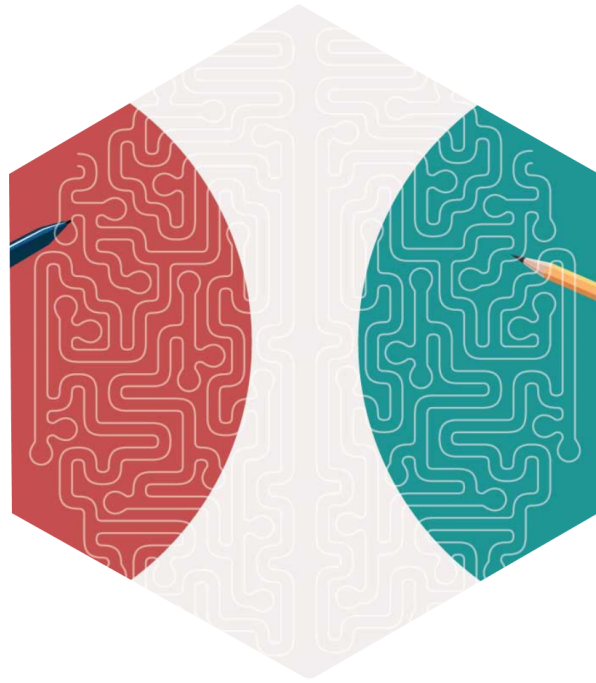


University of British Columbia

Manu 465 Final Project Group 9:

EEG: Brain & Handedness



Josiah Tsang

Lisa Skelton

Martina Chiesa

Muyang Li

Sofia Caltabiano

Simon Cotterill

December 4, 2022

[Abstract](#)

[Introduction](#)

[Motivation](#)

[Project Goal](#)

[Industry Applications](#)

[Electroencephalography \(EEG\)](#)

[Data Collection](#)

[Testing Procedure](#)

[Equipment](#)

[MindMonitor](#)

[Data Output](#)

[Data Preprocessing](#)

[Initial Preprocessing](#)

[Further Processing Steps](#)

[Exploratory Data Analysis](#)

[PCA](#)

[Machine Learning Models](#)

[K-nearest neighbors algorithm \(K-NN\)](#)

[Support vector machines \(SVM\)](#)

[Naive Bayes classifier](#)

[Decision tree](#)

[Random Forest](#)

[Logistic Regression Classifier](#)

[Artificial Neural Network](#)

[Convolutional Neural Network](#)

[Padding and Striding](#)

[3 layers of CNN](#)

[Convolutional layer](#)

[Pooling layer](#)

[Fully connected layer](#)

[Application](#)

[Conclusions](#)

[Sensitivity Analysis](#)

[Results](#)

[Further Work](#)

[Works Cited](#)

[Appendix A – Code](#)

[Appendix B – Sample RAW Data](#)

[Appendix C – CNN Analysis](#)

# Abstract

*Eras have long been defined by their technology, from the stone age to the industrial revolution and now the information age. Born from the information age is a need to process and extract meaning from incomprehensible amounts of data which is the impetus for developing machine learning models. While we try to conceive of how to train machines to make predictions, these models are often based on how we hypothesize our own brains work, ie. the term neural network. It is both ironic and poetic to then apply machine learning to better understand how our brains work. Using eeg data obtained from a Muse 2 device and software which extracts raw eeg signals, this project sought to develop an algorithm which could predict if a person was using their dominant or non-dominant hand. Results of prediction accuracy after applying various algorithms ranged between 60-80%. These results are inline with our expected results given that the best recorded accuracy of EEG classification is ~ 90% and the limitations of our experiment. The data collected and our testing methodology considered hand dominance as a binary value when in reality, people have varying degrees of hand dominance as is demonstrated by the images captured of the writing samples and resulting CNN predictions.*

## Introduction

### Motivation

Training artificial intelligence to read human brainwaves sounds like the beginning of a bad science-fiction Netflix movie, instead, it is the introduction to this study.

Brain computer interfaces (BCI) were first introduced in the 1970's ("Summary of over Fifty Years with Brain-Computer Interfaces—A Review") and in 50 years the progress shows this is much more than science fiction. Most recently, Neuralink released a video of a monkey 'typing' with its brainwave via the Neuralink chip, an implanted BCI device (*Elon Musk Shows Latest Neuralink Demo of Monkey Typing with its Mind*). A core part of BCI's is decoding what brain waves tell us, and is the basis for this project. While only a few studies exist, there is data to support that Left handed individuals perform worse on motor imagery BCI control. Handedness is strongly correlated with brain lateralization especially in right handed individuals who make up ~90% of the population (Price). Our project sought to determine if a person was using their dominant or non-dominant hand giving insight into the differences in the brain activity of right and left handed individuals.

### Project Goal

The goal of this project was to be able to accurately predict if a person was using their dominant or non-dominant hand from EEG data. Participants were fitted with the Muse 2 device and asked to perform the same writing task with both hands sequentially. EEG data was recorded from the Muse2 ("Muse 2") sensor using a third-party application, Mind Monitor ("Mind Monitor").

Participants performed 4 tasks in total and a .csv file for each task was recorded. Additional information including Task ID, Participant ID, gender and if english was their first language was added to each .csv file. Over 90 participants generated more than 360 csv files of various lengths. Evident from the collected writing samples, some participants had a more significant hand dominance which was demonstrated by the differences in both time to complete the writing sample, and quality of penmanship with the non-dominant hand. For the purpose of this study, participants self-identified as right or left handed and no ranking or gradient was applied.

## Industry Applications

The BCI industry is predicted to be an almost 5.5 Billion dollar industry by 2030 (“Brain Computer Interface (BCI) Market Poised to Garner a Valuation of USD 5.48 Billion by 2030”). Additionally, the neurofeedback industry is expected to contribute another 1.9 Billion by 2029 (“Neurofeedback Market Size, Industry Scope, & Forecast Analysis By 2029”). End user classification shows that research and medical applications dominate the sectors but there are emerging new markets including individuals and corporations.

Currently available on the market, there are hundreds of neurofeedback devices each making its own claim to improve various attributes from ADHD to sleep disorders. The validity of these claims remains to be seen. From the interaction with the Muse 2 device used in this study, the technology may need to improve significantly before reliable results can be expected. During our testing, the Muse 2 data output was sensitive to movement, hair, head shape and battery strength. Primarily devices like Muse 2 are marketed towards individuals akin to a fit bit for the brain and do not make health or medical claims.

Another subset of neurofeedback application is in neuromarketing. EEG data is used to validate or invalidate a subject's response to stimuli. This can be applied to product development, UI testing, calls to action and rebranding strategies (Lutkevich).

BCI devices offer promises of radical medical breakthroughs for individuals who have suffered from a neurological injury which includes the brain, spine and nervous system. Applications include helping paralyzed individuals control movement through exoskeletons, rehabilitation after traumatic brain injury or stroke and more.

Highlighting only a few devices and industry applications shows the diverse scope of this technology. At the core of utilizing brainwave data is being able to categorize and understand its meaning.

## Electroencephalography (EEG)

EEG's measure the electrical output, and when applied to brainwaves, the output is in the range of millivolts. Within an EEG output, the waves can be categorized into five main frequencies: delta (0.5 - 4 Hz), theta (4 - 8 Hz), alpha (8 - 13 Hz), beta (13 - 32 Hz), and gamma (32 - 100 Hz).

Alpha waves are the most common type of brain wave. They occur when you're awake but relaxed, such as when you're daydreaming. Beta waves occur when you're alert and focused. They're the least common type of brain wave. Gamma waves are associated with higher cognitive function and memory formation. They're also linked to perception and consciousness. Delta waves are the slowest type of brain wave and may indicate deep sleep. Finally, Theta brain waves occur within sleep, but only in light stages like falling asleep or waking up. They can also appear when you're awake and deeply relaxed (Healthline, 2020).

Consider the task of going from the above generalization of brainwaves and their function to succeeding at BrainPong ("BrainPong") and you will have some scope of the technological challenges with neurofeedback and BCI's. Applying machine learning algorithms to identify features and apply analysis decreases the barrier to entry of utilizing brainwave data.

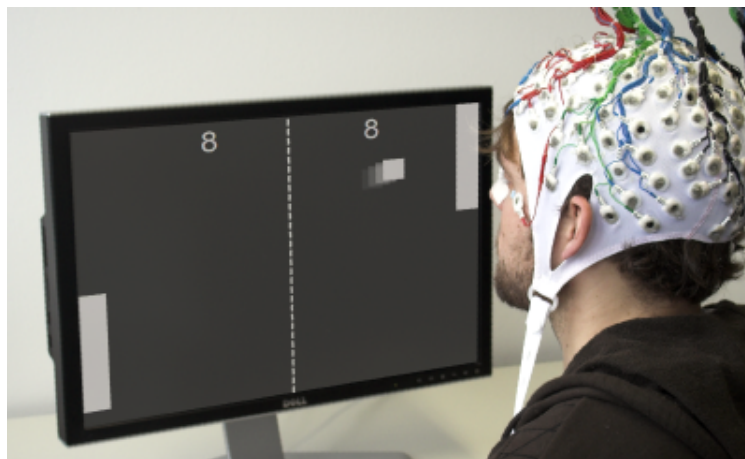


Figure 1 - BrainPong

## Data Collection

### Testing Procedure

The experiment consisted of subjects completing a complex and simple test using their dominant and non dominant hands. For these tests, subjects were asked to reprint a provided sentence, requiring some level of care, and scribble in a circle for 10 seconds, a much simpler task. We also recorded each subject's gender, dominant hand, and whether english was their first language in the event that these parameters had an impact on our results. A completed example is shown below.

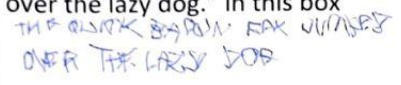
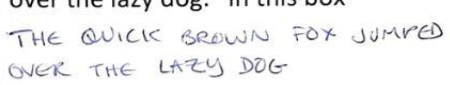
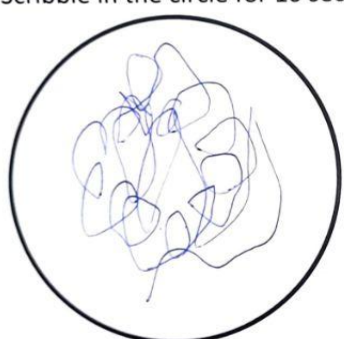

LEFT HAND SAMPLES	RIGHT HAND SAMPLES
Print "The quick brown fox jumped over the lazy dog." in this box 	Print "The quick brown fox jumped over the lazy dog." in this box 
Scribble in the circle for 10 seconds	Scribble in the circle for 10 seconds
	
Testers <u>L+S</u> Hand Dominance <u>RIGHT</u>	
Date <u>OCT 27</u> Time <u>12:30</u>	
Participant Number <u>105</u> Name (Optional) <u>LISA S.</u>	
Circle one: <input checked="" type="radio"/> Female <input type="radio"/> Male <input type="radio"/> Non-Binary <input type="radio"/> Prefer not to answer <span style="margin-left: 20px;">Is English your native language? <input checked="" type="radio"/> Yes <input type="radio"/> No</span>	

Figure 2 - Completed Test Sample

The testing of each subject took roughly five minutes to complete, including time for COVID-19 screening, cleaning the Muse 2 headset, and adding the additional features to each csv file. In total, roughly 90 participants were tested, totaling 365 unique csv files.

## Equipment

The tool provided to us for EEG analysis was the Muse 2, a consumer based EEG device geared towards meditation, sleep, and focus. The relevant sensors are depicted below, being the AF7, AF8, TP9, and TP10 sensors, placed around a user's temples and forehead.

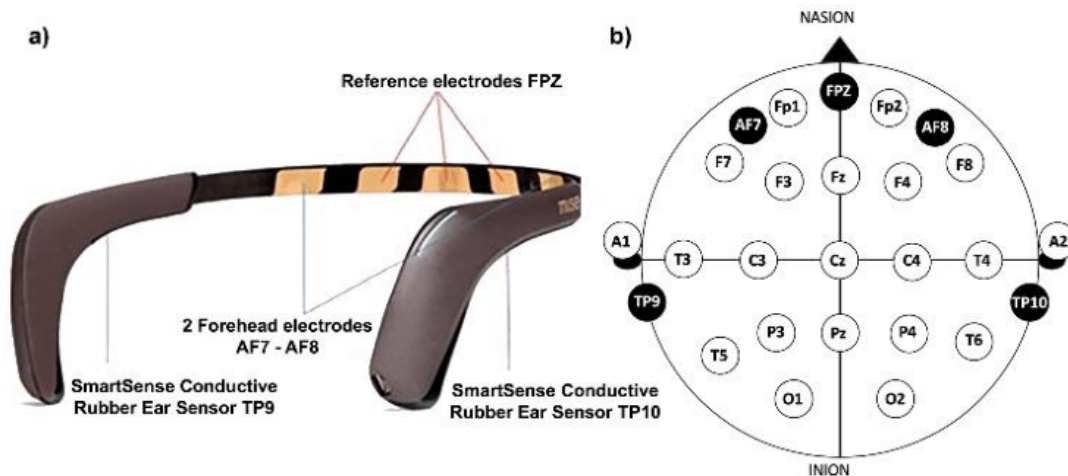


Figure 3 - Muse 2 Sensor Locations (Mansi, S. A., Pigliautile, I., Porcaro, C., Pisello, A. L., & Arnesano, M.)

As the accompanying Muse app did not offer a means to publish data as a csv or compatible file, we were advised to use the MindMonitor app.

## MindMonitor

The MindMonitor app provided us with two useful forms of data. The first form was as raw microvolt measurements from each sensor, ranging from roughly 0 to 1682 mV. The second form was as absolute brain wave values split between five brainwave frequencies, discussed in the Electroencephalography section. The brain wave values are “based on the logarithm of the Power Spectral Density (PSD) of the EEG data for each channel” (“Mind Monitor. (n.d.). Technical Manual. Mind Monitor”), with PSD being “the measure of signal's power content versus frequency ... typically used to characterize broadband random signals” (Schaldenbrand). While the MindMonitor app also included readings for acceleration, angular acceleration, battery power, and subject actions (blinking or jaw clenching), these were not considered in our analysis.

After completing our testing, we found that the default sample rate for data registered by the app was once a second. When considering the Nyquist rate, twice the highest frequency of a signal, we found that we should have taken samples every 0.011 seconds (“Nyquist rate”). This is considering the upper band of the Gamma frequencies sent by the MindMonitor app, being 44Hz, with the required sampling rate equal to 1/the frequency (“Mind Monitor. (n.d.). Technical Manual. Mind Monitor”). Unfortunately, as this was discovered near our deadline, we were unable to recollect our data. Since a Fourier Transform conducted on frequencies outside of this sampling range would equal zero, we would be unable to accurately use Fourier transforms for analysis (“Nyquist–Shannon sampling theorem”).

## Data Output

The final csv's sent by the MindMonitor app had columns for the various forms of data, with each row designating unique samples every second.

Timestamp	Delta_TP9	Delta_AF7	Delta_AF8	Delta_TP10	Theta_TP9	Theta_AF7	Theta_AF8	Theta_TP10	RAW_TP9	RAW_AF7	RAW_AF8	RAW_TP10	Participant Test	Gender	English	Dominance
43:07.5													127 RHC	Male	Yes	Right
43:07.5	0	0.955974	0.63044	1.039788	0	0.231531			811.0989	772.8205	791.7582	790.9524	127 RHC	Male	Yes	Right
43:08.5	0	0.003765	0.072892	0.994671	0	-0.25384			793.3699	775.6411	796.9963	826.0073	127 RHC	Male	Yes	Right
43:09.6	0	-0.3396	-0.26912	0.863073	0	-0.39025			846.9597	785.7143	800.2198	859.0476	127 RHC	Male	Yes	Right
43:10.6	0	-0.14214	-0.44765	0.863073	0	-0.14308			795.3846	778.4616	790.5494	803.4432	127 RHC	Male	Yes	Right
43:11.6	0	-0.37619	-0.33304	0.863073	0	-0.35361			780.8792	770.403	788.9377	772.8205	127 RHC	Male	Yes	Right
43:12.6	0	-0.26752	-0.22857	1.275989	0	-0.05308			875.5678	776.044	798.2051	845.348	127 RHC	Male	Yes	Right
43:13.6	0	-0.12623	0.002218	1.258686	0	-0.0689			794.5787	772.0147	787.7289	800.6227	127 RHC	Male	Yes	Right
43:14.6	0	0.449572	0.18539	1.258686	0	0.170049			689.011	769.1942	786.1172	782.0879	127 RHC	Male	Yes	Right
43:15.6	0	0.304969	0.06374	1.258686	0	-0.07405			803.4432	776.044	788.1319	807.8755	127 RHC	Male	Yes	Right
43:16.6	0	-0.04776	-0.10239	0.675701	0	-0.03897			816.337	785.7143	790.5494	808.6813	127 RHC	Male	Yes	Right
43:17.6	0	0.413807	0.166191	0.969467	0	0.240169			851.7949	778.4616	794.5787	840.1099	127 RHC	Male	Yes	Right
43:18.6	1.649748	0.433776	0.180397	1.19384	1.880369	-0.07054			799.4139	776.8498	792.967	835.6777	127 RHC	Male	Yes	Right
43:19.6	1.628041	0.223583	-0.29801	0.52118	1.092966	-0.16998			809.0842	770.403	788.1319	823.5897	127 RHC	Male	Yes	Right
43:20.6	1.643404	0.205837	0.25472	0.764221	0.940859	0.058531			792.1612	777.2528	792.1612	808.2784	127 RHC	Male	Yes	Right
43:21.7	1.643404	0.184698	0.316756	0.859777	0.940859	-0.15362			795.7875	769.5971	795.3846	811.5018	127 RHC	Male	Yes	Right
43:22.7	1.564125	0.280636	0.00663	0.915244	1.486856	-0.08998			802.6374	773.2235	789.7436	806.2637	127 RHC	Male	Yes	Right
43:23.7	1.638804	0.15045	-0.05948	0.939615	1.391405	0.030641			782.0879	774.8351	789.3406	810.293	127 RHC	Male	Yes	Right

Brain Wave Values

Raw EEG Values

Added Features

Figure 4 - Data csv Breakdown

The brain wave values are separated by the different frequency bands (Delta, Theta, etc.) for each sensor. The raw EEG values are similarly provided by each sensor, but not separated into the different brain waves. Periods where data either remained constant or was given as zero had to be dealt with in our data preprocessing. Finally, the additional features we collected were applied to each data point. The Participant column was used as an identifier for each subject, English noted whether English was a subject's first language, Dominance stated the user's hand dominance, and Gender noted a subject's gender. The Test column noted the hand and complexity of the test the user was completing, with RH or LH for Right Hand or Left Hand, and C or S for Complicated or Simple. Primarily blank rows, like the first 43:07:5 row in the images above, occurred when a subject action was recorded and were consequently removed.

## Data Preprocessing

### Initial Preprocessing

Once we collected all the files, one per test, our goal was to create a unique dataframe in which each row corresponds to a file.

Each csv file contains several features measured once per second, specifically, we focused on the absolute band power, which had a total of 20 columns. Four columns for each wave (Delta\_TP9, Delta\_AF7, Delta\_AF8, Delta\_TP10). The alphanumeric string indicates the sensor: TP9: left ear, AF7: left forehead, AF8: right forehead, TP10: right ear.



TimeStam	Delta_TP9	Delta_AF7	Delta_AF8	Delta_TP1	Theta_TP5	Theta_AF7	Theta_AF8	Theta_TP1	Alpha_TP5	Alpha_AF7	Alpha_AF8	Alpha_TP1	Beta_TP9	Beta_AF7	Beta_AF8
23:50.3	0.966305	0.6541	11.956.85	0.886557	0.302337	0.801309	12.115.39	0.599635	0.968283	0.607008	0.398758	0.683825	0.990106	0.083158	0.10228
23:51.3	0.966305	0.6541	11.956.85	0.886557	0.302337	0.801309	12.115.39	0.599635	0.968283	0.607008	0.398758	0.683825	0.990106	0.083158	0.10228
23:52.3	0.966305	0.6541	10.479.17	0.886557	0.302337	0.801309	10.810.57	0.599635	0.968283	0.607008	0.621275	0.683825	0.990106	0.083158	0.392358
23:53.3	0.966305	0.6541	12.401.86	0.886557	0.302337	0.801309	12.076.97	0.599635	0.968283	0.607008	0.729824	0.683825	0.990106	0.083158	0.324404
23:54.4	0.845319	0.949996	0.986042	0.886557	0.732564	-0.15453	0.053526	0.599635	13.601.99	0.281135	0.142271	0.683825	11.446.54	-0.03584	-0.02398
23:55.4	0.869692	0.711753	0.855981	0.886557	0.910449	-0.03779	-0.14335	0.599635	11.259.25	0.278763	-0.14809	0.683825	0.833421	-0.15176	-0.00347
23:56.4															
23:56.4	0.895842	0.264106	0.705403	0.886557	0.711804	-0.19938	0.062794	0.599635	0.877468	0.115311	-0.03982	0.683825	0.681562	-0.26221	-0.00724
23:57.4	12.203.29	0.527685	0.657792	0.886557	0.839803	-0.06287	0.082181	0.599635	0.732139	0.314547	0.045966	0.683825	0.677239	-0.09085	-0.09316
23:57.8															
23:58.4	12.203.29	0.422569	0.445123	0.886557	0.839803	-0.04118	0.135907	0.599635	0.732139	0.267565	0.020269	0.683825	0.677239	-0.18744	-0.07737
23:59.4	12.203.29	0.241679	0.825814	0.886557	0.839803	0.034736	0.671643	0.599635	0.732139	0.173927	0.536467	0.683825	0.677239	-0.16538	0.294244
24:00.5	12.203.29	-0.11417	0.688784	0.886557	0.839803	-0.21553	0.456665	0.599635	0.732139	0.035313	0.530301	0.683825	0.677239	-0.06641	0.193458
24:01.5	12.203.29	-0.07642	0.130562	0.886557	0.839803	-0.00707	-0.05086	0.599635	0.732139	0.250852	-0.34194	0.683825	0.677239	-0.04739	-0.07169
24:02.5	0.848797	-0.22659	0.885564	0.886557	0.709829	-0.16136	0.414339	0.599635	0.726598	-0.01163	0.644989	0.683825	0.901697	-0.14558	0.035087

Figure 5 - Example csv

In order to summarize a file such as the one in the image above in only one row, without losing information we use feature extraction technique, which allows us to discover compact and informative representations of the csv.

An important note regards the constant values. We observed that frequently the Muse registers constant values for several seconds, corresponding to a flat line on the Mind monitor app. These were shown when the patient was not actively performing the task. To get rid of these values which can influence the analysis, every value that was identical to the one 4 steps ahead was not considered.

Then we built an algorithm that, for each wave, identifies the column names containing the wave name and applies eleven functions on all values associated with these 4 columns.

Thus, for each wave we have 11 values, in total 55 values for a dataset.

These values are stored as a row of a new dataframe, associated with the four categorical variables.

The functions applied are: mean, variance, standard deviation, point to point range, maximum, minimum, arg maximum, arg minimum, skewness, kurtosis, root mean square.

Participant ◆	Dominance ◆	Delta_mean ◆	Delta_std ◆	Delta_ptp ◆	Delta_var ◆
101.0	Dominant	0.000000	0.000000	0.000000	0.000000
102.0	Dominant	0.526442	0.294538	1.336331	0.086752
103.0	NonDominant	0.340579	0.413665	1.775144	0.171119
139.0	Dominant	0.933517	0.513017	1.899951	0.263186
106.0	NonDominant	0.567189	0.504123	2.265507	0.254140
...	...	...	...	...	...
322.0	NonDominant	0.582510	0.434822	1.406913	0.189070
323.0	Dominant	0.504330	0.501552	1.719051	0.251554
324.0	NonDominant	0.204568	0.381588	1.164488	0.145609
325.0	Dominant	0.472270	0.334002	0.989795	0.111557
326.0	Dominant	0.679960	0.580307	1.580237	0.336757

Figure 6 - Resulting Dataset.

Moreover, we decided to code the response variable in 'Dominant' vs 'Non dominant'. Originally, the idea was to predict 'Left' vs 'Right' and compare the results in all the 4 tests between these two classes. However, left-handed people represent 10% of the total population, which was also observed in our data. As a result, also the *naive model* that classifies all observations in the same class will produce a good accuracy, because it will misclassify this 10% of observations and all the others will be placed in the right group ('Right handed').

As a possible solution the balancing techniques can be applied, using the method of oversampling because the dataset is small. The goal is to randomly create observations with Left handed as response, similar to the 10% already existing in the dataset, in order to have 50% of the observations in both classes. However, balancing a dataset is something artificial, and then the results will not be reliable at 100%.

Thus, we decided to consider instead 'Dominant hand' vs 'Non dominant', so the test LHC conducted by a 'Left handed' will be placed in the same class as the RHC test of a right handed. That way, the dataset is perfectly balanced and the results will be more reliable.

## Further Processing Steps

### Exploratory Data Analysis

EDA is useful to investigate relations between variables. It can be done once the data has been cleaned. With this process we can discover interesting features of the phenomenon of interest.

Through some visualization plots we noticed that the distributions of the variables are mostly skewed and not symmetrical around the mean, thus we will proceed with the standardization to make them be on the same scale.

To investigate possible multicollinearity, we plotted the correlation matrix as a square and symmetric matrix in which the  $(i, j)$  entry contains the correlation between the  $i$ -th and the  $j$ -th variables. Generally, we want a situation in which each predictor is correlated with the response variable and the correlation between pairs of predictors is not high. The reason for that comes from the desire to create a model which is able to predict the output variable using the predictors available, and is not complex. Thus, the variable selection is an important step during the construction of each model.

Through a heatmap it is possible to notice the presence of some patterns, this was expected since it is reasonable to think that some of our predictors have some similarity in the distribution. Indeed, the variable 'Alpha\_minimum' is reasonably highly correlated with 'Delta\_minimum'. Another example is the standard deviation variable for one wave and the variance variable for the same wave. The reason for that output comes from the structure of the dataset with the generation of the same features for all the waves. Considering that, we removed all the variables highly correlated.

### PCA

The principal component analysis (PCA) is a technique that transforms high-dimensions data into lower-dimensions while retaining as much information as possible. An example of a reduction from a dimension 3 to a dimension 2 is shown below:

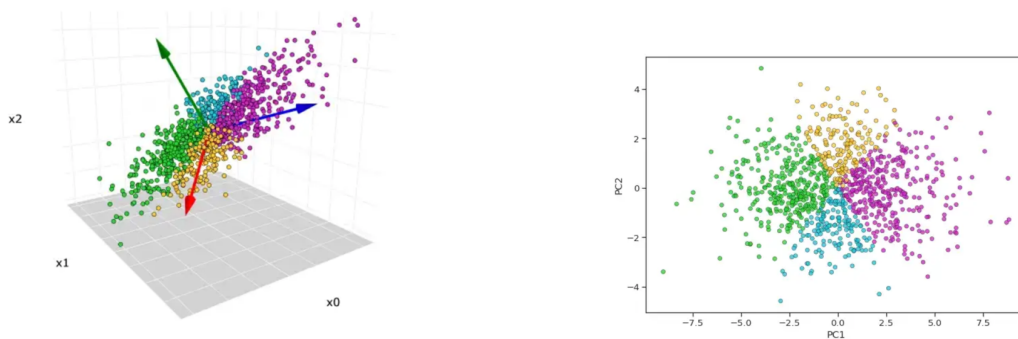


Figure 7 - PCA Reduction

PCA is extremely useful when working with datasets that have a lot of features. While having more data is always great, sometimes they have too much information in them, and we would need impossibly long model training time and that's a problem. Sometimes, less is better.

Fundamentally, PCA transforms the data into directions of maximum variance, these directions are the Eigenvectors and the variances in each direction are the corresponding eigenvalues of the covariance matrix of data. In other words: The greater the variance, the more the information and vice versa.

The basic steps when applying the PCA are: Compute the Covariance Matrix  $C$  for the matrix  $X$  of the input data, compute the eigenvectors ( $V_i$ ) and eigenvalues ( $\lambda_i$ ) of  $C$ , sort the eigenvalues from the maximum to minimum, find the principal components which are given by  $X$  multiplied by the eigenvector  $V_i$ . So,  $PCA_1 = X * V_1$ .

Often, it's not necessary to keep all the principal components but only the first few PCs and

chop off the rest. The final contribution of the PCs we used is described by:  $\frac{\sum_{i=1}^m \lambda_i}{\sum_{i=1}^n \lambda_i}$ .

We applied the Principal Component Analysis (PCA) before building the Artificial Neural Network model using the in-built Python function from SKlearn.

In our project, to compute the PCA, we imported PCA from the SKLearn library.

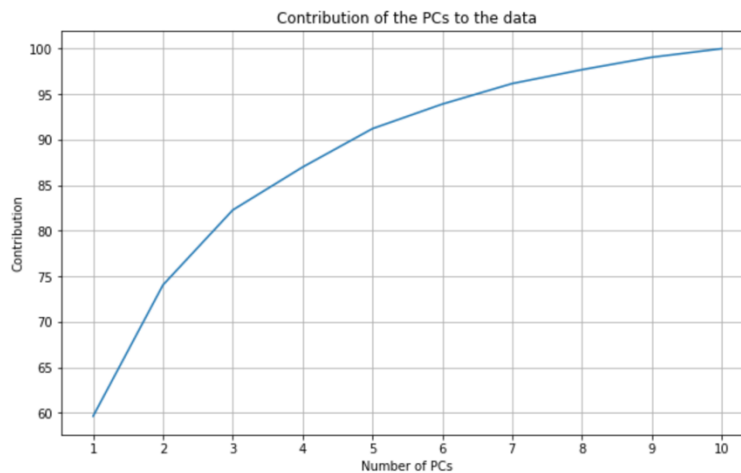


Figure 8 - Number of Principal Components

As we can see from the graph, almost 100% of the data information is contained in the first 10 principal components. Consequently, we decided to keep those 10 to compute the further analysis on SVM and ANN.

The reason why we decided to use it only before these two machine learning algorithms is because the Principal Component technique requires linear relationship between variables, however this is not reflected by all of the features in our dataset. SVM usually performs well after PCA since it reduces the dimension space and this is convenient computationally.

# Machine Learning Models

This section will introduce the algorithms used to train the machine learning models for this project. The theory behind various classification algorithms and Artificial Neural Network (ANN) are explained and the input parameters for each model are justified.

## K-nearest neighbors algorithm (K-NN)

K-nearest neighbors algorithm (k-NN) is a non-parametric supervised learning method. It uses proximity to make classifications of individual data points. The goal of the KNN algorithm is to identify the nearest neighbors of a given point, and this forms a neighborhood. The majority voting among the data records in the neighborhood is used to decide the class label of the point with or without consideration of distance-based weighting (“What is the k-nearest neighbors algorithm?”).

To do this, the distance metrics and the number of neighbors need to be decided properly. For this project, the distance metric is decided to be Euclidean distance and the number of neighbors is set as  $K=3$ . The Euclidean distance is the length of a line segment between the two points. It is commonly used to find the distance between two real-valued vectors; for this project specifically, the distance between two floating point data.

The success of classification is dependent on the selection of the K value. There are many ways of choosing the k value, but a simple one is to run the algorithm many times with different k values and choose the one with the best performance. Various k values ( $k=1,2,3,4$ ) have been tested in a controlled environment, and the result is shown in Table 1. The model accuracy is about the same 68% for all k values. However, when  $k=1$  and 3, the sensitivity and F1 score is significantly higher than the values when  $k=2$  and 4. Therefore, for this project,  $k=1$  or 3 is selected. The test accuracy is 68.06% and the F1\_score is 67.99% when k is set as 1 or 3 for the k-NN model.

Table 1 - Model accuracy for different k values of KNN

Comparison of the model accuracy for different k values of KNN				
K values	1	2	3	4
Model accuracy	0.68	0.67	0.68	0.70
Sensitivity accuracy	0.64	0.39	0.64	0.42
F_1 score	0.68	0.55	0.68	0.58

KNN modeling does not include a training period since the data itself is a model which will be the reference for future prediction (“Advantages And Disadvantages of KNN | by Anuuz Soni”). Therefore, the KNN algorithm is usually faster than other algorithms that require training. However, it also has a high cost of classifying new instances. This low efficiency prohibits its runtime performance if the model will be implemented for real-time applications.

## Support vector machines (SVM)

Support vector machines (SVM) are supervised learning models with associated learning algorithms that analyze data for classification and regression analysis (Ahmed). It can solve linear and non-linear problems as shown in Figure 9. The purpose of an SVM for classification problems is to take labeled data and determine a decision boundary, either a line or a hyperplane that separates the data. The optimal hyperplane is given by the hyperplane with the largest margin. It is calculated as the perpendicular distance from the decision boundary to the closest points, called support vectors (Ahmed).

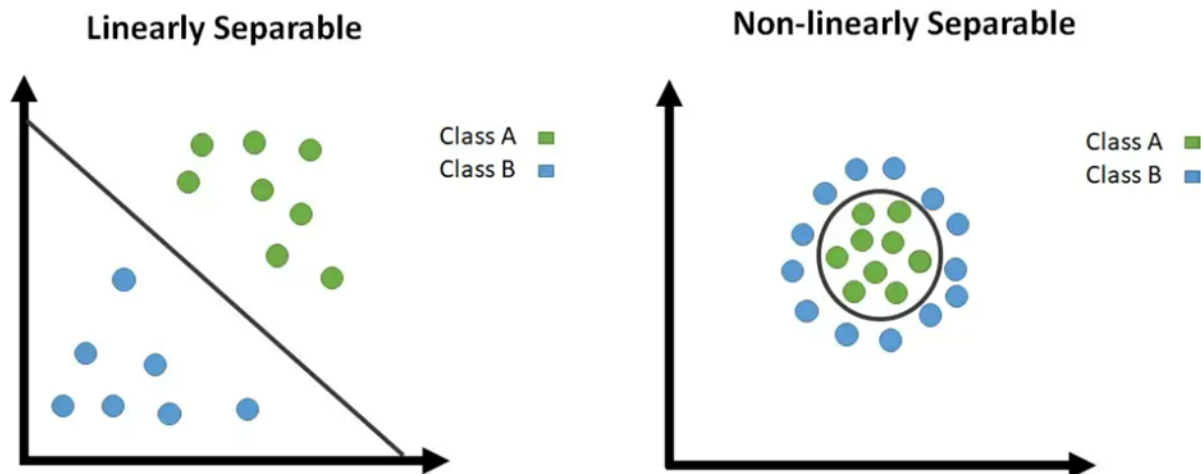


Figure 9 - SVM can solve both linearly separable and non-linearly separable problems (Ahmed)

To produce non-linear support vector machines, kernel functions are used to map the inputs into high-dimensional feature spaces where the data becomes more likely to be linearly separable (Ahmed). There are different types of kernels available in the scikit-learn library, including linear, polynomial, rbf, and sigmoid. The Gaussian radial basis function (rbf) kernel is one of the widely used kernels and is used when there is no prior knowledge about the data (Ahmed). Table 2 is a comparison of running SVM using different kernels. SVM “linear” generates the highest model accuracy and F1\_score of 82%. SVM with “sigmoid” kernel resulted in a model accuracy and F1\_score of 77%. This is followed by SVM with the “rbf” kernel which resulted in a model accuracy and F1\_score of 74%. The “poly” kernel resulted in the lowest model accuracy of 64% and F1\_score of 49%. For all SVM kernels except the “poly”, the model accuracy, sensitivity, and F\_1 score indicates the similar accuracy. For the “poly” kernel, the F1 score and sensitivity are significantly lower than the model accuracy, indicating low precision and recall of the model.

Table 2 - Comparison of different kernels for SVM

SVM kernels	linear	poly	rbf	sigmoid
Model accuracy	0.82	0.64	0.74	0.77
Sensitivity	0.83	0.33	0.72	0.73
F_1 score	0.82	0.49	0.74	0.77

PCA reduces the data dimension, and, in general, it improves algorithm performance (“What are the Pros and cons of the PCA?”). However, as shown in Table 3, both model accuracy and F1 score drop after applying PCA with 10 principal components. For SVM linear, the model accuracy drops from 77% to 71% and the F1 score drops from 77% to 59% after applying PCA. For non-linear SVM with the “rbh” kernel, the model accuracy drops from 81% to 71%, and the F1 score drops from 80% to 62%. This might be due to the information loss and the dataspace has been changed significantly from applying PCA.

Table 3 - Comparison of SVM w/o PCA

	Without PCA		With PCA	
Model	SVM Linear	SVM non-linear “rbf”	SVM Linear	SVM non-linear “rbf”
Model accuracy	77%	81%	71%	71%
F1_score	77%	80%	59%	62%

## Naive Bayes classifier

A Naive Bayes classifier is a probabilistic machine learning model based on the Bayes theorem for the classification task. It assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature and equally contributes to the outcome (Ray). There are mainly three types of naïve Bayes classifiers. Gaussian Naïve Bayes (GaussianNB) is used for this study and it assumes that values associated with each feature are normally distributed (Ray). This resulted in a test accuracy of 68.49% and 61.75%. This low accuracy suggests that the assumption that all features are independent and normally distributed might not hold true and more complex models should be used for this study.

A decision tree is a non-parametric supervised learning algorithm. Figure 10 is the decision tree plotted for this project. It has a hierarchical tree structure, which consists of a root node, branches, internal nodes, and leaf nodes. The topmost node in a decision tree is the root node. The internal nodes represent features; the branch represents a decision rule, and each leaf node represents the outcome (guide).



16



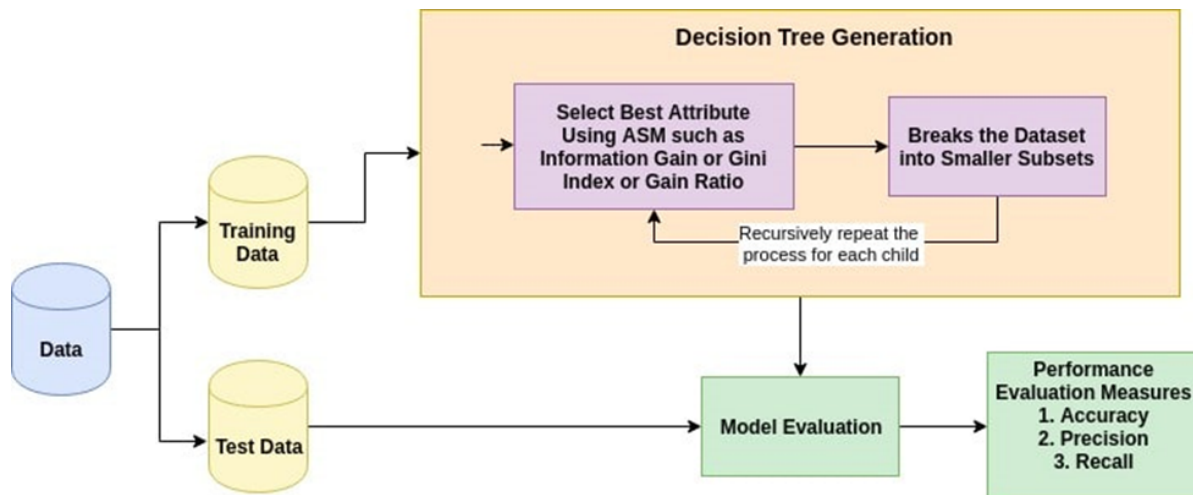


Figure 11 - Architecture of a decision tree (guide)

## Random Forest

Random Forest is a classifier that contains several decision trees on various subsets of the given dataset (Figure 12). It utilizes bagging and features randomness to create an uncorrelated forest of decision trees. Feature randomness generates a random subset of features, which ensures low correlation among decision trees. Bagging is one of the ensemble learning methods. Each tree in the ensemble consists of a data sample drawn from a training set called the bootstrap sample. Those data points can be chosen more than once. After several data samples are generated, these models are then trained independently. The output of the random forest is the class selected by most trees (“What is Random Forest?”).

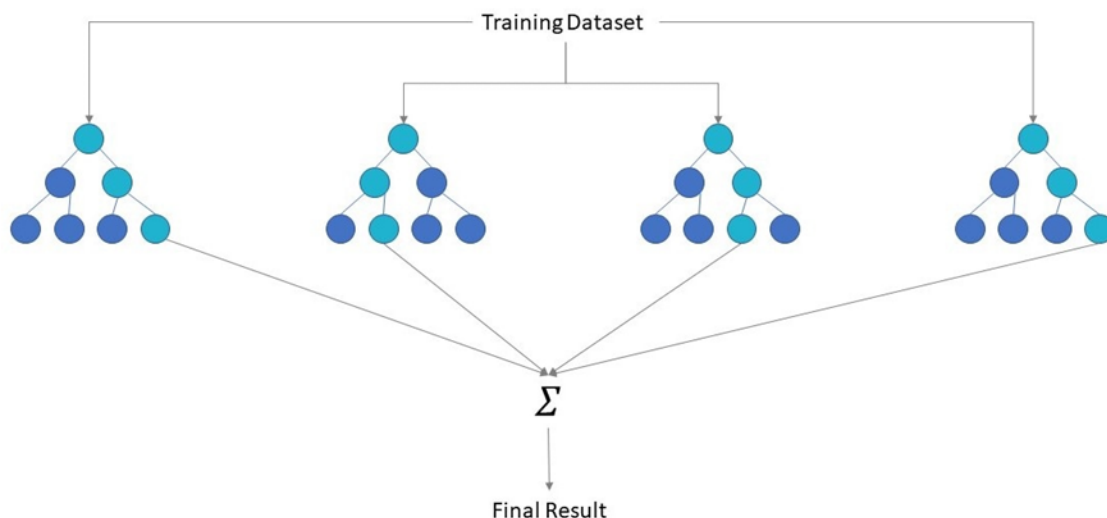


Figure 12 - Schematic of random forest (“What is Random Forest?”)

Usually, the greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting since the averaging of uncorrelated trees lowers the overall variance and prediction error. However, the main limitation of random forest is that many trees can make the algorithm too slow and ineffective for making predictions (“What is Random Forest?”).

For this study, the number of trees (n\_estimators) is set as 10 and the criterion to measure the quality of a split is selected to be “entropy”. There are three criteria available in the scikit-learn library including “gini”, “entropy”, and “log\_loss”. The entropy criterion computes the Shannon entropy, a measure of information that indicates the disorder of the features with the target of the possible classes (Aznar). The optimum split is chosen by the feature with less entropy. Computationally, entropy is more complex since it utilizes logarithms and hence it is usually slower than calculating the Gini Index (Aznar). However, the model with “entropy” criterion provides a model accuracy of 79%, the sensitivity of 69%, and F1 score of 78%, which outperforms the model built with “gini” criterion. As shown in Table 4, the “gini” criterion resulted in a model accuracy of 74%, sensitivity of 64%, and F\_1 score of 72%.

Table 4 - Comparison of random forest criterion

	Gini	Entropy
Model accuracy	0.74	0.79
Sensitivity	0.64	0.69
F_1 score	0.72	0.78

## Logistic Regression Classifier

Logistic regression (LR) estimates the parameters of a logistic sigmoid function to return a probability value. Logistic function is a common type of sigmoid function. A sigmoid function is a mathematical function having a characteristic “S”- shaped curve as shown in Figure 13. It can map any real value into another probability value within a range of 0 and 1 (Pant). The threshold value is used to define the probability of either 0 (Non Dominance) or 1 (Dominance) for classification and the default value 0.5 is used for this project.

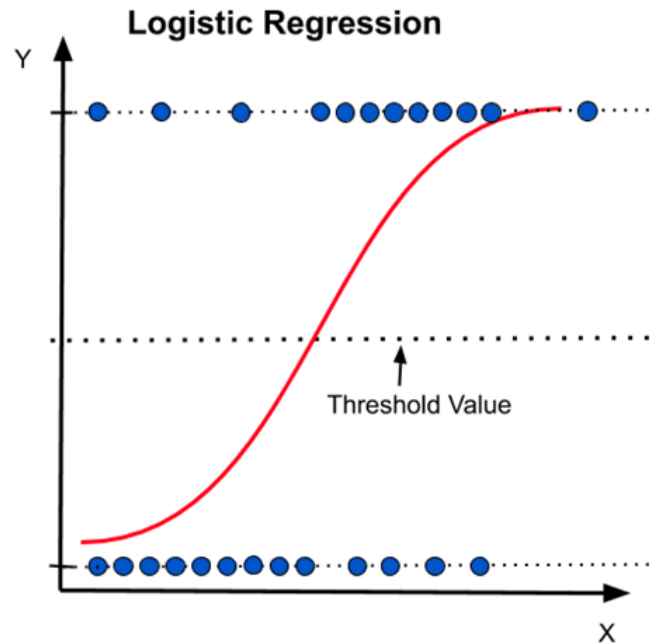


Figure 13 - The logistic regression function (Mehta)

To obtain a highly accurate logistic regression classifier, some basic assumptions that need to be met include the independence of errors, linearity in the logit for continuous variables, absence of multicollinearity, and lack of strongly influential outliers (Leung). In particular, inputs with a correlation coefficient higher than 0.85 are removed to avoid overfitting and this resulted in the test accuracy of 0.71 and the F1 score of 0.7.

## Artificial Neural Network

Artificial neural networks (ANNs) are composed of node layers, containing an input layer, one or more hidden layers, and an output layer. Each node connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network ("What are Neural Networks?").

Neural networks can be classified into several types depending on their purposes, including the perceptron, feedforward neural networks, convolutional neural networks, recurrent neural networks, etc ("What are Neural Networks?").

Feedforward neural networks are the focus of this project. It allows signals to travel one way only, from input to output. The multilayer perceptrons (MLP) architecture is a typical example of feedforward neural networks. MLP is a layered feedforward neural network, meaning that neurons are arranged in successive layers, and the information flows unidirectionally from the input layer to the output layer through the hidden layer(s) (Figure 14). There should not be lateral or feedback connection between nodes.

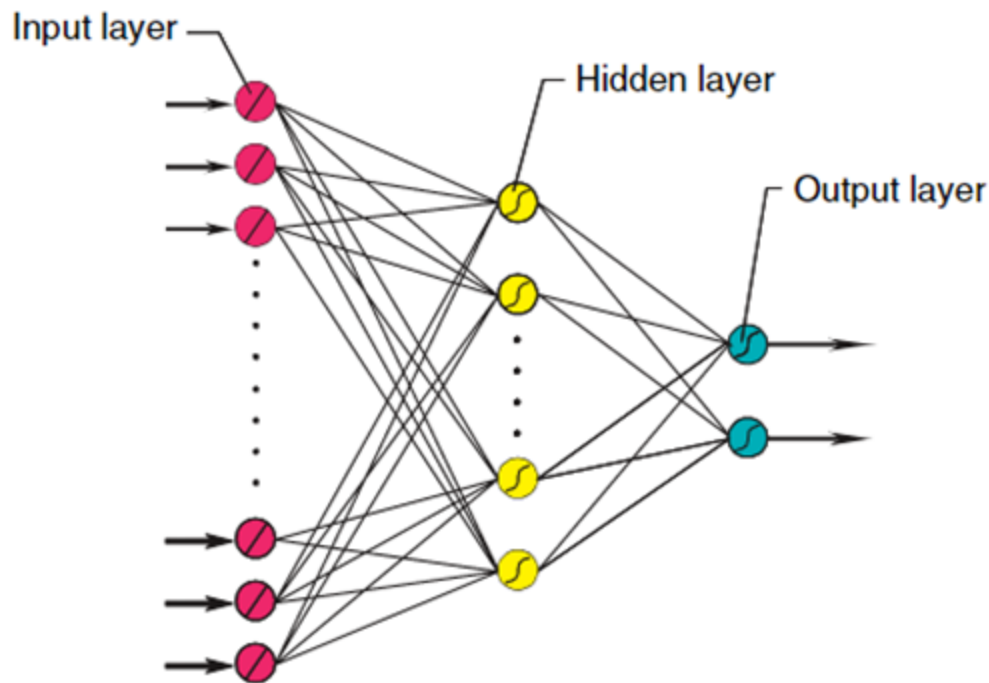


Figure 14 - Schematic of three-layered feedforward neural network (Lek)

The input layer passes on the information to the hidden layer. Hidden layers perform computation on the features passed from the input layer and transfer the result to the output layer. The output layer brings the information learned through the hidden layer and delivers the result (Baheti).

To train an ANN model, an activation function, a function that decides whether a neuron should be activated or not, needs to be determined. For this study, the activation functions for input and hidden layers are selected to be “relu”, and “sigmoid” is decided for the output layer. Both of them are nonlinear functions that allow the nodes to learn complex structures in the data.

Rectified Linear Unit (relu) is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero (Brownlee). It is one of the most used activation functions in ANN since it reduces the saturation and vanishing gradient issues that exist in the “sigmoid” functions (Brownlee).

Table 5 - Summary of the three ANN models

	Model 1	Model 2	Model 3
Numbers of hidden layers	1	1	1
Hidden layers features	Units=64	Units=22	Units=6
Model test accuracy	54.17%	56.94%	58.33%
Averaged prediction accuracy	64.75%	69.71%	64.78%

Three ANN models have been tested. All three models in the study were composed of a single input with 10 neurons, a hidden layer and an output layer with one neuron. Different numbers of hidden layers. Model 1 contains 64 neurons; model 2 contains 22 neurons, and model 3 contains 6 neurons. As shown in Table 5, Model 1 resulted in the highest test accuracy 58.33% and model 3 resulted in the highest prediction accuracy of 69.71%. For all three models, the validation accuracy is higher than the testing accuracy. This might indicate that the validation dataset is not big enough or the test dataset is less complex or random than the test dataset. Moreover, as shown in Figure 15, the loss function diverges after 50 epochs which indicates an overfitting of the model.

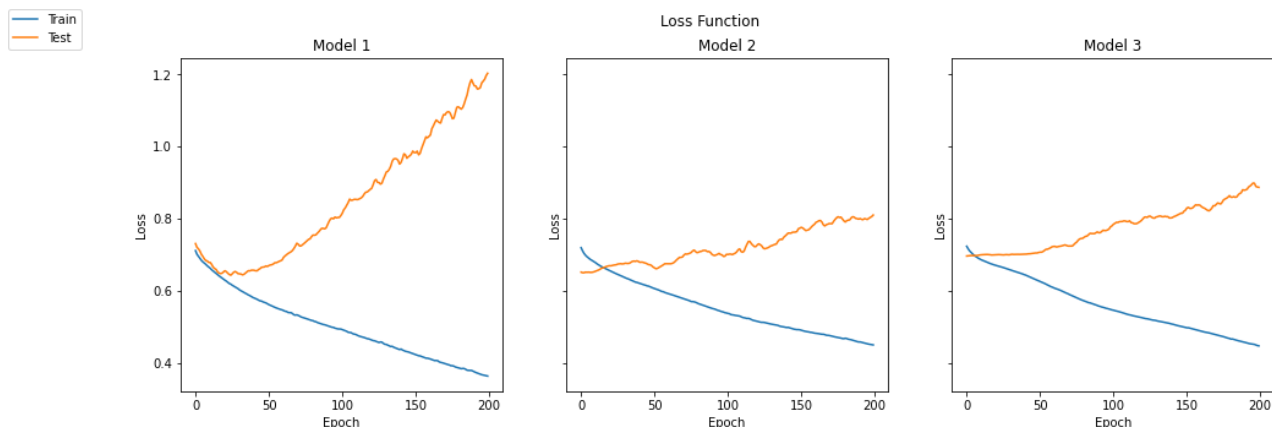


Figure 15 - The loss function of ANN models

# Convolutional Neural Network

Convolutional neural networks are often used in machine learning for image recognition tasks. A convolutional neural network is a neural network that uses convolution, a mathematical operation, that allows the merging of two sets of information. In the case of CNN, convolution is applied to the input data to filter the information and produce a feature map.

Input							Filter / Kernel			
0	1	1	0	1			1	0	1	
0	1	1	0	1			1	1	1	
0	1	1	0	1			0	0	1	
0	1	1	0	1						
0	1	1	0	1						

Figure 16 - Kernel Filter

This filter is also called kernel. To perform convolution, the kernel goes over the input image, doing matrix multiplication element after element. The result is written down in the feature map. It's necessary to continue sliding the filter until the feature map is complete:

Input

0x1	1x0	1x1	0	0
0x1	1x1	0x1	1	0
1x0	1x0	0x1	1	1
0	0	1	1	0
0	1	1	0	0

Filter / Kernel

2		

Figure 17 - Kernel Filter continued

## Padding and Striding

Two techniques that are often used in CNNs are padding and striding. Padding expands the input matrix by adding “fake” pixels to the borders of the matrix. This is done because convolution reduces the size of the matrix. For example, a 5x5 matrix turns into a 3x3 matrix when a filter goes over it.

When working with a convolutional layer, it could happen that we would like to get an output that is smaller than the input. A possible way to achieve this is to use a pooling layer. Another way to achieve this is to use striding. The idea behind stride is to skip some areas when the kernel slides over: for example, skipping every 2 or 3 pixels. It reduces spatial resolution and makes the network more computationally efficient.

Most images have 3 dimensions: height, width, and depth. Depth corresponds to color channels (RGB). So, the convolutional filter needs to be 3-dimensional as well.

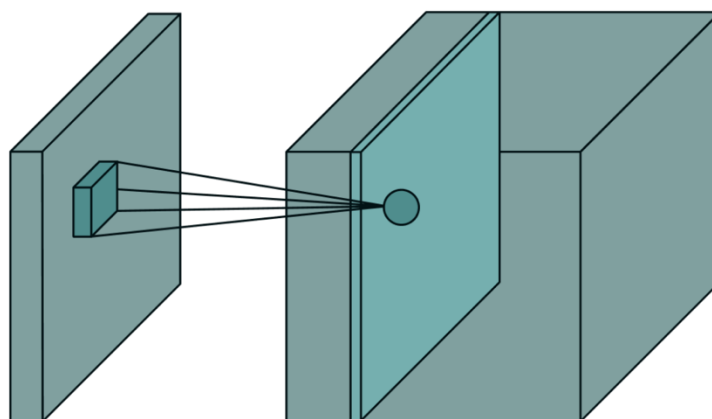


Figure 18 - Padding and Striding

### 3 layers of CNN

The goal of CNN is to reduce the images so that it would be easier to process without losing features that are valuable for accurate prediction.

There exists three different layers: convolutional layer, pooling layer, and fully connected layer. The convolutional layer is responsible for recognizing features in pixels. The pooling layer is responsible for making these features more abstract. The fully connected layer is responsible for using the acquired features for prediction.

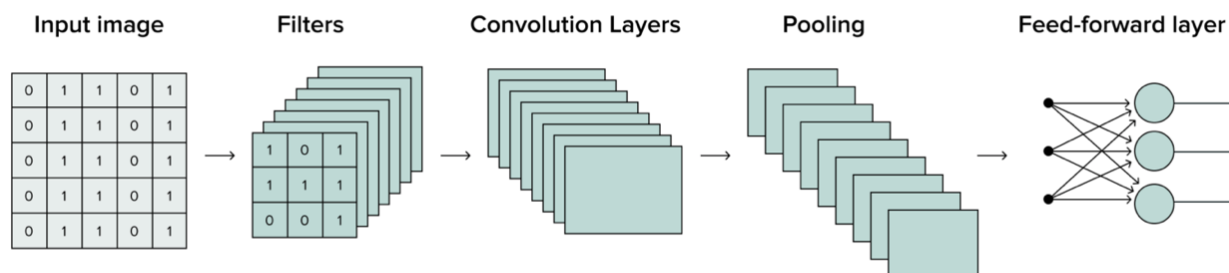


Figure 19 - CNN Process

## Convolutional layer

The convolutional layers are at the center of CNNs, enabling them to autonomously recognize features in the images. This process generates a large amount of data, which makes it hard to train the neural network. For this reason, we need to compress the data using the pooling layer.

## Pooling layer

A pooling layer receives the result from a convolutional layer and compresses it. The filter of a pooling layer is always smaller than a feature map. Usually, it takes a 2x2 square (patch) and compresses it into one value. Multiple different functions can be used for pooling. The most frequent are: Maximum Pooling, which calculates the maximum value for each patch of the feature map and Average pooling, which calculates the average value for each patch on the feature map. After using the pooling layer, you get pooled feature maps that are a summarized version of the features detected in the input. The pooling layer improves the stability of CNN. Without pooling, the slightest fluctuations in pixels would cause the model to misclassify, now small changes in the location of the feature in the input detected by the convolutional layer will result in a pooled feature map, the result will be that the feature is in the same location. The last step is to flatten the input (turn it into a column vector) and pass it to a regular neural network for classification.

## Fully connected layer

The flattened output is fed to a feed-forward neural network and backpropagation is applied at every iteration of training. This layer provides the model with the ability to finally understand images: there is a flow of information between each input pixel and each output class.

## Application

Convolutional Neural Network can be used in many applications, the most common are image classification and object detection. In image classification, CNN identifies different objects on images, by recognizing valuable features.

In our project we trained our data with different machine learning algorithms however the results were not very accurate because the Muse2 instrumentation was affected by noise and disturbances while collecting data. In an effort to validate the predictions of our learning algorithms, we supplemented our EEG ML with a CNN model based on the two different classes provided by the writing samples.

To execute the CNN, we scanned the writing samples and converted them into .jpg pictures, which were divided into two different folders: Dominant and Non-Dominant. We further divided them into training and test sets. The CNN model was built to distinguish the images of the sentence made with the dominant hand from the ones made with the non-dominant. An example of the two handwriting classes is reported below:



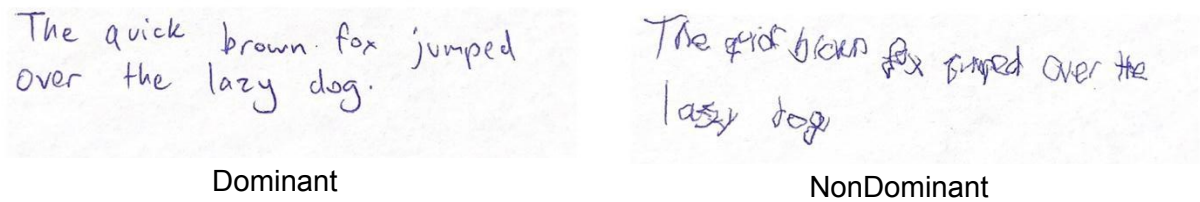


Figure 20 - Dominant and NonDominant Sentence Samples

It was not always easy to distinguish between the writing samples of a person's dominant and nondominant hand as demonstrated in the images above. Different calligraphy and degrees of hand dominance was observed making it difficult to correctly determine a person's dominant hand from the images. For this reason we have obtained a training accuracy of 73% and a test accuracy of 67% after training the CNN model.

With two different datasets and ML algorithms generated from the same test, a goal of convergence of accuracy between the models would demonstrate optimization of both algorithms, and thus reliable results.

## Conclusions

### Sensitivity Analysis

A sensitivity analysis determines how values of an independent variable affect a particular dependent variable based on a set of assumptions. The purpose of sensitivity analysis is to understand how the outputs change over a range of possible input features.

The F-score is another measurement of a mode's accuracy on a dataset, used to evaluate binary classification like in our case ("F-Score Definition"). The F-score formula is as follows:

$$F_1 = \frac{2}{\frac{1}{\text{recall}} \times \frac{1}{\text{precision}}} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

$$= \frac{\text{tp}}{\text{tp} + \frac{1}{2}(\text{fp} + \text{fn})}$$

Figure 21 - F-score

Where:

Precision = fraction of true positives among examples that the model classified as positive

Recall = fraction of examples classified as positive out of total number of positives

tp = number of true positives classified by the model

fn = number of false negatives classified by the model

fp = number of false positive classified by the model

We performed a sensitivity and F-score comparison on each of the machine learning models. The results are shown below in Table 5.

Table 5 - Sensitivity Analysis

<b>Model</b>	<b>Sensitivity</b>	<b>F1_score</b>
K-Nearest-Neighbor	51.72 %	58.25 %
SVM-Linear	72.41 %	76.83 %
SVM Non-Linear	75.86 %	80.10 %
Naive Bayes Classifier	51.72 %	65.93 %
Random Forest	68.97 %	73.55 %
Decision Tree	68.97 %	74.84 %
Logistic Regression	68.97 %	72.20 %

## Results

### Accuracy

The accuracy of each model was determined using accuracy\_score from the SKlearn.metrics library. The accuracy of each machine learning classifier is summarized in Table 6 below:

Table 6 - Classifier Results

<b>Classifier</b>	<b>Accuracy</b>
K-Nearest Neighbor	59.68 %
SVM-Linear	77.42 %
SVM Non-Linear	80.65 %
Naive Bayes Classifier	72.58 %
Random Forest	74.19 %
Decision Tree	75.81 %
Logistic Regression	72.58 %
Artificial Neural Network	69.35 %

From the results above, the classifier with the highest accuracy was the Non-Linear Support Vector Machine model with an accuracy of 80.65%. In general, all models performed decently well and we are satisfied with the outcome of the SVM Non-Linear classifier. However, further work may be conducted that could possibly increase the accuracy of our model.

## Further Work

The nature of our experiment requires a large dataset to make a machine learning model with a high accuracy. Future work on this project can include increasing the volume of data by conducting more tests on individuals. However, we already collected four samples from nearly 100 subjects and reached the limit on the amount of data we could collect for the scope of this project.

In our preliminary literature review, we also reviewed a study by M.Z Baig et al. (Baig #) that implemented a Single-Organized-Map (SOM) algorithm. This article used a pre-existing dataset that required more advanced components to collect data than the MUSE headset we are provided. The SOM based algorithm is an advanced unsupervised model that does not exist with the pre-existing Python tools we have learned in class and achieved an 84.17% classification accuracy according to the study, which is similar to the results that we achieved. However, it would still be worthwhile in the future to build this classifier with our dataset and compare its accuracy with the results of the SVM Non-Linear model.

# Works Cited

“Advantages And Disadvantages of KNN | by Anuuz Soni.” *Medium*, 2 July 2020,

<https://medium.com/@anuuz.soni/advantages-and-disadvantages-of-knn-ee06599b9336>.

Accessed 4 December 2022.

Ahmed, Mazen. “Linear Support Vector Machines Explained | by Mazen Ahmed | Medium.” *Mazen*

*Ahmed*, 11 February 2021,

<https://linguisticmaz.medium.com/support-vector-machines-explained-8804cac06883>. Accessed

4 December 2022.

Ahmed, Mazen. “Non-linear Support Vector Machines Explained | by Mazen Ahmed | Medium.” *Mazen*

*Ahmed*, 2 September 2021,

<https://linguisticmaz.medium.com/support-vector-machines-explained-ii-f2688fbf02ae>. Accessed

4 December 2022.

Aznar, Pablo. “Decision Trees: Gini vs Entropy.” *Quantdare*, 2 December 2020,

<https://quantdare.com/decision-trees-gini-vs-entropy/>. Accessed 4 December 2022.

Baheti, Pragati. “Activation Functions in Neural Networks [12 Types & Use Cases].” *V7 Labs*, 21 October

2022, <https://www.v7labs.com/blog/neural-networks-activation-functions>. Accessed 4 December

2022.

Baheti, Pragati. “Activation Functions in Neural Networks [12 Types & Use Cases].” *V7 Labs*, 21 October

2022, <https://www.v7labs.com/blog/neural-networks-activation-functions>. Accessed 4 December

2022.

Baig, M. Z. *Classification of left/right hand movement from EEG signal by intelligent algorithms*. pp

163-168 ed., IEEE Symposium on Computer Applications and Industrial Electronics, 2014.

“Brain Computer Interface (BCI) Market Poised to Garner a Valuation of USD 5.48 Billion by 2030.”

*GlobeNewswire*, 2 May 2022,

<https://www.globenewswire.com/en/news-release/2022/05/02/2433717/0/en/Brain-Computer-Inter>

[face-BCI-Market-Poised-to-Garner-a-Valuation-of-USD-5-48-Billion-by-2030-Growing-at-a-14-72-](https://www.globenewswire.com/en/news-release/2022/05/02/2433717/0/en/Brain-Computer-Inter)

[CAGR-Report-by-Market-Research-Future-MRFR.html](https://www.globenewswire.com/en/news-release/2022/05/02/2433717/0/en/Brain-Computer-Inter). Accessed 2 December 2022.

“BrainPong.” *Devpost*, 17 January 2021, <https://devpost.com/software/brainpong>. Accessed 2 December 2022.

Brownlee, Jason. “A Gentle Introduction to the Rectified Linear Unit (ReLU) - MachineLearningMastery.com.” *Machine Learning Mastery*, 9 January 2019, <https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/>. Accessed 4 December 2022.

*Elon Musk Shows Latest Neuralink Demo of Monkey Typing with its Mind*. <https://youtu.be/qatNpM3o74w>.

“F-Score Definition.” *DeepAI*, <https://deepai.org/machine-learning-glossary-and-terms/f-score>. Accessed 4 December 2022.

Gavrilova, Yulia. “What Are Convolutional Neural Networks?” *Serokell*, 3 August 2021, <https://serokell.io/blog/introduction-to-convolutional-neural-networks#how-does-a-cnn-work%3F>. Accessed 4 December 2022.

guide, step. “Python Decision Tree Classification Tutorial: Scikit-Learn DecisionTreeClassifier.” *DataCamp*, <https://www.datacamp.com/tutorial/decision-tree-classification-python>. Accessed 4 December 2022.

Lek, S. “Multilayer Perceptron.” *ScienceDirect*, 2008, <https://reader.elsevier.com/reader/sd/pii/B9780080454054001622?token=D0C31A61283E7A6B42869E61F2D8251B647228E5137BC599DFBDCD27EF88BE3279CF5BC492B68252FA35EEEC05882FDE&originRegion=us-east-1&originCreation=20221204185229>. Accessed 4 December 2022.

Leung, Kenneth. “Assumptions of Logistic Regression, Clearly Explained | by Kenneth Leung.” *Towards Data Science*, 4 October 2021, <https://towardsdatascience.com/assumptions-of-logistic-regression-clearly-explained-44d85a22b290>. Accessed 4 December 2022.

Lutkevich, Ben. “What is Neuromarketing? Definition & Examples.” *TechTarget*, <https://www.techtarget.com/searchcustomerexperience/definition/neuromarketing>. Accessed 2 December 2022.

- Mansi, S. A., Pigliautile, I., Porcaro, C., Pisello, A. L., & Arnesano, M. "Application of wearable EEG sensors for indoor thermal comfort measurements." *Acta imeko*, [https://doi.org/10.21014/acta\\_imeko.v10i4.1180](https://doi.org/10.21014/acta_imeko.v10i4.1180). Accessed 4 December 2022.
- Mehta, Ashish. "Why Is Logistic Regression Called "Regression" If It Is A Classification Algorithm?" *Artificial Intelligence in Plain English*, 29 November 2020, <https://ai.plainenglish.io/why-is-logistic-regression-called-regression-if-it-is-a-classification-algorithm-m-9c2a166e7b74>. Accessed 4 December 2022.
- "Mind Monitor." <https://mind-monitor.com/>.
- "Mind Monitor. (n.d.). Technical Manual. Mind Monitor." *Mind Monitor*, [https://mind-monitor.com/Technical\\_Manual.php#help\\_graph\\_absolute](https://mind-monitor.com/Technical_Manual.php#help_graph_absolute). Accessed 4 December 2022.
- "Muse 2." <https://choosemuse.ca/pages/product-landing>.
- Larson, J. (2020, July 1). *What Is the Purpose of Theta Brain Waves?* Healthline. Retrieved December 2, 2022, from <https://www.healthline.com/health/theta-waves>
- "Neurofeedback Market Size, Industry Scope, & Forecast Analysis By 2029." *Data Bridge Market Research*, <https://www.databridgemarketresearch.com/reports/global-neurofeedback-market>. Accessed 2 December 2022.
- "Nyquist rate." *Wikipedia*, 2022, [https://en.wikipedia.org/wiki/Nyquist\\_rate](https://en.wikipedia.org/wiki/Nyquist_rate). Accessed 4 December 2022.
- "Nyquist–Shannon sampling theorem." *Wikipedia*, 2022, [https://en.wikipedia.org/wiki/Nyquist%E2%80%93Shannon\\_sampling\\_theorem](https://en.wikipedia.org/wiki/Nyquist%E2%80%93Shannon_sampling_theorem). Accessed 4 December 2022.
- Pant, Ayush. "Introduction to Logistic Regression | by Ayush Pant." *Towards Data Science*, 22 January 2019, <https://towardsdatascience.com/introduction-to-logistic-regression-66248243c148>. Accessed 4 December 2022.
- Price, Michael. "The left brain knows what the right hand is doing." *American Psychological Association*, 1 January 2009, <https://www.apa.org/monitor/2009/01/brain>. Accessed 2 December 2022.

Ray, Sunil. "Learn Naive Bayes Algorithm | Naive Bayes Classifier Examples." *Analytics Vidhya*, 11

September 2017, <https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/>.

Accessed 4 December 2022.

Schaldenbrand, P. "What is a Power Spectral Density (PSD)?" *Siemens Communities*, 2022,

[https://community.sw.siemens.com/s/article/what-is-a-power-spectral-density-psd#:~:text=A%20Power%20Spectral%20Density%20\(PSD\)%20is%20the%20measure%20of%20signal's,amplitude%20units%20of%20g%2FHz](https://community.sw.siemens.com/s/article/what-is-a-power-spectral-density-psd#:~:text=A%20Power%20Spectral%20Density%20(PSD)%20is%20the%20measure%20of%20signal's,amplitude%20units%20of%20g%2FHz). Accessed 4 December 2022.

"Summary of over Fifty Years with Brain-Computer Interfaces—A Review." *NCBI*, 3 January 2021,

<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7824107/>. Accessed 2 December 2022.

"What are Neural Networks?" *IBM*, 17 August 2020, <https://www.ibm.com/cloud/learn/neural-networks>.

Accessed 4 December 2022.

"What are the Pros and cons of the PCA?" *i2tutorials*, 1 10 2019,

<https://www.i2tutorials.com/what-are-the-pros-and-cons-of-the-pca/>. Accessed 4 December 2022.

"What is Random Forest?" *IBM*, 7 December 2020, <https://www.ibm.com/cloud/learn/random-forest>.

Accessed 4 December 2022.

"What is the k-nearest neighbors algorithm?" *IBM*, <https://www.ibm.com/topics/knn>. Accessed 4

December 2022.

Zapała D, Zabielska-Mendyk E, Augustynowicz P, Cudo A, Jaśkiewicz M, Szewczyk M, Kopiś N, Francuz

P. "The effects of handedness on sensorimotor rhythm desynchronization and motor-imagery BCI control." *Sci Rep*. 2020 Feb 7;10(1):2087. doi: 10.1038/s41598-020-59222-w.,

<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7005877/>.

# Appendix A – Code

Python Script used to add features to individual .csv files.

## Import basic libraries

```
In [28]: import pandas as pd
```

## Manual Inputs

Copy the name of file to import and manually input parameters (only do once)

```
In [29]: # SPECIFIC TO EACH CSV FILE, CHANGE THESE 4 PARAMS
participant_number = 317
# boolean value, set to true or false
male = True
english = True
rightHanded = False
```

## Condensed Script

```
In [30]: # create array of 4 file names per test
fileName = []
fileName.append('LH-C-' + str(participant_number) + '.csv')
fileName.append('LH-S-' + str(participant_number) + '.csv')
fileName.append('RH-S-' + str(participant_number) + '.csv')
fileName.append('RH-C-' + str(participant_number) + '.csv')

for fileName in fileName:

    data_new = pd.read_csv(fileName)    # Read CSV file
    size = len(data_new[:])

    # parse fileName for 'Participant' and 'Test'
    # e.g LH-C-317.csv
    removedCsv = fileName.split('.')[0]
    testNumber = removedCsv[5:]
    test = removedCsv[:2] + removedCsv[3]

    # add missing features
    data_new['Participant'] = [testNumber] * size
    data_new['Test'] = [test] * size
    data_new['Gender'] = ['Male' if male else 'Female'] * size
    data_new['English'] = ['Yes' if english else 'No'] * size
    data_new['Dominance'] = ['Right' if rightHanded else 'Left'] * size
    # export to a new CSV file
    newFileName = test + '-' + testNumber
    print(newFileName)
    data_new.to_csv(newFileName, mode='w')
```

LHC-317  
LHS-317  
RHS-317  
RHC-317



# Appendix C – CNN Analysis

## Description

To ensure the predictions we decided to supplement our EEG Machine Learning models with a CNN model based on the two different classes given by the writing samples.

## Importing the Libraries

```
In [19]: import tensorflow as tf
         from keras.preprocessing.image import ImageDataGenerator
```

## Data Preprocessing

### Preprocessing the Training Set

The input image is on RGB. Every image is made up of pixels that range from 0 to 255. We need to normalize them i.e convert the range between 0 to 1 before passing it to the model.

```
In [20]: train_datagen = ImageDataGenerator(rescale = 1./255,
                                           shear_range = 0.2,
                                           zoom_range = 0.2,
                                           horizontal_flip = True)
         training_set = train_datagen.flow_from_directory('Dataset/Training_Set',
                                                         target_size = (64, 64),
                                                         batch_size = 32,
                                                         class_mode = 'binary')
```

Found 146 images belonging to 2 classes.

The total of images in the training set is given by  $73 * 2 = 146$ .

### Preprocessing the Test Set

```
In [21]: test_datagen = ImageDataGenerator(rescale = 1./255)
         test_set = test_datagen.flow_from_directory('Dataset/Test_Set',
                                                     target_size = (64, 64),
                                                     batch_size = 32,
                                                     class_mode = 'binary')
```

Found 34 images belonging to 2 classes.

The total of images in the test set is given by  $16 * 2 = 34$ .

## Building the CNN Model

```
In [22]: # Initializing the Model
         Model = tf.keras.models.Sequential()
```

```
In [23]: # Adding First Convolution Layer
         Model.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu', input_shape=[64, 64, 3]))
```

The convolution layer is the layer where the filter is applied to our input image to extract or detect its features. A filter is applied to the image multiple times and creates a feature map which helps in classifying the input image.

```
In [24]: # Pooling the First Layer
         Model.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
```

The pooling layer is applied after the Convolutional layer and is used to reduce the dimensions of the feature map which helps in preserving the important information or features of the input image and reduces the computation time.

The pooling operation involves sliding a two-dimensional filter over each channel of feature map and summarising the features lying within the region covered by the filter.

```
In [25]: # Adding a Second Convolutional Layer
Model.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu'))
```

```
In [26]: # Pooling the Second Layer
Model.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
```

```
In [27]: # Flattening
Model.add(tf.keras.layers.Flatten())
```

The flattening step involves taking the pooled feature map that is generated in the pooling step and transforming it into a one-dimensional vector. This vector will now become the input layer of an artificial neural network.

```
In [28]: # Full Connection
Model.add(tf.keras.layers.Dense(units=128, activation='relu'))
```

The full connection step involves chaining an artificial neural network onto our existing convolutional neural network.

```
In [29]: # Output Layer
Model.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
```

We add the output layer. In this project we must have units=1 because we need to classify 2 different classes.

```
In [30]: # Compiling the CNN
Model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

## Training the CNN

```
In [31]: Model.fit(x = training_set, validation_data = test_set, epochs = 50, verbose= False)
```

```
Out[31]: <keras.callbacks.History at 0x1bb5951b100>
```

## Evaluating the Model

Firstly we evaluate the ability of the model in predicting our training set.

```
In [32]: acc_training = Model.evaluate(training_set)
print ("The accuracy of the model on the training set is", round(acc_training[1]*100, 2), "%")

5/5 [=====] - 4s 681ms/step - loss: 0.5529 - accuracy: 0.7192
The accuracy of the model on the training set is 71.92 %
```

```
In [33]: acc_test = Model.evaluate(test_set)
print ("The accuracy of the model on the test set is", round(acc_test[1]*100, 2), "%")

2/2 [=====] - 1s 51ms/step - loss: 0.5589 - accuracy: 0.6765
The accuracy of the model on the test set is 67.65 %
```