CHALMERS
UNIVERSITY OF TECHNOLOGY

Mitigating Label Noise in ECG Data: A comparative Analysis

Master's thesis in Biomedical Engineering

MANOURAS MANOUSOS
PEDIADITIS DIMITRIOS

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Master's thesis 2025

Mitigating Label Noise in ECG Data: A comparative Analysis

MANOUSOS MANOURAS
DIMITRIOS PEDIADITIS

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Signal Processing and Biomedical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2025

Mitigating Label Noise in ECG Data: A comparative Analysis
MANOURAS MANOUSOS
PEDIADITIS DIMITRIOS

Supervisor: Elias Stenhede
Examiner: Xuezhi Zeng

Master's Thesis 2025
Department of Electrical Engineering
Division of Signal Processing and Biomedical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A close-up of an electrocardiogram (ECG) printout showing heart activity `https://www.pexels.com/photo/wavelength-1093161/`.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2025

Mitigating Label Noise in ECG Data: A comparative Analysis
MANOURAS MANOUSOS
PEDIADITIS DIMITRIOS
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Label noise in electrocardiogram (ECG) datasets, where samples are incorrectly labelled, significantly hinders the performance of machine learning models by fitting to the incorrect labels. This type of noise can arise from several factors, such as human error, inter-expert variability, or obsolete automated annotation algorithms, leading to inconsistencies within dataset labelling. In this thesis work, three noise mitigation methods are compared with a baseline model to evaluate both the impact of label noise and the effectiveness of these mitigation strategies in ECG datasets. The mitigation methods chosen are Stochastic co-teaching, Self-learning and DivideMix. Class-dependent label noise was synthetically introduced into two ECG datasets, PTB-XL and CODE15%, comprising of symmetric and asymmetric noise types with rates of 20% and 40%. The best-performing method, as quantified by the AUROC score, was self-learning, with improvements from 4 to 8% over the baseline in CODE15% and from 8 to 12% in PTB-XL. DivideMix demonstrated reduced performance, presumably because it has been optimised for specific image datasets. Stochastic Co-teaching achieved better results on the CODE15% dataset, likely due to the larger sample size of this dataset. Furthermore, an additional ECG dataset obtained from Akershus University Hospital was used to assess the generalisability of the best-performing method under unknown noise conditions. The results did not show an improvement over the baseline model, indicating a strong dependency between the characteristics of the dataset and the effectiveness of noise mitigation strategies.

Keywords: Deep Learning, Label Noise, AI, ECG, Neural Networks, Time-series.

Acknowledgements

We would like to express our gratitude to all those who helped and supported us throughout this thesis work.
Specifically, special thanks to our supervisor, Elias Stenhede, who helped and guided us with his constructive feedback and advice and our examiner and program coordinator, Xuezhi Zeng, for her guidance and support.
Finally, we would like to thank our families and friends for motivating us throughout this whole process.

AI Use Disclaimer

We take full responsibility for the content of this thesis work and we are able to justify all the choices made. With that being said, AI was strictly used for correcting spelling, grammar and syntax mistakes.

Manousos Manouras,
Dimitrios Pediaditis,
Gothenburg, May 2025

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis, listed in alphabetical order:

1dAVb	First-degree Atrioventricular Block
AF	Atrial Fibrillation
AI	Artificial Intelligence
AUROC	Area Under the Receiver Operating Characteristic Curve
BN	Batch Normalization
CD	Conduction Disturbance
CNN	Convolutional Neural Network
ECG	Electrocardiogram
FC	Fully Connected
FPR	False Positive Rate
GMM	Gaussian Mixture Model
GRU	Gated Recurrent Unit
HYP	Hypertrophy
ICD	International Classification of Diseases
KL	Kullback-Leibler
LBBB	Left bundle branch block
LSTM	Long Short-Term Memory
MI	Myocardial Infarction
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
PCA	Principal Component Analysis
PNP	Probabilistic Noise Prediction
ReLU	REctified Linear Unit
RBBB	Right bundle branch block
RNN	Recurrent Neural Networks
ROC	Receiver Operating Characteristic
ResNet	Residual Network
SB	Sinus Bradycardia
SSL	Semi Supervised Learning
ST	Sinus Tachycardia
STTC	ST/T change
TPR	True Positive Rate

Nomenclature

Below is the nomenclature of indices, sets, parameters, and variables that have been used throughout this thesis.

Indices

i, j	Indices for samples or classes
t	Index for training epoch
k	Index for classes

Sets

D	Complete training dataset
$\hat{D}_{\text{clean}}$	Subset of clean samples in training data
$\tilde{D}$	Noisy dataset
C	Set of class labels
U	Unlabelled sample set
X	Labelled sample set

Parameters

α	Weight factor in loss for label correction
γ	Penalty or regularisation coefficient
T	Temperature used in label sharpening
λ	Mixup interpolation coefficient
λ_u	Weight of unlabelled loss
λ_r	Weight of regularisation term
η	Learning rate
N	Total number of samples or training iterations
r	Noise rate
T_{ij}	Transition probability from class i to class j
w_b	Clean probability weighting from GMM

Variables

x_i	Input ECG signal for sample i
y_i	True label for sample i
$\hat{y}_i$	Predicted label for sample i
$\tilde{y}_i$	Noisy label for sample i
$\bar{y}_i$	Corrected label for sample i
$f(x)$	Model output for input x
$f(x;\theta)$	Prediction function for input x with parameters θ
L	Loss function
θ	Model parameters
z	Logits (raw model output before softmax)
p	Class probability after softmax
$\cos(A, B)$	Cosine similarity between feature vectors A and B
L_X	Loss function for labelled set
L_U	Loss function for unlabelled set
L_{reg}	Kullback-Leibler regularisation term
$p(y/x)$	probability for class y given input x

Contents

List of Figures

List of Tables

1

Introduction

This chapter offers a thorough summary of the conceptual foundation and motivation for this thesis. An overview of the Electrocardiogram (ECG), its clinical importance, and standard practices for recording cardiac signals is given at the beginning. The discussion then shifts to how deep learning is revolutionising healthcare, specifically the processing of biomedical signals such as ECG. The use of deep learning methods in ECG interpretation is then introduced, offering new developments and challenges. The problem of label noise, a significant challenge in medical data analysis, is addressed, with an emphasis on how it affects ECG classification tasks. After a survey of previous work on this issue, the chapter concludes with a description of the datasets employed in the study and a summary of the motivation, objectives, and methodological decisions that guided this work.

1.1 Electrocardiography

The ECG is an essential diagnostic tool that records the heart's electrical activity over time using electrodes placed on the skin. It has a wide usage in clinical settings due to its low cost, non-invasive and rapid nature, allowing the identification and monitoring of a wide range of cardiovascular conditions, as seen in Figure 1.1, including myocardial infarction, arrhythmias and conduction disturbances [1]. The detailed information conveyed in the ECG signal is clinically significant and serves as the basis for quantitative analysis during clinical evaluation, thereby cementing its role in routine cardiac examination [2].

A standard 12-lead ECG is based on an arrangement of ten electrodes in a formation to obtain a complete impression of the electrical activity of the heart. The setup has four limb electrodes (placed on the right arm, left arm, right leg, and left leg) and six precordial electrodes (V1–V6) located on the chest, Figure 1.2. The limb leads are combined to yield six frontal plane leads (I, II, III, aVR, aVL, and aVF), and the precordial leads generate six horizontal plane leads. This configuration facilitates single monitoring of different heart regions, namely, the inferior (leads II, III, aVF), lateral (leads I, aVL, V5, V6), and anterior (leads V1–V4) walls, with accurate signal capture and subsequent analysis [3], Figure 1.3.

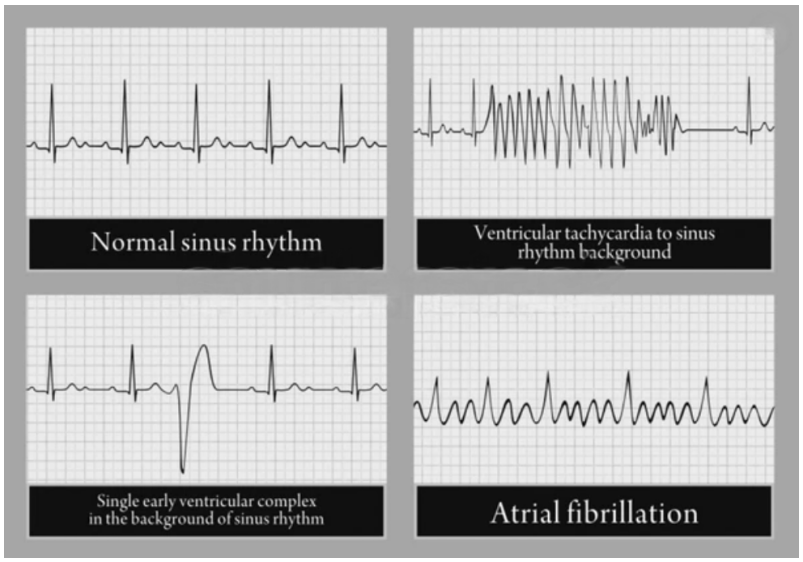


Figure 1.1: Representative ECG waveforms showing four rhythm types. Top left: Normal sinus rhythm with regular P-QRS-T cycles. Top right: Ventricular tachycardia transitioning to sinus rhythm, characterised by rapid, wide QRS complexes. Bottom left: A single premature ventricular complex interrupts an otherwise normal sinus rhythm. Bottom right: Atrial fibrillation with irregular rhythm and absent P waves. [1]

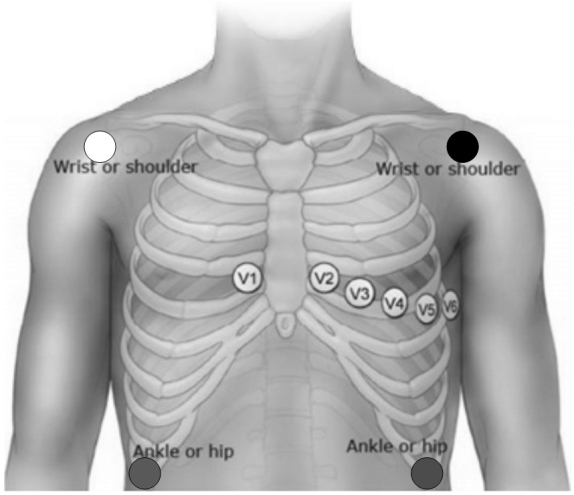


Figure 1.2: Standard precordial (V1–V6) and limb (RA, LA, RL, LL) electrode placement used in a 12-lead ECG recording. [2]

[1] https://www.shutterstock.com/image-vector/examples-normal-abnormal-ecg-vector-professional-86686120

[2] https://clinical.stjohnwa.com.au/clinical-skills/assessment/vital-signs/electrocardiography-%28ecg%29

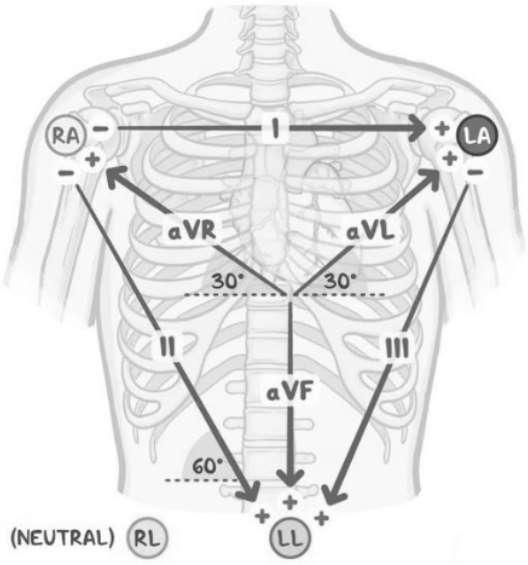


Figure 1.3: Schematic of the 12-lead ECG axis system showing the direction, angles and orientation of the six limb leads (I, II, III, aVR, aVL, aVF) derived from the four limb electrodes. [3]

1.2 Deep learning in healthcare

Deep learning has rapidly emerged as a groundbreaking force in healthcare by enabling the automated processing and analysis of complex biomedical data. Deep learning approaches leverage hierarchical neural network architectures to automatically represent raw data compactly and meaningfully, bypassing the need for handcrafted feature engineering. Consequently, diagnostic performance and decision making in a variety of medical specialities have improved significantly. For instance, such software is now employed routinely to analyse forms of medical images, i.e., X-rays, magnetic resonance imaging (MRI), and computed tomography (CT) scans [4, 5]. Their ability to reveal subtle patterns in imaging data has enhanced the identification of diseases, such as cancer, pulmonary disorders, and neurological conditions, at an early stage. Moreover, deep learning has accelerated advancements in organ segmentation and anomaly detection, thereby simplifying clinical workflows and personalised treatment planning [6].

Apart from image data, deep learning has proved to be highly helpful in the research of other data modalities, including biomedical signals, ECGs, electroencephalograms (EEG) and other physiological signals. Deep learning applied to these signals has led to notable advances in the identification of arrhythmias, ischemic activity and other cardiovascular abnormalities with improved accuracy. Diagnostic rapidity and monitoring of diseases has been facilitated due to the automated detection of subtle morphological changes in waveforms, previously undetectable in normal clinical inspections. This enhanced signal processing feature is essential in both acute care environments and long-term patient management [7].

[3] `https://www.pinterest.com/pin/7177680650242990/`

Beyond imaging and signals, deep learning expands to the analysis of heterogeneous healthcare data such as genomics, clinical records, and laboratory results [8, 9]. Given that deep learning can integrate various data modalities, these models offer a holistic view of patient health, enabling precision medicine initiatives and interventions tailored based on individual risk profiles. This multidisciplinary approach not only promotes early diagnosis but also improves prognostic assessments and prediction of treatment outcomes. The consistent development in deep learning architectures and training techniques, combined with the growing availability of large annotated datasets, points toward a future where these advanced techniques will play an increasingly central role in clinical decision support, leading to better patient outcomes and more efficient healthcare delivery [10].

1.3 Deep learning in Electrocardiograms

Deep learning has revolutionised the analysis of ECG data, enabling more efficient and accurate detection of cardiac abnormalities through sophisticated pattern recognition and data-driven approaches. Convolutional neural networks (CNN) have been particularly effective in automatically extracting spatial and temporal characteristics from raw ECG signals, facilitating the classification of arrhythmias, myocardial infarction, and other cardiac conditions with high accuracy. By leveraging large annotated datasets, these networks learn hierarchical representations of ECG waveforms that capture both subtle and overt abnormalities in the cardiac signal. In many studies, CNNs have outperformed traditional signal processing methods by reducing the need for expert-driven feature extraction and showing robust performance in real-world clinical settings [11].

Beyond CNNs, deep learning techniques such as recurrent neural networks (RNNs) and transformers have been employed to account for the sequential and temporal dependencies inherent in ECG signals. RNNs, including their gated variants like LSTM (long short-term memory) networks and GRUs (gated recurrent units), capture temporal dynamics by retaining the memory of previous signal states, which is particularly useful for detecting gradually evolving arrhythmias [12]. More recently, transformer models have been introduced to handle long-range dependencies and non-linear relationships in ECG data, often leading to improved performance in anomaly detection and classification tasks [13].

In addition, hybrid approaches that combine CNNs with RNNs or attention mechanisms have been developed to further refine the accuracy of these models, allowing better localization of critical ECG characteristics, such as the onset of the QRS complex or changes in T wave morphology. These advancements enhance automated diagnostic capabilities and open avenues for real-time patient monitoring and personalised treatment planning [14, 15].

1.4 Label noise and Electrocardiogram

Label noise in medical data arises from inaccuracies in the labels, which can significantly degrade the performance of deep learning models. This is particularly

problematic for the field of medicine, where precise diagnosis is essential for successful treatment [16].

Label noise in clinical data sets is diverse in nature. First of all, the labelling process is often time-consuming and needs specialised domain experience, while it is subject to inter-observer variability [17, 18]. Different experts have varying degrees of experience, and therefore, the annotations become inconsistent. For ECG data sets in particular, label noise can be very common due to a variety of specific domain difficulties. The majority of ECG datasets rely on retrospective diagnostic codes or annotations retrieved from imperfect electronic health records that may include errors, outdated diagnoses, or written report versus actual signal pattern discrepancies [19]. Additionally, the interpretation of ECG signals is subjective by itself, especially in borderline or noisy cases such as atrial fibrillation(AF) with variable conduction, premature ventricular contractions, or low-amplitude waveforms. In such instances,diagnoses performed by senior cardiologists may present significant variance[20]. Furthermore, automated annotation programs, sometimes utilised to annotate large datasets, can introduce systemic errors or misclassifications based on their application of heuristic rules or outdated algorithms [19]. Signal quality issues, like electrode displacement, baseline wander, or motion artefacts, may also obscure significant ECG features, preventing correct labelling and introducing the likelihood of noise [21]. These factors collectively are the cause of label noise in ECG datasets and pose significant challenges towards building reliable deep learning models for any cardiac event detection task.

The impact of label noise goes beyond a simple reduction in accuracy. Models trained on noisy labels are prone to suboptimal decisions. Furthermore, noisy labels can affect performance evaluation since clean labels are needed to determine a model's effectiveness [22].

Techniques for addressing label noise include noise transition matrix methods, which model the transition probability between noisy and clean labels. Sample selection methods can enhance noise robustness by weakening the influence of back-propagation gradients caused by wrong labels. Label correction methods change the labels that they deem noisy during training into another class to improve performance. Lastly, hybrid approaches, combining multiple techniques, can fully utilise all samples, although they may increase computational costs.

1.5 Previous work

In this section, some of the most important and influential previous work on noise mitigation will be presented, focusing on the three aforementioned technique categories.

Regarding the sample selection category, one of the most frequently used approaches is based on the loss [23]. Smaller loss means the samples are most likely to be clean since the model has higher confidence. The intuition is that the noisier samples will be harder to fit during the training, and therefore, initially, the model will experience larger losses. Another method used for sample selection is neighbour-based [24]. Algorithms like K-Nearest Neighbours [25] attempt to group the samples and then determine whether a sample is clean or not based on the labels

of its neighbouring samples. So, at the beginning of the training, the model again tends to group the clean samples together, thus, if the label of a sample is the same as the ones of its neighbours then it is most likely clean, otherwise, it is considered noisy.

Many strategies have been developed around the notion of sample selection. Han et al.[26] introduced the Co-teaching framework, which trains two networks in parallel and picks a subset of small-loss samples from one to send to the other. The co-training approach reduces the chance of confirmation bias since the two models, which are trained separately, are likely to memorise noisy samples at different rates. Each model shares the samples that are more confident they are clean, exploiting the diverse viewpoints of each model, thereby reducing the influence of noisy labels. Furthermore, by exploiting the strength of the two networks, Co-teaching increases the robustness of label noise mitigation and generalisation.

Malach and Shalev-Shwartz [27] introduced the Decoupling method, which tried to distinguish the 'when to update' from 'how to update'. Instead of only focusing on when a single network decides that a label is likely noisy, it was suggested to combine the disagreement prediction of two networks. So, two networks are trained at the same time, and the sample selection is based on data points where both networks disagree with the result. By exploiting this method, Decoupling aims to enhance the model's robustness by prioritising the sample selection on potentially corrupted or uncertainty-ridden data points. This strategy aims to mitigate the problem of memorising the noisy labels from the diversity of two separately trained networks and selectively updating based on information gained from the samples.

Another approach proposed by, Sun et al. [28] is the Probabilistic Noise Prediction (PNP). Unlike other sample selection methods that depend on small-loss criteria or threshold-based selection, PNP tries to predict the probability of each label being corrupted again by implementing the parallel training, where in this case, one model is used for classifying the labels and the other is used for identifying clean and corrupted samples. This method enables the model to adaptively select cleaner samples without requiring complex hyperparameter tuning based on a specific dataset, a process that is required by other methods.

Label correction is another category of noise mitigation techniques that can be divided into two subcategories: prediction-based and clean sample–based. In prediction-based methods, original and predicted labels are blended based on a variable called 'α' that adjusts this mixture. Section 2.6.2 expands on this in greater detail. For instance, Wang et al. [29] use the entropy of the predicted class distribution to adjust 'α' when predictions are ambiguous, while Arazo et al. [30] rely on the per-sample loss to decide whether a label is likely clean and weight accordingly. To further reduce bias, Chen et al. [31] introduced a dual-network scheme that cross-updates corrections so that each model filters the other's mistakes, and Bahri et al. [32] presented a k-nearest-neighbor consensus among a sample's neighbours that can also inform the choice of 'α'.

In contrast, clean sample–based label correction assumes access to a small, trusted set of noise-free examples. A meta-model trained on these clean labels corrects the larger noisy set [33], although simultaneously updating both meta and classifier models can degrade the quality of the label, leading Tu et al. [34] to

decouple correction and training into separate stages. Other approaches identify clean samples directly, using, for example, topology-based filtering [35], utilising the high-order topological behaviour of data to identify clean labels or the FINE sampling scheme [36], which employs eigenvectors and semi-supervised learning to filter noisy instances. These techniques then assign their labels to noisy instances before the final training objective blends losses on both the original and corrected labels to minimise the impact of any remaining errors.

Model predictions frequently serve as pseudo-labels for iterative correction. Tanaka et al. [37] first introduced a Joint Optimisation framework in which network parameters and class labels are updated simultaneously, using predictions to refine noisy annotations. Yi and Wu [38] then proposed PENCIL, an end-to-end method that treats label distribution as a learnable parameter, initialised by the noisy labels and optimised via back-propagation.

In medical imaging, determining when and how to correct labels is particularly challenging due to class imbalance and pixel-level normalisation differences. Liu et al. [39] observed that early learning and semantic memorisation do not coincide in segmentation tasks, and so designed ADELE to schedule corrections based on the deceleration of Intersection-over-Union (IoU) curves for each class. Jin et al. [40] tackled breast ultrasound tumour segmentation by estimating noisy pixels with a dedicated neural network and applying pixel-level corrections according to model confidence. Qiu et al. [41] addressed inconsistencies between whole-slide and patch-level labels in pathological image classification by deriving patch pseudo-labels from slide-level predictions.

An alternative prototype-based strategy was described by Han et al. [42] demonstrating that employing multiple prototypes per class better captures category characteristics and yields a deep self-learning framework that trains true noise-robust networks.

Hybrid methods combine multiple techniques to create a robust framework to tackle label noise. For example, Li et al. [43] developed the DivideMix framework, which models the sample loss distribution using a Gaussian Mixture Model to dynamically separate the dataset into labelled and unlabelled subsets. A semi-supervised learning strategy is then applied to train the model more effectively under noisy conditions, incorporating ideas from co-teaching and consistency regularisation. DivideMix is analysed in detail in Section 2.6.3.

Zhang et al. [44] proposed a comprehensive framework that integrates loss re-weighting and label correction to achieve robust training in settings with high label noise. Shi et al. [33] introduced a method that jointly learns the parameters of a deep neural network while correcting noisy labels, leveraging techniques such as label quality estimation and interpolation consistency learning. Similarly, Xue et al. [45] developed a co-training framework that employs both global and local feature representations for noise-tolerant classification of skin lesion images. This approach includes dataset partitioning, label correction, and the use of self-supervised learning. Liao et al. [46] proposed a 'divide-and-rule' strategy involving a hybrid network architecture that categorises pulmonary nodules into sets based on annotation reliability and applies targeted learning strategies, such as counterfactual learning and stage-wise training, to each set.

1.6 Motivation and objectives

The main motivation of this thesis is to provide an unbiased comparative analysis of the most popular label-noise mitigation methods used for ECG classification tasks. The objective is to compare the performance of these methods on the same model architecture and different artificial noise levels and types (**??**. After comparing the results and finding the best-performing technique tested on two of the most commonly used ECG datasets, this technique will be used to evaluate its performance on clinical data collected at Akershus University Hospital, Norway. One method for each noise mitigation technique was chosen (Section 2.6), based on their fundamental differences in handling label noise. In addition, the methods chosen do not need to know the noise level beforehand, which makes the experiment more realistic since the best performing method will be tested on the hospital dataset, where the label-noise level is unknown.

Fine-tuning the model architecture and hyperparameters to reach optimal classification accuracy is not in the scope of this thesis work. The main goal is, instead, to assess:

1. How can different noise levels affect the efficacy of the machine learning models?
2. How effective are the chosen noise handling methods for ECG datasets?
3. How is the model's performance on the hospital dataset affected if the label noise is handled?

The chosen methods are Stochastic Co-teaching [18], Self-learning [47] and DivideMix [43]. The theoretical framework behind each of these methods will be analysed later in Chapter 2 and Chapter 3. The model used for the analysis is a typical ResNet architecture that was chosen because it yielded satisfactory results in the chosen benchmark ECG datasets, while being shallow enough to train in an acceptable amount of time, given the limited available computational resources for this work. For all noise settings, the same hyperparameters and the macro-averaged area-under-curve (AUROC) were used as a metric for all comparisons, as it is consistent with previous research that performs comparative analysis [48]. The models were trained for 50 and 100 epochs for the CODE15% and PTB-XL, respectively, due to their size differences. The results are presented in figures and on tables in Chapter 4.

1.7 Description of datasets

Three datasets were evaluated in the scope of this thesis. PTB-XL [49] is a freely accessible clinical 12-lead ECG-waveform dataset consisting of 21,837 records from 18,885 patients of 10 seconds duration. CODE15% [50] is a stratified sample of the CODE [51] dataset (containing 15% of the patients) with 345,779 exams from 233,770 patients. It also has 12-lead ECG recordings with a duration longer than 10 seconds. Due to resource constraints, half of the CODE15% dataset was utilised. Finally, an ECG dataset provided by Akershus University Hospital [52], where data was gathered from 99,116 patients, totalling 284,161 ECGs, with a duration of 10

seconds sampled at 500 Hz. All the details can be seen at Table 1.1.

Each dataset has labels representing cardiovascular conditions that an ECG may record. PTB-XL is annotated with 5 main conditions referred to as superclasses: Conduction Disturbance(CD), Myocardial Infarction (MI), Hypertrophy (HYP), and ST/T change (STTC). CODE15% is annotated with 6 heart conditions: 1st degree AV block (1dAVb), right bundle branch block (RBBB), left bundle branch block (LBBB), sinus bradycardia (SB), atrial fibrillation (AF), and sinus tachycardia (ST). The Hospital dataset, on the other hand, has only two labels: heart failure or no heart failure. The other datasets also have a normal ECG class where no abnormalities were detected.

Table 1.1: Comparison of ECG Datasets Used in This Study

Attribute	**PTB-XL**	**CODE15% (50% used)**	**Hospital Dataset**
No. of Records	21,837	172,889	284,161
No. of Patients	18,885	116,885	99,116
No. of Classes	5	7	2
Leads	12	12	12
Recording Duration	10 seconds	10 seconds	10 seconds
Sampling rate	400 Hz	400 Hz	500 Hz
Annotation Source	Clinical Expert	Physician-reported ICD codes	ICD-10 codes validated with NT-proBNP

2
Theory

In supervised learning, the overarching task to be solved is to find the following probability

$$\mathcal{P}(Y|X_{ECG}) \tag{2.1}$$

where X is the input ECG and Y is the output label(e.g. Normal, Hypertension, etc.). Given a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N},$, where $x_i \in \mathbb{R}^{c \times t}$ represents an ECG signal and $y_i \in \{1, 2, \ldots, C\}$ is the corresponding label along C possible classes. The classification problem is defined as learning a mapping $f : \mathbb{R}^d \to \{1, 2, \ldots, C\}$, that predicts a class based on an ECG signal, and to accomplish that, the minimisation of a loss function is calculated as [53]

$$w^* = \arg\min_w \frac{1}{N} \sum_{i=1}^{N} L\left(y_i, f_w(x_i)\right), \tag{2.2}$$

where $L(\cdot)$ is the loss function, $f_w(X)$ is the model predictions, Y is the ground-truth labels and N the number of samples. For classification tasks, different loss functions can be used, where the most common one is the Cross Entropy Loss [54], defined as

$$\mathcal{L}(w) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{C} y_{ij} \log f_w^{(j)}(x_i). \tag{2.3}$$

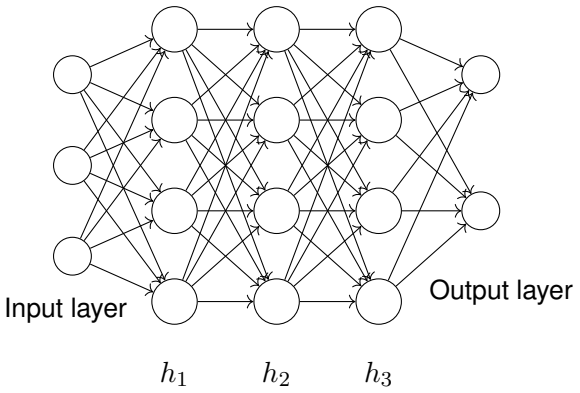
Cross-Entropy assigns a small value if the probability of the model's prediction is high, meaning that the model is confident about the prediction, and it is weighted by the ground truth label. On the other hand, if the model's prediction is low, the loss is penalised by having a higher value. The output of the model is usually raw logits, and to get the probabilities from these raw logits, another non-linear function is applied called softmax [55], which translates them into normalized probabilities

$$\text{softmax}(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{C} e^{z_k}}, \quad \text{for } j = 1, 2, \ldots, C. \tag{2.4}$$

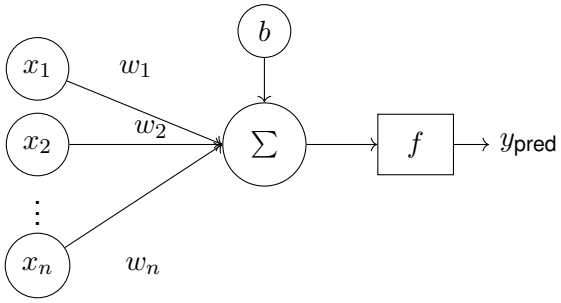
2.1 Multi-Layer Perceptron

To approximate the $f(w)$ that maps an input ECG to a specific label Y, neural networks see Figure 2.1a are used because they can capture complicated and non-linear relations or otherwise called Multi-layer Perceptron (MLP) [56]. Where each neuron computes a weighted sum of its inputs and then applies a non-linear activation

function. These layers enable the model to learn hierarchical data representations, gradually extracting increasingly abstract and task-relevant properties from the original input signal. Given an ECG, which is the heart's electrical activity, this data is being processed and goes as input to the MLP. MLP can process signals of various lengths and complexities, making it suitable for ECG signals. So each signal is passed through the MLP where the model can capture complex patterns and then finally at the last layer, the output is mapped to a specific class, see Figure 2.1b. However, they struggle with learning local structures and spatial patterns, which is crucial for ECGs since they have repeated temporal patterns.



(a) Neural network architecture



(b) Structure of a neuron in the output layer

Figure 2.1: (a) A feedforward neural network with multiple hidden layers and (b) the internal computation of a neuron in the final layer.

2.2 Convolutional Neural Network

This is where CNN comes in, by applying filters along the signal, they are used for extracting local features and temporal patterns, Figure 2.2) [56]. This filter slides over the input and extracts features, see Figure 2.3, like the shape of the ECG signal and other information like the length of the heartbeat, which is crucial in classifying abnormalities in the heart. By combining convolutions with the MLP, Convolutional Neural Networks are formed by extracting the most important features through the convolutions. The MLP helps with modelling these features into global relationships and converting them into class probabilities in the final layer, where the input signal is mapped to a specific class label. CNNs have shown some promising results in image classification. However, the drawback of deep CNNs is that they sometimes struggle with long sequences, especially in time series data, because they have a small receptive field [57]. Another problem of deep CNNs is one known as vanishing/exploding gradients, where the model fails to capture long-term dependencies [58]. Vanishing or exploding gradients happen during backpropagation, when gradients are propagated back to the previous layers from the output layer. Repeated multiplication of gradients over many layers in deep networks can produce extremely tiny (vanishing) or very large (exploding) gradient values.

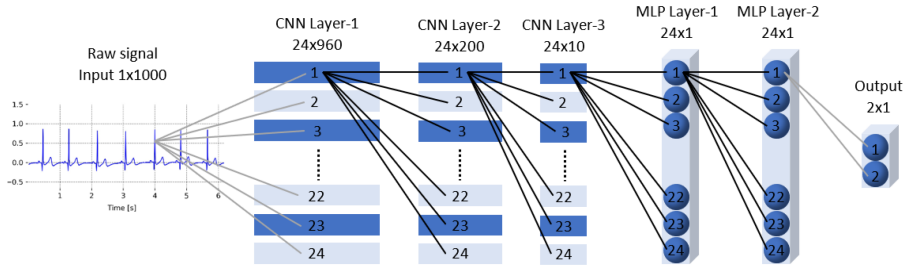


Figure 2.2: Example of CNN architecture with three convolutions and two MLP's [1].

When gradients vanish, weight updates in the early layers become minimal, thereby prohibiting the network to learn. When gradients explode, the updates become disproportionately big, resulting in unstable training and the possibility of a numerical overflow. This issue is particularly obvious in networks with several layers and non-linear activations such as sigmoid or ReLU. To address these issues and enable robust training of very deep models, techniques such as cautious weight initialization, normalization (e.g., batch normalization), and architecture designs such as ResNet's residual connections have been devised.

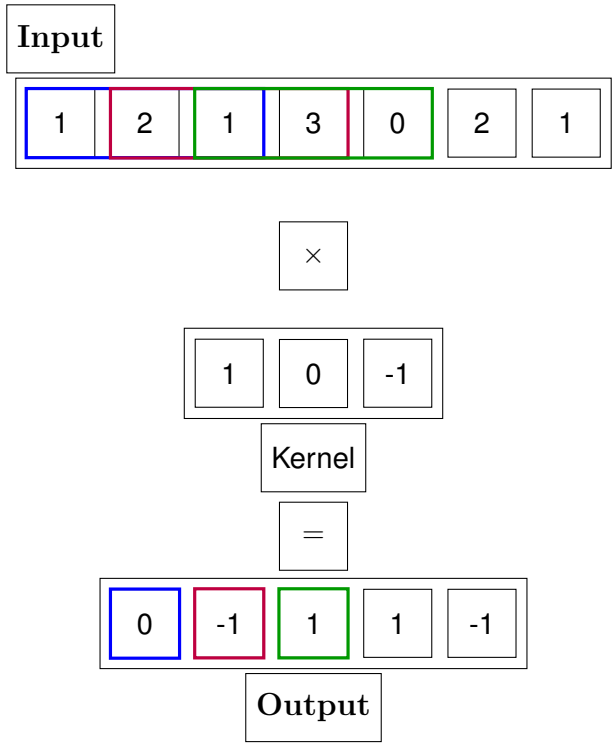


Figure 2.3: Example of 1D- Convolution where the kernel is sliding through the input and produce the output.

[1] https://www.mdpi.com/2076-3417/12/7/3332

2.3 Residual Networks

Residual Networks (ResNet) present a new architectural component called the residual connection, which solves the limits of deep networks caused by vanished gradients. ResNet focuses on learning a residual function, abbreviated as $F(x) = H(x) - x$ where x is the input and $H(x)$ represents the intended underlying mapping, rather than requiring each layer to learn a new transformation [59]. The block's output is then delivered as

$$\text{Output} = F(x) + x. \tag{2.5}$$

During backpropagation, this skip-connection allows the gradient to skip one or more levels and move straight down the identity path, see Equation (2.5). As a result, even in extremely deep networks, the gradients maintain appropriate magnitude. This approach not only stabilises training but also allows for the creation and optimisation of extraordinarily deep structures, often with hundreds of layers, without suffering from performance drop that affects very deep neural networks. In essence, ResNet allows the network to learn residual refinements rather than full transformations, which has been shown to be easier to tune and more successful for difficult tasks like ECG signal recognition, where the data are time-series and it is crucial to preserve information [60].

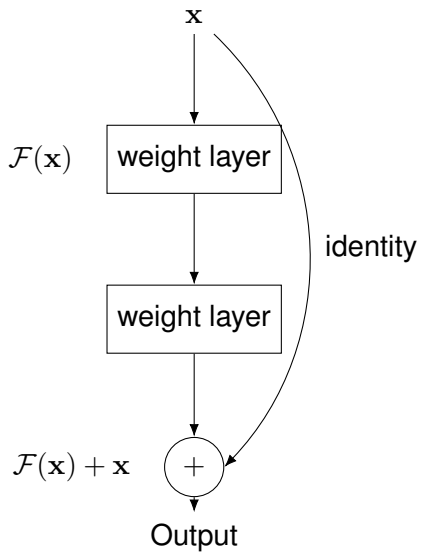


Figure 2.4: Residual connection example

2.4 Classification and noise

The efficacy and performance of a classifier in supervised learning are significantly influenced by the labels present in the training dataset. When labels inside the training consist of errors (label noise), the model faces inaccurate supervision signals, resulting in incorrect outcomes and degrading model performance [61]. In deep

neural networks, the model has the ability to learn from both clean and noisy samples, which leads to the model fitting the noisy data, resulting in incorrect results on unseen data. Given a dataset that contains noisy samples $\tilde{\mathcal{D}} = \{(x_i, \tilde{y}_i)\}_{i=1}^{N}$, where $\tilde{y}_i \neq y_i$ for some samples, the goal of the training is given by

$$w^* = \arg\min_w \frac{1}{N} \sum_{i=1}^{N} L\left(\tilde{y}_i, f_w(x_i)\right). \tag{2.6}$$

So, the model is attempting to minimise a loss function with noisy labels, which will result in poor performance on unknown data.

The memorisation effect further defines this event. Deep neural networks learn from clean input by identifying dominant, 'easier', and accurate patterns first [62]. However, as training advances, the model starts to memorise mislabelled or noisy samples, leading to a drop in performance. For example, a sample that was classified as noisy at the early stages of training it can be classified as clean when the model learns more complex patterns. This effect has been observed across a wide range of applications, and it is particularly causing many problems in critical sectors such as healthcare, where precision is critical. Additionally, the memorisation effect causes the model to overfit the data, which is undesirable because the performance should be equally effective on unseen data. Understanding this effect is crucial because it drives the development of many noise mitigation techniques, as will be described in the next chapters.

2.5 Types of Label Noise

Label noise is often classified into two types: class-dependent (or instance-independent) and feature-dependent [24]. Class-dependent noise can be further divided into symmetric and asymmetric noise.

2.5.1 Class-dependent Noise

Class-dependent label noise (also known as class-conditional or instance-independent label noise) is a noise model in which the chance of a label being corrupted is determined only by its true class and not by individual sample attributes [24]. This requires a constant conditional distribution $P(\tilde{y} \mid y)$ defining how a true label y flips into a noisy label $\tilde{y}$, independent of the input x. The noise process may be described by a $C \times C$ label transition matrix T, where C is the number of individual classes, with each entry $T_{ij} = P(\tilde{y} = j \mid y = i)$ indicating the chance that an occurrence of true class i is mislabelled as class j [63]. The rows of this matrix add to one, with the diagonal values reflecting the chance of the label being correct and the rest representing the probability of the specific class getting mislabelled to the corresponding class, see Figure 2.5. This transition matrix can be used to identify the probability of a label being clean and it can also help understand the distribution of the noise as it can reveal the probabilities of a class getting mislabelled into another class.

0.625	0.125	0.125	0.125
0.125	0.625	0.125	0.125
0.125	0.125	0.625	0.125
0.125	0.125	0.125	0.625

(a) Uniform Transition Matrix

0.5833	0.1875	0.1041	0.1250
0.1250	0.5892	0.1428	0.1428
0.2105	0.2105	0.5000	0.0789
0.1500	0.1750	0.1750	0.5000

(b) Random Transition Matrix

Figure 2.5: Examples of class-dependent label noise represented by transition matrices. Darker diagonal cells indicate the probability of the label being correct. Rows represent the true label and columns the noisy label.

2.5.1.1 Symmetric Noise

Symmetric label noise is a type of label noise in which all labels of the wrong class are assigned with equal probability, regardless of their true class. In other words, the chance of a label being corrupted is equally distributed among all other classes see Figure 2.5a. This indicates that any class label has an equal probability of being switched to any other label, with no systematic bias. Mathematically, if the noise rate is given by $r \in [0, 1))$, the true label stays right with a probability of $1 - r$, while it is flipped to any of the other $C - 1$ classes with equal probability, $r/C - 1$ [24, 23]. With a noise rate of 38.5% in a four-class classification application, the true label is maintained 62.5% of the time, but each of the other three classes has a 12.5% probability of being wrongly classified. Because of its uniform distribution, symmetric noise is sometimes referred to as uniform or unbiased label noise, since it imparts no preference bias toward any single class and acts fully randomly.

2.5.1.2 Asymetric Noise

Asymmetric label noise, often referred to as biased noise, pertains to instances in which the chance of mislabelling varies between classes, frequently reflecting domain-specific systematic confusion patterns. Unlike symmetric noise, where label corruption is uniformly random, asymmetric noise provides systematic mistakes, with certain classes being mislabelled. This mislabelling does not involve the assignment of erroneous classes at random, rather, class corruption is a 1-1 mapping between the sets of erroneous and corrupted classes [24]. This form of noise is very important in healthcare applications, since human annotators may mix classes with similar characteristics (e.g., wrongly categorising an abnormal ECG signal as normal due to close similarities between the signals). Under asymmetric noise, the label transition probability matrix T is non-uniform. This means that for a true class i, the likelihood T_{ij} of being mislabelled as class j is much higher than for other classes j see Figure 2.5b. This causes a shift in the noisy label distribution relative to the clean label distribution, resulting in biased empirical risk reduction during model

training. As a result, classifiers trained on asymmetrically noisy data tend to favour classes that are more frequently confused, lowering model calibration and aggravating performance differences between classes. For example, in a binary classification situation with two classes, if samples from class A are frequently mislabelled as class B, the classifier's decision boundary will be leaned toward predicting class B, embedding the annotation bias into the learnt model.

2.5.2 Feature-dependent noise

Feature-dependent noise, additionally referred to as instance-dependent noise, is label corruption that is conditioned on the individual characteristics of each sample, rather than just on the class label [23]. In this noise model, samples that are ambiguous, noisy, or close to decision-making limits are more likely to be mislabelled than clear or typical instances. The main difference between class-dependent and feature-dependent noise is the underlying dependency: with class-dependent noise, each class has a constant probability of being corrupted into another class, regardless of the particular sample's features. In contrast, feature-dependent noise introduces heterogeneity at the instance level, with the chance of label corruption varying based on each sample's individual properties. This complicates the learning process since the noise cannot be modelled only on class transitions but must also account for complicated feature distributions. In this thesis, only class-dependent noise will be artificially added to the datasets. Further information on the noise-generating process is provided in the following sections.

2.6 Noise mitigation techniques

A dataset can contain both clean and noisy samples, which can lead to a model fitting the noisy samples, leading to poor performance. To overcome this problem, all noisy samples should be identified and corrected, also known as cleaning the dataset [24], in order to generate more accurate results. This process can be computationally expensive since a dataset can contain many samples, and finding both clean and unclean samples lies beyond current methodological capabilities. So, to mitigate this issue, multiple techniques have been developed, which can be divided into sample selection, label correction, and Hybrid approaches that combine both methods.

2.6.1 Sample Selection

Sample selection exploits the memorisation effect, where the model tries to fit the clean data at the early stages of the training and leaves the rest, more complex/unclean samples, for fitting at the later stages of training. Based on this, the goal of sample selection algorithms is to identify dynamically the clean samples and use only these samples for training while disregarding the potential noisy samples [24]. By focusing on more reliable or potentially clean samples, sample selection techniques prevents the model from overfitting the data with corrupted labels and assist with the generalisation on unseen data [16].

Given a training dataset $\mathcal{D}$ that contains both clean and noisy labels, the aim of the sample selection to try to find a dataset $\hat{\mathcal{D}}_{\text{clean}} \subset \mathcal{D}$ that consists of clean samples. So, instead of minimising the empirical risk on the whole dataset, Equation (2.2), the goal is to use the subset of the clean data $\hat{\mathcal{D}}_{\text{clean}}$.

Stochastic Co-teaching [18, 64] is a sample selection noise mitigation technique that extends the traditional Co-teaching technique [26] and is designed to work on scenarios where the noise level is unknown. The basic idea of Co-teaching is that training two networks simultaneously can identify and filter noisy labels by exploiting their differing decision boundaries. Crucially, this division is performed by one network and used to update the other. Network divergence is maintained through differences in initialisation, data partitioning, mini-batch ordering, and training targets.

While traditional Co-teaching depends on prior knowledge of the noise rate and employs a fixed forgetting rate based on it, the stochastic variant introduces randomness into this selection process. So, rather than discarding a fixed proportion of samples based on their loss values, it samples a threshold from a beta distribution [65], Equation (2.7) where α and β are the given positive integers parameters, to determine whether training instances should be retained or rejected based on their posterior probabilities. The retained samples are used for the loss computation, while the rejected ones are temporarily excluded.

During training, the algorithm iterates through mini-batches, draws a random threshold from a beta distribution and selects instances for training based on whether their posterior probability exceeds the sampled threshold. This way, a key limitation of the traditional co-teaching method, knowing a priori the noise level of the dataset, is circumvented. Each of the two models is updated using the subset of instances selected by the other model, mitigating the risk of self-reinforcing erroneous labels. Lastly, to ensure numerical stability, the thresholds are constrained within a range of 0.01 and 0.99.

Like all noise mitigation techniques, to exploit the memorisation effect (Section 2.4), the integration of stochastic co-teaching is gradually implemented in the training procedure, allowing for the model to train without rejecting any samples for a predefined number of epochs, called a warm-up period.

$$\mathcal{B}(\alpha, \beta) = \int_0^1 x^{\alpha-1}(1-x)^{\beta-1}\, dx \tag{2.7}$$

2.6.2 Label Correction

Label correction is a technique used to mitigate the impact of noisy labels in datasets by optimising the labels themselves rather than treating them as fixed. The primary goal is to correct labels in a way that minimises changes to clean samples while maximising the impact on noisy samples. This approach leverages the prediction function $f(x;\theta)$ to control the impact on sample labels, which can be summarised by the label correction function

$$\bar{y} = c\Big(f(x;\theta), \tilde{y}; \phi\Big), \tag{2.8}$$

where c is the correction function, $\tilde{y}$ is the given (noisy) label, $\bar{y}$ is the corrected label, and ϕ represents hyperparameters governing the correction.

The training dataset $\tilde{\mathcal{D}}$ is then transformed into a corrected dataset

$$\bar{\mathcal{D}} = \Big\{(x_i, \bar{y}_i) \mid i = 1, \ldots, n\Big\}, \tag{2.9}$$

where each $\bar{y}_i$ is obtained via Equation (2.8). A classification model parametrised by θ is trained on $\bar{D}$ using gradient descent, with updates of the form

$$\theta \leftarrow \theta - \eta \sum_{(x_i, \bar{y}_i) \in \bar{D}} \nabla \mathcal{L}\Big(f(x_i; \theta), \bar{y}_i\Big), \tag{2.10}$$

where η is the learning rate and L is the loss function.

Label correction methods fall into two principal categories: prediction-based and clean sample–based. In prediction-based label correction, one simply replaces or refines the label of each example using the model's own outputs. A straightforward scheme sets $\bar{y}^{(t)} = f(x; \theta^{(t)})$ (or its hard counterpart $\hat{y}$) at epoch t, then retrains on the corrected set $\bar{D}^{(t)}$. To prevent wholesale corruption of clean data, later variants blend original and predicted labels via

$$\bar{y}^{(t)} = \alpha\, \tilde{y} + (1 - \alpha)\, f(x; \theta^{(t)}),$$

and further stabilise the correction with an exponential moving average:

$$\bar{y}^{(t+1)} = \alpha\, \bar{y}^{(t)} + (1 - \alpha)\, f(x; \theta^{(t+1)}).$$

Adaptive strategies then modulate α on a per-sample basis according to measures of prediction confidence or sample "cleanliness". These techniques are described in the previous work section in Section 1.5.

Self-learning, proposed by Vázquez et. al [47], is a technique that falls under this category. It is based on cosine similarities [66], see Equation (2.11), between the features of ECG signals. Cosine similarity is be defined as:

$$\text{cosine\_similarity}(\mathbf{A}, \mathbf{B}) = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\, \|\mathbf{B}\|}, \tag{2.11}$$

and it tries to measure how similar A is to B. A value closer to 1 means that they are very similar, and closer to 0 means that they are not. The principle of this method is again based on the memorisation effect, where first there is a warm-up period until the network learn from the potential clean samples before starting to memorise the noisy. The difference in this method and co-teaching is that now only a single network is trained, and instead of disregarding the noisy samples, label correction is performed. After the warm-up period of the model, some prototypes or representative classes are selected. For the selection process, all the features for each class extracted from the previous layer of the output classification head are obtained. Then, the cosine similarity between each sample is calculated. After that, the density or the mean cosine similarity value is calculated for each sample and then the values are sorted. High density value means that the sample is really close to the rest of the samples. This phenomenon is exploited to obtain diverse prototype

samples from each class. This way, a better class representation is obtained by capturing all the best possible samples that represent every class.

For the label correction process, the cosine similarity between each sample and every class prototype samples is calculated. If the features of the sample have high similarity with the features of a specific class prototype sample then the label is corrected to this class. So, now the original labels denoted as t and those that were corrected shown as c are used in a weighted loss function, see Equation (2.12), where the $\mathcal{L}$ is the Loss chosen depending on the specific task and a is the weight factor. Higher α value means that the model trusts the corrected labels more, while a lower value favours the original labels.

$$\mathcal{L}_{\text{total}} = \alpha \cdot \mathcal{L}(\hat{y}, c) + (1 - \alpha) \cdot \mathcal{L}(\hat{y}, t). \tag{2.12}$$

To summarise, label correction directly substitutes noisy labels with their estimated true values. However, if these replacements are themselves inaccurate, the alignment between genuinely clean examples and their labels can degrade, impairing overall model performance. Nevertheless, the rationale for pursuing label correction is clear: in datasets characterised by high label noise or limited data, simply discarding questionable samples squanders valuable information, whereas carefully refined labels enable the model to learn from as much of the dataset as possible [16].

2.6.3 Hybrid Approach

Hybrid methods in medical image analysis represent a promising direction for enhancing model robustness by combining diverse learning strategies that exploit information from both clean and noisy data. These approaches integrate complementary paradigms such as re-weighting, semi-supervised learning, and label correction to mitigate the adverse effects of label noise and improve overall performance.

DivideMix [43, 67], proposed by Li et al., introduces a novel and effective hybrid framework for learning with noisy labels by integrating semi-supervised learning (SSL) techniques into the training process. The core idea is to dynamically partition the training dataset into two subsets: a labelled set(X), presumed to contain clean samples, and an unlabelled set(U), presumed to contain noisy samples. Semi-supervised learning is then applied to utilise both subsets, thereby regularising the model and improving its generalisation capability.

To address confirmation bias, where a model tends to reinforce its own misclassifications, which is a common hurdle in self-learning approaches, DivideMix, similar to Co-teach, employs a co-divide strategy involving two neural networks trained in parallel. Each network independently models the distribution of per-sample losses using a two-component Gaussian Mixture Model (GMM) [68] at each epoch. The GMM estimates the probability of a given sample being clean, based on its cross-entropy loss. Specifically, the clean probability is defined as the posterior probability that the loss belongs to the Gaussian component with the smaller mean. GMMs offer improved flexibility in capturing sharp transitions between clean and noisy samples. A threshold is applied to the estimated clean probabilities, enabling each network to divide the data into labelled and unlabelled subsets and update the other network with them.

During the semi-supervised phase of the training, DivideMix builds upon the MixMatch framework [69] by incorporating two mechanisms to effectively handle both the clean and noisy subsets of the data: label co-refinement and label co-guessing.

Label co-refinement adjusts the original ground-truth label of a sample by blending it with the model's prediction, using a weight w_b derived from the clean probability estimated by the peer network. The refined label is computed as a linear combination of the original label y_b and the model's prediction p_b, and subsequently passed through a sharpening function to produce a more confident prediction (see Equation (2.13)). If the probability w_b is high, indicating that the label is most likely correct, then more weight is assigned to the label. Conversely, if the probability w_b is low, suggesting that the label is likely incorrect, then more weight is given to the model's prediction. Then, Temperature $\mathcal{T}$ is applied to boost the confidence of the prediction further. This helps guide the learning process with more regularised and denoised supervision,

$$\hat{y}_b = \text{Sharpen}\left(w_b y_b + (1 - w_b) p_b,\, \mathcal{T}\right). \tag{2.13}$$

Co-guessing is applied to the unlabelled samples where a pseudo-label is generated by averaging both networks' predictions to ensure that the generated labels are in consensus with both models. Using the refined labels for the labelled subset and the pseudo-labels for the unlabelled subset of the data, multiple augmentations are applied using a beta distribution. This is the last part of the MixMatch framework, the mixup operation [70], that is applied to interpolate between pairs of samples (x_a, x_b) and their corresponding labels (y_a, y_b) to create new, mixed or augmented samples to improve the generalization of the model and prevent overfitting, see Equation (2.14), where λ is a parameter drawn from beta distribution with a given α. High values of α imply a soft mix-up of two blended samples, whereas the lower end of the α parameter range indicates the domination of a singular ECG sample in the mix-up. The MixMatch framework is essentially allowing the model to learn more robust and generalizable features, improving its performance on unseen data. By creating new samples as linear interpolations of existing ones, the model is incentivised to learn features that vary smoothly and linearly across the input space.

$$\begin{aligned} &\lambda \sim \text{Beta}(\alpha, \alpha), \quad \lambda \leftarrow \max(\lambda, 1 - \lambda), \\ &\tilde{x} = \lambda x_a + (1 - \lambda) x_b, \\ &\tilde{y} = \lambda y_a + (1 - \lambda) y_b, \end{aligned} \tag{2.14}$$

where $\tilde{x}$ and $\tilde{y}$ is the new mixed ECG sample and label respectively.

Finally, the network is then trained in a semi-supervised manner, utilising these pseudo-labels to exploit information that noisy samples might contain without relying on their potentially noisy labels. This is achieved by using a custom loss function called SemiLoss that combines a supervised loss term with a semi-supervised one. For the labelled samples, a cross-entropy loss is employed, while the unlabelled samples are trained using the mean square error between the predictions and the pseudo-labels. Under severe noise conditions, the model might collapse into

predicting a single class, so a regularisation term, called the "Kullback-Leibler" divergence [71], is added to the loss. This term encourages balanced predictions by penalising the loss Equation (2.15) if the model favors a specific class. The final custom loss formula reads:

$$\mathcal{L} = \mathcal{L}_X + \lambda_u \mathcal{L}_U + \lambda_r \mathcal{L}_{\text{reg}}, \tag{2.15}$$

where $\mathcal{L}_X$, $\mathcal{L}_U$ and $\mathcal{L}_{reg}$ are the supervised, unsupervised and regularization losses respectively and λ_u and λ_r are weights to control the strength of the unsupervised and regularisation losses.

Like the previous noise mitigation techniques, DivideMix is incorporated gradually during training to avoid rapid overfitting to noisy labels during the initial stages of training. In addition to that, an extra mechanism to mitigate potential overfit problems, when asymmetric noise is applied, is utilised, where, during the warmup phase of training, a negative entropy term is added to the cross entropy loss to penalize overconfident (low-entropy) predictions and enable a more robust separation of clean and noisy labels. In conclusion, DivideMix combines Co-teaching techniques with semi-supervised learning and regularisation to address key challenges in learning from noisy labels. The complete workflow of DivideMix can be seen in Figure 2.6.

Although hybrid methods may introduce additional hyperparameters and computational overhead, they offer considerable potential for addressing the inherent challenges of small datasets and high levels of label noise that are characteristic of medical image analysis. Moreover, self-learning label correction techniques can be extended to multi-label classification settings, allowing for the assessment of their impact on both overall model performance and class-specific outcomes under varying levels of label corruption.

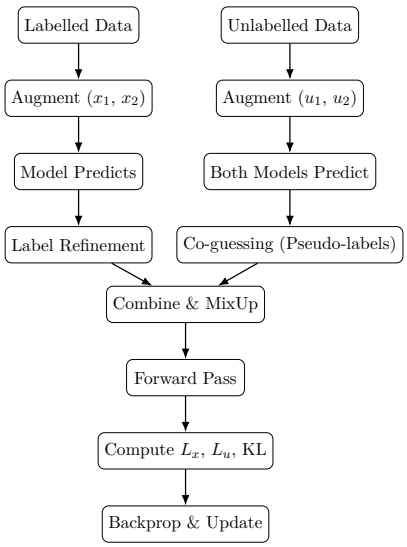


Figure 2.6: Workflow of the DivideMix training procedure. Labelled and unlabelled data are augmented, refined or pseudo-labelled, then combined with MixUp and passed through the network. Losses for labelled and unlabelled samples, along with a KL penalty, are used for training.

3

Methods

In this chapter, the model architecture, or the backbone of the network, is explained. After that, the data pre-processing is introduced in all datasets, and then more details are presented for the noise mitigation techniques that are used and compared in this project. Finally, the training setup for each method is shown along with the chosen hyperparameters and evaluation metrics used.

3.1 Model architecture

In this project, a custom ResNet architecture proposed by Wang et. al. [72] and then implemented by De Vos et. al. [18] is used. This architecture is utilized widely in time-series focused problems as it successfully overcomes the problem of vanishing gradients, and also captures long-term dependencies by having a larger receptive field as mentioned in Section 2.2. Having said that, as seen in Figure 3.2, the model takes as input ECG samples that has the size of (B, C, L) where B is the batch size, C is the number of Channels or leads used to get the ECG and L is the length of the ECG. For example, in the PTB-XL dataset, the length of the ECG is 10 seconds, sampled with 100 Hz, so L is 1000. There are 5 stages in the network configuration, the stem stage, where the input ECG is first convolved and then moves forward to the BasicBlocks see Figure 3.1, where the residual connections are utilised. At Stage 2 and Stage 3, downsampling is performed, and finally the classification Head layer exists, where, at the global information, an average-pooling layer is concatenated with a max-pooling layer [73] followed by a small MLP, see Section 2.1, to perform the classification.

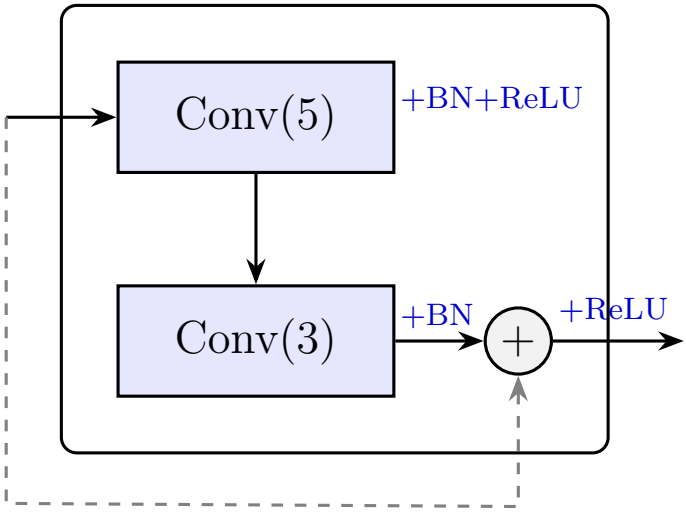


Figure 3.1: Structure of a BasicBlock in ResNet1D. The dashed arrow represents the residual connection bypassing the two convolutions as explained in Section 2.3.

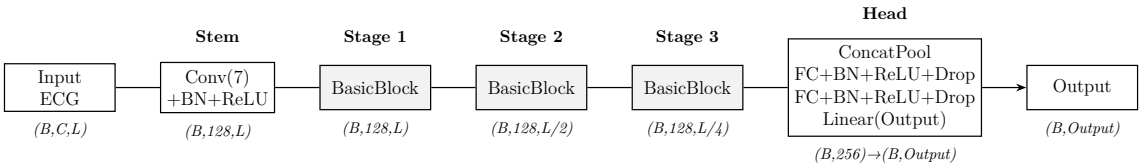


Figure 3.2: ResNet1D architecture using BasicBlock as modular units. **B,C,L**: Batch size, Channels, Length of time-series, **FC**: Fully Connected layer, **Drop**: Dropout layer

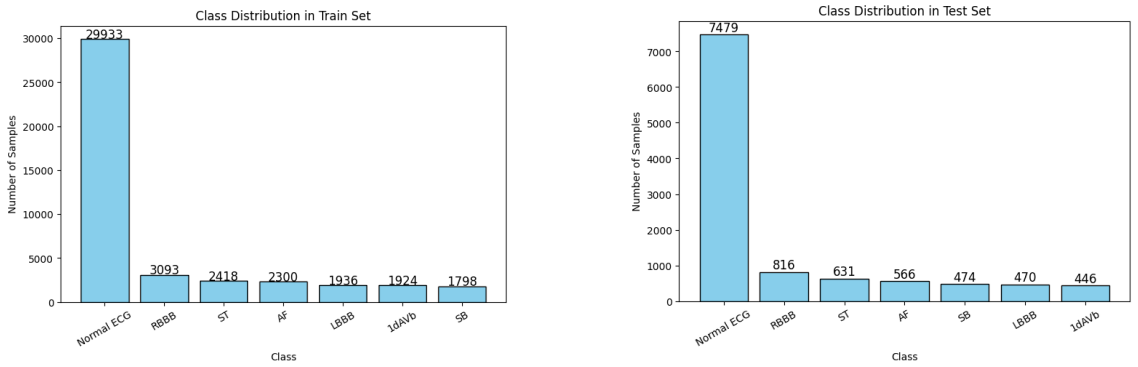
3.2 Pre-Processing

In this section, the preprocessing of each dataset is explained in detail. For each dataset, a robust pipeline is developed to extract useful information from the raw data available online using Python.

3.2.1 CODE15%

The pipeline of this dataset handles ingesting, structuring and partitioning the CODE15% ECG dataset. First, the CSV file ("exams.csv"), downloaded from the website [50], is ingested to associate each unique exam identifier with its corresponding raw ECG trace file and to translate the six condition-flag columns into a seven-element binary label vector. The seventh element indicates the 'normal' ECG where none of the six conditions are present. Since the focus of this work is on single-label classification, any recording with more than one label active is skipped at this stage of the process. Any entries whose referenced trace file is missing or whose ID cannot be located within the file are skipped and logged. Next, each remaining raw HDF5 trace file is opened in turn, the specific recording for the exam ID is extracted as a fixed-length 12-lead time series, paired with its label vector, and

appended into a new aggregated HDF5 container. Finally, once all samples have been consolidated, the accumulated waveforms and labels are randomly permuted to ensure mixing and split according to an 80/20 ratio into separate HDF5 archives for training and testing. These files are then loaded in the main pipeline via a custom Dataloader. This loader further processes the data to make the training faster, more efficient and less resource-intensive. First, it normalises the values to remove outlier values and potential noise. Afterwards, the number of samples is reduced from 4096 to 500 to accelerate training and remove left and right padding that was applied to the ECG data to make their length uniform. These 500 samples are selected from the middle of the ECG recording to capture its most important and useful patterns. Lastly, since the dataset is highly imbalanced towards the 'normal ECG' class, this class is first downsampled, and afterwards the class weights are calculated dynamically and passed to the loss function for better and unbiased results. The final class distribution for the train and test sets is displayed in Figure 3.3

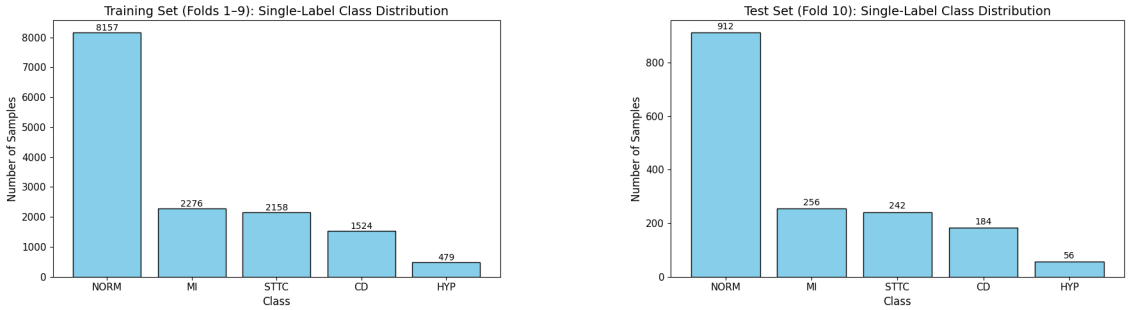


(a) Training Set **(b)** Testing Set

Figure 3.3: Distribution of each class in the CODE15% dataset for both training and testing sets.

3.2.2 PTB-XL

For the PTB-XL pipeline, the pre-processing is done as suggested in [49], where first the 'scp_statements.csv' file is loaded and filtered to contain only the samples where the diagnostic column is 1, thus retaining only the samples with diagnosis statements. Also, for the scope of this thesis, the authors provide the samples into super-classes and sub-classes where the super-classes' implementation is chosen. Another file contained in the dataset is called 'ptbxl_database.csv', which contains some metadata for every ECG sample (e.g., the corresponding raw ECG trace file, patient ID, stratification fold, and many more). At this stage, every sample with more than one label active is skipped. To process the ECG signals, a library called 'wfdb' is used, see [74], and finally, the low-frequency (100 Hz) ECG samples are returned in the form of a dictionary type containing the data and the corresponding label. The data used for training are those with a stratification fold from 1 to 9, and for testing, those with 10. The distribution of each class for the training and testing set can be seen in Figure 3.4.



(a) Training Set **(b)** Testing Set

Figure 3.4: Distribution of each class in the PTB-XL dataset for both training and testing sets.

To further decrease the memory usage, a random window of 500 samples is selected, reducing the total length of each ECG from 1000 to 500. In addition, Z-score normalisation [75] is applied to the training and test sets again to remove potential noise, outliers, and provide numerical stability.

3.2.3 Hospital Dataset

For the Akershus ECG dataset, the ECG signals and the labels are provided by the hospital. The data are already in a tensor type format with the shape of (B,C,N), where, B is 284,161 ECG samples with a duration of 10 seconds sampled at 500 Hz, C is the number of channels or leads where specifically 8 channels are provided and N is the length of the ECG that is set to 5000.

Like the other datasets, this dataset only had Z-score normalisation applied. The labels are in binary form, showing that the ECG is either normal or abnormal, so the model is changed accordingly to account for binary classification instead of multi-class.

3.3 Noise Injection

In Section 2.5, various types of label noise are thoroughly analysed. Within the scope of this thesis, it is assumed that both the training and test sets contain only clean labels before the manual injection of artificial noise into the training set. Also, only class-dependent noise is considered for injection into the datasets, specifically in the form of symmetric (unbiased) and asymmetric (biased) noise at rates of 20% and 40%. These percentages are selected because small levels of noise do not typically affect the performance of the model when appropriate regularisation and overfitting prevention techniques are applied. Conversely, more than 50% noise can be considered unrealistic. Therefore, 20% and 40% noise settings are used. In the case of symmetric noise, each class label has an equal probability (either 20% or 40%) of being randomly changed to any other class, simulating a uniform distribution of mislabelling, see Figure 3.5a. In contrast, asymmetric noise follows a predefined noise transition matrix, in which label changes occur according to probabilities that

reflect realistic patterns of misclassification, see Figure 3.5b.

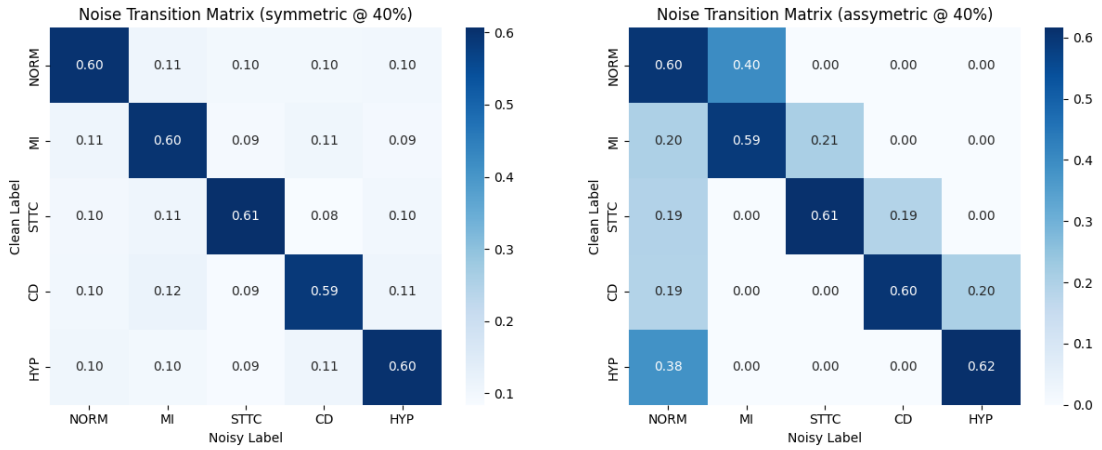
The transition matrix used for asymmetric noise is constructed to simulate real-world annotation errors, particularly those made by inexperienced or fatigued physicians in clinical settings [76]. To ensure medical plausibility, the matrix is developed in consultation with a cardiologist, incorporating domain knowledge about diagnostic categories that can be potentially confused. This medically-informed noise modelling aims to better reflect the types of errors encountered in practical ECG annotation scenarios. Table 3.1 and Table 3.2 present the full noise transition matrix for each dataset. If more than one possible transitions are available then the noisy class is determined randomly.

Table 3.1: Asymmetric noise transition pairs used for label corruption in the CODE15% dataset.

Original Label	**Noisy Label**
Normal	AF or 1dAVb
SB	1dAVb
ST	AF
AF	Normal or ST
RBBB	LBBB
LBBB	RBBB
1dAVb	Normal

Table 3.2: Asymmetric noise transition pairs used for label corruption in the PTB-XL dataset

Original Class	**Noisy Labels**
Normal	MI
MI	Normal or STTC
STTC	Normal or CD
CD	Normal or HYP
HYP	Normal



(a) Symmetric noise **(b)** Asymmetric noise

Figure 3.5: Noise transition matrices when 40% noise is applied to the PTB-XL. On the left the noise is uniformly distributed on the other classes while on the right the noise is applied to certain classes, representing more realistic cases.

3.4 Noise Mitigation

In this chapter, all the noise mitigation techniques used in the scope of this thesis will be explained. Also, to make a thorough evaluation of the noise mitigation techniques, a baseline version is developed, which does not apply any strategies to handle noisy labels and is trained on them. Therefore, the goal is to check how the model's performance changes depending on the percentage and type of the noisy labels.

3.4.1 Baseline

For the baseline version, the training is straightforward and the reason for developing a simple baseline model is to be able to compare how the performance changes from a version that does not apply any noise mitigation techniques. As can be seen from Table 3.3, the model was trained with an Adam optimiser [77], learning rate of 0.001 and batch size 128. The loss function utilised is the Cross Entropy loss [54] with weight decay of 0.00001 [78] to help with the overfitting and generalisation on the unseen data. Finally, to improve the convergence of the model, a cyclical learning rate scheduler is implemented [79], where it adjusts the learning rate after each batch. The baseline version used for the evaluation of the hospital dataset has the same hyperparameters as described above.

Table 3.3: Hyperparameters used for training the baseline model

Training detail	Value
Number of epochs	50/100
Optimizer	Adam
Initial Learning Rate	0.001
Batch Size	128
Learning Rate Scheduler	OneCycleLR
Loss Function	Cross Entropy Loss
Weight Decay	0.00001

3.4.2 Stochastic Co-teaching

Stochastic Co-teaching (Section 2.6.1) is used as a sample selection technique that rejects samples based on the confidence in the output probabilities. Also, it is stochastic because the threshold to decide whether the sample is clean or noisy is drawn from a beta distribution. The parameters used for the distribution are $Beta(32, 2)$, which outputs a mean probability of 0.94. A warm-up period of 10 epochs is used, during which no sample is rejected. Since the model is not yet confident in the early stages of training, the rejection threshold is then gradually increased over the next 10 epochs to ensure that a sufficient number of reliable samples are included. The specific parameter values are displayed in Table 3.4.

Table 3.4: Key Parameters of the Stochastic Co-Teaching Algorithm.

Name	Value
Beta distribution parameters	$\alpha = 32, \beta = 2$
Threshold clamp range	$t_{\min} = 0.01, t_{\max} = 0.99$
Gradual increase	tp_gradual = 10
Loss Function	Custom Loss Function
Scheduler	OneCycleLR
Learning rate	0.001
Batch size	64
Epochs	50/100
Warmup epochs	10

3.4.3 Self-learning

As discussed in Section 2.6.2, self-learning is a sample selection method that changes labels that are deemed too noisy based on their similarities to the prototypes of selected classes. The configuration used for this implementation, which can be seen in Table 3.5, has a batch size of 128, the initial learning rate of 0.001, and a warm-up phase of 5. For each class, 16 prototype samples are obtained, where the threshold to choose diverse samples is set to 0.9, meaning if the density score is below that value, the sample is added as a prototype. To correct a sample's label, the threshold is 0.9

again. Also, a Cosine Annealing learning rate scheduler [80] is utilised to improve the convergence. For the weighting factor at the Loss function, see Equation (2.12), equal weight is given to both the original and the corrected labels. Finally, a weight decay of 0.00001 is used to prevent overfitting the training data.

Table 3.5: Hyperparameters and settings used for the self-learning implementation

Parameter	**Value**
Batch size	128
Initial learning rate	0.001
Total epochs	50/100
Warm-up epochs	5
Learning rate scheduler	CosineAnnealingLR
Prototype samples per class	16
Prototype selection threshold	0.9
Label correction threshold	0.9
Loss weighting factor α	0.5
Maximum features per class	2000
Weight decay	0.00001

Due to its higher performance in the other datasets, Self-learning is used to evaluate the dataset provided by the Akerhus hospital. This dataset is used to further evaluate the performance of the noise mitigation technique when the label noise is unknown, imitating a real-world scenario. The labels are annotated using ICD-10 codes. The method is compared to the baseline version to ensure a fair comparison, and the parameter values are displayed in Table 3.6.

Table 3.6: Hyperparameters and settings used for the self-learning implementation on the hospital dataset

Parameter	**Value**
Batch size	128
Initial learning rate	0.001
Total epochs	50/100
Warm-up epochs	5
Learning rate scheduler	CosineAnnealingLR
Prototype samples per class	32
Prototype selection threshold	0.95
Label correction threshold	0.9
Loss weighting factor α	0.5
Maximum features per class	5000
Weight decay	0.00001

3.4.4 DivideMix

DivideMix, as explained in Section 2.6.3, is used as an alternative method where, instead of rejecting the samples or correcting the labels as the other methods do, the potential noisy samples are treated in a semi-supervised way. Like the other methods, there is a warm-up period of 15 epochs before the noise mitigation starts. After the warm-up time, the decision threshold that is used for the labelled and unlabelled samples is 0.5. For the MixUp process, both beta distribution parameters are 0.4, meaning that the two ECGs are weakly blended. A temperature of 0.5 is used to sharpen the predicted probabilities, increasing the confidence of the model's predictions. The weight of the unsupervised loss is linearly ramped up to a maximum value of 25, leading the model to progressively emphasize more on pseudo-labels throughout training. The specific parameter values used are described in Table 3.7. Every parameter not described here has not been altered from the original implementation.

Table 3.7: Key Parameters of the DivideMix Algorithm

Name	**Value/Type**
Loss Function	Custom Loss Function
initial λ_u	25
λ_r	1
Mixup beta distribution parameter	$\alpha = 0.4$, $\beta = 0.4$
Weight Decay	0.00001
Temperature(T)	0.5
Learning rate	0.0001
Batch size	32
Epochs	50/100
Warmup epochs	15
Scheduler	CosineAnnealingLR
p_threshold	0.5

3.5 Evaluation

Regarding the evaluation metric used, because the datasets are imbalanced, metrics like accuracy, precision and recall do not represent the true performance of the model. For example, if there is a binary classification problem and the dataset is very imbalanced e.g. 90% of the ECGs inside the dataset are normal, and the other 10% are abnormal, a naive model that classifies everything as normal, still achieves an accuracy of 90%. Therefore, the Area Under the Receiver Operating Characteristic Curve (AUROC) [81, 82] is used in this thesis work, a metric that displays the probability of how well a model can distinguish one class from the others. Since it is a multi-class classification problem, 'one versus rest (ovr)' is used with the macro average as a setting. This metric has been used in every method that has been employed in this research and is consistent with the evaluation methods used in the original articles. Also, AUROC is very useful in medical applications, because

there is always a trade-off between sensitivity (True Positive Rate) and specificity (False Positive Rate) and ROC assists in finding the optimal threshold.

4
Results

In this chapter, the results will be presented across all datasets and different noise types and percentages after comparing all the methods. Additionally, the handling of the noisy labels by each noise mitigation technique is presented with an emphasis on the disparities observed across the different datasets.

4.1 AUROC Comparisons

The goal of this thesis is to compare how different noise mitigation techniques can handle different levels of noise and how noisy labels in general can affect the machine learning models. To do that, a baseline version is developed and compared with the stochastic co-teaching, Self-learning and DivideMix. In Table 4.1 and Table 4.2, all the AUROC results can be seen for both data sets, while in Table 4.3, the results for the hospital dataset are displayed. For PTB-XL, the model that showcased the best performance is Self-learning, with significant improvement from the baseline version and all the other methods. For the CODE15% there is no clear method outperforming the others. For 20% symmetric and 40% asymmetric noise, stochastic co-teaching showed the best performance, while for the other noise settings, self-learning had the best results.

Table 4.1: AUROC values under different noise conditions for the PTB-XL dataset

Algorithm	**No Noise**	**Sym-(20%)**	**Sym-(40%)**	**Asym-(20%)**	**Asym-(40%)**
Baseline	0.91	0.81	0.74	0.81	0.78
Stochastic Co-Teaching	0.82	0.84	0.82	0.80	0.78
Self-Learning	**0.92**	**0.89**	**0.86**	**0.90**	**0.87**
DivideMix	0.81	0.80	0.82	0.81	0.78

Table 4.2: AUROC values under different noise conditions for the CODE15% dataset

Algorithm	**No Noise**	**Sym-(20%)**	**Sym-(40%)**	**Asym-(20%)**	**Asym-(40%)**
Baseline	**0.93**	0.83	0.80	0.85	0.82
Stochastic Co-Teaching	0.92	**0.90**	0.83	0.88	**0.88**
Self-Learning	0.92	0.88	**0.88**	**0.89**	0.87
DivideMix	0.82	0.81	0.83	0.79	0.82

Table 4.3: AUROC values of the hospital dataset

Algorithm	No Noise
Baseline	0.80
Self-Learning	0.80

The baseline model performance for both datasets dropped significantly when noise was injected. Lastly, while Stochastic Co-teaching and Self-learning showed the most promising results on how to mitigate the presence of noisy labels, DivideMix did not achieve better results than the baseline. In the case of the hospital dataset, the baseline version and the self-learning method yielded the same results when no artificial noise was injected.

4.2 Stochastic Co-teaching

The Stochastic Co-teaching noise mitigation technique rejects samples that the two models find noisy and improves their predictions and rejection rates over the epochs. Investigating these rejection rates for each noise setting and over each epoch is a useful way to see this technique's performance.

Figure 4.1 shows the rejection rates as a percentage for each dataset and noise setting. Figure 4.2, Figure 4.3 and Figure 4.4 show the percentage of samples that are rejected by the model throughout 100 epochs for the PTB-XL dataset. The noise settings chosen to be displayed here are no noise, symmetric noise with 20% noise rate and asymmetric noise with 40% noise rate, respectively.

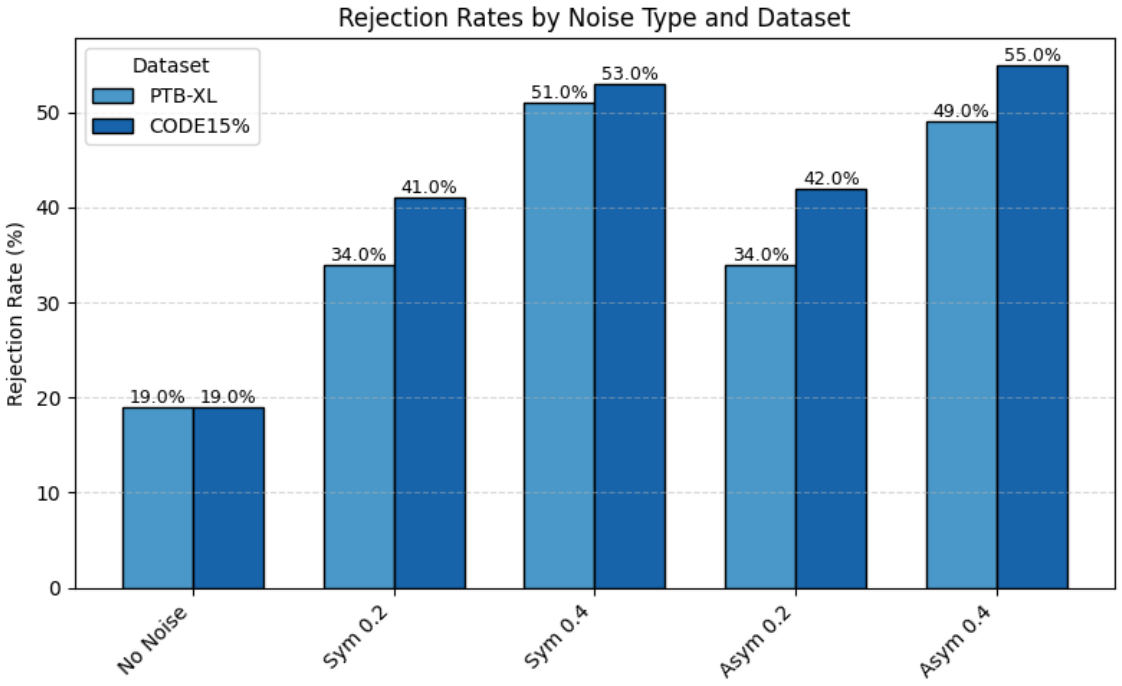


Figure 4.1: Rejection rates for both datasets grouped by all noise settings.

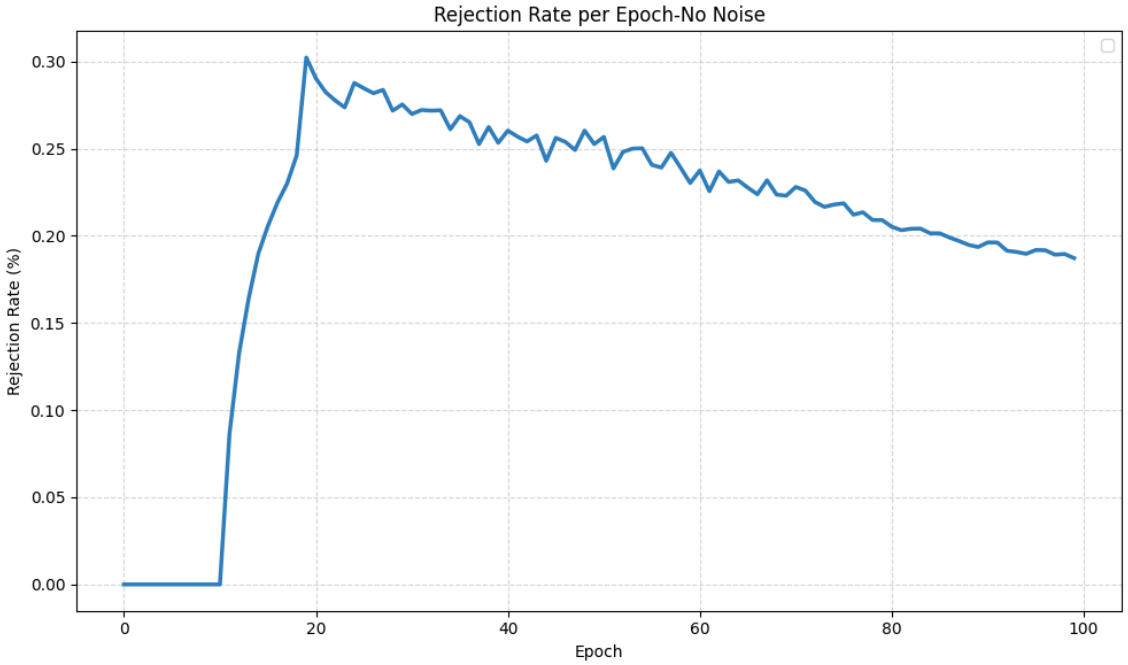


Figure 4.2: Rejection rate of samples over 100 epochs without noise any noise injection.

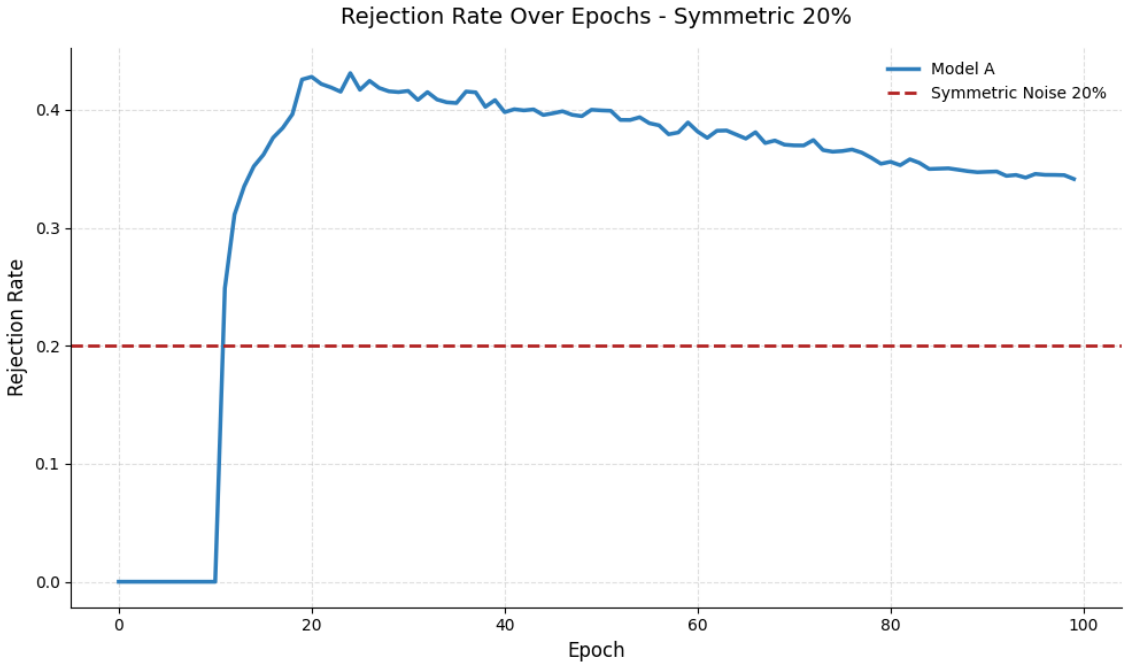


Figure 4.3: Rejection rate of samples over 100 epochs with 20% symmetric noise where the red dashed line indicates the noise percentage.

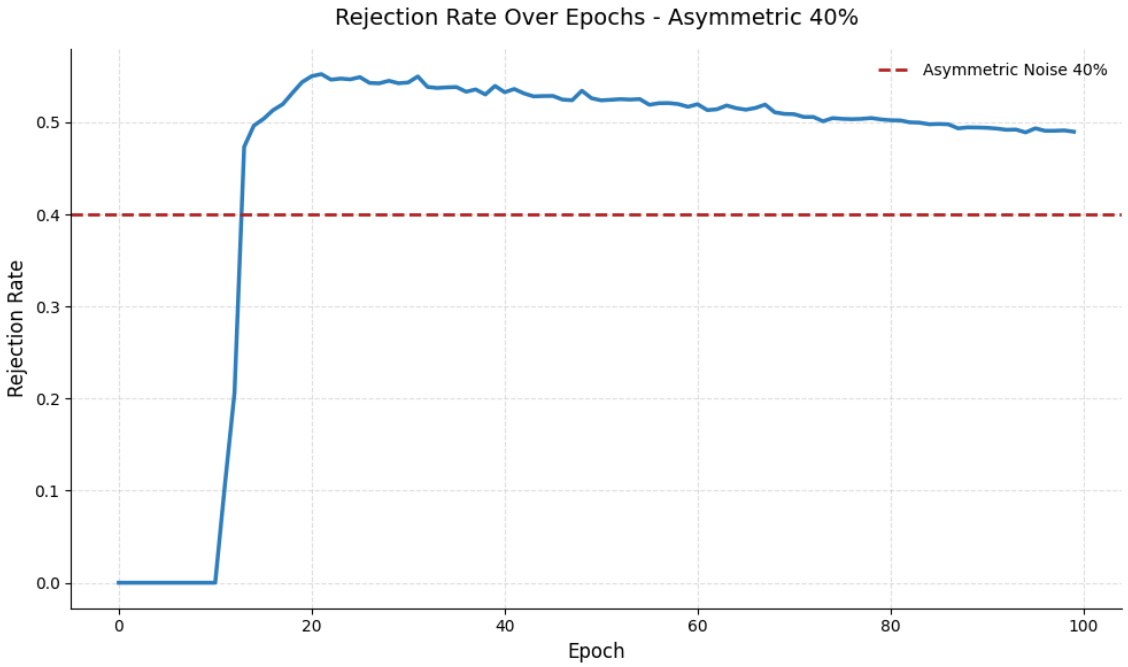
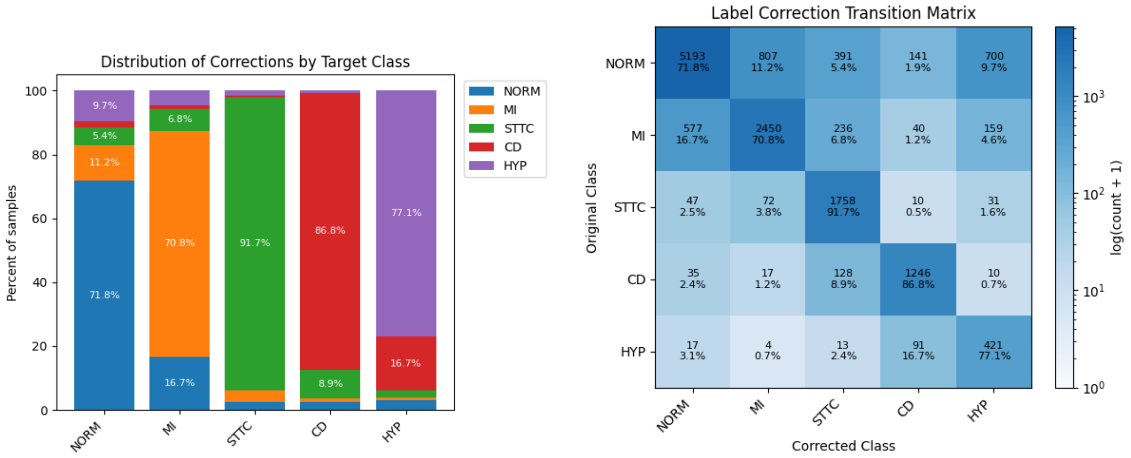


Figure 4.4: Rejection rate of samples over 100 epochs with 40% asymmetric noise where the red dashed line indicates the noise percentage.

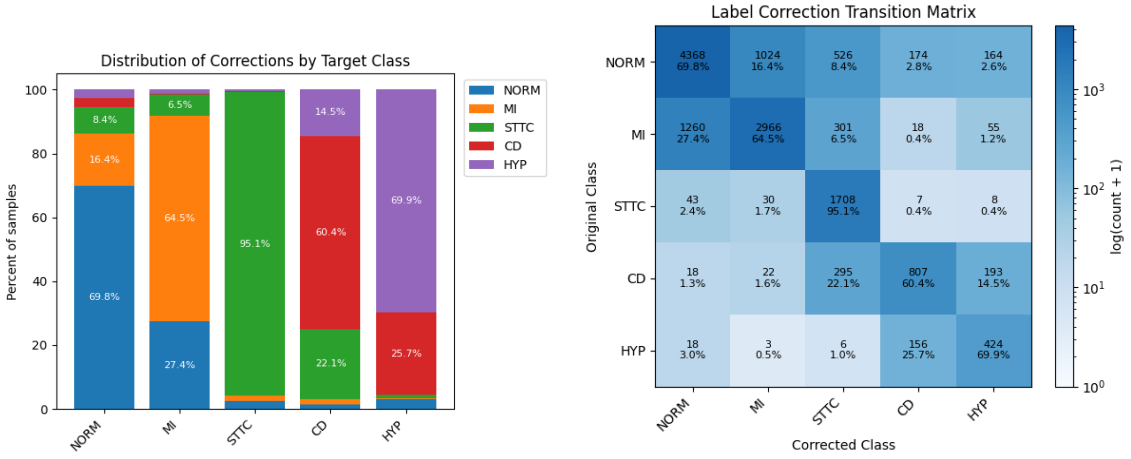
4.3 Self-Learning

The Self-learning method changes the labels that it deems as noisy to the a different class based on similarity metrics. So, investigating these label transitions can provide useful insights. Figure 4.5 and Figure 4.6 show the label transitions that Self-learning performed in the PTB-XL dataset when asymmetric noise is introduced at rates of 20% and 40%, respectively. Figure 4.7 and Figure 4.8 show the label changes performed in the CODE15% dataset when symmetric noise is introduced at the same rates. These changes can be viewed through a bar chart that presents the percentages of each label that changed or stayed the same, and through a label transition matrix with counts and percentages where the rows are the original class and the columns are the class it is changed into. Figure 4.9 and Figure 4.10 show the percentages of each label that changed for each noise type and rate in both datasets.



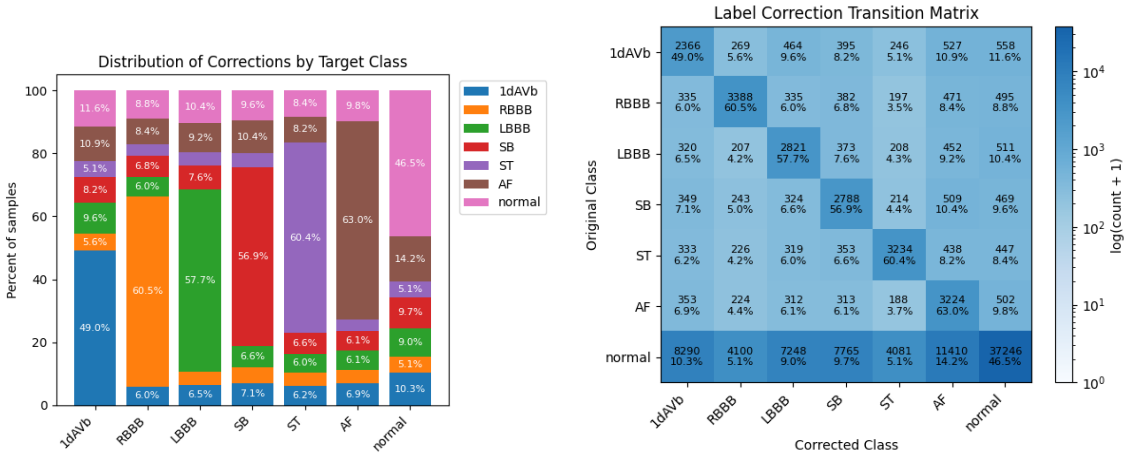
(a) Bar Chart **(b)** Transition Matrix

Figure 4.5: Percentages of label changes when 20% asymmetric noise is applied for each class at the PTB-XL dataset. On the left is the Bar Chart representation and on the right the confusion matrix.



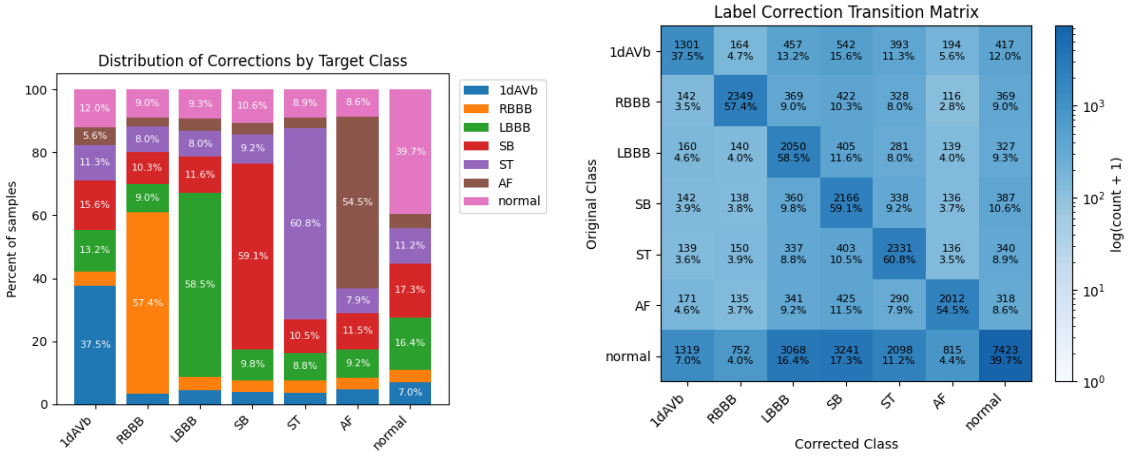
(a) Bar Chart **(b)** Transition Matrix

Figure 4.6: Percentages of label changes when 40% asymmetric noise is applied for each class at the PTB-XL dataset. On the left is the Bar Chart representation, and on the right is the confusion matrix.



(a) Bar Chart **(b)** Transition Matrix

Figure 4.7: Percentages of label changes when 20% symmetric noise is applied for each class at the CODE15% dataset. On the left is the Bar Chart representation, and on the right is the confusion matrix.



(a) Bar Chart **(b)** Transition Matrix

Figure 4.8: Percentages of label changes when 40% symmetric noise is applied for each class at the CODE15% dataset. On the left is the Bar Chart representation and on the right the confusion matrix.

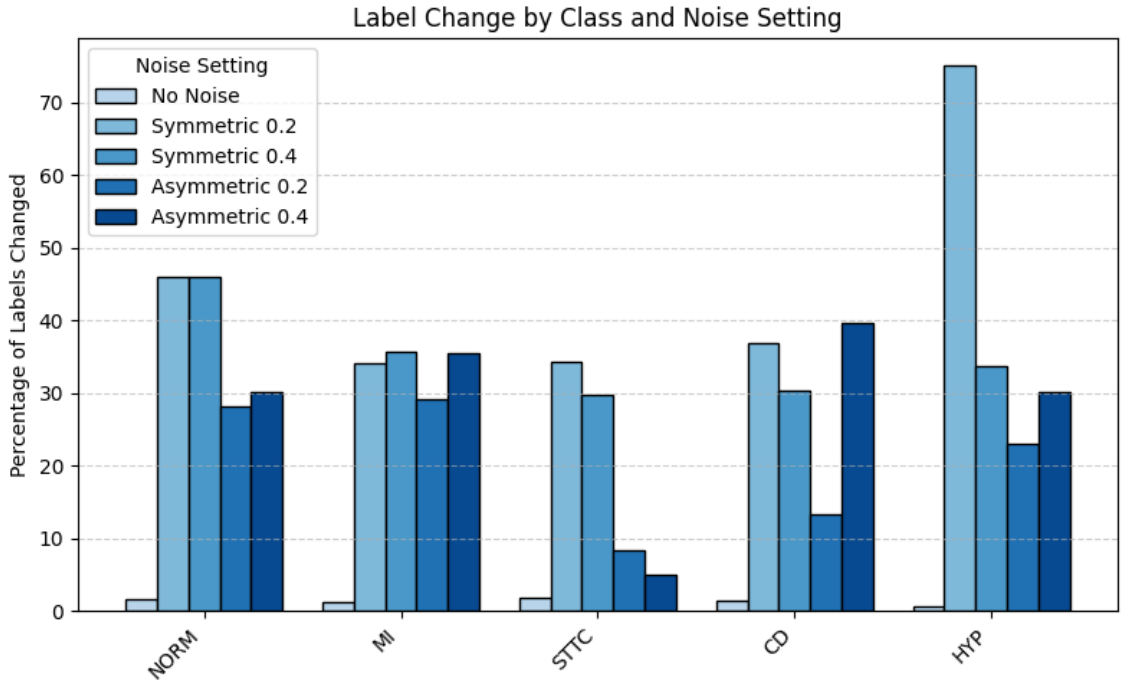


Figure 4.9: Percentages of labels changes grouped by class across all settings for the PTB-XL dataset.

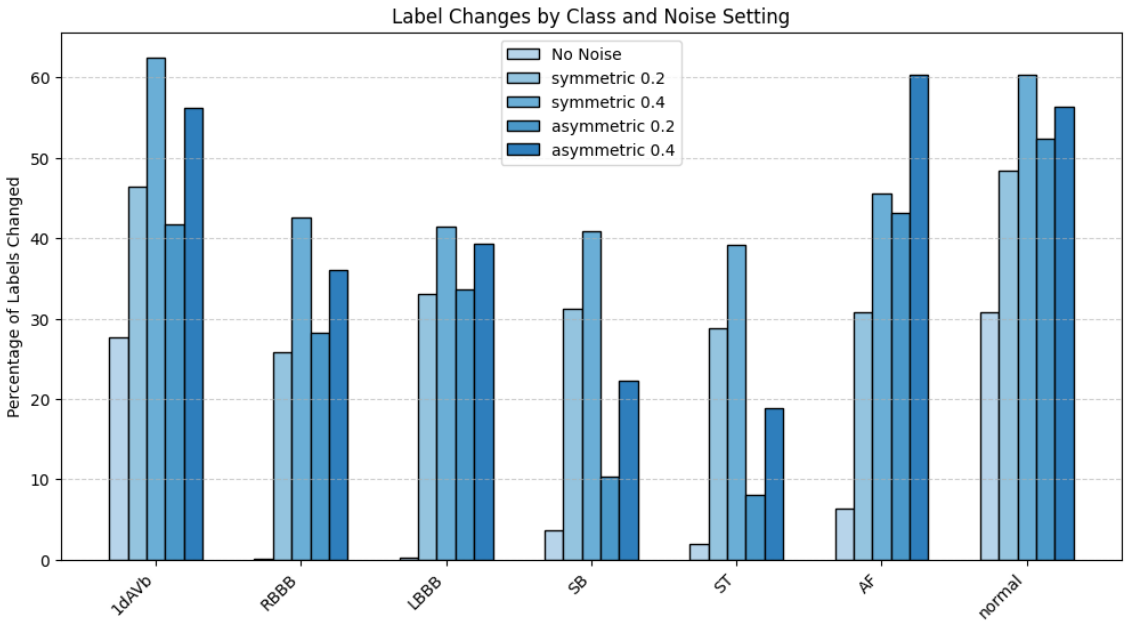


Figure 4.10: Percentages of labels changes grouped by class across all settings for the CODE15% dataset.

4.4 DivideMix

DivideMix differ from other methods, like stochastic co-teaching, where the samples are rejected, and self-learning, where the labels are refined. In DivideMix, the samples are divided into labelled and unlabelled based on the loss, using the MixUp method as the core idea. This technique helps the model with generalisation and prevents overfitting. So, it is essential to understand how the ECGs are combined using MixUp and the percentages of unlabelled or noisy samples that the model defines. In Figure 4.11, two random ECGs are selected, and a combined version is produced after using the MixUP technique. Figure 4.12 shows the percentages of unlabelled samples for both datasets across all noise settings.

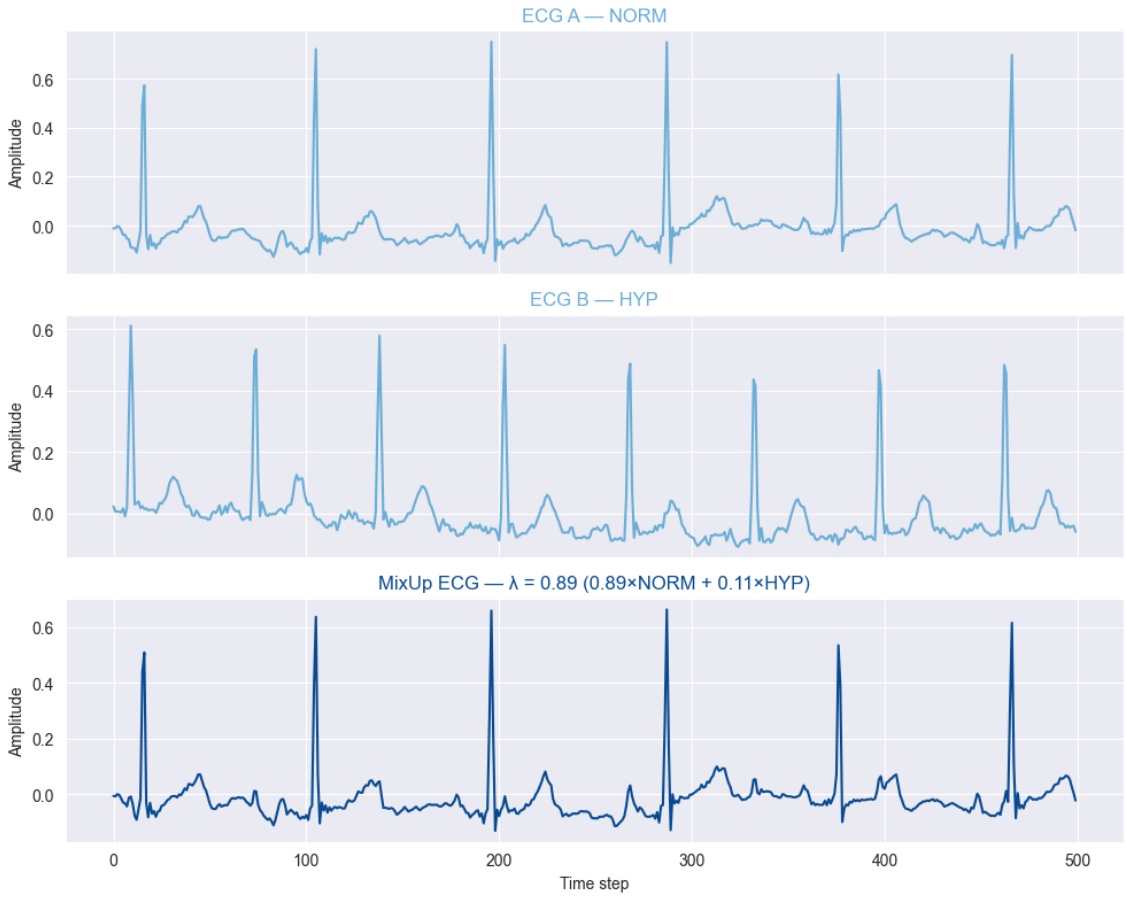


Figure 4.11: MixUP of two ECG signals for augmentation purposes used at Dividemix plotted only for a single lead.

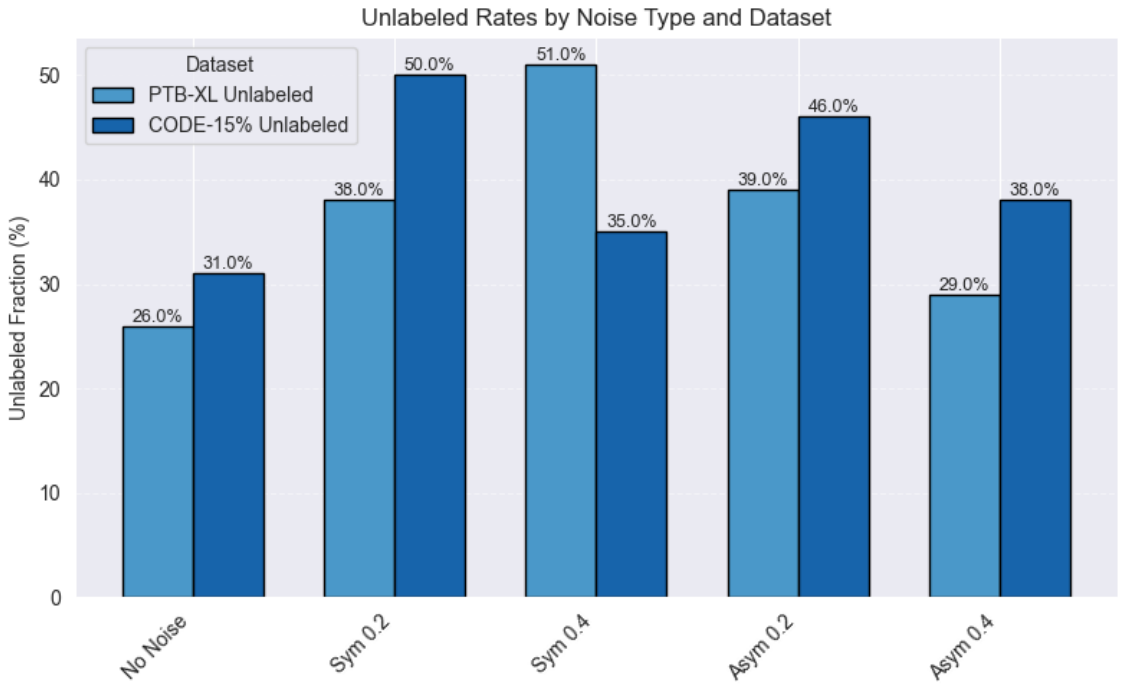


Figure 4.12: Percentages of samples that are used for semi-supervised learning after the networks decide that the sample contains a noisy label by evaluating its loss value.

4.5 Hospital dataset

The main difference between the hospital dataset and the others is that the classification is binary and labels are based off ICD-10 codes and not a posteriori labelling. Only the self-learning method is selected for evaluation, as it showed superior results in the benchmark datasets. In Figure 4.13, the prototype positions of each class are displayed after dimensionality reduction to a 2D plot. To this end and to maximize the variance to capture the most important features, Principal Component Analysis (PCA) [83] is used. The features are already normalised before the dimensionality reduction, so no additional normalisation is needed. In Figure 4.14, a label transition matrix is displayed that shows the label corrections between classes.

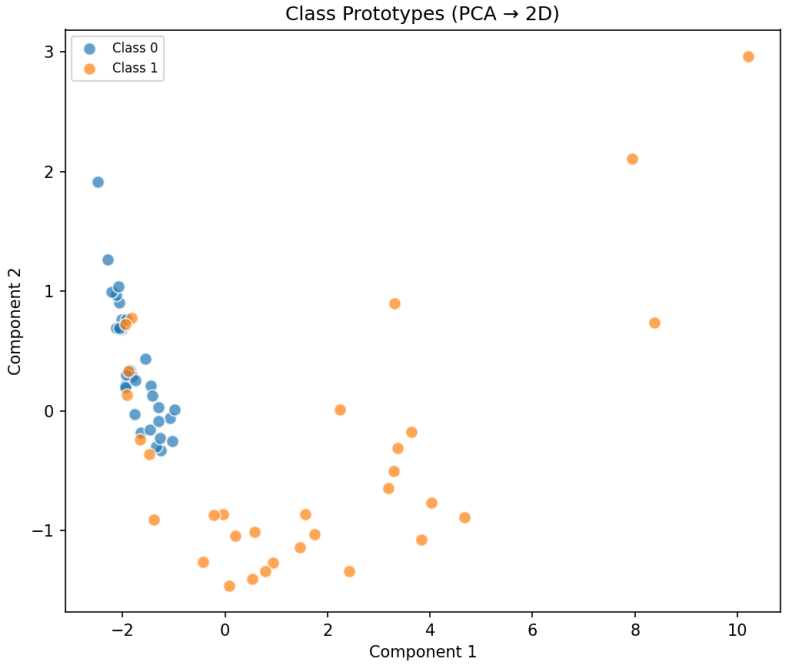


Figure 4.13: 2D plot of prototypes for each class. Class 0: normal ECG, Class 1: abnormal ECG

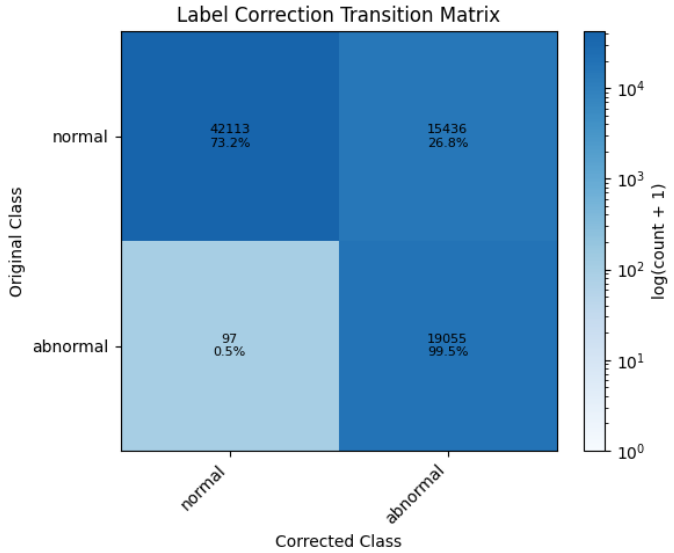


Figure 4.14: Label transition matrix showing corrected label count and percentages.

5
Discussion

In this section, all findings are explained, and how each method performs under all noise settings for the PTB-XL and CODE15% is described. Also, the performance of the best algorithm in the Hospital dataset, where the noise type and percentage are unknown, is discussed. Additionally, a comparison with prior work is conducted and finally, the limitations and future work are presented.

5.1 Method Evaluation

After presenting all the results from Chapter 4 and as can be seen from Table 4.1 and Table 4.2, the performance of the baseline model is constantly dropping in both datasets when artificial noise is introduced, which is expected since the model fits to the noisy samples causing a significant decrease in the performance. So, the memorisation effect occurs when the patterns from noisy labels are memorised during training. Therefore, the AUROC decreases when noise is applied, see Table 4.1 and Table 4.2. Specifically, for PTB-XL, when there is symmetric noise of 20% and 40%, the AUROC drops from 0.91 to 0.81 and 0.74, respectively, and for CODE 15%, it decreases from 0.93 to 0.83 and 0.80. For the asymmetric noise type, in the PTB-XL dataset, the AUROC value drops from 0.91 to 0.81 for the 20% rate and to 0.78 for the 40% rate. The CODE15% dataset demonstrates a smaller decrease in this noise type, dropping from 0.93 to 0.85 with a 20% rate and to 0.82 with a 40% rate.

In the symmetric noise case, because there is always a random chance of label transitioning into any other class, the model can struggle learning patterns that could possibly lead to degradation of the performance. When asymmetric noise is present, then each class is transitioning into a specific other class, which is constant as explained in Section 2.5.1. So the model might find it easier to learn from this type of noise, which explains the difference in the performance for the baseline model between symmetric and asymmetric noise. When comparing the noise rate percentages, the baseline model struggles more as the noise rate increases. In both datasets, the 20% noise rate shows better performance than the 40% rate. This happens because there are more noisy samples that the model learns from, which increases the probability that the memorisation effect occurs.

5.1.1 Stochastic Co-teaching

In Stochastic Co-teaching, as described at Section 2.6.1, as the two networks simultaneously train and each one decides for the other network which samples to keep

and which ones to reject based on the loss values that is then compared to a threshold drawn from beta-distribution, with values $\alpha = 32$ and $\beta = 2$, that makes the decision boundary very strict with an average threshold of 0.94. Therefore, samples might get rejected even though the model is confident with a high probability, but less than the average threshold, which is why some samples are rejected when no noise is applied.

Upon further inspection of the results, this method exhibits a large discrepancy in the AUROC values between the datasets. The main differences between the datasets are the number of classes and the number of samples. PTB-XL contains fewer classes and samples than the CODE15%. This is mainly where this difference in results is attributed. If the rejected sample percentage is high, as can be seen in Figure 4.1, the training procedure is affected more in the dataset with fewer samples. In that case, there is a greater probability that the samples of a specific class are completely rejected, which can greatly affect the model's performance since it will not be able to learn from that class. In general, Stochastic Co-teaching rejects a percentage of samples around 10-20% higher than the actual noise rate, as can be seen in Figure 4.2, Figure 4.3 and Figure 4.4. Especially in the case of no noise, the difference in AUROC between the datasets is 10%, while the rejection rate is 19% in both datasets. This means that rejecting 19% of the clean samples in the training data affects the dataset with fewer overall samples and classes more than it affects the larger dataset. In theory, running the model for a lot more epochs would make the rejection rate converge with the actual noise rate.

5.1.2 Self-learning

The Self-Learning method, instead of excluding the samples from the training process, it corrects the labels based on the similarity between the sample and the class prototypes as mentioned in Section 2.6.2. Compared to the Stochastic Co-teaching, this method uses all the samples during training instead of excluding the noisy ones. It will correct the samples that classifies as noisy into the most similar class, so the difference in the number of samples between the datasets used is not apparent in this method. Self-learning achieved the best performance across all settings for PTB-XL, successfully improving the results and mitigating high and low levels of different types of noise. For CODE15%, the method achieved decent results, improving from the baseline across all noise settings around 5-8%. In Figure 4.5 and Figure 4.6, the label transitions can be seen for the asymmetric noise type. These transitions are consistent with the noise transition matrix as introduced in Figure 3.5b. For example, after applying asymmetric noise, samples with the original class 'CD' are correctly changed into the 'STTC' class at a high rate, since this is their true class before noise is injected. In contrast, in 'STTC' class an insufficient number of labels are corrected, which can be attributed to poor prototype selection for this class. In the symmetric noise case, as demonstrated in Figure 4.7 and Figure 4.8, each class has the same probability of being corrected into another class, which is also consistent with the noise transition matrix as seen in Figure 3.5a. Ideally, the percentage of labels that should remain the same, that is, the diagonal of the transition matrices, is $(100\% - noise\_rate)$ in both noise types. This is also evident in Figure 4.9

and Figure 4.10, where the percentage of labels that are corrected across all noise settings for PTB-XL and CODE15%, respectively, is displayed. For the CODE15% dataset, when 40% noise rate is introduced, more samples have their labels changed across all classes compared to the 20% noise rate, but the percentages are usually higher than the actual noise rate. In contrast, for the PTB-XL dataset, the percentage of labels corrected is closer to the actual noise rates. However, the correction rates are not consistent with the noise rate values, meaning that higher correction rates at a noise rate of 40% compared to a rate of 20% is not always the case in this dataset. In general, the combination of label correction and the custom loss function used, which adds weight to the original and corrected labels, helps the robustness of the training procedure. By making the model less confident in its corrections and not fully trusting the original labels, the training is more consistent and less prone to overfitting to noisy labels.

5.1.3 DivideMix

DivideMix is a hybrid method that uses multiple techniques like co-train, co-guessing and a custom loss function to divide the data into two subsets. The samples that are deemed noisier, based on their loss values, are selected for an unlabelled set and trained in a semi-supervised way, while the presumed clean samples are part of the labelled set. This method utilises many techniques to mitigate label noise and the memorisation effects during training, making it computationally expensive. Label co-refinement, in particular, augments the training samples with artificial ones created from both the labelled and unlabelled subsets, see Figure 4.11. These augmentations, combined with the fact that two models are trained in parallel, make this method extremely complex and non-trivial. From Table 4.1 and Table 4.2, even though sometimes the AUROC is improved for symmetric 40% noise at both datasets, the results are inconsistent for all the other settings, showing poor performance of the specific technique. DivideMix has a similar working principle to Stochastic Co-Teaching, which uses the predicted probabilities to decide whether the sample is clean or not. The decision is made from the GMM that outputs probabilities, which are then thresholded at 0.5, to split them into labelled or unlabelled. From Figure 4.12, all the unlabelled percentages are shown based on the decision threshold. It is expected that the unlabelled percentages will be as close as the noise rate that is applied, but there is no clear correlation between the expected noise rates and the results. The reason why the DivideMix does not have an adequate performance compared to the other methods is that the hyperparameters are fine-tuned to specific image datasets used by the authors. For example, the threshold of 0.5 might need to be tweaked to better distinguish between clean and noisy samples. Also, there is a high chance that, because the pseudo-labels are generated from the two networks, if the label is predicted incorrectly, despite the MixUp feature of the algorithm, the model might still learn from noisy samples, since the custom loss function accounts for these samples as well.

5.2 Hospital Dataset findings

The hospital data contains samples labelled as normal and abnormal. The self-learning noise mitigation method is selected to evaluate the performance and is compared with a baseline model. The AUROC results are the same in both cases, as seen in Table 4.3, meaning that there is no improvement when this mitigation method is utilised. This experiment is performed to evaluate the performance of this method in a dataset with unknown levels of noise, as the dataset is automatically annotated based on ICD-10 codes and other measurements not related to ECG which is not an ideal way of annotating ECGs. In the other datasets, the assumption made is that minimal or no noise is present in the validation/test set. This assumption does not hold in the case of the hospital dataset, which can explain why the noise mitigation method has the same performance as the baseline model. Furthermore, the methods investigated in this thesis work are tested and optimised based on artificial class-dependent noise, but this dataset likely contains feature-dependent noise, which requires different optimisation strategies and typically requires increased training time to effectively address and mitigate such noise.

Regarding the Self-learning method, in Figure 4.14, all the label transitions can be seen. In particular, 26.8% of the normal labels are changed into abnormal and less than 1% are changed from abnormal to normal. This can be explained by Figure 4.13, where all prototypes for class 0 are clustered together, while for class 1 the prototypes are more scattered and some are very close to class 0. Therefore, many normal samples have high similarity with the abnormal prototypes, leading to more normal labels being changed. In theory, while the training epochs increase, the baseline performance will drop or remain unchanged. Although phenomena like grokking [84] suggest that after prolonged training, a significant improvement in accuracy can occur, this effect is delayed when noisy labels are introduced in a dataset. In contrast, the self-learning performance will increase as the model learns to identify more representative prototypes.

5.3 Limitations

One of the most important limitations of this thesis work is the lack of computational resources. When training complex AI networks, especially in different settings where multiple runs are required, adequate GPU memory is mandatory to expedite the training process. The model training is performed on either a personal computer's small GPU or on a limited-resource online platform, which made this process more time-consuming than originally expected. This is the main reason why most models are trained for no more than 100 epochs, and no hyperparameter tuning or ablation study is performed. Also, Stochastic Co-teaching and self-learning are fine-tuned and optimised for the PTB-XL, while DivideMix is tested for the CIFAR10 and CIFAR100 datasets. So, each noise mitigation technique is dataset dependent, so optimising and fine-tuning these parameters for other datasets is not trivial. Additionally, only symmetric and asymmetric types of noise are tested, feature-dependent noise is not examined in this thesis work, so the performance of the methods is un-

known for this specific type of noise. Finally, even though the datasets contained multi-labels, in this implementation, all the multi-labels are disregarded and only single-label samples are kept. Therefore, all the results shown are for single-label tasks.

5.4 Comparison with previous work

In the original article from stochastic co-teaching [18], noise injection is performed on CIFAR10, CIFAR100 and MNIST datasets, which are image datasets. To evaluate the performance in real-world scenarios, PTB-XL is used without noise injection, while in this thesis work, the noise is injected in both the PTB-XL and the CODE15% datasets. The ResNet model architecture used in this thesis work is the same as the best-performing architecture used in that article.

The Self-learning article [47] used a different model architecture to evaluate the results and investigated the PTB-XL and MNIST datasets. The custom loss function proposed is developed for multi-label classification problems, where a sample can contain more than one label, while in the current implementation, this function had to be adapted to work for single-label datasets. As a final evaluation, in the original article, experts are consulted for a visual confirmation of the corrected samples, while this is not the case in this thesis work.

The DivideMix article [43] had the most differences to this work since the noise injection and the result evaluation are performed only on image data. This means that the techniques used in this method are optimised only for image datasets and not time series ones like ECG datasets. Specifically, CIFAR-10, CIFAR-100, clothing1M and WebVision datasets are used. Many of the strategies implemented, like label co-refinement, co-guessing and various data augmentations, are not tested and evaluated in ECG datasets, so their value in these datasets is unknown without an ablation study. This can be further confirmed by the low AUROC values of this method when evaluated in the ECG datasets. In general, while DivideMix can work well in ECG datasets with a lot of unlabelled data due to its semi-supervised approach, the MixUp technique employed reduces the interpretability of the results, which is paramount in the healthcare domain. Additionally, the confidence penalty, custom loss parameters and data augmentations utilised in DivideMix require appropriate adaptations to adjust to an ECG dataset.

Finally, in every article, different noise rates are evaluated, ranging from 10% to 90%, but especially in the case of asymmetric noise, the authors concluded that noise rates of over 50% significantly degrade the model performance.

5.5 Future work

To further evaluate the true performance of the methods, the models should be trained for more epochs. Also, in this thesis, only class-dependent noise is injected, but feature-dependent noise is usually present as well. So, to better simulate more realistic conditions and evaluate the robustness of the noise mitigation techniques under that setting as well, for future work, this type of noise is recommended to be

injected. Additionally, many ECG samples are annotated with more than one label, therefore, the implementation should be adjusted for this setting. Especially for DivideMix, further parameter tuning and an ablation study are required to investigate whether the specific strategies used are improving the performance of this method. Due to resource limitations (see Section 5.3), the ECG signals are truncated to 500 samples and minimal data augmentations are applied. To prevent overfitting during training and increase the model's generalisation capabilities, more augmentations could be applied, like noise injection to the samples, magnitude scaling and time scaling [85], and the full samples should be utilised to further assess the performance of each method. Moreover, rather than simply truncating or augmenting the signals, window-based sampling can be explored to achieve effortless augmentation without additional memory usage. Similarly, leveraging the full CODE [51] dataset for benchmarking can further improve robustness, since modern AI training is heavily data-dependent. A vast research into label noise mitigation exists, and many methods have been developed to tackle this problem, so to find the best-performing technique for ECG datasets, more methods should be tested and compared. For example, PNP [28] claims that the parameters should not be dataset dependent and develops a method that confirms that assumption.

For the hospital dataset, all the methods developed here should be utilised to fairly compare the results, instead of only testing the best-performing one. Lastly, a custom noise mitigation technique can be developed, tailored for this hospital dataset, to accurately mitigate the label noise by combining the strongest techniques of each method. For example, Self-learning could be used in conjunction with the co-training technique to confirm the label corrections with both models, thus improving the confidence of the method when correcting noisy labels.

6
Conclusion

This thesis investigated the impact of label noise on the performance of machine learning models in ECG classification tasks. While many approaches have been proposed to address this issue, three representative methods, Stochastic Co-teaching, Self-learning, and DivideMix, were selected and evaluated. A baseline model was also implemented to enable fair comparison. Two benchmark ECG datasets, PTB-XL and CODE15%, were used under the assumption that both the training and test sets were initially noise-free. Artificial, class-dependent label noise was then introduced at rates of 20% and 40%, comprising both symmetric and asymmetric variants.

The results demonstrated that label noise negatively affects model performance in the absence of any noise mitigation technique. Self-learning exhibited the best performance, particularly in effectively mitigating low levels of label noise. At higher noise levels, performance differences became substantial, highlighting the potential of Self-learning in such scenarios.

Furthermore, an ECG dataset obtained from Akershus University Hospital was used to assess the robustness of the best-performing method under unknown noise conditions. However, no significant improvement was observed on this dataset, likely due to the presence of label noise in the test set as well. Overall, these methods appear to be optimised for specific datasets and, with appropriate adaptation, they may offer effective solutions to the challenges posed by noisy labels in ECG data.

Bibliography

[1] Sandra Śmigiel et al. Ecg signal classification using deep learning techniques based on the ptb-xl dataset. *Entropy*, 23(9):1121, August 2021.

[2] Douglas P Zipes. Clinical application of the electrocardiogram. *JACC*, 36(6):1746–1748, 2000.

[3] Robert T. Tung. Electrocardiographic limb leads placement and its clinical implication: Two cases of electrocardiographic illustrations. *Kansas Journal of Medicine*, 14(3):229–230, September 2021.

[4] Kyu-Hwan Jung, Hyunho Park, and Woochan Hwang. Deep learning for medical image analysis: Applications to computed tomography and magnetic resonance imaging. *Hanyang Medical Reviews*, 37:61, 01 2017.

[5] Mohamed Khalifa and Mona Albadawy. Ai in diagnostic imaging: Revolutionising accuracy and efficiency. *Computer Methods and Programs in Biomedicine Update*, 5:100146, January 2024.

[6] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen A. W. M. van der Laak, Bram van Ginneken, and Clara I. Sánchez. A survey on deep learning in medical image analysis. *Medical Image Analysis*, 42:60–88, December 2017.

[7] Awni Y. Hannun, Pranav Rajpurkar, Masoumeh Haghpanahi, Geoffrey H. Tison, Codie Bourn, Mintu P. Turakhia, and Andrew Y. Ng. Cardiologist-level arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network. *Nature Medicine*, 25(1):65–69, January 2019.

[8] James Zou, Mikael Huss, Abubakar Abid, Pejman Mohammadi, Ali Torkamani, and Amalio Telenti. A primer on deep learning in genomics. *Nature genetics*, 51(1):12–18, January 2019.

[9] Jiheum Park, Michael G. Artin, Kate E. Lee, Yoanna S. Pumpalova, Myles A. Ingram, Benjamin L. May, Michael Park, Chin Hur, and Nicholas P. Tatonetti. Deep learning on time series laboratory test results from electronic health records for early detection of pancreatic cancer. *Journal of Biomedical Informatics*, 131:104095, July 2022.

[10] Andre Esteva, Brett Kuprel, Roberto A. Novoa, Justin Ko, Susan M. Swetter, Helen M. Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115–118, February 2017.

[11] Xinwen Liu et al. Deep learning in ecg diagnosis: A review. *Knowledge-Based Systems*, 227:107187, September 2021.

[12] Temesgen Mehari and Nils Strodthoff. Advancing the state-of-the-art for ecg analysis through structured state space models, 2022.

[13] Hemaxi Narotamo, Mariana Dias, Ricardo Santos, André V. Carreiro, Hugo Gamboa, and Margarida Silveira. Deep learning for ecg classification: A comparative study of 1d and 2d representations and multimodal fusion approaches. *Biomedical Signal Processing and Control*, 93:106141, July 2024.

[14] Negin Alamatsaz, Leyla s Tabatabaei, Mohammadreza Yazdchi, Hamidreza Payan, Nima Alamatsaz, and Fahimeh Nasimi. A lightweight hybrid cnn-lstm model for ecg-based arrhythmia detection, 2022.

[15] Manjur Kolhar and Ahmed M. Al Rajeh. Deep learning hybrid model ecg classification using alexnet and parallel dual branch fusion network model. *Scientific Reports*, 14(1):26919, November 2024.

[16] Jialin Shi, Kailai Zhang, Chenyi Guo, Youquan Yang, Yali Xu, and Ji Wu. A survey of label-noise deep learning for medical image analysis. *Medical Image Analysis*, 95:103166, July 2024.

[17] Yanrui Jin, Zhiyuan Li, Mengxiao Wang, Jinlei Liu, Yuanyuan Tian, Yunqing Liu, Xiaoyang Wei, Liqun Zhao, and Chengliang Liu. Cardiologist-level interpretable knowledge-fused deep neural network for automatic arrhythmia diagnosis. *Communications Medicine*, 4:31, 2024.

[18] Bob D. De Vos, Gino E. Jansen, and Ivana Išgum. Stochastic co-teaching for training neural networks with unknown levels of label noise. *Scientific Reports*, 13(1):16875, October 2023.

[19] Matthew A Reyna, Nadi Sadr, Erick A Perez Alday, Annie Gu, Amit J Shah, Chad Robichaux, Ali Bahrami Rad, Andoni Elola, Salman Seyedi, Sardar Ansari, Hamid Ghanbari, Qiao Li, Ashish Sharma, and Gari D Clifford. Issues in the automated classification of multilead ecgs using heterogeneous labels and populations. *Physiological Measurement*, 43(8):1081, 2022.

[20] Emily Foreman Dewar Finlay Raymond Bond Peter Doggart, Alan Kennedy. Automated identification of label errors in large electrocardiogram datasets. In *Computing in Cardiology*, volume 49, pages 1–4. IEEE, 2022.

[21] Antônio H Ribeiro, Manoel Horta Ribeiro, Gabriela M M Paixão, Derick M Oliveira, Paulo R Gomes, Jéssica A Canazart, Milton P S Ferreira, Wagner Meira Jr., and Antonio Luiz P Ribeiro. Automatic diagnosis of the 12-lead ecg using a deep neural network. *Nature Communications*, 11:1760, 2020.

[22] Yishu Wei, Yu Deng, Cong Sun, Mingquan Lin, Hongmei Jiang, and Yifan Peng. Deep learning with noisy labels in medical prediction problems: a scoping review, 2024.

[23] Görkem Algan and İlkay Ulusoy. Label noise types and their effects on deep learning. (arXiv:2003.10471), March 2020. arXiv:2003.10471 [cs].

[24] Jongmin Shin, Jonghyeon Won, Hyun-Suk Lee, and Jang-Won Lee. A review on label cleaning techniques for learning with noisy labels. *ICT Express*, 10(6):1315–1330, December 2024.

[25] Zhongheng Zhang. Introduction to machine learning: k-nearest neighbors. *Annals of Translational Medicine*, 4(11):218, June 2016.

[26] Bo Han, Quanming Yao, Xingrui Yu, Gang Niu, Miao Xu, Weihua Hu, Ivor Tsang, and Masashi Sugiyama. Co-teaching: Robust training of deep neural networks with extremely noisy labels. (arXiv:1804.06872), October 2018. arXiv:1804.06872 [cs].

[27] Eran Malach and Shai Shalev-Shwartz. Decoupling "when to update" from "how to update". In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[28] Zeren Sun, Fumin Shen, Dan Huang, Qiong Wang, Xiangbo Shu, Yazhou Yao, and Jinhui Tang. Pnp: Robust learning from noisy labels by probabilistic noise prediction. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, page 5301–5310, June 2022.

[29] Xinshao Wang, Yang Hua, Elyor Kodirov, Sankha Subhra Mukherjee, David A. Clifton, and Neil M. Robertson. Proselflc: Progressive self label correction towards a low-temperature entropy state, 2022.

[30] Eric Arazo, Diego Ortego, Paul Albert, Noel E. O'Connor, and Kevin McGuinness. Unsupervised label noise modeling and loss correction, 2019.

[31] Mingcai Chen, Hao Cheng, Yuntao Du, Ming Xu, Wenyu Jiang, and Chongjun Wang. Two wrongs don't make a right: Combating confirmation bias in learning with label noise, 2023.

[32] Dara Bahri, Heinrich Jiang, and Maya Gupta. Deep k-nn for noisy labels, 2020.

[33] Jialin Shi, Zheng Cao, and Ji Wu. Meta joint optimization: a holistic framework for noisy-labeled visual recognition. *Applied Intelligence*, 52(1):875–888, January 2022.

[34] Yuanpeng Tu, Boshen Zhang, Yuxi Li, Liang Liu, Jian Li, Yabiao Wang, Chengjie Wang, and Cai Rong Zhao. Learning from noisy labels with decoupled meta label purifier, 2023.

[35] Pengxiang Wu, Songzhu Zheng, Mayank Goswami, Dimitris Metaxas, and Chao Chen. A topological filter for learning with label noise, 2022.

[36] Taehyeon Kim, Jongwoo Ko, Sangwook Cho, Jinhwan Choi, and Se-Young Yun. Fine samples for learning with noisy labels, 2021.

[37] Daiki Tanaka, Daiki Ikami, Toshihiko Yamasaki, and Kiyoharu Aizawa. Joint optimization framework for learning with noisy labels, 2018.

[38] Kun Yi, Guo-Hua Wang, and Jianxin Wu. Pencil: Deep learning with noisy labels, 2022.

[39] Sheng Liu, Kangning Liu, Weicheng Zhu, Yiqiu Shen, and Carlos Fernandez-Granda. Adaptive early-learning correction for segmentation from noisy annotations, 2022.

[40] Songbai Jin, Wenkai Lu, and Patrice Monkam. Deep neural network-based noisy pixel estimation for breast ultrasound segmentation. In *2022 IEEE International Conference on Image Processing (ICIP)*, pages 1776–1780, 2022.

[41] Runping Hou Wangyuan Zhao Lu Qiu, Lu Zhao. Hierarchical multimodal fusion framework based on noisy label learning and attention mechanism for cancer classification with pathology and genomic features. *Computerized Medical Imaging and Graphics*, 104:102176, March 2023.

[42] Jiangfan Han, Ping Luo, and Xiaogang Wang. Deep self-learning from noisy labels. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[43] Junnan Li, Richard Socher, and Steven C. H. Hoi. Dividemix: Learning with noisy labels as semi-supervised learning, 2020.

[44] Zizhao Zhang, Han Zhang, Sercan O. Arik, Honglak Lee, and Tomas Pfister. Distilling effective supervision from severe label noise, 2020.

[45] Cheng Xue, Lequan Yu, Pengfei Chen, Qi Dou, and Pheng-Ann Heng. Robust medical image classification from noisy labeled data with global and local representation guided co-training. *IEEE Transactions on Medical Imaging*, 41(6):1371–1382, 2022.

[46] Zehui Liao, Yutong Xie, Shishuai Hu, and Yong Xia. Learning from ambiguous labels for lung nodule malignancy prediction. *IEEE Transactions on Medical Imaging*, 41(7):1874–1884, 2022.

[47] Cristina Gallego Vázquez, Alexander Breuss, Oriella Gnarra, Julian Portmann, Antonio Madaffari, and Giulia Da Poian. Label noise and self-learning label correction in cardiac abnormalities classification. *Physiological Measurement*, 43(9):094001, September 2022.

[48] Ruixuan Xiao, Yiwen Dong, Haobo Wang, Lei Feng, Runze Wu, Gang Chen, and Junbo Zhao. Promix: Combating label noise via maximizing clean sample utility. In Edith Elkind, editor, *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, pages 4442–4450. International Joint Conferences on Artificial Intelligence Organization, 8 2023. Main Track.

[49] Patrick Wagner, Nils Strodthoff, Ralf-Dieter Bousseljot, Wojciech Samek, and Tobias Schaeffter. Ptb-xl, a large publicly available electrocardiography dataset.

[50] Antônio H. Ribeiro, Gabriela M.M. Paixao, Emilly M. Lima, Manoel Horta Ribeiro, Marcelo M. Pinto Filho, Paulo R. Gomes, Derick M. Oliveira, Wagner Meira Jr, Thömas B Schon, and Antonio Luiz P. Ribeiro. Code-15June 2021.

[51] Antonio H. Ribeiro, Manoel Horta Ribeiro, Gabriela M. Paixão, Derick M. Oliveira, Paulo R. Gomes, Jéssica A. Canazart, Milton P. Ferreira, Carl R. Andersson, Peter W. Macfarlane, Wagner Meira Jr., Thomas B. Schön, and Antonio Luiz P. Ribeiro. CODE dataset. 11 2021.

[52] Arian Ranjbar, Elias Stenhede, Jesper Ravn, and Henrik Schirmer. Heart failure detection in electrocardiograms using artificial intelligence. January 2025.

[53] Yang Song, Alexander Schwing, Richard, and Raquel Urtasun. Training deep neural networks via direct loss minimization. In *Proceedings of The 33rd International Conference on Machine Learning*, page 2169–2177. PMLR, June 2016.

[54] Zhilu Zhang and Mert Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[55] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. (arXiv:1811.03378), November 2018. arXiv:1811.03378 [cs].

[56] Yaqoob Ansari, Omar Mourad, Khalid Qaraqe, and Erchin Serpedin. Deep learning for ecg arrhythmia detection and classification: an overview of progress for period 2017–2023. *Frontiers in Physiology*, 14, September 2023.

[57] Zhen Zeng, Rachneet Kaur, Suchetha Siddagangappa, Saba Rahimi, Tucker Balch, and Manuela Veloso. Financial time series forecasting using cnn and transformer. (arXiv:2304.04912), April 2023. arXiv:2304.04912 [cs].

[58] Ananya Jain, Aviral Bhardwaj, Kaushik Murali, and Isha Surani. A comparative study of cnn, resnet, and vision transformers for multi-classification of chest diseases. (arXiv:2406.00237), May 2024. arXiv:2406.00237 [eess].

[59] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. (arXiv:1512.03385), December 2015. arXiv:1512.03385 [cs].

[60] R. Anand, S. Vijaya Lakshmi, Digvijay Pandey, and Binay Kumar Pandey. An enhanced resnet-50 deep learning model for arrhythmia detection using electrocardiogram biomedical indicators. *Evolving Systems*, 15(1):83–97, February 2024.

[61] Hwanjun Song, Minseok Kim, Dongmin Park, Yooju Shin, and Jae-Gil Lee. Learning from noisy labels with deep neural networks: A survey. (arXiv:2007.08199), March 2022. arXiv:2007.08199 [cs].

[62] Davood Karimi, Haoran Dou, Simon K. Warfield, and Ali Gholipour. Deep learning with noisy labels: Exploring techniques and remedies in medical image analysis. *Medical Image Analysis*, 65:101759, October 2020.

[63] Diane Oyen, Michal Kucer, Nick Hengartner, and Har Simrat Singh. Robustness to label noise depends on the shape of the noise distribution in feature space. (arXiv:2206.01106), June 2022. arXiv:2206.01106 [cs].

[64] Gino Jansen. Gejansen/stochastic-coteaching, August 2024.

[65] Beta distribution - an overview | sciencedirect topics.

[66] Alfirna Rizqi Lahitani, Adhistya Erna Permanasari, and Noor Akhmad Setiawan. Cosine similarity to determine similarity measure: Study case in online essay assessment. In *2016 4th International Conference on Cyber and IT Service Management*, page 1–6, April 2016.

[67] Junnan Li. Lijunnan1992/dividemix, April 2025.

[68] Douglas Reynolds. Gaussian mixture models. *Encyclopedia of Biometrics*, 01 2008.

[69] David Berthelot, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin Raffel. Mixmatch: A holistic approach to semi-supervised learning. (arXiv:1905.02249), October 2019. arXiv:1905.02249 [cs].

[70] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization, 2018.

[71] Jonathon Shlens. Notes on kullback-leibler divergence and likelihood, 2014.

[72] Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks: A strong baseline. (arXiv:1611.06455), December 2016. arXiv:1611.06455 [cs].

[73] Florentin Bieder, Robin Sandkühler, and Philippe C. Cattin. Comparison of methods generalizing max- and average-pooling. (arXiv:2103.01746), March 2021. arXiv:2103.01746 [cs].

[74]

[75] S. Gopal Krishna Patro and Kishore Kumar Sahu. Normalization: A preprocessing stage. (arXiv:1503.06462), March 2015. arXiv:1503.06462 [cs].

[76] David A. Cook, So Young Oh, and Martin V. Pusic. Accuracy of physicians' electrocardiogram interpretations: A systematic review and meta-analysis. *JAMA internal medicine*, 180(11):1461–1471, November 2020.

[77] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. (arXiv:1412.6980), January 2017. arXiv:1412.6980 [cs].

[78] Francesco D'Angelo, Maksym Andriushchenko, Aditya Varre, and Nicolas Flammarion. Why do we need weight decay in modern deep learning? (arXiv:2310.04415), November 2024. arXiv:2310.04415 [cs].

[79] Leslie N. Smith and Nicholay Topin. Super-convergence: Very fast training of neural networks using large learning rates. (arXiv:1708.07120), May 2018. arXiv:1708.07120 [cs].

[80] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. (arXiv:1608.03983), May 2017. arXiv:1608.03983 [cs].

[81] Pablo Andretta Jaskowiak, Ivan Gesteira Costa, and Ricardo José Gabrielli Barreto Campello. The area under the roc curve as a measure of clustering quality. *Data Mining and Knowledge Discovery*, 36(3):1219–1245, 2022. arXiv:2009.02400 [cs].

[82] J. A. Hanley and B. J. McNeil. The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology*, 143(1):29–36, April 1982.

[83] Ian T. Jolliffe and Jorge Cadima. Principal component analysis: a review and recent developments. *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences*, 374(2065):20150202, April 2016.

[84] Ahmed Imtiaz Humayun, Randall Balestriero, and Richard Baraniuk. Deep networks always grok and here is why. (arXiv:2402.15555), June 2024. arXiv:2402.15555 [cs].

[85] Aniruddh Raghu, Divya Shanmugam, Eugene Pomerantsev, John Guttag, and Collin M. Stultz. Data augmentation for electrocardiograms. (arXiv:2204.04360), April 2022. arXiv:2204.04360 [cs].

A

Appendix 1

A.1 PTB-XL pre-process and dataloader

```
1
2
3
4  class PTBXL_CE_Self_Learning(Dataset):
5      """
6      PTB-XL dataset loader for multi-class classification.
7      pre-processes the dataset and injects symmetric or asymmetric noise.
8      """
9      def __init__(self, split, transform=None, data_dir=None, hr=False, debug=False,
10                  noise_type=None, noise_rate=0.0, random_state=42):
11         self.transform = transform
12         self.noise_type = noise_type
13         self.noise_rate = noise_rate
14         self.random_state = random_state
15
16
17         classes = np.array(['NORM', 'MI', 'STTC', 'CD', 'HYP'])
18         df_statements = pd.read_csv(data_dir / 'scp_statements.csv', index_col=0)
19         df_statements = df_statements[df_statements.diagnostic == 1]
20         df_statements['diagnosis_label'] = df_statements.diagnostic_class.apply(
21             lambda x: int(np.squeeze(np.where(classes == x)))
22         )
23
24         def aggregate_diagnostic_to_encoding(y_dic):
25             tmp = []
26             for key in y_dic.keys():
27                 if key in df_statements.index:
28                     tmp.append(df_statements.loc[key].diagnosis_label)
29             # skip multi-label samples
30             if len(set(tmp)) == 1:
31                 return list(set(tmp))[0]
32             else:
33                 return -1  # set -1 multi-label samples
34
35
36
37
38         df_ptbxl = pd.read_csv(data_dir / 'ptbxl_database.csv', index_col='ecg_id')
39         if split == 'train':
40             df_ptbxl = df_ptbxl[df_ptbxl.strat_fold <= 9]
41         elif split == 'test':
42             df_ptbxl = df_ptbxl[df_ptbxl.strat_fold == 10]
43
44
45         df_ptbxl.scp_codes = df_ptbxl.scp_codes.apply(lambda x: ast.literal_eval(x)
               )
46         df_ptbxl['diagnosis_label'] = df_ptbxl.scp_codes.apply(
               aggregate_diagnostic_to_encoding)
47         df_ptbxl = df_ptbxl[df_ptbxl.diagnosis_label != -1]  # remove multi-label
               samples
48
49         samples = []
```

```
        labels_clean = []
        for ecg_id, row in tqdm(df_ptbxl.iterrows(), total=len(df_ptbxl), desc=f'
            Loading {split} set'):
            fname = row.filename_hr if hr else row.filename_lr
            data, meta = wfdb.rdsamp(str(data_dir / fname))
            # Transpose so that shape becomes (channels, time)
            data = data.T.astype(np.float32)
            y_clean = int(row['diagnosis_label'])
            samples.append(dict(
                data=data,
                fs=meta['fs'],
                pid=ecg_id,
                clean_y=y_clean  # store clean label
            ))
            labels_clean.append(y_clean)

        # If training and noise is specified, inject noise.
        if split == 'train' and noise_type is not None and noise_rate > 0:
            labels_clean = np.array(labels_clean)
            if noise_type == 'assymetric':
                labels_noisy, actual_noise = self.noisify_assymetric(labels_clean,
                    noise_rate, random_state, nb_classes=len(classes))
            elif noise_type == 'symmetric':
                labels_noisy, actual_noise = self.noisify_multiclass_symmetric(
                    labels_clean, noise_rate, random_state, nb_classes=len(classes)
                    )
            else:
                labels_noisy, actual_noise = labels_clean, 0.0
            print(f"Applied noise: {actual_noise*100:.2f}% of labels flipped.")
            for i in range(len(samples)):
                samples[i]['y'] = int(labels_noisy[i])
        else:
            for sample in samples:
                sample['y'] = sample['clean_y']

        self.samples = samples


    @staticmethod
    def multiclass_noisify(y, P, random_state=0):
        """Flip classes according to transition probability matrix P."""
        assert P.shape[0] == P.shape[1]
        m = y.shape[0]
        new_y = y.copy()
        flipper = np.random.RandomState(random_state)
        for idx in range(m):
            i = y[idx]
            flipped = flipper.multinomial(1, P[i, :], 1)[0]
            new_y[idx] = np.where(flipped == 1)[0][0]
        return new_y

    @staticmethod
    def noisify_multiclass_symmetric(y, noise, random_state=None, nb_classes=5):
        """uniformly random label flipping (all classes equally likely)."""
        P = np.ones((nb_classes, nb_classes)) * (noise / (nb_classes - 1))
        np.fill_diagonal(P, 1. - noise)
        if noise > 0.0:
            y_noisy = PTBXL_CE_Self_Learning.multiclass_noisify(y, P=P,
                random_state=random_state)
            actual_noise = (y_noisy != y).mean()
            print(f"Applied symmetric noise: {actual_noise*100:.2f}%")
            return y_noisy, actual_noise
        return y, 0.0

    @staticmethod
    def noisify_assymetric(y, noise, random_state=None, nb_classes=5):
        """
        asymmetric noise: for each class, flip with probability `noise` to a
            specific wrong class.
        transition matrix is defined as follows:
```

```
            0: NORM -> 1: MI,
            1: MI -> {0: NORM, 2: STTC} (choose one),
            2: STTC -> {0: NORM, 3: CD},
            3: CD -> {0: NORM, 4: HYP},
            4: HYP -> 0: NORM
        """
        transitions = {
            0: [1],
            1: [0, 2],
            2: [0, 3],
            3: [0, 4],
            4: [0]
        }
        rng = np.random.RandomState(random_state)
        y_noisy = y.copy()
        for idx, label in enumerate(y):
            if label in transitions and rng.rand() < noise:
                new_label = rng.choice(transitions[label])
                y_noisy[idx] = new_label
        actual_noise = (y_noisy != y).mean()
        print(f"Applied asymmetric noise: {actual_noise*100:.2f}%")
        return y_noisy, actual_noise

    def __len__(self):
        return len(self.samples)

    def __getitem__(self, idx):
        sample = copy.deepcopy(self.samples[idx])
        if self.transform:
            sample = self.transform(sample)
        return sample


    class NormalizeECG:
    def __init__(self, method="z-score", mean=None, std=None, min_val=None, max_val
        =None):
        self.method = method
        self.mean = mean
        self.std = std
        self.min_val = min_val
        self.max_val = max_val

    def fit(self, dataset):
        # Concatenate all samples along time axis (axis=1)
        all_data = np.concatenate([sample['data'] for sample in dataset.samples],
            axis=1)
        if self.method == "z-score":
            self.mean = np.mean(all_data, axis=1, keepdims=True)
            self.std = np.std(all_data, axis=1, keepdims=True)
        elif self.method == "min-max":
            self.min_val = np.min(all_data, axis=1, keepdims=True)
            self.max_val = np.max(all_data, axis=1, keepdims=True)

    def __call__(self, sample):
        data = sample['data']
        if self.method == "z-score" and self.mean is not None and self.std is not
            None:
            data = (data - self.mean) / (self.std + 1e-8)
        elif self.method == "min-max" and self.min_val is not None and self.max_val
             is not None:
            data = (data - self.min_val) / (self.max_val - self.min_val + 1e-8)
        sample['data'] = data
        return sample

class Sample:
    def __init__(self, size, rs=np.random):
        self.size = size
        self.rs = rs

    def __call__(self, sample):
```

```
        data = sample['data']
        overset = data.shape[1] - self.size
        if overset > 0:
            start_idx = self.rs.randint(overset)
            data = data[:, start_idx:start_idx + self.size]
        sample['data'] = data
        return sample
```

A.2 CODE15% pre-process and dataloader

```


class MiddleSample:
    """Extracts a fixed-length segment from the center of ECG data."""
    def __init__(self, size):
        self.size = size  # Target sample size

    def __call__(self, data):
        """Extracts the middle portion of the signal."""
        original_size = data.shape[1]  # Time dimension (4096)
        center = original_size // 2  # Middle index
        start_idx = max(0, center - self.size // 2)  # Ensure start index is non-
            negative
        end_idx = min(original_size, start_idx + self.size)  # Ensure within bounds
        return data[:, start_idx:end_idx]

class ECGDataset_multi(Dataset):
    def __init__(self, h5_path, noise_type="none", noise_rate=0.0, window_size=500,
         normalize=True):
        """
        ECG Dataset for **multiclass classification** with softmax.

        Args:
            h5_path (str): Path to the HDF5 file.
            window_size (int): Number of time steps to keep from each sample.
            normalize (bool): Whether to apply normalization.
            noise_type (str): Type of label noise ("symmetric", "asymmetric", or "
                none").
            noise_rate (float): Percentage of samples to corrupt.
        """
        self.h5_path = h5_path
        self.file = h5py.File(h5_path, "r", swmr=True)  # Keep file open
        self.dataset_size = self.file["x"].shape[0]
        self.window_size = window_size
        self.normalize = normalize
        self.noise_type = noise_type
        self.noise_rate = noise_rate
        self.sampler = MiddleSample(window_size)

        # Load labels (first 6 columns only, ignoring "Normal ECG" column)
        labels = np.array(self.file["y"])  # Shape: (num_samples, 7)

        # Keep only single-label samples
        single_label_mask = labels.sum(axis=1) == 1  # Boolean mask: True if
            exactly one label is active
        self.labels = labels[single_label_mask]
        self.data_indices = np.where(single_label_mask)[0]  # Indices of valid
            samples

        # Convert multi-hot labels to class indices
        self.labels = np.argmax(self.labels, axis=1)

        # Introduce noise
        if self.noise_type == "symmetric":
            print("Applying symmetric noise...")
            self.labels = self.apply_exact_noise(self.labels, self.noise_rate)
```

```
        elif self.noise_type == "asymmetric":
            print("Applying asymmetric noise...")
            label_transitions = {
                0: [3,6],  # 1dAVb     normal
                1: [2,6],  # RBBB     LBBB
                2: [1,6],  # LBBB     RBBB
                3: [0,6],  # SB     1dAVb
                4: [5,6],  # ST     AF
                5: [4,6],  # AF     ST
                6: [0,5]   # normal     AF
            }
            self.labels = self.apply_exact_noise(self.labels, self.noise_rate,
                label_transitions)

        # Compute mean & std for normalization
        if self.normalize:
            self.mean, self.std = self.compute_mean_std()

        self.class_weights = self.compute_class_weights(self)  # Store in class
            attribute

    def __len__(self):
        return len(self.data_indices)  # Updated size after filtering

    def compute_mean_std(self):
        """Compute mean & std across all ECG data for normalization."""
        mean_sum, std_sum, count = 0.0, 0.0, 0
        for idx in self.data_indices:
            x = self.file["x"][idx]  # Shape: (4096, 12)
            mean_sum += x.mean(axis=0)  # Per-channel mean
            std_sum += x.std(axis=0)  # Per-channel std
            count += 1
        mean = mean_sum / count
        std = std_sum / count
        std[std == 0] = 1  # Prevent division by zero
        return mean, std

    def __getitem__(self, idx):
        real_idx = self.data_indices[idx]  # Get original index from filtered
            dataset
        x = torch.tensor(self.file["x"][real_idx], dtype=torch.float32).permute(1,
            0)  # Shape: (12, 4096)
        # Extract middle portion
        x = self.sampler(x)  # Shape now (12, window_size)
        # Apply normalization (per channel)
        if self.normalize:
            x = (x - torch.tensor(self.mean, dtype=torch.float32)[:, None]) / torch
                .tensor(self.std, dtype=torch.float32)[:, None]
        y = torch.tensor(self.labels[idx], dtype=torch.long)  # Single class index
            for softmax
        return x, y

    @staticmethod
    def compute_class_weights(self):
        """Compute dynamic class weights based on dataset labels."""
        class_counts = np.bincount(self.labels, minlength=7)  # Count occurrences
            of each class
        total_samples = len(self.labels)

        # Avoid division by zero
        class_weights = total_samples / (class_counts + 1e-6)
        # Convert to PyTorch tensor
        class_weights = torch.tensor(class_weights, dtype=torch.float32)

        # log-scaling to dampen extreme weights
        log_weights = torch.log(class_weights + 1)
        # Normalize weights (optional)
        normalized_weights = log_weights / log_weights.max()

        return normalized_weights
```

```
115
116     @staticmethod
117     def apply_exact_noise(y_train, noise_rate, transition_matrix=None):
118         """
119         Ensures exactly `noise_rate` fraction of samples are modified.
120
121         Args:
122             y_train (np.ndarray): Class indices (shape: num_samples).
123             noise_rate (float): Fraction of samples to corrupt.
124             transition_matrix (dict, optional): Class transitions for asymmetric
                    noise.
125
126         Returns:
127             np.ndarray: Noisy labels.
128         """
129         y_noisy = y_train.copy()
130         num_samples = len(y_train)
131
132         num_noisy_samples = int(noise_rate * num_samples)
133         if num_noisy_samples > 0:
134             noisy_indices = np.random.choice(num_samples, num_noisy_samples,
                    replace=False)
135
136             for idx in noisy_indices:
137                 original_label = y_train[idx]
138
139                 if transition_matrix:  # Asymmetric noise
140                     if original_label in transition_matrix:
141                         y_noisy[idx] = np.random.choice(transition_matrix[
                                original_label])
142                 else:  # Symmetric noise
143                     possible_labels = [x for x in range(7) if x != original_label]
144                     y_noisy[idx] = np.random.choice(possible_labels)
145
146         actual_noise = np.mean(y_train != y_noisy)
147         print(f'Applied Noise: {actual_noise:.2f}')
148         return y_noisy
149
150
151 def load_exams_csv(csv_path, data_dir):
152     """Load exams.csv and create a mapping of exam_id to trace_file and labels."""
153     print("Loading exams.csv...")
154     trace_file_map = {}
155     labels_map = {}
156     existing_files = set(os.listdir(data_dir))  # Available HDF5 files
157     with open(csv_path, "r") as f:
158         reader = csv.reader(f, delimiter=",")
159         next(reader)  # Skip header
160         for row in reader:
161             exam_id = int(row[0])
162             trace_file = row[14]
163             ecg_conditions = [row[i] == 'True' for i in range(4, 10)]  # 6
                    conditions
164             normal_ecg = not any(ecg_conditions)  # 7th label
165             ecg_conditions.append(normal_ecg)
166             if trace_file in existing_files:
167                 trace_file_map[exam_id] = trace_file
168                 labels_map[exam_id] = ecg_conditions
169             else:
170                 print(f"Skipping exam {exam_id}: Missing file {trace_file}")
171             labels_map[exam_id] = ecg_conditions
172     print(f"Loaded {len(trace_file_map)} exam records.")
173     return trace_file_map, labels_map
174
175 def process_and_save_data(output_file, data_dir, trace_file_map, labels_map):
176     """Process ECG data and save directly to HDF5 to handle large datasets
            efficiently."""
177     print(f"Processing data and saving to {output_file}...")
178     existing_files = set(os.listdir(data_dir))  # Available HDF5 files
179
```

```
    with h5py.File(output_file, "w") as f:
        first_write = True
        for exam_id, trace_file in trace_file_map.items():
            if trace_file not in existing_files:
                print(f"Skipping exam {exam_id}: Missing file {trace_file} ERROR")
                continue

            file_path = os.path.join(data_dir, trace_file)
            with h5py.File(file_path, "r") as h:
                exam_ids = h['exam_id'][()]
                index = np.where(exam_ids == exam_id)[0]
                if len(index) == 0:
                    print(f"Exam ID {exam_id} not found in {trace_file}")
                    continue

                trace = h['tracings'][index[0]]
                label = np.array(labels_map[exam_id], dtype=np.float32)

                # Save directly to HDF5
                if first_write:
                    f.create_dataset("x", data=trace[None, ...], maxshape=(None,
                        4096, 12), chunks=True)
                    f.create_dataset("y", data=label[None, ...], maxshape=(None, 7)
                        , chunks=True)
                    first_write = False
                else:
                    f["x"].resize((f["x"].shape[0] + 1), axis=0)
                    f["x"][-1] = trace
                    f["y"].resize((f["y"].shape[0] + 1), axis=0)
                    f["y"][-1] = label
    print(f"Data successfully saved to {output_file}")

def split_train_test(data_path, train_path, test_path, split_ratio=0.8):
    """Split the processed data into train and test sets."""
    print("Splitting data into train and test sets...")
    with h5py.File(data_path, "r") as f:
        x_data = f['x'][...]
        y_data = f['y'][...]

    total_samples = x_data.shape[0]
    train_size = int(total_samples * split_ratio)
    indices = torch.randperm(total_samples).tolist()
    train_idx, test_idx = indices[:train_size], indices[train_size:]

    with h5py.File(train_path, "w") as f:
        f.create_dataset("x", data=x_data[train_idx])
        f.create_dataset("y", data=y_data[train_idx])
    with h5py.File(test_path, "w") as f:
        f.create_dataset("x", data=x_data[test_idx])
        f.create_dataset("y", data=y_data[test_idx])
    print(f"Train/Test split completed: {train_size} train samples, {total_samples
        - train_size} test samples.")

def main():
    data_dir = os.path.join(os.getcwd(), 'Code', 'RAW_DATA', 'Code15', 'TRAIN')
    csv_path = os.path.join(os.getcwd(), 'Code', 'RAW_DATA', 'Code15', 'exams.csv')
    output_file = os.path.join(os.getcwd(), 'Code', 'RAW_DATA', 'Code15_Processed',
         'full_dataset.hdf5')
    train_file = os.path.join(os.getcwd(), 'Code', 'RAW_DATA', 'Code15_Processed','
        TRAIN_DATA', 'TRAIN_DATA.hdf5')
    test_file = os.path.join(os.getcwd(), 'Code', 'RAW_DATA', 'Code15_Processed','
        TEST_DATA', 'TEST_DATA.hdf5')

    trace_file_map, labels_map = load_exams_csv(csv_path,data_dir)
    process_and_save_data(output_file, data_dir, trace_file_map, labels_map)
    split_train_test(output_file, train_file, test_file)

main()
```

A.3 Useful functions

```
    def get_corrected_class(batch_features, class_prototypes, sim_threshold=0.9):
    """
    for each sample, we compute cosine similarities to each class's prototypes,
    then select the class with the highest similarity if it exceeds the threshold.

    Returns:
      corrected: tensor of shape (B,) with corrected class indices.
      If no class exceeds the threshold, the value is -1.
    """
    B = batch_features.size(0)
    corrected = torch.full((B,), -1, dtype=torch.long, device=batch_features.device
        )
    for i in range(B):
        best_sim = -1.0
        best_cls = -1
        for cls, prototypes in class_prototypes.items():
            # Compute cosine similarity between sample i and all prototypes for
                class cls.
            sims = F.cosine_similarity(batch_features[i].unsqueeze(0), prototypes,
                dim=-1)
            max_sim = sims.max().item()
            if max_sim > best_sim:
                best_sim = max_sim
                best_cls = cls
        if best_sim > sim_threshold:
            corrected[i] = best_cls
    return corrected


    def select_prototypes(features, num_prototypes=32, sim_threshold=0.9):
    """
    selects prototypes from the feature embeddings for one class.

    features: tensor of shape (N, feat_dim) for samples of one class.
    num_prototypes: maximum number of prototypes to select.
    sim_threshold: if the cosine similarity between a candidate and any already
                   selected prototype exceeds this threshold, the candidate is
                       skipped.

    Returns:
      A tensor of shape (P, feat_dim) with P <= num_prototypes prototypes, or None
         if no prototypes.
    """
    prototypes = []
    # Compute cosine similarities
    cos_sim = F.cosine_similarity(features.unsqueeze(1), features.unsqueeze(0), dim
        =-1)
    density = cos_sim.mean(dim=1)  # average similarity for each sample
    sorted_indices = density.argsort(descending=True)

    for idx in sorted_indices:
        candidate = features[idx]
        if not prototypes:
            prototypes.append(candidate)
        else:
            sims = [F.cosine_similarity(candidate.unsqueeze(0), proto.unsqueeze(0))
                .item()
                    for proto in prototypes]
            if max(sims) < sim_threshold:
                prototypes.append(candidate)
        if len(prototypes) >= num_prototypes:
            break

    if prototypes:
        return torch.stack(prototypes)
    else:
        return None
```

```
61
62      def update_prototypes(model, data_loader, num_classes, device, max_batches=40,
            max_features_per_class=1000):
63      model.eval()
64      class_features = {cls: [] for cls in range(num_classes)}
65      batch_count = 0
66      with torch.no_grad():
67          for batch in tqdm(data_loader, desc="Updating Prototypes", leave=False):
68              if batch_count >= max_batches:
69                  break
70              inputs = batch['data'].to(device)
71              labels = batch['y'].to(device).long()
72              _, features = model(inputs, return_features=True)
73              for i in range(features.size(0)):
74                  cls = labels[i].item()
75                  class_features[cls].append(features[i].cpu().detach())
76                  if len(class_features[cls]) > max_features_per_class:
77                      class_features[cls] = class_features[cls][-
                            max_features_per_class:]
78              batch_count += 1
79      for cls in range(num_classes):
80          print(f"Class {cls} collected {len(class_features[cls])} features.")
81      class_prototypes = {}
82      for cls in range(num_classes):
83          if class_features[cls]:
84              feats = torch.stack(class_features[cls])
85              prototypes = select_prototypes(feats, num_prototypes=16, sim_threshold
                    =0.95)
86              if prototypes is not None:
87                  class_prototypes[cls] = prototypes.to(device)
88      import gc
89      gc.collect()
90      return class_prototypes
```

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY