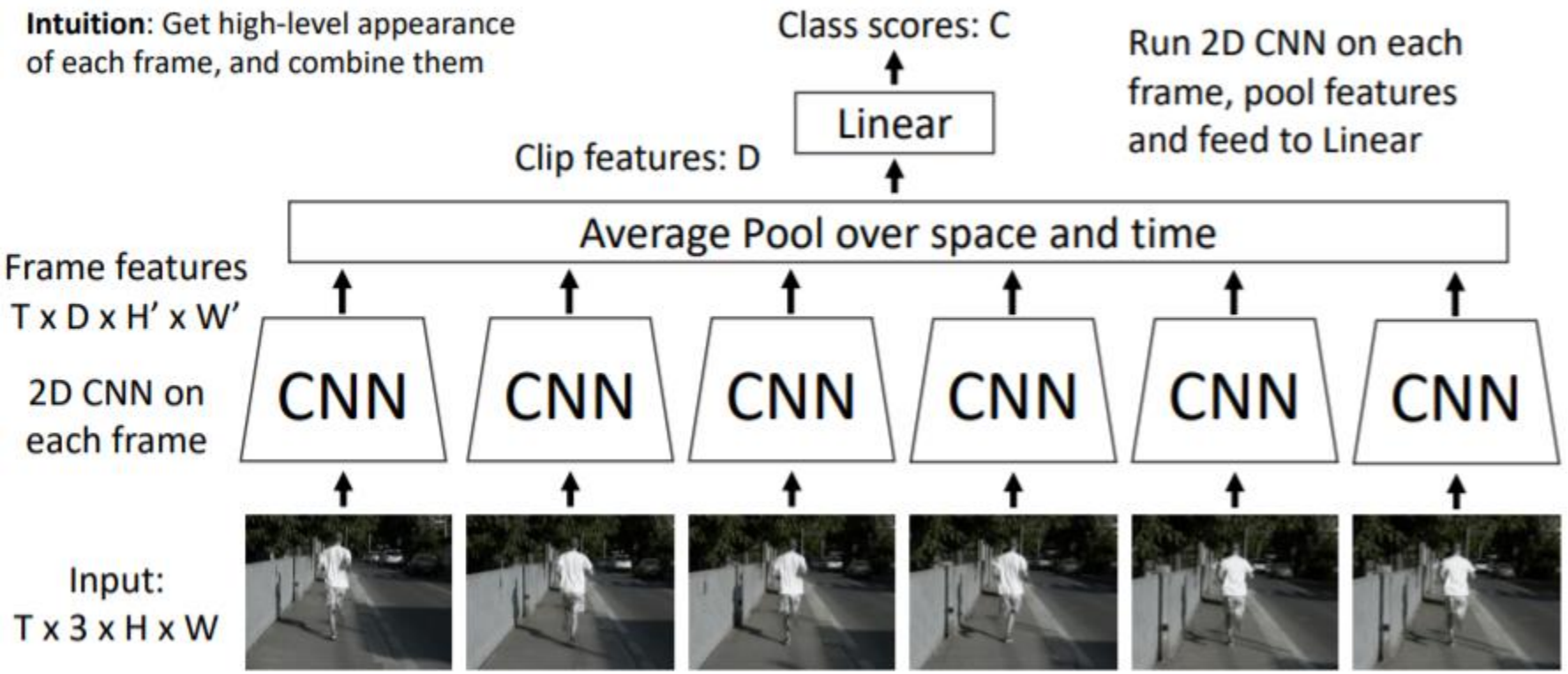
Video Classification(2): Late Fusion (with FC layers)



Input $T \times 3 \times H \times W \rightarrow 2D\ CNN \rightarrow T \times D \times H' \times W' \rightarrow Flatten \rightarrow MLP \rightarrow C$

- When received the initial input video, model didn't take advantage of the time
- Clip feature make some time features.

Video Classification(2): Late Fusion (with pooling)



Instead of **fc-layer** where many parameters and computations take place, use **average pool** to create D dimensional clip features to calculate class scores.

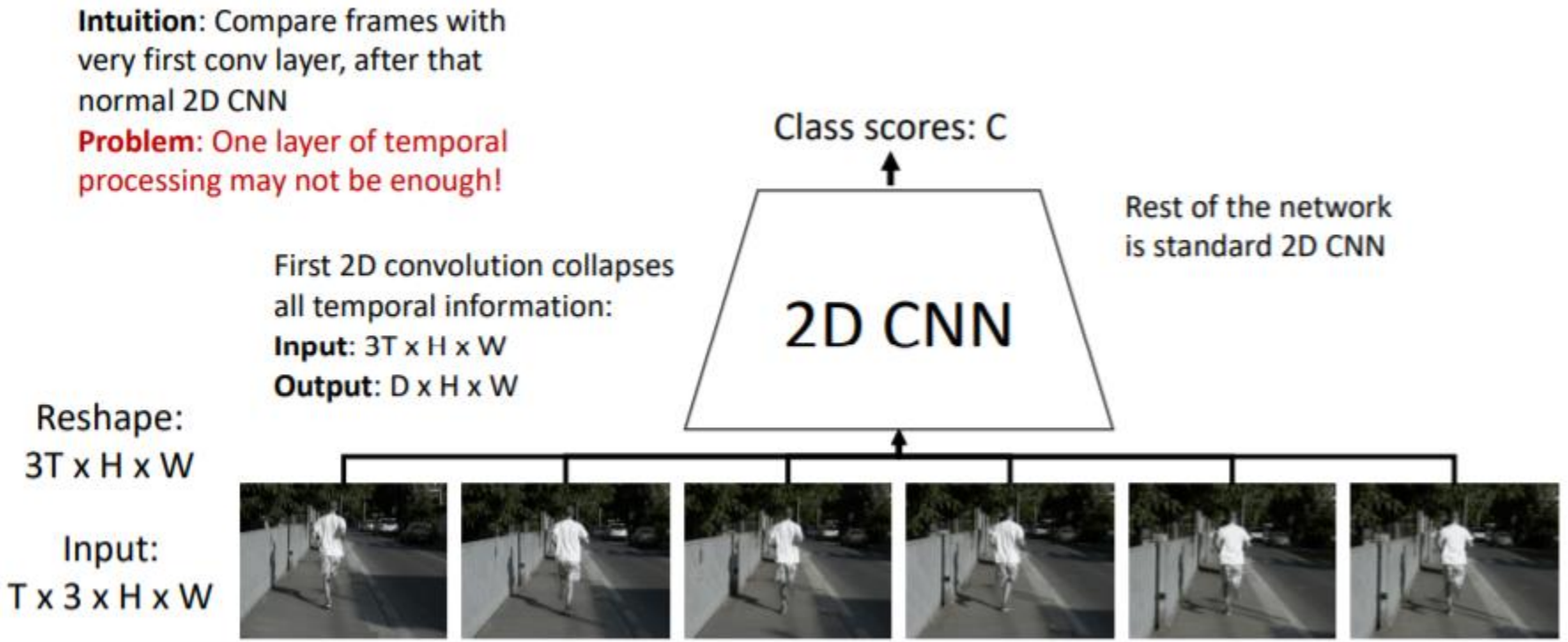
Video Classification(2): Late Fusion (with pooling)

Input:
T x 3 x H x W

Problem: Hard to compare low-level motion between frames

It is difficult to compare fine image differences (in the figure above, one leg goes up and down and is repeated because a person is running) by frame. Because there is only a few pixel differences.

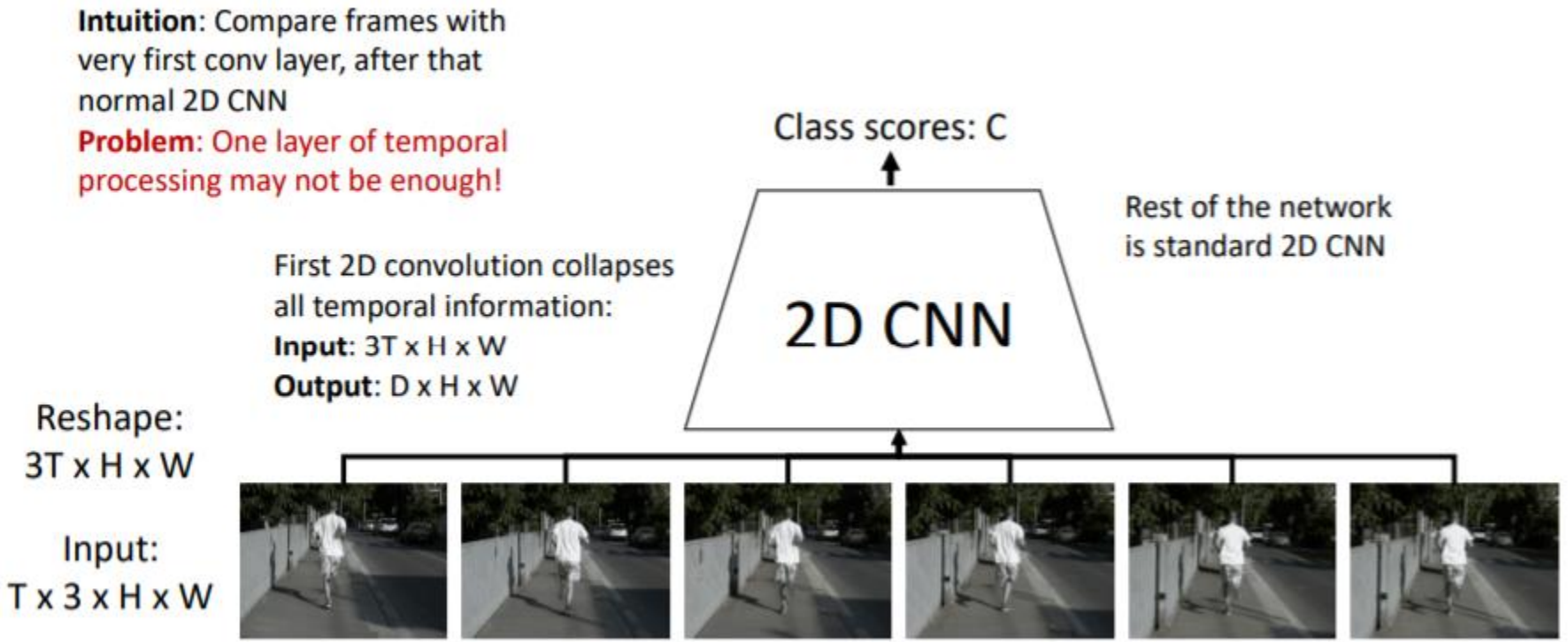
Video Classification: Early Fusion



$Late\ Fusion$ independently passed each video frame to CNN and later linked temporal features. $Early\ Fusion$ first links the temporal properties of the video frame by changing the order.

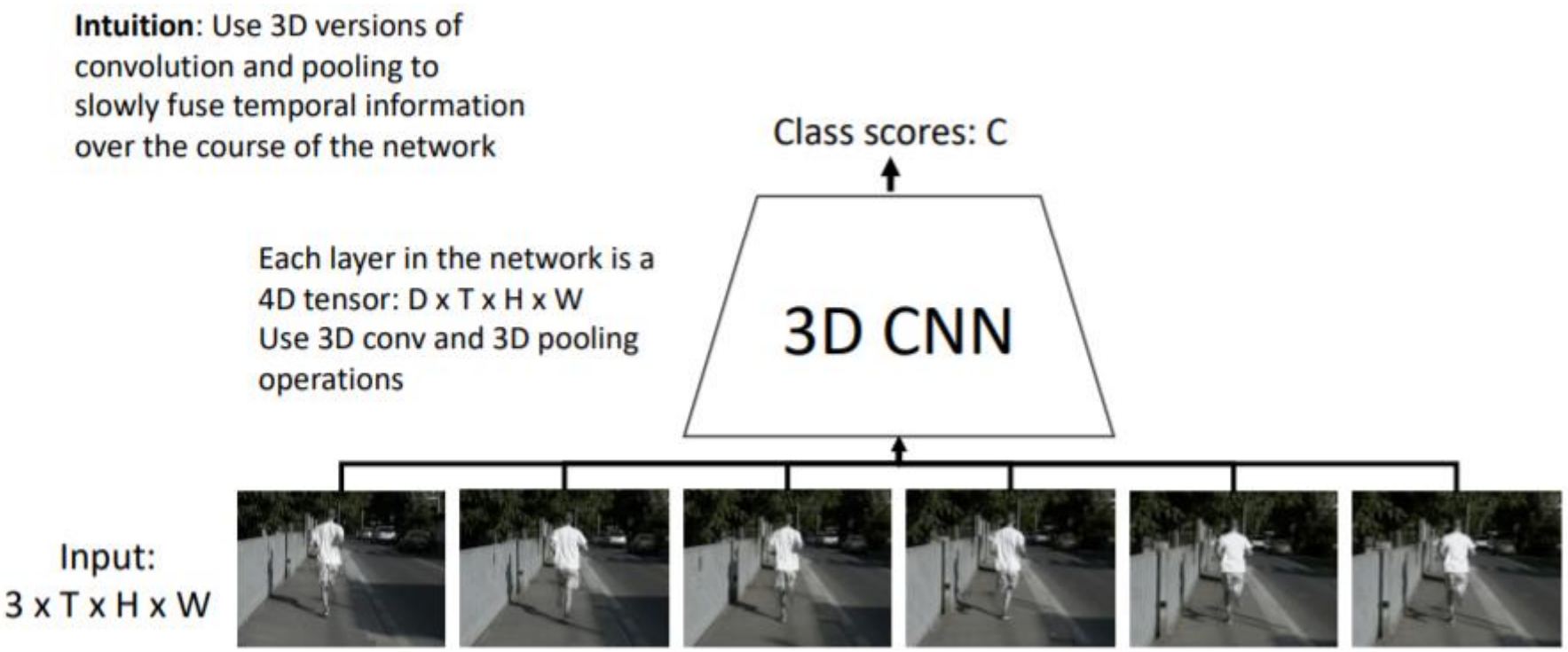
Input $T \times 3 \times H \times W \rightarrow reshape \rightarrow 3T \times H \times W \rightarrow 2D\ CNN$

Video Classification: Early Fusion



There is a problem that it is not enough to do the temporary processing once at first. This is because after going through 2D CNN, the first generated temperature characteristic collapses.

Video Classification: 3D CNN (Slow Fusion Network)



3D CNN maintains the existing 4D Tensor shape $D \times T \times H \times W$

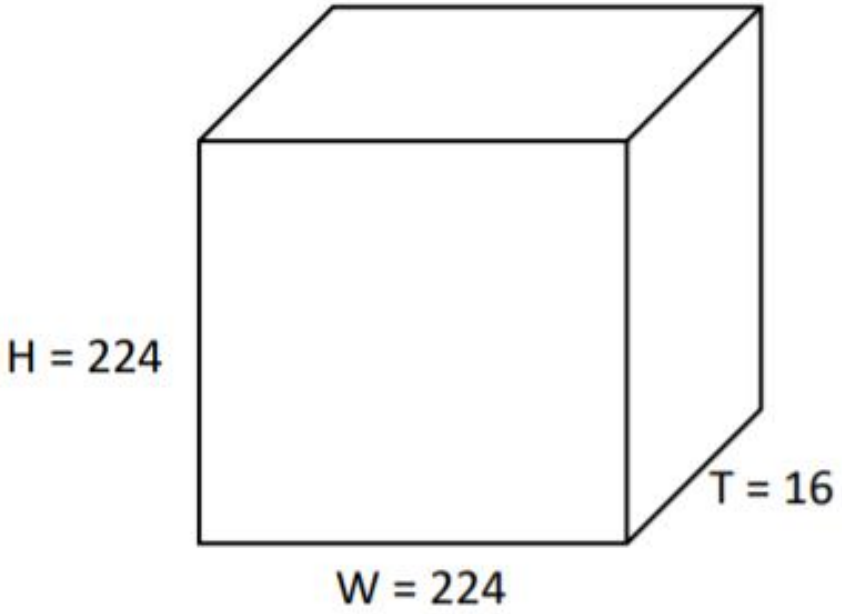
As the network progresses, it combines temporary information overall.

Early Fusion vs Late Fusion vs 3D CNN

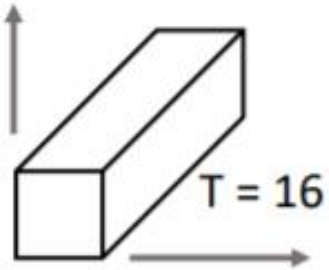
	Layer	Size (C x T x H x W)	Receptive Field (T x H x W)	
Late Fusion	Input	3 x 20 x 64 x 64		
	Conv2D(3x3, 3->12)	12 x 20 x 64 x 64	1 x 3 x 3	Build slowly in space, All-at-once in time at end
	Pool2D(4x4)	12 x 20 x 16 x 16	1 x 6 x 6	
	Conv2D(3x3, 12->24)	24 x 20 x 16 x 16	1 x 14 x 14	
	GlobalAvgPool	24 x 1 x 1 x 1	20 x 64 x 64	
Early Fusion	Input	3 x 20 x 64 x 64		
	Conv2D(3x3, 3*10->12)	12 x 64 x 64	20 x 3 x 3	Build slowly in space, All-at-once in time at start
	Pool2D(4x4)	12 x 16 x 16	20 x 6 x 6	
	Conv2D(3x3, 12->24)	24 x 16 x 16	20 x 14 x 14	
	GlobalAvgPool	24 x 1 x 1	20 x 64 x 64	
3D CNN	Input	3 x 20 x 64 x 64		
	Conv3D(3x3x3, 3->12)	12 x 20 x 64 x 64	3 x 3 x 3	Build slowly in space, Build slowly in time "Slow Fusion"
	Pool3D(4x4x4)	12 x 5 x 16 x 16	6 x 6 x 6	
	Conv3D(3x3x3, 12->24)	24 x 5 x 16 x 16	14 x 14 x 14	
	GlobalAvgPool	24 x 1 x 1	20 x 64 x 64	

2D Conv (Early Fusion)

Input: C_{in} x T x H x W
(3D grid with C_{in}-dim feat at each point)

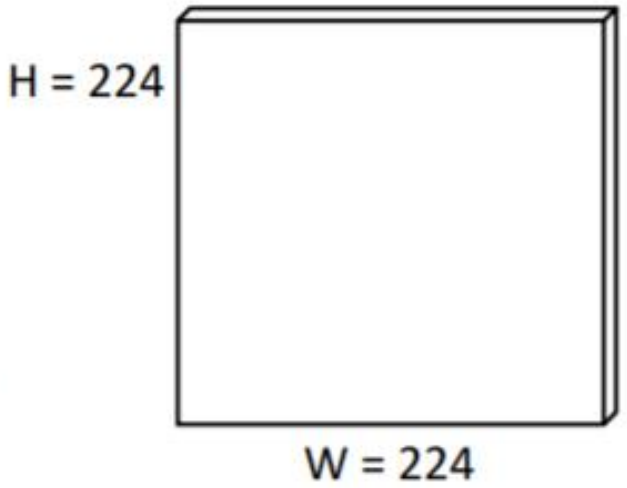


Weight:
C_{out} x C_{in} x T x 3 x 3
Slide over x and y



C_{out} different filters

Output:
C_{out} x H x W
2D grid with C_{out} –dim feat at each point



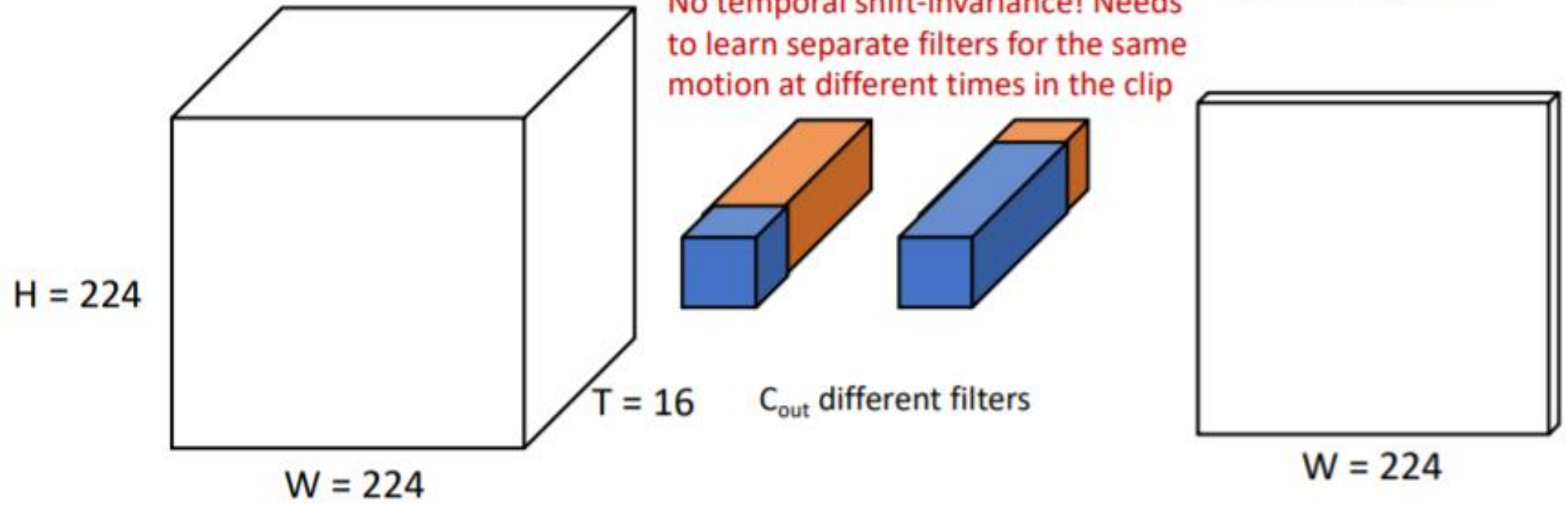
2D Conv (Early Fusion)

Input: C_{in} x T x H x W
(3D grid with C_{in}-dim feat at each point)

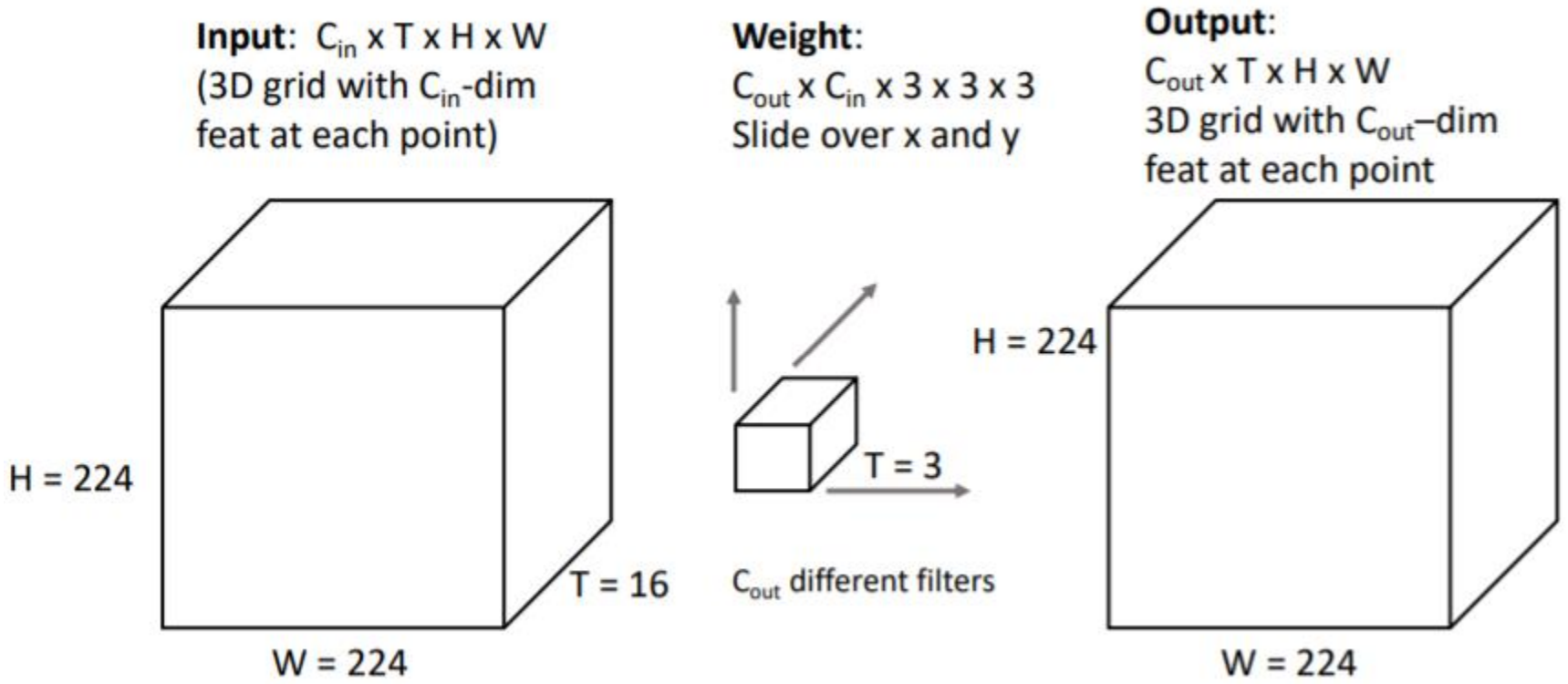
Weight:
C_{out} x C_{in} x T x 3 x 3
Slide over x and y

Output:
C_{out} x H x W
2D grid with C_{out} –dim feat at each point

No temporal shift-invariance! Needs to learn separate filters for the same motion at different times in the clip

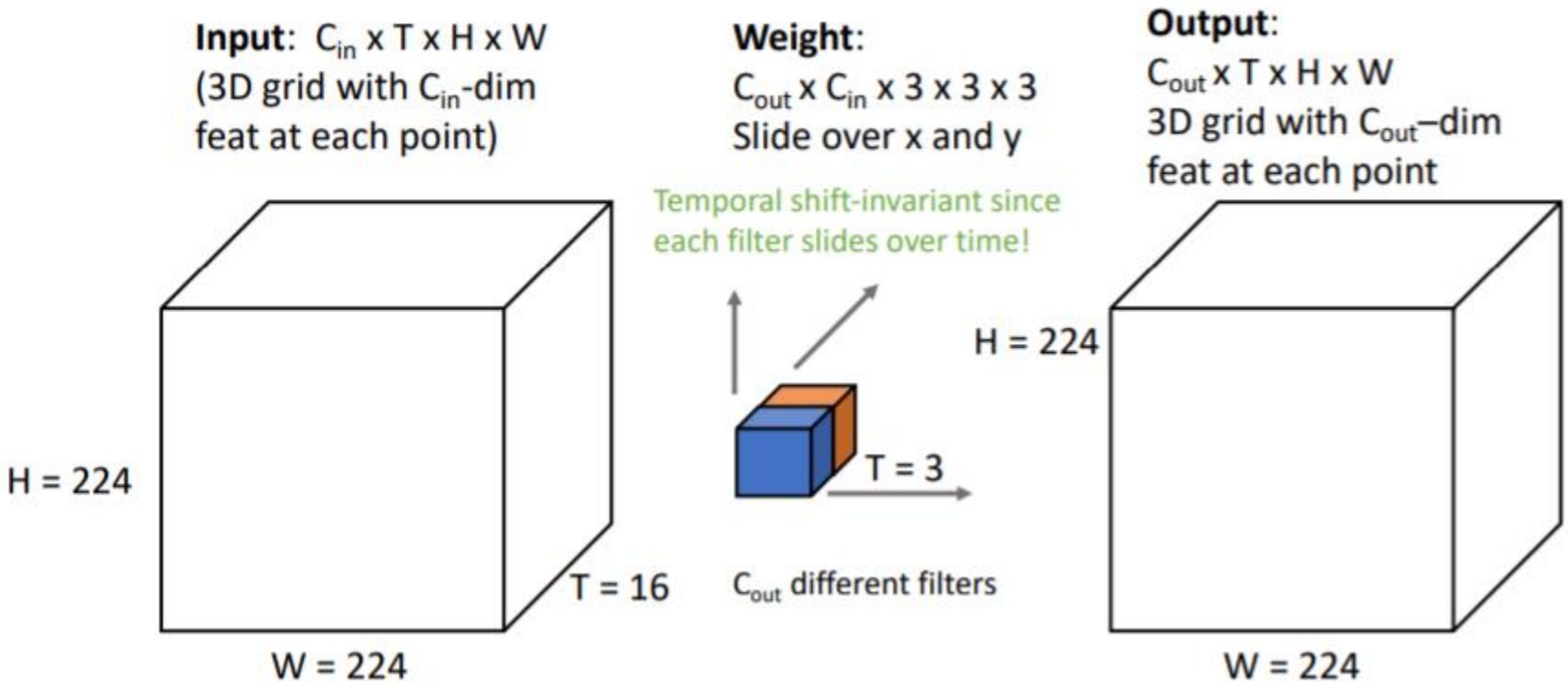


3D Conv (3D CNN)



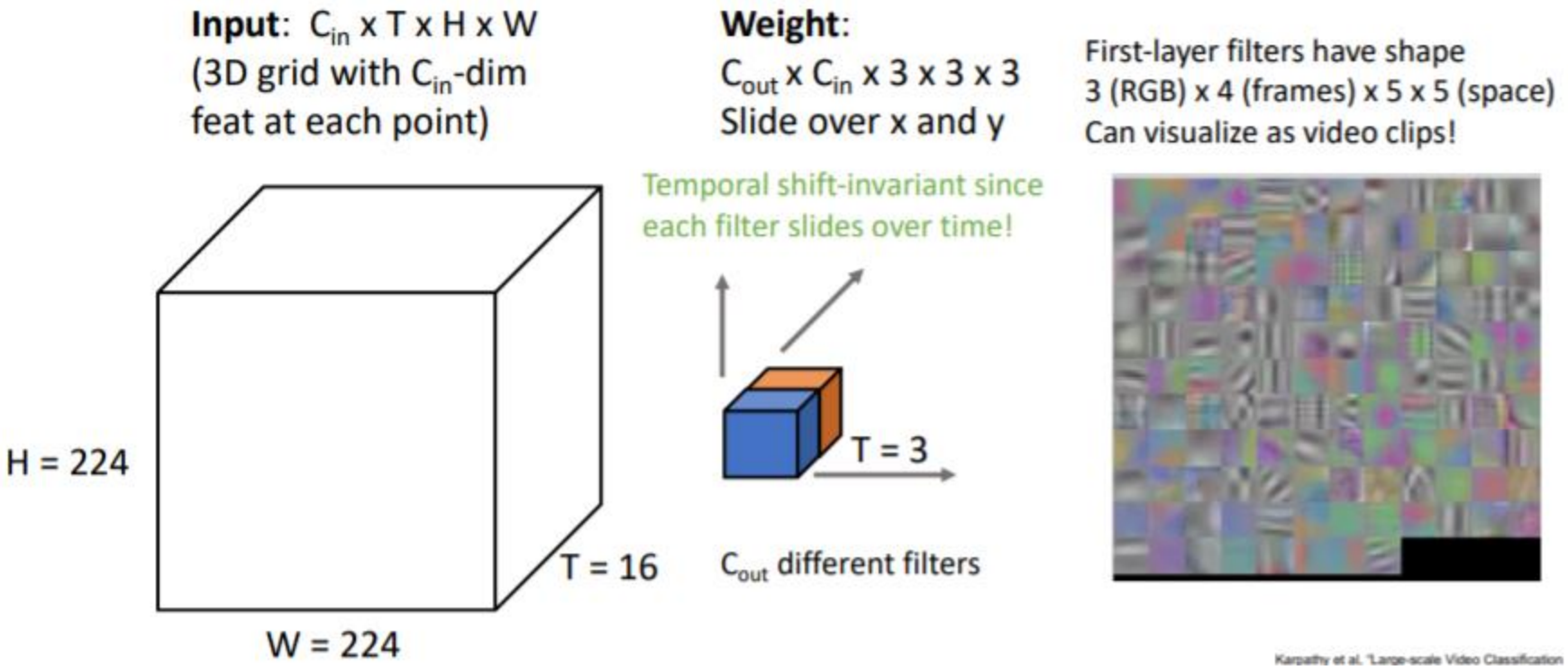
- $3 \times 3 \times 3$ $filter$
- Receptive field is increasing not only $H \times W$ but also T

3D Conv (3D CNN)



Since the filter slides for time, it can be learned with one filter for the same motion.

3D Conv (3D CNN)



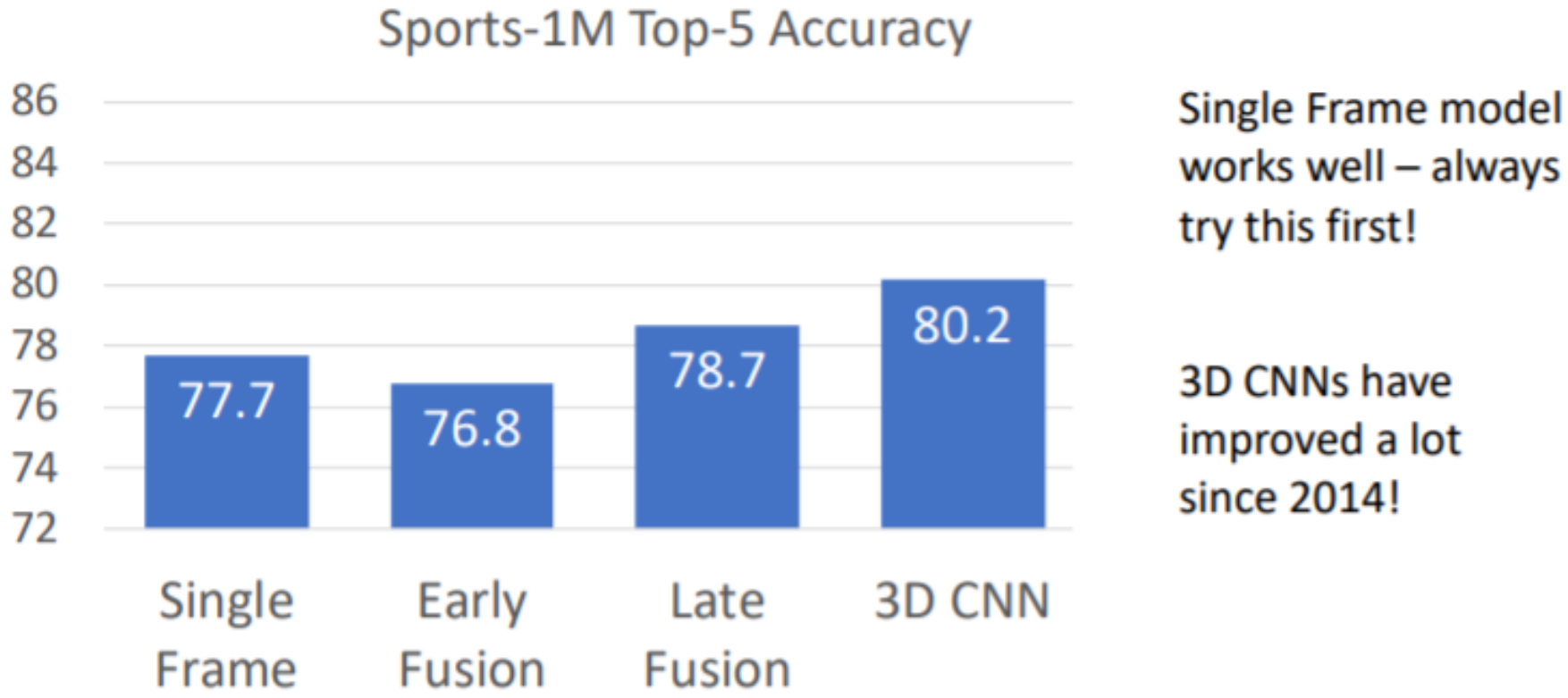
If you look at First-layer's filters, you can see that they are not only looking at line, blobs, etc., but **also learning about motion**.

Example Video Dataset: Sports-1M

1 million YouTube videos
annotated with labels for
487 different types of sports

Ground Truth
Correct prediction
Incorrect prediction

Performance : Early Fusion vs Late Fusion vs 3D CNN



- Single-Frame approach is more powerful than you think
- Late Fusion and 3D CNN aren't performing so well by a large margin.

C3D: The VGG of 3D CNNs

C3D: The VGG of 3D CNNs

3D CNN that uses all 3x3x3 conv and 2x2x2 pooling
(except Pool1 which is 1x2x2)

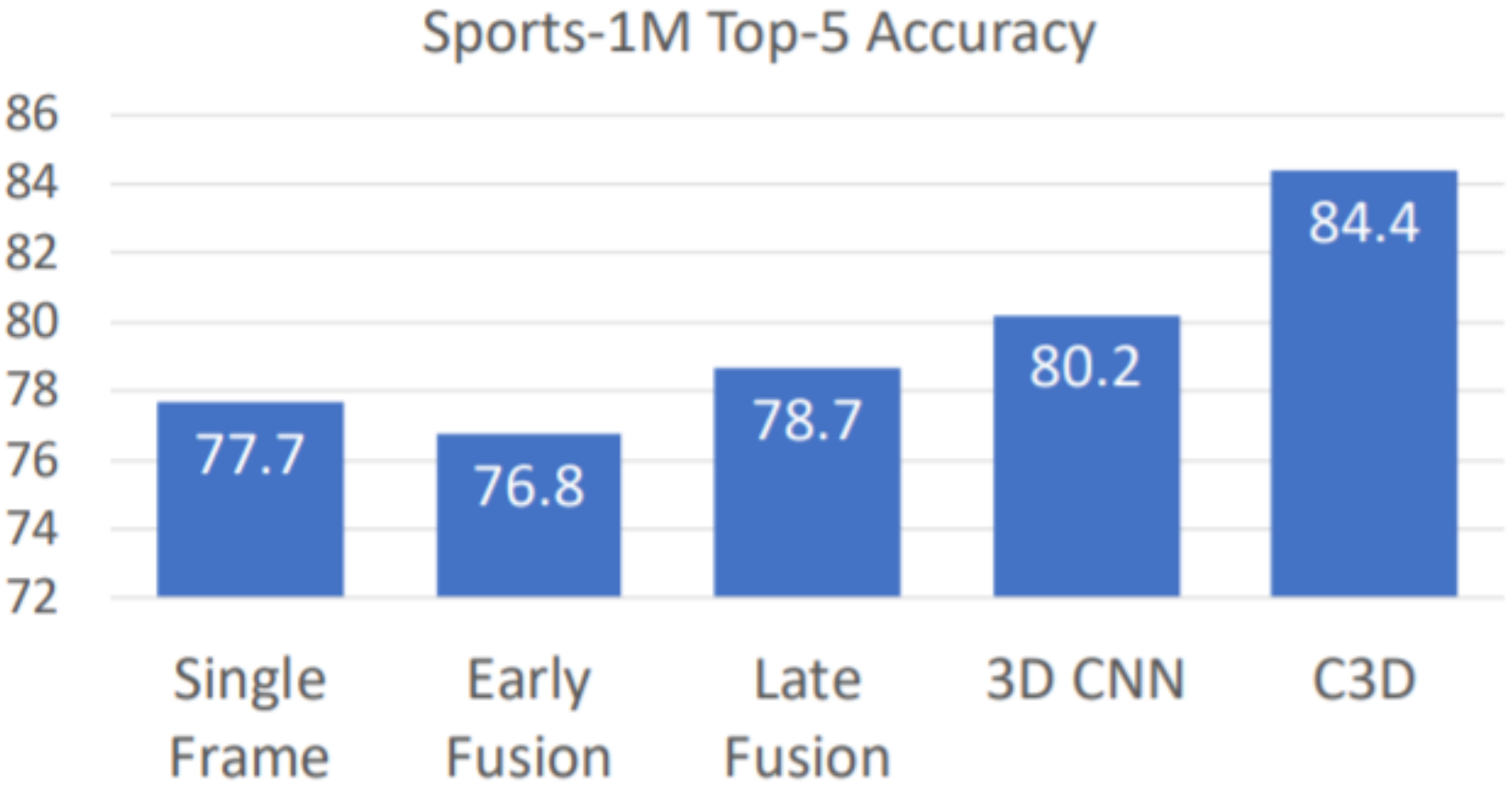
Released model pretrained on Sports-1M: Many people used this as a video feature extractor

Problem: 3x3x3 conv is very expensive!
AlexNet: 0.7 GFLOP
VGG-16: 13.6 GFLOP
C3D: **39.5 GFLOP (2.9x VGG!)**

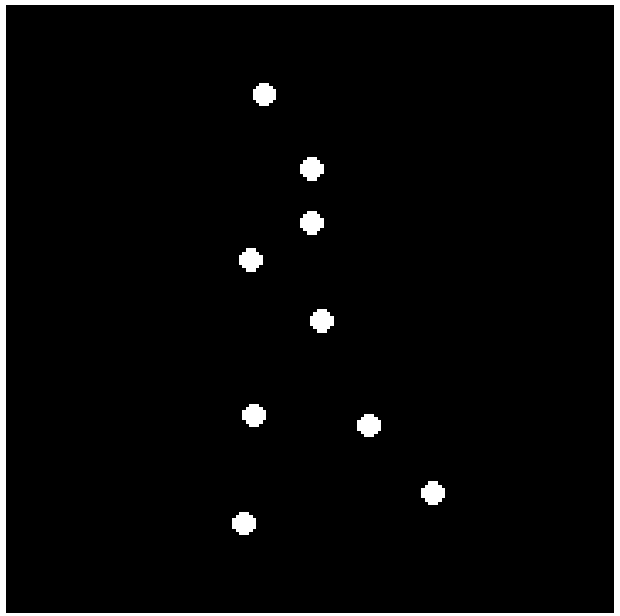
Layer	Size	MFLOPs
Input	3 x 16 x 112 x 112	
Conv1 (3x3x3)	64 x 16 x 112 x 112	1.04
Pool1 (1x2x2)	64 x 16 x 56 x 56	
Conv2 (3x3x3)	128 x 16 x 56 x 56	11.10
Pool2 (2x2x2)	128 x 8 x 28 x 28	
Conv3a (3x3x3)	256 x 8 x 28 x 28	5.55
Conv3b (3x3x3)	256 x 8 x 28 x 28	11.10
Pool3 (2x2x2)	256 x 4 x 14 x 14	
Conv4a (3x3x3)	512 x 4 x 14 x 14	2.77
Conv4b (3x3x3)	512 x 4 x 14 x 14	5.55
Pool4 (2x2x2)	512 x 2 x 7 x 7	
Conv5a (3x3x3)	512 x 2 x 7 x 7	0.69
Conv5b (3x3x3)	512 x 2 x 7 x 7	0.69
Pool5	512 x 1 x 3 x 3	
FC6	4096	0.51
FC7	4096	0.45
FC8	C	0.05

- $3 \times 3 \times 3\ Conv, \quad 2 \times 2 \times 2\ Pooling$
- 3 times more computations than VGG

C3D: The VGG of 3D CNNs

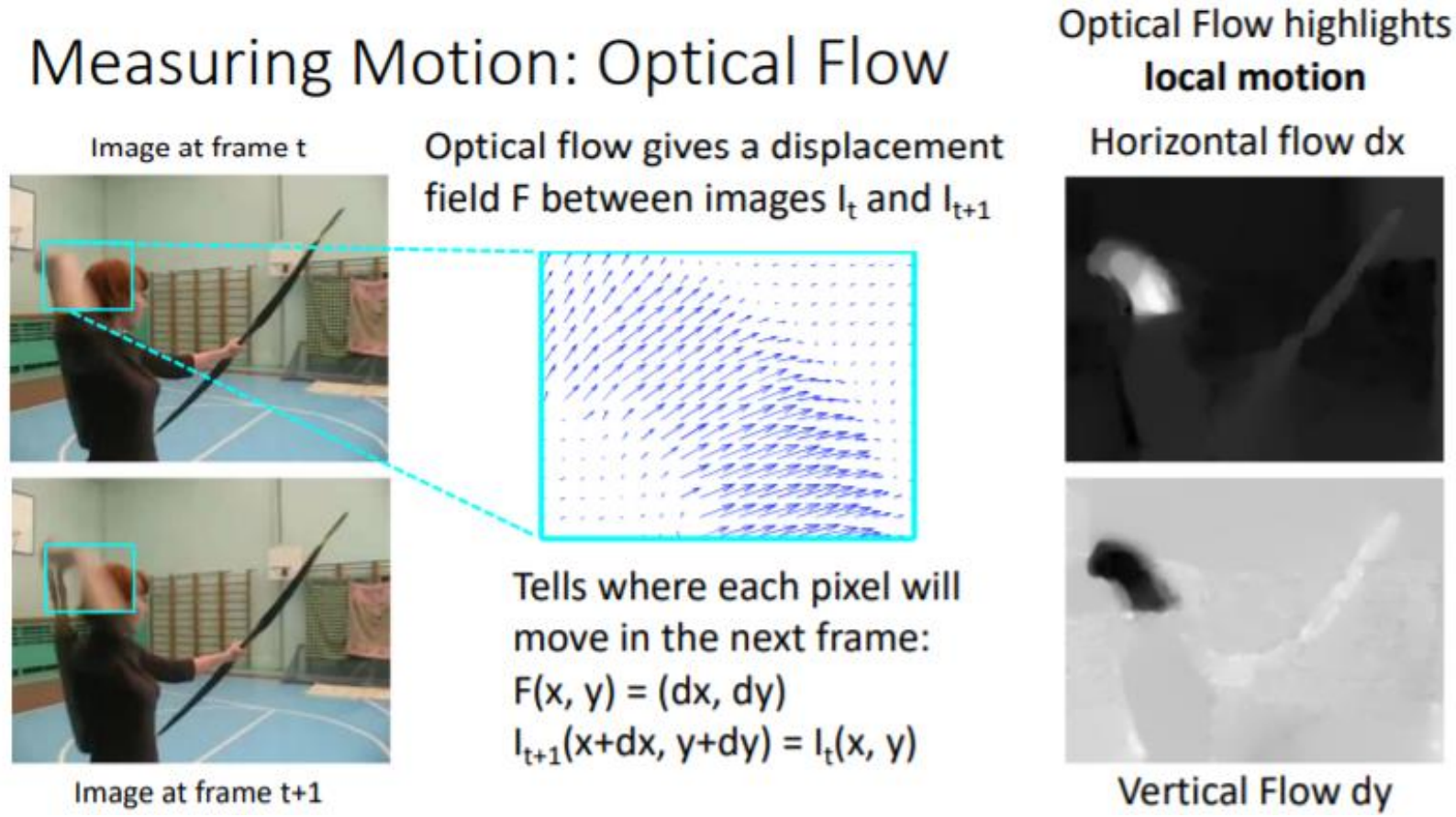


Recognizing Actions from Motion



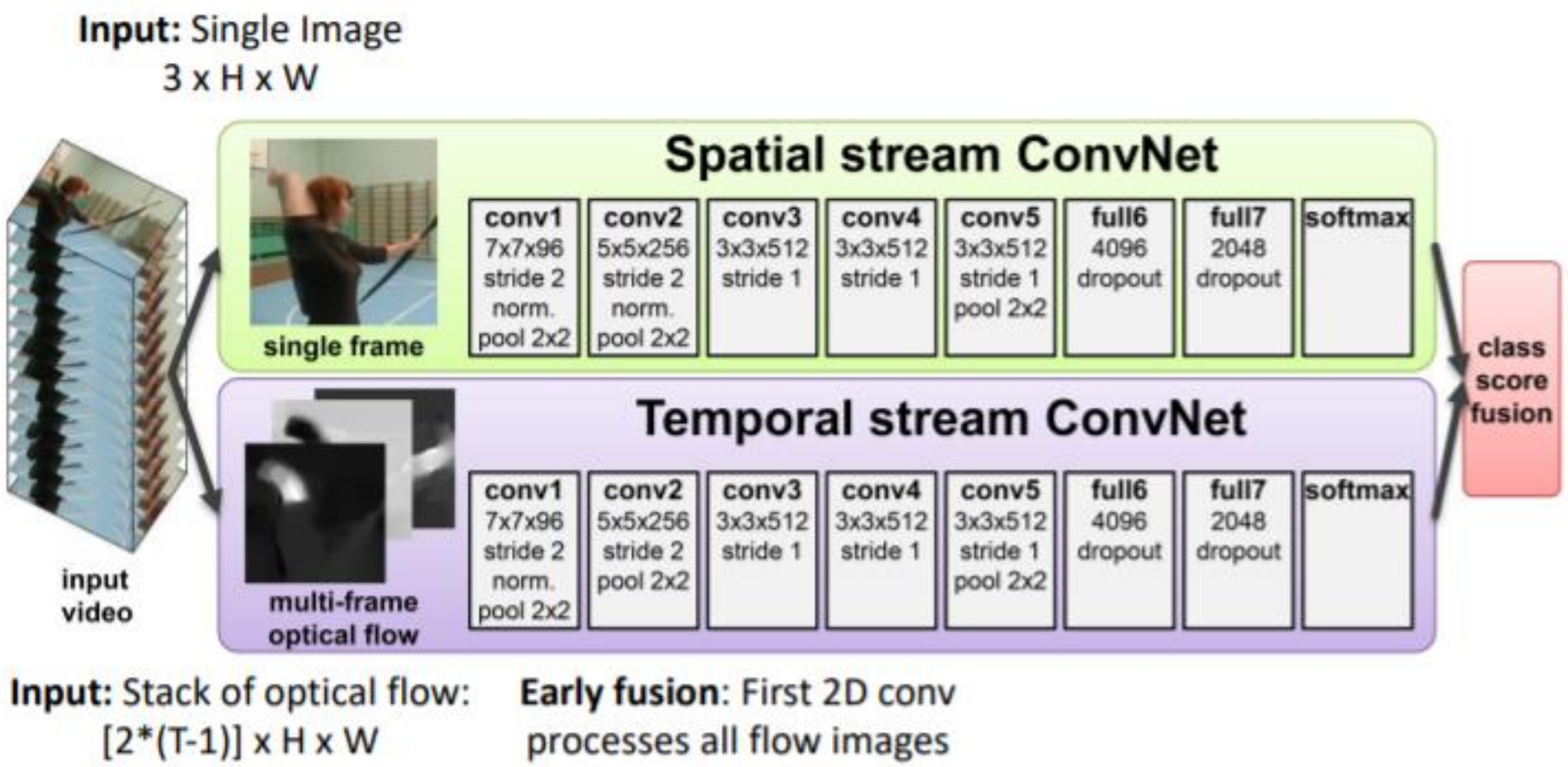
- We can easily recognize actions using only motion information
- There is a method of expressing such motion information explicitly.

Measuring Motion: Optical Flow

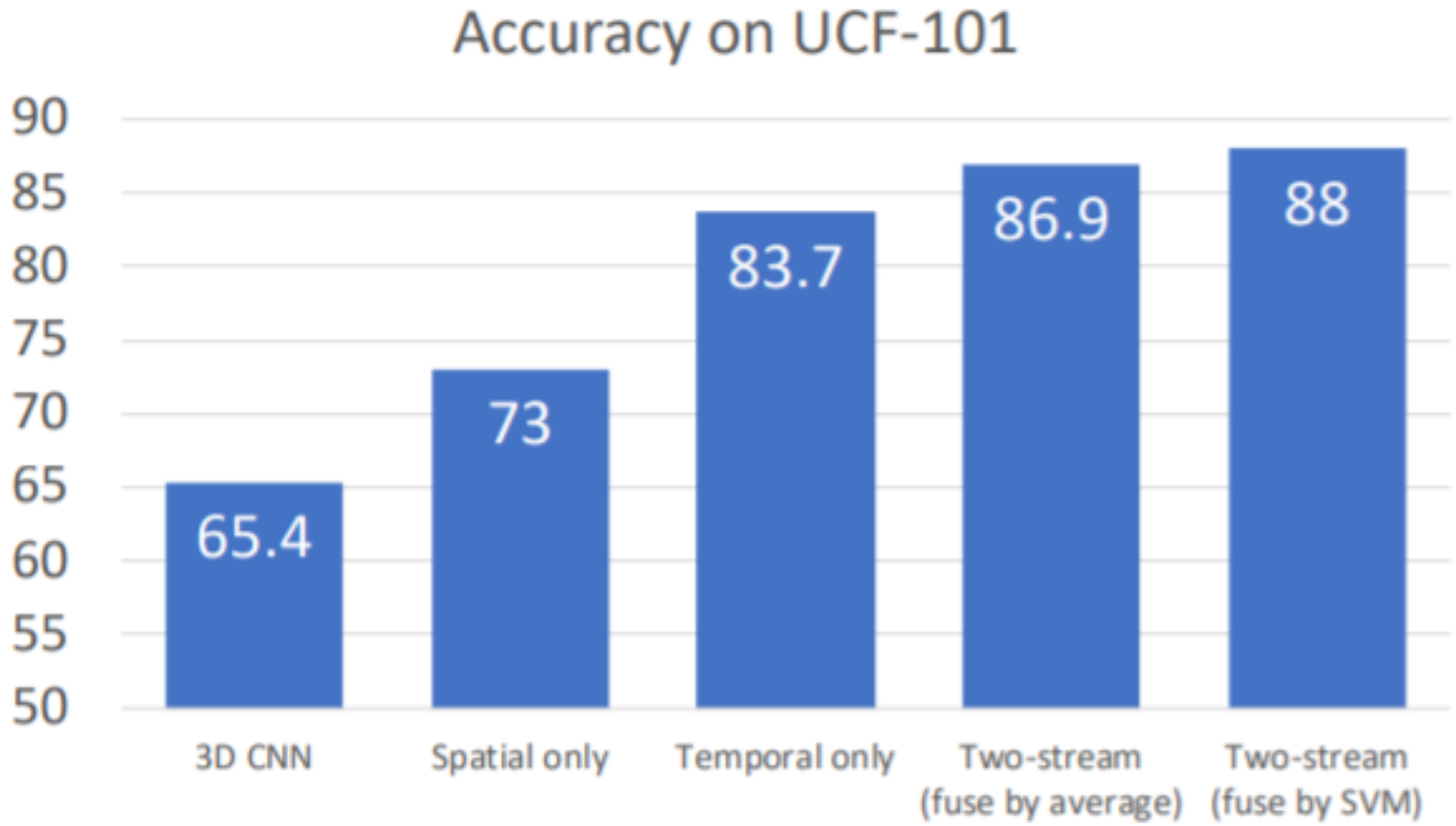


Take a pair of adjunct video frames as input and calculate an optical flow for each pixel that describes **where it will be located in the next frame**.

Separating Motion and Appearance: Two-Stream Networks



Separating Motion and Appearance: Two-Stream Networks



It can be seen that even though only Temporal information was used, it produces tremendous performance. It can be seen that not only humans can easily recognize the action with motion, but also computers can do it.

Thank you!
Q & A