A Resource-Aware Streaming-Based Framework for Big Data Analysis

by

Bita Darvish Rouhani

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree Master of Science

Approved, Thesis Committee:

Prof. Farinaz Koushanfar, Chair
Professor of Electrical and Computer Engineering

Prof. Behnaam Aazhang
J.S. Abercrombie Professor of Electrical and Computer Engineering

Prof. Richard Baraniuk
Victor E. Cameron Professor of Electrical and Computer Engineering

Houston, Texas

December, 2015
ABSTRACT

A Resource-Aware Streaming-Based Framework for Big Data Analysis

by

Bita Darvish Rouhani

The ever growing body of digital data is challenging conventional analytical techniques in machine learning, computer vision, and signal processing. Traditional analytical methods have been mainly developed based on the assumption that designers can work with data within the confines of their own computing environment. The growth of big data, however, is changing that paradigm especially in scenarios where severe memory and computational resource constraints exist. This thesis aims at addressing major challenges in big data learning problem by devising a new customizable computing framework that holistically takes into account the data structure and underlying platform constraints. It targets a widely used class of analytical algorithms that model the data dependencies by iteratively updating a set of matrix parameters, including but not limited to most regression methods, expectation maximization, and stochastic optimizations, as well as the emerging deep learning techniques. The key to our approach is a customizable, streaming-based data projection methodology that adaptively transforms data into a new lower-dimensional embedding by simultaneously considering both data and hardware characteristics. It enables scalable data analysis and rapid prototyping of an arbitrary matrix-based learning task using a sparse-approximation of the collection that is constantly updated inline with the data arrival. Our work is supported by a set of user-friendly Application Program-
mung Interfaces (APIs) that ensure automated adaptation of the proposed framework to various datasets and System on Chip (SoC) platforms including CPUs, GPUs, and FPGAs. Proof of concept evaluations using a variety of large contemporary datasets corroborate the practicability and scalability of our approach in resource-limited settings. For instance, our results demonstrate 50-fold improvement over the best known prior-art in terms of memory, energy, power, and runtime for training and execution of deep learning models in deployment of different sensing applications including indoor localization and speech recognition on constrained embedded platforms used in today’s IoT enabled devices such as autonomous vehicles, robots, and smartphone.
The completion of this thesis would not have been possible without the contribution of several individuals who in one way or another have provided their valuable guidance and assistance.

My deepest gratitude to my advisor Professor Farinaz Koushanfar for her priceless support, encouragement, and guidance. Her dedication, fondness, and motivation towards doing novel research, as well as exceptional care for her students have taught me invaluable lessons and greatly influenced my life at both academic and personal levels.

I would also like to thank my committee members, Professor Behnaam Aazhang and Professor Richard G. Baraniuk for their brilliant comments and suggestions on this thesis.

Last but not the least, I wish to express my profound admiration and gratitude to my beloved parents and brothers for their endless love and continuous support in my life.
## Contents

Abstract 																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																									

1 Introduction

1.1 Hardware-Accelerated Streaming Sketch-based Analysis of Big Data 		3
1.2 Deep Learning of Massive Data on Constrained Platforms 		6
1.3 Contribution 		8

2 Preliminaries and Background 

2.1 Streaming Model 		10
2.2 Orthogonal Matching Pursuit 		11
2.3 Deep Learning 		11
2.4 Notation 		12

3 SSsketch: Automated Big Data Analysis Using Reconfigurable Platforms 

3.1 SSsketch Overview 		14
3.2 SSsketch Methodology 		15
3.2.1 Blocking SSsketch 		20
3.2.2 Theoretical Bound on SSsketch Approximation Error 		21
3.3 SSsketch Automated Hardware-accelerated Implementation 		22
3.4 Automated SSsketch Customization 		25
3.4.1 Motivation 		25
3.4.2 Constraint-Driven Optimization ........................................ 26

4 Automated Deep Learning on Embedded Systems .................. 31
  4.1 MobiDeep Overview .................................................. 31
  4.2 Automated Projection Customization ................................ 32
    4.2.1 Memory Customization ........................................... 33
    4.2.2 Error Customization ............................................. 34
    4.2.3 Runtime Customization .......................................... 34
  4.3 DL-based Classifier .................................................... 35
    4.3.1 Training Phase .................................................. 36
    4.3.2 Execution Phase .................................................. 37

5 Evaluations and Practical Design Experiences ....................... 40
  5.1 SSketch Evaluations .................................................. 40
    5.1.1 Light Field ..................................................... 40
    5.1.2 Hyperspectral .................................................. 42
    5.1.3 SSketch’s Hardware Customized Experiments .................. 44
    5.1.4 Scalability ..................................................... 46
    5.1.5 SSketch’s Hardware Settings and Results ..................... 48
  5.2 MobiDeep Evaluations ................................................ 51
    5.2.1 Applications ................................................... 51
    5.2.2 Pre-processing and Customization ............................ 54
    5.2.3 Performance Evaluation ........................................ 56
    5.2.4 Discussion ..................................................... 59

6 Related Work ..................................................................... 62
  6.1 Streaming-based Data Sketching Using Reconfigurable Platforms . 62
  6.2 Deep Learning on Resource-Constrained Platforms ................ 64
Illustrations

3.1 High level block diagram of SSketch. It takes stream of data as input and adaptively learns a corresponding sketch of the collection by doing computation at the level of matrix rank. The resulting sketch is then sent back to the host for further analysis depending on the application. .................................................. 15

3.2 Schematic depiction of blocking SSketch. ................................. 21

3.3 High level diagram of SSketch API. The constraint-driven customization unit of SSketch takes user-defined properties and hardware limitations as inputs and delivers output parameters that can be used for an optimized sketch computation. ......................... 24

3.4 Overview of automated SSketch customization. ...................... 25

3.5 SSketch’s FPGA-accelerated implementation performance. $l$, $k$, $\epsilon$, and $m_b$ are SSketch’s algorithmic parameters which indicate the number of samples in the dictionary matrix, target sparsity level, error threshold, and block-size respectively. $n_k$ denotes the number of OMP kernels that work in parallel in data sketching unit. In both figures (b), and (c) we set $k = l$ to let $k$ be as large as the dictionary size and use $\epsilon$ as the stopping criteria in the OMP algorithm. .......... 28
4.1 High level block diagram of MobiDeep. The raw data is projected to a lower-dimensional embedding using a new customizable method. The new method is resource-aware and is tailored to benefit the subsequent learning task. 32

4.2 Schematic depiction of a 3-layer DNN. The circles labeled “+1” denote the bias units within each layer. 36

5.1 Experimental evaluations of SSketch. $\alpha$, $\epsilon$, $l$, and $m_b$ are SSketch’s algorithmic parameters which indicate the projection threshold, error threshold, number of samples in the dictionary matrix, and block-size respectively. $X_{err}$ represents the data matrix sketching error and $G_{err}$ represents the approximation error for the corresponding Gram matrix. The spectral transformation error is denoted by $e_k$. 43

5.2 Approximation error ($G_{err}$) vs. $l$ for two different hyperspectral datasets. The SSketch algorithmic parameters are set to $\alpha = 0.1$ and $\epsilon = 0.01$, where $\alpha$ is the projection threshold and $\epsilon$ is the error threshold. 44

5.3 SSketch’s automated constraint-driven customization. Each plot demonstrates the average runtime required to process one sample as a function of the dictionary size. The dashed horizontal line, reflects the user runtime deadline and the star point illustrates the SSketch’s automated customization output. The user power constraint selects the number of OMP kernels that can work in parallel. The crossing point of the dashed line and the corresponding runtime curve (solid curve) is the actual optimized point in each setting. 46

5.4 Computational complexity comparison of different OMP implementations. Using QR decomposition significantly improves OMP’s runtime. 47
5.5 MobiDeep’s pre-processing customization for daily and sport activities recognition. The optimal sparsity level \( k \) that minimizes the inference error can be found by using only a small portion of training samples.

5.6 Inference error over time for different input SNRs. The sparsity level for transformed data is set to be \( k = 19 \), and the dictionary size is \( l = 456 \). The reported runtime correspond to MobiDeep execution on the GPU of Jetson TK1 board.
# Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Power consumption on Virtex 6 for different number of OMP kernels.</td>
<td>29</td>
</tr>
<tr>
<td>5.1</td>
<td>Complexity of different sketching methods.</td>
<td>48</td>
</tr>
<tr>
<td>5.2</td>
<td>Virtex-6 resource utilization.</td>
<td>49</td>
</tr>
<tr>
<td>5.3</td>
<td>SSketch total processing time is linear in terms of the number of processed samples.</td>
<td>50</td>
</tr>
<tr>
<td>5.4</td>
<td>Size of evaluation datasets.</td>
<td>52</td>
</tr>
<tr>
<td>5.5</td>
<td>Pre-processing overhead (seconds) of MobiDeep which accounts for both tuning and data transformation.</td>
<td>56</td>
</tr>
<tr>
<td>5.6</td>
<td>Performance improvement achieved by MobiDeep over conventional deep learning approach.</td>
<td>59</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

The ever growing body of digital data is challenging traditional data analysis techniques in machine learning, computer vision, and signal processing. Several difficulties arise due to impracticality of applying regular data analysis algorithms directly to massive collections of high dimensional data. This disruption of convention changes the way we analyze modern datasets and renders designing scalable data transformation methods, a.k.a., sketching algorithms, a necessity *. A sketch matrix is a compact approximation of the original matrix embedding to lower dimensional subspaces. With a properly designed sketch matrix the intended computations can be performed on an ensemble of lower dimensional structures rather than the original matrix without a significant loss. Note that traditional sketching methods such as Singular Value Decomposition (SVD) and Principal Component Analysis (PCA) incur a large memory footprint with a quadratic computational complexity which limit their practicability in big data regime.

This thesis aims at addressing major challenges in big data learning problem using a combination of hardware and algorithmic approaches. It introduces a set of automated computing frameworks that are devised to adaptively customize conventional machine learning methods (e.g., various regression and classification problems) that are based on iterative matrix updates, as well as the emerging Deep Learning (DL)

*Throughout this thesis, we use transformation or projection approach alternatively.
algorithms to make it amenable to use such techniques for analyzing large amount of data in resource-constrained settings at scales that were not feasible before. The core of our proposed computing frameworks is a new streaming based data transformation (sketching) that holistically takes into the account the application data and underlying platform constraints. Our approach is devised to transform data to an ensemble of lower dimensional subspaces in which applying such learning algorithms is much more efficient in terms of runtime, memory, and energy consumption.

In particular, we corroborate the practicability of our customized data transformation by (i) Devising a generic hardware-accelerated computing framework, called SSketch, for streaming sketch-based analysis of big data on reconfigurable platforms. SSketch takes advantage of a HW/SW co-design approach. It provides an API that can be customized for rapid prototyping of an arbitrary matrix-based analytical algorithm in deployment of various learning applications including but not limited to image super-resolution, and de-noising. Our results show significant speedups in comparison to a single multi-core processing unit, highlighting the importance of incorporating hardware in large-scale streaming applications. (ii) Introducing the first end-to-end automated framework, called MobiDeep that enables both on-chip training and execution of deep neural networks in deployment of resource-constrained autonomous sensing applications. MobiDeep adaptively reduces the data and circuit footprint required for implementing DL which translate to meaningful memory, power, energy, and runtime savings.
1.1 Hardware-Accelerated Streaming Sketch-based Analysis of Big Data

Real time processing of streaming data is critical for many applications in which storage is severely limited and data can be read at most once [1]. In such scenarios, to cope with the dynamics of the streaming content just in time the sketch has to be found inline with the data arrival. A large body of earlier work demonstrated the efficiency of using custom hardware for acceleration of traditional matrix sketching algorithms such as QR [2], LU [3], Cholesky [4], and SVD [5]. However, the existing hardware-accelerated sketching methods either have a higher-than-linear complexity [6], or are non-adaptive for online sketching [5]. They are thus unsuitable for streaming applications and big data analysis with dense correlation matrices. A recent theoretical solution for scalable sketching of big data matrices is presented in [1], which also relies on running SVD on the sketch matrix. Even this method is unable to handle the changing data dynamics in real time as the SVD algorithm incurs a higher than linear computational complexity. Moreover, runtime and power constraints are not addressed in [1] either.

We propose SSketch, a novel automated computing framework for efficient analysis and FPGA-based acceleration of big data with dense (non-sparse) correlation matrices. The stream of input data is used by SSketch for adaptive learning and updating a corresponding ensemble of lower dimensional data structures (a.k.a., sketch matrix). A new sketching methodology is introduced that tailors the problem of transforming the big data with dense correlations to an ensemble of lower dimensional subspaces such that it is suitable for hardware-based acceleration performed by reconfigurable hardware. The new method is scalable, while it significantly re-
duces costly memory interactions and enhances matrix computation performance by leveraging coarse-grained parallelism existing in the dataset. It works by factorizing the original (densely correlated) large matrix into two new matrices: (i) a dense but much smaller dictionary matrix which includes a set of atoms learned from the input data, and (ii) a large block-sparse matrix where the blocks are organized such that the subsequent matrix computations incur a minimal amount of message passings on the target platform.

An important property of SSketch is its capability to customize its sketching approach based upon the user-defined requirements and underlying hardware limitations. More precisely, we provide an automated optimization approach that can be used to customize SSketch framework to compute the best sketch matrix (with the least approximation error) under runtime, power, and memory constraints. As the stream of input data arrives, SSketch adaptively learns from the incoming vectors and updates the sketch of the collection. To facilitate automation, SSketch takes advantage of a HW/SW co-design approach: It provides an API that can be customized for rapid prototyping of an arbitrary matrix-based data analysis algorithm. SSketch and its accompanying API target a broad class of learning algorithms that model the data dependencies by iteratively updating a set of matrix parameters, including but not limited to most regression methods, belief propagation, expectation maximization, and stochastic optimizations [7]. Our proof-of-concept evaluations on a variety of visual datasets with more than 11 million non-zeros demonstrates up to 200-fold speedup on our hardware-accelerated realization of SSketch compared to a software-based deployment on a general purpose processor.

Our framework addresses the big data learning problem by using MobiDeep’s block-sparse matrix and applying an efficient, greedy routine called Orthogonal
Matching Pursuit (OMP) on each sample independently. Note that OMP is a key computational kernel that dominates the performance of many sparse reconstruction algorithms. Given the wide range of applications, it is thus not surprising that a large number of OMP implementations on GPUs, ASICs and FPGAs have been reported in the literature, for example [8], [9], and [10]. However, the prior work on FPGA had focused on fixed-point number format. In addition, most earlier hardware-accelerated OMP designs are restricted to constant and small matrix sizes and can handle only a few number of OMP iterations [11], [10], and [12]. Such designs cannot be readily applied to massive, dynamic datasets. In contrast, MobiDeep’s scalable methodology introduces a novel approach that enables applying OMP on matrices with varying large sizes and supports arbitrary number of iterations.

MobiDeep uses the abundant hardware resources on current FPGAs to provide a scalable, floating-point implementation of OMP for sketching purposes. One may speculate that GPUs may show a better acceleration performance than FPGAs. However, the performance of GPU accelerators is limited in our application because of two main reasons. First, for streaming applications, the memory hierarchy in GPUs increases the overhead in communication and thus reduces the throughput of the whole system. Second, in our sketching approach, the number of required operations to compute the sketch of each individual sample depends on the input data structure and may vary from one sample to the other. Thus, the GPU’s applicability is reduced due to its Single Instruction Multiple Data (SIMD) architecture.

An earlier version of our proposed framework (SSketch) has appeared in the 23rd IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM), 2015 [13].
1.2 Deep Learning of Massive Data on Constrained Platforms

Recent advances in the field of deep learning provide a significant leap in our ability to comprehend raw data in a variety of complex learning and understanding tasks [14, 15, 16]. Deep learning techniques are becoming the key elements in achieving state-of-the-art inference performance by moving beyond traditional linear or polynomial analytical approaches to the modern non-linear, and sophisticated ones. These approaches, inspired by neural activities in the brain, learn complex concepts through a multi-layer data modeling in which data features in higher layers are constructed from those in preceding ones.

While the nonlinear and sophisticated nature of DL empowers it to achieve extraordinary inference capability, it inevitably brings a new set of challenges concerning scalability and resource utilization. The high computational overhead of DL particularly hinders its practicality in resource-constrained platforms such as autonomous vehicles, robots, smartphones, and wearable devices. Given the rich set of embedded sensors in today’s IoT enabled devices (e.g., accelerometers, gyroscopes, microphones, and cameras), and the great promise of deep learning, the ability to locally learn and infer sensing data undoubtedly provides a paradigm shift across a variety of domains, including health care [17], social networks [18], environmental monitoring [19], and transportation [20]. However, the existing implementations of deep learning on resource-constrained System on Chip (SoC) platforms either outsource the costly steps of learning, or they only perform execution of the algorithm after the network has been learned off-line using the power of clouds.

In this thesis, we propose MobiDeep, a novel automated framework that enables
on-chip training and execution of DL models by customizing it to the underlying resource-constrained SoC platform used in IoT enabled devices. There are several advantages in realizing DL on constrained SoCs: (i) Training/updating DL models requires continuous processing of the evolving sensor data. Having consistent access to cloud servers can be highly expensive or even infeasible in many real-world settings. (ii) Offloading data to the cloud brings major privacy concerns, particularly as many sensing data can reveal personal and private user information. (iii) Reducing the overhead of DL models enables executing more smart sensing and environment perception applications on a limited computational resource budget imposed by the pertinent platform.

Our key observation is that the dimensionality of input samples to a deep neural network has a direct impact on the overall size of the network, which subsequently dictates resource utilization for training and executing deep models. The resource utilization can be characterized in terms of runtime, power, energy, and memory. As such, we use our proposed resource-aware data transformation approach as a preprocessing step to adaptively map the stream of input data to a corresponding ensemble of lower-dimensional subspaces. The new data representation highlights the most informative portions of the data, shrinking the deep network training and execution workload. As a result, meaningful reductions in power and energy consumptions, memory footprint, training runtime, as well as testing latency are obtained. The proposed data transformation is adaptive and incurs linear computational complexity within a guaranteed approximation error. Our approach leverages the degree of freedom in producing several possible projection subspaces to enable customizing MobiDeep with respect to the underlying platform. We provide a systematic methodology to perform customization as well as projection error tuning to achieve a
target inference accuracy. Note that our data transformation is computed based on a streaming model which evades the requirement to store the original ever growing data matrix and incurs a fixed, low memory footprint.

We present practical design experiences of using MobiDeep for various smart sensing and visual understanding tasks on Nvidia Tegra K1 SoC. Tegra K1 is a state-of-the-art embedded processor, widely used in contemporary IoT enabled cars, robots, and smartphones [21]. To the best of our knowledge, no prior solution for simultaneous training and execution of DL on an embedded system has been reported in the literature. We also design an API that can implement MobiDeep on any pertinent SoC including CPU and/or CPU-GPU platforms for rapid prototyping of an arbitrary DL application by customizing to the underlying hardware. Our API is built to work with Theano, a highly efficient and popular DL library [22]. In our evaluations, we demonstrate the performance efficiency achieved by MobiDeep in comparison with the conventional implementation of the target sensing DL applications using Theano library (on the same platform). Our results corroborate MobiDeep’s practicability and efficiency in realizing various sensing applications ranging from indoor localization to activity recognition. In particular, we show up to 50-fold improvement in runtime, power, energy, and memory in comparison with the best known prior solutions.

1.3 Contribution

The explicit contributions of this thesis are as follows:

- We develop a novel resource-aware data transformation (sketching) algorithm to adaptively map the stream of input data to a corresponding ensemble of lower-dimensional subspaces. Our algorithm benefits from a fixed, low mem-
ory footprint and incurs linear computational complexity within a guaranteed approximation error.

- We propose SSketch, a generic communication-minimizing computing framework for streaming-based analysis of massive dense datasets using reconfigurable platforms. MobiDeep adaptively learns the embedded hybrid structure of the input, inline with the data arrival while providing machine learning practitioners and data scientist with a user-friendly interface for rapid-prototyping of an arbitrary matrix-based data analysis algorithm.

- We introduce the first automated framework, called MobiDeep, for efficient realization of deep learning methods in resource-limited settings. MobiDeep is the first framework that enables local training and execution of deep networks on portable devices, while customizing memory, energy, power, and runtime utilizations.

- We devise a systematic, resource-aware customization methodology that adaptively leverages the degree of freedom in producing several possible projection subspaces and tunes the suggested data transformation with respect to user-defined and platform constraints.

- We design APIs that can implement our proposed efficient computing frameworks on any pertinent SoC components including CPU, GPU, and FPGA. Our proof-of-concept evaluations using a variety of large visual, audio, and smart-sensing datasets corroborate the practicability and efficiency of our customized streaming-based approach in realizing different learning/sensing applications ranging from indoor localization, and activity recognition to image super-resolution and image de-noising.
Chapter 2

Preliminaries and Background

2.1 Streaming Model

The dimensionality of modern data collections renders usage of traditional learning algorithms infeasible. Data transformation methods enable scalable computations on a more compact structure rather than the original data matrix without a significant loss [23]. To minimize the overhead of data transformation, it is critical to devise scalable and pass-efficient algorithms. In pass-efficient algorithms, data is read at most a constant number of times. A streaming-based method refers to a pass-efficient computational model that requires only one pass through the dataset. By taking advantage of a streaming model, the lower-dimensional embedding of a collection can be obtained inline with data arrival. The compact embedding significantly reduces the required memory footprint and makes storing the original data samples unnecessary [24]. In this thesis, we suggest a streaming-based data transformation method as a pre-processing step prior to costly machine learning algorithms such as deep learning, stochastic optimization, and gradient-based learning approaches; we devise our framework with a systematic optimization approach to adaptively customize it with respect to the user-defined and platform constraints.
2.2 Orthogonal Matching Pursuit

OMP is a well-known greedy algorithm for solving sparse approximation problems. It is a key computational kernel in many compressed sensing algorithms. OMP has wide applications ranging from classification to structural health monitoring. As we describe in Algorithm 2, OMP takes a dictionary and a signal as inputs and iteratively approximates the sparse representation of the signal by adding the best fitting element in every iteration. More details regarding the OMP algorithm are presented in Section 3.2.

2.3 Deep Learning

Deep learning refers to a set of machine learning techniques that learn high level abstractions of a data collection through multiple layers of non-linear information processing. In DL, data features in higher layers are constructed from those in preceding ones. As documented by several recent studies, DL approaches have demonstrated superb learning capabilities in a variety of understanding tasks including classification, feature extraction, pattern recognition, and regression [14, 15]. In this thesis, we use deep learning as a generic classification approach for realizing various mobile sensing inferences. Use of deep network architectures simplifies feature extraction with custom codes, enabling users to hand off a lot of those decisions to the DL algorithms.

DL approaches can be partitioned into two main categories: (i) Deep Neural Networks (DNNs), and (ii) Convolutional Neural Networks (CNNs) [15]. These two types of DL networks share many architectural similarities; CNNs are composed of additional convolution layers on top of fully connected DL networks that are used
in DNNs. CNN models rely on local dependency of neighboring samples in a data collection to extract high level abstractions of data. As such, although CNNs have been shown to be quite effective (in term of accuracy) for interpreting visual datasets, they are not directly applicable to various type of smart sensing data such as those appear in autonomous vehicular and robotic applications. In this thesis, we focus on use of DNNs as they are more generic (suitable for various sensing data) and have been shown to be more resource-efficient than their CNN counterparts [25].

DNN is a well-established artificial neural network with multiple hidden layers in between the input and output layers. The number of hidden layers is a parameter predetermined by the user. In a DNN architecture, units in one layer are fully-connected to the units in the adjacent layer while there is no connection within the same layer. The state of each unit is determined in response to the state of the units in the prior layer after applying a non-linear activation function. Commonly used activation functions for hidden layers include logistic sigmoid, Tangent-hyperbolic (Tanh), and Rectified Linear Unit (ReLu). Output layer is an exception for which a softmax regression is typically used to determine the final inference. Softmax regression (or multinomial logistic regression) is a generalization of logistic regression that maps a $P$-dimensional vector of arbitrary real values to a $P$-dimensional vector of real values in the range $(0, 1)$. The final inference for each input sample can be determined by the output unit that has the largest conditional probability value [15].

2.4 Notation

We write vectors in lowercase script, $x$, and matrices in uppercase script, $A$. Let $A^t$ denote the transpose of $A$. $A_j$ represents the $j^{th}$ column and $A_\lambda$ is a subset of matrix $A$ consisting of the columns defined in the set $\lambda$. $\text{nnz}(A)$ defines the number of non-
zeros in the matrix $A$. $\|x\|_p = \left(\sum_{j=1}^{n} |x(j)|^p\right)^{1/p}$ is used as the $p$-norm of a vector where $p \geq 1$. The Frobenius norm of matrix $A$ is defined by $\|A\|_F = \sqrt{\left(\sum_{i,j} |A(i,j)|^2\right)}$.

The input matrix $A$ is of size $m \times n$ where $n$ is the number of samples, $m$ is the corresponding number of features per sample, and $m \ll n$ for over-complete data collections.
Chapter 3

SSketch: Automated Big Data Analysis Using Reconfigurable Platforms

3.1 SSketch Overview

SSketch uses the abundant hardware resources on current FPGAs to provide a scalable, floating-point implementation of OMP for sketching purposes. One may speculate that GPUs may show a better acceleration performance than FPGAs. However, the performance of GPU accelerators is limited in our application because of two main reasons. First, for streaming applications, the memory hierarchy in GPUs increases the overhead in communication and thus reduces the throughput of the whole system. Second, in our sketching approach, the number of required operations to compute the sketch of each individual sample depends on the input data structure and may vary from one sample to the other. Thus, the GPU’s applicability is reduced due to its Single Instruction Multiple Data (SIMD) architecture.

The global flow of SSketch is presented in Fig. 3.1. SSketch takes the stream of a massive, dynamic dataset in the matrix form and adaptively computes/updates a corresponding lower-dimensional embedding of the collection. The constraint-driven customization unit of SSketch takes user-defined properties and hardware limitations as inputs and customizes the framework accordingly for an optimized data projection. The input parameters of this unit may include runtime, power, and memory constraints and its outputs include sketching algorithmic parameters as well as guide-
lines for the hardware mapping. Our sketch formation algorithm is devised to minimize the costly message passings to/from the memory and cores, thereby it reduces the communication delay and energy. All SSketch’s computations are done in IEEE 754 single precision floating-point format.

Figure 3.1: High level block diagram of SSketch. It takes stream of data as input and adaptively learns a corresponding sketch of the collection by doing computation at the level of matrix rank. The resulting sketch is then sent back to the host for further analysis depending on the application.

An earlier version of SSketch has appeared in the 23rd IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM), 2015 [26].

3.2 SSketch Methodology

Many modern massive datasets are either low-rank or lie on a union of lower dimensional subspaces. This convenient property can be leveraged to efficiently map the data to an ensemble of lower dimensional data structures [23]. The authors in [23] suggest a distributed framework based upon a scalable and sparsity-inducing solution
to find the sketch of large and dense datasets such that:

$$\minimize_{D \in \mathbb{R}^{m \times l}, C \in \mathbb{R}^{l \times n}} \| A - DC \|_F \text{ subject to } \| C \|_0 \leq kn,$$

(3.1)

where $A_{m \times n}$ is the input data, $D_{m \times l}$ is the dictionary matrix, $C_{l \times n}$ is the block-sparse matrix, and $l \ll m \ll n$. $\| C \|_0$ measures the total number of non-zeros in $V$, and $k$ is the target sparsity level for each input sample. Their approach, however, is “static” and does not adaptively update the dictionary at runtime. The only way to update is to redo the dictionary computation which would incur a higher cost and is unsuitable for streaming applications with a single pass requirement and limited memory. We develop SSketch based upon a novel extension of [23] for streaming applications. SSketch tailors the solution of (3.1) according to the underlying platform’s constraints. Our approach incurs a lower memory footprint and is well-suited for scenarios where storage is severely limited.

Our data transformation approach is summarized in Algorithm 1. SSketch algorithm, approximates matrix $A$ as a product of two other matrices ($A_{m \times n} \approx D_{m \times l}C_{l \times n}$) based on a streaming model.

For each newly arriving sample, our algorithm first calculates a projection error, $W(A_i)$, based on the current values of the dictionary matrix $D$. Next, it compares the calculated error with a user-defined projection threshold $\alpha$ and updates the sketch accordingly. SSketch locally updates the dictionary matrix $D$ based on each arriving data sample and makes use of the greedy OMP routine to compute the block-sparse matrix $C$. OMP can be used, either by fixing the number of non-zeros in each column of $C$ (sparsity level $k$) or by fixing the total amount of approximation error (error threshold $\epsilon$). Factorizing the input matrix $A$ as a product of two matrices with much fewer non-zeros than the original data, induces an approximation error that can be controlled by tuning the error threshold ($\epsilon$), dictionary size ($l$), and projection
Algorithm 1 SSketch algorithm

Inputs: Measurement matrix A, projection threshold \( \alpha \), sparsity level \( k \), error threshold \( \epsilon \), and dictionary size \( l \).

Output: Matrix D, and coefficient matrix C.

1: \( D \leftarrow \text{empty} \)
2: \( j \leftarrow 0 \)
3: for \( i = 1, \ldots, n \) do
4: \( W(A_i) = \frac{\|D(D^tD)^{-1}D^tA_i - A_i\|_2}{\|A_i\|_2} \)
5: if \( W(A_i) > \alpha \) and \( j < l \) then
6: \( D_j = A_i / \sqrt{\|A_i\|_2} \)
7: \( C_{ij} = \sqrt{\|A_i\|_2} \)
8: \( j \leftarrow j + 1 \)
9: else
10: \( C_i \leftarrow OMP(D, A_i, k, \epsilon) \)
end if
end for

threshold (\( \alpha \)) in SSketch framework. In our experiments, we consider Frobenius norm error (\( \text{Xerr} = \frac{\|A-DC\|_F}{\|A\|_F} \)), as sketch accuracy metric.

As can be seen, SSketch requires only one pass through each arriving sample. This method only requires storing a single column of the input matrix \( A \) and the matrix \( D \) at a time. Note that the dictionary matrix \( D_{m \times l} \) is constructed by columns of data matrix \( A_{m \times n} \). The column space of \( D \) is contained in the column space of \( A \). Thus, \( \text{rank}(DD^+A) = \text{rank}(D) \leq l \leq m \). It simply implies that for over-complete datasets OMP computation is required for \( n - l \) columns and the overhead time of copying \( D \) is ignorable due to its small size compared to \( A \).
OMP with QR Decomposition. As we describe in Section 5.1.4, computational complexity of the projection step (line 4 of Algorithm 1) is small compared to the $O(mn^2)$ complexity of the OMP algorithm. Thus, the computational bottleneck of SSketch algorithm is OMP. To boost the computational performance of SSketch for analyzing a large amount of data on FPGA, it is necessary to modify the OMP algorithm such that it maximally benefits from the available resources and incurs a scalable computational complexity.

Algorithm 2 demonstrates the pseudocode of OMP where $\epsilon$ is the error threshold, and $k$ is the target sparsity level. The Least Squares (LS) minimization step (line 5 of Algorithm 2) involves a variety of operations with complex data flows that introduce an extra hardware complexity. However, proper use of factorization techniques like QR decomposition or Cholesky method within the OMP algorithm would reduce its hardware implementation complexity and make it well-suited for hardware accelerators [10], and [27].

**Algorithm 2 OMP algorithm**

Inputs: Matrix $D$, measurement $A_i$, sparsity level $k$, threshold error $\epsilon$. 

Output: Coefficient vector $c$.

1: $r \leftarrow A_i$
2: $\Lambda_0 \leftarrow \emptyset$
3: for $i = 1,...,k$ do
4: \hspace{2em} $\Lambda \leftarrow \Lambda \cup \text{argmax}_j | < r^{i-1} , D_j > |$ \hspace{2em} Find best fitting column
5: \hspace{2em} $c^i \leftarrow \text{argmin}_c \| r^{i-1} - D_{\Lambda} c \|^2_2$ \hspace{2em} LS Optimization
6: \hspace{2em} $r_i \leftarrow r^{i-1} - D_{\Lambda} c^i$ \hspace{2em} Residual Update
end for
Thus, to efficiently solve the LS optimization problem in line 5 of Algorithm 2, we decide to use QR decomposition (Algorithm 3). QR decomposition returns an orthogonal matrix $Q$ and an upper-triangular matrix $R$. It iteratively updates the decomposition by reusing the $Q$ and $R$ matrices from the last OMP iteration. In this approach, the residual (line 6 of Algorithm 2) can be updated by $r^i \leftarrow r^{i-1}Q^t(Q^t)^t r^{i-1}$.

The final solution is calculated by performing back substitution to solve the inversion of the matrix $R$ in $v^k = R^{-1}Q^t A_i$.

Algorithm 3 Incremental QR decomposition by modified Gram-Schmidt

Inputs: New column $D_{A^s}$, last iteration $Q^{s-1}$, $R^{s-1}$.

Output: $Q^s$ and $R^s$.

1: 

$$R^s \leftarrow \begin{pmatrix} R^{s-1} & 0 \\ 0 & 0 \end{pmatrix}$$

2: $\xi^s \leftarrow D_{A^s}$

3: for $j = 1, \ldots, s-1$ do

4: $R^s_{js} \leftarrow (Q^s_{j-1})_j^H \xi^s$

5: $\xi^s \leftarrow \xi^s - R^s_{js} Q^s_{j-1}$

end for

6: $R^s_{ss} \leftarrow \sqrt{\|\xi^s\|_2^2}$

7: $Q^s \leftarrow [Q^{s-1}, \frac{\xi^s}{R^s_{ss}}]$

Assuming that matrix $A$ is of size $m \times n$ and $D$ is of size $m \times l$, then the complexity of the OMPQR is $O(mnl^2)$. This complexity is linear in terms of $m$ and $n$ since in many settings $l$ is much smaller in compared to $m$ and $n$. This linear complexity enables SSketch to readily scale up for processing a large amount of data based on a
streaming model.

### 3.2.1 Blocking SSketch

Let \( A = [ A_1; A_2; A_3 ] \) be a matrix consisting of rows \( A_1, A_2, \) and \( A_3 \) that are stacked on the top of one another. Our key observation is that if we obtain the sketch of each block independently and combine the resulting sketches (blocking SSketch) as illustrated in Fig. 3.2, then the combined sketch can be as good as sketching \( A \) directly (nonblocking SSketch) in terms of error-performance trade-off. This property can be generalized to any number of partitions of \( A \). We leverage this convenient property to increase the performance of our proposed framework for sketching massive datasets based on a streaming model. In blocking SSketch, the data matrix \( A \) is divided into more manageable sized blocks such that there exist enough block RAMs on FPGA to store the corresponding \( D \) and a single column of that block. The blocking SSketch achieves a significant bandwidth saving, faster load/store, less communication traffic between kernels, and a fixed memory requirement on FPGA. The methodology also provides the capability of factorizing massive, dense datasets in an online streaming model.

Independent analysis of each block is especially attractive if the data is distributed across multiple machines. In such settings, each platform can independently compute a local sketch. These sketches can then be combined to obtain the sketch of the original collection. Given a fixed memory budget for the matrix \( D \), as it is presented in Section 5.1, blocking SSketch results in a more accurate approximation compared with nonblocking SSketch. The blocking SSketch computations are done on smaller segments of data which confers a higher system performance. The achieved higher accuracy is at the cost of a larger number of non-zero elements in \( C \). Note that as our evaluation
results imply, designers can reduce number of non-zeros in the computed block-sparse matrix by increasing the error threshold $\epsilon$ in SSketch algorithm.

![Figure 3.2: Schematic depiction of blocking SSketch.](image)

### 3.2.2 Theoretical Bound on SSketch Approximation Error

In SSketch methodology, the dictionary matrix $D$ is constructed such that the column space of $D$ is contained in the column space of the data matrix $A$. To bound the reconstruction error using SSketch, we propose Theorem 1.

**Theorem 1** In blocking SSketch, the reconstruction error of a massive, dynamic input data $A$ is

$$\|A - DC\|_F^2 \leq \max(\alpha, \epsilon).$$

**Proof**: Let $a_{uj}$ represent the $u^{th}$ segment of the $j^{th}$ column of input data matrix $A$. In blocking SSketch, for each newly arriving sample, if $W(a_{uj}) \geq \alpha$, then it is added to the corresponding sub-block in dictionary matrix $D$, see Fig. 3.2. The reconstruction error for the added $a_{uj}$ is exactly zero via SSketch’s methodology. For the remaining part of the input data, the greedy OMP routine is used to compute sparse approximation of the sample.

In our methodology, when the dictionary size $l$ is set to be large enough (i.e. $l = mb$ where $mb$ is the block-size), then the set of $l$ samples that are linearly independent will span the ambient dimension of the corresponding data block $\mathbb{R}^{mb}$,
which results in exact decomposition, i.e., \( \| \mathbf{a}_{uj} - \mathbf{D}_u \mathbf{D}_u^+ \mathbf{a}_{uj} \|_F = 0 \). Note that OMP routine does not stop unless either the reconstruction error (\( \frac{\| \mathbf{a}_{uj} - \mathbf{D}_u \mathbf{c}_{uj} \|_F}{\| \mathbf{a}_{uj} \|_F} \)) reaches a value less than or equal to \( \epsilon \), or unless all the column samples in \( \mathbf{D}_u \) are used for reconstructing \( \mathbf{a}_{uj} \), where \( \mathbf{D}_u \) is the corresponding dictionary sub-block for the input segments \( \mathbf{a}_u \). In the latter case, the normalized reconstruction error is less than or equal to \( \alpha \) according to the SSketch algorithm. Therefore, for each \( \mathbf{a}_{uj} \) we have:

\[
\frac{\| \mathbf{a}_{uj} - \mathbf{D}_u \mathbf{c}_{uj} \|_F^2}{\| \mathbf{a}_{uj} \|_F^2} \leq \max(\alpha, \epsilon)
\]

\[
\| \mathbf{a}_{uj} - \mathbf{D}_u \mathbf{c}_{uj} \|_F^2 \leq \max(\alpha, \epsilon) \| \mathbf{a}_{uj} \|_F^2.
\] (3.2)

Summing up (3.2) over all blocks for an input sample \( \mathbf{a}_j \) results in:

\[
\sum_{u=1}^{n} \| \mathbf{a}_{uj} - \mathbf{D}_u \mathbf{c}_{uj} \|_F^2 \leq \sum_{u=1}^{n} \max(\alpha, \epsilon) \| \mathbf{a}_{uj} \|_F^2
\]

\[
\| \mathbf{a}_j - \mathbf{Dc}_j \|_F^2 \leq \max(\alpha, \epsilon) \| \mathbf{a}_j \|_F^2.
\] (3.3)

Equation (3.3) is a result of the blocking structure of the dictionary matrix \( \mathbf{D} \) (Fig. 3.2). Finally, the overall reconstruction error can be presented by summing up (3.3) over all \( n \) input samples.

\[
\sum_{j=1}^{n} \| \mathbf{a}_j - \mathbf{Dc}_j \|_F^2 \leq \sum_{j=1}^{n} \max(\alpha, \epsilon) \| \mathbf{a}_j \|_F^2
\]

\[
\| \mathbf{A} - \mathbf{DC} \|_F^2 \leq \max(\alpha, \epsilon) \| \mathbf{A} \|_F^2
\] (3.4)

### 3.3 SSketch Automated Hardware-accelerated Implementation

In this section, we discuss the details of SSketch hardware-accelerated implementation. After applying preprocessing steps on the stream of the input data for dictionary learning, SSketch sends the data to FPGA through a 1Gbps Ethernet port. SSketch
is devised with multiple OMP kernels and a control unit to efficiently compute the block-sparse matrix $C$. As the stream of data arrives, the control unit looks for availability of OMP kernels and assigns the newly arriving sample to an idle kernel for further processing. The control unit also has the responsibility of reading out the outputs and sending back the results to the host. SSketch API provides designers with a user-friendly interface for rapid prototyping of arbitrary matrix-based data analysis algorithms and realizing streaming applications on FPGAs, see Fig. 3.3. The constraint-driven customization unit of SSketch takes user-defined properties and hardware limitations as inputs and customizes the framework for an optimized sketch computation. The input parameters of the constraint-driven customization unit may include runtime, power, and memory constraints ($T_u$, $P_u$, and $M_u$ respectively) and its outputs include sketching algorithmic parameters as well as guidelines for hardware mapping. Users can then use the transformed data to scalably perform an arbitrary matrix-based data analysis on an ensemble of lower dimensional structures rather than the original matrix without a significant loss. Note that the algorithmic parameters of SSketch including the projection threshold $\alpha$, error threshold $\epsilon$, dictionary size $l$, target block-sparsity level $k$, and block-size $m_b$, can also be easily changed through the SSketch API.

In OMP hardware implementation, we utilize several techniques to reduce the iteration interval of two successive operations and exploit the parallelism within the algorithm. We observe that the OMP algorithm includes multiple dot product computations which result in frequent appearance of for-loops requiring an operation similar to $a += b[i] \times c[i]$.

We use a tree-based reduction module by implementing a tree-based adder to accelerate the dot product and norm computation steps that appear frequently in the
Figure 3.3: High level diagram of SSketch API. The constraint-driven customization unit of SSketch takes user-defined properties and hardware limitations as inputs and delivers output parameters that can be used for an optimized sketch computation.

OMP routine. By means of the reduction module, SSketch is able to reduce the iteration interval and handle more operations simultaneously. As such, SSketch requires multiple concurrent loads and stores from a particular RAM. To cope with the concurrency, instead of having a large block RAM for matrices D and Q, we use multiple smaller sized block memories and fill these block RAMs by cyclic interleaving. Thus, we can perform a faster computation by accessing multiple successive elements of the matrices and removing the dependency in the for-loops.

Using the block RAM is desirable in FPGA implementations because of its fast access time. The number of block RAMs on one FPGA is limited, so it is important to reduce the amount of utilized block memories. We reduce block RAM utilization in our realization by a factor of 2 compared to the naive implementation. This reduction
is a consequence of our observation that none of the columns of matrix $D$ would be selected twice during one call of the OMP algorithm. Thus, for computing line 4 of Algorithm 2 we only use the indices of $D$ that are not selected during the previous OMP iterations. We instead use the memory space that was originally assigned to the selected columns of $D$ to store the matrix $Q$. By doing so we reduce the block RAM utilization, which allows SSKetch to employ more OMP kernels in parallel.

### 3.4 Automated SSKetch Customization

SSSketch framework is devised with an automated constraint-driven customization unit to deliver best data transformation under a given set of user/platform constraints, see Fig. 3.4.

![Figure 3.4: Overview of automated SSKetch customization.](image)

#### 3.4.1 Motivation

SSSketch automatically customizes itself according to different sets of user/platform constraints to achieve a resource-efficient computation of the sketch matrix in streaming applications. As we demonstrate in Section 5.1, there is a trade-off between the
sketch matrix accuracy and SSketch’s performance which can be carefully leveraged to improve the efficiency of sketch matrix computation.

By increasing the dictionary size $l$, SSketch can better capture the hybrid structure of the input data, which results in a higher sparsity level in the block-sparse matrix $C$ and typically higher sketch accuracy. Note that in SSketch framework each dictionary sub-block is constructed from the columns of its corresponding data sub-matrix, see Fig. 3.2. The column space of the dictionary is contained in that of the data matrix which implies that the rank of each dictionary sub-block is less than $m_b$, where $m_b$ is the block-size. Therefore, in blocking SSketch approach, each dictionary sub-block can at most consist of $m_b$ independent samples. This automatically ensures that the memory constraint is satisfied. However, in practice one might need to customize the architecture such that not only memory constraint is addressed but also runtime and power constraints are taken into consideration. SSketch takes all user/platform runtime, power, and memory constraints into account and subsequently tunes its algorithmic parameters to deliver the sketch matrix with least approximation error in each scenario.

3.4.2 Constraint-Driven Optimization

Memory constraint on computing platforms is one of the main limitations in big data regime. Blocking SSketch computes the sketch of dynamic data collections by breaking up the data into more manageable blocks according to the memory budget. In SSketch framework, the memory requirement can be approximated by $((lm_b + m_b) \times (n_k + 1) + n_k l^2) \times 4$ bytes, where $n_k$ is the number of OMP kernels, $m_b$ is the block-size, and $l$ is the number of samples in the dictionary matrix.
Total delay of SSketch ($T_{SSketch}$) can be expressed as:

$$T_{SSketch} \approx T_{\text{dictionary learning}} + T_{\text{Communication Overhead}} + T_{\text{FPGA Computation}}$$

$$\approx \beta_0 mn + \beta_1 mn(l + l^2) + \beta_2 \frac{mn(kl + k^2)}{n_k},$$

where $\beta_i$s are constant coefficients that characterize the runtime requirement per unit of floating point operation. The latter term in (3.5) represents the runtime cost of computing the block-sparse matrix $C$, which is the dominant factor in SSketch's total delay. Fig. 3.5 demonstrates the average runtime of SSketch as a function of sparsity level ($k$), dictionary size ($l$), and number of OMP kernels ($n_k$). As Fig. 3.5 illustrates, the average processing time for a sample in SSketch framework is independent of the total number of samples. This convenient property enables SSketch to readily scale up for analyzing any number of input samples.

According to the OMP algorithm, for each newly arriving sample we expect ($m(kl + k^2)$) operations for updating the block-sparse matrix $C$. As it is illustrated in Fig. 3.5a, and 3.5b our hardware-accelerated implementation of SSketch follows the same trend as predicted by the OMP algorithm complexity. Note that none of the columns of matrix $D$ would be selected twice during one call of the OMP algorithm.

In our hardware-accelerated realization of SSketch, we compute line 4 of Algorithm 2 by searching among those columns of the dictionary that have not been selected during the previous OMP iterations. Our hardware implementation approach results in a milder slope for larger $k$s in Fig. 3.5a. For example this property can be seen for $100 \leq k \leq 128$ versus $80 \leq k \leq 100$ in Fig. 3.5a. In Fig. 3.5c the average runtime of SSketch is illustrated as a function of $n_k$, where $n_k$ is the number of OMP kernels. The kernels are used in parallel to compute the block-sparse matrix $C$. As Fig. 3.5c
Figure 3.5: SSketch’s FPGA-accelerated implementation performance. \( l, k, \epsilon, \) and \( m_b \) are SSketch’s algorithmic parameters which indicate the number of samples in the dictionary matrix, target sparsity level, error threshold, and block-size respectively. \( n_k \) denotes the number of OMP kernels that work in parallel in data sketching unit.

In both figures (b), and (c) we set \( k = l \) to let \( k \) be as large as the dictionary size and use \( \epsilon \) as the stopping criteria in the OMP algorithm.

demonstrates, the average runtime of SSketch is proportional to \( 1/n_k \), which experimentally confirms that the runtime of SSketch \( (T_{SSketch}) \) is dominated by the runtime cost of computing the block-sparse matrix \( C \).

Table 3.1 reflects the total power consumption of our hardware-accelerated implementation of SSketch on Xilinx Virtex-6-XC6VLX240T FPGA ML605 Evaluation Kit. As can be seen in Table 3.1, the total power consumption of SSketch is a linear function of the number of OMP kernels that are employed in data sketching unit.

To deliver the most accurate sketch matrix, SSketch solves an optimization as described in (3.6) and maximally exploit the existing sparsity in a dataset to effectively improve the iterative matrix computations performance while it considers a set of user/platform constraints. The SSketch’s constraint-driven optimization can be
Table 3.1: Power consumption on Virtex 6 for different number of OMP kernels.

<table>
<thead>
<tr>
<th>Number of Kernels ($n_k$)</th>
<th>Power Consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$0.358w$</td>
</tr>
<tr>
<td>2</td>
<td>$0.447w$</td>
</tr>
<tr>
<td>3</td>
<td>$0.546w$</td>
</tr>
<tr>
<td>4</td>
<td>$0.634w$</td>
</tr>
</tbody>
</table>

expressed as:

$$\begin{align*}
\text{minimize} & \quad \text{approximation error}, \\
\text{subject to:} & \quad l \leq m_b, \\
& \quad m_b \leq m, \\
& \quad n_k \in \mathbb{N}, \\
& \quad \beta 2mn^2/n_k \leq T_u, \\
& \quad P_{SSketch} \leq P_u, \\
& \quad (l + 1)(n_k + 1)m_b + n_kl^2 \leq M_u,
\end{align*}$$

where $T_u$, $P_u$, $M_u$ are a set of user defined parameters which imply the user/platform constraints in terms of runtime, power, and memory respectively.

SSketch approximates the solution of (3.6) using the Karush-Kuhn-Tucker (KKT) conditions. To efficiently capture the hybrid structure of streaming data collections, SSketch automatically customizes its framework according to the application and tunes the algorithmic parameters including block-size $m_b$, dictionary size $l$, as well as the number of OMP kernels $n_k$. To facilitate automation, we provide a solver
for our optimization approach. The solver gets the constraints from the user as inputs and uses our Mathematica-based computational software program to solve the optimization. Note that this constraint-driven optimization is a one-time process that incurs a constant, and negligible overhead regardless of data size.
Chapter 4

Automated Deep Learning on Embedded Systems

4.1 MobiDeep Overview

MobiDeep exploits the well-known trade-off between the output solution variations and system performance to provide an efficient approach for realizing various sensing and visual understanding tasks on resource-constrained devices. It leverages the great learning capabilities of deep networks, while customizing resource utilizations in terms of power, energy, and memory corresponding to the underlying platform. To do so, as shown in Figure 4.1 MobiDeep consists of two main phases: (i) Training, and (ii) Execution.

The training phase of our framework includes two major steps: (i) Automated projection customization (Section 4.2), and (ii) Training a DL-based classifier using the projected data (Section 4.3). Our customization is a scalable approach that tailors the subsequent learning task corresponding to user and platform constraints such that the inference error is minimized. Once the DL model is learned, it can be used for classifying newly arriving samples (execution phase). Note that the execution phase only includes a forward pass through the DL model while tuning the network parameters requires multiple rounds of forward and backward propagation (Section 4.3). We also provide an accompanying API to facilitate automation and adoption of MobiDeep for rapid prototyping of various visual and smart sensing tasks on SoC platforms, e.g., CPU-GPU.
Figure 4.1: High level block diagram of MobiDeep. The raw data is projected to a lower-dimensional embedding using a new customizable method. The new method is resource-aware and is tailored to benefit the subsequent learning task.

4.2 Automated Projection Customization

The dimensionality of modern data collections renders usage of traditional machine learning algorithms infeasible. This makes designing customized data projection methods a necessity to enable scalable computations on a more compact structure rather than the original data without a significant loss. MobiDeep’s data projection customization is a pre-processing step. It adaptively transforms the input data to a lower-dimensional embedding such that the subsequent iterative training of the DL model using the projected data becomes much more efficient in terms of runtime, memory footprint, energy, and power compared with the conventional scenario where the raw data is fed into the deep network. Our key observation is that leveraging data geometry can significantly facilitate deep learning by (i) increasing the convergence rate, and (ii) reducing the overall DL network size in terms of the required number of units per layer. As such, MobiDeep adaptively factorizes the original input matrix $A_{m \times n}$ into a dictionary matrix $D_{m \times l}$ and a coefficient matrix $C_{l \times n}$, where $m$ is the
feature space size of the raw input samples and \( n \) is the total number of measurements. In MobiDeep, the dictionary matrix \( \mathbf{D} \) is adaptively learned from the stream of input data and the block-sparse coefficient matrix \( \mathbf{C} \) is computed using the greedy OMP routine.

MobiDeep is devised with an automated customization methodology to adaptively tunes the projection parameters (e.g., the dictionary size \( l \), and sparsity level (or the number of non-zero elements per column of \( \mathbf{C} \)) \( k \)) and customize MobiDeep framework corresponding to the underlying data structure and platform constraints.

### 4.2.1 Memory Customization

The dictionary size indicated by \( l \) is equal to the feature space size of the coefficient matrix \( \mathbf{C} \) (the input layer size of the deep network) and has a direct effect on the memory storage and overall DL network size. The memory footprint of the DL is defined by \((m \times l + l \times n + \text{size(param)})\); \( l \), \( m \), and \( n \) are data-specific variables while \( \text{size(param)} \) depends on the DNN topology (e.g., number of edges and units per layer). MobiDeep adaptively tunes its dictionary size \( l \) with respect to the input feature space size \( m \), and the available memory bandwidth. Note that the dictionary matrix \( \mathbf{D}_{m \times l} \) is constructed from carefully sampling columns of the data matrix \( \mathbf{A}_{m \times n} \). Thus, the column space of \( \mathbf{D} \) is contained in the column space of \( \mathbf{A} \), which in turn implies that \( \text{rank}(\mathbf{D}\mathbf{D}^+\mathbf{A}) = \text{rank}(\mathbf{D}) \leq l \leq m \). Here, \( \mathbf{D}^+ \) denotes the pseudo-inverse of matrix \( \mathbf{D} \). This guarantees that the dictionary matrix \( \mathbf{D} \) at most consists of \( m \) samples unless \( l \) is set to be less than \( m \) to satisfy the memory constraints on the target platform.
4.2.2 Error Customization

Another key observation is that changing the sparsity level $k$ has a significant impact on the ultimate inference accuracy as well as pre-processing overhead, making it necessary to tune $k$ corresponding to the application and available resources. It has been shown that many contemporary large data collections usually can be modeled by a union of lower-dimensional subspaces [23]. Let $A$ be a data with a union of subspace signal model and $A_s$ be a random subset of $A$ such that $|A_s| = n_0$, where $|.|$ denotes the cardinality or number of columns in a data matrix. As we experimentally verify in Section 5.1, $A_s$ closely models the existing underlying lower-dimensional hybrid structure of $A$ as $n_0 \to n$. We build up our customization approach based upon this convenient property to optimize the performance cost with respect to different scenarios. Our evaluations show that MobiDeep can effectively approximate the optimal sparsity level $k$ with respect to the underlying constraints by applying Alg. 4 on a small subset of the original raw data.

4.2.3 Runtime Customization

Transformation overhead of each newly arriving sample is a deterministic function of dictionary size $l$, sparsity level $k$, and the input feature space size $m$. MobiDeep takes the user-defined runtime budget into considerations and tunes the algorithmic parameters accordingly. The runtime budget could be dictated either by the arriving rate of sensor measurements or the buffer size for storing incoming samples in the target platform.
4.3 DL-based Classifier

After pre-processing the incoming samples, MobiDeep trains a DNN-based model using the customized low-dimensional embedding of the data computed as explained in Section 4.2. The goal in training a DNN is to learn the weights and biases between layers such that a loss function is minimized. In our evaluations, we consider the $L_2$ norm difference between network inferences and the ground-truth labeled data as our loss function. Training a DL model usually requires multiple paths through the whole dataset and consists of two main steps: (i) forward propagation, and (ii) backward propagation. In the forward propagation step, the model’s response is computed based on the current values of the network parameters per Eq. (4.1):

$$a_i^{(s+1)} = f\left(\sum_{j=1}^{n(s)} W_{ij}^{(s)} a_j^{(s)} + b_i^{(s)}\right),$$

where $a_i^{(s)}$ is the state of unit $i$ in layer $s$ and $f(\cdot)$ denotes the activation function. For $s = 1$, $a_i^{(1)}$ is equivalent to the $i^{th}$ input feature. $W_{ij}^{(s)}$ specifies the weight associated with the connection between unit $j$ in layer $s$ and unit $i$ in layer $s + 1$, and $b_i^{(s)}$ indicates the bias associated with unit $i$ in layer $s + 1$. In the backward propagation step, a gradient descent based algorithm is applied to adjust (fine-tune) network parameters. The training procedure continues until a local optima is reached via the deep network. Figure 4.2 illustrates a typical DNN topology.

In particular, our framework adopts two different flows for training and executing deep learning in realizing various sensing and understanding tasks, which we further explain in the following:
4.3.1 Training Phase

Alg. 4 provides the pseudocode of our proposed framework for deployment of an arbitrarily DL application. As can be seen in Alg. 4, MobiDeep requires only one pass through each arriving sample to update the coefficient matrix $C$. Note that MobiDeep’s pre-processing step can be highly parallelized as the representation of each arriving sample can be independently computed.

In Alg. 4, $n_{\text{epoch}}$ is a user-defined variable that controls the frequency of updating the classifier input based on the arriving rate of training samples. After every $n_{\text{epoch}}$ new training measurements, MobiDeep updates the acquired DL-based classifier through multiple rounds of forward and backward propagation using the expanded matrix $C$ that contains both the new and previously computed coefficient vectors. The inputs of $DNN$ module include projected training samples $C$, the corresponding label...
for each training sample $L_T$, current network parameters $\text{param} = (\text{weights, biases})$, and the number of units per layer $\text{layer size}$. $\text{layer size}$ is a vector of integers whose first component should be equal to $l$ (number of features in the block-sparse matrix $C$), and its last component indicates number of classes according to the application. Each number in between the first and last components of the $\text{layer size}$ indicates the number of units per hidden layer in the chosen DNN topology. At the end of training process, network parameters (e.g., weights and biases) are updated corresponding to the entire input training samples.

### 4.3.2 Execution Phase

Once the classifier is trained using Alg. 4, there are two main steps to predict the class label for each test measurement (Alg. 5). In the first step, each test sample is projected based on the learned dictionary matrix $D$. Then in the second step, the corresponding coefficient vector $C_i$ is fed into the trained DL-based classifier to obtain the class label. The execution phase only requires a forward propagation (Eq. 4.1) for each transformed data sample; thereby the classification delay (testing latency) is proportional to the number of units per layer ($\text{layer size}$).
Algorithm 4: MobiDeep (Training Phase)

**Inputs:** Measurement matrix (A), Transformation parameters (\(\alpha, k, l\)), Sample size (\(n_{\text{epoch}}\)), Training labels (\(L_T\)), and Model parameters (\(layer_{\text{size}}\))

**Output:** Dictionary matrix \(D\), coefficient matrix \(C\), and Classifier parameters \(param\).

1: \(D \leftarrow \text{empty} \)

2: \(param \leftarrow \text{empty} \)

3: \(j \leftarrow 0 \)

4: \(i \leftarrow 0 \)

5: while (true) do

6: \(W_p(a_i) = \frac{\|D(D'^TD)^{-1}D'a_i-a_i\|_2}{\|a_i\|_2} \)

7: if \(W_p(a_i) > \alpha \) and \(j < l\) then

8: \(D_j = a_i/\sqrt{\|a_i\|_2} \)

9: \(c_{ij} = \sqrt{\|a_i\|_2} \)

10: \(j \leftarrow j + 1 \)

11: else

12: \(c_i \leftarrow OMP(D, a_i, k) \)

end if

13: \(i \leftarrow i + 1 \)

14: if \(i \mod n_{\text{epoch}} == 0\) then

15: \(param \leftarrow DNN(C^t, param, layer_{\text{size}}, L_T) \)

end if

end while
Algorithm 5: MobiDeep (Execution Phase)

Inputs: Measurement matrix ($A^{test}$), Dictionary matrix $D$, Sparsity level $k$, and Model parameters ($param$)

Output: Class label vector $y$

1: $i ← 0$
2: while (true) do
3: $C_i ← OMP(D, A_i^{test}, k)$
4: $y_i ← DNN_{forward}(C_i^t, param)$
5: $i ← i + 1$
end while
Chapter 5

Evaluations and Practical Design Experiences

5.1 SSketch Evaluations

For evaluation purposes, we apply our SSketch’s methodology on three sets of data: (i) Light field, (ii) Hyperspectral images, and (iii) Synthetic data. To ensure that dictionary learning is independent of the data order, for each fixed set of algorithmic parameters we shuffle the data before each calling of SSketch algorithm. The mean error value for 10 calls of SSketch algorithm and its variance are reported in Fig. 5.1, and 5.2. In all cases, we observe that the variance of the error value is two to three orders of magnitude less than the mean value, implying that SSketch algorithm has a low dependency on the data arrival order. This convenient property of SSketch algorithm is promising for adaptive dictionary learning and sketching purposes in streaming applications.

5.1.1 Light Field

A light field image is a set of multi-dimensional array of images that are simultaneously captured from slightly different viewpoints. Promising capabilities of light field imaging include the ability to define the field’s depth, focus or refocus on a part of image, and reconstruct a 3D model of the scene [28]. For evaluating SSketch algorithm accuracy, we run our experiments on a light field data consisting of 2500 samples each of which constructed of 25 $8 \times 8$ patches. The light field data results in a data matrix
with 4 million non-zero elements. We choose this moderate input matrix size to accommodate the SVD algorithm for comparison purposes and enable the exact error measurement especially for correlation matrix (a.k.a., Gram matrix) approximation. The Gram matrix of a data collection consists of the Hamiltonian inner products of data vectors. The core of several important data analysis algorithms is the iterative computation on the data Gram matrix. Examples of Gram matrix usage include but are not limited to kernel-based learning and classification methods, as well as several regression and regularized least square routines [29].

In SSketch framework, the dictionary size $l$ has a direct effect on the achieved approximation error as well as system speed. Transformation with larger $l$ results in a smaller approximation error at the cost of decreasing system performance in terms of runtime due to the increase in computation. Fig. 5.1a and 5.1d report the results of applying both nonblocking and blocking SSketch on the data. We define the approximation error for the input data and its corresponding Gram matrix as $X_{\text{err}} = \frac{\|A - \tilde{A}\|_F}{\|A\|_F}$, and $G_{\text{err}} = \frac{\|A^T A - \tilde{A}^T \tilde{A}\|_F}{\|A^T A\|_F}$, where $\tilde{A} = D V$.

Given a fixed memory budget for the matrix $D$, Fig. 5.1a and 5.1d illustrate that the blocking SSketch results in a more accurate sketch compared with the nonblocking approach. Number of rows in each block of input data (block-size) has a direct effect on SSketch’s performance. Fig. 5.1b, 5.1e, and 5.1c demonstrate the effect of block-size on the data and Gram matrix approximation error as well as matrix compression-rate, where the compression-rate is defined as $\frac{\text{nnz}(D) + \text{nnz}(C)}{\text{nnz}(A)}$. In this setting, the higher accuracy of blocking SSketch is at the cost of a larger number of non-zeros in the block-sparse matrix $C$. As illustrated in Fig. 5.1b, 5.1e, and 5.1c designers can easily reduce the number of non-zeros in the block-sparse matrix $C$ by increasing the SSketch error threshold $\epsilon$. In SSketch algorithm, there is a trade-off
between the sketch accuracy and the number of non-zeros in the block-sparse matrix \( C \). The optimal performance of SSketch methodology is determined based on the error threshold and the user/platform constraints.

Considering the spectral norm error \( (\epsilon_k = \frac{\|A - \tilde{A}\|_2}{\|A\|_2}) \) instead of Frobenius norm error, the theoretical minimal error can be expressed as \( \sigma_{k+1} = \min(\|A - A_k\|_2 : A_k \text{ has rank } k) \), where \( \sigma_k \) is the exact \( k^{th} \) singular value of \( A \) and \( A_k \) is obtained by SVD [30]. Fig. 5.1f compares \( \epsilon_k \) and the theoretical minimal error for the light field dataset.

5.1.2 Hyperspectral

A hyperspectral image is a sequence of images generated by hundreds of detectors that capture the information from across the electromagnetic spectrum. With this collected information, one would obtain a spectrum signature for each pixel of the image that can be used for identifying or detecting the material [31]. Hyperspectral imaging is a new type of high-dimensional image data and a promising tool for applications in earth-based and planetary exploration, geo-sensing, and beyond. This fast and non-destructive technique provides a large amount of spectral and spatial information on a wide number of samples. However, the large size of hyperspectral datasets limits the applicability of this technique, especially for scenarios where online evaluation of a large number of samples is required.

We adapt SSketch framework to capture the sketch of each pixel for the purpose of enhancing the computational performance and reducing the total storage requirement for hyperspectral images. In this experiment, the SSketch algorithm is applied on two different hyperspectral datasets. The first dataset [32] is \( 148 \times 691614 \) and the second one [33] is of size \( 204 \times 54129 \). Our experiments show that SSketch algorithm results
(a) $X_{err}$ vs. $l$ with $\alpha = 0.1$, $m_b = 200$ and $\epsilon = 0.01$.  
(b) $X_{err}$ vs. $m_b$ with $\alpha = 0.1$ and $l=128$.  
(c) Compression-rate vs. $m_b$ with $\alpha = 0.1$ and $l=128$.  
(d) $G_{err}$ vs. $l$ with $\alpha = 0.1$, $m_b=200$ and $\epsilon=0.01$.  
(e) $G_{err}$ vs. $m_b$ with $\alpha = 0.1$  
(f) $e_k$ vs. $l$ compared to the minimal possible error.

Figure 5.1: Experimental evaluations of SSketch. $\alpha$, $\epsilon$, $l$, and $m_b$ are SSketch’s algorithmic parameters which indicate the projection threshold, error threshold, number of samples in the dictionary matrix, and block-size respectively. $X_{err}$ represents the data matrix sketching error and $G_{err}$ represents the approximation error for the corresponding Gram matrix. The spectral transformation error is denoted by $e_k$.

in the same trend as in 5.1a and 5.1d for both hyperspectral datasets. As Fig. 5.2 illustrates, the Gram matrix approximation error reaches to less than $0.2 \times 10^{-6}$ for $l \geq 10$ in both datasets.
Figure 5.2: Approximation error (\(Gerr\)) vs. \(l\) for two different hyperspectral datasets. The SSketch algorithmic parameters are set to \(\alpha = 0.1\) and \(\epsilon = 0.01\), where \(\alpha\) is the projection threshold and \(\epsilon\) is the error threshold.

5.1.3 SSketch’s Hardware Customized Experiments

We provide three experiments to demonstrate the advantage of SSketch’s automated tuning approach for user/platform specific customization and performance optimization. In all these experiments, we compare the approximation results with the actual optimal values. As can be seen, our approximations for SSketch algorithmic parameters are very close to the real optimized values which implies the accuracy of SSketch automated constraint-driven customization. Our experiment platform is a Virtex-6-XC6VLX240T FPGA ML605 Evaluation Kit with 1872kB available block RAM memory [34].

Experiment (i): Here we consider a situation where a user requires computing the sketch matrix of a dynamic data collection with a size of \(m = 256\), and \(n = 5000\) within a runtime budget of 30s. In this case, the memory and power are only limited
by the target platform; in other words there are no user constraints in terms of memory or power. Since no power constraint exists, \( n_k \) (the number of OMP kernels) is set at its maximum value, which is determined by the memory budget. As it is shown in Table 5.2, 4 OMP kernels fit on the Virtex-6-XC6VLX240T FPGA with 1872kB of available block RAM memory. Fig. 5.3 demonstrates the average runtime for a sample sketch update as a function of dictionary size in our real-world evaluation on FPGA. As Fig. 5.3a illustrates, \( l = 107 \) is the optimal dictionary size one can use to compute the data sketch within the aforementioned set of user/platform constraints. On the other hand, solving our constraint-driven optimization in (3.6) results in \( l \approx 98 \) (star-marked point in Fig. 5.3a). Thus, our parametric optimization outputs efficiently model the real-world hardware-accelerated implementation.

Experiment (ii): Here we consider a situation where a user requires computing the sketch matrix of a dynamic data collection with a size of \( m = 256 \), and \( n = 5000 \) within a runtime budget of 25s and a power budget of 0.6 Watts. In this case, the memory is limited by the target platform. Due to the power constraint, one can make use of 3 OMP kernels to compute the sketch (Table 3.1). As Fig. 5.3b illustrates, \( l = 80 \) is the optimal dictionary size one can use to compute the data sketch within the aforementioned set of user/platform constraints. On the other hand, solving our constraint-driven optimization in (3.6) results in \( l \approx 78 \) (star-marked point in Fig. 5.3b).

Experiment (iii): Here we consider a situation where a user requires computing the sketch matrix of a dynamic data collection with a size of \( m = 512 \), and \( n = 2500 \) within a runtime budget of 91s and a power budget of 0.4 Watts. In this case, the memory is limited to the 1872kB block RAM memory available within our experiment platform. Due to the power constraint, only 1 OMP kernel can be used to
compute the sketch matrix (Table 3.1). As Fig. 5.3c illustrates, \( l = 95 \) is the optimal dictionary size that can be used to compute the data sketch within the aforementioned set of user/platform constraints. On the other hand, solving our constraint-driven optimization in (3.6) results in \( l \approx 85 \) (star-marked point in Fig. 5.3c).

The above experiments further demonstrate the applicability and efficiency of our constraint-driven optimization approach proposed in this Section 3.4.

Figure 5.3: SSketch’s automated constraint-driven customization. Each plot demonstrates the average runtime required to process one sample as a function of the dictionary size. The dashed horizontal line reflects the user runtime deadline and the star point illustrates the SSketch’s automated customization output. The user power constraint selects the number of OMP kernels that can work in parallel. The crossing point of the dashed line and the corresponding runtime curve (solid curve) is the actual optimized point in each setting.

5.1.4 Scalability

We provide a comparison between the complexity of different implementations of the OMP routine in Fig. 5.4. Batch OMP (BOMP) is a variation of OMP that is espe-
cially optimized for sparse-coding of large sets of samples over the same dictionary. BOMP requires more memory space compared with the conventional OMP, since it needs to store $D^T D$ along with the matrix $D$. BOMPQR and the OMPQR both have near linear computational complexity. We use OMPQR method in our target architecture as it is more memory-efficient.

![Figure 5.4: Computational complexity comparison of different OMP implementations. Using QR decomposition significantly improves OMP’s runtime.](image)

The complexity of our OMP algorithm is linear both in terms of $m$ and $n$, so dividing $A_{m \times n}$ into several blocks along the dimension of $m$ and processing each block independently does not add to the total computational complexity of the algorithm. However, it shrinks the data size to fit into the FPGA block RAMs and improves the sketching performance.

Let $T_{OMP}(m, l, k)$ stand for the number of operations required to obtain the sketch of a vector of length $m$ with target sparsity level $k$. Then the runtime of the system is a linear function of $T_{OMP}$ which makes the proposed architecture scalable for factorizing large matrices. The complexity of projection step in SSketch algorithm (line 4 of
Algorithm 1) is \((l^3 + 2lm + l^2m)\). However, if we decompose \(D_{m \times l}\) to \(Q_{m \times m} \times R_{m \times l}\) and replace \(DD^+\) with \(QI_tQ^t\), then the projection step’s computational complexity would be reduced to \((2lm + l^2m)\). Assuming \(D = QR\) then the projection matrix can be written as:

\[
D(D'D)^{-1}D^t = QR(R'R'QR)^{-1}R'Q'^t
\]

\[
= Q(RR^{-1})(R'^{-1}R'^t)Q'^t = QI_tQ'^t, \quad (5.1)
\]

which we use to decrease the projection step’s complexity.

Table 5.1 compares different sketching methods with respect to their complexity. The SSketch’s complexity indicates a linear relationship with \(n\) and \(m\). In SSketch, computations can be parallelized as the sparse representation can be independently computed for each column of the sub-blocks of \(A\).

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVD</td>
<td>(m^2n + m^3)</td>
</tr>
<tr>
<td>SPCA</td>
<td>(lmm + m^2n + m^3)</td>
</tr>
<tr>
<td>SSketch (this work)</td>
<td>(n(lm + l^2m) + mnl^2 \approx 2mnl^2)</td>
</tr>
</tbody>
</table>

5.1.5 SSketch’s Hardware Settings and Results

We use Xilinx Virtex-6-XC6VLX240T FPGA ML605 Evaluation Kit as our hardware platform. An Intel core i7-2600K processor with SSE4 architecture running on the Windows OS is utilized as our general purpose processing unit hosting the FPGA. In this work, we employ Xilinx standard IP cores for single precision floating-point
operations. We used Xilinx ISE 14.6 to synthesize, place, route, and program the FPGA.

We leverage Eigen library to implement the dictionary learning and CPU implementation of SSketch (used for comparison purposes) on the general-purpose Intel processor. The Eigen library exploits Intel Stream SIMD Extension (SSE) instructions on Intel processors to enhance the performance of intensive matrix computations. Using Eigen library pushes the performance of our software implementations to the limits of the general purpose processor.

Table 5.2 shows Virtex-6 resource utilization for our heterogeneous architecture. SSketch includes 4 OMP kernels plus an Ethernet interface. For factorizing matrix $A_{m \times n}$, there is no specific limitation on the size $n$ due to the streaming nature of SSketch. However, the FPGA block RAM size is limited. To fit into the RAM, we decide to divide input matrix $A$ to blocks of size $m_b \times n$ where $m_b$ and $k$ are set to be less than 256. Note that these parameters are changeable in SSketch API. So, if a designer decides to choose a higher $m_b$ or $k$ for any reasons, she can easily modify the parameters according to the application.

<table>
<thead>
<tr>
<th></th>
<th>Used</th>
<th>Available</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slice Registers</td>
<td>50888</td>
<td>301440</td>
<td>16%</td>
</tr>
<tr>
<td>Slice LUTs</td>
<td>81585</td>
<td>150720</td>
<td>54%</td>
</tr>
<tr>
<td>RAM B36E1</td>
<td>382</td>
<td>416</td>
<td>91%</td>
</tr>
<tr>
<td>DSP 48E1s</td>
<td>356</td>
<td>768</td>
<td>46%</td>
</tr>
</tbody>
</table>
To corroborate the scalability and practicability of our framework, we use synthetic data with dense (non-sparse) correlations of different sizes as well as a hyperspectral image dataset [33]. The runtime of SSketch for the different-sized synthetic datasets is reported in Table 5.3, where the total delay of SSketch ($T_{SSketch}$) is defined in (3.5).

As it is shown in Table 5.3, the total delay of SSketch is a linear function of the number of processed samples, which experimentally confirms the scalability of our proposed architecture. According to Table 5.3, the whole system including the dictionary learning process takes 21.029s to process 5K samples where each of them has 256 elements. 4872 of these samples pass through OMP kernels and each of them requires 89 iterations on average to complete the process. In this experiment, an average throughput of 169Mbps is achieved by SSketch.

For both hyperspectral data [33] of size $204 \times 54129$ and synthetic dense data, our HW/SW co-design approach (with target sparsity level ($k$) of 128) achieves up to 200 folds speedup compared to the software-only realization on a 3.40 GHz CPU. The average overhead delay for communicating between the processor (host) and accelerator contributes less than 4% to the total delay.

Table 5.3 : SSketch total processing time is linear in terms of the number of processed samples.

<table>
<thead>
<tr>
<th>Size of n</th>
<th>Timing ($l = 128$)</th>
<th>Timing ($l = 64$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n = 1K$</td>
<td>$3.635s$</td>
<td>$2.31s$</td>
</tr>
<tr>
<td>$n = 5K$</td>
<td>$21.029s$</td>
<td>$12.01s$</td>
</tr>
<tr>
<td>$n = 10K$</td>
<td>$43.446s$</td>
<td>$24.32s$</td>
</tr>
<tr>
<td>$n = 20K$</td>
<td>$90.761s$</td>
<td>$48.52s$</td>
</tr>
</tbody>
</table>
5.2 MobiDeep Evaluations

We provide an accompanying API for our implementation of Algorithms 4 and 5. To boost the inference accuracy and avoid over-fitting, we employ dropout technique [35] in our DL implementation. Stochastic Gradient Descent (SGD) with momentum [36] is used for back-propagation. We adopt Theano library in our DL implementation [22]. MobiDeep’s API can be used to implement our framework on any pertinent SoC that includes CPU or CPU/GPU according to the user configuration. The user-defined algorithmic inputs of our API include: stream of raw data $A$, training labels $L_T$, sample size $n_{epoch}$, projection threshold $\alpha$, and model parameters including $layer_{size}$, activation function, and the batch size $b_s$ for SGD. In our evaluations we used $Tanh$ as our activation function for each hidden layer, and $b_s = 100$ for SGD. The projection error $\alpha$ is set to be 0.1 (10%) throughout our experiments.

All our design experiments are carried out on a constrained embedded system named Jetson TK1. The Jetson development kit from NVIDIA is a full-featured platform for realizing computer vision, robotics, security, automotive, and mobile sensing embedded applications [21]. It includes 192 CUDA cores and 4-Plus-1 quad-core ARM Cortex A15 CPU with 2 GB memory. In our evaluations, data transformation is performed by standard Message Passing Interface (MPI) system using the 4 quad-core ARM processors and training and executing DL models have been conducted on the available CUDA cores.

5.2.1 Applications

To corroborate the practicability of MobiDeep, we apply our proposed methodology to three different contemporary sensing applications: (i) daily and sport activities recognition [37], (ii) indoor localization [38], and (iii) speech recognition [39]. Table
5.4 specifies the size of the dataset used for each application. Unless otherwise specified in the text, we used 70% of each data collection for training, 10% for validation, and the remaining 20% for testing our acquired model.

Table 5.4: Size of evaluation datasets.

<table>
<thead>
<tr>
<th>Daily &amp; Sport Activities Recognition [37]</th>
<th>Indoor Localization [38]</th>
<th>Speech Recognition [39]</th>
</tr>
</thead>
<tbody>
<tr>
<td>5625 × 9120</td>
<td>520 × 21048</td>
<td>617 × 7797</td>
</tr>
<tr>
<td>410.4 MB</td>
<td>87.5 MB</td>
<td>38.5 MB</td>
</tr>
</tbody>
</table>

**Daily and Sports Activities Recognition.** Daily and Sports Activities (DSA) dataset is collected using 5 body-worn miniature inertial units with 9 sensors on each unit (x, y, z accelerometers, x, y, z gyroscopes, and x, y, z magnetometers) [37]. The dataset covers a wide range of daily and sport activities including: sitting (A1), standing (A2), lying on the back and on the right side (A3 and A4), ascending and descending stairs (A5 and A6), standing still in an elevator (A7) and moving around in an elevator (A8), walking in a parking lot (A9), walking on a treadmill with a speed of 4 km/h in flat and 15° inclined positions (A10 and A11), running on a treadmill with a speed of 8 km/h (A12), exercising on a stepper (A13), exercising on a cross trainer (A14), cycling on an exercise bike in horizontal and vertical positions (A15 and A16), rowing (A17), jumping (A18) and playing basketball (A19). Eight volunteer subjects (4 female, and 4 male) between ages of 20 and 30 are asked to perform the aforementioned 19 activities in their own style. Total signal duration is 5 minutes for each activity of each subject. Sensor units are calibrated to acquire data at 25Hz sampling frequency. The 5 minutes signals are divided into 5 seconds segments so that 480 (60 × 8) signal segments are obtained for each activity. The data matrix in
this experiment consists of 51 million non-zero elements (5625 × 9120). The goal of training DNN in this experiment is to classify each vector of sensing measurement to the correct activity class.

**Indoor Localization.** “UJIIndoorLoc” is a dataset created for indoor localization applications using 25 Android devices [38]. It covers 3 buildings with 4 or more floors and almost 110,000 m². This dataset can be used for classification (e.g., building and floor identification) or regression (e.g., longitude and latitude estimation). In our experiments, we use this dataset for classification purposes. The dataset contains the WLAN fingerprint (a.k.a., WiFi fingerprint) of 19937 training/reference records and 1111 validation/test measurements. During the dataset creation, 520 different Wireless Access Points (WAPs) were detected. Thus, the WiFi fingerprint is composed of 520 intensity values. The intensity values are represented as negative integer values ranging $-104$ \text{dbm} (extremely poor signal) to 0 \text{dbm}. The positive value 100 is used when a WAP was not detected. The goal of training DNN in this experiment is to identify the actual indoor location based on the WiFi fingerprints.

**Speech Recognition.** Isolet Spoken Letter (ISL) dataset contains the English alphabet letters spoken in isolation under laboratory conditions and associated transcripts [39]. The data consists of two productions of each letter by 150 speakers (75 female, and 75 male) for approximately 1.25 hours of speech. All speakers reported English as their native language. Their ages varied from 14 to 72 years and the average age was 35 years. The goal of training DNN in this experiment is to map each measurement to the correct corresponding phoneme.

To make our experiments robust to the arriving order of the samples, all the aforementioned data collections are completely shuffled before applying Alg. 4.
5.2.2 Pre-processing and Customization

Here, we experimentally verify our claim in Section 4.2 that MobiDeep can adaptively use subsets of data for automated customization. One of the key tunable parameters that characterizes MobiDeep’s performance in terms of inference accuracy and resource utilization is the number of non-zeros in each column of the coefficient matrix $C$(sparsity level $k$). Figure 5.5a illustrates validation error as a function of sparsity level $k$ for different subsets of the DSA dataset. Two observations are evident. First, there is an optimal $k$ that minimizes the inference error for a given input data and selected deep network topology. As demonstrated, MobiDeep can achieve a reasonable estimation of the efficient sparsity level using only a small subset of the data for customization. Second, a lower classification error is achieved when a larger number of samples are used for training. This convenient property enables MobiDeep to make its acquired DL model more accurate over time as the new training samples arrive.

In this experiment, the dictionary size $l$ is 456 and the chosen DL model is a 4-layer DNN of size $(456 \times 500 \times 100 \times 19)$. In Figure 5.5a, $|C_{sub}|$ denotes the number of sample measurements in the selected subset of the data where $|C_{Total}|$ is the total number of samples.

Figures 5.5b and 5.5c show validation error as a function of sparsity level $k$ for different subsets of the indoor localization and ISL datasets respectively. For indoor localization, the dictionary size $l$ is 150 and the chosen DL network is a 3-layer DNN in which each hidden layer has 100 units and the output layer includes 13 classes. For ISL data, the dictionary size is set to be 200 and the selected DL topology is a 3-layer DNN in which the number of units per hidden layer is equal to 50 and the output layer has 26 inferences. As shown, MobiDeep can achieve a reasonable estimation of the effective sparsity level $k$ by using only a small portion of the training samples.
Figure 5.5: MobiDeep’s pre-processing customization for daily and sport activities recognition. The optimal sparsity level $k$ that minimizes the inference error can be found by using only a small portion of training samples.

Table 5.5 shows MobiDeep’s total pre-processing time overhead, which accounts for both tuning the algorithmic parameters using the proposed automated customization approach, and transforming the data. The tuning is done on a small subset of data that contains only 5% of each dataset. Once the tuning is completed, the transformation is performed using the customized parameters $l$ and $k$. As we discuss later, the one-time pre-processing overhead of MobiDeep is amortized when the iterative DL algorithms are run on the transformed data. Let $T_{OMP}(m, l, k)$ stand for the number of operations required to project an input vector of length $m$ with the target sparsity level $k$ using a dictionary matrix of size $m \times l$. The transformation overhead is a linear function of $T_{OMP}$. This linear overhead makes the proposed architecture scalable for pre-processing large amounts of streaming input data. Note that the pre-processing computations of MobiDeep can be parallelized as the sparse representation can be independently computed for each new incoming sample.
Table 5.5: Pre-processing overhead (seconds) of MobiDeep which accounts for both tuning and data transformation.

<table>
<thead>
<tr>
<th></th>
<th>Daily and Sport Activities</th>
<th>Indoor Localization</th>
<th>Speech Recognition</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5625 × 9120</td>
<td>520 × 21048</td>
<td>617 × 7797</td>
</tr>
<tr>
<td>Tuning</td>
<td>91.4</td>
<td>9.0</td>
<td>31.1</td>
</tr>
<tr>
<td>Transformation</td>
<td>11.9</td>
<td>2.5</td>
<td>4.7</td>
</tr>
<tr>
<td>Overall</td>
<td>103.3</td>
<td>11.5</td>
<td>35.8</td>
</tr>
</tbody>
</table>

5.2.3 Performance Evaluation

To evaluate MobiDeep performance in presence of noisy samples, we perform three experiments using DSA dataset with (i) no additive noise, (ii) Signal to Noise Ratio (SNR) = 10, and (iii) SNR = 15. In these experiments, we use $k = 19$, and $l = 456$ to transform the data (Figure 5.5a), while each original input sample has 5625 features. Figure 5.6 shows test error over time for three different input SNRs. As shown in this figure, MobiDeep is not sensitive to noisy inputs (less than 2% variation in final inference error) which makes it a suitable choice for development of sensing inferences in presence of noisy sensor measurements.

The graphs in Figure 5.6 illustrate that our data transformation greatly speeds up the DL training process, while it maintains a competitive inference accuracy. This increase in the convergence rate is mainly due to the block-sparse structure of matrix $C$ that facilitates deep learning by effectively representing data as an ensemble of lower-dimensional subspaces. As the dictionary evolves over time, it better captures dynamic structure of data, and subsequently increases the error reduction rate in MobiDeep. In addition, our data transformation also enables us to achieve the same
Figure 5.6: Inference error over time for different input SNRs. The sparsity level for transformed data is set to be $k = 19$, and the dictionary size is $l = 456$. The reported runtime correspond to MobiDeep execution on the GPU of Jetson TK1 board.

level of error (e.g., 10% in Figure 5.6a) by utilizing smaller number of units per layer compared to the conventional scenario where the original (not transformed) data is fed to the deep network. This reduction in the overall DL network size translates to a lessening in resource utilization as well as training and executing (testing) latency.

Table 5.6 elaborates the practicability and effectiveness of our approach in local training and execution of DL models for processing visual, audio, and smart sensing datasets. Note that no prior systematic DL-based approach has been reported in the literature that is simultaneously applicable to different types of data collections and well-suited for constrained SoCs. Thus, for comparison purposes we implement the conventional DL approach as a baseline where the raw data is fed into the deep network with no pre-processing. In particular, we report performance improvement over conventional deep learning in terms of training and execution latency, memory footprint, energy consumption, as well as the reduction achieved in the overall DL network size as a result of using MobiDeep’s adaptive customization approach. Note that the power consumption is a direct function of energy and runtime.
To achieve 10% error in classifying the DSA dataset, MobiDeep reduces the training runtime ($10.1\times$) compared with the conventional approach and uses ($14.8\times$) less memory. In the same experiment, we also achieve significant reduction in energy consumption ($48.4\times$) as well as the DL execution latency ($43.7\times$) by scaling down the amount of required units per layer. For DSA data, the conventional approach meets 10% classification error using a $(5625 \times 2000 \times 500 \times 19)$ network architecture while MobiDeep reaches the same classification error by training a $(456 \times 500 \times 100 \times 19)$ network (Figure 5.6).

To obtain 5% inference error in the indoor localization application, MobiDeep achieves ($2.7\times$) saving in training runtime and ($3.6\times$) in the total memory footprint. Besides, MobiDeep also gains ($5.6\times$) and ($19.5\times$) improvement in energy consumption and the execution latency, respectively. The different scale of improvement for indoor localization compared with the DSA application is due to the smaller size of feature space for this dataset. As expected, our methodology results in more improvement when applied to datasets with larger number of input features. In this experiment, we use $k = 20$, and $l = 150$ to transform the data (Figure 5.5b), while each input sample originally has 520 features.

For the ISL dataset, MobiDeep achieves 5% classification error with ($2.2\times$) less training runtime and using ($3.1\times$) less memory. In the same application, ($5.5\times$), and ($2.8\times$) reduction in energy consumption and test time latency is obtained respectively. In this application, we use $k = 50$, and $l = 200$ to transform the data (Figure 5.5c), while each original input sample has 617 features.
Table 5.6: Performance improvement achieved by MobiDeep over conventional deep learning approach.

<table>
<thead>
<tr>
<th>Application</th>
<th>Memory Improvement</th>
<th>Energy Improvement</th>
<th>Training Latency Improvement</th>
<th>Execution Latency Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Daily and sport Activities</td>
<td>14.8×</td>
<td>48.4×</td>
<td>10.1×</td>
<td>43.7×</td>
</tr>
<tr>
<td>Indoor Localization</td>
<td>3.6×</td>
<td>5.6×</td>
<td>2.7×</td>
<td>19.5×</td>
</tr>
<tr>
<td>Speech Recognition</td>
<td>3.1×</td>
<td>5.5×</td>
<td>2.2×</td>
<td>2.8×</td>
</tr>
</tbody>
</table>

5.2.4 Discussion

Our evaluations demonstrate the importance of MobiDeep’s pre-processing step to reduce the cost of expensive training and execution process of deep networks. As our results show, training DL models even with a moderate size dataset can be computationally extremely expensive. For instance, to achieve 10% inference error, it takes 1 hour and 51 minutes to train a deep network for the DSA dataset without pre-processing using the GPU of a NVIDIA Jetson development kit (Figure 5.6a). However, MobiDeep’s pre-processing reduces this time to less than only 12 minutes on the same platform (Figure 5.6a). The one-time pre-processing overhead for this dataset is less than 1.7 minutes (Table 5.5). This significant improvement also translates to resource utilization reduction and longer battery life on portable devices. Note that each application demands its own model selection. This means for classification using DL models one requires to perform multiple rounds of training with varying parameters including number of hidden layers, number of units per layer, batch size, activation function (e.g., ReLU, Sigmoid, or Tanh), etc., until the best ones are identified. The complex and iterative nature of the DL methods further
amortizes the transformation tuning cost as a pre-processing step.

One may speculate that traditional data transformation methods such as Principle Component Analysis (PCA), or Singular Value Decomposition (SVD) can replace our pre-processing step and result in comparable performance with MobiDeep. However, during our evaluations we observed poor performance when PCA is used as the primary dimensionality reduction technique before training the DNN. For instance, a test error of 70.23% is obtained with a DNN of size \((456 \times 500 \times 100 \times 19)\) when the DSA data is transformed by the PCA method (MobiDeep achieves 10% classification error with the same DNN configuration). Beside the poor performance in terms of accuracy, there are three main limitations with PCA or SVD. First, the computational complexity of such methods is \(O(m^2n)\) which makes them costly choices for transforming large datasets. Second, these data transformation techniques are not well-suitable for streaming applications where the dataset dynamically evolves over time. Third, such methods are oblivious to the coarse-grained parallelism that exists in the datasets. To be more specific, unlike our transformation approach, methods such as PCA or SVD reduce data dimensionality according to the rank of datasets and do not take into accounts the union of low-rank properties [23].

Auto-encoders are considered as an alternative dimensionality reduction technique that learn the low-rank representation of a dataset within multiple levels of a deep network [40]. Although auto-encoders have been shown to be effective in learning low-rank representation of different datasets, their practicability is limited in mobile inference applications due to two main reasons. First, using auto-encoders for dimensionality reduction adds to the number of parameters that should be learned within DNN training, thereby increasing training runtime, power consumption, and number of required FLOP during test-time. Second, unlike our transformation ap-
proach, auto-encoders are not pass-efficient alternatives for dimensionality reduction in streaming applications, especially in scenarios where memory storage is severely limited. To be more specific, learning the corresponding parameters of auto-encoders requires multiple passes through the whole dataset. As a result, the entire feature sets of training samples should be stored during training, which makes this approach memory inefficient and unsuitable for resource-constrained devices.
Chapter 6

Related Work

6.1 Streaming-based Data Sketching Using Reconfigurable Platforms

Developing support for streaming data is critical in many emerging applications where real time response is required [41]. Several recent studies have focused on system modeling and design techniques to facilitate streaming applications by exploiting task and data level parallelism, for example [42], [41], and [43]. However, none of the prior work have leveraged the data geometry to further accommodate streaming applications. To the best of our knowledge, MobiDeep1 is the first automated framework that proposes a generic online data transformation that enables scalable big data analysis in streaming applications.

It is known that the most accurate low-rank approximation of a data collection is computed by SVD or PCA in settings where the column span of the data admits a lower dimensional embedding [44]. However, the large memory footprint and $\mathcal{O}(m^2n)$ computational complexity of these well-known sketching algorithms make it impractical to use them for analyzing massive, and dynamic datasets. Unlike PCA, Sparse PCA (SPCA) is modified to find principal components with sparse loadings, which is desirable for interpreting data and storage reduction [45]. The computational complexity of SPCA is similar to classic SVD. Thus, even this method is not scalable for analyzing massive datasets [45], and [46].
The efficiency of random subsampling methods to compute the lower dimensional embedding of large datasets has been shown in [47], and [48]. Random Column Subset Selection (rCSS) has been proposed as a scalable strategy for sketching large matrices [47]. Although, authors in [47] had provided a theoretical scalable approach for large matrix sketching, but the hardware constraints are not considered in this work. The large memory footprint and non-adaptive structure of their rCSS approach make it unsuitable for streaming applications.

Recently, a distributed framework has been proposed in [23] based upon a scalable and sparsity-inducing data transformation algorithm. The framework enables efficient execution of large-scale iterative learning algorithms on massive and dense datasets. MobiDeep framework is developed based upon a novel extension of the proposed sketching method in [23]. Unlike the work in [23], our data sketching approach is well-suited for streaming applications and is amenable to FPGA acceleration.

OMP has been shown to be very effective in inducing sparsity, although its complexity makes it costly for streaming applications. A number of implementations on GPU [8], [49], and [50], ASICs [9], and FPGAs [11], [10], and [51] are reported in the literature to speed up this complex reconstruction algorithm. FPGA implementation of OMP for problems of dimension $32 \times 128$ and $256 \times 1024$ are developed for signals with sparseness of 5 and 36 respectively [11], and [10]. To the best of our knowledge, none of the previous implementations of OMP are devised for streaming applications with large and densely correlated data matrices. In addition, use of fixed-point format to compute and store the results limits their applicability for sketching purposes.
6.2 Deep Learning on Resource-Constrained Platforms

The existing approaches for using DL in IoT enabled devices, either operate entirely off-device in the cloud [14] or heavily rely on using pre-trained DL networks developed for particular tasks in an off-line manner [52, 53]. Thereby, they cannot locally adjust their DL model corresponding to the evolving data constantly acquired by many sensors available on today’s IoT platforms such as AVs, robots, and smartphones. Recent works [53, 25] provide measurement studies that analyze the performance of common DL methods such as convolutional neural networks and multi-layer perceptrons on a System on Chip (SoC) used for IoTs. While these works provide insights on resource utilization of DL, they do not provide any mechanism to improve the system’s performance. More importantly, none of the earlier works addresses the problem of resource efficient customization for training DL networks that is known to be more computationally challenging than DL execution. To the best of our knowledge, MobiDeep is the first automated framework that enables users to locally prototype (train and execute) different smart sensing and environment perception applications using deep models while working under a set of power, energy, and memory constraints. Our approach also incurs a lower execution latency compared with the conventional DL models by leveraging data geometry using our new customizable data projection.

Several recent works have focused on using auto-encoders [40] or classic dimensionality reduction approaches for feature extraction to facilitate common classification methods such as nearest neighbor, or support vector machine [54, 47]. However, no prior work has suggested the use of data projection approaches as a way to achieve resource efficiency for training and execution of DL models in resource-limited settings. MobiDeep’s customizable data projection, for the first time, proposes a systematic approach to reduce the data and circuit footprint of DL models and make
them amenable to be implemented on constrained platforms. Unlike traditional classification models (e.g., Gaussian Mixture Model (GMM), and nearest neighbor) DL approaches are capable of scaling gracefully to large number of inference categories. For instance, a GMM-based classifier computes probability score for each class represented by the entire GMM so that an inference with $p$ added categories (classes) is $p \times$ more computationally expensive than one with a single class. Nonetheless, the execution latency of a DL-based classifier is mostly dominated by the propagation from input and multiple hidden layers which are invariant to the number of classes in the output layer. In addition, when the number of input samples increases, methods such as nearest neighbor become computationally prohibitive during execution, while the test-time latency using a DL model is oblivious to the number of input samples and only depends on the chosen DL topology.
Chapter 7

Summary

This thesis proposes an automated end-to-end solution for efficient streaming-based analysis of big and densely correlated data matrices in resource-limited settings. We suggest adoption of a new communication-minimizing methodology to efficiently capture the sketch of a data collection inline with the data arrival. The new method adaptively learns and leverages the hybrid structure of the streaming input data to effectively improve the performance. We propose a constraint-driven optimization method which automatically tunes the sketch matrix computation and embedding properties with respect to user-defined and hardware specifications. To enable handling datasets with arbitrary large number of features, a blocking model is devised that efficiently partitions the feature space for sketch matrix computing. We provide theoretical bounds for the sketch approximation error in the blocking mode.

We further extend our solution by introducing the first resource-efficient computing framework for realization of on-chip training and execution of deep neural networks in deployment of various autonomous sensing applications. Our approach adaptively reduces the data and circuit footprint required for implementing DL which translate to meaningful memory, power, energy, and runtime savings. We compare the performance of our proposed framework in terms of memory, and power consumption, as well as executing (testing) and training latency against the conventional deep learning approaches. Our experiments demonstrate up to 50-fold improvement compared with the best known prior solutions, none of which were implemented on a
mobile platform due to the demanding computational overhead associated with DL methods.

To boost the computational efficiency, we exploit the capabilities of SoC platforms available on current IoT enabled devices such as GPUs and FPGAs. Our accompanying user-friendly APIs enable machine learning practitioners and designers to easily adopt our proposed approach for rapid prototyping and evaluation of an iterative arbitrary matrix-based big data analysis algorithm as well as training and execution of DL models on any pertinent SoC platforms including CPU, CPU-GPU, and FPGA.
Bibliography


automated framework for streaming sketch-based analysis of big data on fpga,” 2015.


