Exploration of Edge Machine Learning-based Stress Detection Using Wearable Devices

by

Sang-Hun Sim

A Thesis Presented in Partial Fulfillment
of the Requirement for the Degree
Master of Science

Approved April 2022 by the
Graduate Supervisory Committee:

Ming Zhao, Chair
Nicole A. Roberts
Jia Zou

ARIZONA STATE UNIVERSITY

May 2022

ABSTRACT

Stress is one of the critical factors in daily lives, as it has a profound impact on performance at work and decision-making processes. With the development of IoT technology, smart wearables can handle diverse operations, including networking and recording biometric signals. Also, it has become easier for individual users to self-detect stress with recorded data since these wearables as well as their accompanying smartphones now have data processing capability. Edge computing on such devices enables real-time feedback and in turn preemptive identification of reactions to stress. This can provide an opportunity to prevent more severe consequences that might result if stress is unaddressed. From a system perspective, leveraging edge computing allows saving energy such as network bandwidth and latency since it processes data in proximity to the data source. It can also strengthen privacy by implementing stress prediction at local devices without transferring personal information to the public cloud.

This thesis presents a framework for real-time stress prediction using Fitbit and machine learning with the support from cloud computing. Fitbit is a wearable tracker that records biometric measurements using optical sensors on the wrist. It also provides developers with platforms to design custom applications. I developed an application for the Fitbit and the user's accompanying mobile device to collect heart rate fluctuations and corresponding stress levels entered by users. I also established the dataset collected from police cadets during their academy training program. Machine learning classifiers for stress prediction are built using classic models and TensorFlow in the cloud. Lastly, the classifiers are optimized using model compression techniques for deploying them on the smartphones and analyzed how efficiently stress prediction can be performed on the edge.

ACKNOWLEDGEMENTS

First of all, I appreciate Dr. Ming Zhao for offering me the opportunity to participate in the project and giving clear guidance for my thesis direction. I also appreciate Dr. Nicole Roberts for your attention to my thesis work, including helpful advice linking psychology and computer science areas and cooperation with the police academy for the project. I also thank Dr. Jia Zou for the direction of my thesis work. I'd also like to thank Tara Paranjpe and Allen Lin, the undergraduate students collaborating with me with creative and sharp ideas and contributions to this project. I also express my gratitude to the VISA lab and the National Science Foundation - Award CNS-1955593 for funding my research. And I also thank the Republic of Korea and the Army for the opportunity and funding to pursue a master's degree in the United States as a military officer. Last and foremost, I appreciate my family, Min Young and Jiah, for always supporting me and being my driving force.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Motivation

Stress often occurs when we get pressured by events that overwhelm our capacity to deal with it. It is also known to be inversely related to job performance at the workplace [5] as it also negatively affects on decision-making process [6]. This thesis focuses on the stress of police officers in the law enforcement occupation who behave and make decisions for the public society. However, policing is a stressful job that encounters many unexpected threats and undergoes challenging demands from the organization [7]. Seizing and handling stress symptoms in advance is critical as the acute stress becomes chronic and induces the loss of society by giving us more severe results, such as loss in performance or recruiting costs [8]. Thus, anticipating stress in advance and taking preemptive reactions to stress is helpful to not only police officers but also the public society.

With the development of IoT technology, it became much easier for individual users to measure stress by themselves using smart wearable devices since they can now network to the Internet and record biometric signals. Self-stress detection enables one to get alert to stress right in time. From the system perspective, one way to implement self-stress detection is to use the server running in the cloud to operate all the tasks and provide the results to the user device. However, implementing stress prediction on the device makes it even faster. The main challenge is that the device has limited computing resources. Thus, we need to transform the models to be suited for resource-constrained devices using model optimization techniques.

1.2 Objectives

This thesis aims to build a framework for real-time stress prediction based on edge machine learning. To accomplish this goal, we set four research questions as follows:

- Can commercial wearable smartwatches provide validated datasets for stress prediction? If yes, how are we going to implement the data collection protocol?
- How effectively do the machine learning models recognize stress, including neural networks? What kind of and how to build stress prediction models?
- Is it feasible to implement stress prediction models on Fitbit's companion? What processes are needed for the Fitbit development platform?
- How efficiently does the model optimization reduce the latency for real-time stress detection? What kind of model compression technique is the most ideal?

1.3 Methods

This thesis suggests the framework for real-time stress management using Fitbit, machine learning, and edge computing. First, we cooperated with Phoenix Regional Police Academy to collect heart rate and stress data needed to build stress prediction models. The participants were the police cadets who participated police academy training program, in which they must conform to a more strict lifestyle without individual actions. The duration of the data collection was four months, and 15 cadets ended up providing the data until the end of the training. Most of the cadets were male and were in their early 20s. We periodically visited the academy during the data collection stage to inform them to update the application and get feedback from them.

There are two types of data needed for stress prediction methods, heart rates and stress levels. We assigned Fitbit Versa 3 to each cadet individually for heart rate recording and asked them to wear them during the training sessions. Fitbit reads and stores heart rates data in its repository by itself, so we did not need to consider a private database. However, for the stress levels data, we programmed Fitbit to generate prompts with five-stress-level buttons when their heart rates go above their resting heart rates by 35% percent for 2 minutes. Since the stress level is the custom type data we created, we considered utilizing a separate database. We provided the cadet's mobile phone with the endpoint to reach our database using Amazon Web Services. To monitor the overall data collection process, we created a web application that has a dashboard showing battery, syncing status, how cadets entered stress levels, etc. The application also has the interface for downloading heart rates and stress data in CSV format.

Next, we built classification models using machine learning based on our established dataset. Since the dataset is already labeled by the stress levels entered by the cadet, we considered supervised learning. We employed binary classification methods (not stressed vs. stressed) using both classic models and a neural network. Then, we deployed the stress classification models on mobile devices so that they process stress prediction in proximity to the user without sending the data to the cloud. We also considered applying quantization to the neural network to reduce the weight size and compared performance with the on-Cloud system. To compare the cloud and the mobile, we not only implemented stress prediction methods on the mobile application but also created a server running in the AWS data center. This thesis exemplifies the effectiveness of model compression techniques regarding model size and latency for stress prediction.

1.4 Outline

The subsequent chapters of this thesis are as follows. Chapter 2 elaborates on the background of three main concepts: wearable devices as a health management tool, fundamentals of machine learning, and the concept of edge machine learning. Chapter 3 goes through literature reviews that are divided into two main parts: machine learning-based stress prediction and deploying machine learning on edge devices. Chapter 4 describes the data collection protocol we established, including the Fitbit application, web application, and cloud server. Chapter 5 explains how we build stress prediction models using machine learning. Chapter 6 introduces stress prediction on the edge devices and compares results between on-cloud and on-device. Lastly, chapter 7 summarizes what we learned and suggests future works that can extend the research scope.

Chapter 2

BACKGROUND

2.1 Wearable Devices as a Health Management Tool

Wearable devices have already permeated every aspect of our lives. With the evolution of the Internet of Things (IoT) industry, the computing power of wearable devices themselves has also improved. The study conducted by IDTechEX specifies that both market value and the number of units sold have increased since 2015 [1]. Wearables are now being used in various fields of society, such as the healthcare industry or automated systems. One of the most widespread applications of the wearable is the health care and fitness [9]. As people are willing to self-check their health status on their own, wearable usage has more than tripled over the past few years [10]. Wearables are equipped with ordinary functions like measuring time but also with sensors to read human biosignals with a networking capability [11].

One primary function of the wearable fitness tracker is a sensor to record biometric signals. For instance, Fitbit has optical sensors at the bottom of the clock face. The main advantage of using smart wearable devices is that they measure biometric signals without interfering with the user's daily life. They minimize disturbing ordinary living patterns because they are merely put on the wrist. The additional function is networking capability with either Bluetooth or WiFi, meaning that they can now exchange data with the outside of the watch. Fitbit, for example, is compatible with smartphones via Bluetooth connection and transmits data by using a bridge of the smartphone.

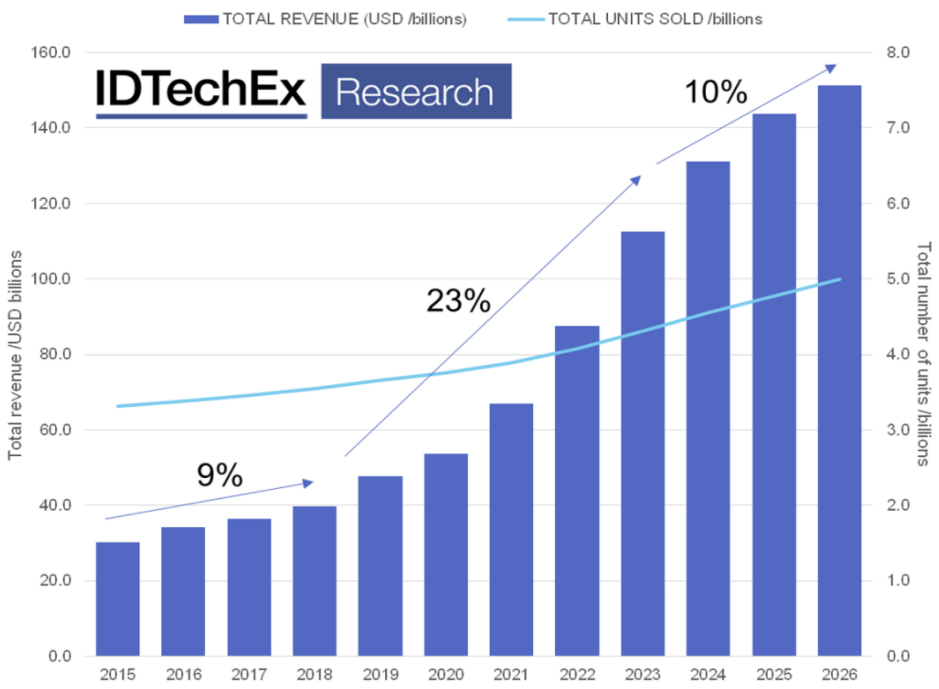


Figure 2.1: Global Wearable Market Forecast [1]

2.1.1 Correlation Between Heart Rates and Stress

The human body system responds to stress in a way that we can measure quantitatively. The Autonomic Nervous System (ANS) is triggered when someone gets stressed. ANS has two parts, the sympathetic nervous system, and the parasympathetic nervous system. The former is related to intense activities, for example, and increases heart rates, while the latter decreases heart rates, especially in resting [12]. There are various devices that can measure heart rate variability. By measuring heart rate variability, it has been proved that stressed and unstressed status indicate different quantitative heart rate value [13]. They demonstrated that the mean and standard deviation (SD) of the heart rates and heart rate variability are differed by the condition: The mean R-R interval and mean SD was higher when resting.

2.2 Fundamentals of Machine Learning

Machine learning creates computer programs that learn and improve from a gathered dataset. [14] elaborates on machine learning techniques that it is a statistical and automated framework extracting valuable pieces of information from the dataset by statistically analyzing the data patterns. The ultimate goal of machine learning is to build a model which interprets "regularities" and "patterns" of the data and "generalizes" to the unseen data [15]. The machine learning model generates different outputs depending on the scenarios. Supervised learning, for example, constructs predictive or decisive models with a dataset labeled, while unsupervised learning develops clustering forms with a unlabeled dataset.

2.2.1 Machine Learning Processes

Machine learning processes can be divided into four stages roughly: 1) data acquisition, 2) data preprocessing, 3) training, and 4) evaluation. The data acquisition stage needs to determine a problem and requirements to specify which type of data would be the most effective. We need to clarify which kind of algorithm is adequate for our goal, considering the dataset's characteristics. Regarding data collection, one way to acquire data is to exploit already collected data open to the public source. This way alleviates the burdens of collecting data and guarantees validated datasets. Another way is to collect data directly. It can reflect our problems more effectively.

Next, there are three typical methods for data pre-processing. 1) It detects and deletes the outliers outside the trust bound. Detecting outliers can purify the dataset and train the models with more invariant input data [16]. 2) It also transforms raw data into a form that the machine learning model can employ more efficiently. For

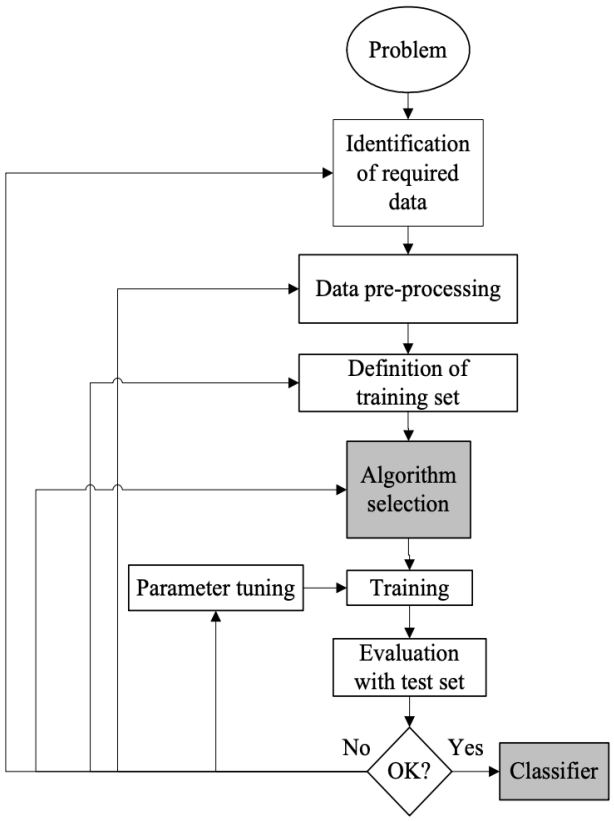


Figure 2.2: The Processes of the Supervised Learning [2]

instance, the standard scaling makes every column of the dataset maintain the same distribution with mean zero and unit standard deviation. It diminishes the operations time. Lastly, 3) It removes nonessential and duplicative parts of the dataset by extracting a subset of features [2]. It reduces the complexity of the models and drives them to proceed quicker.

In the training phase, deciding on appropriate machine learning algorithms is crucial as their performance varies depending on the type of algorithms. Considering the dataset's characteristics we collected, this work is more suitable for the classification algorithm in supervised learning. Our dataset consists of $\{(X_1, y_1), (X_2, y_2), .., (X_m, y_m)\}$, where X_is are the input vectors $X_i = \{x_{i1}, x_{i2}, ..., x_{in}\}$ that has i number of columns in the dataset. The input vector is in the n-dimensional space, where n is the num-

ber of features extracted from heart rates. The output y have the discrete values, where $y = \{1, 2, 3, 4, 5\}$ and indicates the stress levels. The foremost goal of this work is to create classifiers that have the parameters of $f(X) = y$, which minimize the error rates. The representative classifiers for our dataset are K-Nearest Neighbors, Adaptive boost(Adaboost), XGBoost [17], Random Forest, Decision Tree, and Feed Forward Neural Network from TensorFlow [18].

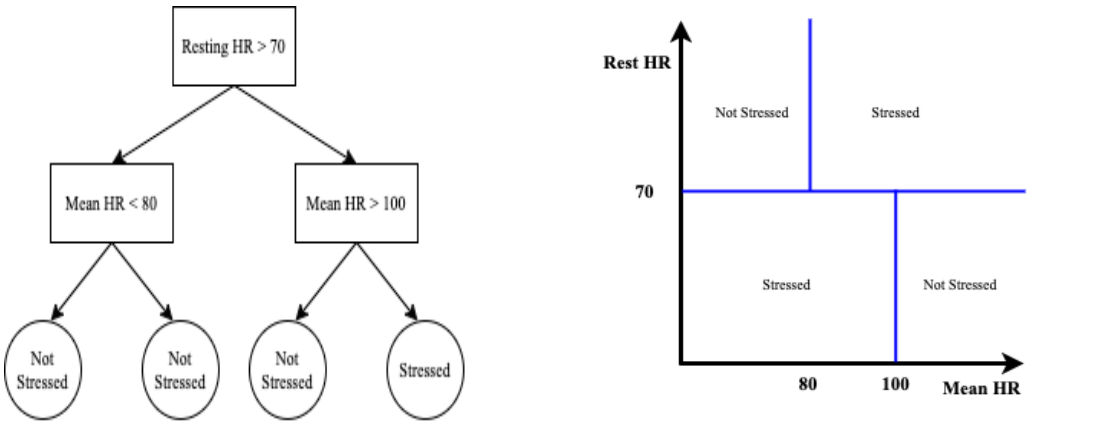
Upon the model selection, the dataset must be split to evaluate it. Typical ratios are training data (60%), validation data (20%), and test data (20%). Suppose the size of the data is incomplete. In that case, cross-validation is performed. Cross-validation divides the dataset into k number of equal-sized segments, uses one segment of the dataset as verification set and the rest as the training set, and repeats it k times while changing the verification set. After training and validation, the model examinations with test data as it encounters unseen data to prove its prediction capability for the new instances.

2.2.2 Classification Algorithms

Our problem needs classification algorithms since it is to build classification models that classify the stress with given dataset. Also, considering that our dataset provides both inputs and outputs, it follows supervised learning algorithms. Standard classification algorithms are tree-based, ensemble, and deep learning algorithms. Tree-based algorithms classify the input data with given rules established by the training dataset. Ensemble algorithm merges insignificant models into one robust model. Ensemble type is divided into two sub-kinds: Bootstrp Aggregating (Bagging) and Boosting. Lastly, deep learning is the state-of-the-art method that uses neural networks to discover the best parameters that minimize the loss.

Decision Tree

Decision Tree is an algorithm that classifies the input vector to the discrete target value by using the tree-like decision rule. The decision rule is a combination of rules for predicting the input data patterns, and the rules are based on the dataset's attributes. Figure 6.3b shows an example of the decision tree. It comprises the root node, intermediate nodes, and leaf nodes. The root node is the node that every input instance encounters at first, and it should be the best splitter of the training set [2]. Every time the input instances pass the node, they are redirected to the following nodes until they get the final class. As shown in Figure 6.3f, the output space is partitioned corresponding to the classification by the decision rule. A good Decision Tree classifier has good discrimination, and it is judged by impurity, representing homogeneity of the classes in each node. The algorithms for impurity calculation are Gini impurity and Entropy.



(a) Example of the Decision Tree (b) Partition Generated by the Decision Tree

Figure 2.3: A Decision Tree and Partitioning

Random Forest

Random Forest is a familiar example of the ensemble bagging algorithm. The ensemble bagging algorithm utilizes Bootstrap to create n number of the subsets of the datasets by sampling randomly. It trains n models with the datasets produced by Bootstrap. We call these models weak models. The output is determined by voting among the models. Figure 2.4 shows the overall schemes of Random Forest. Random Forest uses as many decision tree models as the number of the subsets of the dataset it resampled. One substantial advantage of Random Forest over the decision tree algorithm is that it can overcome overfitting because it resamples the dataset with randomly selected features. Also, when it comes to resampling, it allows duplication of the feature selection.

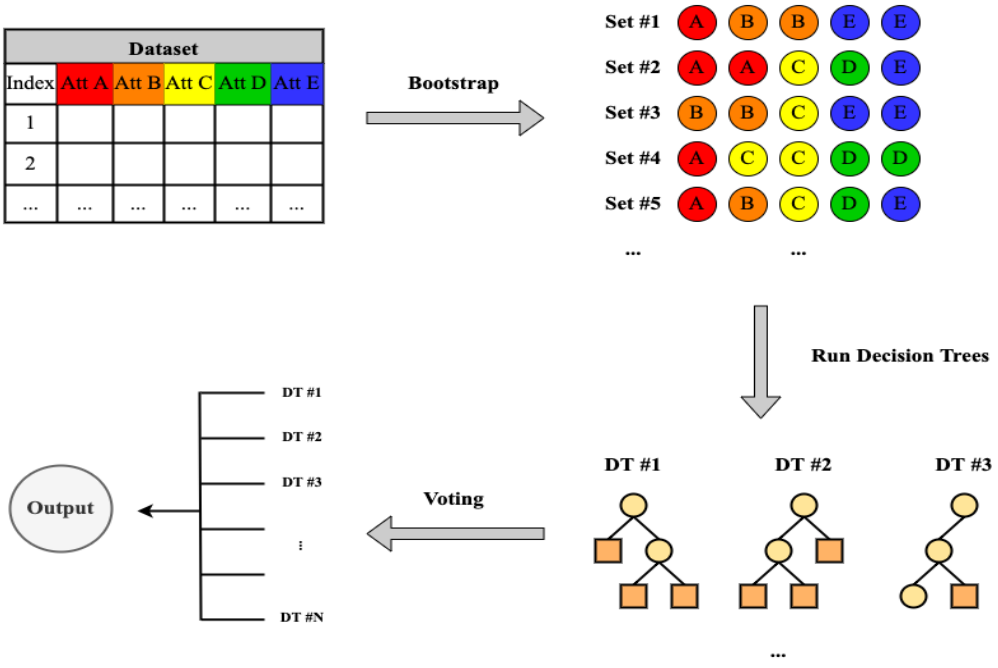


Figure 2.4: Scheme of Random Forest

Adaptive Boost

Adaptive Boost (Adaboost) belongs to the ensemble boosting algorithm. Boosting is similar to Bagging in that it also resamples the dataset and creates multiple weak classifiers. However, unlike Bagging, Boosting utilizes the weighted samples for the following classifiers. Also, while Bagging's models are independent, Boosting's classifiers are interactive among each other—boosting grants the misclassified targets the weights for the next model to recognize and update appropriately. Figure 2.6 shows the training process in Adaboost. Targets misclassified in the leftmost model are weighted, so the next model updates the boundary line mainly on targets with larger weights. It also weights the more precise classifiers. Thus, the final classifiers are determined by voting, but each vote has different weights.

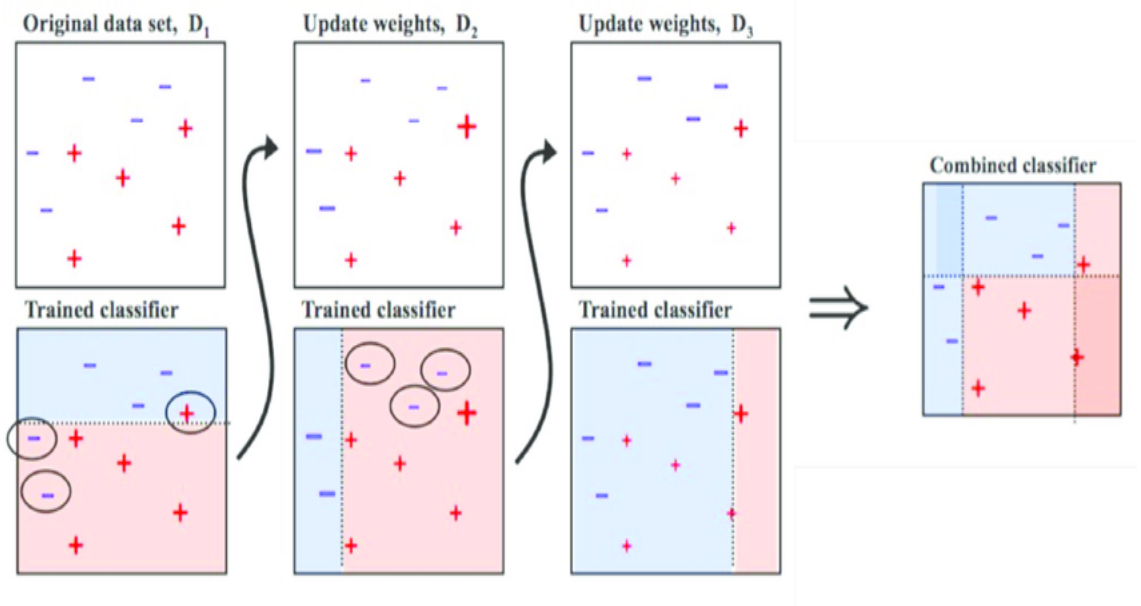


Figure 2.5: Training Process in Adaboost [3]

K-Nearest Neighbor

K-Nearest Neighbor (KNN) algorithm classifies data based on the similarity between a novice sample and the neighbors. The two general similarity measurements are the normalized value of Euclidean distance and the cosine similarity. K implies the number of the closest neighbors, meaning that the model decides the output of the novice sample by voting among the nearest k neighbors and chooses the majority's class. It uses relatively simple algorithms and is easy to implement. However, it is also called a lazy model because it delays the learning process until it embarks on classifying new data. Also, it is expensive in terms of algorithm complexity in that it calculates all the distances from the novice sample to the training samples in the brute force manner. Moreover, distance measurement will be ineffective when features are above a certain number. The KD tree is used for a more efficient approach to avoid calculating all the distances.

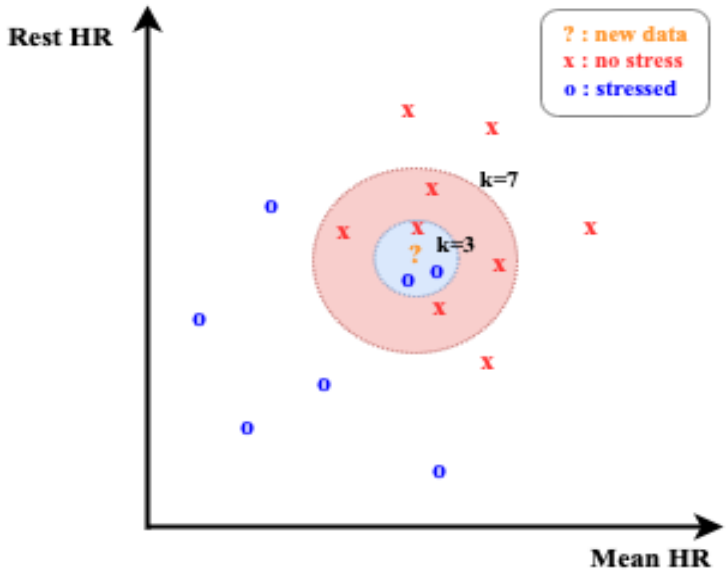


Figure 2.6: Example of the K-nearest Neighbor Algorithm

2.2.3 Evaluation Metrics

The model evaluation is performed on a testing dataset, which the model has not seen during the training stage. The classification problem has to consider not only the mean squared error between the ground truth and prediction but also the other metrics, such as accuracy, confusion matrix, or F1-score. Accuracy is the metrics indicating how many of the predictions match the ground truths.

$$Accuracy = (True\,Positive + True\,Negative) \,/\, Total\,Predictions \tag{2.1}$$

However, the accuracy-only method cannot address the imbalanced dataset correctly. For example, let us assume the dataset contains 90% of label 0 and 10% of label 1. Even though the model can return the data as of only 0, the accuracy of the model is still 90%. Therefore, it is needed to be evaluated with additional types of evaluation metrics, like confusion matrix. The confusion matrix shows how the model classifies the test instances well by showing true-positive (TP), true-negative (TN), false-positive (FP), and false-negative (FN).

		Prediction	
		Predicted Positive	**Predicted Negative**
Actual	**Actaully Postive**	**True Positive (TP)**	False Negative (FN)
	Actually Negative	False Positive (FP)	**True Negative (TF)**

Figure 2.7: Confusion Matrix

Based on these four components of the confusion matrix, we can measure precision and recall. Precision is the ratio of the positive predictions that are actually positive, while recall is the ratio of the positive ground truths that are predicted as positive. However, since precision and recall are in complementary relationships, as one increases, the other decreases. We call this as precision-recall trade-off.

$$Precision = \frac{TP}{(TP + FP)} \tag{2.2}$$

$$Recall = \frac{TP}{(TP + FN)} \tag{2.3}$$

The calculation of the F1-score is from the harmonic mean of precision and recall. The purpose of the F1-score is to merge precision and recall into one evaluation metric. F1-score provides us with insights to analyze the performance of the model. The higher score indicates precision and recall are high together, while the lower score means both are low. Also, if the score is around 50%, then it means the opposite trend that one is high while the other is low.

$$F1score = 2 * \frac{Precision * Recall}{(Precision + Recall)} \tag{2.4}$$

2.3 Concept of Edge Machine Learning

With the advancement of wireless communication technology and wearable devices, the interest in edge computing on wearable has emerged considerably. In the meantime, wearable devices generate enormous amounts of data in such a short period of time. However, this wearable's data generating induces network bottlenecks due to the limited network bandwidth. Edge computing is to proceed computation at the edge devices, where data is originated [19]. Instead of employing cloud-central systems that burden all the responsibilities [20; 21], edge computing relieves the cloud's role and process data locally. For example, many IoT applications, such as language translation or image recognition, require real-time responses by deploying machine learning on edge devices. Yet, it is still challenging for such devices to handle a large amount of data with limited resources.

2.3.1 Advantages of Edge Computing over Cloud-only Computing

There are several challenges when considering the cloud-only system. First, the application must establish a stable network connection to the cloud server [22]. Numerous unpredictable variables interfere with network connections, hindering data exchange on time. Second, the cloud may not be able to handle the amount of data that edge devices generate [22; 23]. As the number of IoT equipment such as wearable devices grows, the amount of data generated increases as well. It will induce the overload of the cloud, which becomes unserviceable afterward. Third, the cloud may not interpret the personal data accurately [22]. A global model trained in the cloud may not interprets individual local data properly because it has learned with a mixed set of data. Lastly, it has to take risks of compromising privacy if data is transferred to public cloud [20; 22; 23].

On the other hand, edge computing can alleviate the aforementioned drawbacks of the cloud-only system. First, edge computing can save computing resources, like network bandwidth, latency, and power consumption [24]. Since the data is processed in proximity to the device, the system does not need to transfer the data over the Internet as often as the cloud-only system. Next, it enables faster inference from the machine learning models than the cloud-only [22]. Applications like natural language processing or image recognition require frameworks with the real-time response from the server. With edge computing, the processing model is deployed on the device and gives faster answers to the user. Lastly, it can preserve the confidentiality of local data by keeping them in private storage still. The device only has to send already-processed data without exposing sensitive raw data to the public cloud. However, implementing edge-based machine learning systems is still challenging due to limited resources. It must go through model optimization processes to transform the models suitable for the edge device.

2.3.2 Model Optimization Techniques

Model Redesign

For classic models, like K-NN or tree-based algorithms, we can diminish the model's size by reducing dataset dimensions with techniques such as PCA. They filter out redundant features from the correlation matrix. The model re-design can be applied to deep neural networks as well. We can consider lowering the number of parameters for the neural networks while preserving their accuracy [25]. SqueezeNet [26], for example, is the convolutional neural network (CNN) with one-fifty times of weights to the standard CNN models but preserves its accuracy. Its strategy is to reduce the kernel size by 1x1 and the input channel to 3x3 filters.

Model Compression

The following contribution is model compression. With the combination of the model compression techniques, including pruning, quantization, and Huffman Coding, the model size can be decreased by 35x to 49x without losing accuracy [27]. This thesis uses only quantization for optimizing TensorFlow models. First of all, **pruning** reduces the complexity of the model by deleting insignificant weights. There are two types of pruning, unstructured pruning, and structured pruning. Unstructured pruning converts unnecessary weights to zero individually. However, it doesn't speed up the inference time as it still maintains the sparse matrix. Structured pruning prunes the entire area, like a channel, and removes matrix operations for the pruned target. While unstructured pruning retains good accuracy but utilizes hardware inefficiently, structured pruning accelerates hardware efficiently but loses accuracy [28]. Next, **weight clustering** lowers the number of individual weights by grouping weights of each layer into N clusters and sharing the representative value. [27] used k-means clustering algorithms to make groups of weights that have similar values and set the same weight to share for a cluster.

The purpose of **quantization** is to diminish the inference time of the neural network. The majority parts of the neural network are matrix multiplications of weights neurons and the activation functions. To raise the accuracy, the ordinary network is represented by 32-bit floating points. Of course, it is challenging for edge devices to operate with 32-bit floating points due to the lack of resources. Thus, we use quantization to reduce the bits to smaller units, like 8-bits or 4-bits. The main advantage of quantization is that the quantized model utilizes lower memory bandwidth. And it is followed by lower power consumption and storage. It also reduces the size of models by 4x when bits are shrunk to 8-bits from 32-bits [25].

However, it is inevitable that the accuracy of the model will get lost because it converts float 32-bits to int8 format, which can represent 256 indexes at maximum. Post-training quantization is to quantize the model after training with floating 32-bits. This approach is suitable for when the size of parameters is large because the decrease in accuracy is relatively low. Quantization-aware training is to quantize during the training by simulating quantization application so that it can minimize the decrease of the model's accuracy.

Hardware

Hardware is also a crucial part of running machine learning models on edge devices. For example, users can leverage **GPUs acceleration**. GPUs hasten the operation speed by parallel programming with more cores and memory bandwidth than CPUs. TensorFlow [18] is the representative framework that can control GPU usage by programming. [22] utilized GPUs to accelerate hardware in the mobile device for training DNN by porting to Tensor. There are also application-specific integrated circuits (**ASIC**) such as Google's TensorFlow Processing Unit for faster inference [29]. TPU is a customized chipset developed by Google for energy-efficient operations, considering that training operations and inference operations have different scales. While training operations require 32-bits floating bits, it is sufficient for inference to be in 8-bits floating bits. TPU has components that quicken the matrix multiplication, including 655,36 multipliers, 256 accumulators, and the unified buffer that can hold intermediate operations and can implement activation functions in parallel [30].

Chapter 3

LITERATURE REVIEW

This chapter gives survey studies on the literature related to our work. It is divided into two parts: studies that used machine learning for stress prediction and studies that applied machine learning on Edge. The former part elaborates on the author's approaches made for machine learning-based stress prediction, such as the data source, features extracted, stressors, the type of machine learning model, and the accuracy. Subsequently, the second part investigates how the author made contributions to deploying machine learning on Edge.

3.1 Machine Learning Based Stress Detection

Today's wearable devices can construct datasets for the machine learning process by recording biometric signals. In this work, we collected data from Fitbit that can record heartbeats in about 5 to 10-second-granularity. It can also read other measurements, such as calories burned or steps. Although Fitbit cannot read more precise heart rates variability, like Electrocardiogram (ECG), several studies have already proved the usage of Fitbit for stress prediction with machine learning classification [31; 32; 33]. The common thing among these studies is that they extracted statistical features from raw data and reduced the feature matrix's dimension. Shrinking feature sets can also eliminate redundant features to increase the efficiency of the prediction processes. They also considered defining stressors by themselves using surveys or customized methods.

Paper	Data Source	Features	ML Model	Stressor	Accuracy
[31]	Fitbit	BMI, Heart Rates, Sleep Patterns, Physical Activities, Demographic Data	Probit, Logistic Regression, Log-Log	Survey	N/E
[32]	Fitbit	Calories, Steps, Heart Rates, Sleeps Resting Heart Rate	KNN, SVM, Decision Tree	Surveys	81.70%
[33]	Custom App, Fitbit	Workplace Acitvities, Heart Rates, Sleep Pattern	KNN, Naive Bayes, J48, Adaboost, Random Forest	Prompts	78%
[34]	Wearable Sensor	HRV features, GSR features, Accelerometers	J48 Decision Tree, Bayes Network, SVM	Stroop Test	95.21%
[35]	SRAD [36]	Foot GSR, Hand GSR, Heart Rates	Conv. Neural Net	Driving Status	95.67%
[37]	WESAD	Statistical features of 10-second window	KNN, RF, SVM, LDA, Adaboost	Questionnaires (pre-labeled)	84.17%
[38]	WESAD	Statistical features, peak frequency, slope of signal	KNN, RF, SVM, LDA, Adaboost, DT, Feed-forward Net	Questionnaires (pre-labeled)	95.67%

Table 3.1: List of Literature About Machine Learning Based Stress Detection

[31] utilized in-body information, physical and sleep-related data, heart rates. All data is retrieved from Fitbit's data archive. They also considered using demographic attributes such as age or gender. They extracted over 30 features by calculating statistical features, such as mean or standard deviation, and applied PCA to reduce the dimension of the dataset. For the stressor, it used the PSS survey. They use only regression models, including logistic regression, probit model, and complementary log-log model. It has not involved the accuracy for each model. Instead, it concludes with AIC metrics, which compare the performance of the used model; the Probit model is most suitable for their dataset. This work proved that Fitbit could provide

data to predict stress and tried to contextualize the origin of stresses. In addition, they sought to reduce the model size by eliminating redundant features using PCA. However, it only used the daily surveys to measure stress and did not measure the accuracy and F1-scores of the models.

The study conducted by [32] collected five data types, including heart rates, resting heart rate, sleep patterns, calories, and steps data, and extracted 17 statistical features such as mean, standard deviation, and summation of heart rates. For stress measurement, it used multiple surveys, Perceived Stress Scale (PSS), General Self-Efficacy Scale (GSE), and General Survey. The models it employed are K-Nearest Neighbor, Support Vector Machine (SVM), and Decision Tree, and it ran the models separately for each survey. The best accuracy it reached is 81.70% from SVM. This work inspired me to use Fitbit for stress prediction as its dataset contains only data recorded by Fitbit. Also, they proved that different survey responses indicate different stress levels. However, they have tried with only three classic models and not used deep neural networks, which is more accurate. Moreover, they have not considered the real-time aspect.

[33] aimed to measure stress among the office workers using heart rates and sleep patterns recorded by Fitbit. They also created a custom application to collect office activities, such as the number of keyboard strokes. The application generated the survey every 210 minutes for stress measurement, and the participants entered the stress levels they considered. It implemented PCA, correlation analysis, and feature importance for data processing to filter out redundant features and finalize the feature set. Classic models, including KNN, Naive Bayes, Random Forest, J48, and Adaboost, were run for classification, and the best accuracy was 78% from Random Forest. Instead of using surveys, this work implemented a novel method for stress measurement using the custom application. The user enters the stress level directly.

It also imported SMOTE [39] to overcome the imbalance of the dataset. However, it should also deploy a deep neural network tuned in custom for its dataset instead of classic models, which are not suitable for the user-specific dataset in the long term.

There are other data sources for stress prediction. Instead of using wearable smart devices, like Fitbit, we can consider using different wearables that require subjects a bit static but read biometric signals in more depth, such as ECG, accelerometers, or GSR [34; 35]. WESAD [40] is also a popular method for stress prediction since it contains multimodal data collection, such as 3-axis accelerometers, ECG, EDA, EMG, RESP and temperature, recorded by a wrist and chest-worn device. [38; 37] have implemented three-class and binary classification models with this dataset for stress prediction.

[34] researched for methods to detect stress based on activity recognition. Using sensor-reading wearable equipment, they collected three types of sensor data, ECG, GSR, and accelerometers (ACC), from 20 participants composed of students, faculty, and staff from their university. They extracted features representing heart rate variability (HRV) from ECG data and skin conductance variables from GSR data. They also calculated the mean and standard deviation of the ACC. They built an application to test participants with a test interface to define stress levels. They segmented each data corresponding to their current status to distinguish activities occurring stress like seating or walking. In addition, to analyze which features affect more on which activity, they excluded one of all types of features. They concluded that ACC data has more influence on active motion, while physiological data is effective in both inactive and active status. Lastly, they implemented classification algorithms, such as Decision Tree and Bayes Network, and reached the best accuracy of 92.4% from Decision Tress with all types of data included.

[37] conducted study using WESAD [40] dataset. The segmented dataset in the 10-second window and calculated statistical features. The best accuracy they got from Random Forest is 84.17% for binary classification and 67.56% for three-class classification. [38] used the same dataset and extracted in a similar way by extracting statistical features. However, it reached 93.20% from a feed-forward neural network.

Lastly, some studies consider neural networks for stress prediction [38; 35]. The first study built a feed-forward neural network for both binary and three-class classification. The network consists of two hidden layers. They applied different activation functions for the output layer on each case. They put a sigmoid function for binary classification that returns a value between 0 to 1. At the same time, they applied a softmax function for three-class classification, which produces three probability values of the three classes so that the largest is the likely target of the input data.

[35] considered using deep neural network, convolutional neural network. It employed a public dataset called Stress Recognition in Automobile Drivers(SRAD) [36], which contains three time-series measurements, including Foot GSR (FGSR), Hand GSR (GSR), and Heart Rate. They extracted 10-second and 30-second windows segments, respectively, when data processing. One thing to notice is that they created a recurrence plot (RP) from the time-series sampling to convert the sequence data to the image. They defined stress levels per where subjects drive. Using the VGG16 [41] model, they extracted features of RPs and generated feature vectors by flattening the feature maps. As a result, they achieved the best accuracy of 95.67% with 30-second windows. In summary, this study shed light on using CNN models for time-series data classification by converting them into images. Also, it proves such a short period of data is sufficient for stress measurement, which is suitable for the real-time framework. However, it does not use any user's response, such as surveys or prompts.

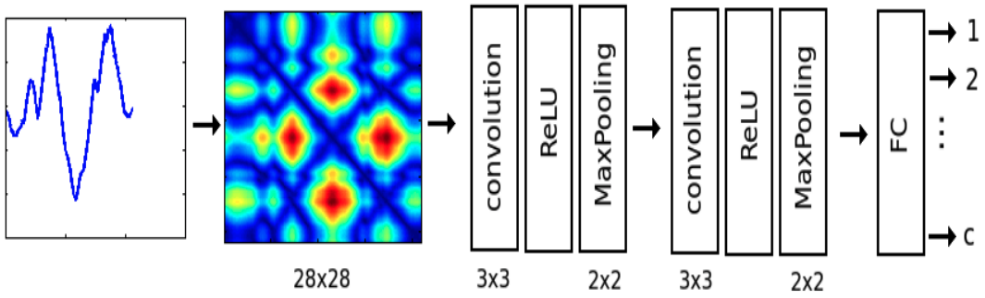


Figure 3.1: Example of the Usage of the Recurrent Plot with CNN [4]

3.2 Deploying Machine Learning on Edge Devices

Many studies have explored the feasibility of leveraging machine learning capability on edge devices. These shed light on applying edge machine learning to our edge device, Fitbit. Some also tell about the efficiency of mobile machine learning by comparison with on-cloud operations.

Paper	Contributions	ML Model
[22]	Extension of TensorFlow, Hardware acceleration, Redesigning the model	CNNs for image classification
[42]	Loading pre-trained model on mobile device	Deep Neural Network, Classic models
[43]	Model compression - Quantization	CNNs for image classification
[44]	Comparison between cloud and mobile inference	CNNs for image classification

Table 3.2: List of Literature About Edge-based Machine Learning

[22] conducted a study that investigates the capabilities of mobile devices for training and inferencing with deep learning models. It first introduces how they ported TensorFlow mobile to enable mobile devices to train the model and utilize GPU for hardware acceleration. They adjusted the interface of TensorFlow mobile by including methods and libraries that are needed for training. Also, they switched the frameworks to accelerate the operations, like convolutional and matrix-matrix

multiplications. It considered testing three CNN models for image classifications with the CIFAR-10 dataset and used four different mobile phones and a server. They also compared the performance among different types of networks, fully connected layer and convolutional layer with the result of that fully-connected layer is more subtle with its width and depth. They found that training operations consume most of the time, especially for the gradient calculation of the backward path. Consequently, they conclude that it is plausible to run both training and inference on mobile devices provided that models' complexity is relieved. They also suggest new approaches for machine learning on mobile, such as federated learning or knowledge transfer.

There is a study that builds a real-time inference framework for fall detection of the elderly [42]. This project has three components: 1) a smartwatch, 2) a smartphone, and 3) the cloud server. The smartwatch monitors and records the accelerometer data and transfers them to the smartphone. The Smartphone has the application calculating for the Fall detection instead of redirecting data to the cloud server. The server is in charge of generating the models with the dataset provided by smartwatches and an external source. They trained SVM, Naive Bayes, and Deep Neural Network (DNN). They infer that DNN outperforms the others because it can catch the feature information more in detail. Even though they do not compare the fall detection performance between on-device and on-cloud, it proves that mobile devices are capable enough to proceed with the machine learning inference.

Also, [43] implemented the frameworks for deep learning inference on the mobile. This framework dynamically decides the inference location and the types of models under the conditions such as desired accuracy or current system status. They mainly focused on which model compression techniques to use, which model to choose for mobile inference, and when to depend on the servers. First, they investigated the results from the model compression by its accuracy and inference time. They proved

that the quantized models have significantly less size of models. Also, they found that loading model time takes most of the inference time except for the 8-bit quantized model. Next, they measured the inference time between different mobile devices. Also, assuming the mobile is not always available, they analyzed the inference time at the edge server according to the type of Internet connection.

Another study conducted by [44] implemented a comparison of the inference performances of three CNN models between cloud-based and on-device with the image-recognition application. For cloud-based inference, they established the server on Apache Server hosted by Amazon Virginia data center for cloud-based inference with both CPU and GPU enabled. The input image has to go through bandwidth from the mobile. On the other hand, mobile-based inference used an Android phone, and they used already-loaded the trained models and images on the device. The evaluation metrics are latency, power consumption, and resource usage. They measure the performance in an end-to-end manner but also break it down into separate steps to analyze more in detail to figure out which consumes more time. They found that the cloud-based approach outperforms the mobile-based. Also, the evaluations are differed by model type in that the Caffe-based model took less time to load model than CCNDroid. Also, they realized that the Android application consumes memory with tasks unrelated to inference tasks, such as garbage collection. The mobile device is also sensitive to the size of the image.

Chapter 4

DATA COLLECTION SYSTEMS

This chapter describes how we established the architecture of data collection. Figure 4.5 shows the architecture and data flows. There are three main functions: (a) Fitbit application, (b) Web application, and (c) Node.js API server. Fitbit application is in charge of data collection right in proximity to users by recording heart rate fluctuation and generating stress level prompts. We built a static web application to provide data visualization and data collection monitoring interfaces. It also holds trained models for the companion to self-predict stress. Lastly, an EC2 instance hosts an API server for on-cloud stress prediction.

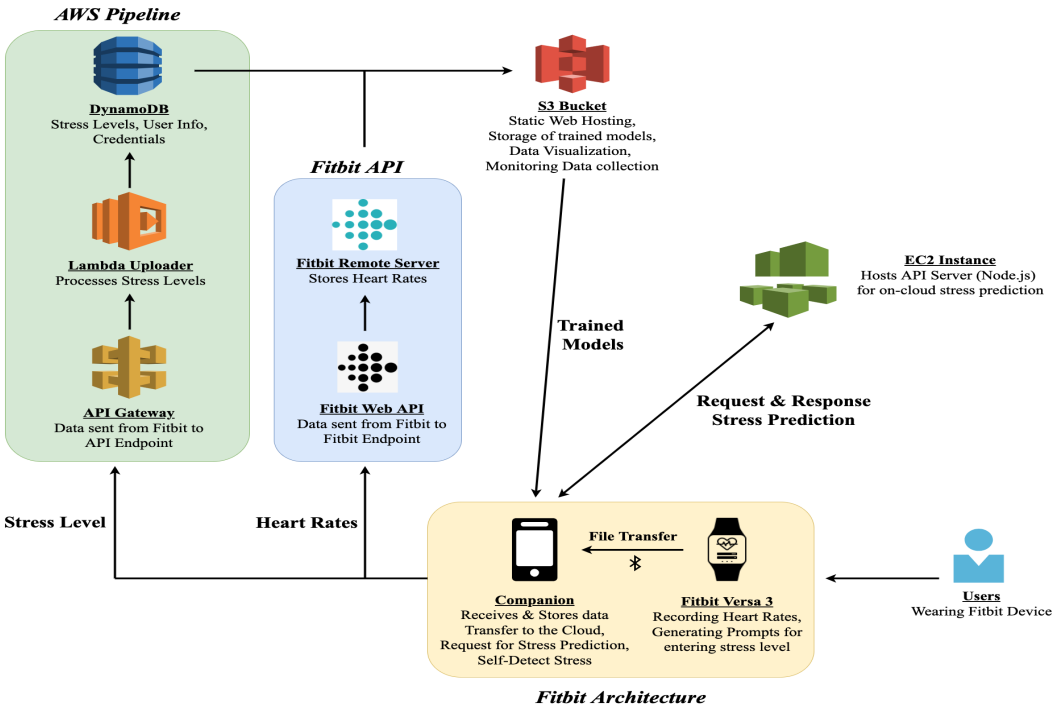


Figure 4.1: The Architecture of the Stress Management

4.1 Fitbit Application

Fitbit supplies a Software Development Kit(SDK) and various Application Programming Interface(API)s to users for developing a customized application. In terms of data recording, Fitbit offers significantly simple processes. Fitbit records and uploads biometric data by default, so it was unnecessary to implement functions for them. Also, we can retrieve data recorded via APIs so that we only need to set individual credentials as parameters to API functions. As this study intended to catch stress occurrence mainly by the heart rate fluctuation, I built an application to detect the patterns indicating stresses by prompting stress level buttons as shown in Figure 4.2. The prompting generation mechanism is when the heart rate goes above the resting heart rate by 35 percent for two minutes, the clock's face changes to the prompted face with four buttons ranging with four levels, 'No Stress,' 'A Little,' 'Moderate,' and 'A Lot.' Users enter their stress levels subjectively. Also, the subsequent prompts are not shown for 30 minutes before the previous prompt to avoid continuous prompts.

Figure 4.2: Clock Face of the Fitbit Application

Basically, Fitbit's network architecture includes the Fitbit watch and the companion. Figure 4.3 shows Fitbit's network architecture. Fitbit needs to be paired with the companion by Bluetooth to push forward the recordings outside the Fitbit Network. Because Fitbit does not have an Internet connection by itself, it depends on the companion for any other operations, like fetching information or storing JSON data, except for recording bio-signals. Upon syncing to the companion, it sends all the biometric signals recorded to the remote server. Fitbit also provides a mobile application on the companion to display statistical data of the recordings by fetching data back from the remote server. While the Fitbit focuses mainly on recording biometric measurements, the companion can do more complex operations, such as fetching data or importing external packages. Fitbit and companion together can build a more elaborate application than Fitbit would do alone as long as the application is designed for mobile devices. On the stress management application we built for this work, once the user enters the stress level, Fitbit keeps the stress input data in the file storage in the CBOR format and sends it to the companion as soon as the Bluetooth connection is established. Companion receives the stress file and concatenates it to the existing data in a key-value format, so the data is preserved in the companion for a while.

Companion is responsible for sending the key-value dataset to the database on the Internet. When the companion receives the file from Fitbit, it converts them into a JSON object using the File API that Fitbit provides. Then it sends the dataset to the database using the fetching function. Since it has way more resources than Fitbit, it can handle a larger dataset size. Another responsibility is to query the device's unique identification (ID). Over 20 cadets are providing their data, so it needs to be distinguishable from each other. The device ID is unique in that it maps the Fitbit account to the device. Companion figure out the device id right after turning it on

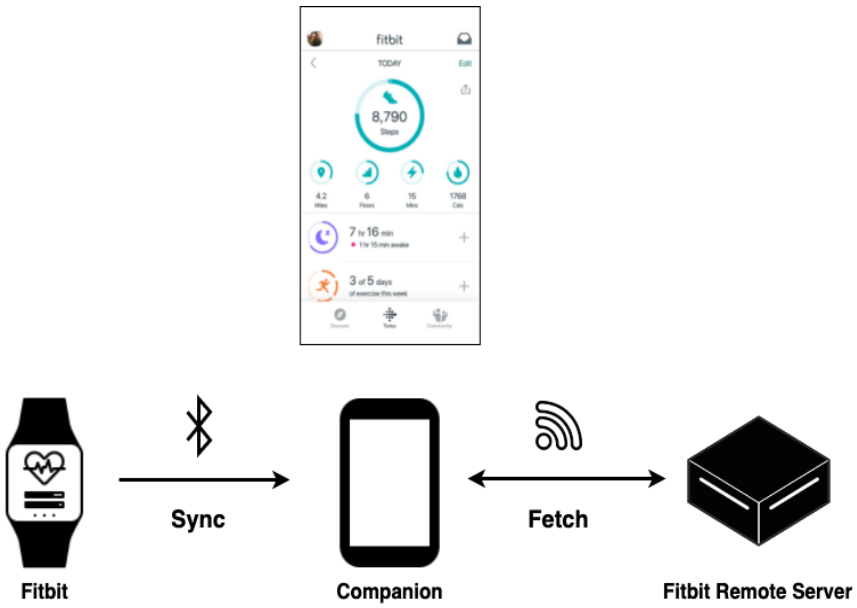


Figure 4.3: Fitbit's Network Architecture

and store them in the storage. Also, the companion can import external libraries such as Node Package Manager (NPM), meaning that the companion is capable of doing more tasks.

In addition to understanding how Fitbit works for recording data, it is also essential to define how to retrieve the data to our end. As mentioned above, Fitbit automatically stores monitored data in its remote server. Fitbit provides several WEB APIs to retrieve the stored data to incorporate them into their application. However, it requires specific authorization credentials, including access tokens and user identification. To validate the access tokens, users must go through the authorization code grant processes, which depends on OAuth 2.0 application, the protocol to allow a third-party user to access the resources. OAuth application creates the client ID and Secret to use them to invoke the access token. And since the access tokens last for only 8 hours, the method of refreshing the tokens also has to be prepared.

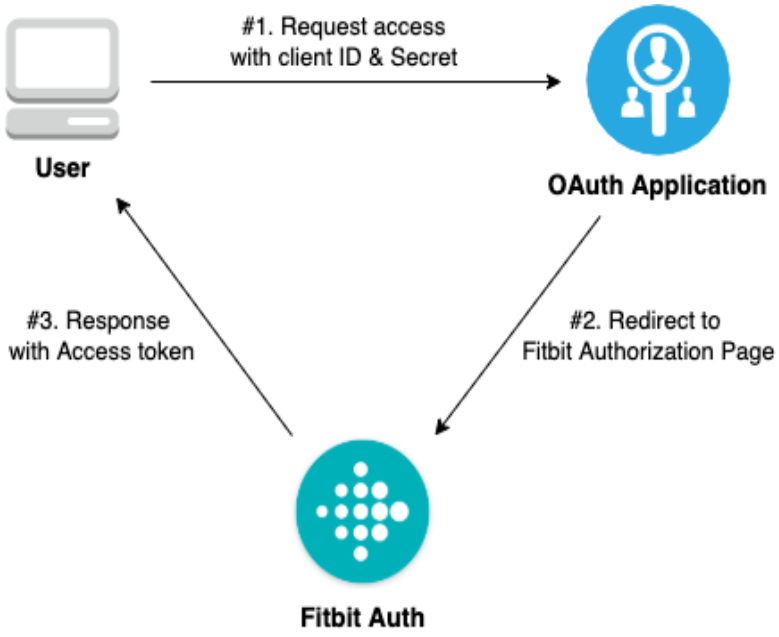


Figure 4.4: Fitbit's Authorization Code Grant Flow

4.2 Web Application

As the number of users grows, we considered building a web application to deal with the inflow of data from multiple devices. The web application enables monitoring of the data collection of each user and shows the device information on the dashboard, such as battery status and the last synced time to their mobile phone, to figure out how well they are wearing the Fitbits. It also provides the tools that handle collected data, including downloading interfaces that convert JSON data to CSV format and visualization page to analyze the heart rates data with stress inputs.

We considered building a serverless static web application using AWS components described in Table 4.1. The potential of static web hosting is that it minimizes the initial cost and eliminates the need for a hosting server like elastic compute cloud (EC2). Instead, the web application runs in the **Simple Storage Service** (S3) bucket, where HTML, CSS, and JS files are stored. The bucket has a feature to

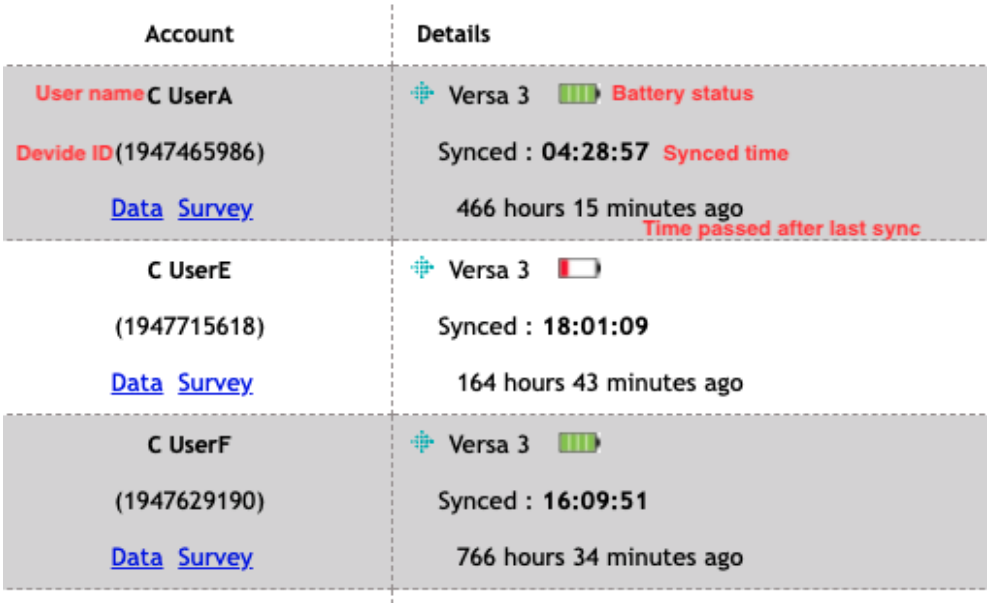


Figure 4.5: The Dashboard on the Web Application

host an application with a designated AWS domain. However, since Fitbit only allows the HTTPS protocol to communicate outside, we purchased the private domain and connected it to the S3 bucket using **Route53**, **CloudFront**, and **Certificate Manager**. Further details are described in Table 4.1

We employed **DynamoDB**, a non-relational key-value NoSQL database. First, the key-value data is beneficial in terms of data partitioning. In other words, it is easy to query by the partition key or sort key. For example, it is easy implementation to query user A's heart rates between specific dates and times. Next, it is a non-relational database that allows scaling both vertically and horizontally. It does not require strict schema that all the rows in a table have to have same attributes. It can be varies. Assuming the data is generated continuously from multiple wearable devices, a key feature for the database is scalability.

Most importantly, **Lambda functions** is AWS's core service that allows a static web application to act as a standard server. AWS Lambda is a serverless computing

service that executes code without establishing a server. It is event-driven that is executed only when the service is requested. It charges as per as invoked. Lambda helps the application build data processing functions by accessing resources at AWS' other services, like DynamoDB or S3. It can handle up to 250MB of code, and the execution time cannot be more than 15 minutes, which is sufficient for our needs. In the case of our application, we build lambda functions for creating/logging in user accounts and uploading/retrieving recorded data to the interface.

API Gateway builds APIs. Our application runs in RESTful API, which requests and responds in JSON format with four methods, including CREATE(post), READ(get), UPDATE(put), and DELETE(delete). AWS's API Gateway provides users in the back-end with the endpoint to access data in other services, such as DynamoDB. It also controls authentication to filter out unidentified requests. In our application, we built endpoints and mapped them to lambda functions so that end-users could access and upload data. An example is that the companion upload the stress level data via an API gateway endpoints to DynamoDB.

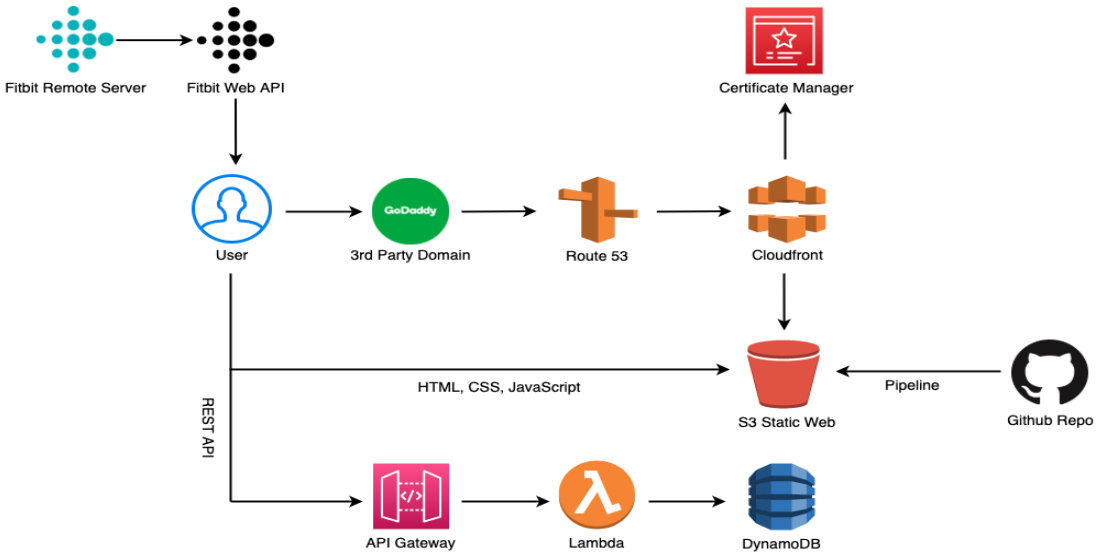


Figure 4.6: The Architecture of the Web Application

Services	Description
S3 Bucket	Stores the Web components (HTML, CSS, JS) and hosts the static web application.
Route 53	Routes the third party domain name service to the S3 buckets.
Certificate Manager	Provides SSL certificates for the static web to communicate in HTTPS protocol.
CloudFront	A contenct delivery network (CDN) that distributes static web contents by caching them on AWS's edge proxy servers.
DynamoDB	Stores data in key-value formats, including account credentials, stress levels input by user, survey data.
Lambda	Implements functions for acting like a server to handle requests and responds from the users.
API Gateway	Provides users with end points to access the resources.
Code Pipeline	Automatically updates the code as soon as the main branch of Github code is renewed.

Table 4.1: Role of Aws Services Used for Web Application

4.3 API Server for on-Cloud Stress Prediction

In the on-cloud-based stress prediction framework, Fitbit generates an array of the features of heart rates and sends it to the companion. Then the companion redirects the array of features to the server on the cloud for stress prediction. JavaScript applications can transfer data via API calls with the fetch function. We set a Node.js server running in an EC2 instance. The instance maintains trained models and prediction logic. When it receives the requests for stress prediction, it loads and predicts stress and responses with the result value.

Chapter 5

MACHINE LEARNING BASED STRESS MANAGEMENT

5.1 Problem Definition

This thesis aims to build real-time stress prediction frameworks using machine learning. Before defining the overall framework, verification of the dataset we collected has to be preceded to see if it is applicable to machine learning algorithms. This chapter shows the machine learning progress for stress prediction step by step according to the fundamentals of machine learning described in Chapter 2. Our problem is to detect the patterns of heart rates indicating stress. The heart rates are collected through Fitbit Versa 3, and the user provides the output data, the stress level. It is suitable for supervised learning algorithms, as the dataset contains the labels for all instances. Also, the labels are categorical variables, so I decided to employ classification algorithms to determine the input instances as 'stressed' or 'not stressed.'

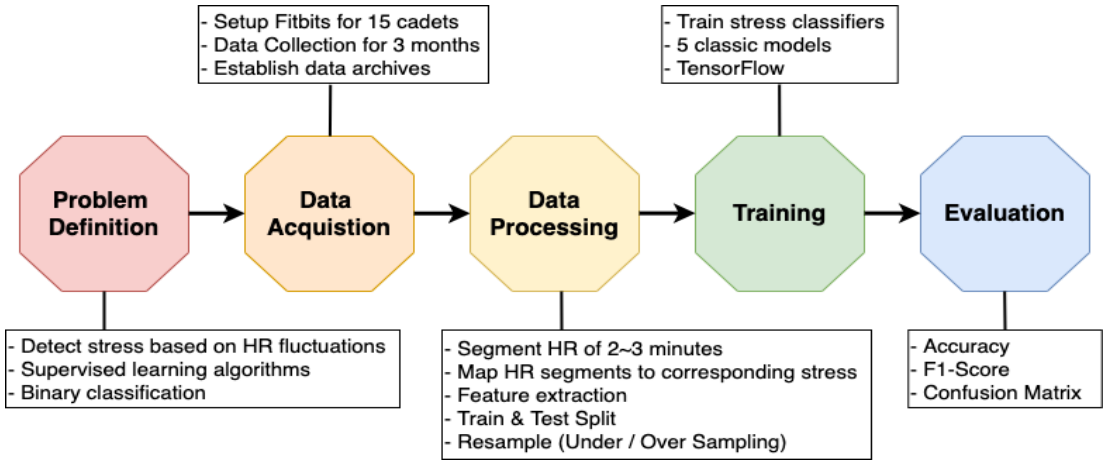


Figure 5.1: Machine Learning Processes for Stress Management

5.2 Data Acquisition

This project implements machine learning algorithms with the dataset collected from the police cadets—data collection has been processed from August to November 2021. Initially, there were 20 cadets, but five cadets quit the program in the middle and eventually established a dataset from 15 cadets at the end. Due to the anonymity of candidates, we excluded personal information from the dataset. Table 5.2 shows the general information of the cadets who provided biometric measurements during the police academic program. The majority were males and in their early 20s, while three females and two people were older than their 20s. Their program often required them to get involved in physically intense activities and paper exams. Since Fitbit has to send the data to the companion, we asked them to periodically sync their Fitbit to the companion for smooth data collection. The stress levels range from one to five, where one means as unstressed and five as extremely stressed. We also set stress level zero when the cadet does not respond to the prompt. Table 5.1 shows the distribution of stress levels. The most stress level entered is zero, which is excluded for data processing. Still, level one is overwhelmingly the majority than the stressed levels ranging from two to five. So I decided to run binary classification.

Stress level	Numerical value	Number of Responses
No Response	0	9578
Not Stressed	1	2935
A bit Stressed	2	355
Moderate	3	89
A lot	4	19
Extremely	5	5

Table 5.1: Distribution of Stress Responses

User	Age	Gender	Height(ft.)	Weight(lb.)
User A	20	M	5.6	170
User B	21	M	5.7	156
User C	22	M	5.8	183
User D	27	F	5.0	110
User F	26	M	5.9	180
User I	21	M	5.1	180
User J	24	M	6.3	240
User K	20	M	5.1	155
User L	42	F	5.4	145
User M	30	M	5.1	150
User N	29	M	6.4	220
User O	24	F	5.3	140
User R	24	M	6.0	220
User S	23	M	5.1	185
User T	22	M	5.8	180

Table 5.2: General Information of Police Cadets

There are two types of data collection pipelines for heart rates and stress responses. Regarding heart rates, we did not have to consider the repository to store heart rate data. Heart rates data is transmitted to the Fitbit's remote server by itself (once a Fitbit gets synced to the mobile phone). And we can retrieve the heart rates via Fitbit Web API with the credential information of the user account. On the other hand, stress levels need the private database since it is the custom type data that we programmed for this thesis. They are stored in key-value format. Lastly, both data can be downloaded from the download interface in the application we built using AWS S3. We then created a data archive that contains all cadets' heart rate and stress responses.

5.3 Data Processing

The main point of the data processing is the transformation of the dataset for them to be applied to the machine learning models that we want to run. As mentioned above, our work is ideal for classification models of supervised learning. The first thing was to extract the segment of heart rates as the black box shown in Figure 5.2. Since the prompts occur when heart rates go above the resting heart rate by 35 percent for 2 minutes, we considered cutting heart rates data of 2-3 minutes based on the time stress generated. Then each heart rates segment got mapped into corresponding stress levels to label each of them as stressed or unstressed.

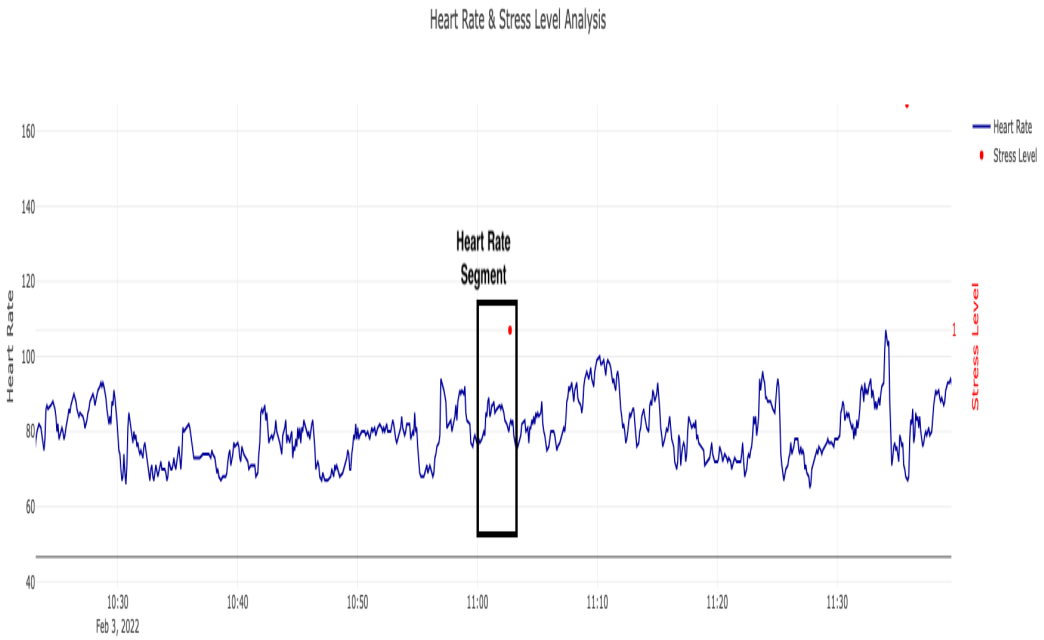


Figure 5.2: An Example of the Heart Rates Segmentation

We converted the stress level to '0' for unstressed and '1' for stress as it considers binary classification. As a result of segmentation of the heart rates data, mapping corresponding stress levels, and converting the labels for binary classes, Figure 5.3 shows the preprocessed raw dataset. The preprocessed raw dataset includes the target value, resting heart rates, and 40 samples of heart rates. The granularity of heart rates recorded by Fitbit is between 5 to 10 seconds, so the 40 samples of heart rates indicate about 2 to 3 minutes. Since the resting heart rates are calculated by Fitbit automatically, we do not have to go through the feature extraction step to get the values. It was attached before the feature extracting stage. One thing to notice is that there are two approaches to selecting 'stressed' data: 1) To include stress levels from 2 to 5, and 2) To include stress levels from 3 to 5. The reason for choosing the stress levels in two ways is that stress level 2 seems ambiguous—the classifiers do not learn when included. Therefore, we processed the datasets in two ways separately according to how to choose stress levels.

Label	RestHR	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
1	60.0	82	81	80	78	79	83	85	84	85	83	82	79	80	82	80	77	78	81	82	81	82	82	80	79	80	81	83	82	80	81	83	82	80	81	79	81	83	82	81	80
1	60.0	84	85	84	86	89	88	86	84	82	83	84	85	83	82	82	84	87	86	84	85	84	83	85	86	84	83	85	86	85	86	87	87	85	85	84	82	83	84	83	84
0	65.0	102	103	102	100	98	98	99	100	99	100	101	99	98	97	96	98	97	96	95	99	101	102	101	100	101	100	97	98	99	99	101	103	102	100	99	99	100	99	98	99
0	65.0	101	102	100	98	99	101	102	101	104	102	101	100	101	96	98	99	100	99	100	101	99	99	99	100	101	102	101	99	99	100	99	101	102	102	100	101	102	103	102	102
1	60.0	90	88	86	85	84	85	86	85	86	87	88	89	89	87	86	88	89	88	86	87	86	90	93	94	95	94	96	94	96	97	100	99	96	92	91	98	105	100	97	93
0	65.0	93	92	90	90	91	95	97	96	95	95	97	99	101	105	100	99	100	101	103	100	98	99	98	97	99	101	102	101	102	100	98	99	101	102	101	104	102	101	100	101
0	65.0	82	81	79	80	79	81	80	77	75	75	75	80	82	83	84	83	84	86	84	87	86	88	89	91	92	93	92	93	92	93	95	96	100	96	97	96	97	96	97	99
1	60.0	89	92	88	89	93	92	94	93	92	89	91	89	89	86	89	88	89	88	90	91	92	90	89	88	87	88	89	88	86	87	83	84	83	86	85	85	86	87	88	86
1	60.0	55	53	54	55	54	53	54	55	54	56	60	61	58	57	56	56	57	56	55	54	53	54	55	54	56	54	55	54	55	55	54	55	56	56	55	54	53	54	56	57
0	65.0	100	98	96	98	99	100	101	98	103	104	101	98	99	101	103	104	102	99	101	101	99	100	102	103	102	100	98	98	99	100	99	100	101	99	98	97	96	98	97	96
1	54.0	70	69	69	70	75	77	77	75	74	73	71	68	69	67	66	62	62	63	64	71	75	79	78	79	80	76	78	77	77	78	80	79	76	75	76	74	75	73	70	68
0	65.0	103	102	102	101	102	100	99	100	99	97	95	98	100	98	96	98	99	100	101	98	103	104	101	98	99	101	103	104	102	99	101	101	99	100	102	103	102	100	98	98
1	54.0	81	82	83	84	87	86	87	85	88	87	85	84	83	84	86	83	86	81	82	79	84	83	86	87	86	85	86	82	83	82	81	80	84	85	87	87	88	86	80	76
0	54.0	71	70	69	70	71	72	71	70	69	71	70	68	70	73	71	69	67	68	70	69	67	66	69	68	65	68	70	72	70	73	71	70	71	69	67	65	68	73	74	72
0	65.0	99	100	101	102	101	100	103	100	99	98	98	99	98	98	95	93	92	93	92	93	92	90	90	91	95	97	96	95	95	97	99	101	105	100	99	100	101	103	100	98
1	60.0	56	55	55	56	56	56	55	56	56	57	61	74	70	65	59	55	53	54	55	56	55	54	54	53	53	54	55	54	55	56	55	56	56	54	55	54	53	54	53	54
1	54.0	75	76	71	70	71	73	71	67	66	69	70	67	66	64	62	61	60	61	63	63	61	59	59	60	60	64	63	62	60	62	60	59	60	62	63	62	61	62	62	61
0	65.0	75	80	82	83	84	83	84	86	84	87	86	88	89	91	92	93	92	93	92	93	95	96	100	96	97	96	97	96	97	99	100	101	102	101	100	103	100	99	98	98

Figure 5.3: Preprocessed Dataset

The next step is feature extraction. The preprocessed dataset shown in Figure 5.3 is merely representing heart rates. Each instance is not distinguishable for stress status for now. Since I set the prompting algorithm concerning how heart rate fluctuates compared to the resting heart rate, I considered extracting the features representing distribution and fluctuation-wise aspects of heart rates. There are seven features extracted from the heart rates. The primary five are 1) mean, 2) standard deviation, 3) minimum and 4) maximum value of the heart rates, and 5) resting heart rate. The additional two features are 6) the difference between mean heart rates from the resting heart rate by percentage (DiffRest) [32], and 7) the root mean squared of successive difference between normal heartbeats (RMSSD), which Fitbit uses for calculating heart rate variability from heartbeats. Figure 5.4 shows the feature matrix containing all the features and the corresponding target value.

Label	RestHR	Mean	Std	Max	Min	DiffResting	RMSSD
1	60.0	91.125	5.182603	105	84	50.609756	90.637983
0	54.0	85.025	3.357734	91	77	56.052394	84.469218
1	60.0	85.300	2.002498	90	82	41.138211	84.795881
0	65.0	99.700	2.111871	104	96	52.082552	99.020446
0	65.0	98.625	3.631029	105	90	50.469043	98.007964
0	65.0	100.450	1.548386	104	96	53.208255	99.747119
1	60.0	58.600	1.113553	62	56	-2.276423	58.644859
0	65.0	99.900	1.700000	103	95	52.382739	99.209190
1	60.0	55.550	1.182159	58	53	-7.235772	55.675015
1	60.0	55.500	1.244990	58	53	-7.317073	55.627682
0	65.0	100.025	1.916866	104	95	52.570356	99.335843
0	65.0	87.375	7.398268	100	75	33.583490	87.204553

Figure 5.4: The Feature Matrix

One issue with our dataset is imbalanced. The dataset contains unstressed instances over 85% of the entire set. If we do not resolve the imbalanced issue before getting into the training stage, the model will learn about only unstressed data. It might be giving us good accuracy, but it will not learn about the stressed instance, so the F1 score will be insignificant. There are two approaches for balancing the dataset: 1) To undersample the majority class by randomly choosing as many instances as the number of the minority, and 2) To oversample the minority instances using Synthetic Minority Over-Sampling Technique (SMOTE) [39]. The former method actually reduced the size of the dataset, so it might be ineffective for training sessions; however, it can learn both stressed and unstressed equivalently. The latter increases the dataset size and will increase the effectiveness of the learning stage, but the sampled instances are similar but not real.

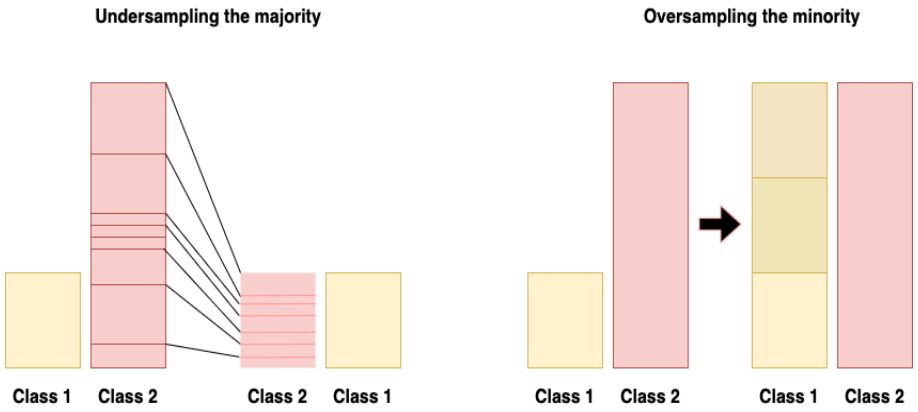


Figure 5.5: Resampling Methods

Lastly, all the attributes in the dataset should have the same impact on the machine learning models, meaning that features in different scales will generate bias on a particular type of variable so that some features will not contribute to the model as we would expect. We implemented the standard scaling on all the dataset columns to make them have the same distribution with 0 means and the unit standard deviation.

5.4 Training Models

There are four methods for training models as specified in Table 5.3. We trained models separately by how we defined the stress levels (level 2 to 5 or 3 to 5) and how we balanced the dataset (undersampling or oversampling). Regarding classification algorithms, we considered four classic models and TensorFlow's feed-forward network. The set of classic models includes Decision Tree, Random Forest, Adaboost, XGBoost. The first three models are imported from Scikit-learn(Sklearn) [45], which provides not only reliable classification models but also useful built-in functions for the machine learning process, such as *train_test_split* or *standard scalaer*. We imported the XGBoost algorithm from [17]. The reason for choosing TensorFlow for the neural network is that it provides a framework for converting to TensorFlow.js, used in Fitbit architecture. Before running the dataset for training, I split the dataset into a training set (80%) and a testing set (20%) to evaluate models' capability to handle unseen data.

Notation	Stress levels	Resampling method
Approach 1	2 to 5	Under-sampling
Approach 2	2 to 5	SMOTE
Approach 3	3 to 5	Under-sampling
Approach 4	3 to 5	SMOTE

Table 5.3: Four Approaches for Training Models

Regrading hyperparameters of the TensorFlow feed-forward network, it has six hidden layers activated by the ReLu function. The output layer is set with a Sigmoid function returning a value ranging from 0 to 1. If the final return value from an input instance is lower or equal to 0.5, it is unstressed, which is labeled as 0. Otherwise, it

is stressed when it is higher than 0.5. Since it is the binary classification, we considered the binary cross-entropy for the loss function. Other pieces of information are described in Table 5.4.

Types	Value
Number of Hidden Layers	6
Learning Rate	0.0005
Loss Function	Binary Cross-Entropy
Epochs	200

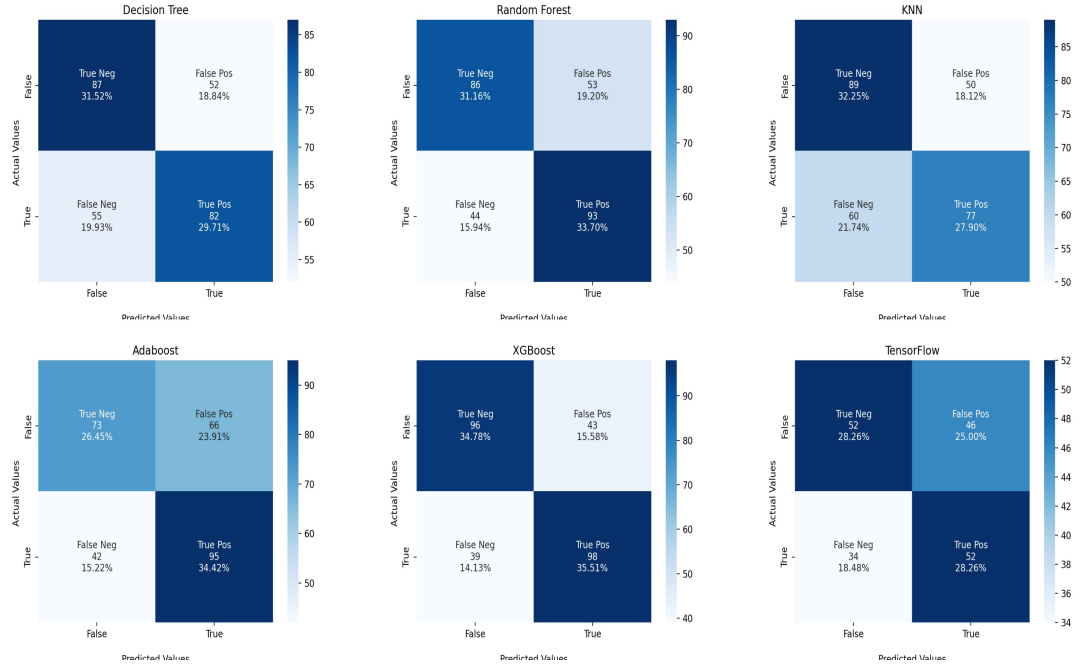
Table 5.4: Hyperparameters of TensorFlow's Feed-Forward Network

5.5 Evaluation

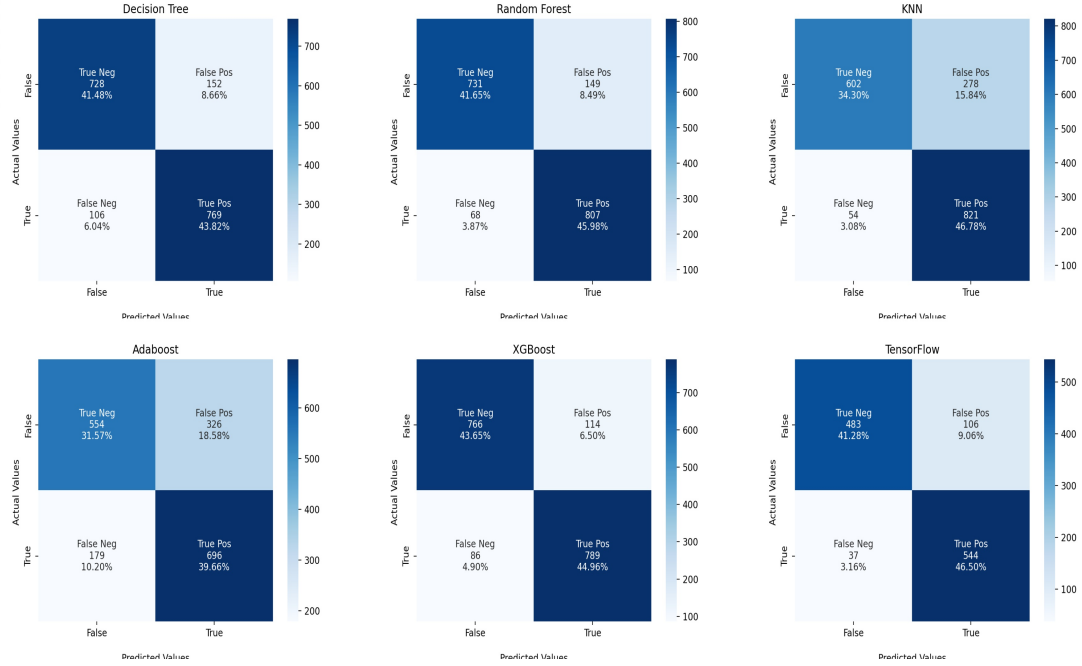
Table 5.5 shows the accuracy and F1 scores for each approach. Approach 1 gave us the worst result. It could not distinguish between unstressed and stressed by heart rates features. It might be because of either insufficient number of instances or ambiguity of features from stress level 2. We could gain better results from approach 2, which oversampled the minority class. Approach 3 showed more reliable results than Approach 1 by not using oversampling with synthetic sampling. The dataset size was reduced even more than approach 1 as it excluded stress level 2 and resampled the unstressed data as many as the number of stress levels ranging from 3 to 5. Although it does not have a sufficient dataset, it showed pretty good accuracy and F1 scores by properly inferencing the testing set. Lastly, approach 4 gave us the best results. It excluded stress level 2 and oversampled the minority class. The best accuracy it reached is 96% from XGBoost and 95% from TensorFlow.

Approach	Model	Accuracy	F1 Score
Approach 1	Decision Tree	61.23%	60.51%
	KNN	60.14%	58.33%
	Random Forest	64.85%	65.72%
	AdaBoost	60.86%	58.33%
	XGBoost	70.28%	70.50%
	TensorFlow	56.52%	56.52%
Approach 2	Decision Tree	85.29%	85.63%
	KNN	81.08%	83.18%
	Random Forest	87.63%	88.14%
	AdaBoost	71.22%	73.37%
	XGBoost	88.60%	88.75%
	TensorFlow	87.77%	88.38%
Approach 3	Decision Tree	77.27%	76.92%
	KNN	77.27%	72.72%
	Random Forest	81.72%	81.83%
	AdaBoost	72.12%	71.64%
	XGBoost	78.78%	78.12%
	TensorFlow	86.36%	88.46%
Approach 4	Decision Tree	91.79%	91.73%
	KNN	88.34%	88.73%
	Random Forest	94.41%	94.34%
	AdaBoost	79.25%	80.15%
	XGBoost	96.98%	96.95%
	TensorFlow	95.98%	96.02%

Table 5.5: Evaluations by Accuracy and F1-score

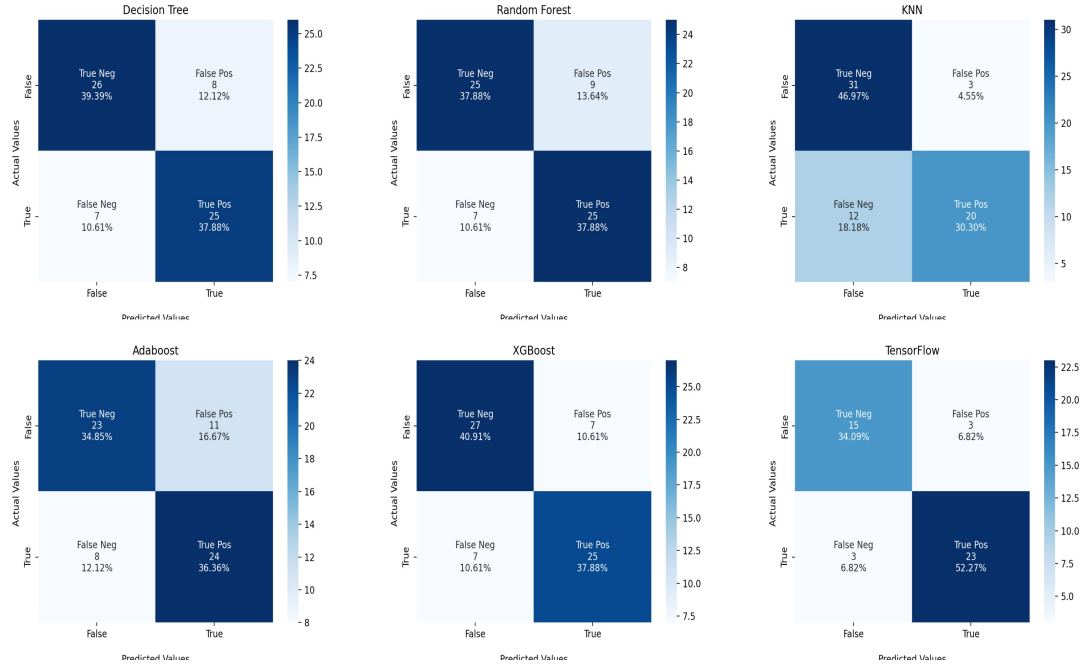


(a) Approach 1

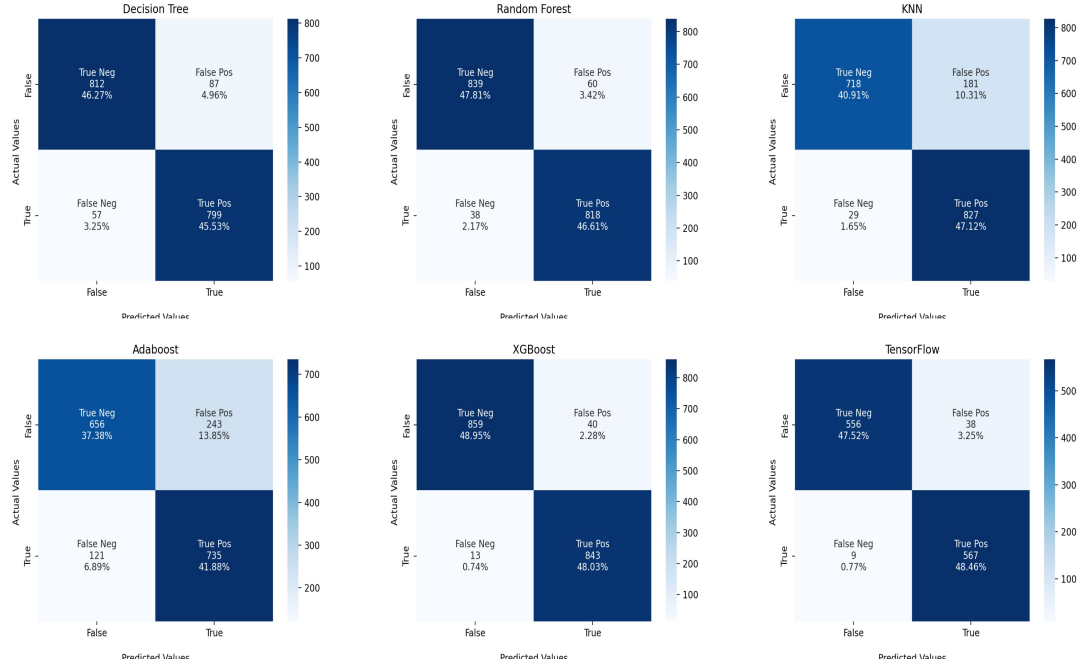


(b) Approach 2

Figure 5.6: Confusion Matrices of Approach 1 and 2



(a) Approach 3



(b) Approach 4

Figure 5.7: Confusion Matrices of Approach 3 and 4

Chapter 6

EDGE COMPUTING FOR STRESS PREDICTION

This chapter deploys stress prediction models on edge devices and explores the effectiveness of model optimization techniques. The performance of edge-based machine learning is evaluated by its latency of model loading, stress prediction, and data transfer between each node. To analyze the edge's capability by comparison with the other method, We established a stress prediction framework on the Cloud. I set API services in the Node.js server running on the AWS EC2 instance in the northern California datacenter.

Fitbit always maintains the latest 2 to 3 minutes of heart rates using the queue data structure. When it needs to request stress prediction, it generates an input instance by calculating the seven features specified in Chapter 5. It sends a request with a feature vector to either the companion or Cloud after extracting features. Companion has to load the pre-trained models from AWS S3 buckets using HTTP protocols, while the API server has the models in its storage.

Since Fitbit and the companion run in JavaScript, we had to convert the models to JavaScript versions from Python. TensorFlow library provided with conversion framework so that it was straightforward to transform the model for Fitbit companion. Among the four approaches experimented in Chapter 5, we chose Approach 4's model, which achieved the highest accuracy and F1 scores. The accuracy is estimated by the model types, such as original and compressed, and the latency measuring includes the end-to-end and milliseconds in predictions.

6.1 Cloud- vs. Edge-based Stress Prediction

To implement on-cloud stress prediction, I built an API server that handles requests from Fitbit, measures stress or not stress, and replies with an answer. Figure 6.1 shows the overall flows of on-cloud stress prediction. Fitbit communicates with the server by Fetch API, which allows us to send requests to the server and receive the responses. The fetch method takes two parameters, the URL and the options object, like *GET* or *POST*. Regarding URL, Fitbit only accepts HTTPS protocols for communication outside its network, so I had to set domain name service and SSL certificates for the API server. In the case of on-cloud prediction, the companion is merely a bridge redirecting requests and responses. When requesting stress prediction, Fitbit uses *POST* method and attaches the input instance, which contains heart rates features. Upon receiving requests, the server proceeds with the stress prediction method with pre-trained models stored in advance. Since the cloud has to focus only on stress prediction for its resources and has superior processing units to companion, the server-side models have not gone through model optimization processes.

Type	Specification	Value
Ubuntu Server 20.04 LTS (Server)	CPU	1 vCPU 2.40GHz
	RAM	1 Gib
	Storage	20 GiB
Samsung Galaxy S9 (Companion)	CPU	Octa-core, 2800 MHz
	RAM	4 Gib
	Storage	64 GiB

Table 6.1: Hardware Specification of the Server and Companion

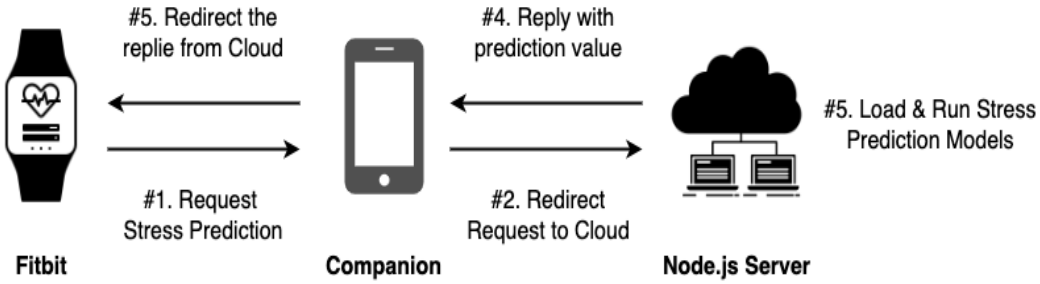


Figure 6.1: On-cloud Stress Prediction

On-companion stress prediction is equivalent to implementing prediction methods on the mobile phone. Unlike the typical Android application, Fitbit's companion cannot load files from the device's storage. In order to access the trained models, the companion needs to run the Fetch method to retrieve them via HTTPS protocol. Once the models are declared with variables at the beginning, they are allocated to the memory location so that the subsequent model loadings take almost 0 milliseconds. For the companion-based stress prediction, I run the original size of the model and reduced-size of models processed by model optimization techniques. And I will verify the possibility for model optimizations to reduce the latency with maintaining good accuracy.

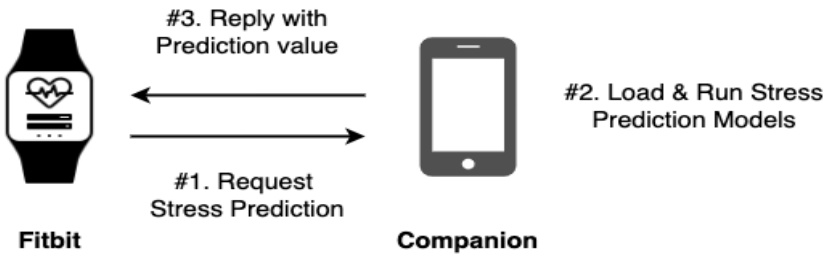


Figure 6.2: On-companion Stress Prediction

6.2 TensorFlow on Edge

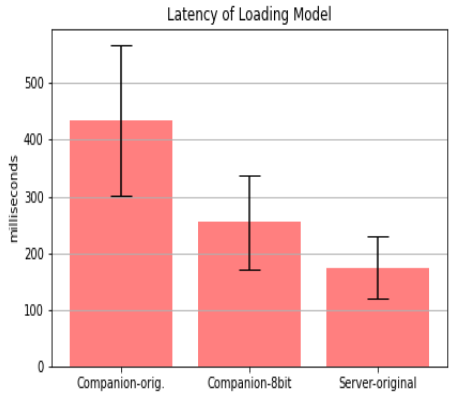
TensorFlow (TF) is an open-source library developed by Google Brain teams that programs diverse data flows for machine learning. It offers various platforms depending on the nature of the project, including TensorFlow.js (TFJS) and TensorFlow lite. And it also provides easy ways to transform the model to the version of the other platforms. To run the model on Fitbit's companion, we converted the stress prediction model into TFJS. The main difference between TF and TFJS is that TFJS is for running operations in the browser, so how to load the models is different. It can be loaded from local storage, indexedDB, or HTTP endpoints. However, the first two do not apply to the companion since it is not based on the browser, meaning that the only way to load TFJS models is via HTTP endpoints. We used AWS S3 buckets to store the model topology and the binary files of the weight information. One thing to notice is that the topology and the weight file must be in the same directory since the topology refers to the path of the weights.

We chose model compression techniques to optimize the neural network model because TFJS supports several model compression tools, including pruning, quantization, and weight clustering. In this work, We applied quantization to reduce the size of models to 8-bits from the original model. For the TFJS model, there are two approaches to quantize, post quantization and quantization-aware training. In this work, we used the quantization-aware training before converting to the TFJS model. We could reduce the model size by 4x when we quantized it into 8-bit from 32-float, preserving accuracy. Table 6.2 shows the accuracy and the size of models. It shows that the model's size decreases more as quantized to the lower bits.

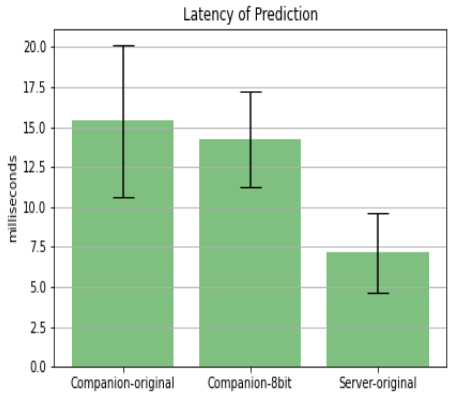
Model	Type	Value
Original Model	Accuracy	85.98%
	Topology Size	5.0 KB
	Weight Size	3.0 MB
Quantized 8-bit	Accuracy	84.10%
	Topology Size	6.7 KB
	Weight Size	774.0 KB

Table 6.2: Accuracy and the Size of Models

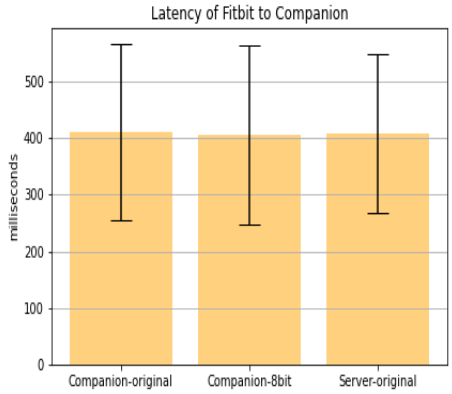
For the latency, we considered measuring: (a) Model loading, (b) Prediction, (c) Data transfer to Fitbit to the companion, (d) Data transfer from companion to Fitbit, (e) End-to-end latency and (f) Data transfer on the Internet. We measured (a) independently from the rest of latency measurements. First, the latency for loading models is insignificant because it is allocated to the memory after the first call. The times for the following loads are like 0 milliseconds. However, whenever the application turns off and on, it has to load the model again. Also, Fitbit restarts its application randomly, so it is worth measuring loading latency. The most effective part of reducing latency through quantization is model loading. If quantization is done for 8 bits, the latency can be decreased by over 2x. For 2) to 4), they were measured together to analyze which are most affecting the latency. The figure 6.3 represents the latencies of each part by the type of the models. The prediction time is not significantly decreased by quantization since the impact of an input instance is subtle. Also, the Bluetooth connection took over twice as much as the Internet connection (Companion to Server), meaning the time for reaching server from the companion was not considerable. The end-to-end latencies were similar to each other.



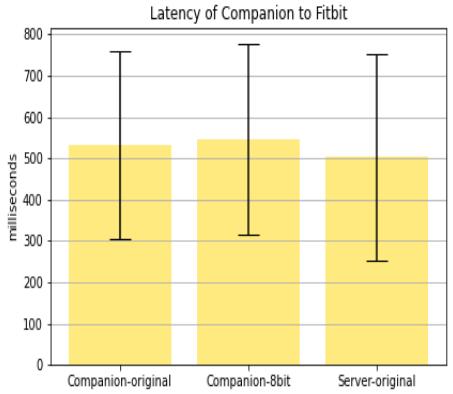
(a) Loading Model (independent)



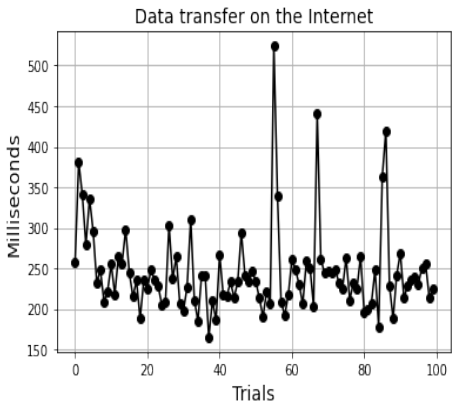
(b) Prediction



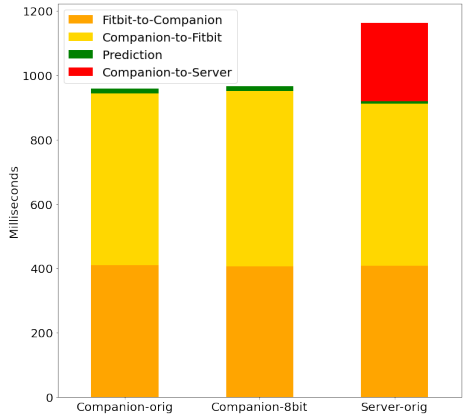
(c) Data Transfer (Fitbit to Companion)



(d) Data Transfer (Companion to Fitbit)



(e) Data Transfer (Companion to Server)



(f) Stacked Measurements

Figure 6.3: Latency Analysis

Chapter 7

CONCLUSION

7.1 Conclusion

In this thesis, we researched the feasibility of edge machine learning-based stress prediction using wearable devices. The summarization of what we learned is as follows:

- First, we verified that the heart rates recorded by commercial wearable devices could be used for stress prediction. Unlike other equipment which records ECG or GSR data in milliseconds, Fitbit reads heartbeats in 5 to 10-minute granularity. However, with features extracted from the heartbeats and stress levels provided by participants in advance, we confirmed that the heartbeat data is also a validated data type indicating stressful or non-stressful conditions.

- Next, we employed five classic classification models and one neural network for binary classification. After segmenting heart rates data into 2 to 3-minute windows and extracting five statistical features and two features representing heart rate variability, we could apply the dataset to machine learning algorithms. We also resolved the imbalance of the dataset by either under-sampling or over-sampling. The best accuracies were 95.34% and 94.94% by XGBoost and TensorFlow from approach 4, which over-sampled and used stress levels 3 to 5.

- Third, since Fitbit and the companion are run on JavaScript, we needed to convert models in Python to JavaScript. TensorFlow offered a conversion framework so that we could transform the neural network model without losing the accuracy

significantly. Also, the Fitbit SDK allowed importing the TensorFlow module on the companion side, so it was doable to implement stress prediction on the mobile device.

- Lastly, our neural network is a feed-forward network for binary classification, and we could only consider quantization techniques. The quantization reduced the model size by 2x and maintained the accuracy as before the quantization. However, it could not reduce the inference time dramatically. The latency for data transfer between a node to node was the same regardless of the model size.

7.2 Future Research

We have discussed about real-time inference through model optimization on edge devices. However, the stages that require the most effort in machine learning processes are data processing and training. To train models in a cloud-intensive structure is still vulnerable in energy efficiency and privacy. We can consider both training and inference on edge devices, but they have limited resources to burden such operations. While preserving the advantage of edge computing, such as reducing bandwidth, faster inference, and protecting privacy, the problem above can be overcome by federated learning. Federated learning is the framework for building a global model in the central cloud by training across multiple small nodes [22; 46]. The edge device still does not have to transfer data generated locally and train with its own data by updating the weights of the model independently. It can also make use of locally trained models directly for quick inference [47]. The central cloud aggregates the weights updates from the edge nodes and establishes a mo4re intelligent model.

For our thesis, we can consider extracting features from recorded heart rates and maintaining a small and personal dataset for model training on the companion. In the initial stage, the companion can utilize the model generated by the central cloud, improving the model with accumulated local data. This case will reflect the personal trend of accepting stress levels more subjectively. In addition, it can contribute to creating a smarter model by reporting the weight of the model learned at regular intervals.

REFERENCES

[1] IDTechEx, "Wearable technology 2016-2026 markets, players and 10-year forecasts." `https://www.idtechex.com/en/research-report/wearable-technology-2016-2026/483`, July 2016. Accessed: 2022-04-13.

[2] S. B. Kotsiantis, I. Zaharakis, P. Pintelas, *et al.*, "Supervised machine learning: A review of classification techniques," *Emerging artificial intelligence applications in computer engineering*, vol. 160, no. 1, pp. 3–24, 2007.

[3] B. Marsh, "Multivariate analysis of the vector boson fusion higgs boson," *University of Missouri*, vol. 8, 2016.

[4] N. Hatami, Y. Gavet, and J. Debayle, "Classification of time-series images using deep convolutional neural networks," in *Tenth international conference on machine vision (ICMV 2017)*, vol. 10696, p. 106960Y, International Society for Optics and Photonics, 2018.

[5] R. F. AbuAlRub, "Job stress, job performance, and social support among hospital nurses," *Journal of nursing scholarship*, vol. 36, no. 1, pp. 73–78, 2004.

[6] K. Gok, N. Atsan, *et al.*, "Decision-making under stress and its implications for managerial decision-making: a review of literature," *International Journal of Business and Social Research*, vol. 6, no. 3, pp. 38–47, 2016.

[7] L. Giessing, R. R. Oudejans, V. Hutter, H. Plessner, J. Strahler, and M. O. Frenkel, "Acute and chronic stress in daily police service: A three-week n-of-1 study," *Psychoneuroendocrinology*, vol. 122, p. 104865, 2020.

[8] J. M. Reingle Gonzalez, K. K. Jetelina, S. A. Bishopp, M. D. Livingston, R. A. Perez, and K. P. Gabriel, "The feasibility of using real-time, objective measurements of physiological stress among law enforcement officers in dallas, texas," *Policing: Int'l J. Police Strat. & Mgmt.*, vol. 42, p. 701, 2019.

[9] A. Ometov, V. Shubina, L. Klus, J. Skibińska, S. Saafi, P. Pascacio, L. Flueratoru, D. Q. Gaibor, N. Chukhno, O. Chukhno, A. Ali, A. Channa, E. Svertoka, W. B. Qaim, R. Casanova-Marqués, S. Holcer, J. Torres-Sospedra, S. Casteleyn, G. Ruggeri, G. Araniti, R. Burget, J. Hosek, and E. S. Lohan, "A survey on wearable technology: History, state-of-the-art and current challenges," *Computer Networks*, vol. 193, p. 108074, 2021.

[10] A. Phaneuf, "Latest trends in medical monitoring devices and wearable health technology." `www.businessinsider.com/wearable-technology-healthcare-medical-devices`, Jan 2021. Accessed: 2022-04-13.

[11] R. Rawassizadeh, B. A. Price, and M. Petre, "Wearables: Has the age of smartwatches finally arrived?," *Communications of the ACM*, vol. 58, no. 1, pp. 45–47, 2014.

[12] B. Hewgill, *Open Source Quantitative Stress Prediction Leveraging Wearable Sensing and Machine Learning Methods*. The University of Vermont and State Agricultural College, 2020.

[13] J. Taelman, S. Vandeput, A. Spaepen, and S. Huffel, *Influence of Mental Stress on Heart Rate and Heart Rate Variability*, vol. 22, pp. 1366–1369. 01 2009.

[14] M. Ivanović and M. Radovanović, "Modern machine learning techniques and their applications," in *International Conference on Electronic, Communication, and Network*, 2014.

[15] I. H. Witten and E. Frank, "Data mining: practical machine learning tools and techniques with java implementations," *Acm Sigmod Record*, vol. 31, no. 1, pp. 76–77, 2002.

[16] V. Hodge and J. Austin, "A survey of outlier detection methodologies," *Artificial intelligence review*, vol. 22, no. 2, pp. 85–126, 2004.

[17] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, (New York, NY, USA), pp. 785–794, ACM, 2016.

[18] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pp. 265–283, 2016.

[19] A. Jain and D. S. Jat, "An edge computing paradigm for time-sensitive applications," in *2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4)*, pp. 798–803, 2020.

[20] M. G. S. Murshed, C. Murphy, D. Hou, N. Khan, G. Ananthanarayanan, and F. Hussain, "Machine learning at the network edge: A survey," *CoRR*, vol. abs/1908.00080, 2019.

[21] S. Patel, R. S. McGinnis, I. Silva, S. DiCristofaro, N. Mahadevan, E. Jortberg, J. Franco, A. Martin, J. Lust, M. Raj, B. McGrane, P. DePetrillo, A. J. Aranyosi, M. Ceruolo, J. Pindado, and R. Ghaffari, "A wearable computing platform for developing cloud-based machine learning models for health monitoring applications," in *2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pp. 5997–6001, 2016.

[22] Y. Chen, S. Biookaghazadeh, and M. Zhao, "Exploring the capabilities of mobile devices supporting deep learning," in *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '18, (New York, NY, USA), p. 17–18, Association for Computing Machinery, 2018.

[23] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proceedings of the IEEE*, vol. PP, pp. 1–20, 07 2019.

[24] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.

[25] M. Merenda, C. Porcaro, and D. Iero, "Edge machine learning for ai-enabled iot devices: A review," *Sensors*, vol. 20, no. 9, p. 2533, 2020.

[26] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and ¡0.5mb model size," 2016.

[27] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[28] K. Zhao, A. Jain, and M. Zhao, "Iterative activation-based structured pruning," *arXiv preprint arXiv:2201.09881*, 2022.

[29] Google, "Edge tpu." `https://cloud.google.com/edge-tpu/`. Accessed: 2022-04-13.

[30] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, pp. 1–12, 2017.

[31] P. Buddi, V. Prasad, and K. Sunitha, "Machine learning approach for stress detection using wireless physical activity tracker," *International Journal of Machine Learning and Computing*, vol. 8, pp. 33–38, 02 2018.

[32] W. Lawanont, P. Mongkolnam, C. Nukoolkit, and M. Inoue, *Daily Stress Recognition System Using Activity Tracker and Smartphone Based on Physical Activity and Heart Rate Data*, pp. 11–21. 01 2019.

[33] W. Sanchez, A. Martínez-Rebollar, Y. Hernandez, H. Estrada Esquivel, and M. Gonzalez-Mendoza, "A predictive model for stress recognition in desk jobs," *Journal of Ambient Intelligence and Humanized Computing*, 12 2018.

[34] F.-T. Sun, C. Kuo, H.-T. Cheng, S. Buthpitiya, P. Collins, and M. Griss, "Activity-aware mental stress detection using physiological sensors," in *Mobile Computing, Applications, and Services* (M. Gris and G. Yang, eds.), (Berlin, Heidelberg), pp. 282–301, Springer Berlin Heidelberg, 2012.

[35] J. Lee, H. Lee, and M. Shin, "Driving stress detection using multimodal convolutional neural networks with nonlinear representation of short-term physiological signals," *Sensors*, vol. 21, no. 7, 2021.

[36] J. A. Healey and R. W. Picard, "Detecting stress during real-world driving tasks using physiological sensors," *IEEE Transactions on intelligent transportation systems*, vol. 6, no. 2, pp. 156–166, 2005.

[37] P. Garg, J. Santhosh, A. Dengel, and S. Ishimaru, "Stress detection by machine learning and wearable sensors," in *26th International Conference on Intelligent User Interfaces-Companion*, pp. 43–45, 2021.

[38] P. Bobade and M. Vani, "Stress detection with machine learning and deep learning using multimodal physiological data," in *2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA)*, pp. 51–57, IEEE, 2020.

[39] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[40] P. Schmidt, A. Reiss, R. Duerichen, C. Marberger, and K. Van Laerhoven, "Introducing wesad, a multimodal dataset for wearable stress and affect detection," in *Proceedings of the 20th ACM international conference on multimodal interaction*, pp. 400–408, 2018.

[41] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[42] T. R. Mauldin, M. E. Canby, V. Metsis, A. H. H. Ngu, and C. C. Rivera, "Smartfall: A smartwatch-based fall detection system using deep learning," *Sensors*, vol. 18, no. 10, 2018.

[43] S. S. Ogden and T. Guo, "MODI: Mobile deep inference made efficient by edge computing," in *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*, (Boston, MA), USENIX Association, July 2018.

[44] T. Guo, "Cloud-based or on-device: An empirical study of mobile deep inference," 2018.

[45] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[46] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[47] Q. Xia, W. Ye, Z. Tao, J. Wu, and Q. Li, "A survey of federated learning for edge computing: Research problems and solutions," *High-Confidence Computing*, vol. 1, no. 1, p. 100008, 2021.