Blind Object Recognition with Soft Grippers

Irene Testa

Robotics Course, University of Pisa
A.Y. 2024/25

Background

Motivation

- **Goal:** Enable robots to recognize objects through physical interaction.
- **Challenge:** Vision can be unreliable in cases of occlusion, clutter, or poor lighting.
- **Alternative:** Tactile sensing enables *blind object recognition*.
- **Advantage of Soft Grippers:** Their compliance generates rich contact signals during interaction.

Related Work (Donato et al. [1])

- **Two-Stage LSTM Architecture:** 1) Predict coarse object properties (e.g., shape, size) from tactile time-series; 2) Use it as a prior to improve object classification.
- **Classical Models on Raw Signals:** Evaluated several standard classifiers (e.g., KNN, SVM, DT, RF, GB) directly on raw temporal data.
- **Multi-modal Fusion:** Combining proprioceptive and exteroceptive signals improves recognition over single-modality inputs.

My approach

- **Single-Stage Temporal Models:** The approach focuses on single-stage pipelines, extending the evaluation by employing models explicitly designed for *temporal data*, as well as traditional classifiers applied to extracted *hand-crafted features*.
- **Few-Shot Learning:** *Metric learning* techniques are explored to enable generalization to previously unseen objects using limited new data.

Problem Setup

Task

Multiclass object classification using *tactile* and *proprioceptive* time-series data from a real-world soft robotic gripper.

Data Overview

- Multivariate time series
- 768 samples total
- 150 timesteps per sample (sampling interval: 0.05 s)
- 4 features per timestep:
  - Left and right force sensing resistors (exteroceptive)
  - Left and right curvature sensors (proprioceptive)
- 17 object classes; generally balanced except the underrepresented `empty` class

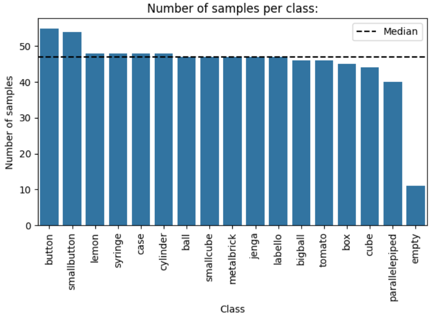

Figure 1: Class distribution.

Table 1: Sensor Data Statistics.

Sensor	Mean	Std Dev	Min	Max
Force left	4738	9630	3.18	24465.15
Force right	5569.7	10233.5	3.18	24465.15
Curvature left	17.91	12.13	-2.63	47.09
Curvature right	13.19	9.20	-3.71	32.50

Data Preprocessing

Data Cleaning and Balancing

- Removed 17 samples due to *non-monotonic time values*, indicating possible data collection errors.
- Applied *random oversampling* of the `empty` class to match the median number of samples per class.

Data Splitting and Scaling

- Two experimental scenarios considered:
  - **Closed set**: all 17 classes used during training.
  - **Few-shot**: 5 novel classes held out from training.
- For both scenarios, the data was split into training, validation, and test sets (80/10/10%) with balanced class distributions.
- Features scaled to the range [0,1] after flattening the time dimension; scaling parameters computed solely on the respective training sets to prevent data leakage.

Few-Shot Learning Setup

Terminology

- **Base dataset:** Dataset containing classes seen during training.
- **Novel classes:** Classes not used during base training, appearing only in few-shot evaluation.
- **Support set:** Small labeled subset of novel class samples used for adaptation.
- **Query set:** Remaining samples of novel classes used for evaluation.

Few-Shot Data Partitioning

- Five novel classes held out: `parallelepiped`, `smallbutton`, `smallcube`, `syringe`, `tomato`.
- Base dataset comprises the remaining 12 classes.
- For each novel class:
  - Created 5-shot and 10-shot support sets.
  - Query set contains all other samples from those classes.

Classifier Overview

Traditional Models (Closed Set)

- Gradient Boosting (GB) trained on:
  - Statistical features extracted from time series
  - Shapelet-based features
- K-Nearest Neighbors (KNN) with Dynamic Time Warping (DTW) distance

Neural Classifiers (Closed Set)

- Convolutional Neural Network (CNN)
- Long Short-Term Memory Network (LSTM)
- Transformer-based model

Neural Metric Learning Models (Closed Set and Few-Shot)

- Siamese Network (SN)
- Deep Attentive Time Warping (DATW)

Gradient Boosting Classifier

Feature Engineering

Statistical Features (50 total):

- **Time domain:** mean, std, median, percentiles (10, 25, 50, 75, 90), covariance, skewness, kurtosis.
- **Frequency domain:** spectral energy and spectral entropy (computed via Discrete Fourier Transform).
- Features with zero standard deviation were removed.

Shapelet Features (150 total):

- **Shapelets**: subsequences that are highly representative of a specific class.
- Each sample is mapped to a vector of distances to discovered shapelets.
- Extracted using `sktime`'s `ShapeletTransformClassifier`.

Classification

- **Gradient Boosting**: sequential tree ensemble correcting prior errors; effective for tabular, high-dimensional data.
- Used the *optimized and scalable* implementation from `xgboost` library.
- Hyperparameters tuned via *grid search* (selected by validation accuracy).

Table 2: GB Hyperparameters. Blue: best for statistical; Magenta: best for shapelets; **Bold**: shared.

Parameter	Values
`n_estimators`	100, 200
`max_depth`	**3**, 5
`learning_rate`	0.05, **0.1**
`subsample`	0.8, 1.0

K-Nearest Neighbors with Dynamic Time Warping

Distance computation

- **Dynamic Time Warping** measures similarity between time series that may differ in speed, timing, or length.
- Minimizes alignment cost by *warping the time axis* using dynamic programming.
- **Time complexity**: $\mathcal{O}(n^2)$ (vs. $\mathcal{O}(n)$ for Euclidean distance).

Classification

- Used `tslearn` library.
- *Grid search* over hyparameters; best configuration selected by validation accuracy.

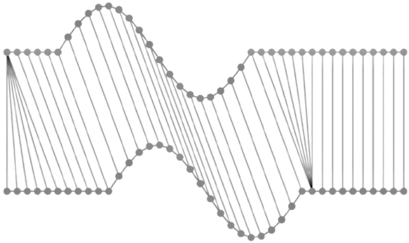

Figure 2: Example of time series alignment via DTW.

Table 3: KNN Hyperparameters. Best values in **bold**.

Parameter	Values
`k`	$\mathbf{1}, 3, 5, 9, 15$
`weights`	**uniform**, distance

Neural Classifiers

Neural Classifiers Overview

Evaluated three neural architectures, each designed to capture distinct data characteristics.

- **CNN**: *local pattern extraction*, baseline for time series.
- **LSTM**: *temporal memory*, captures *sequential dependencies*.
- **Transformer**: *self-attention*, models *global context*.

Training Setup

- Implemented in `Keras`.
- Used *categorical cross-entropy* as loss function.
- Trained for 500 epochs, batch size 16; *early stopping* on validation loss (patience=20, restore best weights).
- Learning rate *reduced* on plateau (patience=10).
- Hyperparameters tuned via *grid search*; best configuration selected by validation accuracy.

Convolutional Neural Network

Overview

- **Convolutional Neural Network**: a deep learning model that applies shared learnable filters to extract *local patterns* in temporal data.
- **Convolutional filters**: detect *short-term temporal dependencies* by sliding over input sequences.

Architecture

- Based on [2], a baseline for time series classification with deep neural networks
- Stack of convolutional blocks:
  - Conv1D (kernel size 3)
  - Batch normalization
  - ReLU activation
  - Dropout (optional)
- Global average pooling
- Dense layer with softmax activation

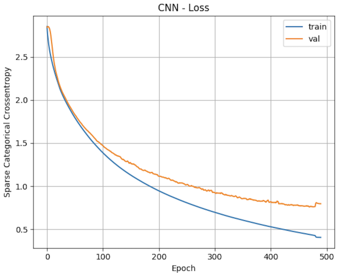

Figure 3: CNN training and validation loss over epochs.

Table 4: CNN Hyperparameters. Best values in **bold**.

Parameter	Values
num_filters	**32**, 64
num_conv_layers	2, 3, **4**
dropout_rate	**0.0**, 0.3
learning_rate	**1e-4**, 1e-3

Long Short-Term Memory Network

Overview

- **Long Short-Term Memory**: are a type of *Recurrent Neural Network* designed to learn from sequences by capturing both short- and long-term dependencies through *gated memory cells*.
- **Gated memory cells**: control what information is stored, forgotten, or passed on.

Architecture

- Stack of LSTM layers
- Optional dropout applied on inputs and recurrent connections
- Final dense layer with softmax activation for classification

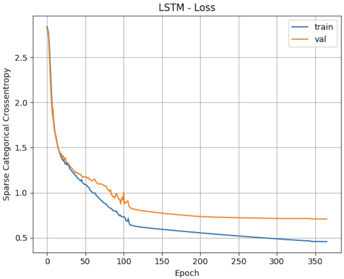

Figure 4: LSTM training and validation loss over epochs.

Table 5: LSTM Hyperparameters. Best values in **bold**.

Parameter	Values
`units`	32, **64**
`num_layers`	1, **2**
`dropout_rate`	**0.0**, 0.3
`learning_rate`	**1e-4**, 1e-3

Transformer Network

Overview

- **Transformer**: a deep learning model that uses *self-attention* to capture both local and global dependencies in sequences.
- **Self-attention mechanism**: each time step's feature vector is updated by a learned weighted average of all time steps' vectors.

Architecture

- A Transformer encoder block, including:
  - *Multi-head self-attention* with residual connection, dropout, and layer normalization
  - *Feedforward network*: two Conv1D layers (kernel size 1) with ReLU and dropout + residual connection and layer normalization
- Global average pooling, followed by a dense MLP and softmax output.

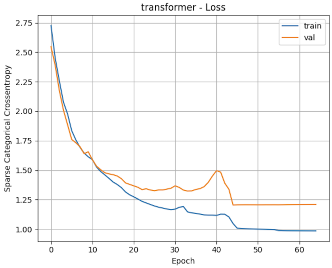

Figure 5: Transformer training and validation loss over epochs.

Table 6: Transformer Hyperparameters. Best values in **bold**.

Parameter	Values
head_size	64
num_heads	2
num_filters	64
mlp_units	64
dropout_rate	**0.0**, 0.3
learning_rate	1e-4, **1e-3**

Metric Learning Approaches

Overview

- **Metric learning**: trains a model to map samples into an embedding space where distance reflects similarity.
- **Pairwise contrastive learning**: learns embeddings by processing pairs of samples, minimizing the distance between embeddings if they share the same label, and maximizing it otherwise.
- *Contrastive loss* with **margin** τ (minimum distance between negative pairs).
- Classification via **KNN** on learned embeddings.
- For *few-shot learning*, compute embeddings for new samples without updating model weights, then apply KNN to classify them.

Implementation Notes

- Implemented in `PyTorch`.
- Grid search was not performed due to the large pairwise dataset and model size required for contrastive learning.
- Best model weights (based on validation accuracy via KNN) were restored after training.

Table 7: Contrastive learning parameters.

Parameter	Values
`learning_rate`	1e-4
`batch_size`	64
`num_epochs`	20
`iterations_per_epoch`	500
`positive_ratio`	1/3
`tau`	1
`k`	3

Siamese Network

Overview

- **Siamese network**: twin networks sharing weights, designed to learn embeddings for pairs of inputs.

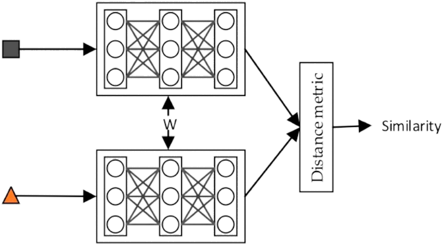

Figure 6: Illustration of a Siamese Network.

Architecture and Training Details

- Architecture:
  - Two convolutional blocks, each with two Conv1D layers (kernel size 3), batch normalization, ReLU, and dropout
  - Max pooling (stride 2) after the first block; final global average pooling
  - Fully connected layer projecting to a 128-dimensional embedding
- Embeddings are ℓ_2-normalized before computing the contrastive loss to prevent trivial minimization via unbounded growth of embedding norms.

Deep Attentive Time Warping

Overview

- Learns an adaptive, differentiable alignment between time series A and B.
- Inputs are reshaped and combined via **outer concatenation**.
- A **U-Net** outputs a real-valued matrix P, followed by row-wise softmax to produce a soft alignment.
- This process parallels an **attention mechanism**, where P acts as an attention weight matrix.
- Warped series are obtained by applying P and compared to compute distance.
- Trained end-to-end with **contrastive loss**.
- U-Net weights **initialized** by mimicking DTW.

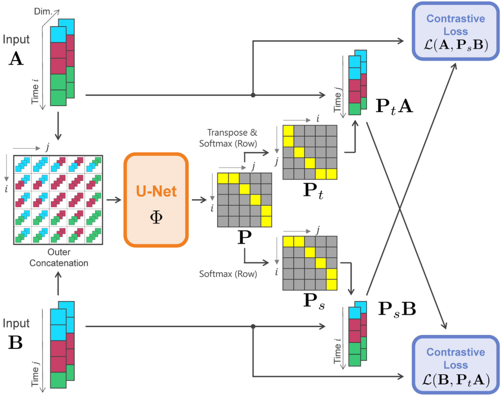

Figure 7: Illustration of the DATW method.

Implementation details

- **U-Net**: encoder-decoder CNN with skip connections (10 Conv2D + BatchNorm, ReLU, max pooling).
- **Pre-training**: 10 epochs, 100 iterations each.
- Ran on **GPU** due to computational cost.

Closed-Set Classification Results

Classification Performance Overview

- Tab. 8 shows *accuracy* and *macro F1*, which averages performance equally across classes to mitigate imbalance; their similarity indicates **balanced class performance**.
- **DATW** achieves the best results, followed closely by **DTW**.
- **stats+GB** outperforms several more complex models.
- **CNNs** and **transformers** perform worse, likely due to *data limitations*.

Table 8: Closed-set classification scores.

	Accuracy	Macro F1
DATW	0.814815	0.813844
DTW	0.802469	0.801356
stats+GB	0.790123	0.793791
SN	0.753086	0.748969
LSTM	0.753086	0.746446
CNN	0.679012	0.680768
shapelet+GB	0.679012	0.669873
transformer	0.641975	0.611932

Confusion Matrix Highlights (e.g. Fig. 8)

- Some classes (`empty`, `metalbrick`, `box`) are **perfectly classified** by all models.
- Others (e.g.,`ball`, `cylinder`, `lemon`) show consistently **high misclassification**, indicating *inherent difficulty*.
- A few instances are **misclassified 100%**, suggesting *ambiguous samples* (not obvious from visual inspection).
- Most samples are classified reliably, indicating **consistent data quality**.

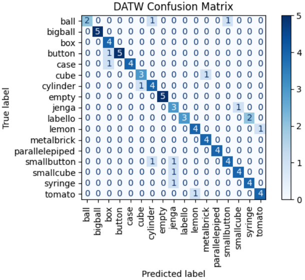

Figure 8: DATW confusion matrix.

Few-Shot Learning Results

Performance summary (Tab. 9)

- Both models learn a **discriminative metric**, successfully classifying novel classes ("New Ref") but sometimes confusing them with seen classes.
- **DATW** consistently outperforms SN across all few-shot settings.
- Using a **subset** of training data + support ("Sub Ref") as KNN reference outperforms using all training data + support ("Full Ref"), likely reducing bias to seen classes.
- Performance improves from 5-shot to 10-shot learning.

Embedding Analysis (Fig. 9)

- t-SNE visualization of Siamese Network embeddings shows that novel classes form **elongated clusters**.
- Some clusters partially **overlap** with seen classes.

Table 9: Few-shot learning classification results.

	Accuracy	F1 Macro
DATW 10-shot New Ref	0.881081	0.884006
DATW 5-shot New Ref	0.861905	0.863981
SN 10-shot New Ref	0.837838	0.836266
SN 5-shot New Ref	0.790476	0.791284
DATW 10-shot Sub Ref	0.664865	0.299011
DATW 5-shot Sub Ref	0.552381	0.221240
SN 10-shot Sub Ref	0.481081	0.207921
SN 10-shot Full Ref	0.427027	0.190487
DATW 10-shot Full Ref	0.464865	0.189817
SN 5-shot Sub Ref	0.366667	0.155858
SN 5-shot Full Ref	0.290476	0.126410
DATW 5-shot Full Ref	0.261905	0.115253

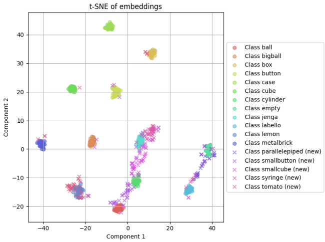

Figure 9: Siamese Network embeddings.

Conclusion and Future Work

Conclusion

- **Challenges**: Limited data for training and testing; *regularization* (e.g., dropout) used to reduce overfitting.
- **Key Findings**: DATW achieved the best results, slightly outperforming DTW.
- **Limitations**: DATW has *high computational cost* due to outer concatenation and use of U-Net, plus slow test-time inference requiring a forward pass through the entire reference set for each query—*limiting real-time suitability*.

Future Work

- Apply **data augmentation** to expand the training set.
- For metric learning approaches:
  - **Fine-tune** the metric model on new class pairs with few steps and low learning rate.
  - Select an **optimal reference subset** for KNN or use **class prototypes** instead.
  - Improve training with **triplet loss** and **hard mining**.

`GitHub` repository:
https://github.com/iretes/blind-object-recognition-soft-grippers

References

[1] E. Donato, D. Pelliccia, M. Hosseinzadeh, M. Amiri, and E. Falotico, "Tactile object recognition with recurrent neural networks through a perceptive soft gripper," *IEEE Robotics and Automation Letters*, 2025.

[2] Z. Wang, W. Yan, and T. Oates, "Time series classification from scratch with deep neural networks: A strong baseline," in *2017 International joint conference on neural networks (IJCNN)*, IEEE, 2017, pp. 1578–1585.

[3] S. Matsuo *et al.*, "Attention to warp: Deep metric learning for multivariate time series," in *Document Analysis and Recognition–ICDAR 2021: 16th International Conference, Lausanne, Switzerland, September 5–10, 2021, Proceedings, Part III 16*, Springer, 2021, pp. 350–365.

[4] M. Kaya and H. Ş. Bilge, "Deep metric learning: A survey," *Symmetry*, vol. 11, no. 9, p. 1066, 2019.

Thank you!