



**Alamein International University
Computer Science & Engineering**

**A Convolutional Neural Network Model Classification
Beat-to-Beat Arterial Pressure Time Series:
Stratification and Cardiovascular Risk Prediction**

by

Ahmed Khaled Asaad Hamed Eladl (20100229)

ahmed.eladl@aiu.edu.eg

Supervised by

Dr. Ahmed El-Yazbi | ayazbi@aiu.edu.eg

Duration & Location

August 2023 : October 2023 (3 months)

Faculty of Pharmacy at Alamein International University

Table of Contents

Abstract	3
Introduction	4
Problem Statement	11
Solution	12
Conclusion.....	50

Abstract

Machine learning (ML) models have demonstrated impressive applicability in classifying hemodynamic signals according to disease-specific features of overt cardiovascular (CV) conditions like myocardial infarction and hypertension. However, there remains a pressing need for identifying subtle, subclinical deviations in hemodynamic time-series associated with early disease states, to predict prognosis and prevent deteriorating processes. Our previous work indicates the predictive capacity of non-linear parameters of blood pressure variability (BPV) in discriminating between healthy and prediabetic rats in a sex- and age-specific manner. Whilst pre-diabetic rats lack signs of gross CV insult, they possess blunted baroreceptor sensitivity (BRS) and dysfunctional endothelial-dependent vasorelaxation only discernible upon complex, invasive hemodynamic manipulations. Our results show that female rats are relatively protected from prediabetic CV manifestations.

Here, we evaluate the capacity of a ML convolutional neural network (CNN), the B2B_Net(), in segregating arterial pressure signals collected from pre-diabetic vs. healthy controls according to disease state, age, and sex. 4- to 5-week-old Sprague Dawley, male or female rats were fed a high-calorie (HC) or a normal-calorie (NC) diet for 12 or 24 weeks, representing young vs. old rats, respectively. Female rats fed for 24 weeks were further divided into three subcategories: a sham group, one undergoing ovariectomy (OVX) at week 12, and another similarly ovariectomized and receiving estrogen treatment for 12 weeks post-surgery. At weeks 12 or 24, anesthetized rats were instrumented for invasive hemodynamics monitoring via a pressure transducer inserted through the carotid artery.

Beat-to-beat arterial pressure (AP) signals of length ~300 seconds were downsampled to 50Hz, sliced to 1000 data points per bin (~16.7s), and Wavelet-transformed. Wavelet plots were cropped to remove graph labels/axes and were then fed into the CNN. Our model consisted of 5 convolutional layers, separated by batch normalization, ReLU activation, and max pooling. The outcome of the 5th convolutional layer had 40% of data dropped out and was delivered to a fully connected layer and then to a softmax function. The model utilized cross entropy for a loss function and ADAM for an optimization function. 80% of the data was randomly allocated for training and 10% each for validation and testing. Test accuracy (TA), AUROC, and AUPRC were used to evaluate the model.

Our CNN successfully classified rats based on diet (test accuracy: 94.497%, AUROC: 0.95201, AUPRC: 0.9369), sex (TA: 96.865%, AUROC: 0.98309, AUPRC: 0.98121), and age (TA: 95.579 %, AUROC: 0.96841, AUPRC: 0.95646).

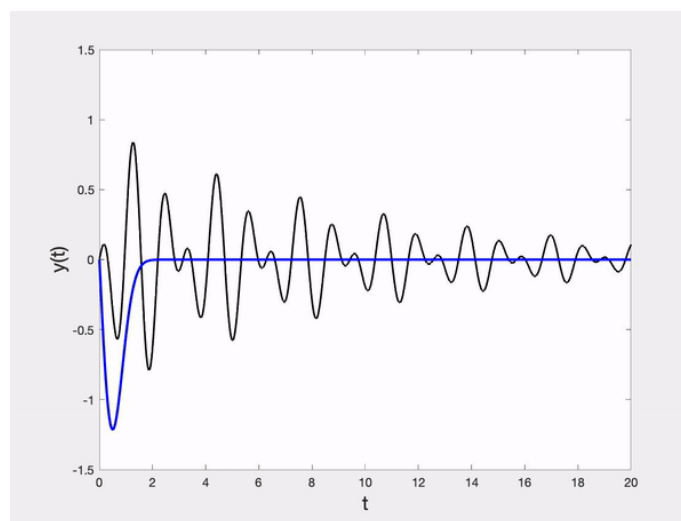
Application of a supervised ML model provides the opportunity of capturing several undefined features of hemodynamic control which cannot be detected by a single isolated measure. Utility of such a CNN overcomes the shortcomings associated with the use of other parameters of hemodynamic fluctuations which measure a single characteristic, by probing and compiling additional layers of differences and synthesizing a global appraisal of CV signals.

Introduction

A wavelet transform is a mathematical technique that decomposes a signal into different frequency components, and then analyzes each component at a resolution that matches its scale. Wavelet transforms are useful for processing signals that have non-stationary or transient features, such as music, images, power, earthquakes, and so on.

How does it work?

Let's take another look at the same animation from before.



Animation of Discrete Wavelet Transform

The **basic idea** is to **compute *how much of a wavelet is in a signal*** for a particular scale and location. For those familiar with convolutions, that is exactly what this is. A signal is convolved with a set wavelets at a variety of scales.

In other words, we pick a wavelet of a particular scale (like the blue wavelet in the gif above). Then, we slide this wavelet across the entire signal i.e. vary its location, where at each time step we multiply the wavelet and signal. The product of this multiplication gives us a coefficient for that wavelet scale at that time step. We then increase the wavelet scale (e.g. the red and green wavelets) and repeat the process.

Difference between Continuous and Discrete

There are two main types of wavelet transforms: continuous and discrete.

- **Discrete Wavelet Transform (DWT)** uses a subset of scales and positions based on powers of two and behaves like a filter bank that splits the signal into approximation and detail coefficients. The DWT can be computed efficiently using a fast algorithm.
- **Continuous Wavelet Transform (CWT)** compares a signal with a family of wavelets that are scaled and shifted versions of a mother wavelet. The result is a coefficient plot that shows how similar the signal is to each wavelet at each scale and position.

Definitions of each type are given in the below figure. The key difference between these two types is the Continuous Wavelet Transform (CWT) uses every possible wavelet over a range of scales and locations i.e., an infinite number of scales and locations. While the Discrete Wavelet Transform (DWT) uses a finite set of wavelets i.e., defined at a particular set of scales and locations.

Continuous Wavelet Transform (CWT)	Discrete Wavelet Transform (DWT)
$T(a, b) = \frac{1}{\sqrt{a}} \int_{-\infty}^{\infty} x(t) \psi^* \frac{(t-b)}{a} dt$	$T_{m,n} = \int_{-\infty}^{\infty} x(t) \psi_{m,n}(t) dt$

Definitions of Continuous and Discrete Wavelet Transforms.

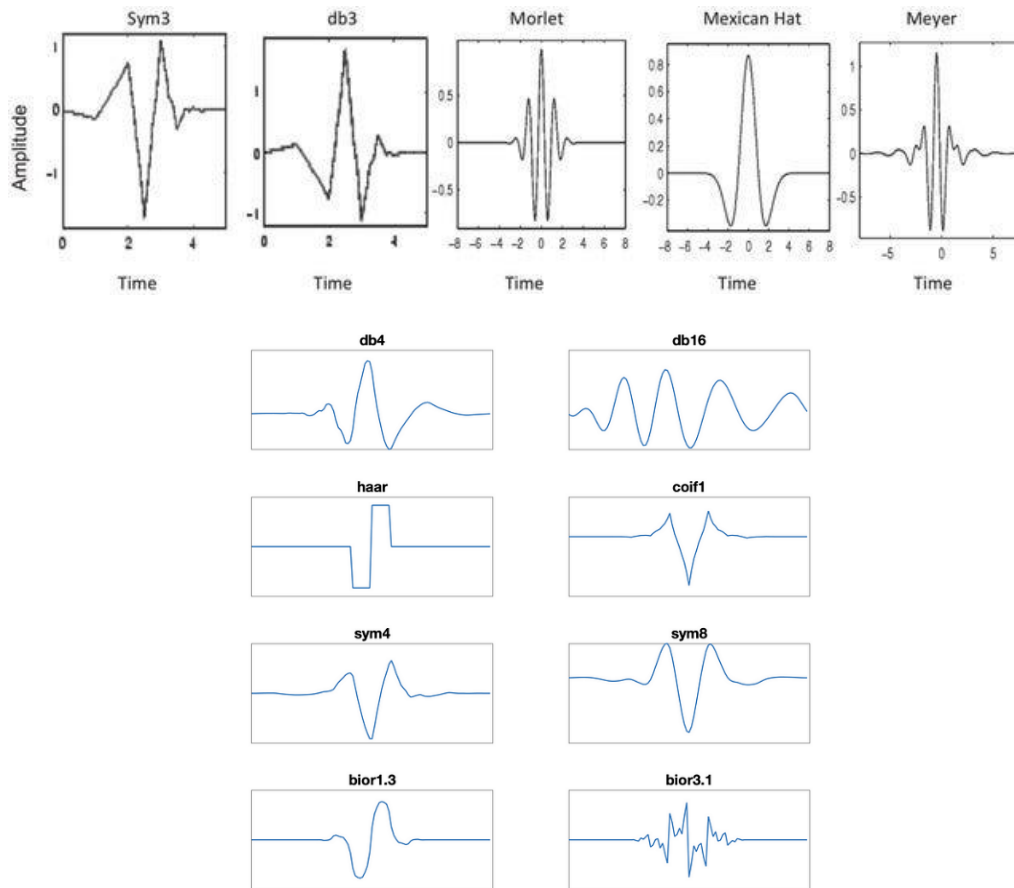
Why wavelets?

A couple of key advantages of the Wavelet Transform are:

- **Wavelet transform** can extract local spectral **and** temporal information simultaneously.
- **Variety of wavelets** to choose from

We have touched on the first key advantage a couple of times already. This is probably the biggest reason to use the Wavelet Transform. This may be preferable to using something like a Short-Time Fourier Transform which requires chopping up a signal into segments and performing a Fourier Transform over each segment.

The second key advantage sounds more like a technical detail. Ultimately, the takeaway here is if you know what characteristic shape you are trying to extract from your signal, there are a wide variety of wavelets to choose from to best *match* that shape. A handful of options are given in the figure below.



Some wavelet families

Why did we use CWT over DWT?

There are several reasons why continuous wavelet transform (CWT) is preferred over discrete wavelet transform (DWT) for heart rate variability (HRV) analysis.

Here are some of them:

- CWT provides a finer resolution of the scale parameter than DWT, which allows for a more detailed and accurate representation of the frequency components of HRV.
- CWT is more robust to noise and artifacts than DWT, which can affect the quality and reliability of HRV analysis.
- CWT can handle non-linear and non-stationary signals better than DWT, which are common characteristics of HRV signals.

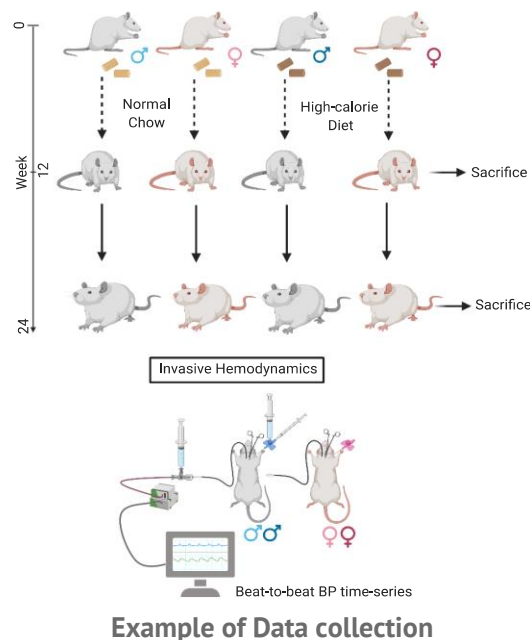
- CWT can use complex wavelets, which can capture the phase information of HRV signals, while DWT is limited to real wavelets.

CWT can use a variety of wavelet shapes that can match the physical properties of HRV signals, while DWT has a restricted choice of wavelets that are based on perfect reconstruction criteria.

Dataset Explanation

We have large datasets derived from experimental animals consist of:

- ID: each column has a unique identifier.
- Sex: male or female.
- Diet: normal chow (NC) or high-fat diet (HFD).
- Age: 24 weeks or 12 weeks.
- Condition: a continuous variable ranging from 0 to 303.999.

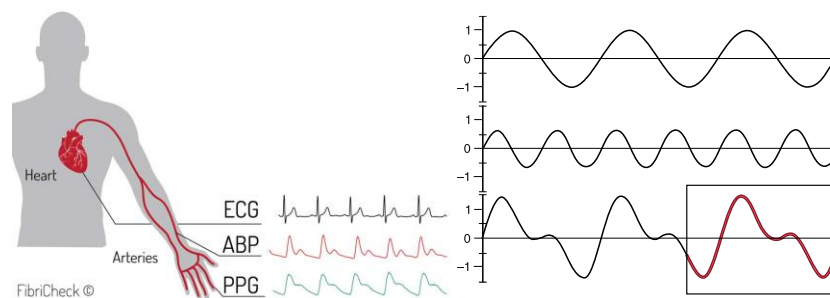


What is a Beat-to-Beat Arterial Pressure Time-Series?

Beat to beat arterial pressure (AP) is a measure of the variation in blood pressure between each heartbeat. It can be used to assess the cardiovascular health and function of an individual, as well as the effects of different interventions or treatments on the heart and blood vessels.

Some of the applications of beat-to-beat AP analysis are:

- A convolutional neural network model that can classify AP time-series according to disease state, age, and sex. This model can help predict the cardiovascular risk and prognosis of patients with pre-diabetes, hypertension, or other conditions.
- A software that can display and calculate various AP parameters, such as systolic, diastolic, mean, pulse pressure, and rate pressure product. This software can help monitor and analyze the hemodynamic signals of animals or humans.
- A formula that can estimate the mean arterial pressure (MAP) from the systolic and diastolic pressures. MAP is an important indicator of the perfusion pressure of vital organs, such as the brain and kidneys.



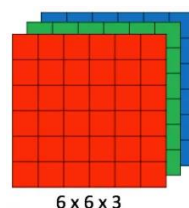
Wavelet transformation of beat-to-beat arterial pressure signals.

Adopted from Mark, J. B. (1998). *Atlas of cardiovascular monitoring*. Churchill Livingstone.

What is the Convolutional Neural Network?

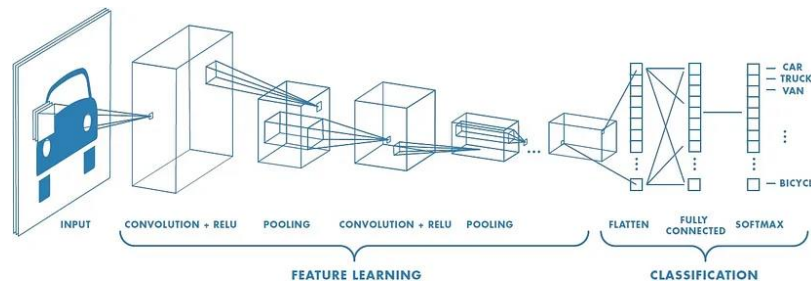
Convolutional neural network (ConvNets or CNNs) is one of the main categories to do images recognition, images classifications. Objects detections, recognition faces etc., are some of the areas where CNNs are widely used.

CNN image classifications takes an input image, process it and classify it under certain categories (Eg., Dog, Cat, Tiger, Lion). Computers sees an input image as array of pixels and it depends on the image resolution. Based on the image resolution, it will see $h \times w \times d$ (h = Height, w = Width, d = Dimension). Eg., An image of $6 \times 6 \times 3$ array of matrix of RGB (3 refers to RGB values) and an image of $4 \times 4 \times 1$ array of matrix of grayscale image.



Array of RGB Matrix

Technically, deep learning CNN models to train and test, each input image will pass it through a series of convolution layers with filters (Kernels), Pooling, fully connected layers (FC) and apply Softmax function to classify an object with probabilistic values between 0 and 1. The below figure is a complete flow of CNN to process an input image and classifies the objects based on values.



Neural network with many convolutional layers

Convolution Layer

Convolution is the first layer to extract features from an input image. Convolution preserves the relationship between pixels by learning image features using small squares of input data. It is a mathematical operation that takes two inputs such as image matrix and a filter or kernel.

- An image matrix (volume) of dimension $(h \times w \times d)$
- A filter $(f_h \times f_w \times d)$
- Outputs a volume dimension $(h - f_h + 1) \times (w - f_w + 1) \times 1$



Image matrix multiplies kernel or filter matrix.

Consider a 5 x 5 whose image pixel values are 0, 1 and filter matrix 3 x 3 as shown in below

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

*

1	0	1
0	1	0
1	0	1

5 x 5 – Image Matrix

3 x 3 – Filter Matrix

Image matrix multiplies kernel or filter matrix.

Then the convolution of 5 x 5 image matrix multiplies with 3 x 3 filter matrix which is called “**Feature Map**” as output shown in below

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

4		

Image

Convolved
Feature

Output matrix

Problem Statement

Pre-diabetes is a condition that affects millions of people worldwide and increases the risk of developing type 2 diabetes and cardiovascular (CV) diseases. However, pre-diabetes is often asymptomatic and undiagnosed, which makes it difficult to prevent or treat its complications. Therefore, there is a need for developing new methods that can detect pre-diabetic individuals and monitor their CV health using non-invasive and reliable biomarkers. One possible source of such biomarkers is the hemodynamic signals, which are the measurements of blood flow and pressure in the CV system. Hemodynamic signals reflect the dynamic interactions between the heart, blood vessels, and autonomic nervous system, and can reveal subtle changes in the CV function that may not be apparent from conventional measures. However, hemodynamic signals are complex and noisy, and require advanced techniques to analyze and interpret them.

Machine learning (ML) is a branch of artificial intelligence that can learn from data and make predictions or classifications based on patterns and features. ML has been widely used in various fields of science and engineering, including CV health. ML models can process large and high-dimensional data sets, such as hemodynamic signals, and extract meaningful information from them. However, most of the existing ML models for CV health focus on overt CV conditions that have clear and obvious symptoms, such as high blood pressure or heart attack. These models may not be able to detect subtle changes in the hemodynamic signals that occur in early stages of CV diseases, such as pre-diabetes. Moreover, these models may not account for the effects of age and sex on the hemodynamic signals, which are known to influence the CV function and risk.

The problem that this study addresses is how to develop a novel ML model that can classify hemodynamic signals based on pre-diabetic status, age, and sex. The proposed model is a convolutional neural network (CNN), which is a type of deep learning model that can process complex and high-dimensional data, such as images or time-series.

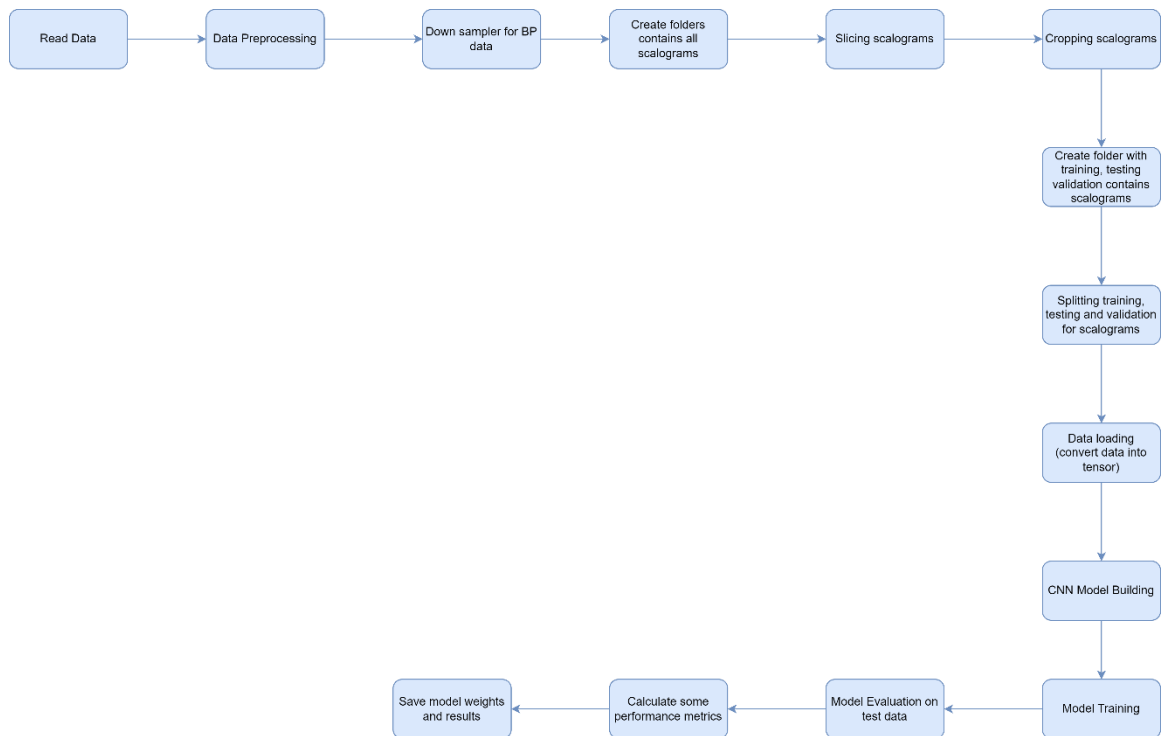
The study uses rats as an animal model because they have similar CV physiology and pathology to humans. The study compares rats that are fed with a high-calorie (HC) diet or a normal (NC) diet for 12 or 24 weeks, representing young or old rats, respectively. The study also compares male and female rats, as well as female rats that undergo ovariectomy (OVX) or estrogen treatment, to investigate the effects of sex hormones on CV health. The study measures the arterial pressure (AP) signals of the rats using an invasive method that involves inserting a pressure transducer through the carotid artery. The study applies a wavelet transform to the AP signals to convert them into images that capture the frequency and time information of the signals. The study then feeds these images into the CNN model, which it calls B2B_Net(), to learn the features that distinguish between different groups of rats. The study evaluates the performance of the CNN model using metrics such as test accuracy, AUROC, and AUPRC.

The goal of this study is to demonstrate the potential of ML in detecting subclinical CV abnormalities in pre-diabetic individuals using hemodynamic signals. The study hopes that the CNN model can provide a new tool for CV diagnosis and prognosis that is more sensitive and specific than conventional methods.

Solution

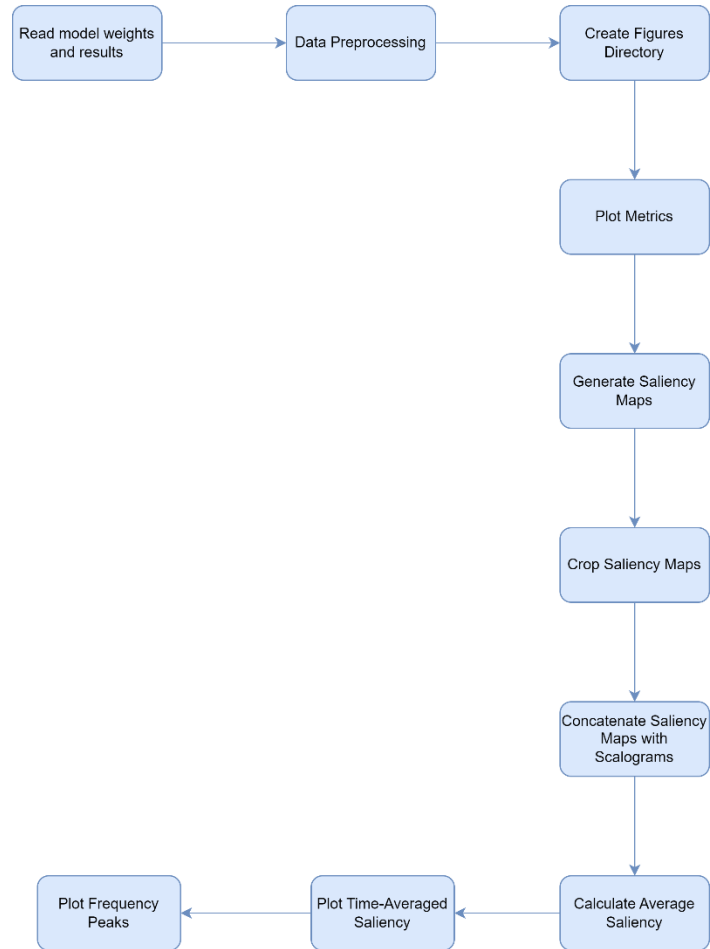
Before discussing the solution in detail, I will first present the entire architecture's pipelines, outlining all the steps involved. Then, I will delve into each step individually, providing comprehensive details for a clear understanding.

The first architecture pipeline pertains to the construction of the “Build a CNN Scalogram Classification Model”. This encompasses the process of designing and training the Convolutional Neural Network (CNN) to classify scalograms generated from continuous blood pressure recordings.



Architecture Pipeline for Build a CNN Scalogram Classification Model

The next crucial step involves the “generation of Saliency Maps for the Scalograms and the subsequent identification of Frequency Peaks”. This process plays a pivotal role in discerning the most significant features within the scalograms that contribute to the classification of healthy and sick subjects based on their blood pressure recordings.



Architecture Pipeline for Generate Saliency Maps for Scalograms and find Frequency Peaks

Build a CNN Scalogram Classification Model

After presenting the comprehensive overview of the entire architecture's pipelines, I will now delve into each one individually, starting with the "Build a CNN Scalogram Classification Model" architecture pipeline. This pipeline encompasses the design, training, and utilization of a Convolutional Neural Network (CNN) for the purpose of classifying scalograms generated from continuous blood pressure recordings. I will provide detailed insights into the intricacies of this pivotal phase.

Read Data

The dataset comprises extensive records obtained from experimental animals. It encompasses several key attributes:

- **ID:** This column serves as a unique identifier for each record, facilitating individual tracking and referencing.
- **Sex:** Indicates the gender of the experimental animal, categorized as either male or female.
- **Diet:** Specifies the dietary regimen to which the animal was subjected. It can be classified as either normal chow (NC) or high-fat diet (HFD).
- **Age:** Provides information about the age of the animals at the time of recording, categorized into two groups: 24 weeks and 12 weeks.
- **Condition:** Represents a continuous variable ranging from 0 to 303.999. This variable holds significant physiological information related to the animals, contributing to the comprehensive characterization of their health status.

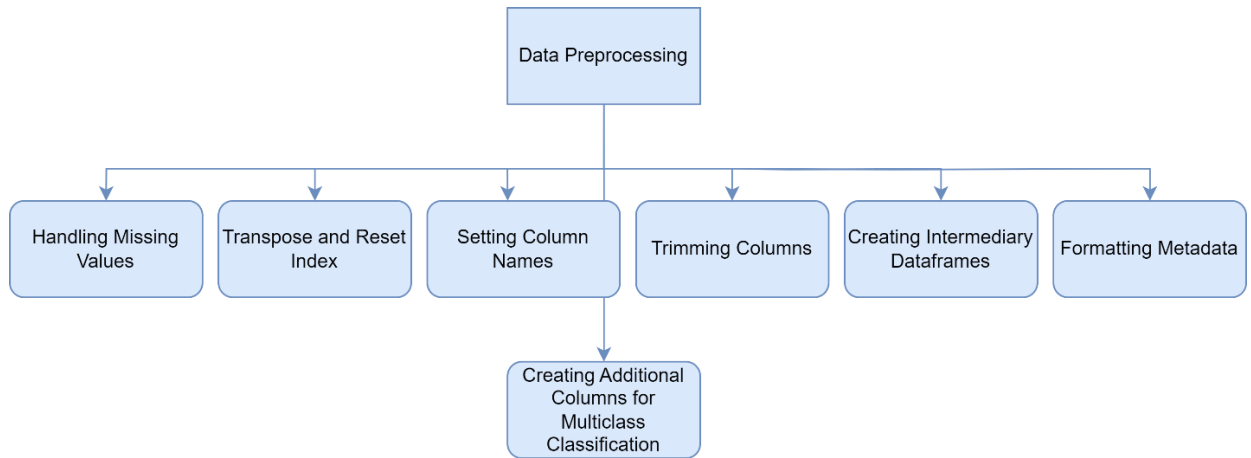
	ID	F1	F9	F10	F17	F18	F25	F3	F4	F11	...	S9	S13	S17	S21	S25
0	Sex	Male	Male	Male	Male	Male	Male	Male	Male	Male	...	Male	Male	Male	Male	Male
1	Diet	NC	NC	NC	NC	NC	NC	HFD	HFD	HFD	...	NC	NC	NC	NC	NC
2	Age	24 wks	24 wks	24 wks	24 wks	24 wks	24 wks	24 wks	24 wks	24 wks	...	12 wks	12 wks	12 wks	12 wks	12 wks
3	Condition	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
4	0	130.1223	130.4467	131.7845	172.6916	154.5692	155.522	131.0142	126.5749	148.0217	...	136.5482	120.8382	156.1301	160.6708	122.845
...
304999	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
305000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
305001	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
305002	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN
305003	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN

305004 rows × 143 columns

View of Arterial Pressure (AP) Dataset

Data Preprocessing

To prepare the dataset for further analysis and modeling, several key preprocessing steps were performed, we will see architecture pipeline in the below figure, and we will talk about each step in order:



Data Preprocessing of Arterial Pressure (AP) Dataset

Handling Missing Values:

- Blank or whitespace values were replaced with Nan (Not a Number) to ensure uniform data quality.

Transpose and Reset Index:

- The data frame was transposed to facilitate further operations, and the index was reset.

Setting Column Names:

- The first row of the transposed data frame was set as the column names.

Trimming Columns:

- The data frame was trimmed to retain only the first 304,005 columns, as per the specified data range.

Creating Intermediary Data frames:

- Two intermediary data frames were created to separate data and metadata for ease of further processing.
- df_data** contains the time series data and was appropriately formatted.

Time	1	2	3	4	5	6	7	8	9	10	...	133	134	135	136
1970-01-01 00:00:00.000	130.1223	130.4467	131.7845	172.6916	154.5692	155.522	131.0142	126.5749	148.0217	155.6233	...	136.5482	120.8382	156.1301	160.6708
1970-01-01 00:00:00.001	129.9399	130.1021	131.4399	173.6038	154.9138	155.2382	130.5075	126.2911	147.7784	156.0085	...	137.2375	120.453	155.8666	159.9005
1970-01-01 00:00:00.002	129.6561	129.6764	131.2778	174.4146	155.4409	154.8733	130.1831	125.9059	147.4946	156.5558	...	138.0888	119.9462	155.6841	159.3532
1970-01-01 00:00:00.003	129.3926	129.1899	130.9737	175.1241	155.7652	154.63	129.7777	125.5613	147.2919	157.0423	...	138.9402	119.6016	155.4814	158.8464
1970-01-01 00:00:00.004	129.0074	128.9466	130.7102	175.9552	156.1098	154.4476	129.575	125.014	147.1906	157.5896	...	139.7511	119.257	155.3193	158.3802

View of df_data

Formatting Metadata:

- df_meta** contains the metadata information such as ID, Sex, Diet, Age, and Surgery (renamed from Condition) which was appropriately formatted.

Creating Additional Columns for Multiclass Classification:

- New columns were created to explore multiclass classification scenarios based on combinations of Sex, Diet, and Age.

	ID	Sex	Diet	Age	Surgery	SexDiet	AgeDiet	SexAge	SexAgeDiet
1	F1	Male	NC	24 wks	NaN	MaleNC	24 wksNC	Male24 wks	Male24 wksNC
2	F9	Male	NC	24 wks	NaN	MaleNC	24 wksNC	Male24 wks	Male24 wksNC
3	F10	Male	NC	24 wks	NaN	MaleNC	24 wksNC	Male24 wks	Male24 wksNC
4	F17	Male	NC	24 wks	NaN	MaleNC	24 wksNC	Male24 wks	Male24 wksNC
5	F18	Male	NC	24 wks	NaN	MaleNC	24 wksNC	Male24 wks	Male24 wksNC
...
138	S29	Male	NC	12 wks	NaN	MaleNC	12 wksNC	Male12 wks	Male12 wksNC
139	S3	Male	HFD	12 wks	NaN	MaleHFD	12 wksHFD	Male12 wks	Male12 wksHFD
140	S7	Male	HFD	12 wks	NaN	MaleHFD	12 wksHFD	Male12 wks	Male12 wksHFD
141	S15	Male	HFD	12 wks	NaN	MaleHFD	12 wksHFD	Male12 wks	Male12 wksHFD
142	S23	Male	HFD	12 wks	NaN	MaleHFD	12 wksHFD	Male12 wks	Male12 wksHFD

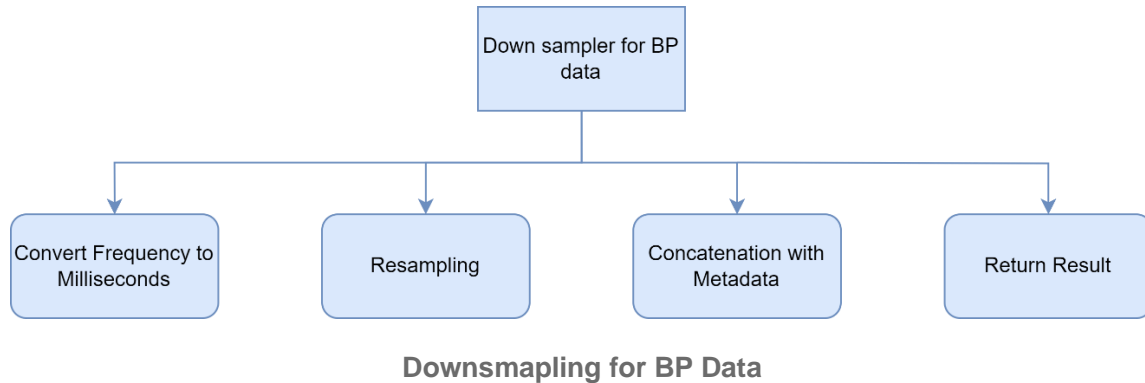
View of df_meta

these preprocessing steps ensure that the data is structured appropriately for subsequent analyses and model development. The resultant dataframes, **df_data** and **df_meta**, serve as the foundation

for further exploration and modeling endeavors.

Downsampling Function

The provided function, `downsample`, is designed to downsample blood pressure data to a specified sample rate in milliseconds, we will see architecture pipeline in the below figure, and we will talk about each step in order:



Function Explanation:

- **freq:** This parameter represents the desired sampling frequency (in Hz) for the downsampling process.
- **df:** This is an optional parameter which refers to the dataframe containing the blood pressure data. By default, it uses the **df_data** dataframe.
- **meta:** This is an optional parameter referring to the metadata dataframe. By default, it uses the **df_meta** dataframe.

Convert Frequency to Milliseconds:

- The function first calculates the time in milliseconds corresponding to the provided sampling frequency.

Resampling:

- The function then utilizes the **df.resample** method to perform the downsampling operation, taking the mean over the specified time intervals in milliseconds.

Concatenation with Metadata:

- The downsampled data is concatenated with the metadata dataframe, ensuring that the information about each record is preserved.

Return Result:

- The resulting dataframe, containing downsampled blood pressure data along with metadata, is returned as the output of the function.

This downsampling function effectively reduces the granularity of the blood pressure data, making it more manageable for further analysis while retaining important trends and patterns.

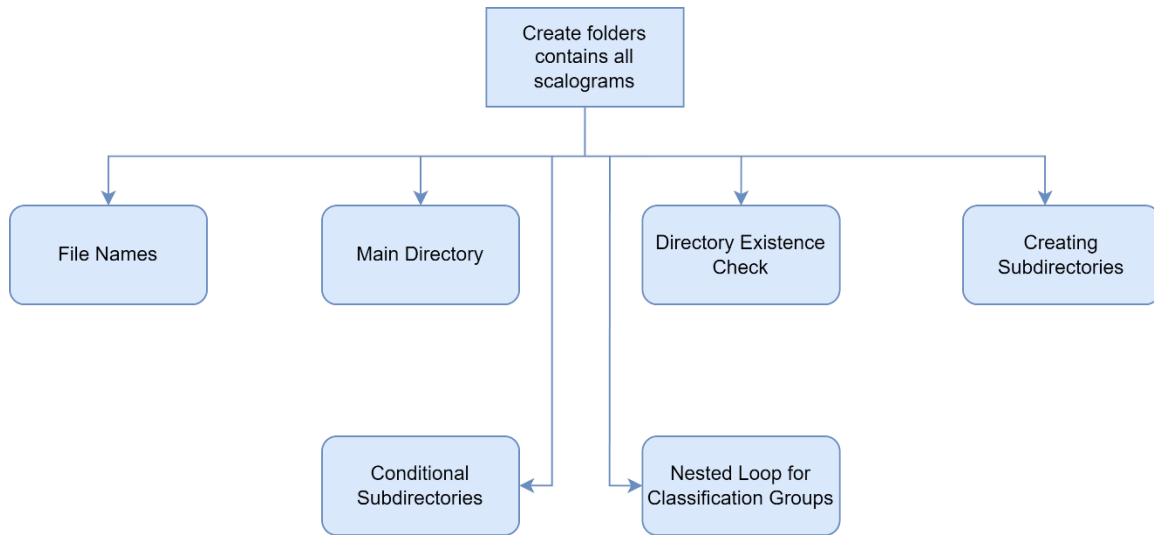
	ID	Sex	Diet	Age	Surgery	SexDiet	AgeDiet	SexAge	SexAgeDiet	0	...	42550	42551	42552	42553	42554	42
1	F1	Male	NC	24 wks	NaN	MaleNC	24 wksNC	Male24 wks	Male24 wksNC	129.194912	...	NaN	NaN	NaN	NaN	NaN	f
2	F9	Male	NC	24 wks	NaN	MaleNC	24 wksNC	Male24 wks	Male24 wksNC	129.111325	...	127.093257	135.334871	143.886388	149.709957	152.432071	153.
3	F10	Male	NC	24 wks	NaN	MaleNC	24 wksNC	Male24 wks	Male24 wksNC	130.890075	...	NaN	NaN	NaN	NaN	NaN	f
4	F17	Male	NC	24 wks	NaN	MaleNC	24 wksNC	Male24 wks	Male24 wksNC	175.478850	...	NaN	NaN	NaN	NaN	NaN	f
5	F18	Male	NC	24 wks	NaN	MaleNC	24 wksNC	Male24 wks	Male24 wksNC	155.957775	...	NaN	NaN	NaN	NaN	NaN	f
...
138	S29	Male	NC	12 wks	NaN	MaleNC	12 wksNC	Male12 wks	Male12 wksNC	169.412712	...	NaN	NaN	NaN	NaN	NaN	f
139	S3	Male	HFD	12 wks	NaN	MaleHFD	12 wksHFD	Male12 wks	Male12 wksHFD	149.879012	...	NaN	NaN	NaN	NaN	NaN	f
140	S7	Male	HFD	12 wks	NaN	MaleHFD	12 wksHFD	Male12 wks	Male12 wksHFD	149.161912	...	NaN	NaN	NaN	NaN	NaN	f
141	S15	Male	HFD	12 wks	NaN	MaleHFD	12 wksHFD	Male12 wks	Male12 wksHFD	157.957013	...	NaN	NaN	NaN	NaN	NaN	f
142	S23	Male	HFD	12 wks	NaN	MaleHFD	12 wksHFD	Male12 wks	Male12 wksHFD	160.224850	...	NaN	NaN	NaN	NaN	NaN	f

142 rows × 42569 columns

View of Down Sampling for BP Data

Create Directories Function

The provided code defines a function, **CrtDir**, which automates the creation of directories for organizing experimental data and scalogram images. we will see architecture pipeline in the below figure, and we will talk about each step-in order:



Create Folders contains all Scalograms

Function Explanation:

- **grp**: Refers to the classification criteria (e.g., Age).
- **col**: Represents the specific categories within the classification (e.g., 12 weeks, 24 weeks).
- **exp**: Denotes the experiment number (in frequency).

File Names:

- **files** is a list containing two strings: "scal" and "scal-cropped". These represent the types of scalogram images to be generated.

Main Directory:

- **main** is a string representing the main directory path where the subdirectories will be created. It includes information about the experiment number and classification.

Directory Existence Check:

- Checks if the main directory already exists. If so, it is removed and recreated.

Creating Subdirectories:

- The function iterates through the list of file names and creates subdirectories within the main directory.

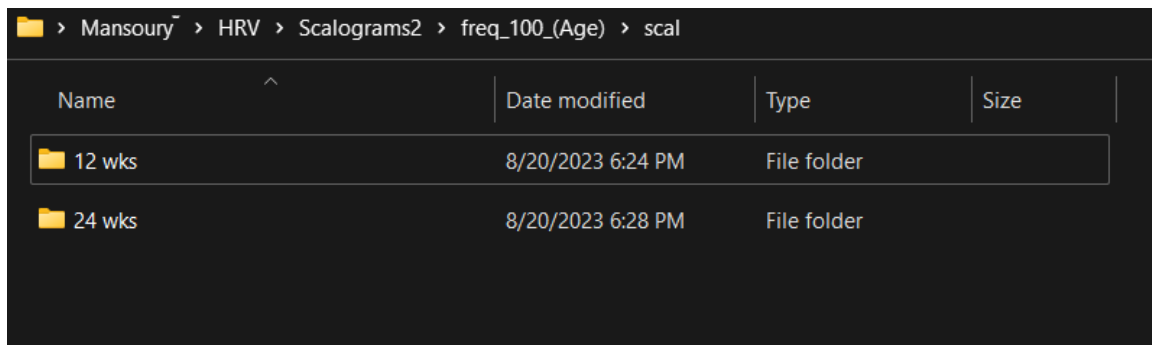
Conditional Subdirectories:

- For "scal-cropped", additional subdirectories for training, testing, validation, and all scalograms are created.

Nested Loop for Classification Groups:

- The function iterates through the classification groups and creates subdirectories within each type of scalogram directory.

This function automates the creation of a structured directory hierarchy for organizing experimental data and generated scalogram images. It ensures that data is properly categorized for subsequent processing and analysis.



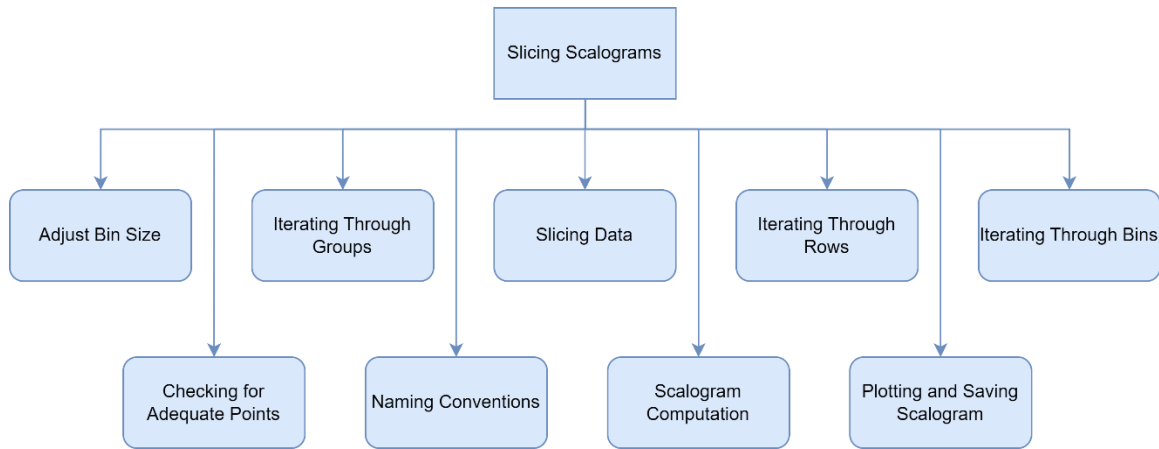
The screenshot shows a file explorer window with the path: Mansoury > HRV > Scalograms2 > freq_100_(Age) > scal. The window displays a table of files and folders.

Name	Date modified	Type	Size
12 wks	8/20/2023 6:24 PM	File folder	
24 wks	8/20/2023 6:28 PM	File folder	

View of Creation Scalograms Folders

Slicing Scalograms

The provided code defines a function, `slice_scal`, which performs the slicing of timeseries data into bins and computes scalograms for each bin. we will see architecture pipeline in the below figure, and we will talk about each step-in order:



Slicing Scalograms Architecture

Function Explanation:

- **grp**: The values to be classified.
- **col**: Name of the variable to be classified.
- **freq**: Sampling rate in the timeseries data.
- **df**: Dataframe containing timeseries data.
- **exp**: Experiment number. Used for naming folders.
- **w**: Bin size in seconds (default is 16.67 seconds).

Adjust Bin Size:

- **w** is adjusted to represent the number of points depending on the sampling rate.

Iterating Through Groups:

- The function iterates through the provided classification groups.

Slicing Data:

- The dataframe is sliced to include only the records corresponding to the current classification group.

Iterating Through Rows:

- The function iterates through the rows of the sliced dataframe.

Iterating Through Bins:

- It further iterates through the bins of data within each row.

Checking for Adequate Points:

- If there are not enough points to create a new scalogram with the current window, the remaining points are discarded.

Naming Conventions:

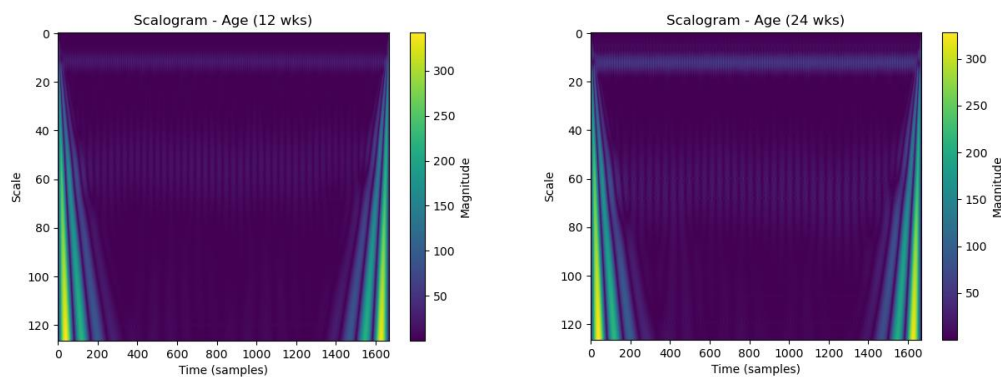
- The function uses a naming convention to numerically name the scalogram images.

Scalogram Computation:

- It computes the scalogram for the current bin.

Plotting and Saving Scalogram:

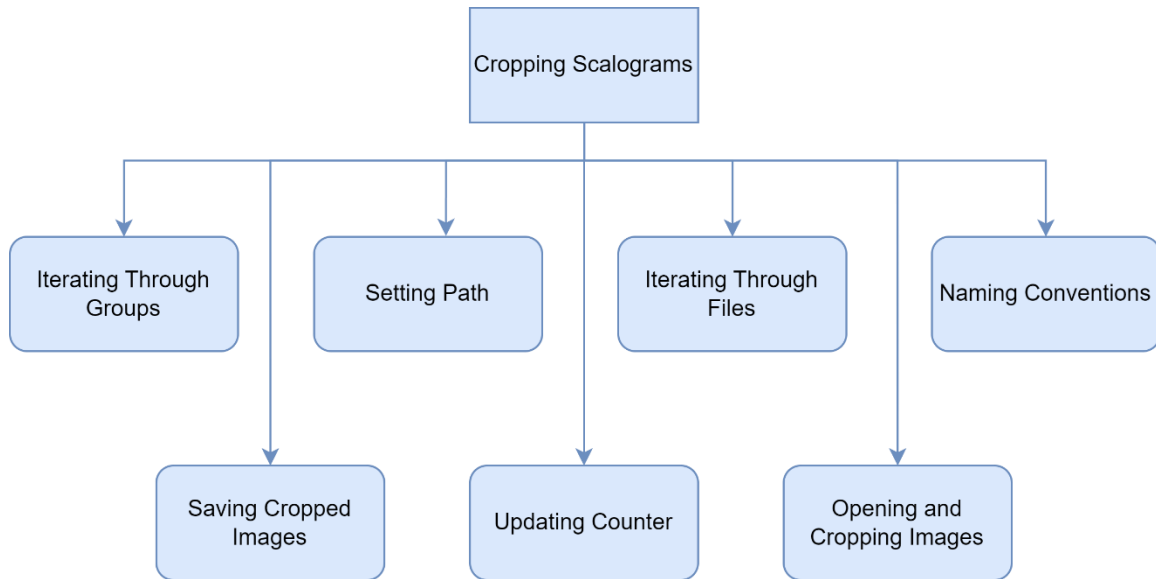
- The scalogram is plotted, and the image is saved.



View of Slicing Scalogram Age (12-24 wks) with Morlet wavelet from BP Data

Cropping Scalograms Function

The provided code defines a function, `crop_scal`, which crops scalogram images to remove all white spaces and axes, leaving only the plot. we will see architecture pipeline in the below figure, and we will talk about each step-in order:



Slicing Scalograms Architecture

Function Explanation:

- **grp**: The values being classified. This is used to retrieve images and save them to the appropriate directory.
- **exp**: Experiment number. Used for naming folders.
- **box**: This parameter defines the cropping parameters in the format **[left, upper, right, lower]**. It specifies the region of the image to keep.

Iterating Through Groups:

- The function iterates through the provided classification groups.

Setting Path:

- The path to the directory containing the original scalogram images is created.

Iterating Through Files:

- The function iterates through the files in the directory.

Naming Conventions:

- It uses a naming convention to numerically name the cropped scalogram images.

Opening and Cropping Images:

- The scalogram image is opened and cropped using the specified cropping parameters.

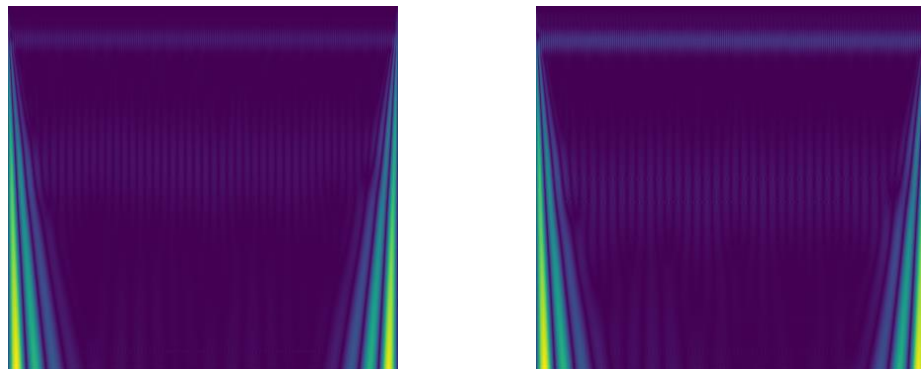
Saving Cropped Images:

- The cropped images are saved in both the respective group folder and the folder containing all scalograms.

Updating Counter:

- The counter **j** is incremented.

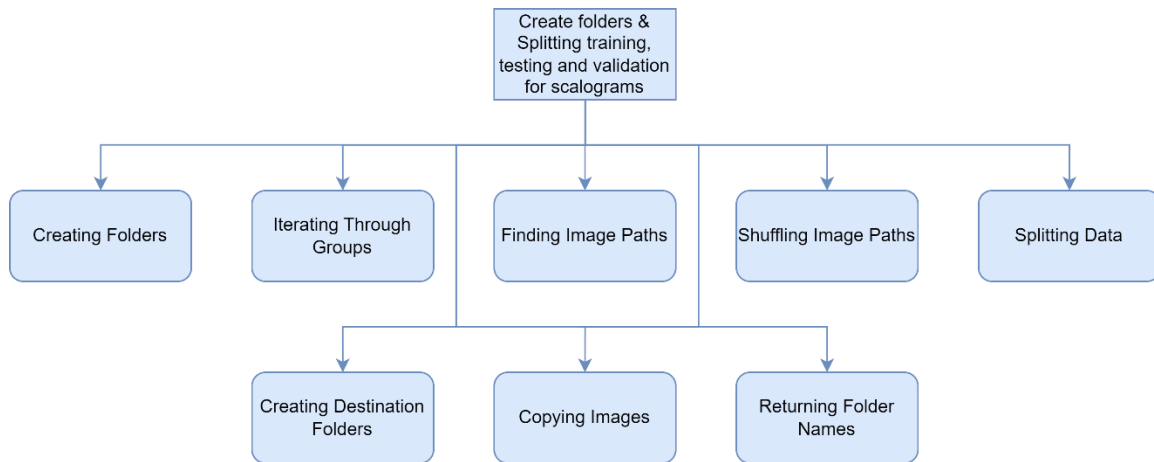
This function effectively crops the scalogram images, removing unnecessary whitespace and axes, leaving only the plot for further analysis.



View of Cropping Scalogram Age (12-24 wks) after Slicing Scalogram

Create Folders & Splitting Training, Testing and Validation for all Scalograms Function

The provided code defines a function, **splt_scal**, which creates folders for training, testing, and validation data and shuffles images into them using an 8:1:1 ratio. we will see architecture pipeline in the below figure, and we will talk about each step-in order:



Creating Folders for Training, Testing, and Validation Data Architecture

Function Explanation:

- **grp**: The values being classified.
- **exp**: Experiment number. Used for naming folders.
- **scalograms_dir**: List containing the directory paths of the scalograms.
- **folder_names**: List containing the paths for the train, test, and validation folders.

Creating Folders:

- The function first creates the train, test, and validation folders.

Iterating Through Groups:

- The function iterates through the provided classification groups.

Finding Image Paths:

- It finds all image paths within the specified scalogram directory.

Shuffling Image Paths:

- The image paths are randomly shuffled.

Splitting Data:

- The function then splits the images into training, validation, and testing sets using an 8:1:1 ratio.

Creating Destination Folders:

- It creates destination folders for train and test images.

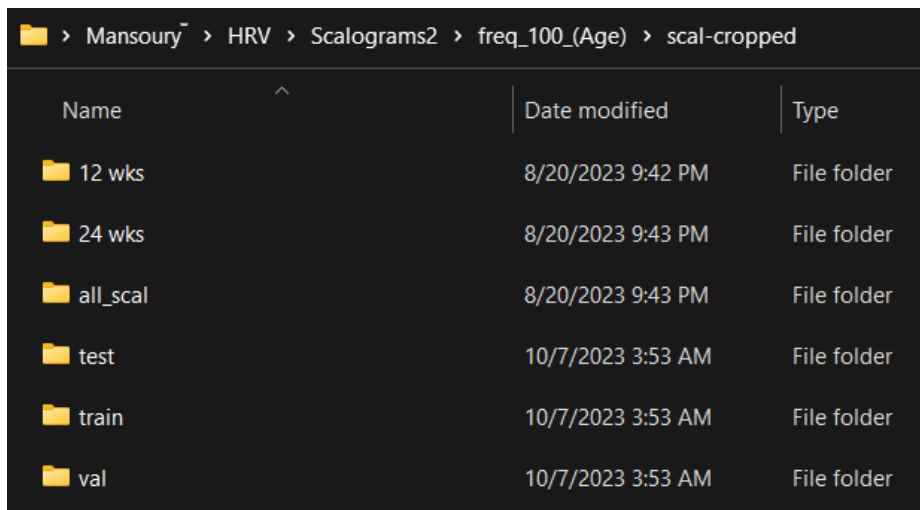
Copying Images:

- The images are then copied into their respective folders.

Returning Folder Names:

- The function returns the paths of the created folders.

This function sets up the directory structure for training, testing, and validation data, and shuffles images into these folders for further processing.

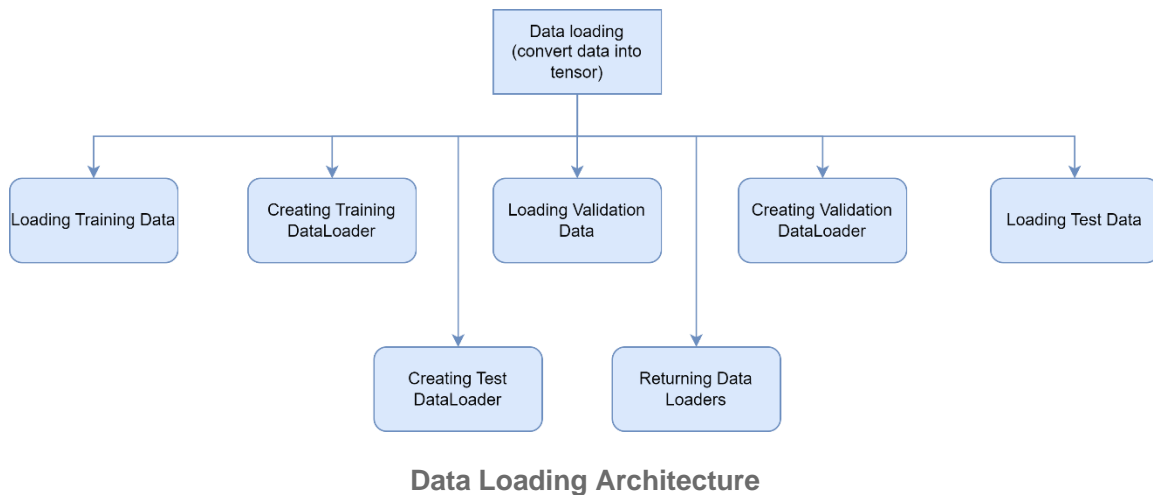


Name	Date modified	Type
12 wks	8/20/2023 9:42 PM	File folder
24 wks	8/20/2023 9:43 PM	File folder
all_scal	8/20/2023 9:43 PM	File folder
test	10/7/2023 3:53 AM	File folder
train	10/7/2023 3:53 AM	File folder
val	10/7/2023 3:53 AM	File folder

View of Creation Training, Testing, and Validation (8:1:1) ratio Folders

Data Loading

The provided code defines a function, `load_scal`, which loads scalogram images and sets up data loaders for training, validation, and testing sets. we will see architecture pipeline in the below figure, and we will talk about each step-in order:



Function Explanation:

- **folder_names**: A list containing paths for train, test, and validation folders.
- **batch_size**: The number of samples in each batch (default is 25).
- **shuffle**: A boolean indicating whether to shuffle the data before each epoch (default is True).

Loading Training Data:

- The function loads the training dataset using **`datasets.ImageFolder`**. It applies a transformation to convert the images to tensors.

Creating Training DataLoader:

- The function then creates a data loader for the training set using **`torch.utils.data.DataLoader`**.

Loading Validation Data:

- The validation dataset is loaded similarly to the training dataset.

Creating Validation DataLoader:

- A data loader for the validation set is created.

Loading Test Data:

- The test dataset is loaded in the same manner as the training and validation datasets.

Creating Test DataLoader:

- A data loader for the test set is created.

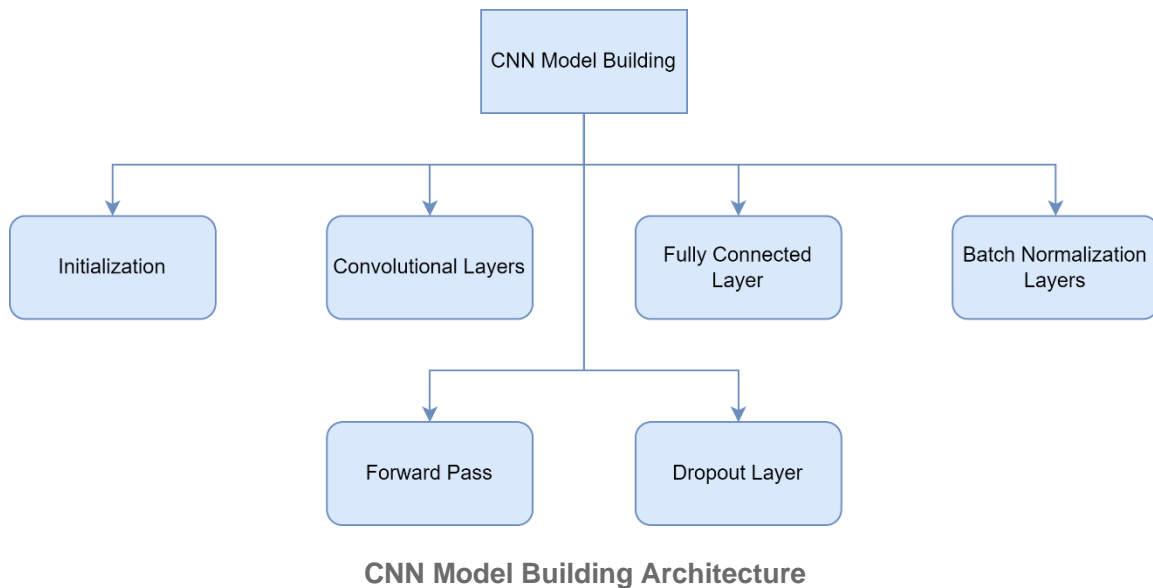
Returning Data Loaders:

- Finally, the function returns the data loaders for training, validation, and testing.

This function sets up the data loaders, making the datasets ready for training, validation, and testing.

CNN Model Building

The provided code defines a CNN model for the task at hand. we will see architecture pipeline in the below figure, and we will talk about each step-in order:



CNN Model Explanation:

1. Initialization:

- The class **b2b_net** inherits from **nn.Module** and initializes the layers of the neural network.

2. Convolutional Layers:

- There are five convolutional layers (**conv1** to **conv5**). Each layer performs a convolution operation on the input data to extract features.

3. Fully Connected Layer:

- There is one fully connected layer (**fc1**). It takes the output from the convolutional layers and produces the final output.
-

4. Batch Normalization Layers:

- Batch normalization is applied after each convolutional layer (**batchnorm1** to **batchnorm5**). It helps stabilize and speed up the training process.

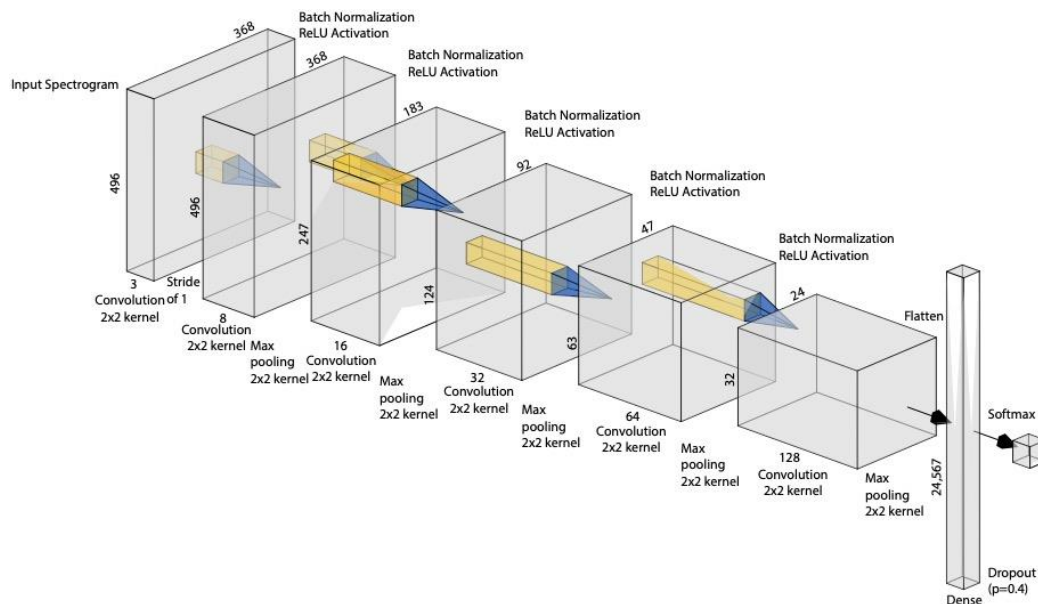
5. Dropout Layer:

- A dropout layer with a dropout rate of 0.4 is applied to the flattened output before the fully connected layer. This helps prevent overfitting.

6. Forward Pass:

- The **forward** method defines the forward pass of the network. It specifies how the input data is processed through the layers.
- Each convolutional layer is followed by a ReLU activation function and max pooling operation.
- The output from the last layer is flattened and passed through the fully connected layer. The softmax function is applied to get the final probabilities.
- The model's architecture consists of multiple convolutional layers, each followed by batch normalization, ReLU activation, and max pooling.

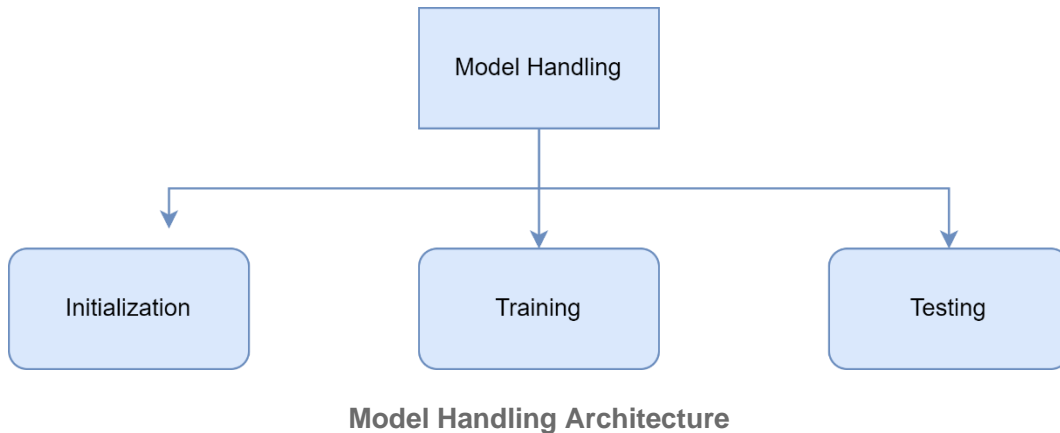
This model architecture is designed for processing scalogram images for classification tasks.



CNN Model Architecture

Model Handling

The provided code defines a class `ModelHandler` that is responsible for managing the training, validation, and testing of the CNN model. we will see architecture pipeline in the below figure, and we will talk about each step-in order:



Model Handling Explanation:

1. Initialization:

- The `ModelHandler` class is initialized with the following attributes:
 - `model`: The CNN model.
 - `device`: The device (CPU or GPU) on which the model is trained.
 - `epochs`: The number of training epochs.
 - `learning_rate`: The learning rate for the optimizer (default is 0.00005).

2. Training:

- The `train` method is responsible for training the model. It takes two data loaders (`train_loader` and `validation_loader`) as input.
- The method loops over the specified number of epochs and performs the following steps:
 - Sets the model to training mode.
 - Initializes variables for tracking loss, correct predictions, and total samples.

- Loops over the batches in the training data:
 - Zeroes out the gradients.
 - Transfers data and target to the specified device.
 - Performs the forward pass.
 - Computes and applies gradients using gradient scaling.
 - Updates the optimizer.
- Appends the training loss and accuracy for the epoch.
- After training for one epoch, the model is switched to evaluation mode and evaluated on the validation data.
- The method also appends validation loss and accuracy.

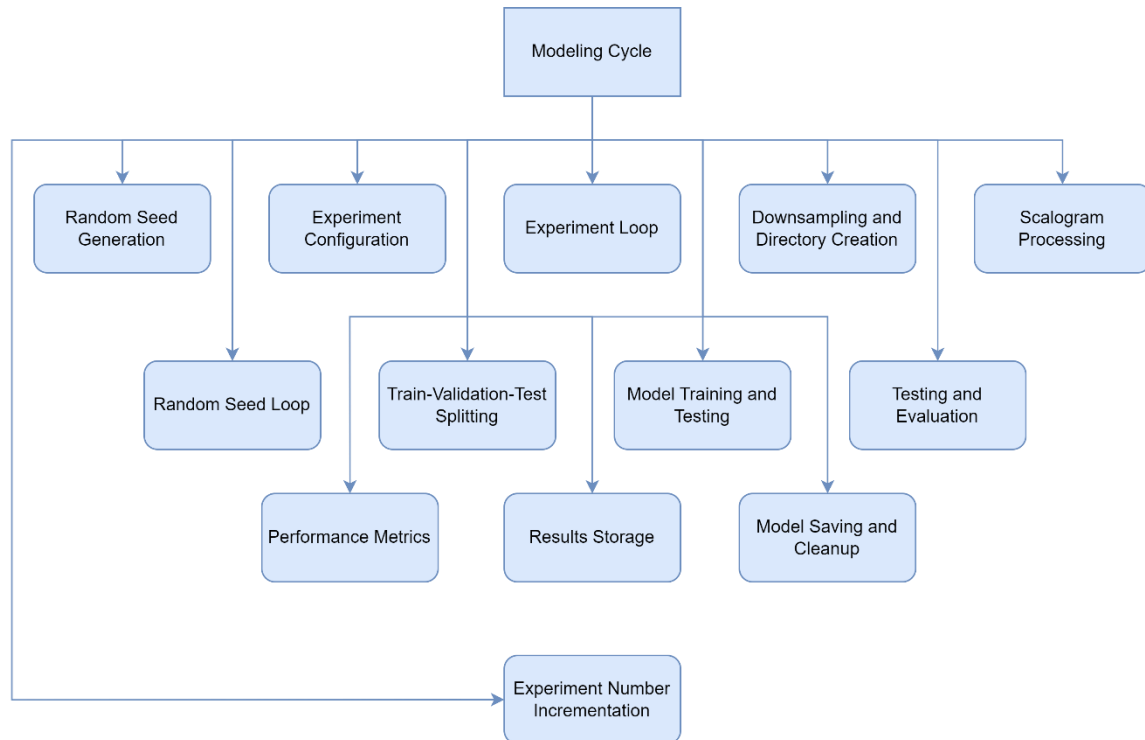
3. Testing:

- The test method is responsible for testing the trained model. It takes a test data loader as input.
- The method loops over the batches in the test data and performs the following steps:
 - Sets the model to evaluation mode.
 - Transfers data and target to the specified device.
 - Performs the forward pass.
 - Computes test accuracy.
 - Concatenates the model outputs and targets for later analysis.
- Returns the model outputs, target labels, and test accuracy.

This class encapsulates the training, validation, and testing process for the CNN model. It provides a clean and organized way to handle the entire pipeline.

Modeling Cycle

The provided code implements a modeling cycle for a deep learning project. It involves several steps, including data preprocessing, model training, evaluation, and result recording. We will see the architecture pipeline in the below figure, and we will talk about each step in order:



Modeling Cycle Architecture

Random Seed Generation:

- Initially, a list of 10 random seed values is generated. These seeds will be used to ensure reproducibility in your experiments.

Experiment Configuration:

- The variable **exp** is set to 1 to track the experiment number.
- A list of **freqs** is defined, representing different frequencies.
- An empty dictionary **multiclassresults** and an empty DataFrame with the same name are created to store the results of the experiments.

Experiment Loop:

- The code iterates through different experiments based on the combinations of classification groups defined in **grps**.

Downsampling and Directory Creation:

- For each experiment, the data is downsampled based on the chosen frequency.
- Directories are created to organize the scalogram images.

Scalogram Processing:

- The scalograms are sliced, cropped, and saved into appropriate directories.

Random Seed Loop:

- The code enters a loop that iterates through the list of random seeds (**randomlist**).
- For each seed value, it sets the random seed for Python, PyTorch, and GPU computations to ensure reproducibility.

Train-Validation-Test Splitting:

- The data is split into training, validation, and testing sets using the **splt_scal** function.
- The resulting folder names are stored in **folder_names**.

Model Training and Testing:

- A CNN model (**b2b_net**) is created and initialized.
- The **ModelHandler** class is used to train the model. Training and validation losses and accuracies are recorded.

Testing and Evaluation:

- The trained model is tested on the test data, and the test accuracy is computed.
- The confusion matrix is calculated to further evaluate model performance.

Performance Metrics:

- Various performance metrics such as specificity, sensitivity, positive predictive value (ppv), negative predictive value (npv), area under the ROC curve (AUROC), and average precision (AUPRC) are calculated based on the confusion matrix and model outputs.

	Frequency	Test_Accuracy	AUROC	AUPRC	Specificity	Sensitivity	PPV	NPV	Classification	RN
0	140	96.84	0.9611	0.9531	0.9649	0.9712	0.9712	0.9649	Age	200396
1	140	89.33	0.8871	0.8718	0.8333	0.9424	0.8733	0.9223	Age	544352
2	140	96.44	0.9513	0.9339	0.9386	0.9856	0.9514	0.9817	Age	524927
3	140	94.07	0.9483	0.9398	0.9035	0.9712	0.9247	0.9626	Age	635473
4	140	96.05	0.9672	0.9569	0.9386	0.9784	0.9510	0.9727	Age	609072
5	140	94.86	0.9516	0.9360	0.9298	0.9640	0.9437	0.9550	Age	680561
6	140	94.86	0.9368	0.9123	0.9386	0.9568	0.9500	0.9469	Age	227999
7	140	97.23	0.9626	0.9490	0.9561	0.9856	0.9648	0.9820	Age	156037
8	140	96.05	0.9523	0.9344	0.9298	0.9856	0.9448	0.9815	Age	535490
9	140	96.05	0.9530	0.9358	0.9211	0.9928	0.9388	0.9906	Age	208398
10	120	95.26	0.9484	0.9260	0.9298	0.9712	0.9441	0.9636	Age	200396
11	120	95.26	0.9462	0.9304	0.9123	0.9856	0.9320	0.9811	Age	544352
12	120	96.84	0.9589	0.9513	0.9298	1.0000	0.9456	1.0000	Age	524927
13	120	94.86	0.9571	0.9422	0.9123	0.9784	0.9315	0.9720	Age	635473
14	120	95.26	0.9678	0.9532	0.9035	0.9928	0.9262	0.9904	Age	609072
15	120	96.05	0.9493	0.9299	0.9474	0.9712	0.9574	0.9643	Age	680561
16	120	96.05	0.9394	0.9132	0.9211	0.9928	0.9388	0.9906	Age	227999

Example of view Performance Metrics

Results Storage:

- The results, including test accuracy and performance metrics, are stored in the **multiclassresults** DataFrame.

	140_Age_	140_Age_	140_Age_	140_Age_	140_Age_	140_Age_	140_Age_	140_Age_	140_Age_	140_Age_	120_Age_	120_Age_	120_Age_	120_Age_	120_Age_
0	96.84	89.33	96.44	94.07	96.05	94.86	94.86	97.23	96.05	96.05	95.26	95.26	96.84	94.86	95.26
1	tensor(0.9	tensor(0.8	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9
2	tensor(0.9	tensor(0.8	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9
3	tensor(0.9	tensor(0.8	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9
4	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(1.)	tensor(0.9
5	tensor(0.9	tensor(0.8	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9
6	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(0.9	tensor(1.)	tensor(0.9

View of multiclassresults

Model Saving and Cleanup:

- The trained model weights are saved to a file, and the results are saved to a CSV file.
- GPU memory is freed by deleting the model which is reset every time when finished model, but it is saved in the hard disk.

Experiment Number Incrementation:

- The experiment number (**exp**) is incremented for the next experiment.

This entire process represents a modeling cycle in which you systematically train, evaluate, and record the performance of your CNN model for different experimental conditions. The results are organized in the **multiclassresults** DataFrame, making it easy to analyze and compare the model's performance across various experiments.

```
100% ██████████ 8/8 [13:46:13<00:00, 6849.64s/it]

140
Train val test splitting is done!

Training: 100% ██████████ 30/30 [04:16<00:00, 8.16s/epoch, loss=0.321]

Training is done!
Testing is done!
0.9683794466403162
Exp 2 is finally done!
140
Train val test splitting is done!

Training: 100% ██████████ 30/30 [04:15<00:00, 8.08s/epoch, loss=0.543]

Training is done!
Testing is done!
0.8932806324110671
Exp 3 is finally done!
140
Train val test splitting is done!

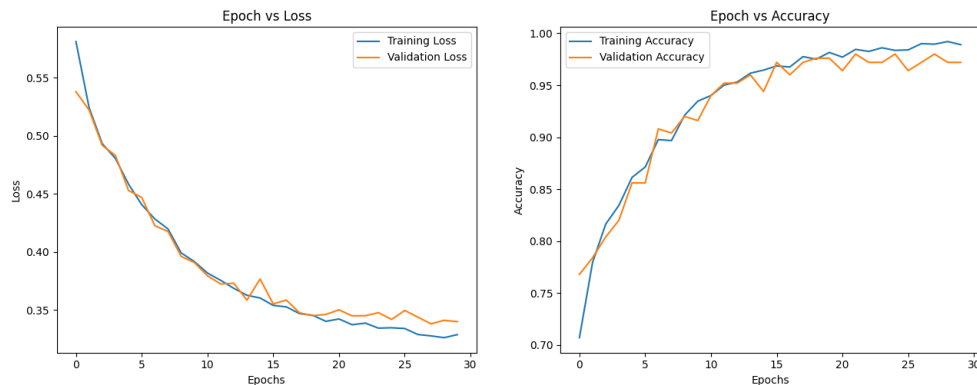
Training: 100% ██████████ 30/30 [04:16<00:00, 8.08s/epoch, loss=0.328]

Training is done!
Testing is done!
0.9644268774703557
Exp 4 is finally done!
140
Train val test splitting is done!

Training: 100% ██████████ 30/30 [04:14<00:00, 8.12s/epoch, loss=0.813]

Training is done!
Testing is done!
0.932806324110672
Exp 5 is finally done!
140
```

View the Results Modeling cycle as it trains, tests, and validation.



View of Epoch vs Loss and Epoch vs Accuracy

Generate Saliency Maps for Scalograms and find Frequency Peaks

Once we have trained a CNN scalogram classification model, we can use it to generate saliency maps and find frequency peaks. This can be useful for understanding which parts of the scalogram are most important for the model's decision-making process.

To generate a saliency map, we first feed the scalogram to the model and get its prediction. Then, we calculate the gradient of the prediction with respect to the input scalogram. This gradient represents the sensitivity of the prediction to changes in the input. Finally, we take the absolute value of the gradient and normalize it to produce the saliency map.

The saliency map is a heatmap where brighter pixels represent more important parts of the scalogram. We can use the saliency map to identify the frequency peaks that are most important for the model's classification decision.

Read MultiClassResult Dataset

In this section, you read the experiment results stored in a CSV file named "multiclassresults.csv" using the Pandas library. The loaded data is stored in a DataFrame called result. This DataFrame contains the performance metrics and results of your experiments, making it easy to analyze and visualize the outcomes.

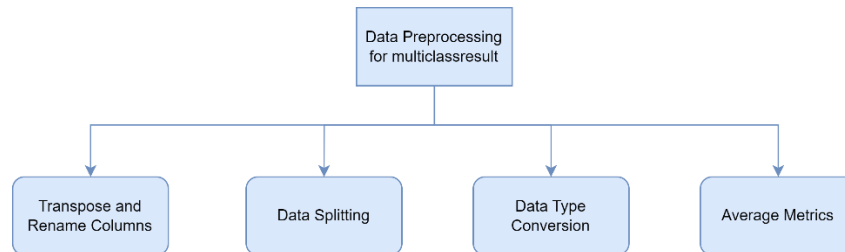
Unnamed: 0										
		140_Age_200396	140_Age_544352	140_Age_524927	140_Age_635473	140_Age_609072	140_Age_680561	140_Age_227999	140_Age_156037	140_Age_156037
0	0	96.84	89.33	96.44	94.07	96.05	94.86	94.86	97.23	97.23
1	1	tensor(0.9611)	tensor(0.8871)	tensor(0.9513)	tensor(0.9483)	tensor(0.9672)	tensor(0.9516)	tensor(0.9368)	tensor(0.9626)	tensor(0.9626)
2	2	tensor(0.9531)	tensor(0.8718)	tensor(0.9339)	tensor(0.9398)	tensor(0.9569)	tensor(0.9360)	tensor(0.9123)	tensor(0.9490)	tensor(0.9490)
3	3	tensor(0.9649)	tensor(0.8333)	tensor(0.9386)	tensor(0.9035)	tensor(0.9386)	tensor(0.9298)	tensor(0.9386)	tensor(0.9561)	tensor(0.9561)
4	4	tensor(0.9712)	tensor(0.9424)	tensor(0.9856)	tensor(0.9712)	tensor(0.9784)	tensor(0.9640)	tensor(0.9568)	tensor(0.9856)	tensor(0.9856)
5	5	tensor(0.9712)	tensor(0.8733)	tensor(0.9514)	tensor(0.9247)	tensor(0.9510)	tensor(0.9437)	tensor(0.9500)	tensor(0.9648)	tensor(0.9648)
6	6	tensor(0.9649)	tensor(0.9223)	tensor(0.9817)	tensor(0.9626)	tensor(0.9727)	tensor(0.9550)	tensor(0.9469)	tensor(0.9820)	tensor(0.9820)

7 rows × 561 columns

View of MultiClassResult Dataset after Modeling Cycle

Data Preprocessing

in this section, you perform data preprocessing on the experiment results to organize and clean the data for further analysis. we will see architecture pipeline in the below figure, and we will talk about each step-in order:



Data Preprocessing for multiclassresult Architecture

Transpose and Rename Columns:

- You transpose the DataFrame result to have the frequencies as rows and metrics as columns. Then, you rename the columns to make them more descriptive.

Data Splitting:

- You split the "Frequency" column into "Frequency," "Classification," and "RN" (Random Number) columns to separate these pieces of information.

Data Type Conversion:

- You convert the metrics columns from strings (with parentheses) to float values for further analysis.

Average Metrics:

- You calculate the mean and confidence intervals (CI) for each metric, including specificity, sensitivity, PPV, and NPV, for each combination of frequency and classification.

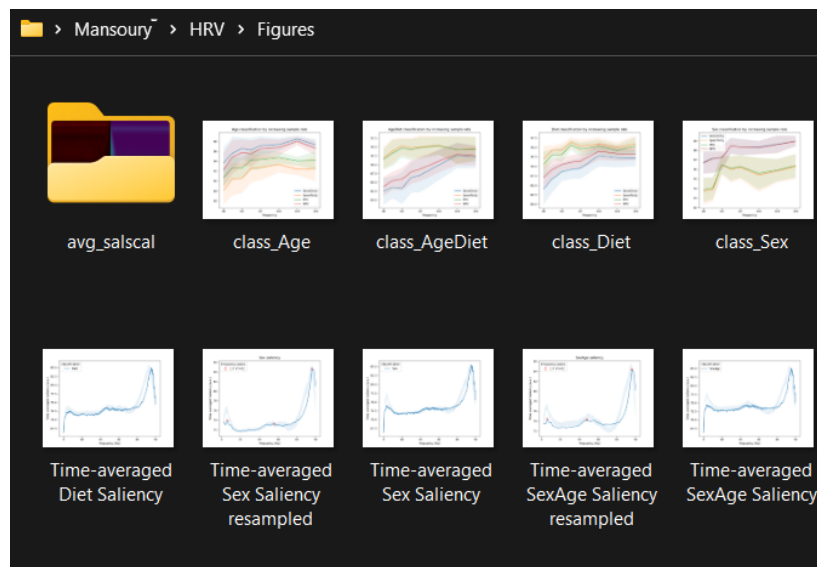
The resulting DataFrame 'resavg' provides a structured and summarized view of your experiment results, which can be easily used for analysis and visualization. It contains the mean values and CIs for each metric, organized by frequency and classification.

	Frequency	Test_Accuracy	AUROC	AUPRC	Specificity	Sensitivity	PPV	NPV	Classification	RN
0	140	96.84	0.9611	0.9531	0.9649	0.9712	0.9712	0.9649	Age	200396
1	140	89.33	0.8871	0.8718	0.8333	0.9424	0.8733	0.9223	Age	544352
2	140	96.44	0.9513	0.9339	0.9386	0.9856	0.9514	0.9817	Age	524927
3	140	94.07	0.9483	0.9398	0.9035	0.9712	0.9247	0.9626	Age	635473
4	140	96.05	0.9672	0.9569	0.9386	0.9784	0.9510	0.9727	Age	609072
5	140	94.86	0.9516	0.9360	0.9298	0.9640	0.9437	0.9550	Age	680561
6	140	94.86	0.9368	0.9123	0.9386	0.9568	0.9500	0.9469	Age	227999
7	140	97.23	0.9626	0.9490	0.9561	0.9856	0.9648	0.9820	Age	156037
8	140	96.05	0.9523	0.9344	0.9298	0.9856	0.9448	0.9815	Age	535490
9	140	96.05	0.9530	0.9358	0.9211	0.9928	0.9388	0.9906	Age	208398
10	120	95.26	0.9484	0.9260	0.9298	0.9712	0.9441	0.9636	Age	200396
11	120	95.26	0.9462	0.9304	0.9123	0.9856	0.9320	0.9811	Age	544352
12	120	96.84	0.9589	0.9513	0.9298	1.0000	0.9456	1.0000	Age	524927
13	120	94.86	0.9571	0.9422	0.9123	0.9784	0.9315	0.9720	Age	635473
14	120	95.26	0.9678	0.9532	0.9035	0.9928	0.9262	0.9904	Age	609072
15	120	96.05	0.9493	0.9299	0.9474	0.9712	0.9574	0.9643	Age	680561
16	120	96.05	0.9394	0.9132	0.9211	0.9928	0.9388	0.9906	Age	227999

View of multiclassresult after preprocessing

Create Figures Directory

Create a directory named "Figures" unless it already exists. This directory will be used to store any visualizations or figures generated during your analysis. Having a dedicated directory for figures helps keep your project organized and makes it easy to locate and reference visual outputs.



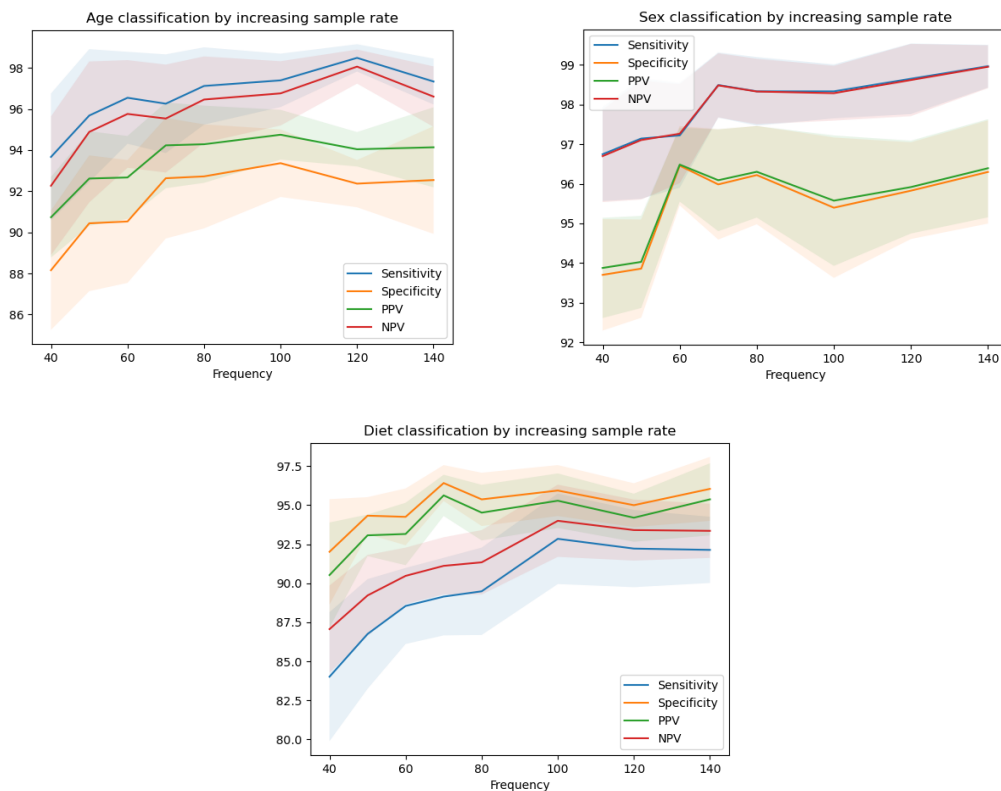
View of Figures Folder

Plot Metrics Function

In this section, you define a function `plot_line(metric)` that takes a metric name as input and generates a line plot. The function plots the mean value of the specified metric across different frequencies, with shaded regions indicating confidence intervals.

For each classification type ('Age', 'Sex', 'Diet', 'SexDiet', 'AgeDiet', 'SexAge', 'SexAgeDiet'), the function calls `plot_line` to generate line plots for the metrics Sensitivity, Specificity, PPV, and NPV. The resulting plots are saved in the "Figures" directory with appropriate labels.

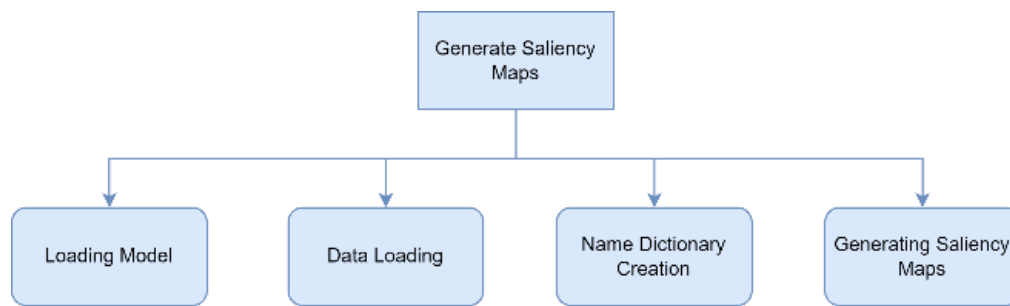
This section enables you to visually analyze how the classification metrics change with different frequencies and classification types.



View of Classification by increasing Sample rate Figure

Generate Saliency Maps

This section focuses on generating saliency maps for each classification at different frequencies. The code iterates over a list of frequencies (**freq_list**) and for each frequency, it further iterates over the different classification types (**col**) and random seeds (**rn**).. we will see architecture pipeline in the below figure, and we will talk about each step-in order:



Generating Saliency Maps Architecture

Loading Model:

- The code first loads the trained model for the specific frequency, classification, and random seed combination.

Data Loading:

- It then loads the test data for generating the saliency maps. In this case, batch size is set to 1 to process images one at a time.

Name Dictionary Creation:

- A name dictionary (**name_dic**) is created using the **create_name_dic** function, which is likely defined elsewhere in your code. This dictionary is used for naming the generated saliency maps.

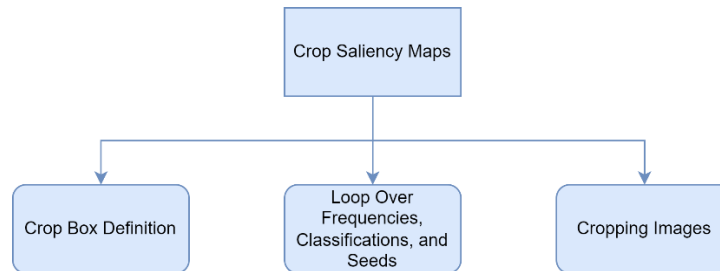
Generating Saliency Maps:

- The **handler.visualize** function is called to generate the saliency maps. The **visualize** function likely takes the test loader, name dictionary, frequency, classification type, and random seed as arguments to generate and save the saliency maps.

This section allows you to generate and save saliency maps for each classification type at different frequencies, providing valuable insights into the features the model is focusing on for making predictions.

Crop Saliency Maps

This section is dedicated to cropping the produced saliency maps to remove any unnecessary white spaces. It iterates over a list of frequencies (freq_list) and for each frequency, it further iterates over the different classification types (col) and random seeds (rn). we will see architecture pipeline in the below figure, and we will talk about each step-in order:



Crop Saliency Maps Architecture

Crop Box Definition:

- The variable **box** represents the cropping box, which specifies the dimensions to crop from the images. In this example, the box is defined as **[left, upper, right, lower]** for cropping a 6.4 x 4.8 inch image with a 4:3 aspect ratio.

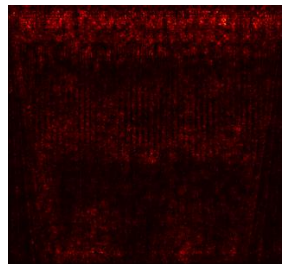
Loop Over Frequencies, Classifications, and Seeds:

- The code iterates over each frequency, classification, and random seed combination. For each combination, it processes the saliency maps located in the corresponding folder.

Cropping Images:

- It opens each image in the folder and crops it using the **crop** method with the specified **box**. The cropped image is then saved back to the same location, effectively removing any unwanted white spaces.

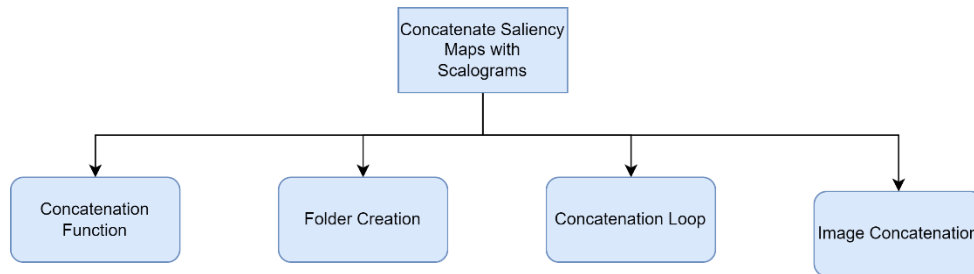
By cropping the saliency maps, you ensure that the generated visualizations are more focused on the regions of interest, making them more interpretable and informative.



View of Saliency Map after Cropped

Concatenate Saliency Maps with Scalograms

This section is responsible for concatenating saliency maps with their corresponding scalograms to create informative visualizations. Here's an overview and we will see architecture pipeline in the below figure, and we will talk about each step-in order:



Concatenate Saliency Maps with Scalograms Architecture

Concatenation Function:

- The function `get_concat_h(im1, im2)` is defined. This function concatenates two images horizontally (**im1** on the left and **im2** on the right) and returns the resulting image.

Folder Creation:

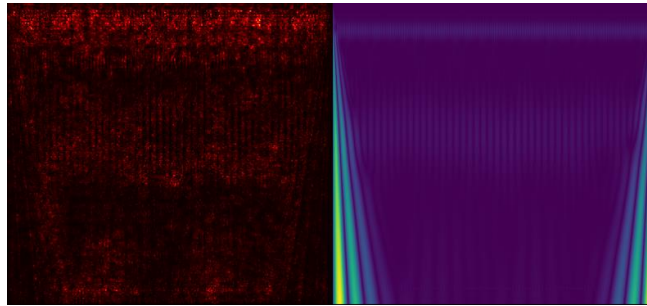
- There's a commented section of code that appears to be intended for creating folders to contain pairs of saliency maps and scalograms. It creates a structure where there are multiple folders (presumably one for each classification or frequency) that will hold the paired images.

Concatenation Loop:

- The code iterates over the specified frequencies (**freq_list**), classifications (**col**), and random seeds (**rn**).
- For each combination, it prepares the paths for the saliency maps (**sal**) and scalograms (**scal**).

Image Concatenation:

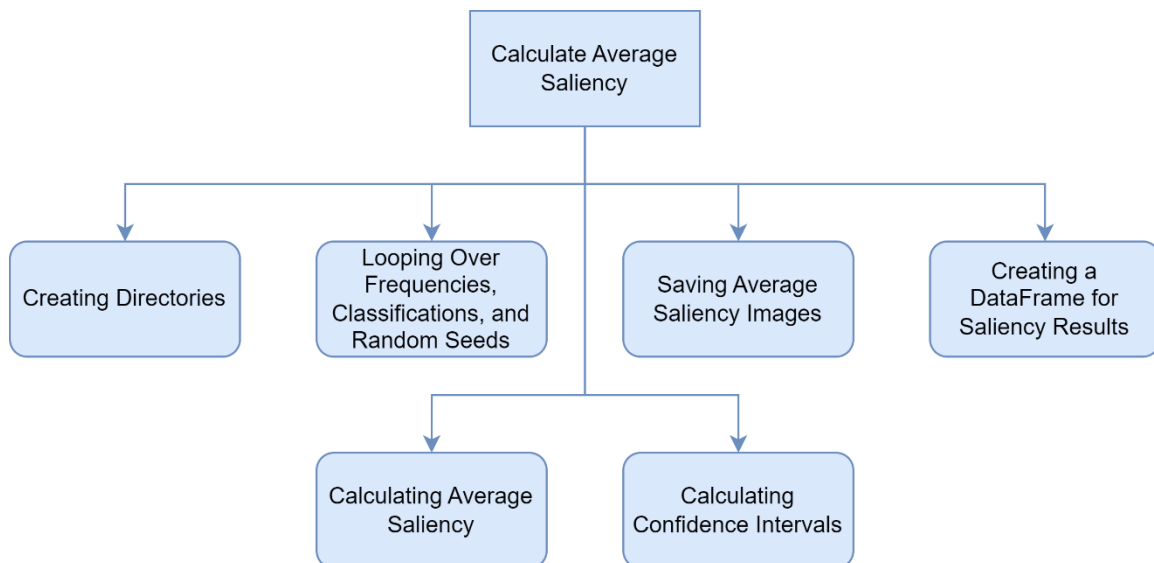
- Inside the loop, it opens each image in the **sal** and **scal** directories, concatenates them using the **get_concat_h** function, and saves the resulting image in the **salscal** directory.



View of Concatenate Saliency Maps with Scalograms

Calculate Average Saliency

This section of code calculates the average saliency for each classification and model, and also computes the confidence intervals (CI) for the average saliency values. Here's an overview and we will see architecture pipeline in the below figure, and we will talk about each step-in order:



Calculate Average Saliency Architecture

Creating Directories:

- You're creating a new folder named `avg_salscal` to save the average saliency-scalogram pairs for each model. If the folder already exists, this code will not create a new one.

Looping Over Frequencies, Classifications, and Random Seeds:

- The code iterates over different frequencies, classifications, and random seeds.
- For each combination, it calculates the average of the saliency maps for the corresponding saliency-scalogram pairs. This is done by summing up all the arrays and then dividing by the total number of images.

Saving Average Saliency Images:

- The resulting average array is converted back to an image format and saved in the `avg_salscal` folder.

Creating a DataFrame for Saliency Results:

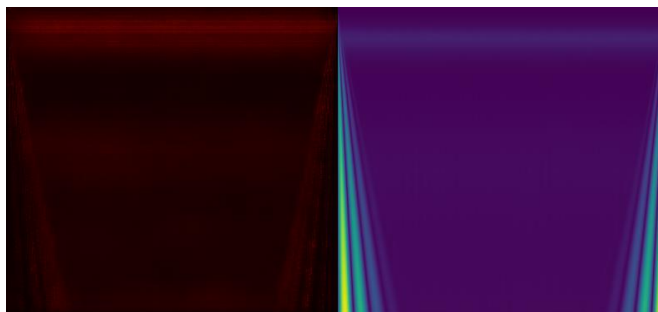
- A DataFrame named `results_sal` is created to store the average saliency values.

Calculating Average Saliency:

- For each classification, the code calculates the average saliency value for each pixel position.

Calculating Confidence Intervals:

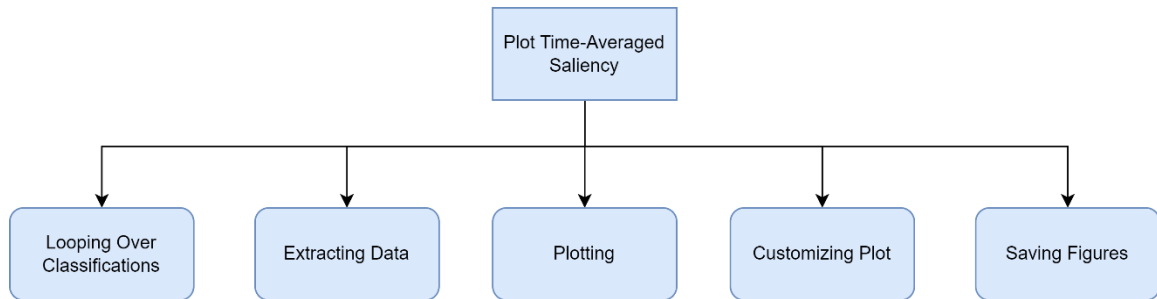
- The confidence intervals (CI-upper and CI-lower) are also calculated for each pixel position.



View of Average Scalograms and Saliency Maps

Plot Time-Averaged Saliency

In this section, you are plotting the time-averaged saliency for different classifications. Here's an overview and we will see architecture pipeline in the below figure, and we will talk about each step-in order:



Plot Time-Averaged Saliency Architecture

Looping Over Classifications:

- You're iterating over different classifications ('Age', 'Sex', 'Diet', 'SexDiet', 'AgeDiet', 'SexAge', 'SexAgeDiet').

Extracting Data:

- For each classification, you extract the relevant data from the results_sal DataFrame. Specifically, you're interested in the mean, CI-lower, and CI-upper values.

Plotting:

- You create a plot where the x-axis represents frequency (ranging from 0 to 50 Hz) and the y-axis represents time-averaged saliency (in arbitrary units).
- You plot the mean time-averaged saliency, and fill the area between the CI-lower and CI-upper values to represent the confidence interval.

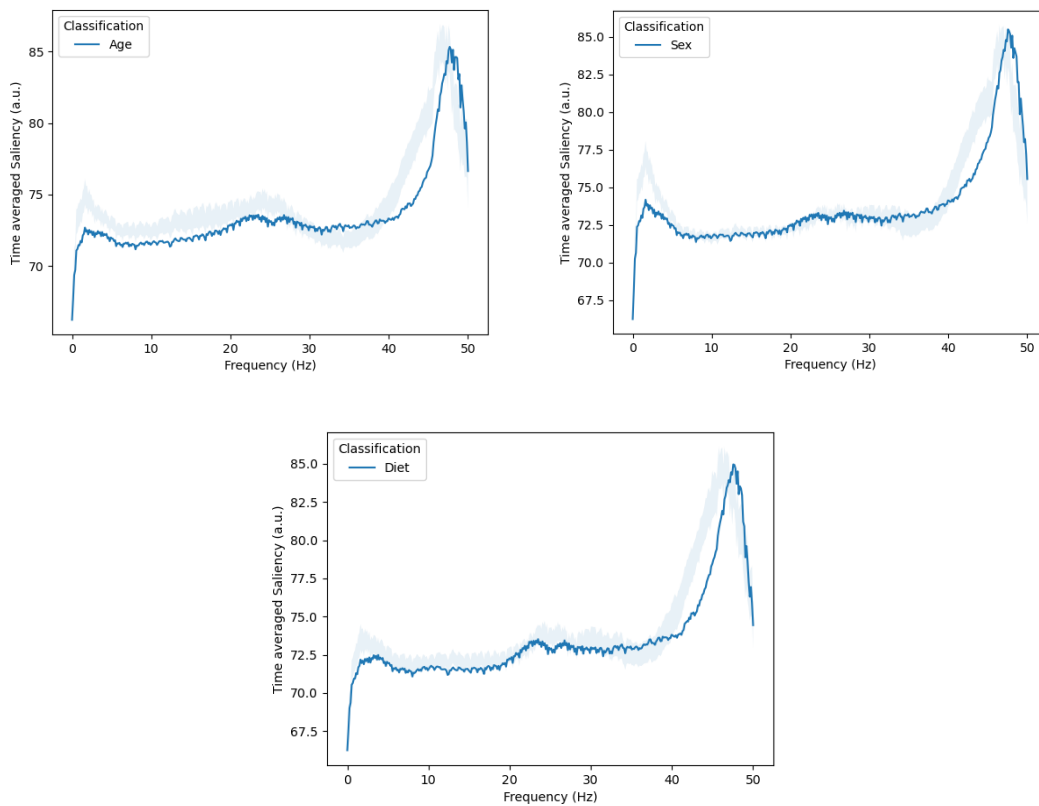
Customizing Plot:

- You add labels for the x-axis and y-axis.
- You include a legend with the title "Classification" to differentiate between the different classifications.

Saving Figures:

- The generated plot is saved as a PNG file in the Figures directory with an appropriate filename.

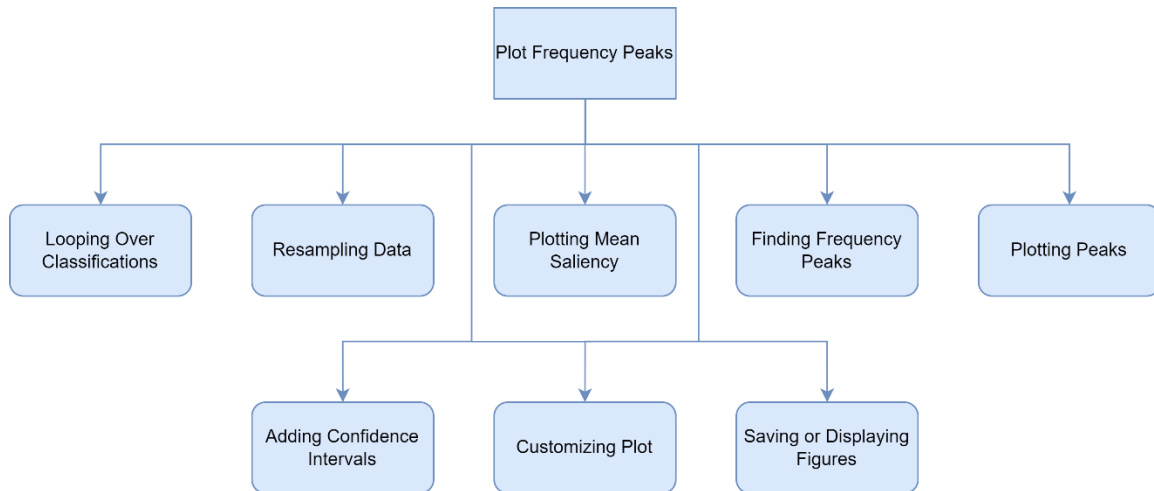
Overall, this section provides visual representations of the time-averaged saliency for different classifications, along with confidence intervals for each classification. This enables a clear comparison of the saliency patterns across different frequency ranges.



View of Time-averaged Age, Sex and Diet Saliency Figure

Plot Frequency Peaks

In this section, you're visualizing the frequency peaks for different classifications. Here's an overview and we will see architecture pipeline in the below figure, and we will talk about each step-in order:



Plot Frequency Peaks Architecture

Looping Over Classifications:

- You're iterating over different classifications ('Age', 'Sex', 'Diet', 'SexDiet', 'AgeDiet', 'SexAge', 'SexAgeDiet').

Resampling Data:

- For each classification, you're resampling the mean, CI-lower, and CI-upper values to have 50 data points. This helps in visualizing the frequency peaks more clearly.

Plotting Mean Saliency:

- You plot the resampled mean time-averaged saliency against the frequency. This provides an overview of the saliency pattern.

Finding Frequency Peaks:

- You identify the frequency peaks using the `sp.signal.find_peaks` function. These peaks represent significant points in the saliency pattern.

Plotting Peaks:

- You mark the identified frequency peaks on the plot with red dots.

Adding Confidence Intervals:

- The area between the resampled CI-lower and CI-upper values is filled to represent the confidence interval.

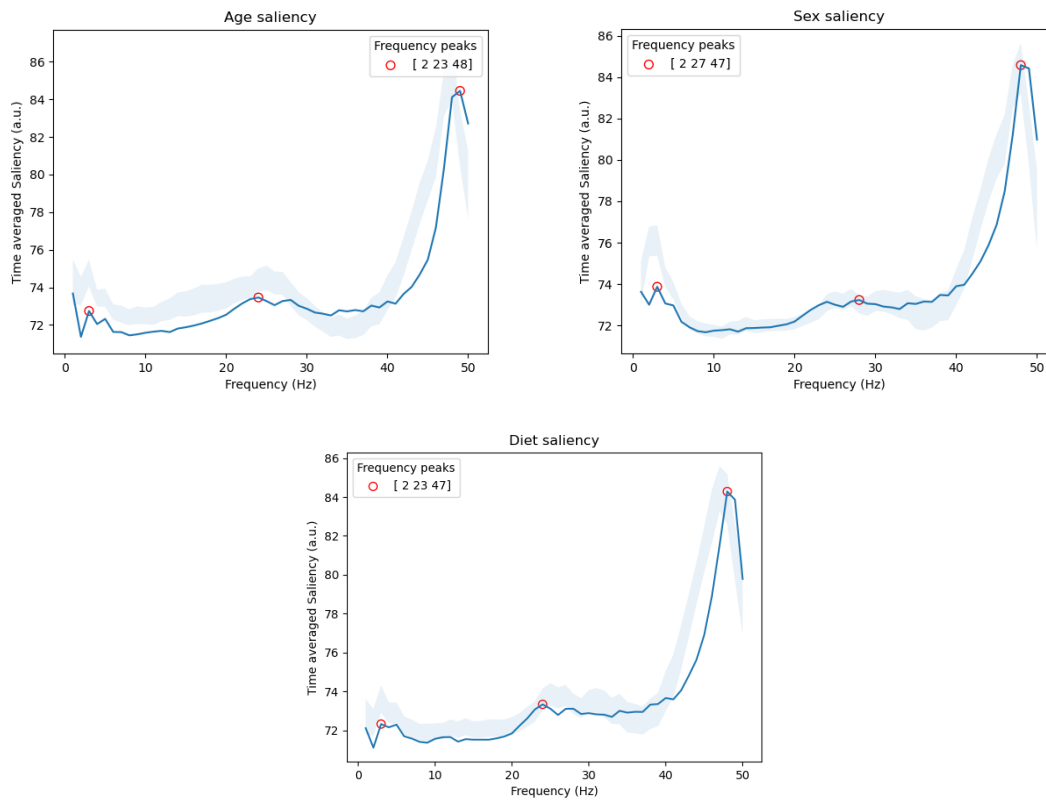
Customizing Plot:

- You add labels for the x-axis and y-axis.
- You include a legend with the title "Frequency peaks" to indicate the marked peaks.

Saving or Displaying Figures:

- The generated plot can be saved as a PNG file (uncomment the plt.savefig line) or displayed in the notebook.

Overall, this section provides visualizations of frequency peaks in the time-averaged saliency, which helps in identifying important frequency components for different classifications.



View of Time-averaged Age, Sex and Diet Saliency resampled Figure

Conclusion

Employing a convolutional neural network overcomes the shortcomings associated with conventional cardiovascular risk prediction measures such as average blood pressure and metrics related to blood pressure variability, including dispersion, chaos, detrended fluctuation analysis, and power spectral density analysis.

A beat-to-beat arterial pressure scalogram encapsulates latent and less readily identified information across its temporal, frequency, and power spectra. These elements prove invaluable in evaluating the state of cardioautonomic control.

The characterization of prominent features within arterial pressure scalograms can serve as a powerful tool for the early detection of irregular blood pressure fluctuations. This information is instrumental in informing timely diagnoses and guiding appropriate therapeutic protocols.