Methodology: Generative Adversarial Networks for PCG Arrhythmia Detection

SR-TEEM2-013

February 28, 2021

Contents

1 Data Management **2**

1.1 Data Collection . . . 2

1.2 Data Preprocessing . . . 4

1.3 Data Segmentation . . . 4

1.4 Data Augmentation . . . 4

2 Model Development **5**

2.1 Model Architecture . . . 5

2.2 Model Complexity . . . 5

3 Model Learning **5**

3.1 Model Training . . . 5

3.2 Model Training . . . 5

3.3 Model Evaluation . . . 5

4 Model Deployment **6**

4.1 Integration . . . 6

4.2 Monitoring & Maintaining . . . 6

5 Model Structure **6**

5.1 PCG-Net: Dense Generator . . . 6

5.2 PCG-Net: Convolutional Discriminator . . . 6

5.3 VQGAN: Architecture . . . 8

5.4 VQGAN: Data Transformation . . . 8

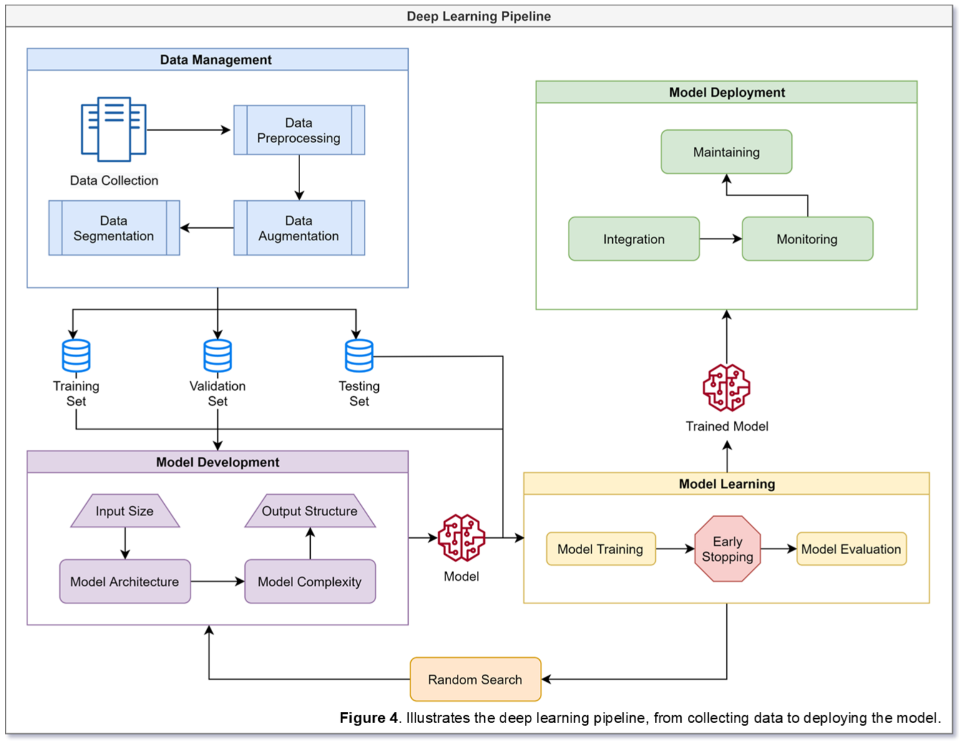

Figure 1: Illustrates the deep learning pipeline, from collecting data to deploying the model.

1 Data Management

1.1 Data Collection

Although PCG signals are analyzed less often than ECG signals, these signals are rather analyzed in real-time by physicians and healthcare workers. Preliminary studies were done on PCG segmentation and classification primarily used private datasets. Hence, there existed no publicly available datasets until recently. Since then, many public datasets have been developed aiding researchers in their studies and creating open benchmarks for researchers to use in comparing similar findings. However, these datasets are still limited by the number of classes that are collected, when compared to ECG datasets.

Currently, only three major supervised PCG datasets exist: PhysioNet Classification of Heart Sound Recording Challenge dataset, PASCAL Heart Sound Challenge dataset, and the Heart Sound and Murmur Library. These datasets are all anonymized and de-identified for the safety of their subjects, and thus includes no personal information such as name, income, age, etc.

The PhysioNet Classification of Heart Sound Recording Challenge dataset was produced as a part of the 2016 PhysioNet Computing in Cardiology Challenge. The heart sounds were collected from both clinical and non-clinical environments (in-home visits). The challenge focused on creating an accurate dataset of normal and abnormal heart sound recordings, especially in real-world (extremely noisy and low signal quality) scenarios. These recordings were sourced from nine independent databases and in total, contain 4,593 heart sound recordings from 1072 subjects, lasting from 5-120 seconds. Of which, 409 recordings that were collected from 121 patients contain one PCG lead and one simultaneously recorded ECG. Though, all recordings were resampled to 2,000 Hz using an anti-alias filter. Furthermore, the dataset is comprised of 3 classes: normal, abnormal, and unsure (this is due to poor recording quality), and have the following proportion respectively: 77.1%, 12,0%, 10.9%.

The PASCAL Classifying Heart Sounds Challenge dataset was released to the general public in 2011. The challenge consisted of two sub-challenges: heart sound segmentation, and heart sounds classification; these sub-challenges corresponded with dataset A, and dataset B respectively. Both datasets have recordings of varying lengths, between 1 second and 30 seconds. Dataset A was collected via the iSethoscope Pro iPhone app, and contained 176 heart sound recordings. 124 of which are divided into four classes: Normal (31 recordings), Murmur (34 recordings), Extra heart

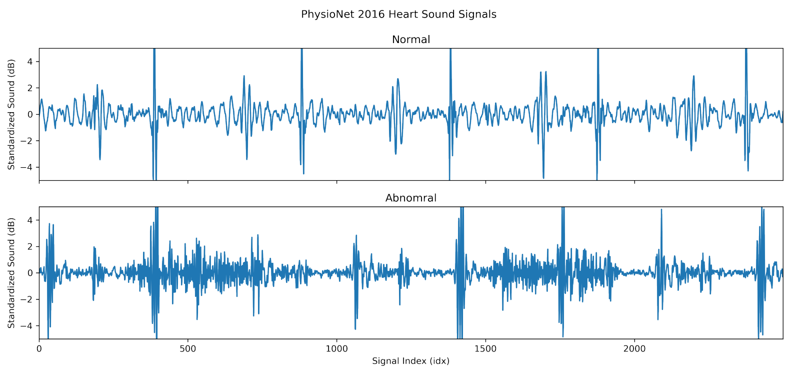

Figure 2: Plots the classes of the PhyioNet 2016 heart sound dataset (Normal and Abnormal).

sound (19 recordings), and Artifact (40 recordings); the rest of the records are unlabeled for testing purposes. Dataset B was collected using a DigiScope (a digital stethoscope), and included 656 heart sounds. All except 370 were separated into three classes: Normal (320 recordings), Murmur (95 recordings), and Extra-systole (46 recordings). Both datasets A and B vary in sound recordings between lengths of 1 second and 30 seconds.

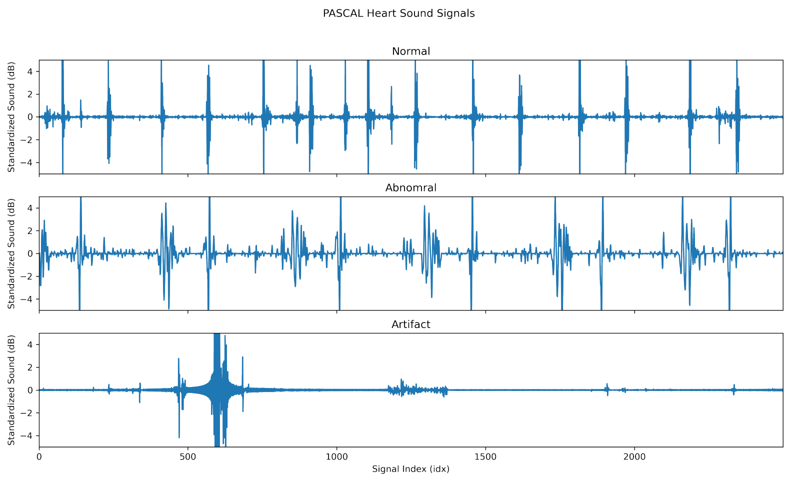

Figure 3: Plots the classes present in the PASCAL dataset (Normal, Abnormal, and Artifact).

More than 300 million ECG recordings are analyzed yearly, and thus create an exceptional tool for arrhythmia classification. Coupled with the recent surge in research interest in 2015, many massive publicly available datasets have been published, notable by PhysioNet - the moniker of the Research Resource for Complex Physiologic Signals. Numerous, datasets ECG exist, however, many are limited to few classes (Normal and Abnormal). At present, three public datasets exist that have more than 4 classes: AF Classification Challenge 2017, PTB Diagnostic ECG, and PTB-XL dataset. Additionally, iRhythm Technologies have developed a semi-public dataset, that is available upon request, that contains 12 classes.

The PTB-XL is the largest publicly available dataset for ECGs and contains 21,837 clinical 12-lead ECG recordings from 18,885 patients of 10 second length. These recordings are separated into 5 super-classes: Normal, Myocardial Infraction, Hypertrophy, ST/T-Change, and Conduction Disturbance. These super-classes are further split into 71 sub-classes that range from AV Block to Posterior Myocardial Infraction. The raw signal data were downsampled to 100 Hz and annotated by up to two cardiologists, who assigned potentially multiple ECG statements to each record. iRhythm Technologies developed a large, 12 classes ECG dataset using raw single-lead ECG inputs. The 12 classes include Atrial fibrillation and flutter, AVB, Bigeminy, EAR, IVR, Junctional rhythm, Noise, Sinus rhythm, SVT, Trigeminy, Ventricular tachycardia, and Wenckebach. The dataset consists of 91,232 ECG recordings from 53,549 patients. This training dataset is available upon request under license from iRhythm Technologies, Inc. The publicly available test dataset contains 328 records collected from 328 unique patients, split between 6 classes. Both datasets were recorded using a Zio monitor, which monitors the heart through a single-lead sensor at 200 Hz. The annotation was done by a consensus committee of expert cardiologists.

The PhysioNet AF Classification database, presented in 2017 for the Computing in Cardiology Challenge, contains 8,528 ECG recordings, divided into 4 classes: Normal (5154 recordings), Atrial Fibrillation (771 recordings), Other arrhythmias (2557 recordings), and Noisy (46 recordings). The single-lead recordings last from 9 - 61 seconds, with a mean of 32.5 seconds and a standard deviation of 10.9 seconds. The ECG recordings were sampled to 300 Hz and provided in MATLAB V4 WFDB-compliant format.

1.2 Data Preprocessing

PCG recordings often are recording in non-ideal environments that are filled with unwanted background noise and interference. Data preprocessing is the process of altering the data in the signal, often by denoising, normalizing, standardizing, and transforming the signal. These steps are crucial for automatic localization and classification tasks. Preprocessing the data allows a model to extract meaning features efficiently and reveals the physiological structure of the heart sounds [Latif et al.]. Furthermore, preprocessing helps ensure that the data that is fed into the model is always in the same domain. This allows the model to generalize more easily.

We first resample the data to 500 Hz, to decrease the spatial resolution of the heart sound recordings, but still retain important features. Thus, helping the model to converge faster. The resampled data is standardized using the standard score equation. This scales the mean of the distribution to 0, artificially scaling all data into similar ranges, thus, helping combat the exploding gradient problem.

The standardized data is then fed into a CycleGAN that has learned to denoise data. The CycleGAN is fed synthetically noised PCG signal and attempts to construct the denoised data from the noisy data. This synthetic noise consists of white noise, pink noise, and real background noise collected from audio recordings. The noise is added to each PCG signal recording and then treated as the input to the CycleGAN. The CycleGAN's output is compared to the original, non-noise, PCG recording. In this way, the CycleGAN eventually learns to denoise PCG recordings.

1.3 Data Segmentation

Data segmentation refers to the process of creating cross-validation datasets. This process assists in validating if the model is overfitting to the dataset. These datasets include the training set, validation set, and testing set. Typically, the training set is 70%-80% of the dataset, the reset of the dataset is split among the validation set and testing set. Here, we split the data 80% training, 10% validation, and 10% testing.

1.4 Data Augmentation

Data augmentation is a strategy that enables a significant increase in the diversity of data available while training a model, without actually collecting new data. Data augmentation techniques aim to slightly alter existing data to a point where the model cannot recognize the augmented data as one it has trained on before, but still retains the characteristics of the data's category. This helps in reinforcing important features within the data and is only done during the training portion of the workflow.

A common misconception arises when comparing preprocessing and augmentation. To be clear, preprocessing aims to clean the data of unwanted artifacts that are not meant for classification and is done in place. Augmentation, on the other hand, is solely done for expanding the dataset's size, often to combat overfitting. Augmenting the data before preprocessing further obscures the data unrealistically, and beyond classification.

We use to resample the heart sound recordings to different frequencies to simulate slower and faster beats per minute (bpm). The normal bpm for a human is between 60-100 bpm. Thus, measuring the sample distance between the first S1 (the start of systole) and the second S1, we calculate the bps and resample accordingly.

Furthermore, we use noise injection directly to preprocessed PCG recordings [Messner et al.]. This process is identical to the process of synthetically adding noise to PCG recordings described in the preprocessing step. A variety of noises, like white noise, is added to the signal to increase the sample of recordings per class. This method is extremely beneficial for training on small datasets, like the PASCAL dataset.

2 Model Development

2.1 Model Architecture

Here we propose using Generative Adversarial Networks (GANs) for increased success in PCG heart sound detection. GANs pose a unique advantage over traditional machine learning and deep learning methods, in that a model learns to mimic a dataset by creating its own data, and tries to fool a discriminator into thinking the generated data is real. In a supervised approach, a GAN consists of two parts, a generator and a discriminator. The generator is responsible for creating fake heart sound data, while the discriminator tries to predict where the incoming data is fake or real. In a semi-supervised approach, however, the is fed data from a real dataset and the generator. Here, the discriminator tries to classify the generator's fake data, as well as predict the classes form the real dataset.

2.2 Model Complexity

Traditionally, generators are dense layers that slowly increase the dimensionality of the generated data to match that of the real dataset. Discriminators, on the other hand, are commonly CNNs because the majority of their applications work with images. However, it is possible to use a wide variety of architectures; such as LSTMs, RNNs, SVMs, DNNs, ANNs, Transformers. As mentioned previously, there are many types of model architecture, some are used for classification, and others for feature extraction. Optimizing the combination of feature extraction layers and classification layers is extremely time-consuming and computationally taxing. This is because there exist many combinations of hyperparameters, thus making it difficult to optimize each parameter. To optimize hyperparameters, we used hyperparameter sweeps to make the optimization process more efficient. This method involves using one of three methods: grid search, random search, and Bayesian search. Grid search computes each possible combination of all hyperparameters and tests them all. Although this is very effective, it can be computationally costly. Random search selects a new combination at random, provided a distribution of values. This method is surprisingly effective and scales very well. Bayesian search creates a probabilistic model of metrics and suggests parameters that have a high probability of improving metrics. This works well for small-scale projects, but scales poorly as the complexity of parameter relationships increases. Here, we used a random search to optimize our hyperparameters.

3 Model Learning

3.1 Model Training

During the training phase, the model is trained using backpropagation in conjunction with a cost function. Backpropagation attempts to calculate the gradient of the cost function with respect to the weight and biases of the model. This process involves an optimizer, which optimizes the model's parameters and a cost function that measure the correctness or incorrectness of the model. The goal of the optimizer is to minimize the cost function's error by adjusting the parameters to the given label. In this study, we used the Adam optimizer in union with Cross-Entropy Loss. The Adam optimizer uses a hyperparameter that dictated the change in the model's parameters on each backpropagation step, this is called the learning rate. Here we choose a learning rate of 0.0001.

The model is only trained on the training set; thus, backpropagation only occurs on the training set. Additionally, for each step in the training set, the optimizer backpropagates and optimizes the parameters and calculates metrics to further evaluate the model. The amount of steps in the training set is dictated by the batch size, the number of signals the model is trained on, in a single forward pass. Here we use a batch size of 32, meaning that the model is fed 32 signals per input. This significantly speeds up the process of training as more signals are passed through the model every time the model is optimized. A full pass of the training set is called an Epoch, here we train the model on 100 Epochs.

3.2 Model Training

To ensure the model is not overfitting, but generalizing to the training set, we use a validation set to track the metrics of the model. In theory, the metrics on the training set equal to that of the validation set. In practicality, after many epochs of training the metrics of the validation set become static, but the metrics of the training set still increase. This suggests that the model is overfitting. Thus, we stop training the model on the training set and test it as a testing set.

3.3 Model Evaluation

Testing sets or hold-out sets are used to validate the metrics of the model, this is because both the validation set and the testing set have been tested by the model; thus, the model has developed a latent bias to both sets. Therefore, a third set is needed to assess the model's ability to generalize on an independent dataset. The metrics calculated on the testing set include the Accuracy, Sensitive, and Specificity (and MSELoss in the case of the VQGAN).

4 Model Deployment

Model Deployment is one of the last stages of any machine learning project and involves releasing the model to the public.

4.1 Integration

Integration consists of implementing the model in a system, whether it happens on the client-side or the backend. The most popular backend model integration tools involve Flask, Azure, and FastAPI. These tools create APIs that encapsulate the model prediction, given a GET request with the desired input.

4.2 Monitoring & Maintaining

Following model integration and deployment, we move onto the next phase, monitoring and maintaining the system. As more and more data passes through the model, it increases the opportunity for the model to learn from a more generalized dataset. Though such data would be unsupervised, we could use unsupervised techniques to categories the data. Based on the improvement of the model, the model and be reintegrated and deployed. In essence, looping the whole process from data management to model learning.

5 Model Structure

The GAN model contains two sub-models, a Generator and a Discriminator. The Generator is responsible for extracting relevant features from ECG signals and constructing a PCG signal from extracted latent features (in the case of VQGAN). The Discriminator is responsible for extracting relevant features from the PCG signal and creating predictions of whether or not the signal was generated by the Generator.

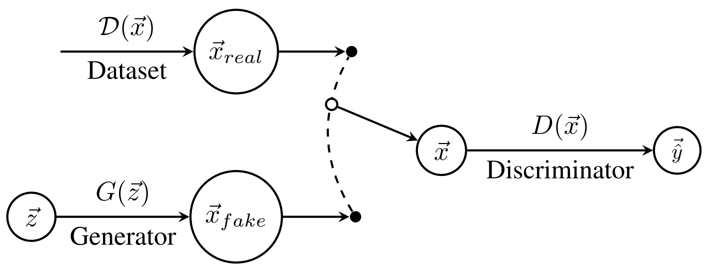

Figure 4: GAN Structure.

5.1 PCG-Net: Dense Generator

The input consists of a 2-dimensional tensor (batch size x signal) with a length of 128. This input is randomly generated. This input is then fed into the Dense layers upscale the length of the input vector until the length reaches that of a real input signal (2500). This allows for the output of the generator to be directly fed into the discriminator. Between each dense layer, a Rectified Linear activation function alters the range of the incoming data by setting all numbers below 0 to 0 and leaving all positive numbers intact.

5.2 PCG-Net: Convolutional Discriminator

The PCG arrhythmia detection algorithm is a sequence-to-sequence Generative Adversarial Network, with the architecture that contains a Generator (G) and a Discriminator (D). The function of a Generator is to generate data such that, the Discriminator will classify the generated data as real, while the Discriminator aims to classify the generated data as fake. Hence, the Generator requires a series of noise signals $z_i \sim N\left(\mu, \sigma^2\right)$; such that $\mu = 0$ and $\sigma = 1$. Thus, $\overrightarrow{z} \in \{-1, 1\}$ where every element of z is between -1 and 1, with a fixed length z_{100}. The Generator then outputs $\overrightarrow{x}_{fake} = [x_1, ...x_n]$ biased on $\overrightarrow{z}$. Conversely, the Discriminator takes inputs $\overrightarrow{x} = [x_1, ...x_n]$ as features

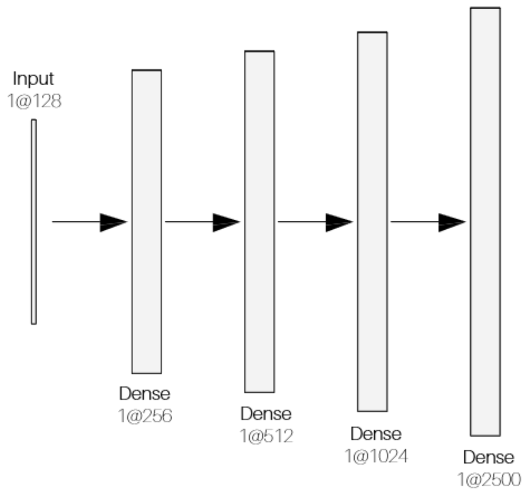

Figure 5: Representation of PCG-Net dense generator structure.

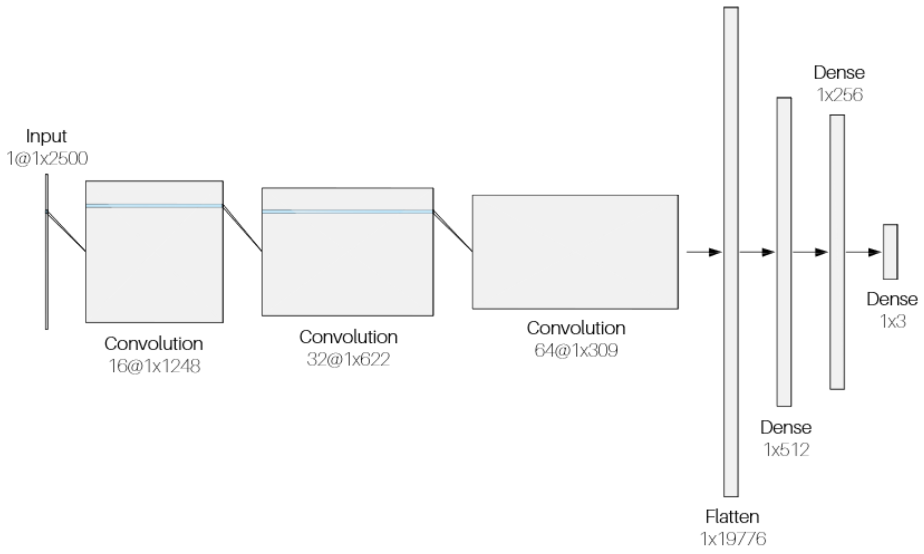

Figure 9: Representation of PCG-Net convolutional discriminator structure.

from a dataset, $\mathcal{D}$, and calculates the outputs $\overrightarrow{y} = [\hat{y_1}, ...\hat{y_c}]$ based on latent features extracted from $\overrightarrow{x}$, where $\hat{y_i} - 1$ represents each class in the dataset. Given $|\overrightarrow{x}_{fake}| \sim |\overrightarrow{x}|$, meaning the Generator output, $\overrightarrow{x}_{fake}$ is similar in structure and cardinality with the input of the Discriminator, $\overrightarrow{x}$.

The input consists of a 3-dimensional tensor (batch size x channel size x signal) with a length of 2500. Then then the convolution decreases the size of the input vector. A vector (with a size of 1x5) filters across the data by multiplying all values in the input vector by the filter vector. This method also assists the model in finding features. The Rectified Linear activation function, between each convolutional and dense layer, alters the range of the incoming data by setting all numbers below 0 to 0 and leaving all positive numbers intact. The liner function flattens the incoming result (batch size x 64 x 309) into a 2D tensor (batch size x19776). The Dense layers downscale the length of the input vector until the length reaches that of the number of classes. This allows for the output of the discriminator to be directly interpreted by the cost function. The Linear Output layer transforms the Linear layer output into a 1x3. Each column in the tensor represents a class's likelihood of being the correct class in the dataset. Thus, the column with the largest values is the model prediction for the input.

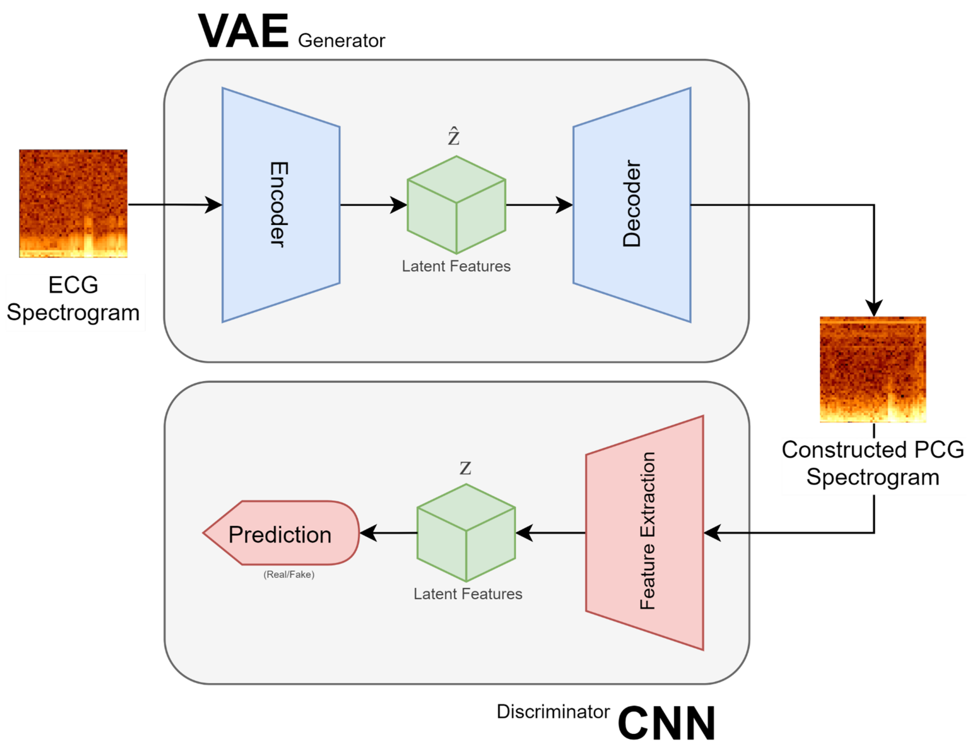

Figure 20: Diagram of VQGAN, composed of a VAE Generator and a CNN Discriminator.

5.3 VQGAN: Architecture

Variational Autoencoders (VAE) is an architecture that attempts to map the initial data into an encoded space ($\hat{z}$) that stores latent features, the encoder is responsible for doing this task. The encoded space is regularized to avoid overfitting. In the context of ECG and PCG signals, both signals come from different latent spaces, which makes it difficult to travel from one domain to another. Regularizing this encoded latent space ensures that the latent space has good properties. This space is then reconstructed by a decoder into an encoded-decoded space, identical to the initial data in shape. With respect to ECG and PCG signals, the input, a spectrogram representation of the ECG signals, is encoded into the encoded space. The encoder consists of ResNet and Multi-head attention blocks that introduce a method to negate the vanishing gradient problem and weights for each pixel in the spectrogram respectively. The encoded space represents important structural elements, such as the positions of the PQRST complex. This space is then decoded using a revered structured encoder, into an encoded-decoded state that respectable a spectrogram of the PCG signal. These reconstructed PCG signals are sent to a convolutional discriminator, which attempts to classify the generated PCG spectrogram as real or fake (reconstructed). Of course, the goal of the discriminator is to classify the constructed spectrograms as fake and spectrograms from a PCG dataset as real. Whereas, the generator attempts to fool the discriminator into classifying the generated spectrogram as real. This system ensures that the generated spectrogram looks authentic and genuine.

5.4 VQGAN: Data Transformation

Traditional machine learning reconstruction techniques use images because they contain dense information in a low dimensionality. However, time-series data, such as heart sound signals, are one-dimensional. Thus, we use time and frequency analysis, specifically log spectrograms to represent heart sound recordings in a two-dimensional manner. These spectrograms show the frequency of the signal against time. This process is done on both the PCG and ECG signals and results in a 48 x 48 image.

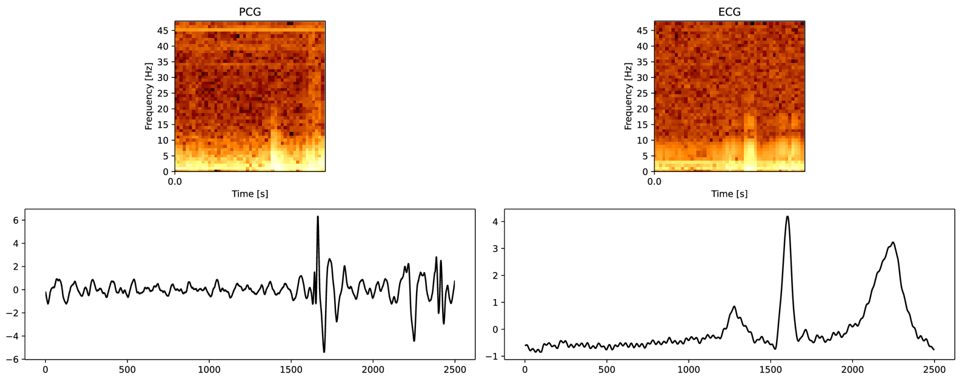

Figure 19: Comparison of PCG and ECG log spectrograms.