Transforming 1-D Machine Learning Problems to 2-D Towards Using Convolutional Neural Networks

Georgios Lekkas

gxlekkas@gmail.com

Introduction

Convolutional Neural Networks (CNNs) have revolutionized the field of image processing and computer vision, particularly in the processing and classification of two-dimensional (2D) data. This importance is even more pronounced when it comes to analyzing data that has been transformed from one-dimensional (1D) to 2D formats. The capability of CNNs to effectively manage spatial hierarchy in data makes them exceptionally suitable for tasks involving images, time series, and any form of spatially structured data.

The use of 2D Convolutional Neural Networks (CNNs) for analyzing signals such as Electrocardiograms (ECGs) that have been transformed into a two-dimensional format represents a significant advancement in medical signal processing and diagnostic methodologies. This transformation and subsequent analysis allow for the leveraging of CNNs' spatial pattern recognition capabilities, offering a more nuanced and comprehensive understanding of the underlying physiological conditions.

ECG signals, traditionally analyzed as one-dimensional time series, capture the electrical activity of the heart and are fundamental in diagnosing a wide range of cardiac abnormalities. However, the complexity and subtlety of certain cardiac events can make them challenging to detect and classify using conventional 1D signal analysis techniques. By transforming these 1D ECG signals into 2D representations, it becomes possible to visualize and analyze the frequency and time-based characteristics of the heart's electrical activity in a spatial context.

Dataset and preprocessing

In this project we made use of the MIT-BIH Arrhythmia Database (https://physionet.org/content/mitdb/1.0.0/). . A widely used dataset in the field of biomedical signal processing, particularly for the development and evaluation of algorithms for cardiac signal analysis, including ECG (electrocardiogram) classification. This database contains 48 half-hour excerpts of two-channel ambulatory ECG recordings, obtained from 47 subjects studied by the BIH Arrhythmia Laboratory between 1975 and 1979. The recordings were digitized at a sampling rate of 360 samples per second.

Baring in mind the complexity of the raw ECG signals we have utilized only one record with normal heart beats and only one record with paced heart beats – the cardiac rhythm is controlled by an electrical impulse from an artificial cardiac pacemaker, in order to compare the dimension transformation methodologies and implement our 2D CNN model.

The preprocessing methodology that we implemented begins as follows:

First, we converted the raw 1D signals which were found in a .dat format inside the data-set, into data-frames for easier and straightforward use. Then we applied our first preprocessing on the csv files (Signal Preprocessing.py):

- **Noise filtering**: In order to reduce the noise within the ECG signal. A median non-linear filter was applied with the aim to replace each sample in the signal with the median of neighboring samples, using a kernel size of 3, meaning that for each point in the signal, the median value of it and its immediate neighbors is calculated.
- **Baseline wander removal**: Baseline wander is a low-frequency noise introduced into ECG recordings by patient movement, respiration, or poor electrode contact. It can obscure the true morphology of the ECG waveform, making accurate analysis challenging. The removal is being done by fitting a linear model to the signal and then subtracting this model from the signal.
- **Normalization**: Normalization is performed to standardize the amplitude range of ECG signals. The function standardizes the signal to have a mean of 0 and a standard deviation of 1. This is done by subtracting the mean of the signal from each sample and then dividing by the standard deviation of the signal.

These preprocessing functions collectively clean the ECG signals by reducing noise, correcting baseline wander, and standardizing the signals' amplitude. The on the processed signals we apply the Pan-Tompkins algorithm (Pan–Tompkins algorithm.py).

The Pan-Tompkins algorithm is a widely recognized method for detecting QRS complexes in electrocardiogram (ECG) signals. The QRS complex is a crucial component of the ECG waveform, representing the depolarization of the right and left ventricles of the human heart. Accurate detection of QRS complexes is essential for arrhythmia detection. The algorithm applies the following functions on the processed signals:

- **Band-pass filter**: This function applies a band-pass filter to the ECG signal to isolate the frequency range where the QRS complex primarily exists, reducing noise outside this band. Parameters include low and high cutoff frequencies, the signal sampling frequency, and the filter order.
- **Derivative**: This function approximates the first derivative of the ECG signal, highlighting the rate of change in the signal, which are characteristic of the QRS complex. This step enhances the QRS complexes' slopes, making them more distinguishable from other components of the ECG signal, like T and P waves.
- **Squaring**: This function squares each sample of the input signal. This step serves to emphasize the larger values (typically associated with the QRS complexes) over the smaller ones, effectively increasing the signal-to-noise ratio.
- **Moving Window Integration**: This function smooths the squared signal by averaging it over a moving window. The size of the window is determined by the ‘window_size’ parameter. This process helps in consolidating the QRS complex information spread across several samples, making the complexes more pronounced and easier to detect.
- **Find peaks**: This is a custom function designed to identify peaks in the smoothed signal that exceed a certain threshold. It implements a basic peak detection algorithm by comparing each point to its neighbors.

After processing the signal and applying the Pan-Tompkins Algorithm to detect the QRS complexes, we move to the last implementation, the signal segmentation. Signal segmentation refers to the process of dividing a continuous signal into smaller segments or portions.

For each QRS complex, the script calculates the nearest point in the ECG data based on time stamps, ensuring precision by rounding time values to 8 decimal places. It then determines a segment of the ECG signal to extract, centered around the QRS complex, with a predefined length in samples. This segment length is chosen to sufficiently capture the QRS complex and its context within the ECG waveform. The start and end points of each segment are carefully calculated to ensure they do not exceed the bounds of the ECG data.

The processed segmented signal is illustrated in Figure 1 below:

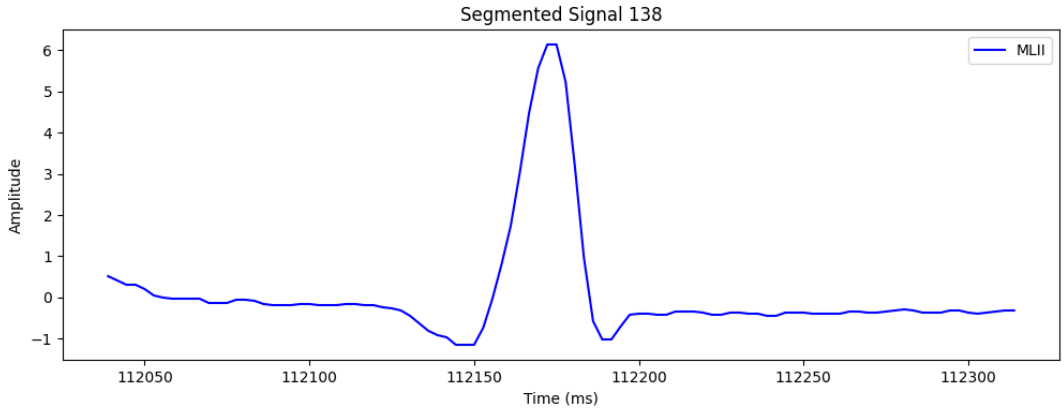


Figure 1: Segmented signal

After implementing our preprocessing sequence and conducting signal segmentation, we are ready to implement all our dimension transformation methodologies.

Dimension Transformation

Transforming one-dimensional (1D) signals into two-dimensional (2D) formats is a crucial step in enabling the application of 2D Convolutional Neural Networks (CNNs) for advanced analysis. This transformation can be achieved through various methods, each offering unique insights into the signal's characteristics. In this work we have used and compared the following six dimension transformation methods: The Reshaping Method, Recurrence Plots, Discrete Fourier Transform (DFT), Fast Fourier Transform (FFT), Short-Time Fourier Transform (STFT), and Continuous Wavelet Transform (CWT), with the aim to examine which method has greater performance when used with the same 2D CNN model.

To preserve homogeneity across all the processed 1D segments after transforming them to 2D images with the aforementioned methods, we kept the dimensions at 224 x 224 pixels and gray-scale.

Reshaping Method

The reshaping method is perhaps the simplest form of data transformation and involves reorganizing the data points of a 1D signal into a 2D matrix. For a 1D array of length N, reshaping can convert it into a 2D array of dimensions $m \times n$, where $m \times n = N$. This operation is essential when the input data for an algorithm must be in a 2D format, such as images in CNNs. The reshaping method is purely structural, meaning it does not change the data's underlying meaning or representation but makes it compatible for algorithms that require 2D input (Reshaping method.py).

The script transforms the 1D ECG signals into 2-D grayscale images.Initially, it scales the ECG data to a 0-255 gray-scale range, ensuring that the signal's dynamic range is preserved in the image format. The script extracts the signal data applies a gray-scale conversion to normalize and scale the intensity values. Subsequently, it adjusts the ECG signal's size to fit a predefined image dimension, carefully stretching the signal's height and width to match the target shape while maintaining the aspect ratio. The resizing process is achieved through the zoom function, which interpolates the signal data for a smooth transition to the new size. A 2D image canvas is prepared, with the resized signal centrally placed, ensuring that the ECG occupies the intended portion of the image space. The transformed 2D Image is illustrated in Figure 2:

Figure 2: 2D Image with the Reshaping Method

Recurrence Plots

Recurrence plots are a more complex transformation that visualizes the dynamics of a system. The idea is to plot a point at coordinates (i,j) in a 2D space whenever the state of a system at time i is close to its state at time j, according to some predefined criterion. This method is based on the concept of phase space reconstruction from time series data and is useful for identifying patterns, changes, or recurring states within the system. For signals, this translates into capturing repetitions and similarities over time (Recurrence Plots method.py).

This script transforms the 1D ECG signal data into recurrence plot (RP) images. It operates by first setting a proximity threshold and steps to define when points in the ECG signal are considered recurrent, based on their closeness in value. For each signal, it constructs a recurrence matrix by comparing each point in the ECG signal against others within a defined range, marking points as recurrent if they meet the proximity criteria. This binary matrix is then resized to a 224 x 224 dimension, converting it into a grayscale image where recurrences are highlighted. The transformed 2D Image is illustrated in Figure 3:

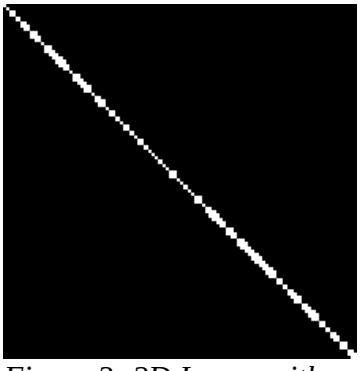


Figure 3: 2D Image with the Recurrence Plot Method

Discrete Fourier Transform (DFT) and Fast Fourier Transform (FFT)

DFT converts a sequence of values into components of different frequencies, providing a frequency domain representation of the original 1D signal (Discrete Fourier Transform (DFT).py).

This script converts ECG signals into 2D gray-scale images based on their Discrete Fourier Transform (DFT) representations, facilitating analysis with image processing techniques. It follows a straightforward process: The DFT is applied to the ECG signals to highlight frequency components and then the DFT spectrum is converted into a 2D image. This image is normalized and scaled to a 224 x 244 size, and finally, converted to gray-scale to emphasize the frequency magnitude spectrum. The transformed 2D Image is illustrated in Figure 4:

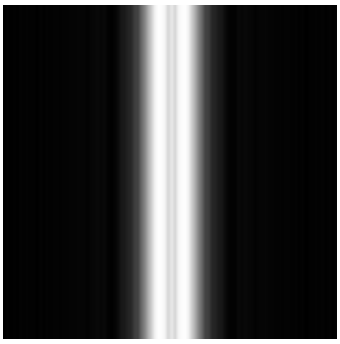


Figure 4: 2D Image with the DFT Method

FFT is an algorithm to compute the DFT efficiently, significantly reducing the computational complexity, from O(N2) to O(NlogN) (Fast Fourier Transform (FFT) method.py).

This script is designed to process ECG signals transforming them into 2D grayscale images based on their Fast Fourier Transform (FFT) representations. It works by first applying FFT to the ECG data to obtain the frequency spectrum, then shifts the zero-frequency component to the center of the spectrum. The magnitude of the FFT, which represents the frequency components' strengths, is scaled to match a target height. This scaled spectrum is then extended horizontally to fit a target 2D shape, ensuring the final image represents the full frequency content of the original ECG signal in a 224 x 224 dimension. The image data is normalized and converted to gray-scale, enhancing the visibility of the frequency components.

The transformed 2D Image is illustrated in Figure 5:

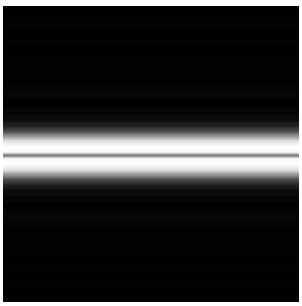


Figure 5: 2D Image with the FFT Method

By applying DFT or FFT, one can analyze the frequency content of signals, which is especially useful in filtering, signal reconstruction, and spectral analysis. The result can be visualized in a 2D plot with frequency on one axis and the magnitude or phase of the frequency components on the other.

Short-Time Fourier Transform (STFT)

STFT addresses the limitation of DFT/FFT in analyzing signals whose frequency content changes over time. It divides the signal into short, overlapping segments and applies FFT to each segment. This process provides a time-frequency representation of the signal, visualized as a 2D spectrogram with time on the horizontal axis, frequency on the vertical axis, and intensity/color representing the amplitude of a frequency at a given time. STFT is particularly useful for analyzing non-stationary signals where frequency components vary over time (Short-Time Fourier Transform (STFT).py).

This script transforms ECG signals into 2D images using the Short-Time Fourier Transform (STFT) to capture the frequency and time characteristics of the signals. The STFT is applied with a Hann window to minimize spectral leakage, and the magnitude of the STFT result is used to represent the signal's frequency content over time. The script focuses on a meaningful frequency range, up to 50 Hz, to ensure relevance to cardiac signals. After computing the STFT, the data is scaled to match a target image size (224 x 224) and converted to grayscale using logarithmic scaling to effectively manage the dynamic range. To enhance visual contrast, adaptive histogram equalization is applied, followed by Gaussian smoothing to reduce noise and smooth the image. The transformed 2D Image is illustrated in Figure 6:

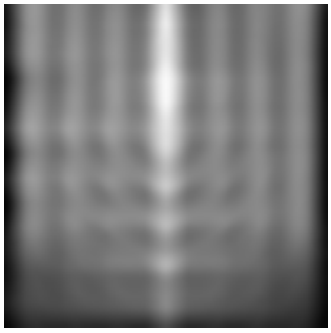


Figure 6: 2D Image with the STFT Method

Continuous Wavelet Transform (CWT)

CWT provides a multi-resolution analysis of a signal by decomposing it into wavelets. Unlike STFT, which uses a fixed window size, CWT analyzes a signal at different scales and translations using wavelets—functions defined over a limited duration with an average value of zero. This results in a 2D representation where one axis represents time, the other represents scale or frequency, and the color/intensity represents the magnitude of the wavelet coefficient. CWT is especially useful for analyzing non-stationary signals where frequency components vary over time (Continuous Wavelet Transform (CWT).py).

This script processes ECG signals to generate 2D images using the Continuous Wavelet Transform (CWT). Applies CWT to capture both time and frequency information, and scales the resulting coefficients to the desired image size (224 x 224). The script employs a specific wavelet ('morl') for transformation, capturing the ECG's nuanced patterns through scales that range widely, allowing detailed analysis of various frequency components. After transformation, the wavelet coefficients are normalized and converted to grayscale images, then undergo contrast enhancement and Gaussian smoothing to improve visual clarity and highlight significant features.

The transformed 2D Image is illustrated in Figure 7:

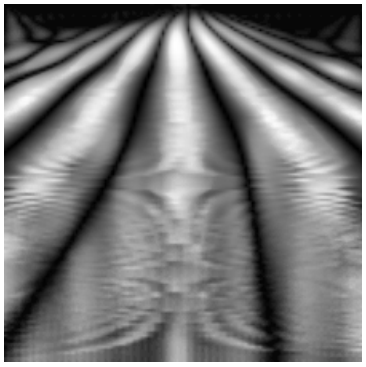


Figure 7: 2D Image with the CWT Method

2D CNN Model

After transforming the 1D signal with the aforementioned methods, we are ready to implement a 2D CNN model in order to compare the performance of all the dimension transformation methods.

We tested our dimension transformation methods with various 2D CNN models. Both our train and test data-sets did not have adequate number of transformed signals to properly train and test the majority of the 2D CNN models. In this work we have implemented the following models: A ResNet50, LeNet-5, MobileNet and a minimal 2D CNN model.

The dataset, consisting of 'normal' and 'paced' 2D images were split to two parts. 85% (70% train + 15% validation) for the training of the model and 15% for testing and evaluating the performance of the model. The three bigger models (ResNet50, LeNet-5 and MobileNet) produced all the performance metrics at 100% and a correct confusion matrix, correctly examining the signals as True Positives and True Negatives. This happened for all the dimension transformation methods. This may indicate two things. First, that the dataset was too easy for all these models in correctly predicting the 2D images as normal or paced and secondly, that due to the fact that the dataset was easy and small, all the models suffered from overfitting.

To overcome this issue, in order to correctly examine and compare the various transformation methods, we setup a minimal 2D CNN in order to challenge it's performance and better illustrate the overall score of its transformation method. At the training phase of the minimal 2D CNN model, we implemented the k-fold cross validation method due to the fact that the dataset was small, in order to avoid overfitting (Minimal 2D CNN k-fold cross val learning rate decay early stopping.py).

The model architecture is explained below:

- **Input layer (shape=(224, 224, 1))**: This layer specifies the shape of the input data, which is set to 224x224 pixels with 1 channel (grayscale).
- **Convolutional layer (8, (3, 3), activation='relu')**: This is the first layer that performs convolutional operations on the input data. It has 8 filters with a size of 3x3 pixels. The convolutional layer scans the input image with each filter, producing 8 different 2D feature maps. The ReLU (Rectified Linear Unit) activation function is used to introduce non-linearity, allowing the model to learn complex patterns.

- **MaxPooling layer (2,2)**: Following the convolutional layer, the max pooling layer reduces the spatial dimensions of the feature maps by half, effectively downsampling the input by taking the maximum value over a 2x2 pooling window. This operation reduces the computational complexity and helps in extracting the most prominent features while reducing overfitting.
- **Flattening layer**: This layer transforms the 2D feature maps into a 1D feature vector, preparing the data for the fully connected (dense) layers that follow. This step is necessary because dense layers expect 1D inputs.
- **Dense layer (64, activation='relu')**: A fully connected layer that receives the flattened feature vector from the previous layer. It has 64 neurons and uses the ReLU activation function. This layer allows the network to learn non-linear combinations of the high-level features extracted by the convolutional and pooling layers.
- **Dropout layer (0.5)**: This layer randomly sets half of the input units to 0 during training at each update, helping to prevent overfitting by making the learning process noisy and forcing the model to learn more robust features that generalize better to unseen data.
- **Output layer (dense(2, activation='softmax')**: The final layer of the model has 2 neurons, corresponding to the number of classes (e.g., 'normal' and 'paced'). It uses the softmax activation function, which outputs the probabilities of each class, ensuring that the sum of these probabilities is 1. The class with the highest probability is considered the model's prediction.

Other model features and techniques:

- **Optimizer:** Adam optimizer is used with a learning rate of 0.001. Adam is an adaptive learning rate optimization algorithm designed specifically for training deep neural networks.
- **Loss Function:** Categorical crossentropy is chosen as the loss function because it's suitable for multi-class classification problems where each class is mutually exclusive.
- **K-Fold Cross Validation**: Used to evaluate the performance of a machine learning model and ensure that it generalizes well to unseen data. The main idea is to divide the entire dataset into 'k' equal-sized folds or subsets. The model is then trained on 'k-1' folds and validated

on the remaining fold. This process is repeated 'k' times, each time with a different fold used as the validation set. The performance measure reported by k-fold cross-validation is then the average of the values computed in the loop.

- **Early stopping**: A regularization method used to prevent overfitting, which occurs when a model learns the training data too well, including its noise, resulting in poor performance on unseen data. It monitors the model's performance on a validation dataset and stops training when the performance on the validation dataset stops improving, by monitoring the validation loss metric.
- **Learning rate scheduler**: A hyperparameter that controls how much we are adjusting the weights of our network concerning the loss gradient. The learning rate scheduler adjusts the learning rate during training according to a predefined schedule or function. This helps in fine-tuning the model training, potentially leading to better performance and faster convergence.
- **Parameters**: Batch Size = 8, Epoch = 5, Learning Rate = 0.001 & Number of K – Folds = 5

Dimension Transformation Comparison

The last part and the main idea of our project is to compare the performance of the 2D CNN model based on the transformed 2D Images that were extracted from its transformation method. Thus, comparing the performance of the dimension transformation methods. To preserve homogeneity in the data, all the signals were processed, transformed to 2D images and fed in the 2D CNN model in the same way. We opted for a dimension of 224 x 224 and gray-scale to simplify the input 2D images. Below all the performance metrics of each dimension transformation together with their confusion matrix are presented. The confusion matrix is shaped as:

- Top left corner = True Positives
- Top right corner = False Positives
- Bottom left corner = False Negatives
- Bottom right corner = True Negatives

Reshaping Method

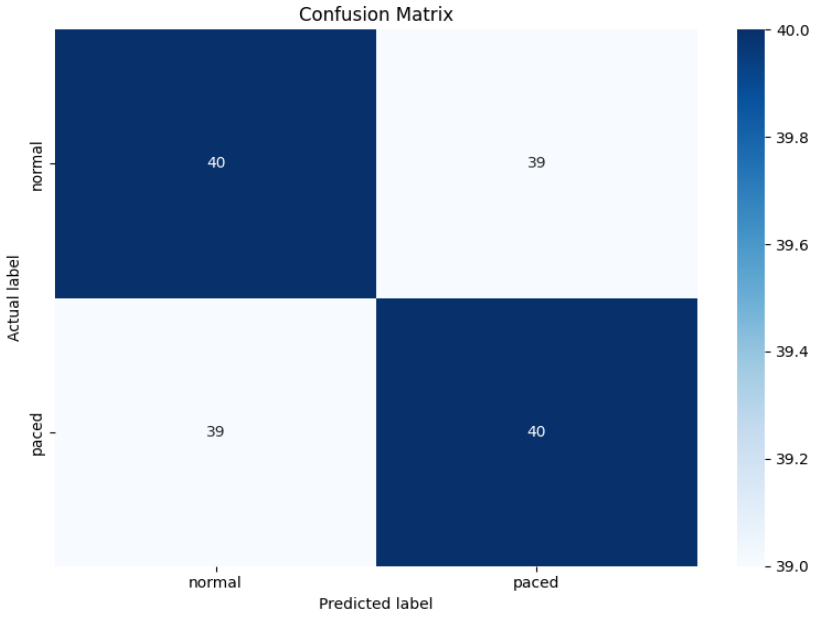
Performance metrics:

Accuracy: 0.506 (50.6 %)

Precision: 0.506 (50.6 %)

Recall: 0.506 (50.6 %)

F1 Score: 0.506 (50.6 %)



Recurrence Plots

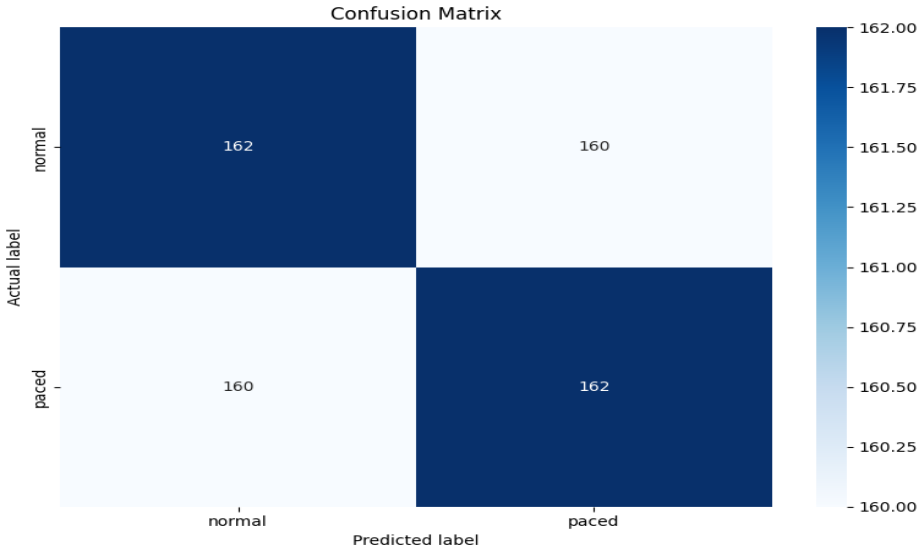
Performance metrics:

Accuracy: 0.503 (50.3 %)

Precision: 0.503 (50.3 %)

Recall: 0.503 (50.3 %)

F1 Score: 0.503 (50.3 %)



Discrete Fourier Transform (DFT)

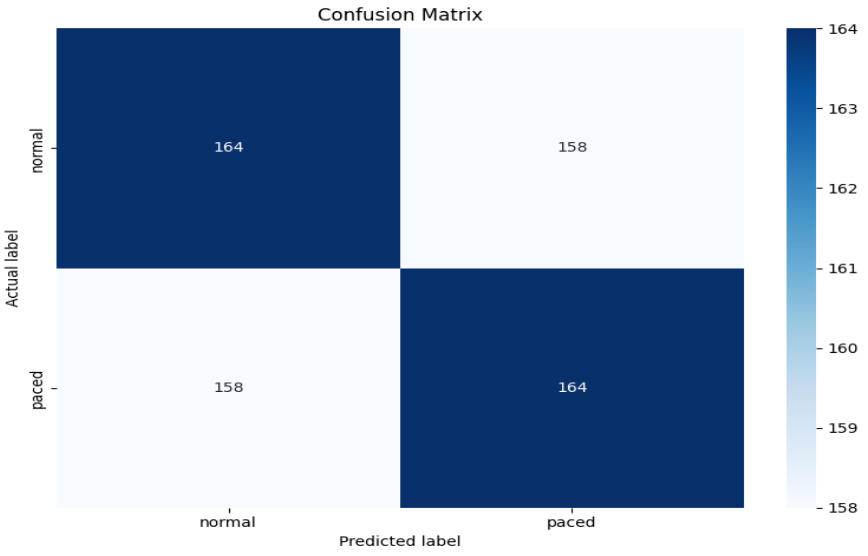
Performance metrics:

Accuracy: 0.509 (50.9 %)

Precision: 0.509 (50.9 %)

Recall: 0.509 (50.9 %)

F1 Score: 0.509 (50.9 %)



Fast Fourier Transform (FFT)

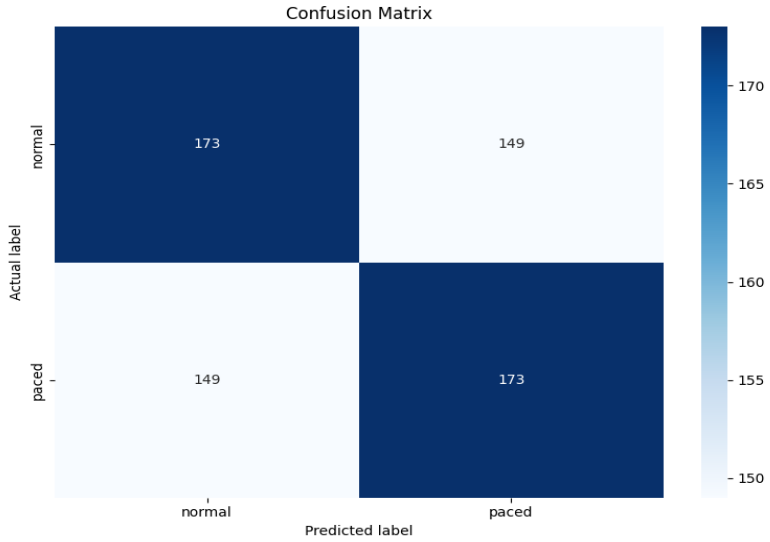
Performance metrics:

Accuracy: 0.537 (53.7 %)

Precision: 0.537 (53.7 %)

Recall: 0.537 (53.7 %)

F1 Score: 0.537 (53.7 %)



Short-Time Fourier Transform (STFT)

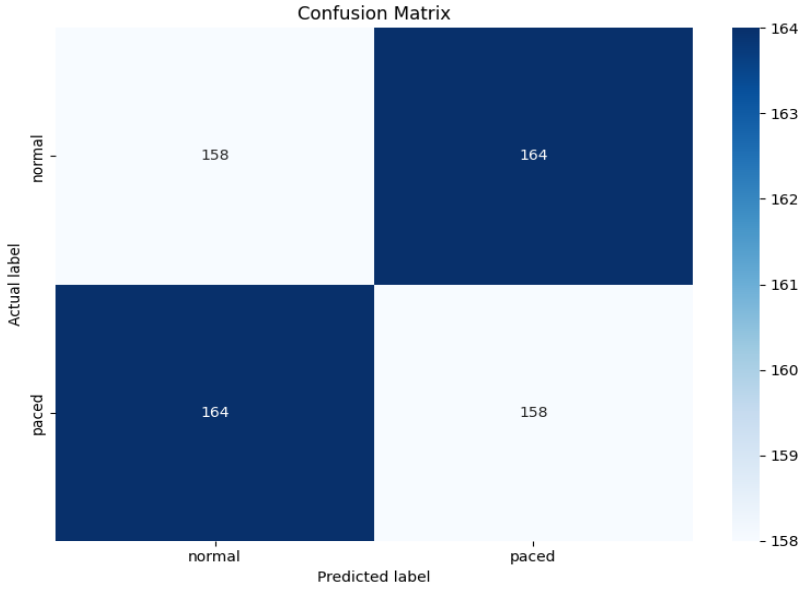
Performance metrics:

Accuracy: 0.490 (49 %)

Precision: 0.490 (49 %)

Recall: 0.490 (49 %)

F1 Score: 0.490 (49 %)



Continuous Wavelet Transform (CWT)

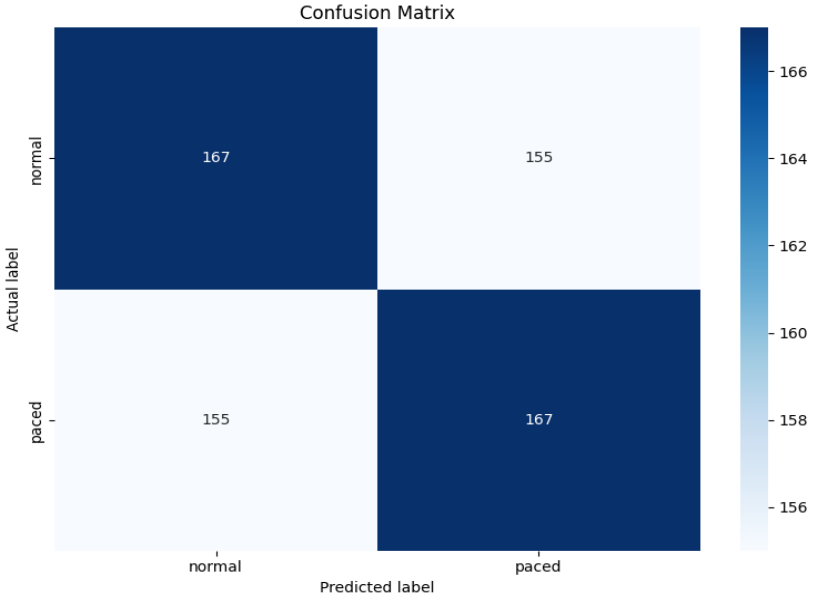
Performance metrics:

Accuracy: 0.518 (51.8 %)

Precision: 0.518 (51.8 %)

Recall: 0.518 (51.8 %)

F1 Score: 0.518 (51.8 %)



Results & Discussion

After training and evaluating our minimal 2D CNN model, we came to a conclusion about the performance of its transformation method. As we saw before, the FFT method, followed by the CWT method and then all the rest methods, give me biggest accuracy score. However, not only all the methods have generally similar performance metrics but each separate method give the exact performance metrics. This can also be intepreted by the produced confusion matrix. Each model evaluation with a specific dimension transformation method each time, has the same number of TP,FP,FN and FP. This may be caused by a number of reasons:

- **The model may predicts only one class**: To create this binary classification problem we made use of two classes, the 'normal' beats and the 'paced' beats. Both beats may appear the same. The former is a normal heart beat while the latter is a electronically controlled heart beat from a cardiac pacemaker which in the end may appear as a normal beat. This outcome could indicate that the model has some predictive power but is struggling to effectively discriminate between classes.
- **Model under-fitting**: The minimal 2D CNN was used due to the fact that the bigger and more complex CNN models, showed over-fitting issues. Although, a simpler model with the appropriate modifications will be challenged to produce sufficient results and avoid over-fitting, a minimal 2D CNN may be too simple for the required task, thus showing evidence of under-fitting.
- **Poor data quality or feature representation**: The data might not contain enough informative features, or the features used might not be represented in a way that allows the model to learn more complex patterns. This scenario can limit the model's ability to perform better, essentially creating a performance ceiling.

A number of actions can be done in order to combat the unwanted results and help produce meaningful insights from the comparison of the dimension transformation methods:

- **Larger data-set and better preprocessing**: The MIT-BIH database has sufficient data with many kinds of arrhythmia. Extracting signals that correspond to arrhythmia and implementing a careful

preprocessing methodology can not only produce a much larger data-set but also a more reliable data-set, ready for feeding a 2D CNN model.

- **Model complexity**: Upon acquiring a larger data-set with more than two labels, we can implement a more complex 2D CNN model (VGG16, ResNet50 etc) in order to both classify arrtyhmia events and conduct a thorough dimension transformation comparison on the methods we discussed earlier.

Conclusion

Summing up this endeavor to compare various dimension transformation methodologies, as a takeaway message we noticed that the FFT and CWT performed slightly better than other transformation methods, but all methods were within a certain percentage range. This may be a result of the possible errors we discussed earlier and future modifications need to take place in order to confirm our findings or discover new ones. All the scripts that have been used in this project are uploaded at my GitHub account (https://github.com/GeoLek/Transforming-1-D-Machine-Learning-Problems-to-2-D-Towards-Using-Convolutional-Neural-Networks).