Deep Learning-Enabled Cerebrovascular Reactivity Processing Software

by

Yashesh Dasari

A thesis

presented to the University of Waterloo

in fulfillment of the

thesis requirement for the degree of

Master of Applied Science

in

Mechanical and Mechatronics Engineering

Waterloo, Ontario, Canada, 2022

Abstract

Magnetic Resonance Imaging (MRI) is a non-invasive medical imaging technique that is used to generate high-resolution images of the brain. Blood oxygenation level dependent (BOLD) imaging is a functional MRI technique that maps the differences in cerebral blood flow (CBF). Cerebrovascular Reactivity (CVR) is a provocative test used with BOLD-MRI studies where a vasoactive stimulus is applied and the corresponding changes in the CBF differences are analyzed. This test is analogous to a cardiac stress test where the patient exercises and the change in heart blood flow are observed.

CVR is measured as the ratio of the change in BOLD signals to the change in the vasoactive stimulus. The vasoactive stimulus used in this work is the arterial partial pressure of carbon dioxide. The gas control for applying the stimulus is accomplished using a computer-controlled device called RespirAct RA-MR^{TM}. CVR studies can highlight abnormalities in the brain vasculature and therefore, indicate underlying pathological conditions. Studies have demonstrated a close correlation between an irregular CVR distribution and cerebrovascular diseases like Alzheimer's disease, steno-occlusive disease (SOD), stroke, and traumatic brain injury.

SOD is the most common cause of ischemic stroke all over the world. Patients with symptomatic SOD are at a high risk of recurrent ischemic stroke. Therefore, information about the severity and spatial location of irregular CVR at the brain tissue level can help guide clinical treatment. The current generation of CVR analyses and assessments are conducted manually by a team of doctors and radiologists, using their subject expertise and years of experience.

In this work, a next-generation CVR processing and visualization software application is presented that furthers the research capabilities of CVR analyses. The proposed software is capable of processing raw BOLD-MRI files and generating CVR maps. It is developed using open-source tools and deployed as a stand-alone application that runs on a virtual machine. Additionally, convolutional neural networks (CNNs) are designed to facilitate the screening of SOD patients by classifying the CVR maps into healthy and unhealthy patients. Some popular pre-trained networks, like, EfficientNetB0, InceptionV3, ResNet50, and VGG16 are fine-tuned to accomplish the target classification.

For training the CNNs, the original dataset consisted of 68 healthy and 163 unhealthy images. To increase the number of trainable samples and address the data imbalance, image augmentation techniques were applied. An empirical evaluation-based design strategy is implemented for optimizing the network architecture. The performance of different CNN architectures along with fine-tuning of hyperparameters was analyzed and the most optimal network is presented. Results from transfer learning are compared as well. Experiments indicated that a customized CNN with two convolution layers with 32 filters and a hidden fully connected layer with 32 neurons produced the best results. This model used batch normalizations and dropout regularization after the convolution layers and the fully connected layer. It achieves high training/validation/prediction results consistent with expert clinical readings.

The proposed software integrates the complete CVR research workflow, including file management, processing MRI files, data visualization, and producing CVR maps, and serves as a clinical decision support system, automating the workflow by 75% and providing a one-stop software solution. It is suggested that this software is used as a research tool to produce CVR maps and make data-driven decisions on SOD screenings, along with validation by experts.

Keywords: artificial intelligence- AI, cerebrovascular reactivity, convolutional neural network, deep learning, magnetic resonance imaging, software development, steno-occlusive disease

Acknowledgements

First, I would like to express my sincere gratitude to my supervisor, Professor Mir Behrad Khamesee, for providing me with the opportunity to work under his invaluable supervision and for the patient guidance and support he has provided throughout my master's research.

I would also like to sincerely thank Dr. Joseph A. Fisher, Dr. James Duffin, Dr. David J. Mikulis, Dr. Olivia Sobczyk, Mr. Julien Poublanc, Ms. Ece Su Sayin, and Mr. Harrison Levine for providing their guidance and insights on the research topics. Special thanks to Mr. Kevin Morwood for his continuous guidance and mentorship in software-related domains during this research.

I consider myself incredibly lucky to have worked with such a knowledgeable and encouraging supervisor and research group, who have always cared about my work and well-being (especially during the COVID period).

I am truly grateful to Professor Stewart McLachlin and Professor William Melek for being my thesis readers and for taking out the time to provide valuable feedback and suggestions.

I also wish to acknowledge the financial support provided by the Natural Sciences and Engineering Research Council of Canada (NSERC), Thornhill Research Inc., and the University of Waterloo.

And finally, I would like to thank my family and friends in Canada and back home in India, for always believing in me.

Dedication

I would like to dedicate my thesis to four beloved people who play an incredibly important role in my life and are my pillars of strength: my mom (Dr. Shridevi Dasari), my dad (Mr. Shridhar Babu Dasari), and my sisters (Ms. Poorvi Dasari, and Ms. Renuka Dasari). My ambitious plans of pursuing graduate studies would not have been possible without their unwavering love and support.

I am truly grateful and honored to call you my family. Thank you for everything.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AI	Artificial Intelligence
BOLD	Blood Oxygenation Level Dependent
CBF	Cerebral Blood Flow
CVR	Cerebrovascular Reactivity
CNN	Convolutional Neural Network
MRI	Magnetic Resonance Imaging
SOD	Steno-Occlusive Disease

Chapter 1

Introduction

“The chief function of the body is to carry the brain around.”

-Thomas Alva Edison

1.1 Project Overview

Magnetic Resonance Imaging (MRI) is a non-invasive medical imaging technique that is used to generate high-resolution images of the body’s organs and tissues. It uses magnetic fields and computer-generated radio waves to create these images [1]. Cerebrovascular Reactivity (CVR) is a provocative test used with MRI studies to analyze the changes in cerebral blood flow (CBF) when an external stimulus is applied, to highlight abnormalities in regions of the brain [2]. CVR closely relates to the health of the brain vasculature and is a key parameter used to identify underlying cerebrovascular diseases like stroke, small vessel disease, and dementia [3].

Steno-occlusive disease (SOD) is a brain disorder that involves an arterial occlusion (blockage) or stenosis (narrowing) in regions of the brain [4]. A compensatory mechanism for SOD is the development of new vessels, called collateral blood vessels, to bypass the obstruction. CVR analysis can be used to assess the net hemodynamic consequences of SOD and developed collateral blood flow [5].

The current generation of CVR processing and assessment is conducted manually by a team of experts. The master’s research discussed in this thesis developed software tools to further the processing, visualization, and automation standards for CVR studies. The research involved conceiving and engineering an in-house CVR processing and visualization software application to process raw BOLD-MRI files and produce the corresponding CVR maps. Using the CVR maps produced, convolutional neural networks (CNNs) are implemented to facilitate the screening of SOD patients by discriminating between healthy control subjects and unhealthy patients.

1.2 Motivation

Cerebrovascular diseases include several different types of conditions that affect cerebral blood flow. Of the many forms, stroke is the most common type. According to the World Stroke Organization (WSO), 1 in 4 people over the age of 25 will have a stroke in their lifetime [6]. As per the statistical reports from Centers for Disease Control and Prevention, 6.5 million people suffered from a form of stroke in the United States alone in 2015 [7]. According to the World Health Organization (WHO) website, around 15 million people worldwide suffer from stroke annually. From this stroke-affected population, 5 million patients die, and another 5 million patients suffer from a permanent disability [8]. SOD is the most common cause of ischemic stroke all over the world [9].

As discussed above, CVR can play a vital role in identification and diagnosis of underlying cerebrovascular diseases. Therefore, information about the severity and spatial location of irregular CVR at the brain tissue level can help guide clinical treatment.

The current generation of CVR processing is conducted using a variety of software products. Some of the intermediate steps of the CVR research workflow also required the use of paid software, for example, MATLAB [10], to produce the output files. The workflow consisted of executing individual shell scripts in a particular sequence to accomplish the required CVR analysis. Additionally, researchers at participating institutions used efficient, but self-made, programs to realize the workflow. Although these practices produce the CVR maps, there can be differences in the produced output which makes it difficult to standardize the findings. Additionally, depending on customized shell scripts requires subject expertise and therefore limits the broad utilization of this workflow. Finally, CVR assessment is conducted manually by a team of doctors and radiologists using their subject expertise.

Therefore, there was an opportunity to build a software solution that could integrate and standardize the research workflow across all participating researchers, transitioning from licensed software to open-source tools, and allowing the distribution of a standalone application package to the collaborators. Extending the CVR software capabilities, this research implements deep learning models to support clinicians and researchers in making data-driven decisions and facilitate the screening of patients with SOD.

1.3 Objectives

The primary objective of this thesis is to explain the design and development an in-house CVR processing software that integrates and standardized the CVR research workflow and propose a clinical decision support system to screen SOD patients using deep learning. The developed software is proposed as an advancement over the current processing tools as it transitions from the use of licensed tools to open-source tools. The software package, being a standalone tool, is intended to allow easy testing, troubleshooting, distribution, and maintenance. Finally, the easy-to-operate software functionalities along with the streamlined workflow would allow users with minimal AFNI expertise to process MRI data and produce CVR maps.

1.4 Contribution to the Field

The main contributions of this research work are as follows.

1) The in-house CVR Processing software developed in this work serves as a next-generation processing and visualization tool enabling researchers access to a standardized and streamlined CVR research workflow.

2) The CVR Processing software is developed using open-source tools only, addressing the challenge of using licensed software that was used in the current workflow.

3) The CVR Processing software runs on a virtual machine which is operating system agnostic, overcoming the barrier of operating system constraints.

4) To the best of the author’s knowledge, the research of implementing CNN and transfer learning for classifying CVR maps obtained from prospective end-tidal targeting of carbon dioxide (CO2), which is a specialized type of external stimulus in MRI studies, is novel.

5) The CNN proposed in this work successfully discriminates between healthy and unhealthy patients with high accuracy on both the training and validation datasets. The network’s generalization capabilities are consistent with clinical readings as it predicts the correct labels for unseen samples in the prediction dataset with high accuracy.

1.5 Thesis Outline

Chapter 1 introduces the research discussed in this thesis. It begins with a project overview, followed by the motivations behind pursuing this research, the objectives of this thesis, and the contributions to the field.

Chapter 2 explains the relevant background and the main findings from the literature review. The discussions include the concepts of CVR, the products used to administer vasoactive stimulus, underlying concepts of deep learning and convolutional neural networks, and the application of deep learning in medical imaging. The current state of the application of deep learning in CVR is highlighted to evaluate the novelty of this work.

Chapter 3 analyzes the problem space, detailing the software process flow diagrams and the software requirements that were identified. Functional, non-functional, and user interface requirements are discussed, and the software architecture is highlighted as well.

Chapter 4 details the underlying concepts and calculations to realize the CVR research workflow, and the software development methodology used to build the CVR Processing software application. The different components of the software are discussed, including the Graphical User Interface (GUI) design, tools for file management, MRI processing software, and Matplotlib's plotting modules.

Chapter 5 describes the design and optimization of CNNs to classify CVR maps into healthy and unhealthy groups. Data pre-processing is described including the data analysis of the original datasets, and image augmentation techniques, followed by the experimental setup, and the design methodology. The methodology entails the design of customized CNNs and their optimization by hyperparameter finetuning. Transfer learning is implemented using a few popular pre-trained networks. The model performances are compared, and the most optimal network is proposed.

Chapter 6 discusses the results and main findings of the research. This includes discussions on the capabilities and limitations of the CVR Processing software, analysis of the performance of different CNN architectures investigated, and the most optimal network's performance and limitations. A CVR research and assessment workflow is proposed, and the future scope of this research is discussed.

Finally, Chapter 7 draws conclusions based on the research outcomes.

Appendix-A discusses the workflow of the CVR software developed in this work along with the GUI components at the different stages of the processing and visualization. Appendix-B presents and explains the programs and scripts used in this work.

Chapter 2

Literature Review and Background

2.1 Cerebrovascular Reactivity

Functional MRI (fMRI) is a type of MRI that produces images of blood flow to areas of the brain [2]. Blood oxygenation level dependent (BOLD) imaging is an fMRI technique that maps the differences in cerebral blood flow (CBF) when an external stimulus is administered [11]. Cerebrovascular Reactivity (CVR) is a provocative test used with BOLD-MRI studies, that measures the ability of the brain to adjust the diameter of the vessels to compensate the cerebral blood flow when an external vasoactive stimulus is applied [3]. Vasoactive stimulus is an agent that affects the blood vessels diameters, and in turn manipulating the blood flow. The agent can either constrict the blood vessels (narrowing it) or dilate them (widening it) [12].

In BOLD-MRI studies, the change in CBF is monitored by studying the corresponding changes in the BOLD signals [13]. CVR is calculated as the ratio of the change in BOLD signal(s) to the change in vasoactive stimulus applied. The abnormalities in the brain vasculature can then be identified by measuring the difference in CVR between different regions of the brain. There are different types of stimuli that can be used to conduct CVR studies, for example, transient reduction in mean arterial blood pressure [14], injection of chemicals (like acetazolamide) [15], and an increase in blood partial pressure of CO2 [16]. In this work, the partial pressure of CO2 is used as the stimulus. A detailed explanation is presented in section 2.1.1.

An irregular CVR is associated with abnormalities in the brain vasculature, indicating underlying pathological conditions [17]. Several studies have demonstrated the close correlation between an impaired CVR and cerebrovascular diseases such as Steno-Occlusive Disease [18] [19], Alzheimer's disease [20] [21] [22], and small vessel disease [23]. The CVR values are calculated voxel-by-voxel across the 3-dimensional volume of the brain and mapped over the corresponding anatomical scans using a well-defined color scale. These color-coded maps can then be plotted as 2-dimensional slice-by-slice images for clinical evaluation. CVR values can be color-coded and mapped onto the corresponding MRI anatomical map. Figure 2.1 shows CVR maps measured in a healthy and an unhealthy subject. The underlying CVR calculations are discussed in section 2.1.3.

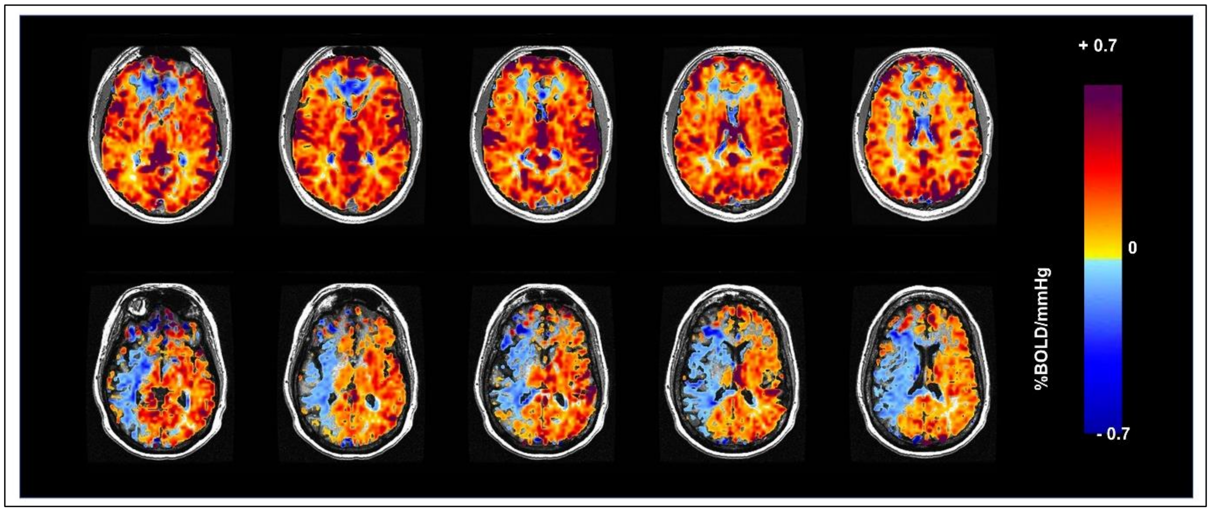


Figure 2.1: CVR maps of two subjects color coded as per the color scale shown and placed over the corresponding anatomical scans. Maps in the top are measured in a healthy subject. Note the increase in CVR (which is an expected response with respect to the change in the vasoactive stimulus for a healthy subject) indicated by positive values for CVR and colored red. Maps in the bottom show an abnormal CVR response measured in a patient with SOD. Note the reduced CVR in the left hemisphere of the brain, indicated by negative values for CVR and colored blue

Steno-occlusive disease (SOD) is one of the most common causes of stroke around the world, and arterial blockage is responsible for about 85% of stroke cases [9]. Additionally, another research highlighted that patients with SOD were at high risks to develop recurrent ischemic stroke. The study found a close correlation between impaired BOLD-CVR and the risk for recurrent ischemic stroke [24]. And information about the spatial distribution of the irregular CVR and its intensity can help highlight the presence of SOD. Therefore, accurate assessment of CVR can play a vital role in guiding clinical treatment.

2.1.1 Vasoactive stimuli

There is a wide variety of vasoactive stimuli that have been used by researchers to monitor CBF. When there are changes in brain activity or perfusion pressure, metabolic coupling mechanisms regulate the blood flow. These coupling mechanisms also react to changes in carbon dioxide and hypoxia. Researchers and investigators have used this phenomenon to externally challenge the system using a vasoactive stimulus and study the response [2].

Carbon Dioxide (CO2) as the vasoactive stimulus is a non-invasive technique that can be effectively controlled and terminated. CBF has a close correlation with the changes in the partial pressure of arterial CO2 in BOLD MRI studies. CO2 administration is considered safe. Despite these advantages, one of the main challenges in controlling CO2 was the requirement of a ventilatory system or voluntary rapid breathing. Also, when compared to chemicals whose dosage can be monitored, controlling CO2 stimulus is difficult [2].

To address the issues discussed, the sponsors developed an automated device to reliably administer changes in the end-tidal partial pressure of CO2 ($P_{a,CO2}$) [2]. The device is called the RespirAct RA-MR and is shown in Figure 2.2. The details about its working mechanism and the components are presented in the next section.

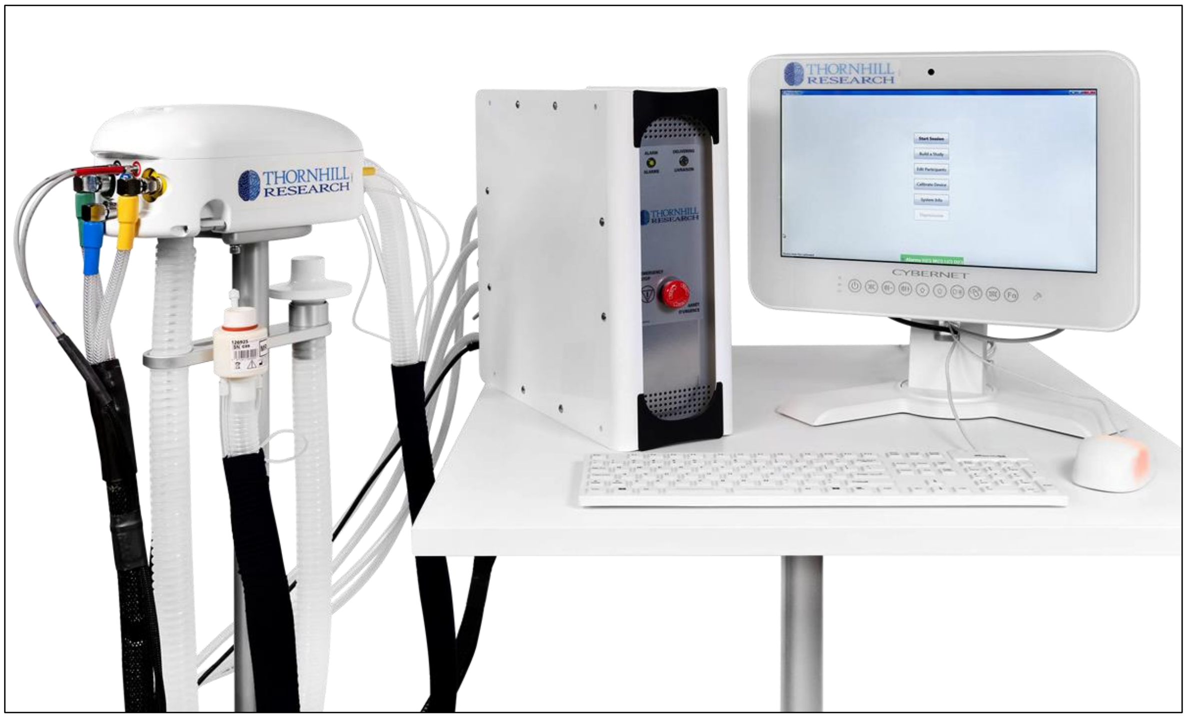


Figure 2.2: The RespirAct RA-MR device used for controlling gas delivery in breathing studies [25]

2.1.1.1 Prospective end-tidal targeting

Prospective end-tidal targeting is a gas flow method that is used to control the end-tidal partial pressures of carbon dioxide (CO_2) and oxygen (O_2) gas concentration in spontaneous breath studies [26]. The sponsors developed an innovative computer-controlled gas blender, called the RespirAct RA-MR device, that can be used to control the gas delivery in breathing studies. The RA-MR device can be used with MRI, Transcranial Doppler (TCD), and other imaging modalities. In an MRI study, the RA-MR user interface and the control room unit are placed in the control room which communicates with the MRI unit placed next to the MRI bed [27].

The main purpose of the RespirAct RA-MR device is to control the end-tidal partial pressure of CO_2 and O_2 in spontaneously breathing subjects. The device has four main components as listed below [28]:

1) RA-MR Control Room Unit
2) RA-MR User Interface

3) RA-MR MRI Unit

4) RA-MR Subject Breathing Circuit and RA-MR Consumable Mask and Circuit.

A system block diagram of the RespirAct RA-MR device used in an MRI study is shown in Figure 2.3. The device saves the test files required for conducting CVR research workflow and obtaining the CVR maps. The *EndTidal* file contains the data collected over the course of the entire test. Specifically, it contains the measured end-tidal CO2 values for the breath during the test. This is used for CVR calculations as explained in section 2.1.3. The *Events* file saves the time-stamped events information over the course of the test. This information is used to define the area of interest for conducting a specific study, by specifying the MRI sequence start and end values.

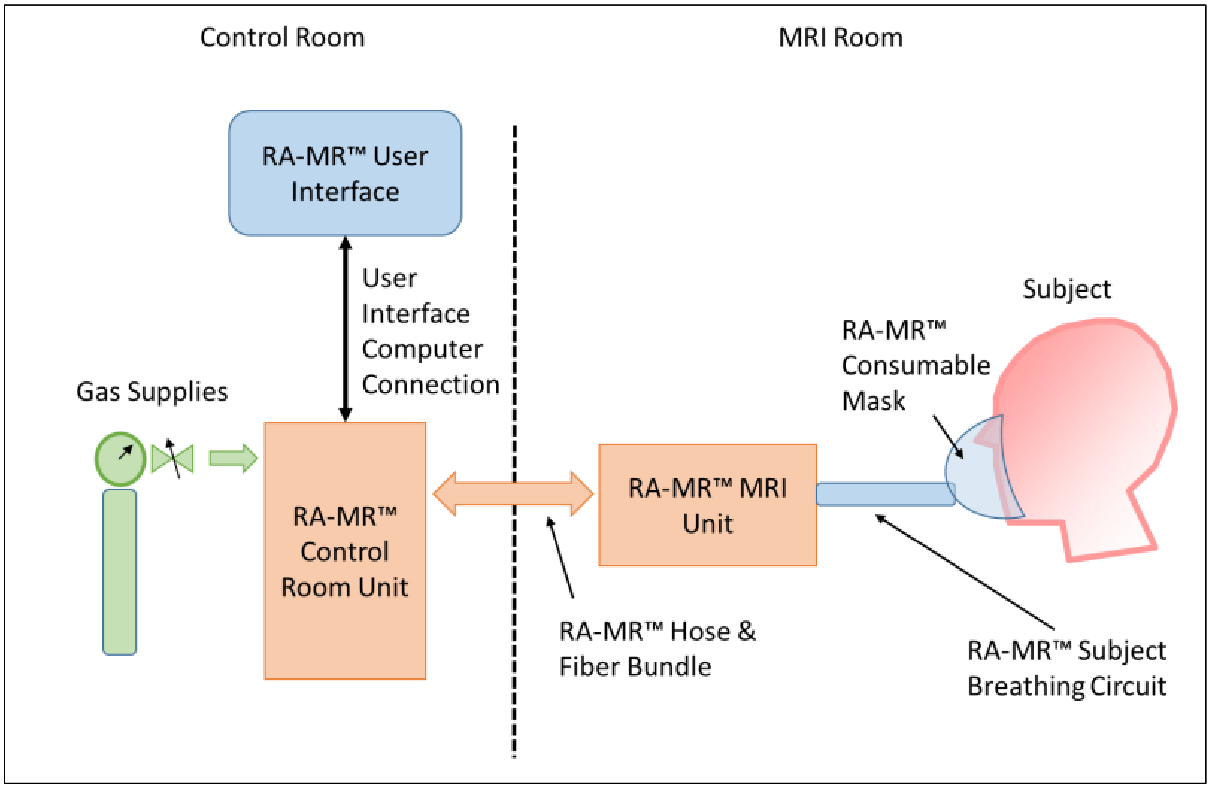


Figure 2.3: A system block diagram of the RespirAct RA-MR device for an MRI study. The Control Room is shown on the left and the MRI Room on the right [28]

2.1.2 DICOM and AFNI

Digital Imaging and Communications in Medicine (DICOM) is the international standard for medical imaging and related information. DICOM is recognized by the International Organization for

Standardization as the ISO 12052 standard and is universally adopted for medical imaging [29]. DICOM interface defines the form and flow of biomedical images and related information between computers using electronic messages [30].

Analysis of Functional NeuroImages (AFNI) is a software suite used for the analysis and visualization of anatomical and functional MRI data. Its development began in 1994 and was mainly written by RW Cox. AFNI is freely available for research purposes. AFNI runs under Unix+X11+Motif systems, however, is not readily available for Windows operating systems [31].

AFNI can process MRI files produced using different types of scanners and in different formats. The fundamental data storage produced by AFNI is a 3-dimensional (3-D) dataset that contains multiple 3D arrays of voxel values. A voxel represents a value in a 3-D space, similar to what a pixel does in a 2-D space. In this work, MRI formats such as DICOM (.DCM), MR image files (.MR), and NIfTI-1 formatted datasets (.nii) were tested.

2.1.3 CVR calculations

For the protocol used in this thesis, CVR is calculated as the ratio of change in BOLD signals ($\Delta BOLD$) to the increase in blood partial pressure of CO_2 ($\Delta PCO2$), as shown in equation (2.1) [32].

$$CVR = \frac{\Delta\, BOLD}{\Delta\, PCO2} \tag{2.1}$$

In this thesis, the workflow to obtain CVR values consisted of the following steps:

1. Anatomical and BOLD DICOM files from the BOLD-MRI study are processed using AFNI, producing intermediate AFNI format HEAD and BRIK files. The output file consists of coordinates of the brain voxel (which is like pixels but in a 3-dimensional space) and the corresponding BOLD values for the time duration of the MRI study. An example is shown in Table 2.1. A detailed sample output file is shown in Figure 2.4. The column headers x, y, and z values represent the coordinates of the voxel in the standard space. Similarly, i, j, and k represent the voxel coordinates in the MNI (Montreal Neurological Institute) space. T_i (**i** from 1 to n; n being the last time step of the MRI sequence) represent the time series and the corresponding BOLD values are registered at each time step. The right arrows show that the table extends based on the time duration of the study, and the down arrows show that more rows contain BOLD values for the different voxels across the brain.

2. An average BOLD sequence is calculated by averaging the BOLD responses at a given time step across all voxels (averaging the columns corresponding to the time steps). This average BOLD value dataset is then plotted on a new time series based on an MRI parameter called the response time [TR].

3. The PCO2 data in the units of mmHg is supplied from the RespirAct RA-MR device. This PCO2 data is then resampled on the same time series as the BOLD values, the TR. This is achieved by resampling the PCO2 data using a data interpolation technique.

4. Generally, there is a time delay between the external stimulus (PCO2) and the BOLD response in the brain. Therefore, to calculate the CVR values, the two plots are synchronized, and an area of study is defined using graphical tools. This is further explained in section 4.7.

5. For a given voxel, the percentage change in BOLD over time is calculated. The formula used to calculate the average BOLD is shown in equation (2.2). This is repeated for each voxel.

$$Mean_{BOLD} = \frac{\sum Individual\_BOLD}{Number\ of\ time\ steps} \tag{2.2}$$

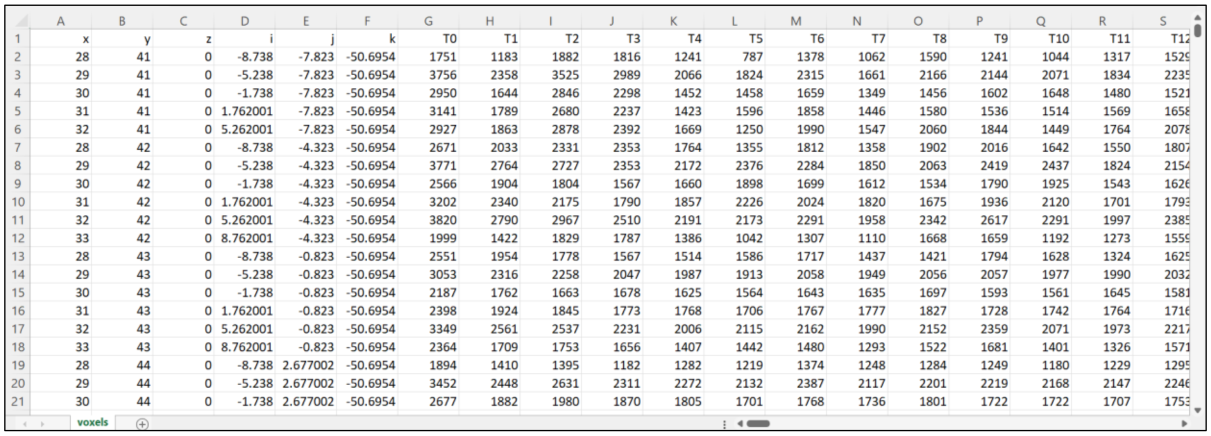
6. For each BOLD value, the percentage change with respect to the mean BOLD is calculated, as shown in equation (2.3).

$$Percentage_{change_{BOLD}} = \frac{Individual_{BOLD} * 100}{Mean_{BOLD}} \tag{2.3}$$

7. Finally, CVR is calculated for each voxel by computing the slope between the percentage change in BOLD values and the corresponding PCO2 values over the time series. This is again repeated for each voxel. Once the CVR values at different voxels (spatial locations) are calculated, a color-coded CVR map can be plotted over the corresponding anatomical scan to visualize and interpret the results.

Table 2.1: A sample output file for the BOLD values obtained from an MRI analysis

x	y	z	i	j	k	T1	T2	T3	→
28	41	0	-8.73	-7.82	-4.5	1702	1183	1862	→
↓	↓	↓							→



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	x	y	z	i	j	k	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12
2	28	41	0	-8.738	-7.823	-50.6954	1751	1183	1882	1816	1241	787	1378	1062	1590	1241	1044	1317	1529
3	29	41	0	-5.238	-7.823	-50.6954	3756	2358	3525	2989	2066	1824	2315	1661	2166	2144	2071	1834	2235
4	30	41	0	-1.738	-7.823	-50.6954	2950	1644	2846	2298	1452	1458	1659	1349	1456	1602	1648	1480	1521
5	31	41	0	1.762001	-7.823	-50.6954	3141	1789	2680	2237	1423	1596	1858	1446	1580	1536	1514	1569	1658
6	32	41	0	5.262001	-7.823	-50.6954	2927	1863	2878	2392	1669	1250	1990	1547	2060	1844	1449	1764	2078
7	28	42	0	-8.738	-4.323	-50.6954	2671	2033	2331	2353	1764	1355	1812	1358	1902	2016	1642	1550	1807
8	29	42	0	-5.238	-4.323	-50.6954	3771	2764	2727	2353	2172	2376	2284	1850	2063	2419	2437	1824	2154
9	30	42	0	-1.738	-4.323	-50.6954	2566	1904	1804	1567	1660	1898	1699	1612	1534	1790	1925	1543	1626
10	31	42	0	1.762001	-4.323	-50.6954	3202	2340	2175	1790	1857	2226	2024	1820	1675	1936	2120	1701	1793
11	32	42	0	5.262001	-4.323	-50.6954	3820	2790	2967	2510	2191	2173	2291	1958	2342	2617	2291	1997	2385
12	33	42	0	8.762001	-4.323	-50.6954	1999	1422	1829	1787	1386	1042	1307	1110	1668	1659	1192	1273	1559
13	28	43	0	-8.738	-0.823	-50.6954	2551	1954	1778	1567	1514	1586	1717	1437	1421	1794	1628	1324	1625
14	29	43	0	-5.238	-0.823	-50.6954	3053	2316	2258	2047	1987	1913	2058	1949	2056	2057	1977	1990	2032
15	30	43	0	-1.738	-0.823	-50.6954	2187	1762	1663	1678	1625	1564	1643	1635	1697	1593	1561	1645	1581
16	31	43	0	1.762001	-0.823	-50.6954	2398	1924	1845	1773	1768	1706	1767	1777	1827	1728	1742	1764	1716
17	32	43	0	5.262001	-0.823	-50.6954	3349	2561	2537	2231	2006	2115	2162	1990	2152	2359	2071	1973	2217
18	33	43	0	8.762001	-0.823	-50.6954	2364	1709	1753	1656	1407	1442	1480	1293	1522	1681	1401	1326	1571
19	28	44	0	-8.738	2.677002	-50.6954	1894	1410	1395	1182	1282	1219	1374	1248	1284	1249	1180	1229	1295
20	29	44	0	-5.238	2.677002	-50.6954	3452	2448	2631	2311	2272	2132	2387	2117	2201	2219	2168	2147	2246
21	30	44	0	-1.738	2.677002	-50.6954	2677	1882	1980	1870	1805	1701	1768	1736	1801	1722	1722	1707	1753

Figure 2.4: A detailed view of the voxels.1D file that contains the BOLD response across brain voxels over the MRI duration

2.2 Software Development using Python

Python is an interpreted, object-oriented, high-level programming language. It can be used for application development and scripting to work with existing components. It supports modules and packages encouraging program modularity and code reuse. Python's syntax is simple, easy to read, and therefore has a low cost of program maintenance. Another major advantage of using Python is that there is no compilation step which provides increased productivity [33].

Python can be used in several application domains, including, web and internet development, computing, desktop graphical user interfaces (GUIs), software development, and e-commerce systems [34]. Software developed using Python libraries can be packaged and published to a repository called the Python Package Index (PyPI) [35]. The same can be used to install Python software developed by others in the Python community.

2.2.1 Software Development Life Cycle

Software Development Life Cycle (SDLC) consists of the stages typically used for a well-designed software development process. SDLC defines the tasks that are required at various stages of the development process. There are mainly six stages in an SDLC which are explained below [36].

Stage 1: Planning and requirement analysis

This is the planning stage where a project timeline is defined, and the high-level project requirements are analyzed by the software developer based on the customer needs. This stage requires that the developer assesses the customer inputs and conducts surveys to understand the products and market.

Stage 2: Requirements Analysis

In this stage, the software requirements are outlined. Aspects like the functional requirements, non-functional requirements and user interface requirements are analyzed. Developers often create a software requirement specification (SRS) document to specify and list all the requirements.

Stage 3: Architecture Design

Based on the software requirements identified in stage 2, the next step is to define the best architecture for the software design. The best design is proposed based on the customer's needs and other factors.

Stage 4: Developing software

This stage involved the development activities for the target software product. A specific programming language and corresponding development tools are used for this. The development is sometimes accompanied by simple testing routines depending on the testing plan.

The software development process is often tracked using a source-control tool, like GitHub [37].

Stage 5: Software Testing and Integration

Once the software is developed, the next step is to rigorously test the software. There are different types of testing, like unit tests, integration tests, functional tests, end-to-end tests, acceptance testing, performance testing, and smoke testing [38].

Stage 6: Deployment and Maintenance

Once the testing is complete, the final product is ready for deployment and release. Once deployed, it undergoes final testing in a real customer environment. If the software passes all the tests, it is distributed to the customers. Based on customer feedback, the software undergoes further changes and maintenance.

2.3 Artificial Intelligence

The invention of Artificial Intelligence (AI) stemmed from inventors dreaming of creating machines that can think [39]. AI is a field of science and engineering that utilizes computers and machines to enable problem-solving and decision-making. Today AI is making steep advancements with many practical applications such as speech recognition, computer vision, understanding natural language, heuristic classification, and game playing [40]. More commonly used terms, Machine Learning (ML) and Deep Learning (DL) are subsets of AI. This is shown in Figure 2.5.

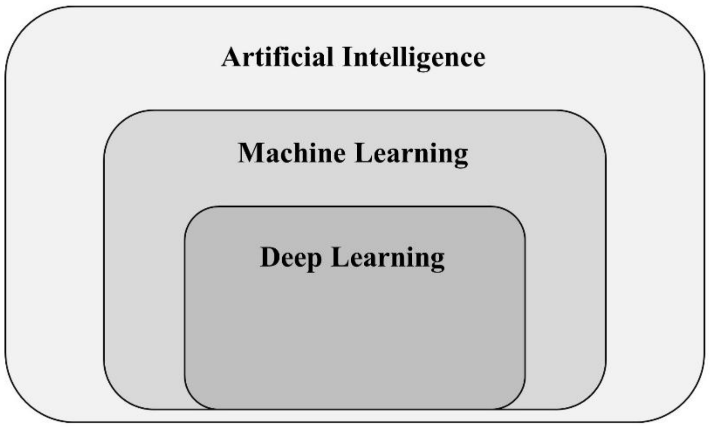


Figure 2.5: Representation of Artificial Intelligence and its subsets, Machine Learning, and Deep Learning

A machine learning algorithm is an algorithm that learns from the data [39]. A commonly accepted definition is: “A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E” [41]. Conventional machine learning models require the input parameters to be structured and well-defined to obtain meaningful relations or patterns. Processing raw input data to produce a suitable representation requires careful engineering and subject expertise and this step is called feature engineering [42].

Representation Learning consists of algorithms that train a machine to automatically discover the representations needed for making decisions using unprocessed and raw data. Deep Learning is a subset of machine learning, and a type of representation learning, that is capable of automatically extracting meaningful representations from the raw data. Deep Learning models consist of multiple layers, each layer consisting of non-linear modules that transform the representation [39]. The building blocks of a deep learning model are artificial neural networks (or neural networks). This is discussed in section 2.4.

2.3.1 Splitting Input Data

In a machine learning task, the input data is typically split into three categories to train the model and evaluate its performance. These categories are train, validation, and test sets, and each contains distinct samples from the original input data. This is explained below.

1) Training set: It is a subset of the original input dataset that is used to train the model to learn the patterns and key features. In each training cycle (epoch), the same training data is supplied to the model.

2) Validation set: The models use the validation set to update the model hyperparameters and configurations based on the performance during training. This process is repeated after each epoch.

3) Test set: This is a separate dataset that is used after the model training and optimization are complete. It essentially provides an unbiased final model performance and generalization capabilities in terms of model accuracy, and precision, on unseen data.

The most common ratio used for splitting the input data is 80%-10%-10% for the training, validation, and test sets, respectively. However, this ratio can be customized based on the specific requirements of the project [43].

2.3.2 Deep Learning and Small Datasets

Deep learning heavily relies on the amount of input data and typically requires large amounts of data to achieve a stable performance. This requirement makes it challenging to implement deep models in medical datasets given the limited number of input samples as compared to the learning parameters [44].

A small dataset contains a small number of samples. The quantification of small depends on the nature of the problem. Some of the challenges while working with a small dataset is [45]:

1) Outliers: An outlier is a sample that is significantly different from the other samples in the dataset
2) Overfitting: The network performs well with the training set but performs poorly with the unseen data
3) Sampling bias: the dataset fails to represent reality
4) Missing features: there might be a possibility that the network fails to identify all features from the small dataset.

The shortage of training samples enables the non-linear hidden layers to learn complicated relationships from the sampling noise which does not exist in the validation data [46]. This leads to the network fitting too closely against the training data. This causes the network to perform poorly against previously unseen data. This phenomenon is called overfitting [47].

Data augmentation and transfer learning are two commonly used countermeasures to address the issues arising from data scarcity. The first technique involves artificially producing synthetic data from the original dataset by implementing realistic modifications. In image analysis tasks, modifications can include resizing, rescaling, cropping, horizontal and vertical flips, rotations, shearing, and changes to brightness and contrast.

In the latter technique, the weights and parameters from a network trained on a different dataset are transferred and used for the target objective. The concept of transfer learning is elaborated in section 2.6.

2.4 Artificial Neural Networks

Artificial neural networks (or neural networks) are a type of ML model that is inspired by the human brain, mimicking the processing patterns of biological neurons [48]. A neural network uses a network of functions to establish interpretations from the input data and map it to a specific output. The uniqueness of these models lies in their ability to learn key characteristics, known as features, directly from the input data. The learning algorithm is capable of learning key characteristics from the labeled input dataset that is supplied during training. This means that the neural network does not need to be provided with specific rules or characteristics for the input data [49]. The term "deep" in deep neural networks essentially means that the network consists of three or more neural layers.

Neural networks are made up of node layers, consisting of the input layer, one or more hidden layers, and an output layer. Each node is connected to the other nodes in the next layer and has an associated weight and threshold. A node is activated if the output value in that node exceeds the threshold value, and the information is passed on to the next layer. Otherwise, the node is not activated, and the data is not forwarded [50].

The quintessential deep learning models are the multilayer perceptrons (MLPs), also called feedforward neural networks. In abstract terms, the goal of an MLP is to approximate a function f* to accomplish a learning task. Mathematically, for a given input "x" and output "y" and a learning parameter "θ", the network defines a mapping y = f (x, θ) and calculates the best values of θ that can improve the function approximation [39]. In this study, this function is required to map an input image (CVR map) to a patient category (healthy or unhealthy).

2.4.1.1 Working Algorithm

Each node can be considered a linear regression model. It will consist of input data, weights, a threshold (or bias), and an output. Depending on the activation function, the formula used to calculate the output from a node varies. This is discussed in section 2.5.2.2. For a binary step activation function, the formula used by a node to calculate the output function *[f(x)]* for an input *[x_i]*, based on the weights *[w_i]*, and a bias, is shown in equations (2.4) and (2.5) [50].

$$\sum_{i=1}^{m} wixi + bias = w1x1 + w2x2 + w3x3 + bias \qquad \textbf{(2.4)}$$

$$Output = f(x) = \begin{cases} 1\ if \displaystyle\sum_{i=1}^{m} wixi + bias \geq 0 \\ 0\ if \displaystyle\sum_{i=1}^{m} wixi + bias < 0 \end{cases} \tag{2.5}$$

A deep neural network or deep learning model is essentially a neural network with three or more layers. A deep learning model breaks the complicated process of a learning task into a series of simple mappings. Each layer of the network is responsible for extracting a specific interpretation of the input data. In an image processing task, the hidden layers of the network extract abstract features from the input images, for example, by identifying edges, corners, and contours [39]. A deep learning model has multiple levels of representation that obtain more and more abstract representation with each level [42].

To understand how a neural network works, consider the image classification task for a cat. There are several possibilities for how an image of a cat might look and writing a code for every scenario would be difficult. Neural networks use a generalized approach to extract characteristic information from an image. Using multiple layers, the neural network identifies trends from the diverse dataset and classifies the images by their similarities [49]. A typical neural network with the input, hidden, and output layers is shown in Figure 2.6.

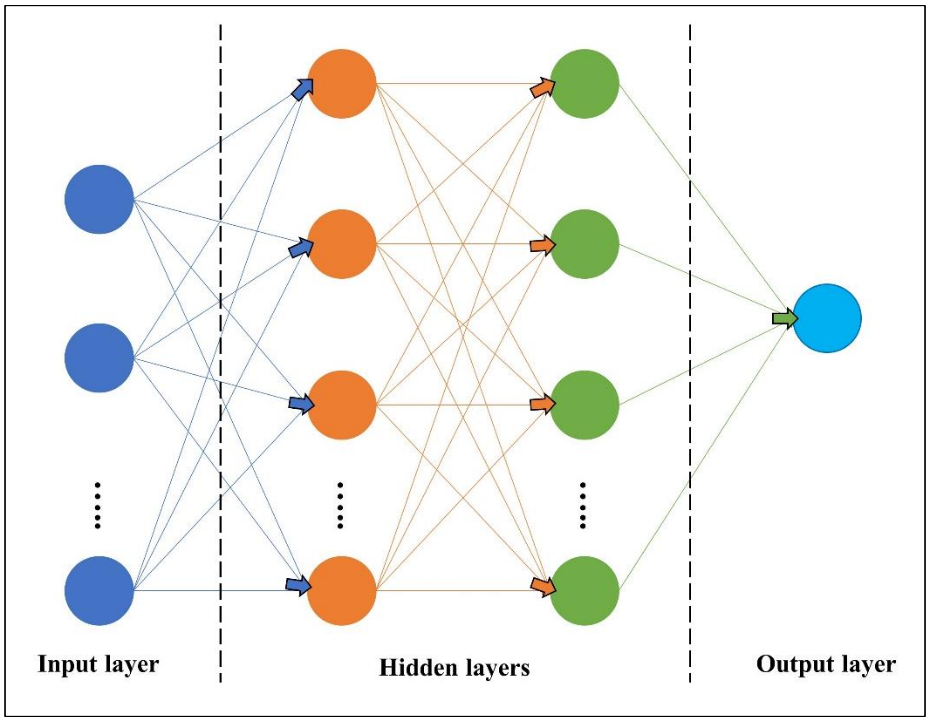


Figure 2.6: A typical neural network with the input layers, hidden layers, and output layers (during the forward pass). The dots represent additional neurons in each layer depending on the network architecture. The dashed lines separate the three types of layers

2.5 Convolutional Neural Networks

2.5.1 Introduction

A Convolutional Neural Network (CNN) is a specialized type of deep learning model that employs a linear mathematical operation called convolution. The uniqueness and benefits of using a CNN is it reduces the number of parameters in an artificial neural network. They are one of the most popular DL algorithms in processing data with grid-like topology, for example, images. The application of CNNs in tasks related to computer vision rapidly increased after a breakthrough work that used this network to almost halve the error rate for object detection using the ImageNet LSVRC-2010 contest's dataset [42][51].

A CNN is mainly comprised of convolution layers and fully connected layers. The convolution layers extract meaningful representations from the raw input data and feed it to the fully connected layers for the classification task. The first step is called feature extraction and the fully connected layers act as the classifier. Therefore, a CNN can be thought of as a combination of feature extractors and a classifier. The different components of a CNN are explained in section 2.5.2.

In mathematical terms, convolution is a linear operation that multiplies a set of weights with the input. CNNs are purposed to process data that can be represented as multiple arrays [52]. In an image-related learning task, a CNN takes an order 3 tensor as its input, for example, an image with dimensions of h rows, w columns, and 3 channels (R, G, B colors). This input then goes through a series of layers, where each layer is responsible for a specific processing task. The different types of layers in a CNN can be a convolution layer, a pooling layer, a normalization layer, a fully connected layer, a loss layer, and so on [52]. In medical image analysis, CNNs are particularly effective because of their ability to preserve local spatial relationships when filtering input images [53].

2.5.2 Components of a CNN

2.5.2.1 Convolutional Layer

In an image analysis task, the input function consists of pixel values at a position in the image represented as an array of numbers. When the convolution is performed, the dot product between the input and the filter is performed. And the filter is shifted to the next position in the image. This shift length is defined as the stride. This process is repeated until the entire area of the image is covered, and the obtained output is called a feature map [53]. Figure 2.7 shows the convolution operation on a 7 x 7 matrix with a 3 x 3 kernel.

The convolution operation is denoted with an asterisk (*). A convolution operation on input I(t), with a filter K(a), and producing a feature map s(t) can be represented by equation (2.6) [39].

$$s(t) = (I * K)(t) \tag{2.6}$$

If t is limited to integer values only, the discretized equation is represented by equation (2.7) [39].

$$s(t) = \sum_{a} I(a).K(t-a) \tag{2.7}$$

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 4 & 3 & 4 & 1 \\ 1 & 2 & 4 & 3 & 3 \\ 1 & 2 & 3 & 4 & 1 \\ 1 & 3 & 3 & 1 & 1 \\ 3 & 3 & 1 & 1 & 0 \end{bmatrix}$$

Figure 2.7: The output matrix [right] obtained by convolving a 7x7 input matrix [left] against a 3x3 kernel [middle] with a stride of 1

While using a convolution operation, the first argument is called the input, the second argument is called the kernel (or a filter), and the output is commonly referred to as the feature map [39]. The effects of different convolution matrices are explained in Table 2.2.

Table 2.2: Different convolution matrix and its effect on the input image [54]

Operation	Filter (3x3 matrix)
Identity	$\begin{matrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{matrix}$
Edge Detection	$\begin{matrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{matrix}$
	$\begin{matrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{matrix}$
	$\begin{matrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{matrix}$
Sharpen	$\begin{matrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{matrix}$
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$

2.5.2.2 Activation Functions

The layer following the convolution layer is an activation function that adds non-linearity to the output. This is used to adjust or cut off the generated output after convolution, saturating the output [55].

Sigmoid

The sigmoid [56] or logistic activation function produces values between 0 and 1. It is especially used for models where a probability needs to be predicted. However, it can cause issues like a vanishing gradient problem [57]. Therefore, it is commonly used in the output layer. Another disadvantage of the sigmoid function is that it can disable the network from learning to delay the convergence [58]. The function is shown in equation (2.8).

$$\theta(x) = \frac{1}{1 + e^{-x}} \tag{2.8}$$

Tanh

Tanh is a shifted version of the sigmoid function and is bounded between -1 and 1. Tanh also has the same disadvantage as the sigmoid function, and it can lead to a vanishing gradient problem [58]. The function is shown in equation (2.9).

$$f(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{2.9}$$

Rectified Linear Unit (ReLU)

The ReLU activation function [59] converts any negative values to zero. ReLU accelerates the neural network calculations and avoids the vanishing gradient problem [59]. For an input x to the neuron, the function is shown in equation (2.10).

$$f(x) = \begin{cases} x, & if\ x \geq 0 \\ 0, & otherwise \end{cases} \tag{2.10}$$

Exponential Linear Unit (eLu)

The eLu activation function is similar to the ReLU but has a non-zero value for negative output values. This helps the network converge faster and mitigate the vanishing gradient problem [58]. The function is shown in equation (2.11).

$$f(x) = \begin{cases} x, & if\ x \geq 0 \\ \alpha(e^{x} - 1), & otherwise \end{cases} \tag{2.11}$$

2.5.2.3 Pooling Layer

The pooling layer reduces the number of parameters to be calculated and is used between the convolutional and the activation layers. Its main purpose is down-sampling which reduces the complexity of the feature maps for the succeeding layers [60]. In image analysis, it reduces the width and height of the image, thereby reducing the dimensionality of the features. The different types of pooling operations are max-pooling, average pooling, and L2-normalization pooling. In this work, a max-pooling layer is used, which essentially extracts the largest input value from the filter [59]. Figure 2.8 shows a simple max pooling operation.

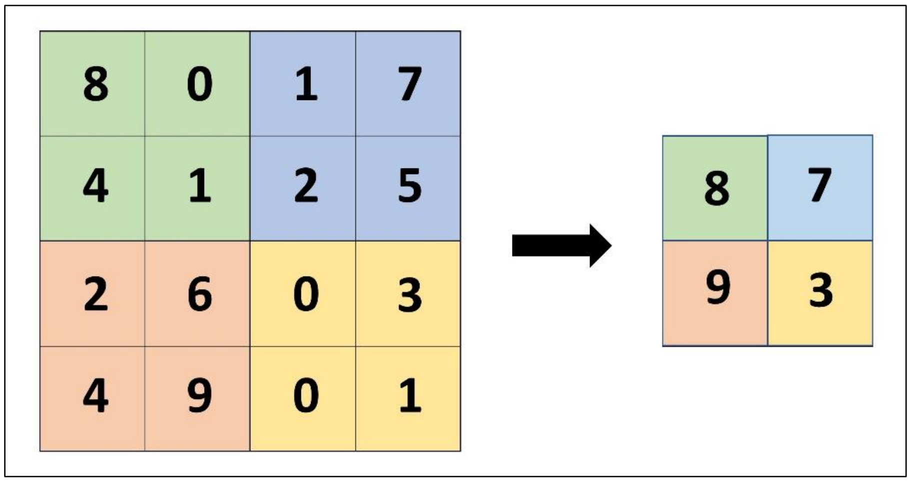


Figure 2.8: Max pooling operation with a 2x2 filter and a stride of 2. Notice how it extracts the maximum value from the filters [represented by the different colors]

2.5.2.4 Fully Connected Layer

The last layer in a CNN is the Fully Connected Layer. This essentially means that every neuron in the layer before is connected to every neuron in the Fully Connected Layer. Like other layers, there can be more than one fully connected layer depending on the required level of feature abstraction. The main function of this layer is to use the output from the preceding layer as input and computes the probability score for classification [59].

Figure 2.9 shows a sample CNN architecture composed of an input layer, a convolution layer, a max pooling layer, a flatten layer, a fully connected layer, and an output layer.

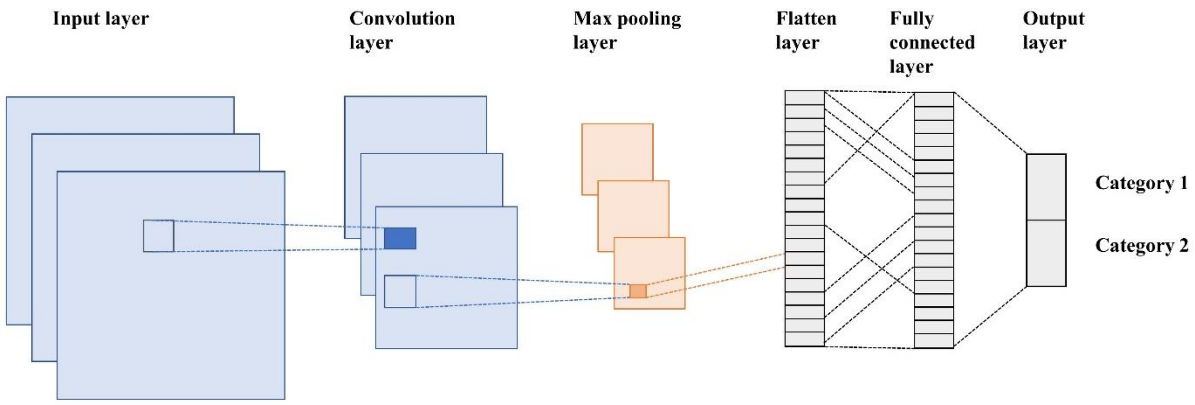


Figure 2.9: A sample CNN architecture with the different types of layers

2.5.2.5 Dropout Layer

Dropout [61] is a regularization method used to stochastically set the activations of hidden units to zero [52]. The number of units to be removed is defined using a probability $p \in (0, 1)$ value, also called the dropout rate. The dropout layer functions only during the training phase. A dropout layer is generally applied between two fully connected layers. Although dropout is not commonly used after convolutional layers because max-pooling already accounts for spatial reduction, research has indicated that adding a dropout layer after convolution layers can improve the model performance [52] [62]. In this study, both these techniques are investigated.

2.5.3 CNN Working Algorithm

CNN mainly operates using two operations called forward propagation (also called a forward pass) and backward propagation (also called a backward pass). During a forward pass, the outputs of the layers are computed using the input variables from the previous layer. It calculates an output function f(x, y) based on the inputs x and y. In a backward pass, the internal network parameters are updated to optimize the loss function [63].

In a classification problem, the final output of the CNN model is calculated using a sigmoid or softmax function. For a sigmoid function [equation (2.8)], as explained in section 2.4.1.2, the output value is between 0 and 1. Based on the probability value for the given sample (either <0.5, or >= 0.5), the output class is assigned. The cross-entropy loss function is most commonly used for evaluating the model performance. A Stochastic Gradient Descent (SGD) algorithm is typically used by a CNN to optimize the network parameters and achieve an optimal loss function value. The performance of the CNN can be evaluated using metrics like the train and validation accuracy and the loss functions.

2.5.4 CNN Architecture

The CNN architecture designed in this study was accomplished using Python-based machine learning libraries, called TensorFlow [64] and Keras [65]. TensorFlow is an end-to-end machine learning platform developed by Google. It operates on multidimensional arrays, also called tensors, represented as tf.Tensor objects [64]. Keras is a Python-based deep learning Application Programming Interface (API) that runs on top of TensorFlow [65]. The main data structures are layers and models.

The main steps involved in designing a CNN are:

1) Data pre-processing
2) Defining a base model
3) Identifying the critical hyperparameters
4) Fine-tuning the identified hyperparameters

Data pre-processing involves performing data augmentations for small datasets, implementing required image processing steps, splitting the entire dataset into the train, validation, and test sets, and normalizing the input data before training [66]. The train set is the dataset used to fit the network and on which the network learns. The validation set is used to provide an unbiased evaluation of the network. This is typically used to fine-tune the hyperparameters. The test set is used to evaluate the final model. It is used as a final testing step once the network is completely trained with the best hyperparameter settings [67].

Hyperparameter is a configurable value that is set before the network training starts. The different hyperparameters for a neural network are [66]:

1) Number of layers [network depth]
2) Number of neurons in each layer [network width]
3) Activation functions

4) Optimizer
5) Learning rate
6) Batch size
7) Number of epochs
8) Dropout
9) L1/L2 regularization

The hyperparameters unique to CNN are [66]:

1) Kernel size
2) Padding
3) Stride
4) Number of channels
5) Pooling parameters

2.6 Transfer Learning

Transfer learning with CNN is the concept of transferring knowledge between networks at the parametric level. The objective of such an experiment is to use a pre-trained network, which is trained on generic features learned from a different image classification task and use these pre-trained weights to accomplish the target image classification task (in our case, classifying CVR maps). There are several publicly accessible pre-trained CNNs that can be downloaded and customized as per the project's needs [68].

There are broadly two approaches commonly applied for transfer learning activities. The first one is to use the pre-trained network as a feature extractor and build a custom CNN model using the features obtained. This is called a feature extractor. And the second one, called fine-tuning, involves updating the parameters of the pre-trained network during model fitting [68].

Transfer Learning proves very effective in implementing CNNs to small datasets. If the dataset is relatively small, the model starts to overfit. However, if a pre-trained model is used which has been trained on a much larger dataset, the learned features are more generic. This helps in addressing the overfitting issue when this knowledge is transferred to a small dataset [69].

In this work, some of the popular models trained using the ImageNet dataset (ImageNet large scale visual recognition challenge; ILSVRC) [70] were fine-tuned. AlexNet [71] is a first-generation CNN model that was developed in 2012. It is a shallow model as compared to other ImageNet-trained models that were developed more recently. The success of AlexNet boosted CNN research in computer vision tasks after it won the ImageNet large scale visual recognition challenge (ILSVRC) in 2012 [68].

Another popular model is the Inception [72], also called GoogLeNet, which outperformed AlexNet and set new benchmarks in the ILSVRC, in 2014. This model introduced a block concept employing different sized filters and then concatenating the multiple outputs before passing it to the deep networks [68]. The latest model of the inception family is the InceptionV3. VGG [73] is another deep CNN consisting of 16 (VGG-16) or 19 (VGG-19) layers. VGG-16 is another popular pre-trained model for image classification. VGG models stack consecutive convolution layers followed by the pooling layers. In a VGG-16 network, there are 13 convolution layers, 5 pooling layers, and 3 dense layers.

A big challenge while using very deep networks is the vanishing gradient problem, where the parameters of the initial layers in the network are poorly updated. ResNet (also called the Residual Net) [74] addressed this issue by proposing residual blocks and bypass or skip connections between layers.

EfficientNet [75] proposed a compound coefficient-based scaling method to provide a more structured way to scale up CNNs. The family of EfficientNet networks achieved state-of-the-art accuracy in several widely used transfer learning datasets, including, CIFAR-100, and Flowers.

A typical transfer learning architecture is shown in Figure 2.10. The figure demonstrates the mechanism of fine-tuning the Inception V3 model which is trained originally using the ImageNet dataset. The pre-trained weights of the Inception model are used and the last layer(s), for example, the output layer, are reconstructed to suit the target objective.

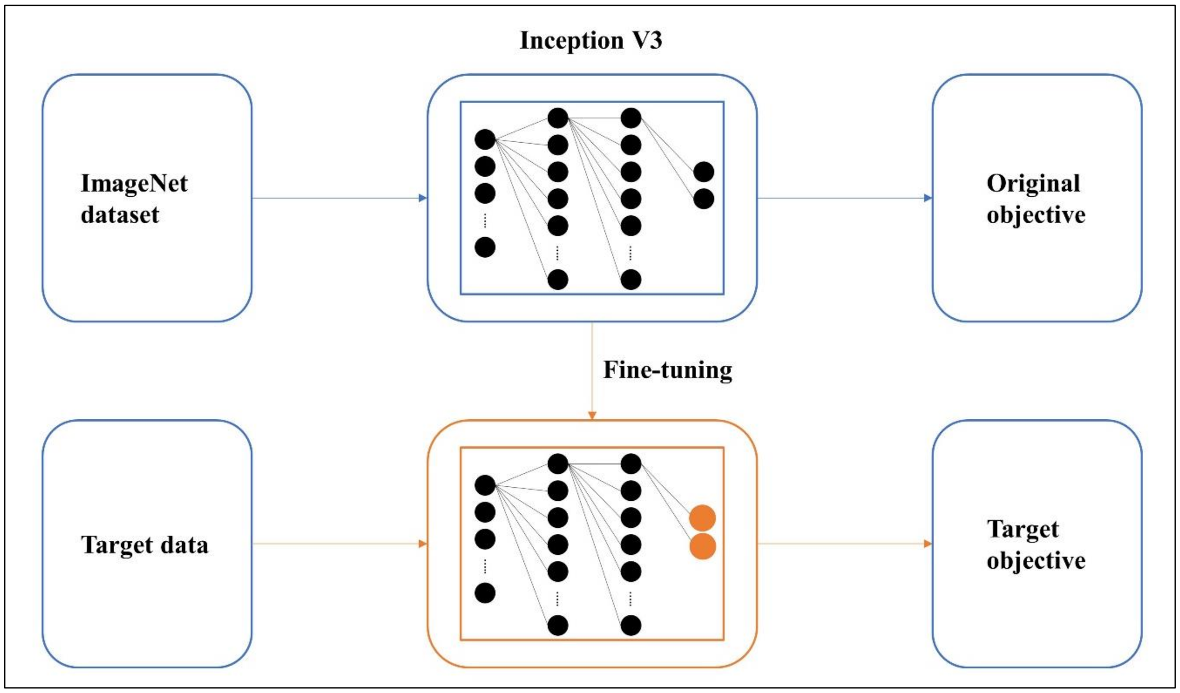


Figure 2.10: Representation of the transfer learning mechanism. Inception V3, trained on the ImageNet dataset is fine-tuned to use its pre-trained weights on the new dataset and accomplish the target objective

2.7 Related work: Machine Learning and CVR

Some of the widely used machine learning techniques in medical imaging are support vector machines (SVMs) [76], neural networks [77], k-nearest neighbor classifier [78], linear discriminant analysis [79], and boosting-based learning [80]. Kloppel et al. successfully used SVMs to classify structural MRI for AD detection and obtained an accuracy of 96% [81]. Waddle et al. (2020) investigated the application of support vector machine (SVM) algorithms to identify cerebrovascular impairment [82]. A study investigated LDA with Fisher's discriminant rule, k-nearest neighbor (k-NN), and nonlinear SVMs to classify brain tumor types using MRI [83].

2.7.1 Application of Deep Learning in MRI

The main advantages of using a deep learning model with medical image processing are [84]:

1. Deep learning is capable of extracting features automatically from the input dataset. Therefore, it does not require subject expertise or manual feature engineering.
2. Deep learning has the ability to extract unique features customized to a particular application, that might not be discovered by manual feature engineering.

Several studies have investigated the application of deep learning models to accomplish wide-ranging medical imaging tasks including image classification, detection of anatomical and cellular structures, localization, object detection, segmentation, registration, and computer-aided disease diagnosis [85], [86], [87].

Farooq et al. designed a deep CNN pipeline for a 4-way classification task using MRI scans. The models are experimented on the ADNI dataset on a GPU-based processing system and obtained state-of-the-art results. The proposed model successfully classified three diseases, including Alzheimer's (AD), mild cognitive impairment (MCI), late mild cognitive impairment (LMCI), and healthy group. The prediction accuracy achieved is 98.8% [88]. Hashemzehi et al. used a hybrid model consisting of a neural autoregressive distribution estimation (NADE) and a CNN for brain tumor detection using brain MRI images. This model achieves a high classification performance using images belonging to three classes of brain tumors [89].

Salehi et al. designed a CNN and achieved an accuracy of 99% in classifying AD using MRI datasets. Chen et al. used a CNN to predict the cerebrovascular reserve in Moyamoya disease using a simultaneous [15O]-water PET/MRI dataset [90]. Hussein et al. designed a two-phase deep learning model that translated MRI images to Positron Emission Tomography (PET) and then classified the images into healthy control, Moyamoya disease, intracranial atherosclerotic steno-occlusive disease (ICSD), and stroke. The average accuracy for the classification task was 96.38% [91].

Chen et. al. proposed a deep learning network that predicts cerebrovascular reserve by using PET-plus-MRI and MRI-only images, thereby making the use of an acetazolamide vasodilator redundant. The study used simultaneous oxygen 15-labeled water PET and MRI and assessed results before and after the vasosimulation. The analyses were conducted on patients with Moyamoya disease and healthy subjects [92].

Nie et al. proposed a 3D CNN-based architecture to use multi-channel data for feature extraction. The deep learning models are implemented for the task of feature extraction from several brain imaging datasets, like, T1 MRI, fMRI, and DTI. Using the features, a support vector machine is used to predict the overall survival time for tumor patients [93].

Hou et al. investigated the application of deep learning to detect vascular abnormalities using cerebrovascular reactivity (CVR) and bolus arrival time (BAT) maps of the brain obtained using resting state fMRI studies [94]. Dhar et al. investigated the application of single 2D U-net deep learning architecture is discussed, which delineated the intracranial hemorrhage and perihematomal edema [95].

So far, the research discussed involved the detection, classification, and prediction of diseases using various medical imaging techniques. Zhu et al. reviewed the other domain of the application of deep learning in medical imaging, for creating and enhancing stroke imaging. In the review, the application of deep learning in stroke imaging particularly using Computed Tomography (CT) and MRI is discussed [96].

2.7.2 Application of Transfer Learning in MRI

Many researchers have implemented the concept of transfer learning in accomplishing tasks related to medical imaging. Talo et. al. implemented the CNN-based ResNet34 model to successfully classify brain images into normal and abnormal classes using 613 MR images. Techniques like image augmentation, optimal learning rate finder, and fine-tuning were used [69].

Maqsood et al. used the popular pre-trained network, AlexNet, for both binary and multiclass classification of 3D MRI images. The network is trained over segmented and unsegmented images. The overall accuracy obtained during the classification of multi-class unsegmented images was 92.85% [97].

Yuan et. Al. developed a novel multiparametric MR model using transfer learning to classify prostate cancer. This study uses the pre-trained model as a feature extractor. The model is then fine-tuned to classify the images and the achieved accuracy is 86.92%. The feature extractor achieves performance better than similar deep learning models built on hand-crafted features [98].

A very interesting work, which provides direction to the future of the research discussed in this thesis, investigates the application of transfer learning methods for domain adaptation to accommodate the differences in MRI scan parameters. The study investigated white matter hyperintensity segmentation by training CNN on MRI images. The findings conclude that transferring pre-trained weights of the model trained on one set of MRI images to a different set of MRI images achieved a dice score of 0.63, and outperformed a new network trained on the new MRI images from scratch [99]. This is an indication that the weights obtained in this thesis could be potentially extended to a different CVR dataset.

Based on this literature review, it can be established that several studies have successfully implemented deep learning models to classify MRI images using various modalities and using different types of stimuli to map CVR responses. Researchers have also successfully used pre-trained models to suit the target classification objectives, achieving high model performance. This research extends the application of deep learning models to classify CVR maps obtained using the novel prospective end-tidal targeting of carbon dioxide.

Chapter 3

Problem Analysis

3.1 Problem Description

As described in the introduction, the current generation of CVR analyses is conducted using shell scripts and a variety of programs that need to be executed in a specific order to obtain the CVR maps. Several shortcomings were identified in the current generation of CVR processing tools. Therefore, the objective of this project was to develop an in-house CVR processing and visualization software application, engineered using open-source programming tools. The software was intended to be used as a research tool to process raw BOLD-MRI files in the DICOM format and produce the corresponding CVR maps using AFNI modules. The developed application was required to have a simple and clear workflow, allowing it to be operated with minimal subject expertise. Another key requirement of this software was to allow users with little to no AFNI expertise to process and visualize CVR maps. Finally, the software would provide a framework to investigate the application of deep learning models to classify CVR maps. This deep learning-based classification tool is integrated as an independent module into the proposed software and is discussed in the next chapter.

Additionally, the clinical assessments of CVR maps relied on a team of radiologists and doctors who made judgments based on their subject expertise and years of experience. The objective of this research was to extend the screening capabilities of the CVR software. Convolutional Neural Networks (CNNs) are investigated to classify CVR maps into healthy and unhealthy categories. Although existing literature has research that used conventional machine learning and deep learning models for CVR analysis and classification, the implementation of CNNs to classify CVR maps obtained using the prospective end-tidal targeting of CO2 is unprecedented as per the author's best knowledge.

3.2 Software Requirements

As discussed in section 1, the current generation of CVR processing tools had several shortcomings. Researchers at the participating institutions were consulted to evaluate the requirements and outline the research workflow currently used. Based on the consultation and the sponsor's needs, the functional, non-function, and user interface requirements of the software were identified. These are discussed below.

3.2.1 Functional Requirements

The functional requirements of the CVR Processing software are listed below.

1) The software should create a main input folder with the patient's name and three sub-folders for anatomical, BOLD, and MR sequence information
2) The software should create the main output folder
3) The software should allow users to view the input files in the corresponding folders
4) The software should allow users to upload the input files in the corresponding folders
5) The software should check if the input files are placed in the correct sub-folders
6) The software should display the status of the input files
7) The software should execute a series of AFNI executable files to realize the processing workflow, using the raw DICOM input files and producing AFNI HEAD/BRIK files
8) The software should plot the BOLD signals and PCO2 values on a dual-axis plot
9) The software should allow users to synchronize the PCO2 curve with the BOLD curve
10) The software should allow users to select an area of interest once the plots are synchronized
11) The software should calculate the CVR values across the brain voxels and write to an output file
12) The software should plot the CVR maps using AFNI executable file
13) The software should save the CVR maps in a JPEG format in the main output folder

3.2.2 Non-Functional Requirements

The non-functional requirements of the CVR Processing software are listed below.

1) The software should be operating system agnostic
2) The software should not experience critical failures
3) The software should support one patient analysis at a time
4) The software should allow multiple CVR studies for the same patient
5) The software should not use any paid or licensed tools
6) The software should be easy-to-use
7) The software should allow easy distribution to the participating institutions

3.2.3 User Interface Requirements

The user interface requirements of the CVR Processing software are listed below.

1) The software should be a standalone GUI application; with the exception of using AFNI modules when required
2) The software should display a warning message stating that the software is an investigational device and must be used by qualified investigators only
3) The GUI should display readable and easy-to-understand instructions for the users
4) The software should display a status message when a task is completed

3.3 Process Flow Diagrams

As described in the background, CVR studies require a series of computational steps to produce the CVR maps for a given subject. Based on the research workflow used by our sponsors, a process flow diagram was developed highlighting the key steps required to accomplish a CVR analysis. This is shown in Figure 3.1.

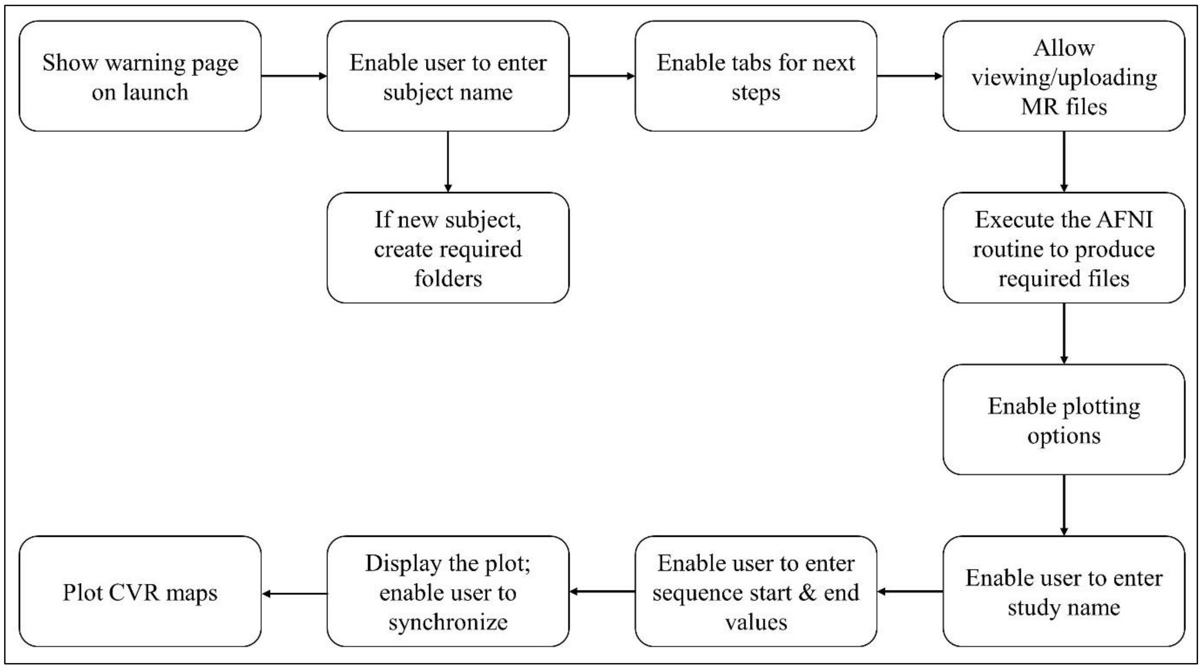

Show warning page on launch
Enable user to enter subject name
Enable tabs for next steps
Allow viewing/uploading MR files
If new subject, create required folders
Execute the AFNI routine to produce required files
Enable plotting options
Enable user to enter study name
Enable user to enter sequence start & end values
Display the plot; enable user to synchronize
Plot CVR maps

Figure 3.1: Process flow diagram of the CVR Processing Software

With a framework available to conduct CVR analyses, the CNN classifier furthers the CVR research workflow. Since the original software was developed following object-oriented design principles, the CNN model was also developed as a separate module that can be easily integrated into the software workflow. The process flow diagram for the CVR Processing-Assessment workflow is shown in Figure 3.2.

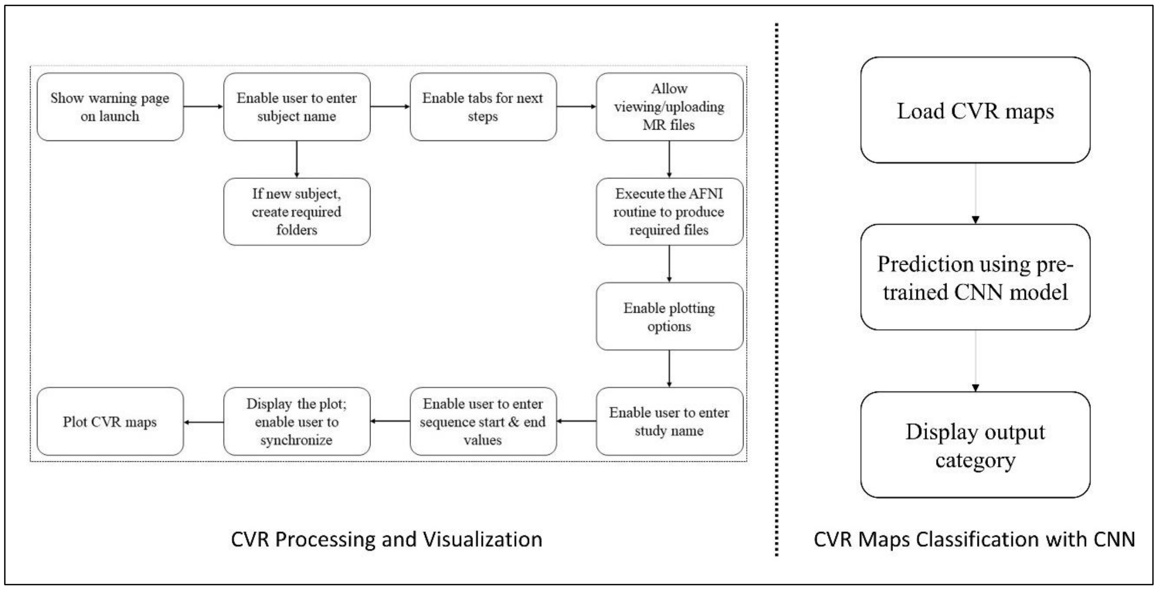

Show warning page on launch
Enable user to enter subject name
Enable tabs for next steps
Allow viewing/uploading MR files
If new subject, create required folders
Execute the AFNI routine to produce required files
Enable plotting options
Enable user to enter study name
Enable user to enter sequence start & end values
Display the plot; enable user to synchronize
Plot CVR maps
Load CVR maps
Prediction using pre-trained CNN model
Display output category
CVR Processing and Visualization
CVR Maps Classification with CNN

Figure 3.2: Process flow diagram of the CVR Processing-Assessment Software workflow

3.4 Software Architecture

The first step in developing the software was to identify the appropriate development tools. Based on the software requirements listed in section 3.2, Python was identified as a well-suited programming language. Python is an open-source language that has a relatively easier syntax and integrates with other programs seamlessly. It also supports object-oriented techniques allowing quick modifications to the application modules. Additionally, Python enables GUI applications that can be ported to several Python libraries and system calls. Finally, Python provides a better structure to be scaled as compared to shell scripts [100].

The software was developed using PyCharm IDE [101] on a Linux-based environment installed on a virtual machine (VM). VM is a software-only compute resource that runs on a physical host machine [102]. Bitbucket is used as the version-control tool [103].

Chapter 4

CVR Processing Software Development

4.1 Project Organization

An object-oriented software development methodology was followed [104]. Classes were used to create customized data structures and methods to define the functions. The program codes were developed as separate modules, and these modules were imported as per the requirement. This section discusses the functionality and features of the proposed software.

The object-oriented principles allowed the software to be flexible and the main GUI design attributes can be changed by modifying a few parameters in the GUI module. All other modules, and therefore GUI pages, inherit these features automatically. Using object-oriented principles also allowed for a faster development and deployment process because modules communicate with one another, and codes do not have to be written from the scratch. A modular code structure allowed easy feature upgrades and testing of the modules individually and independently using a unit test suite. Modules also allowed us to separate the code into smaller parts storing related data and functionality [105].

The project repository was organized following the standard recommendations [105]. The main folder is called cvr_proc denoting CVR_Processing (software). In the main folder, the src folder contains the scripts. The docs folder contains the images and files required for the application interface. The tests folder has the unit test scripts used to test the individual modules from the src folder. The README.md is a markdown file that contains basic information about the project, installation steps, and a guide to running the program. Figure 4.1 shows the repository structure. The module-based scripts developed for the application are shown in Figure 4.2. The file default.txt contains meta-information for the color scale of the CVR maps.

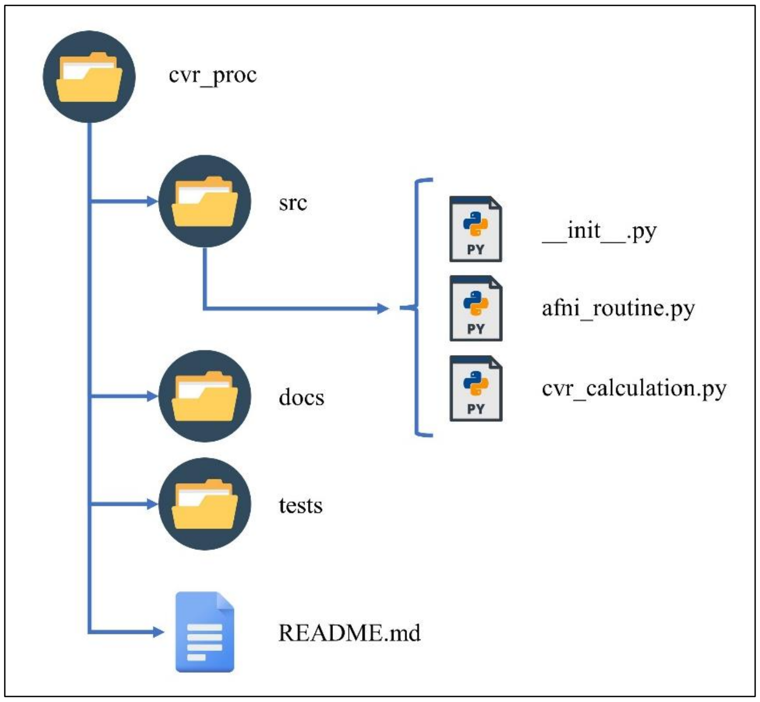


Figure 4.1: The working directory used for the CVR Processing software development. The icons used in this figure are from Flaticon [106]

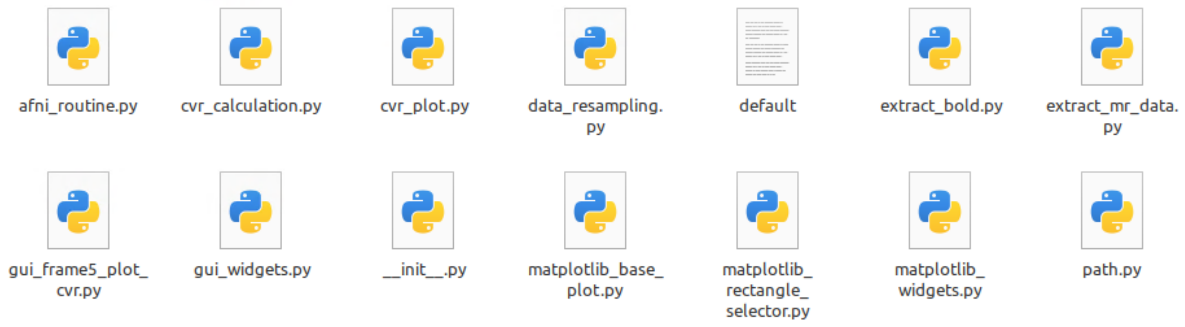


Figure 4.2: Python modules developed for the CVR Processing software, located in the src folder

4.2 Calculations and Interpolations

4.2.1 BOLD Calculations

As discussed in section 2.1.3, CVR is calculated as the ratio of the change in BOLD values with respect to the change in the end-tidal PCO2. There were two different approaches that were used to assess the BOLD changes. The first formula calculated the percentage changes in BOLD by subtracting the mean BOLD value from each of the individual BOLD values in each voxel. The second formula did not subtract the mean BOLD value. The two approaches are shown in equations (4.1) and (4.2).

$$\Delta BOLDpercentage(1) = \frac{BOLD(i) - BOLDMean}{BOLDMean} * 100 \tag{4.1}$$

$$\Delta BOLDpercentage(2) = \frac{BOLD(i)}{BOLDMean} * 100 \tag{4.2}$$

The difference in the corresponding percent change in BOLD obtained using the two approaches for the given subject was less than 0.01. This difference did not cause a significant difference in the CVR maps produced. This is observed in Figure 4.3 as the key regions of blue and red remains the same. As discussed in section 2.1, as per the color scale used, the red regions represent a positive CVR value, highlighting a regular CVR response in the brain regions. The blue regions represent an abnormal CVR response in the brain regions, indicated by negative CVR values. Based on the feedback from researchers and senior scientists, equation (4.2) was used as the default formula for CVR studies. However, a separate module allows the user to change to the other method. This was implemented as a beta feature and is accessible only by internal researchers.

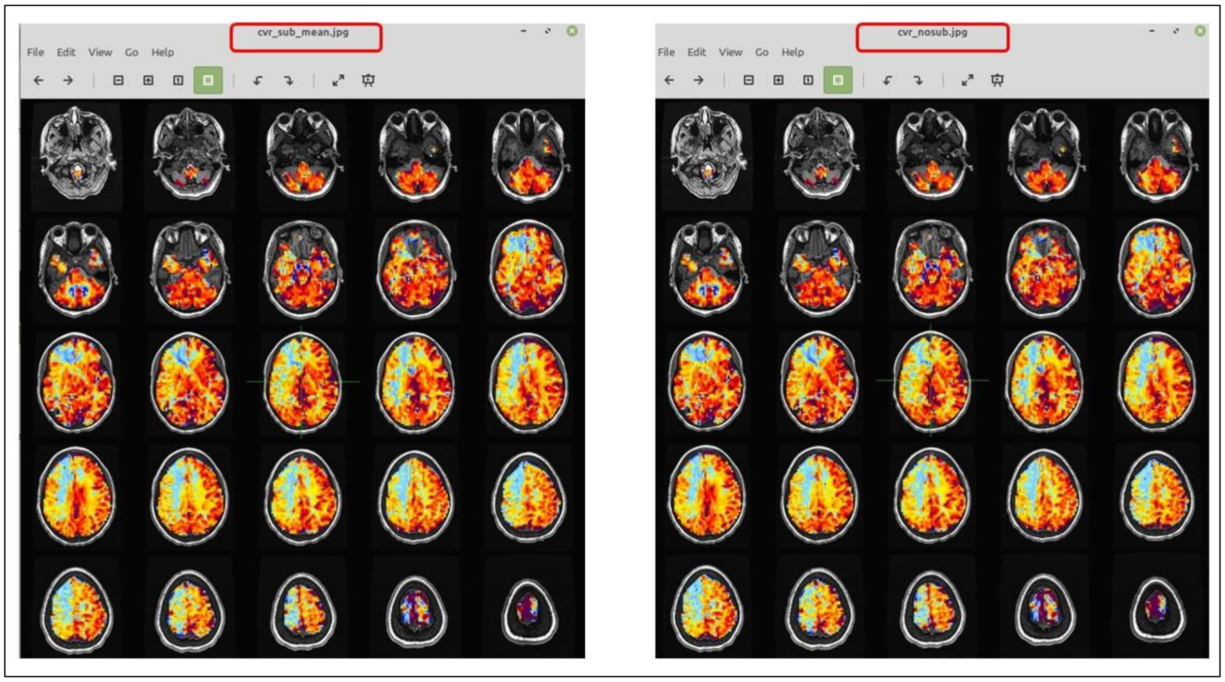


Figure 4.3: Comparison of the CVR maps obtained from two approaches of calculating the percentage difference in BOLD. The maps in the left image were obtained by subtracting the mean from individual BOLD values and then dividing by the mean BOLD to calculate the percentage BOLD change, represented by equation (4.1). In the maps in the right image, the ratio of individual BOLD values to mean BOLD was used to calculate the percentage BOLD change, represented by equation (4.2)

In an MRI study, the repetition time is the duration of time between successive pulse sequences applied to the same brain slice [107]. The repetition time (TR value) can be extracted from any of the DICOM images using the Pydicom library. The script to extract the TR value is presented below. The repetition_time method reads the first DICOM file in the BOLD dataset and extracts the metainformation stored in the [0x0018, 0x0080] index. This corresponds to the repetition time used for the given analysis.

Once the average change in BOLD values is calculated for all the voxels and the TR values are extracted, we create a new time series dataset based on the TR scale. For example, if there are 4 BOLD values for a given voxel, the time series is constructed as a list: [0, 2.4, 4.8, 7.2]. The BOLD values are then plotted against this new time series instead of linear x-values. The program used to extract the TR value is described in Appendix-B, section B.1.

4.2.2 PCO_2 Interpolations

The administered end-tidal PCO2 is logged in the EndTidal file generated by the RespirAct RA-MR device [25]. The PCO2 values are observed in real-time with the MRI study. The software reads the column headers and stores the data under "PCO2 (mmHg)" and "MR Time(s)". Due to various reasons, some entries may be missing, and the files store them as none values. The software ensures that these values are not stored. The program that accomplishes these functions is presented in Appendix-B, section B.2.

As discussed in section 2.1, CVR is calculated as the slope between the change in BOLD signal(s) with respect to the change in PCO2 administered. Therefore, the PCO2 is interpolated from its original real-time MRI time scale to a common time series used for BOLD. This creates a data point for PCO2 at the same time series as the BOLD values, allowing further CVR calculations.

The interpolation to convert the PCO2 data to the time series of BOLD was achieved using the Cubic spline data interpolator (scipy.interpolate.CubicSpline) [108].

4.3 Plotting MRI images with Pydicom

As the first step towards conceiving an MRI processing software, plotting the MRI images was accomplished using the Pydicom package [109]. The Pydicom module allows the development of Python programs to interact with the DICOM datasets. We used the anatomical and BOLD DICOM files for a subject to plot the respective MRI results. The program prompted the user to enter a slice number to view the 2-dimensional image of that slice. When the slice number is entered, the program creates the plot using the Matplotlib modules [110]. The images visualized for a test case are shown in Figure 4.4 and Figure 4.5.

Although the anatomical plots have a good resolution, the BOLD images were poorly constructed, as shown in Figure 4.4. Additionally, the research workflow heavily relied on AFNI programs. Therefore, it was decided that the plotting workflow would be accomplished using AFNI programs and not Pydicom.

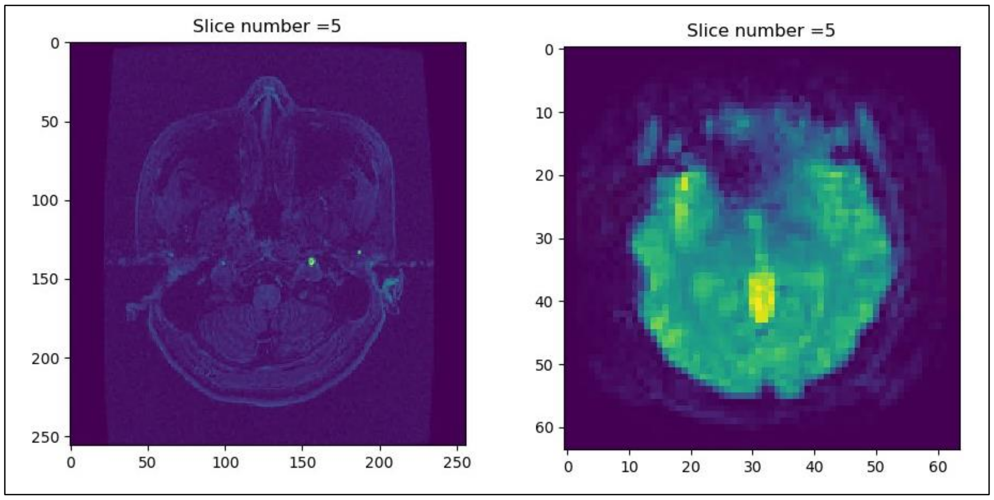


Figure 4.4: Anatomical and BOLD images for a subject plotted using the Pydicom and Matplotlib libraries. Left image shows the anatomical MRI image corresponding to the fifth slice. And right image shows the BOLD MRI image for the same slice

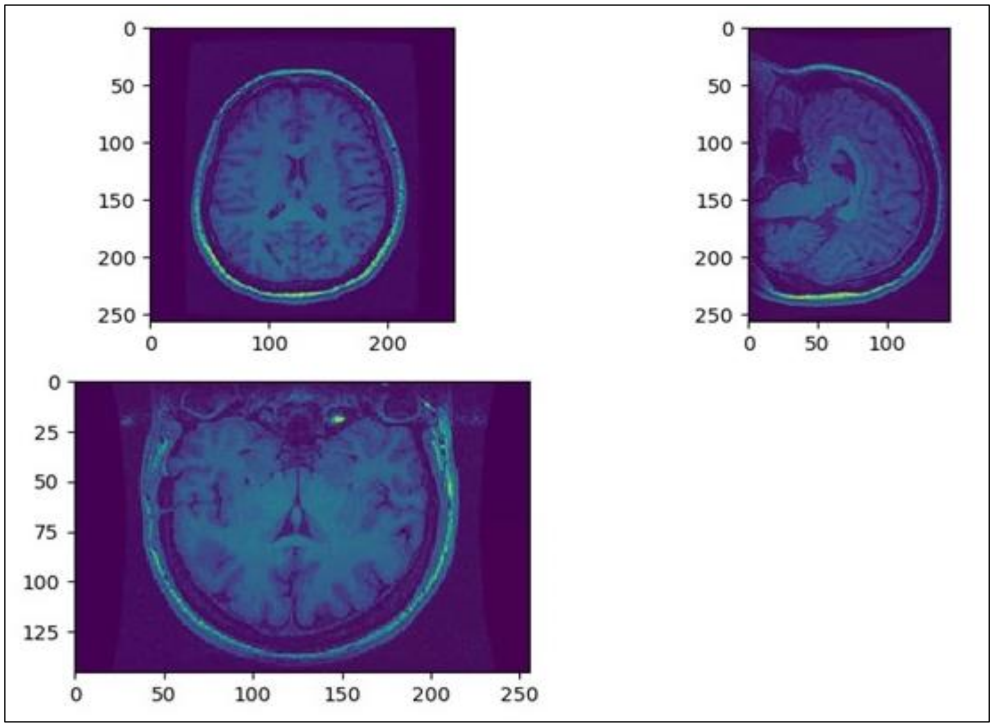


Figure 4.5: The three views for the Anatomical MRI images. Top left image shows the axial view, top right image shows the sagittal view, and the bottom image shows the coronal view

4.4 GUI Design

The first page that the software displayed, the landing page, required to have a user warning as per the company policies. The Tkinter package [111] was used for designing the GUI of the application. The GUI consisted of the main application window, a notebook that contained multiple tabs, and widgets placed on the different tab frames. The notebook design allowed the software to be a single-windowed application. Some of the Tkinter widgets used in this work are shown in Table 4.1 with a brief description of their functionality. The different components of the GUI are shown in Figure 4.6.

Table 4.1: Tkinter widgets used and their respective functions [112], [113], [114]

Sr. No.	Widget Name	Functionality
1	Notebook	Allows creation of tabs and corresponding pages. Creates a child pane associated with the selected tab
2	Frame	A container widget inside which other widgets can be organized
3	Label	Creates single-line captions or images
4	Entry	Used to accept a single-line text field entry from user
5	Button	Displays a button in the application
6	Canvas	Used to draw shapes in the application
7	Scrollbar	Enables scrolling feature to other widgets
8	ImageTk (From PIL)	Used to create Tkinter Photoimage from PIL images

For regulatory purposes, a warning message was required stating that the software is an investigational device that was to be used by qualified investigators only. To ensure that the user reads and agrees to the warning, an OK button was added to the landing page. And in case, the user is not qualified, the software can be closed using the Cancel button. The OK button hides the tab containing the warning message and opens the tabs used for the next steps in the same notebook.

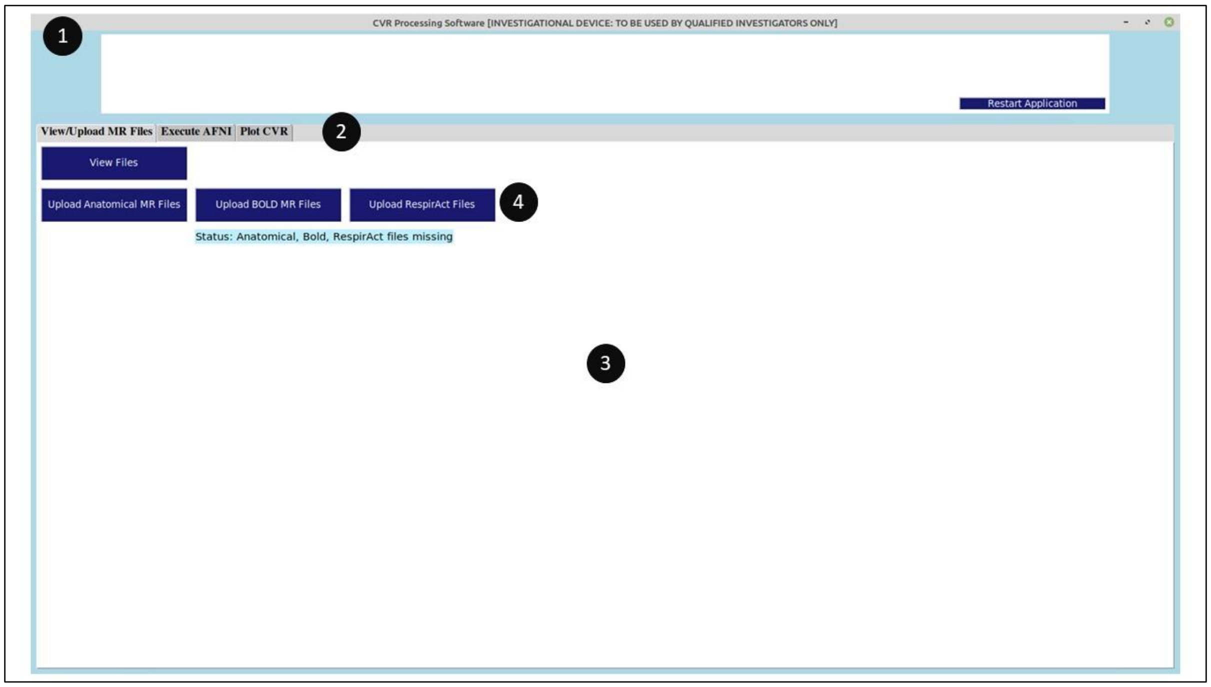

CVR Processing Software [INVESTIGATIONAL DEVICE: TO BE USED BY QUALIFIED INVESTIGATORS ONLY]
1
Restart Application
View/Upload MR Files
Execute AFNI
Plot CVR
2
View Files
Upload Anatomical MR Files
Upload BOLD MR Files
Upload RespirAct Files
4
Status: Anatomical, Bold, RespirAct files missing
3

Figure 4.6: CVR Processing Software GUI design. Label 1 shows the main window. Label 2 shows the notebook style frames with 3 tabs [View/Upload MR files; Execute AFNI; Plot CVR]. Label 3 shows the frame corresponding to tab 1: View/Upload MR Files. Label 4 shows the different widgets (buttons and text) on the frame of tab 1

4.5 File Management

To streamline the research workflow to conduct a CVR analysis, each analysis was linked to a subject ID. The subject ID acted as the first step to the CVR workflow and linked the input and output directories for the corresponding study. If the entered subject ID corresponded to a previously completed analysis, the directories were pre-populated according to the steps completed previously. And in case of a new analysis, the software creates the respective input and output folders linked to the subject ID.

For a given study, three types of input files are required, and the software creates three folders. The Anat folder contains the anatomical files of the subject in the DICOM format. The Bold folder contains the corresponding BOLD-MRI files of the subject in the DICOM format. The RA Files folder contains the files generated by the RespirAct RA-MR device including Events.txt and EndTidal.txt, which are the files of interest.

The software pastes the output files generated from processing the raw files from the input folder into the Output folder corresponding to the subject ID. These are mainly two categories of the output files, the first generated using the AFNI modules, and the second generated using a mix of plotting tools and AFNI modules.

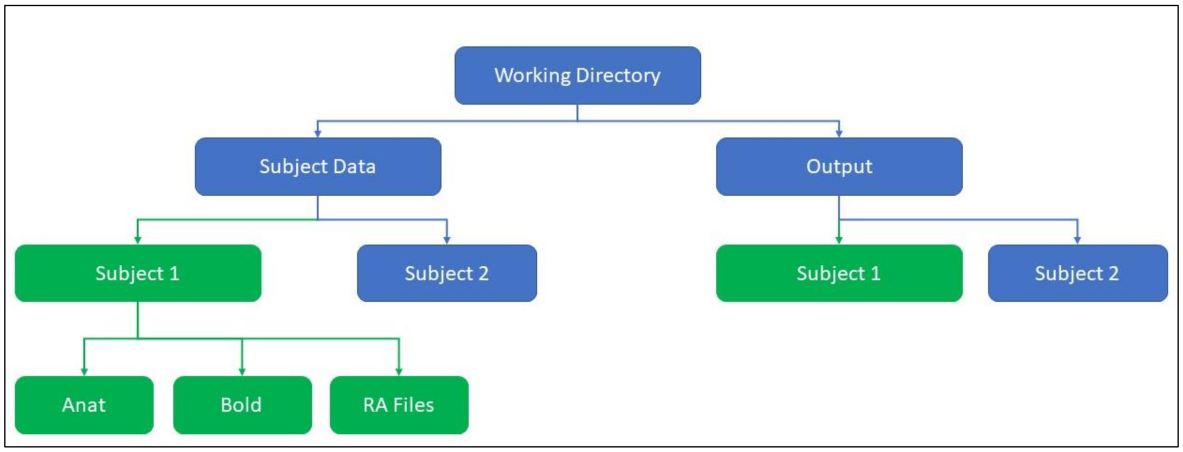


Figure 4.7: Folders and structured directory created for "Subject 1" as the subject ID

When a new analysis is started, with a unique subject ID, the software creates the folders in the directory structure shown in Figure 4.7. The software then allows users to view and upload the required input files from the user interface itself. The View Files and Upload Files buttons open a pop-up window that allows the user to handle the files. The software recognizes MRI files in two formats, namely, *.MR* (MR scan image), and *.DCM* (DICOM). To accommodate the NIfTY formats, a separate module was developed that used a slightly modified AFNI workflow. The file-uploading interface is shown in Figure 4.8.

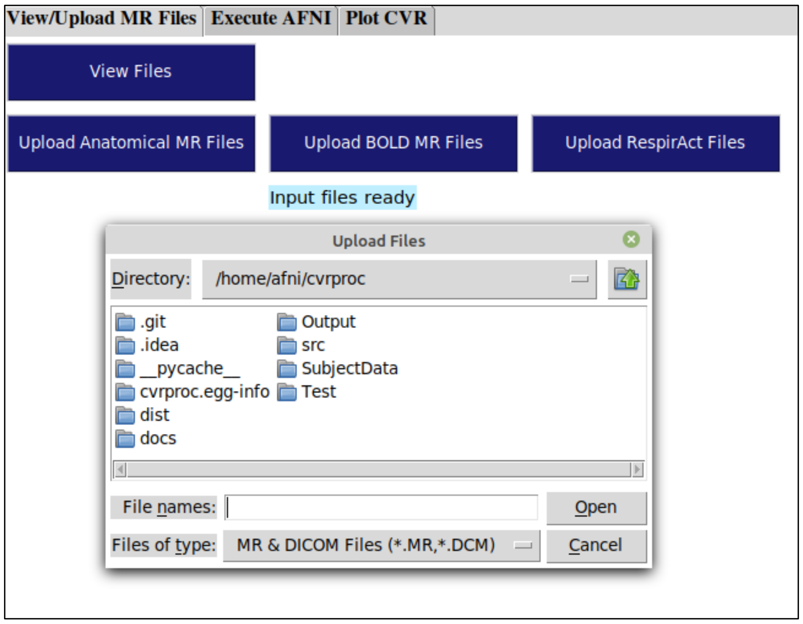


Figure 4.8: File uploading interface

4.6 Processing MRI files

Once the required input files are organized in the corresponding folders, the next step is to use AFNI modules to process the input DICOM files and produce the intermediate files that can be used for further processing. For processing the input files, the CVR software makes a series of system calls to execute the AFNI programs. For this research, the required AFNI programs were installed on a Linux-based virtual machine.

The first program is Dimon [115], which is used to monitor and organize the acquisition of the DICOM image files. It uses the meta-information contained within the DICOM file and uses the ‘image number’ to organize the sequence of the input files. The Dimon program allows for some additional options which can be used to realize specific functionality. Once the files are organized, the to3d program is used which creates 3-dimensional datasets from 2-dimensional image files. The AFNI documentation recommends using to3d as an additional option with the Dimon program.

The next AFNI program used is the 3dTshift [115]. It shifts the voxel (which is like pixels but in a 3-D space) time series in the input dataset to ensure that all the slices are aligned to the same temporal origin. The 3dvolreg program is used to register the volume of each 3-D sub-brick from the input dataset to the base brick. The 3dAutomask converts the 3-D and time series from the input dataset to produce brain-only mask dataset. The 3dmaskdump produces ASCII format file values from the input dataset, producing the voxels.1D file. This is the intermediate output file of interest which is used to extract BOLD values for the CVR calculations [115].

The software executes additional AFNI programs to further obtain more intermediate files. The complete series of AFNI executables used for the CVR workflow in this work are listed in Table 4.2.

Table 4.2: A step-by-step list of AFNI modules used to realize the CVR workflow [115]

Step No.	AFNI module	Description
1	Dimon	Reads and organizes the DICOM files in the specified folder
2	to3d	Converts the 2-D image files into 3-D AFNI format datasets
3	3dtshift	Shifts the voxel time series in the input dataset to ensure that the separate slices are aligned on the same temporal origin
4	3dvolreg	Registers the sub-bricks from the BOLD input dataset to the base brick (the original space)
5	3dAutomask	Converts the input EPI 3D+time or a skull-stripped anatomical input dataset into a brain-only mask
6	3dmaskdump	Uses the input dataset to write an ASCII file values. voxels.1D file is created
7	ANAT to3d	to3d function used for the anatomical dataset
8	3dAllineate	Converts the anatomical dataset to the MNI space using affine (matrix) transformation of space
9	3dAllineate	Converts the BOLD dataset to the MNI space
10	3dAutomask	Same as step 5 but using the MNI space datasets
11	3dmaskdump	Uses the input dataset to write an ASCII file values but for the MNI dataset
12	3dresample	Rewrite the dataset in the new orientation, with the new voxel size

Once these steps are completed, we have both the BOLD and the anatomical datasets in the same space (original space) with a resolution of 64x64x39 pixels. Some of the steps used for producing the voxels.1D file are repeated to generate the respective anatomical and BOLD datasets in the MNI space.

4.7 Plotting

Once the intermediate HEAD/BRIK files are produced in the output folder, the software allows for multiple CVR analyses for the same subject. Therefore, for a given subject ID, the steps required to obtain the intermediate HEAD/BRIK files need to be performed only once. Now, for each CVR analysis, a study name has to be entered. If an already completed study is being plotted again, the corresponding study name can be entered. The software displays the names of the already completed studies.

For creating the required plots, Matplotlib was used. It is a Python library that enables creation of static, animated, and interactive visualizations [110]. The CVR software uses two of these features- static plots of BOLD and PCO2, and interactive alignment of the two plots.

The main functions of the plotting modules identified were:

1) Import the BOLD data
2) Change the x-axis values to a new time series based on the repetition time
3) Import the PCO2 data
4) Resample the PCO2 data using a cubic spline interpolation on the same time series as BOLD
5) Plot BOLD and PCO2 on a dual y-axis plot, enabling visualizing both on the same plot
6) Allow the user to displace the PCO2 curve and align it with the BOLD plot
7) Allow the user to select an area to calculate the corresponding CVR values

4.7.1 Base Plot

The first step for the plotting module was to create the static plots of BOLD and PCO2 data. The data to be plotted was imported using the os modules [116] to read the corresponding files. Both these data are plotted on a common time series x-axis. However, given the nature of the study, the BOLD and PCO2 curves are not aligned naturally. There is usually a time delay between the time stamps when the external stimulus is administered and when the brain responds to compensate. As shown in Figure 4.9, the PCO2 curve is offset to the right for the two study sequences, step, and ramp. The secondary axis for creating a dual plot was accomplished using the Axes.twinx function [117].

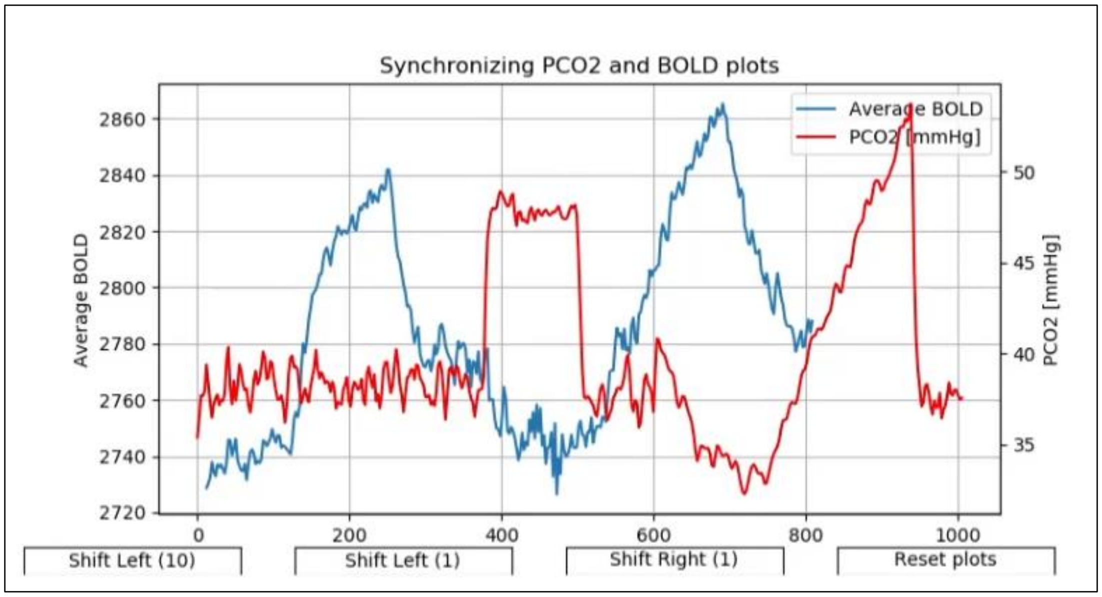


Figure 4.9: Static plots of the BOLD and PCO2 data on a common time scale. The shifting buttons that allow plot synchronization are placed below the plotting interface

4.7.2 Plot Synchronization

To calculate CVR, it is required that the two plots are aligned. For this synchronization, Matplotlib buttons were added to the GUI window, as seen in Figure 4.9. In all the use cases, the PCO2 curve needed to be shifted left. Therefore, two options were provided for left shifts. The plot could be shifted by 1 unit or 10 units depending on the requirement. In case of excessive shifting to the left, the shift right button shifts the PCO2 curve to the right by 1 unit. The reset button plots the base PCO2 plot in the raw form.

Once the two plots are aligned, the next step is to define an area of analysis. For example, for the analysis of the step sequence, the start and end points for the dataset are required. The software allows for defining the endpoints by drawing a rectangle over the plotting interface. This feature was realized using the rectangle selector widget in the Matplotlib library [118]. It allows the user to use the mouse button and movement to select the start and end points on the plot and saves the x- and y- coordinates for the four corners. The plotting interface was embedded in the application window, to the Plot CVR tab, using Matplotlib's backends module [119].

In the study shown in Figure 4.9, two sequences of the vasoactive stimulus were applied. The first sequence is a step response (highlighted inside the red rectangle). The red curve represents the vasosimulation, in the form of end-tidal PCO2. The blue curve represents the BOLD response to the applied step sequence. After a time delay, a ramp sequence is applied, which is plotted as the pointed curve following the step sequence.

The synchronized plots along with the area of interest selected (the step response) on the plotting interface can be seen in Figure 4.10.

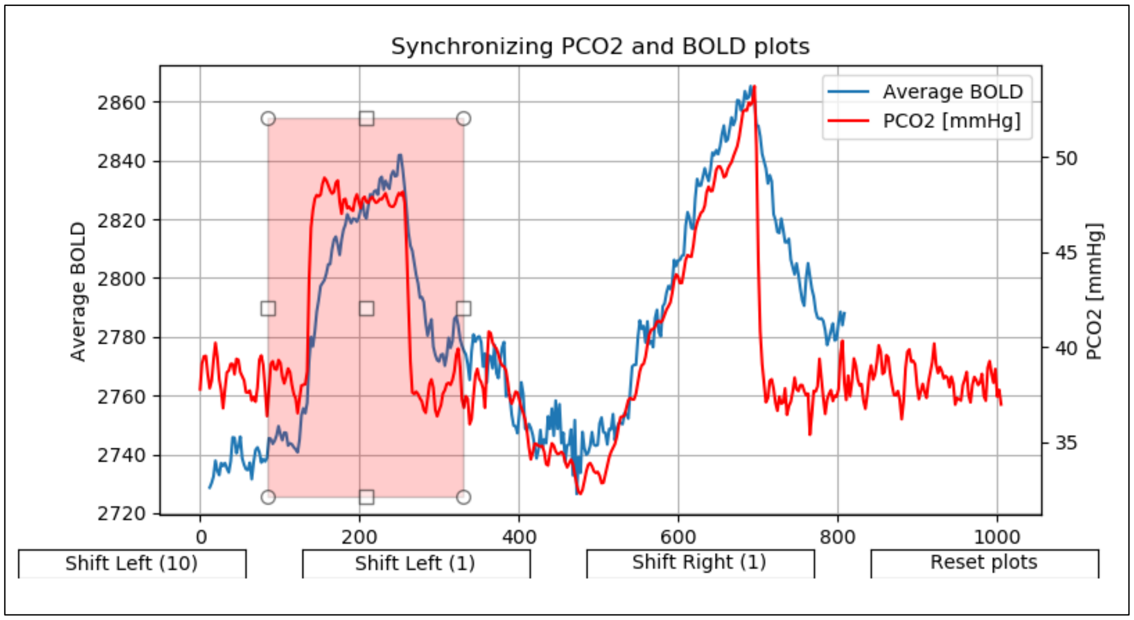


Figure 4.10: The interactive matplotlib interface for achieving the synchronization requirements. The user is required to define the area of analysis using a rectangular selector tool, shown as the red-colored rectangular area on the plot. The small squares and circles around the red rectangle allow the user to resize the selected area

4.8 CVR Calculations and Visualization

The final step of the CVR research workflow is to calculate the CVR values and plot them voxel-by-voxel across the brain. The software uses the scipy.stats module linregress to calculate the slope of the regression line [120]. The slope values for the different data points of BOLD and PCO2, along with the voxel coordinates are written in an output file.

Finally, the AFNI programs are used to visualize the CVR maps. This is shown in Figure 4.11. The interface shows the sagittal, coronal, and axial views of the CVR maps plotted over the anatomical images of the subject. As it can be observed in the CVR maps, the subject has reduced CVR response in the left hemisphere of the brain, indicated by the blue gradients. This indicates an irregular CVR response and highlights the presence of an underlying abnormality.

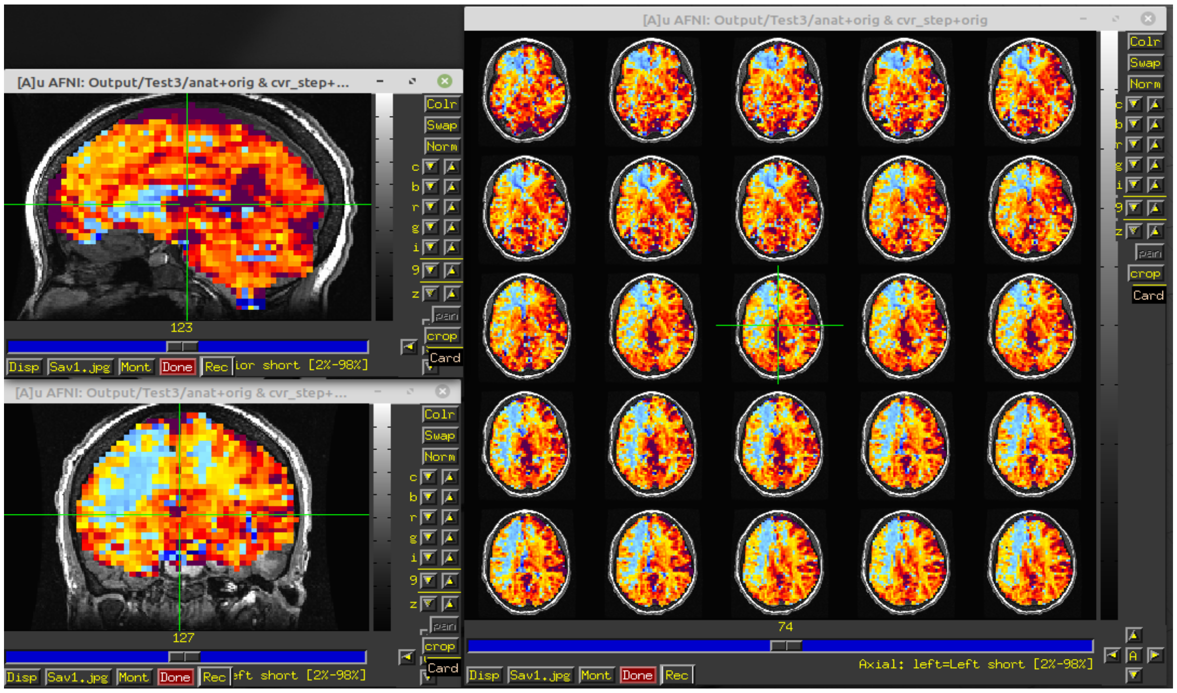

[A]u AFNI: Output/Test3/anat+orig & cvr_step+...
[A]u AFNI: Output/Test3/anat+orig & cvr_step+...
[A]u AFNI: Output/Test3/anat+orig & cvr_step+orig
Colr
Swap
Norm
crop
Card
123
127
74
Disp
Sav1.jpg
Mont
Done
Rec
Axial: left=Left short [2%-98%]

Figure 4.11: CVR maps produced using AFNI programs. The three views of the brain MRI images are shown: Sagittal (top left), Coronal (bottom left), and a 5 x 5 mosaic stack of the Axial images (right)

4.9 Software Validation

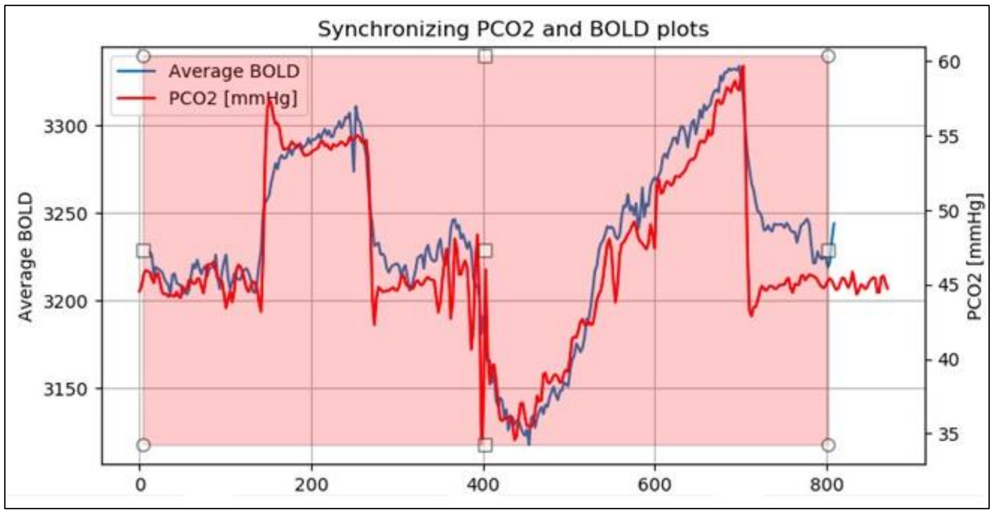
To validate that the software's workflow and calculations met the research standards, the CVR maps obtained from the software were compared with the ones obtained from the currently used processing tools (shell scripts and licensed software). The key aspects that needed validation were the interpolation process and plotting interface, and the final CVR maps. Two test cases were analyzed using the two methods and the results were compared. Step and ramp increases in the CO2 were applied to both these subjects.

4.9.1 Study 1

This study involved a step and a ramp stimulus administered over 800 seconds. The files were processed using the software and the static plots were created. A comparison of the plot generated using the software and that obtained from the shell scripts is shown in Figure 4.12.

A few differences were observed between the two figures. First, the x-axis scale used in the CVR software pertained to the normal time series. Whereas it represents the repetition time (TR = 2.4 s) on the other plot. Additionally, the y-axis values on the two plots use two different ways of representing the change in BOLD signals. The CVR software plots the change in average BOLD values, whereas the right image plots the percentage change in BOLD. Although these are visible differences, they do not affect the CVR workflow directly. The main objective of this plotting step is to synchronize the BOLD and PCO2 data and to define the endpoints of the area of interest.

The CVR maps obtained using the two software tools were plotted and compared. This is shown in Figure 4.13. For the given study, both maps look very similar with only minor differences. The regions of blue color are consistent in both, and the interpretability of the maps is the same.



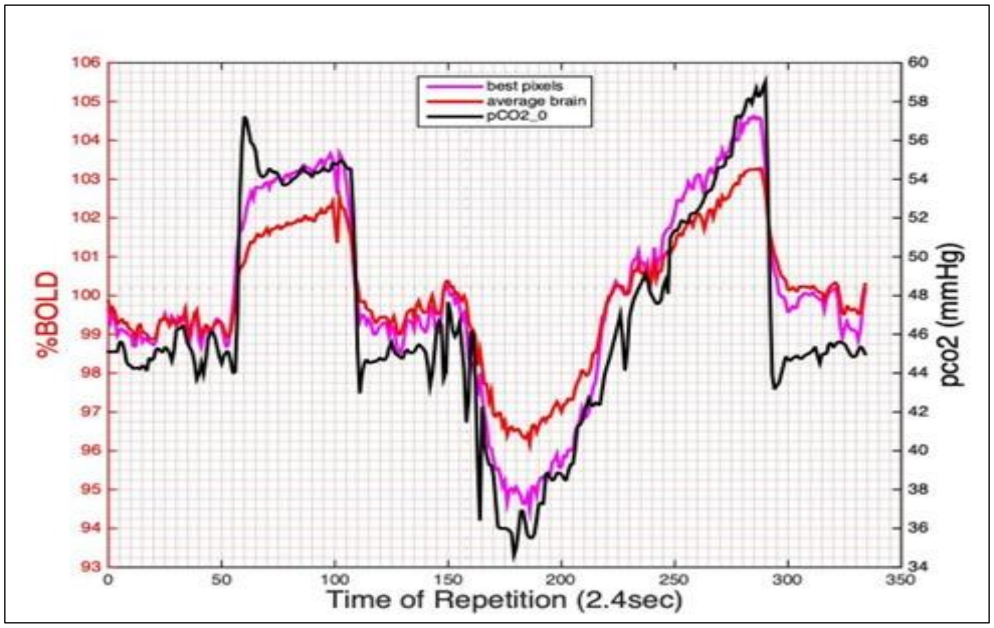


Figure 4.12: A comparison of the BOLD and PCO2 plot synchronization obtained using the CVR software developed in this work [top] and the current generation tools [bottom]

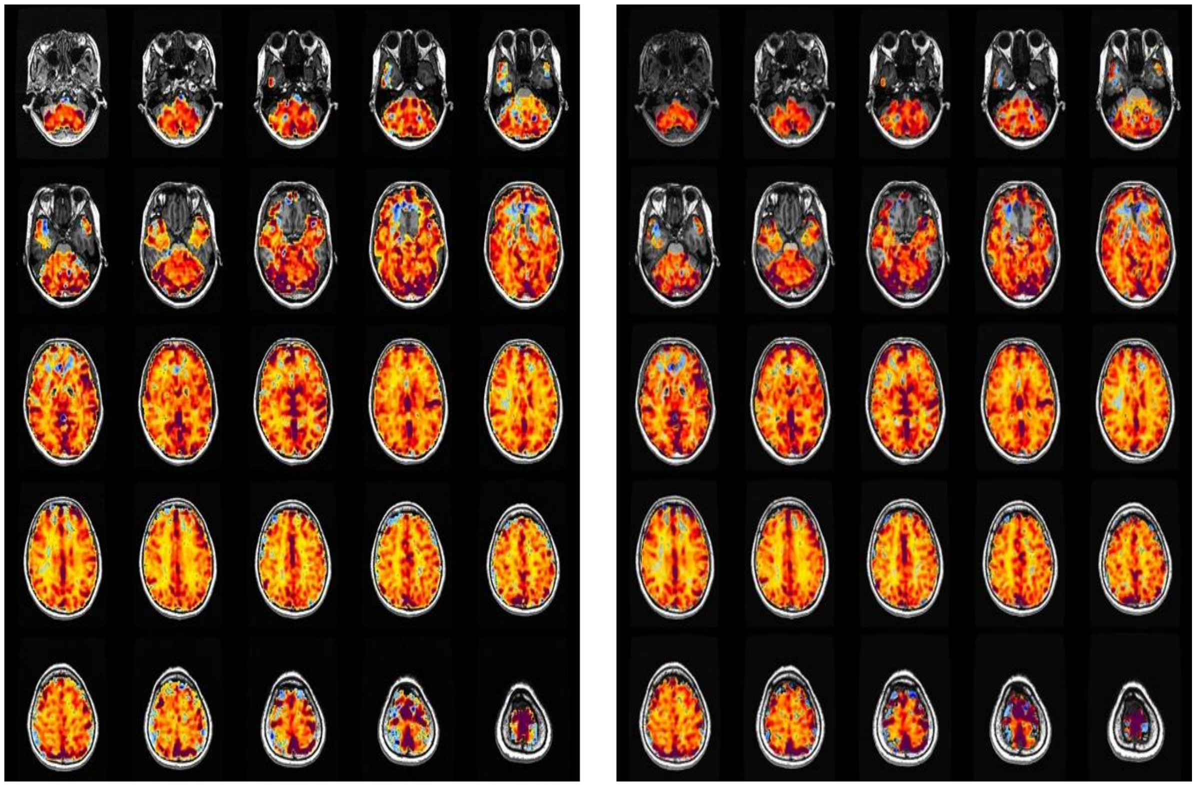
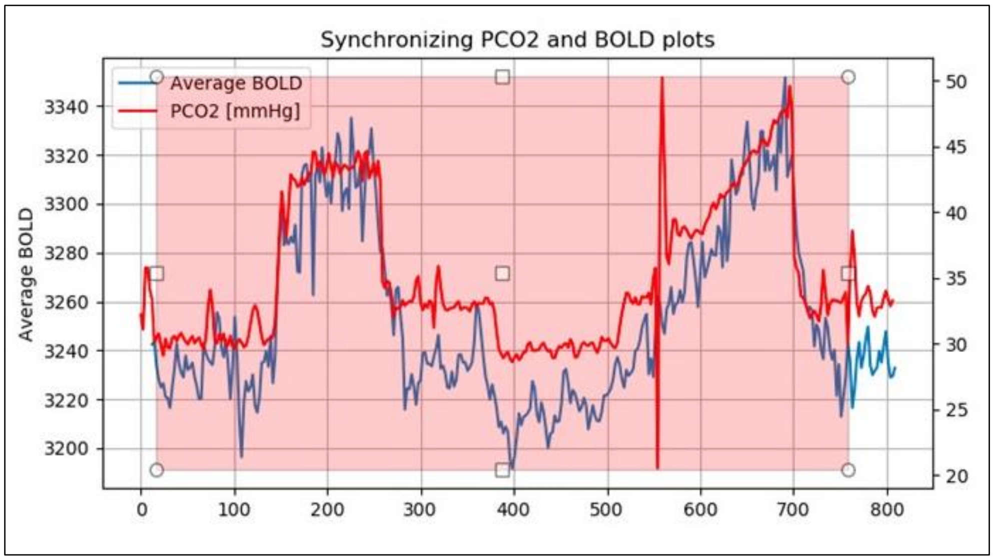


Figure 4.13: A comparison of the CVR maps of a healthy control subject produced using the CVR software [left] and the current generation tools [right]. As observed, there are very minimal differences in the two sets of maps

4.9.2 Study 2

The second study used to validate the software also consisted of a step and a ramp response. However, the data in this study had more noise as compared to study 1. The software successfully created the plot, which was interpretable and comparable with the plots developed using existing methods. Figure 4.14 shows a comparison of the two plots.



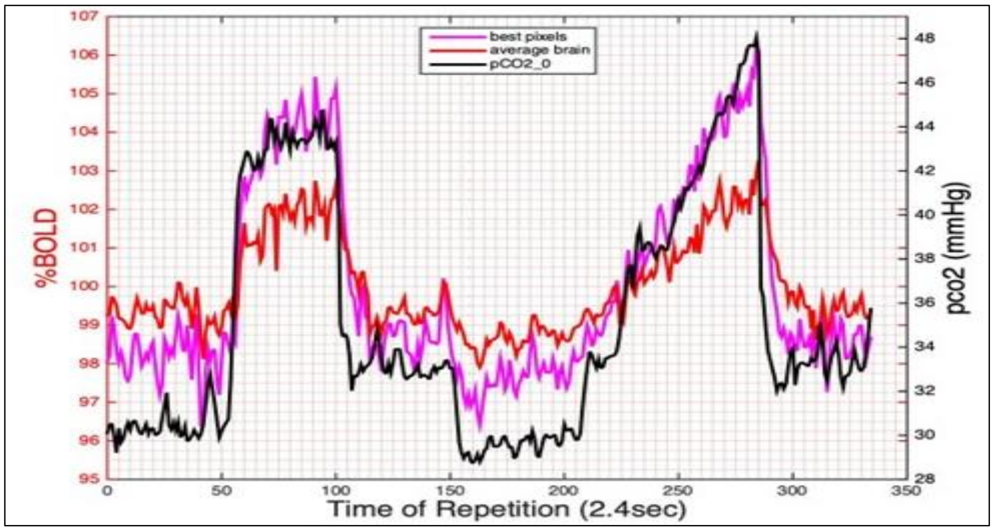


Figure 4.14: A comparison of the BOLD and PCO2 plot synchronization obtained using the CVR software developed in this work [top] and the current generation tools [bottom] for study 2

Once again, the corresponding CVR maps were visualized, as shown in Figure 4.15. As can be noticed, there were some differences between the two sets of plots. First, the maps produced using the CVR software do not smoothen the edges of the blue and red-colored areas. The reason behind this was identified as a difference in thresholding values set while the CVR maps were generated. Secondly, minor deviations can be observed in the spatial distribution of the blue areas, particularly noticeable in the fourth row. This was attributed to the difference in the sequence start and end values. The CVR software enables the user to manually drag and select the area of interest. In the current generation tool, the user manually enters the start and end values. In conclusion, the CVR maps still convey the same information, with negligible differences, therefore, validating the software’s outputs.

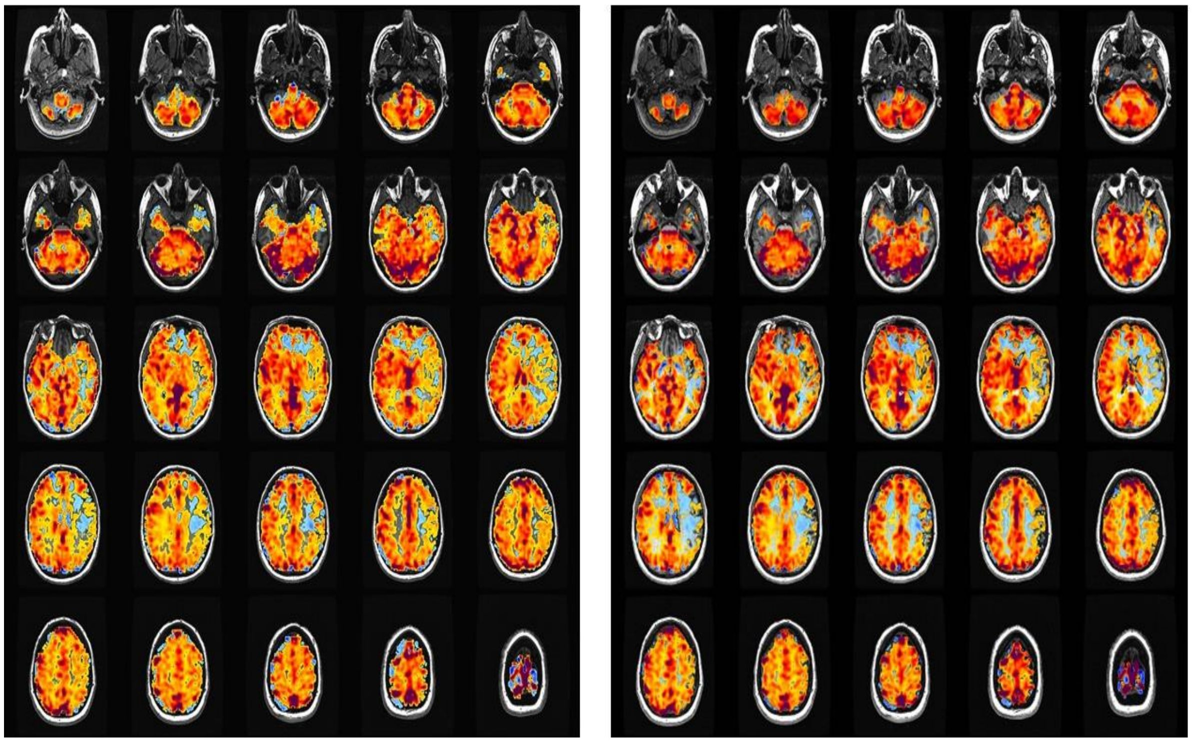


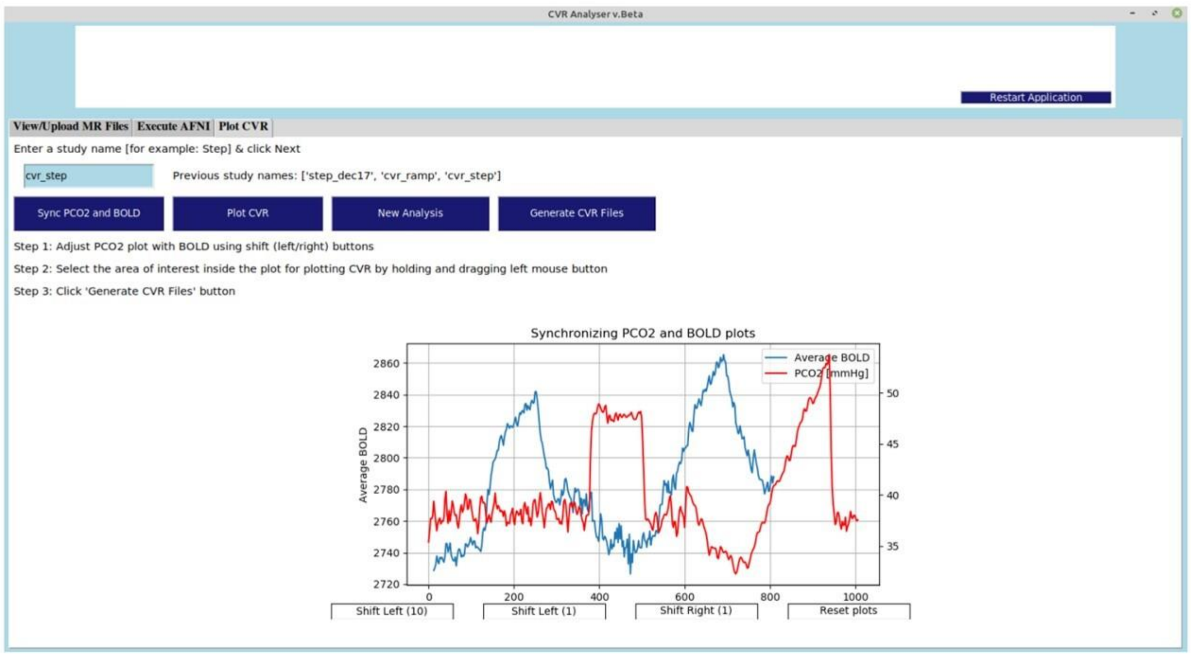
Figure 4.15: A comparison of the CVR maps of an unhealthy subject produced using the CVR software [left] and the current generation tools [right] for test case 2

4.10 Final Product

Once the software package was developed and tested on the local machine, it needed to be built, packaged, and published to a repository for distribution. Poetry, a Python dependency management and packaging tool were used for this purpose [88]. The packaged software file was uploaded to a private repository.

The CVR Processing Software developed in this work successfully accomplishes the CVR research workflow. It is a standalone software that runs on a virtual machine. The virtual machine already comes pre-installed with the required AFNI modules and Python libraries required to complete the CVR research. The CVR maps generated at the end of the workflow represent linear CVR. The final version of the proposed CVR software is shown in Figure 4.16.

A detailed software workflow is explained in Appendix A.


Enter a study name [for example: Step] & click Next

cvr_step   Previous study names: ['step_dec17', 'cvr_ramp', 'cvr_step']

Step 1: Adjust PCO2 plot with BOLD using shift (left/right) buttons

Step 2: Select the area of interest inside the plot for plotting CVR by holding and dragging left mouse button

Step 3: Click 'Generate CVR Files' button

Figure 4.16: The final GUI of the CVR software. This interface shows the plotting tab of the software, the plotting interface along with the interactive buttons to align the BOLD and PCO2 plots

Chapter 5

Convolutional Neural Networks for Screening SOD

In this chapter, the design and optimization of the convolutional neural network (CNN) architecture to classify CVR maps into healthy control subjects and unhealthy categories is discussed. CVR maps stacked in a 5x5 mosaic format were supplied to the networks for training and an empirical evaluation-based design methodology was implemented to optimize the network architecture and fine-tune the network's hyperparameters. The CNN is designed using the Tensorflow [64] and Keras [65] libraries. The section begins by describing the experimental setup, followed by an analysis of the input dataset, the pre-processing of the input data, and finally the network design methodology. Some common issues encountered while using our limited dataset, for example, overfitting and noise, are also discussed. Finally, transfer learning is implemented using some of the popular pre-trained models, and the results are analyzed.

5.1 Experimental Setup

The first topic to be discussed is the computational setup used to conduct the experiments. The design and optimization of the convolutional neural networks were conducted on the university lab server, physically located in the Maglev Microrobotics laboratory [121]. The server specifications are explained in Table 5.1. The same computational resources were used consistently for all experiments discussed in this chapter.

Table 5.1: Server specifications used for training the networks

Parameter	Specifications
Processor	Intel XII Gold 5218 CPU @ 2.30GHz (2 processors)
RAM	256 GB
System type	64-bit Operating System
Hard disk	3.62 TB

The Python libraries and tools used for the experimentations are listed below in Table 5.2.

Table 5.2: The different packages and libraries used for the experimentations of CNNs

Package	Version
Python	3.10.4
TensorFlow	2.9.1
Keras	2.9.0
Keras-Applications	1.0.8
Keras-Preprocessing	1.1.2
NumPy	1.22.4
Matplotlib	3.5.2

5.2 Data Analysis

The input images used to train the CNNs comprised 2-dimensional RGB images of the CVR maps of different slices of the brain viewed in an axial orientation stacked in a mosaic format of 5x5. The spacing between the axial CVR maps is set to 5 slices, meaning every 5th slice is plotted. The original resolution of these mosaic images was 1280 x 1280 x 3 pixels. The CVR maps used in this work were generated in the standard space. The CVR maps generated in the Montreal Neurological Institute (MNI) standards were not considered due to the shortage of such samples. The dataset is obtained from a diverse patient cohort from studies conducted by our sponsoring institute. The input data was labeled by a team of experts and belonged to two classes: healthy subjects, and patients with the steno-occlusive disease (SOD).

Figure 5.1 shows two sample images belonging to each class. Note the difference in the distribution of the blue gradients across the brain regions, which represents decreased CVR in those regions. And a decreased CVR highlights an abnormal vascular response to the stimulus. The spatial distribution of the blue gradients is the key difference between a healthy and an unhealthy subject, and this is the feature attribute that needs to be learned by the deep learning models to successfully classify the maps.

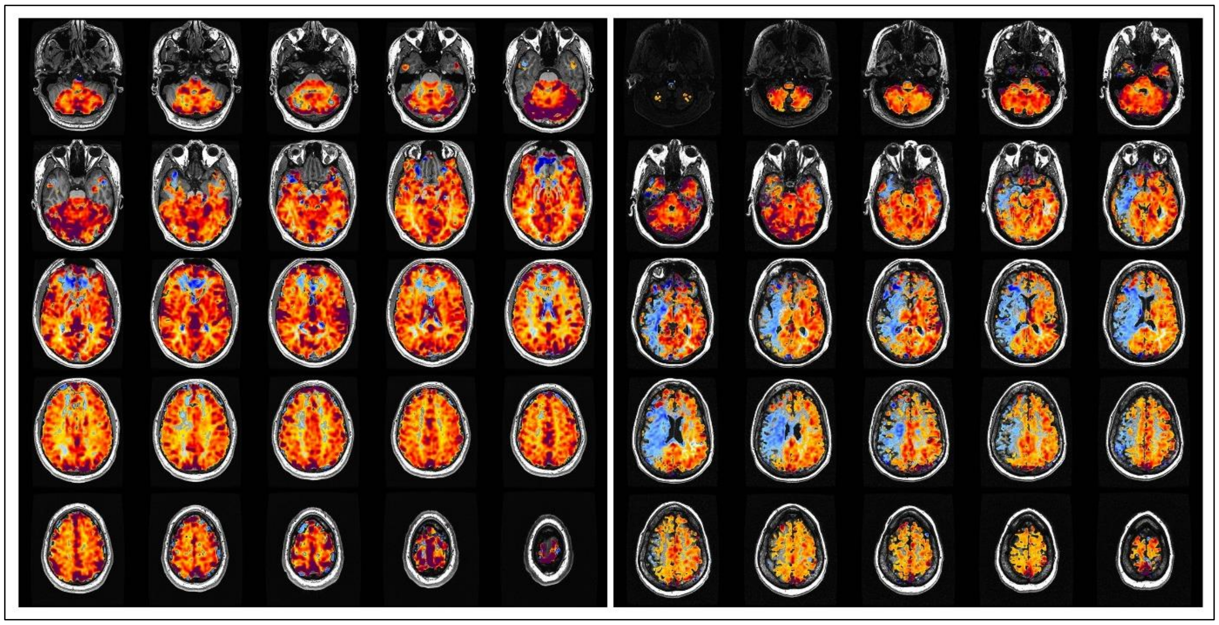


Figure 5.1: 5x5 mosaic images of the CVR maps used as the input data for training the convolutional neural networks. CVR maps in images (A) [left] and (B) [right] were measured in a healthy subject, and a patient with the SOD, respectively

The clinical procedure used to obtain the CVR maps using the prospective end-tidal targeting (RespirAct RA-MR device) [122] is novel, and therefore the data availability was limited. The original dataset consisted of mosaic images of 68 healthy subjects and 163 unhealthy patients. We consider this a small dataset given the small number of trainable samples.

As discussed in section 2.3.2, in this work, image augmentation techniques and transfer learning were implemented to address the challenges of using a small dataset. The augmentations applied are discussed in section 5.3, and the pre-trained models used for transfer learning are discussed in section 5.7.

5.3 Data Pre-Processing

The first step to using the medical dataset was to anonymize the file names and remove any patient information attached to the images. This was accomplished by developing a Python script that copied only the CVR JPEG images and renamed the files to remove any patient-specific label or information. The program is explained in Appendix-B, section B.3.

The original dataset, consisting of 68 healthy subjects and 163 unhealthy patients, was split into three categories, as described in section 2.3.2. The training dataset is used for training the model where the model learns the features and patterns in the input data. The validation dataset is distinct from the training set and is used to validate the model performance during training and update the network parameters. This dataset is used to validate if the model is successful in learning important parameters and indicates if the hyperparameters need to be tuned. Finally, the test set is used to provide an unbiased performance metric for the model [123]. The dataset split is shown in Figure 5.2.

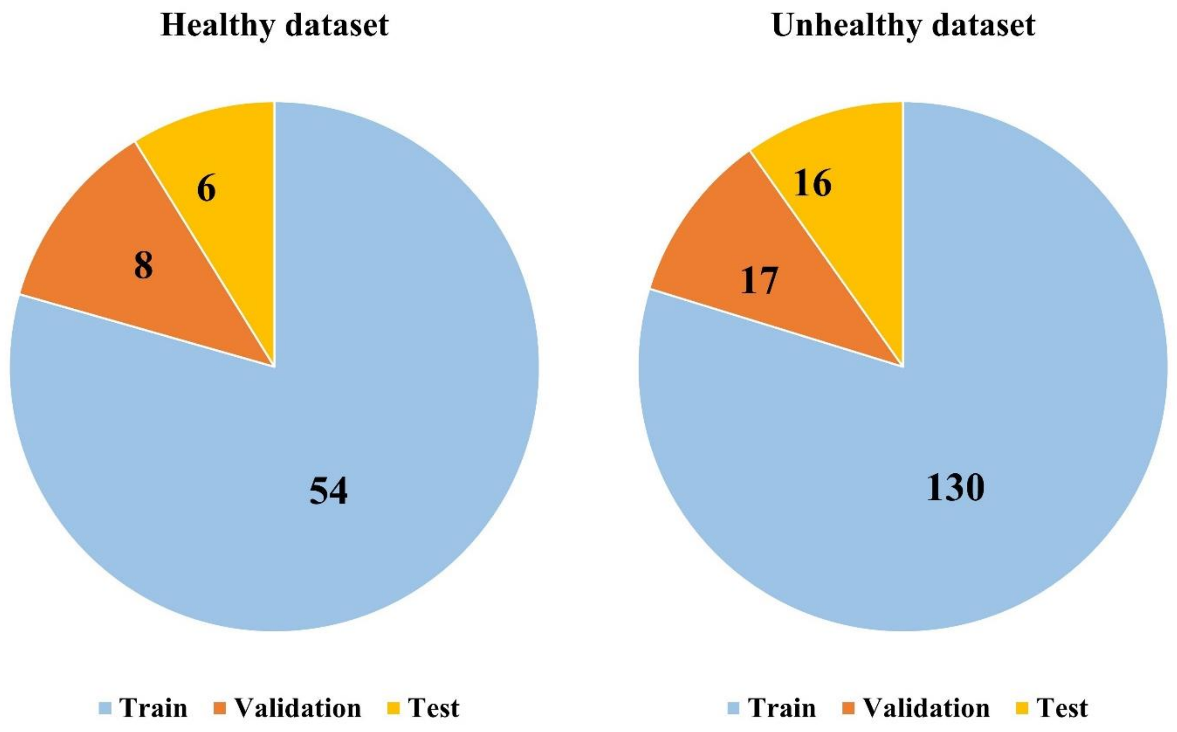


Figure 5.2: A comparison of the healthy and unhealthy datasets, and the distribution of the original dataset into train, validation, and test sets

5.3.1 Data Augmentation

To address the shortage of the number of trainable samples and the data imbalance between the two categories, data augmentation techniques were implemented in the training set. Additionally, to accommodate real-world test cases, small augmentations were applied to the validation set. The spatial resolution and orientation of the input mosaic images needed to be preserved to maintain a standardized CVR analysis protocol (for example, the slice number and its relative placement on the mosaic). Therefore, the augmentations applied were small-scale and limited to the extent of not affecting the original attributes. The augmentation operations used included rotation (range: 0-2), shearing (range: 0.2), modifications to the brightness (range: 0.3-1.0), and horizontal flips. A few participating institutions used horizontally flipped mosaic images for assessing CVR maps and therefore the horizontal flips were included to extend the model capabilities to such studies. The test set consisted of the images from the original dataset, without any modifications. The final sample size used for training and validation is shown in Table 5.3.

Table 5.3: The final train, validation, and test datasets after applying the image augmentations

Dataset	Healthy	Unhealthy
Train	594	650
Validation	168	186
Test	6	16

5.4 Network Design Methodology

5.4.1 Key Definitions

Some of the terminologies used in this section are defined below [124].

1) Model parameters: The internal network parameters that are updated during the learning process. These primarily include the model's weights and biases.

2) Hyperparameters: The network variables that are adjusted (manually or using an automated service) during successive trials of network training.

3) Batch size: Number of samples used in one iteration of the training. An iteration is a single update of the model's parameters.

4) Epoch: A complete training pass performed on the complete training dataset. The total number of training iterations in an epoch is calculated by N/batch size, where N is the total number of training samples.

5.4.2 Network Design

The design methodology used in this work consisted of an empirical evaluation-based optimization strategy which involved fine-tuning based on the model performance. The idea was to start the experiments with a simple and shallow network and optimize the network configuration based on the results. This small network was the baseline and the network hyperparameters were fine-tuned to develop more optimized models. Some of the popular pre-trained models were also investigated to implement the concept of transfer learning. The results of all the networks were compared and the most optimal network is proposed.

The hyperparameters investigated during the fine-tuning of the customized CNNs are listed in Table 5.4.

Table 5.4: The hyperparameters investigated to optimize the CNN architecture

Hyperparameter	**Values**
Input image resolution	256 x 256, 512 x 512 pixels
Batch size	8, 16, 32, 64
Batch Normalization	Default settings
Dropout regularization	No dropout; rate = 0.2, 0.5
Network depth (number of layers)	2, 3 hidden layers
Network width (filters and nodes in each layer)	16, 32, 64, 128
Learning rate	0.001 (default), 0.0005
Kernel size	3x3, 5x5, 7x7

5.4.3 Customized CNNs: Common Network Settings

To train the neural networks considered in this study, the Adam optimization algorithm [125], which is an extension of the stochastic gradient descent (SGD), is used. For all the models investigated in this study, a Rectified Linear Units (ReLU) activation function [59] is used with the convolution layers. The convolution layers are followed by a 2-dimensional max pooling layer [126] with a pool size of 2x2, and a flatten layer [127]. The flatten layer flattens the multi-dimensional input tensor into a single dimension. The pooling operation was discussed in section 2.4.1.3. The hidden fully connected (FC) layers use a ReLU activation function, and the output fully connected layer uses a Sigmoid activation function. A binary cross-entropy loss function is used to compute the cross-entropy loss between the true and the predicted outputs.

The networks were trained for up to 200 epochs. Early stopping callback is used while training the model. It stops the training for the given network when the specified metric does not improve over a specified epoch. Two arguments were used in this work which is monitor and patience. The metric to be monitored is set using the monitor argument, and patience defines the number of epochs without improvement when the model training can be stopped [128]. In this work, the validation loss (val_loss) was monitored, with patience of 10 epochs.

5.4.4 Convolution Operation: Feature Extraction

This sub-section explains the feature extraction process of the convolution layers. The main learning objective during feature extraction is to identify the key parameters from the image. Convolution layers do this by capturing the spatial relations in the image using the filter. Additionally, these layers reduce the number of trainable parameters while still preserving the important spatial relations.

Figure 5.3 shows the overall process of feature extraction when one of the samples was fed to a convolution layer with a 3x3 kernel. A detailed explanation of the process is explained below.

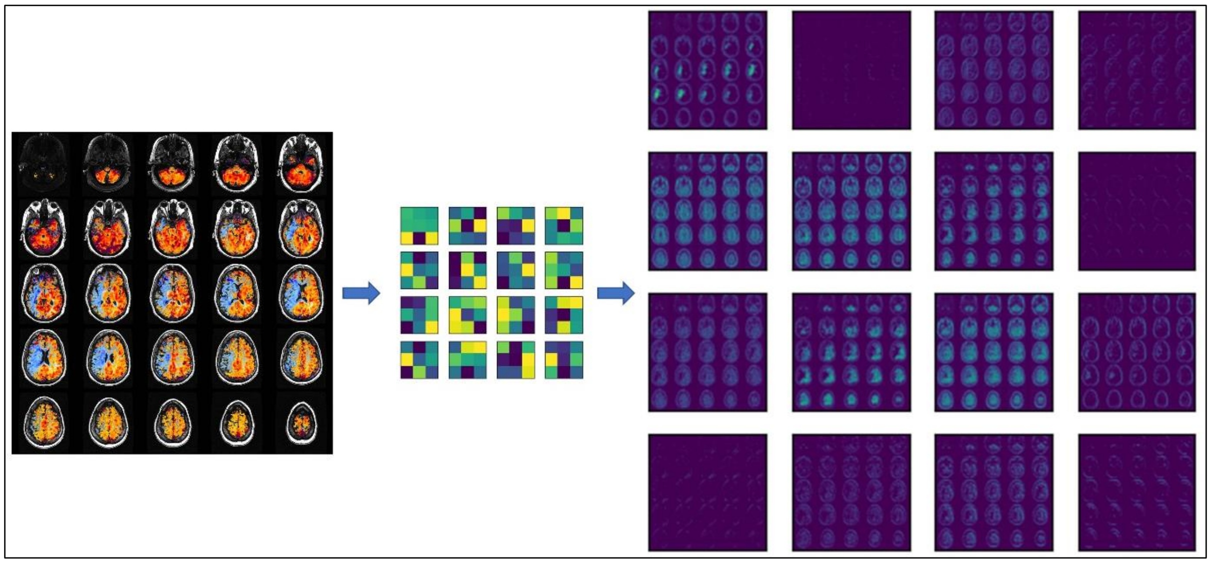


Figure 5.3: A set of feature maps produced after the convolution operation using a 3x3 kernel. Note how the convolutional layer preserves the spatial resolution of the input image

Consider a sample input mosaic image of the CVR maps as shown in Figure 5.4. When passed through a convolution layer, features like the edges, the corners, the boundaries, and the color gradients are identified. This can be observed in Figure 5.5. The figure shows four of the feature maps obtained after the convolution operation, using a convolution layer of 16 filters and a kernel size of 3 x 3, and a stride of 1. The ReLU activation function was used. The plots are obtained using Matplotlib, using the ‘Viridis’ color scale.

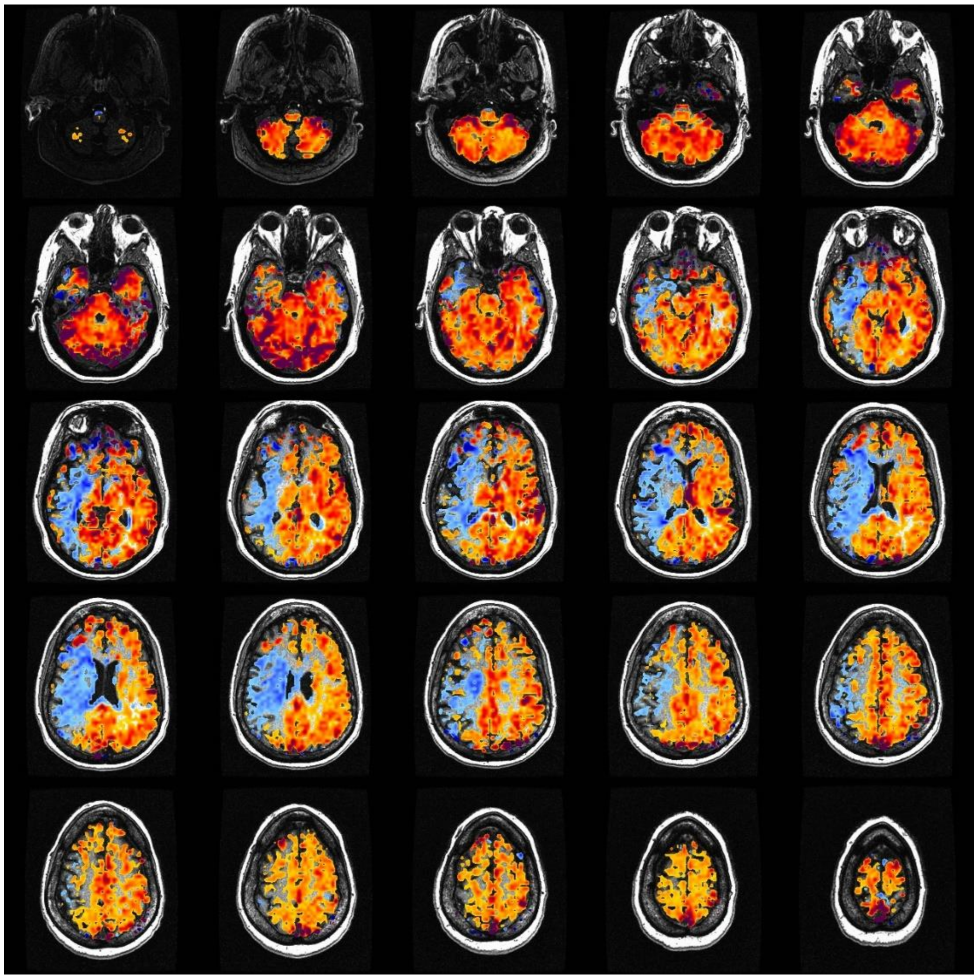


Figure 5.4: A sample input image belonging to an unhealthy patient. Note the spread of blue gradients on the left hemisphere of the brain

Notice how the different feature maps represent a specific representation in Figure 5.5. For example, the map on the top left establishes boundaries between the blue and red color gradients. And the map on the bottom right registers the location and spread of the blue color gradients. As discussed in section 5.2, the main attribute that discriminates a healthy CVR map from an unhealthy one is the spatial distribution of the blue color gradients. And as can be observed by the feature maps, the convolution layer successfully highlights this attribute in some of its representations.

These attributes are then associated with the labeled class by the fully connected layers when it tries to learn to discriminate between the two classes. Therefore, when these feature maps are passed on to the next layers, the corresponding relationships are used to calculate the probability of the image being healthy or unhealthy. Based on this discussion, the network will significantly improve its performance if a wide variety of CVR maps are used for training.

Figure 5.6 shows four additional samples from the unhealthy dataset to highlight the variations in the spatial distribution of the blue color gradients. When convolution layers with 16 filters are used as a feature extractor, multiple representations can be extracted from these images. And as the convolution layer passes these representations to the next layers, there are more attributes associated with the unhealthy class (and likewise the healthy class). This helps in improving the generalization capabilities of the network and is expected to improve the model's performance on the test set.

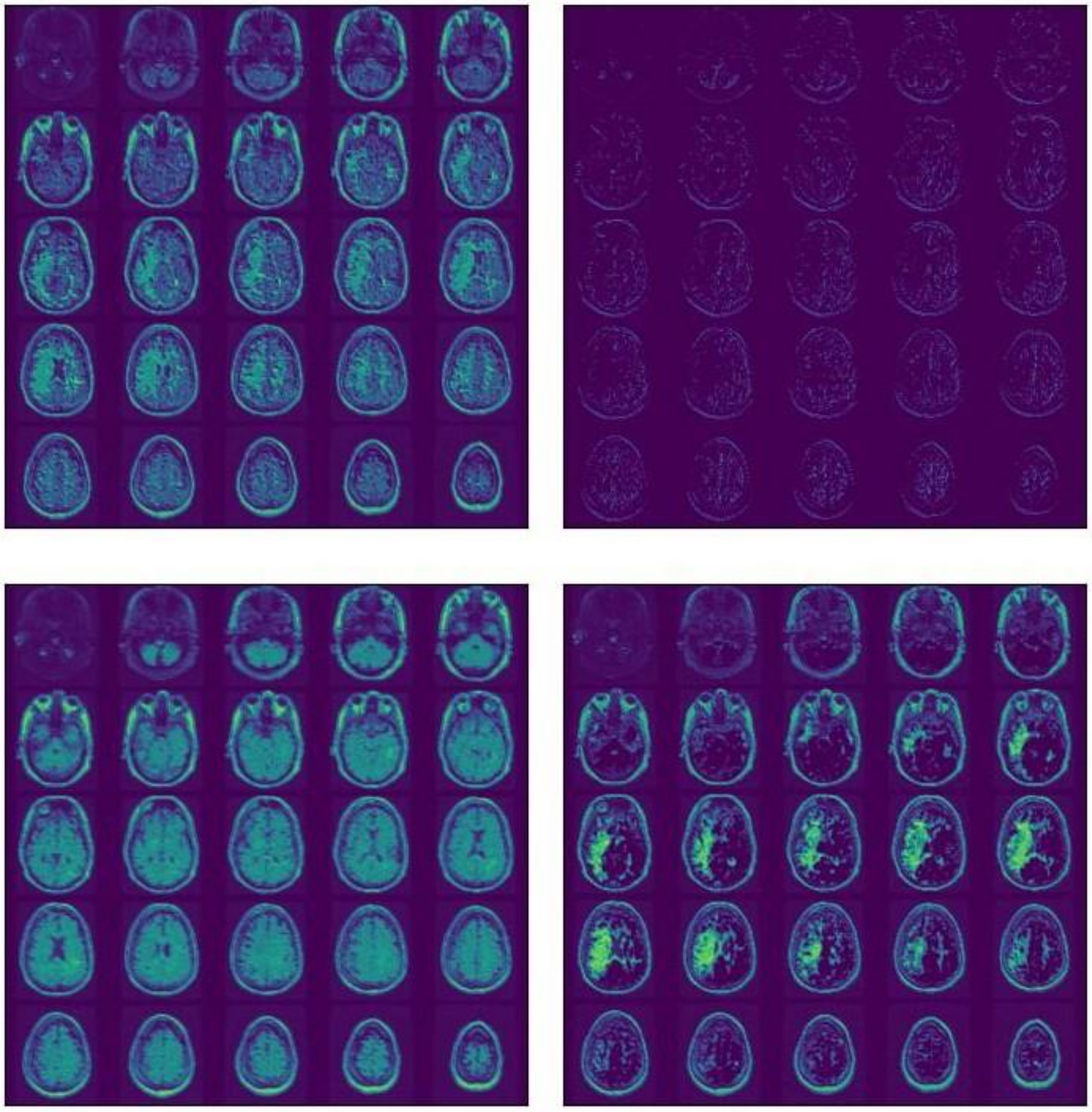


Figure 5.5: Four feature maps obtained by feeding the sample input image into a convolution layer of 16 filters with a 3 x 3 kernel size. The feature maps are plotted using the 'Viridis' color scale in Matplotlib. Observe how each of the feature map highlights a different attribute from the input image. For example, in the top left, the boundaries between the blue and red color regions are established. The top right image presumably highlights the textures created by the color distribution. The bottom right image highlights the spatial distribution and intensity of the blue gradients

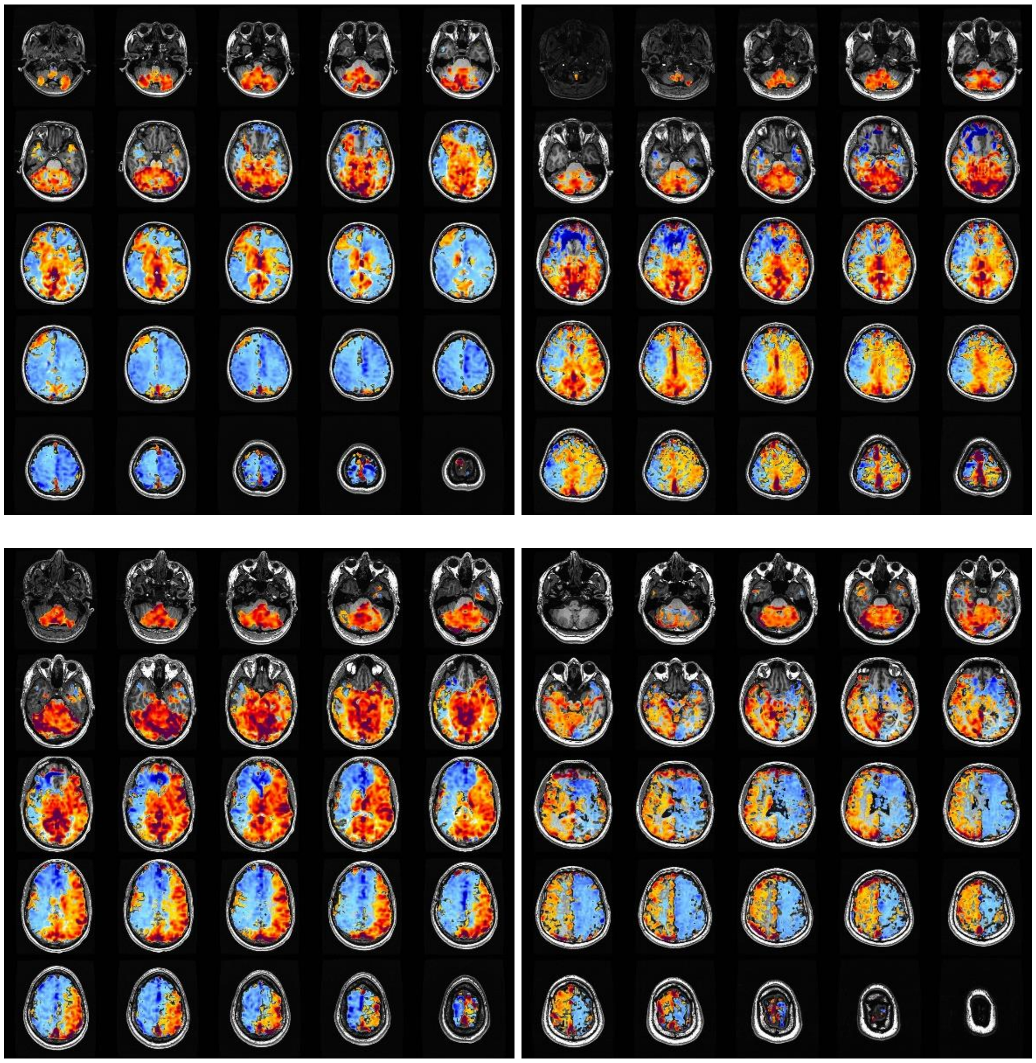


Figure 5.6: CVR maps belonging to the unhealthy category. The variations in the distribution of the blue color gradients across the four samples can be observed. The feature extractor highlights these different spatial locations of the blue gradients, and the feature attributes are used to label them accordingly

5.5 Baseline CNN Model

As discussed earlier, the baseline model was a simple network. It consisted of an input layer, a convolution layer with 8 filters, followed by a fully connected layer with 8 neurons, and an output fully connected layer with 1 neuron.

The baseline model was trained with two input image resolutions, 256 x 256 and 512 x 512 pixels, as the first set of experiments. The results obtained from training the baseline model indicated that the input image resolution after downsizing had an insignificant effect on the training and validation accuracy. However, the computational resources, specifically the training time and the memory, required were higher for 512 x 512 pixels. Additionally, the baseline model was able to successfully retrieve key spatial information from the 256 x 256 resolution images. Therefore, it was decided to conduct the experiments using a 256 x 256 input image resolution.

The baseline model achieved a poor performance on the training and validation sets, achieving an accuracy of 52.3% and 52.56%, respectively, after training for 54 epochs. There were signs of overfitting on the training data as the binary cross-entropy loss for the validation set began to increase with the training cycles. To address the overfitting, a dropout layer [61] was added after the fully connected layer (with 8 nodes) with dropout rates of 0.2 (dropping 20% of the input nodes in the hidden layer) and 0.5, respectively. A dropout of 0.2 did not improve the overfitting, however, increasing it to 0.5 showed improved results.

5.5.1 Dropout Regularization

To address the issue of overfitting observed in the baseline model, the dropout rate used for the fully connected layer was increased from 0.2 to 0.5, and model 2 was conceived. The optimized model with a 0.5 dropout rate after the fully connected layer achieved a training and validation accuracy of 72.77% and 90.90%, respectively, after training for 21 epochs. As compared to the baseline model, model 2 demonstrated improvements in the validation loss.

To extend the experimentations of dropout, an additional dropout layer was added after the convolution layer. The dropout rate was set to 0.5. This further improved the network’s performance on both the training and validation sets.

5.5.2 Batch Normalization

Batch normalization, or batchnorm, is an effective normalization technique used between the layers of a neural network to standardize data, and therefore, reparametrize deep networks. It essentially applies a transformation to normalize the inputs supplied to a layer. It significantly accelerates the learning process of a network and may improve the overall model performance through a small regularization effect [129].

5.5.3 Batch Size

The next hyperparameter investigated was the batch size. The batch size represents the number of samples fed into the model before the model is updated. Results produced using a small batch size of 8 demonstrated fluctuations (noise) in the validation accuracy. Four batch sizes were experimented with model 2: 8, 16, 32, and 64. It was observed that an increase in the batch size reduced the noise, improving the model performance.

The experiments conducted agreed with several studies that have investigated the effects of batch size on the performance of CNNs [130, 131]. In this study, the models are trained with a batch size of 32 based on the experimental results and in agreement with the recommendations from Bengio [132].

5.5.4 Network Performance

The main findings and network performance obtained from the sensitivity studies using the baseline network architectures are summarized in Table 5.5. As demonstrated by the performance of the Baseline-3 model, adding dropout layers after both the convolution and the fully connected layer improved the model performance, and the accuracy obtained on both the training and validation sets demonstrated significant improvements. Adding a batch normalization layer after the convolution layer further improved the results, however, the validation loss curve was not stable. Using two batchnorm layers one after each of the hidden layers (Baseline-5) produced the best results. The network details like the architecture, weights, and state of the trained models were saved using the Keras save and load model Application Programming Interfaces (APIs) [133].

Table 5.5: Comparison of model performance in terms of training and validation accuracy and loss. In the architecture, conv*X* represents a convolution layer with *X* filters, dense*X* represents a fully connected layer with *X* nodes, drop*X* represents a dropout layer with a dropout rate of *X*, and BN represents a batch normalization layer. Accuracy is on a scale of 0-1, 1 representing a 100% accuracy

Model Name	Architecture	Epochs	Train Accuracy	Train Loss	Validation Accuracy	Validation Loss
Baseline-1	conv8, dense8, dense1	54	0.5231	0.69	0.5255	0.70
Baseline-2	conv8, dense8, drop0.5, dense1	20	0.7277	0.49	0.9091	0.37
Baseline-3 (*Model-1*)	conv8, drop0.5, dense8, drop0.5, dense1	16	0.8977	0.21	0.8835	0.29
Baseline-4	conv8, BN, drop0.5, dense8, drop0.5, dense1	40	0.8985	0.31	0.9176	1.89
Baseline-5 (*Model-2*)	conv8, BN, drop0.5, dense8, BN, drop0.5, dense1	42	**0.967**	**0.06**	**0.97**	**0.09**

The baseline-3 model achieved high accuracy and therefore the concept of using a dropout layer after the convolution layer was of interest in this study. This model will be regarded as Model-1.

The performance of the network with two batchnorm layers (Baseline-5) on the training and validation sets is shown in Figure 5.7. As can be observed, the accuracy plots begin to converge as the model continues training. This model will be regarded as Model-2, representing the best performing model from the baseline experiments.

5.5.5 Test set

To test the generalization capabilities of the Baseline-5 model, the saved model was used to classify the CVR maps in the test set [6 healthy samples, 16 unhealthy samples]. The model accurately classified all the healthy samples as healthy. However, the model inference for the unhealthy dataset was 62.5% accurate with 10 correct and 6 incorrect predictions. Therefore, the overall prediction accuracy was 72.7%.

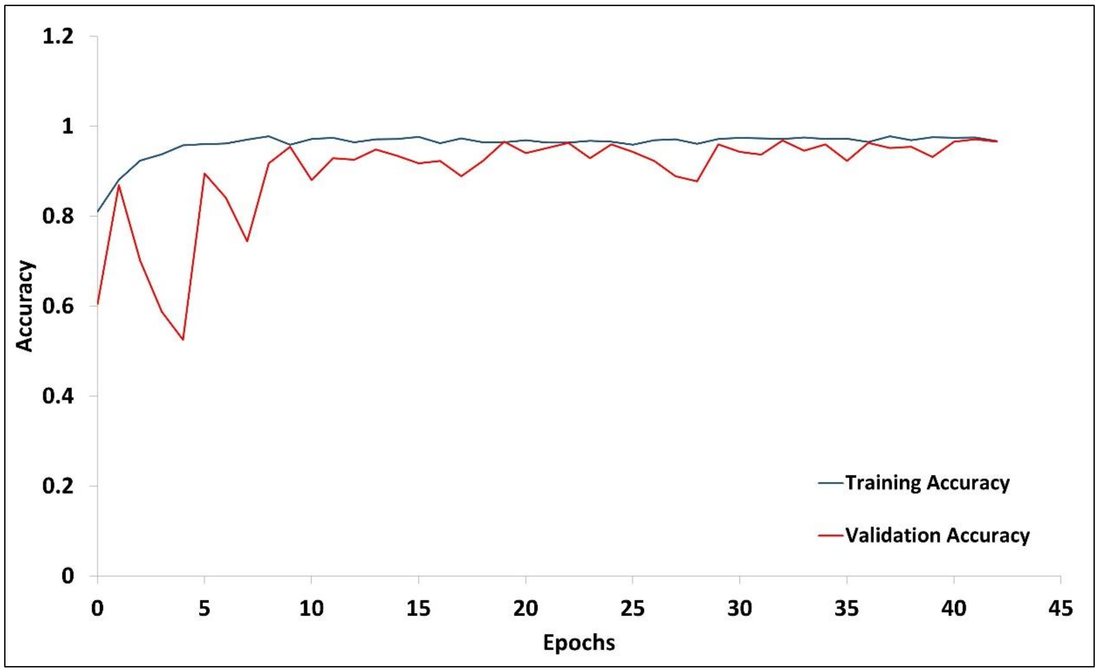


Figure 5.7: The accuracy plots obtained from training the network with two batch normalization layers after the convolution and the fully connected layers (Baseline-5)

5.5.6 Modeling and Execution Scripts

All the experiments are conducted using server processing units (CPU) and the GPU is disabled. A sequential model is defined, and the different layers are added using the Keras model.add() function. The model is then compiled using the adam optimizer, a binary cross-entropy loss, and the metric to be monitored is set as accuracy. The program used to define the model is presented in Appendix-B, section B.4.

To train the networks with TensorFlow, the ImageDataGenerator [134] is used. The ImageDataGenerator module creates tensor image data from the original datasets that can be used with TensorFlow. It allows real-time data augmentation. This work used a rescaling argument to normalize the input images. A ratio is defined for the rescaling that is multiplied by each pixel of the image. A ratio of 1/255 is used to rescale the pixel values from the range of 0-255 to 0-1. The module used to begin the training is explained in Appendix-B, section B.5.

5.6 CNN Model Optimization

It was evident from training the baseline models that a training accuracy of around 90% was successfully achieved, meaning the networks were performing well with the training dataset, also indicating overfitting to the training set in some experiments. Additionally, based on the model performance on the test set, there was an opportunity to improve the network's performance on unseen data. This implied that the generalization capabilities of the networks were the main area to be evaluated. Therefore, a key emphasis was placed on validation accuracy. Setting the performance of the best performing baseline model (Model-2) as a benchmark, the network performance in terms of training and validation accuracy and loss were analyzed for other CNN architectures, and the other network hyperparameters were fine-tuned to improve the model performance.

5.7 Network Width and Depth

The network width and depth were investigated next. These experiments included changing the number of filters and nodes in the convolution layer and fully connected layer to change the network width, respectively, and adding layers to change the network depth. The layers included dropout layers, batch normalization layers, convolution layers, and fully connected layers. Two sets of experiments regarding the network depth were conducted for a fixed network width- (a) adding dropout layers, and (b) adding batch normalization layers.

In Model-3, the filters used in the convolution layer were increased from 8 to 16, and the nodes in the hidden fully connected layer were increased from 8 to 16. The architecture comprised of an input layer, a convolution layer with 16 filters, followed by a hidden fully connected layer with 16 nodes, and an output fully connected layer. The experiments involved adding dropout layers after the convolution layer and the fully connected layer, one at a time, followed by adding two dropout layers. Similar experiments were conducted for batch normalization.

The network's performance declined as compared to Model-2. The network could only achieve an accuracy slightly upwards of 50% during training and validation for all but one configuration. However, the model performance was significantly better when two batch normalization layers were added. The training and validation accuracy, in this case, were 99% and 92%, respectively. A comparison between network performance in terms of validation accuracy between the two architectures is shown in Figure 5.8. Both the networks used dropout layers with a dropout rate of 0.5 after the hidden layers and one of these used batch normalizations before the two dropout layers.

Like Model-3, a series of experiments were conducted using a wider hidden fully connected layer, increasing the nodes from 16 to 32 (Model-4). The convolution layer was unchanged, with 16 filters. The best model performance achieved using this network configuration for the training and validation sets was 85% and 91%. Adding batch normalizations after the two hidden layers improved the network's performance on the training set but decreased the validation accuracy.

Continuing the experimentation, the number of filters in the convolution layer and the nodes in the dense fully connected layer were changed to conduct a sensitivity study. The combinations included 32, 64, and 128 filters and nodes.

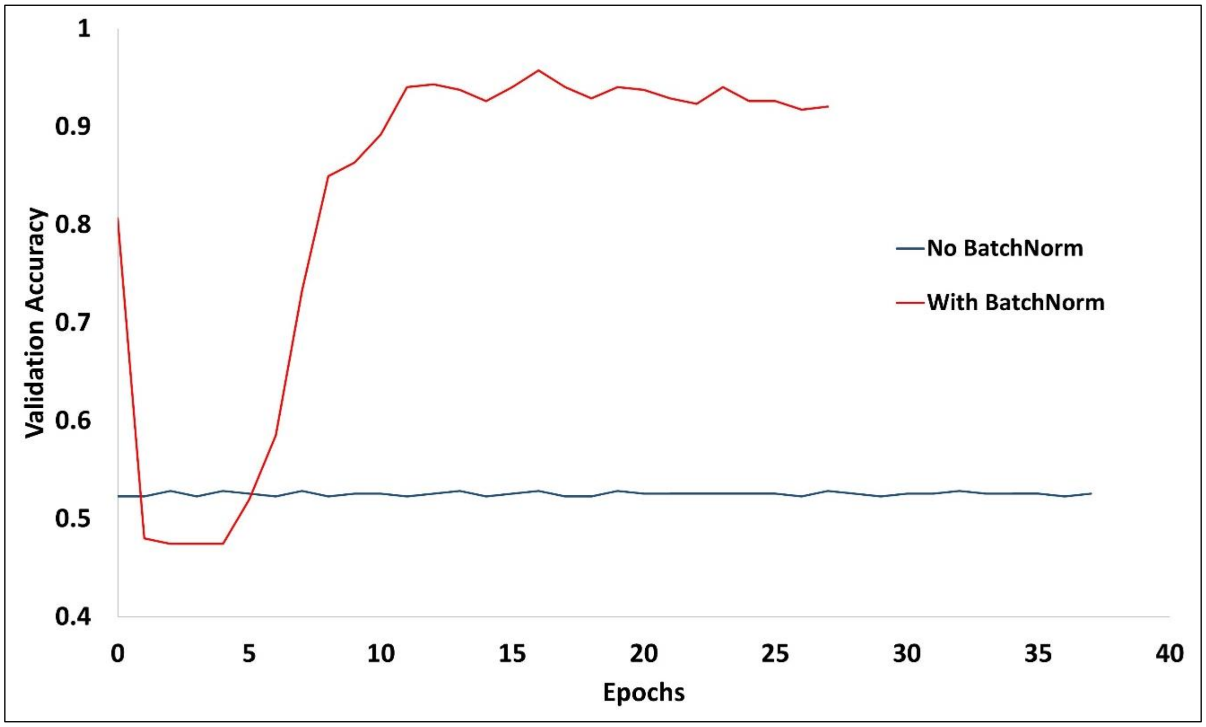


Figure 5.8: A comparison of validation accuracy obtained with and without batch normalization using the architecture of Model-3

Similar to the findings from the baseline experiments, adding a dropout layer with a dropout rate of 0.5 after the convolution layer produced improved results. Batch normalizations improved the performance of the smaller networks (8 and 16 filters [and nodes]), however, deteriorated the performance of the larger networks (32, 64, and 128 filters [and nodes]).

Based on this empirical evaluation, the network with 64 filters produced the best results on both the training and validation sets, on much fewer training cycles. The best performing network consisted of a convolution layer with 64 filters and a hidden fully connected layer with 64 neurons. This model used two dropout layers, one after each hidden layer, but no batch normalization. Using batch normalizations in this case significantly declined the validation accuracy, dropping it to 52%.

5.7.1 Additional Convolutional Layers

The next hyperparameter investigated was the network depth. The number of layers was increased, and additional convolutional layers were added. Some of these architectures were inspired by the VGG network architecture [73] where convolution layers are stacked.

The first model consisted of two consecutive convolution layers of 8 filters each, followed a fully connected layer with 8 neurons, and a dropout layer with a dropout rate of 0.5. The number of parameters for this network was 246,841. The best results were achieved when two dropout layers were used, one after the two convolution layers, and the other after the dense layer, along with batch normalizations right before the dropout layers. The network achieved a training and validation accuracy of 98.84% and 92.04%, respectively, after training for 31 epochs. This is represented as Model-10.

The second network developed consisted of two consecutive convolution layers of 8 filters each, followed by a batchnorm layer, a dropout layer with a rate of 0.5, a fully connected layer with 16 neurons, and a dropout layer with a rate of 0.5. The best results achieved by the network obtained a training and validation accuracy of 99.75% and 93.18%, respectively, after training for 36 epochs.

Similarly, experiments were conducted with stacked convolution layers with 16, 32, and 64 filters. It was observed that the larger networks were overfitting to the training set. A complete list of experiments conducted while investigating the effects of network width and depth is presented in Table 5.6. The table shows the best performing network from each series of experiments on the given network width. In the table, *conv* represents a convolution layer, *dense* represents a fully connected layer, *drop* represents a dropout layer with a dropout rate of 0.5, and *BN* represents a batchnorm layer.

Table 5.6: The different CNN architectures investigated to study the effect of network width on the performance. The dropout layers (represented as *drop*) used a dropout rate of 0.5

Model name	# of parameters	Network architecture
Model-3	986,993	conv16, BN, drop, dense16, BN, drop, dense1
Model-4	8,258,561	conv16, drop, dense32, drop, dense1
Model-5	16,516,673	conv16, drop, dense64, drop, dense1
Model-6	16,517,313	conv32, BN, drop, dense32, BN, drop, dense1
Model-7	33,033,601	conv32, BN, drop, dense64, BN, drop, dense1
Model-8	66,066,305	conv64, drop, dense64, drop, dense1
Model-9	132,131,585	conv64, dense128, drop, dense1
Model-10	246,905	conv8, conv8, BN, drop, dense8, BN, drop, dense1
Model-11	492,969	conv8, conv8, drop, dense16, drop, dense1
Model-12	986,993	conv16, conv16, BN, drop, dense16, BN, drop, dense1
Model-13	1,971,153	conv16, conv16, BN, drop, dense32, BN, drop, dense1
Model-14	3,946,721	conv32, conv32, BN, drop, dense32, BN, drop, dense1
Model-15	15,784,385	conv64, conv64, BN, drop, dense64, BN, drop, dense1

5.8 Performance Evaluation

The performance of the different CNN architectures investigated is shown in Table 5.7. The performance metrics evaluated are the accuracy and loss for the training and validation datasets.

Table 5.7: Best network performance achieved in terms of training and validation accuracy [scale: 0-1] and loss, along with the number of training cycles (Epochs). The best values are highlighted in bold. Since the validation performance was defined as the primary metric in this work, the best model is identified based on validation accuracy and loss.

Model Name	Epochs	Train Accuracy	Train Loss	Validation Accuracy	Validation Loss
Model-1	16	0.8977	0.21	0.8835	0.29
Model-2	42	0.967	0.06	0.97	0.09
Model-3	14	0.9991	0.005	0.9204	0.177
Model-4	53	0.8531	0.3465	0.9176	0.3149
Model-5	16	0.9801	0.0459	0.9091	0.301
Model-6	10	1.00	0.01	0.5284	0.8745
Model-7	10	1.00	0.0054	0.5250	2.3896
Model-8	23	0.9867	0.0316	0.9488	0.2335
Model-9	12	1.00	0.0018	0.5284	1.1150
Model-10	31	0.9884	0.0285	0.9204	0.2554
Model-11	36	0.9975	0.0191	0.9318	0.19
Model-12	40	1.00	0.0043	0.9715	0.0776
Model-13	27	0.9983	0.0095	0.9176	0.3017
Model-14	**46**	**1.00**	**0.0006**	**0.9801**	**0.0671**
Model-15	10	1.00	0.0024	0.7642	0.5937

As can be observed, the best performance on the training and validation datasets was achieved by Model-14 which stacks two convolution layers followed by a hidden fully connected layer. After training for 46 epochs, the model achieved a training and validation accuracy of 100% and 98%, respectively. There were other models which achieved comparable performance on the validation dataset, including Model-2 (97%), Model-8 (94.88%), and Model-12 (97.15%). These networks were tested on the test set to evaluate the generalization capabilities on unseen data. This is discussed next.

5.9 Prediction Capabilities

The classification performance is also evaluated using some additional evaluation metrics [135]. In a binary classification task, a confusion matrix is commonly used to measure network performance and assess the model performance on unseen data. Based on the predictions, the outcomes can be categorized into 4 types:

1) True Positive (TP): Number of samples that are classified correctly as yes or success
2) True Negative (TN): Number of samples that are classified correctly as no or failure
3) False Positive (FP): Number of samples that are classified incorrectly as yes or success
4) False Negative (FN): Number of samples that are classified incorrectly as no or failure

Based on these numbers, the performance can be evaluated using the formulas presented in equations (5.1) and (5.2).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad \textbf{(5.1)}$$

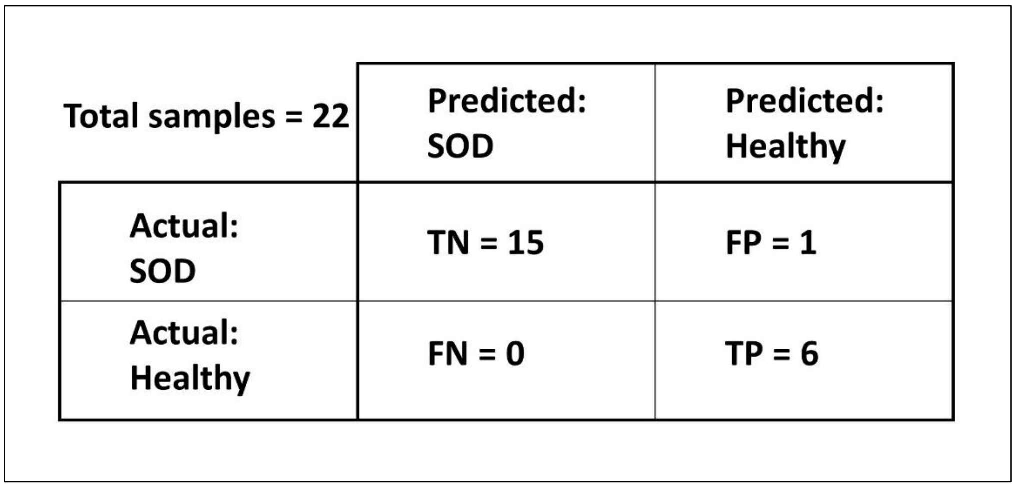
$$Error_{rate} = \frac{FP + FN}{TP + TN + FP + FN} = 1 - Accuracy \quad \textbf{(5.2)}$$

To evaluate the performance of the proposed CNN and the transfer learning models, the test dataset used for the prediction task comprised of 6 healthy cases and 16 cases belonging to SOD patients.

Model-14, which was trained from the scratch, successfully classified the healthy test samples. The model successfully labelled 15 out of 16 samples from the SOD dataset. Based on these numbers, the accuracy can be calculated using equation (5.1). Therefore, the overall accuracy of this model is 95.45%

$$Accuracy = \frac{6 + 15}{0 + 6 + 1 + 15} = \frac{21}{22} = 0.9545 \quad \textbf{(5.3)}$$

Model-12 correctly labelled the healthy samples in the test set. However, it labelled 12 out of 16 samples correctly in the SOD set. The overall prediction accuracy of this model is 81.82%. Similarly, Model-2 correctly labelled the healthy samples, and incorrectly labelled 6 out of 16 SOD samples. The overall prediction accuracy is 72.72%. The confusion matrix corresponding to this is shown in Figure 5.9.



Total samples = 22	Predicted: SOD	Predicted: Healthy
Actual: SOD	TN = 15	FP = 1
Actual: Healthy	FN = 0	TP = 6

Figure 5.9: Confusion matrix for the best performing customized CNN on the test dataset. TN is a true negative, FN is a false negative, FP is a false positive, and TP is a true positive

5.10 Advanced Hyperparameters

On changing the kernel size using Model-14, there was a negligible effect on the network's performance on both training and validation datasets. Therefore, a 3 x 3 kernel size is recommended for this dataset.

For the learning rate, three settings were experimented with: 0.001 (default), 0.0005, and the module ReduceLROnPlateau [136]. ReduceLROnPlateau is a Keras module that reduces the learning rate when the defined performance metric stops improving. Changing the learning rate from the default value of 0.001 extended the training time but failed to improve the performance on the training and validation datasets. Therefore, the default learning rate is recommended.

5.11 Transfer Learning

As explained in section 2.5, using pre-trained models is an effective technique to implement CNNs using relatively small datasets. In this work, we investigated some of the popular pre-trained models trained using the ImageNet dataset (ImageNet large scale visual recognition challenge; ILSVRC). These models included AlexNet [71], InceptionV3 [72], ResNet50 [74], and VGG16 [73]. AlexNet and VGG16 are shallow networks with 8 and 16 trainable layers respectively. InceptionV3 and ResNet50 are deeper networks with 48 and 51 trainable layers respectively [68]. These models, except for AlexNet, are downloaded and imported from the Keras libraries.

The pre-trained models were imported with the pre-trained ImageNet weights. The imported models were then fine-tuned to suit our classification task by freezing all the layers in the pre-trained networks except for the classifier. We drop the final fully connected layer from these models since this is tailored to classify the ImageNet dataset. Finally, we add some fully connected layer(s) to achieve the classification task for our CVR dataset. The network architectures are explained in the sub-sections below.

The transfer learning is applied as a fine-tuning workflow and consisted of four steps as described below.

1) Load the base model (the pre-trained network) and load the pre-trained weights
2) Freeze all the layers in the base model. This is done by setting trainable = False
3) Customize the model layers, for example, the input and the output layers, on top of the output from one of the layers from the base model
4) Train this new model using the CVR dataset.

5.11.1 Common Network Settings

Like the experiments in the design of customized CNN, the input images were re-sized to 224 x 224 pixels for training the pre-trained models, and for all the networks investigated in this section, an early stopping criterion was used. A batch size of 32 was used for all networks. The monitored metric was the validation loss (“val_loss”).

5.11.2 AlexNet-based CNN

Keras does not have a readily available pre-trained AlexNet network that can be downloaded and fine-tuned. Therefore, an AlexNet-inspired CNN was designed and trained from the scratch, like the other customized CNNs. The architecture used in this experiment is shown in Table 5.8. The layers in bold represent the core layers from the pre-trained network, and the layers starting from Flatten are customized to classify the CVR maps. The program used to design the AlexNet is explained in Appendix-B, section B.6.

Table 5.8: The AlexNet-based architecture designed for CVR classification. The layers native to the AlexNet network are highlighted in bold. The second half of the network divided in the table represent the customized layers added for the target classification.

Layer	# filters @ kernel size | stride	Size of feature maps	Activation Function	Parameters
Input	-	227 x 227 x 3	-	-
Conv 1	96 @ 11 x 11 | 4	55 x 55 x 96	ReLU	34944
Max Pool 1	3 x 3 | 2	27 x 27 x 96	-	-
Conv 2	256 @ 5 x 5 | 1	27 x 27 x 256	ReLU	614656
Max Pool 2	3 x 3 | 2	13 x 13 x 256	-	-
Conv 3	384 @ 3 x 3 | 1	13 x 13 x 384	ReLU	885120
Conv 4	384 @ 3 x 3 | 1	13 x 13 x 384	ReLU	1327488
Conv 5	256 @ 3 x 3 | 1	13 x 13 x 256	ReLU	884992
Max Pool 3	3 x 3 | 2	6 x 6 x 256	-	-
Flatten	9216	-	-	-
Fully connected, dropout = 0.5	4096	-	ReLU	37752832
Fully connected, dropout = 0.5	4096	-	ReLU	16781312
Fully connected	1	-	Softmax	4097
Total Parameters				58,290,945

5.11.3 Inception V3

The program for loading the InceptionV3 pre-trained model and customizing it by adding final layers is presented in Appendix-B, section B.7. As it can be seen, the top layer is excluded (*include_top = False*) and the core layers are frozen using *pre_trained_model.trainable = False*. A flatten layer is added after the output layer of the pre-trained model. After the flatten layer, the output shape is (None, 51200). If an output layer is used directly, the difference between the input shape for this layer coming from the flatten layer, and the classification output of (None, 1) would be huge. Therefore, an additional fully connected layer (represented as *dense*) is used with 1024 neurons and a dropout with a 0.5 probability. This method returns the model which can be used with our training dataset. As a part of the fine-tuning process, the top two layers were unfrozen and re-trained with the CVR dataset to utilize these layers during the training process. However, there were no improvements in the results.

5.11.4 Other Pre-Trained Networks

Like the Inception V3 network design, the ResNet50 pre-trained model was imported while freezing the lower convolution layers and adding fully connected layers. Only one fully connected layer was added on top of the pre-trained network, with a Sigmoid activation function.

The VGG 16 pre-trained network was imported and customized to classify the CVR maps. Like the ResNet50 architecture, only one classification output layer was added after the VGG network's feature transformer. And finally, the EfficientNetB0 model was imported, and the final layer was customized, with one classification output layer.

5.11.5 Transfer Learning: Results

The early stopping module used while training the models stopped the training for the AlexNet and InceptionV3 models after 17 and 14 epochs, respectively. Similarly, the EfficientNetB0, ResNet50, and VGG16 models trained for 27, 85, and 25 epochs, respectively.

The AlexNet-based model achieved only 50% accuracy for both the training and validation datasets. The EfficientNetB0 model achieved a similar accuracy of 51.5% on the training dataset and a similar accuracy of 52.27% on the validation dataset. The InceptionV3 model achieved similar performance on both datasets. The ResNet50 model continued training for longer before it could stabilize the validation loss. After training for 85 epochs, the model achieved an accuracy of 52%.

The VGG16 model produced the best results, achieving 99.83% and 94.88% accuracy on the training and the validation datasets, respectively. A comparison of the training and validation accuracy obtained from these models is shown in Figure 5.10 and Figure 5.11.

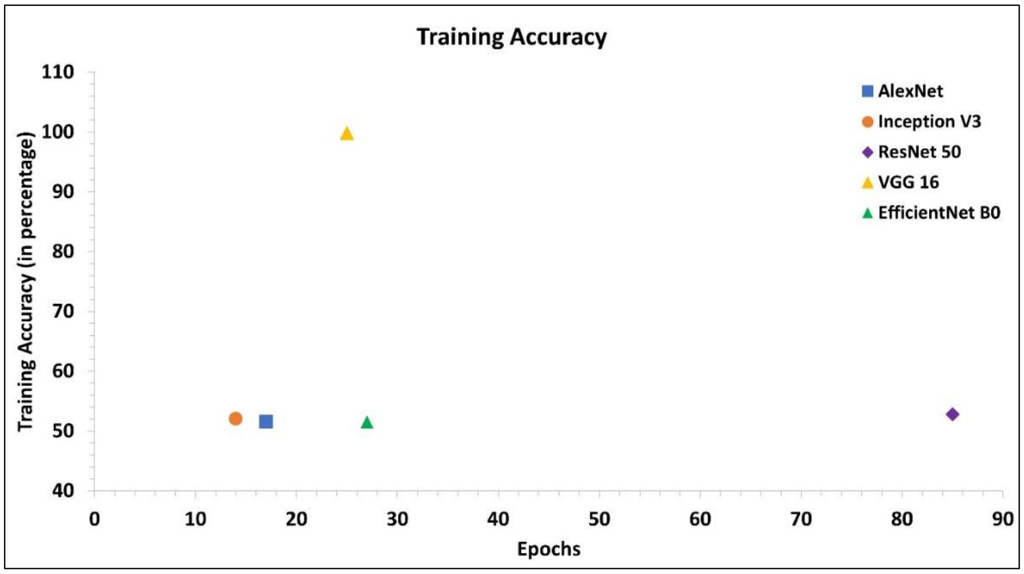


Figure 5.10: Comparison of the training accuracy obtained by the pre-trained networks and AlexNet-based CNN after fine-tuning for the classification of CVR maps

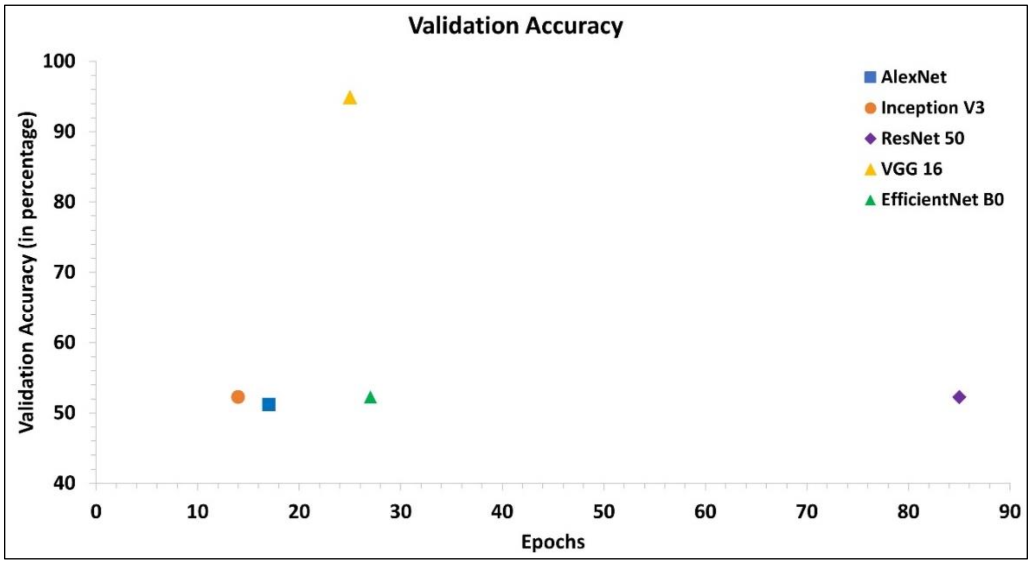


Figure 5.11: Comparison of the validation accuracy obtained by the fine-tuned pre-trained networks and AlexNet-based CNN

5.11.6 Transfer Learning: Prediction Results

The best performing pre-trained network VGG correctly labeled all the samples from the unhealthy class as SOD patients. However, it was not able to label healthy samples correctly, inferring that the samples were SOD patients. Therefore, using equation (5.1), the overall prediction accuracy of the VGG model is 72.72%. The same behavior was observed in other pre-trained model performances.

5.12 Results Analysis

The performance of all the pre-trained networks and the most optimal customized network is compared in Table 5.9. The accuracy of the different models on the train, validation, and test datasets are highlighted.

Table 5.9: Performance of all the CNN models experimented on the CVR datasets. CNN[1] is the most optimal customized network trained from scratch. Other networks are the pre-trained networks fine-tuned for CVR classification. The best results are shown in bold.

Model	Network Design	Training Accuracy (%)	Validation Accuracy (%)	Prediction Accuracy (%)
CNN[1]	**Trained from scratch**	**100**	**98.01**	**95.45**
AlexNet	Trained from scratch	51.54	51.2	72.72
EfficientNetB0	Pre-trained network	51.5	52.3	72.72
InceptionV3	Pre-trained network	52.06	52.27	72.72
ResNet50	Pre-trained network	52.8	52.27	72.72
VGG16	Pre-trained network	99.83	94.88	72.72

Based on the model performance, the pre-trained networks are complicated for the small dataset used in this study. The huge number of parameters and the complexity of the architecture of these pre-trained network produces poor performance. However, VGG, a comparatively simpler and smaller network, produced high accuracy and improved prediction results as compared to other networks. The pre-trained networks have proven very successful with small datasets and there is an opportunity to further fine-tune them to improve performance for the CVR dataset. For example, some experiments with the InceptionV3 included unfreezing the top two layers. This did not improve the results, but more experiments are needed to optimize the transfer learning process. The customized network trained from scratch produced the best results.

5.12.1 Proposed CNN

Based on the results discussed in section 5.8 and comparing them with the results from the pre-trained networks, a customized CNN is proposed as the best-suited network for the considered classification task with the CVR datasets to screen steno-occlusive disease patients. The network achieves high accuracy on both the training and validation datasets and this performance translates to the test dataset. Although the test dataset is small, consisting of 22 samples, the proposed network correctly labels all but one, achieving a prediction accuracy of 95.45%.

The proposed CNN is a shallow network with two convolution layers with 32 filters and a 3x3 kernel stacked one after the other with a max pooling layer with a pool size of 2x2 in between, followed by another max pooling layer (2x2 pool size), a batch normalization layer, a dropout layer with a dropout rate of 0.5, a flatten layer, a fully connected layer with 32 neurons, another dropout layer with a dropout rate of 0.5, and an output layer with 1 neuron. A schematic representation of this network is shown in Figure 5.12.

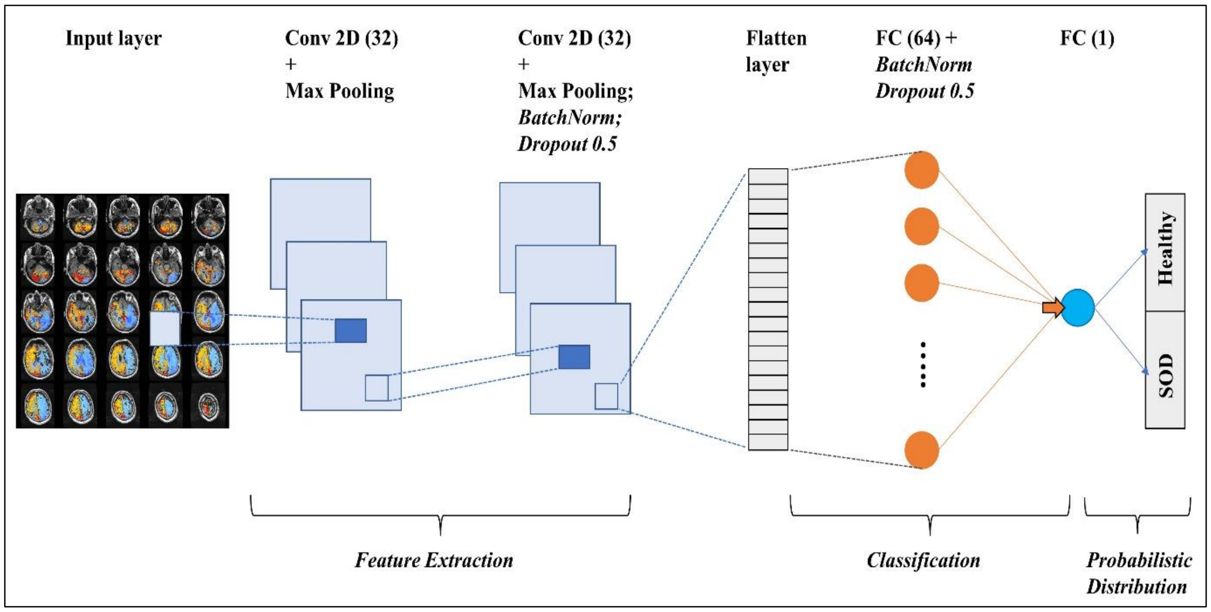

Input layer
Conv 2D (32) + Max Pooling
Conv 2D (32) + Max Pooling; BatchNorm; Dropout 0.5
Flatten layer
FC (64) + BatchNorm Dropout 0.5
FC (1)
Healthy
SOD
Feature Extraction
Classification
Probabilistic Distribution

Figure 5.12: A schematic representation of the proposed CNN architecture. Conv 2D is a 2-dimensional convolution layer and FC is a fully connected layer. The number of filters (or neurons) are shown inside the round brackets

The network architecture is detailed in Table 5.10. The input layer consists of images of 256 x 256 pixels and 3 channels (R, G, and B). The total trainable parameters of this network are 3,946,721.

Table 5.10: Architecture of the proposed CNN model. The dotted line separates the feature extractor (the convolution layers) from the fully connected layers which classify the CVR maps.

Layer	**# filters @ kernel size | stride**	**Output shape**	**Activation Function**	**Parameters**
Input	-	256 x 256 x 3	-	-
2D Convolution	32 @ 3 x 3 | 1	254 x 254 x 32	ReLU	896
2D Max Pooling	2 x 2 | 3	127 x 127 x 32	-	0
2D Convolution	32 @ 3 x 3 | 1	125 x 125 x 32	ReLU	9248
2D Max Pooling	2 x 2 | 3	62 x 62 x 32	-	0
Batch Normalization	-	62 x 62 x 32	-	128
Dropout (0.5)	-	62 x 62 x 32	-	0
Flatten	-	123008	-	0
Fully connected	32	32	ReLU	3936288
Batch Normalization	-	32	-	128
Dropout (0.5)	-	32	-	0
Fully connected	1	1	Sigmoid	33
Total Parameters				**3,946,721**

Chapter 6

Discussions

6.1 Key Contributions

The Cerebrovascular Reactivity (CVR) software proposed in this research serves as a next-generation tool that furthers the capabilities of CVR analyses, supporting experts in making judgments and decisions. It successfully addressed the major concerns of the existing workflow, and the project objectives were met. As discussed earlier, independent shell scripts were used to realize the workflow. To use these scripts, special subject expertise and experience are required to execute the scripts in the required order. Since the existing workflow did not have a standard application package, troubleshooting and maintenance of the shell scripts is another challenge. And using other software tools as a part of the workflow creates chances of non-standardized results.

The CVR software overcomes all these barriers as it requires minimal subject knowledge to be operated, serves as a standalone tool, and is uploaded on a private repository which allows systematic troubleshooting and maintenance. The CVR software standardizes the research workflow, automating the overall process by an estimated 75%. A simple workflow and easy-to-use modules ensure that users with minimal AFNI (Analysis of Functional NeuroImages) experience can also complete CVR analyses using raw MRI files. From the developers' point of view, the software is designed using object-oriented principles and has a modular design. This ensures that new features can be seamlessly added and helps in testing individual components using unit tests. The software can also be easily scaled.

The CVR software package was deployed to a private repository, allowing easy distribution to collaborators and participating institutions. As this research concluded, one of the partnering companies, based in the United States, began testing their native datasets using the software. This software successfully scaled to the new dataset in the NIfTI format and produced accurate CVR maps. Using the same software and workflow across participating institutions will ensure that the results are standardized and comparable. The virtual machine-based software solution also overcomes operating system concerns, as it can be easily set up on any machine irrespective of the operating system. The virtual machine developed to support this work has the AFNI programs pre-installed, therefore, acting as a one-stop solution to CVR analyses and assessments.

To further expedite the CVR assessment process, Convolutional Neural Networks (CNNs) were investigated to be used with the CVR datasets. The first step was to create sufficient trainable samples to be used with the CNNs. Data augmentation techniques were used to address data scarcity. However, the data augmentations were small-scale to preserve the spatial features of the CVR maps. The proposed CNN, a customized shallow network with two stacked convolution layers, produced results consistent with clinical readings.

Since the CVR software is developed as a research tool and is undergoing testing before it can be distributed, the financial aspects of the software have not been evaluated.

6.1.1 CVR Software: Testing and Scalability

The CVR software was rigorously tested using real-world clinical trials and CVR analyses. The results obtained from the CVR software agreed with the results obtained from the existing tools. The proposed software is capable of processing three types of raw MRI datasets: DICOM (.dcm), MR (.mr), and NIfTY (.nii). Since the software uses the AFNI routine to process these files, the proposed workflow is independent of the scanner used. However, the software can only process one type of vasoactive stimulus, the prospective end-tidal targeting of partial pressure of arterial CO2 (Pa,CO_2).

The software was used to process BOLD-MRI data obtained from one of the collaborators and it successfully scaled to the much larger datasets. To compare the size of the two datasets, the voxels datasets produced from in-house analyses were 100 MB and the ones from the collaborator were 1.9 GB. The algorithm was optimized to accommodate such large datasets, and the proposed upgrades were successfully tested and published.

6.1.2 CNN Design: Main Findings

From the quantitative results shown in Table 5.7, several inferences can be made on the hyperparameter fine-tuning experiments. First, the dropout rate played a significant role in addressing the overfitting tendency of the network. Increasing the dropout probability to 0.5 improved the model performance on the validation dataset and addressed the overfitting. Next, the batch-size experiments indicated that increasing the batch size stabilized the noise in the validation results, improving the performance. The recommended batch size of 32 was effective and produced consistent results in the experiments conducted. In terms of the network width and depth, several experiments were conducted to understand how different layers and width affected the model performance. It was observed that, given the relatively small datasets, shallower networks outperformed the deeper networks.

Four of the popular pre-trained networks were fine-tuned to be used with the CVR datasets. AlexNet-based model underperformed in the target classification task. This experiment was technically not a transfer learning workflow since the model was trained from scratch. It was intended to investigate the performance of an AlexNet-inspired network. EfficientB0, Inception, and ResNet did not perform well with the dataset and produced low accuracy. However, VGG produced significantly improved results. The generalization capabilities of the pre-trained networks were not adequate. These networks are unable to identify any healthy sample correctly.

Given the small datasets used to train the proposed CNN, there is a need to extend the network training and conduct more experiments to further the generalization capabilities of the network. It is suggested that the proposed CNN may be used as a research tool and the results obtained are validated by human experts. As with any neural network, the scope of the proposed CNN classifier is limited to the CVR studies conducted using the clinical workflow and color scales implemented at our institute.

6.1.3 Customized CNN: Incorrect prediction

The proposed CNN incorrectly labeled one SOD patient as a healthy patient. The sample is shown in Figure 6.1. As can be observed, this sample is an edge case and does not have a lot of blue gradients in the grey matter of the brain. The blue gradients are not prominent, and the patient has a mild cerebrovascular condition. A notable difference in this sample is that the reduced CVR response is observed in the bottom slices of the brain (the last 2 elements in the bottommost row).

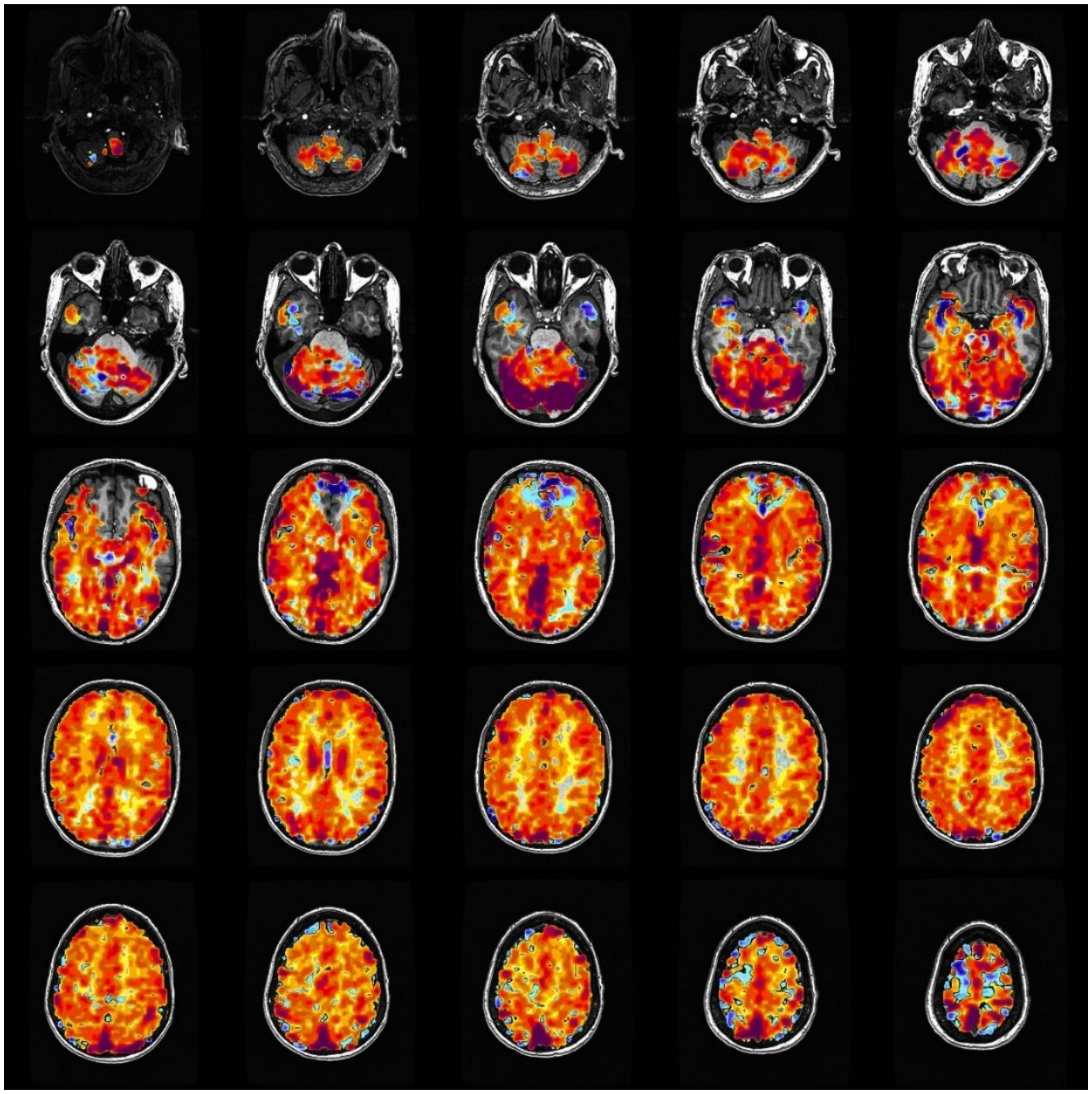


Figure 6.1: The SOD patient sample in the test set that was incorrectly labelled by the proposed customized CNN

6.2 Proposed Workflow

The CVR processing and assessment software can be used as a start-to-end research workflow. The proposed workflow shall begin in the Magnetic Resonance Imaging (MRI) room where the breathing study is conducted by trained professionals to obtain the raw MRI datasets. The datasets can be transferred to a local device that has the CVR software installed on it. Following this, the input images can be processed to produce the corresponding CVR maps. The pre-trained CNN-based classifier then calculates the probability of the class of the CVR map produced and labels it as a healthy subject or a steno-occlusive disease (SOD) patient. The final output CVR map along with the label is presented to the expert radiologists for validation and final comments. The workflow is demonstrated in Figure 6.2.

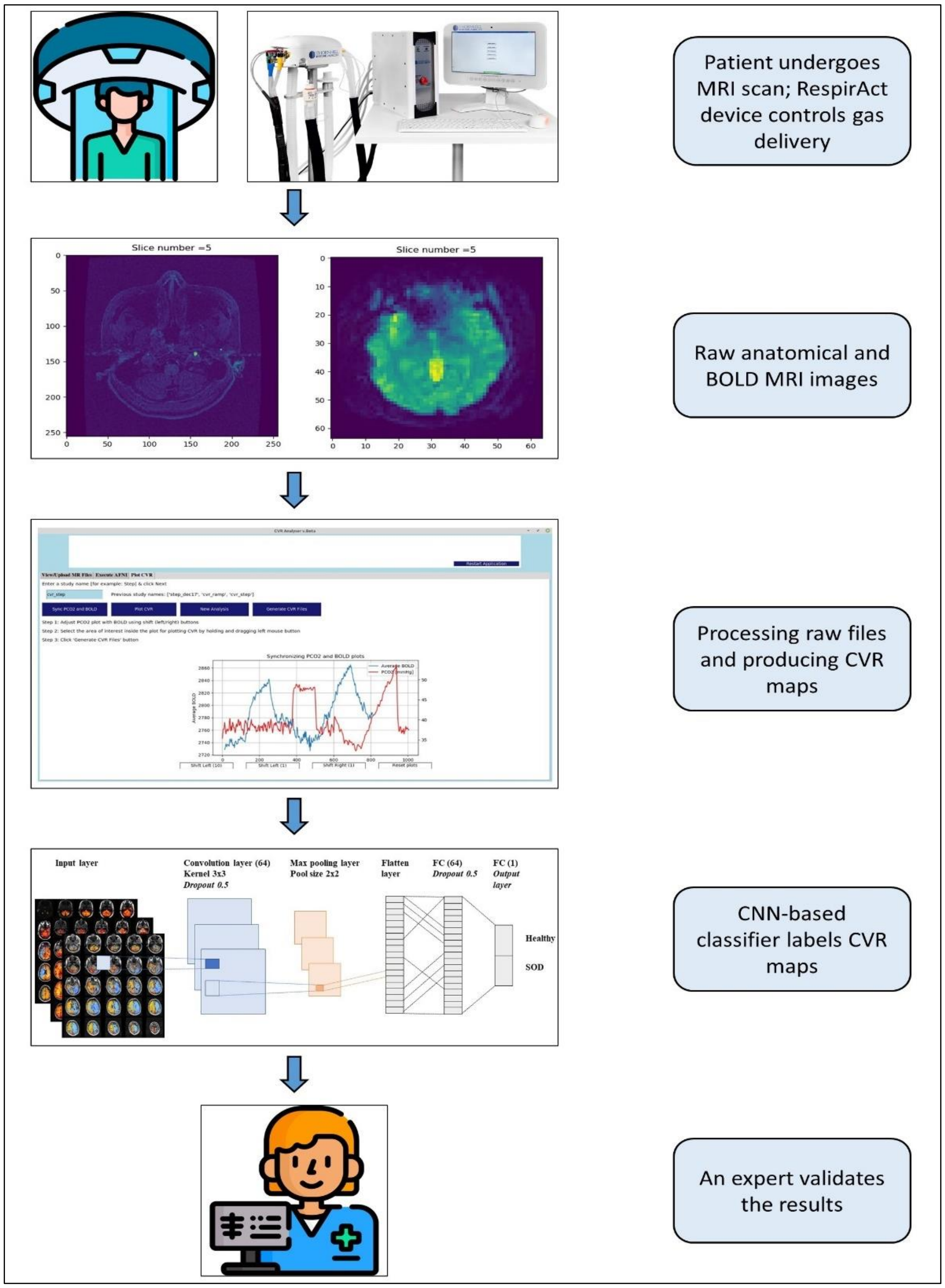

Patient undergoes MRI scan; RespirAct device controls gas delivery
Slice number =5
Slice number =5
Raw anatomical and BOLD MRI images
Synchronizing PCO2 and BOLD plots
Processing raw files and producing CVR maps
Input layer
Convolution layer (64) Kernel 3x3 Dropout 0.5
Max pooling layer Pool size 2x2
Flatten layer
FC (64) Dropout 0.5
FC (1) Output layer
Healthy
SOD
CNN-based classifier labels CVR maps
An expert validates the results

Figure 6.2: A schematic representation of the proposed CVR processing and assessment workflow using the CVR software and CNN-based classifier tool. The icons used in this figure are from Flaticon [115]

6.3 Future Work

The success of this research can be established by the myriad of future directions it provides to the sponsoring institutions in the field of computer-aided diagnostics (CAD). First, as discussed in section 4.6, the AFNI programs produce intermediate output files in the MNI space too, in addition to the standard space. However, the CVR software's plotting capabilities are currently limited to the standard space. An additional module can be easily developed and seamlessly integrated that can plot CVR maps in the MNI space.

Second, currently, the software plots linear CVR maps based on the slope calculations between the BOLD changes and the applied PCO2. There are several other types of maps that are used along with the linear CVR maps to make clinical judgments. One such method is calculating the z-score for each voxel and mapping the values using a z-map. There is an opportunity to add these features to the CVR software.

Another future direction to the CVR software is overcoming the dependence on AFNI programs. Although the AFNI program suite is well-established and comfortably integrates with the CVR software, the required functionality can be achieved by developing in-house programs as well. The source codes for AFNI modules are available publicly. The source code for these modules is written in C programming.

Additionally, the senior software manager at a partnering institution identified an opportunity to scale the CVR software as a cloud-based application. If successful, this project would add numerous advantages over the current virtual machine-based software, including, cost savings, mobility, increased collaboration, automatic software updates, and sustainability.

For furthering the CNN-based classification capabilities, it is important to collect more data. Since the prospective end-tidal gas targeting is a recent discovery, there have not been many studies. As more patients are analyzed, the proposed CNN should be retrained. Another direction to Artificial Intelligence-based decisions would be to train the machine learning models on more abstract datasets. For example, instead of using CVR maps to train the networks, extract features from the raw BOLD signals and feed them into the networks.

Finally, this study did not investigate if the CNN models trained on the selected CVR dataset can generalize to other CVR analyses, conducted using a different vasoactive stimulus, for example, breath-hold, fixed inspired CO2, and chemical stimuli. This can potentially help establish a framework for using the pre-trained weights and biases from the model proposed in this study and fine-tuning it to other CVR maps. As discussed in section 2.6.2, a study has found that such transfer learning achieves better performance than training a new model from the scratch.

Chapter 7

Conclusions

In this research, a next-generation Cerebrovascular Reactivity (CVR) processing and assessment software was engineered that provides a platform for furthering the processing, visualization, and research capabilities of the CVR research. CVR is a provocative test used with Blood Oxygenation Level Dependent (BOLD)-MRI studies where a vasoactive stimulus is applied and the corresponding changes in the CBF differences are analyzed. The research workflow used in this study uses prospective end-tidal targeting of partial pressure of carbon dioxide as the vasostimulus. CVR studies can highlight abnormalities in the brain vasculature and help identify the presence of cerebrovascular diseases like Alzheimer’s disease, steno-occlusive disease (SOD), stroke, and traumatic brain injury. SOD is the most common cause of stroke all over the world and therefore its detection can help guide clinical treatment.

The current generation of CVR analysis and assessments was conducted manually by a team of doctors and radiologists, using shell-scripts-based programs. The proposed software, built using open-source tools, standardizes the CVR research workflow and is capable of processing anatomical and BOLD MRI files and visualizing CVR maps. The software allows viewing and uploading input MRI files, automatic screening functions to ensure the input files are correctly placed, and saves the output files in a well-structured directory, providing a well-organized data uploading and extraction platform. It provides advanced plotting features, allowing users to accomplish the plotting steps in the same software interface. Finally, it makes system calls to AFNI programs to visualize the CVR maps.

The customized convolutional neural networks as well as the fine-tuned pre-trained networks are capable of successfully identifying the key parameters in the CVR maps to discriminate between healthy and unhealthy groups. An empirical evaluation-based network design methodology was implemented to optimize the customized CNN architecture by fine-tuning the network hyperparameters. The influence of the different hyperparameter settings is evaluated. The model weights and biases are saved, and the model can be used to readily make predictions on new CVR maps.

During the transfer learning experiments, the fine-tuned VGG16 model produces the best results on both training and validation datasets, after training for 22 epochs. The model achieves a training and validation accuracy of 99.83% and 94.88% respectively. While this was a relatively simple network, other pre-trained networks failed to perform well and their accuracy values were around 50%.

Based on the experiments, a shallow network trained from scratch is proposed as the most optimal classifier. The generalization results of this model are consistent with expert clinical readings. To the best of the authors' knowledge, this research work on developing software tools for processing CVR and screening SOD patients using convolutional neural networks does not put anyone at any potential disadvantage. The proposed software is intended to be used as a research tool only, and it is recommended that the results obtained are validated using human expertise.

References

[1] Mayo Clinic. (2017). MRI - Mayo Clinic. Mayoclinic.org. Retrieved November 14, 2022, from https://www.mayoclinic.org/tests-procedures/mri/about/pac-20384768

[2] Fierstra, J., Sobczyk, O., Battisti-Charbonney, A., Mandell, D. M., Poublanc, J., Crawley, A. P., Mikulis, D. J., Duffin, J., & Fisher, J. A. (2013). Measuring cerebrovascular reactivity: what stimulus to use? The Journal of Physiology, 591(23), 5809–5821. https://doi.org/10.1113/jphysiol.2013.259150

[3] Emilie Sleight, Michael S. Stringer, Ian Marshall, Joanna M. Wardlaw, and Michael J. Thrippleton. Cerebrovascular reactivity measurement using magnetic resonance imaging: A systematic review. Frontiers in Physiology, 12, 2021

[4] Martinez-Rodriguez J. E. Munteis E. Gomis M. Rodr´ıguez-Campello A. Jimenez-Conde J. Cuadrado-Godia E. amp; Roquer J. Ois, A. Stenoocclusive arterial disease and early neurological deterioration in acute ischemic stroke. 2008

[5] Fisher, J.A. (2016). The CO2 stimulus for cerebrovascular reactivity: Fixing inspired concentrations vs. targeting end-tidal partial pressures. Journal of Cerebral Blood Flow & Metabolism, 36(6), pp.1004–1011. Doi:10.1177/0271678x16639326

[6] WSO | World Stroke Organization. (2020). World Stroke Organization. Retrieved on November 14, 2022, from https://www.world-stroke.org/

[7] FastStats - Cerebrovascular Disease or Stroke. (2020). Retrieved on November 14, 2022, from https://www.cdc.gov/nchs/fastats/stroke.htm

[8] WHO EMRO | Stroke, Cerebrovascular accident | Health topics. (2022). World Health Organization - Regional Office for the Eastern Mediterranean. https://www.emro.who.int/health-topics/stroke-cerebrovascular-accident/index.html

[9] Lee PH, Oh SH, Bang OY, et alIsolated middle cerebral artery disease: clinical and neuroradiological features depending on the pathogenesisJournal of Neurology, Neurosurgery & Psychiatry 2004;75:727-732

[10] MATLAB - MathWorks. (2019). Mathworks.com. Retrieved November 14, 2022, from https://www.mathworks.com/products/matlab.html

[11] Gaillard, F. (n.d.). BOLD imaging | Radiology Reference Article | Radiopaedia.org. Radiopaedia. Retrieved November 14, 2022, from https://radiopaedia.org/articles/bold-imaging

[12] https://www.cancer.gov/publications/dictionaries/cancer-terms/def/vasoactive. (2011, February 2). www.cancer.gov. https://www.cancer.gov/publications/dictionaries/cancer-terms/def/vasoactive

[13] Joseph A. Fisher and David J. Mikulis. Cerebrovascular reactivity: Purpose, optimizing methods, and limitations to interpretation – a personal 20-year odyssey of (re)searching. Frontiers in Physiology, 12, 2021

[14] MacKenzie, E. T., Farrar, J. K., Fitch, W., Graham, D. I., Gregory, P. C., & Harper, A. M. (1979). Effects of hemorrhagic hypotension on the cerebral circulation. I. Cerebral blood flow and pial arteriolar caliber. Stroke, 10(6), 711-718

[15] Vorstrup, S., Brun, B., & Lassen, N. A. (1986). Evaluation of the cerebral vasodilatory capacity by the acetazolamide test before EC-IC bypass surgery in patients with occlusion of the internal carotid artery. Stroke, 17(6), 1291-1298

[16] Poulin, M. J., Liang, P. J., & Robbins, P. A. (1996). Dynamics of the cerebral blood flow response to step changes in end-tidal PCO2 and PO2 in humans. Journal of applied physiology, 81(3), 1084-1095

[17] Renata F Leoni, Kelley C Mazzetto-Betti, Afonso C Silva, Antonio C Dos Santos, Draulio B de Araujo, Joao P Leite, and Octavio M Pontes-˜ Neto. Assessing cerebrovascular reactivity in carotid steno-occlusive disease using mri bold and asl techniques, Jun 2012

[18] Daniel M Mandell, Jay S Han, Julien Poublanc, Adrian P Crawley, Andrea Kassner, Joseph A Fisher, and David J Mikulis. Selective reduction of blood flow to white matter during hypercapnia corresponds with leukoaraiosis. Stroke, 39(7):1993–1998, 2008

[19] Daniel M Mandell, Jay S Han, Julien Poublanc, Adrian P Crawley, Jeff A Stainsby, Joseph A Fisher, and David J Mikulis. Mapping cerebrovascular reactivity using blood oxygen level-dependent mri in patients with arterial steno-occlusive disease: comparison with arterial spin labeling mri. Stroke, 39(7):2021–2028, 2008

[20] Chen, J. J. (2018). Cerebrovascular-Reactivity Mapping Using MRI: Considerations for Alzheimer’s Disease. Frontiers in Aging Neuroscience, 10. https://doi.org/10.3389/fnagi.2018.00170

[21] Lidia Glodzik, Catherine Randall, Henry Rusinek, and Mony J de Leon. Cerebrovascular reactivity to carbon dioxi eason iaheimer’s disease. Journal of Alzheimer’s disease, 35(3):427–440, 2013

[22] Mauro Silvestrini, Patrizio Pasqualetti, Roberto Baruffaldi, Marco Bartolini, Yasmin Handouk, Maria Matteis, Filomena Moffa, Leandro Provinciali, and Fabrizio Vernieri. Cerebrovascular reactivity and cognitive decline in patients eason iaheimer disease. Stroke, 37(4):1010–1015, 2006

[23] Spencer L Waddle, Meher R Juttukonda, Sarah K Lants, Larry T Davis, Rohan Chitale, Matthew R Fusco, Lori C Jordan, and Manus J Donahue. Classifying intracranial stenosis disease severity from functional mri data using machine learning. Journal of Cerebral Blood Flow & Metabolism, 40(4):705–719, 2020

[24] Sebök, M., van Niftrik, C. H. B., Germans, M. R., Katan, M., Kulcsar, Z., Luft, A., Regli, L., & Fierstra, J. (2022). Recurrent stroke in symptomatic steno-occlusive disease: identifying patients at high-risk using impaired BOLD cerebrovascular reactivity. Brain and Spine, 2, 101215. https://doi.org/10.1016/j.bas.2022.101215

[25] Chris. (2019, October 18). Made-in-Canada Technology may Help to Take Nobel Medicine Prize Research from Test Tube to Patient Bedside. Thornhill Medical USA. Retrieved November 14, 2022, from https://thornhillmedical.com/oct-18-2019-made-in-canada-technology-may-help-to-take-nobel-medicine-prize-research-from-test-tube-to-patient-bedside/

[26] Slessarev, M., Han, J., Mardimae, A., Prisman, E., Preiss, D., Volgyesi, G., Ansel, C., Duffin, J., & Fisher, J. A. (2007). Prospective targeting and control of end-tidal CO2 and O2

concentrations. The Journal of Physiology, 581(3), 1207–1219. https://doi.org/10.1113/jphysiol.2007.129395

[27] RespirAct® RA-MRTM. (n.d.). Thornhill Medical Canada. Retrieved November 14, 2022, from https://thornhillmedical.ca/research/respiract-ra-mr/

[28] RespirAct® RA-MRTM Operator's Manual. (n.d.). Thornhill Medical Canada. Retrieved November 14, 2022, from https://thornhillmedical.ca/resources/respiract-ra-mr/manual/

[29] About DICOM- Overview. (n.d.). DICOM. https://www.dicomstandard.org/about-home

[30] Bidgood, W. D., Horii, S. C., Prior, F. W., & Van Syckle, D. E. (1997). Understanding and Using DICOM, the Data Interchange Standard for Biomedical Imaging. Journal of the American Medical Informatics Association, 4(3), 199–212. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC61235/

[31] afni.nimh.nih.gov. (n.d.). Afni.nimh.nih.gov. https://afni.nimh.nih.gov/

[32] Poublanc, J., Crawley, A. P., Sobczyk, O., Montandon, G., Sam, K., Mandell, D. M., Dufort, P., Venkatraghavan, L., Duffin, J., Mikulis, D. J., & Fisher, J. A. (2015). Measuring Cerebrovascular Reactivity: The Dynamic Response to a Step Hypercapnic Stimulus. Journal of Cerebral Blood Flow & Metabolism, 35(11), 1746–1756. https://doi.org/10.1038/jcbfm.2015.114

[33] Python Software Foundation. (2019). What is Python? Executive Summary. Python.org; Python.org. Retrieved November 14, 2022, from https://www.python.org/doc/essays/blurb/

[34] Applications for Python. (n.d.). Python.org. Retrieved November 14, 2022, from https://www.python.org/about/apps/

[35] PyPI · The Python Package Index. (n.d.). PyPI. Retrieved November 14, 2022, from https://pypi.org/

[36] Software Development Life Cycle (SDLC). (2020, February 26). GeeksforGeeks. https://www.geeksforgeeks.org/software-development-life-cycle-sdlc/

[37] GitHub. (2018). https://github.com/. Accessed December 1, 2022

[38] Pittet, S. (2019). The different types of testing in Software | Atlassian. Atlassian. https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing

[39] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT Press

[40] What is Artificial Intelligence (AI)? (n.d.). Www.ibm.com. Retrieved November 14, 2022, from https://www.ibm.com/cloud/learn/what-is-artificial-intelligence#toc-deep-learn-md_Q_Of3

[41] Mitchell, T. M. (1997). Machine learning. Mcgraw-Hill

[42] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep Learning. Nature, 521(7553), 436–444. https://doi.org/10.1038/nature14539

[43] Train Test Validation Split: How To & Best Practices [2022]. (n.d.). Www.v7labs.com. Retrieved December 15, 2022, from https://www.v7labs.com/blog/train-validation-test-set#:~:text=Validation%20split%20helps%20to%20improve

[44] Alzubaidi, L., Zhang, J., Humaidi, A.J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Santamaría, J., Fadhel, M.A., Al-Amidie, M. and Farhan, L. (2021). Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. Journal of Big Data, 8(1). Doi:10.1186/s40537-021-00444-8

[45] Duca, A. L. (2022, February 19). Is a Small Dataset Risky? Medium. https://towardsdatascience.com/is-a-small-dataset-risky-b664b8569a21

[46] Dietterich, T. (1995). Overfitting and undercomputing in machine learning. ACM Computing Surveys, 27(3), pp.326–327. Doi:10.1145/212094.212114

[47] IBM Cloud Education. (2021, March 3). What is Overfitting? Www.ibm.com. https://www.ibm.com/cloud/learn/overfitting#:~:text=Overfitting%20is%20a%20concept%20in

[48] IBM Cloud Education. (2020, August 17). What are Neural Networks? Www.ibm.com; IBM. https://www.ibm.com/cloud/learn/neural-networks

[49] Build with AI | DeepAI. (2017). DeepAI. https://deepai.org/machine-learning-glossary-and-terms/neural-network

[50] What are Neural Networks? (n.d.). www.ibm.com. Retrieved November 14, 2022, from https://www.ibm.com/cloud/learn/neural-networks#:~:text=Neural%20networks%20reflect%20the%20behavior

[51] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. Communications of the ACM, 60(6), 84–90

[52] Wu, H. and Gu, X. (2015). Towards dropout training for convolutional neural networks. Neural Networks, 71, pp.1–10. Doi:10.1016/j.neunet.2015.07.007

[53] Ker, J., Wang, L., Rao, J., & Lim, T. (2018). Deep Learning Applications in Medical Image Analysis. IEEE Access, 6, 9375–9389. https://doi.org/10.1109/ACCESS.2017.2788044

[54] Albawi, S., Mohammed, T. A., & Al-Zawi, S. (2017). Understanding of a Convolutional Neural Network. 2017 International Conference on Engineering and Technology (ICET), 1–6. https://doi.org/10.1109/icengtechnol.2017.8308186

[55] Amor, E. (2020, May 29). Understanding Non-Linear Activation Functions in Neural Networks. ML Cheat Sheet. https://medium.com/ml-cheat-sheet/understanding-non-linear-activation-functions-in-neural-networks-152f5e101eeb#:~:text=What%20does%20non%2Dlinearity%20mean

[56] Sharma, S., Sharma, S., & Athaiya, A. (2017). Activation functions in neural networks. towards data science, 6(12), 310-316

[57] Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 6(02), 107-116

[58] Shahriar Real (2020). Triple Pool Net: A novel robust Convolution neural network for image/content classification. UWSpace. http://hdl.handle.net/10012/16539

[59] Agarap, Abien Fred (2018). Deep Learning using Rectified Linear Units (ReLU). [online] arXiv.org. Available at: https://arxiv.org/abs/1803.08375

[60] Albawi, S., Mohammed, T. A., & Al-Zawi, S. (2017). Understanding of a Convolutional Neural Network. 2017 International Conference on Engineering and Technology (ICET), 1–6. https://doi.org/10.1109/icengtechnol.2017.8308186

[61] Srivastava, N., Hinton, G., Krizhevsky, A. and Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research,

[online] 15, pp.1929–1958. Available at: https://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf

[62] Badrinarayanan, V., Kendall, A., & Cipolla, R. (2017). SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 39(12), 2481–2495. https://doi.org/10.1109/tpami.2016.2644615

[63] Shivam Kalra (2018). Content-based Image Retrieval of Gigapixel Histopathology Scans: A Comparative Study of Convolution Neural Network, Local Binary Pattern, and Bag of visual Words. UWSpace. http://hdl.handle.net/10012/13226

[64] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., & Levenberg, J. (n.d.). TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/45166.pdf

[65] Keras. (2019). Home - Keras Documentation. Keras.io. https://keras.io

[66] Ramesh, S. (2018, May 26). A guide to an efficient way to build neural network architectures- Part II: Hyper-parameter…. Medium. https://towardsdatascience.com/a-guide-to-an-efficient-way-to-build-neural-network-architectures-part-ii-hyper-parameter-42efca01e5d7

[67] Shah, T. (2017, December 6). About Train, Validation and Test Sets in Machine Learning. Towards Data Science; Towards Data Science. https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7

[68] Kim, H.E., Cosa-Linan, A., Santhanam, N. et al. Transfer learning for medical image classification: a literature review. BMC Med Imaging 22, 69 (2022). https://doi.org/10.1186/s12880-022-00793-7

[69] Talo, M., Baloglu, U. B., Yıldırım, Ö., & Acharya, U. R. (2019). Application of deep transfer learning for automated brain abnormality classification using MR images. Cognitive Systems Research, 54, 176-188

[70] ImageNet. https://www.image-net.org/update-mar-11-2021.php. Accessed 14 November 2022

[71] Krizhevsky A, Sutskever I, Hinton GE. ImageNet classification with deep convolutional neural networks. In: Pereira F, Burges CJC, Bottou L, Weinberger KQ, editors. Advances in neural information processing systems 25. Curran Associates, Inc.; 2012. p. 1097–105

[72] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2014). Going Deeper with Convolutions. ArXiv.org. https://arxiv.org/abs/1409.4842

[73] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. arXiv:14091556 [cs]. 2015

[74] He K, Zhang X, Ren S, Sun J. Deep Residual Learning for Image Recognition. In: 2016 IEEE conference on computer vision and pattern recognition (CVPR). Las Vegas, NV, USA: IEEE; 2016. P. 770–8

[75] Tan, M., & Le, Q. V. (2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. ArXiv.org. https://arxiv.org/abs/1905.11946

[76] Vapnik VN (1999). The nature of statistical learning theory (Statistics for Engineering and Information Science). Springer, Heidelberg

[77] Jiang, J., Trundle, P., & Ren, J. (2010). Medical image analysis with artificial neural networks. Computerized Medical Imaging and Graphics, 34(8), 617–631. https://doi.org/10.1016/j.compmedimag.2010.07.003

[78] Handbook of Medical Imaging, Volume 2. Medical Image Processing and Analysis | (2000) | Fitzpatrick | Publications | Spie. (n.d.). Spie.org. Retrieved November 14, 2022, from https://spie.org/Publications/Book/831079?SSO=1

[79] Bankman, I. N. (2008). Handbook of Medical Image Processing and Analysis (Academic Press series in biomedical engineering). Academic Press

[80] J. H. Morra, Z. Tu, L. G. Apostolova, A. E. Green, A. W. Toga and P. M. Thompson, "Comparison of AdaBoost and Support Vector Machines for Detecting Alzheimer's Disease Through Automated Hippocampal Segmentation," in IEEE Transactions on Medical Imaging, vol. 29, no. 1, pp. 30-43, Jan. 2010, doi: 10.1109/TMI.2009.2021941

[81] Klöppel, S., Stonnington, C. M., Chu, C., Draganski, B., Scahill, R. I., Rohrer, J. D., Fox, N. C., Jack, C. R., Jr, Ashburner, J., & Frackowiak, R. S. (2008). Automatic classification of MR scans in Alzheimer's disease. Brain : a journal of neurology, 131(Pt 3), 681–689. https://doi.org/10.1093/brain/awm319

[82] Waddle SL, Juttukonda MR, Lants SK, et al. Classifying intracranial stenosis disease severity from functional MRI data using machine learning. Journal of Cerebral Blood Flow & Metabolism. 2020;40(4):705-719. Doi:10.1177/0271678X19848098

[83] Zacharaki, E.I., Wang, S., Chawla, S., Soo Yoo, D., Wolf, R., Melhem, E.R. and Davatzikos, C. (2009), Classification of brain tumor type and grade using MRI texture and shape in a machine learning scheme. https://doi.org/10.1002/mrm.22147

[84] Applications of deep learning to MRI images: A survey. (2018). Big Data Mining and Analytics, 1(1), 1–18. https://doi.org/10.26599/bdma.2018.9020001

[85] Shen, D., Wu, G., & Suk, H. I. (2017). Deep Learning in Medical Image Analysis. Annual review of biomedical engineering, 19, 221–248. https://doi.org/10.1146/annurev-bioeng-071516-044442

[86] Justin Ker, Lipo Wang, Jai Rao, and Tchoyoson Lim. Deep learning applications in medical image analysis. IEEE Access, 6:9375–9389, 2018

[87] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen A. W. M. van der Laak, Bram van Ginneken, and Clara I. Sanchez. A survey on deep learning in medical image analysis, Jun 2017

[88] A. Farooq, S. Anwar, M. Awais and S. Rehman, "A deep CNN based multi-class classification of Alz'eimer's disease using MRI," 2017 IEEE International Conference on Imaging Systems and Techniques (IST), 2017, pp. 1-6, doi: 10.1109/IST.2017.8261460

[89] Hashemzehi, R., Mahdavi, S. J. S., Kheirabadi, M., & Kamel, S. R. (2020). Detection of brain tumors from MRI images base on deep learning using hybrid model CNN and NADE. Biocybernetics and Biomedical Engineering, 40(3), 1225–1232. https://doi.org/10.1016/j.bbe.2020.06.001

[90] Chen, D. Y. T., Ishii, Y., Zhao, M. Y., Fan, A. P., & Zaharchuk, G. Using Deep Learning to Predict PET Cerebrovascular Reserve in Moyamoya Disease from Baseline MRI

[91] Hussein, R., Zhao, M., Shin, D., Guo, J., Chen, K. T., Armindo, R. D., Davidzon, G., Moseley, M., & Zaharchuk, G. (2022). Multi-task Deep Learning for Cerebrovascular Disease Classification and MRI-to-PET Translation. ArXiv:2202.06142 [Cs, Eess]. https://arxiv.org/abs/2202.06142

[92] Chen DYT, Ishii Y, Fan AP, Guo J, Zhao MY, Steinberg GK, Zaharchuk G. Predicting PET Cerebrovascular Reserve with Deep Learning by Using Baseline MRI: A Pilot Investigation of a Drug-Free Brain Stress Test. Radiology. 2020 Sep;296(3):627-637. doi: 10.1148/radiol.2020192793. Epub 2020 Jul 14. PMID: 32662761; PMCID: PMC7457949

[93] Nie, D., Zhang, H., Adeli, E., Liu, L., Shen, D. (2016). 3D Deep Learning for Multi-modal Imaging-Guided Survival Time Prediction of Brain Tumor Patients. In: Ourselin, S., Joskowicz, L., Sabuncu, M., Unal, G., Wells, W. (eds) Medical Image Computing and Computer-Assisted Intervention – MICCAI 2016. MICCAI 2016. Lecture Notes in Computer Science(), vol 9901. Springer, Cham. https://doi.org/10.1007/978-3-319-46723-8_25

[94] Hou, X., Guo, P., Wang, P., Liu, P., Lin, D. D. M., Fan, H., Li, Y., Wei, Z., Lin, Z., Jiang, D., Jin, J., Kelly, C., Pillai, J. J., Huang, J., Pinho, M. C., Thomas, B. P., Welch, B. G., Park, D. C., Patel, V. M., & Hillis, A. E. (2022). Deep-learning-enabled Brain Hemodynamic Mapping Using Resting-state fMRI. ArXiv:2204.11669 [Physics]. https://arxiv.org/abs/2204.11669

[95] Dhar, R., Falcone, G. J., Chen, Y., Hamzehloo, A., Kirsch, E. P., Noche, R. B., ... & Lee, J. M. (2020). Deep learning for automated measurement of hemorrhage and perihematomal edema in supratentorial intracerebral hemorrhage. Stroke, 51(2), 648-651

[96] Zhu, G., Chen, H., Jiang, B., Chen, F., Xie, Y., & Wintermark, M. (2022). Application of Deep Learning to Ischemic and Hemorrhagic Stroke Computed Tomography and Magnetic Resonance Imaging. Seminars in Ultrasound, CT and MRI, 43(2), 147–152. https://doi.org/10.1053/j.sult.2022.02.004

[97] Maqsood, M., Nazir, F., Khan, U., Aadil, F., Jamal, H., Mehmood, I., & Song, O. Y. (2019). Transfer learning assisted classification and detection of Alzheimer's disease stages using 3D MRI scans. Sensors, 19(11), 2645

[98] Yuan, Y., Qin, W., Buyyounouski, M., Ibragimov, B., Hancock, S., Han, B., & Xing, L. (2019). Prostate cancer classification with multiparametric MRI transfer learning model. Medical physics, 46(2), 756–765. https://doi.org/10.1002/mp.13367

[99] Ghafoorian, M., Mehrtash, A., Kapur, T., Karssemeijer, N., Marchiori, E., Pesteie, M., Guttmann, C. R. G., de Leeuw, F.-E., Tempany, C. M., van Ginneken, B., Fedorov, A., Abolmaesumi, P., Platel, B., & Wells III, W. M. (2017). Transfer Learning for Domain Adaptation in MRI: Application in Brain Lesion Segmentation. ArXiv:1702.07841 [Cs], 10435, 516–524. https://doi.org/10.1007/978-3-319-66179-7_59

[100] 10 Reasons Why Use Python For Web Development in 2022. (2021). https://www.monocubed.com/blog/why-use-python/

[101] JetBrains. (2019). PyCharm. JetBrains; JetBrains. https://www.jetbrains.com/pycharm/

[102] VMware. (2022, January 27). What is a Virtual Machine? | VMware Glossary. VMware. https://www.vmware.com/topics/glossary/content/virtual-machine.html

[103] Atlassian. (2019). Bitbucket. Bitbucket. https://bitbucket.org/product

[104] Python OOPs Concepts. (2016, May 16). GeeksforGeeks. https://www.geeksforgeeks.org/python-oops-concepts/

[105] Structuring Your Project — The Hitchhiker's Guide to Python. (n.d.). Docs.python-Guide.org. Retrieved November 14, 2022, from https://docs.python-guide.org/writing/structure/#structure-of-the-repository

[106] Folder free icons designed by Vectors Market. (n.d.). Flaticon. Retrieved November 14, 2022, from https://www.flaticon.com/free-icon/folder_235251

[107] MRI Basics. (2020). Case.edu. https://case.edu/med/neurology/NR/MRI%20Basics.htm#:~:text=Repetition%20Time%20(TR)%20is%20the

[108] scipy.interpolate.CubicSpline — SciPy v1.9.3 Manual. (n.d.). Docs.scipy.org. Retrieved November 14, 2022, from https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.CubicSpline.html

[109] Pydicom. (n.d.). Pydicom.github.io. https://pydicom.github.io/

[110] Matplotlib. (2012). Matplotlib: Python plotting — Matplotlib 3.1.1 documentation. Matplotlib.org. https://matplotlib.org/

[111] Python Software Foundation. (2019). tkinter — Python interface to Tcl/Tk — Python 3.7.2 documentation. Python.org. https://docs.python.org/3/library/tkinter.html

[112] Python 3 - GUI Programming (Tkinter) - Tutorialspoint. (n.d.). Www.tutorialspoint.com. https://www.tutorialspoint.com/python3/python_gui_programming.htm

[113] Tkinter Notebook Widget. (2021, January 6). Python Tutorial - Master Python Programming for Beginners from Scratch. https://www.pythontutorial.net/tkinter/tkinter-notebook/

[114] ImageTk Module. (n.d.). Pillow.readthedocs.io. https://pillow.readthedocs.io/en/stable/reference/ImageTk.html

[115] AFNI program -help files. (n.d.). Afni.nimh.nih.gov. Retrieved November 14, 2022, from https://afni.nimh.nih.gov/afni/doc/program_help/index.html

[116] os — Miscellaneous operating system interfaces — Python 3.8.0 documentation. (2019). Python.org. https://docs.python.org/3/library/os.html

[117] matplotlib.axes.Axes.twinx — Matplotlib 3.6.2 documentation. (n.d.). Matplotlib.org. Retrieved November 14, 2022, from https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.twinx.html

[118] Rectangle and ellipse selectors — Matplotlib 3.6.2 documentation. (n.d.). Matplotlib.org. Retrieved November 14, 2022, from https://matplotlib.org/stable/gallery/widgets/rectangle_selector.html

[119] Embedding in Tk — Matplotlib 3.6.0 documentation. (n.d.). Matplotlib.org. https://matplotlib.org/stable/gallery/user_interfaces/embedding_in_tk_sgskip.html

[120] scipy.stats.linregress — SciPy v1.5.4 Reference Guide. (n.d.). Docs.scipy.org. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.linregress.html

[121] Home. (2014, May 16). Maglev Microrobotics Laboratory. https://uwaterloo.ca/maglev-microrobotics-laboratory/

[122] Mutch, W.A.C., Mandell, D.M., Fisher, J.A., Mikulis, D.J., Crawley, A.P., Pucci, O. and Duffin, J. (2012). Approaches to brain stress testing: BOLD magnetic resonance imaging with

computer-controlled delivery of carbon dioxide. PloS One, [online] 7(11), p.e47443. doi:10.1371/journal.pone.0047443

[123] Baheti, P. (2021). Train, Validation, and Test Set: How to Split Your Machine Learning Data. V7labs.com. https://www.v7labs.com/blog/train-validation-test-set

[124] Machine Learning Glossary. (n.d.). Google Developers. https://developers.google.com/machine-learning/glossary

[125] Kingma, D.P. and Ba, J. (2014). Adam: A Method for Stochastic Optimization. [online] arXiv.org. Available at: https://arxiv.org/abs/1412.6980

[126] Team, K. (n.d.). Keras documentation: MaxPooling2D layer. Keras.io. https://keras.io/api/layers/pooling_layers/max_pooling2d/

[127] Team, K. (n.d.). Keras documentation: Flatten layer. Keras.io. https://keras.io/api/layers/reshaping_layers/flatten/

[128] tf.keras.callbacks.EarlyStopping | TensorFlow Core v2.4.0. (n.d.). TensorFlow. https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/EarlyStopping

[129] Brownlee, J. (2019, January 17). How to Accelerate Learning of Deep Neural Networks With Batch Normalization. Machine Learning Mastery. https://machinelearningmastery.com/how-to-accelerate-learning-of-deep-neural-networks-with-batch-normalization/

[130] Kandel, I. and Castelli, M. (2020).The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset. ICT Express. [online] doi:10.1016/j.icte.2020.04.010.

[131] Radiuk, P.M. (2017). Impact of Training Set Batch Size on the Performance of Convolutional Neural Networks for Diverse Datasets. Information Technology and Management Science, [online] 20(1), pp.20–24. Available at: https://itms-journals.rtu.lv/article/view/itms-2017-0003

[132] Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. arXiv:1206.5533 [cs]. [online] Available at: https://arxiv.org/abs/1206.5533

[133] Save and load Keras models | TensorFlow Core. (n.d.). TensorFlow. https://www.tensorflow.org/guide/keras/save_and_serialize

[134] tf.keras.preprocessing.image.ImageDataGenerator. (n.d.). TensorFlow. https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerato r

[135] Dinov, I.D. Model Performance Assessment. In Data Science and Predictive Analytics; Springer: Singapore, 2018; pp. 475–496

[136] Team, K. (n.d.). Keras documentation: ReduceLROnPlateau. Keras.io. https://keras.io/api/callbacks/reduce_lr_on_plateau/

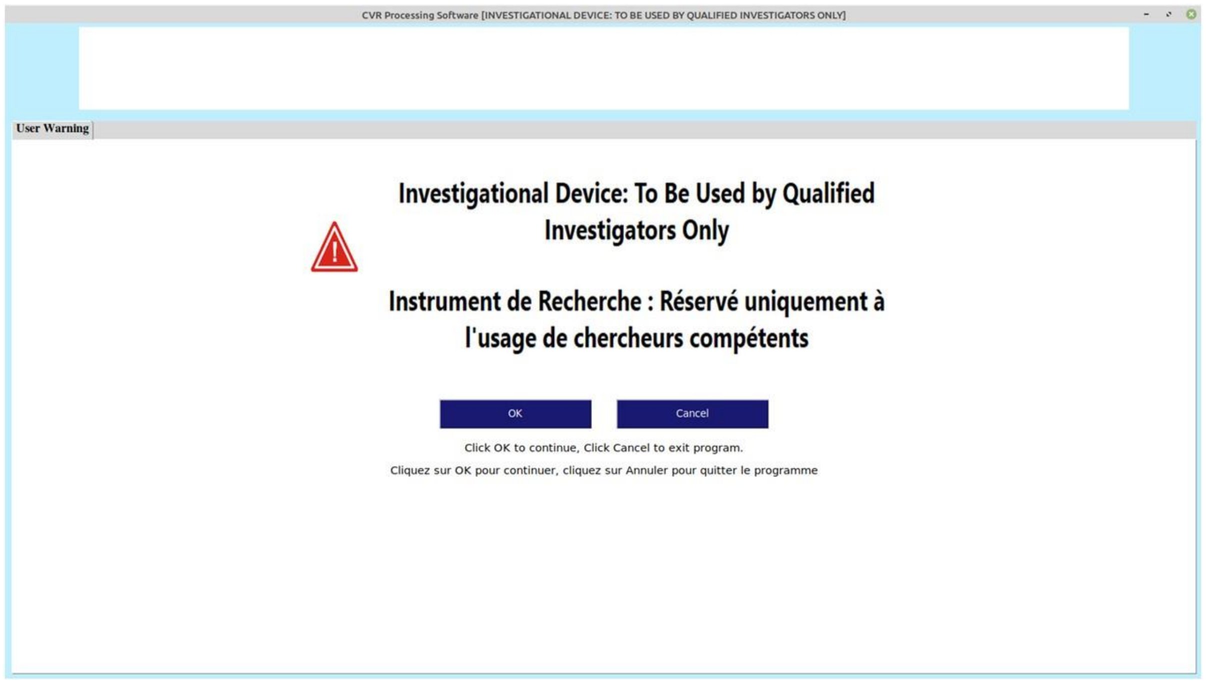
Appendices

Appendix A: CVR Software Workflow

This appendix presents a detailed workflow of the CVR processing software developed in this work.

A.1 Software Launch

On the virtual machine, the software can be launched by double-clicking on the desktop icon named "**CVR Proc Local**". A CVR Processing Software window is opened with a user warning page, as shown in Figure A.1. The warning page explicitly highlights the user's qualifications requirements.



CVR Processing Software [INVESTIGATIONAL DEVICE: TO BE USED BY QUALIFIED INVESTIGATORS ONLY]

User Warning

Investigational Device: To Be Used by Qualified Investigators Only

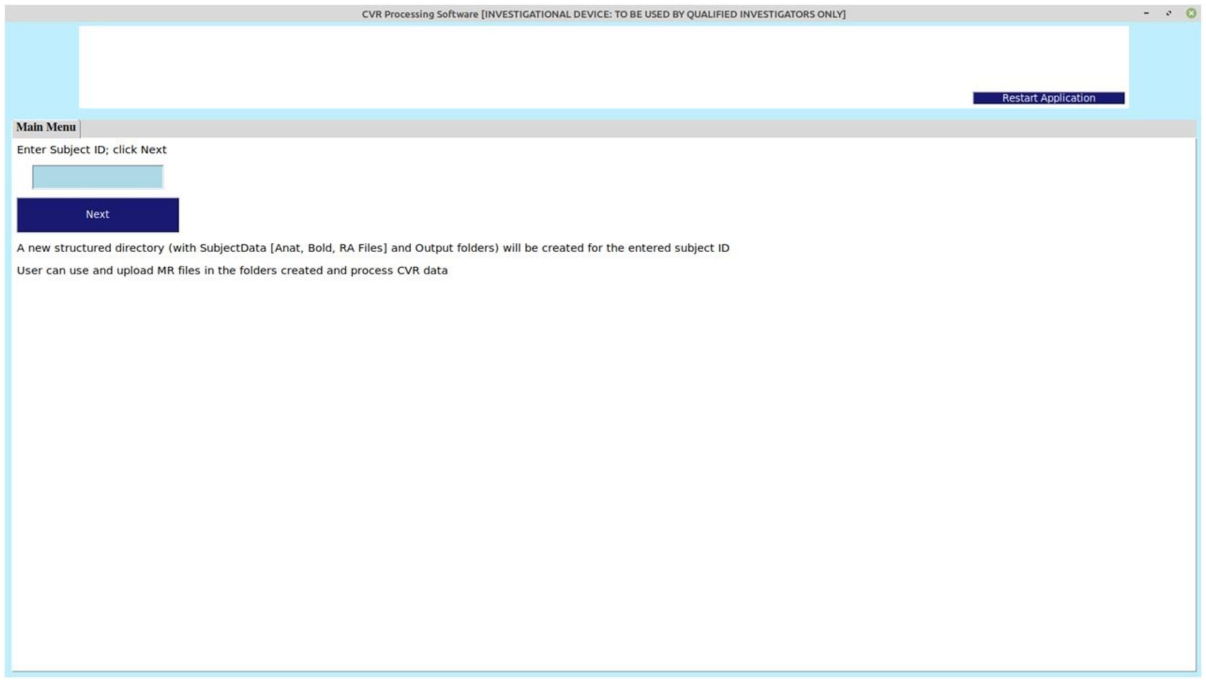
Instrument de Recherche : Réservé uniquement à l'usage de chercheurs compétents

Click OK to continue, Click Cancel to exit program.

Cliquez sur OK pour continuer, cliquez sur Annuler pour quitter le programme

Figure A.1: Software launch page. This is the first window that is displayed on executing the software application. Clicking the OK button displays the next interface. The cancel button closes the application window

The main menu tab is displayed once the user eligibility is confirmed, as shown in Figure A.2. The user is required to enter a subject ID. A subject ID corresponds to the subject under investigation. This can correspond to an existing analysis or a new one. If it is a new analysis, the software creates a structured directory for the input and output files. If the required folders already exist (from a previously completed analysis), the software reads the input and output files from these pre-populated folders.


CVR Processing Software [INVESTIGATIONAL DEVICE: TO BE USED BY QUALIFIED INVESTIGATORS ONLY]

Enter Subject ID; click Next

A new structured directory (with SubjectData [Anat, Bold, RA Files] and Output folders) will be created for the entered subject ID

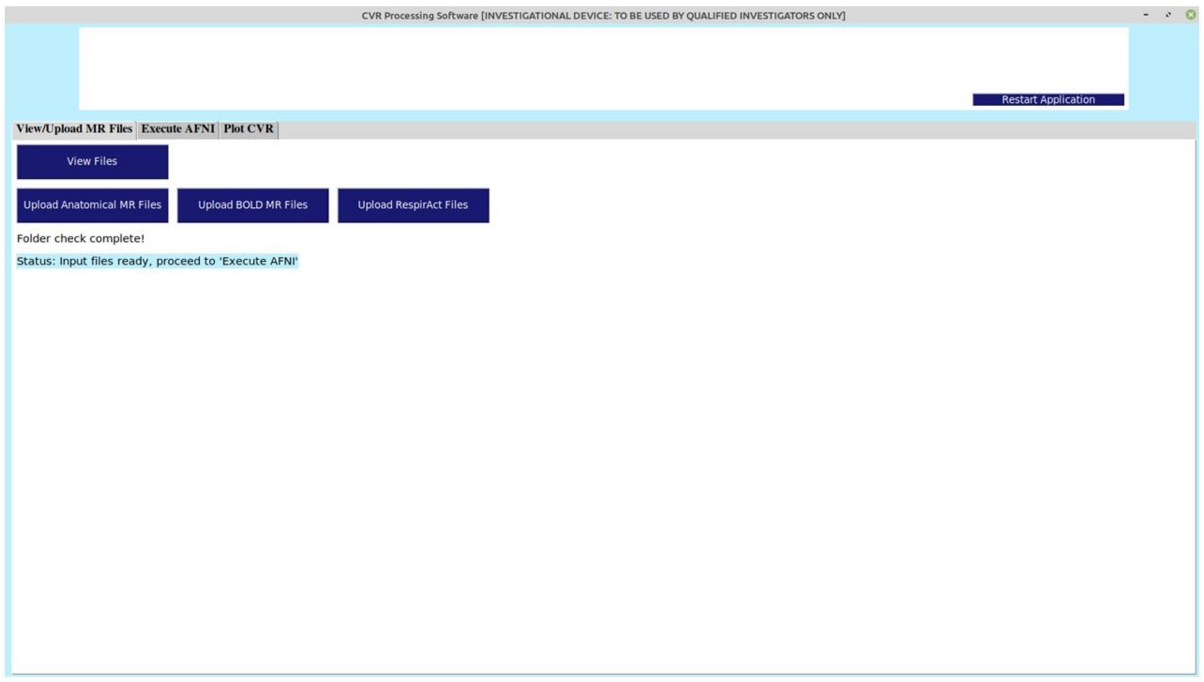
User can use and upload MR files in the folders created and process CVR data

Figure A.2: The main menu tab. The subject ID can be entered here. The Next button hides the main menu tab and opens new tabs corresponding to the next steps in the software workflow

A.2 File Management

After entering the subject ID, all the other functionalities of the software, including, View/Upload MR Files, Execute AFNI, and Plot CVR are enabled. **View/Upload MR Files** tab allows the user to view and upload the input MR files in the respective input folders, as shown in Figure A.3**.** The steps to upload new files is described below:

1. Click on the "Upload Anatomical MR Files" button.
2. Browse to the directory where your files are located.
3. Select the files to be uploaded. To select multiple files, left click on the first file using your mouse, hold the shift button on your keyboard, and left click on the last file. This will highlight all the selected files.
4. Click on the "Open" button to upload.


CVR Processing Software [INVESTIGATIONAL DEVICE: TO BE USED BY QUALIFIED INVESTIGATORS ONLY]
Restart Application
View/Upload MR Files
Execute AFNI
Plot CVR
View Files
Upload Anatomical MR Files
Upload BOLD MR Files
Upload RespirAct Files

Folder check complete!

Status: Input files ready, proceed to 'Execute AFNI'

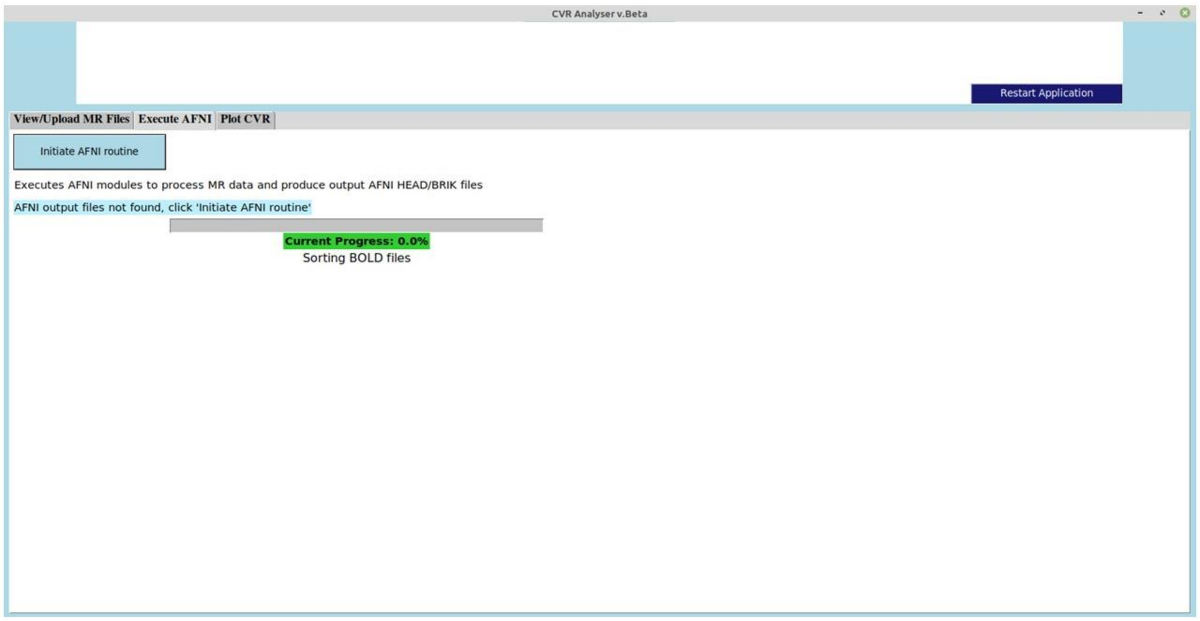
Figure A.3: View/Upload MR Files tab. The View Files button displays the input files in the respective folders. Upload buttons allow the user to upload the required input files. The status below the button provides information about the file checks

A.3 Processing MRI Files

As described in **section 4.6**, the next step uses the AFNI programs to produce the HEAD and BRIK files required to produce CVR maps. These output files are stored in the output directory.

Execute AFNI tab is used to view the interface to accomplish the task. The "Initiate AFNI routine" button makes a system call to the required AFNI programs. The software starts the AFNI routine, and a progress bar is displayed to show the progress, as shown in Figure A.4.

If the user has already completed this step for the entered subject ID, the software prompts a status, *AFNI output files already exist*, *proceed to 'Plot CVR'*, and this step of executing AFNI workflow can be skipped. In this case, the button on display is "Re-initiate AFNI routine". Re-initiating the AFNI routine deletes the existing AFNI HEAD/BRIK files in the output directory and executes the AFNI routine from the first step.



Executes AFNI modules to process MR data and produce output AFNI HEAD/BRIK files

AFNI output files not found, click 'Initiate AFNI routine'

Current Progress: 0.0%

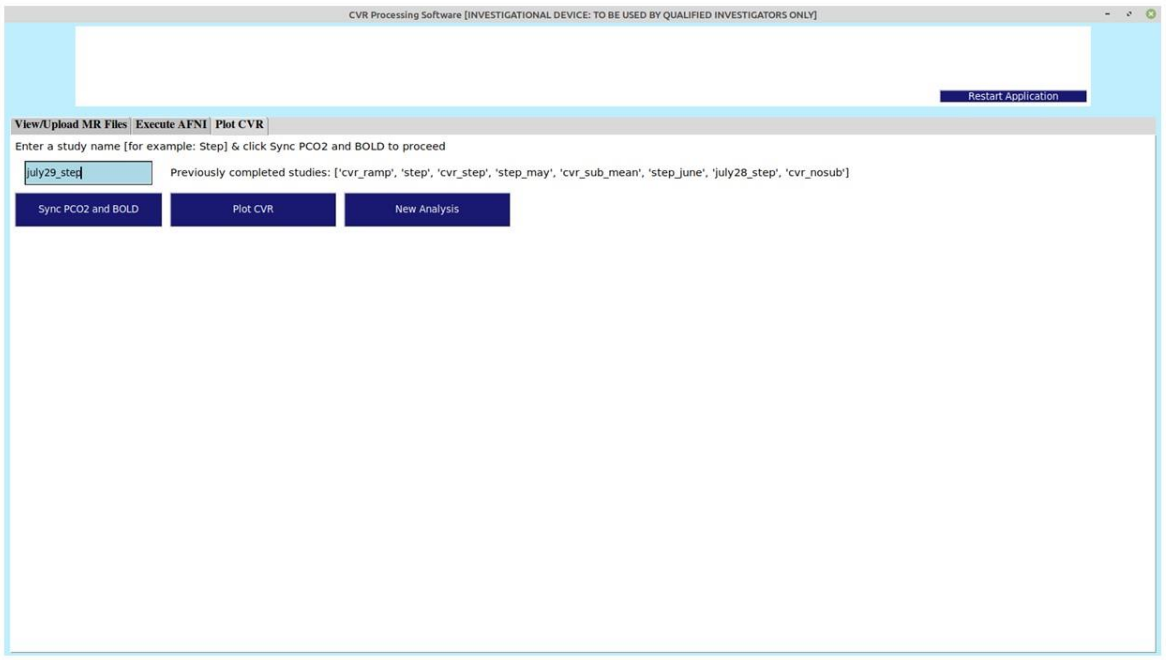
Sorting BOLD files

Figure A.4: Execute AFNI tab after initiating the processing routine. Note the progress bar shows the current progress along with a brief description of the undergoing process.

A.4 Plotting Workflow

Once the AFNI output files are ready in the output directory (as described in section 4.6), the next step is to extract BOLD information from the output file [voxels.1D], perform CVR calculations, and display the CVR maps. To access the plotting options, **Plot CVR** tab is used, as shown in Figure A.5.

To begin an analysis, a study name is required. The software looks for the existing output files and lists the analyses already completed. The user could view one of these existing maps by entering the same name as displayed [without the quotes].



Enter a study name [for example: Step] & click Sync PCO2 and BOLD to proceed

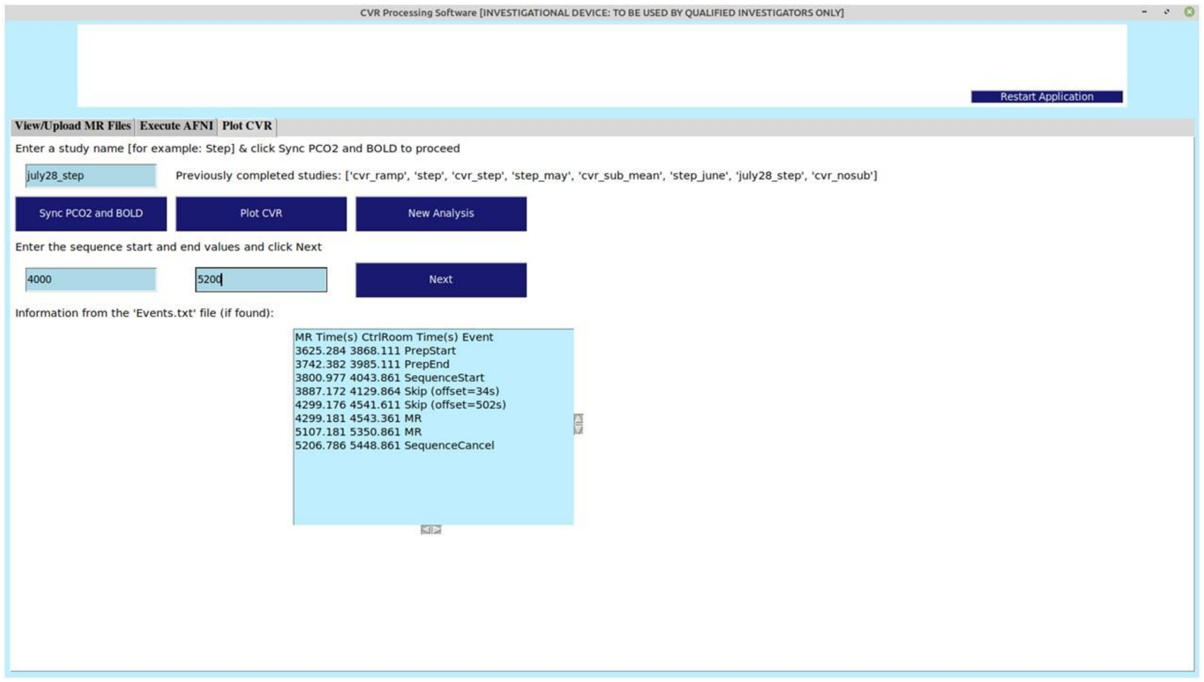
july29_step

Previously completed studies: ['cvr_ramp', 'step', 'cvr_step', 'step_may', 'cvr_sub_mean', 'step_june', 'july28_step', 'cvr_nosub']

Figure A.5: Plot CVR tab. A study name corresponding to the current analysis is required in the text entry box. A list of previously completed analyses is displayed to the right of the entry box. These can be used to directly display the CVR maps

A.4.1 Aligning the BOLD and PCO2 plots

The “Sync PCO2 and BOLD” button loads up and displays the MR sequence information from the *Events.txt* file located in the ‘RA Files’ folder. The sequence start and end values need to be entered in the two input boxes, and the “Next” button saves this information for the next step. This interface is shown in Figure A.6



Enter a study name [for example: Step] & click Sync PCO2 and BOLD to proceed

july28_step

Previously completed studies: ['cvr_ramp', 'step', 'cvr_step', 'step_may', 'cvr_sub_mean', 'step_june', 'july28_step', 'cvr_nosub']

Enter the sequence start and end values and click Next

4000

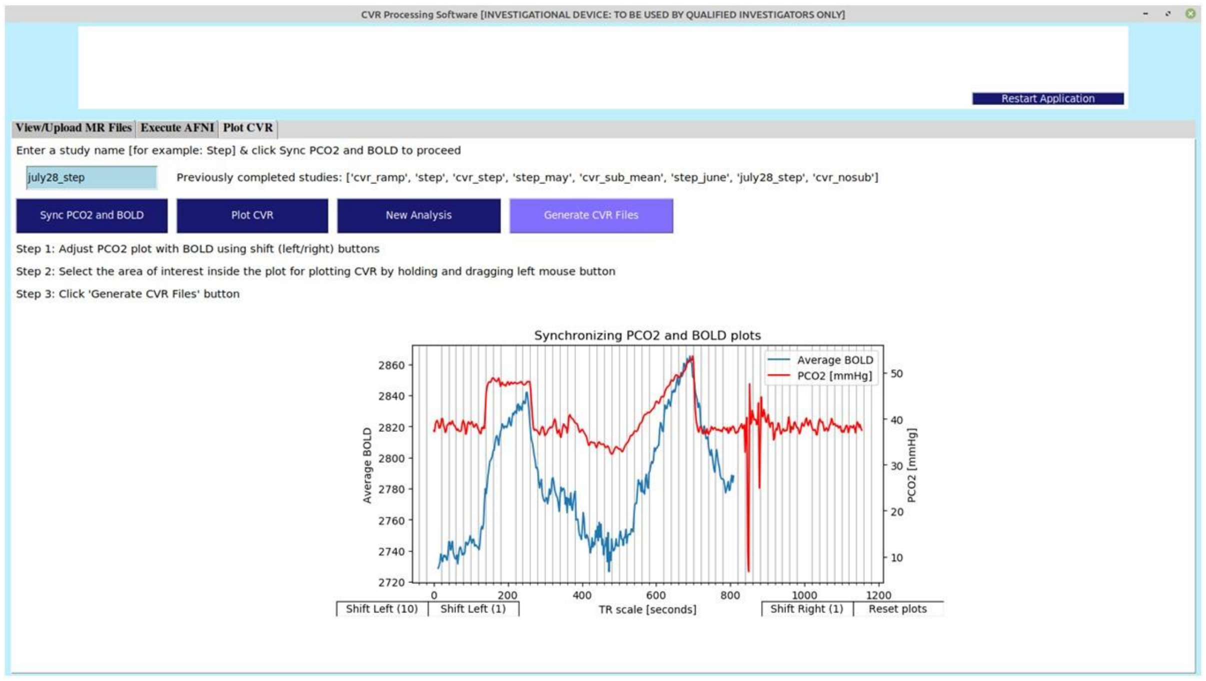
5200

Information from the 'Events.txt' file (if found):

```
MR Time(s) CtrlRoom Time(s) Event
3625.284 3868.111 PrepStart
3742.382 3985.111 PrepEnd
3800.977 4043.861 SequenceStart
3887.172 4129.864 Skip (offset=34s)
4299.176 4541.611 Skip (offset=502s)
4299.181 4543.361 MR
5107.181 5350.861 MR
5206.786 5448.861 SequenceCancel
```

Figure A.6: Displaying the MR sequence information extracted from “Events.txt” file. The start and end values are entered in the input text boxes and the Next button is clicked

On clicking the “Next” button, a set of instructions are displayed along with an interactive plot that allows the user to synchronize the PCO2 time series with the corresponding BOLD time series. The instructions and the plotting interface are shown in Figure A.7.



CVR Processing Software [INVESTIGATIONAL DEVICE: TO BE USED BY QUALIFIED INVESTIGATORS ONLY]

Enter a study name [for example: Step] & click Sync PCO2 and BOLD to proceed

july28_step

Previously completed studies: ['cvr_ramp', 'step', 'cvr_step', 'step_may', 'cvr_sub_mean', 'step_june', 'july28_step', 'cvr_nosub']

Step 1: Adjust PCO2 plot with BOLD using shift (left/right) buttons

Step 2: Select the area of interest inside the plot for plotting CVR by holding and dragging left mouse button

Step 3: Click 'Generate CVR Files' button

Figure A.7: Synchronizing PCO2 and BOLD time series. The plot interface is displayed along with a set of instructions to synchronize the two plots. Once aligned, the Generate CVR Files button is clicked

The alignment of the plots can be achieved using the steps below:

1) Use the shift buttons placed below the plotting interface to adjust the PCO2 graph along its x-axis.
2) Once the time series of both the plots are synchronized, select the area to be analyzed by dragging the left mouse button inside the plot area, as shown in Figure A.8.
3) Click on the “Generate CVR Files” button to calculate the CVR slopes for the area selected.

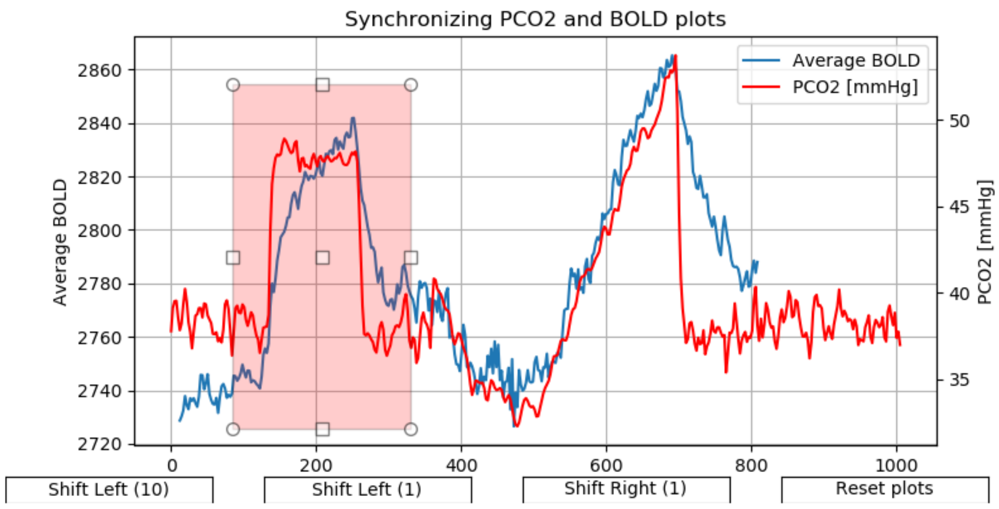


Figure A.8: Selecting the area of interest using the rectangle selector tool. The small square buttons allow re-sizing the selected area

A.4.1 Plotting CVR Maps

To plot the CVR maps for the given analysis, click on the "Plot CVR" button in the same tab, as shown in Figure A.7. This button launches an AFNI executable window that displays the CVR maps in different views. The produced maps are discussed in detail in **section 4.8**.

This is the end of the CVR processing and visualization workflow.

Appendix B: Programs and Scripts

This appendix presents and explains the Python programs and scripts used in this work.

B.1 Extracting Repetition Time

First, the Pydicom library reads the DICOM file using the read_file module. Next, the TR value (in milliseconds) is stored in a variable by searching for the corresponding index in the meta information of the DICOM file. Finally, the stored value is divided by 1000 to convert the time units to seconds. The method returns the TR value which can be imported in other modules when required.

```python
import pydicom as dcm
import os
# path is a module developed to point the software to the corresponding directories
import path

def repetition_time(self):

    mr_files = os.listdir(self.working_path.dirBold)
    ds = dcm.read_file(os.path.join(self.working_path.dirBold, mr_files[0]))
    ds_value = ds[0x0018, 0x0080]
    pieces = str(ds_value).split('"')
    # TR value is stored in milliseconds which needs to be converted to seconds, therefore division by 1000.0
    time_scale = float(pieces[-2]) / 1000.0
    return time_scale
```

B.2 Importing CO2 data

The program developed to import and process the stimulus information (the PCO2 values) from the RespirAct RA-MR device is explained here.

First, the import_pco_file module imports the file and creates a list of the entries found in the file. The data is stored in a 2D list. The first row contains the column header, and the subsequent rows contain the values.

The data_pco method imports the PCO2 data as a 2D list using the import_pco_file method. Two additional lists are created that store the real time MRI time information and PCO2 values respectively. Finally, the “None” entries are excluded from both the lists and the updated lists are returned.

```python
def import_pco_file(self):
    endtidal_file = "EndTidal.txt"
    pco_data = list()
    with open(self.working_path.dirRa + "/" + endtidal_file, "r") as d:
        for line in d:
            pco_data.append(line.strip().split('\t'))

    return pco_data

def data_pco(self, sequence_start, sequence_end):

    pco_data = self.import_pco_file()

    pco_column = 0
    mr_time_column = 0

    for i in range(len(pco_data[0])):
        if pco_data[0][i] == "PCO2 (mmHg)":
            pco_column = i
        if pco_data[0][i] == "MR Time(s)":
            mr_time_column = i

    pco_x_values = [None]*len(pco_data)
    pco_y_values = [None]*len(pco_data)
    for i in range(1, len(pco_data)):
        if sequence_start <= float(pco_data[i][mr_time_column]) <= sequence_end:
            pco_x_values[i] = pco_data[i][mr_time_column]
            pco_y_values[i] = pco_data[i][pco_column]

    # Filtering "None" values:
    pco_y_values = [float(x) for x in pco_y_values if x is not None]
    pco_x_values = [float(x) for x in pco_x_values if x is not None]

    return pco_x_values, pco_y_values
```

B.3 Data Pre-Processing

The program developed to copy the CVR maps from the complex data directories from the CVR studies is presented here.

The glob.glob modules were used to scan through the directories and sub-directories and locate all the files with the name “CVR_slope.jpg”. As these files were located, the shutil module was used to copy and paste the files while anonymizing the file names.

```
Import os
import shutil
import glob


def main():
    # Enter the path of directory which has all patient folders
    patient_data_working_path = “Directory_with_patient_folders”

    # The files are pasted into the Paste folder
    paste_path = os.path.join(os.getcwd(), “Paste”)

    # Search for all files with the name CVR_slope.jpg and copy the file with a new name
    files = glob.glob(os.path.join(“r”, patient_data_working_path, “**”, “CVR_slope.jpg”), recursive=True)
    file_itr = 1
    for f in files:
        shutil.copyfile(f, os.path.join(“r”, paste_path, “patient_” + “_” + str(file_itr) + “.jpg”))
        file_itr += 1


if __name__ == ‘__main__’:
    main()
```

B.4 Defining CNN architectures

The program used to define and modify the CNN architecture is presented here.

```
import keras.callbacks
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Dropout, Flatten, Dense

def model_definition(self):
   # Defining the sequential model
   model = Sequential()
   model.add(Conv2D(8, (3, 3), activation='relu', input_shape=(self.img_h, self.img_w, 3)))
   model.add(MaxPooling2D(pool_size=(2, 2)))
   model.add(Flatten())
   model.add(Dense(8, activation='relu'))
   model.add(Dense(1, activation='sigmoid'))

   opt = tf.keras.optimizers.Adam()
   model.compile(loss='binary_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])
   model.summary()

   return model
```

B.5 Training CNN models

The module used to import the model defined using the module described in section B.4 is explained here. Two tensor image data generators were created: train_datagen, and validation_datagen. These were used to extract data from the training and the validation directories, respectively. The callback was defined to realize the early stopping. The model that was defined earlier was imported and model.fit() was used to begin the training. As discussed in section 5.5.4, the weights and bias of the model were saved using the model.save() function.

```
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import CSVLogger


def run_model(self):

   # Training and test directories
```

```
training_dir = “The_Training_Directory”
validation_dir = “The_Validation_Directory”
train_length = 1711
validation_length = 428

# Rescaling the images
train_datagen = ImageDataGenerator(rescale=1. / 255)
validation_datagen = ImageDataGenerator(rescale=1. / 255)

train_generator = train_datagen.flow_from_directory(
   training_dir,
   target_size=(self.img_h, self.img_w),
   batch_size=self.batch_size_input,
   class_mode=’binary’)

validation_generator = validation_datagen.flow_from_directory(
   validation_dir,
   target_size=(self.img_h, self.img_w),
   batch_size=self.batch_size_input,
   class_mode=’binary’)

model = self.model_definition()

csv_logger = CSVLogger(self.model_name+”_history_log.csv”, separator=”,”, append=True)

callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10)

history = model.fit(train_generator,
               steps_per_epoch=train_length // self.batch_size_input,
               epochs=self.epochs_input,
               validation_data=validation_generator,
               validation_steps=validation_length // self.batch_size_input,
               callbacks=[csv_logger])

# Saving the trained CNN model
model.save(self.model_name+’.h5’)

return None
```

B.6 AlexNet Modeling

As discussed while designing customized CNN, a sequential model is used to define the AlexNet-inspired network. All the layers are added successively, and the layer parameters are defined for each layer. This method returns the model architecture which can be trained using the CVR dataset.

```python
import keras
import tensorflow as tf
from keras import layers


def AlexNet(self):
   model = keras.models.Sequential([
      keras.layers.Conv2D(filters=96, kernel_size=(11, 11), strides=(4, 4), activation='relu',
                      input_shape=(227, 227, 3)),
      keras.layers.BatchNormalization(),
      keras.layers.MaxPool2D(pool_size=(3, 3), strides=(2, 2)),
      keras.layers.Conv2D(filters=256, kernel_size=(5, 5), strides=(1, 1), activation='relu', padding="same"),
      keras.layers.BatchNormalization(),
      keras.layers.MaxPool2D(pool_size=(3, 3), strides=(2, 2)),
      keras.layers.Conv2D(filters=384, kernel_size=(3, 3), strides=(1, 1), activation='relu', padding="same"),
      keras.layers.BatchNormalization(),
      keras.layers.Conv2D(filters=384, kernel_size=(3, 3), strides=(1, 1), activation='relu', padding="same"),
      keras.layers.BatchNormalization(),
      keras.layers.Conv2D(filters=256, kernel_size=(3, 3), strides=(1, 1), activation='relu', padding="same"),
      keras.layers.BatchNormalization(),
      keras.layers.MaxPool2D(pool_size=(3, 3), strides=(2, 2)),
      keras.layers.Flatten(),
      keras.layers.Dense(4096, activation='relu'),
      keras.layers.Dropout(0.5),
      keras.layers.Dense(4096, activation='relu'),
      keras.layers.Dropout(0.5),
      keras.layers.Dense(1, activation='softmax')
   ])

   model.summary()
   return model
```

B.7 Importing InceptionV3

```
import keras
import tensorflow as tf
from keras import layers

from keras.applications import InceptionV3

def Inception(self):
   pre_trained_model = InceptionV3(weights='imagenet',
                                   input_shape=(224, 224, 3),
                                   include_top=False)
   pre_trained_model.trainable = False
   x = layers.Flatten()(pre_trained_model.output)
   x = layers.Dense(1024, activation='relu')(x)
   x = layers.Dropout(0.5)(x)
   x = layers.Dense(1, activation='sigmoid')(x)

   model = Model(pre_trained_model.input, x)

   model.summary()

   return model
```