Visual AI

CPSC 532R/533R – 2019/2020 Term 2

Lecture 5. Modelling 3D skeletons and point clouds

Helge Rhodin

Overview

- 9 Lectures (~ once a week)
  - Introduction
  - Deep learning basics and best practices
  - Network architectures for image processing
  - Representing images and sparse 2D keypoints
  - Representing dense and 3D keypoints
  - GANs and unpaired image translation (moved)
  - Representing geometry and shape
  - Representation learning
  - Attention models

- 3x Assignments
  - Playing with pytorch (5% of points)
  - Pose estimation (10% of points)
  - Shape generation (10% of points)

- 1x Paper presentation (Weeks 3 – 12)
  - Presentation, once per student (25% of points) (15 min + 15 min discussion)
  - Read and review one out of the two papers presented per session (10% of points)

- 1x Project (**40 % of points**)
  - Project pitch (3 min, week 6&7)
  - Project presentation (10 min, week 13&14)
  - Project report (6 pages, Dec 14)

Course projects

Conditions

- groups of 2-3 students
- a CV or CG topic of your choice

Project proposal

- 3-minute pitch

Project scope

- Motivation (intro & abstract)
- Literature review
- Method development and coding
- Evaluation

Project report

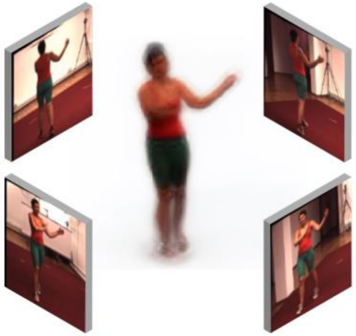
- 6 pages in CVPR double column format
- Sections: introduction/motivation, related work, method description, and evaluation

Project presentation

- 10 min talk per group

Possible project directions I

Improve visual quality



New network architectures + X?

Character animation

handle mesh and skeleton sequences

Movie editing

“movie reshaping”

Possible project directions II

Killer whale identification

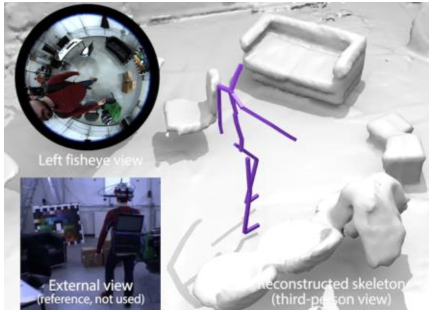
Andrew W Trites

Professor and Director

Institute for the Oceans and Fisheries UBC

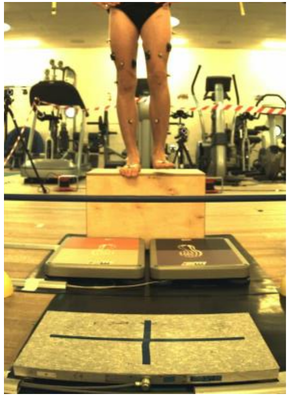
See www.facebook.com/marinemammal

Prevent foot sliding



IMU-based?

force & 3D pose estimation



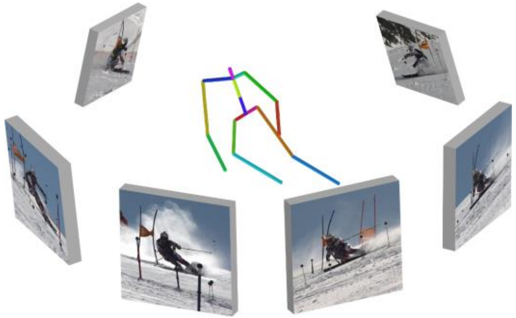
Dr. Jörg Spörri

Sport medicine head

University Hospital Balgrist

Possible project directions III

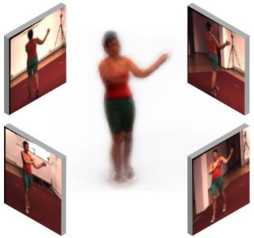
Fast motion capture



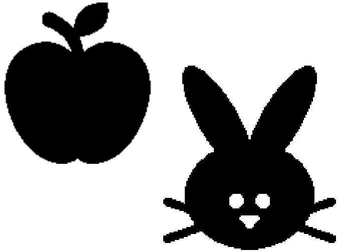
Exploit fast-moving background

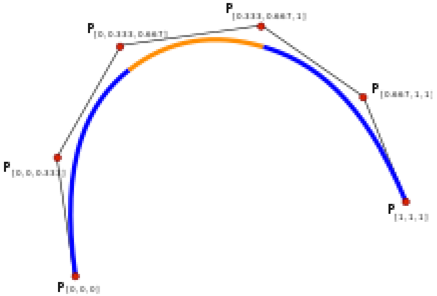
Computer graphics (simulation)

+ Computer vision (real world)



Your own idea!





Last year’s project examples

Reinforcement learning from visual feedback (egocentric)

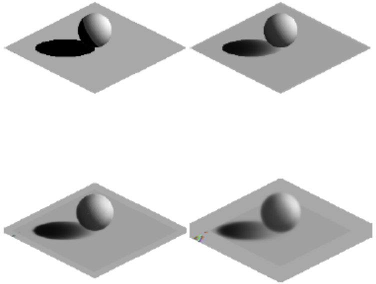
by Daniele Reda and Tianxin Tao

Virtual keyboard

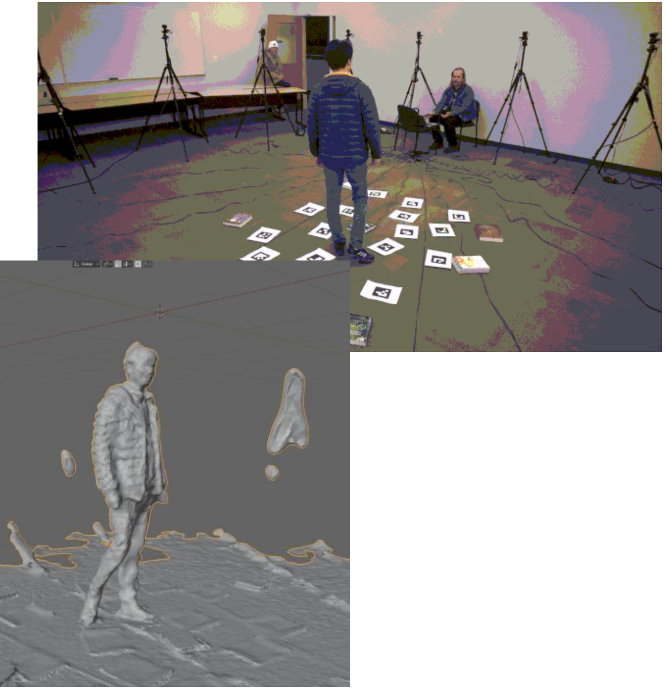
by Willis Peng

Differentiable shadow rendering

Jerry Yin and Dave Pagurek van Mossel



New playgrounds (CS internal)



Multi-cam setups

Accelerometer sensors

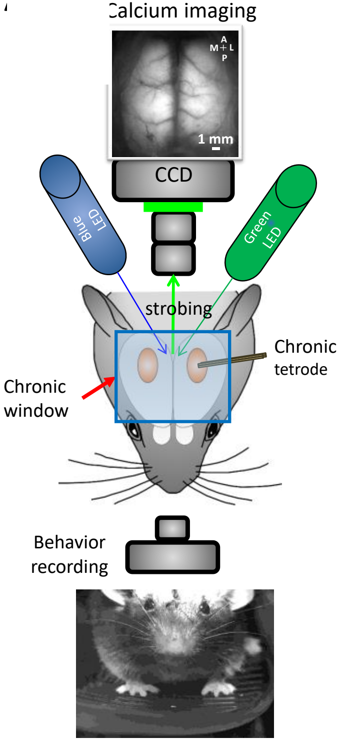
360 degree camera

New playgrounds (within UBC, outside CS)

Psychology and VR

People think and behave differently in VR

Alan Kingstone (Psychology)



Neuroscience

Link between neural firing and motion?

Centre for Brain Health

2D pose estimation cont.

Recap Integral Regression-based 2D pose estimation

A combination of classification and regression

1. Detection network to produce heatmaps
   - same CNN as for heatmap prediction
2. Soft-max layer to turn heatmap H into probability map P
   - normalizing all pixels in each heatmap H

$$P[u,v] = \text{soft-max}(H,(u,v)) = \frac{e^{H[u,v]}}{\sum_{x=1}^{\text{width}} \sum_{y=1}^{\text{height}} e^{H[x,y]}}$$

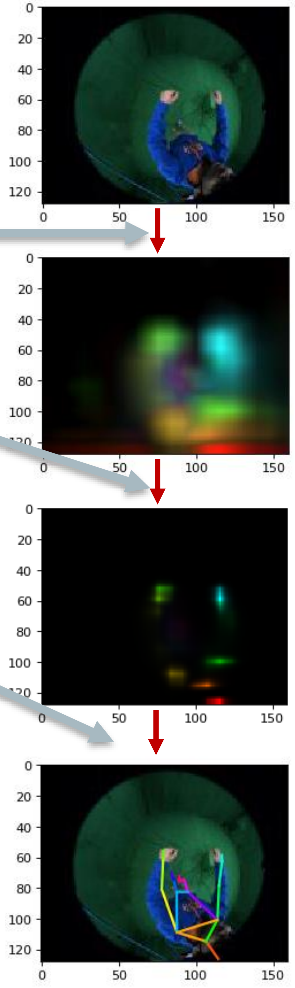
3. Integration layer to regress joint position (expected position)
   - can be interpreted as voting/weighted average

     each pixel votes for its own position, weighted by its probability

$$\text{pose}_x = \sum_{x=1}^{\text{width}} \sum_{y=1}^{\text{height}} x P[x,y]$$

$$\text{pose}_y = \sum_{x=1}^{\text{width}} \sum_{y=1}^{\text{height}} y P[x,y]$$

[Sun et al., Integral Human Pose Regression.]



Part affinity fields for associating joints of multiple persons

An extension of heatmaps (positions) to vectors (directions)

- Ground truth affinity field L* between joints *c,k*

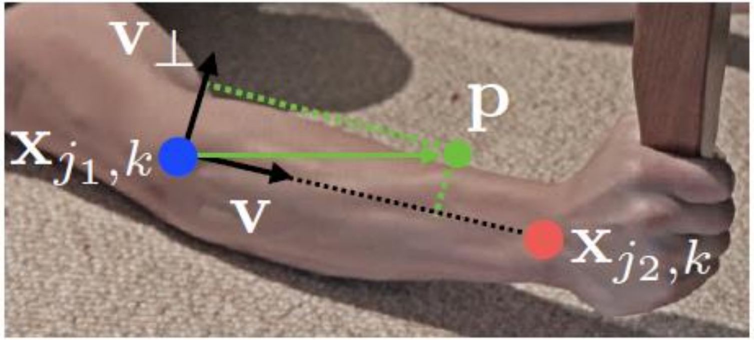
$$\mathbf{L}^*_{c,k}(\mathbf{p}) = \begin{cases} \mathbf{v} & \text{if } \mathbf{p} \text{ on limb } c, k \\ \mathbf{0} & \text{otherwise.} \end{cases}$$

Determine presence by

$$0 \le \mathbf{v} \cdot (\mathbf{p} - \mathbf{x}_{j_1,k}) \le l_{c,k} \text{ and } |\mathbf{v}_\perp \cdot (\mathbf{p} - \mathbf{x}_{j_1,k})| \le \sigma_l,$$

with v defined as

$$\mathbf{v} = (\mathbf{x}_{j_2,k} - \mathbf{x}_{j_1,k}) / ||\mathbf{x}_{j_2,k} - \mathbf{x}_{j_1,k}||_2$$



[Cao et al., Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields, CVPR 2017]

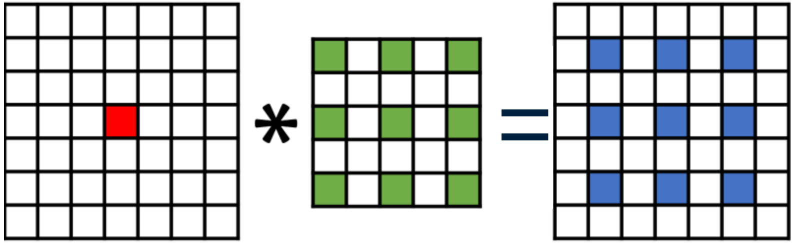
Dilated/Atrous Convolution and ESP Net

Idea: increase the receptive field

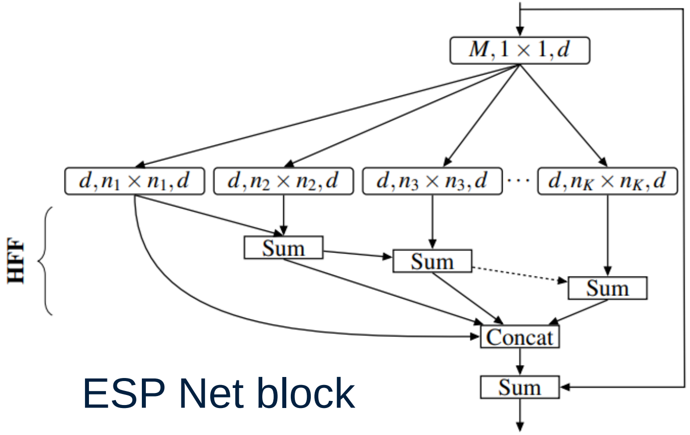
- inserting zeros in the convolutional kernel
  - the effective size of n × n dilated convolutional kernel with dilation rate r, is (n−1)r +1 x (n−1)r +1
  - no increase in parameters
- use a set of dilated filters for multi-scale information

Problem: checkerboard patterns

- Fix: Hierarchical feature fusion (HFF)
  - add output from different dilations before concat



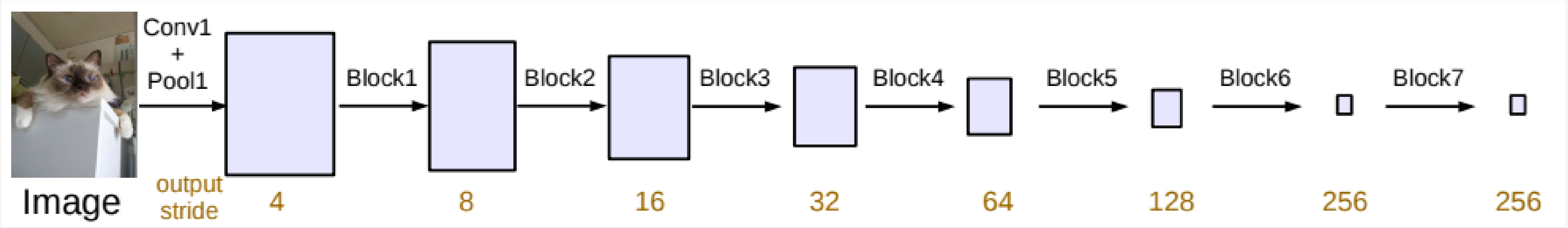
[Mehta et al. ESPNet: Efficient Spatial Pyramid of Dilated Convolutions for Semantic Segmentation]



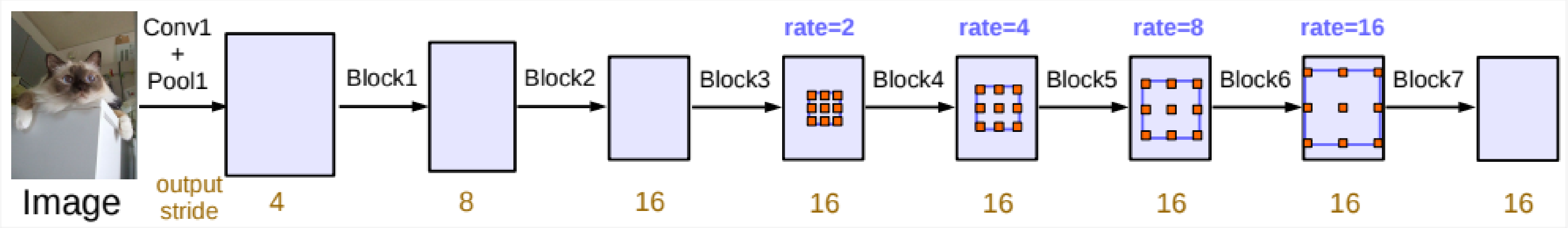
ESP Net block

Sequential application of dilated convolution

- maintains high resolution
- increases receptive field of subsequent layers



(a) Going deeper without atrous convolution.



[Chen et al., Rethinking Atrous Convolution for Semantic Image Segmentation]

Objective functions

Recap: MSE, MAE, Cross Entropy, and log-likelihoods

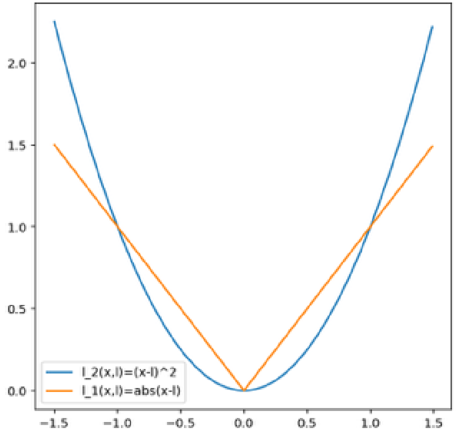
So far:

- simple losses operating element-wise
  - the l_2 loss / MSE
  - the l_1 loss / MAE
- connecting all elements, but treating them equally
  - soft-max + log-likelihood
  - cross entropy
  - Gaussian log-likelihood, (Mixture) Density networks

$$l_{\text{log-likelihood}}(x, y) = -\log(\text{soft-max}(f(x), y))$$

$$l_{\text{cross entropy}}(x, y) = -\sum_{j=1}^{K} y_{[j]} \log(\text{soft-max}(f_{[j]}(x)))$$

$$l_{\text{density network}} = \log\left(\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(y-\mu)^2}{2\sigma^2}}\right)$$



Quadratic loss

$$l_2(y, l) = (y - l)^2$$

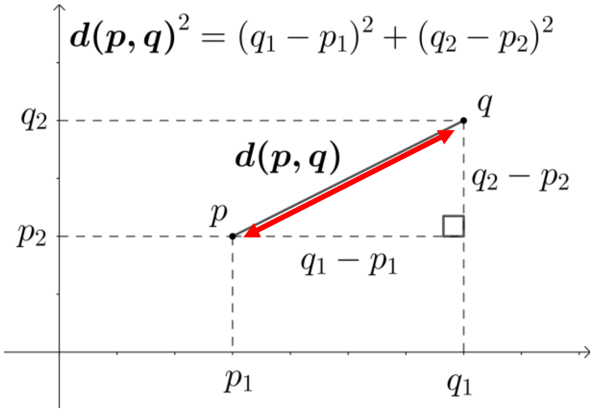
Absolute loss

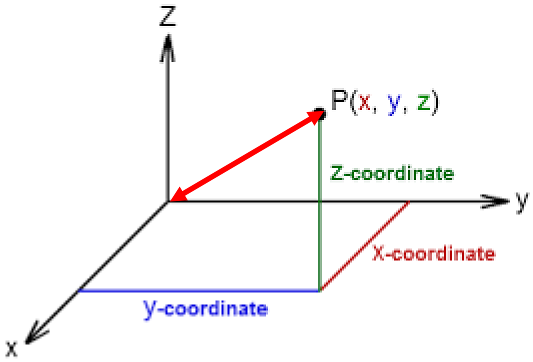
$$l_1(y, l) = |y - l|$$

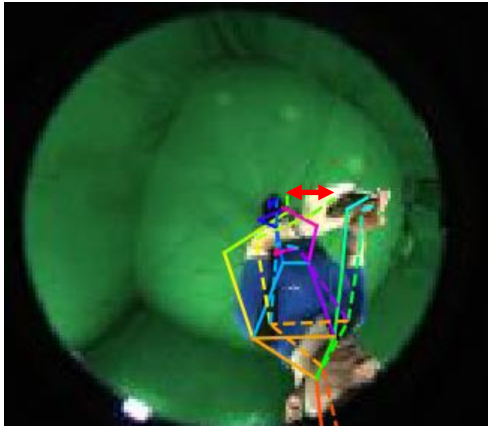
Mean Per-Joint Position Error (MPJPE)

Euclidean distance d(p,q)

- the square root of the sum of squared coordinate offsets







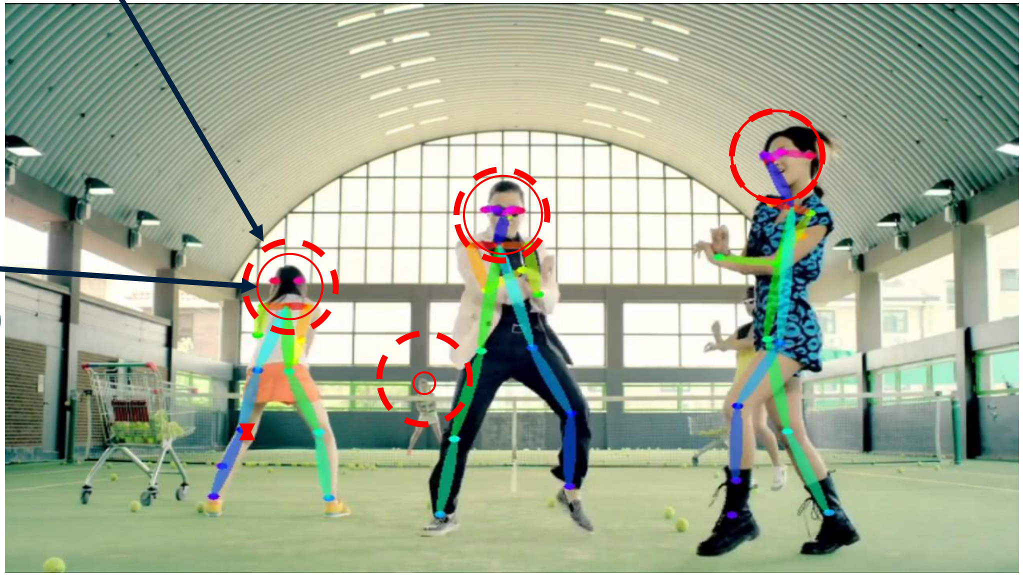
Distance of prediction (solid) to ground truth (dashed)

- averaged over all points
  - groups elements
    - 2D: group of 2 elements, e.g., tensor of N x 18 x 2 for a skeleton with 18 joints
    - 3D: group of 3 elements

Percentage of Correct Keypoints (PCK)

- The number of keypoints below a threshold
  - usually using Euclidean distance
  - less sensitive to outliers
  - scale sensitive

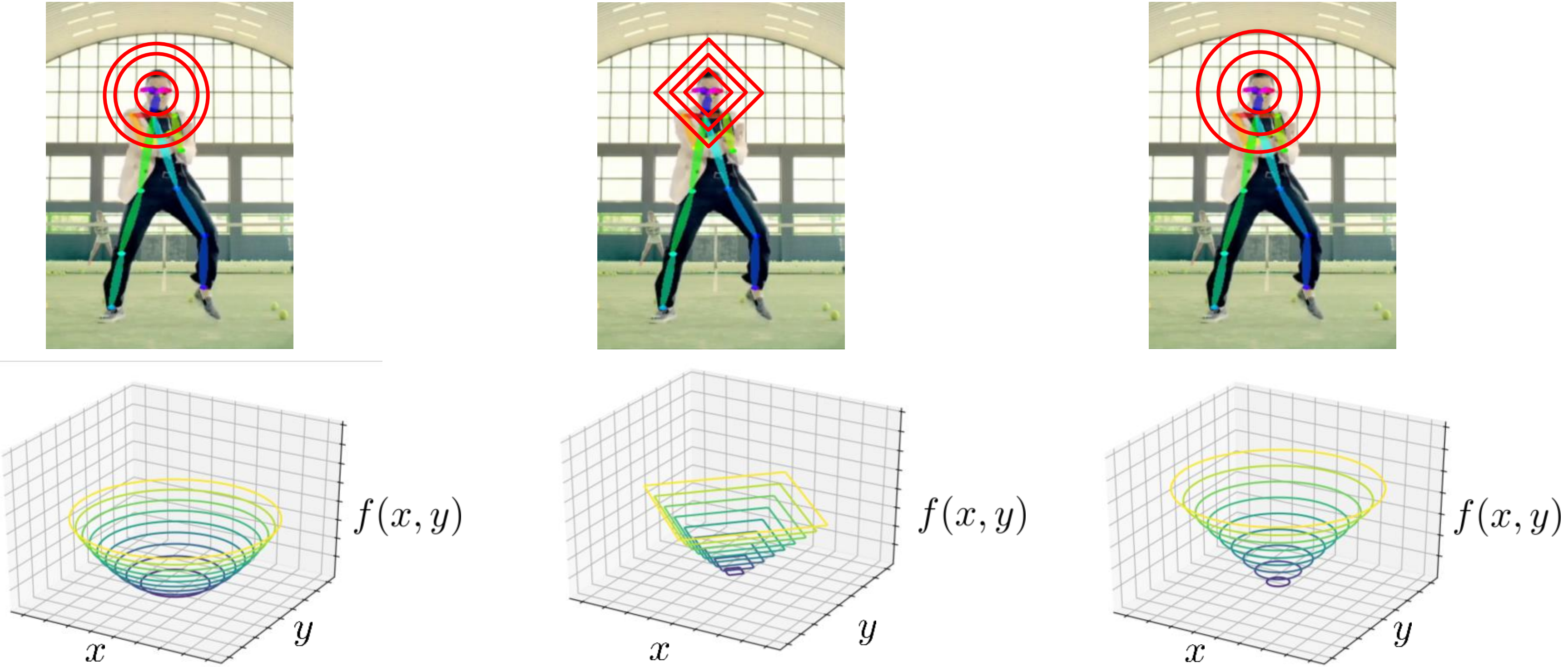
- Scale invariant version: PCKh
  - relative to the scale of the GT annotatio
    - e.g. halt the head-neck distance is common for 2D human pose



Loss comparison

Contour lines

3D slope



Squared error
MSE

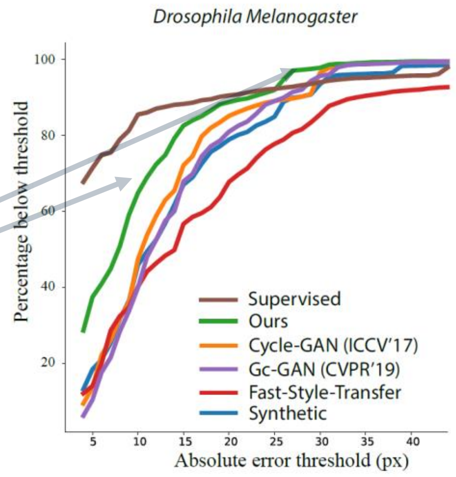
Absolute error
MAE

Euclidean distance
MPJPE

ROC and AUC

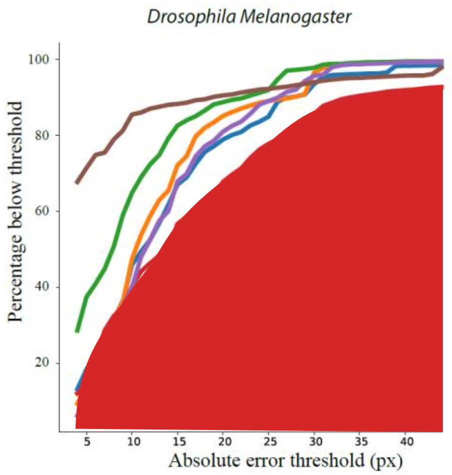
Receiver operating characteristic (ROC)

- true positive rate (TPR) against the false positive rate (FPR)
- defined for binary classification
- applicable for any binary metric (e.g., PCK)
- often reveals important details!



Area Under Curve (AUC)

- a score for consistency
- the integral (sum) of PCK over different thresholds
- summarizes the ROC curve in single value
  - good for ranking approaches with different precision-recall tradeoffs



Chamfer distance

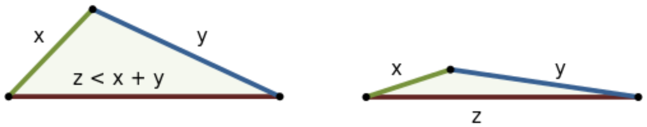
Important for Assignment 3!

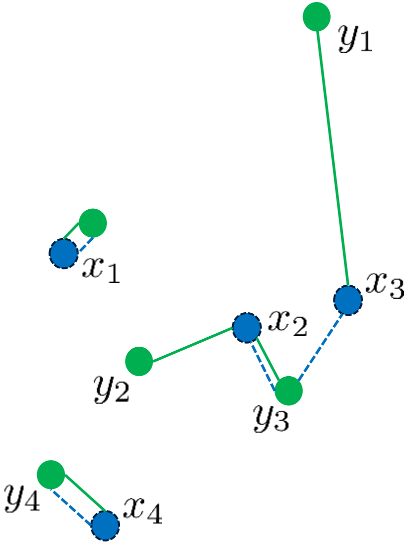
A distance between point clouds without correspondence

- sum of distances between closest points
- bi-directional
  - closest point of y in Y for all x in X
  - closest point of x in X for all y in Y

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

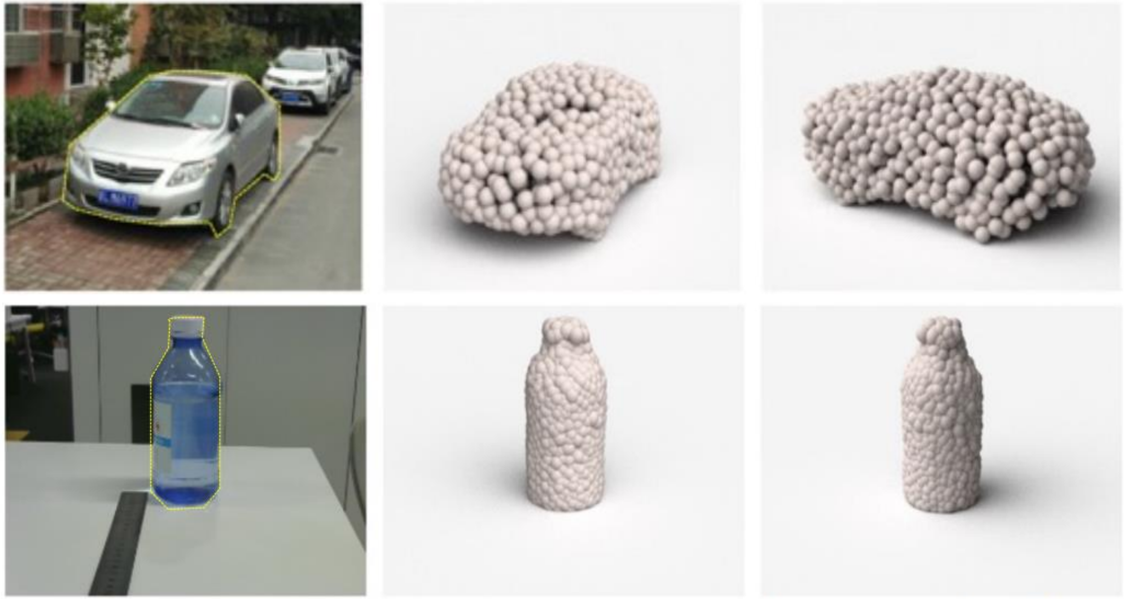
  - is not a *distance function* in the mathematical sense, because the triangle inequality does not hold





A Point Set Generation Network for 3D Object Reconstruction from a Single Image

The chamfer distance is good for cases where points don't have a semantic meaning, by contrast to human keypoints.



Input Reconstructed 3D point cloud

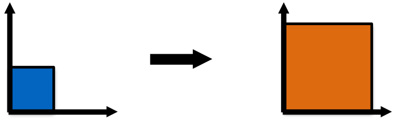
Shape completion

3D transformations

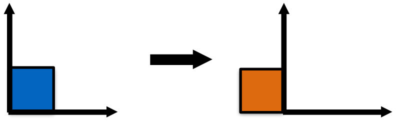
Literature: Multiple View Geometry in Computer Vision
by Richard Hartley and Andrew Zisserman
PDF available online. E.g.: https://github.com/darknight1900/books

Linear transformations in 2D

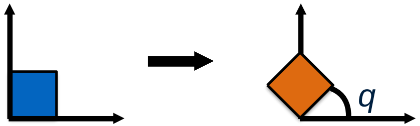
scaling: $\begin{bmatrix} \mathbf{v}'_x \\ \mathbf{v}'_y \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} \mathbf{v}_x \\ \mathbf{v}_y \end{bmatrix}$



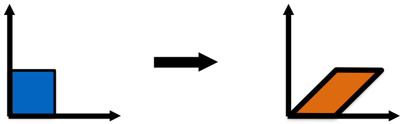
reflection: $\begin{bmatrix} \mathbf{v}'_x \\ \mathbf{v}'_y \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{v}_x \\ \mathbf{v}_y \end{bmatrix}$



rotation: $\begin{bmatrix} \mathbf{v}'_x \\ \mathbf{v}'_y \end{bmatrix} = \begin{bmatrix} \cos(q) & -\sin(q) \\ \sin(q) & \cos(q) \end{bmatrix} \begin{bmatrix} \mathbf{v}_x \\ \mathbf{v}_y \end{bmatrix}$



shear: $\begin{bmatrix} \mathbf{v}'_x \\ \mathbf{v}'_y \end{bmatrix} = \begin{bmatrix} 1 & c \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{v}_x \\ \mathbf{v}_y \end{bmatrix}$

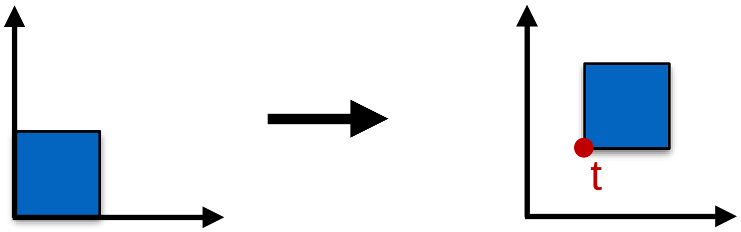


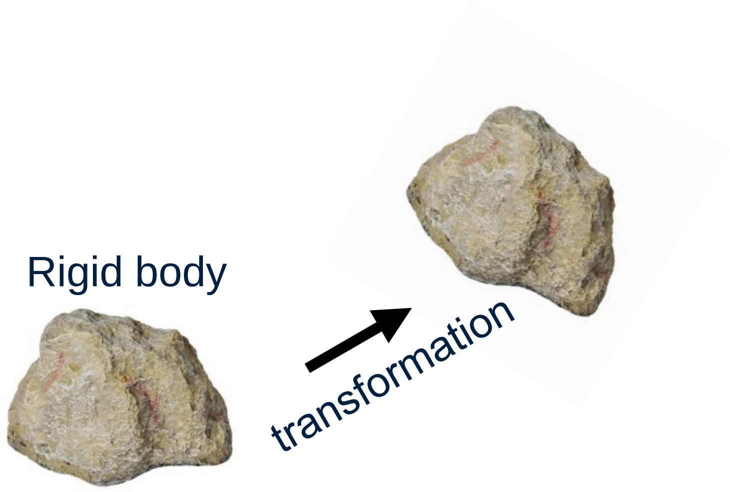
Rigid transformations (isometries)

Definition: Transformations that don't change the shape of an object, i.e. preserve lengths (an isometry)

- Rotation (linear)
- Reflection (linear)
- Translation (non-linear)

$$\begin{bmatrix} \mathbf{v}'_x \\ \mathbf{v}'_y \end{bmatrix} = \begin{bmatrix} t_x \\ t_y \end{bmatrix} + \begin{bmatrix} \mathbf{v}_x \\ \mathbf{v}_y \end{bmatrix}$$



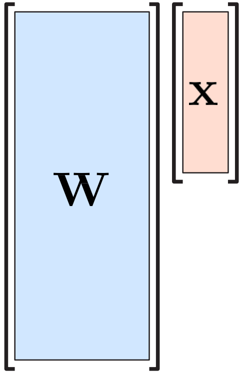


Affine transformations & augmented matrix and vector

- Can express rigid transformations
  - **Translation**
  - Rotation
  - Reflection
- And any other linear transformation
  - shear
  - scale

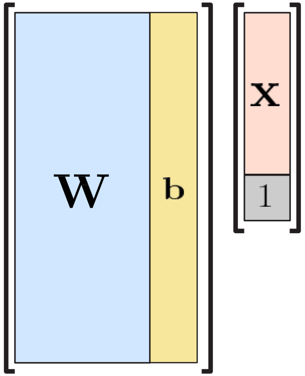
Linear

$$f(\mathbf{x}) = \mathbf{W}\mathbf{x}$$



Affine

$$f(\mathbf{x}) = \tilde{\mathbf{W}} \cdot \tilde{\mathbf{x}}$$



$$\text{with } \tilde{\mathbf{W}} = \begin{pmatrix} \mathbf{w}_{1,1} & \mathbf{w}_{1,2} & \cdots & \mathbf{w}_{1,n} & b_1 \\ \mathbf{w}_{2,1} & \mathbf{w}_{2,2} & \cdots & \mathbf{w}_{2,n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \end{pmatrix}$$

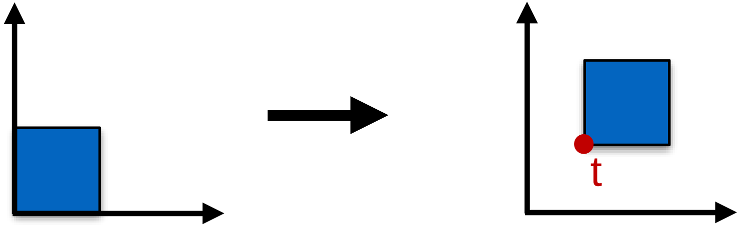
$$\tilde{\mathbf{x}} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n, {\color{red}1})$$

Rigid transformations

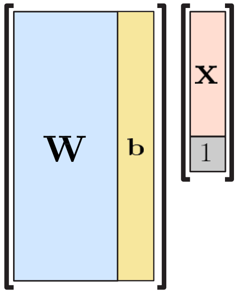
Definition: Transformations that don’t change the shape of an object, i.e. preserve lengths (an isometry)

- Rotation (linear)
- Reflection (linear)
- Translation (affine)

$$\begin{bmatrix} \mathbf{v}'_x \\ \mathbf{v}'_y \end{bmatrix} = \begin{bmatrix} t_x \\ t_y \end{bmatrix} + \begin{bmatrix} \mathbf{v}_x \\ \mathbf{v}_y \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix} \begin{bmatrix} \mathbf{v}_x \\ \mathbf{v}_y \\ 1 \end{bmatrix}$$



General shape



Rigid body

Rigid transformations in 3D

Example: Camera transformation, mapping a point p from world to camera coordinates

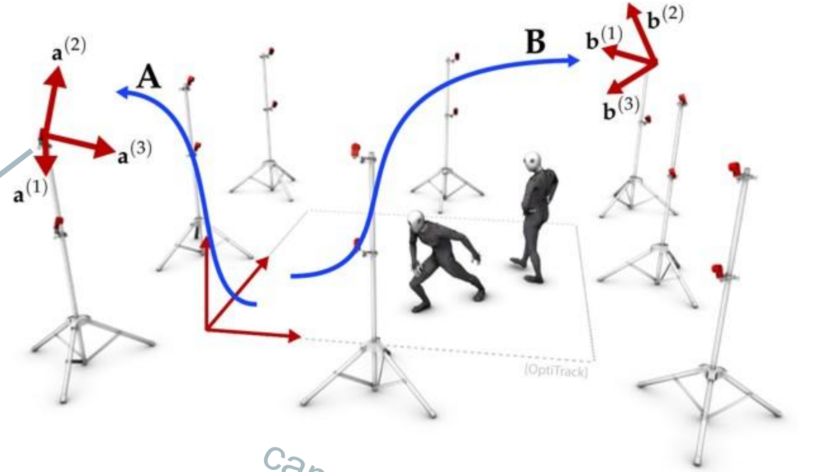
$$\mathbf{p}_{\text{cam}} = \left[\ \mathbf{R}_{\text{world}\rightarrow\text{cam}} \ \middle| \ \mathbf{t}_{\text{world}\rightarrow\text{cam}} \ \right] \mathbf{p}_{\text{world}}$$

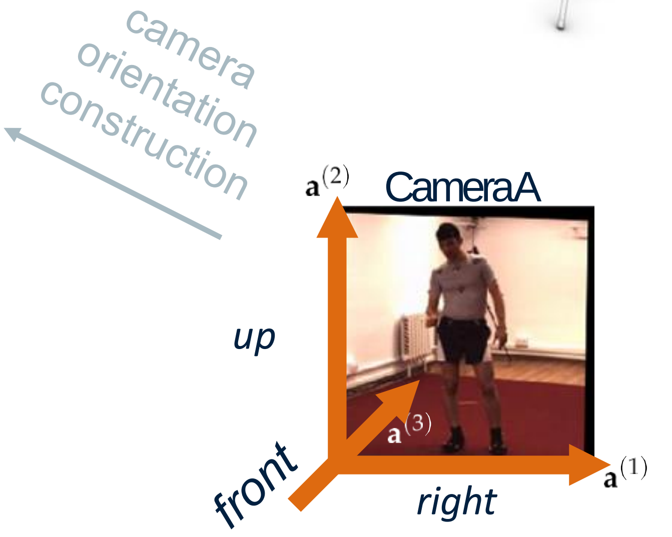
$$\mathbf{t}_{\text{cam}\rightarrow\text{world}} = \mathbf{c} = \text{camera position}$$

$$\mathbf{R}_{\text{cam}\rightarrow\text{world}} = \begin{pmatrix} a_x^{(1)} & a_x^{(2)} & a_x^{(3)} \\ a_y^{(1)} & a_y^{(2)} & a_y^{(3)} \\ a_z^{(1)} & a_z^{(2)} & a_z^{(3)} \end{pmatrix} = \begin{pmatrix} \text{right}_x & \text{up}_x & \text{front}_x \\ \text{right}_y & \text{up}_y & \text{front}_y \\ \text{right}_z & \text{up}_z & \text{front}_z \end{pmatrix}$$

$$\mathbf{R}_{\text{world}\rightarrow\text{cam}} = \mathbf{R}^{-1}_{\text{world}\rightarrow\text{cam}} = \mathbf{R}^{\top}_{\text{world}\rightarrow\text{cam}}$$

$$\mathbf{t}_{\text{world}\rightarrow\text{cam}} = -\mathbf{R}^{\top}_{\text{world}\rightarrow\text{cam}} \text{camera position}$$





Simple & intuitive in affine transformation matrix form

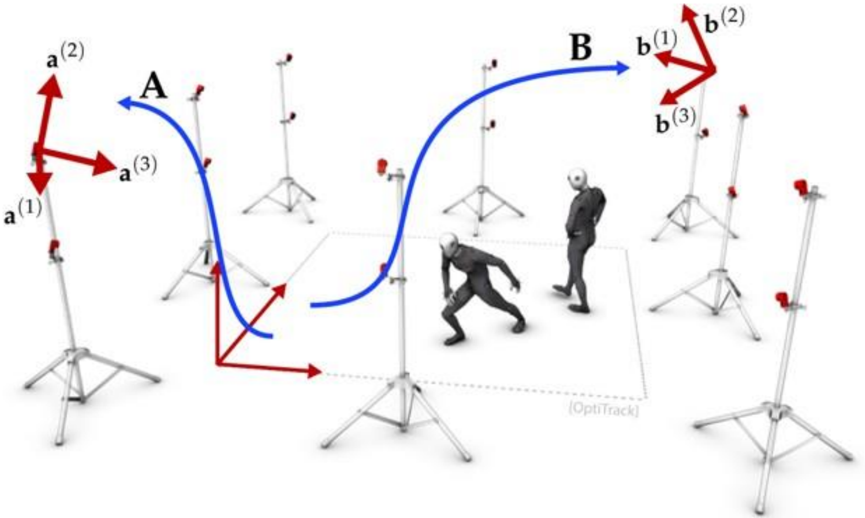
$$\left[\ \mathbf{R}_{\text{world}\rightarrow\text{cam}} \ \middle| \ \mathbf{t}_{\text{world}\rightarrow\text{cam}} \ \right] = \left[\ \mathbf{R}_{\text{cam}\rightarrow\text{world}} \ \middle| \ \mathbf{t}_{\text{cam}\rightarrow\text{world}} \ \right]^{-1}$$

3D affine transformations

- widely used in computer graphics and computer vision
- a chain of linear maps is a linear map
  - to map from one camera to the other
    - via world coordinates

$$\left[\ \mathbf{R}_{\mathrm{cam}_a \to \mathrm{cam}_b}\ \middle|\ \mathbf{t}_{\mathrm{cam}_a \to \mathrm{cam}_b}\ \right] = \left[\ \mathbf{R}_{\mathrm{cam}_b \to \mathrm{world}}\ \middle|\ \mathbf{t}_{\mathrm{cam}_b \to \mathrm{world}}\ \right]^{-1} \left[\ \mathbf{R}_{\mathrm{cam}_a \to \mathrm{world}}\ \middle|\ \mathbf{t}_{\mathrm{cam}_a \to \mathrm{world}}\ \right]$$

- a chain of affine transformation matrices is an affine transformation matrix



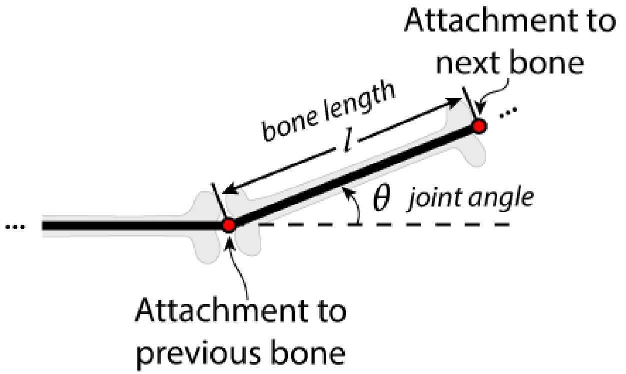
Skeleton representation

Representation: Bones connected by rotational joints

Size: J x 3 + J x 3 (J: # joints, 3: axis + angle, 3: 3D position)

or size: J x 3 + B x 1 (3: axis + angle, B: # bones)

- A hierarchical skeleton approximating anthropology
- Joint rotation is modelled by axis+angle (3 DOF), exponential maps (3-4 DOF) , quaternions (4 DOF) and euler angles (3 DOF)

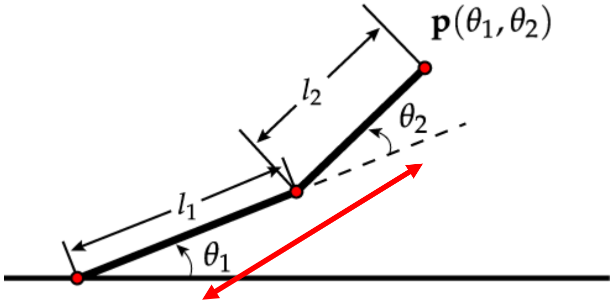


Benefits

- Common for human and animal motion capture
- Enforces skeleton constraints explicitly
- Is efficient to optimize (human tree/star skeleton structure)

Drawbacks

- Only approximates the human skeleton (e.g., the shoulder joint is complex to model properly)
- Indirect representation
  - the end effector position depends on all parent joints



Forward and inverse kinematics

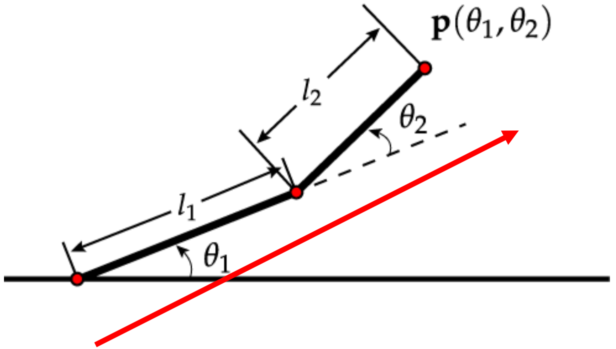
Forward and inverse kinematics

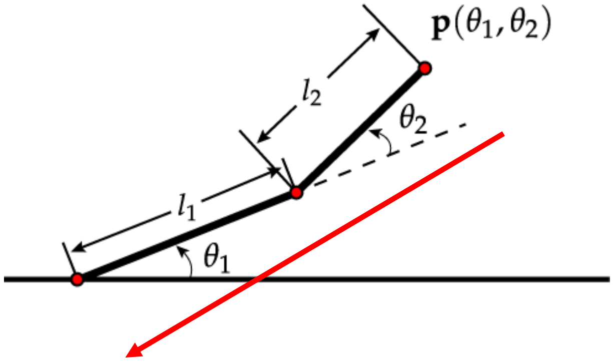
Forward kinematics

- given joint axis, angle, and skeleton hierarchy
- compute joint locations
  - start at the root (neck or head)
    - rotate all child joints (down the hierarchy) by θ
  - iteratively continue from parent to child
  - until end-effector is reached
- *a chain of affine transformations!*

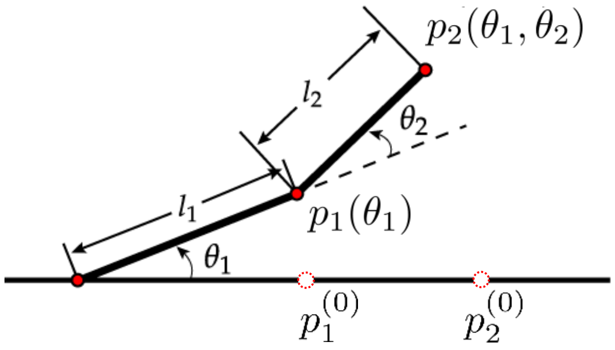
Inverse kinematics

- given skeleton hierarchy and goal location
- optimize joint angles
  - iteratively, gradient descent (as for NNs)
- minimize distance between end effector (computed by forward kinematics) and goal locations





Forward kinematics, linear or not?



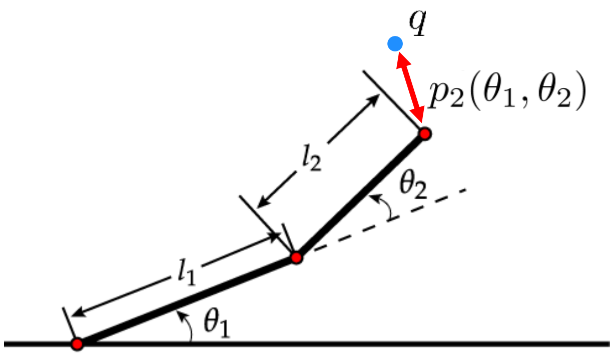
Forward kinematics

- non-linear in the angle (due to cos and sin)

$$R_1 = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 \\ \sin\theta_1 & \cos\theta_1 \end{bmatrix} \qquad R_2 = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 \\ \sin\theta_2 & \cos\theta_2 \end{bmatrix}$$

- linear/affine given a set of rotation matrices

$$p_2(\theta_1, \theta_2) = R_1 p_1^{(0)} + R_2 R_1 \left(p_2^{(0)} - p_1^{(0)}\right)$$



Inverse kinematics

- minimize objective to reach goal location q

$$O(\theta_1, \theta_2) = \|q - p_2(\theta_1, \theta_2)\|$$

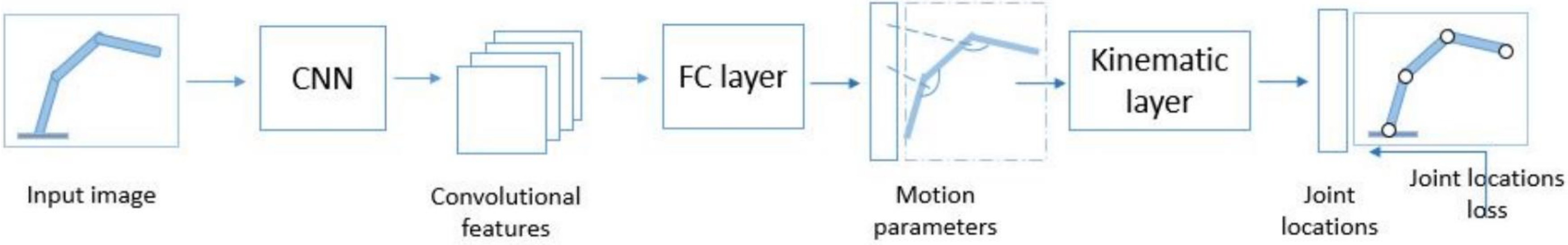
- difficult, due to nonlinear dependency on theta

Bonus: implement IK yourself (perhaps use PyTorch?)
https://rgl.s3.eu-central-1.amazonaws.com/media/pages/hw4/CS328_-_Homework_4_3.ipynb

Deep Kinematic Pose Regression

Regressing joint angles and bone length instead of joint position

- Change of coordinates enforces prior information
  - bone length symmetry
  - constant bone length (over time)



- Is better than predicting points and enforcing symmetry explicitly
  [Imposing Hard Constraints on Deep Networks: Promises and Limitations]
  - Feasible using Karush-Kuhn-Tucker Conditions
  - Did not work well in practice

<u>Positively Negative</u>

Workshop on Negative Results in Computer Vision. CVPR 2017

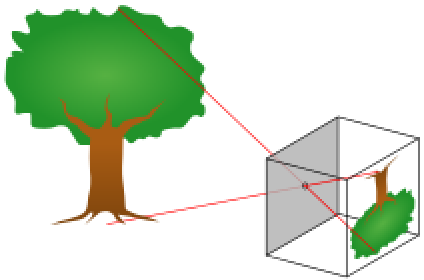
Keep it SMPL: Automatic Estimation of 3D Human Pose and Shape from a Single Image

Regression of SMPL parameters from images using deep learning

parameters:

- axis-angle of all J joints
- a surface mesh
- skinning weights that associate each vertex to neighboring joints (weighted sum)

Projective transformation



Pinhole camera model

[https://en.wikipedia.org/wiki/Pinhole_camera_model]

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \frac{f}{x_3} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

Projection in 3D Euclidean coordinates

Perspective projection

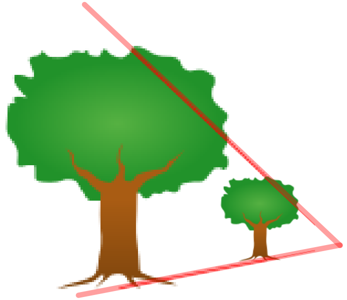
- inversely proportional to depth
  - usually the third coordinate, denoted by x_3 or z
  - proportional to the focal length, the distance of the focal point to the image plane
- non-linear, non-affine
- studied in the field of projective geometry, a sub-field of algebraic geometry

Projective transformation & Homogeneous coordinates

Equivalence in homogeneous coordinates

- Definition: vectors scaled by any constant lambda are equivalent

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{m-1} \\ x_m \end{bmatrix} = \begin{bmatrix} x_1\lambda \\ x_2\lambda \\ \vdots \\ x_{m-1}\lambda \\ x_m\lambda \end{bmatrix} = \begin{bmatrix} x_1/x_m \\ x_2/x_m \\ \vdots \\ x_{m-1}/x_m \\ 1 \end{bmatrix}$$



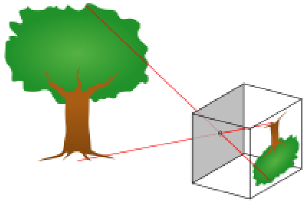
- models perspective transformations (projection) as a linear transformation

$$\begin{pmatrix} y_1 \\ y_2 \\ 1 \end{pmatrix} \sim \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ 1 \end{pmatrix}$$

Projection in Homogeneous coordinates

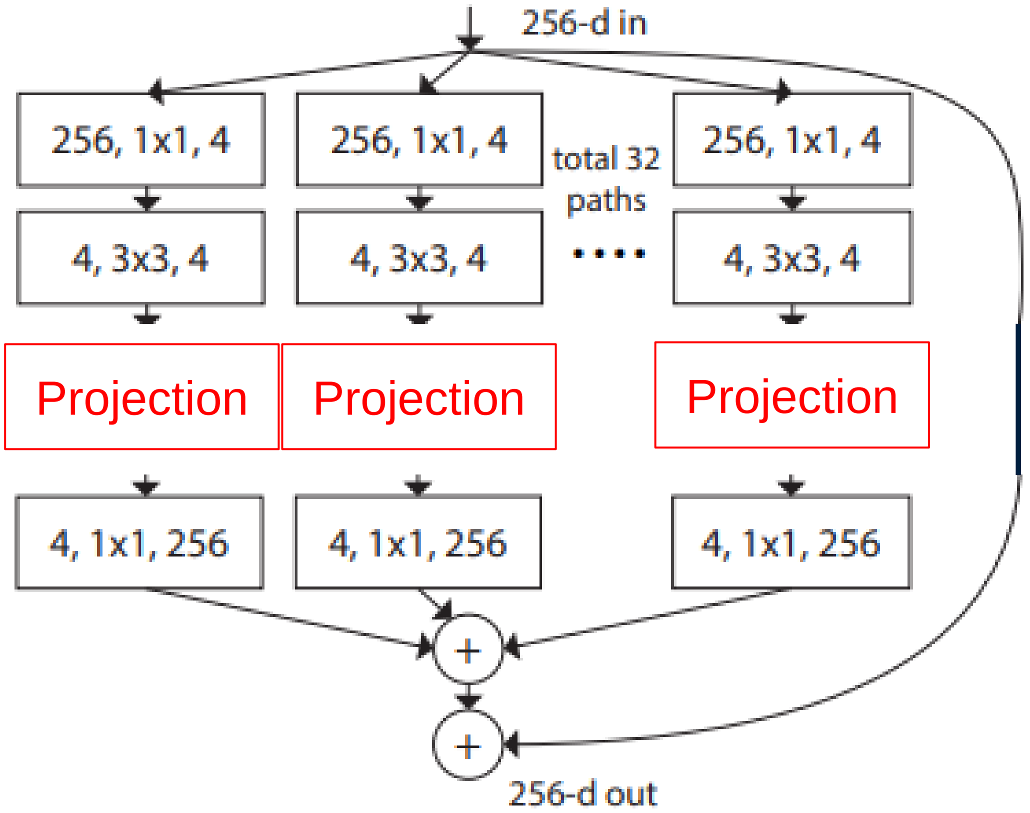
$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = -\frac{f}{x_3} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

Projection in Euclidean coordinates



Project Idea: Projective transformations within CNNs (ProjResNext)

- The basis building block of NNs are affine transformations (linear + bias)
- Idea: Use projective transformations instead
- Tasks:
  - Literature review, has this been tried?
  - How to initialize (to prevent vanishing gradients)
  - Do we need to adapt other NN structures, e.g., Batch Norm?
  - Will it be better?



3D representations

Depth maps

Representation: a depth value per pixel

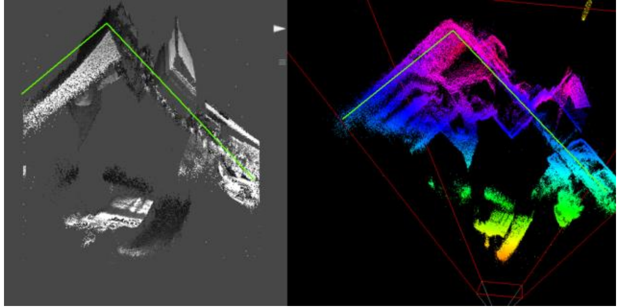
- Size: W x H (Width x Height)
- A 2.5 D representation
  - Continuous in Z (depth)
  - Discrete in X,Y (horizontal and vertical)

Use cases

- Monocular and stereo reconstruction
- Novel view synthesis
- Well-suited for 2D convolution operations

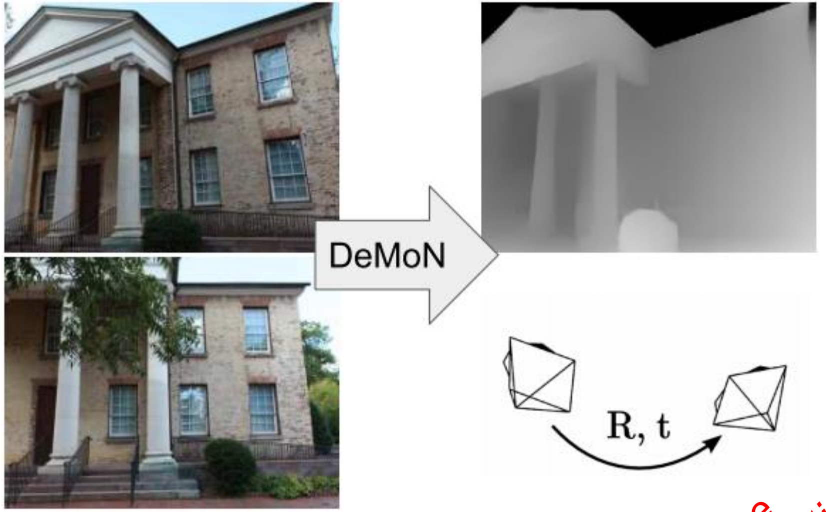
Drawbacks

- Missing parts and holes
- No semantics/correspondence between frames



https://stackoverflow.com/questions/37198974/microsoft-kinect-v2-unity-3d-depth-warping

Kinect depth map viewed from the top. *its sparse!*



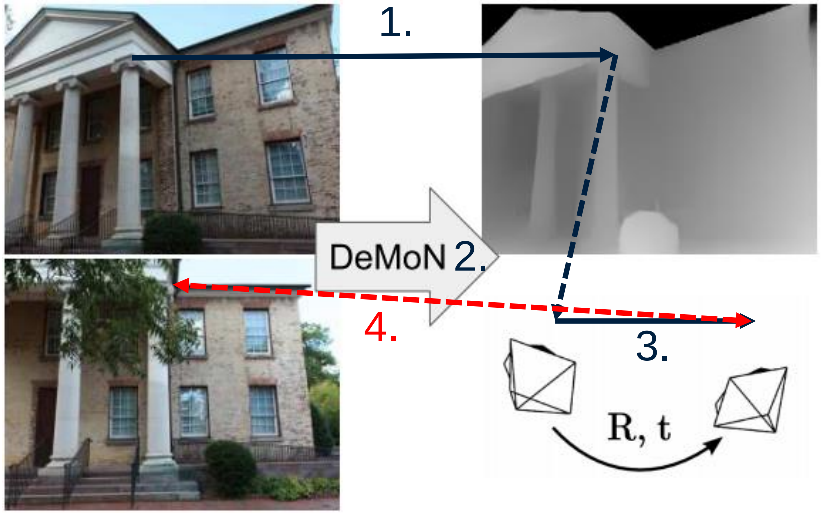
[Ummenhofer et al. DeMoN: Depth and Motion Network for Learning Monocular Stereo]

affine transformation

Self-supervision in a nutshell

a) Remove part of the input
   - e.g. right from left image

b) Train a network to predict the removed part
   - enforce additional constraints
     - geometric
     - temporal
     - …



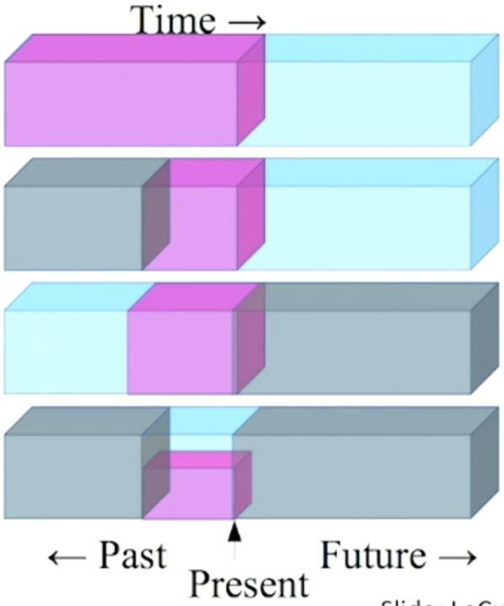
[Ummenhofer et al. DeMoN: Depth and Motion Network for Learning Monocular Stereo]

DeMoN

1. Estimate depth from the image with a NN
2. Estimate camera motion from image pair with a NN
3. Project depth map from first image to second image
   - copy associated pixel color
4. Compute loss between the pixel color of the first image projected on the second

Self-supervision by LeCun

- Predict any part of the input from any other part.
- Predict the future from the past.
- Predict the future from the recent past.
- Predict the past from the present.
- Predict the top from the bottom.
- Predict the occluded from the visible
- Pretend there is a part of the input you don't know and predict that.



Slide: LeCun

Point cloud

Representation: A collection of 3D points

- Size: N x D (Number of points, space dimension)
- Sparse 3 D locations (usually, can be in a higher-dimensional)
  - Continuous and adaptive detail

Benefits

- Well suited for structure from motion form keypoints
- Compact representation of sparse keypoint locations
  - human joints, object edges, …
- Ordered point clouds carry semantics (e.g., first point is the head, the second the neck position)

Drawbacks

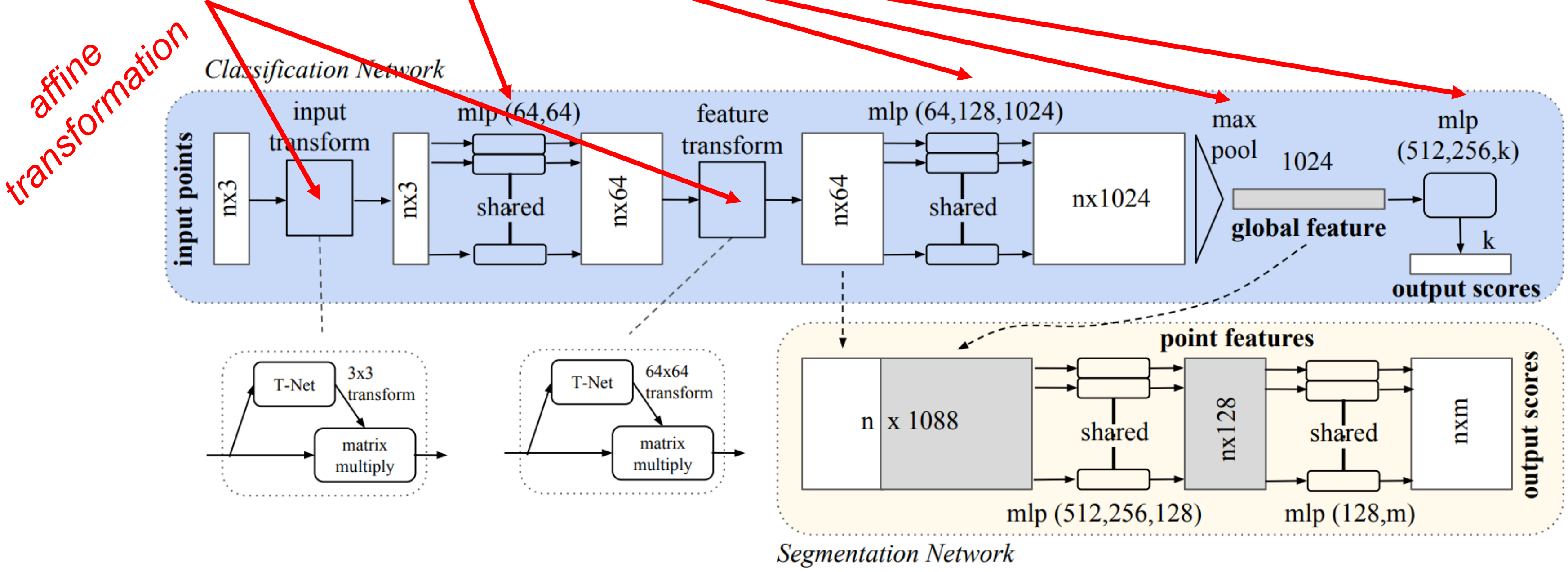
- Unstructured, not well suited for convolutions etc.
- No orientation information

[Snavely et al., Photo Tourism: Exploring Photo Collections in 3D]

PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation

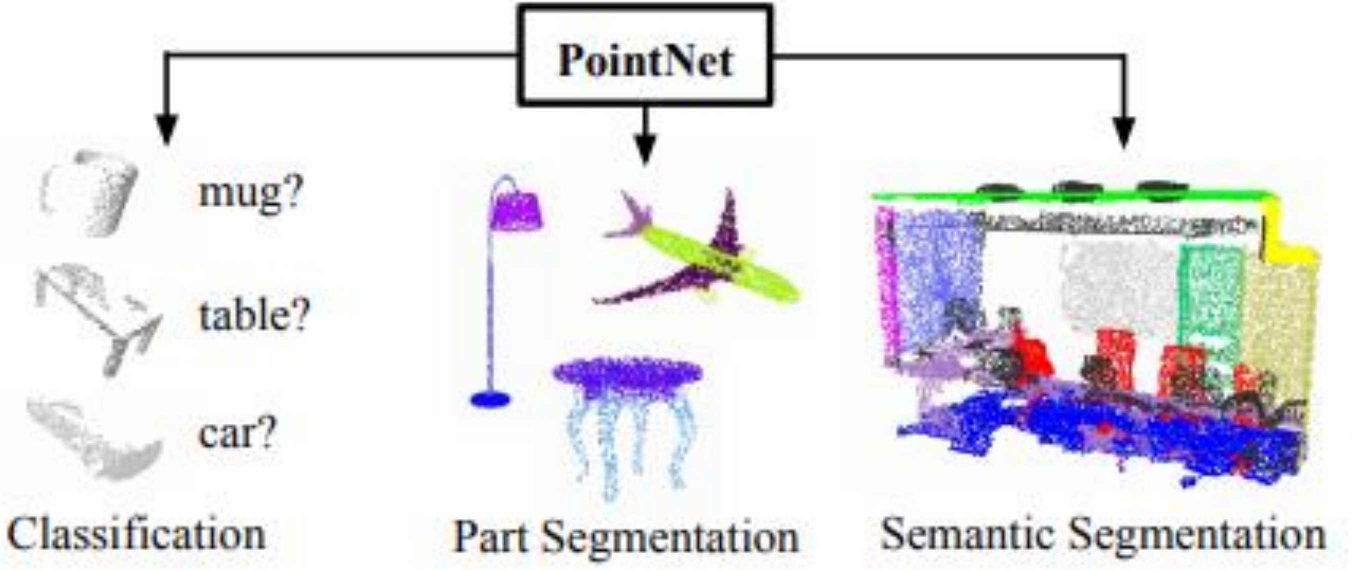
A network architecture to make point cloud processing invariant to

- the point cloud order
- global rigid transform.



PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation

Applications



MonoPerfCap: Human Performance Capture from Monocular Video