CSE3020 – Data Visualization

Project Report

Heart Disease Detection from Phonocardiogram data using Gramian Angular Field Transform and CoAtNet-7

By

Reg. No	Name
19BCE1180	Avireddy NVSRK Rohan
19BCE1195	Hrudhvik Nangineni
19BCE1627	Preetham Sagireddy

B. Tech Computer Science and Engineering

Submitted to

Dr. Pattabiraman V

School of Computer Science and Engineering

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

April 2022

DECLARATION

I hereby declare that the report titled "**Heart Disease Detection from Phonocardiogram data using Gramian Angular Field Transform and CoAtNet-7"** submitted by me to VIT Chennai is a record of bona-fide work undertaken by me under the supervision of **Dr.Pattabiraman V**, School of Computer Science and Engineering, Vellore Institute of Technology, Chennai.

Signature of the Candidate

Register Number	***Student Name***	***Signature***
19BCE1180	Avireddy NVSRK Rohan	*Avireddy Rohan*
19BCE1195	Hrudhvik Nangineni	*Hrudhvik*
19BCE1627	Preetham Sagireddy	*Preetham*

CERTIFICATE

Certified that this project report entitled "**Heart Disease Detection from Phonocardiogram data using Gramian Angular Field Transform and CoAtNet-7"** is a bonafide work of **Avireddy NVSRK Rohan (19BCE1180), Hrudhvik Nangineni (19BCE1195) , Preetham Sagireddy (19BCE1627)** and they carried out the Project work under my supervision and guidance for CSE3020 - Data Visualization.

Dr. Pattabiraman V
SCOPE, VIT Chennai

ACKNOWLEDGEMENT

We would like to express our sincere thanks and gratitude to our Data Visualization project guide Dr. Pattabiraman V for giving us the opportunity to work on this project. We are very grateful for the support and guidance in completing this project.

Register Number	***Student Name***
19BCE1180	Avireddy NVSRK Rohan
19BCE1195	Hrudhvik Nangineni
19BCE1627	Preetham Sagireddy

ABSTRACT

The human heart produces many types of sounds that take a significant amount of experience for doctors to discern with a stethoscope. The main thing to detect heart disease are high frequency, noise-like sounds called heart murmurs that are generated due to turbulence in blood flow through narrow cardiac valves or reflow through the atrioventricular valves. Phonocardiography, unlike electrocardiography measures the mechanical vibration signals produced by the heart and are very helpful to detect presence of such diseases. We plan to convert the time series signal data generated into images using Gramian Angular Difference Field and applying a deep neural network on the produced images to detect the presence of heart murmurs.

CONTENTS

	Declaration	i
	Certificate	ii
	Acknowledgement	iii
	Abstract	iv
1	Introduction	6
	1.1 Objective and goal of the project	6
	1.2 Problem Statement	6
	1.3 Motivation	6
2	Literature Survey	7
3	Requirements Specification	9
	3.1 Hardware Requirements	9
	3.2 Software Requirements	9
4	Methodology	10
	4.1 Data Preprocessing	10
	4.2 Data Segmentation	11
	4.3 Gramian Angular Field	13
	4.4 CoAtNet	15
5	Implementation of System	
6	Results & Discussion	
7	Conclusion and Future Work	
8	References	
	Appendix	

1. Introduction

Features in univariate and multivariate time series data have problems like temporal correlation and difficulty in training with RNNs and the existing models struggle with multiple input variables in input data. Since image classification algorithms have received a huge boom in the world of machine learning, computers have had a multi-fold increase in pattern recognition and classification capabilities of images. A technique pioneered in the field is image style transfer application is the use of a Gram Matrix to convert time-series data into visual cues that can be trained by an image classification algorithm which produces much better results. A classic challenge for image classification datasets is the ImageNet problem proposed in 2010. As of 2022 the top of the leaderboard belongs to the Convolution and self-Attention Neural Network model (CoAtNet7). By fine-tuning this pre-trained model we aim to create a novel PCG signal classification algorithm.

1.1 Objective and goal of the project

The goal of the project is to create a PCG signal classification model which will output - abnormal for signals with heart murmurs and normal for a healthy heart. The main objectives will be to segment the heart signal to find the primary heart sounds: S1 and S2. Then the segmented PCG time-series data is converted into an image using Gramian Angular Sum Field (GASF) and Gramian Angular Difference Field (GADF). The final objective is to train an image classification model that will classify abnormal and normal heart signals.

1.2 Problem Statement

Categorizing a phonocardiogram into either of two two states - normal or abnormal based on the present of irregular vibrations called heart murmurs.

1.3 Motivation

An automated system that would assist medical professionals to identify patients-at-risk is the need of the hour. We believe our system can help the populous by informing

them about the danger-factors of heart murmurs and awareness about any congenital heart disease.

2. Literature Survey

Li. et,al proposed a 1D convolutional neural network (CNN) model which directly classifies heart sound signals into either normal or abnormal without the assistance of an ECG signal. Deep feature extraction was employed using a Denoising Auto encoder. The extracted features are then given as the input to the 1-D CNN. The author also mentioned that using Deep features rather than conventional MFCC features improved classification performance but the greater number of layers in the 1-D CNN prolonged the training time.

Potes.et.al approached the problem via an ensembling solution combined with the CNN algorithm.A modified Adaboost model(Adaboost-abstain) and a dropout included CNN were used to generate probability scores for the normal and the abnormal classification.Those probabilities were then given as an input to an MLP which made the final decision. As the research interest in the application of machine learning on biomedical signal data is increasing, we went on to explore more such sol.

Jiaming Wang et .al, the dataset that has been used for the model was the PCG signals of 86 pediatric patients where 24 are normal and 62 have CHD murmur. The signals are downsampled and normalized to get the values between -1 and 1 to remove the differences among the magnitudes. Normalized signals are then denoised using a discrete wavelet transform. The denoised PCG signals are then segmented using discrete wavelet transform with Daubechies 6 wavelet basis function and Hardmard product for finding S1 and S2. 10 time-frequency features and extracted from the segmented PCG signals. The authors used ANN(artificial neural networks) model which contains 10 neurons in the input layer, 10 neurons in the hidden layer and 1 output layer. Levenberg-Marquardt optimization algorithm [26] was used for optimizing the weight parameters. The learning rate with initial start of 0.01, maximum 1000 iterations and target of 10^-6. The result achieved by this model is 93% accuracy, sensitivity of 93.5%% and specificity of 91.7%. The drawbacks are the denoising method used does not remove strong noises of children like crying and moving during the recording. The sample size was relatively low.

Wei Chen et .al, performed a comparative study of heart sound segmentation studies and found patterns and best practices in techniques. Their observation was that the commonly used denoising methods are wavelet denoising, empirical mode decomposition denoising, and digital filter denoising. And new avenues of denoising lie in wavelet basis function. For heartbeat segmentation the most popular methods include envelope-based methods , ECG or carotid signal methods, probabilistic model methods, feature-based methods, and time–frequency analysis methods. Among these methods, those that utilize the cardiac cycle and an ECG signal, based on the correspondence between the ECG QRS waveform and the heart sound signals, have been found to yield better segmentation performance.

Alaxar et al, proposed a model which uses a convolution neural network called AlexNet.The authors focused on two approaches for distinguishing abnormality in PCG signals with 2016 PhysioNet/CinC Challenge dataset. Alexnet was used as feature extractor and Support vector machine was used for classification on the scalogram of pcg dataset.The Alexnet and SVM models achieved an accuracy of 87.65%, this was an improvement of 85% accuracy obtained by end-to-end learning AlexNet in comparison to the benchmarked technique. Scalograms use discrete wavelet transforms which have shift sensitivity, poor directionality, and lack of phase information.

Chao-Lung Yang et .al, The authors of this paper used a time-series dataset obtained from semiconductor manufacturing to generate an aggregated image of two dimensions using the Gramian Angular Field technique. Their main aim was to classify defective and non-defective wafers. After obtaining the images via the Gramian Angular Summation and Difference Fields they used a convolutional neural network consisting of two convolutional layers with a kernel size of 5 x 5, two pooling layers of size 2x2 and one output layer. Their approach yielded much lower error than others that used just the one-dimensional time-series data.

Springer et al. proposed a probabilistic model using the Hidden Markov Principle to segment PCG signals. They further improved upon it by using the heart sound annotations from corresponding ECG waves to make it a Hidden Semi Markov Model. They trained the model on over 10000 real world PCG recordings and achieved a F1 score of 95.8% which is more than 10% over the previous best model. Based on the probabilities they created an extended Viterbi to classify the signals into four states of the heartbeat: S1, Systole, S2, Diastole.

Krishnan et .al classified PCG signals directly using Deep Neural Networks. They pre-processed the signals to remove noise and then fed the unsegmented signal to a 1D-CNN for classification.

3 Requirements Specification

3.1 Software Requirements

Since we had to do heavy pre-processing and generate images we used a multi-pronged approach with various tools in our arsenal. These include:

- GNU Octave / Mathworks MATLAB for Schmidt noise removal, Butterworth filter and Springer Segmentation of PCG waves
- The neurokit2 python library to generate r-peak and end-t wave annotations for ECG files in training-a of the dataset
- Python provided us with many packages to handle the segmented wav data and ready it for training. This includes the standard assortment of numpy, pandas and scipy for preprocessing. Pyts was used for generating the time correlations among the heartbeats
- Training was done using the Tensorflow and keras frameworks

3.1 Hardware Requirements

Google Colab Compute Engine was used to train the model. It has the following specifications:

- Nvidia K80 or Nvidia P100 GPU depending on the allocation provided
- 2 vCPUs
- 24 GB of RAM
- Upto 16 GB of Graphics RAM

4 Methodology

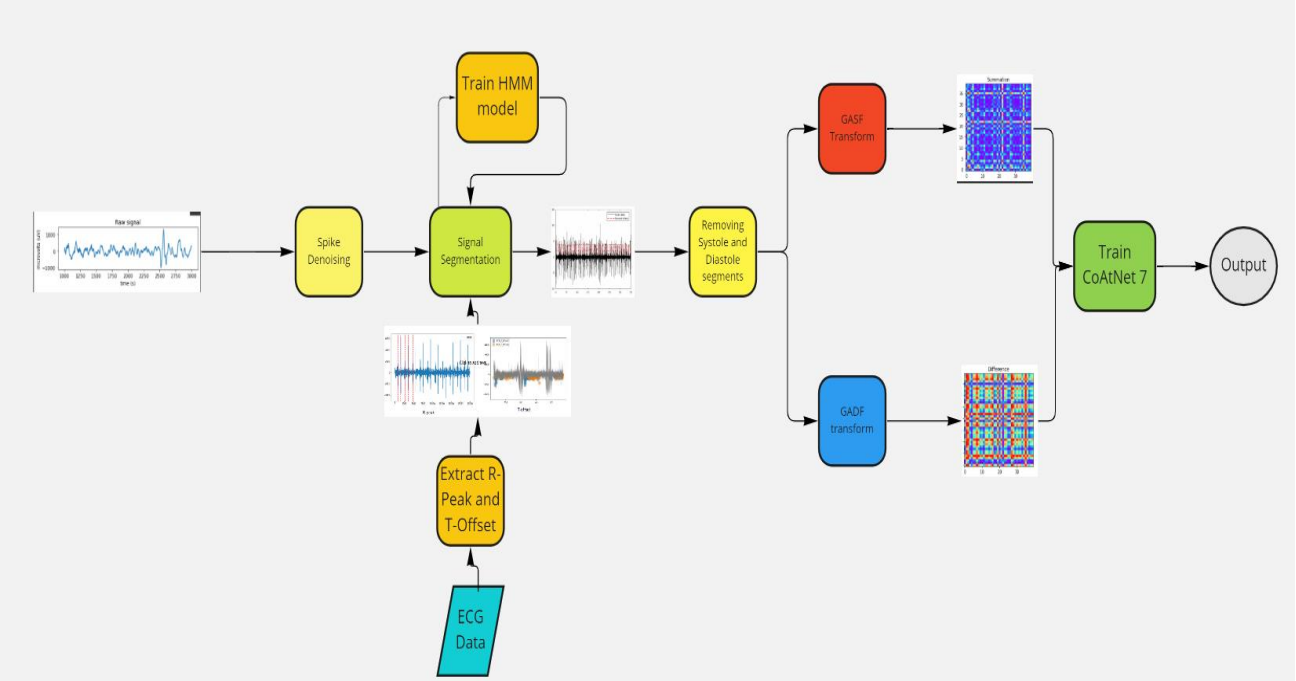

Figure 1 - Project Methodology

4.1 Data pre-processing

For the pre-processing of the pcg signals,First, we applied a spike denoising algorithm. This algorithm splits the signal into 500ms windows. The maximum absolute amplitude (MAA) in each window is found. If at least one MAA exceeds three times the median value of the MAA's, the following steps were carried out. If not continue to

(a) The window with the highest MAA was chosen.

(b) In the chosen window, the location of the MAA point was identified as the top of the noise spike.

(c) The beginning of the noise spike was defined as the last zero-crossing point before the MAA point.

(d) The end of the spike was defined as the first zero-crossing point after the maximum point.

(e) The defined noise spike was replaced by zeroes.

This process is repeated until the condition is met.

4.2 **Data Segmentation**

Developed by David Springer this algorithm breaks down the heart signals into the 4 parts of the beat viz a vis: S1,Systole,S2,Diastole.Murmurs are mostly found in S1 and S2 phase and hence we need segmentation to find out the parts of the signal.Springer proposed the use of a Hidden Semi-Markov Model to segment PCG signals.

A Hidden Markov Model is one where the observable result is caused due to the influence of hidden states without information about the duration of the state the observation is in. Additional parameters of R-peak and T-offset are extracted from ECGs in training-a portion of the dataset to train this model. This gives information about the duration of each state in the heart signal. Thus, making this a Hidden Semi-Markov Model.

R-peak: The R-peak time in a specific ECG lead is the interval from the earliest onset of the QRS complex, preferably determined from multiple simultaneously recorded leads, to the peak (maximum) of the R wave or R' if present.

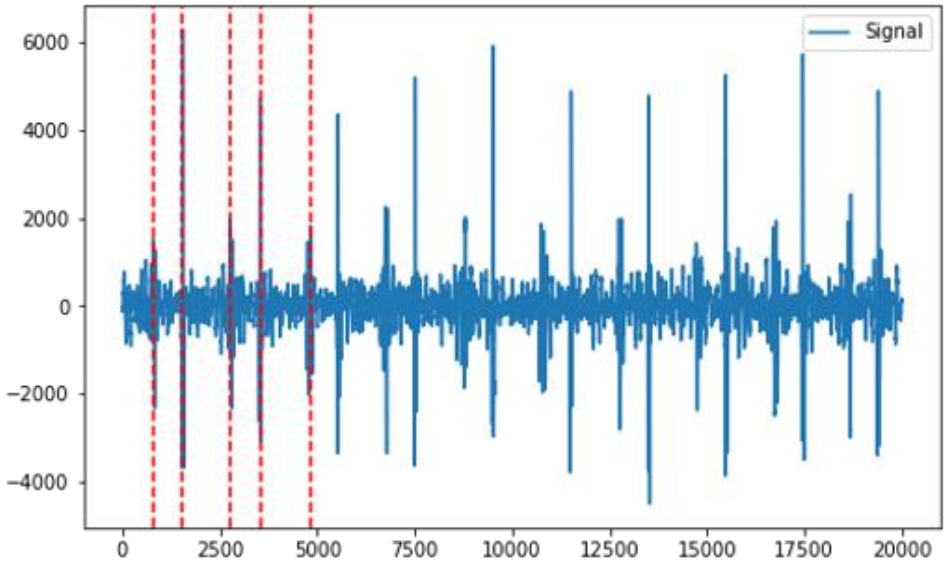

Figure 2 - R-Peak

T-offset: The end of the T wave which represents the relative refractory period. Also called the end T wave.

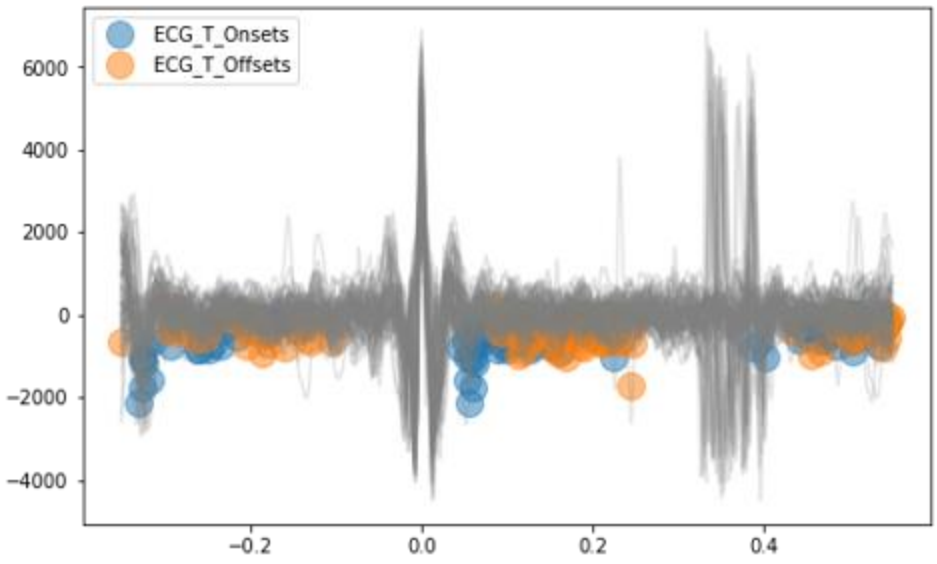

Figure 3 - T-Offset

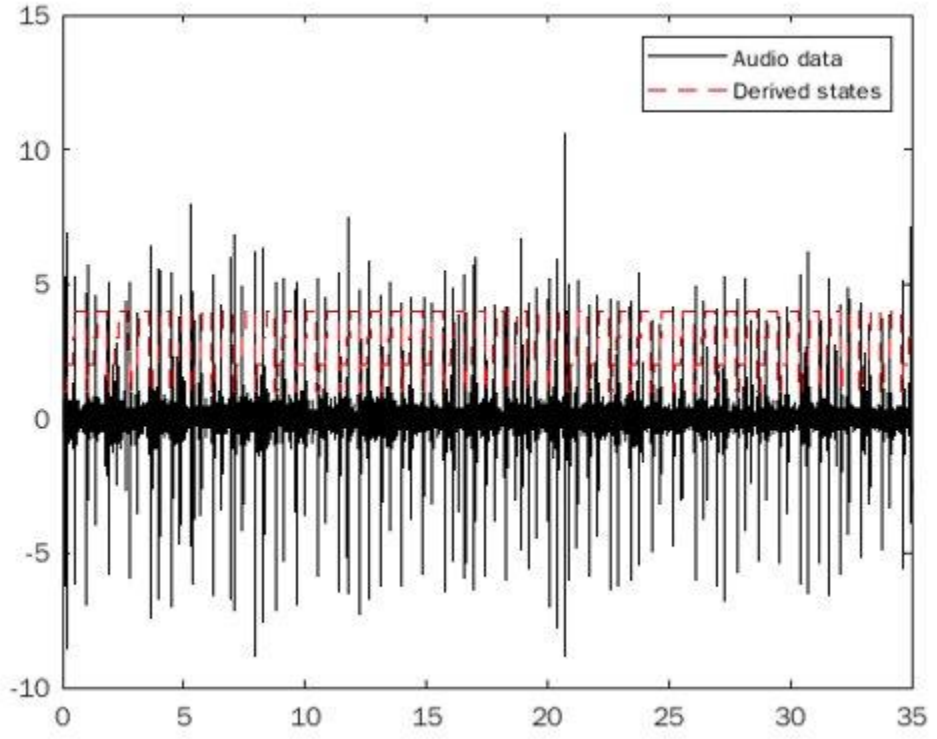

Figure 4 - Sample Segmented Signal

4.3 Gramian Angular Field

Gramian angular transform is a polar-encoding technique for time-series data. This technique elegantly uses the polar coordinates system to generate a 2-D image from the cartesian coordinates. There are two types of GAF namely Gramian Angular Summation Field(GASF) and Gramian Angular Difference Field(GADF).

During mapping into the polar coordinate system, the normalized time series value is encoded as angular cosine, and time stamps as the radius in the polar coordinate system.Given a time series,the map produces one and only one result in the polar coordinate system with a unique inverse map. Second, as opposed to Cartesian coordinates, polar coordinates preserve absolute temporal relations(Imaging Time-Series to Improve Classification and Imputation). .The absolute temporal relationship has been preserved in the polar coordinate system as compared with the Cartesian coordinate system("Classification of time-series images using deep convolutional neural networks,) However, 2-d image generated because the output image will be noisy. So we encode the time stamp as well to have a bijective mapping between the 1-d time series with 2-d image so as not to lose any information.

$$\begin{cases} \emptyset = \arccos(\tilde{x_i}), -1 \leq \tilde{x_i} \leq 1, \tilde{x_i} \in \tilde{X} \\ r = \dfrac{t_i}{N}, t_i \in \mathbb{N} \end{cases}$$

GASF and GADF assist in exploiting the angular perspective by considering the trigonometric sum/difference between each point to identify the temporal correlation within different time intervals((Imaging Time-Series to Improve Classification and Imputation)

$$GASF = \left[\cos\left(\emptyset_i + \emptyset_j\right)\right] \quad (4)$$

$$= \tilde{X}'.\tilde{X} - \sqrt{I - \tilde{X}'^2}.\sqrt{I - \tilde{X}^2} \quad (5)$$

$$GASF = \left[\sin\left(\emptyset_i - \emptyset_j\right)\right] \quad (6)$$

$$= \sqrt{I - \tilde{X}'^2}.\tilde{X} - \tilde{X}'.\sqrt{I - \tilde{X}^2} \quad (7)$$

I is the unit row vector [1, 1, ..., 1]. After transforming to the polar coordinate system, we take the time series at each time step as a 1-D metric space. By defining the inner product < x, y >= x·y− √ 1 − x 2 · p 1 − y 2 and < x, y >= √ 1 − x 2 · y−x· p 1 − y 2 a gramian matrix has been formed and each element of the matrix is the trigonometric function of sum or difference within different time intervals. The GASF and GADF can be obtained by (6) and (7) respectively.

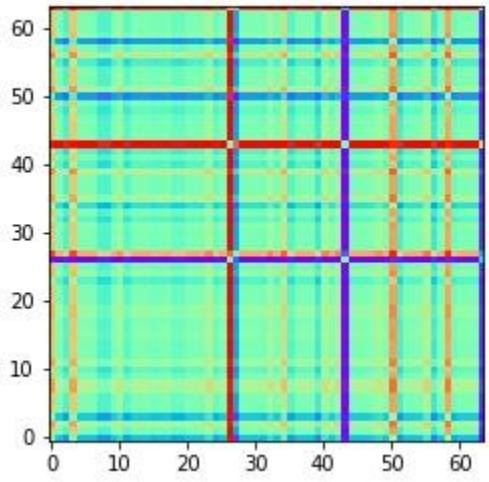

Figure 5 - Gramian Angular Difference Field for PCG signal

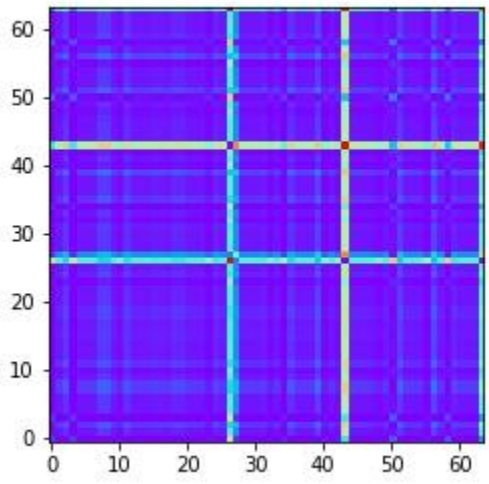

Figure 6 - Gramian Angular Summation Field for PCG signal

4.4 **CoAtNet**

CoAtNet is a model made by Google's Brain Team that seeks to combine Convolution and Attention mechanisms into a single image classification model. It was trained on the ImageNet benchmark dataset and holds the current record with a score of 90.7%. The use of attention mechanisms was popularized in the field of Natural Language Processing by Sadhwani et .al in their 2017 paper - Attention Is All You Need. The main factor for their rapid growth in usage is that they require much less training data to achieve comparable performance to existing machine learning models. However, when we come back into the field of image classification Convolutional Neural Networks (CNNs) still reigned supreme. However they come with a major drawback of requiring huge piles of training data which is simply not available for niche topics. Thus the team at Google Brain sought to combine the performance of CNNs with the learning capacity of Attention mechanisms and CoAtNet was born.

A CoAtNet block is a combination of the MConv convolution cell and a self-attention cell. Stacking them in a specific manner leads to significant increase in performance compared to pure CNNs.

CNNs have the upper edge in Translation Equivariance whereas Attention based models lead in Input-adaptive Weighting and Global Receptive Field.

The Google Brain team employed the following methods to implement the model

(A) Perform some down-sampling to reduce the spatial size and employ the global relative attention after the feature map reaches manageable level.

(B) Enforce local attention, which restricts the global receptive field G in attention to a local field L just like in convolution.

(C) Replace the quadratic Softmax attention with certain linear attention variant which only has a linear complexity w.r.t. the spatial size.

The main advantage of this model is the configuration of Attention and Convolution cells possible, leading to its usage in diverse situations depending, on whether model capacity or generalization is the main factor.

Since this model needs to be fine-tuned we used a final convolution layer followed by binning based on probability values to receive results on the test dataset.

5 Implementation

The physionet training data was separated into 5 folders each containing the wav files of the pcg data. We used the Spike denoising algorithm to remove the noise from the PCG signals. However the first folder also contains some ECG data. We extracted the R-peak and T-offset from the ECG data and then used the

R-peak and T-offset as annotations for segments from the PCG data. We then trained the Springer Segmentation algorithm which inherently depends on the Hidden Markov model to generate the segments for the PCG data. All the PCG signals were segmented.

Segment Label	PCG data segment
1	S1
2	Systole
3	S2
4	Diastole

Table 1 – Four Phases Of A Heartbeat

We then set the amplitudes of Systole and Diastole to 0 as mur-murs exist in the S1 and S2 part of the PCG signal. Now we applied Gramian Transform and generated a summation and a different field dataset. We then split the data into 70% training and 30% testing and started training the CoatNet model for 10 epochs. Batch size was set to 1 and image input size was set to 64x64. Adam optimizer was used for optimizing the loss function. and Sparse categorical accuracy was used as the optimizing metric.

6. Results and Discussion

The CoAtNet Model was run for 10 epochs for GASF and GADF respectively. The model was trained for 10 epochs with batch size as 1. The following table shows various performance metrics while training the model. There was significant improvement in the training sparse categorical accuracy, there was also significant improvement in the Loss.

Epoch	Loss	Sparse Categorical Accuracy	Validation Loss	Validation Sparse Categorical Accuracy
1	0.6525	0.7710	0.7523	0.7975
2	0.5445	0.7927	0.5054	0.7975
3	0.5220	0.7972	0.5214	0.7975
4	0.5307	0.7954	0.5045	0.7975
5	0.5147	0.7972	0.5561	0.7975
6	0.5166	0.7972	0.5096	0.7975
7	0.5188	0.7967	0.5064	0.7975
8	0.5177	0.7972	0.5078	0.7975
9	0.5125	0.7972	0.5145	0.7975
10	0.5138	0.7972	0.5104	0.7975

Table 2 - CoAtNet and GASF Training and Validation results

Figure 7 - Training Loss plot for CoAtNet and GASF model

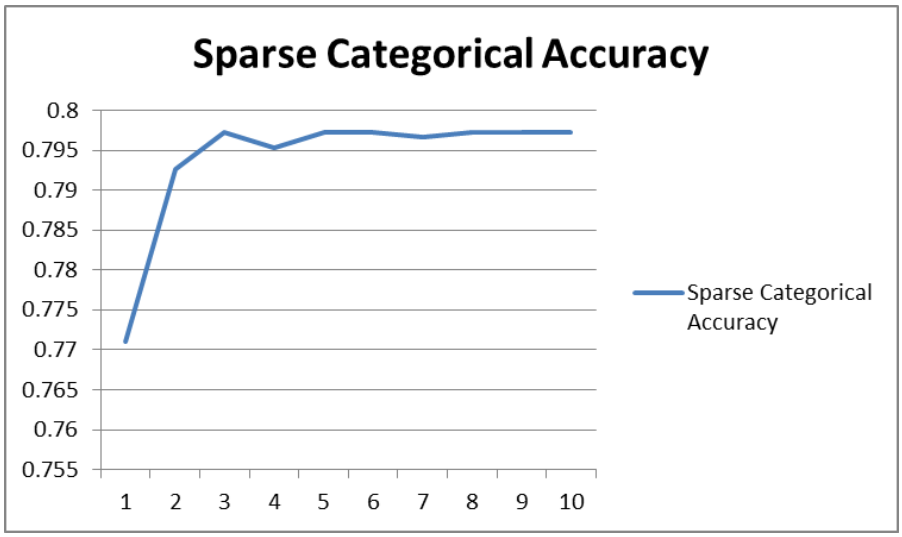

Figure 8 - Training Sparse Categorical Accuracy plot for CoAtNet and GASF model

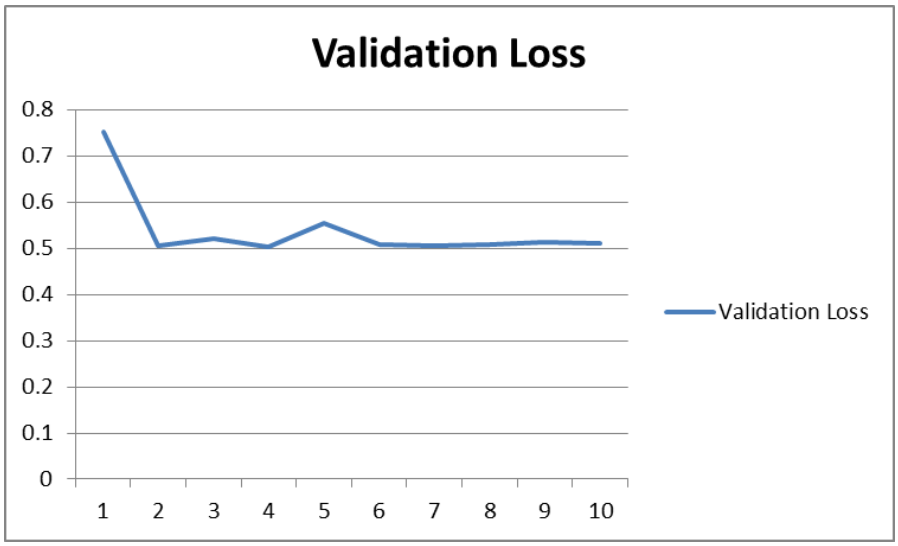

Figure 9 - Validation Loss plot for CoAtNet and GASF model

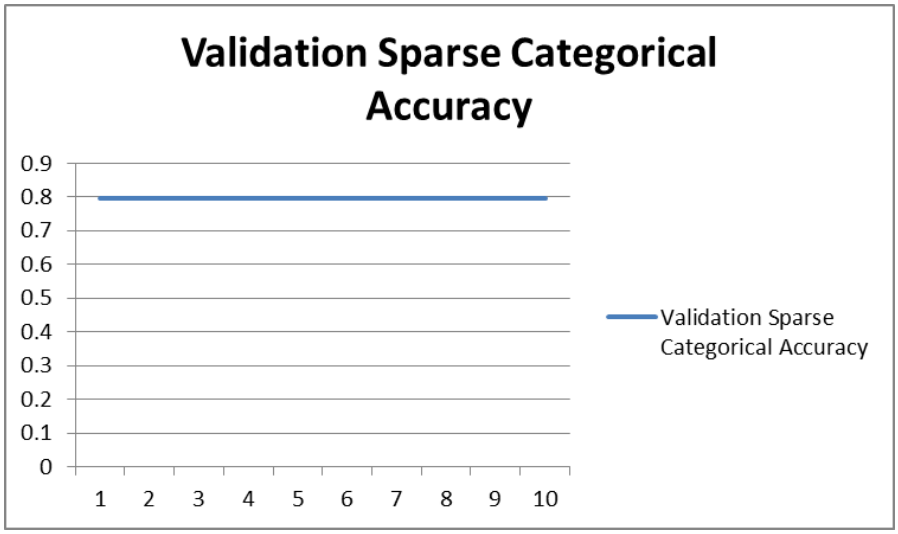

Figure 10 - Validation Sparse Categorical Accuracy plot for CoAtNet and GASF model

Epoch	Loss	Sparse Categorical Accuracy	Validation Loss	Validation Sparse Categorical Accuracy
1	0.6590	0.7768	0.5076	0.7975
2	0.5392	0.7941	0.5083	0.7975
3	0.5284	0.7972	0.5039	0.7975
4	0.5305	0.7963	0.5155	0.7975
5	0.5183	0.7972	0.5039	0.7975
6	0.5136	0.7972	0.5055	0.7975
7	0.5163	0.7972	0.5039	0.7975
8	0.5171	0.7958	0.5091	0.7975
9	0.5300	0.7967	0.5038	0.7975
10	0.5142	0.7972	0.5069	0.7975

Table 3 - CoAtNet and GADF Training and Validation results

Figure 11 - Training Loss plot for CoAtNet and GADF model

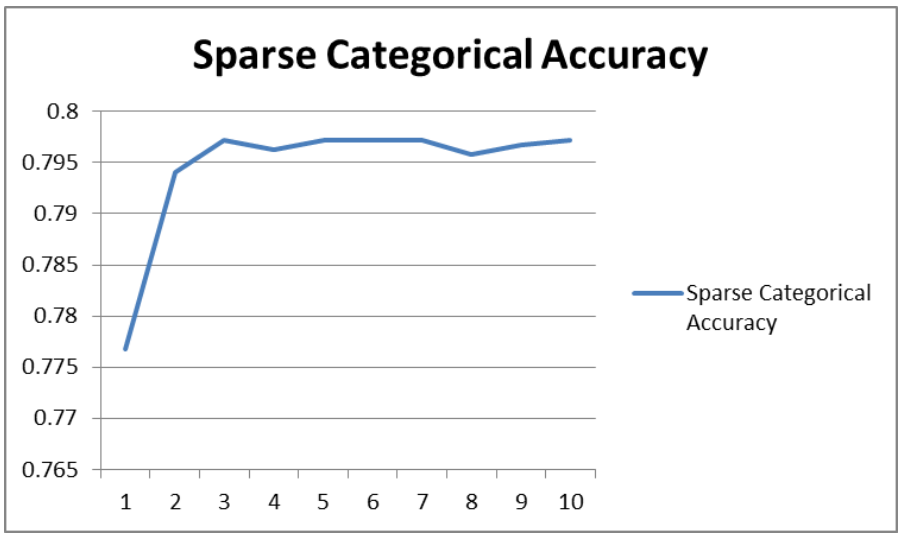

Figure 12 - Training Sparse Categorical Accuracy plot for CoAtNet and GADF model

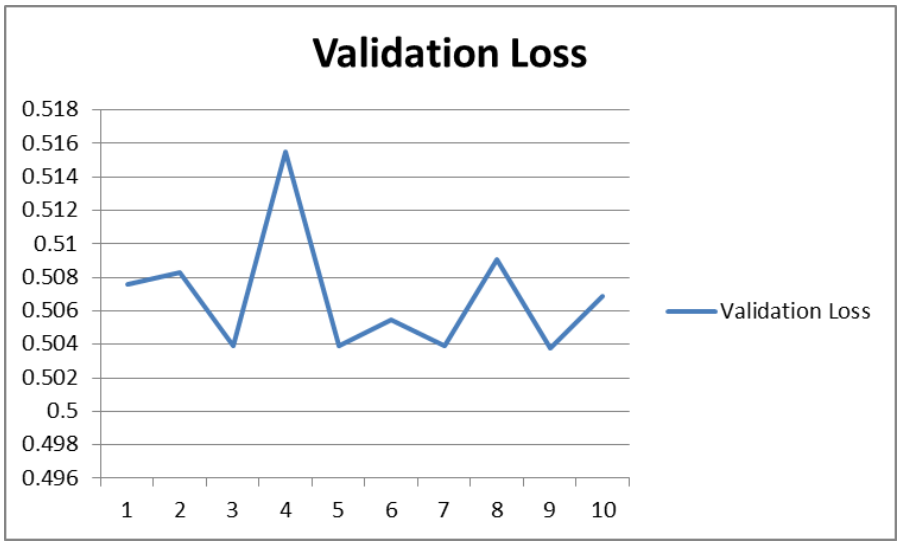

Figure 13 - Validation Loss plot for CoAtNet and GADF model

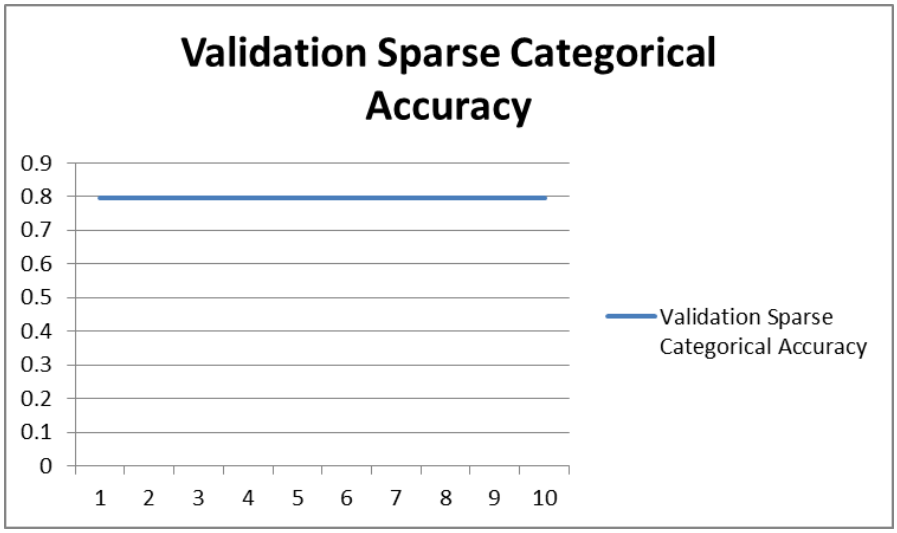

Figure 14 - Validation Sparse Categorical Accuracy plot for CoAtNet and GADF model

Testing Results

CoAtNet and GASF model gave a loss of 0.5104 and sparse categorical accuracy of 0.79752.CoAtNet and GADF model gave a loss of 0.5069 and sparse categorical accuracy of 0.79752.

Data	**Sparse Categorical test accuracy**	**Loss**
GASF	0.79752.	0.5104
GADF	0.79752	0.5069

Table 4 – Comparision Of Test Accuracy Of GASF and GADF

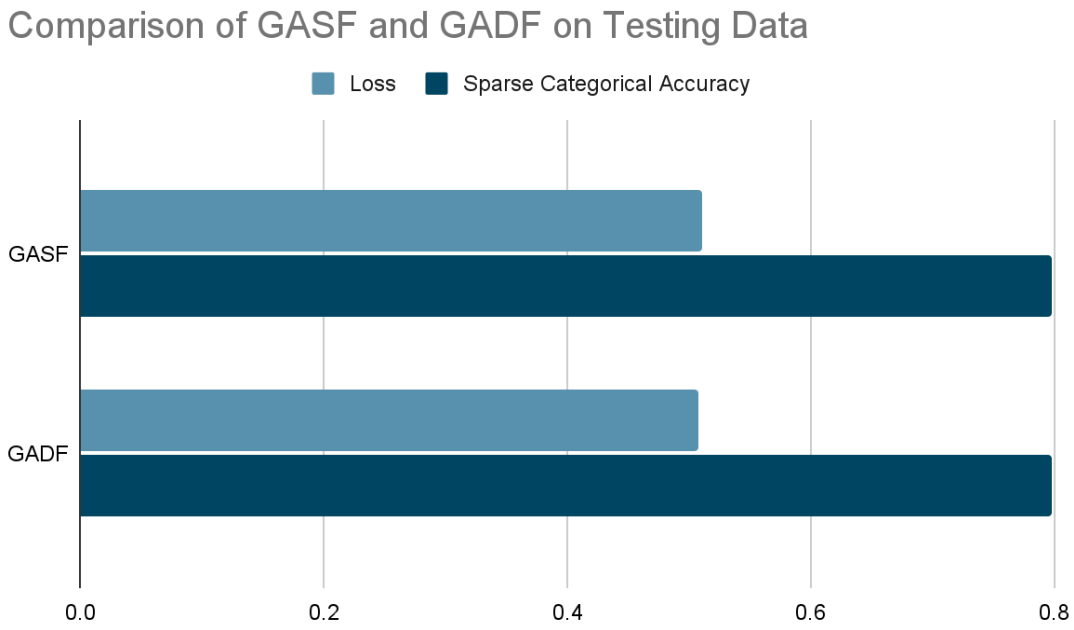

Figure 15 – Barchart Comparing Accuracy of GASF and GADF

7. Conclusion and Future Work

Our proposed model utilizes the technique of image-encoding of time series data by using both gramian summation and gramian difference fields. We first denoised the signals and segmented it by using the Springer algorithm, We then applied the Gramian transform on the segmented signals and then trained and tested our CoAtNet 7 model. We achieved identical performance on both GASF and GADF datasets using CoAtNet 7. Although the accuracy of the model was at 80% , we aim to further optimize the model and experiment with new models which are performing very well on the ImageNet data.

8. REFERENCES

[1] Shankar, A., Khaing, H. K., Dandapat, S., & Barma, S. (2020, October). Epileptic seizure classification based on Gramian angular field transformation and deep learning. In 2020 IEEE Applied Signal Processing Conference (ASPCON) (pp. 147-151). IEEE.

[2] Krishnan, P. T., Balasubramanian, P., & Umapathy, S. (2020). Automated heart sound classification system from unsegmented phonocardiogram (PCG) using deep neural network. Physical and Engineering Sciences in Medicine, 43(2), 505-515.

[3] Liu, X., Hu, Q., Yuan, H., & Yang, C. (2020, October). Motion artifact detection in ppg signals based on gramian angular field and 2-d-cnn. In 2020 13th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI) (pp. 743-747). IEEE.

[4] Xu, G., Ren, T., Chen, Y., & Che, W. (2020). A one-dimensional cnn-lstm model for epileptic seizure recognition using eeg signal analysis. Frontiers in Neuroscience, 14, 1253.

[5] Wang, Z., & Oates, T. (2015, April). Encoding time series as images for visual inspection and classification using tiled convolutional neural networks. In Workshops at the twenty-ninth AAAI conference
on artificial intelligence.

[6] Sun, X., Xv, H., Dong, J., Zhou, H., Chen, C., & Li, Q. (2020). Few-shot learning for domain-specific finegrained image classification. IEEE Transactions on Industrial Electronics, 68(4), 3588-3598.

[7] Zhang, G., Si, Y., Wang, D., Yang, W., & Sun, Y. (2019). Automated detection of myocardial infarction using a gramian angular field and principal component analysis network. IEEE Access, 7, 171570-171583

[8]Krishnan, P. T., Balasubramanian, P., & Umapathy, S. (2020). Automated heart sound classification system from unsegmented phonocardiogram (PCG) using deep neural network. Physical and Engineering Sciences in Medicine, 43(2), 505-515.

[9]Sun, S., & Ren, J. (2021). GASF–MSNN: A New Fault Diagnosis Model for Spatiotemporal Information Extraction. Industrial & Engineering Chemistry Research, 60(17), 6235-6248.

[10]Yang, C. L., Yang, C. Y., Chen, Z. X., & Lo, N. W. (2019, January). Multivariate time series data transformation for convolutional neural network. In 2019 IEEE/SICE International Symposium on System Integration (SII) (pp. 188-192). IEEE.
[11]Potes, C., Parvaneh, S., Rahman, A., & Conroy, B. (2016, September). Ensemble of feature-based and deep learning-based classifiers for detection of abnormal heart sounds. In 2016 computing in cardiology conference (CinC) (pp. 621-624). IEEE.
[12]Liu, X., Hu, Q., Yuan, H., & Yang, C. (2020, October). Motion artifact detection in ppg signals based on gramian angular field and 2-d-cnn. In 2020 13th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI) (pp. 743-747). IEEE.
[13]Bahekar, L., Misal, A., Rawate, M. R., kumar Singh, V., Mandurkar, S., Pardhi, A., & Gosatwar, P. (2019). Segmentation of PCG Signal: A Survey.
[14]Dai, Z., Liu, H., Le, Q. V., & Tan, M. (2021). Coatnet: Marrying convolution and attention for all data sizes. Advances in Neural Information Processing Systems, 34, 3965-3977.

APPENDIX

Github-Link: here

Codes:

Denoising :

```
%path =
"/common/VIT/sem6/data_viz_theory/data_viz_project/2016_dataset/training/training
_combined"
path =
"/common/VIT/sem6/data_viz_theory/data_viz_project/2016_dataset/training/remaini
ng_wav/"
files = dir(path)
for file = files'
  search = strfind(file.name, "wav")
  if(length(search) > 0)
    new_name = strcat("denoised_", file.name)
    [audio, sample_rate] = audioread(strcat(path, file.name))
    result = schmidt_spike_removal(audio, sample_rate)
    audiowrite(new_name, result, 2000)
  endif
end
```

Segmentation:

```
import os
import re
import pandas as pd
dir =
"/common/VIT/sem6/data_viz_theory/data_viz_project/2016_dataset/training/hea_co
mbined/"
labels = []pcg_0 = csvread("0.csv");
pcg_0_transpose = transpose(pcg_0);
```

```
pcg_0_training.wav_data = pcg_0_transpose;
save("pcg_0_training.mat", "pcg_0_training");
pcg_1 = csvread("1.csv");
pcg_1_transpose = transpose(pcg_1);
pcg_1_training.wav_data = pcg_1_transpose;
save("pcg_1_training.mat", "pcg_1_training");
pcg_2 = csvread("2.csv");
pcg_2_transpose = transpose(pcg_2);
pcg_2_training.wav_data = pcg_2_transpose;
save("pcg_2_training.mat", "pcg_2_training");
pcg_3 = csvread("3.csv");
pcg_3_transpose = transpose(pcg_3);
pcg_3_training.wav_data = pcg_3_transpose;
save("pcg_3_training.mat", "pcg_3_training");
pcg_4 = csvread("4.csv");
pcg_4_transpose = transpose(pcg_4);
pcg_4_training.wav_data = pcg_4_transpose;
save("pcg_4_training.mat", "pcg_4_training");
pcg_5 = csvread("5.csv");
pcg_5_transpose = transpose(pcg_5);
pcg_5_training.wav_data = pcg_5_transpose;
save("pcg_5_training.mat", "pcg_5_training");
pcg_6 = csvread("6.csv");
pcg_6_transpose = transpose(pcg_6);
pcg_6_training.wav_data = pcg_6_transpose;
save("pcg_6_training.mat", "pcg_6_training");
```

```
files_with_errors = [[], [438, 441, 451, 452, 456, 462, 464, 465, 468, 470, 472, 482],
[454], [339, 351], [88, 187], [], []]
x = sorted(os.listdir(dir))
```

```
for i in range(len(x)):
    with open(os.path.join(dir, x[i])) as f:
        text = f.read()
        list = re.findall(r"# (\w+)", text)
        labels.append(list[0])
print(len(labels))
labels_chunks = [labels[x:x+500] for x in range(0, len(labels), 500)]
for i in range(len(labels_chunks)):
    if len(files_with_errors[i]) > 0:
        for index in files_with_errors[i]:
            # labels_chunks[i].pop(index-1)
            labels_chunks[i][index-1] = "ERROR"
i = 0
for chunk in labels_chunks:
    a = pd.DataFrame(chunk)
    a.to_csv(".".join(["complete_labels_"+str(i), "csv"]), header = False, index = None)
    i+=1
```

```
clc;
clear all;

load B_matrix.mat;
load pi_vector.mat;
load('pcg_0_training_mat7.mat');
% load('pcg_1_training_mat7.mat');
% load('pcg_2_training_mat7.mat');
% load('pcg_3_training_mat7.mat');
% load('pcg_4_training_mat7.mat');
% load('pcg_5_training_mat7.mat');
% load('pcg_6_training_mat7.mat');
wav_cell = num2cell(pcg_0_training.wav_data, 1);
test_recordings = wav_cell([1:409]);
%test_recordings = pcg_1_training.wav_data;
%test_recordings = pcg_2_training.wav_data;
```

```
%test_recordings = pcg_3_training.wav_data;
%test_recordings = pcg_4_training.wav_data;
%test_recordings = pcg_5_training.wav_data;
%test_recordings = pcg_6_training.wav_data;
% test_annotations = pcg_a_training.annotations([21:50],:);
size(test_recordings)
%% Run the HMM on an unseen test recording:
% And display the resulting segmentation
numPCGs = length(test_recordings);
saved_states = [];
errors = [];
for PCGi = 1:numPCGs
   try
      [assigned_states] = runSchmidtSegmentationAlgorithm(test_recordings{PCGi}, 2000, B_matrix, pi_vector, true);
      saved_states(PCGi,:) = assigned_states;
   catch
      errors(end+1) = PCGi;
      fprintf('Inconsistent data in iteration %d, skipped.\n', PCGi);
   end
end

save("saved_states_0_training_a.mat", "saved_states");
csvwrite("saved_states_0_training_a.csv", saved_states);
csvwrite("errors0_training_a.csv", errors);
% save("saved_states_1.mat", "saved_states");
% save("saved_states_2.mat", "saved_states");
% save("saved_states_3.mat", "saved_states");
% save("saved_states_4.mat", "saved_states");
% save("saved_states_5.mat", "saved_states");
% save("saved_states_6.mat", "saved_states");
```

Gramian:

```
# -*- coding: utf-8 -*-
"""Graimian Angular Field.ipynb

Automatically generated by Colaboratory.

Original file is located at
   https://colab.research.google.com/drive/1Nwa8txsfX-Zw0hX2Igl534SZss1VsTZB
"""

from google.colab import drive
drive.mount('/content/drive')
```

```
pip install pyts

import numpy as np
import matplotlib.pyplot as plt
from pyts.image import GramianAngularField
from mpl_toolkits.axes_grid1 import ImageGrid
import pandas as pd

data = pd.read_csv('/content/drive/MyDrive/DATA VISUALIZATION/2016_dataset/final_segmented_data/set1.csv')

data

data=data.fillna(0)

data.isna().sum()

for i in range(500):
  temp=data.iloc[i,1:].to_numpy()
  temp = temp.reshape(1, -1)
  gasf = GramianAngularField(method='summation',image_size=64)
  X_gasf = gasf.fit_transform(temp)

  fig = plt.figure()
  grid = ImageGrid(fig, 111,
                   nrows_ncols=(1, 1),
                   axes_pad=0.15,
                   share_all=True,
                   )
  images = [X_gasf[0]]
  #titles = ['Summation', 'Difference']
  for image, ax in zip(images, grid):
    im = ax.imshow(image, cmap='rainbow', origin='lower')
    #ax.set_title(title, fontdict={'fontsize': 12})
  #ax.cax.colorbar(im)
  #ax.cax.toggle_label(True)
  #plt.suptitle('Gramian Angular Fields', y=0.98, fontsize=16)
  if(data.iloc[i,0]=='Abnormal'):
    plt.savefig('/content/drive/MyDrive/DATA
VISUALIZATION/2016_dataset/Image_data/GASF/Abnormal/gasf_set1_'+str(i)+'.jpg')
  elif(data.iloc[i,0]=='Normal'):
    plt.savefig('/content/drive/MyDrive/DATA
VISUALIZATION/2016_dataset/Image_data/GASF/Normal/gasf_set1_'+str(i)+'.jpg')
  elif(data.iloc[i,0]=='ERROR'):
```

```
   plt.savefig('/content/drive/MyDrive/DATA
VISUALIZATION/2016_dataset/Image_data/GASF/Error/gasf_set1_'+str(i)+'.jpg')
  else:
   plt.savefig('/content/drive/MyDrive/DATA
VISUALIZATION/2016_dataset/Image_data/GASF/Unsure/gasf_set1_'+str(i)+'.jpg')
  plt.show()

for i in range(500):
  temp=data.iloc[i,1:].to_numpy()
  temp = temp.reshape(1, -1)
  gasf = GramianAngularField(method='difference',image_size=64)
  X_gadf = gasf.fit_transform(temp)

  fig = plt.figure()
  grid = ImageGrid(fig, 111,
                nrows_ncols=(1, 1),
                axes_pad=0.15,
                share_all=True,
                )
  images = [X_gadf[0]]
  #titles = ['Summation', 'Difference']
  for image, ax in zip(images, grid):
   im = ax.imshow(image, cmap='rainbow', origin='lower')
   #ax.set_title(title, fontdict={'fontsize': 12})
  #ax.cax.colorbar(im)
  #ax.cax.toggle_label(True)
  #plt.suptitle('Gramian Angular Fields', y=0.98, fontsize=16)
  if(data.iloc[i,0]=='Abnormal'):
   plt.savefig('/content/drive/MyDrive/DATA
VISUALIZATION/2016_dataset/Image_data/GADF/Abnormal/gadf_set1_'+str(i)+'.jpg')
  elif(data.iloc[i,0]=='Normal'):
   plt.savefig('/content/drive/MyDrive/DATA
VISUALIZATION/2016_dataset/Image_data/GADF/Normal/gadf_set1_'+str(i)+'.jpg')
  elif(data.iloc[i,0]=='ERROR'):
   plt.savefig('/content/drive/MyDrive/DATA
VISUALIZATION/2016_dataset/Image_data/GADF/Error/gadf_set1_'+str(i)+'.jpg')
  else:
   plt.savefig('/content/drive/MyDrive/DATA
VISUALIZATION/2016_dataset/Image_data/GADF/Unsure/gadf_set1_'+str(i)+'.jpg')
  plt.show()
```

COAT NET 7:

```
import tensorflow as tf
```

```
import ssl
# ssl._create_default_https_context = ssl._create_unverified_context

# def pipe(data, batch_size = 128, shuffle = False):
#     dataset = tf.data.Dataset.from_tensor_slices(data)
#     if shuffle:
#         dataset = dataset.shuffle(buffer_size = batch_size * 10)
#     dataset = dataset.batch(batch_size)
#     #dataset = dataset.prefetch((batch_size * 2) + 1)
#     dataset = dataset.prefetch(tf.data.experimental.AUTOTUNE)
#     return dataset

# (tr_x, tr_y), (te_x, te_y) = tf.keras.datasets.cifar10.load_data()
```

In [1]:

In [2]:

```
# tr_x = tr_x * 1/255
# te_x = te_x * 1/255

# batch_size = 128

# tr_data = pipe((tr_x, tr_y), batch_size = batch_size, shuffle = True)
# te_data = pipe((te_x, te_y), batch_size = batch_size, shuffle = False)
```

In [3]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [4]:

```
%cd '/content/drive/MyDrive/DATA VISUALIZATION/2016_dataset/gaf_images'
```

/content/drive/.shortcut-targets-by-id/1psvNNdzjOZtPf9i4b7gJzkYmUTZL7w5O/DATA VISUALIZATION/2016_dataset/gaf_images

In [5]:

```
import pandas as pd
import numpy as np
import os
from keras.preprocessing.image import ImageDataGenerator, img_to_array, load_img, array_to_img
import matplotlib.image as mpimg
from  matplotlib import pyplot as plt
%matplotlib inline
```

In [6]:

```
IMG_WIDTH=64
IMG_HEIGHT=64
batch_size=1

train_dir = r'/content/drive/MyDrive/DATA VISUALIZATION/2016_dataset/gaf_images/Image_data/gadf_split/train'
test_dir =  r'/content/drive/MyDrive/DATA VISUALIZATION/2016_dataset/gaf_images/Image_data/gadf_split/test'
val_dir =  r'/content/drive/MyDrive/DATA VISUALIZATION/2016_dataset/gaf_images/Image_data/gadf_split/val'
```

In [7]:

```
image_gen_train = ImageDataGenerator(rescale=1./255,
                                     zoom_range=0.2,
                                     rotation_range=65,
                                     shear_range=0.09,
                                     horizontal_flip=True,
                                     vertical_flip=True)
```

In [8]:

```
train_data_gen = image_gen_train.flow_from_directory(batch_size=batch_size,
```

```
                                        directory=train_dir,
                                        shuffle=True,
                                        target_size=(IMG_HEIGHT, IMG_WIDTH),
                                        class_mode='sparse')
```

```
Found 2258 images belonging to 2 classes.
```

In [9]:

```
image_gen_val = ImageDataGenerator(rescale=1./255)
val_data_gen = image_gen_val.flow_from_directory(batch_size=batch_size,
                                                 directory=val_dir,
                                                 target_size=(IMG_HEIGHT, IMG_WIDTH),
                                                 class_mode='sparse')
```

```
Found 484 images belonging to 2 classes.
```

In [10]:

```
train_data_gen.class_indices.keys()
```

Out[10]:

```
dict_keys(['Abnormal', 'Normal'])
```

In [11]:

```
!pip install einops
```

```
Collecting einops
  Downloading einops-0.4.1-py3-none-any.whl (28 kB)
Installing collected packages: einops
Successfully installed einops-0.4.1
```

In [12]:

```
def pipe(data, batch_size = 128, shuffle = False):
    dataset = tf.data.Dataset.from_tensor_slices(data)
    if shuffle:
        dataset = dataset.shuffle(buffer_size = batch_size * 10)
    dataset = dataset.batch(batch_size)
```

```
    #dataset = dataset.prefetch((batch_size * 2) + 1)
    dataset = dataset.prefetch(tf.data.experimental.AUTOTUNE)
    return dataset
```

In [13]:

```
# tr_data = pipe((train_data_gen, train_data_gen.labels), batch_size = batch_size, shuffle
= True)
# val_data = pipe((val_data_gen, val_data_gen.labels), batch_size = batch_size, shuffle =
False)
```

In [14]:

```
import coatnet

model = coatnet.coatnet7(input_shape = (64, 64, 3), include_top = False,classes=2)

flatten = tf.keras.layers.GlobalAveragePooling2D()(model.output)
drop_out = tf.keras.layers.Dropout(0.5)(flatten)
dense = tf.keras.layers.Dense(2048, activation = "relu")(drop_out)
prediction = tf.keras.layers.Dense(10, activation = "softmax", name = "prediction")(dense)
model = tf.keras.Model(model.input, prediction)
```

In [15]:

```
loss = tf.keras.losses.sparse_categorical_crossentropy
opt = tf.keras.optimizers.Adam(1e-4)
metric = [tf.keras.metrics.sparse_categorical_accuracy]
model.compile(loss = loss, optimizer = opt, metrics = metric)
```

In [16]:

```
model.fit(train_data_gen, validation_data = val_data_gen, epochs = 10)
```

```
Epoch 1/10
2258/2258 [==============================] - 1028s 419ms/step - loss: 0.6590 -
sparse_categorical_accuracy: 0.7768 - val_loss: 0.5076 -
val_sparse_categorical_accuracy: 0.7975
Epoch 2/10
```

2258/2258 [==============================] - 772s 342ms/step - loss: 0.5392 - sparse_categorical_accuracy: 0.7941 - val_loss: 0.5083 - val_sparse_categorical_accuracy: 0.7975
Epoch 3/10
2258/2258 [==============================] - 770s 341ms/step - loss: 0.5284 - sparse_categorical_accuracy: 0.7972 - val_loss: 0.5039 - val_sparse_categorical_accuracy: 0.7975
Epoch 4/10
2258/2258 [==============================] - 771s 342ms/step - loss: 0.5305 - sparse_categorical_accuracy: 0.7963 - val_loss: 0.5155 - val_sparse_categorical_accuracy: 0.7975
Epoch 5/10
2258/2258 [==============================] - 769s 340ms/step - loss: 0.5183 - sparse_categorical_accuracy: 0.7972 - val_loss: 0.5039 - val_sparse_categorical_accuracy: 0.7975
Epoch 6/10
2258/2258 [==============================] - 767s 340ms/step - loss: 0.5136 - sparse_categorical_accuracy: 0.7972 - val_loss: 0.5055 - val_sparse_categorical_accuracy: 0.7975
Epoch 7/10
2258/2258 [==============================] - 767s 340ms/step - loss: 0.5163 - sparse_categorical_accuracy: 0.7972 - val_loss: 0.5039 - val_sparse_categorical_accuracy: 0.7975
Epoch 8/10
2258/2258 [==============================] - 767s 340ms/step - loss: 0.5171 - sparse_categorical_accuracy: 0.7958 - val_loss: 0.5091 - val_sparse_categorical_accuracy: 0.7975
Epoch 9/10
2258/2258 [==============================] - 762s 337ms/step - loss: 0.5300 - sparse_categorical_accuracy: 0.7967 - val_loss: 0.5038 - val_sparse_categorical_accuracy: 0.7975
Epoch 10/10
2258/2258 [==============================] - 764s 338ms/step - loss: 0.5142 - sparse_categorical_accuracy: 0.7972 - val_loss: 0.5069 - val_sparse_categorical_accuracy: 0.7975

Out[16]:

```
<keras.callbacks.History at 0x7f7334b68fd0>
```

In [17]:

```
model.save_weights("model_gadf.h5")
```

In [18]:

```
model.load_weights("model_gadf.h5")
```

In [19]:

```
image_gen_test = ImageDataGenerator(rescale=1./255)
test_data_gen = image_gen_test.flow_from_directory(batch_size=batch_size,
                                                   directory=val_dir,
                                                   target_size=(IMG_HEIGHT, IMG_WIDTH),
                                                   class_mode='sparse')
```

```
Found 484 images belonging to 2 classes.
```

In [20]:

```
loss = tf.keras.losses.sparse_categorical_crossentropy
metric = [tf.keras.metrics.sparse_categorical_accuracy]
model.compile(loss = loss, metrics = metric)
model.evaluate(test_data_gen)
```

```
484/484 [==============================] - 84s 128ms/step - loss: 0.5069 -
sparse_categorical_accuracy: 0.7975
```

Out[20]:

```
[0.5069025754928589, 0.797520637512207]
```