ProtoNN: Compressed and Accurate kNN for Resource-scarce Devices

Chirag Gupta [1] **Arun Sai Suggala** [1 2] **Ankit Goyal** [1 3] **Harsha Vardhan Simhadri** [1]
Bhargavi Paranjape [1] **Ashish Kumar** [1] **Saurabh Goyal** [4] **Raghavendra Udupa** [1] **Manik Varma** [1]
Prateek Jain [1]

Abstract

Several real-world applications require real-time prediction on resource-scarce devices such as an Internet of Things (IoT) sensor. Such applications demand prediction models with small storage and computational complexity that do not compromise significantly on accuracy. In this work, we propose ProtoNN, a novel algorithm that addresses the problem of real-time and accurate prediction on resource-scarce devices. ProtoNN is inspired by k-Nearest Neighbor (KNN) but has several orders lower storage and prediction complexity. ProtoNN models can be deployed even on devices with puny storage and computational power (e.g. an Arduino UNO with 2kB RAM) to get excellent prediction accuracy. ProtoNN derives its strength from three key ideas: a) learning a small number of prototypes to represent the entire training set, b) sparse low dimensional projection of data, c) joint discriminative learning of the projection and prototypes with explicit model size constraint. We conduct systematic empirical evaluation of ProtoNN on a variety of supervised learning tasks (binary, multi-class, multi-label classification) and show that it gives nearly state-of-the-art prediction accuracy on resource-scarce devices while consuming several orders lower storage, and using minimal working memory.

1. Introduction

Real-time and accurate prediction on resource-constrained devices is critical for several Machine Learning (ML) domains. Internet-of-things (IoT) is one such rapidly growing domain. IoT devices have the potential to provide real-time, local, sensor-based solutions for a variety of areas like housing, factories, farming, even everyday utilities like toothbrushes and spoons. The ability to use machine learning on data collected from IoT sensors opens up a myriad of possibilities. For example, *smart* factories measure temperature, noise and various other parameters of their machines. ML based anomaly detection models can then be applied on this sensor data to preemptively schedule maintenance of a machine and avoid failure.

However, machine learning in IoT scenarios is so far limited to cloud-based predictions where large deep learning models are deployed to provide accurate predictions. The sensors/embedded devices have limited compute/storage abilities and are tasked only with sensing and transmitting data to the cloud. Such a solution does not take into account several practical concerns like privacy, bandwidth, latency and battery issues. For example, consider the energy costs of communication if each IoT device on each machine in a smart factory has to continuously send data and receive predictions from the cloud.

Consider a typical IoT device that has $\leq$ 32kB RAM and a 16MHz processor. Most existing ML models cannot be deployed on such tiny devices. Recently, several methods (Han et al., 2016; Nan et al., 2015; Kusner et al., 2014) have been proposed to produce models that are compressed compared to large DNN/kernel-SVM/decision-tree based classifiers. However, none of these methods work well at the scale of IoT devices. Moreover, they do not offer natural extensions to supervised learning problems other than the ones they were initially designed for.

In this paper, we propose a novel kNN based algorithm (ProtoNN) that can be deployed on the tiniest of devices, can handle general supervised learning problems, and can produce state-of-the-art accuracies with just $\approx$16kB of model size on many benchmark datasets. A key reason for selecting kNN as the algorithm of choice is due to its generality, ease of implementation on tiny devices, and small number of parameters to avoid overfitting. However, kNN suffers from three issues which limit its applicability in

[1]Microsoft Research, India [2]Carnegie Mellon University, Pittsburgh [3]University of Michigan, Ann Arbor [4]IIT Delhi, India. Correspondence to: Arun Sai Suggala <asuggala@andrew.cmu.edu>, Prateek Jain <prajain@microsoft.com>.

Proceedings of the 34^{th} International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017.

practice, especially in the small devices setting: a) *Poor accuracy*: kNN is an ill-specified algorithm as it is not a priori clear which distance metric one should use to compare a given set of points. Standard metrics like Euclidean distance, ℓ_1 distance etc. are not task-specific and lead to poor accuracies. b) *Model size*: kNN requires the entire training data for prediction, so its model size is too large for the IoT setting. c) *Prediction time*: kNN requires computing the distance of a given test point w.r.t. each training point, making it prohibitive for prediction in real-time.

Several methods have been proposed to address some of these concerns. For example, metric learning (Weinberger & Saul, 2009) learns a task-specific metric that provides better accuracies but ends up increasing model-size and prediction time. KD-trees (Bentley, 1975) can decrease the prediction time, but they increase the model size and lead to loss in accuracy. Finally, recent methods like Stochastic Neighborhood Compression (SNC) (Kusner et al., 2014) can decrease model size and prediction time by learning a small number of prototypes to represent the entire training dataset. However, as our experiments show, their predictions are relatively inaccurate, especially in the tiny model-size regime. Moreover, their formulations limit applicability to binary and multi-class classification problems (see Section 2 for a detailed comparison to SNC).

In contrast, ProtoNN is able to address the above-mentioned concerns by using three key ideas:

a) *Sparse low-d projection*: we project the entire data in low-d using a sparse projection matrix that is jointly *learned* to provide good accuracy in the projected space.

b) *Prototypes*: we learn prototypes to represent the entire training dataset. Moreover, we learn labels for each prototype to further boost accuracy. This provides additional *flexibility*, and allows us to seamlessly generalize ProtoNN for multi-label or ranking problems.

c) *Joint optimization*: we learn the projection matrix jointly with the prototypes and their labels. Explicit sparsity constraints are imposed on our parameters during the optimization itself so that we can obtain an optimal model within the given model size de-facto, instead of post-facto pruning to force the model to fit in memory.

Unfortunately, our optimization problem is non-convex with hard ℓ_0 constraints. Yet, we show that simple stochastic gradient descent (SGD) with iterative hard-thresholding (IHT) works well for optimization. ProtoNN can be implemented efficiently, can handle datasets with millions of points, and obtains state-of-the-art accuracies.

We analyze ProtoNN in a simple binary classification setting where the data is sampled from a mixture of two well-separated Gaussians, each Gaussian representing one class. We show that if we fix the projection matrix and prototype labels, the prototypes themselves can be learned optimally in polynomial time with at least a constant probability. Moreover, assuming a strong initialization condition we observe that our SGD+IHT method when supplied a small number of samples, proportional to the sparsity of means, converges to the global optima. Although the data model is simple, it nicely captures the main idea behind our problem formulation. Further, our analysis is the first such analysis for any method in this regime that tries to learn a compressed non-linear model for binary classification.

Finally, we conduct extensive experiments to benchmark ProtoNN against existing state-of-the-art methods for various learning tasks. First, we show that on several binary (multi-class) problems, ProtoNN with a 2kB (16kB) memory budget significantly outperforms all the existing methods in this regime. Moreover, in the binary classification case, we show that ProtoNN with just $\approx$ 16kB of model-size, provides nearly the same accuracy as most popular methods like GBDT, RBF-SVM, 1-hidden layer NN, etc, which might require up to 1GB of RAM on the same datasets. Similarly, on multilabel datasets, ProtoNN can give $100\times$ compression with $\leq 1\%$ loss in accuracy. Finally, we demonstrate that ProtoNN can be deployed on a tiny Arduino Uno device[1] and leads to better accuracies than existing methods while incurring significantly lesser energy and prediction time costs. We have implemented ProtoNN as part of an open source embedded device ML library and it can be downloaded online[2].

2. Related Works

kNN is a popular ML algorithm owing to its simplicity, generality, and interpretability (Cover & Hart, 2006). In particular, kNN can learn complex decision boundaries and has only one hyperparameter k. However, vanilla kNN suffers from several issues as mentioned in the previous section. A number of methods, which try to address these issues, exist in the literature. Broadly, these methods can be divided into three sub-categories.

Several existing methods reduce prediction time of kNN using fast nearest neighbor retrieval. For example Bentley (1975); Beygelzimer et al. (2006) use tree data structures and Gionis et al. (1999); Weiss et al. (2008); Kulis & Darrell (2009); Norouzi et al. (2012); Liu et al. (2012) learn binary embeddings for fast nearest neighbor operations. These methods, although helpful in reducing the prediction time, lead to loss in accuracy and require the entire training data to be in memory leading to large model sizes that cannot be deployed on tiny IoT devices.

[1] https://www.arduino.cc/en/Main/ArduinoBoardUno

[2] https://github.com/Microsoft/ELL

Another class of methods improve accuracy of kNN by learning a *better* metric to compare, given a pair of points (Goldberger et al., 2004; Davis et al., 2007). For example, (Weinberger & Saul, 2009) proposed a Large Margin Nearest Neighbor (LMNN) classifier which transforms the input space such that in the transformed space points from same class are closer compared to points from disparate classes. LMNN's transformation matrix can map data into lower dimensions and reduce overall model size compared to kNN, but it is still too large for most resource-scarce devices.

Finally, another class of methods constructs a set of prototypes to represent the entire training data. In some approaches (Angiulli, 2005; Devi & Murty, 2002), the prototypes are chosen from the original training data, while some other approaches (Mollineda et al., 2002) construct artificial points for prototypes. Of these approaches, SNC, Deep SNC (DSNC) (Wang et al., 2016), Binary Neighbor Compression (BNC) (Zhong et al., 2017) are the current state-of-the-art.

SNC learns artificial prototypes such that the likelihood of a particular class probability model is maximized. Thus, SNC applies only to multi-class problems and its extension to multilabel/ranking problems is non-trivial. In contrast, we have a more direct discriminative formulation that can be applied to arbitrary supervised learning problems. To decrease the model size, SNC introduces a pre-processing step of low-d projection of the data via LMNN based projection matrix and then learns prototypes in the projected space. The SNC parameters (projection matrix, prototypes) might have to be hard-thresholded *post-facto* to fit within the memory budget. In contrast, ProtoNN's parameters are *de-facto* learnt jointly with model size constraints imposed during optimization. This leads to significant improvements over SNC and other state-of-the-art methods in the small model-size regime; see Figure 1, 3.

DSNC is a non-linear extension of SNC in that it learns a non-linear low-d transformation jointly with the prototypes. It has similar drawbacks as SNC: a) it only applies to multi-class problems and b) model size of DSNC can be significantly larger than SNC as it uses a feedforward network to learn the non-linear transformation.

BNC is a binary embedding technique, which jointly learns a binary embedding and a set of artificial binary prototypes. Although BNC learns binary embeddings, its dimensionality can be significantly higher, so it need not result in significant model compression. Moreover, the optimization in BNC is difficult because of the discrete optimization space.

3. Problem Formulation

Given n data points $X = [\mathbf{x}_1, \mathbf{x}_2, \dots \mathbf{x}_n]^T$ and the corresponding target output $Y = [\mathbf{y}_1, \mathbf{y}_2 \dots \mathbf{y}_n]^T$, where $\mathbf{x}_i \in \mathbb{R}^d, \mathbf{y}_i \in \mathcal{Y}$, our goal is to learn a model that accurately predicts the desired output of a given test point. In addition, we also want our model to have small size. For both multi-label/multi-class problems with L labels, $\mathbf{y}_i \in \{0,1\}^L$ but in multi-class $\|\mathbf{y}_i\| = 1$. Similarly, for ranking problems, the output y_i is a permutation.

Let's consider a smooth version of kNN prediction function for the above given general supervised learning problem

$$\hat{\mathbf{y}} = \rho(\hat{s}) = \rho\left(\sum_{i=1}^{n} \sigma(\mathbf{y}_i) K(\mathbf{x}, \mathbf{x}_i)\right), \tag{1}$$

where $\hat{\mathbf{y}}$ is the predicted output for a given input $\mathbf{x}$, $\hat{s} = \sum_{i=1}^{n} \sigma(\mathbf{y}_i) K(\mathbf{x}, \mathbf{x}_i)$ is the score vector for $\mathbf{x}$. $\sigma : \mathcal{Y} \to \mathbb{R}^L$ maps a given output into a score vector and $\rho : \mathbb{R}^L \to \mathcal{Y}$ maps the score function back to the output space. For example, in the multi-class classification, σ is the identity function while $\rho = \text{Top}_1$, where $[\text{Top}_1(s)]_j = 1$ if s_j is the largest element and 0 otherwise. $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is the similarity function, i.e., $K(\mathbf{x}_i, \mathbf{x}_j)$ computes similarity between $\mathbf{x}_i$ and $\mathbf{x}_j$. For example, standard kNN uses $K(\mathbf{x}, \mathbf{x}_i) = \mathbb{I}[\mathbf{x}_i \in \mathcal{N}_k(\mathbf{x})]$ where $\mathcal{N}_k(\mathbf{x})$ is the set of k nearest neighbors of $\mathbf{x}$ in X.

Note that kNN requires entire X to be stored in memory for prediction, so its model size and prediction time are prohibitive for resource constrained devices. So, to bring down model and prediction complexity of kNN, we propose using prototypes that represent the entire training data. That is, we learn prototypes $B = [\mathbf{b}_1, \dots, \mathbf{b}_m]$ and the corresponding score vectors $Z = [\mathbf{z}_1, \dots, \mathbf{z}_m] \in \mathbb{R}^{L \times m}$, so that the decision function is given by: $\hat{\mathbf{y}} = \rho\left(\sum_{j=1}^{m} \mathbf{z}_j K(\mathbf{x}, \mathbf{b}_j)\right)$.

Existing prototype based approaches like SNC, DSNC have a specific probabilistic model for multi-class problems with the prototypes as the model parameters. In contrast, we take a more direct discriminative learning approach that allows us to obtain better accuracies in several settings along with generalization to any supervised learning problem, e.g., multi-label classification, regression, ranking, etc.

However, K is a fixed similarity function like RBF kernel which is not tuned for the task at hand and can lead to inaccurate results. We propose to solve this issue by learning a low-dimensional matrix $W \in \mathbb{R}^{\hat{d} \times d}$ that further brings down model/prediction complexity as well as transforms data into a space where prediction is more accurate.That is, our proposed algorithm ProtoNN uses the following prediction function that is based on *three* sets of learned parameters $W \in \mathbb{R}^{\hat{d} \times d}$, $B = [\mathbf{b}_1, \dots, \mathbf{b}_m] \in \mathbb{R}^{\hat{d} \times m}$, and $Z = [\mathbf{z}_1, \dots, \mathbf{z}_m] \in \mathbb{R}^{L \times m}$: $\hat{\mathbf{y}} = \rho\left(\sum_{j=1}^{m} \mathbf{z}_j K(W\mathbf{x}, \mathbf{b}_j)\right)$.

To further reduce the model/prediction complexity, we learn sparse set of Z, B, W. Selecting the correct simi-

larity function K is crucial to the performance of the algorithm. In this work we choose K to be the Gaussian kernel: $K_\gamma(x, y) = \exp\{-\gamma^2\|x - y\|_2^2\}$, which is a popular choice in many non-parametric methods (including regression, classification, density estimation).

Note that if $m = n$, and $W = I_{d\times d}$, then our prediction function reduces to the standard RBF kernel-SVM's decision function for binary classification. That is, our function class is universal: we can learn any arbitrary function given enough data and model complexity. We observe a similar trend in our experiments, where even with reasonably small amount of model complexity, ProotNN nearly matches RBF-SVM's prediction error.

Training Objective: We now provide the formal optimization problem to learn parameters Z, B, W. Let $\mathcal{L}(\hat{\mathbf{s}}, \mathbf{y})$ be the loss (or) risk of predicting score vector $\hat{\mathbf{s}}$ for a point with label vector $\mathbf{y}$. For example, the loss function can be standard hinge-loss for binary classification, or NDCG loss function for ranking problems.

Now, define the empirical risk associated with Z, B, W as

$$\mathcal{R}_{emp}(Z, B, W) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}\left(\mathbf{y}_i, \sum_{j=1}^{m} \mathbf{z}_j K_\gamma(\mathbf{b}_j, W\mathbf{x}_i)\right).$$

In the sequel, to simplify the notation, we denote the risk at i^{th} data point by $\mathcal{L}_i(Z, B, W)$ i.e., $\mathcal{L}_i(Z, B, W) = \mathcal{L}\left(\mathbf{y}_i, \sum_{j=1}^{m} \mathbf{z}_j K_\gamma(\mathbf{b}_j, W\mathbf{x}_i)\right)$. To jointly learn Z, B, W, we minimize the empirical risk with explicit sparsity constraints:

$$\min_{Z:\|Z\|_0\le s_Z, B:\|B\|_0\le s_B, W:\|W\|_0\le s_W} \mathcal{R}_{emp}(Z, B, W), \quad (2)$$

where $\|Z\|_0$ is equal to the number of non-zero entries in Z. For all our expeirments (multi-class/multi-label), we used the squared ℓ_2 loss function as it helps us write down the gradients easily and allows our algorithm to converge faster and in a robust manner. That is, $\mathcal{R}_{emp}(Z, B, W) = \frac{1}{n}\sum_{i=1}^{n} \|\mathbf{y}_i - \sum_{j=1}^{m} \mathbf{z}_j K_\gamma(\mathbf{b}_j, W\mathbf{x}_i)\|_2^2$. Note that the sparsity constraints in the above objective gives us explicit control over the model size. Furthermore, as we show in our experiments, jointly optimizing all the three parameters, Z, B, W, leads to better accuracies than optimizing only a subset of parameters.

4. Algorithm

We now present our algorithm for optimization of (2). Note that the objective in (2) is non-convex and is difficult to optimize. However, we present a simple alternating minimization technique for its optimization. In this technique, we alternately minimize Z, B, W while fixing the other two parameters. Note that the resulting optimization problem in each of the alternating steps is still non-convex. To optimize these sub-problems we use projected Stochastic Gradient Descent (SGD) for large datasets and projected Gradient Descent (GD) for small datasets.

Algorithm 1 ProtoNN: Train Algorithm

Input: data (X, Y), sparsities (s_Z, s_B, s_W), kernel parameter γ, projection dimension $\hat{d}$, no. of prototypes m, iterations T, SGD epochs e.
Initialize Z, B, W
for $t = 1$ **to** T **do** {alternating minimization}
  repeat {minimization of Z}
    randomly sample $S \subseteq [1, \dots n]$
    $Z \leftarrow HT_{s_Z}\left(Z - \eta_r \sum_{i\in S} \nabla_Z \mathcal{L}_i(Z, B, W)\right)$
  until e epochs
  repeat {minimization of B}
    randomly sample $S \subseteq [1, \dots n]$
    $B \leftarrow HT_{s_B}\left(B - \eta_r \sum_{i\in S} \nabla_B \mathcal{L}_i(Z, B, W)\right)$
  until e epochs
  repeat {minimization of W}
    randomly sample $S \subseteq [1, \dots n]$
    $W \leftarrow HT_{s_W}\left(W - \eta_r \sum_{i\in S} \nabla_W \mathcal{L}_i(Z, B, W)\right)$
  until e epochs
end for
Output: Z, B, W

Suppose we want to minimize the objective w.r.t Z by fixing B, W. Then in each iteration of SGD we randomly sample a mini-batch $S \subseteq [1, \dots n]$ and update Z as: $Z \leftarrow HT_{s_Z}\left(Z - \eta \sum_{i\in S} \nabla_Z \mathcal{L}_i(Z, B, W)\right)$, where $HT_{s_Z}(A)$ is the hard thresholding operator that thresholds the smallest $L \times m - s_Z$ entries (in magnitude) of A and $\nabla_Z \mathcal{L}_i(Z, B, W)$ denotes the partial derivative of $\mathcal{L}_i$ w.r.t Z. Note that GD procedure is just SGD with batch size $|S| = n$. Algorithm 1 presents pseudo-code for our entire training procedure.

Step-size: Setting correct step-size is critical to convergence of SGD methods, especially for non-convex optimization problems. For our algorithm, we select the initial step size using Armijo rule. Subsequent step sizes are selected as $\eta_t = \eta_0/t$ where η_0 is the initial step-size.

Initialization: Since our objective function (2) is non-convex, good initialization for Z, B, W is critical in converging efficiently to a good local optima. We used a randomly sampled Gaussian matrix to initialize W for binary and small multi-class benchmarks. However, for large multi class datasets (*aloi*) we use LMNN based initialization of W. Similarly, for multi-label datasets we use SLEEC (Bhatia et al., 2015) for initialization of W; SLEEC is an embedding technique for large multi-label problems.

For initialization of prototypes, we experimented with two different approaches. In one, we randomly sample training

data points in the transformed space and assign them as the prototypes; this is a useful technique for multilabel problems. In the other approach, we run k-means clustering in the transformed space on data points belonging to each class and pick the cluster centers as our prototypes. We use this approach for binary and multi-class problems.

Convergence: Although Algorithm 1 optimizes an ℓ_0 constrained optimization problem, we can still show that it converges to a local minimum due to smoothness of objective function (Blumensath & Davies, 2008). Moreover, if the objective function satisfies strong convexity in a small ball around optima, then appropriate initialization leads to convergence to that optima (Jain et al., 2014). In fact, our next section presents such a strong convexity result (wrt B) if the data is generated from a mixture of well-separated Gaussians. Finally, our empirical results (Section 6) indicate that the objective function indeed converges at a fast rate to a good local optimum leading to accurate models.

5. Analysis

In this section, we present an analysis of our approach for when data is generated from the following generative model: let each point $\mathbf{x}_i$ be sampled from a mixture of two Gaussians, i.e., $\mathbf{x}_i \overset{i.i.d}{\sim} 0.5{\cdot}\mathcal{N}(\mu_+, I){+}0.5{\cdot}\mathcal{N}(\mu_-, I) \in \mathbb{R}^d$ and the corresponding label $\mathbf{y}_i$ be the indicator of the Gaussian from which $\mathbf{x}_i$ is sampled. Now, it is easy to see that if the Gaussians are *well-separated* then one can design 2 prototypes ${\mathbf{b}_+}^*$, ${\mathbf{b}_-}^*$ such that the error of our method with $W = I$ and fixed $Z = [\mathbf{e}_1, \mathbf{e}_2]$ will lead to nearly Bayes' optimal classifier; $\mathbf{e}_i$ is the i-th canonical basis vector.

The goal of this section is to show that our method that optimizes the squared ℓ_2 loss objective (2) w.r.t. prototypes B, converges at a linear rate to a solution that is in a *small ball* around the global optima, and hence leads to nearly optimal classification accuracy.

We would like to stress that the goal of our analysis is to justify our proposed approach in a simple and easy to study setting. We do not claim new bounds for the mixture of Gaussians problem; it is a well-studied problem with several solid solutions. Our goal is to show that our method in this simple setting indeed converges to a nearly optimal solution at linear rate, thus providing some intuition for its success in practice. Also, our current analysis only studies optimization w.r.t. the prototypes B while fixing projection matrix W and prototype label vectors Z. Studying the problem w.r.t. all the three parameters is significantly more challenging, and is beyond the scope of this paper.

Despite the simplicity of our model, ours is one of the first rigorous studies of a classification method that is designed for resource constrained problems. Typically, the proposed methods in this regime are only validated using empirical results as theoretical study is quite challenging owing to the obtained non-convex optimization surface and complicated modeling assumptions.

For our first result, we ignore sparsity of B, i.e., $s_B = 2 \cdot d$. We consider the RBF-kernel for K with $\gamma^2 = \frac{1}{2}$.

Theorem 1. *Let $X = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$ and $Y = [\mathbf{y}_1, \ldots, \mathbf{y}_n]$ be generated from the above mentioned generative model. Set $W = I$, $Z = [\mathbf{e}_1, \mathbf{e}_2]$ and let $\mathbf{b}_+, \mathbf{b}_-$ be the prototypes. Let $n \to \infty$, $\bar{\mu} := \mu_+ - \mu_-$. Also, let $\mathbf{\Delta}_+ := \mathbf{b}_+ - \mu_+$, $\mathbf{\Delta}_- := \mathbf{b}_+ - \mu_-$, and let ${\mathbf{\Delta}_+}^T \bar{\mu} \geq -\frac{(1-\delta)}{2}\|\bar{\mu}\|^2$ for some fixed constant $\delta > 0$, and $d \geq 8(\alpha - \delta)\|\bar{\mu}\|^2$ for some constant $\alpha > 0$. Then, the following holds for the gradient descent step $\mathbf{b}_+{}' = \mathbf{b}_+ - \eta \nabla_{\mathbf{b}_+} \mathcal{R}$ where $\mathcal{R} = \mathbb{E}[\mathcal{R}_{emp}]$, and $\eta \geq 0$ is appropriately chosen:*

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

if $\|\mathbf{\Delta}_+\| \geq 8\|\bar{\mu}\| \exp\left\{-\frac{\alpha\|\bar{\mu}\|^2}{4}\right\}$.

See Appendix 8 for a detailed proof of this as well as the below given theorem. The above theorem shows that if the Gaussians are well-separated and the starting $\mathbf{b}_+$ is closer to μ_+ than μ_-, then the gradient descent step decreases the distance between $\mathbf{b}_+$ and μ_+ geometrically until $\mathbf{b}_+$ converges to a small ball around μ_+, the radius of the ball is exponentially small in $\|\mu_+ - \mu_-\|$. Note that our initialization method indeed satisfies the above mentioned assumption with at least a constant probability.

It is easy to see that in this setting, the loss function decomposes over independent terms from $\mathbf{b}_+$ and $\mathbf{b}_-$, and hence an identical result can be obtained for $\mathbf{b}_-$. For simplicity, we present the result for $n \to \infty$ (hence, expected value). Extension to finite samples should be fairly straightforward using standard tail bounds. The tail bounds will also lead to a similar result for SGD but with an added variance term.

Next, we show that if the $\mathbf{b}_+$ is even closer to μ_+, then the objective function becomes *strongly convex* in $\mathbf{b}_+, \mathbf{b}_-$.

Theorem 2. *Let $X, Y, \bar{\mu}, \mathbf{\Delta}_+, \mathbf{\Delta}_-$ be as given in Theorem 1. Also, let ${\mathbf{\Delta}_+}^T \bar{\mu} \geq -\frac{(1-\delta)}{2}\|\bar{\mu}\|^2$, for some small constant $\delta > 0$, $\|\bar{\mu}\|^2 \geq \frac{4}{(\ln 0.1)\delta}$, and $\|\mathbf{\Delta}_+\|^2 \leq 0.5$. Then, $\mathcal{R}$ with $W = I$ and $Z = [\mathbf{e}_1, \mathbf{e}_2]$ is a strongly convex function of B with condition number bounded by* 20.

Note that the initialization assumptions are much more strict here, but strong convexity with bounded condition number provides significantly faster convergence to optima. Moreover, this theorem also justifies our IHT based method. Using standard tail bounds, it is easy to show that if n grows linearly with s_B rather than d, the condition number bound still holds over sparse set of vectors, i.e., for sparse μ_+, μ_- and sparse $\mathbf{b}_+, \mathbf{b}_-$. Using this restricted strong convexity with (Jain et al., 2014) guarantees

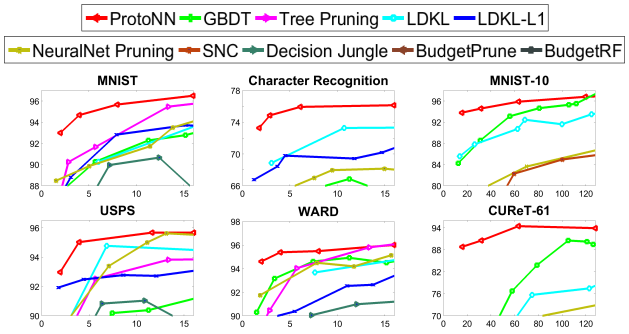

Figure 1. Model size (kB, X-axis) vs Accuracy (%, Y-axis): comparison of ProtoNN against existing compression algorithms on various datasets. The left two columns show the plots for binary datasets and the right most column shows the plots for multiclass datasets. For small model size, ProtoNN is significantly more accurate than baselines.

that with just $O(s_B \log d)$ samples, our method will converge to a small ball around sparse μ_+ in polynomial time. We skip these standard details as they are orthogonal to the main point of this analysis section.

6. Experiments

In this section we present the performance of ProtoNN on various benchmark binary, multiclass and multilabel datasets with a goal to demonstrate the following aspects:

a) In severely resource constrained settings where we require model sizes to be less than 2kB (which occur routinely for IoT devices like Arduino Uno), we outperform all state-of-the art compressed methods.
b) For model sizes in the range $16 - 32$ kB, we achieve comparable accuracies to the best uncompressed methods.
c) In multiclass and multilabel problems we achieve near state-of-the-art accuracies with an order of magnitude reduction in model size, thus showing our approach is flexible and general enough to handle a wide variety of problems.

Experimental Settings: ***Datasets:*** Table 3 in Appendix 9.1 lists the binary, multiclass and multilabel datasets used in our experiments. For binary and multiclass datasets, we standardize each feature in the data to zero-mean and unit-variance. For multilabel datasets, we normalize the feature vector of each data point by projecting it onto a unit norm ball which preserves data sparsity. ***Hyperparameters:*** In all our experiments, we fix the no. of alternating minimization iterations(T) to 150. Each such iteration does e-many epochs each over the 3 parameters, W, B, and Z. For small binary and multiclass datasets we do GD with e set to 20. For multilabel and large multiclass (*aloi*) datasets, we do SGD with e set to 5, batch size to 512. Kernel parameter γ is computed after initializing B, W as $\frac{2.5}{\text{median}(D)}$, where D is the set of distances between prototypes and training points in the transformed space and is defined as $D = \{\|\mathbf{b}_j - W\mathbf{x}_i\|_2\}_{i\in[n],j\in[m]}$.

ProtoNN vs. Uncompressed Baselines: In this experiment we compare the performance of ProtoNN with uncompressed baselines and demonstrate that even with compression, ProtoNN achieves near state-of-the-art accuracies. We restrict the model size of ProtoNN to 16kB for binary datasets and to 64kB for multiclass datasets and don't place any constraints on the model sizes of baselines. We compare ProtoNN with: GBDT, RBF-SVM, 1-Hidden Layer Neural Network (1-hidden NN), kNN, BNC and SNC. For baselines the optimal hyper-parameters are selected through cross-validation. For SNC, BNC we set projection dimensions to 100, 1280 respectively and compression ratios to 16%, 1%. For ProtoNN, hyper-parameters are set based on the following heuristics which ensure that the model size constraints are satisfied: **Binary:** $\hat{d} = 10$, $s_Z = s_B = 0.8$. $m = 40$ if $s_W = 1.0$ gives model larger than 16kB. Else, $s_W = 1.0$ and m is increased to reach 16 kB model. **Multiclass:** $\hat{d} = 15$, $s_Z = s_B = 0.8$. $m = 5$/class if $s_W = 1.0$ gives model larger than 64kb. Else, m is increased to reach 64 kB model. *CUReT* which has 61 classes, requires smaller s_Z to satisfy model size constraints.
We use the above parameter settings for all binary, multiclass datasets except for binary versions of *usps*, *character* and *eye* which require 5-fold cross validation. Table 1 presents the results on binary datasets and Table 2 presents the results on multiclass datasets. For most of the datasets, ProtoNN gets to within $1-2\%$ accuracy of the best uncompressed baseline with $1-2$ *orders of magnitude* reduction in model size. For example on *character recognition*, ProtoNN is 0.5% more accurate than the best method (RBF-SVM) while getting $\approx 400\times$ compression in model size. Similarly, on *letter-26*, our method is within 0.5% accuracy of RBF-SVM while getting $\approx 9\times$ compression. Also note that ProtoNN with 16kB models is still able to outperform BNC, SNC on most of the datasets.

ProtoNN vs. Compressed Baselines: In this experiment we compare the performance of ProtoNN with other state-of-the-art compressed methods in the 2-16kB model size regime: BudgetRF (Nan et al., 2015), Decision Jungle (Shotton et al., 2013), LDKL (Jose et al., 2013), Tree Pruning (Dekel et al., 2016), GBDT (Friedman, 1999), Budget Prune (Nan et al., 2016), SNC and NeuralNet Pruning (Han et al., 2016). All baselines plots are obtained via cross-validation. Figure 1 presents the memory vs. accuracy plots. Hyper-parameters of ProtoNN are set as follows: **Binary:** $s_B = s_Z = 0.8$. For $[2, 4, 8, 16]$ kB, $\hat{d} = [5, 5, 10, 15]$. s_W, m are set using the same heuristic mentioned in the previous paragraph. **Multiclass:** $s_B = 0.8$. For $[16, 32, 64, 128]$ kB, $\hat{d} = [10, 15, 15, 20]$. s_W, s_Z, m are set as defined in the previous paragraph.
ProtoNN values obtained with the above hyper-parameters

Table 1. Comparison of ProtoNN with uncompressed baselines on binary datasets. Model size is computed as $\#parameters \times 4$ bytes; sparse matrices taking an extra 4 bytes for each non-zero entry, for the index. For BNC it is computed as $\#parameters/8$ bytes. GBDT model size is computed using the file size on disk.

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**

Table 2. Comparison of ProtoNN with uncompressed baselines on multiclass datasets. First number in each cell refers to the model size in kB and the second number denotes accuracy. Refer to Table 1 for details about calculation of model size.

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

are reported for all datasets, except *usps* and *character recognition* which require 5-fold cross validation. ProtoNN performs significantly better than the baselines on all the datasets. This is especially true in the 2kB regime, where ProtoNN is $\geq 5\%$ more accurate on most of the datasets.

ProtoNN on Multilabel and Large Multiclass Datasets: We now present the performance of ProtoNN on larger datasets. Here, we experimented with the following datasets: *aloi* dataset which is a relatively large multiclass dataset , *mediamill, delicious, eurlex* which are small-medium sized multilabel datasets.

We set the hyper-parameters of ProtoNN as follows. $\hat{d}$ is set to 30 for all datasets, except for eurlex for which we set it to 100. Other parameters are set as follows: $s_W = 1, s_B = 1$, $s_Z = 5/L$ for *aloi* and $s_Z = 2(\text{avg. number of labels per training point})/L$ for multilabel datasets, $m = 2 \cdot L$ for multilabel datasets.

For *aloi*, we compare ProtoNN with the following baselines: 1vsA L2 Logistic Regression (1vsA-Logi), RBF-SVM, FastXML: a large-scale multilabel method (Prabhu & Varma, 2014), Recall Tree: a scalable method for large multiclass problems (Daume III et al., 2016). For 1vsA-Logi, Recall Tree we perform cross validation to pick the best tuning parameter. For FastXML we use the default parameters. For RBF-SVM we set γ to the default value $1/d$ and do a limited tuning of the regularization parameter.

Left table of Figure 2 shows that ProtoNN (with $m = 5000$) gets to within 1% of the accuracy of RBF-SVM with just $(1/50)^{th}$ of its model size and 50 times fewer floating point computations per prediction. For a better comparison of ProtoNN with FastXML, we set the number of prototypes ($m = 1500$) such that computations/prediction of both the methods are almost the same. We can see that ProtoNN gets similar accuracy as FastXML but with a model size 2 orders of magnitude smaller than FastXML. Finally, our method has almost same prediction cost as Recall-Tree but with 10% higher accuracy and $4\times$ smaller model size.

Right table of Figure 2 presents preliminary results on multilabel datasets. Here, we compare ProtoNN with SLEEC, FastXML and DiSMEC (Babbar & Shölkopf, 2016), which learns a 1vsA linear-SVM in a distributed fashion. ProtoNN almost matches the performance of all baselines with huge reduction in model size.

These results show that ProtoNN is very flexible and can handle a wide variety of problems very efficiently. SNC doesn't have such flexibility. For example, it can't be naturally extended to handle multilabel classification problems.

ProtoNN vs. BNC, SNC: In this experiment, we do a thorough performance comparison of ProtoNN with BNC and SNC. To show that ProtoNN learns better prototypes than BNC, SNC, we fix the projection dimension $\hat{d}$ of all the methods and vary the number of prototypes m. To show that ProtoNN learns a better embedding, we fix m and vary $\hat{d}$. For BNC, which learns a binary embedding, $\hat{d}$ is chosen such that the $\#parameters$ in its transformation matrix is 32 times the $\#parameters$ in transformation matrices of ProtoNN, SNC. m is chosen similarly. Figure 3 presents the results from this experiment on *mnist* binary dataset. We use the following hyper parameters for ProtoNN: $s_W = 0.1, s_Z = s_B = 1.0$. For SNC, we hard threshold the input transformation matrix so that it has sparsity 0.1. Note that for small $\hat{d}$ our method is as much as

Figure 2. Left Table: ProtoNN vs baselines on *aloi* dataset. For Recall Tree we couldn't compute the avg. number of computations needed per prediction, instead we report the prediction time w.r.t 1vsA-Logi. Right Table: ProtoNN vs baselines on multilabel datasets. For SLEEC and FastXML we use the default parameters from the respective papers. Both the tables show that our method achieves similar accuracies as the baselines, but often with $1-2$ orders of magnitude compression in model size. On *aloi* our method is at most 2 slower than 1-vs-all while RBF-SVM is $115\times$ slower.

Method	Accuracy	Model Size(MB)	computations/prediction w.r.t 1vsA-Logi
1vsA-Logi	86.96	0.512	1
RBF-SVM	**94.77**	38.74	115.7
FastXML	89.86	254.53	**0.752**
Recall Tree	85.15	3.69	2.89*
ProtoNN (m = 1500)	89.6	**0.315**	0.792
ProtoNN (m = 5000)	94.05	0.815	2.17

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

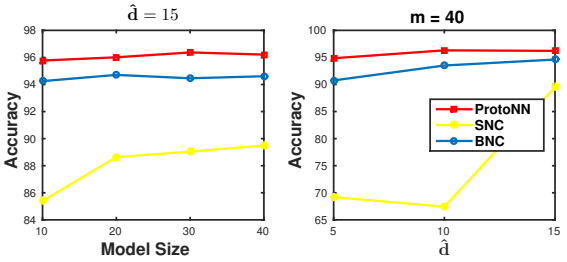

Figure 3. Comparison of ProtoNN with BNC, SNC on *mnist* binary dataset with varying projection dimension $\hat{d}$ or number of prototypes m.

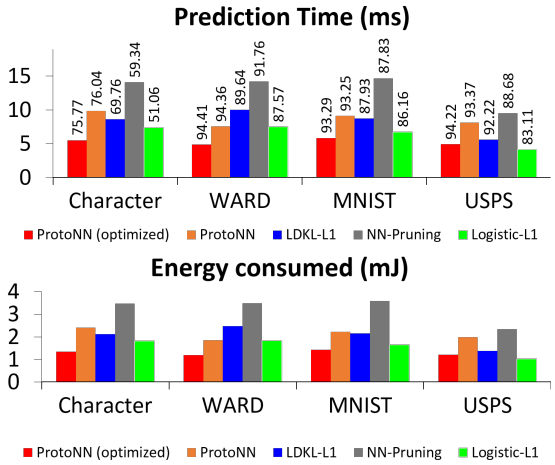

Figure 4. Prediction time and energy consumed by ProtoNN (2kB) and its optimized version against baselines. The accuracy of each model is on top of its prediction time bar.

20% more accurate than SNC, 5% more accurate than BNC and reaches nearly optimal accuracy for small $\hat{d}$ or m.

Remark 1. *Before we conclude the section we provide some practical guidelines for hyper-parameter selection in ProtoNN. Consider the following two cases:*
a) Small L ($L \lessapprox 0.1d$): *In this case, parameters $\hat{d}$ and s_W govern the model size. Given a model size constraint, fixing one parameter fixes the other, so that we effectively have one hyper-parameter to cross-validate. Choosing m such that $10 \leq m/L \leq 20$ typically gives good accuracies.*
b) Large L ($L \gtrapprox 0.1d$): *In this case, s_Z also governs the model size. s_Z, s_W and $\hat{d}$ can be selected through cross-validation. If the model size allows it, increasing $\hat{d}$ typically helps. Fixing m/L to a reasonable value such as 3-10 for medium L, 1-2 for large L typically gives good accuracies.*

7. Experiments on tiny IoT devices

In the previous section, we showed that ProtoNN gets better accuracies than other compressed baselines at low model size regimes. For small devices, it is also critical to study other aspects like energy consumption, which severely impact the effectiveness of a method in practice. In this section, we study the energy consumption and prediction time of ProtoNN model of size 2kB when deployed on an Arduino Uno.The Arduino Uno has an 8 bit, 16 MHz Atmega328P microcontroller, with 2kB of SRAM and 32kB of read-only flash. We compare ProtoNN with 3 baselines (LDKL-L1, NeuralNet Pruning, L1 Logistic) on 4 binary datasets. Figure 4 presents the results from this experiment. ProtoNN shows almost the same characteristics as a simple linear model (L1-logistic) in most cases while providing significantly more accurate predictions.

Further optimization: The Atmega328P microcontroller supports native integer arithmetic at $\approx 0.1\mu$s/operation, software-based floating point arithmetic at $\approx 6\mu$s/operation; exponentials are a further order slower. It is thus desirable to perform prediction only using integers. We implemented an integer version of ProtoNN to leverage this. We factor out a common float value from the parameters and round the residuals by 1-byte integers. To avoid computing the exponentials, we store a pre-computed table of approximate exponential values. As can be seen in Figure 4, this optimized version of ProtoNN loses only a little accuracy, but obtains $\approx 2\times$ reduction in energy and prediction cost.

References

Angiulli, Fabrizio. Fast condensed nearest neighbor rule. In *ICML*, 2005.

Babbar, Rohit and Shölkopf, Bernhard. Dismec-distributed sparse machines for extreme multi-label classification. In *arXiv preprint arXiv:1609.02521, Accepted for Web Search and Data Mining Conference (WSDM) 2017*, 2016.

Bentley, Jon Louis. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18:509–517, 1975.

Beygelzimer, Alina, Kakade, Sham, and Langford, John. Cover trees for nearest neighbor. In *ICML*, 2006.

Bhatia, Kush, Jain, Himanshu, Kar, Purushottam, Varma, Manik, and Jain, Prateek. Sparse local embeddings for extreme multi-label classification. In *NIPS*, pp. 730–738, 2015.

Blumensath, Thomas and Davies, Mike E. Iterative thresholding for sparse approximations. *Journal of Fourier Analysis and Applications*, 14:629–654, 2008.

Cover, T. and Hart, P. Nearest neighbor pattern classification. *IEEE Trans. Inf. Theor.*, 13:21–27, 2006.

Daume III, Hal, Karampatziakis, Nikos, Langford, John, and Mineiro, Paul. Logarithmic time one-against-some. *arXiv preprint arXiv:1606.04988*, 2016.

Davis, Jason V., Kulis, Brian, Jain, Prateek, Sra, Suvrit, and Dhillon, Inderjit S. Information-theoretic metric learning. In *ICML*, 2007.

Dekel, O., Jacobbs, C., and Xiao, L. Pruning decision forests. In *Personal Communications*, 2016.

Devi, V Susheela and Murty, M Narasimha. An incremental prototype set building technique. *Pattern Recognition*, 35, 2002.

Friedman, Jerome H. Stochastic gradient boosting. *Computational Statistics and Data Analysis*, 38:367–378, 1999.

Gionis, Aristides, Indyk, Piotr, Motwani, Rajeev, et al. Similarity search in high dimensions via hashing. In *VLDB*, volume 99, pp. 518–529, 1999.

Goldberger, Jacob, Roweis, Sam T., Hinton, Geoffrey E., and Salakhutdinov, Ruslan. Neighbourhood components analysis. In *NIPS*, 2004.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *ICLR*, 2016.

Jain, Prateek, Tewari, Ambuj, and Kar, Purushottam. On iterative hard thresholding methods for high-dimensional m-estimation. In *NIPS*, pp. 685–693, 2014.

Jose, Cijo, Goyal, Prasoon, Aggrwal, Parv, and Varma, Manik. Local deep kernel learning for efficient non-linear SVM prediction. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, pp. 486–494, 2013.

Kulis, Brian and Darrell, Trevor. Learning to hash with binary reconstructive embeddings. In *Advances in neural information processing systems*, pp. 1042–1050, 2009.

Kusner, Matt J., Tyree, Stephen, Weinberger, Kilian, and Agrawal, Kunal. Stochastic neighbor compression. In *ICML*, 2014.

Liu, Wei, Wang, Jun, Ji, Rongrong, Jiang, Yu-Gang, and Chang, Shih-Fu. Supervised hashing with kernels. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pp. 2074–2081. IEEE, 2012.

Mollineda, Ramón Alberto, Ferri, Francesc J, and Vidal, Enrique. An efficient prototype merging strategy for the condensed 1-nn rule through class-conditional hierarchical clustering. *Pattern Recognition*, 35:2771–2782, 2002.

Nan, F., Wang, J., and Saligrama, V. Feature-budgeted random forest. In *ICML*, 2015.

Nan, F., Wang, J., and Saligrama, V. Pruning random forests for prediction on a budget. 2016.

Norouzi, Mohammad, Fleet, David J, and Salakhutdinov, Ruslan R. Hamming distance metric learning. In *Advances in neural information processing systems*, pp. 1061–1069, 2012.

Prabhu, Yashoteja and Varma, Manik. Fastxml: A fast, accurate and stable tree-classifier for extreme multi-label learning. In *KDD*, 2014.

Shotton, J., Sharp, T., Kohli, P., Nowozin, S., Winn, J., and Criminisi, A. Decision jungles: Compact and rich models for classification. In *NIPS*, 2013.

Wang, Wenlin, Chen, Changyou, Chen, Wenlin, Rai, Piyush, and Carin, Lawrence. Deep metric learning with data summarization. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 777–794. Springer, 2016.

Weinberger, Kilian Q. and Saul, Lawrence K. Distance metric learning for large margin nearest neighbor classification. *J. Mach. Learn. Res.*, 10:207–244, 2009.

Weiss, Yair, Torralba, Antonio, and Fergus, Robert. Spectral hashing. In *NIPS*, pp. 1753–1760. Curran Associates, Inc, 2008.

Zhong, Kai, Guo, Ruiqi, Kumar, Sanjiv, Yan, Bowei, Simcha, David, and Dhillon, Inderjit. Fast Classification with Binary Prototypes. In Singh, Aarti and Zhu, Jerry (eds.), *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pp. 1255–1263, Fort Lauderdale, FL, USA, 20–22 Apr 2017. PMLR. URL http://proceedings.mlr.press/v54/zhong17a.html.