NANYANG TECHNOLOGICAL UNIVERSITY

Federated Learning in Smart Healthcare Applications (ECG Anomaly Detection)

Rani Vigneshkumar Rajendrakumar

School of Computer Science and Engineering

2022

NANYANG TECHNOLOGICAL UNIVERSITY

MSAI Master Project MSAI/21/003

Adaptive Attention Layer Freezing in Federated Learning

Submitted by: Rani Vigneshkumar Rajendrakumar
under the supervision of
Professor Dusit Niyato

School of Computer Science and Engineering

2022

Contents

Abstract

Can there always be a generalized model in the Federated Learning context? Is there a way to optimize model training in a Federated environment? In the line of answering these questions, this ambitious project hypothesizes that controlling the freezing of attention layers independently can help to enhance the training process and learn new data by shifting the focus of existing features in the domain of ECG. Through this process, a generalized model can be formed by just controlling the learning of the attention layers to learn new data which is advantageous in the context of Federated Learning. A pre-trained model (pre-trained on MIT-BIH data) with SE-Nets as attention networks are first obtained which is then trained on two other datasets namely PTB and ECG5000. Adaptive Attention Layer Freezing (AALF) is implemented and applied to optimally train the attention layers by monitoring the mean F1 score and customized coverage metric for each layer. The experiments conducted in this project have not produced sufficient favourable results, leading to the rejection of the hypothesis. The results, however, provide insightful suggestions on the capability of shifting attention layers' focus and on other areas of the FL training process in TFF.

Keywords: Dissertation, Federated Learning, TensorFlow Federated, ECG, Adaptive Attention Layer Freezing

Acknowledgement

I would like to acknowledge and convey my warmest thanks to my Supervisor, Professor Dusit Niyato, for his patience and guidance throughout the project. His advice and constructive comments have helped to smooth the project journey.

Acronyms

AALF - Adaptive Attention Layer Freezing
AI - Artificial Intelligence
APF - Adaptive Parameter Freezing
CCE - categorical cross-entropy
CNN - Convolution Neural Network
DL - Deep Learning
ECG - Electrocardiogram
EMA - Exponential Moving Average
FedAvg - Federated Averaging
FL - Federated Learning
GANs - Generative Adversarial Networks
GPUs - Graphical Processing Units
IID - Independent and identically distributed
IoT - Internet of Things
MAE - Mean absolute error
ML - Machine learning
MSE - Mean squared error
SE Net - Squeezing and Exciting Network
SGD - Stochastic Gradient Descent
TF 2.0 - TensorFlow 2.0
TFF - TensorFlow Federated

List of Figures

Chapter 1

Introduction

1.1 Background

The scope of the project lies within the aspects of Federated Learning (FL) and anomaly detection for ECG. Hence this section has been categorized accordingly to provide background knowledge for the respective aspects.

1.1.1 Anomaly Detection in ECG

Anomaly detection refers to the discovery or identification of any deviation from the normal characteristics of the data. Despite the early stages when traditional ML methodologies have been used, anomaly detection still poses a great challenge and is still under active research. With the advancement of deep learning techniques, deep anomaly detection techniques are ubiquitous in recent research papers. Anomaly detection in the healthcare or medical domain is a very crucial AI application that can help patients to elevate their standard of living.

1.1.1.1 ECG

Electrocardiogram (ECG) in the healthcare domain refers to the heart measurements comprising of rhythm and rate[1]. In terms of data, it can be considered as a 1 dimensional signal over time for each measurement of the heart. ECG can be used to understand how the heart has been functioning by attaching electrodes onto the body[1]. A normal ECG signal contains cycles of recurring waves that follow a certain pattern.

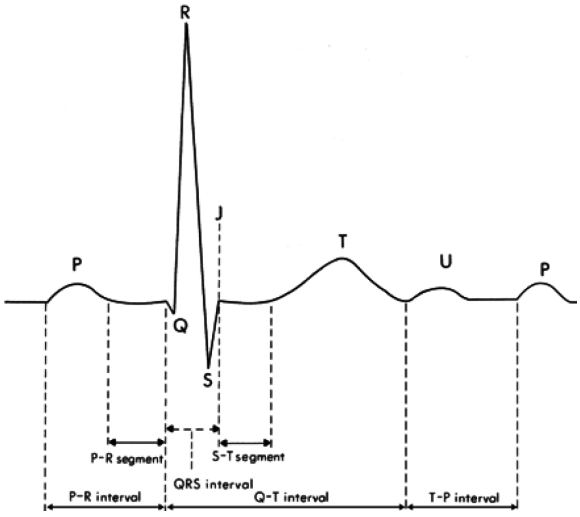


Figure 1.1: Normal ECG wave pattern [2].

The Figure above illustrates a normal ECG wave. The main segments of an ECG wave are the P wave (atrial depolarisation), QRS complex (ventricular depolarization) and T wave (ventricular re-polarisation) as labelled in the Figure above[2]. Based on the diagram, a complete wave cycle is a R-R segment[2]. Each segment of the wave corresponds to the functions of the respective parts of the heart. Hence, any abnormal ECG signal refers to any anomalies of these segments in a wave or irregularities of the ECG wave patterns over time[2]. There are various types of abnormal ECG signals under anomaly detection such as heart block, irregular rhythms or even congestive heart failure.

1.1.1.2 Characteristics of Anomaly Detection

Anomalies contain certain characteristics that should be taken note of and here are some that are associated with ECG signal data.

1. ***Rare*** - Anomalies occur in small numbers and thus it is essential to maximise the learning capability of the models with limited data. This usually results in class imbalance and has to be mitigated with sampling or data augmentation. Rarity also means that more instances of abnormal cases are required for data collection and these cannot be recreated or easily accessible specifically for sensitive data such as the ECG indicators of patients[3, 4].

2. ***Pattern Variety*** - Anomalies have different variants and thus it is essential for models to be able to detect known patterns from the variants. It is not always possible for a model to see known anomalies during application as it is highly dependent on the availability of all known anomaly patterns in the training dataset. Due to the vast amount of features needed to learn for every anomaly, it is common to observe a low recall rate when detecting anomalies[3]. In another perspective, detecting deviating patterns are not straightforward in a higher dimension where these abnormal patterns can be hidden and/or it is expensive to account for patterns in every feature space[3, 4, 6].

3. ***Anomaly Types*** - Anomalies can be categorized based on their nature.

   (a) ***Point*** - Point anomalies are very common and they include abnormal or missing data points given an instance. Taking a person's ECG time series data as an example, abnormal heartbeats are considered point anomalies[3].

(b) ***Contextual*** - Anomalies can be based on scenarios. Reading ECG heartbeat of 140 beats per minute at 2 am is an example of contextual anomalies. 140 beats per minute are normal when exercising but exercising at 2 am seems peculiar and thus is considered an anomaly[3]. Unlike the point anomalies, contextual anomalies may require additional information such as exact timestamp, temporal or spatial information[3].

(c) ***Collective*** - Collective anomalies refer to anomalies observed when grouping data instances from different sources. Having all ECG data instances from 10 people with 140 beats per minute at the same time is peculiar and thus classified as an anomaly as everyone exercising at the same time is indeed rare[3].

4. ***Noisy*** - Normal and abnormal signals are noisy and thus anomalies may not be easily detected. This requires models to be noise resilient and learn to differentiate noise and anomalies[3, 4]. This is challenging when no proper de-noising processing of data is done before being fed to the models.

5. ***Not Easily Explainable*** - It is favourable for the model to understand the reason for abnormal signals. It can be explained via pattern recognition of the abnormal cases mentioned above. With high rarity and noise addition, the signals become complex for the model to identify the cause[3, 4, 5]. Other factors can complicate the nature of signals. Moreover, pattern recognition may not always provide explicit reasons. Hence, the need for experts for manual inspection has been highlighted[4].

1.1.1.3 Deep Learning Approaches

There are numerous strategies for utilizing deep learning approaches for anomaly detection. The authors of [3] have categorised the approaches into 3 major groups listed below.

1. ***Feature Extraction*** - Similar to any other application, this technique leverages the existing pre-trained models to extract essential components of the data ranging from low level to high-level features via transfer learning. Some pre-trained models help to reduce data from a higher dimension to a lower dimension for reduced computation complexity[3, 6]. The extracted features can then be classified with traditional methods such as regressions or SVM[3] or even XGBoost trees[6]. The advantages of such techniques are that pre-trained models are readily available and no training is required which speeds up the anomaly detection process. However, the model weights may not be enough to extract relevant information. At such times, fine-tuning is required for the information extraction to be relevant to the data[3, 4].

2. ***End-to-End Score Learning*** - The techniques under this category have the data as direct inputs and types of anomalies as outputs from the models. In other words, the models learn to differentiate and classify anomalies using objective functions such as softmax/MSE/MAE functions[3, 4]. The advantage is that the model can multi-task by identifying and classifying the anomaly features. This multi-task can also limit the learning process by not identifying the appropriate features. Thus, the learning process is highly dependent on the quality of input data[3, 4].

3. ***Normality Learning*** - DL techniques are used to learn the features of data that are not considered as an anomaly. The ideology is that anomalies are rare and thus easier to obtain non-abnormal data for training[3]. DL techniques can range from sequential prediction models such as autoregressive models to generative models such as AutoEncoders or GANs. For generative models, the objective function involves mean absolute error(MAE) or mean squared error(MSE)[3, 5]. The advantages and disadvantages vary according to

the type of models used. For sequential models, advantages include learning and retaining time-sensitive features (temporal/spatial) while one of the disadvantages is the expensive computation required when dealing with high dimensional data[4]. Furthermore, it may not be explicitly possible to differentiate anomalies based on their distances away from normal features. Generative models such as GANs are very powerful as they can learn the essential features for data recreation if trained appropriately. At the same time, the training can be very difficult due to reasons such as model collapse and many other model convergence failures[3, 4].

This project focuses on the End-to-End Score Learning DL technique in the Federated Learning environment where the DL models learn to predict ECG anomaly classes directly.

1.1.2 Federated Learning

1.1.2.1 Overview

Machine learning (ML) has evolved tremendously over the years specifically due to the advancement in Deep Learning (DL) implementation with increased computing capacity provided by the Graphical Processing Units (GPUs). Originally, ML development was predominantly performed with local computing where the model is trained and evaluated locally. With limited computing power in the local environment, a better platform was required which led to Cloud Computing[7]. Cloud Computing allows one to train and evaluate the model in the cloud environment where GPUs or any other form of computing capacity are provided by cloud services such as Amazon Web Services (AWS). This means that anyone without local computing capacity is still capable of training ML models for large datasets sent to the cloud for centralized learning[7].

Despite the advantages mentioned, cloud computing ensures no guarantee of the privacy of data. Certain data from healthcare or finance are highly sensitive. To deal with privacy issues, the computing paradigm has been slowly shifting from cloud computing to edge computing (or hybrid of cloud and edge) where computing is brought closer to the data sources[8, 9]. One of the major focuses in areas of edge computing in recent years in ML is Federated Learning (FL) where the model is sent to the edge devices where data resides for training in private[8, 9]. This decentralised framework is an idea for the model to learn from sensitive data at edge devices without having the data exposed to the public[8, 9, 10]. This shift in focus is partly also due to the rapid development of the Internet of Things (IoT) for applications in industries such as manufacturing and medical centres[9, 10, 11].

In summary, the objective of FL is to collaboratively learn from distributed and heterogeneous data on heterogeneous devices with limited compute capacity.

With this decentralised ML framework, it can be advantageous to tackle data privacy and security issues[8, 9, 10] which would allow models to learn from sensitive data without compromising privacy and security. Furthermore, training at the devices translates to lower latency due to reduced data transmission which in turn relates to faster AI analysis responses in real time[11].

1.1.2.2 Lifecycle

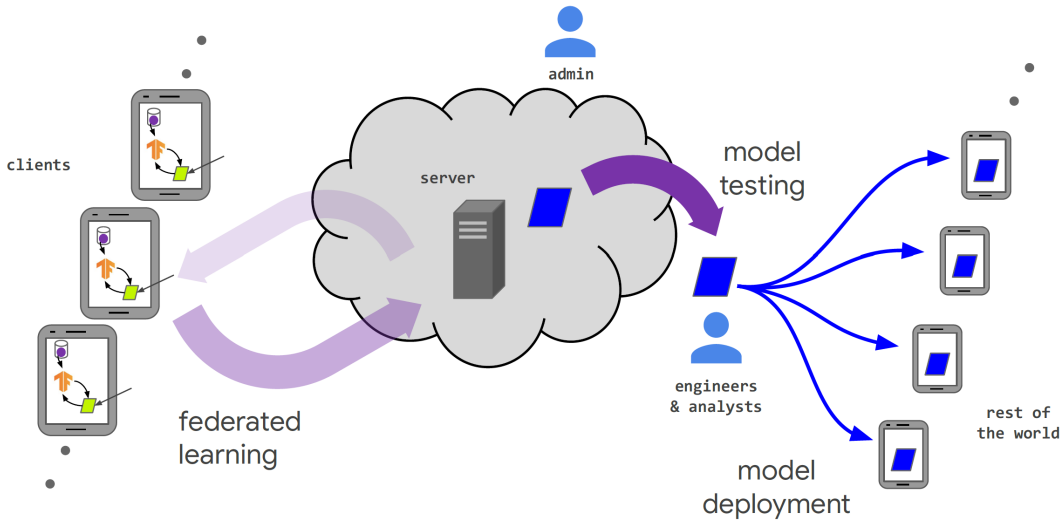


Figure 1.2: Lifecycle of model in FL system [12].

The lifecycle of a global model in a central server in FL comprises a few steps denoted below based on [8, 9, 12]. The steps are generalised for a server and few clients.

1. ***Type of problem*** - It is about the identification of a problem to be solved via FL environment. This project is specific to ECG anomaly detection.

2. ***Client set-ups*** - Selected clients should be able to have the necessary technology platforms, software and libraries for the model training.

3. ***Simulation*** - A simulation environment for training and testing is required to ensure learning is possible for the model without destroying its weights. The capacity of the model can also be examined in the simulation environment, thereby reducing operational costs.

4. ***FL Training and Evaluation*** - The model undergoes FL training and evaluation which involves optimizing algorithms, search space and exchange of model weights between the server and clients on various types of datasets. The training process is elaborated in the next subsection.

5. ***Model Deployment*** - Once tested successfully, the model is deployed to many devices progressively from the central server while monitoring the live performance.

These steps have been summarized visually in the Figure 1.2 above.

1.1.2.3 Training Process

The FL training process constitutes 4 sequential steps as listed below based on [9, 12].

1. ***Server Broadcast*** - The server sends the model weights/parameters to the selected clients.
2. ***Client Computation*** - The clients update their local model by training on local data using backpropagation with optimizers such as Adam and SGD. The clients then send the updated model parameters to the server.
3. ***Aggregation*** - Aggregation can be referred to as the merging of model parameters appropriately and efficiently. Upon receiving the updated model parameters from respective clients, the server performs aggregation techniques such as Federated Averaging (FedAvg) which calculates the mean of all updated model parameters[9, 12]. FedAvg is the most common and naive technique due to its simplicity. Other techniques are further discussed under the sub-section titled "Open problems in FL".
4. ***Model Update*** - With appropriate or desired aggregation technique, the model in server updates its parameters for future communication rounds if training is in progress or for evaluation if training has ended.

Going through the above 4 steps sequentially once is equivalent to 1 complete communication round. These steps are repeated for multiple communication rounds with different clients on data of varying sizes.

It is also essential to note that only model parameters are exchanged between the clients and server. In a simulation environment, there is an additional transmission of datasets from the server to the clients which is absent in reality.

1.1.2.4 FL Environments

There are 2 types of FL environments - cross-silo and cross-device. Their properties have been briefly outlined.

A cross-silo environment consists of clients that are organizations or data centres. There are in a small number (typically in 100s) and thus can be considered as low scaled. The computing and storage capacities are high and there is network stability for reliable transmission of model parameters. All clients can participate in all communication rounds of FL. In addition, each client can be identified and access can be specific to the client[9, 11, 12].

A cross-device environment consists of clients that are low-powered devices such as handphones. The number of clients usually tend toward a power of 10s. The clients have limited computing and storage capacities. The network can be unreliable such as WiFi and thus transmission may not be reliable. Furthermore, the number of participating clients can vary in each communication round and they cannot be distinguished from one another[9, 11, 12].

1.1.2.5 Open problems in FL

There are many open problems in FL that require attention in the research area before proceeding with deployment. The section has highlighted 3 of the active issues in FL as listed below.

1. ***Aggregation At Server*** - The aggregation of model parameters from the clients at the server poses a complicated problem. FedAvg takes the mean (can be weighted mean) of all the model parameters[7, 8, 9, 12] for the stochastic update in the server. However, such an algorithm is suitable for independent and identically distributed (IID) datasets but may not be suitable at all times. The type of aggregation algorithm highly depends on the non-independent and non identically distributed (non-IID) datasets trained locally in the clients.

   There are many types of non-IID datasets listed below collectively based on [8, 11, 12]. In the following scenarios, let's take P(y) as the marginal probability of targets,y and P(x) as the marginal probability of features, x. Let's take P(y|x) as conditional probabilities of target labels given the features and P(x|y) as conditional probabilities of features given target labels.

   (a) ***Covariate skew*** - Covariate skew refers to the change in the marginal distributions of features P(x) in the clients. Noise variations in the features are examples of such cases where the features are the same but noise modifies the marginal distribution of the same features. The underlying pattern of the de-noised features and the same noisy features are similar[8, 12].

   (b) ***Target distribution*** - Datasets among clients can be imbalanced in terms of target distribution P(y) and thus model parameters may be biased to the largest target size and thus not appropriate for model updates[8, 9, 12].

   (c) ***Concept shift*** - Concept shift refers to the change in the conditional probabilities of target labels given the features P(y|x) or/and conditional probabilities of features given target labels P(x|y). This translates to client datasets with the same targets but with differing features and client datasets with the same features but differing targets respectively[12].

   (d) ***Quantity skew*** - Different clients have different sizes of data and thus equal importance cannot be provided to all clients[8, 9, 12].

   The types of non-IID data listed above pose a problem to the widely used FedAvg algorithm and thus better algorithms are required. A countless number of aggregation algorithms have been proposed in the research area. SCAFFOLD is an already proposed algorithm that mainly addresses the issue of heterogeneous data shifts in clients by reducing client variance[14].

2. ***Efficiency*** - This category contains various types of subcategories namely resource efficiency, communication efficiency and optimizing training process. Some of the subcategories have been listed below.

   (a) ***Communication Cost*** - Communication cost refers to the cost of transmitting the information. Here, it is mostly about the transfer of trained models weights from the clients to the servers. Having reduced transmission costs speeds up the communication between clients and servers and is thus less expensive. This can be a bottleneck when communication networks such as WiFi are not reliable. The following are some techniques to mitigate such issues[8, 9, 11, 12].

i. ***Gradient Compression*** - Gradient compression refers to reducing the gradient values transferred. One way is to transfer the model weight differences after training from the clients to the servers which reduce the risk of transmitting large values[11, 12].

ii. ***Model Compression*** - Model compression refers to the amount of data transferred regarding the model weights. Thus, it is essential to identify scenarios where only certain model layers require local training so that only these trained weights can be sent to the server[8, 11, 12].

iii. ***Model Sparsity*** - There are scenarios where only a few of the model parameters are updated in each model layer. This translates to sparse model weight differences as it mostly contains zero values. Only the non-zero values can be sent to the server[12].

(b) ***Training Optimization*** - Optimizing the training process reduces training time in clients and appropriate procedure reduces the trained parameters that are required to be transferred to the server. These help to minimize communication issues mentioned in the previous point. Any techniques that reduce computation capacity can help to reduce communication costs as the time consumed for local training is lower[8, 11, 12]. Techniques include model parameter quantization or even improving the model layer freezing process. More details are provided in the next chapter on Literature Review.

(c) ***Neural Network and Hyperparameter Search*** - The search for the right neural network architecture and the right combination of hyperparameters can be time-consuming when there are numerous clients in FL. It is possible to apply existing search tools in the non-FL environment such as random or Gaussian search FL environment. However, high time consumption can be complicated if the communication is not reliable.

3. ***Privacy Preservation*** : Privacy preservation is the core motive of having FL framework. Privacy can be compromised as long as the intruders can access the client data directly or able to replicate the data by finding the patterns. These are possible when the intruders can obtain the model parameters or attack the server itself where all the aggregations from the clients occur[8, 9, 12]. Reproducing similar client data with model parameters is easily possible with 2 sets of neural networks known as Generative Adversarial Networks (GANs). With the fixed stolen model parameters, GANs can learn to generate fake data that is very similar to client data by minimising the loss produced by the predictions from fake data This is an indirect privacy leakage[8, 11].

   There are existing/proposed techniques to avoid such issues. These techniques are mostly classified into 2 following categories.

   (a) ***Noise Addition*** - Augmenting the trained parameters with noise is one of the existing successful ways. This can be done via Differential Privacy (DP) which adds Gaussian distributed noise or Laplace distributed noise to the parameters at the client side before aggregation at server[8, 11, 12].

   (b) ***Encryption*** - Encryption based methods are usually applied for the communication between the server and clients to prevent any attacks on the communication lines where the model parameters are exchanged. One of such techniques includes the use of secret key pairs between the server and the client[8, 11, 12].

There are other types of open problems that are not outlined here such as scheduling or even security issues that dwell with data/model poisoning and many more. The high number of open

issues with FL calls for attention in the research area. In addition, many techniques address multiple open issues. FedBoost is one example that aims to handle communication efficiency while preserving privacy, speeding up computations and adapting to the data drift[14].

In this project, the focus is on the area on efficiency, specifically on training optimization as a means to achieve its goals. More on the existing/proposed solutions for efficiency are elaborated in the next chapter under Literature Review.

1.2 Problem Statement

To have a consistent understanding of the project, let's have some pre-defined terms. There are 2 types of model weights namely - weights from attention layers and non-attention layers.

Attention and non-attention layers serve different purposes. Non-attention layers are mandatory layers that learn all possible features from the inputs while attention layers are supplementary to the model (models can exist without them) and provide focus on selected features.

1.2.1 Hypothesis

Hypothesis: Controlling the freezing of attention layers independently can help to enhance the efficiency of FL training and learn new data not seen before, by changing the focus of existing features. This ideally means that we can have a generalized model for FL as it is and just train the attention layers optimally for clients' data.

Successful fulfillment of the hypothesis above reduces training time for clients as the normal weights do not have to be trained if possible. This reduces the number of parameter changes which promotes cost effectiveness with better training optimization. This tackles the issue of efficiency as one of the open problems in FL.

1.2.2 Objective

To test the hypothesis above, the main objective of this project is to design and implement a custom attention layer freezing technique in TFF environment for ECG anomaly classification.

1.3 Motivation

The initial motivation originated from layer freezing techniques to optimize the training process so that only necessary parameters are trained while retaining pre-trained features as much as possible. This can be applied without the FL environment. As an attention network focuses on what is required for that dataset, it also means that there may be features that do not gain much attention or are not required at all. Hence, there is a question of whether changing weights of attention networks alone suits other datasets and aids in learning new features.

The other essential motivation came from the question "is training a single global model even the right goal?" that authors of [12] have posed. FL is implemented for the models to collaboratively learn as much as possible including from private data. At the end of the day, can one model learn everything and can be applied in many instances? Authors from [12] suggests having localized training for specific uses and general training to have a general model or having more models. However, having more models for the same specific domain may end up increasing non-generalized models which contradicts the idea of FL.

The narrowed healthcare topic on ECG domain in this project presents an opportunity to explore into merging the motivations at a small scale. For a given specific domain, is it possible to have the non-attention weights (general weights) with different combinations of attention weights for each specific use case? Thus, this project aims to experiment on a possible solution that can help redefine generalised and local models while optimising FL training process.

1.4 Major contribution of the Dissertation

1. The project leverages on existing works for improvement that integrates 2 papers of interests.

   (a) ***Paper 1*** - Raza et al. "Designing ECG Monitoring Healthcare System with Federated Transfer Learning and Explainable AI" [15] : The project improves on the model architecture from this paper.

   (b) ***Paper 2*** - Chen et al. "Communication-Efficient Federated Learning with Adaptive Parameter Freezing" [16] : The project improves on the existing adaptive parameter freezing (APF) technique from this paper.

2. Based on my research, I believe that this project will be the first to focus on adaptive freezing for attention layers in the model (technique inspired by APF). This technique will be called as Adaptive Attention Layer Freezing (AALF).

3. The project is unique to a large extent as it explores into distinguishing a global model and local model for client use-case with the same model architecture with AALF in the specific medical domain.

4. The project aims to handle anomaly ECG classes in different feature space with AALF not seen previously when pre-trained while being cost effective.

5. Previous works in FL have been accomplished with frameworks such as FATE and PySyft (uses PyTorch at the background). Here, the project uses TFF which is still in its initial state (Version 0) and thus not able to support fully for practical implementation except simulation. Thus, this paper contributes to the application/research in TFF.

1.5 Implementation Platform

The project is completed with the following essential platform specifications:

1. TensorFlow 2.3 (TF)
2. TensorFlow Federated 0.17 (TFF)
3. Keras-Tuner 1.0.3
4. Tensorboard 2.6.0
5. Nvidia Geforce GTX 1660 Ti GPU
6. CUDA 10.2

1.6 Organisation of the Dissertation

The project report is structured chronologically as chapters following the project implementation steps, starting from introduction to results discussion. The chapters are listed below in order.

1. **Introduction**: The introduction outlines the background information on anomaly detection in ECG and FL, the problem statement and the motivation of the scope of the project.
2. **Literature Review**: This chapter summarizes the existing techniques and 2 papers are analysed which this project leverages on.
3. **Approach**: This chapter presents the generic steps involved in carrying out the experimentation with the improved model architecture and pseudo-code of the proposed Adaptive Attention Layer Freezing (AALF) algorithm for layer freezing.
4. **Exploratory Analysis Datasets**: This chapter explores and analyses various datasets utilized at different stages of the project.
5. **Obtaining Pre-Trained Model**: The implementation steps and results of obtaining the pre-trained model are explained in this chapter.
6. **FL Experimentation**: The steps in experimenting with the proposed AALF algorithm and the datasets for model training in FL environment are explained in this chapter.
7. **FL Results and Analysis**: The collection and analysis of FL experimental results regarding proposed AALF algorithm are discussed in this chapter.
8. **Limitations and Recommendations**: All types of limitations of this project comprising of the algorithm, implementation platform and others are outlined. Any further improvements complementary to this project are outlined.
9. **Conclusion**: The project scope and the findings are summarized highlighted as the end of the project.

Chapter 2

Literature Review

This chapter presents the literature review of the 2 papers that this project is based upon. In addition, other existing techniques from their corresponding papers are briefly highlighted in the first section to highlight great works in the area of model layer freezing.

2.1 Paper 1

The first project provides essential information on modelling for this project specific to ECG anomaly detection.

2.1.1 Overview

The authors of [15] have developed an explainable AI system for federated ECG anomaly detection on MIT-BIH Arrhythmia data which contain 5 classes. The datasets have been split into 80%-20% for training and evaluation respectively. They have used an AutoEncoder (comprising of Encoder and Decoder) and a Classifier for the anomaly classification for MIT-BIH Arrhythmia data[15]. The AutoEncoder is first trained with the training data augmented with noise[15]. The Encoder of AutoEncoder is then integrated with the fully connected layers of Classifier for classification training where the Encoder layers are frozen and only the fully connected layers are trained[15]. After training, the last layer of Encoder is fed into several customized GradCam layers [15] which are then overlapped with the output of AutoEncoder to present a visual understanding as depicted by Figure 2.1 below.

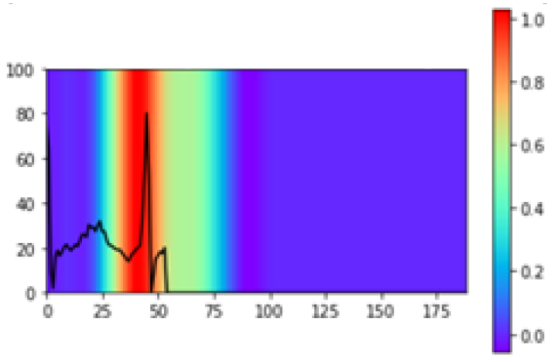


Figure 2.1: Example of Final output at the GradCam for Explainable AI [15].

There are 2 objective functions to accommodate the 2 types of model training. For the recreation of signals with AutoEncoder, Mean Absolute Error (MAE) is minimised[15]. The

classification training utilizes categorical cross-entropy (CCE) as an objective function that is minimised[15] with the softmax function to ensure the sum of class probabilities is 1.

MAE represents the expected absolute difference between the true and predicted data points as shown by Equation 2.1[15].

$$\begin{aligned} &\text{MAE} = (\frac{1}{n}) \sum_{i=1}^{n} |y_i - \hat{y}_i| \,; \\ &y_i = true\ outputs\ , \\ &\hat{y}_i = predicted\ outputs. \end{aligned} \tag{2.1}$$

CCE represents the sum of negative log loss of the classes per sample as expressed by the Equation in 2.2[15].

$$\begin{aligned} &\text{CCE per sample} = -\sum_{i=1}^{\text{No. of classes}} y_i \cdot \log(\hat{y}_i); \\ &y_i = true\ label \\ &\hat{y}_i = predicted\ label. \end{aligned} \tag{2.2}$$

The softmax equation can be mathematically be expressed as the ratios of the exponential of a class to the sum of the exponential of all classes as shown by Equation 2.3[15].

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \tag{2.3}$$

The entire training is situated in an FL environment with 3 clients (Raspberry Pi devices[15]), each given the same dataset of a certain noise level. The initial models do not contain pre-trained parameters and thus are trained from the aggregated updates from the clients. The FL technique is further elaborated in subsection 2.2.3 ”FL Algorithm”.

2.1.2 Model Architectures

The purpose of AutoEncoder here is to be able to learn minimum relevant features to recreate the original ECG signal from the noisy signal. The purpose of the Classifier is to be able to detect and classify the anomalies based on the relevant condensed information found in the last layer of the Encoder.

2.1.2.1 AutoEncoder

As illustrated by Figure 2.2 below, the encoder contains mainly 3 convolution layers and 3 max-pooling layers and the decoder mainly contains 4 convolution layers and 3 upsampling layers[15].

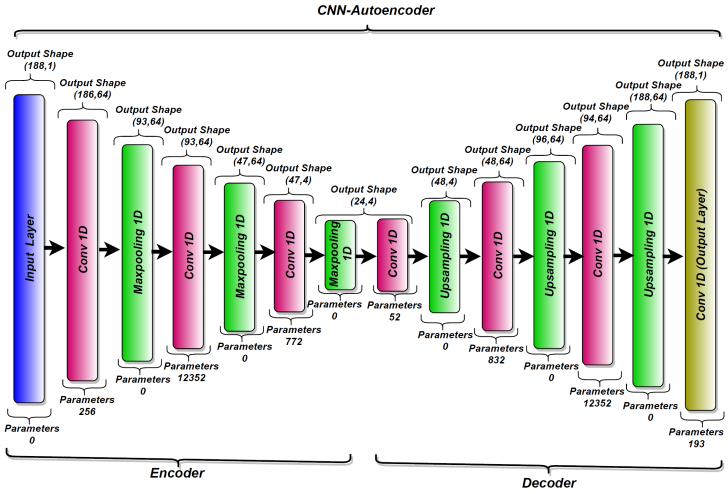


Figure 2.2: Architecture of AutoEncoder [15].

The purpose of max-pooling is to downsize and extract only large values over a sliding window from previous layers that approximate the feature for that particular window of data. The upsampling layers duplicate the edges of the data stream to enlarge the size of the data.

2.1.2.2 Classifier

The Classifier contains the frozen Encoder of the AutoEncoder and fully connected layers built with 1 convolution layer and 2 linear layers[15] as graphically shown in Figure 2.3 below.

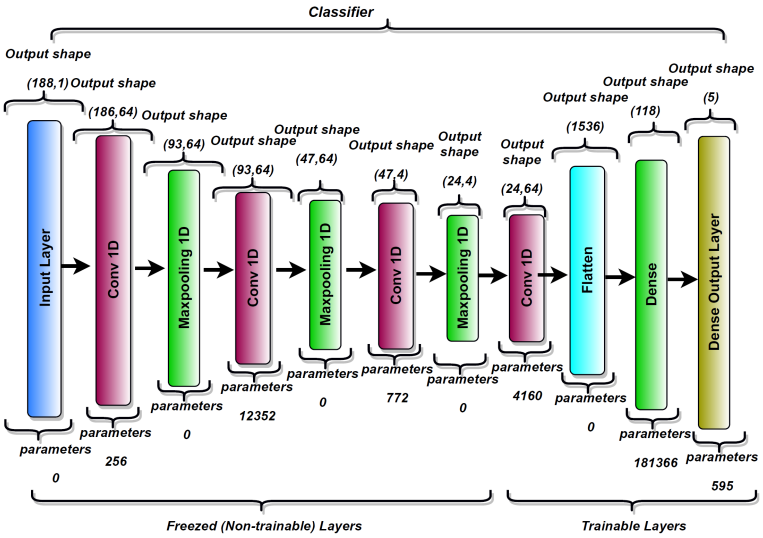


Figure 2.3: Architecture of AutoEncoder [15].

The purpose of the fully connected layers is to flatten the data with many channels from previous layers into single-dimensional data that can be fed into linear layers for classification

decision making. The output is 5 probabilities for every sample of data after the softmax activation function at the last linear layer.

2.1.3 FL Algorithm

The FL algorithm outlines the interaction between the server and the 3 edge devices as clients from broadcasting to aggregation[15].

1. Server initiates AutoEncoder model.
2. Server broadcasts AutoEncoder weights to all 3 clients.
3. Each client performs local training on the AutoEncoder model with private data.
4. Each client sends its trained AutoEncoder weights to the server.
5. The server aggregates the trained AutoEncoder weights from the 3 clients with Federated Averaging.
6. The server then initiates a Classifier model by merging the Encoder of AutoEncoder and fully connected layers. The Encoder layers are frozen.
7. Server broadcasts Classifier weights to all 3 clients.
8. Each client performs local training on the Classifier model with private data.
9. Each client sends its trained Classifier weights to the server.
10. The server aggregates the trained Classifier weights from the 3 clients with Federated Averaging expressed as below.

$$\begin{aligned} &\mathrm{F}edAvg[15] = \frac{\sum_{i=1}^{N=Clients} \omega_i}{N}; \\ &\omega_i = model\ weights \end{aligned} \tag{2.4}$$

The above interaction completes 1 communication round successfully.

2.1.4 Results

The metrics used for results evaluation comprises accuracy, recall, precision and F1-score. They were able to achieve scores ranging from 94% to 98% depending on the degree of noise added while training[15].

2.1.5 Analysis of Paper 1

This paper has provided many positive outcomes based on the high evaluation metrics from the experiments mentioned previously. The advantage of the first model architecture is that the AutoEncoder allows denoising to be more accurate as the compression of data at the end of Encoder contains highly relevant information.

However, having 2 architecture doubles the communication cost between the server and the clients as the time taken for each complete communication round is longer. The entire experiment has been implemented on one type of dataset (MIT-BIH) despite the augmentation with noise. This means that the feature space is similar for clients and server. This questions the model capability when handling unbalanced data or data from other feature spaces. Last but not least, the FedAvg algorithm simply takes the mean of the weights which may pose a problem for unbalanced or even non-IID datasets.

Nevertheless, this paper is significant as it has implemented a system with Explainable AI for ECG classification which can help data scientists to understand the model learning visually.

2.2 Paper 2

The second paper of interest accentuates layer freezing techniques for optimized training in the FL environment.

2.2.1 Overview

The authors of [16] have proposed the Adaptive Parameter Freezing (APF) technique as the most optimised training methodology to train model parameters in an FL environment with proper data privacy techniques without compromising model accuracy. Only the layer freezing technique is explained in this project as it would be adopted in this project. This paper summarizes the importance of APF which helps to speed up the training process by only training necessary parameters which in turn optimizes the number of parameters during the communication between the server and clients. APF is developed in Pytorch and evaluated on the Amazon EC2 platform with 50 instances as clients[16]. Models that have been experimented with are LeNet-5, LSTM and ResNet-18[16].

This paper presents interesting findings that led to the build-up of APF. The model parameter stability does not necessarily mean that the parameter values remain constant over time. It is possible for parameters to change but still considered stable when the changes are minutely oscillating[16]. In addition, the authors of [16] have discovered model parameters within the same layer can undergo changes that vary in magnitude and direction that APF accounts for.

Discovering the parameter value oscillations require new metrics to identify parameter stability. The authors of this paper have proposed effective perturbation to measure and check for parameter stability and is mathematically expressed as Equation 2.5 below[16].

$$\begin{aligned} &\mathrm{P}_k = \frac{||\sum_{i=0}^{Iterations} u_{k-i}||}{\sum_{i=0}^{Iterations} ||u_{k-i}||} = \frac{|E_K|}{E_K^{abs}}; \\ &u_k = weight\ difference\ for\ consecutive\ iterations \end{aligned} \tag{2.5}$$

Effective perturbation refers to the ratio of modulus of summation of parameter changes to the sum of modulus of parameter changes over a window of training iterations[16]. It ranges from 0 to 1. The numerator detects oscillations as the changes in opposing directions cancel each other whiles the denominator adds the changes regardless of the direction[16]. Thus, oscillations result in values towards 0 while non-oscillatory changes result in values towards 1. Stable parameters tend towards 0[16].

The calculation of effective perturbation involves monitoring of weights over the specified window intervals. To simplify the calculation, the authors have come up with an approximation calculation known as Exponential Moving Average (EMA) that is expressed in Equations 2.6 [16] below.

$$\begin{aligned} &\mathrm{P}_k = \frac{|E_K|}{E_K^{abs}}; \\ &E_K = \alpha E_{K-1} + (1-\alpha)\Delta_K; \\ &E_K^{abs} = \alpha E_{K-1}^{abs} + (1-\alpha)|\Delta_K|; \\ &\alpha = smoothing\ factor, \\ &\Delta_K = cumulative\ changes. \end{aligned} \tag{2.6}$$

The calculation of the metric at the current iteration takes in an α percentage of the sum from previous iterations and (1-α) of the current weight changes[16]. This simplifies the calculation process without having to sum over the window of iterations every time.

Many of the layer freezing techniques highlight freezing techniques only. This paper, on the other hand, also suggests unfreezing as part of the APF technique. The authors of [16] have discovered that frozen model parameters have to be unfrozen for training after a certain period. Hence, the duration of freezing can be temporary. APF adopts the classical Transmission Control Protocol(TCP) to control the model parameter freezing and unfreezing during training[16]. The parameters are frozen in an additive manner and unfrozen in a multiplicative manner after exceeding the stable metric criteria mentioned in the previous paragraph[16].

2.2.2 Adaptive Parameter Freezing (APF)

There are 3 main components in the APF workflow namely Clients, Central Server and the APF Manager. The role of APF Manager is to act as an intermediate platform between a client to monitor the metric, effective perturbation, and control the number of parameters frozen[16]. The following indicates the summary of the workflow with the defined variables as requirements[16].

F_s = Frequency at which aggregation is applied
F_c = Frequency at which stability of parameters are checked
T_s = Threshold for effective perturbation which defines the lower boundary for being stable
P_k = Calculation of effective perturbation
N = Number of clients
k = Number of iterations

1. The server initiates the model and sends it to the clients.
2. The clients perform local training and update the model weights.
3. The clients send the unfrozen weights to their respective APF Managers.
4. When k is a multiple of F_s,
   (a) The APF Managers send the unfrozen weight parameters to the server.
   (b) The server performs FedAvg as described on the unfrozen model weights in Expression 2.4.
   (c) The server sends the averaged weights back to APF Managers.
   (d) The APF Managers merge the unfrozen and frozen weights.
   (e) When k is a multiple of F_c, a stability check is performed which comprises the steps listed below.
      i. Increments the freezing duration incrementally by F_c for unfrozen weights when $P_k \leq T_s$.
      ii. Halve the freezing duration $P_k > T_s$ for unfrozen weights.
      iii. Create deadlines for expiry of freezing by summing k with freezing duration. Hence, the deadlines represent the iterations at which unfreezing supposedly starts.
      iv. The frozen model weights remain frozen till k iteration exceeds the deadline iteration created previously.

5. The APF Managers sends the model with new weights to the clients for the next round of communication repeating from the 2nd step.

2.2.3 Results

The results presented are for 3 models - LeNet-5, LSTM and ResNet-18. CIFAR-10 dataset is used for LeNet-5 and ResNet-18 while KWS dataset is used for LSTM[16].

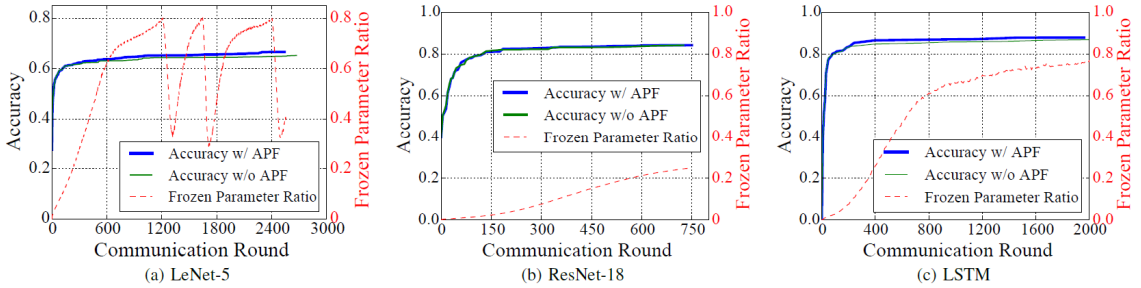


Figure 2.4: Test accuracy for various models [16].

Based on Figure 2.4 above, APF has not compromised the accuracy of the models. APF has improved the accuracy slightly over time for LeNet-5 and LSTM[16].

Model	LeNet-5	ResNet-18	LSTM
Transmission-Volume w/ APF	239 MB	2.62 (1.44) GB	194 MB
Transmission-Volume w/o APF	651 MB	3.12 GB	428 MB
APF Improvement	63.3%	16.0% (53.8%)	54.7%

(a)

Model	LeNet-5	ResNet-18	LSTM
Per-round Time w/APF	0.74 s	139 (95) s	1.8 s
Per-round Time w/o APF	1.02 s	158 s	2.2 s
Improvement	27.5%	12.1% (39.8%)	18.2%

(b)

Figure 2.5: (a) Cumulative transmission volume ;(b) Average time taken per round[16]. Values in brackets are from advanced form of APF known as Aggressive APF.

Based on Figure 2.5 above, the tables in (a) and (b) indicate that APF speeds up the training process in reaching similar accuracy values and reduces the number of trained model parameters during communication[16]. This proves that APF has made the FL training process efficient overall by optimizing the freezing and unfreezing of model parameters.

2.2.4 Analysis of Paper 2

The experimentation results have successfully outlined the positive outcomes of the APF in terms of metrics such as accuracy and thus this technique should be considered and adopted for future use. The unfreezing aspect of APF has helped parameters to continue learning after a designated period. EMA also saves time by only considering the latest update for calculating metric effective perturbation and not over a window every time. The communication between the server and clients only comprises trainable parameters and not the entire model. From the operations perspective, the APF has been implemented as a module that can be easily integrated into existing codes and that helps to be user friendly. As both image and text-based datasets have been used for experimentation, it is appropriate to say that the methodology works on a variety of data and more datasets will add to the credibility.

The APF accounts for every parameter within all trainable layers. The computation can be heavy and complex as the models become larger with more trainable parameters. This may not suitable for low-powered devices with low computing capacity. Furthermore, the TCP control technique manipulates the parameter freeing and unfreezing duration. Though this helps all parameters to be flexible in being frozen and unfrozen, it raises the question of the need for such detailed tracking when perhaps only a very small percentage of parameters within the same layers are required to remain unfrozen. In addition, there is a requirement of threshold value T_s and thus the question of identifying the appropriate range of values remains. Last but not least, the FedAvg algorithm simply takes the mean of the weights which may pose a problem for unbalanced or even non-IID datasets.

Nonetheless, this paper is highly regarded for its contributions to the FL with APF due to positive results that outweigh the disadvantages.

2.3 Samples of Other Existing Layer Freezing Techniques

1) AdaFilter: AdaFilter focuses on DL models with Convolution Neural Network (CNN) architectures as they fine-tune the CNN filter layers when processing images. The pre-trained filter layers and fully fine-tuned filter layers act as references for an external simple DL model to decide which of the filter layers should remain unchanged and which should be fine-tuned[17]. The pre-trained filters are available from pre-trained models while the fully fine-tuned layers are based on the optimal choice of previous filter layers[17].

1. *Advantages* - They can leverage on transfer learning as well as decision to fine-tune is done in real-time by the external model.

2. *Disadvantages* - Extremely time consuming for large CNN models and with numerous CNN filters as the optimal decision requires both fully fine-tuned filter layers and pre-trained filter layers.

2) AutoFreeze: Autofreeze sequentially fine-tunes a small window of layers along with the model from the starting layer. As the window of layers to be fine-tuned shifts towards the last layer, the previously fine-tuned layers are frozen[18]. This has been experimented successfully with large NLP models such as Bert[18].

1. *Advantages* - It is easy to implement and is efficiently fast for large models as the frozen layers are cached.

2. *Disadvantages* - Large memory space is required for caching many frozen layers and thus may not be memory efficient.

3) Bayesian Optimized Incremental Learning: A frozen pre-trained Deep CNN model is first obtained. This technique shares the selected first few frozen layers, clones and branches of the remaining trainable layers to determine the optimal number of layers that requires to be fine-tuned[19]. Hence, this is more of a network architectural search with optimization[19].

1. *Advantages* - The technique is relatively memory-efficient due to sharing of frozen layers rather than caching and also faster search due to Bayesian Optimization.

2. *Disadvantages* - The technique may still up a lot of memory ultimately especially when the model is very deep which also increases the time consumption to search for the optimal number of layers to be fine-tuned.

4) SpotTune: SpotTune is designed by IBM that contains a frozen pre-trained model which is segmented into blocks. Each block of the pre-trained model is connected to its corresponding block of the trainable pre-trained model. At each of these connections, there is a switch to allow the forward pass in either through the frozen block or trainable block[20]. This switch is controlled by an external policy DL network that is also trainable[20].

1. *Advantages* - The external policy network is capable of learning during training and thus no prior knowledge is required. Despite the doubling of model parameters, the forward pass and backpropagation will only involve half the total parameters due to switch control. Hence, the time taken for backpropagation is similar to that of a single model.

2. *Disadvantages* - SpotTune essentially contains twice the model parameters which make it memory inefficient. As the external policy network is a DL model, training may require additional time depending on the use cases.

5) SemifreddoNets: SemifreddoNets is developed by Intel Corporation which uniquely presents vertical layer freezing, unlike many others that perform horizontal layer freezing[21]. Vertical layer freezing means that only certain portions of each layer are frozen while other portions are trainable. SemifreddoNets have the central portion of each layer to be frozen while the extreme portions to be trainable[21]. After each layer or block of layers, the features from frozen and trainable portions of layers are combined and sent as inputs to other layers or blocks of layers[21]. This repeats depending on the depth of the model.

1. *Advantages* - One of the interesting features of this technique is that the combination of features from frozen and trainable portions at each layer or block of layers enhances the learning capacity of the model.

2. *Disadvantages* - Some level of hard wiring is needed to allow portions of layers to be trainable and frozen for memory efficiency.

The list of techniques above briefly summarizes descriptions and quick analysis. Respective papers have identified ways to overcome disadvantages or improve their techniques that are not mentioned here. Despite the unique traits of the individual techniques listed above, all are developed in a local environment. Thus, recent work in the FL environment (paper 2) has been selected for the project that would be best to leverage.

Chapter 3

Approach

This chapter presents the overview of the important steps involved to investigate the hypothesis which involves the use of attention weights of a model. First, we will obtain a pre-trained model with an attention mechanism by training it from scratch on a particular dataset of 1D data as there is no off-the-shelf model readily available for ECG signals. After having the pre-trained model, we will apply the improved layer freezing algorithm AALF in the FL environment.

3.1 Obtaining pre-trained model

During the process of obtaining the pre-trained model, we will have 2 models - base and improved model (a.k.a attention model) - for comparison to safely assume that the attention model performs minimally the same as the base model. This to eliminate or minimize model prediction errors when experimenting with AALF.

The base model that is used as control will closely follow [1] the model from Figure 2.3 as shown below.

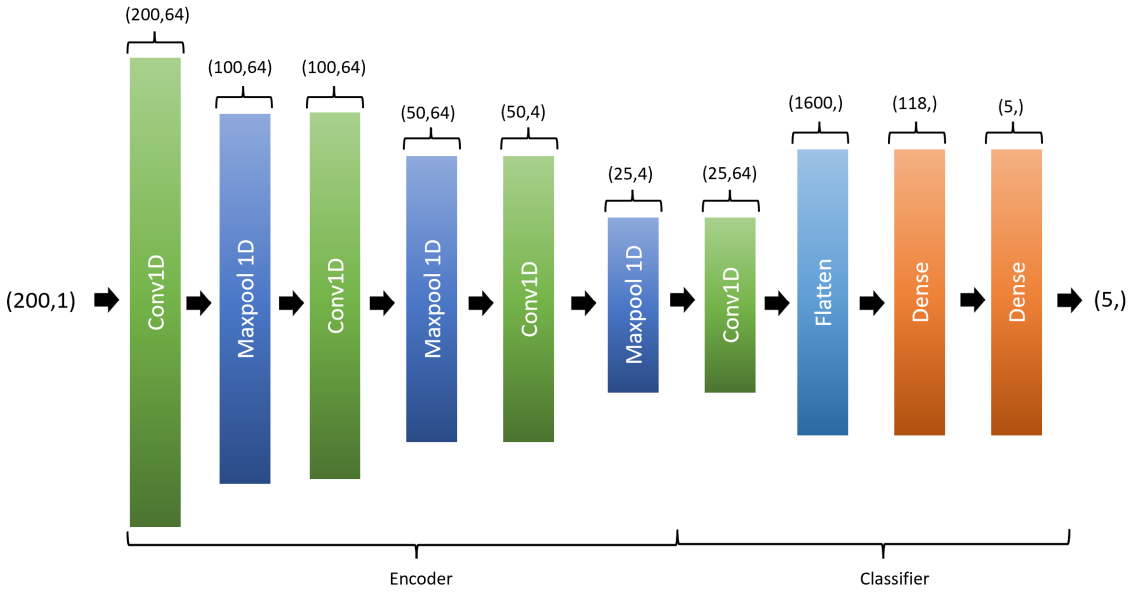


Figure 3.1: Architecture of base model with output feature sizes.

[1] Note that the term Classifier in Figure 2.3 has a different reference as compared to Figure 3.1.

Though the number and type of layers are the same, there is a notable difference in Figure 3.1. The input feature size has been increased to 200 from 188 in Figure 2.3 which in return led to slight differences in the output feature sizes of subsequent layers.

The attention model has the base model architecture modified with the addition of residual blocks of CNN layers and SE Net(Squeezing and Exciting Network) as attention layers. The modified architecture is depicted below in Figure 3.2.

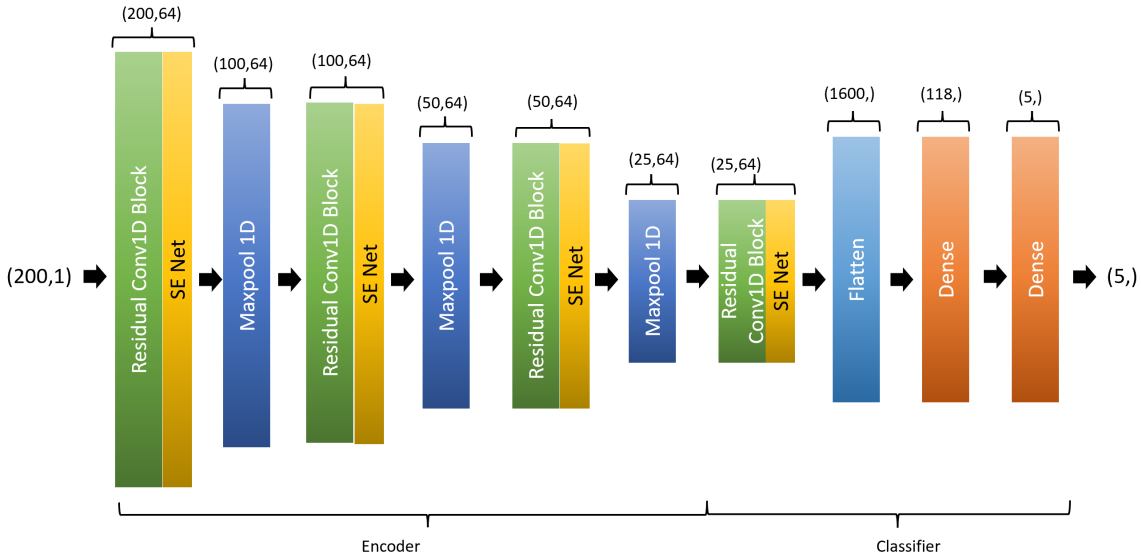


Figure 3.2: Architecture of attention model with output feature sizes.

The architecture of the Residual CNN block imitates that of a typical residual block in ResNet as displayed in Figure 3.3 below. The weight layers are replaced with Convolution 1D layers.

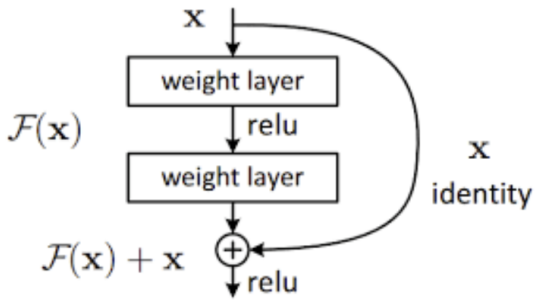


Figure 3.3: Architecture of Residual CNN block[25].

The purpose of having Residual CNN blocks here is to help preserve features from previous layers with skip connections during training when vanishing gradients exist[25]. Ideally, the skip connections can be used to skip layers when necessary during training.

SE Net is another block of linear layers added after the residual block as illustrated in Figure 3.4 below.

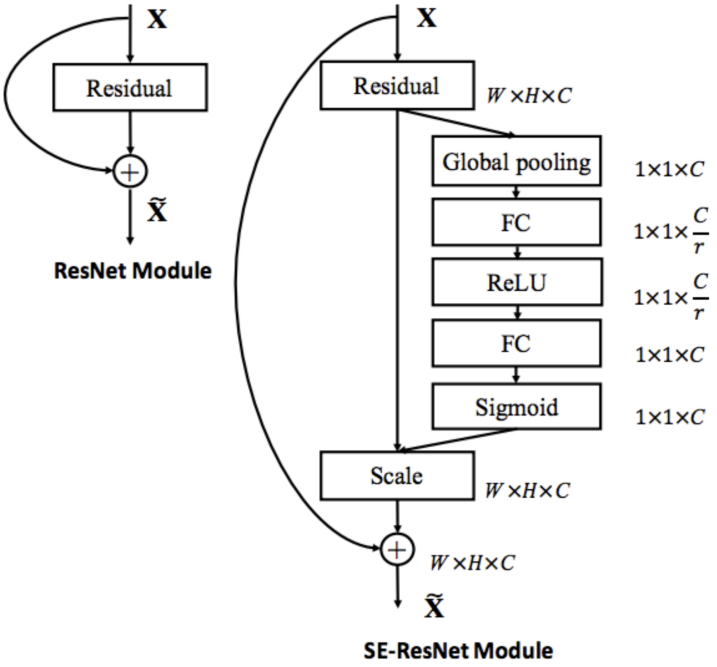


Figure 3.4: Left: Architecture of residual block, Right: Architecture residual block with SE Net[26].

The Global pooling layer as the initial layer of SE Net squeezes the features along all dimensions except the channels. The role of SE Net is to identify the degree of attention required by all the channels in the feature space[26]. Without SE Net, all channels in the features have equal attention. The linear layers learn and determine the right amount of attention required during training. The results from the linear layers are scalar values of length corresponding to the number of channels in the feature space[26]. These scalar values are also known as attention scores. Scaling is applied where the features are multiplied with the attention scores to accentuate the important channels that are relevant[26]. The overall process has been summarized visually below in Figure 3.5.

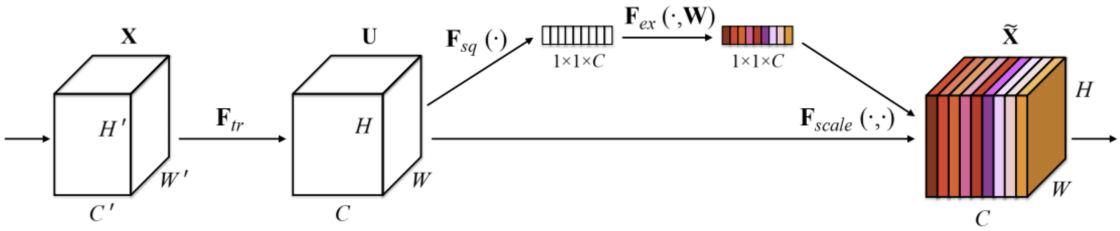


Figure 3.5: Visual summary of SE Net's role[26].

In the context of ECG 1D data, the 1D data are transformed into features containing 64 channels from the original single channel. With SE Net, more attention can be provided to any of the 64 channels in the feature space that are relevant to anomaly detection. This occurs at every SE Net architecture present in the model. SE Net is light-weighted as it has very few parameters as compared to the model which adds less complexity and is easy to train[26]. Having few parameters means no additional large communication cost in the FL process. With more attention to essential features, SE Net also helps to train the model faster which benefits the FL process[26]. Hence, SE Net is an ideal attention mechanism to be used in this project.

The code development for Residual CNN block and SE Net are adapted from [25] and [26] respectively. After building the base and attention models, they will be trained on MIT-BIH Arrhythmia Dataset for comparison.

3.2 FL Workflow

The high-level algorithm of the FL workflow below illustrates the communication process between the central server and the 3 clients. An existing built-in function for FL training [2] has been utilized as it was suitable for gradual learning of the model weights. The choice of using the built-in function has further simplified the FL training process and reduced any worries about communication overheads. AALF is applied at the beginning of the FL training process after every 4 epochs at the central server. More on AALF is discussed in the subsequent section.

Algorithm 1 FL Workflow

Pre-process data and duplicate for each of the 3 clients in the central server.
$n \leftarrow 0$
while $n \neq 100$ **do** ▷ Run 100 epochs with stride of 4.
  Apply AALF algorithm after 1st epoch. ▷ Explained in next section
  $i \leftarrow 0$
  while $i \leq 3$ **do** ▷ Track epochs to ensure periodic application of AALF.
    Step 1: $clients \leftarrow server$ ▷ Steps 1-6 simplified by built in function
    Step 2: Perform FL training in clients on training set.
    Step 3: Clients update their models with certain learning rate.
    Step 4: $server \leftarrow clients$
    Step 5: Server updates their models with certain learning rate.
    Step 6: Aggregate metrics.
    Validate the updated model on validation set and record at server.
    $i \leftarrow i + 1$
  end while
  $n \leftarrow n + 1$
end while

[2] The built-in function is tff.learning.build_federated_averaging_process.

3.3 AALF Algorithm

The model begins training with all layers frozen except for fully connected layers. The AALF algorithm gradually allows an optimized number of attention layers to be trained or frozen depending on the metrics. There are a total of 3 steps.

1. **get_permissible_attn_layers**:: Gradually increases the number of attention layers that can be frozen or trained and these are known as permissible [3] layers based on F1 score threshold. The rest remain frozen.

2. **check_perm_trainable_status**:: The permissible attention layers are then determined to be frozen or trainable based on the custom coverage metric.

3. **modify_model_for_AALF_loop**:: Simply modifies the model layers' trainable status based on the updated results from previous 2 steps.

The steps above have been illustrated below in Figure 3.6 to visually define terms such as permissible layer and frozen layer.

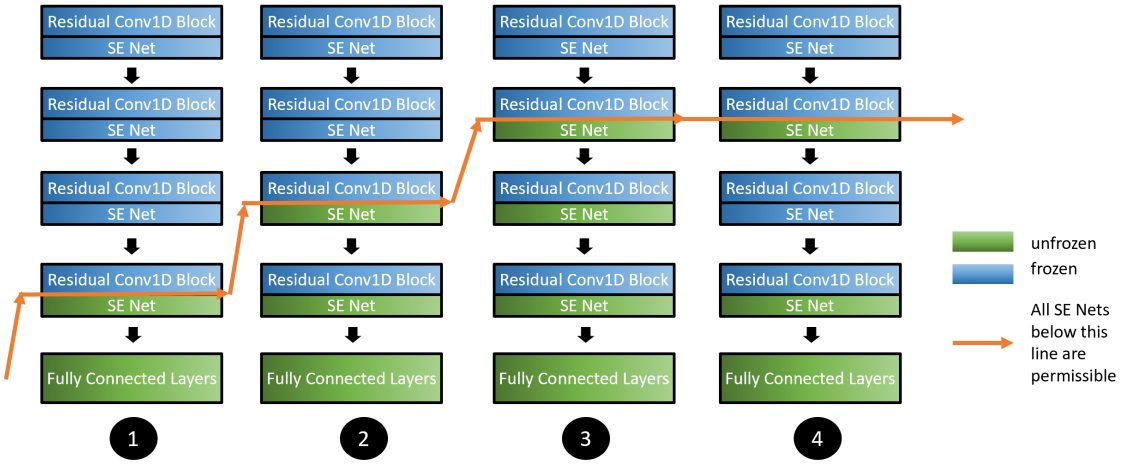


Figure 3.6: Demonstration of AALF on parametric layers over 4 epochs.

The main aspects of the AALF algorithm are concentrated in steps 1 and 2. From epochs 1 to 3 in Figure 3.6, the number of unfrozen attention (SE Net) layers in the model gradually increases which is the function of step 1 listed above. These are also the permissible layers as they are below the orange line in Figure 3.6 above. The layers below the orange line can be considered as the outputs from Step 1. Step 2 freezes or unfreezes the permissible SE Net layers below the orange line. This is transparent in epoch 4 where the number of permissible layers remain the same as the model in epoch 3 but has 1 SE Net layer frozen. Hence, a permissible layer can be frozen or unfrozen. The combination of steps 1 and 2 supposedly produces the optimized number of attention layers required for training the model. The steps above have been illustrated below in Figure 3.6 to visually define terms such as permissible layer and frozen layer.

[3] permissible layer has the liberty to be trained or frozen

The following Algorithm 2 represents for step 1 where the permissible layers are identified.

Algorithm 2 Algorithm: get_permissible_attn_layers

Function get_permissible_attn_layers:

mean_f1_diff = mean score of all f1 score differences
mean_curr_f1 = mean score of all current f1 scores
mean_prev_f1 = mean score of all previous f1 scores
curr_permissable_list = list of possibly trainable attention layers that can be either frozen or unfrozen
all_attn_list = list of all attention layers' indices
f1_diff_thresh = minimum value for each parameter in layers to be considered as significant change

if $absolute(mean\_f1\_diff) \leq f1\_diff\_thresh \;||\; mean\_curr\_f1 < mean\_prev\_f1$ **then**
    $permissable\_num \leftarrow permissable\_num + 1$ ▷ Permissable layer has liberty to be frozen or not
    Update curr_permissable_list
end if
return curr_permissable_list

The following Algorithm 3 represents step 2 where the layers in the list of permissible layers (from step 1) are determined to be trained or frozen. This is done with the comparison of coverage metric score that indicates the proportion of parameters per layer experiencing significant changes. Significant changes refer to changes in parameter values in each layer that exceed a threshold range known as min_val. This threshold range accounts for parameters that experience small oscillations (as suggested by [16]) that can be frozen.

The higher the number of parameters experiencing significant changes, the higher is the coverage score for a particular layer. If the coverage score exceeds the minimum coverage (min_coverage), that layer will be unfrozen to be trained. Else, the layer is frozen. These frozen layers are then unfrozen when the AALF algorithm is applied again after 4 epochs based on our FL workflow in the previous section.

Algorithm 3 Algorithm: check_perm_trainable_status

Function check_perm_trainable_status:
min_coverage = threshold for freezing
for *layer in curr_permissible_list from step* 1 **do**
  if *layer's status* $= Unfrozen$ **then**
    $coverage \leftarrow gets\_coverage()$ ▷ Obtain the coverage value for the layer
    if $coverage < min\_coverage$ **then**
      $layer'sstatus \leftarrow Frozen$
    end if
  else ▷ Skip checking the threshold for frozen layers
    $layer'sstatus \leftarrow Unfrozen$ ▷ Unfreeze frozen layer
  end if
end for
return updated status of model layers

Function get_coverage:
min_val = threshold for parameter value change
$diff \leftarrow currentweights - previousweights$ ▷ Matrix of parameter weights
$mask \leftarrow sum(absolute(diff) > min\_val)$ ▷ Number of parameters beyond min value
return mask / total number of parameters

With the 2 steps above, AALF is expected to reduce the computational cost by determining the model's trainable status at a layer level with the concept of custom coverage metric instead of parameter level in the APF. With the threshold value range, parameters in layers that undergo small oscillations are exempted from coverage calculation and thus is expected to fulfil a similar purpose as APF but at a faster and cheaper cost. The additional inclusion of validation f1 score in identifying permissible layers in step 1 is expected to align the optimum training towards maximum learning and not just focus on the stability of parameters.

3.4 Metrics

The metrics required for assessment at the end are similar to those mentioned in the Literature papers in Chapter 2. Precision, recall and f1-scores on the test datasets at the central server are the scalar metrics for evaluation. Other scalar metrics include the number of trainable attention layers that indicates the efficiency of the FL training process. Confusion matrices are great indicators of insights on the model performance. The weight distribution of the attention layers during training is monitored to identify changes. Reduction in computational cost can be tracked via training time.

3.5 Modifications from Literature papers

As the project adopts and merges content from 2 Literature papers, it is essential to highlight the modifications which are outlined in this section.

The project has indeed adopted the second of the 2 model architectures from Paper 1 in Chapter 2. However, the model in this project has been modified with the addition of residual CNN blocks and SE Nets. A single FL training loop in paper 1 involves 2 model exchanges between the server and the clients while it has been reduced to 1 in this project. The project also trains the model on the MIT-BIH dataset to obtain a pre-trained model which is then used to train and evaluate on other datasets [4] in the FL environment. This is contrary to the training and evaluation process in Paper 1 which focuses only on the MIT-BIH dataset. Furthermore, the aggregation process in the server for this project utilizes stochastic gradient descent with a learning rate while Paper 1 makes use of averaging. Likewise, Paper 2 applies averaging as an aggregation method.

The project has introduced the AALF algorithm for 1D data which has been modified from APF in Paper 2 for 2D data. FL training with APF in Paper 2 begins with unfrozen model layers with initial values[16]. AALF, however, has been initialized with pre-trained weights (discussed in Chapter 5) and thus FL training with AALF begins with all frozen layers except for the fully connected layers. AALF focuses only on the attention SE Net layers, unlike in Paper 2 where there are no attention layers and APF has been applied on all model layers. AALF attempts to simplify the granular approach of APF by measuring custom coverage metric scores for each layer to determine their trainable status. The APF involves the multiplicative increase and additive decrease layer frozen duration[16]. AALF has standardized the frozen duration as 4 epochs. APF's purpose is to detect stability for optimum training but AALF aligns optimum training towards maximum learning with the account for validation f1 scores.

Lastly, one of the essential differences of this project is the code implementation in the TFF simulation environment.

[4] PTB and ECG5000 datasets are used in the FL environment in this project. More details are discussed in the subsequent Chapters.

Chapter 4

Exploratory Analysis of Datasets

There are 3 types of datasets that are used for the experimentation namely MIT-BIH, ECG5000 and PTB. MIT-BIH and PTB are obtained from the source [22] and ECG5000 is obtained from the source [23].

4.1 MIT-BIH

4.1.1 Overview

The datasets orginate from the MIT-BIH Arrhythmia Dataset[22]. The samples have been pre-processed via down-sampling where necessary and padding with zeros to reach desired length[22]. The train and test datasets are available for heartbeat classification based on various Arrhythmias. There are a total of 87,554 samples for the training set and a total of 21,892 samples for test set. Each sample contains 187 features with a label from one of the 5 classes. The 5 classes are listed below with its corresponding class index[22, 24].

1. Class index 0: N (Normal)
2. Class index 1: S (Supraventricular ectopic)
3. Class index 2: V (Ventricular ectopic)
4. Class index 3: F (Fusion beat)
5. Class index 4: Q (Unknown beat)

4.1.2 Target Distribution

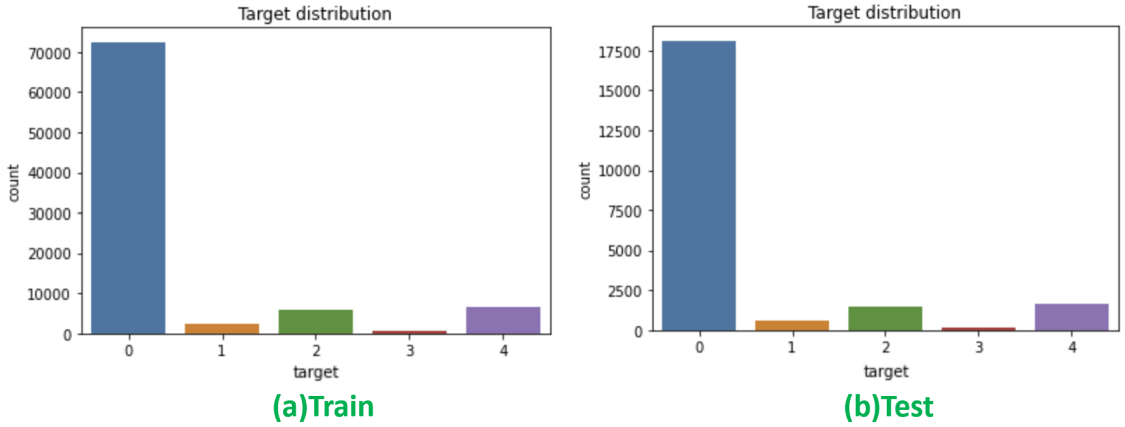


Figure 4.1: Target distribution of MIT-BIH train and test datasets.

From Figure 4.1 above, the distribution of classes in training and test sets are similar. However, both the datasets are highly imbalanced with class 0 having the highest proportion while class 3 having the least proportion of samples. This means that the datasets have to be resampled which is further discussed in Chapter 5.

4.1.3 Visualization of Individual Class

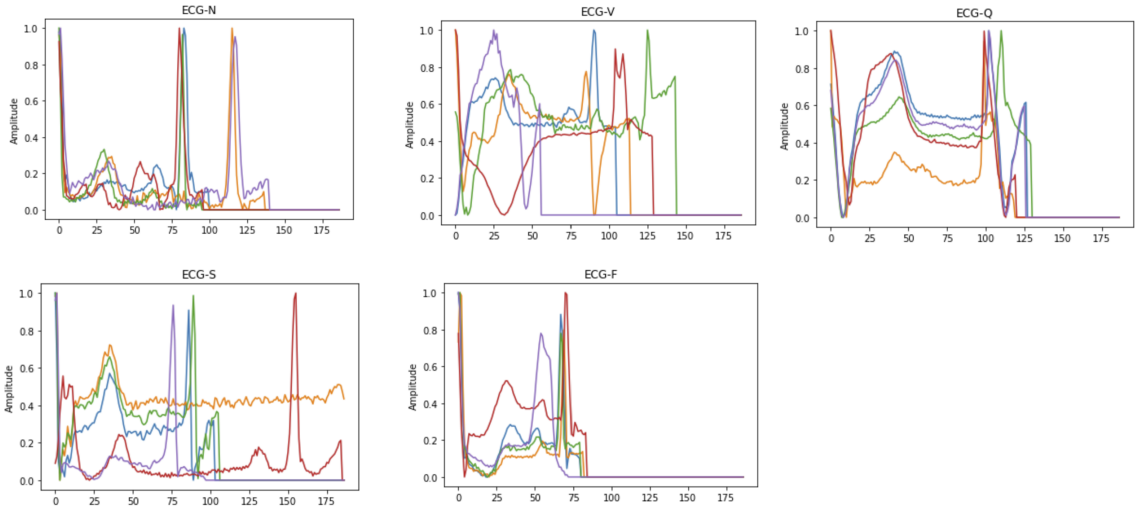


Figure 4.2: Visualization of 5 samples per class in MIT-BIH datasets.

Figure 4.2 above displays 5 samples from each class. This is to provide an approximation of how much the signals vary. Classes 0(N) and 4(F) constitute samples that are very similar in the respective classes and hence it can be approximated that the variance is small. The curves are rather different in each of the other classes, indicating high variances in these 3 classes. Having smaller variance means the model can quickly learn from fewer samples due to their high similarity among the samples.

4.2 ECG5000

4.2.1 Overview

The pre-processed dataset originates from BIDMC Congestive Heart Failure Database and is originally 20-hour long[23]. The pre-processing steps (prior downloading) involve extracting and interpolating each heartbeat to result in equal lengths[23]. The train and test datasets are available for download for ECG Classification. There are a total of 500 samples for the training set and a total of 4,500 samples for the test set. Each sample contains 139 features with a label from one of the 5 classes. The 5 classes are listed below with their corresponding class index[23, 24].

1. Class index 0: N (Normal)
2. Class index 1: R-on-T (R-on-T premature ventricular contraction)
3. Class index 2: PVC (Premature ventricular contraction)
4. Class index 3: SP (Supraventricular premature beat)
5. Class index 4: UB (Unclassified beat)

4.2.2 Target Distribution

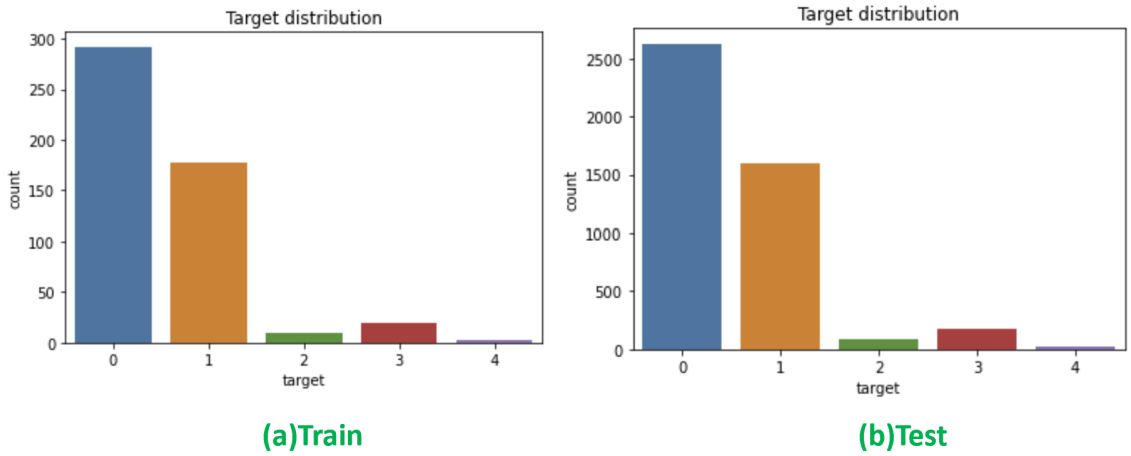


Figure 4.3: Target distribution of ECG5000 train and test datasets.

From Figure 4.3 above, the distribution of classes in training and test sets are similar. However, both the datasets are highly imbalanced with class 0 having the highest proportion while class 4 having the least proportion of samples. This means that the datasets have to be re-sampled which is further discussed in Chapter 6.

4.2.3 Visualization of Individual Class

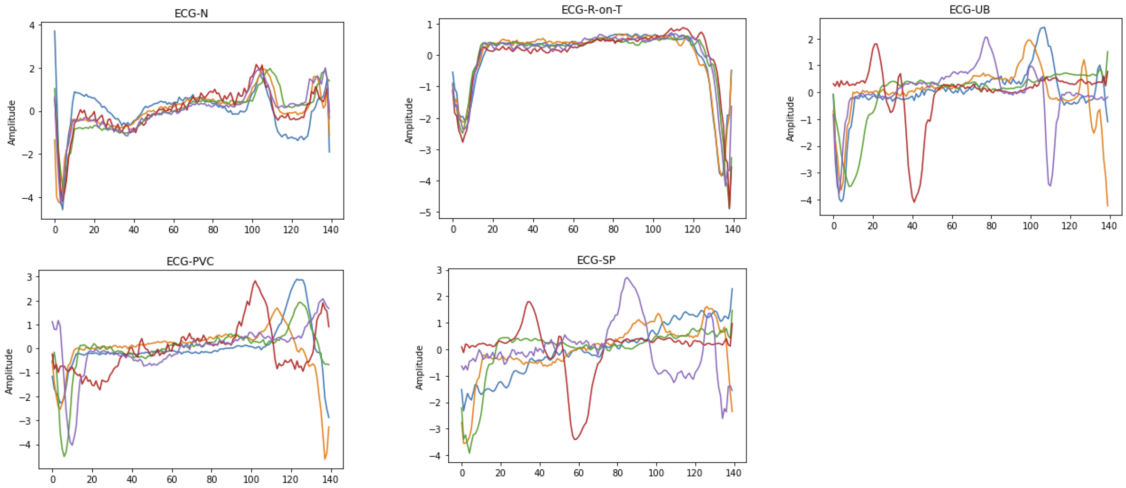


Figure 4.4: Visualization of 5 samples per class in ECG5000 datasets.

Figure 4.4 above displays 5 samples from each class. This is to provide an approximation of how much the signals vary. Classes 0(N), 1(R-on-T) and 3(PVC) constitute samples that are very similar in the respective classes and hence it can be approximated that the variance is small. The curves are different in each of the other 2 classes, indicating high variances.

4.3 PTB

4.3.1 Overview

The datasets originate from the PTB Diagnostic ECG Database[22]. The samples have been pre-processed via down-sampling where necessary and padding with zeros to reach desired length(prior downloading)[22]. The source contains 2 datasets containing normal and abnormal samples in respective datasets. Unlike, the other 2 datasets earlier, the samples have not been split into training and test datasets. Furthermore, this is binary classification as there are only 2 classes. There are a total of 4,046 normal samples and 10,506 abnormal samples. Each sample contains 187 features with a label from one of the 2 classes. The 2 classes are listed below with their corresponding class index[22].

1. Class index 0: Normal
2. Class index 1: Abnormal

4.3.2 Target Distribution

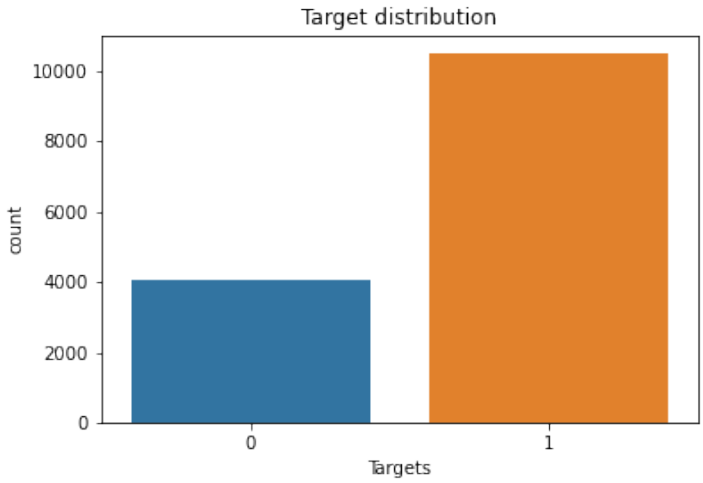


Figure 4.5: Target distribution of PTB datasets.

From Figure 4.5 above, the distribution of classes is uneven with class 0 having half the proportion of class 1. This means that the datasets have to be re-sampled which is further discussed in Chapter 6.

4.3.3 Visualization of Individual Class

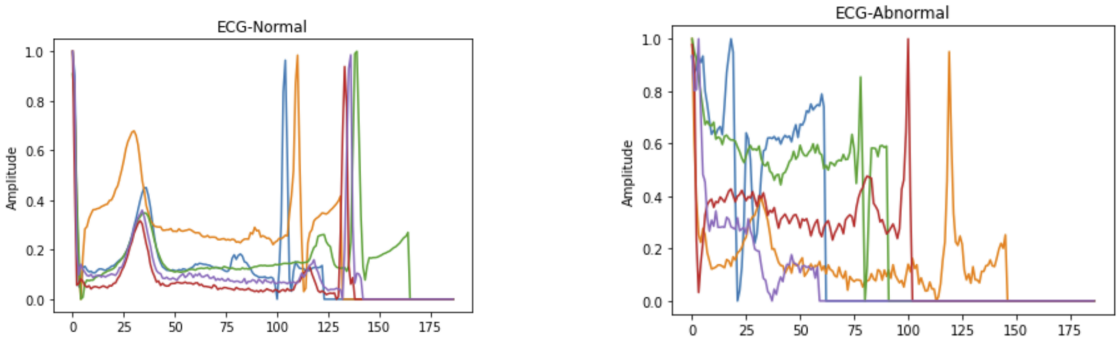


Figure 4.6: Visualization of 5 samples per class in PTB datasets.

Figure 4.6 above displays 5 samples from each class. This is to provide an approximation of how much the signals vary. The Normal class constitutes samples that are very similar to one another, indicating small variances. The Abnormal class constitutes curves that are different from one another, indicating high variances.

Chapter 5

Obtaining Pre-trained Model

It is essential to have the pre-trained attention model (our improved architecture) before moving to the FL environment with Transfer Learning. There are 2 models that are trained for comparison - namely the 2 architectures from Figure 3.1 and 3.2 for base and attention models respectively. The goal of the experiment in this section is not just to have a pre-trained attention model but also to ensure that the attention model produces equivalent or better results than the base model. This eliminates or minimizes resulting errors from the model predictions from being carried forward.

5.1 Data Pre-processing

The dataset used for pre-training is the MIT-BIH Arrhythmia dataset. There are some transformation steps to balance the original data via sampling and data augmentation.

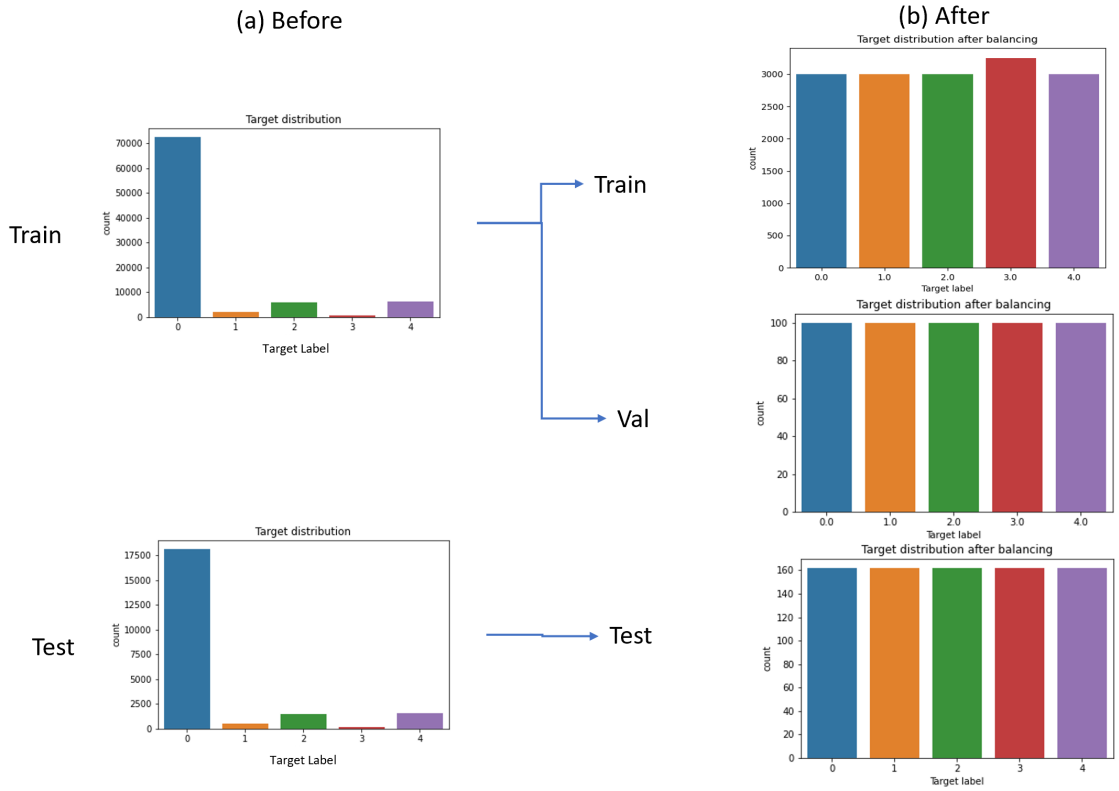


Figure 5.1: (a) Original datasets (b) pre-processed datasets.

Figure 5.1 presents the target distributions before and after the pre-processing steps. The original imbalanced train dataset has been split into balanced train and validation datasets. The final validation dataset contains 100 samples for each target with random down-sampling and the final train dataset contains approximately 3000 samples for each target. The original imbalanced test dataset is randomly down-sampled to a balanced pre-processed test dataset containing 162 samples for each target.

Unlike the test and validation datasets, the train dataset could not be obtained simply with random downsampling. As there has to be sufficient data for the model to learn, too much under-sampling may not lead to desired results. Hence, 1000 samples for each target label are to be randomly sampled from the original train dataset after removing the samples for the validation dataset. Class 3 had approximately 500 samples and thus had to be doubled to approximate balance with other classes. This is done via augmenting with Gaussian noise of sigma 0.005. The visualization of this intermediate step is presented as the middle plot in Figure 5.2 below.

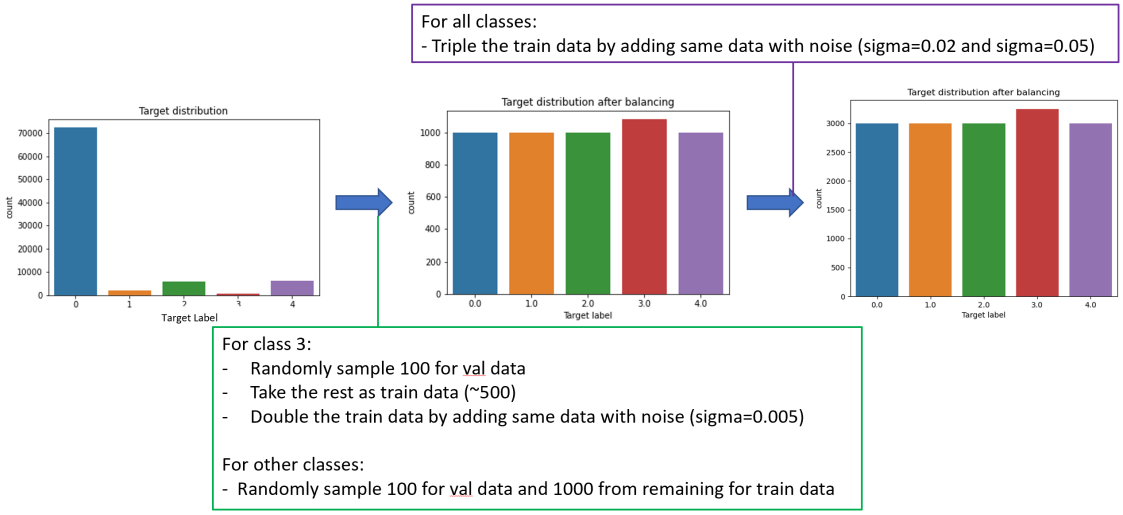


Figure 5.2: Overview of pre-processing steps for train dataset.

All the samples of all target labels are tripled in quantity with the addition of noises sigma 0.02 and sigma 0.05. The end product is presented as the plot on the right of Figure 5.2. The outcome of the additions of respective Gaussian noise has been illustrated below in Figure 5.3 to highlight the difference among them.

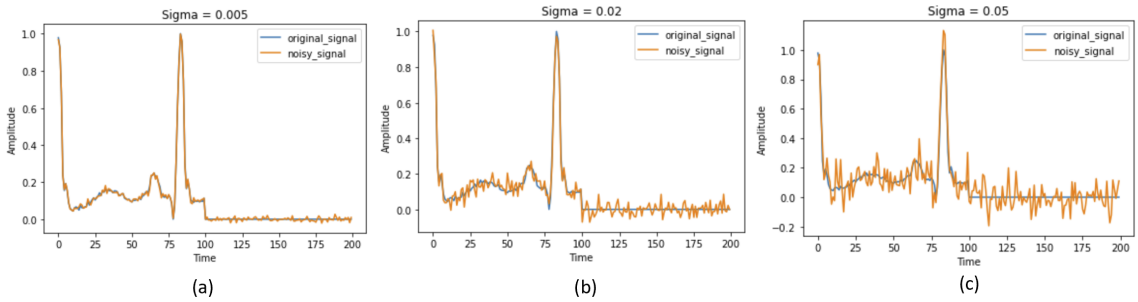


Figure 5.3: Augmenting with (a)Gaussian noise sigma 0.005 (b)Gaussian noise sigma 0.02 (c)Gaussian noise sigma 0.05.

According to Figure 5.3, the lower the sigma value, the lesser are the variations from the

original data points. As such, noise addition allows the data to be more realistic apart from performing the role of data augmentation as an oversampling strategy. Each data sample has been padded with zeros to accommodate an array length of 200 data points, equivalent to 200 variables.

5.2 Hyper-parameter Tuning

The optimizers - SGD and Adam - and various learning rates have been the hyper-parameters for model tuning. The total number of trials [1] is 30 and each trial ran for 50 epochs. The search for the best combination of hyper-parameters is completed with Bayesian Optimization. The validation plots in Figure 5.4 below illustrates the concentration of a few trials at the respective best scores.

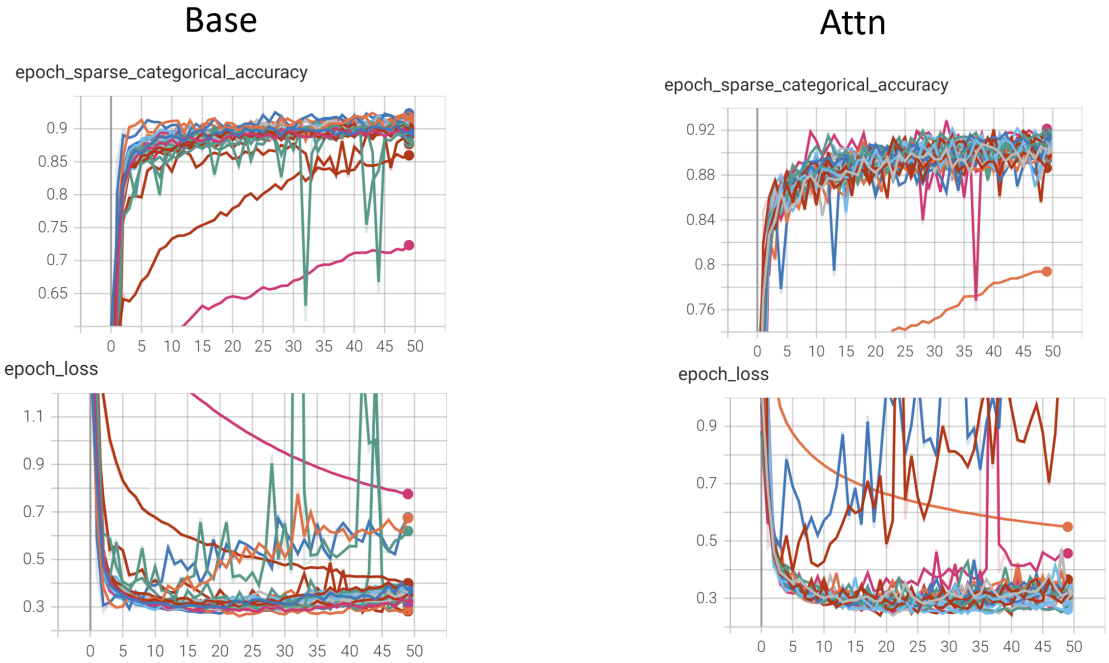


Figure 5.4: Plots of validation categorical accuracies and losses of the base (on the left) and attention (on the right) model for selected hyper-parameters.

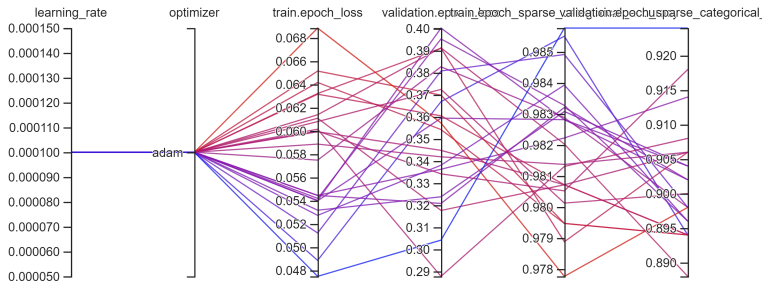


Figure 5.5: Trials of best hyper-parameter combination for base model concentrating validation accuracy range between 0.895 and 0.905.

[1] Trial refers to each run of a model for 50 epochs with a certain combination of hyper-parameters.

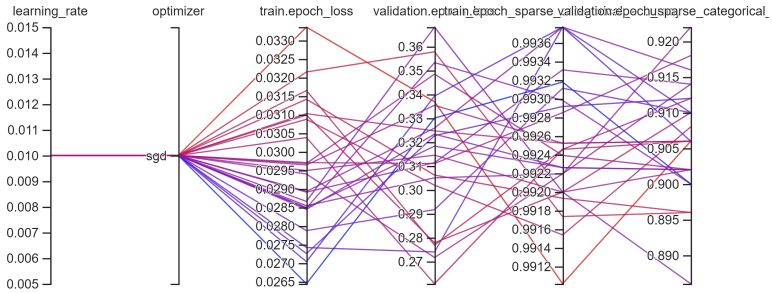


Figure 5.6: Trials of best hyper-parameter combination for attention model concentrating validation accuracy range between 0.905 and 0.920.

Based on Figures 5.5 and 5.6, the respective best hyper-parameter combinations are Adam optimizer with a learning rate of 0.0001 for the base model and SGD optimizer with a learning rate of 0.01 for the attention model. The 2 models are then retrained with the best hyper-parameters for 100 epochs. The categorical accuracies and the losses of both models are plotted below in Figure 5.7.

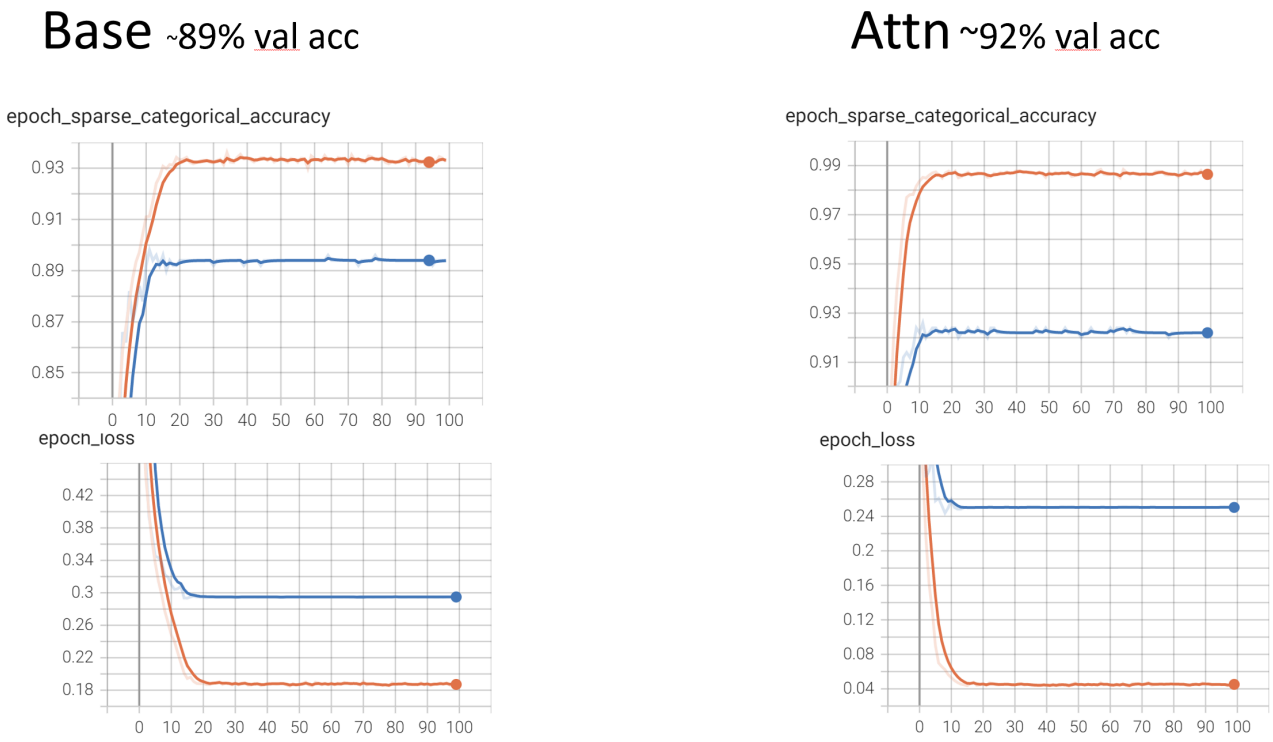


Figure 5.7: Accuracy and loss plots for base and attention models (Blue: Training and Red: Validation).

The validation accuracies of the base and attention model are approximately 89% and 92% respectively based on Figure 5.7. The respective validation losses are approximately 0.19 and 0.05 from Figure 5.7. Based on the metrics on the validation set, the attention model has performed better than the base model with higher accuracy and lower loss.

5.3 Results

The models are then evaluated on the test set. The overall precision, recall and f1-scores are 0.89 each for the base model and 0.91 each for the attention model. The higher scores for the attention model highlight its better performance based on Figure 5.8.

Base

	precision	recall	f1-score	support
N	0.79	0.85	0.82	162
V	0.88	0.78	0.83	162
Q	0.88	0.90	0.89	162
S	0.89	0.92	0.91	162
F	0.99	0.97	0.98	162
accuracy			0.89	810
macro avg	0.89	0.89	0.89	810
weighted avg	0.89	0.89	0.89	810

(a) Metrics table for base model

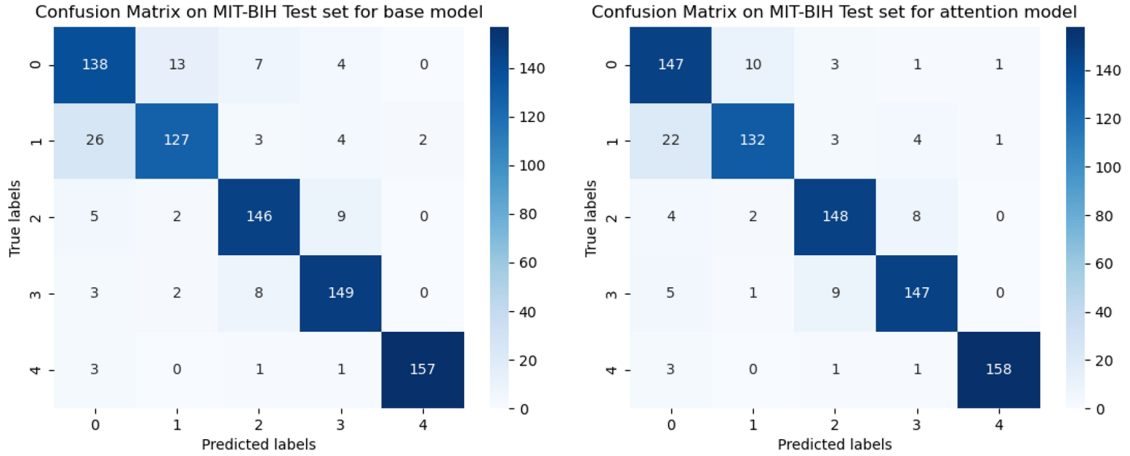
Attn

	precision	recall	f1-score	support
N	0.81	0.91	0.86	162
V	0.91	0.81	0.86	162
Q	0.90	0.91	0.91	162
S	0.92	0.91	0.91	162
F	0.99	0.98	0.98	162
accuracy			0.91	810
macro avg	0.91	0.91	0.91	810
weighted avg	0.91	0.91	0.91	810

(b) Metrics table for attention model

Figure 5.8: Metric tables of test dataset for base and attention models.

It is also clear that all the 3 metric scores for many target labels are better or minimally the same for the attention model as compared to the base model from Figure 5.8. Despite the decrease of 0.01 in recall for label S, the precision has increased by 0.03 when comparing the attention and base models. Hence, the attention model still outperforms the base model. Figure 5.9 below presents the confusion matrices on the test set by both models.



(a) Confusion Matrix for base model

(b) Confusion Matrix for attention model

Figure 5.9: Confusion matrices on test dataset for base and attention models.

From Figure 5.9, the confusion matrix of the attention model contains higher values along the diagonal direction, in which the confusion matrix plot contains blue boxes. One interesting observation is that both models tend to perform worst on the target label N (Normal). Intuitively, it is desired to have a model that predicts ECG anomalies better than the normal class, corresponding to the current result of the models.

As the attention model has performed better than the other, the goal of the pre-training has been successfully fulfilled. The attention model is used for further experimentation in the FL environment which are documented in the upcoming chapters.

Chapter 6

FL Experimentation Set-up

6.1 Experiment Set-up

Previously when obtaining the pre-trained model, there was a baseline model for comparison with the improved model. Similarly, there are baselines for FL training as well. There are two baseline FL training loops namely lower and upper bounds which contain the same models with differing trainable layers.

The lower bound FL training loop consists of the model with only the fully connected layers unfrozen. The upper bound consists of the model with all the attention and fully connected layers unfrozen. Other aspects of the training remain constant. These two bounds represent the two extremes of minimum and maximum attention layers trained in the model.

Ideally, the AALF identifies the optimal number of attention layers that requires training which is in between the lower and upper bounds. It is possible for the results to be equivalent to those of the lower or upper bounds. These two extremes will serve as great baselines for other metric comparisons such as time taken. Time taken determines whether AALF can optimally train at a faster rate compared to the baselines without compromising much on the accuracy.

The 3 types of FL training loops have been summarized in the Figure 6.1 below.

Figure 6.1: Left: Lower bound FL loop as baseline — Middle: FL loop with AALF — Right: Upper bound FL loop as baseline.

Note that 3 types of FL training loops translate to 3 models for each dataset. In this project,

there are 2 datasets for FL training which translate to 6 models used for comparison with their results in the next chapter.

The FL workflow steps are simplified as below. In the FL experiments, three clients are communicating with the central server.

1. Initiate model at the central server
2. Central server sends the model to the clients for FL training. Clients send back model weight updates to the central server after 1 round of training. Aggregation of model weights and metrics occur at the central server.
3. At every 4 epochs, apply AALF based on the metrics.
4. Repeat steps 2 to 4 till 100 epochs

The details of the FL workflow are referenced from Section 3.2 under Chapter 3.

6.2 FL Datasets

There are two datasets used in FL training - ECG5000 and PTB. The data exploration has been documented previously in Chapter 4. The pre-processing steps closely follow the steps for obtaining a pre-trained model in Chapter 5 such as the train-validation-test splits, addition of noise for balancing and padding to the length of 200 with zeros for standardized input sizes with a few minor additional changes.

Furthermore, the data size remains the same for all clients when training for a particular dataset. This condition minimises the issue of quantity skew (described in Section 1.1.2) among the clients for the same dataset.

6.2.1 ECG5000

Based on the data exploration in Chapter 4, the original train data has fewer data compared to training data. To accommodate sufficient samples for each class, the train and test data have been swapped. From here onwards, the train data refers to original test data and vice versa.

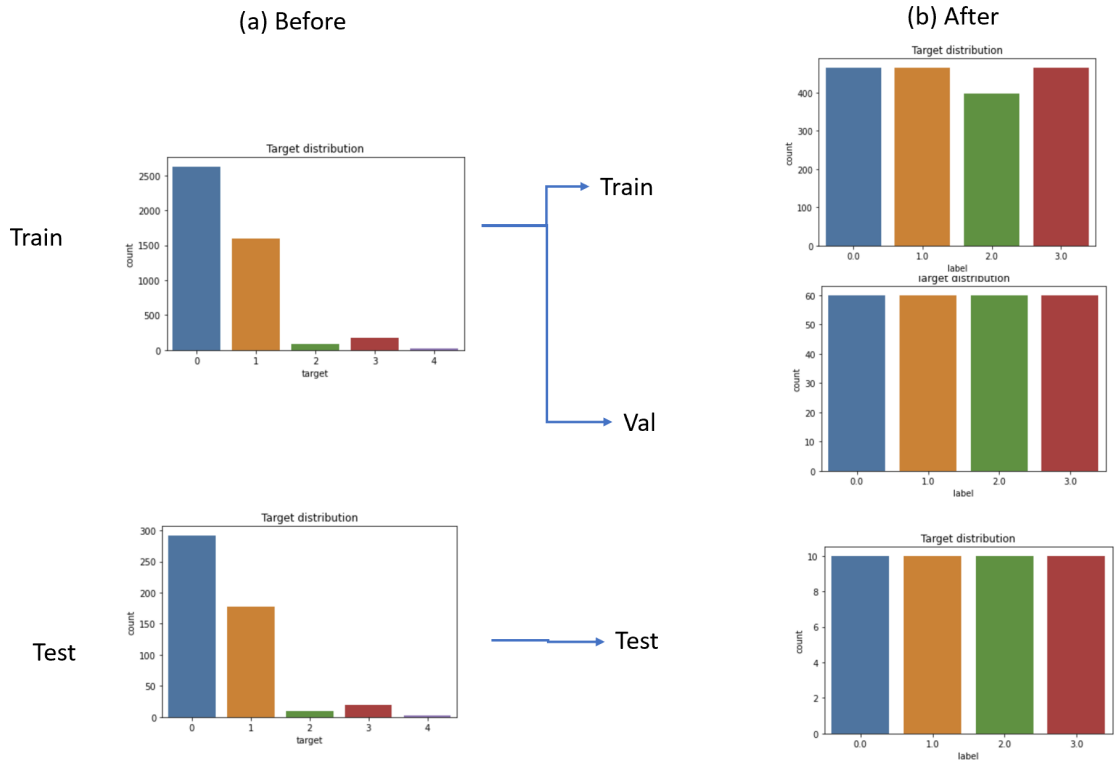


Figure 6.2: ECG5000: (a) original datasets (b) pre-processed datasets.

Figure 6.2 presents the target distributions before and after the pre-processing steps. The imbalanced train dataset has been split into balanced train and validation datasets. The final validation dataset contains 60 samples for each target with random down-sampling and the final train dataset contains approximately 150 samples for each target. The imbalanced test dataset is randomly down-sampled to a balanced pre-processed test dataset containing 10 samples for each target. The number of training samples has been limited to cater for parallel training and the computing capacity of FL in the hardware. A very essential point is that class 4 (Unclassified beat) has been dropped due to the lack of sample count. There are only 4 classes involved.

Unlike the test and validation datasets, the training dataset could not be obtained simply with random downsampling. 150 samples for each target label are randomly sampled from the original train dataset for classes 0,1 and 3 after removing the samples for the validation dataset. The number of samples in Class 2 had to be doubled to approximate balance with other classes. This is done via augmenting with Gaussian noise of sigma 0.005. The visualization of this intermediate step is presented as the middle plot in Figure 6.3 below.

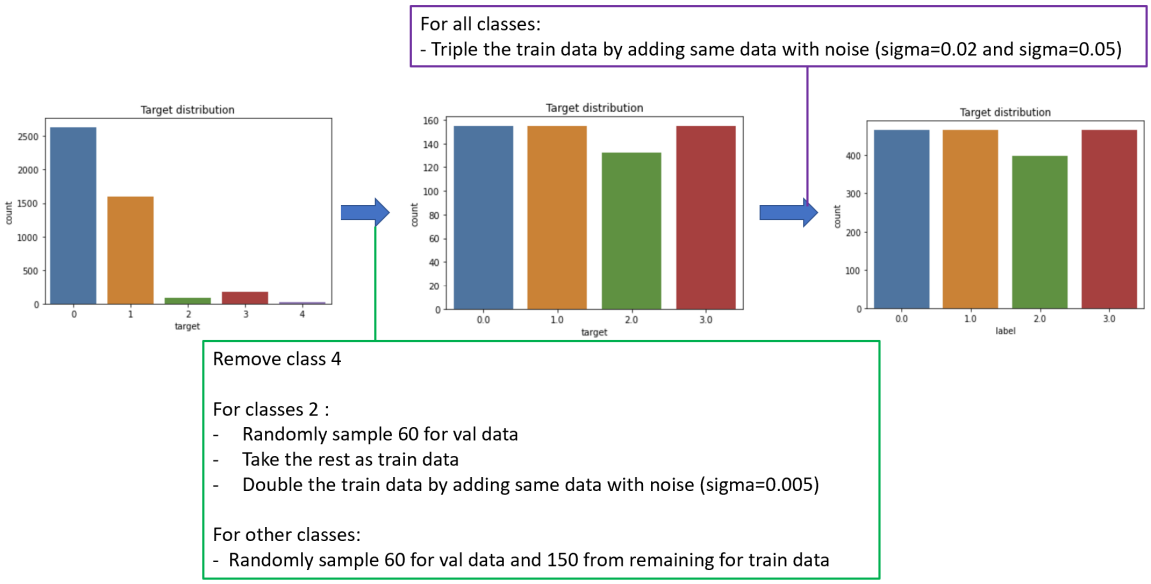


Figure 6.3: ECG5000: Overview of pre-processing steps for train dataset.

Upon completion of previous steps, the entire balanced train set has been tripled from 150 to 450 for each class with augmentation with Gaussian noise of sigma 0.02 and 0.05. The three clients receive the balanced train set with various levels of Gaussian noise added (sigma = 0, 0.02, 0.05) respectively.

6.2.2 PTB

Based on the data exploration in Chapter 4, the original data is imbalanced. As there are a large number of samples, the train, validation and test datasets are easily formed with random down-sampling to the sizes of 200, 150 and 50 respectively, observed in Figures 6.4 and 6.5 below.

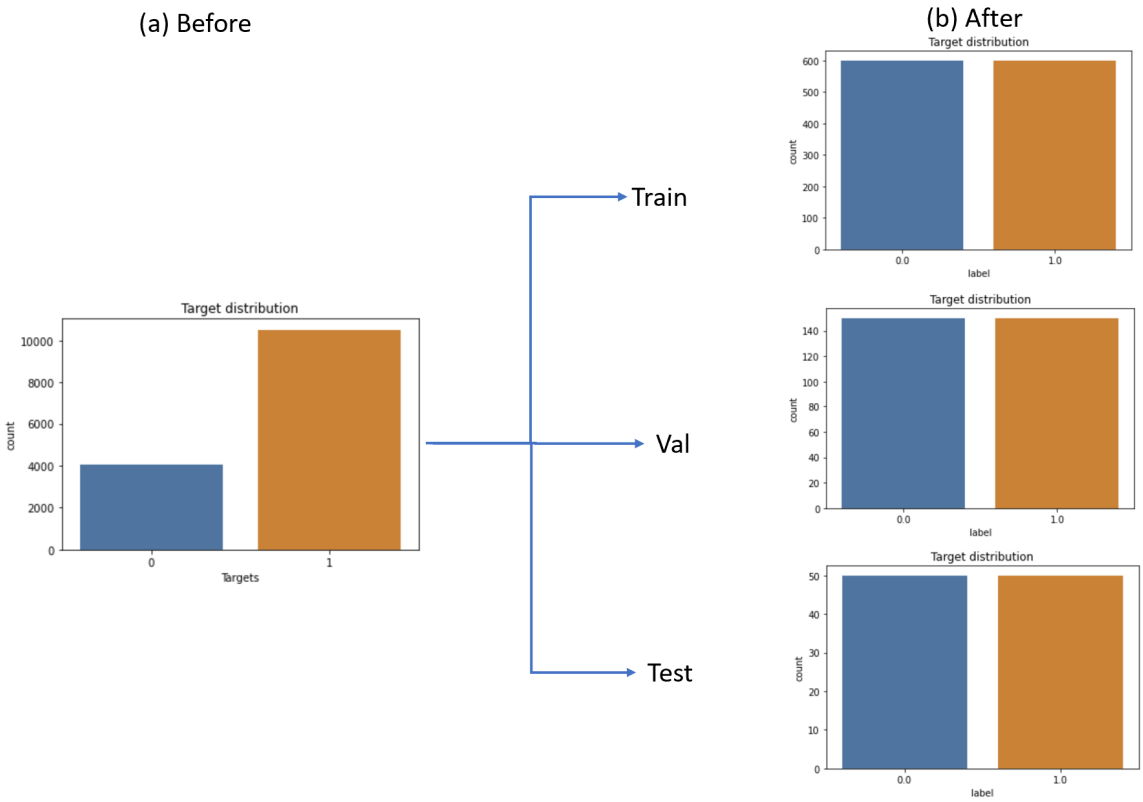


Figure 6.4: PTB: (a) original datasets (b) pre-processed datasets.

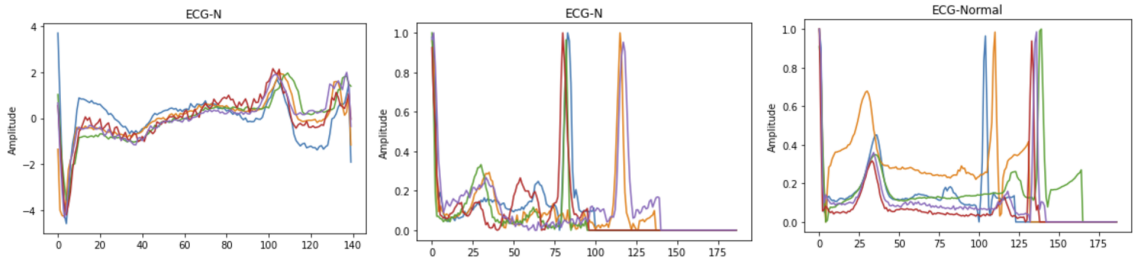
Figure 6.5: PTB: Overview of pre-processing steps for train dataset.

Upon completion of previous steps, the entire balanced train set has been tripled from 200 to 600 per class with augmentation with Gaussian noise of sigma 0.02 and 0.05. The three clients receive the balanced train set with various levels of Gaussian noise added (sigma = 0, 0.02, 0.05) respectively, similar to the process of the ECG5000 dataset.

Chapter 7

FL Results and Analysis

It is essential to highlight the presence of concept shift (described in Section 1.1.2) when comparing the pair PTB and MIT-BIH datasets used for the pre-trained model with the pair ECG5000 and MIT-BIH datasets. The features of PTB are more similar to those of MIT-BIH than the ECG5000 features. An example is illustrated in Figure 7.2 below where the samples of normal ECGs from the 3 datasets are compared.



(a) ECG5000 (b) MITBIH (c) PTB

Figure 7.1: Samples of Normal ECGs from (a)ECG5000, (b)MIT-BIH and (c)PTB

Based on the shapes and amplitude range of values in Figure 7.2, The feature space is similar for MIT-BIH and PTB and dissimilar for ECG5000. This observation is due to the nature of datasets as to the biological meaning of the ECG representations. The issue of concept shift is still deep in research based on Section 1.1.2. Therefore, the results of these 2 datasets are concurrently compared in the next 2 subsections if AALF can help to minimise this open problem as part of learning new classes in anomaly detection.

7.1 Training and Validation Results

Several results from training and validation data are used to capture the learning of the AALF algorithm.

7.1.1 Losses

The losses for every type of training loop for training and validation datatsets have been captured over 100 epochs and plotted below in Figure 7.2.

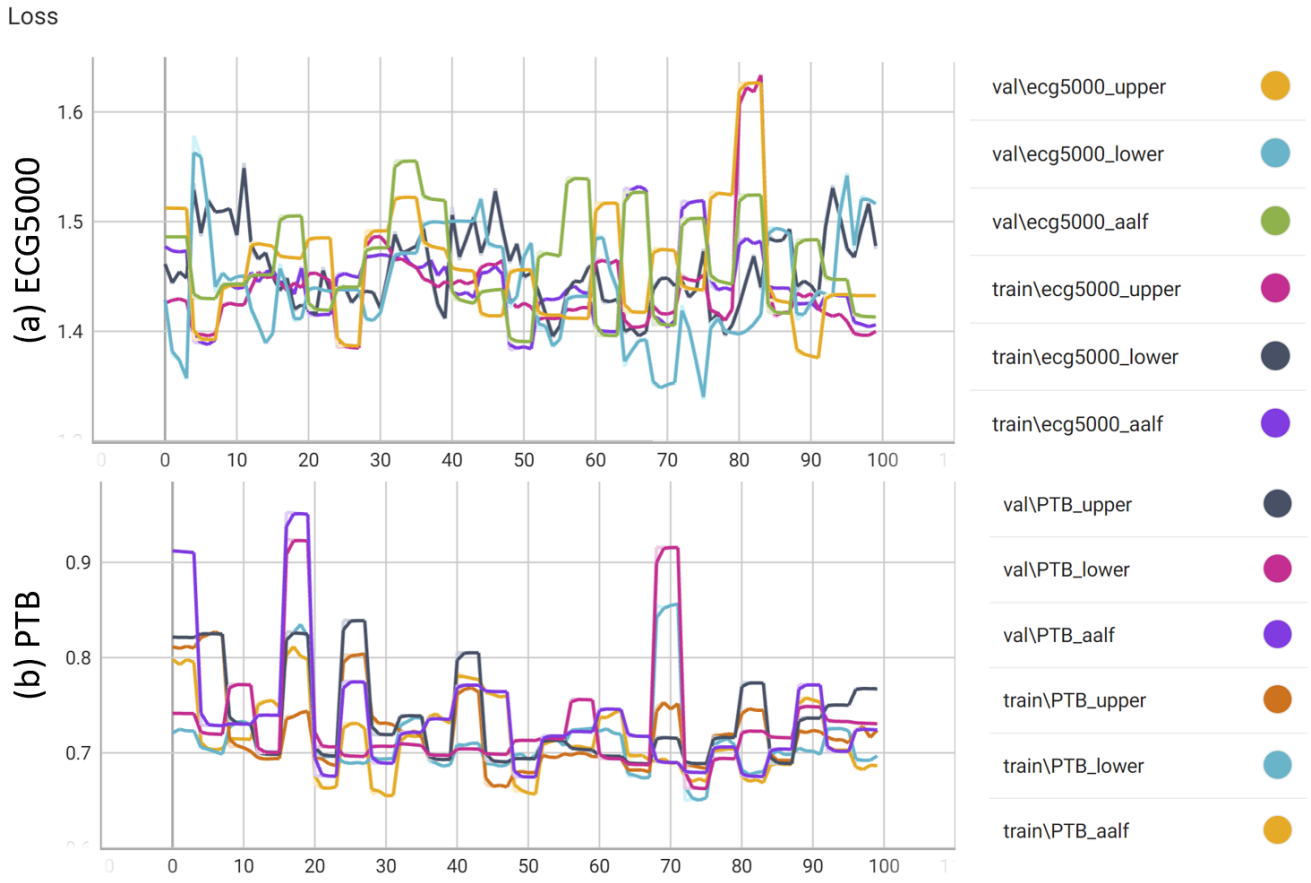


Figure 7.2: Training and validation losses over 100 epochs of 3 types of training loops containing (a)ECG5000 and (b)MIT-BIH

Based on Figure 7.2, all losses for ECG5000 kept fluctuating and the general trend is that the losses remain the same over the epochs. The fluctuations are also observed in all losses for PTB but there is a general decreasing trend observed. These indicate that the models may not have learnt from the data. The slight general decreasing trend seen for PTB is probably due to similar feature space. Despite the negative outlook from the loss plots, it would be interesting to see other aspects of measurements on how the experiments fair.

7.1.2 Metrics

The metrics Precision, Recall and F1 scores for each class and for each type of training loop are plotted in Figures 7.3 and 7.4 respectively.

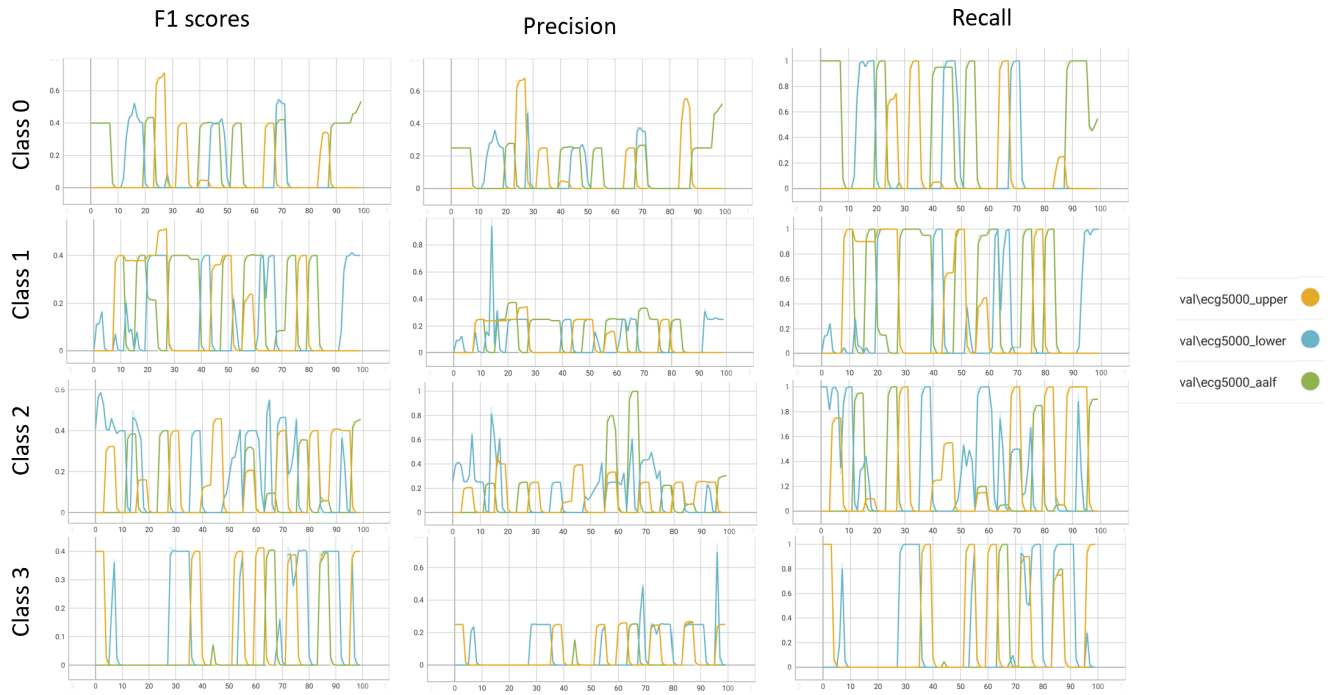


Figure 7.3: Training and validation metric plots for ECG5000. Each row represents a class and each column represents a metric.

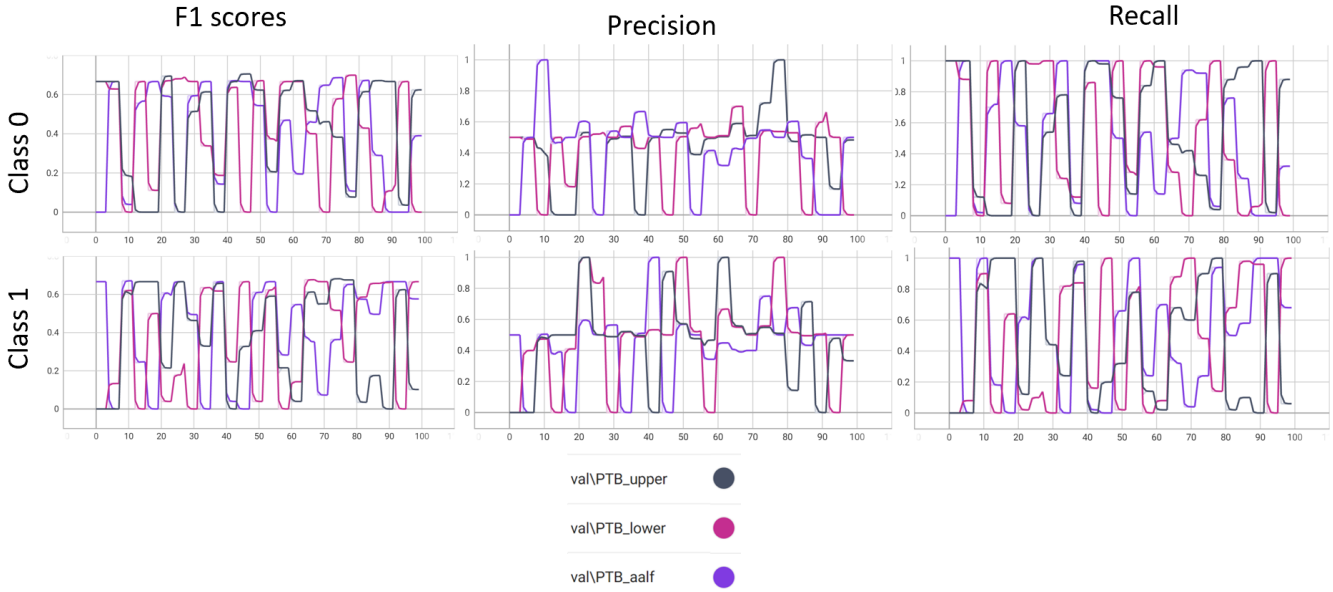


Figure 7.4: Training and validation metric plots for PTB. Each row represents a class and each column represents a metric.

From Figure 7.3, the metrics fluctuate over the epochs for ECG5000. The precision scores are much lower as compared to recall scores. Class 3 seems to be the most difficult to learn as there are more instances at which the metric scores of all FL training loops are zero compared to the rest. Among the 3 types of FL training loops, the lower bound results in the poorest performance in learning with most of the scores being zero while the upper bound seems to have the best performance scores. The performance of the loop with AALF seems to be similar to the upper bound as the number of epochs increases.

This is probably due to the fact that the number of training model layers for the loop with AALF has increased to a maximum towards the end of 100 epochs. Moreover, all the metrics tend to reach zero several times during the fluctuations. This perhaps hints that the attention layers are not able to learn all classes at the same time due to the nature of their purpose to focus on essential layer weights. This is supported by the uncommon observations on having metric scores above zero at the same time.

These patterns are more clearly visible in Figure 7.4 on the PTB dataset that only contains 2 classes. At several epochs, the metric score for one class is closer to zero when the other is further from zero. Between the 2 classes, it is ambiguous to identify the better class for learning. This is intriguing as the samples for the PTB normal class are very similar to one another whereas the samples for PTB abnormal class are different from one another as various types of anomalies have been grouped into this class. Hence, it is much easier to learn the normal class theoretically but the results show no such clear patterns.

Comparing plots for ECG5000 and PTB, the metric scores of ECG5000 tend to reach zero at more instances than the scores of PTB, though there is no clear distinction as to which data the models learnt best from. The source of the issues mentioned can be due to the aggregation of model weights from 3 clients every epoch where the new model weights for the next epoch may not be the perfect learnt model weights from the previous epoch. Hence, the best way to identify learning is to use the results from test set which is discussed in subsequent section 7.2.

7.1.3 Time Taken

The time consumed by the whole process has been recorded to measure its speed.

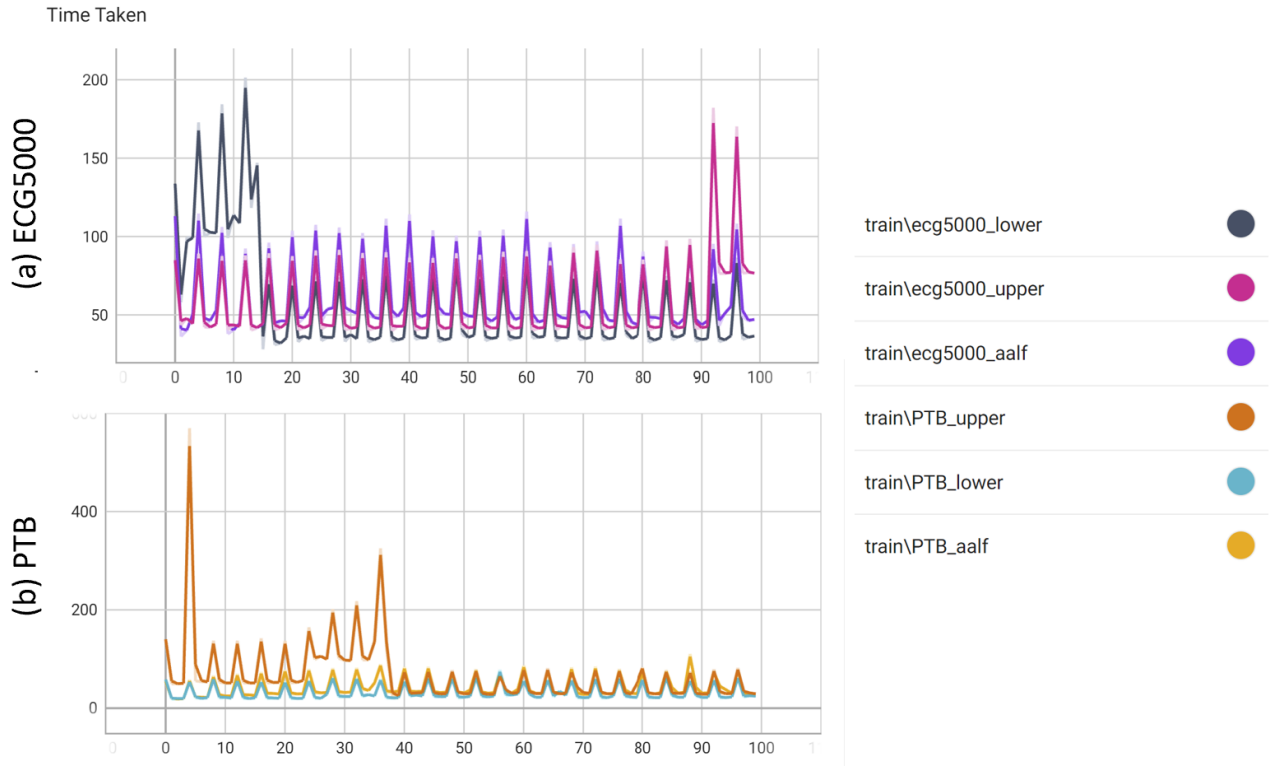


Figure 7.5: Time consumption over the epochs of all FL training loops for (a)ECG5000 and (b) PTB datasets

In Figure 7.5, the abnormally increased time taken at the beginning and end of the training observed are considered to be anomalies and the cause have been identified to be the memory issue in the hardware that lowered the computation capacity at those instances.

The common pattern seen in all plots of FL training loops in Figure 7.5 is the presence of spikes at every 4 epochs. This is due to the post-processing steps at every 4 epochs such as saving model weights which requires the temporary pause of using the built-in FL function. To continue using, the same datasets had to be re-ingested and sent to clients, thus the constant spikes. These spikes are slightly larger for training loops with AALF due to the application of the AALF algorithm.

Among the training loops for ECG5000, the loop with AALF was approximately 10s slower than the upper bound and 20s slower than the lower bound (from 20 to 90 epochs). For PTB, the loop with AALF is approximately 2s slower than the lower bound and has the same speed as the upper bound.

Overall, the AALF algorithm increases the overall computation time but can be considered minimal. It is an efficient and light architecture.

7.1.4 Number of Trainable Attention Layers

The number of trainable attention layers reflects how AALF decides to train layers. The number of trainable attention layers is constant for lower and upper bounds. These can be observed in Figure 7.6 below.

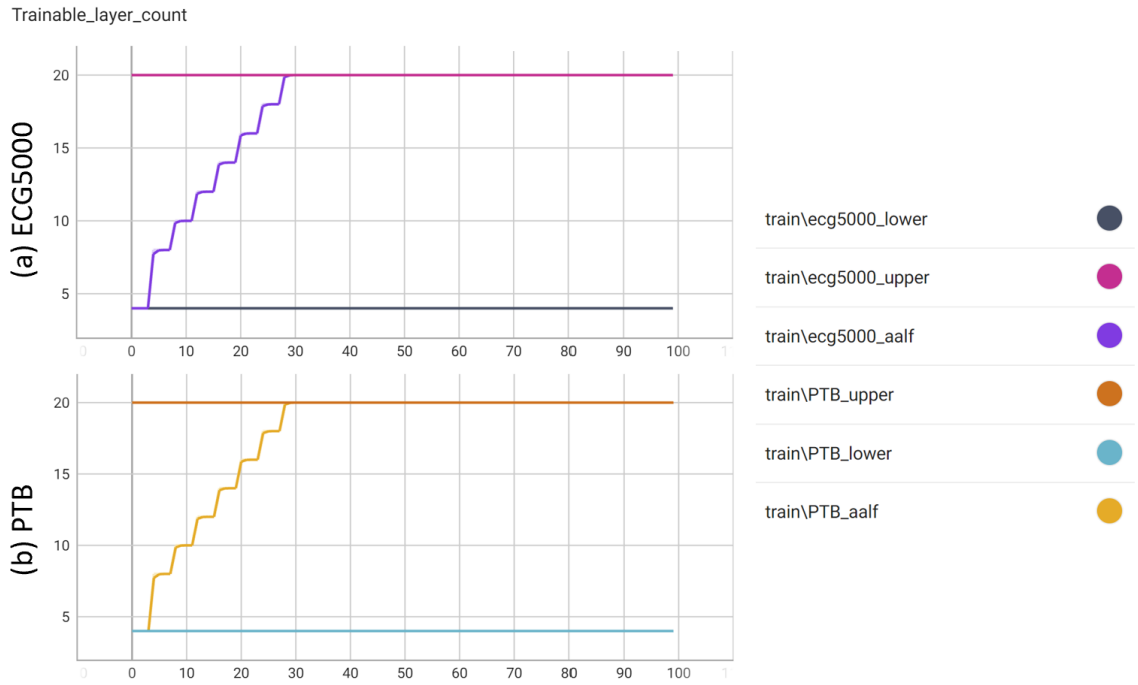


Figure 7.6: Samples of Normal ECGs from (a)ECG5000, (b)MIT-BIH and (c)PTB

Based on the plots in Figure 7.6, the increased rates of the number of trainable layers are the same. Approximately after epoch 30, the FL training loop with AALF behaves exactly like the upper bound as all attention layers are trained. This suggests that the AALF algorithm was not able to decide appropriately and the cause probably originates from the manual input of values for metrics and threshold ranges. In other words, these metrics and threshold values had to be more cautiously selected. This issue is more apparent based on Figure 7.7 below that displays the difference of mean f1 scores (difference of mean f1 scores between current and previous epoch) of all classes over the epochs.

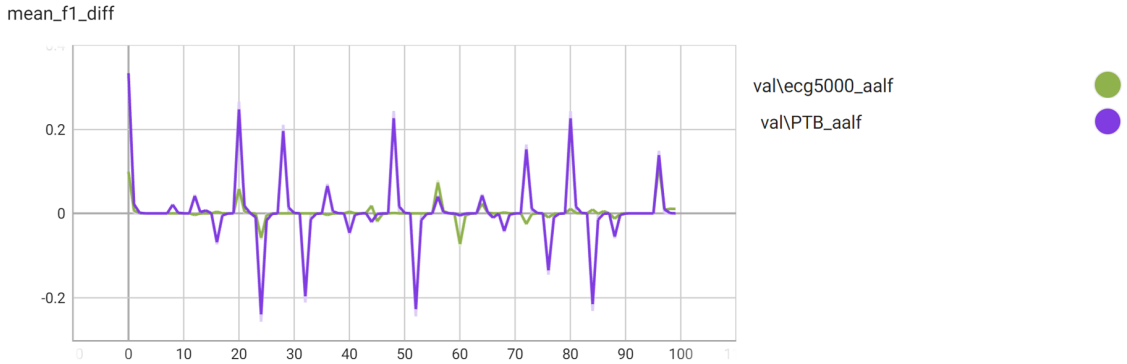


Figure 7.7: Samples of Normal ECGs from (a)ECG5000, (b)MIT-BIH and (c)PTB

The mean f1 score difference is used to determine if more permissible layers (step 1 in

Section 3.3) are required in the AALF algorithm (concerning section 3.3 in Chapter 3). From Figure 7.7, it is evident that the mean f1 score differences are different for both PTB and ECG5000 datasets. The PTB dataset has higher mean f1 score differences as compared to ECG5000. All exceed the current threshold of 0.03 which means the number of permissible layers will always increase. Based on the non-improving F1 scores observed in Figures 7.3 and 7.4, all the permissible layers are then trained probably due to drastic changes in the model layer parameters, exceeding the coverage metric threshold (step 2 in Section 3.3) which in turn results in the graphs in Figure 7.7.

It must be recalled that the AALF algorithm is only applied every 4 epochs and thus there are temporary moments for the models to learn. Perhaps, the interval of 4 epochs could be too low for learning.

Another interesting observation is that the values fluctuated from positive to negative and vice versa. This could also suggest that training attention layers can only help to shift focus but not learn all classes at the same time.

7.1.5 Attention Layer Weights

In the model from the central server, there are 4 SE Nets (attention network) with each containing 2 linear layers. Hence, there are a total of 8 linear layers for the attention network. Additional, the model contains 2 linear layers as fully-connected layers. Combining the attention network and fully connected layers, the weight distributions of the 10 linear layers have been captured. Figures 10.1 to 10.10 in Appendix A illustrate the kernel weight distributions of the 10 linear layers over epochs and Figures 10.11 to 10.20 in Appendix B illustrate the corresponding bias weight distributions of the 10 linear layers over epochs. Figures 10.1 to 10.8 and 10.11 to 10.18 exclude attention layer weight distributions as they are not trained.

Do note that Figures 10.9, 10.10, 10.19 and 10.20 show the most changes as they belong to fully-connected layers that are trained throughout the epochs. Hence, their distributions may differ from those of the attention network where the focus lies.

Based on the kernel weight distributions from Figures 10.1 to 10.8, peaks are observed for the upper bound and loop with AALF with the range of values being small. No distinct patterns, however, are observed in the changes over the epochs. Between the loop with AALF and the upper bound, the distribution of bias weights in the former consistently has a single peak per epoch while many small peaks are mostly observed in the latter. Sometimes, the latter does contain single peaks per epoch.

Despite the differences in peaks, they do not reflect on how the learning has progressed over the epochs for individual classes. This approach intends to merely spot possible patterns in the FL environment.

7.2 Test Results

The results of test sets will be based on the final metrics such as Recall, Precision and F1 scores and visual aspects such as the confusion matrices.

7.2.1 Metrics

The metrics comprising of precision, recall and F1 scores are tabulated below for every class and overall in Figures 7.8 and 7.9 for ECG5000 and PTB respectively.

ecg5000upper:

	precision	recall	f1-score	support
N	0.00	0.00	0.00	10
R-on-T	0.00	0.00	0.00	10
PVC	0.00	0.00	0.00	10
SP	0.26	1.00	0.41	10
accuracy			0.25	40
macro avg	0.06	0.25	0.10	40
weighted avg	0.06	0.25	0.10	40

(a)

ecg5000lower:

	precision	recall	f1-score	support
N	0.00	0.00	0.00	10
R-on-T	0.00	0.00	0.00	10
PVC	0.25	1.00	0.40	10
SP	0.00	0.00	0.00	10
accuracy			0.25	40
macro avg	0.06	0.25	0.10	40
weighted avg	0.06	0.25	0.10	40

(b)

ecg5000aalf:

	precision	recall	f1-score	support
N	0.56	0.50	0.53	10
R-on-T	0.00	0.00	0.00	10
PVC	0.29	0.90	0.44	10
SP	0.00	0.00	0.00	10
accuracy			0.35	40
macro avg	0.21	0.35	0.24	40
weighted avg	0.21	0.35	0.24	40

(c)

Figure 7.8: ECG5000 metric table from (a)upper bound, (b)lower bound and (c)loop with AALF

Based on Figure 7.8, the model from the upper bound in (a) is only able to learn and predict the SP class with a perfect recall of 1 and low precision of 0.26. On the other hand, the model from lower bound i (b) is only able to learn and predict the PVC class with a perfect recall of 1 and low precision of 0.25. For both cases, there seems to be over-fitting to one particular class as the models are not able to learn from other classes. The models learnt features from a single class but are not able to distinguish from other classes which resulted in high recall but low precision.

The model from the loop with AALF in (c) can learn Normal class with a recall of 0.5 and moderate precision of 0.56 and PVC with a moderate-high of 0.9 and low precision of 0.29. The model from the latter seems to have performed better on the ECG5000 test set as its weighted F1 score is higher as compared to both upper and lower bounds.

ptbupper:

	precision	recall	f1-score	support
Normal	0.53	0.94	0.68	50
Abnormal	0.73	0.16	0.26	50
accuracy			0.55	100
macro avg	0.63	0.55	0.47	100
weighted avg	0.63	0.55	0.47	100

(a)

ptblower:

	precision	recall	f1-score	support
Normal	0.00	0.00	0.00	50
Abnormal	0.50	1.00	0.67	50
accuracy			0.50	100
macro avg	0.25	0.50	0.33	100
weighted avg	0.25	0.50	0.33	100

(b)

ptbaalf:

	precision	recall	f1-score	support
Normal	0.59	0.38	0.46	50
Abnormal	0.54	0.74	0.63	50
accuracy			0.56	100
macro avg	0.57	0.56	0.55	100
weighted avg	0.57	0.56	0.55	100

(c)

Figure 7.9: PTB metric table from (a)upper bound, (b)lower bound and (c)loop with AALF

Based on Figure 7.9, the model from the upper bound in (a) can predict both normal and abnormal classes. For the normal class, the recall is very high and precision is moderate. For abnormal classes, the precision is high and the recall is low. The difference between (a) and (b) is perhaps because the normal features are more consistent than the abnormal ones which have a wider range of unique patterns. Thus, it is easier for the model to learn more normal features than the abnormal. The model from the lower bound in (b), however, performed weakest as it is only able to predict abnormal class with high recall and moderate precision. No predictions for the normal class were observed. This is expected as the model from (a) has all attention layers trained compared to (b).

The precision-recall values of the model from the loop with AALF in (c) are more balanced but of moderate level. The model seems to have performed better on the PTB test set as its weighted F1 score is higher as compared to both upper and lower bounds.

7.2.2 Confusion Matrices

The results from metric table from the previous subsection can also be reflected via confusion matrices in Figures 7.10 and 7.11 below.

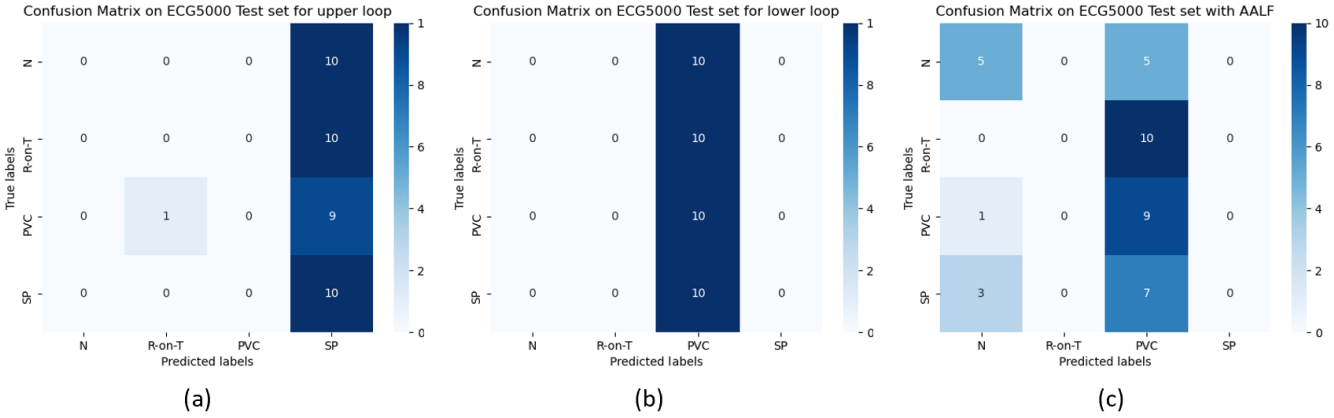


Figure 7.10: ECG5000 confusion matrix from (a)upper bound, (b)lower bound and (c)loop with AALF

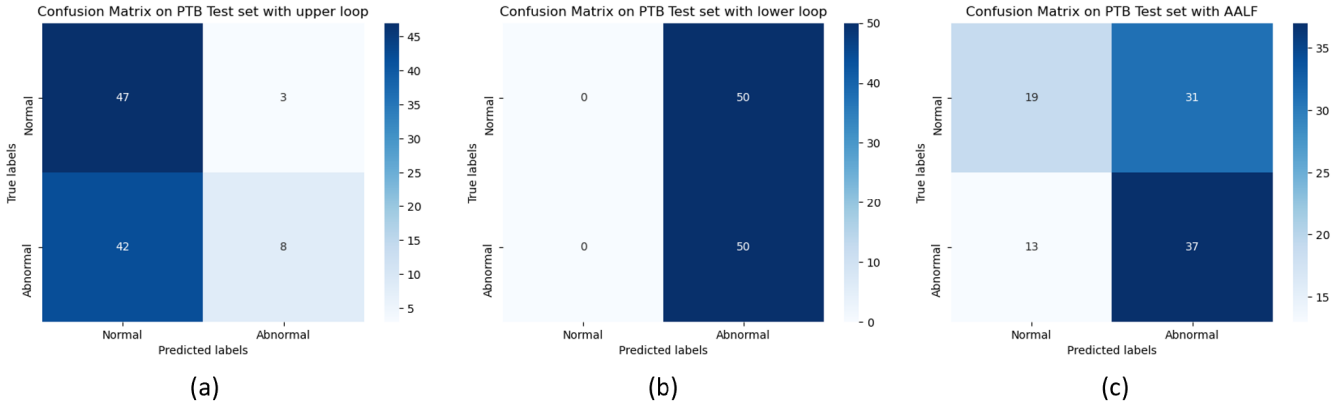


Figure 7.11: PTB confusion matrix from (a)upper bound, (b)lower bound and (c)loop with AALF

Ideally, we only want the diagonally placed boxes to be in dark blue from top left to bottom right. From all the confusion matrices above, no such diagonal observation is captured which translates to poor performances for all models. In each of the upper and lower bounds for PTB and ECG5000 test sets, only one vertical dark blue column is observed which signifies that the respective models predict all data as the same class. On the contrary to the lower and upper bounds, the loops with AALF result in more than 1 vertical dark blue column.

Based on the metric tables and confusion matrices, the models from the loop with AALF seemed to have performed slightly better than the rest on the test sets. However, these results may not hold if the model learning has been fluctuating over the epochs as suggested from Figure 7.7 and attention weights.

7.3 Summary of Experiments

Overall, the experiments have shown that the AALF algorithm is not able to adequately learn and classify all ECG signal classes by shifting the focus of attention networks though the AALF have presented better results as compared to the 2 baselines. The experiments do show that having AALF is a light overhead architecture that does not adversely affect the speed of model training. Moreover, the experimentation results with standard mean f1 scores and coverage metric scores for layer freezing is insufficient to conclude that the approach to determine optimum layers for training is incorrect or correct.

Chapter 8

Limitations and Recommendations

8.1 Limitations

8.1.1 Stability Threshold Range and Metrics in AALF

Currently, the stability threshold range and coverage metric values are manually selected. These values may not be the optimum values. Furthermore, the values have been constant for both datasets which in reality may not be true. The learning capacity of the model layers is different for both datasets. PTB only has 2 classes while ECG5000 contains 4 classes in this project. Hence, the same model has to learn to differentiate more classes for ECG5000 as compared to PTB with the same number of parameters.

In addition, the presence of concept shift has to be considered in FL where the features to learn in PTB are similar to those of the MIT-BIH dataset, unlike ECG5000. These reasons favour the need for higher learning capacity for the model to learn from ECG500. This would translate to different coverage metric values required to freeze or unfreeze model layers and the stability of parameters in terms of constant oscillation may be of different range as well. Therefore, the algorithm may require different needs based on the model and datasets.

These values can be altered with hyperparameter tuning to identify the right values or can be further experimented with dynamic modification while training.

8.1.2 More Generalized Pre-Trained Model

We can have a more generalized pre-trained model that is trained on numerous datasets for FL application. In this way, there is a higher probability of capturing more feature spaces while learning new classes due to rich information from transfer learning. Currently, transfer learning is solely reliable on learning from 1 type of data.

8.1.3 Limited Learning and Applicability from Attention Layers

As observed from the FL experimental results in the previous chapter, training attention layers alone provide very limited learning. This has a great negative impact on the use of AALF as it solely depends on attention networks. Shifting the focus of attention does not necessarily lead to appropriate learning. Hence, the algorithm itself is a limitation inherently.

In addition, the AALF is highly restrictive as such a technique is best to experiment within the same domain such as ECG here in the FL environment. Hence, the algorithm itself is a limitation inherently due to limited learning and restrictive applicability.

Having said that, it is essential to note that SE Nets have been used as attention layers here. Thus, further experiments can be conducted to test the learning of attention with other types of attention networks such as self-attention networks which is applied to each neuron instead of channels.

8.1.4 Location of AALF Implementation

Currently, the AALF has been implemented in the central server. Hence, the model weights had to be retrieved and aggregated periodically from the clients which increases the computation and communication cost.

In this project, it was discovered that it was not easy to implement the AALF algorithm in the clients within the time constraint. There were many source scripts of TFF that required modifications. In addition, the model has been serialized when the central server sends it to clients. Hence, only layers can be accessed in serialized models and are not able to modify the training status of the model layers best to my knowledge. There may have been a better way to perform the above.

8.1.5 Data Duplication in FL and Client Limitation

For every client, the dataset used to be sent for communication will be duplicated. Here, the dataset has been duplicated thrice as there are 3 clients. This takes up a huge amount of memory when training in an FL environment. This worsens the situation when the computing capacity of hardware is limited. Hence, only a few clients could be selected.

It is also essential to note that each of the 3 clients was given the same dataset with varying noise addition to mimic real-world scenarios. Eventually, the aggregation of model weights heavily rely on one model per type of dataset with noise. Thus, it is more prone to model weights that are outliers.

Having more clients means more model weights per dataset when aggregating. This, in turn, increases the reliability of learning from the model weights as it will be less prone to learning from model weights that are outliers.

8.2 Recommendations

8.2.1 Hyper Parameter Tuning in FL

Hyper-parameter tuning was done when obtaining the pre-trained model but not for the FL training due to the hardware memory capacity. Hyper-parameter tuning for FL can be developed to obtain the best possible optimised model. Furthermore, more parameters such as the coverage metrics and stability thresholds mentioned in Section 7.1.1 can be added as Hyper-parameters and determine the best range. This can help to understand the model learning at the granular level. An example will be to understand if the stability thresholds vary for every model layer to maintain the same coverage metric since different layers have different learning capabilities.

8.2.2 Tracking of Clients

Currently, only the model layers in the central server are tracked since that is the location of AALF applied. Tracking the clients can additionally provide valuable information as to which datasets in the clients led to similar and varying results. Concerning the approach taken in this project, we can identify the impact of the noise level on the outcome by tracking their metrics individually and their learning capabilities by tracking their model layers' weight distributions.

8.2.3 Extension of AALF to All Layers

The results observed indicate that the learning from attention networks are limited and insufficient to deal with datasets in the FL environment. Despite the negative outlook, it is possible to modify AALF and extend its application to all model layers. This extension helps to minimise/eliminate the limited learning capabilities of the attention networks. Though the computation time increases, it is worth further research and experiments to continue to answer the question of "is training a single global model even the right goal?" that authors of [12] have posed. This translates to a partial continuation of this project that focuses on other techniques of having a generalized model in the FL environment.

Chapter 9

Conclusion

This is an ambitious project which leverages the narrowed healthcare topic of ECG to help redefine generalised and local models while enhancing the efficiency of the FL training process. This is aimed to be achieved via the implementation of a custom attention layer freezing technique with attention networks (SE-Nets). Thus, the project hypothesizes that controlling the freezing of attention layers independently can help to optimize the FL training process and learn new data by shifting the focus of existing features. This ideally allows a generalized model for FL to be formed as it is sufficient to train the attention layers to learn the client's data. This translates to less number of model layers trained to lead to better training efficiency.

To achieve the results, the project first builds a pre-trained model (base model + SE-Nets) trained on the MIT-BIH dataset with an accuracy of 91% on the test set. Transfer learning is then applied to the model during FL training. The model is sent to 3 clients to be trained on ECG5000 and PTB datasets respectively using the AALF algorithm. The AALF algorithm progressively and dynamically unfreezes the attention layers with mean F1-score and coverage metric respectively. The model's weighted f1 score is 0.24 on the ECG5000 test set and 0.55 on the PTB test set.

Both results are not as expected as the model is not successful in learning all of the new features by shifting the focus of attention networks. Integrating the AALF did not reduce training time significantly which partially fulfils the efficacy of the FL training process. There is, however, insufficient evidence to outline that the standard metric values and thresholds in AALF have optimized the layer freezing technique which in return optimizes training. Therefore, the results are inconclusive as further experimentation is required.

Considering the project results only, the hypothesis is rejected. Despite the unsuccessful outcome, the results are insightful in understanding how the FL training process has occurred in the TFF environment and understanding the workings and limitations of the AALF algorithm. Such includes that the training attention networks solely seem to cause oscillations in the model learning in the FL environment. These can help to solidify the application of AALF or any similar of its kind for future hypothesis testing.

References

[1] M. Sampson and A. McGrath, “Understanding the ECG. Part 1: Anatomy and physiology,” British Journal of Cardiac Nursing, vol. 10, no. 11, pp. 548–554, Nov. 2015, doi: 10.12968/bjca.2015.10.11.548.

[2] E. Hockstad, “2020 American Heart Association Virtual Cardiac Symposium,” p. 65.

[3] G. Pang, C. Shen, L. Cao, and A. van den Hengel, “Deep Learning for Anomaly Detection: A Review,” ACM Comput. Surv., vol. 54, no. 2, pp. 1–38, Apr. 2021, doi: 10.1145/3439950.

[4] S. Thudumu, P. Branch, J. Jin, and J. Singh, “A comprehensive survey of anomaly detection techniques for high dimensional big data,” J Big Data, vol. 7, no. 1, p. 42, Dec. 2020, doi: 10.1186/s40537-020-00320-x.

[5] G. Michau and O. Fink, “Unsupervised transfer learning for anomaly detection: Application to complementary operating condition transfer,” Knowledge-Based Systems, vol. 216, p. 106816, Mar. 2021, doi: 10.1016/j.knosys.2021.106816.

[6] D. Preuveneers, V. Rimmer, I. Tsingenopoulos, J. Spooren, W. Joosen, and E. Ilie-Zudor, “Chained Anomaly Detection Models for Federated Learning: An Intrusion Detection Case Study,” Applied Sciences, vol. 8, no. 12, p. 2663, Dec. 2018, doi: 10.3390/app8122663.

[7] C. Renaud, ”Designing for Edge and IoT Success,” Commisioned by Dell Technologies, 2020.

[8] 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, Jun. 2021, doi: 10.1016/j.hcc.2021.100008.

[9] I. Kholod et al., “Open-Source Federated Learning Frameworks for IoT: A Comparative Review and Analysis,” Sensors, vol. 21, no. 1, p. 167, Dec. 2020, doi: 10.3390/s21010167.

[10] T.-M. H. Hsu, H. Qi, and M. Brown, “Federated Visual Classification with Real-World Data Distribution,” in Computer Vision – ECCV 2020, vol. 12355, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds. Cham: Springer International Publishing, 2020, pp. 76–92. doi: 10.1007/978-3-030-58607-2_5.

[11] W. Y. B. Lim et al., “Federated Learning in Mobile Edge Networks: A Comprehensive Survey,” IEEE Commun. Surv. Tutorials, vol. 22, no. 3, pp. 2031–2063, 2020, doi: 10.1109/COMST.2020.2986024.

[12] P. Kairouz et al., “Advances and Open Problems in Federated Learning,” p. 122.

[13] T. S. Ananda et al., ”SCAFFOLD: Stochastic Controlled Averaging for Federated Learning,” 2021.

[14] J. Hamer, M. Mohri, and A. T. Suresh, “FedBoost: Communication-Efficient Algorithms for Federated Learning,” p. 11.

[15] A. Raza, K. P. Tran, and L. Koehl, “Designing ECG Monitoring Healthcare System with Federated Transfer Learning and Explainable AI,” p. 16.

[16] C. Chen et al., “Communication-Efficient Federated Learning with Adaptive Parameter Freezing,” p. 11.

[17] Y. Guo, Y. Li, L. Wang, and T. Rosing, “AdaFilter: Adaptive Filter Fine-tuning for Deep Transfer Learning,” arXiv:1911.09659 [cs], Dec. 2019, Accessed: Oct. 25, 2021. [Online]. Available: http://arxiv.org/abs/1911.09659

[18] S. A. S. V. Yuhan Liu, ”AutoFreeze: Automatically Freezing Model Blocks to Accelerate Fine-tuning,” 2021.

[19] B. Kim, T. Kim, and Y. Choe, “Bayesian Optimization Based Efficient Layer Sharing for Incremental Learning,” Applied Sciences, vol. 11, no. 5, p. 2171, Mar. 2021, doi: 10.3390/app11052171.

[20] Y. Guo, H. Shi, A. Kumar, K. Grauman, T. Rosing, and R. Feris, “SpotTune: Transfer Learning through Adaptive Fine-tuning,” arXiv:1811.08737 [cs, stat], Nov. 2018, Accessed: Oct. 25, 2021. [Online]. Available: http://arxiv.org/abs/1811.08737

[21] L. F. Isikdogan, B. V. Nayak, C.-T. Wu, J. P. Moreira, S. Rao, and G. Michael, “SemifreddoNets: Partially Frozen Neural Networks for Efficient Computer Vision Systems,” in Computer Vision – ECCV 2020, vol. 12372, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds. Cham: Springer International Publishing, 2020, pp. 193–208. doi: 10.1007/978-3-030-58583-9_12.

[22] Shayan Fazeli, “ECG Heartbeat Categorization Dataset — Kaggle,
” Kaggle. https://www.kaggle.com/shayanfazeli/heartbeat (accessed Nov. 23, 2021).

[23] E. K. Y. Chen, “Time Series Classification Website.”
http://www.timeseriesclassification.com/description.php?Dataset=ECG5000 (accessed Nov. 23, 2021).

[24] P. Matias, D. Folgado, H. Gamboa, and A. Carreiro, “Robust Anomaly Detection in Time Series through Variational AutoEncoders and a Local Similarity Score:,” in Proceedings of the 14th International Joint Conference on Biomedical Engineering Systems and Technologies, Online Streaming, — Select a Country —, 2021, pp. 91–102. doi: 10.5220/0010320500910102.

[25] “python - How to code a residual block using two layers of a basic CNN algorithm built with ‘tensorflow.keras’? - Stack Overflow.” https://stackoverflow.com/questions/64792460/how-to-code-a-residual-block-using-two-layers-of-a-basic-cnn-algorithm-built-wit (accessed Nov. 29, 2021).

[26] titu1994, “GitHub - titu1994/keras-squeeze-excite-network: Implementation of Squeeze and Excitation Networks in Keras,” GitHub. https://github.com/titu1994/keras-squeeze-excite-network (accessed Nov. 29, 2021).

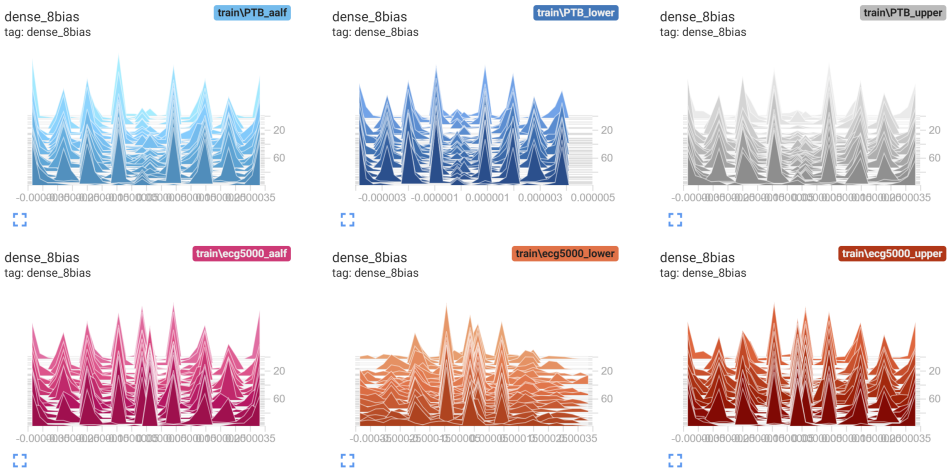


Figure 10.19: Bias 8 Weight Distribution

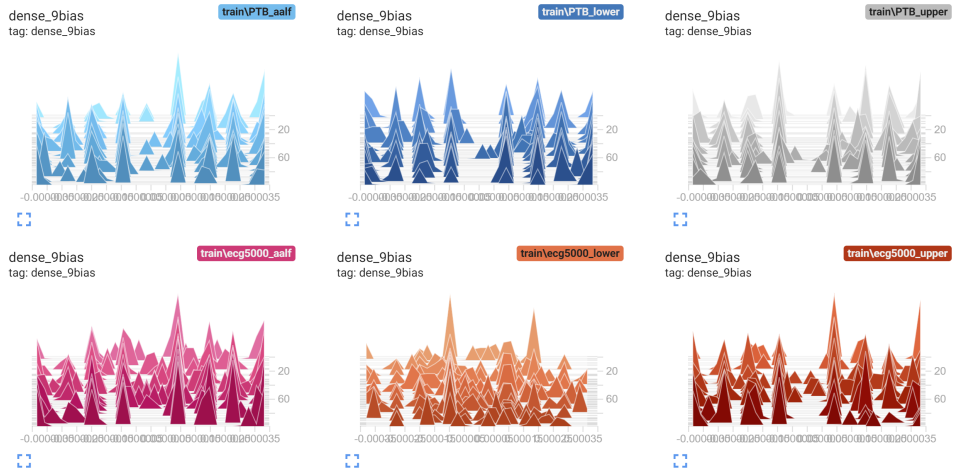


Figure 10.20: Bias 9 Weight Distribution