Differentiable Approaches for Robust Estimation

René Ranftl, Intel Labs

RANSAC in 2020: CVPR tutorial
Differentiable approaches

Differentiable == learnable

• Deep networks in the loop
  - Trained with stochastic gradient descent
  - Learn statistical properties of the data
  - Can integrate additional information

• Challenges:
  - High-dimensional, unordered data
  - Acquiring training data
  - Guaranteeing transfer
  - Design of loss functions
Differentiable approaches
An active research topic

- Current work mostly targeting specific applications
  - Need for training data
  - Requirements for integrating domain knowledge
- Two research clusters:
  - Two-view correspondence filtering (2D-2D)
  - 3D Point cloud registration (3D-3D)
- Other applications: Camera localisation (2D-3D), vanishing point estimation
Differentiable approaches
An active research topic

• Current work mostly targeting specific applications
  - Need for training data
  - Requirements for integrating domain knowledge

• Two research clusters:
  - Two-view correspondence filtering (2D-2D)
  - 3D Point cloud registration (3D-3D)

• Other applications: Camera localisation (2D-3D), vanishing point estimation
### Differentiable vs. classic

<table>
<thead>
<tr>
<th></th>
<th>Classic approach</th>
<th>Differentiable approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training data</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Architecture specification</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Loss function</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>All parts differentiable</td>
<td>No</td>
<td>(Yes)</td>
</tr>
<tr>
<td>Minimal solvers</td>
<td>(Yes)</td>
<td>(No)</td>
</tr>
<tr>
<td>Sampling-based</td>
<td>Yes</td>
<td>(No)</td>
</tr>
<tr>
<td>IRLS</td>
<td>No</td>
<td>(Yes)</td>
</tr>
</tbody>
</table>

- Promises of differentiable methods:
  - Adapt to scenario
  - Improve with more data and capacity
  - Leverage additional information
  - Complementary to classic approaches
Overview
A prototypical pipeline

Training

Input → Trainable → Feature extraction + Inlier classification → Inliers → Classification loss → “Groundtruth”
A prototypical pipeline

Training

Input → Trainable → Feature extraction + Inlier classification → Inliers → Model extraction → Model

Regression loss

“Groundtruth”
A prototypical pipeline

Training

Input → Feature extraction + Inlier classification → Inliers → Model extraction → Model

Classification loss → “Groundtruth”

Regression loss → “Groundtruth”
A prototypical pipeline

Part I: Feature extractors

Input \rightarrow \text{Trainable} \rightarrow \text{Feature extraction + Inlier classification} \rightarrow \text{Inliers} \rightarrow \text{Model extraction} \rightarrow \text{Model}

\text{Classification loss} \downarrow \quad \text{Regression loss} \downarrow

\text{“Groundtruth”} \quad \text{“Groundtruth”}
A prototypical pipeline

Training

Input -> Feature extraction + Inlier classification -> Inliers

Classification loss -> "Groundtruth"

Model extraction -> Model

Regression loss -> "Groundtruth"

Part II: Integrating geometry
A prototypical pipeline

Training

- Feature extraction + Inlier classification
- Inliers
- Model extraction
- Model

Part III: Loss functions

- Classification loss
  - “Groundtruth”
- Regression loss
  - “Groundtruth”
A prototypical pipeline

Training

Part IV: Practical considerations

Input → Feature extraction + Inlier classification → Inliers → Model extraction → Model

Trainable

Classification loss → “Groundtruth”

Regression loss → “Groundtruth”
A prototypical pipeline

Testing

Input → Feature extraction + Inlier classification → Model extraction → Postprocessing → Model

Part IV: Postprocessing
Part I: Feature extractors

Trainable

Input

Feature extraction + Inlier classification

Inliers

Model extraction

Model

Classification loss

“Groundtruth”

Regression loss

“Groundtruth”
Input and output data

**Input**

- Putative correspondences
- Putative correspondences + additional properties
- Points in high-dimensional space

```
N    [ p_1  p_2  ...  p_N ]
      ^      ^        ^
      P      P        P
```

**Output**

- Fundamental/Essential Matrix, Homography
- Camera pose
- Parametric curve

Images © Wikimedia Commons
Requirements

• Problems require networks defined on sets:
  - Permutation-equivariance
  - Independent of size

• Prototypical networks:
  - PointNet [Qi 2016]
  - DeepSets [Zaheer 2017]
Set functions in deep learning

Permutation-equivariance

A function is permutation-equivariant if and only if

$$f([p_{\pi(i)}, \ldots, p_{\pi(N)}]) = [f_{\pi(1)}(p), \ldots, f_{\pi(N)}(p)]$$

for all permutations $\pi$

A network layer $f_\Theta(p) = \sigma(\Theta p)$

is permutation-equivariant if and only if

$$\Theta = \lambda \begin{bmatrix} 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 1 \end{bmatrix} + \gamma \begin{bmatrix} 1 & 1 & \ldots & 1 \\ 1 & 1 & \ldots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \ldots & 1 \end{bmatrix}$$

$\lambda, \gamma \in \mathbb{R}$

Set functions in deep learning

Permutation-equivariance

A function is permutation-equivariant if and only if

$$f([p_{\pi(1)}, \ldots, p_{\pi(N)}]) = [f_{\pi(1)}(p), \ldots, f_{\pi(N)}(p)]$$

for all permutations $\pi$

A network layer $f_\Theta(p) = \sigma(\Theta p)$

is permutation-equivariant if and only if

$$\Theta = \lambda \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} + \gamma \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix}$$

$\lambda, \gamma \in \mathbb{R}$

Not the complete space of permutation-equivariant functions!

Simplistic model!

PointNet-based architectures

Basic building block

\[ \Theta = \lambda \begin{bmatrix} 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 1 \end{bmatrix} + \gamma \begin{bmatrix} 1 & 1 & \ldots & 1 \\ 1 & 1 & \ldots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \ldots & 1 \end{bmatrix} \]
PointNet-based architectures

Basic building block

\[ \Theta = \lambda \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} + \gamma \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix} \]
PointNet-based architectures

Basic building block

Feature extraction block
\[ \mathbb{R}^{N \times F_i} \rightarrow \mathbb{R}^{N \times F_j} \]
PointNet-based architectures

Basic building block
PointNet-based architectures

Basic building block

Input (NxP) → Context → Context → Context → Context
PointNet-based architectures

Basic building block

• Additional components
  - Residual connections
  - Batch normalization
  - All standard non-linearities
PointNet-based architectures

Existing architectures

• Permutation-equivariant context:
  - Context Normalization [Yi 2018, Ranftl 2018]
  - Canonical ordering [Zhao 2019, Zhang 2019]
  - Attention [Sun 2020]

• Other architectures
  - High-dimensional convolutions [Choy 2020a]
Context Normalization

AKA Instance Normalization

• Normalize feature to zero-mean and unit variance

\[
\mu = \frac{1}{N} \sum_{i=1}^{N} f_i
\]

\[
CN(f_i) = \frac{f_i - \mu}{\sigma}
\]

\[
\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (f_i - \mu)^2}
\]

• Operation distributes information globally

• Simple and surprisingly effective


Image by Eduard Trulls
(https://etrulls.github.io/slides/cvpr18_slides.pdf)
Attentive Context Normalization

Robust version of CN

- Context Normalization treats every point equally
- Attentive CN uses weighted global statistics to focus normalization on inliers

\[
ACN(f_i) = \frac{f_i - \mu}{\sigma}
\]

\[
\mu = \frac{1}{N} \sum_{i=1}^{N} w_i f_i
\]

\[
\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^{N} w_i (f_i - \mu)^2}
\]

Order-Aware Networks
Inferring a canonical ordering

- Proposes three operations
  - Map points to a fixed set of clusters in canonical order
  - Correlate clusters feature space
  - Unpool to append context to points

Order-Aware Networks

Inferring a canonical ordering

• Proposes three operations
  - Map points to a fixed set of clusters in canonical order
  - Correlate clusters feature space
  - Unpool to append context to points

Order-Aware Networks v2
Inferring a canonical ordering

- Original OANet proposed one layer
- Can stack multiple networks
  - “Stacked Hourglass”
  - Bottleneck layer for reduced computation

<table>
<thead>
<tr>
<th>Dataset</th>
<th>mAP (w/ &amp; w/o RANSAC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>YFCC100M</td>
<td></td>
</tr>
<tr>
<td>OANetV1</td>
<td>52.2/39.3</td>
</tr>
<tr>
<td>OANetV2</td>
<td>56.8/50.5</td>
</tr>
</tbody>
</table>

Thanks to Chen Hongkai for providing slides!
NM-Net
A graph-based view

- Adaptively build KNN graph based on geometric consistency
- Based on local structural compatibility
- Perform grouping and convolution on graph

C. Zhao, Z. Cao, C. Li, X. Li, and J. Yang. “NM-Net: Mining Reliable Neighbors for Robust Feature Correspondences”, CVPR 2019
NM-Net v2

A graph-based view

- Iterate multiple times to refine weights
High-Dimensional Convolutional Networks
Leveraging insights from CNNs

• Define neighborhoods directly in voxelized high-dimensional space:
  - 4D for image matching
  - 6D for 3D point cloud registration

• Define sparse operations in this space
  - Dense not tractable
  - Efficient implementation in MinkowskiEngine

https://github.com/StanfordVL/MinkowskiEngine

High-Dimensional Convolutional Networks
Leveraging insights from CNNs

- U-Net-style network
- Expanding receptive field

High-Dimensional Convolutional Networks
Leveraging insights from CNNs

- U-Net-style network
- Expanding receptive field

High-Dimensional Convolutional Networks
Leveraging insights from CNNs

- U-Net-style network
- Expanding receptive field
Feature extractors

An intermediate conclusion

• Data is different from other applications
  - Unordered data
  - High-dimensional
  - Non-metric relations
• Many techniques from “standard” DL carry over
  - Multi-scale processing
  - Normalizing batch statistics
  - Residual connection
  - Attention
Part II: Integrating Geometry

- Feature extraction + Inlier classification
- Model extraction
- Classification loss
- Training loss
- "Groundtruth"
Domain Knowledge

• Feature extractors are generally applicable
  - Example I: Append a regression layer that directly outputs model parameters (usually doesn’t work well)
  - Example II: Append a layer that outputs inlier probabilities for each point

• Domain knowledge can help
  - Faster training
  - Less data
  - Better accuracy
Domain Knowledge

- Good news 😊
  - Problems can be defined as **optimization problems**
  - Optimization can be integrated in deep networks
    - Closed-form solutions
    - Implicit differentiation
    - Unrolled optimization
  - Least-squares as a workhorse
Least Squares

Basic setup

• Outlier-free problems can be naturally formulated as Least-Squares
  
  - Standard least-squares

\[
\mathbf{x} = \underset{\mathbf{x}}{\text{arg min}} \, \sum_{i=1}^{N} \| \mathbf{(A)}_i \cdot \mathbf{x} - \mathbf{b}_i \|^2
\]

  - Homogenous least-squares (for DLT-type problems)

\[
\mathbf{x} = \underset{\mathbf{x}: \| \mathbf{x} \| = 1}{\text{arg min}} \, \sum_{i=1}^{N} \| \mathbf{(A)}_i \cdot \mathbf{x} \|^2
\]
Least Squares

Examples

• Outlier-free problems can be naturally formulated as Least-Squares

  - Homogenous least-squares (for DLT-type problems)

Hyperplanes:

\[(A)_i = p_i\]

Essential/Fundamental Matrix:

\[(A)_i = \text{vec}(p_i(p'_i)^\top)\]

Homography:

\[(A)_{2i-1:2i} = \begin{pmatrix}
-p_i^\top & 0 \\
0 & -p_i^\top & (p'_i)_1p_i^\top \\
0 & (p'_i)_2p_i^\top
\end{pmatrix}\]
Solving Least Squares

Is differentiable (almost everywhere) for overdetermined systems

- Least Squares problems solved using Singular Value Decomposition (SVD)
- SVD is differentiable (as long as there are no repeated singular values)

\[ \begin{align*}
\frac{\partial g}{\partial X} &= U \left\{ 2 \Sigma \left( K^T \circ \left( V^T \frac{\partial g}{\partial v_d} \right)_{\text{sym}} \right) \right\} V^T, \\
\text{where} \\
K_{ij} &= \begin{cases} 
\frac{1}{\sigma_i^2 - \sigma_j^2}, & \text{if } i \neq j \\
0, & \text{otherwise}
\end{cases} \\
\text{and } \sigma_i \text{ denotes the } i\text{-th singular value.}
\end{align*} \]

- Already implemented in PyTorch and Tensorflow

```python
import torch
import tensorflow as tf

# Example in PyTorch
U, S, V = torch.svd(input=True, some=True, compute_uv=True, out=None)

# Example in Tensorflow
U, S, V = tf.linalg.svd(tensor=True, full_matrices=False, compute_uv=True, name=None)
```

This function returns a namedtuple \((U, S, V)\) which is the singular value decomposition of a input real matrix or batches of real matrices \(input\) such that \(input = U \times \text{diag}(S) \times V^T\).
Consider the problem

\[
\arg \min_x \sum_{i=1}^{N} \| (A)_i \cdot x - b_i \|_p
\]

IRLS algorithm:

\[
x^{t+1} = \arg \min_x \sum_{i=1}^{N} w_i(x^t) \| (A)_i \cdot x - b_i \|^2
\]

\[
w_i(x^t) = \| (A)_i \cdot x^t - b_i \|^{(p-2)}
\]

Also applicable for robust penalties
Iteratively Reweighted Least Squares
A classic tool in robust statistics

• Consider the problem

\[
\arg \min_x \sum_{i=1}^{N} \| (A)_i \cdot x - b_i \|_p
\]

• IRLS algorithm:

\[
x^{t+1} = \arg \min_x \sum_{i=1}^{N} w_i(x^t) \| (A)_i \cdot x - b_i \|^2
\]

\[
w_i(x^t) = \| (A)_i \cdot x^t - b_i \|^{(p-2)}
\]

• Also applicable for robust penalties

Can still be solved using SVD
Reweighted Least Squares

- Standard least squares

$$x^{t+1} = \arg \min_x \sum_{i=1}^{N} w_i(x^t) \| (A)_i \cdot x - b_i \|^2$$

- Homogenous least squares

$$x^{t+1} = \arg \min_{x: \|x\|=1} \sum_{i=1}^{N} w_i(x^t) \| (A)_i \cdot x \|^2$$
Reweighted Least Squares

- Standard least squares

\[ x^{t+1} = \arg \min_x \sum_{i=1}^N w_i(x^t) \| (A)_i \cdot x - b_i \|^2 \]

- Homogenous least squares

\[ x^{t+1} = \arg \min_x \sum_{i=1}^N w_i(x^t) \| (A)_i \cdot x \|^2 \]

Can be represented by neural network
Reweighted Least Squares

- Standard least squares

\[ x^{t+1} = \arg \min_x \sum_{i=1}^{N} w_i(x^t) \| (A)_i \cdot x - b_i \|^2 \]

- Homogenous least squares

\[ x^{t+1} = \arg \min_{x: \|x\|=1} \sum_{i=1}^{N} w_i(x^t) \| (A)_i \cdot x \|^2 \]
Neural Reweighted Least-Squares

Examples

• Define a regression loss [Yi 2017]

\[ \mathcal{L} = \min \{ \| E_{gt} \pm g(P, w) \| \} \]

• Iterate weight estimation and model estimation [Ranftl 2018]
Neural Reweighted Least-Squares

Examples

• Define a regression loss [Yi 2017]

\[ \mathcal{L} = \min \{ \|E_{gt} \pm g(P, w)\| \} \]

• Iterate weight estimation and model estimation [Ranftl 2018]

| PointCN UnA UnB OF L3 Geo Iter Known Unknown |  |
|---|---|---|---|---|---|---|---|---|---|
| ✓ | ✓ | ✓ | 34.36/13.93 | 47.98/23.55 |
| ✓ | ✓ | 34.38/14.04 | 47.93/24.10 |
| ✓ | ✓ | 36.33/17.88 | 49.65/28.78 |
| ✓ | ✓ | ✓ | 40.78/25.94 | 51.63/32.55 |
| ✓ | ✓ | ✓ | 39.69/26.04 | 50.70/30.48 |
| ✓ | ✓ | ✓ | 40.79/28.39 | 51.10/33.68 |
| ✓ | ✓ | ✓ | ✓ | 42.46/33.06 | 52.18/39.33 |

Table from [Zhang 2019]
Neural Reweighted Least-Squares

Examples

- Weighted procrustes analysis
- Estimate rotation and translation between 3D pointclouds

\[
\min \operatorname{Tr}((Y - RX - t1^T)W(Y - RX - t1^T)^T)
\]

- Network provides initialization for iterative refinement

RANSAC in the training loop

Neural Guidance

- Learn a hypothesis sampling distribution
  - Select outlier-free hypothesis sets with high probability
  - Parametrized by network

\[
\mathcal{L}(w) = \mathbb{E}_{H \sim p(H; w)}[l(h)]
\]

Learned hypothesis distribution
Hypothesis scoring

RANSAC in the training loop

Neural Guidance

- Learn a hypothesis sampling distribution
  - Select outlier-free hypothesis sets with high probability
  - Parametrized by network
    \[ \mathcal{L}(w) = \mathbb{E}_{H \sim p(H; w)}[l(h)] \]
    
- Hypothesis selection (and potentially scoring non-differentiable)
  - Stochastic estimate of the gradient
    \[ \frac{\partial}{\partial w} \mathcal{L}(w) = \mathbb{E}[l(h) \frac{\partial}{\partial w} \log p(H; w)] \]
  - Variance reduction

RANSAC in the training loop

Neural Guidance

- In the spirit of guided RANSAC variants
  - PROSAC
  - USAC
- Can be trained self-supervised
- Very flexible
- Can be combined with DSAC to also learn observations

Part III: Loss functions

Trainable

Feature extraction + Inlier classification

Inliers

Model extraction

Model

Classification loss

“Groundtruth”

Regression loss

“Groundtruth”
Loss functions

General tips

• Algebraic usually problematic
  - Errors arbitrarily scaled

• Use geometrically meaningful metrics
  - Sampson error, epipolar distance
  - Reprojection error
  - Point-to-point distance, point-to-plane distance

• Generally needs robust loss formulations
  - Certain points can have large influence
  - Clipping/robust loss formulations
Loss functions

An example

- From $F_{gt}$ can derive a set of ground truth inliers $p_j$
  ➤ Can always cast the problem as a classification problem!

- $F_{gt}$ is defined up to scale
  - Reasonable to have
    \[ \mathcal{L}(F') = \mathcal{L}(aF') \]
  - Naive loss $\mathcal{L}(F) = \|F - F_{gt}\|$ doesn’t have this property
  - Algebraic error can be scaled arbitrarily
    \[ \mathcal{L}(F') = \sum_j \|p_j^T F p_j\|^2 \]
Loss functions

An example

- From $F_{gt}$ can derive a set of ground truth inliers $p_j$

  ➤ Can always cast the problem as a classification problem!

- $F_{gt}$ is defined up to scale
  - Reasonable to have
    \[
    \mathcal{L}(F') = \mathcal{L}(aF')
    \]
  - Naive loss $\mathcal{L}(F') = \|F - F_{gt}\|$ doesn’t have this property
  - Algebraic error can be scaled arbitrarily
    \[
    \mathcal{L}(F') = \sum_j \|p_j^T F p_j\|^2
    \]
Loss functions

Design considerations

- Problem has a dual classification/regression character
  - Classify model inliers
    - Requires definition of inlier
    - Simply use binary cross-entropy (might require re-balancing if many more outliers than inliers)
Loss functions
Design considerations

• Problem has a dual classification/regression character
  - Classify model inliers
    • Requires definition of inlier
    • Simply use binary cross-entropy (might require re-balancing if many more outliers than inliers)
  - Regress a model
    • Requires geometry-aware layers
    • Enforce constraints (rank, orthogonality)
    • Parametrization can be an issue
Loss functions

An example for epipolar geometry

Find compatible points

\[
\min_{\hat{p}, p} \left( d(\hat{p}, p) + d(\hat{p}', p') \right) \quad \text{subject to} \quad p^\top Fp' = 0
\]

Measure distance of estimate to these points

\[
d(p, F_{gt}) < t
\]

Part IV: Practical Considerations

Trainable

Feature extraction + Inlier classification

Input → Inliers → Model extraction → Model

Classification loss → “Groundtruth”

Regression loss → “Groundtruth”
Training data

There is never enough

• Perfect training data is hard to get
  - No manual labeling possible
  - No measurement device exists

• Need sufficient size

• Need sufficient diversity

• Very application dependent

• Your method is only as good as your training data
Datasets

Two-view geometry

- Use SfM reconstructions as proxy-groundtruth

<table>
<thead>
<tr>
<th></th>
<th>Type</th>
<th>Size</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>YFCC100M</td>
<td>Phototourism</td>
<td>72 scenes</td>
<td>SfM</td>
</tr>
<tr>
<td>SUN3D</td>
<td>Indoor</td>
<td>~250 scenes</td>
<td>RGB-D</td>
</tr>
<tr>
<td>GL3D-v2</td>
<td>Drone footage + Small objects</td>
<td>540 scenes</td>
<td>SfM</td>
</tr>
<tr>
<td>Megadepth</td>
<td>PhotoTourism</td>
<td>196 scenes</td>
<td>SfM</td>
</tr>
<tr>
<td>Tanks and Temples</td>
<td>Single Camera</td>
<td>30 sequences</td>
<td>SfM</td>
</tr>
<tr>
<td>KITTI</td>
<td>Road Scenes</td>
<td>22 sequences</td>
<td>LiDaR</td>
</tr>
</tbody>
</table>

- Possible to use synthetic data (but mostly unexplored)
Datasets

Two-view geometry

- Use SfM reconstructions as proxy-groundtruth

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>YFCC100M</td>
<td>Phototourism</td>
<td>SfM</td>
</tr>
<tr>
<td>SUN3D</td>
<td>Indoor</td>
<td>RGB-D</td>
</tr>
<tr>
<td>GL3D-v2</td>
<td>Drone footage + Small objects</td>
<td>SfM</td>
</tr>
<tr>
<td>Megadepth</td>
<td>PhotoTourism</td>
<td>SfM</td>
</tr>
<tr>
<td>Tanks and Temples</td>
<td>Single Camera</td>
<td>SfM</td>
</tr>
<tr>
<td>KITTI</td>
<td>Road Scenes</td>
<td>LiDaR</td>
</tr>
</tbody>
</table>

- Possible to use synthetic data (but mostly unexplored)
Datasets

Two-view geometry

- Diversity (mostly) dependent on capture mode

Unconstrained

Handheld video

Constrained video
# Datasets

## 3D Registration

<table>
<thead>
<tr>
<th>Datasets</th>
<th>Type</th>
<th>Size</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D Match</td>
<td>Indoor</td>
<td>62 scenes</td>
<td>RGB-D</td>
</tr>
<tr>
<td>KITTI</td>
<td>Outdoor</td>
<td>~70 sequences</td>
<td>LiDaR</td>
</tr>
<tr>
<td>ICL-NIUM</td>
<td>Indoor</td>
<td>4 scenes</td>
<td>Synthetic</td>
</tr>
<tr>
<td>S3-DIS</td>
<td>Indoor</td>
<td>6 areas</td>
<td>RGB-D</td>
</tr>
<tr>
<td>ModelNet40</td>
<td>CAD</td>
<td>40 models</td>
<td>Synthetic</td>
</tr>
<tr>
<td>ApolloScapes</td>
<td>Outdoor</td>
<td>80k point clouds</td>
<td>LiDAR</td>
</tr>
</tbody>
</table>
## Datasets

### 3D Registration

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Type</th>
<th>Size</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D Match</td>
<td>Indoor</td>
<td>62 scenes</td>
<td>RGB-D</td>
</tr>
<tr>
<td>KITTI</td>
<td>Outdoor</td>
<td>~70 sequences</td>
<td>LiDaR</td>
</tr>
<tr>
<td>ICL-NIUM</td>
<td>Indoor</td>
<td>4 scenes</td>
<td>Synthetic</td>
</tr>
<tr>
<td>S3-DIS</td>
<td>Indoor</td>
<td>6 areas</td>
<td>RGB-D</td>
</tr>
<tr>
<td>ModelNet40</td>
<td>CAD</td>
<td>40 models</td>
<td>Synthetic</td>
</tr>
<tr>
<td>ApolloScapes</td>
<td>Outdoor</td>
<td>80k point clouds</td>
<td>LiDAR</td>
</tr>
</tbody>
</table>
Datasets

3D Point Clouds

- Strong dependence on acquisition sensor!

Images from [Choy 2020b]
### Datasets

#### Other applications

<table>
<thead>
<tr>
<th>[Dang 2018]</th>
<th>Problem</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Absolute pose</td>
<td>Synthetic</td>
</tr>
<tr>
<td>[Kendall 2015]</td>
<td>PnP</td>
<td>SfM</td>
</tr>
<tr>
<td>[Ranftl 2018]</td>
<td>Homography</td>
<td>Synthetic</td>
</tr>
<tr>
<td>[Choy 2020a]</td>
<td>Linear subspaces</td>
<td>Synthetic</td>
</tr>
<tr>
<td>[Sun 2020]</td>
<td>Line</td>
<td>Synthetic</td>
</tr>
</tbody>
</table>

- For PnP transfer to real data demonstrated [Dang 2018]
- Acquiring data is both important and non-trivial!
Numerical Stability

- Training can be unstable
  - SVD non-differentiable if repeated singular values
  - Eigen-vector switching
  - Large contributions from gross outliers in the beginning
- You are likely to see NaNs

\[ f(X) = v_\theta \]

The gradient of \( g \) with respect to the input \( X \) is given by

\[
\frac{\partial g}{\partial X} = U \left( 2\Sigma \left( \mathbf{K}^T \circ \left( \mathbf{V}^\top \frac{\partial g}{\partial \mathbf{V}} \right)_{\text{sym}} \right) \right) \mathbf{V}^\top,
\]

where

\[
K_{ij} = \begin{cases} 
\sigma_i^{-1} \sigma_j, & \text{if } i \neq j \\
0, & \text{otherwise}
\end{cases}
\]

and \( \sigma_i \) denotes the \( i \)-th singular value.

Image from [Dang 2018]
Numerical Stability
Ways to mitigate

- Eigendecomposition-free training [Dang 2018]

\[ \mathcal{L}(W) = \tilde{e}X^TWX\tilde{e} + \alpha \exp(-\beta \text{tr}(\tilde{X}^TW\tilde{X})) \]

Weight from network, Groundtruth EV, Regularization

with \( f(X) = v_g \). The gradient of \( g \) with respect to the input \( X \) is given by

\[ \frac{\partial g}{\partial X} = U \{ 2\Sigma (K^T \circ (V^T \frac{\partial g}{\partial v_g})_{\text{sym}}) \} V^T, \]

where

\[ K_{ij} = \begin{cases} \frac{1}{\sigma_i^2 - \sigma_j^2}, & \text{if } i \neq j \\ 0, & \text{otherwise} \end{cases} \]

and \( \sigma_i \) denotes the \( i \)-th singular value.
Numerical Stability

Ways to mitigate

• Eigendecomposition-free training [Dang 2018]

\[ \mathcal{L}(W) = \hat{e}X^T WX\hat{e} + \alpha \exp(-\beta \text{tr}(\hat{X}^T W \hat{X})) \]

• Alternative:
  - Careful normalization
  - Clip loss function
  - Code safeguard

with \( f(X) = v_q \). The gradient of \( g \) with respect to the input \( X \) is given by

\[
\frac{\partial g}{\partial X} = U \left( 2E \left( K^T \circ \left( V^T \frac{\partial g}{\partial v_q} \right)_{\text{sym}} \right) \right) V^T,
\]

where

\[
K_{ij} = \begin{cases} \sigma_i^{-2} - 1, & \text{if } i \neq j \\ 0, & \text{otherwise} \end{cases}
\]

and \( \sigma_i \) denotes the \( i \)-th singular value.

Image from [Dang 2018]
Mini-Batching

- Mini-batching stabilizes and speeds up learning
- Different number of points per sample
- Makes effective batching hard
- Two possible solutions:
  1. Resample to a fixed size
  2. Retain sample indices and use them for aggregation ops
Part V: Postprocessing
Postprocessing
Everything is allowed at test time

- Simple integration
  - Remove samples with small weights
  - Send surviving points to *SAC
- Other approaches possible
  - Evaluate different percentiles of points
- Classic and learned are complementary

### Table from [Sun 2020]

<table>
<thead>
<tr>
<th>Method</th>
<th>Outdoor (%)</th>
<th>Indoor (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Known</td>
<td>Unknown</td>
</tr>
<tr>
<td>RANSAC*</td>
<td>4.82%</td>
<td>9.08%</td>
</tr>
<tr>
<td>PointCN[21]</td>
<td>34.35/13.93</td>
<td>47.98/23.55</td>
</tr>
<tr>
<td>PointNet++[27]</td>
<td>34.11/9.28</td>
<td>46.23/14.04</td>
</tr>
<tr>
<td>N*Net[26]</td>
<td>34.18/12.49</td>
<td>49.13/23.18</td>
</tr>
<tr>
<td>Ours</td>
<td>40.78/25.94</td>
<td>51.63/32.55</td>
</tr>
<tr>
<td>Ours++</td>
<td>42.46/33.06</td>
<td>52.18/39.33</td>
</tr>
</tbody>
</table>

### Table from [Zhang 2019]

<table>
<thead>
<tr>
<th>Method</th>
<th>Outdoor (%)</th>
<th>Indoor (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Known</td>
<td>Unknown</td>
</tr>
<tr>
<td>RANSAC*</td>
<td>15.1%</td>
<td>21.95%</td>
</tr>
<tr>
<td>PointCN[21]</td>
<td>30.45/13.83</td>
<td>43.18/24.83</td>
</tr>
<tr>
<td>Ours*</td>
<td>32.37/13.83</td>
<td>46.28/22.18</td>
</tr>
</tbody>
</table>
Conclusion

• A novel and exciting research field
• Novel type of data in the context of deep learning
  - Feature extractors are likely not optimal
• Can integrate geometric knowledge
  - Need to keep differentiability in mind
• Opportunity to improve with more and better data
• Some questions are (and will be hard to answer)
  - Break-down points
  - Dependence on dimensionality of data
Open questions

- Many avenues to make progress
  - Feature extractors
  - Tighter integration of non-differentiable parts
  - Complete end-to-end pipelines
  - Theoretical questions
  - Loss functions
  - Differentiable minimal solvers
  - Handling degeneracy
  - Datasets
  - New applications
Shall we only focus on deep learning?

Check out the next talk by Dmytro Mishkin!

There might be surprises :)
Questions?
References


