NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATION

MSc THESIS

Tiny ML in Microcontroller to Classify EEG Signal into Three States

Thuy T. Pham

Supervisor (or supervisors): **Van-Tam Nguyen,**
Professor at Telecom ParisTech, France

ATHENS

SEPTEMBER 2022

MSc THESIS

Tiny ML in Microcontroller to Classify EEG Signal into Three States

Thuy T. Pham

S.N.: 7115192100007

Supervisor (or supervisors): **Van-Tam Nguyen,**
Professor at Telecom ParisTech, France

ABSTRACT

This thesis investigates how to implement an own-built neural network for electroencephalography signals classification on an STM32L475VG microcontroller unit. The original dataset is analyzed and processed to better understand the brain signals. There is a comparison between three machine learning algorithms (linear support vector machine, extreme gradient boosting, and deep neural network) in three testing paradigms: *specific-subject*, *all-subject*, and *adaptable* to select the most appropriate approach for deploying on the microcontroller. The implementation procedure with detailed notation is presented, and the inference is also performed to feasible observation. Finally, possible improvement solutions are proposed within a clear demonstration.

SUBJECT AREA: Signal processing, machine learning, embedded system

KEYWORDS: electroencephalography, artificial neural network, STM32 microcontroller, SVM, XGBoost

Co-funded by the
Erasmus+ Programme
of the European Union

Erasmus Mundus Joint Master's Degree
"SMART Telecom and Sensing NETworks" (SMARTNET) (**2019/2021 intake**)
Aston University, Triangle, B4 7ET / Birmingham, UK
Email: aipt_smartnet@aston.ac.uk / Web-site: smartnet.astonphotonics.uk/

Acknowledgement

This Master Thesis has been accomplished in the framework of the European Funded Project: **SMART Telecom and Sensing Networks (SMARTNET)** - Erasmus+ Programme Key Action 1: Erasmus Mundus Joint Master Degrees – Ref. Number 2017 – 2734/001 – 001, Project number - 586686-EPP-1-2017-1-UK-EPPKA1-JMD-MOB, coordinated by **Aston University,** and with the participation of **Télécom SudParis, member of IP Paris** and **National and Kapodistrian University of Athens.**

Aston University
BIRMINGHAM UK

TELECOM SudParis TELECOM ParisTech

National and Kapodistrian
UNIVERSITY OF ATHENS

ACKNOWLEDGEMENT

Khi ngồi viết những dòng này, em biết rằng hành trình hai năm của mình với SMARTNET sắp đi tới hồi kết. Thời gian trôi qua chẳng chờ đợi ai, nhưng kỷ niệm là thứ sẽ còn tồn tại mãi mãi!

Cảm ơn SMARTNET đã tin tưởng và cho em cơ hội để có thể thực hiện hóa ước mơ của mình. Em không nghĩ rằng mình lại có thể "sống sót" qua những thử thách khắc nghiệt trong suốt khóa học này: những khó khăn lúc làm visa, lo chỗ ăn ở; gần một năm trời chỉ quanh quẩn với căn phòng vì dịch Covid, những môn học mà em chưa bao giờ được tiếp xúc, những đề thi dài 10, 11 trang giấy…^_^ Nhưng có lẽ, chính những thử thách ấy đã rèn luyện em trưởng thành và tự tin hơn trong cuộc sống. SMARTNET không chỉ là một chương trình học đơn thuần mà với em đó là một gia đình, nơi có các bạn từ khắp nơi trên thế giới cùng nhau chia sẻ những khoảnh khắc, có những thầy cô thân thiện cởi mở và nhiệt tình với sinh viên. Cảm ơn tất cả những gì thuộc về SMARTNET!

Em cũng muốn gửi lời cảm ơn chân thành nhất tới thầy Văn-Tâm Nguyễn, người đã hướng dẫn cũng như định hướng cho em trong suốt quá trình thực hiện thực tập. Không chỉ là kiến thức học thuật mà những kiến thức đời sống, thầy đều chia sẻ nhiệt tình với em, giúp em không cảm thấy tự ti trong nghiên cứu. Bên cạnh đó, em cũng xin cảm ơn sự hỗ trợ tận tình từ các thầy cô trong bộ môn COMELEC, Telecom Paris; đặc biệt là thầy Germain Pham – người luôn luôn truyền những năng lượng tích cực, dạy em những kiến thức rất hữu ích trong đề tài này.

Những lời tri ân này em cũng xin gửi tới tất cả các thầy cô, những điều phối chương trình trong SMARTNET, đặc biệt là tại Telecom SudParis và NKUA. Chặng đường này của em sẽ không thể nào hoàn thiện được nếu không có sự giảng dạy và hỗ trợ từ mọi người.

Cảm ơn những người bạn luôn sát cánh, động viên và "lôi kéo" mình đi ăn trong những lúc stress nặng nề. Cảm ơn Juhyun Kim và Rilwanu Kasno – được làm bạn với hai người là một trong những thành công lớn của mình trong chương trình thạc sỹ này :D.

Cuối cùng, con xin cảm ơn gia đình ở Việt Nam, cảm ơn bố mẹ và em trai đã luôn ủng hộ sự lựa chọn của con, luôn quan tâm, lo lắng và giúp con có một tinh thần mạnh mẽ trong lần đầu tiên xa nhà. Con sắp hoàn thành chặng đường này rồi để về thăm gia đình mình rồi ạ!

Thank you so much! Merci beaucoup! Ευχαριστώ από τα βάθη της καρδιάς μου! SMARTNET.

PHẠM TRỌNG Thủy

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Using electroencephalography (EEG) signals in neuroscience and brain disease diagnosis was started in the first half of the twentieth century. Even today, the principles of operation have unchanged, but the field of study for electroencephalography signals has now considerably broadened along with the development of science and technology. Many patterns were found between electroencephalography signals and studies of motor activity, mental state, and brain activity. However, the valuable information extracted from the electroencephalography is still limited. Science in this area is still in the early stages of development.

The evolution of machine learning has made significant strides in the previous ten years, influencing several industries, including signal processing for EEG. Among them, neural network (NN) is a flourishing tool for working with EEG signals. Thousands of publications about NN applications for EEG signals were published in various areas, such as diagnosing diseases, lie recognition, researching the physiology process, image classification, control artifacts, etc. However, the research on implementing NN for EEG application on embedded systems is limited. This thesis will deal with designing NN for EEG classification and implementing NN on an embedded system as the STM32 microcontroller.

1.1 Related work

For the comfort of understanding, the related work will be divided into two domains: the first is works about neural networks in EEG signal recognition/classification, and the second is works about machine learning on microcontrollers.

In [1], the authors used machine learning to automatically detect alertness/drowsiness from the combination of EEG and electrooculography signals. An efficient extremely learning machine (ELM) was employed for state classification. The proposed algorithm performed a high accuracy and also computed in a fast speed. The best state-detection accuracy when using ELM within radial basis function is 97.3%.

Xiaojun Bi et al. [2] applied deep learning for EEG spectral images to detect the early Alzheimer's disease. EEG data was collected from 12 Alzheimer and Mild Cognitive Impairment patients to build a whole dataset with 12000 EEG spectral images with 32×32 resolution. The outcome was impressive with 95.04% accuracy, and it had a better performance compared with SVM method.

Another work on proposing a Brain-Computer Interface system for mental state recognition based on real time EEG signals was introduced by Li et al. in [3]. A k-NN classifier built using the Self-Assessment Manikin (SAM) model identified three different degrees of attentiveness. Although the average accuracy peak is 57.03% but the method's advanced aspects are low latency in computation and real-time.

EEG data recorded from six people on the cognitive tasks was analyzed and classified by the SVM algorithm in [4]. The multiclass SVM classifiers were designed to detect five cognitive activities for each participant. The average accuracy estimated for all candidates was $93.33 \pm 8.16\%$. However, there was not a standard paradigm for all participants, and this work is only used for studying with the less practical contribution.

Aci et al. [5] developed a passive brain-computer interface using machine learning approaches for observing the attention states of human being. They designed the SVM model to classify three attention levels (focused, unfocused, and drowsy), then made a comparison with two other methods as k-Nearest Neighbor and Adaptive Neuro-Fuzzy System. The results were promising for the future work when the individual's attention identification reached 96.7% (best), and 91.72% (average) accuracy.

The volume of Internet of Things (IoT) devices is growing explosively with more than 75 billion connections to the Internet by 2025 as estimated [6]. It makes the trending of shifting computation to the edge devices is becoming reasonable. Additionally, this pattern is applicable to machine learning methods, particularly for inference runs, which requires significantly less processing power than the earlier training phase [7].

Several industry giants released the platforms to support implementing machine learning on embedded systems such as: Google has the Tensoreflow Lite, which supplies the powerful engines to convert the original models into the simplified and lighter version; ARM also released a free library that is only compatible with their Cortex-M processors; even STMicroelectronics introduced the X-CUBE-AI extension for STM32CubeIDE software to deploy deep neural networks on STM 32-bit microcontrollers feasibly.

A convolutional neural network on STM Nucleo-L476RG for human presence detection was presented by Cerutti et al. [8]. They used the Cortex Microcontroller Software Interface Standard Neural Network (CMSIS-NN) library for maximizing the NN efficiency. The network performed 76.7% of accuracy while solely used 6 kB of RAM, and consumed 16.5 mW in steady mode.

A substance detector named MobileNet-Single Shot Detector (SSD) was introduced by Zhang et al. [9], and used the well-known Caffe framework in a deep convolutional NN. That model was implemented on NanoPi2, using Samsung Cortex-A9 Quad-Core 1.4GHz, and 1 GB DDR3 RAM.

Emotion detection by a bracelet which could run multilayer NN was published by Magno et al. [10]. The power measurement proved that application could fit the mW power ARM Cortex M4F microcontroller. The emotion was detected with 100% of accuracy surprisingly while using only 2% of available memory.

Several examples were presented in [11] to give the fundamental knowledge of tiny machine learning. These projects were deployed on Arduino Nano 33 BLE Sense board, STM32F746G Discovery kit, and SparkFun Edge board.

There are a bunch of works on implementing NN on embedded systems, however most of them used the robust edge devices (e.g., Cortex-A9, Raspberry PI, Cortex-A53). A great work in [7] showed the effort to compensate the lack of works on mainstream microcontrollers.

1.2 Project duration

The time duration for this project is from 2nd May, 2022 to 31st August, 2022 (including holidays and weekends). The time estimation is an approximate evaluation and could be affected by exteriors (hardware and software resources such as boards, devices for collecting the own dataset; or project scope, etc.)

2. BACKGROUND

This chapter will supply the basic concepts about electroencephalography signals, machine learning classifiers, neural networks, and edge device computation. A section on electroencephalography which includes definition and fundamental features will be presented. Furthermore, there are parts describing the conventional classifying methods in machine learning. Lastly, embedded deployment contexts such as number representations and quantization will be investigated.

2.1 Electroencephalography

Electroencephalography (EEG) is the study of capturing and figuring out the electrical activity generated from the brain's surface. When the brain receives the impact from senses such as sight, hearing, taste, etc. it will produce the biological electrical signals which was transmitted through the neural system. Electrical activity can be recorded with electrodes placed on the scalp; each electrode as considered as a channel will capture the electrical pulse in each specific area. Depending on the application, the number of electrodes can be used in range of 2 to 512.

The EEG is recorded and displayed as waveforms of varying frequency and amplitude measured in voltage [12]. EEG consists of mainly 4 standard patterns: delta (0.5-4 Hz), theta (4-8 Hz), alpha (8-12 Hz), beta (13-30 Hz) as shown in Figure 2-1. Delta and theta signals are often monitored while human is in drowsy or asleep state. Graphic chart shows the amplitudes varies from 0.5 – 1.5 mV and reaches several millivolts at peaks. However, these values on scalp are within 10 – 100 µV regularly.

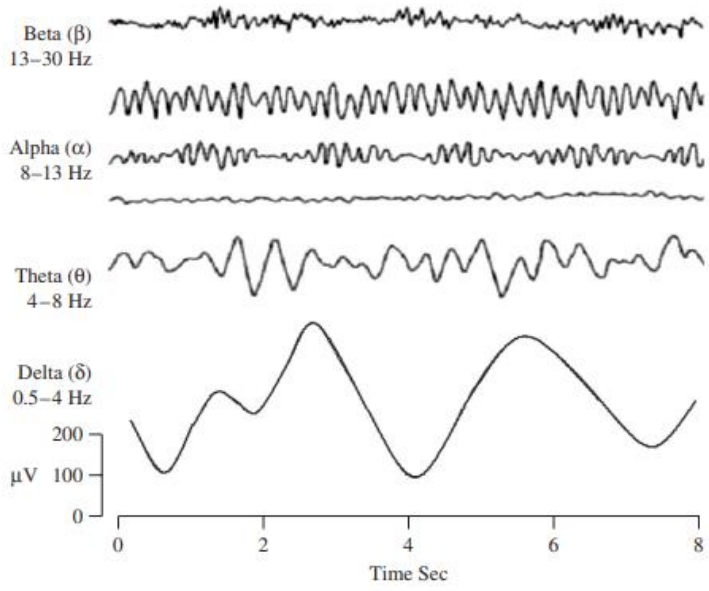


Figure 2-1: 4 typical dominant brain normal rhythms [13]

As proven, each frequency band for brain activity is identified with particular cognitive function. There is a considerable quantity of information in EEG signals which indicates the spatial, temporal, and spectral aspects. These benefits make EEG an alternative option to consider in not only neuroscience but also clinical treatments and disease diagnosis. However, in contrast, the EEG method also pays the cost of data processing complexity due to high dimensionality, non-stationary, and a low signal-to-noise ratio. Following the technology tendency, machine learning has been considered as a sufficient solution to engage in natural challenges of EEG approach.

In order to determine the user's mental state, raw EEG signals must be processed into the group of these signals. The two basic steps of a pattern recognition approach that is typically used to accomplish this translation are as follows:

- Feature extraction: the initial signal processing phase, tries to characterize the EEG signals by a small number of relevant values referred to "features" [14]. Such features should exclude noise and other irrelevant information when capturing the information included in EEG signals that is pertinent to describing the mental states to be identified. The arrangement of all extracted features into a single vector is called a feature vector.
- Classification: the second stage is where a class is assigned to a set of features derived from the signals. This class is appropriate for the type of identified mental state. Each classification algorithm is named as "classifier".

For example, the imagined left-right hand movement labeled process is shown in Figure 2-2. There are two mental states (imagined right hand and imagined left-hand movements) are distinguished. Band power features, or the strength of the EEG signal in a particular frequency range, are common characteristics that can be used to distinguish them from EEG signals. The following step is using a Linear Discriminant Analysis (LDA) classifier to detect the states.

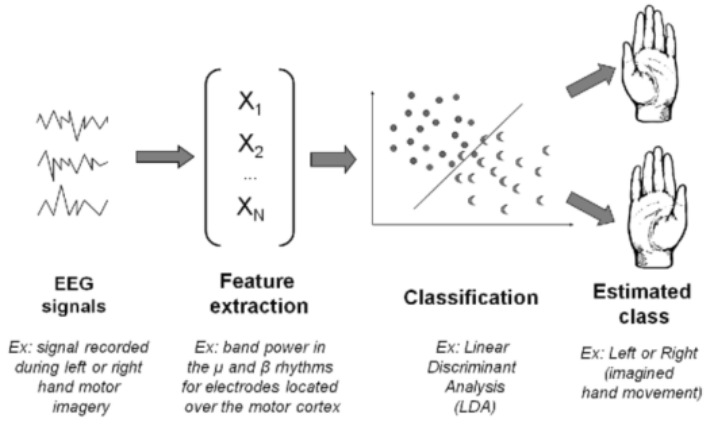


Figure 2-2: EEG signal processing pipeline [15]

2.2 Machine learning classifiers

Machine learning frequently encounters categorization problems, where the model must give anticipated class labels to a set of input data. Binary classification is used when there are only two classes from which to choose, and multi-class classification is used when there are more than two groups [16]. The classification accuracy is good parameter for evaluating performance and one can get additional details through monitor a confusion matrix. The confusion matrix plays a role as highlight potential issues such as whether the model frequently conflates two classes. Each row of the matrix corresponds to the instances in an actual class while the columns represent the classes that the model predicted [17]. The illustration of a confusion matrix is shown in Figure 2-3.

It is crucial that the input data must be balanced for categorization accuracy to be significant and pertinent. It means that each class should appear in the training dataset equally about time, number of samples, etc. A bias toward one class maybe occurred through an imbalanced class distribution. In the worst scenario, the accuracy of model will only perform the underlying class distribution [18]. There are some alternative indices using for measuring model's performance as precision and recall. Along with accuracy, precision and recall can also be estimated through true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN) quantities.

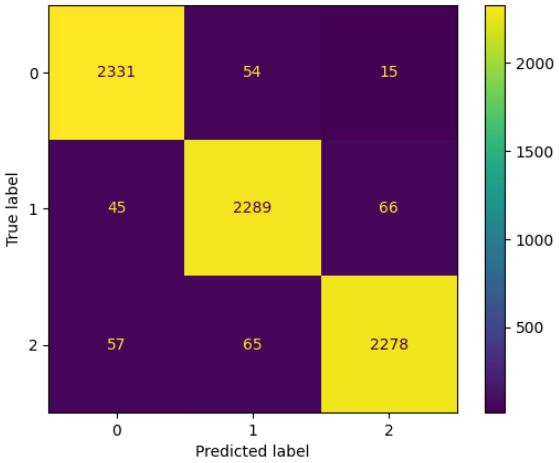


Figure 2-3: Confusion matrix for 3 classes

The following equations will be used for calculating each type of metric:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP} \qquad \textbf{Equation 2-1}$$

$$Recall = \frac{TP}{TP + FN}$$

Lastly, F1-score is defined as the metric that regard to both precision and recall with expression as:

$$F1_{score} = 2 * \frac{Precision * Recall}{Precision + Recall} \qquad \textbf{Equation 2-2}$$

In next subsections, some widely used classifiers in machine learning will be discussed briefly. They all support for classification effectively and belong to supervised learning algorithms.

2.2.1 Linear Kernel Support Vector Machine (SVM)

The SVM is a supervised machine learning model used for both classification and regression aims [19]. However, SVMs are usually applied in classification issues than by computing the hyperplane that best separates a dataset into two subsets. The advantages of SVMs are memory efficiency and able to determine the complex constraint between data samples. However, if the dataset is massive and noisy, the time execution will increase [20]. The simplest implementation of SVM is a linear kernel while the linear model only performs the first order function: $y = w * x + b$, where w and b are the support vector and the bias correspondingly.

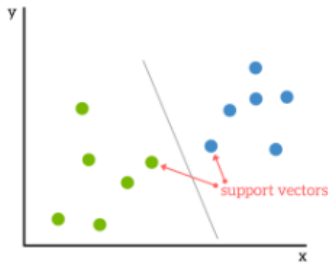


Figure 2-4: SVM mechanism illustration [19]

2.2.2 Extreme Gradient Boost (XGBoost)

XGBoost is a tree-based algorithm that also belongs to the supervised machine learning class as SVMs [21]. Although using the same tree-based algorithm as Gradient Boosting approach, XGBoost has a different way of building a tree decision where it drives the best node division through Similarity Score and Gain values. The node split that achieves the highest Gain is the best choice for the tree [22].

2.3 Artificial neural networks

Artificial neural network (ANN) is a concept that was presented in mid-1940s and was defined as "… a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs." – by Dr. Robert Hecht-Nielsen – the inventor of the first neurocomputer [23]. Many ideas supposed that artificial neural networks were first inspired by neurology and biological information processing. Simply, ANNs are composed of several computational layers; each layer is made of multiple artificial nodes, which play a role as biological neurons of the human brain. The nodes interact with each other by *links* and each link is presented by a *weight*. The input of ANNs can be the raw data or features, then the nodes will do computations on input data and pass the results to other neurons. Output at each neuron is defined as *activation* or *node value.* Figure 2-5 shows the simple architecture of neural network that consists of several dense layers. A dense layer or a fully connected layer takes responsible for connects all outputs from every node from the previous layer to the inputs of neurons in the next layer.

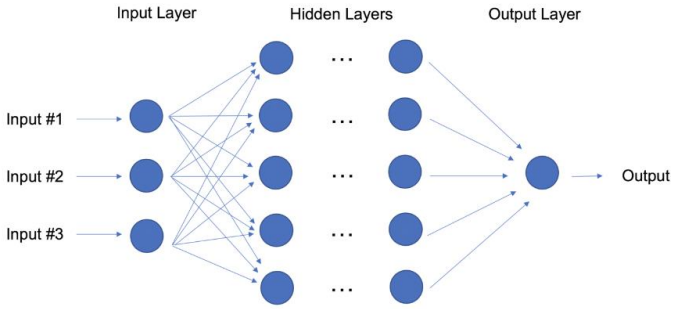


Figure 2-5: An ANN with multiple hidden layers

ANNs have the ability to learn, which happens through changing the weight values. It is one of the most impressive and renowned machine learning algorithms, which can be deployed in a diversity of applications, such as natural language processing (NLP), image recognition, prediction of stocks, medical analysis or disease diagnosis, etc.

Deep learning was introduced as the branch of machine learning area using artificial NNs. The definition of *deep* can be understood as neural networks include multiple (hidden) layers in its architecture [24], or maybe it associated with the *deeper* understanding through learning on data directly rather than using handcrafted features as input. There are variety of deep-learning architectures such as: deep NNs, deep reinforcement learning, convolutional NNs and transformer, however these frameworks are out of this work scope, but can be recognized for future improvements and implementations.

2.3.1 The perceptron

The fundamental block of most artificial neural networks is a single neuron – the perceptron as considered. The perceptron is also known as a single-layer neural network that composes of input values, weights and bias, net sum, and an activation function. Basically, it calculates the output *y* from input signals $x_1, x_2, \ldots, x_n$ by multiplying each input x_n with a specific weight w_n then adding them together into the weighted sum [25].

The bias is optional to put in the sum. Lastly, an activation function is used to map the data to the final output. The schematic of a perceptron is illustrated in Figure 2-6.

From the schematic, the output can be formed consequently as:

$$y = \varphi(w_1x_1 + w_2x_2 + w_3x_3 + \cdots + w_nx_n + b) \quad \textbf{Equation 2-3}$$

Where φ is an arbitrary activation function. Moreover, the formular can be rewritten in a matrix form:

$$y = \varphi(\boldsymbol{X^T W} + b) \quad \textbf{Equation 2-4}$$

With: $\boldsymbol{X^T} = [x_1\ x_2\ \dots x_n]$ and $\boldsymbol{W^T} = [w_1\ w_2\ \dots w_n]$.

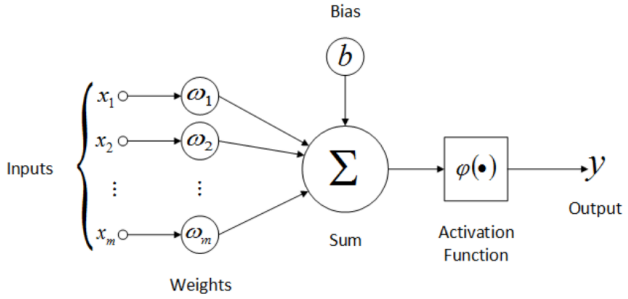


Figure 2-6: A perceptron schematic

As consequently, the neuron can only solve the linear problems when missing the activation function. So that non-linear features of activation functions can help the ANN to be able to deal with more complicated problems and performing arbitrary functions.

2.3.2 Activation functions

There are 2 common activation functions used in NNs: the *Sigmoid* function, and Rectified Linear Unit or *ReLu* function as Figure 2-7. The Sigmoid function (Equation 2-5) is suitable for solving probabilistic problems. However, an unexpected issue for this precise function is the *vanishing gradient problem*. Briefly, the gradient will be strikingly small at the ends of the output space due to the function's derivation, the weights in neural networks can be updated barely. This cause leads the neural network to work less effectively or even stop for further training.

$$f(x) = \frac{1}{1 + e^{-x}} \quad \textbf{Equation 2-5}$$

To compensate for the disadvantage of the *Sigmoid* function, in 2010, Vinod Nair and Geoffrey E. Hinton introduced the new activation function: *ReLu* [26]. The *ReLu* (Equation 2-6) is a non-linear function that if the input is positive, the output will be unchanged and if the input is below zero, it will output zero. This function is also the most used function at the moment because of its simplicity and efficiency.

$$f(x) = \max(0, x) \quad \textbf{Equation 2-6}$$

At notion, in the last layer of neural networks, a *Softmax* function (Equation 2-7) is often handled because the sum of all output values equals 1 and can be described as a probability distribution. The summation of data will be arranged in the range of 0 to 1 and represents the predicted output of the neural networks.

$$f(z) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad \textbf{Equation 2-7}$$

Figure 2-7: Activation functions: Sigmoid (left), ReLu (right)

2.3.3 Training the network

The training of the network is the loss function optimization step, also called a cost function. The loss function indicates the error of the network, then the purpose of optimization is to minimize the value of the loss function. Equation 2-3 shows the relationship between input and output through the set of weight values. Determining the correct weight values is done by using the concept *gradient descent* of the loss function. If the gradient descent vector moves toward the negative gradient, the loss will be reduced as quickly as possible.

There are two main stages in the training process: the training phase and the validation phase. The original data set is separated into two subsets; the training set gives the parameters to the network for learning, and error estimation will be executed through validation data. The ratio between two subsets is defined as the validation split. The usual ratio is 80 percent for training data and the remaining amount for validation evidently.

Another significant aspect that needs to be considered carefully in the training process is hyperparameters adjustment. Several factors can be used for optimizing the neural networks: structure of the neural networks, kernel size, learning rate, etc. Especially when implementing a neural network on microcontrollers that has restrained memory, the arrangement of hyperparameters is extremely essential.

2.4 Machine learning on microcontrollers

Nowadays, the trend to expand machine learning in edge devices and microcontrollers is attractive to more researchers, especially in Internet of Things (IoT), sensor fusion, and synthetics sensors areas. Conventionally, all the steps of machine learning have been completed on the cloud or the server computers, which have powerful computation abilities and limitless storage. However, this method creates many concerns: latency, scalability, and privacy [27], [28]. Dividing the computation portions to the edge devices helps to alleviate the shortcomings of the approach. For example, data privacy will be secured since less data is sent to the server and mostly done on the devices. Furthermore, the bandwidth is also retained because fewer frequency resources are used for transmitting data.

In contrast to sufficient pros, there are still limitations to deploy machine learning on embedded devices. Due to resource-scarce property, the complex computation and large memory size are obstacles in machine learning deployment on microcontrollers. Typically, there are some target applications that are already invested in as activity recognition, voice recognition, and simple classification.

In this work, the current method for implementing machine learning on microcontrollers is to first train a model (neural network) on the computer or the cloud, then convert model to a simplified version, offload it on the target device, finally perform the inference. Tensorflow Lite platform is a tool from Google to convert a neural network model and an

proper open source library CMSIS-NN is specifically manipulated for microcontrollers with Cortex-M processor in implementing optimized neural network algorithms [29].

2.4.1 Fixed-point quantization

In order to represent data, there are two frequent ways: *fixed-point format* and *floating-point format*. In deep learning, the 32-bit floating-point format is utilized mostly due to the good accuracy. The drawback of this format are the memory occupation and complexity of operation. Otherwise, fixed-point numbers are often applied to perform values on microcontrollers and digital signal processors (DSP) for productivity and reducing the memory. The technique converting 32-bit floating-point numbers to the fixed-point numbers is called *quantization*.

Impressively, the only difference between a fixed-point number and its floating-point counterpart is the range that each one represents. The range of fixed-point numbers is always linear. It can conserve that the smallest error will always be well-defined and constant as the step between two successive values. Moreover, the real numbers performed by fixed-point numbers will be stored as integers in the memory with limited resolution. As defined in CMSIS library [30], Q-notation is used to express the schema for converting these integers to real numbers, or vice versa. Q-notation is form of $Qx.y$ that x is number of bits for integer part (also include sign bit) while y specifies for decimal part. The range of representable values depends on what kind of notation used. In general, the value ranges can be expanded from -2^{x-1} to $2^{x-1} - 2^{-y}$ (signed number) [31].

Post-training quantization is one of the quantization approaches that can be used. This strategy is easy to put into practice; there is no changes to training procedure required. Quantization of learnable parameters and quantization of activations make up post-training quantization process. The first class – quantization of weights and biases, is simplified because these parameters are set, and the quantization range is accessible determined. As shown in Figure 2-8, the activation quantization depends on the input data, and is also needed to bypass the de-quantization of weights and biases in previous steps.

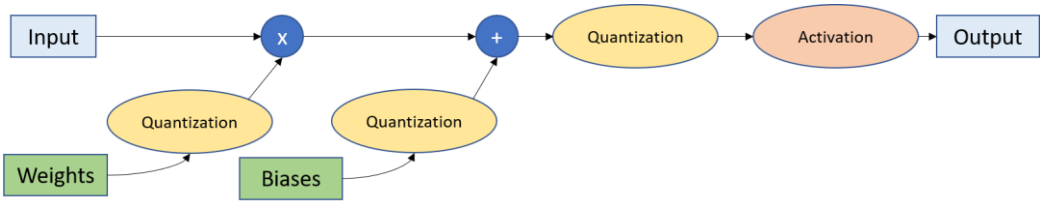


Figure 2-8: Post-quantization in an ANN layer

In this work, only quantization of weights and biases stage is invested and deployed. Because the model after quantization has sufficient memory size to implement on microcontrollers.

2.4.2 The STM32L475 discovery kit

It is essential to choose an algorithm to operate on an embedded device. However, parallelly selecting an optimal hardware solution is also tricky. The criteria for hardware choice are according to accuracy, energy consumption, and cost [32]. Several microcontrollers can be used for artificial intelligence applications but invoking the algorithms on them requires more effort. Nevertheless, microcontrollers are excellent choices if they can run networks that are not too large for rare data fusion activities [28]. The X-CUBE-AI [33], which is only compatible with STMicroelectronics microcontrollers, is a valuable tool for facilitating the deep neural networks implementation on microcontrollers. The tool is an extension of the STM32CubeMX environment, which

enables automatic conversion of pre-trained NNs to scarce-resource hardware. Moreover, X-CUBE-AI also improves libraries by modifying layers and mitigating the number of weights. It helps the model is slighter and more friendly in term of memory. Among some tiny hardware for IoT purposes recommended on the Tensorflow Lite site [29], STM32 microcontrollers are a great option. Therefore, in this work, the microcontroller STM32L475VG based on an Arm Cortex M4 core is chosen for executing inference. Its features include, among other things, 80 MHz of maximum operating frequency, 1 MB of flash memory, 128 kB RAM memory including 32 kB with hardware parity check, 5 embedded universal synchronous/asynchronous receiver transmitter (USART) using baud rate up to 204 Kbaud [34]. As any Arm Cortex-M4 processors, this microcontroller features a floating-point unit (FPU) and using ultra-low-power. The peak current in the *Standby* mode is 420 nA, which proves that it also meets the requirement of hardware choice.

Table 2-1: Board specification

Board	MCU	Clock speed	Flash memory	SRAM	Cost
STM32 B-L475E-IOT01A1	32-bit Arm Cortex-M4	80 MHz	1 MB	128 kB	$53

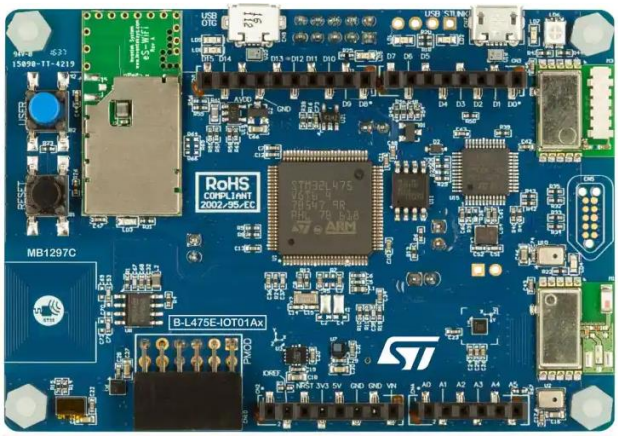


Figure 2-9: The B-L475E-IOT01A discovery kit [35]

3. IMPLEMENTATION

This chapter describes the details of the using dataset firstly. In the next step, the signal processing procedure and training phase (including NN training) are presented in different scenarios. Finally, implementing a neural network on the microcontroller is illustrated as well as evaluating and verifying criteria.

Due to unexpected causes, the self-collected dataset was not done then the used dataset in this project is available on [36].

3.1 Dataset

The data collection was built for monitoring the attention states in human being using passive EEG Brain-computer Interfaces (BCI). The original dataset of 25-hour EEG recordings from 5 individuals engaged in a low-intensity control task was used in this investigation. The task entailed using the “Microsoft Train Simulator” program to control a computer-simulated train. In each experiment, participants used the simulation tool to drive the train for 35 to 55 minutes over a mostly nondescript route [5]. Each person joined 7 experiments, in which 2 first exams were used for subject familiarize with the process, and the last 5 records were helpful data. All the EEG data was collected by EMOTIV device.

However, there were some points needed paid attention to:

- Total number of records is 34 instead of 35 because the last participant only took 6 experiments. All the export files are available as .mat format that can be imported to Matlab or Python.
- There are 3 mental states labeled in dataset: the focused, the unfocused and the drowsy state. Time distribution for each state is: the focused was measured during first 10 minutes, then the unfocused occupied the next 10 minutes, lastly the remaining slot was taken by the drowsy.
- The 14 channels feasible in the dataset are AF3, F7, F3, FC5, T7, P7, O1, O2, P8, T8, FC6, F4, F8, and AF4. However, only 7 channels named F7, F3, P7, O1, O2, P8, and AF4 have non-corrupt data. It is the reason this work only considered them as the proper channels for processing.
- The sampling frequency is $F_s = 128\ Hz$.

Table 3-1: Sample data

Cnt	Intp	Chan. F7	Chan. F3	Chan. P7	Chan. O1	Chan. O2	Chan. P8	Chan. AF4	X	Y
30	0	4332.8	5312.8	4566.7	4651.8	4338.5	4445.6	4486.7	1571	1716
31	0	4334.4	5315.4	4569.2	4655.4	4343.6	4451.8	4485.1	1572	1717
32	0	4343.1	5319.5	4569.7	4662.6	4349.7	4465.6	4485.1	1572	1718
33	0	4340.5	5319.5	4561.5	4659	4351.3	4465.1	4489.2	1572	1717
34	0	4331.8	5316.4	4555.4	4651.8	4343.1	4459.5	4490.3	1572	1718
35	0	4328.7	5309.7	4554.9	4651.8	4329.2	4453.8	4481	1573	1720
36	0	4327.2	5302.6	4552.3	4650.3	4327.7	4445.6	4473.1	1572	1720
37	0	4324.1	5300	4549.7	4645.1	4331.8	4442.6	4464.6	1571	1720
38	0	4324.1	5301	4551.3	4643.1	4328.7	4439	4455.9	1569	1720
39	0	4326.7	5302.6	4551.8	4643.1	4327.7	4435.4	4446.7	1568	1716
40	0	4325.1	5302.6	4553.3	4643.1	4334.9	4440	4446.7	1566	1717

With Cnt = sample counter; intp = indicate if data is interpolated; X, Y = gyroscope axis.

The general block diagram of implementing neural network for EEG classification on microcontroller (Figure 3-1) shows that raw data needs to be processed before training the machine learning models.

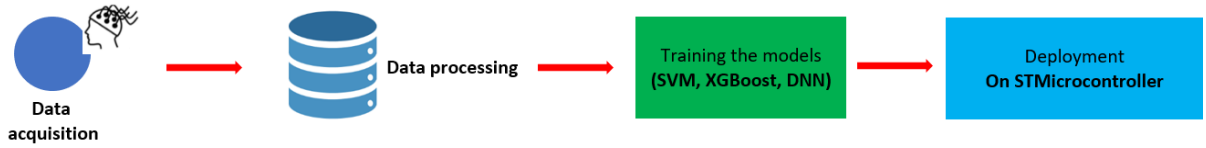


Figure 3-1: The general work flow of implementing neural network on microcontroller

Thus, the next subsection will present the data processing procedure.

3.2 Data processing

The flowchart of data processing was inspired by the original work in [5] with illustration as shown in Figure 3-2.

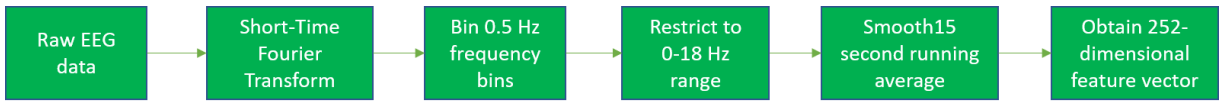


Figure 3-2: Feature extraction step

Processing the time-series signals as EEG signal can be solved through several approach. One of the most convenient tools is Fourier transform. In this step, EEG signals in each channel will be represented in time-frequency domain, using a Fourier-related transform – the short-time Fourier transform (STFT). Because the continuous EEG signals were sampled thus it can be considered that the obtained data is discrete-time data. The discrete-time STFT can be expressed as [37]:

$$STFT\{x[n]\}(m,\omega) \equiv X(m,\omega) = \sum_{n=-\infty}^{\infty} x[n]w[n-m]e^{-j\omega n} \qquad \textbf{Equation 3-1}$$

Here, $x[n]$ is the EEG signal in single channel, $w[n]$ is window while m is discrete and ω is continuous. The spectrogram is calculated by raising the STFT magnitude to the power of 2:

$$S(m,\omega) = |X(m,\omega)|^2 \qquad \textbf{Equation 3-2}$$

STFT is calculated for each channel. The STFT's characteristic is dividing the time signal into equal length segments and then computing the Fourier transform in each segment. Hence, after doing STFT for each $\Delta T = 15$ second fragment, the Blackman window was applied to subside the EEG signal at two sides of each segment. The Blackman window function is described as [38]:

$$w(k) = \begin{cases} 0.42 - 0.5cos\dfrac{2\pi k}{M-1} + 0.08cos\dfrac{4\pi k}{M-1}, & 0 \le k < M \\ 0, otherwise \end{cases} \qquad \textbf{Equation 3-3}$$

With $M = F_s . \Delta T$ is the total time points in the window, and k is discrete-time index.

After determining STFT in each channel, the achieved spectrum represents power density distributed over $\frac{n_{fft}}{2} + 1$ frequencies with n_{fft} is fast discrete Fourier transform length. The bandwidth of each sub-carrier is $\omega_l = lF_s/n_{fft}$ where l is in range 0 to $n_{fft}/2$. The following steps as binning frequency and frequency range restriction, will be explained details below.

In this work, the necessary parameters are given: $n_{fft} = 2048$, $M = 128 . 15 = 1920$. Thus $\omega_l = 0.0625l\ Hz$ with l changed from 0 to 1024. Binning 16 sub-carriers into the 0.5Hz frequency bin by using average, the frequency band of EEG signal after STFT spreads from 0 to 64 Hz with step equals 0.5 Hz. The frequency range was limited in range of 0 to18 Hz. Thus, there are only 36 frequencies at the final product of signal processing step. Finally, the spectrogram was softened through a smoothing window. As discussed in [5], the temporal width of STFT's window and the smoothing window is essential parameter. The selection for these parameters was done by experiments, and the choice of 15-second gave a good compromise.

There are 7 channels in each EEG record, the final feature vector was composed by forming the spectrum distribution from all 7 channels into a single. The feature vector used for training neural network has the dimension of $36 . 7 = 252$.

3.3 Model training

The training stage was completed in Python running on Google Colaboratory (or Google Colab). As mentioned in subsection 3.1, the lengths of three mental states are unbalanced. The drowsy class is kept for only first 10-minute period (which will be explained more in the next section) to avoid the bias effect in the classified prediction. The evolution of this work performs through 3 evaluation paradigms specifically as:

- Specific-subject paradigm: the classifiers were trained for each subject individually based on that participant's data records only. The sub-dataset will be split into 80% for the training and 20% for validation (or testing). For each participant, there is a mental state detector was used.
- All-subject paradigm: in this case, a single classifier was built for all subjects. 80% data of all EEG records was randomly chosen for the training stage, and the remaining data is spent for evaluation.
- *Adaptable* paradigm: Beside the implementation neural network on microcontroller purpose, other target of EEG mental detector is having a model which can predict at good accuracy for the new data. In this situation, the dataset was divided into 3 subsets: training, validation and testing set with special ratio. The training set was selected from randomly three participants' data, the data of 2 resting people will take responsible one for the validation data, and other one for testing set.

Unlike SVM or XGBoost which can be utilized from the supporting libraries, deep neural network is built through the Tensorflow and its wrapper Keras packages [39], [40]. The model composed of one Flatten layer (which reshapes the input), two fully connected layers with the *ReLu* activation function, followed by a single dense layer with a *softmax* activation function. The neural network architecture can be observed in Figure 3-3.

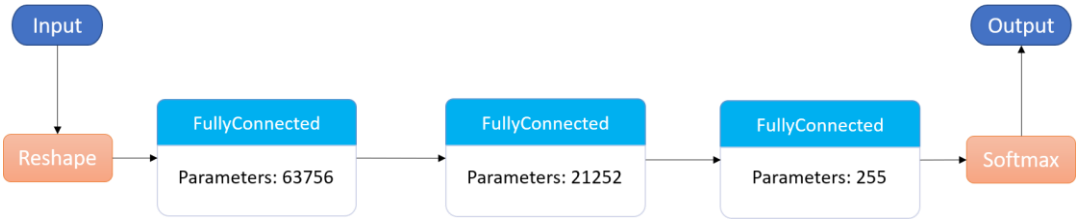


Figure 3-3: Neural network diagram

The model has total of 85263 parameters of size; therefore, it could pretty fit the STM32L475VG kit specifications. During the training phase, the optimizer algorithm Adam was used to mitigate the cross-entropy loss between true labels and predictions. In order to avoid overfitting, an early stopping was utilized with the *patience* of 2 epochs. The total epochs are 100, and batch size was fixed as default. Finally, the check-point

function was used to save the best weight model. The model can be saved in some formats as: *SavedModel* – a Tensorflow model saved on disk, *Keras model* – a model created using the high level Keras Application Programming Interface (API), or *Keras H5 format* – a light-weight version of *SavedModel* format supported by Keras API [41]. The model built in this work was in *Keras H5* format because X-CUBE-AI supports to import 3 types of models into STM32 microcontrollers: Keras, TFLite and ONNX (Open Neural Network Exchange). Coincidently, the saved model has light size so that it does not require the conversion step.

3.4 Neural network deployment and inference

Finally, the trained neural network was deployed into microcontroller for the mental state classification. The details of this step will be described in the section ANNEX I. In short term, after configuring all the peripherals, connectivity standard, etc. the model needs to be validated in two mechanisms supported by X-CUBE-AI tool: *validation on desktop* and *validation on target.* The aims of these works are comparing the original deep learning model with its generated X86 C model (runs on the host/computer) and C model (runs on the microcontroller).

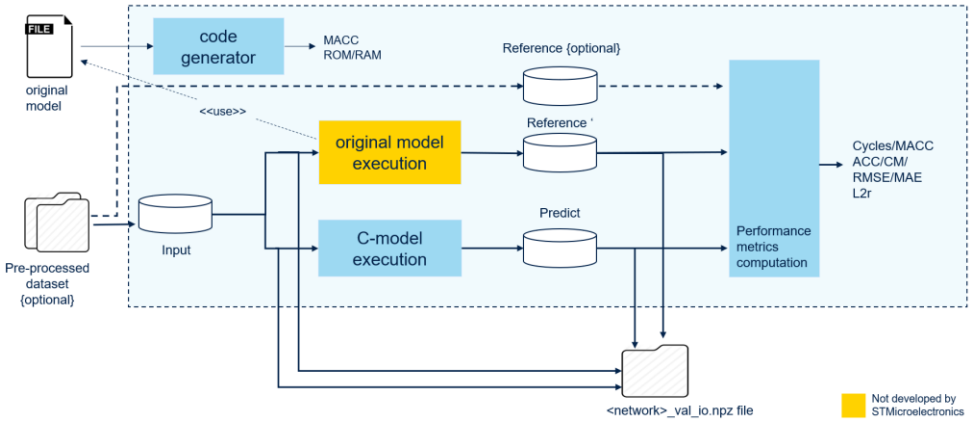


Figure 3-4: Validation flow overview [33]

The program was written in C and using Hardware Abstraction Layer (HAL) in STM32CubeIDE – the development environment for ST microcontrollers. At specified times, a test data in *.txt* format will be sent from the computer (with Linux operating system) to the board, the microcontroller will run the AI model and gives the prediction. There are 2 ways for displaying the result: by LED on the board or display on the computer's interface. The board communicated with the computer through USART protocol.

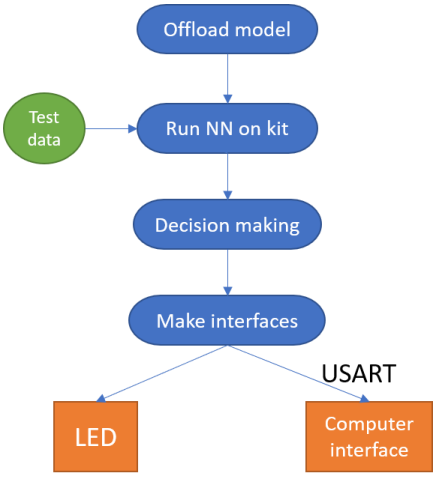


Figure 3-5: Flowchart of implemented AI model on STM32L475VG

4. RESULTS

This chapter presents the results of classifiers on three paradigms and also evaluates the contribution of each EEG channel in distinguishing the different attention states. Finally, comparing different method's results will explain this work orientation.

4.1 Specific-subject paradigm

The performances of classifiers are evaluated in terms of accuracy. Only SVM and XGBoost are tested in this pattern.

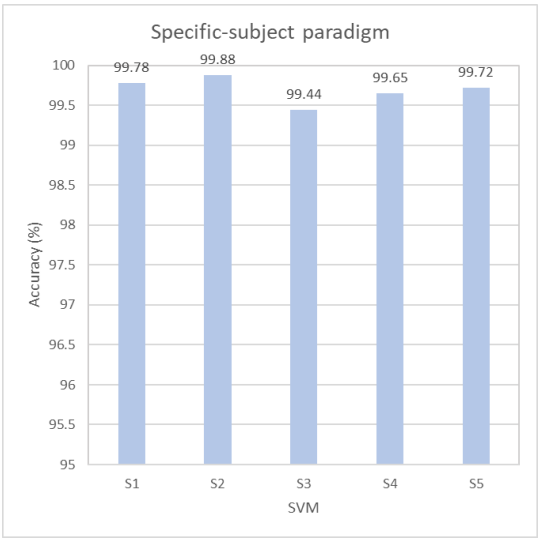


Figure 4-1: Accuracy in SVM method

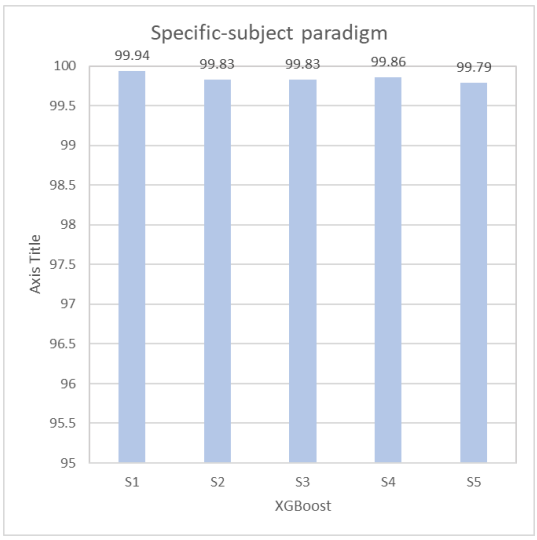


Figure 4-2: Accuracy in XGBoost method

Figure 4-1 and Figure 4-2 show the accuracy of SVM and XGBoost for the specific-subject situation. As observed in the bar charts, the average accuracy for SVM lasts from 99.44 to 99.88%, and XGBoost is even more successful with a slightly higher 99.79 to 99.94%. These results can be explained clearly because the classifier was used for training and test sets of one subject. The data was homogenous for all records of each participant.

Moreover, the channels have been estimated the weights to monitor which electrode contributes more valuable data for mental state detection. The procedure of this task is, each channel was ranked in list of weight values, the electrode with smallest portion will be removed permanent from dataset. The new data set will be spent for training the

models. This process repeated until only one channel left. The channels were listed in a decreasing order as: F7 – F3 – O2 – P7 – O1 – P8 – AF4. The below figures will present the accuracy done by two algorithms SVM and XGBoost.

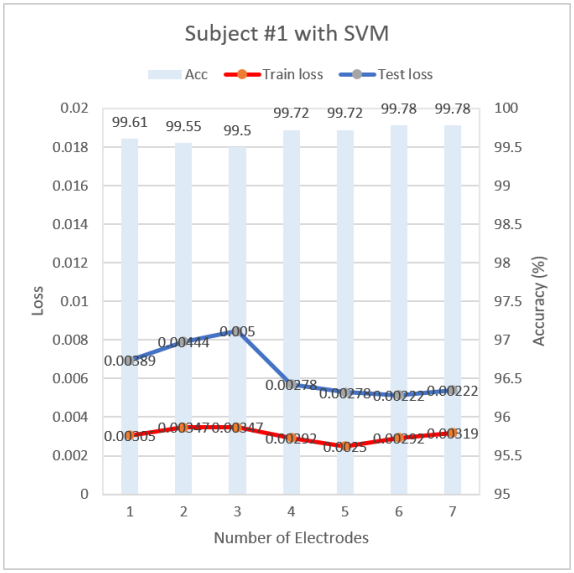


Figure 4-3: Accuracy varies with number of electrodes in SVM

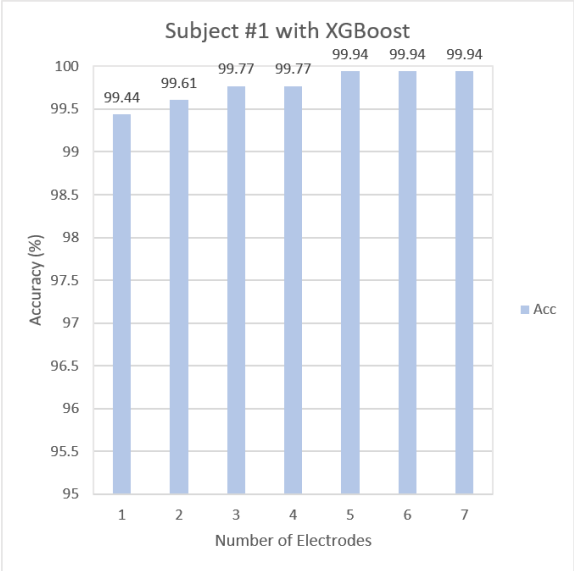


Figure 4-4: Accuracy varies with number of electrodes in XGBoost

4.2 All-subject paradigm

In this case, there is a small difference of feeding models. In particular, the whole dataset was divided into two subsets (training and testing sets) by *split* function with ratio of 80:20 randomly, then fed to train SVM and XGBoost. While protocol of selecting dataset for neural network is: choosing one of the participant's recording files as testing set, all remaining files is occupied for training set. The performance of SVM and XGBoost is given in Table 4-1.

Table 4-1: The accuracy results for all-subject paradigm

Method	Accuracy
SVM	**99.72%**
XGBoost	**99.56%**

About neural network, the results of prediction were expressed as shown in below figures.

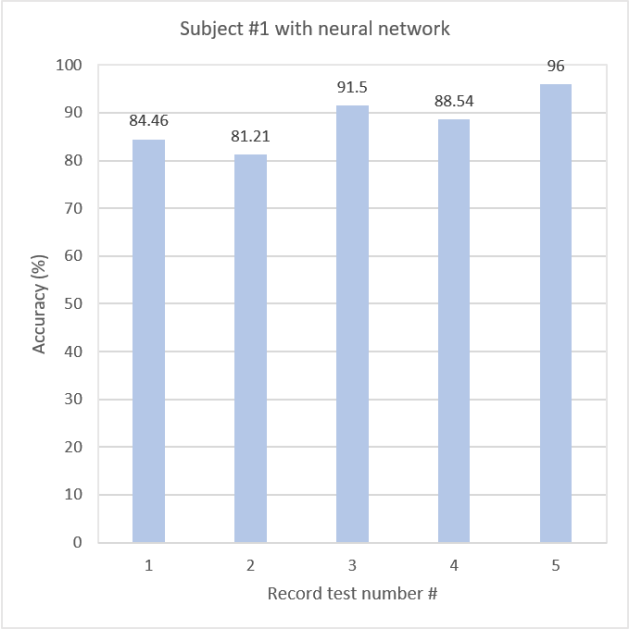


Figure 4-5: Accuracy of NN over each record of 1st subject

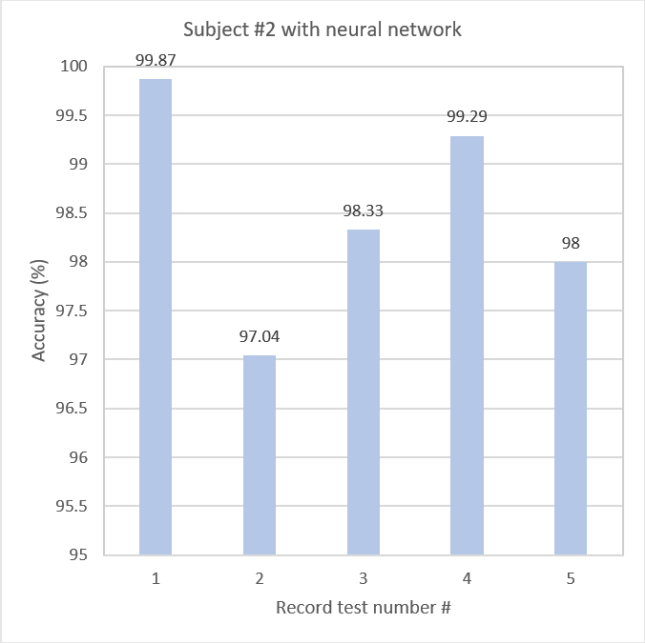


Figure 4-6: Accuracy of NN over each record of 2nd subject

Obviously, the obtained accuracy is good enough to classify the mental attention states when considering about values. It only proves that the model can work. However, its efficiency is still questionable. Although the training and testing sets were separated, the two subsets were still mixed in learning features. For example, if the testing set is one record of 1st participant, and all others were used for the training set, the model will learn the features from the 1st person apparently and test exactly on that person. It increases the prediction rate but gives less scientific contribution.

4.3 Adaptable paradigm

The last paradigm results will explain why the neural network is essential and beneficial for EEG signals classification task rather than conventional approaches. Because the

XGBoost's accuracy is better than SVM's so this task only acknowledges XGBoost and neural network performance. Firstly, the XGBoost's achievement in classification is presented through Figure 4-7

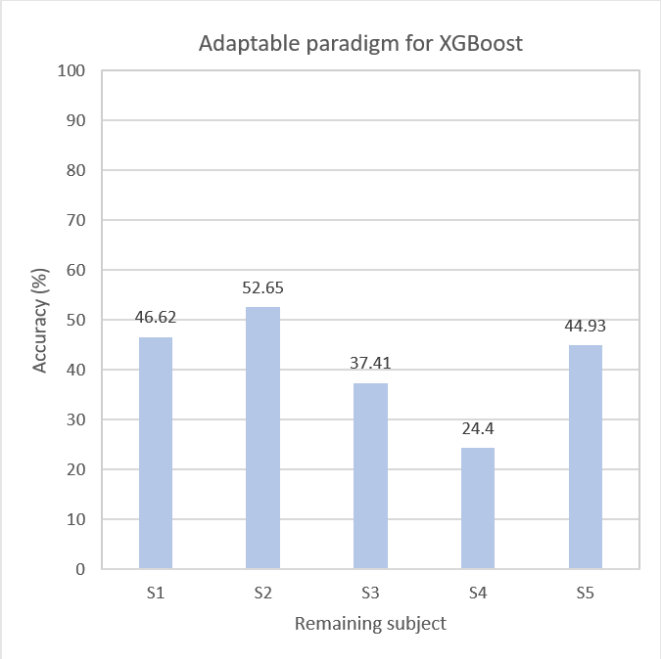


Figure 4-7: Accuracy results of XGBoost in adaptable paradigm

It is effortless to analyze that the accuracy dropped drastically to compare with two previous examinations, just from 24.4 to 52.65% (41.2% on average). Precisely, three sets of data were manipulated for the neural network: training set (occupies the data of first three people), validation set (occupies the data of fourth person), and the testing set (occupies the last participant). The accuracy is only 33.34%.

The possible cause for the low rates could be the dataset is not large enough. Thus, the models do not have a sufficient number of features to learn. One of common solutions for the scarce dataset is using data augmentation to increase data volume. Because of the personal computer's limitation, only *jittering* and *scaling* transformations were applied to enlarge the dataset. In a compressed explanation, *jittering* is the step that adds the noise into the original signals, and *scaling* is the step that scales each time series by a constant amount. The argument to adjust *jittering* and *scaling* quantities is *standard deviation* σ.

Although applying data augmentation method, XGBoost did not show a significant improvement. The average accuracy stayed at 44.14%. In contrast, the neural network performed the significant jump. By tuning the standard deviation to shape the fit dataset, accuracy of the neural network increased approximately 26% up to 59.75% with the deviation pair of 0.3 and 0.05 for the jittering and scaling correspondingly.

For this work, the neural network showed the domination to the conventional approach, even with simple architecture. From here on, only discussion and analysis of the built neural network will be presented.

4.4 Inference

The neural network after deploying into microcontroller will be evaluated based on the accuracy, time execution and memory consumption. As mentioned above, the model needs to validated through X-CUBE-AI extension. For testing inference, another neural architecture was built with more simple structure. It consists of 2 dense layers (252 and 3 nodes corresponding) instead of 3 layers as illustrated Figure 3-3. On basis, the procedure on running inference is remained.

The accuracy here is the cross accuracy between the reference and C model, as shown in Table 4-2. Because the model was kept the original size (~269 kB), there was not the effects from quantization step; thus, the cross-accuracy reached 100%. It means that the C-model's performance is exactly same as the reference model.

Table 4-2: The cross-accuracy report

Output	Accuracy	RMSE	Mean	Std
X-cross	**100%**	**0.000000116**	**0.000000001**	**0.000000118**

With RMSE = Root Mean Square Error, Std = Standard deviation

The total time execution is dependent on the device workload. Hence, on time evaluation criterion, only time execution per each layer in percent was assumed.

Table 4-3: Execution time per layer

c_id	Layer type	Time (ms)*	%
0	**Dense**	**0.127**	**92.5%**
1	**NL**	**0.003**	**2.2%**
2	**Dense**	**0.003**	**2.5%**
3	**NL**	**0.004**	**2.9%**
		0.138 ms	

*: *accurate to 3 decimal places*

The first layer captured a large portion of time due to its size being dominant over other layers. The last concern is memory usage which is approximately 269 kB for both flash and RAMs. The total parameters in the C-model are 64515 items, and multiply-and-accumulate operations are 64812.

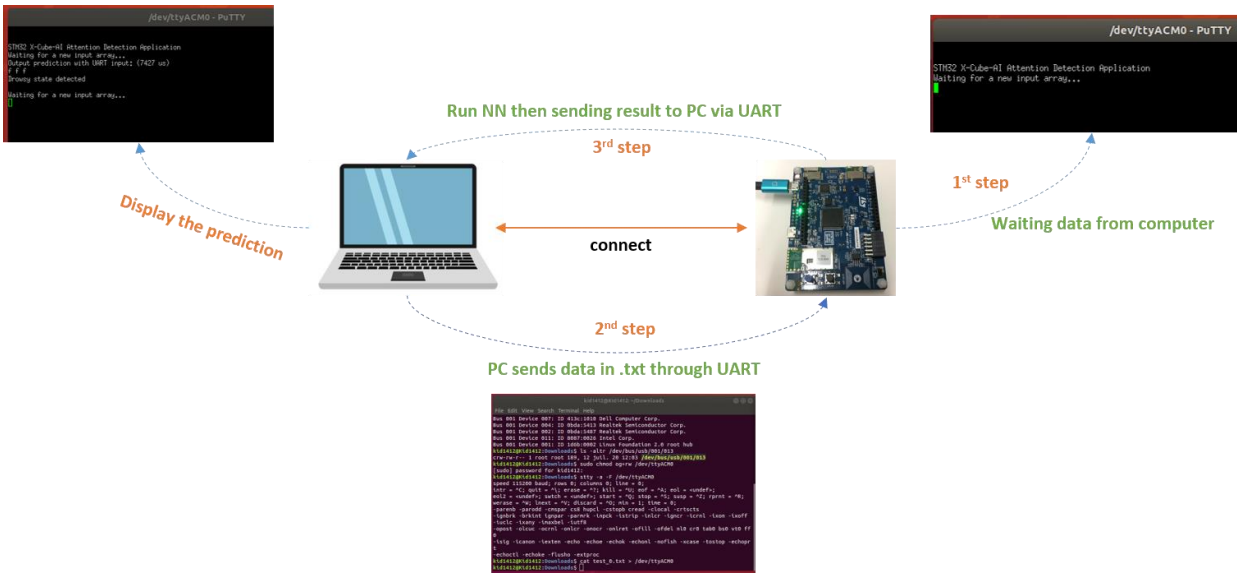


Figure 4-8: Testing inference

Figure 4-8 illustrated whole procedure of performing inference. The board is ready to receive data from the computer. After data sent from computer, the AI model will run and give the prediction. In this demonstration, the drowsy state was detected and notified by displaying on the computer via an SSH and telnet client – PuTTY. The green LED on the board also indicates the drowsy state (as defined in program).

CONCLUSION AND FUTURE WORK

This work presented a deployment of EEG signal classification on a microcontroller, principally on STM32L475VG microcontroller from STMicroelectronics. It performs the comparison between three supervised machine learning algorithms (SVM, XGBoost, and neural network). The conventional methods gave better accuracy with the *homogenous* data (dataset from only one subject). While in the more logical and natural case as the *Adaptable* paradigm, neural network took the preference.

The neural network could classify three mental states *focused*, *unfocused*, and *drowsy*, and be implemented on the STM32 IOT kit by handling the STM X-CUBE-AI package. The model was also evaluated for performance in time execution, accuracy, and memory usage. In advance, this work proves that the application of brain signal study is possibly accomplished on scarce-resource devices. In terms of practice, I believe this work can be a good practical lab for students in tiny machine learning courses.

However, there are some problems needed to invest: 1). the network accuracy is still low (approximate 60%), and 2). The data is still processed manually before feeding the network. To deal with these issues, I proposed the possible solutions:

i). **Building the own dataset with more participants**: as proven, with the enriched dataset (after augmentation), the neural network performed a better outcome. The own dataset creation is a time-consuming task but it brings more benefits and has deeper vision for researching.

ii). **Manipulating or designing an architecture for neural networks combining the new data processing approach**: the model implemented in this work was very simple in both structure and algorithm. Thus, the ideas of new architectures to improve performance is necessary. However, the complex model will increase the computing complication and size of the model. How to balance these constraints is also the future work to develop this work.

I also tested the built model named *Vision Transformer* from [42], [43] for this work, and the result was surprising. The model ran during 200 epochs, and the obtained accuracy for the *Adaptable* paradigm reached 71.76%. It was a promising achievement. The train, validation losses, and confusion matrix are presented below:

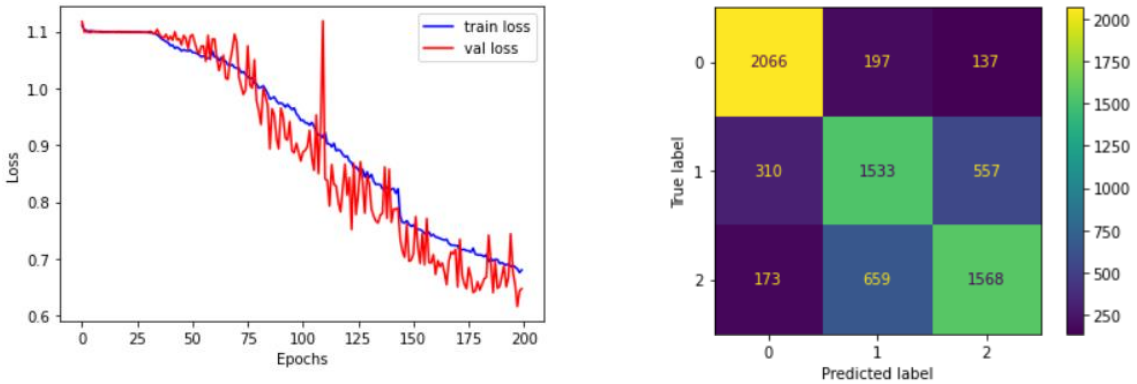


Figure 0-1: Transformer model's performance

The best quality of this approach is it used the raw data (in time series) as input. Thus, it does not need to do the feature extraction step manually. That is why this direction is so natural and promising. However, the size of pre-trained model is large (approximate 12.6 MB), it prevented the implementation for the STM32L475VG with only 1 MB flash. This issue can be done with quantization or pruning techniques in future work.

ABBREVIATIONS - ACRONYMS

ANN	Artificial Neural Network
BCI	Brain-Computer Interface
CMSIS	Cortex Microcontroller Software Interface Standard
DSP	Digital Signal Processor
EEG	Electroencephalography
FN	False Negative
FP	False Positive
FPU	Floating Point Unit
HAL	Hardware Abstraction Layer
IoT	Internet of Things
k-NN	k-Nearest Neighbors
LDA	Linear Discriminant Analysis
NLP	Natural Language Processing
NN	Neural Network
SAM	Self-Assessment Manikin
STFT	Short-time Fourier Transform
SVM	Support Vector Machine
TN	True Negative
TP	True Positive
USART	Universal Synchronous/Asynchronous Receiver Transmitter
XGBoost	Extreme Gradient Boosting

ANNEX I

A. Creating a project on X-CUBE-AI and inference performance for the pre-trained neural network on STM32L475VG

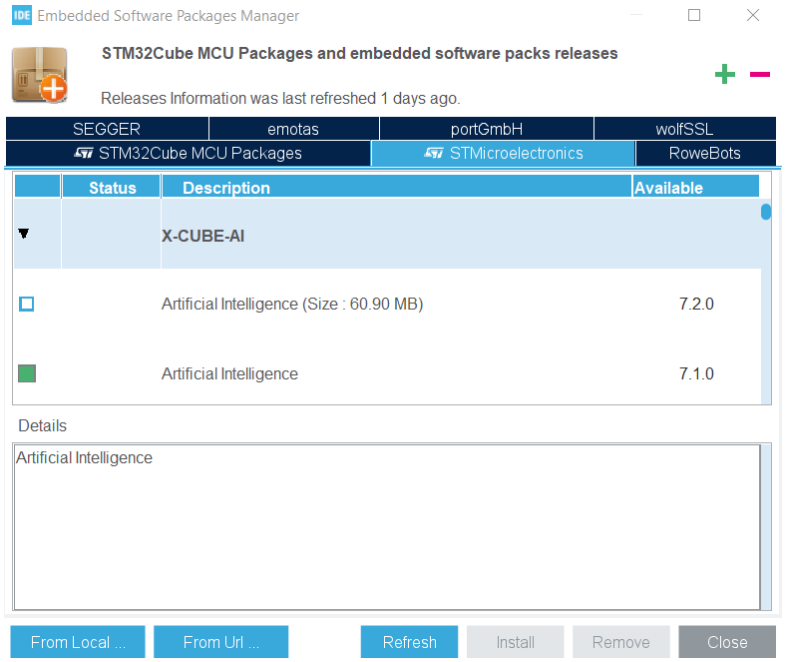
The work flow took inspiration from [44]**, however the difference comes from the configuration for microcontroller and the .c program.**

1. Pre-trained model

The neural network was trained on Google Colab then saved in .h5 format to computer as named *my_model.h5*.

2. X-CUBE-AI installation

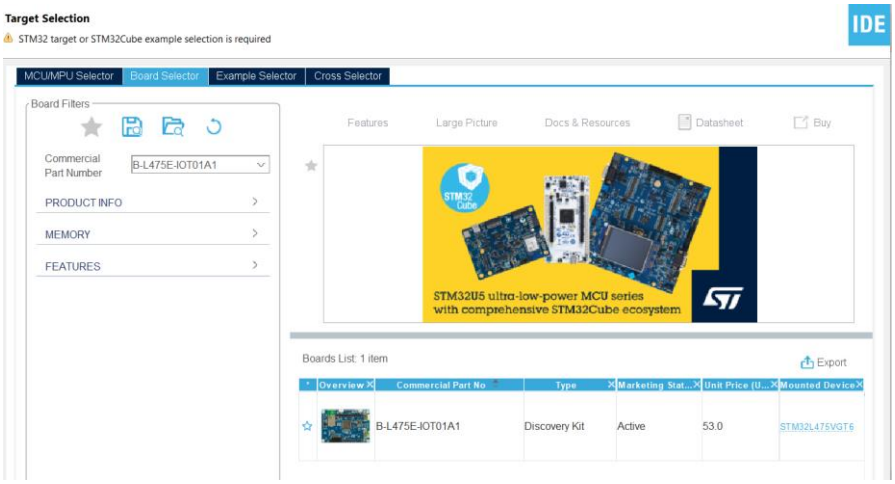
In STM32CubeIDE interface, choose **Help > Manage embedded software packages**. A pop-up window appears, select the **STMicroelectronics** tab. Click the drop-down arrow of **X-CUBE-AI** then select the suitable version. The most recent version will be chosen usually, but this work selected version 7.1.0. Then click **Install.**



This extension will be downloaded and installed automatically after click, just accept the license agreement then close the pop-up window.

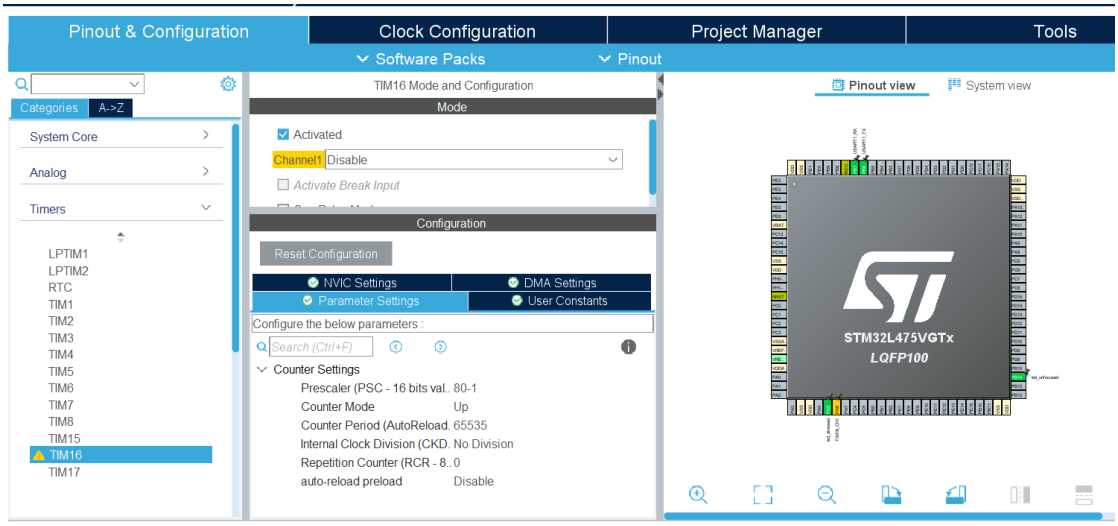
3. Configuration

To start the new project in STM32CubeIDE, select **File > New > STM32 project**. The *Target Selection* window will be open, in **Board Selector,** find B-L475E-IOT01A1.

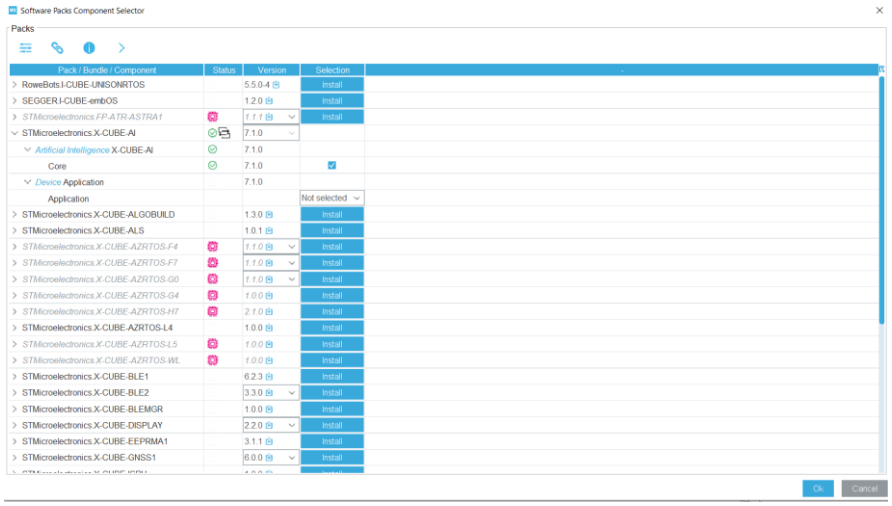


Click **Next**, put the project name then click **Finish**. Choose **Yes** if the system asks to initialize peripheral as default. However, this work only configures the essential pins. Particularly, pins PB6 and PB7 were set for USART1 transmit and receive. 2 pins PB14 and PA5 are connected with LEDs also selected. In **Categories** tab, choosing **Timers** with **TIM16**, **USART1** in **Connectivity,** and select **CRC** (cyclic redundancy check) in **Computing** for AI applications.

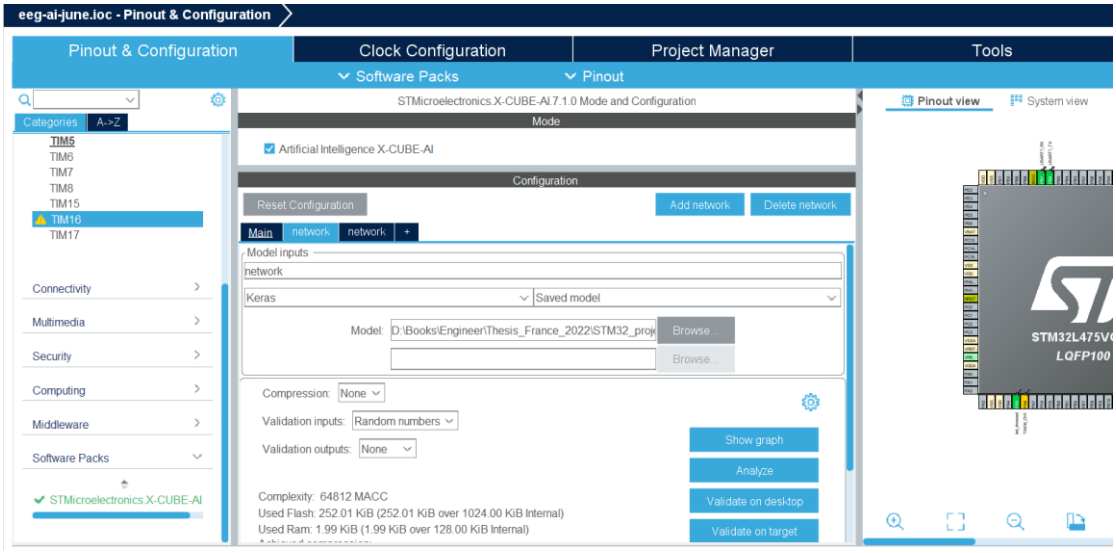
The values in timer were set: Prescaler = 80 – 1 = 79 (for 80 MHz system clock) and Counter Period = maximum value of 16-bit timer = 65535.



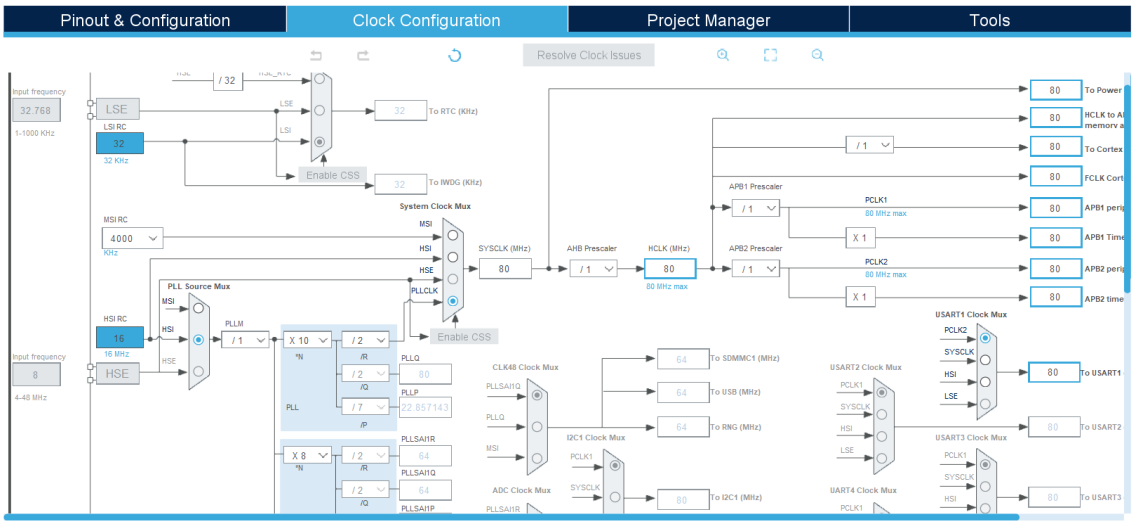
In the tab **Software Packs** of **Pinout & Configuration**, click down arrow and choose **Select Components**. In option **STMicroelectronics.X-CUBE-AI** ensure that all components are activated. For **Device Application**, choose *Not selected*, if choose other options, they are default modes of STM32CubeIDE, you can not modify the code in programming.



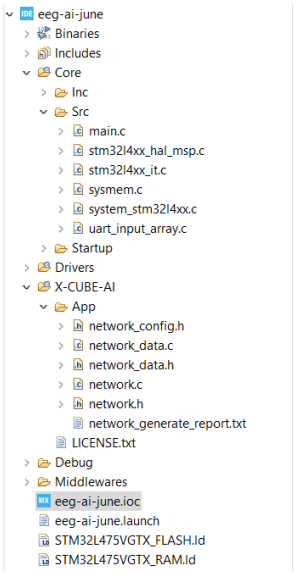
Click **STMicroelectronics.X-CUBE-AI** in **Software Packs** of **Categories** tab, click **Add Network**. Select type of model is Keras, give model a name as *network*. Then scroll down, click **Analyze** to see overview of neural network model. The complexity, used Flash and used RAM are also statistical and display in the interface.



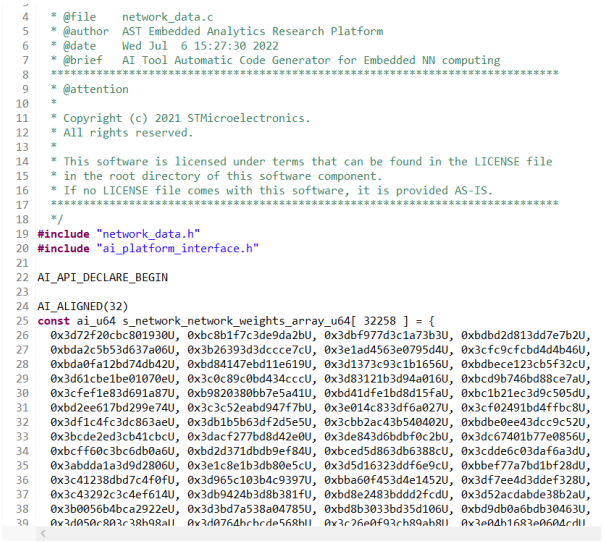
Next step is modifying parameters in tab **Clock Configuration**.In *PLL Source Mux* block, switch the input to the high-speed internal clock as **HIS**. The clock speed was set at 80 for label **HCLKB** then press “enter”, the CubeMX will calculated all the relevant parameters automatically to build an 80 MHz system clock.



Click **File > Save** then **Yes** if asked to generate code.



The project tree will be display. The network was converted into C-array by X-CUBE-AI.



```
  * @file    network_data.c
  * @author  AST Embedded Analytics Research Platform
  * @date    Wed Jul  6 15:27:30 2022
  * @brief   AI Tool Automatic Code Generator for Embedded NN computing
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2021 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  ******************************************************************************
  */
#include "network_data.h"
#include "ai_platform_interface.h"

AI_API_DECLARE_BEGIN

AI_ALIGNED(32)
const ai_u64 s_network_network_weights_array_u64[ 32258 ] = {
  0x3d72f20cbc801930U, 0xbc8b1f7c3de9da2bU, 0x3dbf977d3c1a73b3U, 0xbdbd2d813dd7e7b2U,
  0xbda2c5b53d637a06U, 0x3b26393d3dccce7cU, 0x3e1ad4563e0795d4U, 0x3cfc9cfcbd4d4b46U,
  0xbda0fa12bd74db42U, 0xbd84147ebd11e619U, 0x3d1373c93c1b1656U, 0xbdbece123cb5f32cU,
  0x3d61cbe1be01070eU, 0x3c0c89c0bd434cccU, 0x3d83121b3d94a016U, 0xbcd9b746bd88ce7aU,
  0x3cfef1e83d691a87U, 0xb9820380bb7e5a41U, 0xbd41dfe1bd8d15faU, 0xbc1b21ec3d9c505dU,
  0xbd2ee617bd299e74U, 0x3c3c52eabd947f7bU, 0x3e014c833df6a027U, 0x3cf02491bd4ffbc8U,
  0x3df1c4fc3dc863aeU, 0x3db1b5b63df2d5e5U, 0x3cbb2ac43b540402U, 0xbdbe0ee43dcc9c52U,
  0x3bcde2ed3cb41cbcU, 0x3dacf277bd8d42e0U, 0x3de843d6bdbf0c2bU, 0x3dc67401b77e0856U,
  0xbcff60c3bc6db0a6U, 0xbd2d371dbdb9ef84U, 0xbced5d863db6388cU, 0x3cdde6c03daf6a3dU,
  0x3abdda1a3d9d2806U, 0x3e1c8e1b3db80e5cU, 0x3d5d16323ddf6e9cU, 0xbbef77a7bd1bf28dU,
  0x3c41238dbd7c4f0fU, 0x3d965c103b4c9397U, 0xbba60f453d4e1452U, 0x3df7ee4d3ddef328U,
  0x3c43292c3c4ef614U, 0x3db9424b3d8b381fU, 0xbd8e2483bddd2fcdU, 0x3d52acdabde38b2aU,
  0x3b0056b4bca2922eU, 0x3d3bd7a538a04785U, 0xbd8b3033bd35d106U, 0xbd9db0a6bdb30463U,
```

Any necessary modification or to execute program will be implemented in *main.c* function.

At notion, if there is syntax *printf* in the program, it will appear the error because *printf* and variants do not support floating point values by default in STM32CubeIDE. Thus, click **Project > Properties > C/C++ Build > Settings > Tool Settings > MCU GCC Complier > Miscellaneous**. In the *Other flags* put the command: -u_printf_float

Executing this step for both **Debug** and **Release** configurations. Then save the code.

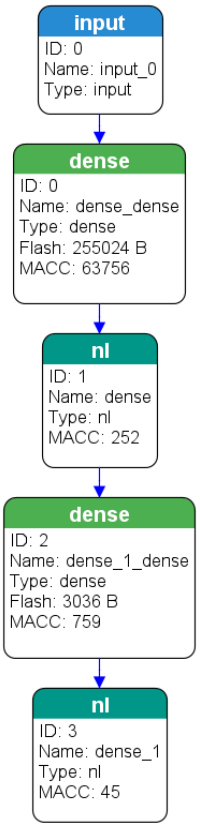
The final step is connecting board with computer through cable (communicate with ST-Link first). Then click **Project > Build Project**.

The details of code were uploaded on Github:

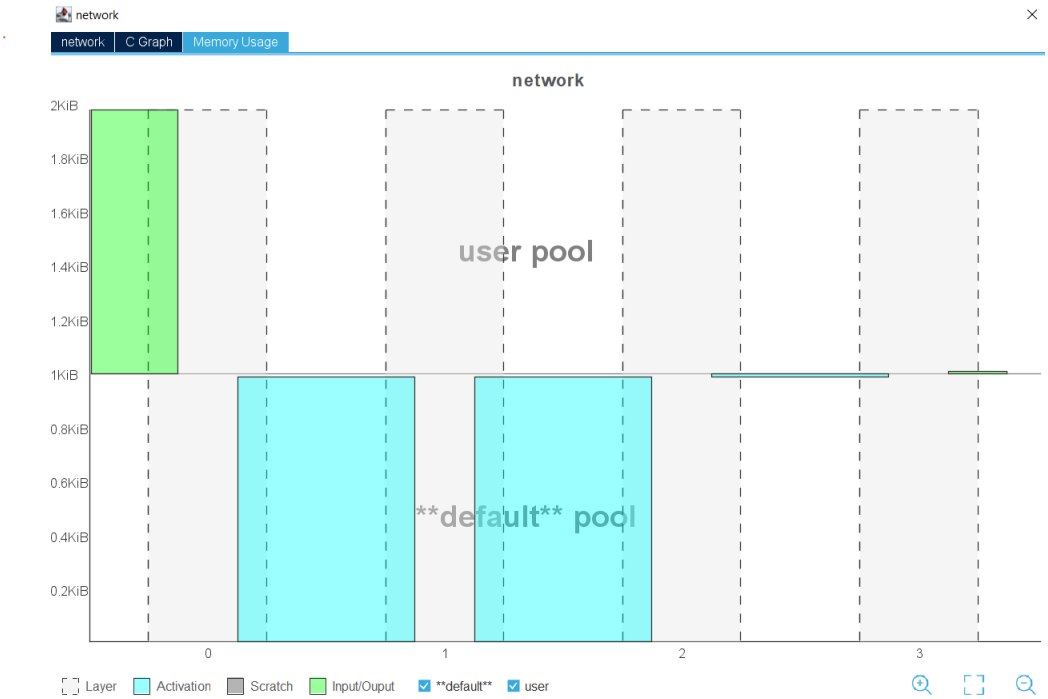
https://github.com/thuypt1402/Intership_code

However, due to privacy of project, this repository is set private status.

B. Visualization of model



C. Memory Usage in Graph



REFERENCES

[1] L. Chen, Y. Zhao, J. Zhang, and J. Zou, “Automatic detection of alertness/drowsiness from physiological signals using wavelet-based nonlinear features and machine learning,” *Expert Systems with Applications*, vol. 42, no. 21, pp. 7344–7355, Nov. 2015, doi: 10.1016/j.eswa.2015.05.028.

[2] X. Bi and H. Wang, “Early Alzheimer’s disease diagnosis based on EEG spectral images using deep learning,” *Neural Networks*, vol. 114, pp. 119–135, Jun. 2019, doi: 10.1016/j.neunet.2019.02.005.

[3] Y. Li, X. Li, M. Ratcliffe, L. Liu, Y. Qi, and Q. Liu, “A real-time EEG-based BCI system for attention recognition in ubiquitous environment,” in *Proceedings of 2011 international workshop on Ubiquitous affective awareness and intelligent interaction - UAAII ’11*, Beijing, China, 2011, p. 33. doi: 10.1145/2030092.2030099.

[4] J. K. Nuamah and Y. Seong, “Support vector machine (SVM) classification of cognitive tasks based on electroencephalography (EEG) engagement index,” *Brain-Computer Interfaces*, vol. 5, no. 1, pp. 1–12, Jan. 2018, doi: 10.1080/2326263X.2017.1338012.

[5] Ç. İ. Acı, M. Kaya, and Y. Mishchenko, “Distinguishing mental attention states of humans via an EEG-based passive BCI using machine learning methods,” *Expert Systems with Applications*, vol. 134, pp. 153–166, Nov. 2019, doi: 10.1016/j.eswa.2019.05.057.

[6] “IoT devices installed base worldwide 2015-2025,” *Statista*. https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/ (accessed Sep. 11, 2022).

[7] F. Sakr, F. Bellotti, R. Berta, and A. De Gloria, “Machine Learning on Mainstream Microcontrollers,” *Sensors*, vol. 20, no. 9, p. 2638, May 2020, doi: 10.3390/s20092638.

[8] G. Cerutti, R. Prasad, and E. Farella, “Convolutional Neural Network on Embedded Platform for People Presence Detection in Low Resolution Thermal Images,” in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Brighton, United Kingdom, May 2019, pp. 7610–7614. doi: 10.1109/ICASSP.2019.8682998.

[9] Y. Zhang, S. Bi, M. Dong, and Y. Liu, “The Implementation of CNN-Based Object Detector on ARM Embedded Platforms,” in *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*, Athens, Aug. 2018, pp. 379–382. doi: 10.1109/DASC/PiCom/DataCom/CyberSciTec.2018.00074.

[10] M. Magno, M. Pritz, P. Mayer, and L. Benini, “DeepEmote: Towards multi-layer neural networks in a low power wearable multi-sensors bracelet,” in *2017 7th IEEE International Workshop on Advances in Sensors and Interfaces (IWASI)*, Vieste, Italy, Jun. 2017, pp. 32–37. doi: 10.1109/IWASI.2017.7974208.

[11] “Supplemental material TinyML,” *TinyML Book*, Dec. 12, 2019. https://tinymlbook.com/supplemental/ (accessed Sep. 11, 2022).

[12] “Normal EEG Waveforms: Overview, Frequency, Morphology,” Mar. 2022, Accessed: Aug. 26, 2022. [Online]. Available: https://emedicine.medscape.com/article/1139332-overview

[13] S. Sanei and J. A. Chambers, *EEG signal processing*, Reprinted with corrections. Chichester: John Wiley & Sons, Ltd, 2009.

[14] A. Bashashati, M. Fatourechi, R. K. Ward, and G. E. Birch, “A survey of signal processing algorithms in brain–computer interfaces based on electrical brain signals,” *J. Neural Eng.*, vol. 4, no. 2, pp. R32–R57, Jun. 2007, doi: 10.1088/1741-2560/4/2/R03.

[15] F. Lotte, “A Tutorial on EEG Signal-processing Techniques for Mental-state Recognition in Brain–Computer Interfaces,” in *Guide to Brain-Computer Music Interfacing*, E. R. Miranda and J. Castet, Eds. London: Springer London, 2014, pp. 133–161. doi: 10.1007/978-1-4471-6584-2_7.

[16] J. Brownlee, “4 Types of Classification Tasks in Machine Learning,” *Machine Learning Mastery*, Apr. 07, 2020. https://machinelearningmastery.com/types-of-classification-in-machine-learning/

[17] “Confusion matrix,” *Wikipedia*. Aug. 31, 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Confusion_matrix&oldid=1107701525

[18] J. Brownlee, “8 Tactics to Combat Imbalanced Classes in Your Machine Learning Dataset,” *Machine Learning Mastery*, Aug. 18, 2015. https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/

[19] “Support Vector Machines: A Simple Explanation,” *KDnuggets*. https://www.kdnuggets.com/support-vector-machines-a-simple-explanation.html/

[20] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: from theory to algorithms*. New York, NY, USA: Cambridge University Press, 2014.

[21] S. Dobilas, “XGBoost: Extreme Gradient Boosting — How to Improve on Regular Gradient Boosting?,” *Medium*, Feb. 05, 2022. https://towardsdatascience.com/xgboost-extreme-gradient-boosting-how-to-improve-on-regular-gradient-boosting-5c6acf66c70a

[22] J. H. Friedman, “Greedy function approximation: A gradient boosting machine.,” *Ann. Statist.*, vol. 29, no. 5, Oct. 2001, doi: 10.1214/aos/1013203451.
[23] “Artificial Intelligence - Neural Networks.” https://www.tutorialspoint.com/artificial_intelligence/artificial_intelligence_neural_networks.htm
[24] “Deep learning,” *Wikipedia*. Sep. 02, 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Deep_learning&oldid=1108086973
[25] “Perceptron in Machine Learning - Javatpoint,” *www.javatpoint.com*. https://www.javatpoint.com/perceptron-in-machine-learning (accessed Sep. 05, 2022).
[26] V. Nair and G. E. Hinton, “Rectified Linear Units Improve Restricted Boltzmann Machines,” p. 8.
[27] J. Chen and X. Ran, “Deep Learning With Edge Computing: A Review,” *Proc. IEEE*, vol. 107, no. 8, pp. 1655–1674, Aug. 2019, doi: 10.1109/JPROC.2019.2921977.
[28] M. Merenda, C. Porcaro, and D. Iero, “Edge Machine Learning for AI-Enabled IoT Devices: A Review,” *Sensors*, vol. 20, no. 9, p. 2533, Apr. 2020, doi: 10.3390/s20092533.
[29] “TensorFlow Lite | ML for Mobile and Edge Devices,” *TensorFlow*. https://www.tensorflow.org/lite
[30] “ARM Developer Suite AXD and armsd Debuggers Guide.” https://developer.arm.com/documentation/dui0066/d
[31] “Q (number format),” *Wikipedia*. Sep. 03, 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Q_(number_format)&oldid=1108245902
[32] V. Sze, Y.-H. Chen, J. Emer, A. Suleiman, and Z. Zhang, “Hardware for machine learning: Challenges and opportunities,” in *2018 IEEE Custom Integrated Circuits Conference (CICC)*, San Diego, CA, USA, Apr. 2018, pp. 1–8. doi: 10.1109/CICC.2018.8357072.
[33] “X-CUBE-AI - AI expansion pack for STM32CubeMX - STMicroelectronics.” https://www.st.com/en/embedded-software/x-cube-ai.html
[34] “stm32l475vg.pdf.” Accessed: Sep. 06, 2022. [Online]. Available: https://www.st.com/resource/en/datasheet/stm32l475vg.pdf
[35] “B-L475E-IOT01A - STM32L4 Discovery kit IoT node, low-power wireless, BLE, NFC, SubGHz, Wi-Fi - STMicroelectronics.” https://www.st.com/en/evaluation-tools/b-l475e-iot01a.html
[36] “EEG data for Mental Attention State Detection.” https://www.kaggle.com/datasets/inancigdem/eeg-data-for-mental-attention-state-detection
[37] “Short-time Fourier transform,” *Wikipedia*. Aug. 16, 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Short-time_Fourier_transform&oldid=1104785770
[38] “Blackman window - MATLAB blackman.” https://www.mathworks.com/help/signal/ref/blackman.html
[39] “TensorFlow,” *TensorFlow*. https://www.tensorflow.org/
[40] “Module: tf.keras | TensorFlow v2.9.1,” *TensorFlow*. https://www.tensorflow.org/api_docs/python/tf/keras (accessed Sep. 07, 2022).
[41] “Model conversion overview | TensorFlow Lite,” *TensorFlow*. https://www.tensorflow.org/lite/models/convert (accessed Sep. 08, 2022).
[42] A. Vaswani *et al.*, “Attention is All you Need,” in *Advances in Neural Information Processing Systems*, 2017, vol. 30. Accessed: Sep. 08, 2022. [Online]. Available: https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html
[43] A. Dosovitskiy *et al.*, “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale,” 2020, doi: 10.48550/ARXIV.2010.11929.
[44] “TinyML: Getting Started with STM32 X-CUBE-AI,” *Digi-Key Electronics*. https://www.digikey.com/en/maker/projects/f94e1c8bfc1e4b6291d0f672d780d2c0 (accessed Sep. 09, 2022).