**BLINDOPHILE**

**MINOR PROJECT REPORT**

Submitted in partial fulfilment of the requirements for the award of the degree

*of*

**BACHELOR OF TECHNOLOGY**

in

**COMPUTER SCIENCE & ENGINEERING**

by

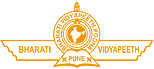
**Udit Kumar Ishita Agarwal Rachit Jain  Ruchika Chugh**

03851202716 40151202716               40551202716 40651202716

**Guided by**

**Ms. Shilpa Gupta**

**Asst. Professor**



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**BHARATI VIDYAPEETH’S COLLEGE OF ENGINEERING**

**PASCHIM VIHAR, NEW DELHI-110063**

**AFFILIATED TO**

**GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY, DELHI**

**NOVEMBER, 2019**

**CANDIDATE’S DECLARATION**

It is hereby certified that the work which is being presented in the B. Tech Minor Project Report entitled “**BLINDOPHILE**” in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology** and submitted in the **Department of Computer Science & Engineering of BHARATI VIDYAPEETH'S COLLEGE OF ENGINEERING, New Delhi (Affiliated to Guru Gobind Singh Indraprastha University, Delhi)** is an authentic record of our own work carried out during the period from **August 2019 to November 2019** under the guidance of **Ms. Shilpa Gupta, Assistant Professor**.

The matter presented in the B. Tech Minor Project Report has not been submitted by us for the award of any other degree of this or any other Institute.

**Udit Kumar Ishita Agarwal Rachit Jain  Ruchika Chugh**

03851202716 40151202716               40551202716 40651202716

This is to certify that the above statement made by the candidate is correct to the best of my knowledge. They are permitted to appear in the External Minor Project Examination.

**Ms. Shilpa Cupta                                                                            Prof. (Dr.) Kirti Gupta**

**Assistant Professor                                                                         Head, CSE Dept.**

The B. Tech Minor Project Viva Voice Examination of **Udit Kumar (03851202716), Ishita Agarwal (40151202716), Rachit Jain (40551202716)** and **Ruchika Chugh (40651202716)** has been held on             2020.

                                                                                      (Signature of External Examiner)

(Project Coordinator)       (Project Coordinator)

**ABSTRACT**

People with visual impairments face enormous restrictions in terms of their mobility and in today’s world there is lack of infrastructure to make it easier. Our objective is to propose an efficient solution to the above problem by using Internet of Things and Machine Learning algorithms. The principal concept in various applications of IoT reveals that the IoT device is not intelligent by itself. It merely senses its environment and sends the sensor readings to the cloud where all the intelligence resides and the decision making takes place. We envision a paradigm where even tiny, resource-constrained IoT devices can run machine learning algorithms locally without necessarily connecting to the cloud. Recent advances in machine learning prognosticate that it is possible to train models for classical supervised learning problems with memory requirements less than other modern ML algorithms. Eventually, the trained models can be loaded onto IoT devices/sensors to carry out fast and accurate real time analysis. So we intend to develop a device/module called “Blindophile” which will act as a helping hand to the visually impaired person(s) by enabling easy access to smartphones and other home devices. A key technical contribution of this work will also be the development of a robust real-time gesture recognition system based on ML models that can be deployed on tiny microcontrollers.

**ACKNOWLEDGEMENT**

We express our deep gratitude to **Ms. Shilpa Gupta**, Asst. Professor, Department of Computer Science & Engineering for her valuable guidance and suggestions throughout our project work. We are thankful to **Mr. Mohit Tiwari** and **Mr. Harsh Taneja**, Project Coordinators, for their valuable guidance.

We would like to extend our sincere thanks to **Head of the Department, Prof. (Dr.) Kirti Gupta** for her timely suggestions to complete our project work. We are also thankful to **Principal, Dr. Dharmender Saini** for providing us with the facilities to carry out our project work.

**Udit Kumar Ishita Agarwal Rachit Jain  Ruchika Chugh**

03851202716 40151202716               40551202716 40651202716

**TABLE OF CONTENTS**

**CANDIDATE’S DECLARATION ii**

**ABSTRACT iii**

**ACKNOWLEDGEMENT iv**

**TABLE OF CONTENTS v**

**LIST OF FIGURES vii**

**LIST OF TABLES viii**

**LIST OF ABBREVIATIONS ix**

**CHAPTER 1: INTRODUCTION 1 - 2**

**1.1 OBJECTIVE 1**

**1.2 SCOPE 2**

**CHAPTER 2: LITERATURE REVIEW 3 - 16**

**2.1 RELATED PREVIOUS FINDINGS 3**

**2.1.1 CLOUD AND EDGE DEVICES 3**

**2.1.2 FASTGRNN AND RNN 9**

**2.1.3 KNN & PROTONN 10**

**2.2 PREDICTION 12**

**2.3 CLASSIFICATION 13**

**2.4 PROPOSED FEATURES 14**

**2.5 TRAINING MODEL 15**

**2.6 PREDICTION PIPELINE 15**

**CHAPTER 3: METHODOLOGY 17 - 27**

**3.1 MODEL 17**

**3.1.1 KNN ALGORITHM 17**

**3.1.2 PROTONN ALGORITHM 19**

**3.1.3 BONSAI ALGORITHM 20**

**3.2 DATA ACQUISITON 21**

**3.3 MODEL SELECTION 23**

**3.4 IMPLEMENTATION 25**

**CHAPTER 4: RESULTS AND DISCUSSION 28 - 30**

**4.1 ANALYSIS 28**

**4.2 FUTURE SCOPE 30**

**REFERENCES 31**

**APPENDIX 33**

**LIST OF FIGURES**

Figure 2.1 Interconnection of IoT with Network……………………………….…………………….6

Figure 2.2 Inferencing with MNIST data………………………………………………………………….6

Figure 2.3 Image of Sensor……………………………………………………...…………………………….7

Figure 2.4 Self-Learning Device………………………………………………………………………………7

Figure 2.5 Communication of Device with Cloud……………………………………………………8

Figure 2.6 Application Logic in Cloud……………………………………………………………………..8

Figure 2.7 Data Featurization……………………………………………………………………………….11

Figure 2.8 Model Training…………………………………………………………………………………….14

Figure 2.9 Machine Learning Inference flowchart…………………………………………………15

Figure 3.1 Analysis of blindophile dataset with respect to acceleration………………..16

Figure 3.2 Analysis of blindophile with respect to gyration…………………………………..22

Figure 3.3 Dataset of IMU sensor to recognise double tap gesture………………………22

Figure 3.4 Accuracy of kNN model with model size………………………………………………23

Figure 3.5 Accuracy of ProtoNN model with model size……………………………………….24

Figure 3.6 Flowchart for training and deploying the device………………………………….25

Figure 4.1 List and symbols of different Machine learning models……………………….26

Figure 4.2 Analysis of MNIST dataset with different models…………………………………28

Figure 4.3 Analysis of USPS dataset with different models……………………………………29

Figure 4.4 Analysis of MNIST-10 dataset with different models…………………………...29

Figure 4.5 Analysis of Character Recognition dataset with different models…………29

**LIST OF TABLES**

Table 1 Dataset Statistics and Links……………………………………………………………….12

**LIST OF ABBREVIATIONS**

KNN                   K-nearest Neighbours classifier

ML                     Machine Learning

IoT                     Internet of Things

ProtoNN            Prototype based K-nearest neighbours (KNN) classifier

DL                     Deep Learning

RL                     Reinforcement Learning

SNC                  Support Network Classifier

BNC                  Bayesian Network Classifier

NN-prune          Pruning Neural Networks

IMU Inertial Measurement Unit

**CHAPTER 1**

**INTRODUCTION**

**1.1     OBJECTIVE**

We envision to create a model called “Blindophile” (meaning Blind lover) that can ameliorate the experience of visually impaired people. We find it prudent to follow a two-pronged to accomplish the same. Firstly, we will try to infuse intelligence into an IoT device by leveraging a multitude of KB-sized machine learning algorithms that can work on resource-constrained edge and endpoint IoT devices like Arduino and Raspberry Pi or even better alternatives like MKR microprocessors. Then different sensors and modules would be employed to develop gesture based solutions for visually impaired people by incorporating the above concept. Using machine learning and other signal processing algorithms, different off-the-shelf sensors can be combined into a synthetic sensor. These types of sensors are capable of detecting complex events. These sensors have lower cost and more energy efficient comparison to camera based systems. Devices can make continuous improvements after they are deployed in the field. These individual improvements are aggregated on a central service and every device is then updated with the combined result. Neural networks can be partitioned such that some layers are evaluated on the device and the rest in the cloud. This enables the balancing of workload and latency. The initial layers of a network can be viewed as feature-abstraction functions. As information propagates through the network, they abstract into high-level features. These high-level features take up much less room than the original data, as a result, making them much easier to transmit over the network. The application logic in the cloud is fairly easy to change. This hot-swapping of the network layer enables the same devices to be used for different applications. The practice of modifying part of the network to perform different tasks can employ the concept of transfer learning.

**1.2       SCOPE**

A number of critical scenarios would be enabled by the application of this concept of Blindophile named device. One such novel ideal, which we propose, would help visually impaired people to carry out their daily tasks in a simpler and efficient manner. Building on recent research on abandonment of specialized devices, we explore a new touch free mode of interaction, wherein a person with visual impairment employs action-based gestures to trigger tasks on their smartphones. We present an integrated solution that can be used with different physical objects to perform a diverse number of functions and operations. This will also allow management of a smartphone without touch, or removing the phone from the pocket. In future, different physical devices can leverage this functionality by using sub-modules of Blindophile leading to various applications in different fields.

**CHAPTER 2**

**LITERATURE REVIEW**

**2.1   RELATED PREVIOUS READINGS**

**2.1.1** **Cloud and Edge Devices**

Edge devices are generating more data and more complex data than ever before - Devices ranging from sensors to robots are generating vast amounts of sensor readings, images, sounds, and videos but multiple levels of computation can exist between the data generation point and the final data repository. From an analytics point of view, each of these compute elements may see different liveness, history, and perspective of the data:  edge compute nodes closest to the data generation point have highest liveness but little or no history and no perspective. And compute nodes further away from the sensor and closer to the cloud have more history and perspective but older data. In real world production, data patterns change without warning. Previously trained ML/DL models can become irrelevant and subsequent predictions inaccurate. Model re-training is therefore needed and can require human intervention. Even determining when to retrain can be difficult. A common practice is to retrain frequently, but this is expensive. An ideal ML/DL solution would maintain accuracy in the face of changing data via dynamic models that can adapt, as well as a management system that can refresh models as needed. While analytics can run virtually anywhere, the resources at each layer can be very different. An edge node could be a single server on a manufacturing floor, while a higher layer could be a public or private cloud. An ideal ML/DL solution would reduce management cost by combining the best suited analytic engines at each level of the topology into a seamless unified workflow. Edge computing is a distributed computing paradigm which brings computation and data storage closer to the location where it is needed, to improve response times and save bandwidth. Edge computing connects “things” to a compute/storage infrastructure. Challenges in using Edge devices to process ML algorithms are manifold. Latency sensitive applications must generate insights or react to data changes quickly. For example: In a Smart Building application such as video based intruder detection, it is not always possible to send the image data to the cloud for processing and wait seconds for a response. In Industrial IoT applications, fault detection windows can sometimes be in the 10s of milliseconds, precluding a round trip to a cloud. Even latency tolerant applications (such as health diagnostics) may require edge ML to operate in disconnected scenarios. These types of scenarios necessitate at a minimum ML/DL inference at edge, possibly with training in cloud, edge, or across both.

Then comes the issue of security and cost. The cost of sending petabytes or terabytes of data per hour to cloud devices can be prohibitive and may incur security or privacy concerns. For example: It is possible for industrial IoT applications to generate several MB/s of time series sensor data, and for other apps such as smart transportation (that capture image or video data) to generate even more. Using precious backbone bandwidth to transmit data is not always beneficial if value can be generated by bringing the compute to the data. Such constraints can affect how and where ML/Dl training is deployed. Sensor data can also be complex, mandating significant resources. Edge devices need to reduce cost and power, thus requiring the complex processing on rich data be efficiently distributed across both edge and cloud. To enable effective edge ML, we need to perform retraining, code propagation and diagnostics. Since the models trained via ML are data dependent, one would have to periodically retrain when the data pattern changes. Retraining is compute intensive and iterative. Recent advancements in ML have enabled a combination of both online and offline training methods to be used to improve accuracy. This is particularly relevant for edge devices since many edge devices may deploy the same centrally trained model but then evolve it online to meet edge-specific scenarios. There are many ML/DL Engines (such as Spark and TensorFlow). Moreover, ML pipelines from the same application may run on different analytic engines. For example: It is a common application pattern to run training as a batch job on Spark and predictions in a streaming context or REST service. For latency-sensitive edge environments, streaming inference can be quite common, but the model used therein could have been batch-trained on the cloud. ML requires models to be periodically updated. Model updates and deployment are critical to the health of ML initiatives. A single model from training could be deployed over thousands of edge nodes. For a large-scale edge installation, one may not want to deploy a new model to all of the edge instances at the same time, but rather incrementally deploy the new model with increasing confidence in the field. This requires a way to seamlessly deploy both code and models with different deployment policies. Debugging issues in a decentralized environment is challenging. In addition to issues with runtimes, algorithms, configuration parameters, model deployment, etc. ML adds an additional dimension since it is data dependent and model performance is loosely coupled with its training pipeline(s). One has to enable collection of information not only about the distributed ML pipelines but also their runtime statistics, logs, and any user-defined statistics. It is most productive to debug an ML application as a single entity and not as a set of independent entities.

Use Cases for Machine Learning at the Edge can be manifold. The applications for AI/ML at the edge go well beyond security and surveillance. In the consumer world, features like facial recognition or iris recognition on the latest smartphones are other examples of machine learning that are independent of the cloud. Modern phones with these features are trained to “learn” specific details of your unique identity through their setup process. Machine learning algorithms running locally on the phones then verify your identity every time you sign into your device. The local operation of these efforts is essential for many reasons. First, smartphones are used in all kinds of environments where connectivity can’t be guaranteed. Second, the sign-in process occurs frequently so needs to be as fast and frictionless as possible. Finally, data privacy and security concerns associated with biometric data make it extremely important to only use the data locally on the device and not send it out over a cloud connection.

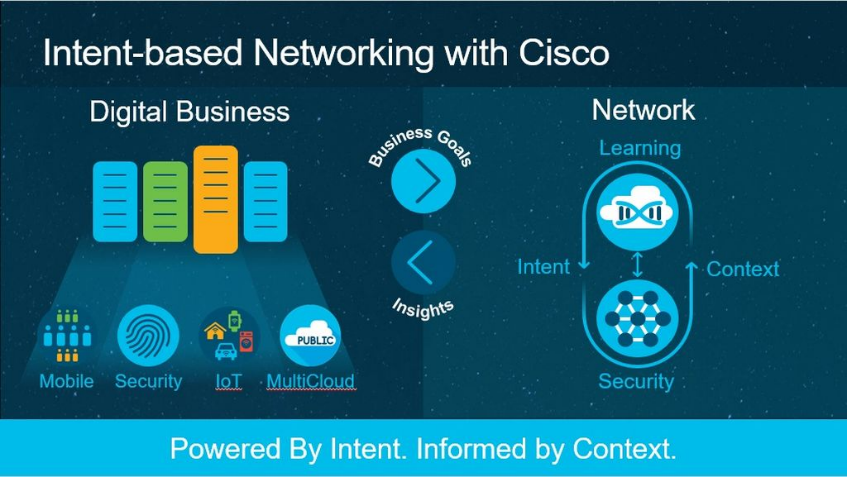


Figure 2.1 Interconnection of IoT with Network

Adding AI to edge devices can lead to various advantages. Simple image classification, gesture recognition, acoustic detection and motion analysis can be done on the edge device. Because only the final result is transmitted, we can minimize delay, improve privacy and conserve the bandwidth in IoT systems. The image on the left shows the classic hand-written-digit dataset, MNIST, in a projected space.

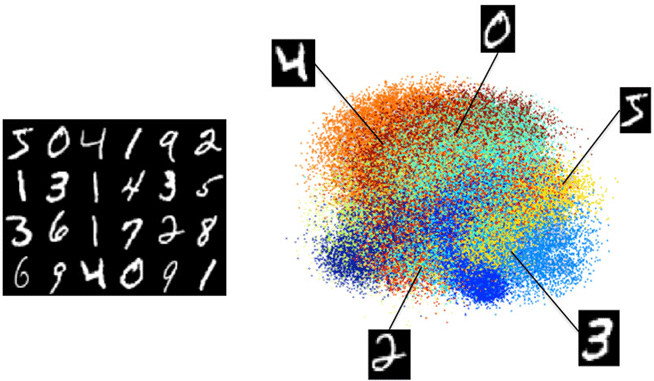


Figure 2.2 Inferencing with MNIST data

Using machine learning and other signal processing algorithms, different off-the-shelf sensors can be combined into a synthetic sensor.

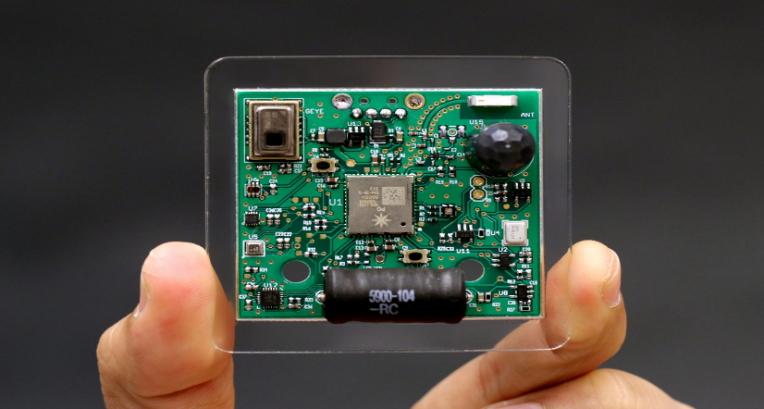


Figure 2.3 Image of sensor

These types of sensors are capable of detecting complex events. These sensors are of lower cost and more energy efficient comparison to camera based systems. Devices can make continuous improvements after they are deployed in the field. Google’s Gboard uses a technique called federated learning, that involves every device collecting data and making individual improvements. These individual improvements are aggregated on a central service and every device is then updated with the combined result.

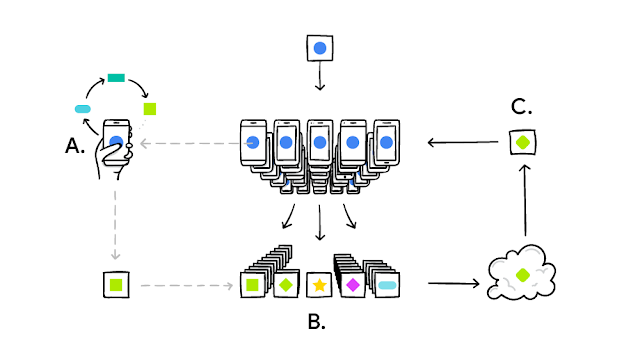


Figure 2.4 Self-learning device

Neural networks can be partitioned such that some layers are evaluated on the device and the rest in the cloud. This enables the balancing of workload and latency. The initial layers of a network can be viewed as feature-abstraction functions. As information propagates through the network, they abstract into high-level features. These high-level features take up much less room than the original data, as a result, making them much easier to transmit over the network. IoT communication technologies, such as Lora and NB-IoT have very limited payload size. Feature-extraction helps to pack the most relevant information in limited payloads.

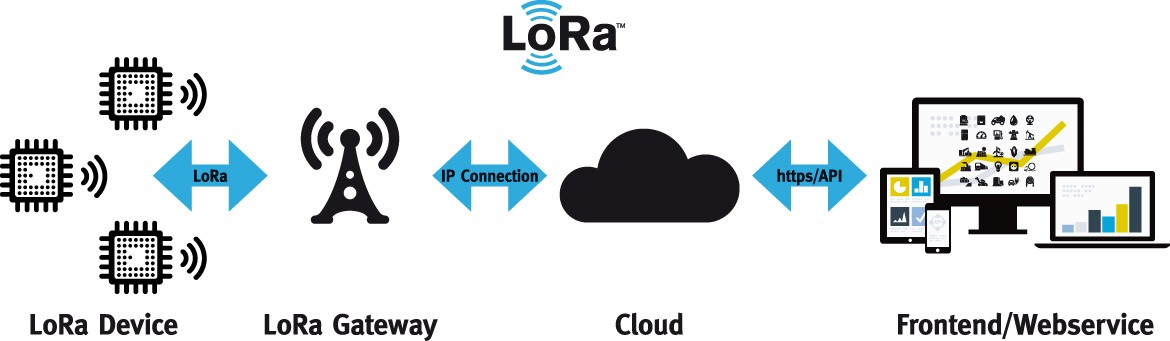


Figure 2.5 Communication of device with Cloud

In the bandwidth example above, the neural network is distributed between device and cloud. In some cases, it is possible to repurpose the network for a completely different application by just changing the layers in the cloud.



Figure 2.6 Application logic in Cloud

The application logic in the cloud is fairly easy to change. This hot-swapping of the network layer enables the same devices to be used for different applications. The practice of modifying part of the network to perform different tasks is an example of transfer learning.

**2.1.2. FastGRNN and RNN**

FastGRNN (an acronym for a Fast, Accurate, Stable and Tiny Gated Recurrent Neural Network) is an algorithm to address the twin RNN limitations of inaccurate training and inefficient prediction. FastGRNN almost matches the accuracies and training times of state-of-the-art other RNNs but has significantly lower prediction costs with models ranging from 1 to 6 Kilobytes for real-world applications. Previous RNN algorithms have improved accuracy at the expense of prediction costs making them infeasible for resource-constrained and real-time applications. Enforcing FastGRNN’s matrices to be low-rank, sparse and quantized resulted in accurate models that could be up to 35x smaller than leading gated and unitary RNNs. This allowed FastGRNN be deployed on severely resource-constrained IoT microcontrollers too tiny to store other RNN models.

In particular, FastRNN’s prediction accuracies could be:

         (a) Upto 19% higher than a standard RNN;

         (b) Could often surpass the accuracies of all unitary RNNs and

         (c) Could be just shy of the accuracies of leading gated RNNs.

Squeezing the RNN model and code into a few Kilobytes could allow RNNs to be deployed on billions of Internet of Things (IoT) endpoints having just 2 KB RAM and 32 KB flash memory. Similarly, squeezing the RNN model and code into a few Kilobytes of the 32 KB cache of a Raspberry Pi or smartphone, could significantly reduce the prediction time and energy consumption and make RNNs feasible for real-time applications such as predictive maintenance, human activity recognition, *etc*. Enforcing FastGRNN’s matrices to be low-rank, sparse and quantized led to a minor decrease in the prediction accuracy but resulted in models that could be up to 35x smaller and fit in 1-6 Kilobytes for many applications. For instance, using a 1 KB model, FastGRNN could match the prediction accuracies of all other RNNs. This allowed FastGRNN to be deployed on IoT endpoints, such as the Arduino Uno, which were too small to hold other RNN models. On slightly larger endpoints, such as the Arduino Duo, FastGRNN was found to be 18-42x faster at making predictions than other leading RNN methods.

Hence, FastRNN could lead to provably stable training by incorporating a residual connection with two scalar parameters into the standard RNN architecture. FastRNN was demonstrated to have lower training times, lower prediction costs and higher prediction accuracies than leading unitary RNNs in most cases. FastGRNN’s model could be compressed to 1-6 KB without compromising accuracy in many cases by enforcing that its parameters be low-rank, sparse and quantized. This allowed FastGRNN to make accurate predictions efficiently on severely resource-constrained IoT devices too tiny to hold other RNN models.

**2.1.3. kNN and ProtoNN**

Internet of Things (IoT) sensor applications demand prediction models with small storage and computational complexity that do not compromise significantly on accuracy. ProtoNN, an algorithm that addresses the problem of real-time and accurate prediction on resource-scarce devices. ProtoNN is inspired by k-Nearest Neighbour (KNN) but has several orders lower storage and prediction complexity. ProtoNN models can be deployed even on devices with puny storage and computational power (e.g. an Arduino UNO with 2kB RAM) to get excellent prediction accuracy.

A key reason for selecting kNN as the algorithm of choice is due to its generality, ease of implementation on tiny devices, and small number of parameters to avoid overfitting. However, kNN suffers from three issues which limit its applicability in practice, especially in the small devices setting:

1. Poor accuracy: kNN is an ill-specified algorithm as it is not a priori clear which distance metric one should use to com- pare a given set of points.
2. Model size: kNN requires the entire training data for prediction, so its model size is too large for the IoT setting.
3. Prediction time: kNN requires computing the distance of a given test point with respect to each training point, making it prohibitive for prediction in real-time.

ProtoNN is able to address the above mentioned concerns by using the following key ideas:

      a) Learning a small number of prototypes to represent the entire training set

      b) Sparse low dimensional projection of data

      c) Joint discriminative learning of the projection

1. Prototypes with explicit model size constraint

We project the entire data in low-d using a sparse projection matrix that is jointly learned to provide good accuracy in the projected space. We learn prototypes to represent the entire training dataset. Moreover, we learn labels for each prototype to further boost accuracy. This provides additional flexibility, and allows us to seamlessly generalize ProtoNN for multi-label or ranking problems. We learn the projection matrix jointly with the prototypes and their labels. ProtoNN can be implemented efficiently, can handle datasets with millions of points, and obtains state-of-the-art accuracies. On multilabel datasets, ProtoNN can give 100× compression with ≤ 1% loss in accuracy. Finally, we demonstrate that ProtoNN can be deployed on a tiny Arduino Uno device and leads to better accuracies than existing methods while incurring significantly less energy and prediction time costs.

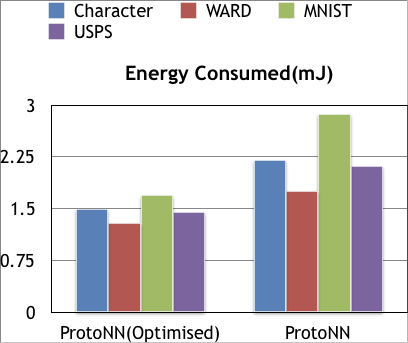
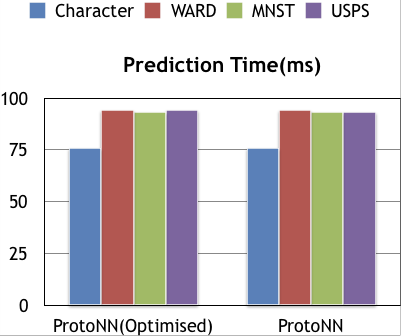


Figure 2.7 Graphs of prediction of time and energy consumed by ProtoNN (2kB) and its optimised version

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Dataset** | **n** | **d** | **L** | **Sources** |
| Character Recognition | 4397 | 400 | 2 | Kaggle |
| MNIST | 60000 | 784 | 2 | CMU Libraries |
| USPS | 7291 | 256 | 2 | Kaggle |
| WARD | 4503 | 1000 | 2 | Kaggle |
| Human recognition Using Smartphones | 10299 | 561 | 2 | UCI Machine Learning Repository |

Table 2.1 Dataset statistics and links

**2.2 PREDICTION**

Gesture and activity recognition is an extensively studied problem and Inertial Measurement Unit (IMU) sensors like accelerometer and gyroscope have been successfully used for detecting gestures in a variety of applications. While gesture recognition has been studied extensively, most of the existing solutions either:

a) Rely on hand-tuned rules for a small number of gestures/activities like running, sleeping;

b) Apply simple ML algorithms that can handle simple gestures in restricted settings; and use a powerful device to detect gestures/activities via computationally expensive ML algorithms.

Finally, due to weight, battery and latency constraints, we are required to predict gestures on a low-powered microcontroller without any external computer help. To the best of our knowledge, our method is the first gesture detection system that can detect complicated gestures accurately and robustly despite access to tiny computing devices. Recently, there have been a couple of ML algorithms intended which can produce tiny but effective ML models. Examples include SNC, BNC, NN-prune, BONSAI, ProtoNN, etc. In this work, we use ProtoNN algorithm which compresses standard k-Nearest Neighbour (kNN) algorithm to decrease both the model size as well as prediction time. Similar to kNN, ProtoNN can also learn non-linear and complicated decision boundaries. However, ProtoNN still requires “featurized" data and does not perform well on unprocessed sensor values. Hence, designing a “featurizer" for unprocessed sensor values is equally crucial to the success of the entire system.

**2.3 CLASSIFICATION**

ML prediction typically consists of two parts:

1) Data featurization that converts raw sensor data into features that are suitable for the ML model, and

2) A classification algorithm that classifies the featurized data into one of the gestures (including no gesture).

For our classification algorithm, we use the multi-class formulation of the recently-proposed ProtoNN algorithm which is specifically designed to generate models small enough to run on microcontrollers. However, when we employed ProtoNN with standard ML features (e.g., FFT features, clustering-based features), it exceeded the time budget due to data featurization cost. Therefore, we had to design a set of features that are easy enough to compute, consume small amount of memory and are still robust enough to be able to discern practical gestures.

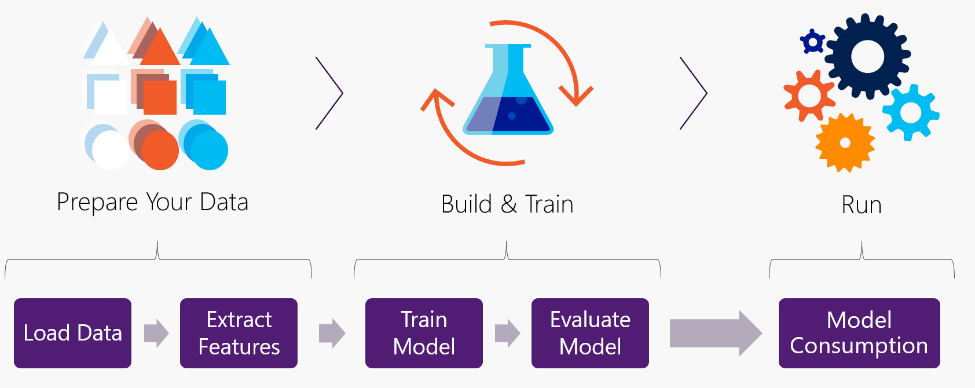


Figure 2.7 Data Featurization

**2.4 PROPOSED FEATURES**

The Featurization step converts unprocessed sensor data into a set of features. The unprocessed sensor data consists of six dimensions, three each from the accelerometer and the gyroscope. For 2-seconds, when sampled at 200 Hz, the raw data for each prediction instance consist of 400 values in each of the six dimensions. We design two kinds of features.

First, for each of the 6 dimensions, we group the 400 values into 20 equally-spaced bins in their range and count the number of values in each bin. Such equally-spaced bin counts are particularly efficient to compute in our strided-window setting.

Second, bin count features discard phase in the gyroscope values (clockwise vs. anti-clockwise). To capture this phase information, we add four additional features: the index and length of the longest positive and negative sequence of values from each dimension of the gyroscope. However, typically, the only axis of rotation is along the vertical axis of the cane. Therefore, we only needed phase information along this axis of the gyroscope. In total, we compute 124 features for each training sample.

**2.5 TRAINING MODEL**

With the new set of features, we trained the ProtoNN model on our dataset using a commodity PC. We randomly split the collected training dataset into 80% training samples and 20% testing samples. We tuned the ProtoNN hyper-parameters simultaneously to achieve high accuracy and low model size. Our final model was just 6 KB in size and achieved an accuracy of 99.9% on the test data. In the real world, our model achieves 92% accuracy.

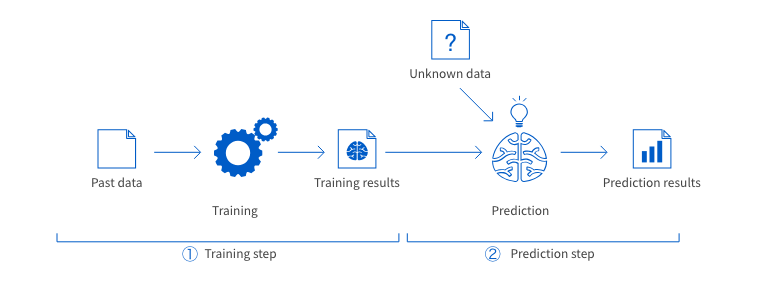
****

Figure 2.8 Model Training

**2.6 PREDICTION PIPELINE**

Our gesture prediction channel runs a continuous cycle of:-

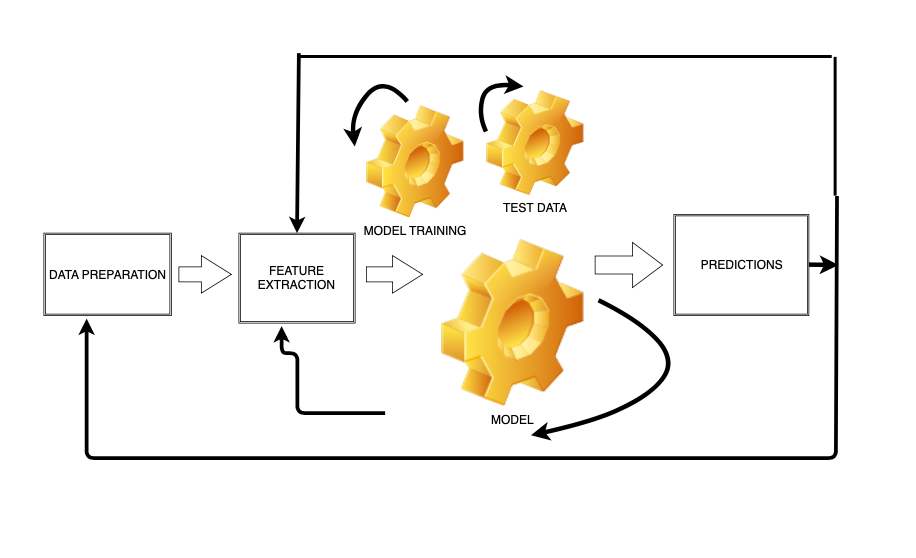
(a) Data collection from IMU,

(b) Feature computation,

(c) ProtoNN inference algorithm on the model generated by the ProtoNN training algorithm; and

 (d) BLE communication for relaying the gestures detected to the phone.

With our new feature set and our 6 KB ProtoNN model, the microcontroller can complete data featurization and the ML classification in 27 ms and 16 ms, respectively. To reduce false positives, we use a secondary filter wherein our algorithm keeps track of the latest n predictions. If the majority of these predictions point to a particular gesture, then our algorithm confirms the presence of that gesture and communicates the gesture to the smartphone.

Figure 2.9 Machine Learning Inference flowchart

**CHAPTER 3**

**METHODOLOGY**

**3.1 MODEL**

* + 1. **KNN Algorithm**
* Supervised machine learning algorithm
* Non parametric as it does not make an assumption about the underlying data distribution pattern
* Lazy algorithm as KNN does not have a training step. All data points will be used only at the time of prediction. With no training step, prediction step is costly.
* An eager learner algorithm
* Used for both classification and regression
* Uses feature similarity to predict the cluster that the new point will fall into

K is a number used to identify similar neighbours for the new data point. KNN takes K nearest neighbours to decide where the new data point with belong to. This decision is based on feature similarity. Choice of K has a drastic impact on the results we obtain from KNN. Value of K at the elbow of test error rate gives us the optimal value of K.

KNN works using the following steps:

Step 1: Choose a value for K. K should be an odd number.

Step2: Find the distance of the new point to each of the training data.

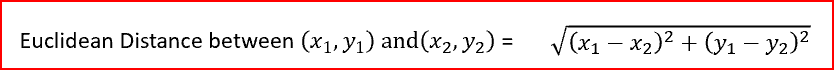
Step 3: Find the K nearest neighbours to the new data point.

Step 4: For classification, count the number of data points in each category among the k neighbours. New data point will belong to class that has the most neighbours.

For regression, value for the new data point will be the average of the k neighbours. Distance can be calculated using

* **Euclidean distance**
* **Manhattan distance**
* **Hamming Distance**
* **Minkowski Distance**

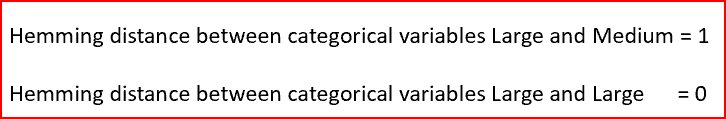
Euclidean distance is the square root of the sum of squared distance between two points. It is also known as L2 norm.



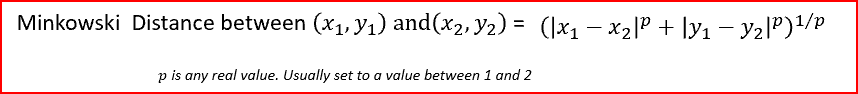
Manhattan distance is the sum of the absolute values of the differences between two points



Hamming distance is used for categorical variables. In simple terms it tells us if the two categorical variables are same or not.



Minkowski distance is the used to find distance similarity between two points. When p=1, it becomes Manhattan distance and when p=2, it becomes Euclidean distance



**Pros of K Nearest Neighbours:-**

* Simple algorithm and hence easy to interpret the prediction
* Non parametric, so makes no assumption about the underlying data pattern
* Used for both classification and Regression
* Training step is much faster for nearest neighbour compared to other machine learning algorithms.

**Cons of K Nearest Neighbours:-**

* KNN is computationally expensive as it searches the nearest neighbours for the new point at the prediction stage
* High memory requirement as KNN has to store all the data points
* Prediction stage is very costly
* Sensitive to outliers, accuracy is impacted by noise or irrelevant data.

**3.1.2 ProtoNN Algorithm**

ProtoNN is inspired by k-Nearest Neighbour (KNN) but has several orders lower storage and prediction complexity. ProtoNN models can be deployed even on devices with puny storage and computational power (e.g. an Arduino UNO with 2kB RAM) to get excellent prediction accuracy. ProtoNN derives its strength from three key ideas:

a) Learning a small number of prototypes to represent the entire training set,

b) Sparse low dimensional projection of data, and

c) Joint discriminative learning of the projection and prototypes with explicit model size constraint.

ProtoNN on a variety of supervised learning tasks (binary, multi-class, multi-label classification) gives nearly state-of-the-art prediction accuracy on resource-scarce devices while consuming several orders lower storage, and using minimal working memory.

ProtoNN is able to address the three key ideas:

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

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

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

**3.1.3. Bonsai Algorithm**

Bonsai is Tree based algorithm for efficient prediction on IoT devices. Bonsai maintains prediction accuracy while minimizing model size and prediction costs by:

(a) Developing a tree model which learns a single, shallow, sparse tree with powerful nodes;

(b) Sparsely projecting all data into a low-dimensional space in which the tree is learnt; and

(c) Jointly learning all tree and projection parameters.

Bonsai can make predictions in milliseconds even on slow microcontrollers, can fit in KB of memory, and has lower battery consumption than all other algorithms while achieving prediction accuracies that can be as much as 30% higher than state- of-the-art methods for resource-efficient machine learning. Bonsai is also shown to generalize to other resource constrained settings beyond IoT by generating significantly better search results.

First, Bonsai learns a single, shallow, sparse tree so as to reduce model size but with powerful nodes for accurate prediction. Second, both internal and leaf nodes in Bonsai make non-linear predictions. Bonsai’s overall prediction for a point is the sum of the individual node predictions along the path traversed by the point. Path based prediction allows Bonsai to accurately learn non-linear decision boundaries while sharing parameters along paths to further reduce model size. Third, Bonsai learns a sparse matrix which projects all data points into a low-dimensional space in which the tree is learnt. This allows Bonsai to fit in a few KB of flash.

* 1. **DATA ACQUISITION**

The dataset contains the double tap readings measured from the IMU(Inertial Measurement Unit) sensor readings which contain 6 readings in one go, which comprises of acceleration across x-, y- and z- axis and gyros across x-, y- and z- axis. IMU comprises of accelerometer for sensing the tilt and gyroscope for angular velocity.

Accelerometers are used to sense both static (e.g. gravity) and dynamic (e.g. sudden starts/stops) acceleration. As tilts are affected by the acceleration of gravity, an accelerometer can tell you how it's oriented with respect to the Earth's surface. An accelerometer can also be used to sense motion.

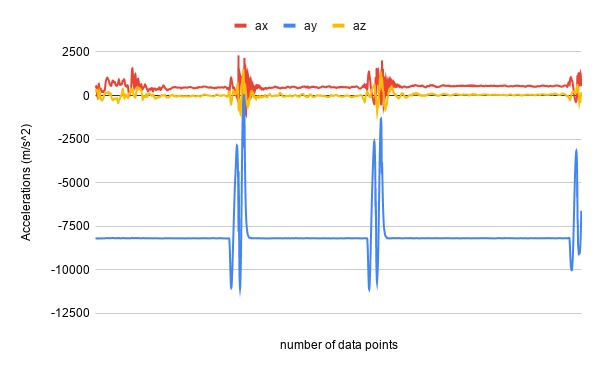


Figure 3.1 Analysis of blindophile dataset with respect to acceleration

Gyroscopes measure angular velocity, how fast something is spinning about an axis. If someone is trying to monitor the orientation of an object in motion, an accelerometer may not give you enough information to know exactly how it's oriented. Unlike accelerometers gyros are not affected by gravity, so they make a great complement to each other.

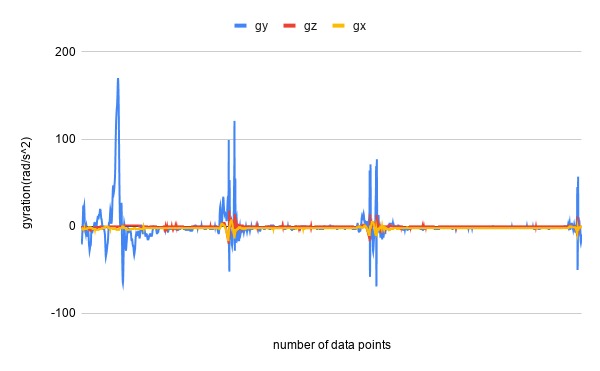


Figure 3.2 Analysis of blindophile with respect to gyration

IMU provides two to six degrees of freedom (DOF). IMUs are widely used in devices that require knowledge of their exact position, for example robotic arms, guided missiles, and tools used in the study of body motion.

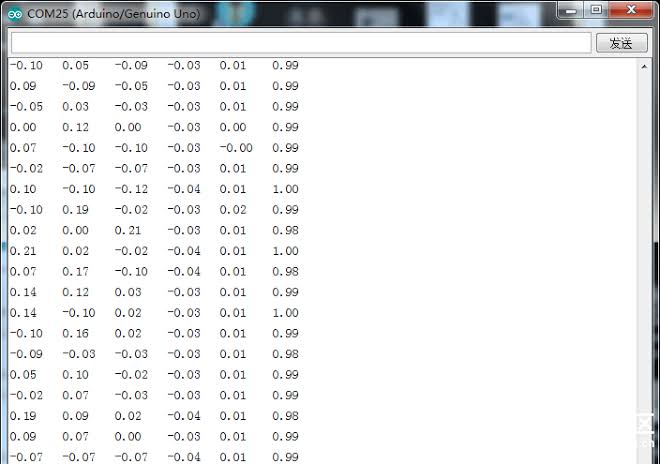


Figure 3.3 Dataset of IMU sensor to recognise double tap gesture

* 1. **MODEL SELECTION**

For gesture recognition we are using ProtoNN and Bonsai algorithm as we have achieved state-of-the-art predictions and accuracies with them and as compared to the conventional kNN algorithm (as kNN suffers from three limitations which affects its applicability in practice, especially in the tiny devices settings). kNN is an ill-specified algorithm as it is not analytically clear which distance metric one should use to compare a given set of points. kNN requires the whole training data for prediction, so its model size is too big for the IoT settings and our main objective is to minimize the model size so that it can run smoothly on dumb and tiny IoT devices. Additionally, kNN requires computing the distance of a given test point w.r.t. each training point, making it prohibitive for prediction in real-time.

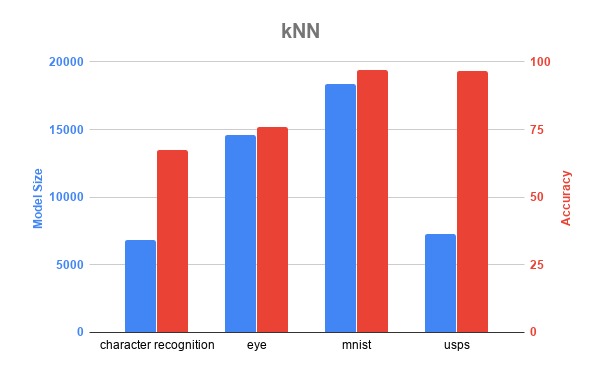


Figure 3.4 Accuracy of kNN model with model size

ProtoNN in a simple binary classification setting where the data is sampled from a mixture of two well-separated Gaussians, each Gaussian representing one class. The performance of ProtoNN is tested on different benchmark binary, multiclass and multilabel datasets with an aim to demonstrate the aspects such as; Firstly, In severely resource constrained settings where we require model sizes to be less than 2kB (which occur routinely for IoT devices like Arduino Uno), we outperform all state-of-the art compressed methods. Secondly, for model sizes in the range 16 − 32 kB, we achieve comparable accuracies to the best uncompressed methods. And lastly, in multiclass and multilabel problems we achieve near state-of-the-art accuracies with an order of magnitude reduction in model size, thus showing our approach is flexible and general enough to handle a wide variety of problems.

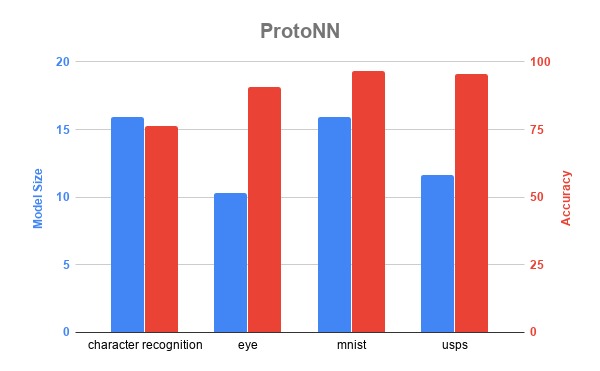


Figure 3.5 Accuracy of ProtoNN model with model size

Bonsai reduces its prediction costs on the Arduino and Micro controllers. It is found to be even lower than that of a non-optimized linear classifier. This allows Bonsai to enjoy the prediction accuracy of a non-linear classifier while paying less than linear costs. It was observed that Bonsai make predictions in milliseconds even on slow microcontrollers, fit in a few KB of flash and extend battery life beyond all other algorithms. Bonsai is shown to generalize to other resource constrained settings beyond IoT by producing significantly better search results as compared to other algorithms where the model size is restricted to 300 bytes.

* 1. **IMPLEMENTATION**

The implementation required data pre-processing which further involved gathering the dataset, sampling the data, scrubbing for duplicate data and finally analysing it. Data pre-processing is done to get efficient data in proper format. Data was first cleaned to avoid missing and noisy data. After data cleaning, data transformation was performed to transform the data in appropriate forms suitable for mining process. We used discretization as a method for data transformation as it replaces the raw values of numeric attribute by interval levels or conceptual. While working with huge volume of data, analysis became harder in such cases. In order to get rid of this, we used data reduction technique. It aims to increase the storage efficiency and reduce data storage and analysis costs. Lastly, for data reduction we used dimensionality reduction method as it reduces the size of data by encoding mechanisms. It can be lossy or lossless. If after reconstruction from compressed data, the original data can be retrieved, such reduction is called lossless reduction else it is called lossy reduction. The two effective methods of dimensionality reduction are: Wavelet transforms and PCA (Principal Component Analysis).

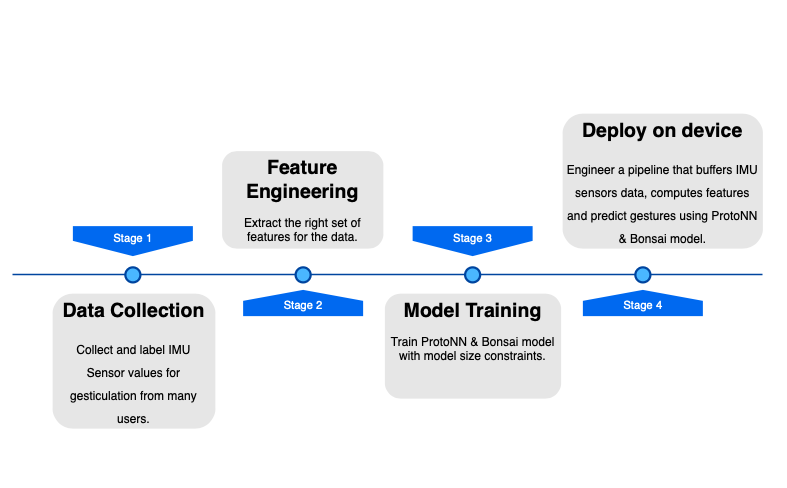


Figure 3.6 Flowchart for training and deploying the device

After the data pre-processing was done completely on the dataset comprising of double taps, we did feature engineering for the dataset. This used the domain knowledge of the data, in this case IMU sensor readings, to create [features](https://en.wikipedia.org/wiki/Feature_(machine_learning)) that helped in making our chosen [machine learning](https://en.wikipedia.org/wiki/Machine_learning) algorithms work.

The feature engineering process is:

* [Brainstorming](https://en.wikipedia.org/wiki/Brainstorming) or [testing](https://en.wikipedia.org/wiki/Software_testing) features;
* Deciding what features to create;
* Creating features;
* Checking how the features work with our model;
* Improving our features if needed;
* Going back to brainstorming/creating more features until the work is completely done.

After the features were engineered we went for model training. For getting the accurate predictions and results we used kNN (k-nearest Neighbours), ProtoNN and Bonsai algorithms. Next came model training; the goal of training is to answer a question or make a prediction correctly as often as possible. Each iteration of process is a training step. Then, evaluation of the model was done to use some metric or combination of metrics to "measure" objective performance of model and test it against previously unseen data. This unseen data was meant to be somewhat represent model performance in the real world, but still helps tuned the model. After that, parameter tuning was performed which also refers to hyper parameter tuning. In this step model parameters are tuned for improved performance. Simple model hyper parameters may include: number of training steps, learning rate, initialization values and distribution, etc.

After all this we split the dataset into training data and testing data. If you evaluate your model on the same data you used to train it, your model could be overfit. A model should be judged on its ability to predict new, unseen data. Training sets are used to fit and tune the models. Test sets are put aside as "unseen" data to evaluate the models.

Finally predictions are made using further (test set) data which has, until this point, been withheld from the model (and for which class labels are known), are used to test the model; a better approximation of how the model will perform in the real world.

Lastly the algorithms were deployed on the tiny and dumb IoT device for recognising gestures.

**CHAPTER 4**

**RESULTS & DISCUSSION**

**4.1 ANALYSIS**

The analysis of different dataset accuracy was checked using different models like SNC, NeuralNet Pruning, Tree Pruning, Decision Jungle, etc.

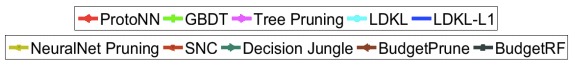


Figure 4.1 List and symbols of different Machine learning models

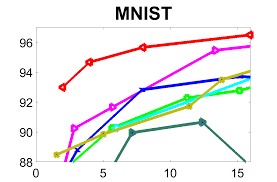


Figure 4.2 Analysis of MNIST dataset with different models

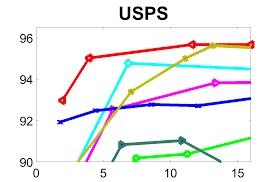


Figure 4.3 Analysis of USPS dataset with different models

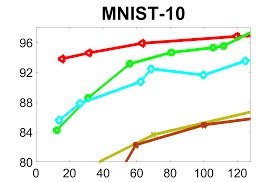


Figure 4.4 Analysis of MNIST-10 dataset with different models

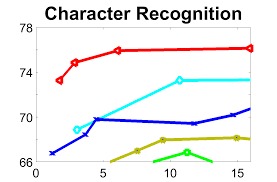


Figure 4.5 Analysis of Character Recognition dataset with different models

**4.2 FUTURE SCOPE**

Due to lack of time we were not able to produce real life device which could help the visually impaired. So in the coming time we would like to design a device that could recognise numerous gestures which can further trigger an event in the respective smartphone. However, the ML model could accurately recognize the set of 5 gestures that included double tap, right twist, left twist, twirl and double swipe. In future, the interactive device will enable faster completion of common smartphone-related tasks. The device would enable visually impaired people to easily adapt to the basic smartphone functionality and benefit from additional functions such as location. Furthermore, the triggered actions would be receiving a phone call from a test phone, calling back the last caller from a missed call notification, starting and stopping audio recordings on the phone, reading out the current geographic location, checking for notifications and reading out the time. We would also like to deploy real-time ML algorithms to detect gestures robustly and effectively despite variations in users. To facilitate access to certain smartphone tasks, we will be building a custom Android app that has an internal state machine. The Android app will be connected to the dumb and tiny IoT device through BLE (Bluetooth Module) using a Bluetooth terminal application which will further help in triggering the concerned event. Going forward, we will be working on detecting more nuanced gestures that can enable access to many more phone apps.

**REFERENCES**

[1] Gopinath, S., Ghanathe, N., Seshadri, V., & Sharma, R. (2019, June). Compiling KB-sized machine learning models to tiny IoT devices. In Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation (pp. 79-95). ACM.

[2] Shishir, G. Patil, Don Kurian, Dennis, C. Pabbaraju, R. Deshmukh, Harsha Vardhan, Simhadri, M.Varma and Prateek Jain (2019, June). Programmable Gesture Recognition for Augmenting Assistive Devices. In Proceedings of the Microsoft Research Conference on Data Mining and knowledge extraction (pp. 69-101). Microsoft.

[3] Talagala, N., Sundararaman, S., Sridhar, V., Arteaga, D., Luo, Q., Subramanian, S., & Roselli, D. (2018). {ECO}: Harmonizing Edge and Cloud with ML/DL Orchestration. In {USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 18).

[4] Dennis, D., Pabbaraju, C., Simhadri, H. V., & Jain, P. (2018). Multiple instance learning for efficient sequential data classification on resource-constrained devices. In Advances in Neural Information Processing Systems (pp. 10953-10964).

[5] Kusupati, A., Singh, M., Bhatia, K., Kumar, A., Jain, P., & Varma, M. (2018). Fastgrnn: A fast, accurate, stable and tiny kilobyte sized gated recurrent neural network. In Advances in Neural Information Processing Systems (pp. 9017-9028).

[6] Gupta, C., Suggala, A. S., Goyal, A., Simhadri, H. V., Paranjape, B., Kumar, A., & Jain, P. (2017, August). ProtoNN: compressed and accurate kNN for resource-scarce devices. In Proceedings of the 34th International Conference on Machine Learning-Volume 70 (pp. 1331-1340). JMLR. org.

[7] Li, H., Ota, K., & Dong, M. (2018). Learning IoT in edge: Deep learning for the Internet of Things with edge computing. IEEE Network, 32(1), 96-101.

[8] Batool, S., Saqib, N. A., & Khan, M. A. (2017, May). Internet of Things data analytics for user authentication and activity recognition. In 2017 Second International Conference on Fog and Mobile Edge Computing (FMEC) (pp. 183-187). IEEE.

[9] Patel, P., Ali, M. I., & Sheth, A. (2017). On using the intelligent edge for IoT analytics. IEEE Intelligent Systems, 32(5), 64-69.

[10] Bevilacqua, F., Zamborlin, B., Sypniewski, A., Schnell, N., Guédy, F., & Rasamimanana, N. (2009, February). Continuous real-time gesture following and recognition. In International gesture workshop (pp. 73-84). Springer, Berlin, Heidelberg.

[11] Freeman, W. T., & Roth, M. (1995, June). Orientation histograms for hand gesture recognition. In International workshop on automatic face and gesture recognition (Vol. 12, pp. 296-301).Chicago

[12] Specht, D. F. (1991). A general regression neural network. IEEE transactions on neural networks, 2(6), 568-576.Chicago

[13] McCarthy, J., & Hayes, P. J. (1981). Some philosophical problems from the standpoint of artificial intelligence. In Readings in artificial intelligence (pp. 431-450). Morgan Kaufmann.

[14] Chua, L. O., & Roska, T. (1993). The CNN paradigm. IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications, 40(3), 147-156.Chicago

[15] Muniyandi, A. P., Rajeswari, R., & Rajaram, R. (2012). Network anomaly detection by cascading k-Means clustering and C4. 5 decision tree algorithm. Procedia Engineering, 30, 174- 182.Chicago

**APPENDIX**

#include "protoNN.h"

int8\_t ProtoNNF::getInitErrorCode(){

this->errorCode = 0;

int d = this->featDim;

int d\_cap = this->ldDim;

int m = this->numPrototypes;

int L = this->numLabels;

float gamma = this->gamma;

if ((d !=protoNNParam::featDim))

this->errorCode |= 1;

if (d\_cap !=protoNNParam::ldDim)

this->errorCode |= 2;

if (m !=protoNNParam::numPrototypes)

this->errorCode |= 4;

if(L !=protoNNParam::numLabels)

this->errorCode |= 8;

if ((gamma - (protoNNParam::gamma)) >= 0.001 ||

(gamma - (protoNNParam::gamma)) <= -0.001){

this->errorCode |= 16;

}

return this->errorCode;

}

ProtoNNF::ProtoNNF() {

this->featDim =protoNNParam::featDim;

this->ldDim =protoNNParam::ldDim;

this->numPrototypes =protoNNParam::numPrototypes;

this->numLabels =protoNNParam::numLabels;

this->gamma = protoNNParam::gamma;

this->errorCode = getInitErrorCode();

}

ProtoNNF::ProtoNNF(unsigned d, unsigned d\_cap,

unsigned m, unsigned L, float gamma){

this->featDim = d;

this->ldDim = d\_cap;

this->numPrototypes = m;

this->numLabels = L;

this->gamma = gamma;

this->errorCode = getInitErrorCode();

}

int8\_t ProtoNNF::denseLDProjection(float\* x, float\* x\_cap){

unsigned int d = this->featDim;

unsigned int d\_cap = this->ldDim;

for (int i = 0; i < d\_cap; i++){

float dotProd = 0.0;

for(int j = 0; j < d; j++){

float component = getProjectionComponent(i, j);

dotProd += x[j] \* component;

}

x\_cap[i] = dotProd;

}

return 0;

}

float ProtoNNF::gaussian(const float \*x, const float \*y,

unsigned length, float gamma = 1.0) {

float sumSq = 0.0;

for(unsigned i = 0; i < length; i++){

sumSq += (x[i] - y[i])\*(x[i] - y[i]);

}

sumSq = -1\*gamma\*gamma\*sumSq;

sumSq = exp(sumSq);

return sumSq;

}

int8\_t ProtoNNF::scalarVectorMul(float \*vec, unsigned length,

float scalar) {

for(unsigned i = 0; i < length; i++) {

vec[i] = vec[i] \* scalar;

}

return 0;

}

int8\_t ProtoNNF::vectorVectorAdd(float \*dstVec, float \*srcVec,

unsigned length){

for(unsigned i = 0; i < length; i++)

dstVec[i] += srcVec[i];

return 0;

}

int8\_t ProtoNNF::getPrototype(unsigned index, float \*prototype){

unsigned int d\_cap = this->ldDim;

for(unsigned i = 0; i < d\_cap; i++){

float component = protoNNParam::prototypeMatrix[index \* d\_cap + i];

prototype[i] = component;

}

return 0;

}

int8\_t ProtoNNF::getPrototypeLabel(unsigned index, float \*prototypeLabel){

unsigned int L = this->numLabels;

for(unsigned i = 0; i < L; i++){

float component = protoNNParam::prototypeLabelMatrix[index \* L + i];

prototypeLabel[i] = component;

}

return 0;

}

float ProtoNNF::getProjectionComponent(unsigned i, unsigned j){

unsigned int d = this->featDim;

return protoNNParam::ldProjectionMatrix[i \* d + j];

}

float ProtoNNF::rho(float\* labelScores, unsigned length) {

float maxScore = -FLT\_MAX;

float maxIndex = 0;

for(int i = 0; i < length; i++){

if (labelScores[i] > maxScore){

maxIndex = i;

maxScore = labelScores[i];

}

}

return maxIndex;

}

float ProtoNNF::predict(float \*x, unsigned length,

int \*scores) {

unsigned m = this->numPrototypes;

unsigned int d = this->featDim;

unsigned int d\_cap = this->ldDim;

unsigned int L = this->numLabels;

float gamma = this->gamma;

if (length != d)

return -1.0;

float x\_cap[d\_cap];

float y\_cap[L];

float prototype[d\_cap];

float prototypeLabel[L];

float weight = 0.0;

for(unsigned i = 0; i < L; i++){

y\_cap[i] = 0.0;

}

denseLDProjection(x, x\_cap);

for(unsigned i = 0; i < m; i++){

// at this stage, we are holding a feature vector

// its LD projection and a prototype in memory

getPrototype(i, prototype);

weight = gaussian(x\_cap, prototype, d\_cap, gamma);

getPrototypeLabel(i, prototypeLabel);

scalarVectorMul(prototypeLabel, L, weight);

vectorVectorAdd(y\_cap, prototypeLabel, L);

}

if (scores != nullptr)

for(int i = 0; i < L; i++)

scores[i] = (int)(100000 \* y\_cap[i]);

return rho(y\_cap, L);

}

int8\_t ProtoNNF::getErrorCode(){

return this->errorCode;

}