ProtoNN: Compressed and Accurate kNN for Resource-scarce Devices

Chirag Gupta[1], Arun Sai Suggala[12], Ankit Goyal[13], Harsha Vardhan Simhadri[1],
Bhargavi Paranjape[1], Ashish Kumar[1], Saurabh Goyal[4], Raghavendra Udupa[1], Manik Varma[1], Prateek Jain[1]
[1]Microsoft Research, India, [2]Carnegie Mellon University, [3]University of Michigan, Ann Arbor, [4]IIT Delhi, India

ICML 2017
The 34th International Conference on Machine Learning
Sydney, Australia
August 6th - 11th, 2017

Abstract

- **Problem:** Can we perform machine learning *locally* on tiny devices with puny storage (<2kB) and computational capacity?

Case in point: **Internet-of-Things (IoT)**
  - For example, an **Arduino uno** with **2kB RAM, 32kB flash**.
  - State of the art: IoT devices just collect data and send it to the cloud for prediction.
  - Factors to consider: **battery, latency, privacy**

- **ProtoNN** improves **k-Nearest Neighbours** on critical metrics:

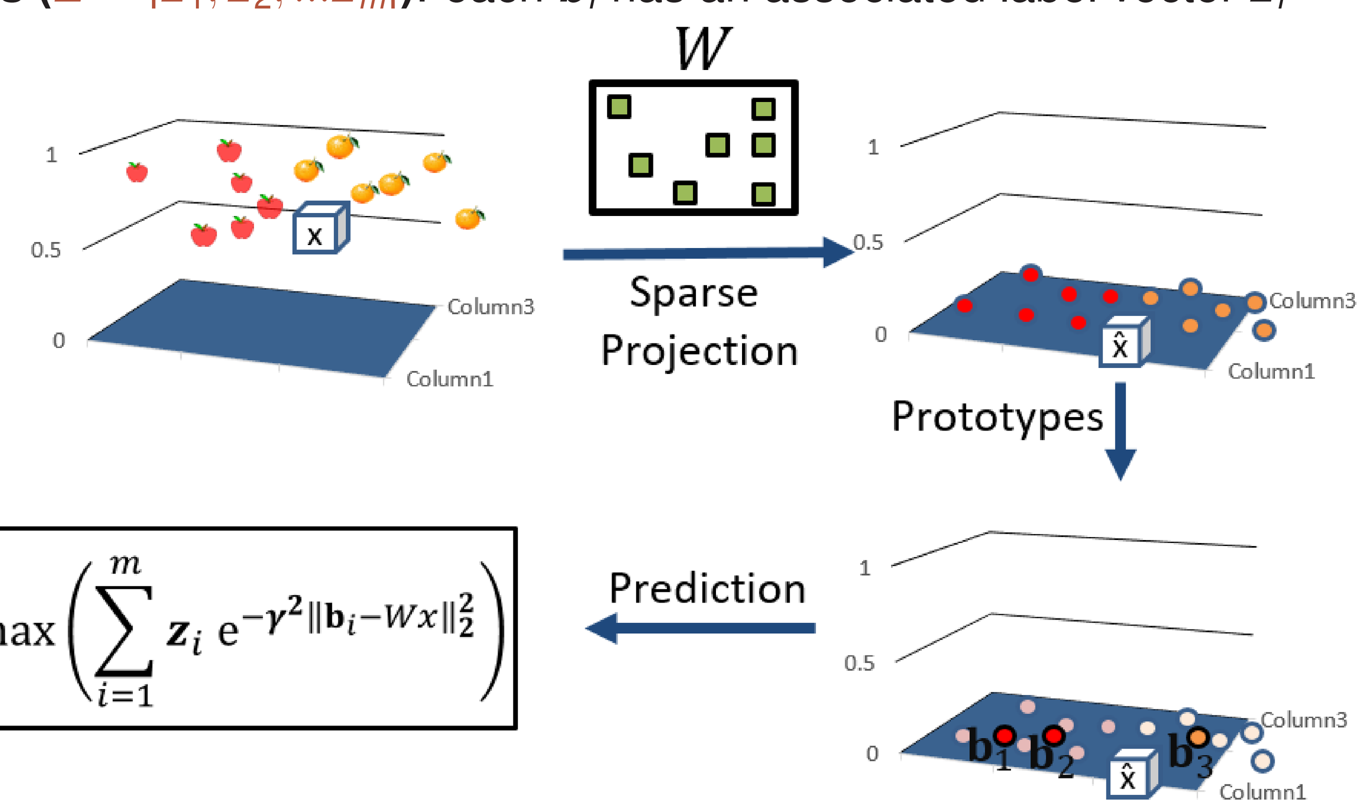
Metric	k-Nearest Neighbours	ProtoNN (Prototype-based Nearest Neighbours)
Model size	Size of the entire training data	Often smaller than the size of a single training point
Prediction time	Few seconds	Few milliseconds
Distance metric	Must be specified a-priori	Learnt as part of training

- ProtoNN stays within 2% of the best accuracy on most datasets!
- ProtoNN seamlessly generalizes to **multi-label**, **multi-class**, **ranking** problems. Our implementation also scales to **large datasets**.

ProtoNN model

Smooth k-NN prediction	ProtoNN prediction
$\hat{\mathbf{y}} = \rho\left(\sum_{i=1}^{n} \sigma(\mathbf{y}_i) K(\mathbf{x}, \mathbf{x}_i)\right)$	$\hat{\mathbf{y}} = \rho\left(\sum_{j=1}^{m} \mathbf{z}_j K(W\mathbf{x}, \mathbf{b}_j)\right)$

- **Sparse low dimensional projection** ($W \in \mathbb{R}^{d \times D}$):
  - reduces the space we work in
  - gives us a handle over the metric to use for the data
- **Prototypes** ($B = [\mathbf{b}_1, \mathbf{b}_2, \ldots \mathbf{b}_m]$):
  - each $\mathbf{b}_i$ is in the low dimensional space
  - B is a representative sample of training points in projected space
  - **Labels** ($Z = [\mathbf{z}_1, \mathbf{z}_2, \ldots \mathbf{z}_m]$): each $\mathbf{b}_i$ has an associated label vector $\mathbf{z}_i$



- K is the Gaussian kernel: $K(x, y) = \exp\{-\gamma^2 \|x - y\|_2^2\}$
- ρ is a selector function: the highest ranked label for binary and multi-class problems. Can be easily extended to ranking or multi-label problems.
- **Joint optimization**:
  - we learn B, Z, and W jointly
  - explicit sparsity constraints are imposed during the optimization itself

Learning algorithm

We minimize squared ℓ_2 loss, with explicit sparsity constraints:

$$\mathcal{R}_{emp}(Z, B, W) = \frac{1}{n}\left(\sum_{i=1}^{n} \left\| \mathbf{y}_i - \sum_{j=1}^{m} \mathbf{z}_j K(W\mathbf{x}_i, \mathbf{b}_j) \right\|_2^2\right)$$

$$\text{Minimize } \mathcal{R}_{emp}(Z, B, W),$$
$$s.t. \|Z\|_0 \le s_Z, \|B\|_0 \le s_B, \|W\|_0 \le s_W$$

- **Alternating minimization** over the 3 parameters:
  - We take e epochs each over Z, B, W one by one
  - We perform this entire process for T iterations.
- In each epoch, we do **mini-batch stochastic gradient descent**:
  - Nesterov's accelerated SGD with tail-averaging
- **Hard-Thresholding** after each SGD step to satisfy sparsity constraints
- Step size: $\eta_t = \frac{\eta}{\sqrt{t}}$. η is selected using the **Armijo** rule

Analysis

- A simple generative model for the data:
  - Mixture of two well-separated spherical Gaussians with centers μ_+, μ_-
  - Points from the μ_+, μ_- Gaussians are positive, negative respectively
- Informally, we show that with constant probability, the 2 prototypes of ProtoNN converge to the centers of these Gaussians at a geometric rate.

Theorem (simplified)

- *Set $W = I$, $Z = [\mathbf{e}_1, \ \mathbf{e}_2]$ and let $\mathbf{b}_+$, $\mathbf{b}_-$ be the prototypes.*
- *Define $\bar{\mu} := \mu_+ - \mu_-$, $\mathcal{R} = \mathbb{E}[\mathcal{R}_{emp}]$.*
- *Suppose:*
  - $(\mathbf{b}_+ - \mu_+)^T \bar{\mu} \ge -\frac{\|\bar{\mu}\|^2}{4}$
  - $d \ge 4\|\bar{\mu}\|^2$
  - $\|\mathbf{b}_+ - \mu_+\| \ge 8\|\bar{\mu}\| \exp\left\{-\frac{\|\bar{\mu}\|^2}{4}\right\}$
- *Consider the update $\mathbf{b}'_+ = \mathbf{b}_+ - \eta \nabla_{\mathbf{b}_+} \mathcal{R}$, with appropriate $\eta \ge 0$*

Then with constant probability:

$$\|\mathbf{b}'_+ - \mu_+\|^2 \le \|\mathbf{b}_+ - \mu_+\|^2 \left(1 - 0.01 \exp\left\{-\frac{\|\bar{\mu}\|^2}{4}\right\}\right)$$

Comparison with compressed baselines

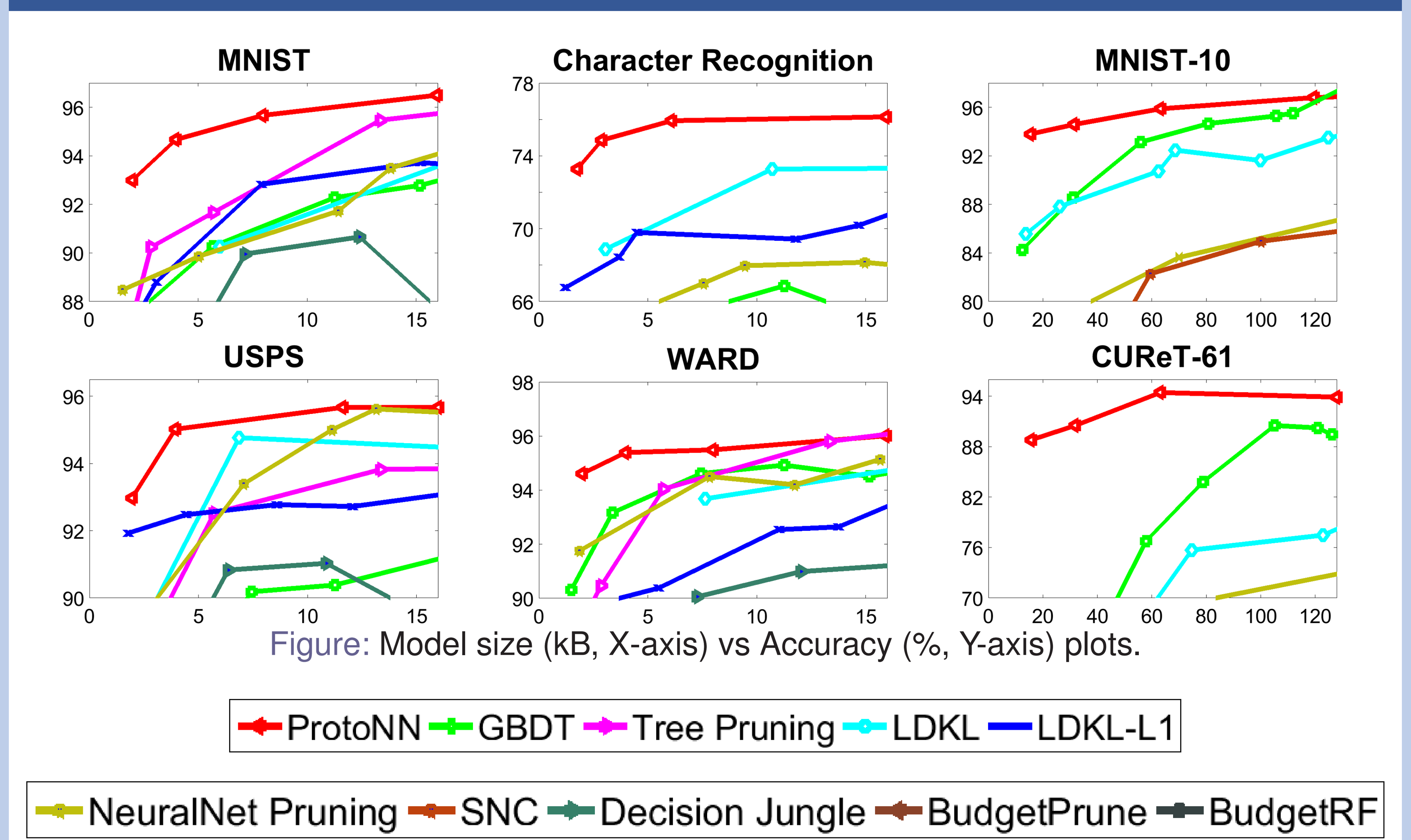


Figure: Model size (kB, X-axis) vs Accuracy (%, Y-axis) plots.

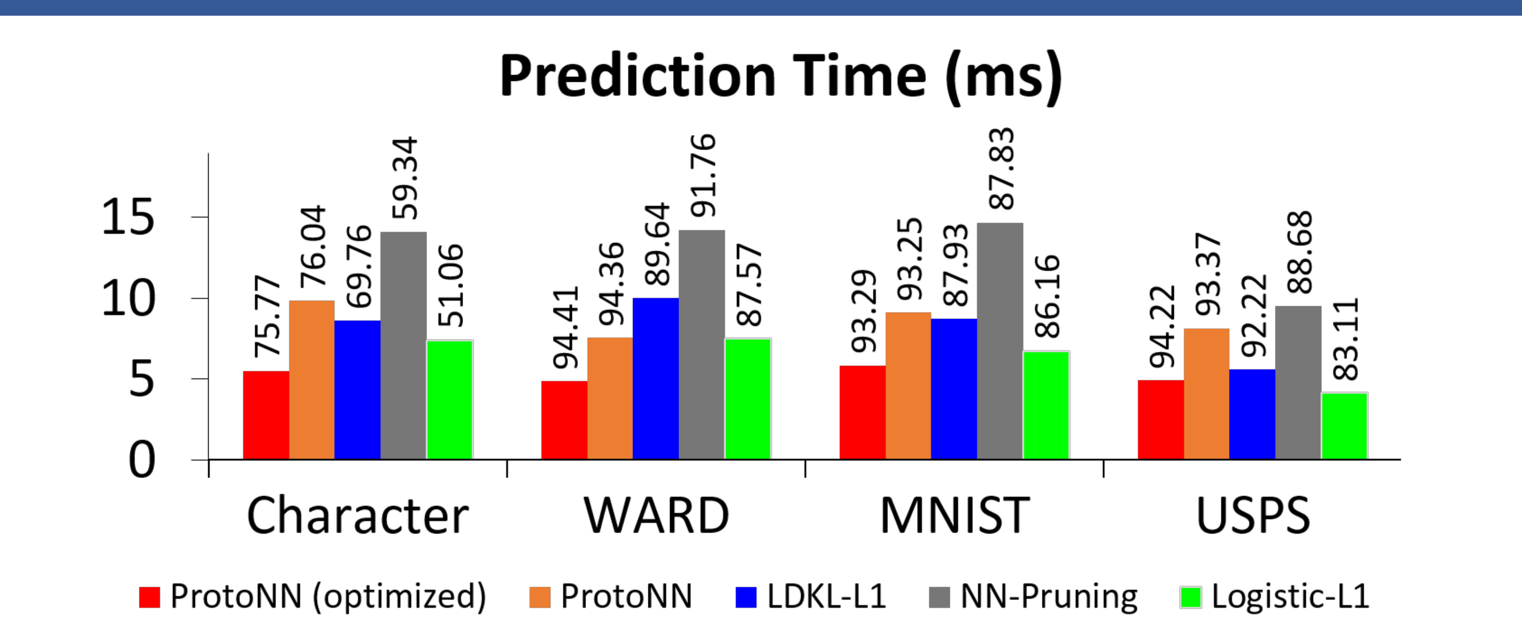
Comparison with uncompressed baselines

Dataset		ProtoNN	kNN	SNC	BNC	GBDT	1-hidden NN	RBF-SVM
character recognition	model size (kB)	15.94	6870.3	441.2	70.88	625	314.06	6061.71
	accuracy	**76.14**	67.28	74.87	70.68	72.38	72.53	75.6
eye	model size (kB)	10.32	14592	3305	1311.4	234.37	6401.56	7937.45
	accuracy	90.82	76.02	87.76	80.61	83.16	90.31	**93.88**
mnist	model size (kB)	15.96	183750	4153.6	221.35	1171.87	3070	35159.4
	accuracy	96.5	96.9	95.74	98.16	**98.36**	98.33	98.08
usps	model size (kB)	11.625	7291	568.8	52.49	234.37	504	1659.9
	accuracy	95.67	96.7	**97.16**	95.47	95.91	95.86	96.86
ward	model size (kB)	15.94	17589.8	688	167.04	1171.87	3914.06	7221.75
	accuracy	96.01	94.98	96.01	93.84	**97.77**	92.75	96.42
cifar	model size (kB)	15.94	78125	3360	144.06	1562.5	314.06	63934.2
	accuracy	76.35	73.7	76.96	73.74	77.19	75.9	**81.68**

Dataset		ProtoNN (64kB)	kNN	SNC	BNC	GBDT	1-hidden NN	RBF SVM
letter-26	model size (kB)	63.4	1237.8	145.08	31.95	20312	164.06	568.14
	accuracy	97.10	95.26	96.36	92.5	97.16	96.38	**97.64**
mnist-10	model size (kB)	63.4	183984.4	4172	220.46	5859.37	4652.34	39083.7
	accuracy	95.88	94.34	93.6	96.68	97.9	**98.44**	97.3
usps-10	model size (kB)	63.83	7291.4	568.8	51.87	390.62	519.53	1559.6
	accuracy	94.92	94.07	94.77	91.23	94.32	94.32	**95.4**
curet-61	model size (kB)	63.14	10037.5	513.3	146.70	2382.81	1310	8940.8
	accuracy	94.44	89.81	95.87	91.87	90.81	95.51	**97.43**

Dataset		FastXML	DiSMEC	SLEEC	ProtoNN
mediamill	model size	7.64M	**48.48K**	57.95M	54.8K
$n = 30993$	P@1	83.65	**87.25**	86.12	85.19
$d = 120$	P@3	66.92	69.3	**70.31**	69.01
$L = 101$	P@5	52.51	54.19	**56.33**	54.39
delicious	model size	36.87M	1.97M	7.34M	**925.04K**
$n = 12920$	P@1	**69.41**	66.14	67.77	68.92
$d = 500$	P@3	**64.2**	61.26	61.27	63.04
$L = 983$	P@5	**59.83**	56.30	56.62	58.32
eurlex	model size	410.8M	79.86M	61.74M	**5.03M**
$n = 15539$	P@1	71.36	**82.40**	79.34	77.74
$d = 5000$	P@3	59.85	**68.50**	64.25	65.01
$L = 3993$	P@5	50.51	**57.70**	52.29	53.98

Actual implementation on an Arduino Uno



Energy (mJ) = (0.2455 J/s) * Prediction time (ms)

Library for machine learning on the Edge (EdgeML)

- ProtoNN and Bonsai [Kumar et al., ICML '17] will be made publicly available as part of a general machine learning library for tiny devices.
- https://aka.ms/EdgeML
- We are also working on more algorithms like anomaly detection.

Mail: asuggala@andrew.cmu.edu, prajain@microsoft.com