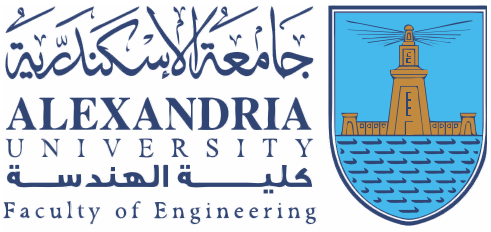


Alexandria University
Faculty of Engineering
Department of Computer & Communications Engineering -SSP-

Non-Invasive Brain Controlled Bionic Arm

Prepared By:
Abdelrahman Ayman El-Kharadeley
Aly Mohsen Mahmoud
Hebatallah Mohamed Nasef
Mina Girgis Helmy

Supervised By:
Dr.Ali Gaber

A Graduation Project submitted to the Department of Computer & Communications in partial fulfillment of the requirements for the degree of B.Sc. in Computer Engineering

Alexandria,Egypt
June, 2022

Abstract

According to Many studies there are nearly one in each 50 people that is Paralyzed and incapable of controlling his environment. The commonly known and used treatment for these individuals is an invasive brain surgery to regain their control;however these surgeries entail high risks of bleeding, infections , blood clots, strokes and commas ,also these surgeries are expensive. A relatively new technology is accessing the brain's waveforms non-invasively. This non-invasive technique is called electroencephalography (EEG) based brain-computer interfacing;weak electrical activity of the subjects' brain is recorded through a specialized, high-tech EEG cap fitted with electrodes and collects the brain's waveforms(thoughts in our case) where it gets transmitted to an attached microcontroller (raspberry pi, Nvidia Jetson,Arduino) to be processed in a trained Neural Network Machine Learning model where these signals get transformed into actions manifested into the Bionic arm Movements. In order to attempt maximum accuracy of the machine multiple machine learning models were researched & experimented with ;including [Deep Neural network"DNN" ,Convolutional Neural Network "CNN" & fully Convolutional Neural Network"FCNN", Residual Neural Network"ResNet",Recurrent Neural Network"RNN" & Recurrent Neural Network with Attention]. Maximum Accuracy has appeared in the case of CNN & FCNN models which returned nearly similar results.]

By applying this technology , we can offer safe non-invasive "mind control" devices which will allow many people to control their environment while avoiding surgical risks and expenses.

Table of Contents

List of Figures

Chapter 1

Introduction and Motivation

Contents

1.1 Problem Statement

Amputations have a devastating impact on patients' health with consequent psychological distress, economic loss, and difficult reintegration into society.

In most cases, the predominant experience of the amputee is one of loss not only the obvious loss of the limb but also resulting losses in function, self-image, career and relationships

For most of history, prosthetics have been designed to make life more comfortable for adults.It can help to perform daily activities such as walking, eating, or dressing. Some artificial limbs let people function nearly as well as before.

One of the most important effects is that it improves the amputee's mental health clearly as he regains self esteem.

People can still experience regular reflexes to aid with daily life which is important for human safety.

Robotic arms were originally designed to assist in mass production such as automotive production lines, They were also implemented to mitigate the risk of injury for workers, they are especially valued in the industrial production, manufacturing, machining and assembly sectors and to undertake monotonous tasks, so as to free workers to concentrate on the more complex elements of production.

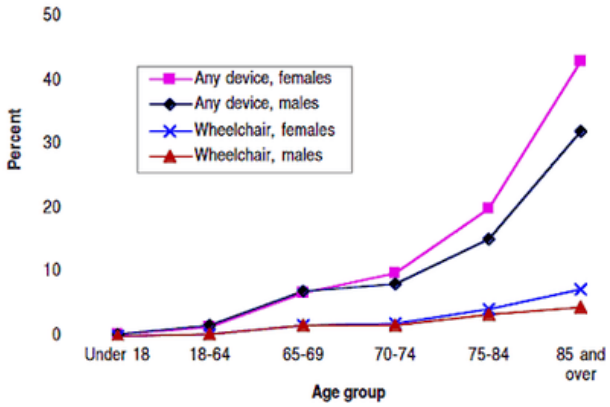


Figure 1.1: Proportion of population using mobility devices and wheelchairs,by age and gender

1.2 Methodology & Requirements

In order to achieve our previously discussed functionality, our requirements have been dissected into two types as identified below:

1.2.1 Software Requirements

- Signal preprocessing.
- Supervised Machine Learning Model.
- EEG Algorithm SDK
- EEG Dataset
- Bionic arm interface

1.2.2 Hardware Requirements

- 3D printed bionic arm model.
- EMOTIV EEG Headset (Neurosky Mindwave Headset
- Microcontroller (Raspberry PI 4, Nvidia Jetson,Arduino).
- 5 Servo motors.
- Bluetooth Module.
- Power Source "Battery".
- Hardwires.

Chapter 2

Background

Contents

2.1 Brain-computer interfaces (BCI)

A brain-computer interface (BCI) translates complex patterns of brain activity into commands that can be used to control a computer and other electronic devices. Thus, a BCI can provide a communication and control channel, which by-passes conventional neuromuscular pathways involved in speaking or making movements to manipulate objects. BCI systems are anticipated to play an important role in the development of assistive and therapeutic technologies for paralyzed patients, for prosthesis or orthosis control, and in movement rehabilitation

Data Collection in our case is defined in reading the generated Brain electrical signals from the brain,for that we need to first understand the basic functionality of the brain as well as understanding the method of data extraction which is the EEG scan;as previously mentioned,we also have to understand the type of extracted data and their expected range of values as well as what they represent.

2.2 Brain Functionality

2.2.1 Brain anatomy

The brain resides in three parts:

- Cerebrum- The bulkiest part of the human brain.
- Cerebellum- The smallest part of the human brain carries out body activities such as balancing.
- Brain stem- Every primary life functions originates inside the brain stem, like beating of heart, pressure of blood, inhalation and exhalation.

2.2.2 Cerebrum

The cerebrum is split into multiple regions:

- Frontal lobes- In control for problem solving, and decision making.
- Parietal lobes – Directs sensitivity, handwriting and body posture.
- Temporal lobes- In control for memory and hearing.
- Occipital lobes- Comprise the brain‘s visual processing system.
- motor cortex-in control of motor functions of the body.
- sensory cortex -in control of sensory receptors of the body.

Each nerve is connected to millions of different nerves where the electrical signal is passed through dendrite where the axon end converges this electrical signal to chemical signal where the neurotransmitter passes this signal. When the current leaves;it leaves a positive polarity and when it enters,it creates a negative polarity.
So we will use Electroencephalogram (EEG) to record this brain activity with their respective frequencies.

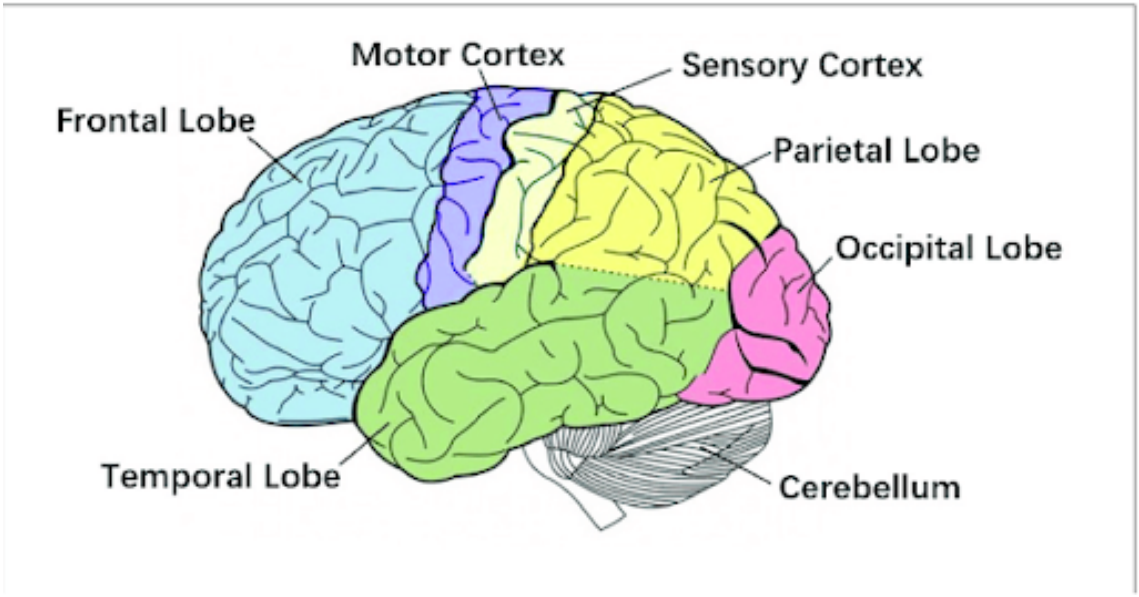


Figure 2.1: Brain parts

2.3 Electroencephalogram (EEG)

2.3.1 EEG definition

An EEG is a test that detects abnormalities in your brain waves, or in the electrical activity of your brain. During the procedure, electrodes consisting of small metal discs with thin wires are pasted onto your scalp. The electrodes detect tiny electrical charges that result from the activity of your brain cells. The charges are amplified and appear as a graph on a computer screen, or as a recording that may be printed out on paper.

Your healthcare provider then interprets the reading. During an EEG, your healthcare provider typically evaluates about 100 pages, or computer screens, of activity. He or she pays special attention to the basic waveform, but also examines brief bursts of energy and responses to stimuli, such as flashing lights. Evoked potential studies are related procedures that also may be done. These studies measure electrical activity in your brain in response to stimulation of sight, sound, or touch.

2.3.2 EEG mechanism

Electrodes are placed on the scalp to pick up the electrical current generated by the brain.
There are significant research efforts using EEG-based BCI to restore limb function in patients suffering neurological damage either from accident or disease.
The main idea is to have a control algorithm which processes EEG signals according to the user's intention.
EEG signals can be differentiated based on their:

- Amplitude.
- Frequency.
- The position of the electrodes.

Figure 2.2: EEG signals with different frequencies,amplitudes and electrodes

2.3.3 EEG Components

EEGs have multiple components of different frequencies:

- A delta component if its dominant frequency component, f<4 Hz
- A theta component if 4Hz <= f <8Hz
- A low alpha component if 8Hz <= f <12Hz
- A high alpha component if 12Hz <= f <30Hz
- A low beta component if f >= 30Hz
- A high beta component if f >= 30Hz
- A low gamma component if f >= 30Hz
- A high gamma component if f >= 30Hz

EEG montages

Montage means the placement of the electrodes. The EEG can be monitored with either a bipolar montage or a referential one. Bipolar means that you have two electrodes per one channel, so you have a reference electrode for each channel. The referential montage means that you have a common reference electrode for all the channels.

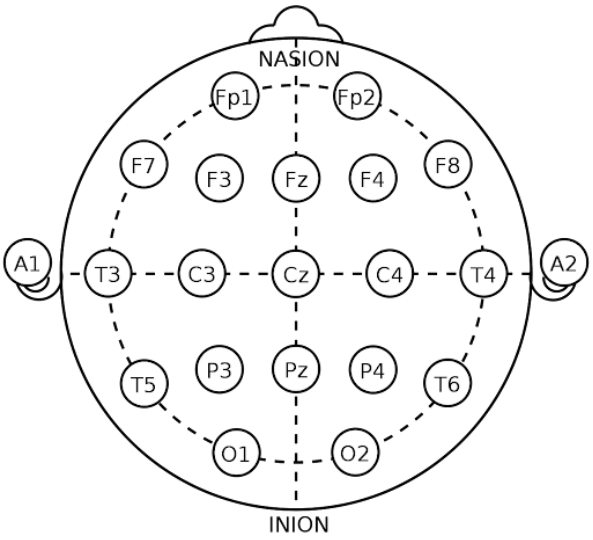


Figure 2.3: EEG electrodes positions

2.3.4 Artifacts

The biggest challenge with monitoring EEG is artifact recognition and elimination. There are patient related artifacts (e.g. movement, sweating, ECG, eye movements) and technical artifacts (cable movements, electrode paste-related), which have to be handled differently. There are some tools for finding the artifacts.
For example,Facial electromyography "FEMG" and impedance measurements can be used for indicating contaminated signal. By looking at different parameters on a monitor, other interference may be found.

2.4 EEG HEADSET NEUROHEADSET

NeuroHeadsets are devices that facilitate the EEG scanning partition as well as enhance portability.The selection of a commercially available BCI headset depended on the number of sensing channels, signal quality, price, and ease of use. The medical EEG equipment causes inconvenience to the subjects by the application of conductive gel to the scalp and the preparation routine is also time consuming. After surveying for wireless user friendly EEG headsets, we found the following headsets most suitable for research as they also provided raw EEG data.

2.4.1 Mindwave

This is a compact, reliable and low cost brain signal sensor kit, mindwave mobile from NeuroSky. Now you can not only read your mind, but also make wireless control of any device just by thinking in a certain manner!! (yes it is true).
This EEG headset can safely measures and transfers the power spectrum (alpha waves, beta waves, etc) data via RF module to wirelessly communicate with your computer, iOS, or Android device or Arduino. The output data from sensors on the headset can be simply used to enable you to see your brainwaves change in real time.
The Mindwave Mobile is simple consisting only of a headset, an ear-clip, and a sensor arm. The headset's reference and ground electrodes are on the ear clip, while the EEG electrode is on the sensor arm, resting on the forehead above the eye.

Figure 2.4: Neurosky mindwave headset

There is more than 100 brain training games, educational apps, and development tools available through the NeuroSky, iOS, and Android. You can also write your own programs to interact with MindWave Mobile by using the free developer tools.

2.4.2 Emotiv headset

The Emotiv EPOC+headset provides excellent access to professional-level brain data.As shown in Figure 3, this helmet contains fourteen active electrodes with two reference electrodes, which are Driven Right Leg (DRL) and Common Mode Sense (CMS). The electrodes are mounted around the participant's scalp in the structures of the following zones: frontal and anterior parietal (AF3, AF4, F3, F4, F7, F8, FC5, FC6), temporal (T7, T8),and occipital-parietal (O1, O2, P7, P8).

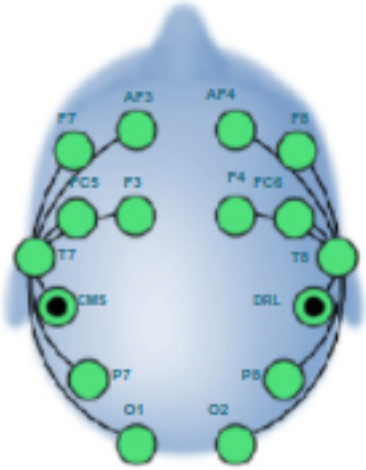


Figure 2.5: Emotiv electrodes positions

The characteristics of the Emotiv EPOC+helmet.

Characteristics	EEG Headset
Number of channels	14 (plus 2 references CMS and DRL)
Channel names	AF3, F7, F3, FC5, T7, P7, O1, O2, P8, T8, FC6, F4, F8, AF4
Sampling rate	128 SPS (2048 Hz internal)
Sampling method	Sequential sampling
Bandwidth	0.2–45 Hz, Digital notch filters at 50 Hz and 60 Hz
Resolution	14 bits
Filtration	Sinc filter
Dynamic range	8400 µV (microvolts)

Figure 2.6: Emotiv characteristics

TheEmotiv EPOC and NeuroSky MindWave were considered initially. Both devices export raw EEG as well as processed, automatically classified mental state data.
Between them, sensing capabilities were considered, where the NeuroSky MindWave uses one sensor that can provide only three values: attention, meditation, and eye blinking. The Emotiv EPOC uses a series of 14 sensors plus 2 references, which are capable of detecting specific conscious thoughts, levels of attention, facial expressions, and head movements (the latter using the embedded gyroscope). The sampling frequency of the Emotiv EPOC is 4 times greater than the NeuroSky MindWave making it comparable

to more complex EEG devices. Occasional unreliability of signal quality, is noticed with both the devices because of the use of dry electrodes. For this reason, the designers of the Emotiv headset suggest that users further improve skin conductance by the moistening of the sensors using a saline solution.
We selected the Emotiv EPOC for use in our robotic arm design, since it integrates the largest number of sensors at the highest sampling rate among all portable low-cost BCI headsets available in the market

2.5 Bionic arm

InMoov is a DIY mostly 3D printable humanoid robot controlled by Arduino micro controllers that was originally designed as a prosthetic arm by French designer Gael Langevin in 2012. Since then it has turned into a personal project for Gael with him posting new improved pieces every couple of months

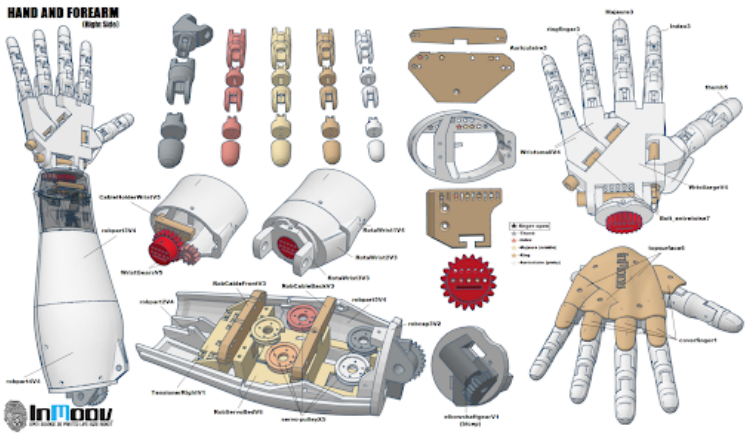


Figure 2.7: Inmoov arm components

2.6 Servo motors

There are lots of servo motors available in the market and each one has its own specialty and applications. The following two paragraphs will help you identify the right type of servo motor for your project/system. Most of the hobby Servo motors operates from 4.8V to 6.5V, the higher the voltage higher the torque we can achieve, but most commonly they are operated at +5V. The MG996R is a metal gear servo motor with a maximum stall torque of 11 kg/cm. Like other RC servos the motor rotates from 0 to 180 degree based on the duty cycle of the PWM wave supplied to its signal pin.

Figure 2.8: Servo motor

After selecting the right Servo motor for the project, comes the question how to use it. As we know there are three wires coming out of this motor. The description of the same is given on top of this page. To make this motor rotate, we have to power the motor with +5V using the Red"VCC" and Brown"Ground" wire and send PWM signals to the Orange"Control" colour wire. Hence we need something that could generate PWM signals to make this motor work

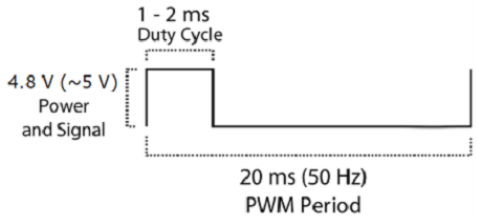


Figure 2.9: Servo torque mechanism

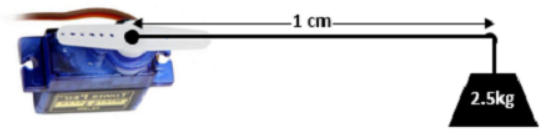


Figure 2.10: Servo torque mechanism

2.7 Adafruit PCA9685 16-Channel Servo Driver

This module is a breakout board for the NXP PCA9685 16 channel PWM controller. It features 16 fully programmable PWM outputs with a 12 bit resolution giving a total of 4096 programmable steps with a duty cycle being adjustable from 0 to 100 . Additionally, the output frequency of all 16 channels can be programmed from 24Hz to 1526Hz.

In fact, this module has been designed with this purpose in mind with 16 sets of headers that allow for any servo with a standard header to be directly plugged into the module. A screw terminal block provides a means of powering the attached servos from an external 5V PSU and so the number of servos you can drive from your microcontroller and so is not limited by the microcontroller's own power supply.

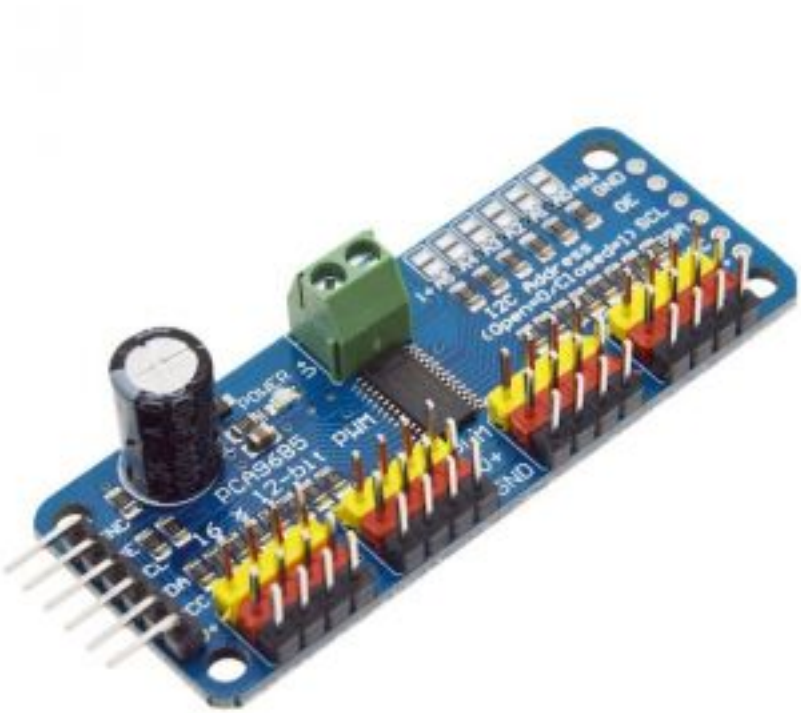


Figure 2.11: Adafruit PCA9685 16-Channel Servo Driver

The module also includes an I2C header with 10K pullup resistors and so only requires two data pins (SDA and SCL) to control the module. Solderable pads on the module provide a means of changing the default I2C address (0x40) to one of 62 options, meaning more than one module can be connected to the same I2C bus.

2.8 Arduino microcontroller

Arduino is an open-source platform used for building electronics projects. Arduino consists of both a physical programmable circuit board (often referred to as a microcontroller) and a piece of software, or IDE (Integrated Development Environment) that runs on your computer, used to write and upload computer code to the physical board.

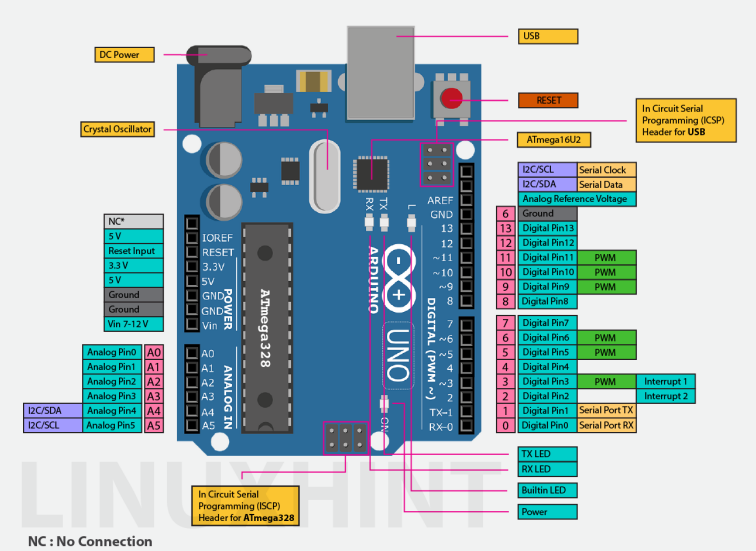


Figure 2.12: Arduino uno

The Arduino platform has become quite popular with people just starting out with electronics, and for good reason. Unlike most previous programmable circuit boards, the Arduino does not need a separate piece of hardware (called a programmer) in order to load new code onto the board – you can simply use a USB cable. Additionally, the Arduino IDE uses a simplified version of C++, making it easier to learn to program. Finally, Arduino provides a standard form factor that breaks out the functions of the micro-controller into a more accessible package.

Chapter 3

Literature Review—Neural Network Models

Contents

3.1 Dataset

This dataset was created and contributed to PhysioNet by the developers of the BCI2000 instrumentation system, which they used in making these recordings. The system is described in: Schalk, G., McFarland, D.J., Hinterberger, T., Birbaumer, N., Wolpaw, J.R. BCI2000: A General-Purpose Brain-Computer Interface (BCI) System. IEEE Transactions on Biomedical Engineering 51(6):1034-1043, 2004. [In 2008, this paper received the Best Paper Award from IEEE TBME.

3.1.1 Overview

This dataset that is being used consists of 1500 1 & 2 minutes recordings obtained from 109 subjects/ volunteers
The subjects performed different motor/imagery tasks while 64-channel EEG was being recorded using BCI 2000

3.1.2 Experiment procedures

Each subject performed 14 experimental runs: two one-minute baseline runs (one with eyes open, one with eyes closed), and three two-minute runs of each of the four following tasks:
1-A target appears on either the left or the right side of the screen. The subject opens and closes the corresponding fist until the target disappears. Then the subject relaxes.
2-A target appears on either the left or the right side of the screen. The subject imagines opening and closing the corresponding fist until the target disappears. Then the subject relaxes.
3-A target appears on either the top or the bottom of the screen. The subject opens and closes either both fists (if the target is on top) or both feet (if the target is on the bottom) until the target disappears. Then the subject relaxes.
4-A target appears on either the top or the bottom of the screen. The subject imagines opening and closing either both fists (if the target is on top) or both feet (if the target is on the bottom) until the target disappears. Then the subject relaxes.
In summary, the experimental runs were:
Baseline, eyes open
Baseline, eyes closed
Task 1 (hand opening)
Task 2 (hand clutching)
Task 3 (clockwise wrist movement)
Task 4 (anticlockwise wrist movement)
Task 1
Task 2
Task 3
Task 4
Task 1
Task 2
Task 3
Task 4

The data provided here are in EDF+ format (containing 64 EEG signals, each sampled at 160 samples per second, and an annotation channel). For use with PhysioToolkit software, rdedfann generated a separate PhysioBank-compatible annotation file (with the suffix .event) for each recording containing identical data to that of the annotation channel in the corresponding .edf file.
Number of trials is 84 trials (six trials per experiment run), which amounts to 640 data readings and each reading is done in 4 seconds.

3.1.3 Montage

The EEGs were recorded from 64 electrodes as per the international 10-10 system, as shown below . The numbers below each electrode name indicate the order in which they appear in the records; note that signals in the records are numbered from 0 to 63, while the numbers in the figure range from 1 to 64.

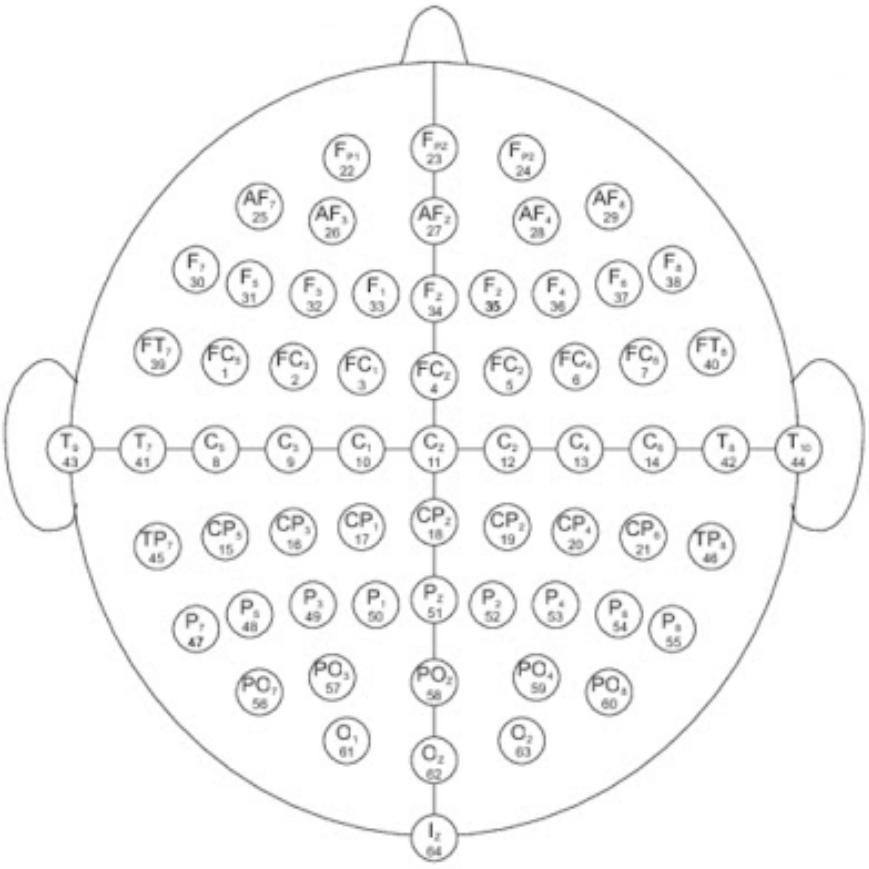


Figure 3.1: Electrodes positions

3.2 Regularization in Machine Learning

In the context of machine learning, the term 'regularization' refers to a set of techniques that help the machine to learn more than just memorize. Before we explore the concept of regularization in detail, let's discuss what the terms 'learning' and 'memorizing' mean from the perspective of machine learning.

When you train a machine learning model, and it is able to deliver accurate results on training data, but provides relatively poor results on unseen data or test dataset, you can say your model is memorizing more than generalizing.

3.2.1 Overfitting

In mathematical modeling, overfitting is "the production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit to additional data or predict future observations reliably".An overfitted model is a mathematical model that contains more parameters than can be justified by the data. The essence of overfitting is to have unknowingly extracted some of the residual variation (i.e., the noise) as if that variation represented underlying model structure

The potential for overfitting depends not only on the number of parameters and data but also the conformability of the model structure with the data shape, and the magnitude of model error compared to the expected level of noise or error in the data. Even when the fitted model does not have an excessive number of parameters, it is to be expected that the fitted relationship will appear to perform less well on a new data set than on the data set used for fitting (a phenomenon sometimes known as shrinkage).In particular, the value of the coefficient of determination will shrink relative to the original data.

To train our machine learning model, we give it some data to learn from. The process of plotting a series of data points and drawing the best fit line to understand the relationship between the variables is called Data Fitting. Our model is the best fit when it can find all necessary patterns in our data and avoid the random data points and unnecessary patterns called Noise.

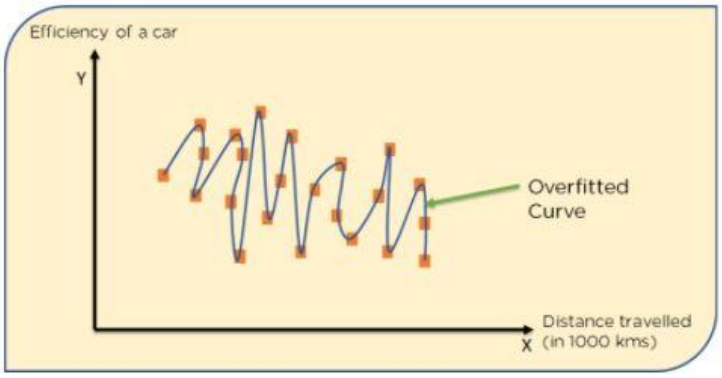


Figure 3.2: example illustrating the overfitting

If we allow our machine learning model to look at the data too many times, it will find a lot of patterns in our data, including the ones which are unnecessary. It will learn really well on the test dataset and fit very well to it. It will learn important patterns, but it will

also learn from the noise in our data and will not be able to predict on other datasets. A scenario where the machine learning model tries to learn from the details along with the noise in the data and tries to fit each data point on the curve is called Overfitting. In the figure depicted below, we can see that the model is fit for every point in our data. If given new data, the model curves may not correspond to the patterns in the new data, and the model cannot predict very well in it.

Conversely, in a scenario where the model has not been allowed to look at our data a sufficient number of times, the model won't be able to find patterns in our test dataset. It will not fit properly to our test dataset and fail to perform on new data too.

A scenario where a machine learning model can neither learn the relationship between variables in the testing data nor predict or classify a new data point is called Underfitting. The below diagram shows an under-fitted model. We can see that it has not fit properly to the data given to it. It has not found patterns in the data and has ignored a large part of the dataset. It cannot perform on both known and unknown data.

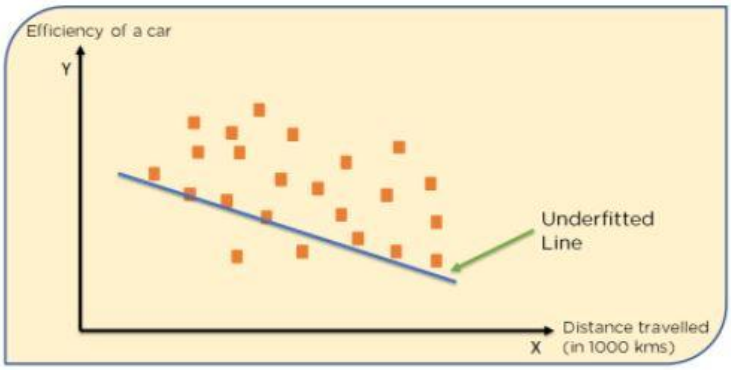


Figure 3.3: example illustrating under-fitting

3.2.2 Bias and Variance

A Bias occurs when an algorithm has limited flexibility to learn from data. Such models pay very little attention to the training data and oversimplify the model therefore the validation error or prediction error and training error follow similar trends. Such models always lead to a high error on training and test data. High Bias causes underfitting in our model.

Variance defines the algorithm's sensitivity to specific sets of data. A model with a high variance pays a lot of attention to training data and does not generalize therefore the validation error or prediction error are far apart from each other. Such models usually perform very well on training data but have high error rates on test data. High Variance causes overfitting in our model.

An optimal model is one in which the model is sensitive to the pattern in our model, but at the same time can generalize to new data. This happens when Bias and Variance are both optimal. We call this Bias-Variance Tradeoff and we can achieve it in over or under fitted models by using Regression.

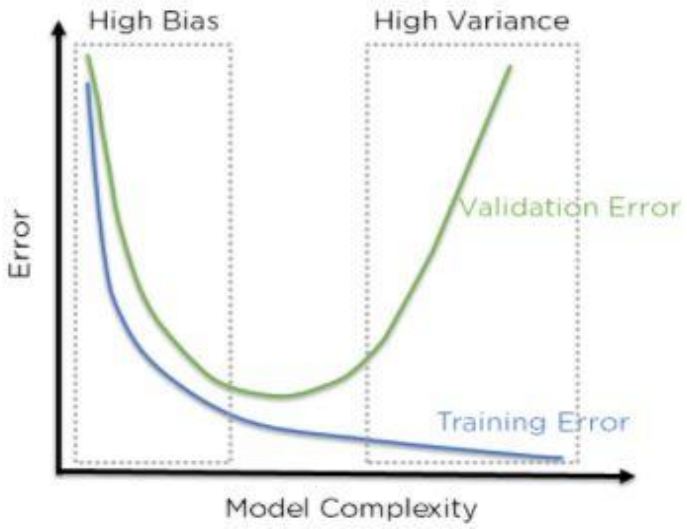


Figure 3.4: Error in testing and training datasets with high bias and variance

In the figure below, we can see that when bias is high, the error in both testing and training set is also high. When Variance is high, the model performs well on our training set and gives a low error, but the error in our testing set is very high. In the middle of this exists a region where the bias and variance are in perfect balance to each other, and here, but the training and testing errors are low.

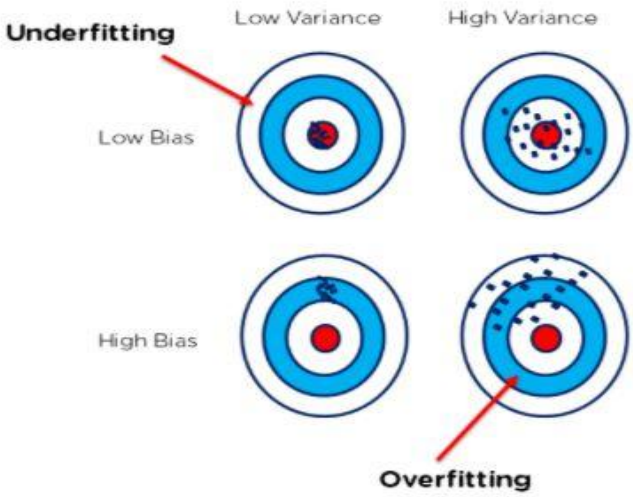


Figure 3.5: illustrating correlations between bias,variance and their effect on data fitting

One of the major aspects of training your machine learning model is avoiding overfitting. The model will have a low accuracy if it is overfitting. This happens because your model is trying too hard to capture the noise in your training dataset. By noise we mean the data points that don't really represent the true properties of your data, but random chance. Learning such data points, makes your model more flexible, at the risk of overfitting.

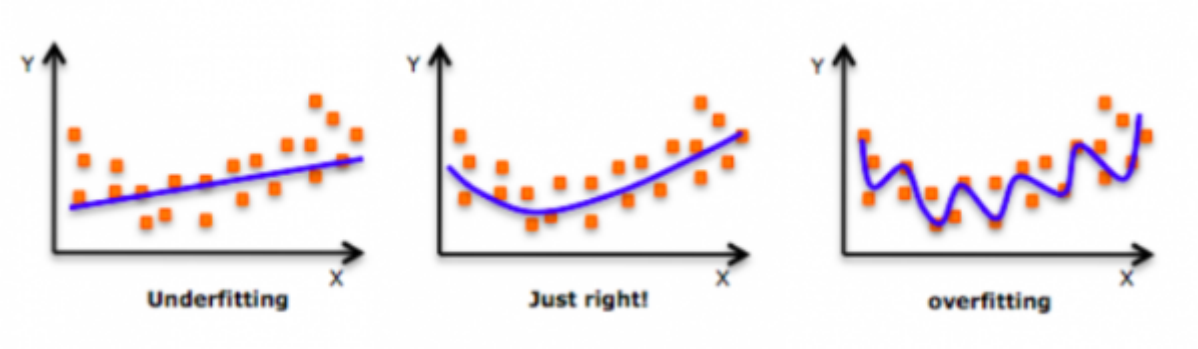


Figure 3.6: simplified example illustrating the difference between under-fitting,optimal fitting & over-fitting

3.2.3 Regularization

The term 'regularization' refers to a set of techniques that regularizes learning from particular features for traditional algorithms or neurons in the case of neural network algorithms. It normalizes and moderates weights attached to a feature or a neuron so that algorithms do not rely on just a few features or neurons to predict the result. This technique helps to avoid the problem of overfitting. There are three main regularization techniques, namely:

- Ridge Regression(L2 Norm)
- Lasso (L1 Norm)
- Dropout"primarily used in any kind of neural networks e.g. ANN, DNN, CNN or RNN to moderate the learning. Let's take a closer look at each of the techniques."

3.2.4 Ridge Regression (L2 Regularization)

Ridge regression is also called L2 norm or regularization.

When using this technique, we add the sum of weight's square to a loss function and thus create a new loss function which is denoted thus:

$$\text{Loss} = \sum_{j=1}^{m}\left(Yi - Wo - \sum_{i=1}^{n} Wi\,Xji\right)^2 + \lambda \sum_{i=1}^{n} Wi^2$$

Figure 3.7: Ridge Regression Technique Equation

As seen above, the original loss function is modified by adding normalized weights. Here normalized weights are in the form of squares.

You may have noticed parameters lambda along with normalized weights. lambda is the parameter that needs to be tuned using a cross-validation dataset. When you use lambda=0, it returns the residual sum of square as loss function which you chose initially. For a very high value of lambda, loss will ignore core loss function and minimize weight's square and will end up taking the parameters' value as zero.

Now the parameters are learned using a modified loss function. To minimize the above function, parameters need to be as small as possible. Thus, L2 norm prevents weights from rising too high.

3.2.5 Lasso Regression (L1 Regularization)

Also called lasso regression and denoted as below:

$$\text{Loss} = \sum_{j=1}^{m}\left(Yi - Wo - \sum_{i=1}^{n} Wi\,Xji\right)^2 + \lambda \sum_{i=1}^{n} |Wi|$$

Figure 3.8: Lasso Regression Technique Equation

This technique is different from ridge regression as it uses absolute weight values for normalization. lambda is again a tuning parameter and behaves in the same as it does when using ridge regression.

As loss function only considers absolute weights, optimization algorithms penalize higher weight values.

In ridge regression, loss function along with the optimization algorithm brings parameters near to zero but not actually zero, while lasso eliminates less important features and sets respective weight values to zero. Thus, lasso also performs feature selection along with regularization.

3.2.6 Dropout

Dropout is a regularization technique used in neural networks. It prevents complex co-adaptations from other neurons.

In neural nets, fully connected layers are more prone to overfit on training data. Using dropout, you can drop connections with 1-p probability for each of the specified layers. Where p is called keep probability parameter and which needs to be tuned.

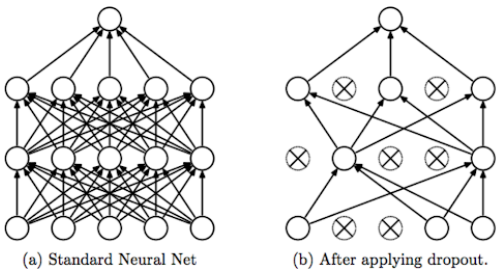


Figure 3.9: Illustration of the same neural network before & after using Dropout Technique

With dropout, you are left with a reduced network as dropped out neurons are left out during that training iteration. Dropout decreases overfitting by avoiding training all the neurons on the complete training data in one go. It also improves training speed and learns more robust internal functions that generalize better on unseen data. However, it is important to note that Dropout takes more epochs to train compared to training

without Dropout (If you have 10000 observations in your training data, then using 10000 examples for training is considered as 1 epoch).

Compared to other regularization methods such as weight decay, or early stopping, dropout also makes the network more robust. This is because when applying dropout, you are removing different neurons on every pass through the network. Thus, you are actually training multiple networks with different compositions of neurons and averaging their results.

One common way of achieving model robustness in machine learning is to train a collection of models and average their results. This approach, known as ensemble learning, helps correct the mistakes produced by single models. Ensemble methods work best when the models differ in their architectures and are trained on different subsets of the training data.

In deep learning, this approach would become prohibitively expensive since training a single neural network already takes lots of time and computational power. This is especially true for applications in computer vision and natural language processing, where datasets commonly consist of many millions of training examples. Furthermore, there may not be enough labeled training data to train different models on different subsets.

Dropout mitigates these problems. Since the model drops random neurons with every pass through the network, it essentially creates a new network on every pass. But weights are still shared between these networks contrary to ensemble methods, where each model needs to be trained from scratch.

Deep neural networks are arguably the most powerful machine learning models available to us today. Due to a large number of parameters, they can learn extremely complex functions. But this also makes them very prone to overfitting the training data.

Compared to other regularization methods such as weight decay, or early stopping, dropout also makes the network more robust. This is because when applying dropout, you are removing different neurons on every pass through the network. Thus, you are actually training multiple networks with different compositions of neurons and averaging their results.

One common way of achieving model robustness in machine learning is to train a collection of models and average their results. This approach, known as ensemble learning, helps correct the mistakes produced by single models. Ensemble methods work best when the models differ in their architectures and are trained on different subsets of the training data.

In deep learning, this approach would become prohibitively expensive since training a single neural network already takes lots of time and computational power. This is especially true for applications in computer vision and natural language processing, where datasets commonly consist of many millions of training examples. Furthermore, there may not be enough labeled training data to train different models on different subsets.

Dropout mitigates these problems. Since the model drops random neurons with every pass through the network, it essentially creates a new network on every pass. But weights are still shared between these networks contrary to ensemble methods, where each model needs to be trained from scratch.

The authors who first proposed dropout (Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky,

Ilya Sutskever, and Ruslan Salakhutdinov) explain the main benefit of dropout as reducing the occurrence of coadaptations between neurons.

Coadaptions occur when neurons learn to fix the mistakes made by other neurons on the training data. The network thus becomes very good at fitting the training data. But it also becomes more volatile because the coadaptions are so attuned to the peculiarities of the training data that they won't generalize to the test data.

3.2.7 Dropout Mechanism

To apply dropout, you need to set a retention probability for each layer. The retention probability specifies the probability that a unit is not dropped. For example, if you set the retention probability to 0.8, the units in that layer have an 80% chance of remaining active and a 20% chance of being dropped.

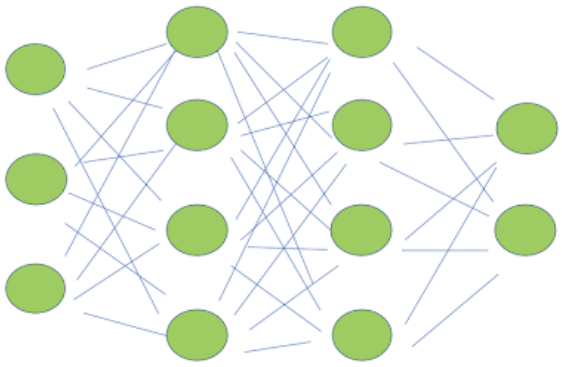


Figure 3.10: Illustration of neural network before Technique

Standard practice is to set the retention probability to 0.5 for hidden layers and to something close to 1, like 0.8 or 0.9 on the input layer. Output layers generally do not apply dropout.

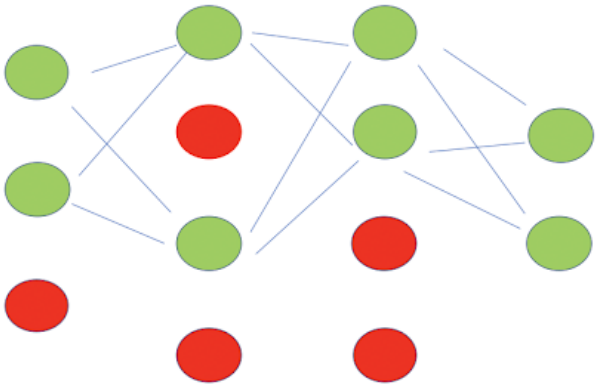


Figure 3.11: Illustration of neural network after using Dropout Technique

In practice, dropout is applied by creating a mask for each layer and filling it with values between 0 and 1 generated by a random number generator according to the retention probability. Each neuron with a corresponding retention probability below the specified threshold is kept, while the other ones are removed. For example, for the first hidden layer in the network above, we would create a mask with four entries.

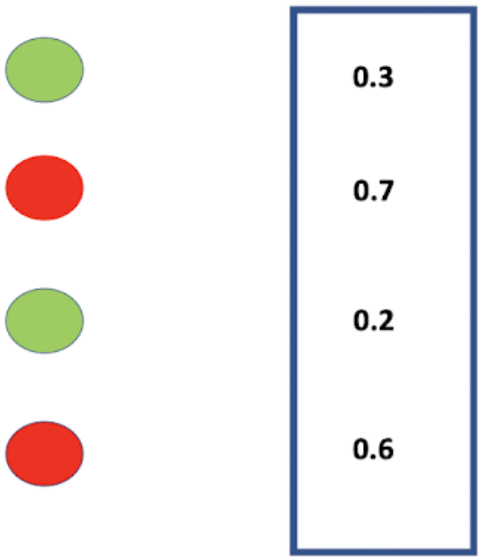


Figure 3.12: Illustration of masking layers with respective retention probabilities

Alternatively, we could also fill the mask with random boolean values according to the retention probability. Neurons with a corresponding "True" entry are kept while those with a "False" value are discarded.

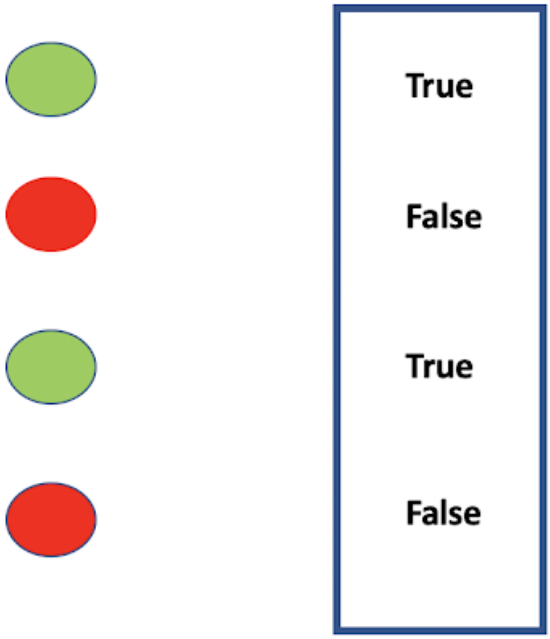


Figure 3.13: Illustration of masking layers with True & false Values

Dropout is only used during training to make the network more robust to fluctuations in the training data. At test time, however, you want to use the full network in all its glory. In other words, you do not apply dropout with the test data and during inference in production.

But that means your neurons will receive more connections and therefore more activations during inference than what they were used to during training. For example, if you use a dropout rate of 50% dropping two out of four neurons in a layer during training, the neurons in the next layer will receive twice the activations during inference and thus become overexcited. Accordingly, the values produced by these neurons will, on average, be too large by 50%. To correct this overactivation at test and inference time, you multiply the weights of the overexcited neurons by the retention probability (1 – dropout rate) and thus scale them down. The following graphic by the user Dmytro Prylipko on Datascience Stackexchange nicely illustrates how this works in practice.

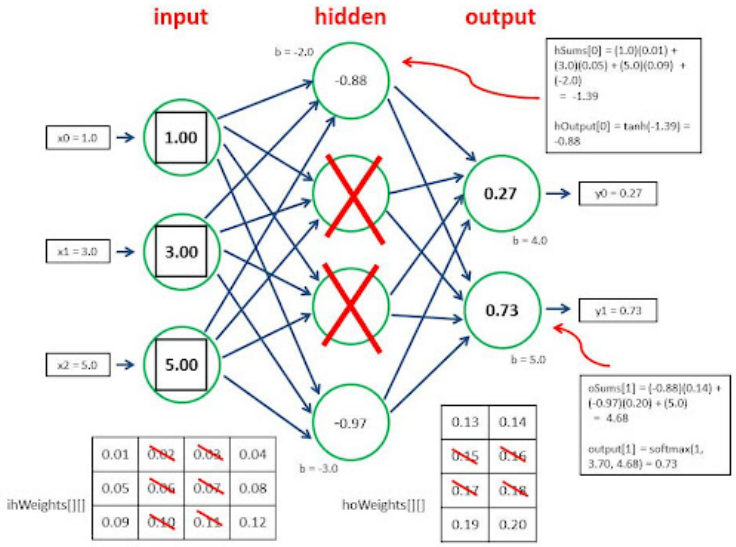


Figure 3.14: Illustration of how to correct for dropout at test time

3.2.8 Inverted Dropout

An alternative to scaling the activations at test and inference time by the retention probability is to scale them at training time.

You do this by dropping out the neurons and immediately afterward scaling them by the inverse retention probability.

This operation scales the activations of the remaining neurons up to make up for the signal from the other neurons that were dropped.

This corrects the activations right at training time. Accordingly, it is often the preferred option.

	Conv Direction	Input	Filter	Output
tf.nn.conv1d	1-direction →	3-dim	3-dim	2-dim
tf.nn.conv2d	2-direction ↓→	4-dim	4-dim	3-dim
tf.nn.conv3d	3-direction	5-dim	5-dim	4-dim

Figure 3.15: Direction of operation for 1D, 2D, and 3D CNN in TensorFlow.

3.3 Deep Neural Network—DNN

A formal definition of deep learning is:

Deep learning is a particular kind of machine learning that achieves great power and flexibility by learning to represent the world as a nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones.

It can also be recognized as: A deep neural network (DNN) is an artificial neural network (ANN) with multiple layers between the input and output layers.There are different types of neural networks but they always consist of the same components: neurons, synapses, weights, biases, and functions.These components functioning similar to the human brains and can be trained like any other ML algorithm.

Nodes are little parts of the system, and they are like neurons of the human brain. When a stimulus hits them, a process takes place in these nodes. Some of them are connected and marked, and some are not, but in general, nodes are grouped into layers.

The system must process layers of data between the input and output to solve a task. The more layers it has to process to get the result, the deeper the network is considered. There is a concept of Credit Assignment Path (CAP) which means the number of such layers needed for the system to complete the task. The neural network is deep if the CAP index is more than two.

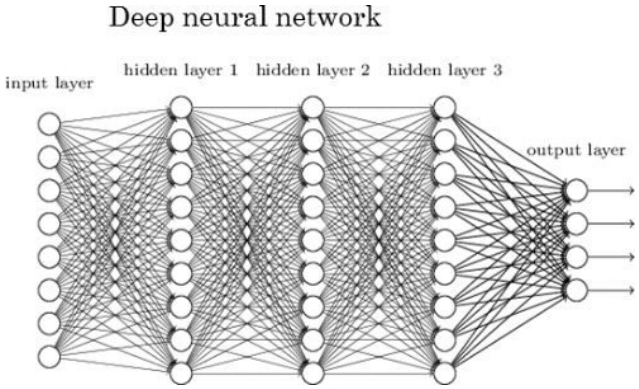


Figure 3.16: Deep Neural Network example

3.3.1 Difference Between the Neural Network and Deep Neural Network

You can compare a neural network to a chess game with a computer. It has algorithms, according to which it determines tactics, depending on your moves and actions. The programmer enters data on how each figure moves into the computer's database, determines the boundaries of the chessboard, introduces a huge number of strategies that chess players play by. At the same time, the computer may, for example, be able to learn from you and other people, and it can become a deep neural network. In a while, playing with different players, it can become invincible.

The neural network is not a creative system, but a deep neural network is much more complicated than the first one. It can recognize voice commands, recognize sound and

graphics, do an expert review, and in our case interpret something as complicated as electrical brain signals. The neural network can get one result (a word, an action, a number, or a solution), while the deep neural network solves the problem more globally and can draw conclusions or predictions depending on the information supplied and the desired result. The neural network requires a specific input of data and algorithms of solutions, and the deep neural network can solve a problem without a significant amount of marked data.

3.3.2 Challenges of DNN

As with ANNs, many issues can arise with naively trained DNNs. Two common issues are over-fitting and computation time.

DNNs are prone to over-fitting because of the added layers of abstraction, which allow them to model rare dependencies in the training data. there are a few ways to attempt to partially overcome these challenges,some of these methods are stated below:

- Regularization methods can be applied during training to combat over-fitting.
- Alternatively dropout regularization randomly omits units from the hidden layers during training. This helps to exclude rare dependencies.
- Finally, data can be augmented via methods such as cropping and rotating such that smaller training sets can be increased in size to reduce the chances of overfitting.

3.3.3 The architecture of DNN

The leftmost layer in the network displayed below is called the input layer, and the neurons within the layer are called input neurons. The rightmost or output layer contains the output neurons, or, as in this case, a single output neuron. The middle layer is called a hidden layer, since the neurons in this layer are neither inputs nor outputs.

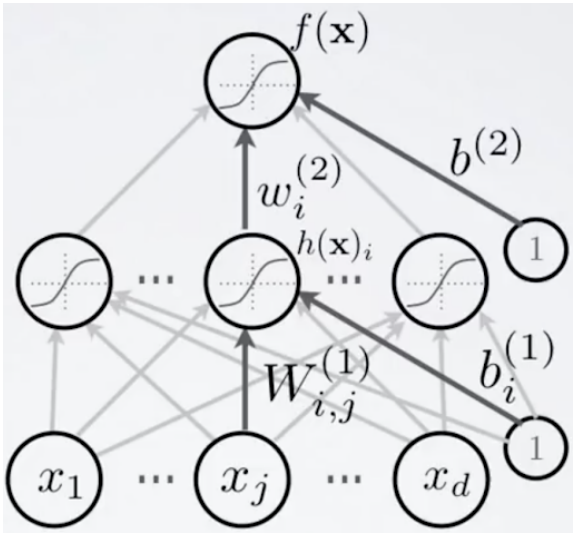


Figure 3.17: DNN layers

As denoted in figure wiá denote the weight of the ith neuron of lth layer. W1(i,j) denote the weight for the connection from the jth neuron in the (l-1)th layer to the ith

neuron in the lth layer. We use bá for the bias for the ith neuron. h(x) is the activation function and for now we are using sigmoid for this purpose. f(x) is output function. Refer Fig 5. Interesting thing about Feedforward networks with hidden layers is that, it provides a universal approximation framework. Specifically, the universal approximation theorem(Horniket al., 1989; Cybenko, 1989) states that a feedforward network with a linear output layer and at least one hidden layer with any "squashing" activation function (such as the sigmoid activation function) can approximate any Borel measurable function from one finite-dimensional space to another with any desired non-zero amount of error, provided that the network is given enough hidden units. The universal approximation theorem says that there exists a network large enough to achieve any degree of accuracy we desire, but the theorem does not say how large this network will be.

Let see how it works by taking an example. the figure below illustrates how a network with absolute value rectification creates mirror images of the function computed on top of some hidden unit, with respect to the input of that hidden unit. Each hidden unit specifies where to fold the input space in order to create mirror responses (on both sides of the absolute value non-linearity). By composing these folding operations, we obtain an exponentially large number of piece-wise linear regions which can capture all kinds of regular (e.g., repeating) patterns.

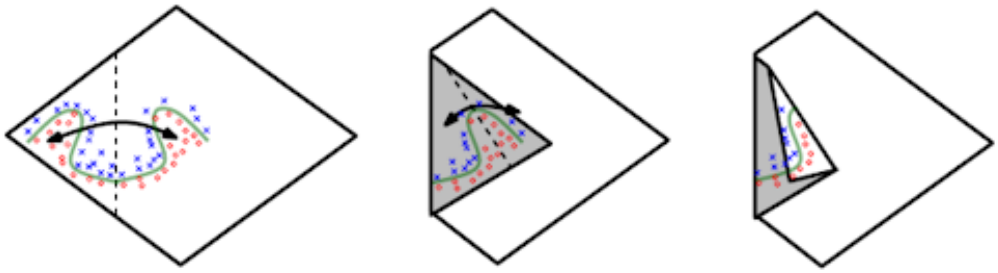


Figure 3.18: Dnn repeating patterns

- (Left) An absolute value rectification unit has the same output for every pair of mirror points in its input. The mirror axis of symmetry is given by the hyperplane defined by the weights and bias of the unit. A function computed on top of that unit (the green decision surface) will be a mirror image of a simpler pattern across that axis of symmetry.
- (Center)The function can be obtained by folding the space around the axis of symmetry.
- (Right)Another repeating pattern can be folded on top of the first (by another downstream unit) to obtain another symmetry(which is now repeated four times, with two hidden layers).

3.3.4 Cost Function

We will introduce a cost function for the purpose of solving and training our model. Now you must be thinking, Why not try to maximize that number of correct output directly, rather than minimizing a proxy measure like the quadratic cost? The problem with that is that the number of correct classified data point is not a smooth function of the weights and

biases in the network. For the most part, making small changes to the weights and biases won't cause any change at all in the number of training data point classified correctly. That makes it difficult to figure out how to change the weights and biases to get improved performance. If we instead use a smooth cost function like the quadratic cost it turns out to be easy to figure out how to make small changes in the weights and biases so as to get an improvement in the cost. That's why we focus first on minimizing the quadratic cost, and only after that will we examine the classification accuracy. In this post we will only see mean square error cost function.

$$C(w,b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2.$$

Figure 3.19: mean square error cost function

Here, w denotes the collection of all weights in the network, b all the biases, n is the total number of training inputs, a is the vector of outputs from the network when x is input, and the sum is over all training inputs, x. Of course, the output a depends on x, w and b, but to keep the notation simple I haven't explicitly indicated this dependence. The notation ‖v‖ just denotes the usual length function for a vector v. We'll call C the quadratic cost function; it's also sometimes known as the mean squared error or just MSE.

3.3.5 Gradient-Based Learning

Designing and training a neural network is not much different from training any other machine learning model with gradient descent. The largest difference between the linear models we have seen so far and neural networks is that the nonlinearity of a neural network causes most interesting loss functions to become non-convex. This means that neural networks are usually trained by using iterative, gradient-based optimizers that merely drive the cost function to a very low value, rather than the linear equation solvers used to train linear regression models or the convex optimization algorithms with global convergence guarantees used to train logistic regression or SVMs.

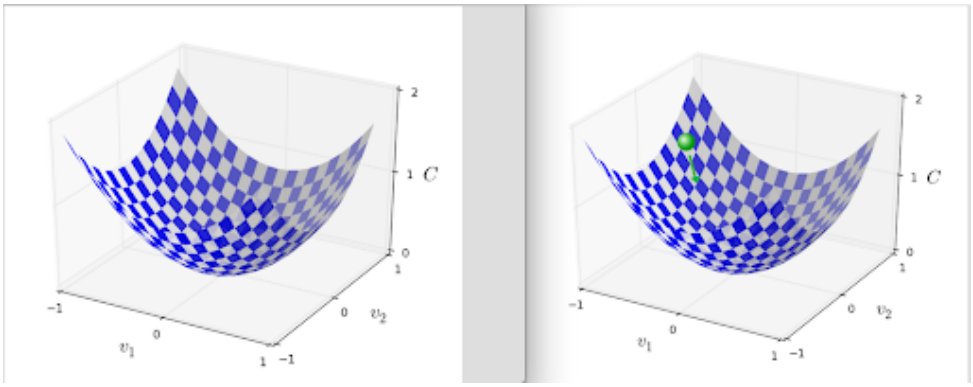


Figure 3.20: logistic regression

As we can see at right of the above figure, an analogy of a ball dropped in a deep valley and it settle downs at the bottom of the valley. Similarly we want our cost function to

get minimize and get to the minimum value possible. When we move the ball a small amount Delta(v1) in the v1 direction, and a small amount Delta(v2) in the v2 direction. Calculus tells us that C changes as follows:

$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2.$$

Figure 3.21: DNN cost function

Change of the v1 and v2 such that the change in cost is negative is desirable. We can also denote delta(C) is approximately equal to VC.delta(v). where VC is,

$$\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \ldots, \frac{\partial C}{\partial v_m}\right)^T.$$

Figure 3.22: DNN cost function

and delta(V) is:

$$\Delta v = (\Delta v_1, \ldots, \Delta v_m)^T$$

Figure 3.23

Indeed, there's even a sense in which gradient descent is the optimal strategy for searching for a minimum. Let's suppose that we're trying to make a move delta(v) in position so as to decrease C as much as possible. We'll constrain the size of the move so that $\|\text{delta(v)}\| = \epsilon for some small fixed eta > 0. In other words, we want a move that is a small step of a fixed size, and we'retry \eta C, where = \epsilon/VC is determined by the size constraint Delta(v) = . So gradient descent can be viewed as a wa$

$$w_k \to w_k' = w_k - \eta\frac{\partial C}{\partial w_k}$$
$$b_l \to b_l' = b_l - \eta\frac{\partial C}{\partial b_l}.$$

Figure 3.24: gradient descent function

Now there are many challenges in training gradient based learning. But for now I just want to mention one problem. When the number of training inputs is very large this can take a long time, and learning thus occurs slowly. An idea called stochastic gradient descent can be used to speed up learning. To make these ideas more precise, stochastic gradient descent works by randomly picking out a small number m of randomly chosen training inputs. We'll label those random training inputs X1,X2,. . . ,Xm and refer to them

as a mini-batch. Provided the sample size mm is large enough we expect that the average value of the VCXj will be roughly equal to the average over all VCx, that is:

$$\frac{\sum_{j=1}^{m} \nabla C_{X_j}}{m} \approx \frac{\sum_x \nabla C_x}{n} = \nabla C,$$

Figure 3.25: stochastic gradient descent

This modification helps us in reducing a good amount of computational load. Stochastic gradient descent applied to non-convex loss functions has no such convergence guarantee, and is sensitive to the values of the initial parameters. For feedforward neural networks, it is important to initialize all weights to small random values. The biases may be initialized to zero or to small positive values.

3.4 Convolutional Neural Network—CNN

3.4.1 Motivation

The human brain processes a huge amount of information the second we see an image. Each neuron works in its own receptive field and is connected to other neurons in a way that they cover the entire visual field. Just as each neuron responds to stimuli only in the restricted region of the visual field called the receptive field in the biological vision system, each neuron in a CNN processes data only in its receptive field as well. The layers are arranged in such a way so that they detect simpler patterns first (lines, curves, etc.) and more complex patterns (faces, objects, etc.) further along. By using a CNN, one can enable sight to computers.

3.4.2 CNN architecture

There are two main parts to a CNN architecture

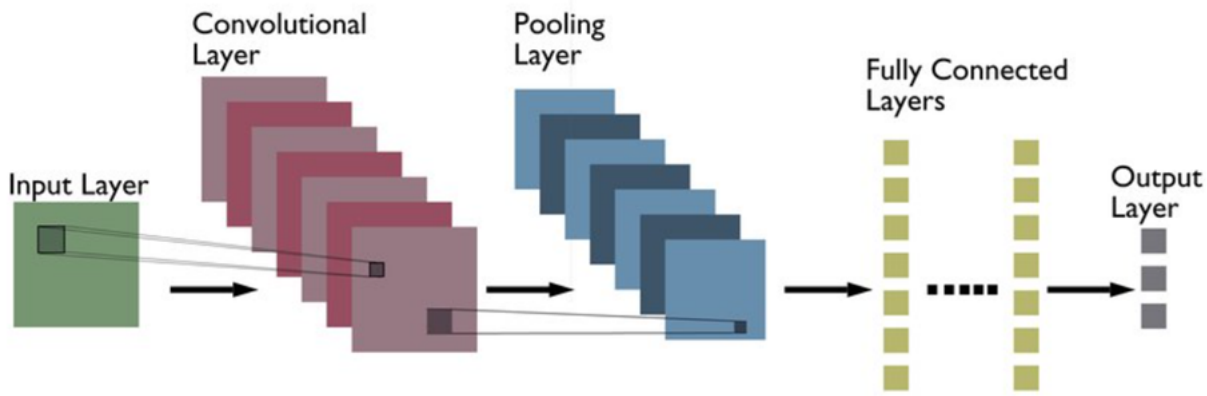


Figure 3.26: CNN layers types

• A convolution tool that separates and identifies the various features of the image for analysis in a process called as Feature Extraction
• A fully connected layer that utilizes the output from the convolution process and predicts the class of the image based on the features extracted in previous stages.
Convolution Layers
There are three types of layers that make up the CNN which are
convolutional layers pooling layers fully-connected (FC) layers.
When these layers are stacked, a CNN architecture will be formed. In addition to these three layers, there are two more important parameters which are the dropout layer and the activation function which are defined below.

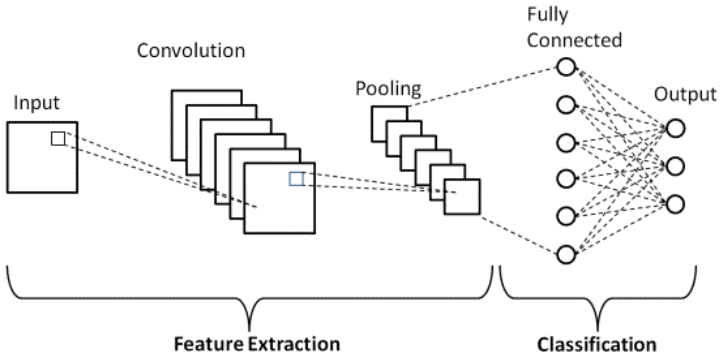


Figure 3.27: the two main parts of CNN

1. Convolutional Layer This layer is the first layer that is used to extract the various features from the input images. In this layer, the mathematical operation of convolution is performed between the input image and a filter of a particular size MxM. By sliding the filter over the input image, the dot product is taken between the filter and the parts of the input image with respect to the size of the filter (MxM). The output is termed as the Feature map which gives us information about the image such as the corners and edges. Later, this feature map is fed to other layers to learn several other features of the input image.

2. Pooling Layer In most cases, a Convolutional Layer is followed by a Pooling Layer. The primary aim of this layer is to decrease the size of the convolved feature map to reduce the computational costs. This is performed by decreasing the connections between layers and independently operates on each feature map. Depending upon method used, there are several types of Pooling operations. In Max Pooling, the largest element is taken from feature map. Average Pooling calculates the average of the elements in a predefined sized Image section. The total sum of the elements in the predefined section is computed in Sum Pooling. The Pooling Layer usually serves as a bridge between the Convolutional Layer and the FC Layer Must Read: Neural Network Project Ideas

3. Fully Connected Layer The Fully Connected (FC) layer consists of the weights and biases along with the neurons and is used to connect the neurons between two different layers. These layers are usually placed before the output layer and form the last few layers of a CNN Architecture. In this, the input image from the previous layers are flattened and fed to the FC layer. The flattened vector then undergoes few more FC layers where the mathematical functions operations usually take place. In this stage, the classification process begins to take place.

4. Dropout Usually, when all the features are connected to the FC layer, it can cause overfitting in the training dataset. Overfitting occurs when a particular model works so well on the training data causing a negative impact in the model's performance when used on a new data. To overcome this problem, a dropout layer is utilised wherein a few neurons are dropped from the neural network during training process resulting in reduced size of the model. On passing a dropout of 0.3, 30 % of the nodes are dropped out randomly from the neural network.

5. Activation Functions Finally, one of the most important parameters of the CNN model is the activation function. They are used to learn and approximate any kind of continuous and complex relationship between variables of the network. In simple words, it decides which information of the model should fire in the forward direction and which ones should not at the end of the network. It adds non-linearity to the network. There are several commonly used activation functions such as the ReLU, Softmax, tanH and the Sigmoid functions. Each of these functions have a specific usage. For a binary classification CNN model, sigmoid and softmax functions are preferred an for a multi-class classification, generally softmax us used.

3.4.3 Activation Functions

An activation function is a function that is added into an artificial neural network in order to help the network learn complex patterns in the data. When comparing with a neuron-based model that is in our brains, the activation function is at the end deciding what is to be fired to the next neuron. That is exactly what an activation function does in an ANN as well. It takes in the output signal from the previous cell and converts it into some form that can be taken as input to the next cell. The comparison can be summarized in the figure below.
Well, the purpose of an activation function is to add non-linearity to the neural network.

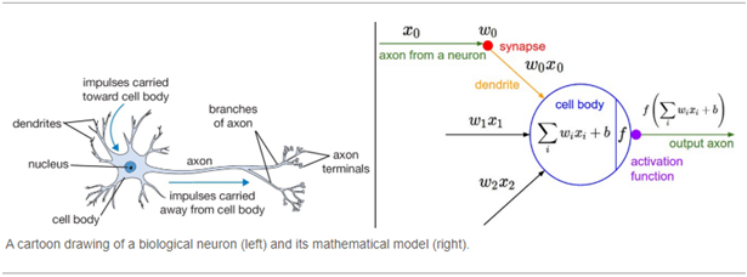


Figure 3.28: A drawing of biological neural(left) and it's mathematical model (right)

The importance of the derivative/differentiation:
When updating the curve, to know in which direction and how much to change or update the curve depending upon the slope.That is why we use differentiation in almost every part of Machine Learning and Deep Learning.

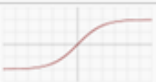
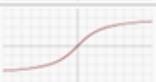
Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

Figure 3.29: Activation functions with it's derivatives

3.4.4 ReLU (Rectified Linear Unit) Activation Function

The ReLU is the most used activation function in the world right now.Since, it is used in almost all the convolutional neural networks or deep learning.

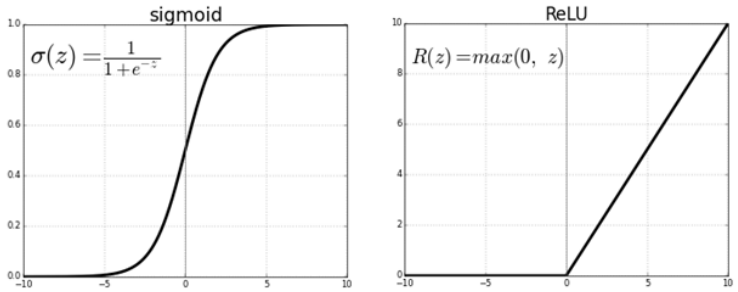


Figure 3.30: ReLU vs Logistic sigmoid

As you can see, the ReLU is half rectified (from bottom). f(z) is zero when z is less than zero and f(z) is equal to z when z is above or equal to zero.
Range: [0 to infinity)

The function and its derivative both are monotonic.
But the issue is that all the negative values become zero immediately which decreases

the ability of the model to fit or train from the data properly. That means any negative input given to the ReLU activation function turns the value into zero immediately in the graph, which in turns affects the resulting graph by not mapping the negative values appropriately.

3.4.5 Leaky ReLU

It is an attempt to solve the dying ReLU problem

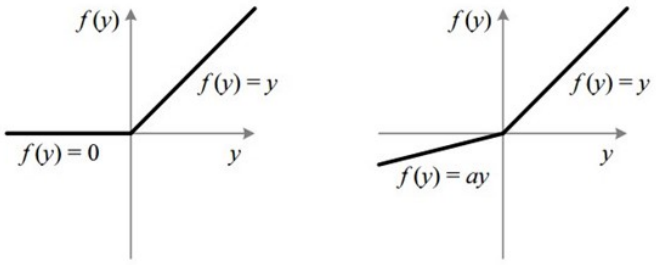


Figure 3.31: ReLU VS Leaky ReLu

Can you see the Leak?
The leak helps to increase the range of the ReLU function.
Usually, the value of a is 0.01 or so. When a is not 0.01 then it is called Randomized ReLU.
Therefore the range of the Leaky ReLU is (-infinity to infinity).
Both Leaky and Randomized ReLU functions are monotonic in nature. Also, their derivatives also monotonic in nature.

3.4.6 Softmax Function

Softmax History
The first known use of the softmax function predates machine learning. The softmax function is in fact borrowed from physics and statistical mechanics, where it is known as the Boltzmann distribution or the Gibbs distribution. It was formulated by the Austrian physicist and philosopher Ludwig Boltzmann in 1868.
Boltzmann was studying the statistical mechanics of gases in thermal equilibrium. He found that the Boltzmann distribution could describe the probability of finding a system in a certain state, given that state's energy, and the temperature of the system. His version of the formula was similar to that used in reinforcement learning. Indeed, the parameter is called temperature in the field of reinforcement learning as a homage to Boltzmann.
In 1902 the American physicist and chemist Josiah Willard Gibbs popularized the Boltzmann distribution when he used it to lay the foundation for thermodynamics and his definition of entropy. It also forms the basis of spectroscopy, that is the analysis of materials by looking at the light that they absorb and reflect.
In 1959 Robert Duncan Luce proposed the use of the softmax function for reinforcement learning in his book Individual Choice Behavior: A Theoretical Analysis. Finally in 1989 John S. Bridle suggested that the argmax in feedforward neural networks should be replaced by softmax because it "preserves the rank order of its input values, and is

a differentiable generalisation of the 'winner-take-all' operation of picking the maximum value". In recent years, as neural networks have become widely used, the softmax has become well known thanks to these properties.

The softmax function is a function that turns a vector of K real values into a vector of K real values that sum to 1. The input values can be positive, negative, zero, or greater than one, but the softmax transforms them into values between 0 and 1, so that they can be interpreted as probabilities. If one of the inputs is small or negative, the softmax turns it into a small probability, and if an input is large, then it turns it into a large probability, but it will always remain between 0 and 1.
The softmax function can be used in a classifier only when the classes are mutually exclusive.
Many multi-layer neural networks end in a penultimate layer which outputs real-valued scores that are not conveniently scaled and which may be difficult to work with. Here the softmax is very useful because it converts the scores to a normalized probability distribution, which can be displayed to a user or used as input to other systems. For this reason it is usual to append a softmax function as the final layer of the neural network.

The softmax formula is as follows:

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Figure 3.32: Softmax formula

Softmax Formula Symbols Explained

$\vec{z}$ The input vector to the softmax function, made up of (z0, ... zK)

z_i :All the zi values are the elements of the input vector to the softmax function, and they can take any real value, positive, zero or negative. For example a neural network could have output a vector such as (-0.62, 8.12, 2.53), which is not a valid probability distribution, hence why the softmax would be necessary.

e^{z_i} :The standard exponential function is applied to each element of the input vector. This gives a positive value above 0, which will be very small if the input was negative, and very large if the input was large. However, it is still not fixed in the range (0, 1) which is what is required of a probability.

$\sum_{j=1}^{K} e^{z_j}$:The term on the bottom of the formula is the normalization term. It ensures that all the output values of the function will sum to 1 and each be in the range (0, 1), thus constituting a valid probability distribution.

K :The number of classes in the multi-class classifier.

3.4.7 Adam optimization algorithm

Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data.
Adaptive Moment Estimation is an algorithm for optimization technique for gradient descent. The method is really efficient when working with large problem involving a lot of data or parameters. It requires less memory and is efficient. Intuitively, it is a combination of the 'gradient descent with momentum' algorithm and the 'RMSP' algorithm.
Adam is different to classical stochastic gradient descent.
Stochastic gradient descent maintains a single learning rate (termed alpha) for all weight updates and the learning rate does not change during training. A learning rate is maintained for each network weight (parameter) and separately adapted as learning unfolds.
The authors describe Adam as combining the advantages of two other extensions of stochastic gradient descent. Specifically:
Adaptive Gradient Algorithm (AdaGrad) that maintains a per-parameter learning rate that improves performance on problems with sparse gradients (e.g. natural language and computer vision problems).
Root Mean Square Propagation (RMSProp) that also maintains per-parameter learning rates that are adapted based on the average of recent magnitudes of the gradients for the weight (e.g. how quickly it is changing). This means the algorithm does well on online and non-stationary problems (e.g. noisy).

3.4.8 Adam optimization mechanism

How Adam works?

Adam optimizer involves a combination of two gradient descent methodologies:
Momentum:

This algorithm is used to accelerate the gradient descent algorithm by taking into consideration the 'exponentially weighted average' of the gradients. Using averages makes the algorithm converge towards the minima in a faster pace.
where

PICTURES
NOTE: Time (t) could be interpreted as an Iteration (i).
Adam Optimizer inherits the strengths or the positive attributes of the above two methods and builds upon them to give a more optimized gradient descent.

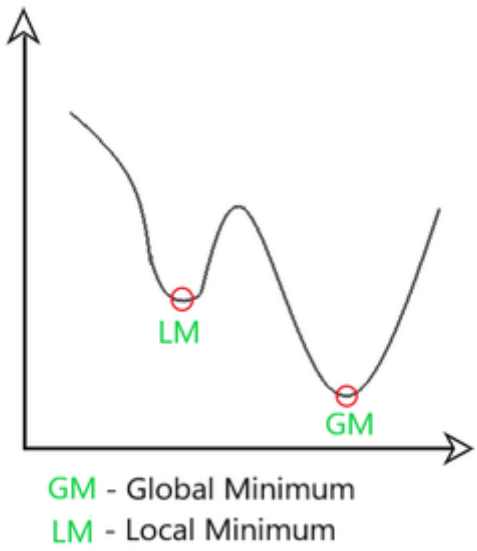


Figure 3.33: Calculation of global max and global min

Here, we control the rate of gradient descent in such a way that there is minimum oscillation when it reaches the global minimum while taking big enough steps (step-size) so as to pass the local minima hurdles along the way. Hence, combining the features of the above methods to reach the global minimum efficiently. Mathematical Aspect of Adam Optimizer Taking the formulas used in the above two methods, we get

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\left[\frac{\delta L}{\delta w_t}\right] v_t = \beta_2 v_{t-1} + (1-\beta_2)\left[\frac{\delta L}{\delta w_t}\right]^2$$

Figure 3.34: Mathematical formula for ADAM optimizer

Parameters Used :

1. = a small +ve constant to avoid 'division by 0' error when (vt -> 0). (10-8)

2. 1 2 = decay rates of average of gradients in the above two methods. (1 = 0.9 2 = 0.999))

3. — Step size parameter / learning rate (0.001)

Since mt and vt have both initialized as 0 (based on the above methods), it is observed that they gain a tendency to be 'biased towards 0' as both 1 2 1. This Optimizer fixes

this problem by computing 'bias-corrected' mt and vt. This is also done to control the weights while reaching the global minimum to prevent high oscillations when near it. The formulas used are:

$$\widehat{m_t} = \frac{m_t}{1-\beta_1^t} \widehat{v_t} = \frac{v_t}{1-\beta_2^t}$$

Figure 3.35: optimizer function

Intuitively, we are adapting to the gradient descent after every iteration so that it remains controlled and unbiased throughout the process, hence the name Adam. Now, instead of our normal weight parameters mt and vt , we take the bias-corrected weight parameters (m hat)t and (v hat)t. Putting them into our general equation, we get

$$w_{t+1} = w_t - \widehat{m_t} \left(\frac{a}{\sqrt{\widehat{v_t}+e}} \right)$$

Figure 3.36: ADAM optimizer function

Performance:
Building upon the strengths of previous models, Adam optimizer gives much higher performance than the previously used and outperforms them by a big margin into giving an optimized gradient descent. The plot is shown below clearly depicts how Adam Optimizer outperforms the rest of the optimizer by a considerable margin in terms of training cost (low) and performance (high).

3.4.9 Model Architecture

The model consists of
six-layer convolution network with two max-pooling layers and three fully connected layers
With regard to the Physionet database, at first, we used Deep ConvNet architecture which included four convolutional layers, four max-pooling layers and one fully-connected layer. In consideration of the pooling kernel size (a 2D kernel such as 2×2 in our work versus a 1D kernel such as 3×1) and stride size (stride both two for height and width in our work versus stride one for height and three for), we used two max-pooling layers instead of four empirically to better suit for our data size. Besides, we have tried a different number of convolutional and fully connected layers. While adding more convolutional layers, higher accuracy on the test set was produced. When the number of convolutional layers was greater than six, the accuracy barely stopped growing. Finally, six layers of convolution operation and the three fully connected layers in our paper are the best-performing ones in the experiments. Adam was used as the optimization algorithm for this work.

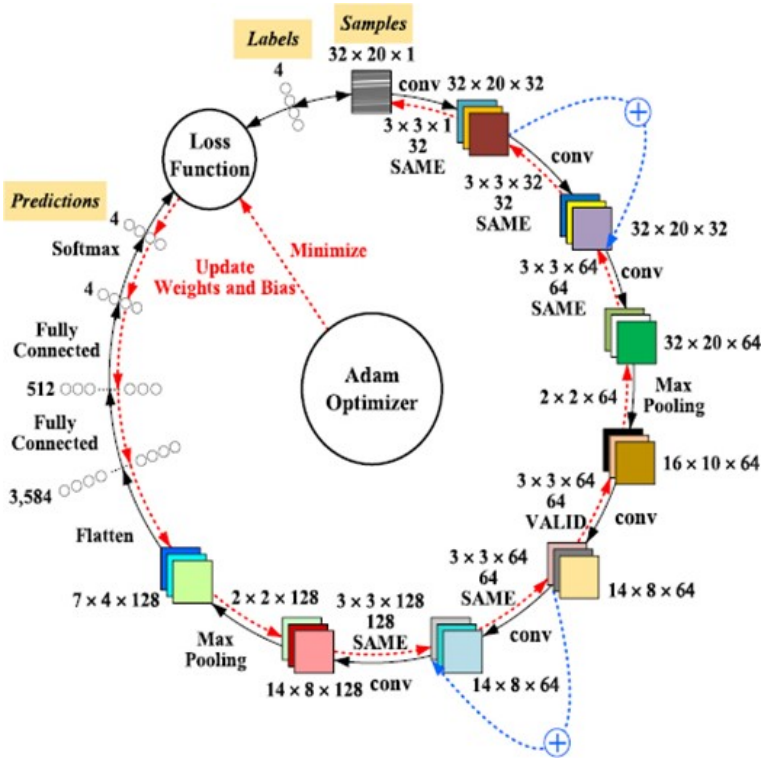


Figure 3.37: The Model Network

Dropout, batch normalization and concatenation deployment are implemented which can lead to faster training of the network, better conservation of information throughout the hierarchical process, and avoids overfitting of the network based on their nature. On the other hand, the model without them may be prone to overfitting because the EEG signal has low signal-to-noise ratio and the difference between subjects is large resulting in inconsistent distribution.
PICS

3.5 Fully Convolutional Neural Network—FCN

3.5.1 Introduction

FCN is a network that does not contain any "Dense" layers (as in traditional CNNs) instead it contains 1x1 convolutions that perform the task of fully connected layers (Dense layers). Though the absence of dense layers makes it possible to feed in variable inputs,making the networks faster to train. It also means that an FCN can handle a wide range of image sizes since all connections are local. fully convolution network (FCN) is a neural network that only performs convolution (and subsampling or upsampling) operations. Equivalently, an FCN is a CNN without fully connected layers. It is primarily used for semantic segmentation. Convolution, pooling, and upsampling are the only locally linked layers they use. Since dense layers aren't used, A fully connected neural network consists of a series of fully connected layers. A fully connected layer is a function from m to n . Each output dimension depends on each input dimension. Pictorially, a fully connected layer is represented as follows

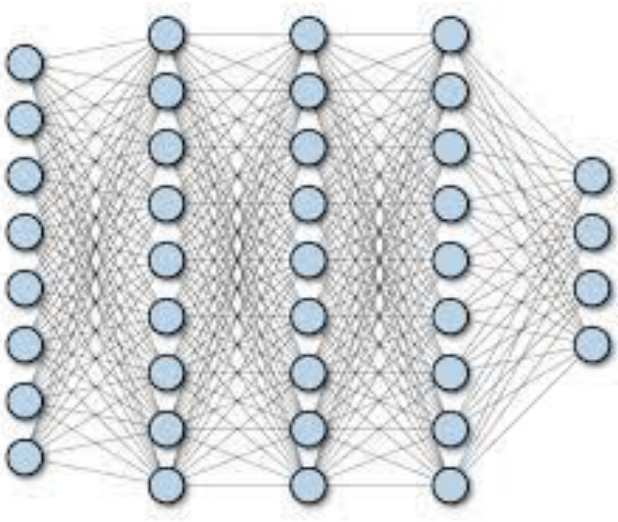


Figure 3.38: Fully connected layer

3.5.2 Conversion between FCN and CNN

The typical convolution neural network (CNN) is not fully convolutional because it often contains fully connected layers too (which do not perform the convolution operation), which are parameter-rich, in the sense that they have many parameters (compared to their equivalent convolution layers), although the fully connected layers can also be viewed as convolutions with kernels that cover the entire input regions.

However, in both types of layers, neurons calculate dot products, so their functional form is the same. Therefore, it is possible to transform the two into each other

For any convolutional layer, there is a fully connected layer that can achieve the same forward propagation function as it. The weight matrix is a huge matrix, except for some specific blocks, the rest are all zeros. In most of these blocks, the elements are equal. On the contrary, any fully connected layer can be transformed into a convolutional layer. For example, a fully connected layer with K = 4096 , the size of the input data volume is 7 7

512 , this fully connected layer can be equivalently regarded as a F = 7 , P = 0 , S = 1 , K = 4096 The convolutional layer. In other words, the size of the filter is set to match the size of the input data body. Because there is only a single depth column covering and sliding over the input data volume, the output will become 1 1 4096. This result is the same as using the original fully connected layer

In the two transformations, converting a fully connected layer to a convolutional layer is more useful in practical applications. Assuming that the input of a convolutional neural network is an image of 224 x 224 x 3 , a series of convolutional layers and downsampling layers transform the image data into an activation data volume of size 7 x 7 x 512 . AlexNet uses two fully connected layers with a size of 4096, and the last fully connected layer with 1000 neurons is used to calculate the classification score. We can convert any of these 3 fully connected layers into a convolutional layer: For the fully connected layer whose first connection area is [7x7x512], set the filter size to F=7, so that the output data body is [1x1x4096]. For the second fully connected layer, set its filter size to F=1, so the output data body is [1x1x4096]. Do the same for the last fully connected layer, set F=1, and the final output is [1x1x1000] In actual operation, each such transformation needs to reshape the weight W of the fully connected layer into a filter of the convolutional layer.

importance It can be more efficient in the following situations: let the convolutional network slide on a larger input picture to get multiple outputs. This conversion allows us to complete the above operations in a single forward propagation process.

Semantic Segmentation PIC

3.5.3 Semantic Segmentation

Semantic Segmentation: Semantic segmentation is a natural step in the progression from coarse to fine inference:The origin could be located at classification, which consists of making a prediction for a whole input.The next step is localization / detection, which provide not only the classes but also additional information regarding the spatial location of those classes.Finally, semantic segmentation achieves fine-grained inference by making dense predictions inferring labels for every pixel, so that each pixel is labeled with the class of its enclosing object ore region It is also worthy to review some standard deep networks that have made significant contributions to the field of computer vision, as they are often used as the basis of semantic segmentation systems:

AlexNet: Toronto's pioneering deep CNN that won the 2012 ImageNet competition with a test accuracy of 84.6VGG-16: This Oxford's model won the 2013 ImageNet competition with 92.7GoogLeNet: This Google's network won the 2014 ImageNet competition with accuracy of 93.3ResNet: This Microsoft's model won the 2016 ImageNet competition with 96.4

A general semantic segmentation architecture can be broadly thought of as an encoder network followed by a decoder network:

The encoder is usually is a pre-trained classification network like VGG/ResNet followed by a decoder network. The task of the decoder is to semantically project the discriminative features (lower resolution) learnt by the encoder onto the pixel space (higher resolution) to get a dense classification. Unlike classification where the end result of the very deep network is the only important thing, semantic segmentation not only requires discrimination at pixel level but also a mechanism to project the discriminative features learnt at

different stages of the encoder onto the pixel space.

Different approaches employ different mechanisms as a part of the decoding mechanism. Let's explore the 3 main approaches: 1—Region-Based Semantic Segmentation 2—Weakly Supervised Semantic Segmentation 3—Fully Convolutional Network-Based Semantic Segmentation

Semantic Segmentation with Fully-Convolutional Network

The original Fully Convolutional Network (FCN) learns a mapping from pixels to pixels, without extracting the region proposals. The FCN network pipeline is an extension of the classical CNN. The main idea is to make the classical CNN take as input arbitrary-sized images. The restriction of CNNs to accept and produce labels only for specific sized inputs comes from the fully-connected layers which are fixed. Contrary to them, FCNs only have convolutional and pooling layers which give them the ability to make predictions on arbitrary-sized inputs.

One issue in FCN is that by propagating through several alternated convolutional and pooling layers, the resolution of the output feature maps is down sampled. Therefore, the direct predictions of FCN are typically in low resolution, resulting in relatively fuzzy object boundaries. A variety of more advanced FCN-based approaches have been proposed to address this issue, including SegNet, DeepLab-CRF, and Dilated Convolutions.

FCN Architecture Each layer output in a convnet is a three-dimensional array of size h×w×d, where h and w are spatial dimensions, and d is the feature or channel dimension. The first layer is the image, with pixel size h × w, and d channels. Locations in higher layers correspond to the locations in the image they are path-connected to, which are called their receptive fields. Convnets are inherently translation invariant. Their basic components (convolution, pooling, and activation functions) operate on local input regions, and depend only on relative spatial coordinates. Writing xij for the data vector at location (i, j) in a particular layer, and yij for the following layer, these functions compute outputs yij by yij = fks (xsi+i,sj+j0i,j<k) where k is called the kernel size, s is the stride or subsampling factor, and fks determines the layer type: a matrix multiplication for convolution or average pooling, a spatial max for max pooling, or an elementwise nonlinearity for an activation function, and so on for other types of layers. This functional form is maintained under composition, with kernel size and stride obeying the transformation rule fks ∘ gk0s 0 = (f ∘ g)k0+(k1)s 0 ,ss0 . While a general net computes a general nonlinear function, a net with only layers of this form computes a nonlinear filter, which we call a deep filter or fully convolutional network. An FCN naturally operates on an input of any size, and produces an output of corresponding (possibly resampled) spatial dimensions. A real-valued loss function composed with an FCN defines a task. If the loss function is a sum over the spatial dimensions of the final layer, ‘(x;) = P ij ‘ 0 (xij ;), its parameter gradient will be a sum over the parameter gradients of each of its spatial components. Thus stochastic gradient descent on ‘ computed on whole images will be the same as stochastic gradient descent on ‘ 0 , taking all of the final layer receptive fields as a minibatch. When these receptive fields overlap significantly, both feedforward computation and backpropagation are much more efficient when computed layer-by-layer over an entire image instead of independently patch-by-patch.

3.5.4 Upsampling

One of the ways to upsample the compressed image is by Unpooling (the reverse of pooling) using Nearest Neighbor or by max unpooling.

The Convolution operation reduces the spatial dimensions as we go deeper down the network and creates an abstract representation of the input image. This feature of CNN's is very useful for tasks like image classification where you just have to predict whether a particular object is present in the input image or not. But this feature might cause problems for tasks like Object Localization, Segmentation where the spatial dimensions of the object in the original image are necessary to predict the output bounding box or segment the object. To fix this problem various techniques are used such as fully convolutional neural networks where we preserve the input dimensions using 'same' padding. Though this technique solves the problem to a great extent, it also increases the computation cost as now the convolution operation has to be applied to original input dimensions throughout the network.

Another approach used for image segmentation is dividing the network into two parts i.e An Downsampling network and then an Upsampling network. In the Downsampling network, simple CNN architectures are used and abstract representations of the input image are produced. In the Upsampling network, the abstract image representations are upsampled using various techniques to make their spatial dimensions equal to the input image. This kind of architecture is famously known as the Encoder-Decoder network.

Another approach used for image segmentation is dividing the network into two parts i.e An Downsampling network and then an Upsampling network. In the Downsampling network, simple CNN architectures are used and abstract representations of the input image are produced. In the Upsampling network, the abstract image representations are upsampled using various techniques to make their spatial dimensions equal to the input image. This kind of architecture is famously known as the Encoder-Decoder network.

Upsampling Techniques The Downsampling network is intuitive and well known to all of us but very little is discussed about the various techniques used for Upsampling. The most widely used techniques for upsampling in Encoder-Decoder Networks are: 1. Nearest Neighbors: In Nearest Neighbors, as the name suggests we take an input pixel value and copy it to the K-Nearest Neighbors where K depends on the expected output.

2. Bi-Linear Interpolation: In Bi-Linear Interpolation, we take the 4 nearest pixel value of the input pixel and perform a weighted average based on the distance of the four nearest cells smoothing the output.

3. Bed Of Nails: In Bed of Nails, we copy the value of the input pixel at the corresponding position in the output image and filling zeros in the remaining positions.

4. Max-Unpooling: The Max-Pooling layer in CNN takes the maximum among all the values in the kernel. To perform max-unpooling, first, the index of the maximum value is saved for every max-pooling layer during the encoding step. The saved index is then used during the Decoding step where the input pixel is mapped to the saved index, filling zeros everywhere else.

All the above-mentioned techniques are predefined and do not depend on data, which makes them task-specific. They do not learn from data and hence are not a generalized technique.

3.6 Model Architecture

The model consists of six-layer convolution network with two pooling layers

Layer Type Maps Size Kernel size Stride Padding Parameters

Input Input 1 32×20 — — — — 0

C1 Convolution 32 32×20 3×3×1 1 SAME 320

C2 Convolution 32 32×20 3×3×32 1 SAME 9248

C3 Convolution 64 32×20 3×3×64 1 SAME 36x928

C4 Convolution 64 14×8 3×3×64 1 VALID 36x928

C5 Convolution 64 14×8 3×3×64 1 SAME 36x928

C6 Convolution 128 14×8 3×3×128 1 SAME 147x584

P1 pooling 64 16×10 2×2×64 2 -

P2 pooling 128 7×4 2×2×128 2 -

3.7 Residual Neural Network—ResNet

After the first CNN-based architecture (AlexNet) that win the ImageNet 2012 competition, Every subsequent winning architecture uses more layers in a deep neural network to reduce the error rate. This works for less number of layers, but when we increase the number of layers, there is a common problem in deep learning associated with that called Vanishing/Exploding gradient. This causes the gradient to become 0 or too large. Thus when we increases number of layers, the training and test error rate also increases.

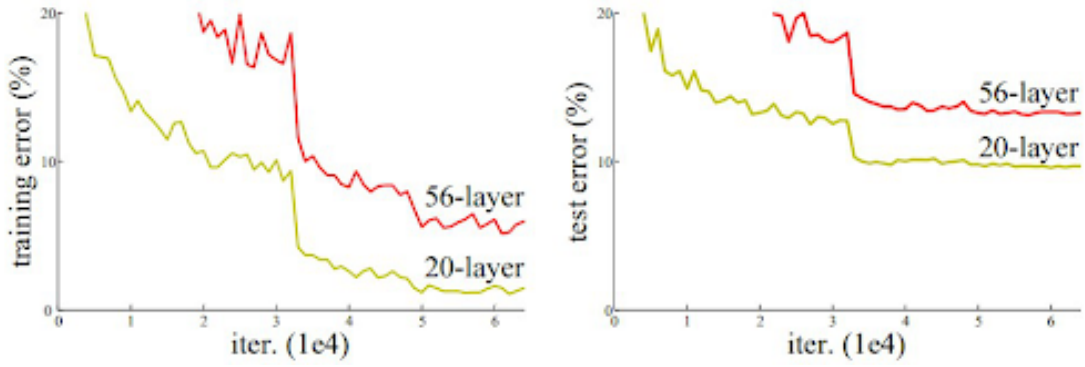


Figure 3.39

In the above plot, we can observe that a 56-layer CNN gives more error rate on both training and testing dataset than a 20-layer CNN architecture, If this was the result of over fitting, then we should have lower training error in 56-layer CNN but then it also has higher training error.

After analysing more on error rate the authors were able to reach a conclusion that it is caused by vanishing/exploding gradient.

ResNet, which was proposed in 2015 by researchers at Microsoft Research, introduced a new architecture called Residual Network.

The degradation (of training accuracy) indicates that not all systems are similarly easy to optimise.

3.7.1 Residual Block

In order to solve the problem of the vanishing/exploding gradient, this architecture introduced the concept called Residual Network. In this network we use a technique called skip connections . The skip connection skips training from a few layers and connects directly to the output.

The approach behind this network is instead of layers learning the underlying mapping, we allow the network to fit the residual mapping. So, instead of say H(x), initial mapping, let the network fit, F(x) := H(x) – x which gives H(x) := F(x) + x.

The advantage of adding this type of skip connection is because if any layer hurt the performance of architecture then it will be skipped by regularisation. So, this results in training very deep neural network without the problems caused by vanishing/exploding gradient.

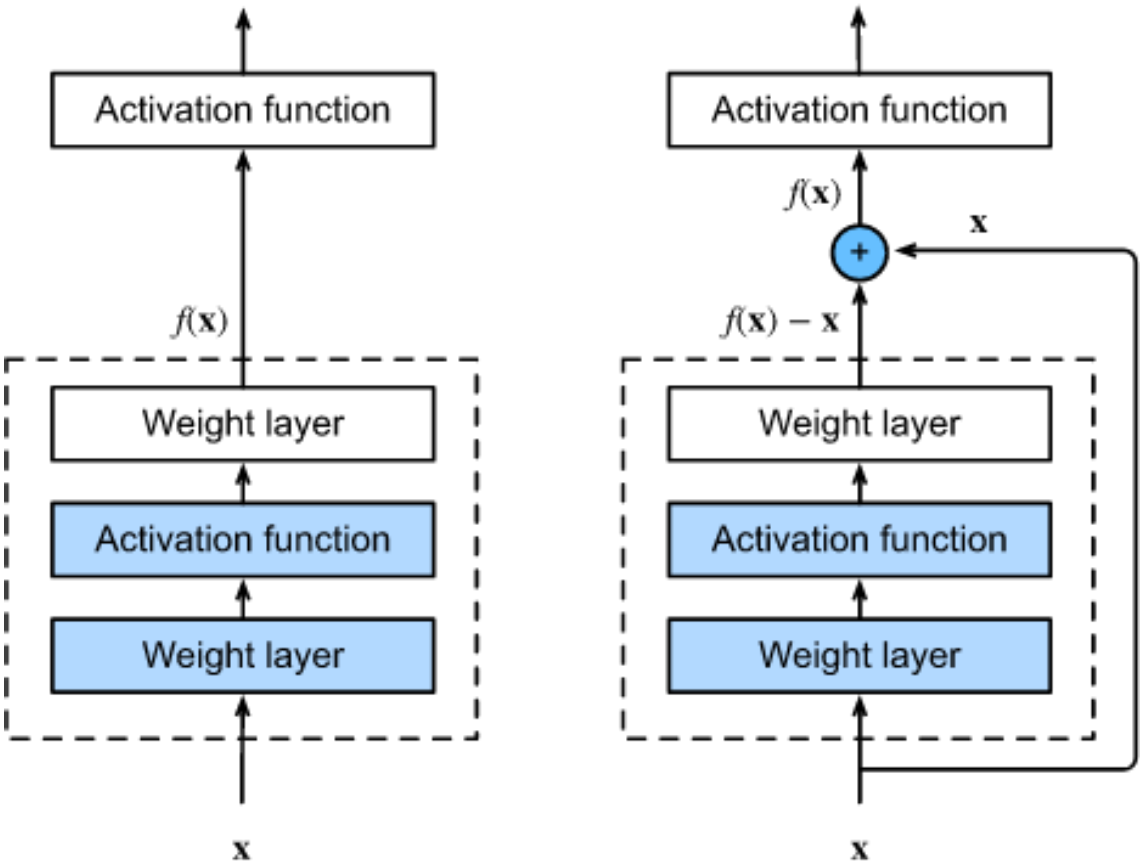


Figure 3.40: Comparison between basic architectures of ANN & ResNet

Let us focus on a local part of a neural network, as depicted in the figure above. Denote the input by x. We assume that the desired underlying mapping we want to obtain by learning is f(x), to be used as the input to the activation function on the top. On the left of Figure above, the portion within the dotted-line box must directly learn the mapping f(x).

On the right, the portion within the dotted-line box needs to learn the residual mapping f(x)-x, which is how the residual block derives its name. If the identity mapping f(x)=x is the desired underlying mapping, the residual mapping is easier to learn: we only need to push the weights and biases of the upper weight layer (e.g., fully-connected layer and convolutional layer) within the dotted-line box to zero. The right figure in Figure above illustrates the residual block of ResNet, where the solid line carrying the layer input x to the addition operator is called a residual connection (or shortcut connection).

With residual blocks, inputs can forward propagate faster through the residual connections across layers.

ResNet follows VGG's full 3×3 convolutional layer design. The residual block has two 3×3 convolutional layers with the same number of output channels. Each convolutional layer is followed by a batch normalisation layer and a ReLU activation function,Then, we skip these two convolution operations and add the input directly before the final ReLU activation function. This kind of design requires that the output of the two convolutional layers has to be of the same shape as the input, so that they can be added together. If we want to change the number of channels, we need to introduce an additional 1×1 convolutional

layer to transform the input into the desired shape for the addition operation.

3.7.2 Deep Residual Learning

Let us consider H(x) as an underlying mapping to be fit by a few stacked layers (not necessarily the entire net), with x denoting the inputs to the first of these layers.

If one hypothesizes that multiple nonlinear layers can asymptotically approximate complicated functions , then it is equivalent to hypothesize that they can asymptotically approximate the residual functions, i.e., H(x) x (assuming that the input and output are of the same dimensions). So rather than expect stacked layers to approximate H(x), we explicitly let these layers approximate a residual function F(x) := H(x) x. The original function thus becomes F(x)+x. Although both forms should be able to asymptotically approximate the desired functions (as hypothesized), the ease of learning might be different. This reformulation is motivated by the counter-intuitive phenomena about the degradation problem. As we discussed in the introduction, if the added layers can be constructed as identity mappings, a deeper model should have training error no greater than its shallower counterpart. The degradation problem suggests that the solvers might have difficulties in approximating identity mappings by multiple nonlinear layers. With the residual learning reformulation, if identity mappings are optimal, the solvers may simply drive the weights of the multiple nonlinear layers toward zero to approach identity mappings. In real cases, it is unlikely that identity mappings are optimal, but our reformulation may help to precondition the problem. If the optimal function is closer to an identity mapping than to a zero mapping, it should be easier for the solver to find the perturbations with reference to an identity mapping, than to learn the function as a new one.

3.7.3 Identity Mapping by Shortcuts

We adopt residual learning to every few stacked layers. Formally, in this paper we consider a building block defined as:

$$y = F(x, Wi) + x \tag{3.1}$$

Here x and y are the input and output vectors of the layers considered. The function F(x, Wi) represents the residual mapping to be learned. For the example in Fig. 2 that has two layers, F = W2(W1x) in which denotes ReLU and the biases are omitted for simplifying notations. The operation F + x is performed by a shortcut connection and element-wise addition. We adopt the second nonlinearity after the addition (i.e., (y)). The shortcut connections in Eqn.(3.1) introduce neither extra parameter nor computation complexity. This is not only attractive in practice but also important in our comparisons between plain and residual networks. We can fairly compare plain/residual networks that simultaneously have the same number of parameters, depth, width, and computational cost (except for the negligible element-wise addition). The dimensions of x and F must be equal in Eqn.(3.1). If this is not the case (e.g., when changing the input/output channels), we can perform a linear projection Ws by the shortcut connections to match the dimensions:

$$y = F(x, Wi) + Wsx \tag{3.2}$$

We can also use a square matrix Ws in Eqn.(3.1). But we will show by experiments that the identity mapping is sufficient for addressing the degradation problem and is economical, and thus Ws is only used when matching dimensions. The form of the residual function F is flexible. Experiments in this paper involve a function F that has two or three layers, while more layers are possible. But if F has only a single layer, Eqn.(3.1) is similar to a linear layer: y = W1x + x, for which we have not observed advantages. We also note that although the above notations are about fully-connected layers for simplicity, they are applicable to convolutional layers. The function F(x, Wi) can represent multiple convolutional layers. The element-wise addition is performed on two feature maps, channel by channel.

3.7.4 Network Architecture(ResNet-34):

the network uses a 34-layer plain network architecture inspired by VGG-19 in which then the shortcut connection is added. These shortcut connections then convert the architecture into a residual network.

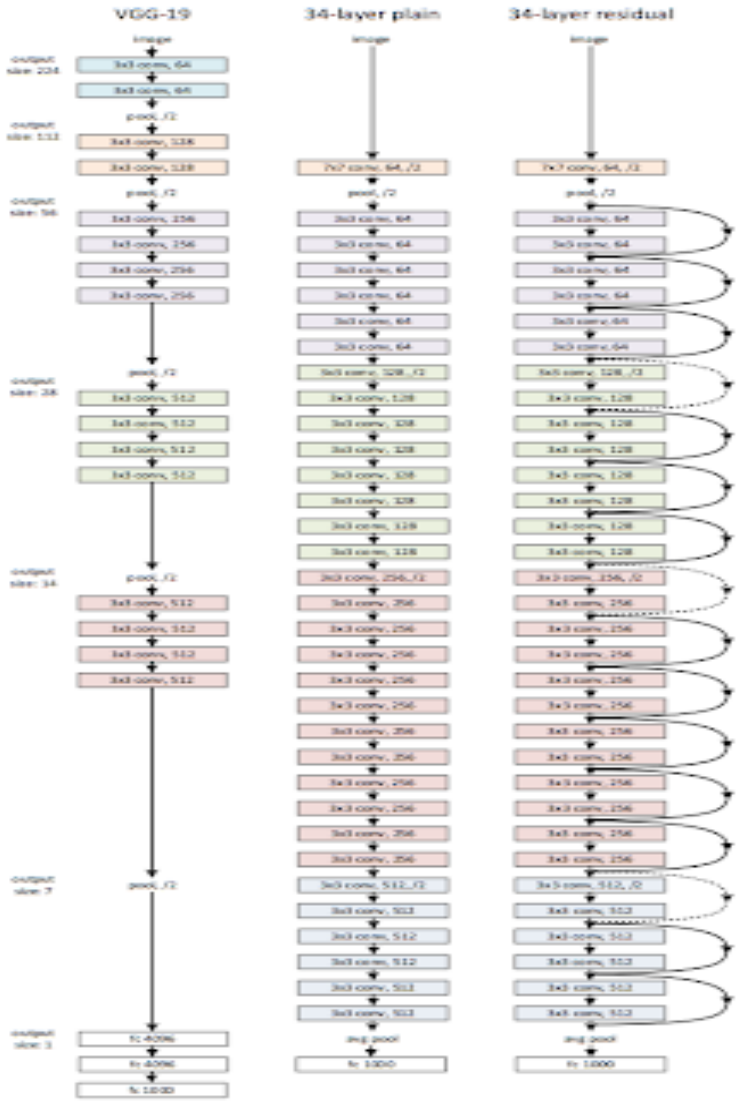


Figure 3.41: Example network architectures for ImageNet.

Left: theVGG-19 model [41] (19.6 billion FLOPs) as a reference.

Middle: a plain network with 34 parameter layers (3.6 billion FLOPs).

Right: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions.

Plain Network: Our plain baselines (Figure3.40, middle) are mainly inspired by the philosophy of VGG nets (Figure3.4 left The convolutional layers mostly have 3×3 filters and follow two simple design rules:

- For the same output feature map size, the layers have the same number of filters.
- If the feature map size is halved, the number of filters is doubled so as to preserve the time complexity per layer. We perform down-sampling directly by convolutional layers that have a stride of 2. The network ends with a global average pooling layer and a 1000-way fully-connected layer with softmax. The total number of weighted layers is 34 in Fig. 3.40 (middle).

It is worth noticing that our model has fewer filters and lower complexity than VGG nets (Fig. 3.40, left). Our 34- layer baseline has 3.6 billion FLOPs (multiply-adds), which is only 18% of VGG-19 (19.6 billion FLOPs).

Residual Network. Based on the above plain network, we insert shortcut connections (Figure, right) which turn the network into its counterpart residual version. The identity shortcuts (Eqn.(3.1)) can be directly used when the input and output are of the same dimensions (solid line shortcuts in Figure). When the dimensions increase (dotted line shortcuts in Figure3.40), we consider two options:

- The shortcut still performs identity mapping, with extra zero entries padded for increasing dimensions. This option introduces no extra parameter
- The projection shortcut in Eqn.(3.2) is used to match dimensions (done by 1×1 convolutions). For both options, when the shortcuts go across feature maps of two sizes, they are performed with a stride of 2.

3.8 Recurrent Neural Network—RNN

A recurrent neural network (RNN) is a type of artificial neural network which uses sequential data or time series data. These deep learning algorithms are commonly used for ordinal or temporal problems, such as language translation, natural language processing (nlp), speech recognition, and image captioning; they are incorporated into popular applications such as Siri, voice search, and Google Translate. Like feedforward and convolutional neural networks (CNNs), recurrent neural networks utilise training data to learn. They are distinguished by their "memory" as they take information from prior inputs to influence the current input and output.

While traditional deep neural networks assume that inputs and outputs are independent of each other, the output of recurrent neural networks depend on the prior elements within the sequence. While future events would also be helpful in determining the output of a given sequence, unidirectional recurrent neural networks cannot account for these events in their predictions.

3.8.1 Recurrent Neural Network vs. Feed forward Neural Network

distinguishing characteristic of recurrent networks is that they share parameters across each layer of the network. While feed forward networks have different weights across each node, recurrent neural networks share the same weight parameter within each layer of the network. That said, these weights are still adjusted in the through the processes of back propagation and gradient descent to facilitate reinforcement learning.

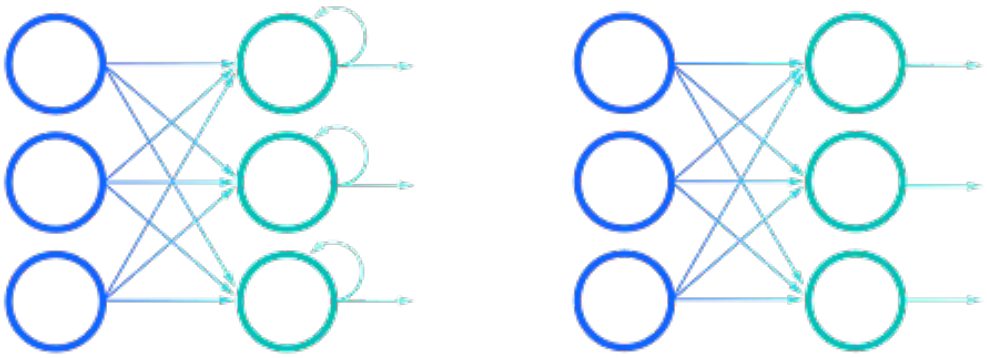


Figure 3.42: Comparison of Recurrent Neural Networks (on the left) and Feed forward Neural Networks (on the right)

3.8.2 Overview

Recurrent neural networks leverage back propagation through time (BPTT) algorithm to determine the gradients, which is slightly different from traditional back propagation as it is specific to sequence data. The principles of BPTT are the same as traditional back propagation, where the model trains itself by calculating errors from its output layer to its input layer. These calculations allow us to adjust and fit the parameters of the model

appropriately. BPTT differs from the traditional approach in that BPTT sums errors at each time step whereas feed forward networks do not need to sum errors as they do not share parameters across each layer.

Through this process, RNNs tend to run into two problems, known as exploding gradients and vanishing gradients. These issues are defined by the size of the gradient, which is the slope of the loss function along the error curve. When the gradient is too small, it continues to become smaller, updating the weight parameters until they become insignificant—i.e. 0. When that occurs, the algorithm is no longer learning. Exploding gradients occur when the gradient is too large, creating an unstable model. In this case, the model weights will grow too large, and they will eventually be represented as NaN. One solution to these issues is to reduce the number of hidden layers within the neural network, eliminating some of the complexity in the RNN model.
The simplest form of an RNN [24, 30] is a layer where the output is connected to the input. Unlike FF layers, in RNN, the result at time t (y(t)) is a function of both the current input xt and the previous state:

$$y(t1) : y(t) = \sigma(x(t)W + y(t1)U + b) \tag{3.3}$$

Here, again is a non-linear activation function. The structure of an RNN layer allows the network to contain memory, since it has access to information from previous time-stamps. RNNs are known to suffer from a phenomenons called "vanishing gradient" and "exploding gradient" , while training, the gradient of the loss function may not propagate to the first layers (i.e., the layers closer to the input layer) or may reach very large values (thus updating the layer weights too much). These problems prevent RNNs from learning long temporal dependencies . A common solution for these problems is called Long Short Term Memory layer.

3.8.3 RNN topography

A Recurrent Neural Network,trained using the Backpropagation

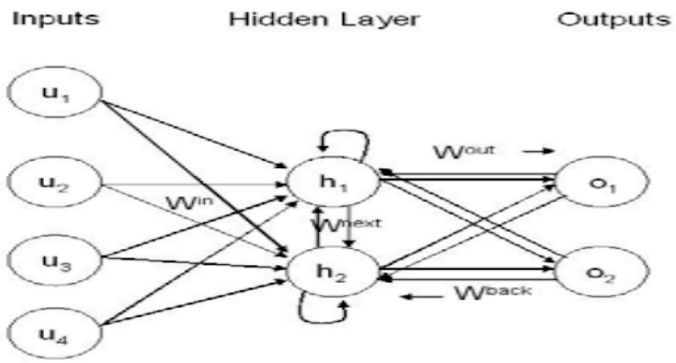


Figure 3.43: Topography of a RNN with 4 Input neurons, 2 Hidden neurons and a single Output neuron

The RNN has three sets of neurons, Input (I), Hidden (H) and a single Output neuron (Fig. 1 shows the topography). The following weight matrices define the connections

between these layers. Input to Hidden neurons:

- a I × H matrix Win.
- Hidden to Hidden neurons: a H × H matrix Wnext.
- Input and Hidden to Output neurons: a (I + H) × 1 matrix Wout.
- Output neurons to Hidden neurons: a 1 × H matrix Wback.

Using these weight matrices, the activation functions were defined as follows:

$$h(t+1) = tanh(u(t+1)Win + h(t)Wnext + o(t)Wback) \tag{3.4}$$

$$o(t+1) = [u(t+1), h(t+1)]Wout(2) \tag{3.5}$$

To train the RNN the activation function is used to calculate the value of each hidden neuron and each output neuron at all times t. The error of the output neurons and hidden neurons are calculated for the last time step by:

$$\gamma(t) = class(t)o(t) \tag{3.6}$$

$$\sigma(t) = (1h(t)^2)(t)Wout \tag{3.7}$$

The errors of each previous time step are then calculated by: $\gamma(t) = class(t)o(t) + h(t+1)Wback$
$\sigma(t) = (1h(t)2)(t)Wout + h(t+1)Wback$
Finally the weights are updated by these error values using:

$$Wini, j+ = lTt = 0i(t)Ij(t) \tag{3.8}$$

$$Wouti+ = lTt = 0\gamma(t)hi(t) \tag{3.9}$$

$$Wnexti+ = lT1t = 0o(t)(t+1) \tag{3.10}$$

$$Wbacki, j+ = lT1t = 0(t+1)hi(t) \tag{3.11}$$

Where l is the learning rate. To classify using a trained RNN, the error between each individual class is calculated by

$$error(class) = Tt = 0(classo(t))2(11) \tag{3.12}$$

and the smallest is selected.

3.9 Recurrent Neural Network with Attention

3.9.1 Attention

In psychology, attention is the cognitive process of selectively concentrating on one or a few things while ignoring others.

A neural network is considered to be an effort to mimic human brain actions in a simplified manner. Attention Mechanism is also an attempt to implement the same action of selectively concentrating on a few relevant things, while ignoring others in deep neural networks.

Let me explain what this means. Let's say you are seeing a group photo of your first school. Typically, there will be a group of children sitting across several rows, and the teacher will sit somewhere in between. Now, if anyone asks the question, "How many people are there?", how will you answer it?

Simply by counting heads, right? You don't need to consider any other things in the photo. Now, if anyone asks a different question, "Who is the teacher in the photo?", your brain knows exactly what to do. It will simply start looking for the features of an adult in the photo. The rest of the features will simply be ignored. This is the 'Attention' which our brain is very adept at implementing.

3.9.2 Problem definition

Recurrent Neural Networks (RNNs) have been used successfully for many tasks involving sequential data such as machine translation, sentiment analysis, image captioning, time-series prediction etc.

Attention is a mechanism combined in the RNN allowing it to focus on certain parts of the input sequence when predicting a certain part of the output sequence, enabling easier learning and of higher quality. Combination of attention mechanisms enabled improved performance in many tasks making it an integral part of modern RNN networks.

We start by briefly going over basic RNNs. The RNN encoder-decoder architecture we will focus on looks like this:

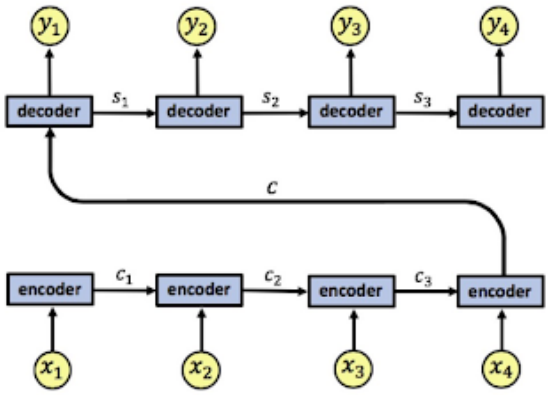


Figure 3.44: RNN encoder-decoder architecture

An RNN encoder-decoder architecture, we take an architecture with 4 time steps for

simplicity The RNN encoder has an input sequence x1, x2, x3, x4. We denote the encoder states by c1, c2, c3. The encoder outputs a single output vector c which is passed as input to the decoder. Like the encoder, the decoder is also a single-layered RNN, we denote the decoder states by s1, s2, s3 and the network's output by y1, y2, y3, y4. A problem with this architecture lies in the fact that the decoder needs to represent the entire input sequence x1, x2, x3, x4 as a single vector c, which can cause information loss. Moreover, the decoder needs to decipher the passed information from this single vector, a complex task in itself. A potential issue with this encoder–decoder approach is that a neural network needs to be able to compress all the necessary information of a source sentence into a fixed-length vector. This may make it difficult for the neural network to cope with long sentences, especially those that are longer than the sentences in the training corpus.

3.9.3 Attention in RNN

An attention RNN looks like this:

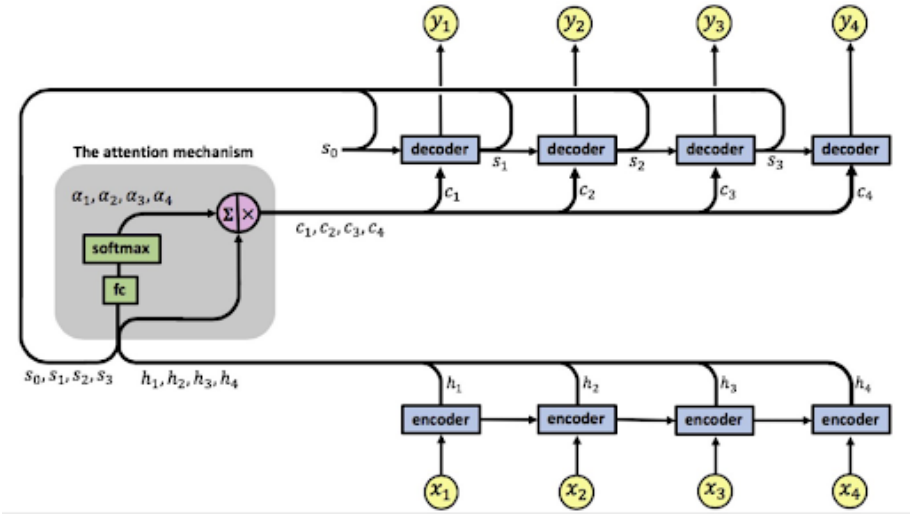


Figure 3.45: a single layer RNN encoder

Our attention model has a single layer RNN encoder, again with 4-time steps. We denote the encoder's input vectors by x1, x2, x3, x4 and the output vectors by h1, h2, h3, h4.

The attention mechanism is located between the encoder and the decoder, its input is composed of the encoder's output vectors h1, h2, h3, h4 and the states of the decoder s0, s1, s2, s3, the attention's output is a sequence of vectors called context vectors denoted by c1, c2, c3, c4.

3.9.4 context vectors

The context vectors enable the decoder to focus on certain parts of the input when predicting its output. Each context vector is a weighted sum of the encoder's output vectors h1, h2, h3, h4, each vector hi contains information about the whole input sequence (since it has access to the encoder states during its computation) with a strong focus on the parts surrounding the i-th vector of the input sequence. The vectors h1, h2, h3, h4 are scaled by weights ij capturing the degree of relevance of input xj to output at time i, yi.

The context vectors c1, c2, c3, c4 are given by:

$$c_i = \sum_{j=1}^{4} \alpha_{ij} h_j$$

Figure 3.46: context vectors

The attention weights are learned using an additional fully-connected shallow network, denoted by fc, this is where the s0, s1, s2, s3 part of the attention mechanism's input comes into play. Computation of the attention weights is given by:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{4} exp(e_{ik})}$$

$$where \qquad e_{ij} = \mathrm{fc}(s_{i-1}, h_j)$$

Figure 3.47: Computation of the attention weights

The attention weights are learned using the attention fully-connected network and a softmax function:

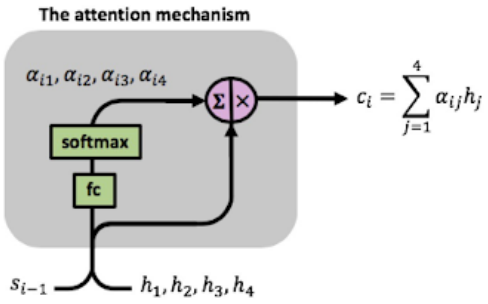


Figure 3.48: attention weights are learned using the attention fully-connected network and a softmax function

At time step i, the mechanism has h1,h2,h3,h4 and si-1 as inputs, it uses the fc neural network and the softmax function to compute the attention weights i1, i2, i3, i4, these are then used in the computation of the context vector ci.

As can be seen in the above image, the fully-connected network receives the concatenation of vectors [si-1,hi] as input at time step i. The network has a single fully-connected layer, the outputs of the layer, denoted by eij, are passed through a softmax function computing the attention weights, which lie in [0,1].

Notice that we are using the same fully-connected network for all the concatenated pairs [si-1,h1], [si-1,h2], [si-1,h3], [si-1,h4], meaning there is a single network learning the attention weights.

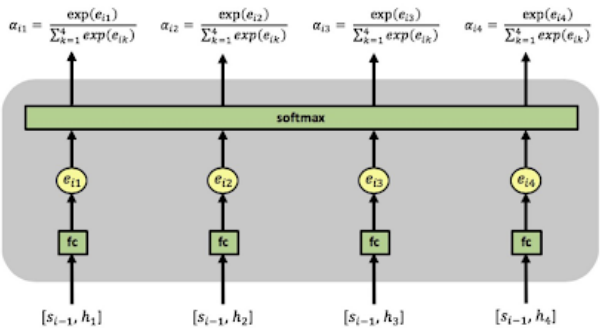


Figure 3.49: single network learning the attention weights

Computation of attention weights at time step i, notice how this needs to be computed separately on every time step since the computation at time step i involves si-1, the decoder's state form time step i-1

The attention weights ij reflect the importance of hj with respect to the previous hidden state si1 in deciding the next state si and generating yi. A large ij attention weight causes the RNN to focus on input xj (represented by the encoder's output hj), when predicting the output yi.

The fc network is trained along with the encoder and decoder using back propagation, the RNN's prediction error terms are back propagated backward through the decoder, then through the fc attention network and from there to the encoder.

Notice that since the attention weights are learned using an additional neural network fc, we have an additional set of weights allowing this learning to take place, we denote this weight matrix by Wa.

An RNN with 4 input time steps and 4 output time steps will have the following weight matrices fine-tuned during the training process. Note the dimensions 4×4 of the attention matrix, connecting between every input to every output:

RNN encoder weights matrix W_e
RNN decoder weights matrix W_d
RNN attention weights matrix $\alpha_{4\times4}$
fc *weights matrix* W_a

Figure 3.50: weight matrices

This mechanism enables the decoder to decide which parts of the input sequence to pay attention to. By letting the decoder have an attention mechanism, we relieve the encoder from having to encode all information in the input sequence into a single vector. The information can be spread throughout the sequence h1, h2, h3, h4 which can be selectively retrieved by the decoder.

3.9.5 Computing the attention weights and context vectors

The first act performed is the computation of vectors h1, h2, h3, h4 by the encoder. These are then used as inputs of the attention mechanism. This is where the decoder is first involved by inputting its initial state vector s0 and we have the first attention input sequence [s0, h1], [s0, h2], [s0, h3], [s0, h4].

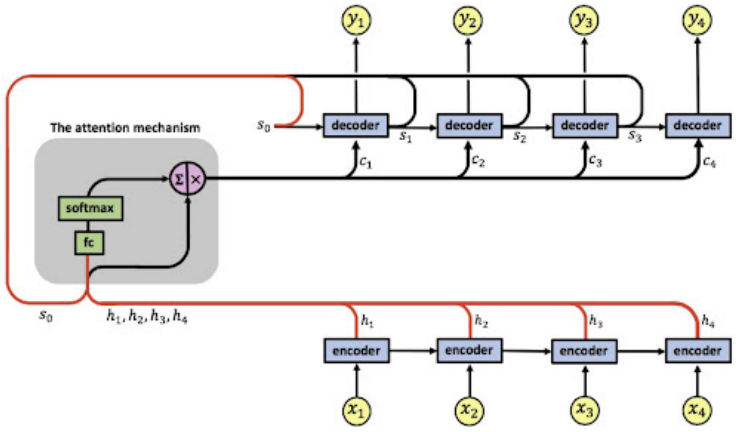


Figure 3.51: Computing the attention weights and context vectors

The attention mechanism computes the first set of attention weights 11, 12, 13, 14 enabling the computation of the first context vector c1. The decoder now uses [s0,c1] and computes the first RNN output y1

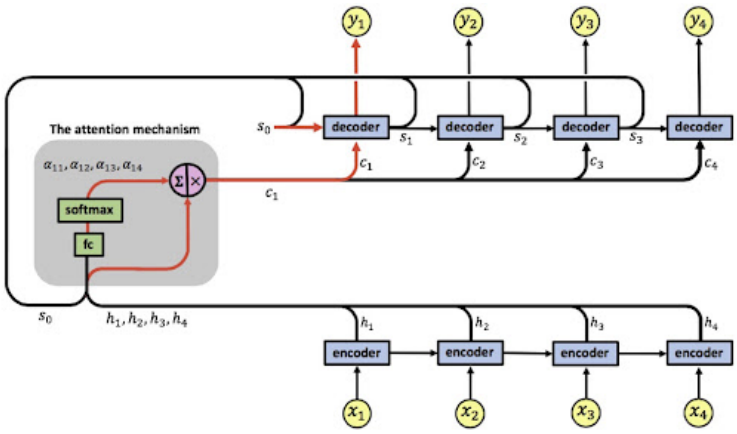


Figure 3.52: Computing the attention weights and context vectors

At the following time step the attention mechanism has as input the sequence [s1, h1], [s1, h2], [s1, h3], [s1, h4].

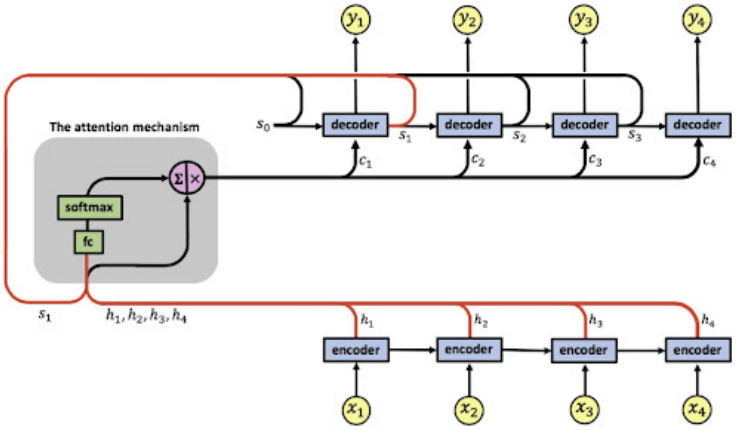


Figure 3.53: Computing the attention weights and context vectors

It computes a second set of attention weights 21, 22, 23, 24 enabling the computation of the first context vector c2. The decoder uses [s1,c2] and computes the second RNN output y2.

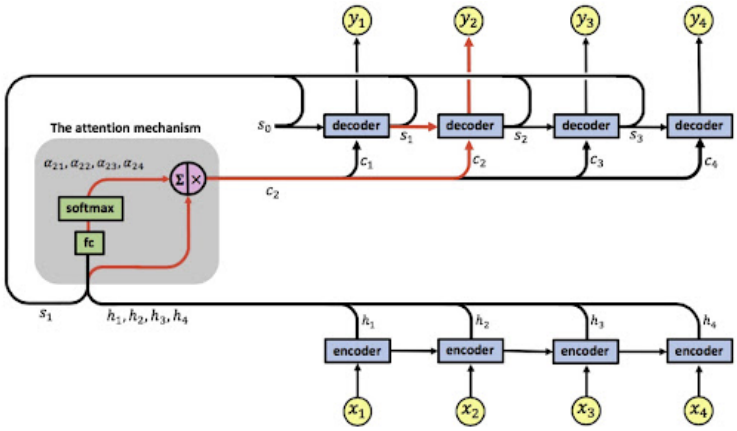


Figure 3.54: Computing the attention weights and context vectors

At the next time step the attention mechanism has an input sequence [s2, h1], [s2, h2], [s2, h3], [s2, h4].

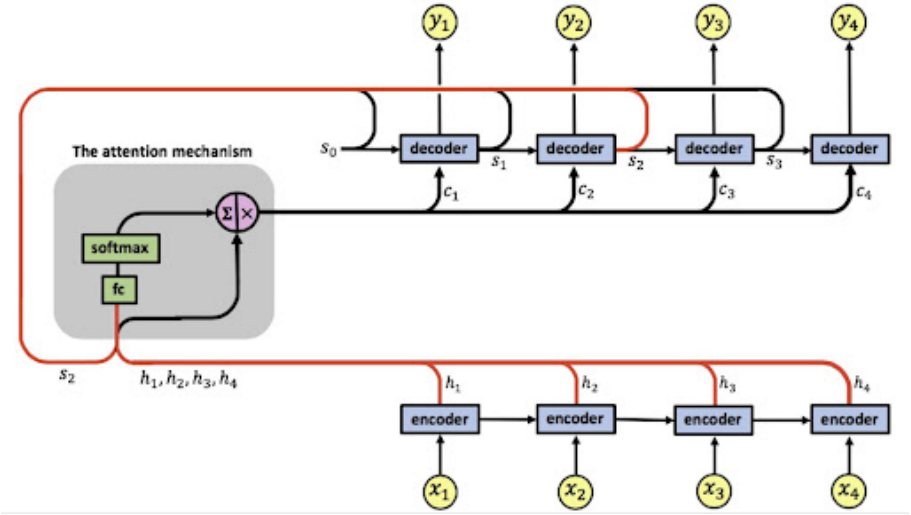


Figure 3.55: Computing the attention weights and context vectors

And computes a third set of attention weights 31, 32, 33, 34 enabling the computation of the third context vector c3. The decoder uses [s2,c3] and computes the following output y3.

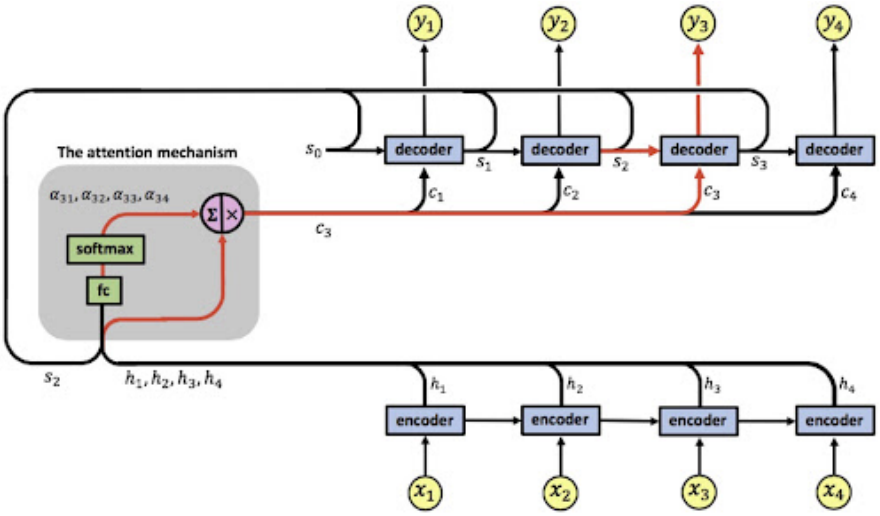


Figure 3.56: Computing the attention weights and context vectors

At the next time step the attention mechanism has an input sequence [s3, h1], [s3, h2], [s3, h3], [s3, h4].

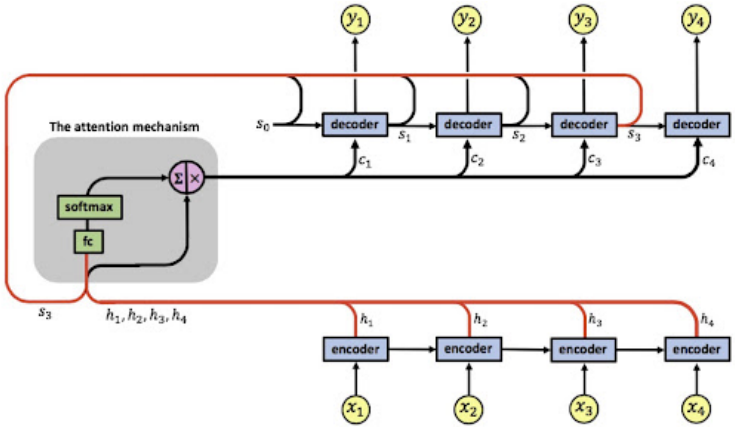


Figure 3.57: Computing the attention weights and context vectors

It computes a fourth set of attention weights 41, 42, 43, 44 enabling the computation of the fourth context vector c4. The decoder uses [s3,c4] and computes the final output y4.

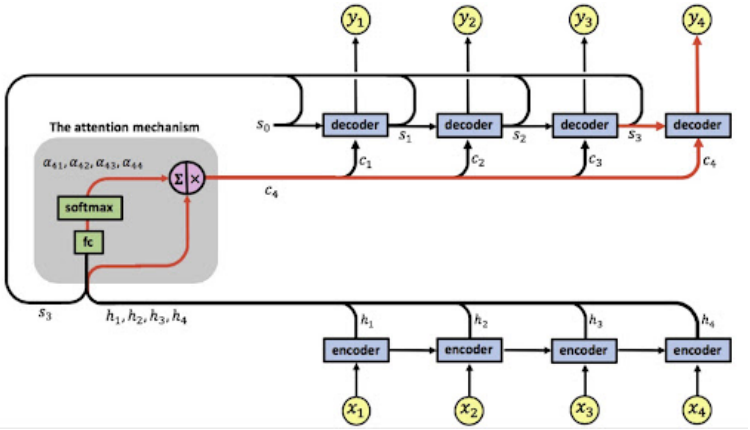


Figure 3.58: Computing the attention weights and context vectors

We end with an example from [1], the task is English-French machine translation. Below are two alignments found by the attention RNN. The x-axis and y-axis of each plot correspond to the words in the source sentence (English) and the generated translation (French), respectively. Each pixel shows the weight ij of the j-th source word and the i-th target word, in gray scale (0: black, 1: white).

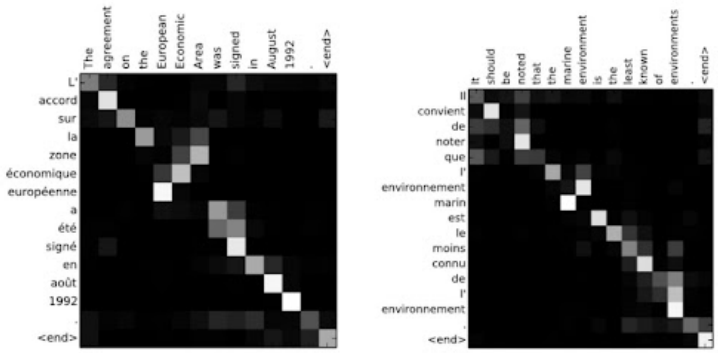


Figure 3.59: the weight ij of the j-th source word and the i-th target word,

We see how the attention mechanism allows the RNN to focus on a small part of the input sentence when outputting the translation. Notice how the larger attention parameters (given by the white pixels) connect corresponding parts of the English and French sentences enabling the network in [1] to achieve state of the art results

3.10 Evaluation Metrics

The idea of building machine learning models works on a constructive feedback principle. You build a model, get feedback from metrics, make improvements and continue until you achieve a desirable accuracy. Evaluation metrics explain the performance of a model.

An important aspect of evaluation metrics is their capability to discriminate among model results.

Performance metrics are a part of every machine learning pipeline. They tell you if you're making progress, and put a number on it. All machine learning models, whether it's linear regression, classification, need a metric to judge performance.

the model you have trained will always perform better on the data set you have trained it. But we train machine learning models to perform well while solving real-world problems where data flows continuously. If we are using a model that is not capable enough to perform well, there is no point in using machine learning to solve your problems. This is where performance evaluation metrics come in. A performance evaluation metric calculates whether your trained machine learning model will perform well in solving the problem it was trained for or not.

Metrics are different from loss functions. Loss functions show a measure of model performance. They're used to train a machine learning model (using some kind of optimization like Gradient Descent), and they're usually differentiable in the model's parameters. Metrics are used to monitor and measure the performance of a model (during training and testing), and don't need to be differentiable. However, if for some tasks the performance metric is differentiable, it can also be used as a loss function (perhaps with some regularization added to it), such as MSE.

choice of metrics influences how the performance of machine learning algorithms is measured and compared. They influence how we weight the importance of different characteristics in the results.

3.10.1 Types of evaluation metrics

Regression	Classification	Recommender System
• Mean Absolute Error (MAE) • Root Mean Squared Error (RMSE) • R-Squared and Adjusted R-Squared	• Recall • Precision • F1-Score • Accuracy • Area Under the Curve (AUC)	• Mean Reciprocal Rank • Root Mean Squared Error (RMSE)

Figure 3.60: types of evaluations metrics

3.10.2 Confusion Matrix

It is a common way of presenting true positive (tp), true negative (tn), false positive (fp) and false negative (fn) predictions. Those values are presented in the form of a matrix where the Y-axis shows the true classes while the X-axis shows the predicted classes. Confusion matrices represent counts from predicted and actual values.

3.10.3 elements of confusion matrix

It represents the different combinations of Actual VS Predicted values. Let's define them one by one.

TP: True Positive: The values which were actually positive and were predicted positive.

FP: False Positive: The values which were actually negative but falsely predicted as positive. Also known as Type I Error.

FN: False Negative: The values which were actually positive but falsely predicted as negative. Also known as Type II Error.

TN: True Negative: The values which were actually negative and were predicted negative.

3.10.4 Types of evaluation metrics

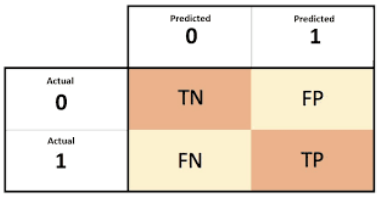


Figure 3.61: confusion matrix

A perfect model would be that which has 0 False Positives and 0 False Negatives, but this is practically impossible in reality. When assessing the results, there is no standard

action plan to follow on what you should minimise. This would depend completely on the business scenario you are working on.

It can be applied to binary classification as well as for multi class classification problems It is calculated on class predictions, which means the outputs from your model need to be thresholds first.

3.10.5 Accuracy

It is one of the simplest performance measures and simply the ratio of correctly predicted observation and total observation. Accuracy is calculated as the summation of true positive and true negative divided by total number of subject in the study

$$Accuracy = (TP + TN)/(TP + TN + FP + FN) \tag{3.13}$$

3.10.6 Precision

Refers to the ability of a classification model to identify only the relevant points. Precision iscalculated as the number of true positive divided by the summation of the number of true positive and truenegative.

$$precision = TP/(TP + TN) \tag{3.14}$$

3.10.7 Recall

It refers to the ability of a model to find all the relevant cases within a dataset. Recall is calculatedas the number of true positive divided by the summation of the number of true positive and false negative

$$Recall = TP/(TP + FN) \tag{3.15}$$

3.10.8 Kappa

Kappa is an important measure of classifier performance on imbalanced dataset. It compares an observed accuracy with an expected accuracy (random chance). Kappa is calculated as total accuracy minus random accuracy divided by one minus random accuracy.

$$Kappa = (totalaccuracy - RandomAccracy)/(1 - RandomAccuracy) \tag{3.16}$$

$$TotalAccuracy = (TP + TN)(TP + TN + FP + FN) \tag{3.17}$$

$$RandomAccuracy = (TN/FP)*(TN+FN)+(FN+FP)*(FP+TP)/TOTAL*TOTAL \tag{3.18}$$

$$Total = TP + TN + FP + FN \tag{3.19}$$

3.10.9 Receiver Operating Characteristic (ROC) Curve

An ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters:
1-True Positive Rate

2-False Positive Rate

True Positive Rate (TPR) is a synonym for recall and is therefore defined as follows:

$$TPR = TP/TP + FN \tag{3.20}$$

False Positive Rate (FPR) is defined as follows:

$$FPR = FP/FP + TN \tag{3.21}$$

An ROC curve plots TPR vs. FPR at different classification thresholds. Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives. The following figure shows a typical ROC curve.

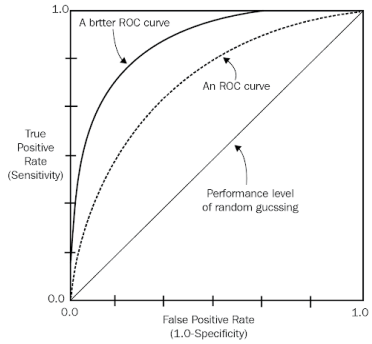


Figure 3.62: ROC curve

The Area Under the Curve (AUC) is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve. The higher the AUC, the better the performance of the model at distinguishing between the positive and negative classes.
When AUC = 1, then the classifier is able to perfectly distinguish between all the Positive and the Negative class points correctly. If, however, the AUC had been 0, then the classifier would be predicting all Negatives as Positives, and all Positives as Negatives
When 0.5<AUC<1, there is a high chance that the classifier will be able to distinguish the positive class values from the negative class values. This is so because the classifier is able to detect more numbers of True positives and True negatives than False negatives and False positives.
When AUC=0.5, then the classifier is not able to distinguish between Positive and Negative class points. Meaning either the classifier is predicting random class or constant class for all the data points. So, the higher the AUC value for a classifier, the better its ability to distinguish between positive and negative classes.

Chapter 4

Proposed Work

Contents

4.1 Data Preprocessing

4.1.1 Data Type Conversions —EDF

Our Dataset is of EDF file type which is "European Data Format" (EDF) ;it is a standard file format designed for exchange and storage of medical time series. Being an open and non-proprietary format, EDF(+) is commonly used to archive, exchange and analyse data from commercial devices in a format that is independent of the acquisition system.

In our case it is used to store the EEG Readings.

The original EDF files are particularly easy to read, they consist of two header blocks followed by all the data. The first header block contains various information about the recording, the device specification, ADC range and filters. Pertaining to the data it contains the number of data records, the length of one record and the number of "signals" (electrodes for EEG). All of the fields in the two headers are plain human readable ASCII characters. The header is 256 bytes long, the entries are summarised below.

Bytes	Description
8	version of this data format, usually 0 for original EDF
80	patient identification
80	local recording identification
8	startdate of recording (dd.mm.yy)
8	starttime of recording (hh.mm.ss)
8	number of bytes in header record
44	not used in original EDF specification
8	number of data records
8	duration of a single data record in seconds
4	number of signals in data record

The second header has 256 bytes for each signal, but these 256 bytes are split on a per signal basis. For example, the first entry is a label for the signal, it is 16 bytes long. The second header will start with the labels for each signal. This is followed by all the transducer types, and so on. The second header entries are as follows.

Bytes	Description
16	label for the signal
80	transducer type
8	units
8	minimum possible value in units
8	maximum possible value in units
8	minimum value numerically
8	maximum value numerically
80	type of any prefiltering
8	number of samples in each data record
32	reserved

The data follows, in the original EDF format the data was represented in two byte signed integers, little endian.

To parse an EDF file the minimum might be as follows.

Read 256 byte header. Extract the number of records (nrecords), the number of signals (nsignals) and optionally the duration of a record if you want to compute sampling frequency.

Read nsignals times each of the second header entries. Extract the number of samples in each data record (nsamples).

Read nrecords * nsamples * nsignals of data values.

Explicitly the ordering of the samples is as follows.

```
for (i=0;i<nrecords;i++) {
  for (j=0;j<nsamples;j++) {
     for (ns=0;ns<nsignals;ns++) {
        read 2 byte integer sample
```

```
        }
      }
   }
```

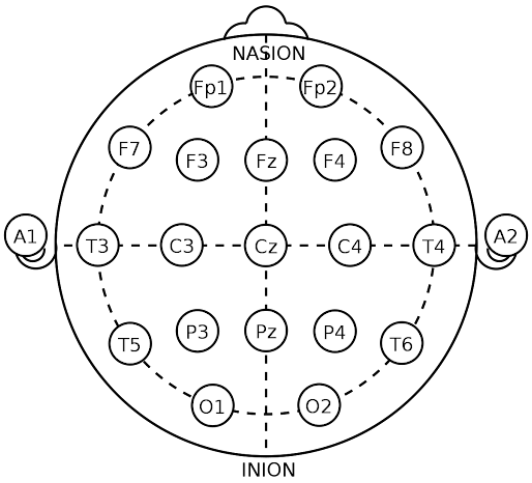
electrode positions are extracted where from it as shown below.

```
def get_electrode_positions():
    positions = dict()
    with io.open("electrode_positions.txt", "r") as pos_file:
        for line in pos_file:
            parts = line.split()
            positions[parts[0]] = tuple([float(part) for part in parts[1:]])
    return positions
```

the function above Returns a dictionary (Name) -> (x,y,z) of electrode name in the extended 10-20 system and its Cartesian coordinates in unit sphere.

```
PHYSIONET_ELECTRODES = {
    1:  "FC5",  2: "FC3",  3: "FC1",  4: "FCz",  5: "FC2",  6: "FC4",
    7:  "FC6",  8: "C5",   9: "C3",  10: "C1",  11: "Cz",  12: "C2",
   13: "C4",  14: "C6",  15: "CP5", 16: "CP3", 17: "CP1", 18: "CPz",
   19: "CP2", 20: "CP4", 21: "CP6", 22: "Fp1", 23: "Fpz", 24: "Fp2",
   25: "AF7", 26: "AF3", 27: "AFz", 28: "AF4", 29: "AF8", 30: "F7",
   31: "F5",  32: "F3",  33: "F1",  34: "Fz",  35: "F2",  36: "F4",
   37: "F6",  38: "F8",  39: "FT7", 40: "FT8", 41: "T7",  42: "T8",
   43: "T9",  44: "T10", 45: "TP7", 46: "TP8", 47: "P7",  48: "P5",
   49: "P3",  50: "P1",  51: "Pz",  52: "P2",  53: "P4",  54: "P6",
   55: "P8",  56: "PO7", 57: "PO3", 58: "POz", 59: "PO4", 60: "PO8",
   61: "O1",  62: "Oz",  63: "O2",  64: "Iz"}
```

(a) class 1 Accuracy



(b) class 1 F1 score

Figure 4.1: illustration of electrode locations

4.1.2 filters

Basic preprocessing is needed to be performed on the raw data before transformation In this case a band-pass filter is used to limit the EEG frequencies to the one of interest & in order to do that some specification are needed to be defined, such specifications are listed below.

- sampling frequency fs= 160.0 HZ
- low-cutoff frequency = 5.0 HZ
- high-cutoff frequency = 30.0 HZ
- Order = 5

Here is a code snippet of our used Filter

```
def butter_bandpass(lowcut, highcut, fs, order=5):
    nyq = 0.5 * fs
    low = lowcut / nyq
    high = highcut / nyq
    b, a = butter(order, [low, high], btype='band')
return b, a

def butter_bandpass_filter(data, lowcut, highcut, fs, order=5):
    b, a = butter_bandpass(lowcut, highcut, fs, order=order)
    y = filtfilt(b, a, data)
return y
```

we then get the 2D projection of the data from the EDF file to convert the extracted data into matrix form that can be further processed in a matlab environment.

Azimuthal equidistant projection (AEP)was used because the "AEP' of 3D carthesian coordinates Preserves distance to origin while projecting to 2D carthesian space.

loc: N x 3 array of 3D points returns: N x 2 array of projected 2D points

```
def projection_2d(loc):

x, y, z = loc[:, 0], loc[:, 1], loc[:, 2]
theta = np.arctan2(y, x)  # theta = azimuth
rho = np.pi / 2 - np.arctan2(z, np.hypot(x, y))  # rho = pi/2 - elevation
return np.stack((np.multiply(rho, np.cos(theta)), np.multiply(rho, np.sin(theta))
```

After That the data of the 20 subjects are save where each subject has 64 separate recording for each electrodein a .mat file type.

4.2 Creating & Training The Machine Learning Models

As Previously stated, we created multiple models in order to conclude optimal model in terms of our selected performance evaluation Metrics.

This section is to discuss each model's Architecture as well as it's specifications. here are some global specifications that are dictated by our used dataset.

- number of subject = 20
- number of trial = 84
- number of channels per subject = 64
- number of data samples = 640
- Time considered "time sampling" = 4 / 10
- Data points = Time consider * 160

4.3 Classification

The classification process is working as water fall model where the the raw EDF data is converted to Matlab files where the pre-proccessing takes place as it become ready in the .CSV files to be injected as the input of the machine learning models proposed previously.

Tensor flow one hot representation (encoding) is the result of models as it produces the different probabilities to each class.

	A	B	C	D
1	0,422148913145065	0,170379713177681	0,375659346580505	0,031812109053135
2	0,213474109768867	0,363638013601303	0,3406081199646	0,082279734313488
3	0,496592938899994	0,235594645142555	0,205106168985367	0,062706209719181
4	0,164040699601173	0,271115481853485	0,118775635957718	0,44606825709343
5	0,25213423371315	0,247895166277885	0,345590949058533	0,154379591345787

Figure 4.2: example illustrating under-fitting

As shown in the previous figure that the model predict different probabilities for each class then this output is translated by help of Tensor Flow one shot encoding technique into one class only as it select the class with the highest probability as shown in the figure below.

	A	B	C	D
1	1	0	0	0
2	0	1	0	0
3	1	0	0	0
4	0	0	0	1
5	0	0	1	0

Figure 4.3: One shot encoding

4.4 Arm Structure

InMoov bionic arm is structured same as the human hand to preform the same functionalities, as it consists of 6 servo motors acting as the muscles of the hands with small rings where the cords that acts as the tendon of the hand is attached when the servo moves, this movement is translated into a Contraction or a relaxation of the cord which is responsible for the movement of the fingers as each servo is responsible for the movement of only one finger, while the sixth servo is responsible for the wrist movement.

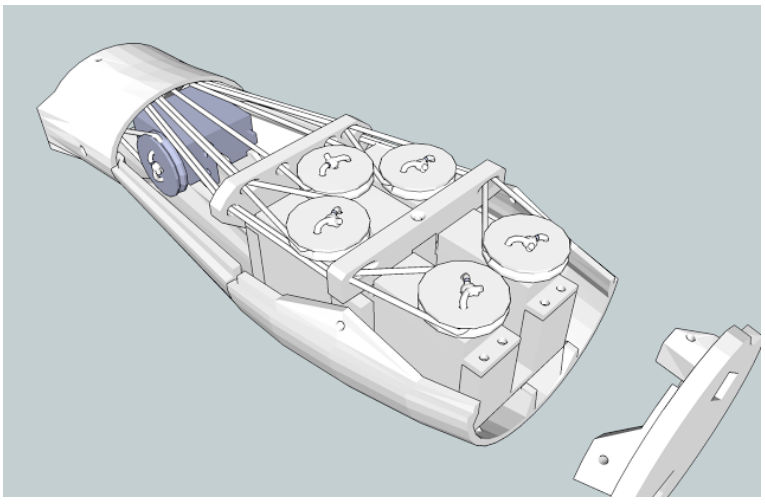


Figure 4.4: InMoov hand structure

while the movement of the wrist has different technique as it depends on a two gears , small gear is attached to a servo while other bigger gear is attached to the wrist which is responsible to change the direction of movement to preform the same as human wrist movement.

4.5 Mechanical Functionality

After the classification of the signals ,each class is then translated into its respective mechanical motion to be performed attached robotic arm. The current objective for the bionic arm's range of motion is creating 4 basic movements:

- Hand opening "Class 1"
- Hand clutching "Class 2"
- Anticlockwise wrist movement "Class 3"
- Clockwise wrist movement "Class 4"

4.5.1 First Classification —hand opening

This Movement is comprised of the relaxation of all five fingers of the human hand at the same time, which is to be Translated into the resetting of all five servo motors to the rest position angle of "0 degrees"

4.5.2 Second Classification —Hand Clutching

Opposite the it's predecessor This Movement is comprised of the Contraction of all five fingers of the human hand at the same time, which is to be Translated into the setting of all five servo motors to an angle of approximately "90 degrees"

4.5.3 Third Classification —Anticlockwise Wrist Movement

This motion is only dependant on one servo motor located in the wrist Compartment of the Bionic Arm. Since the human arm can only rotate inward and is incapable of rotating outwards ,our default position is set to "90 degrees" while maximum rotation is set to "180 degrees".

4.5.4 Fourth Classification —clockwise Wrist Movement

This motion is also only dependant on the one servo motor located in the wrist compartment of the Bionic Arm where it is the reversing of the Anticlockwise motion where the motion is mechanically defined by resetting this particular servo motor's angle to a default of "90 degrees"

Chapter 5

Project Plan

5.1 Project schedule

Task	Time frame	Status
Creating Project Requirements	2 Week	Done
Getting schematics of the 3D model for the bionic Arm	2 Week	Done
Assembling The Arm	2 Weeks	Done
Interfacing with the arm	1 Week	Done
Interfacing with the Brain	2 Weeks	Done
Building/Acquisition of the Dataset	4 Week	Done
Data Preprocessing	3 Weeks	Done
Creating the Machine Learning Model	4 Weeks	Done
Training the Model	2 Week	Done
Coding the Arm's driver with microprocessor	2 Weeks	Done
Extracting Results and Conclusions	2 Week	Done

Chapter 6

Models Results

Contents

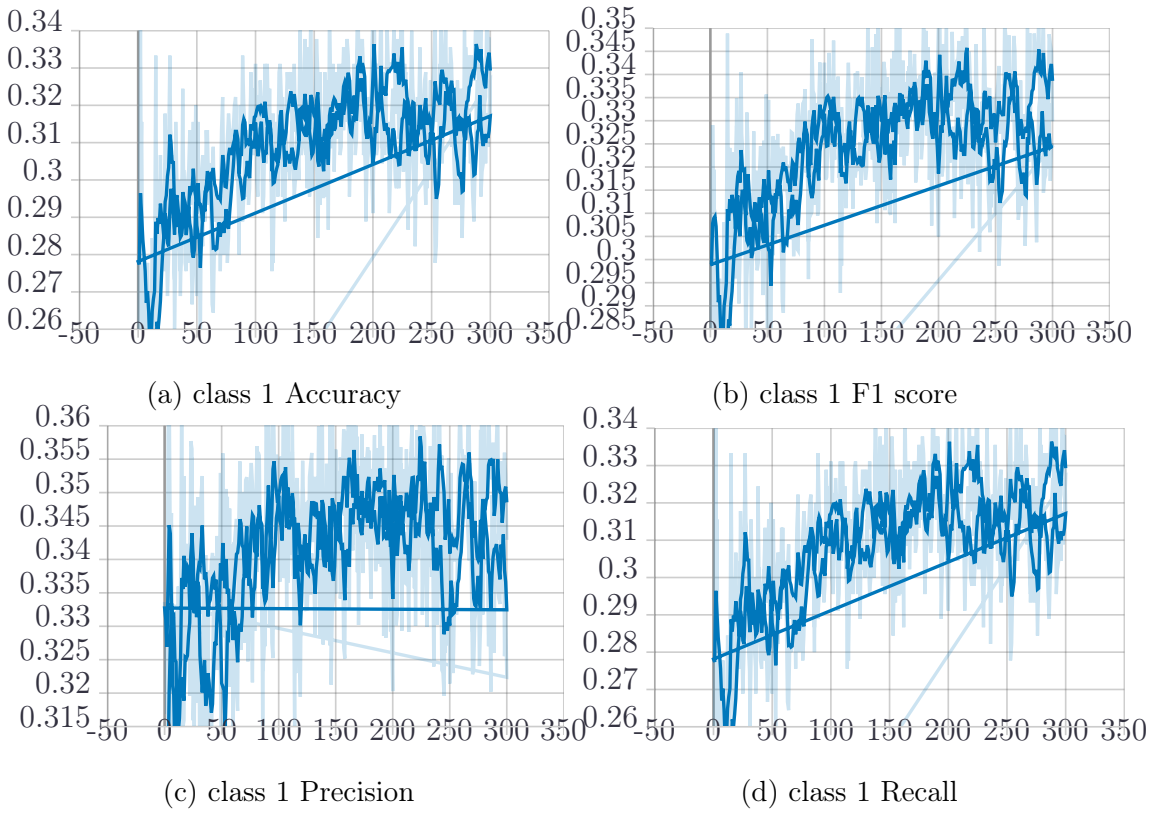
This section shows the classfication of the four motor imaginary functions.

All the results that will be shown in this chapter are performed only on 20 subjects from the dataset due to system specification that don't have the sufficient abilities to preform the training process on the whole dataset (109 subject).

6.1 Deep Neural Network —DNN

This section include the detailed results of the Deep Neural Network DNN which include the evaluation metrics for the four classes. Accuracy,F1 Score,Precision & Recall will be used to evaluate each class separately . The same four Metrics mentioned above as well as the Confusion Matrix & The receiver Operating Characteristics Curve"ROC Curve".

6.1.1 Results of the first class



(a) class 1 Accuracy

(b) class 1 F1 score

(c) class 1 Precision

(d) class 1 Recall

Figure 6.1: DNN first class results

Performance Metrics for this class are calculated to be as follows:

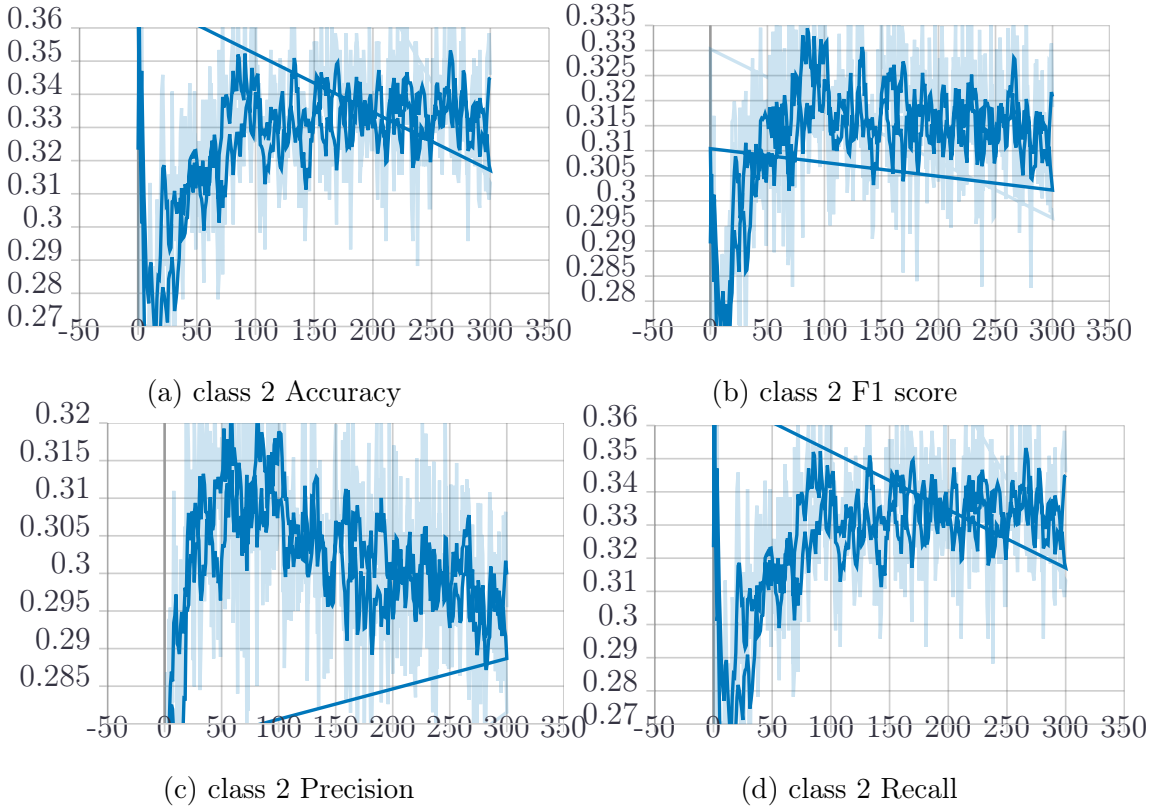
Accuracy performance of this class is calculated to be 33.5 %

Precision is calculated to be 35.5 %

The ability of the model to find relevant cases “Recall” is calculated to be 33.5%.

F1 score is calculated to be 34.5%.

6.1.2 Results of the second class



(a) class 2 Accuracy

(b) class 2 F1 score

(c) class 2 Precision

(d) class 2 Recall

Figure 6.2: DNN second class results

Performance Metrics for this class are calculated to be as follows:

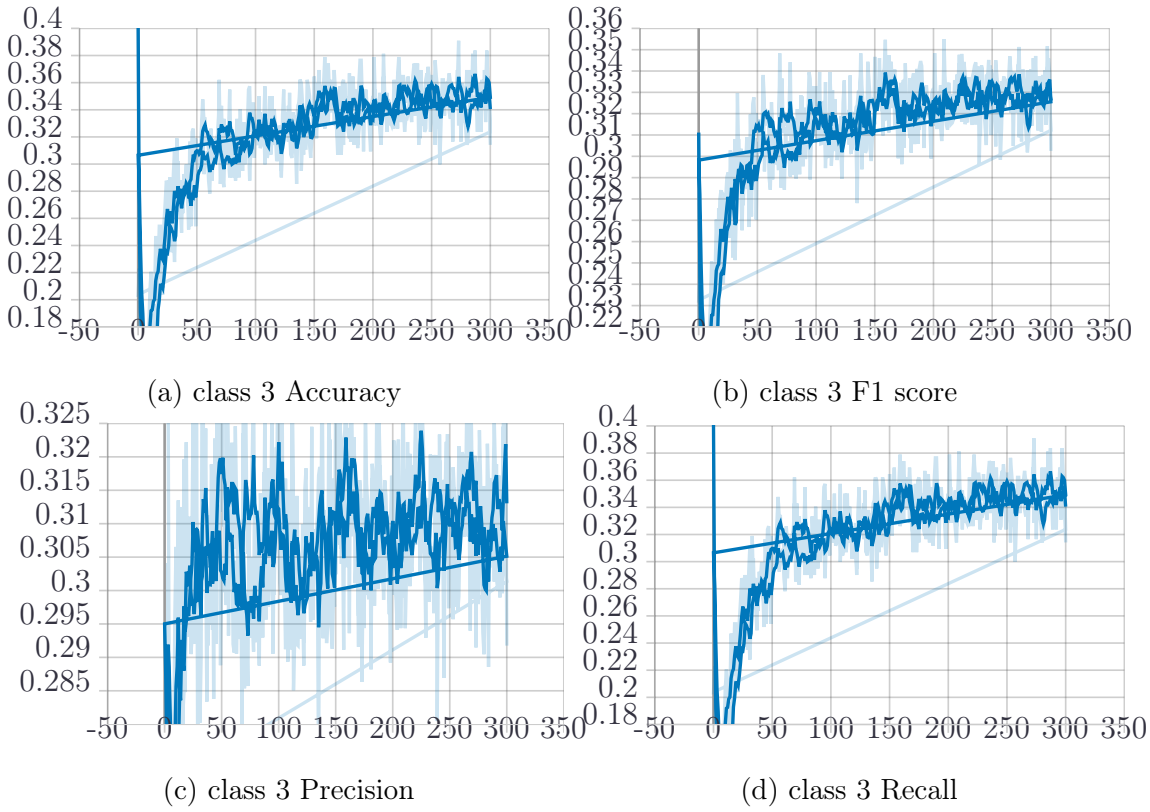
Accuracy performance of this class is calculated to be 35 %

Precision is calculated to be 32 %

The ability of the model to find relevant cases "Recall" is calculated to be 35%.

F1 score is calculated to be 33%.

6.1.3 Results of the third class



(a) class 3 Accuracy

(b) class 3 F1 score

(c) class 3 Precision

(d) class 3 Recall

Figure 6.3: DNN third class results

Performance Metrics for this class are calculated to be as follows:

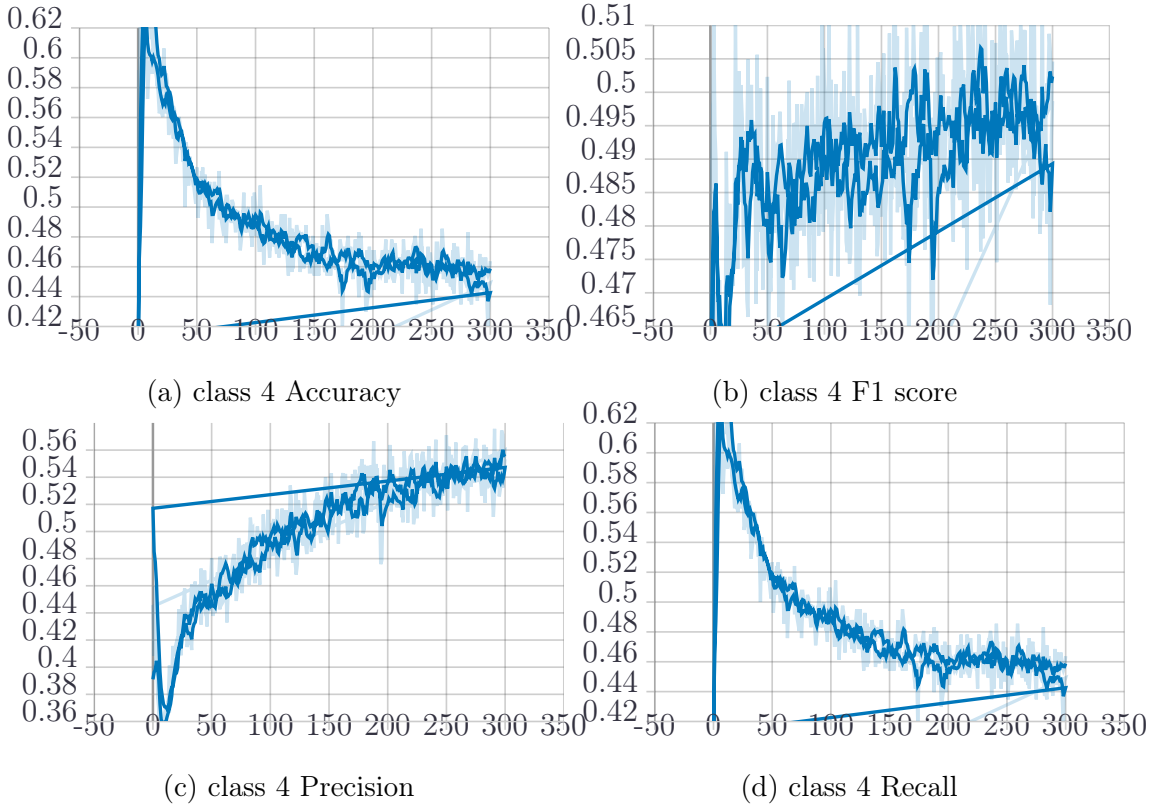
Accuracy performance of this class is calculated to be 36 %

Precision is calculated to be 32.5 %

The ability of the model to find relevant cases “Recall” is calculated to be 34%.

F1 score is calculated to be 34%.

6.1.4 Results of the fourth class



(a) class 4 Accuracy

(b) class 4 F1 score

(c) class 4 Precision

(d) class 4 Recall

Figure 6.4: DNN fourth class results

Performance Metrics for this class are calculated to be as follows:

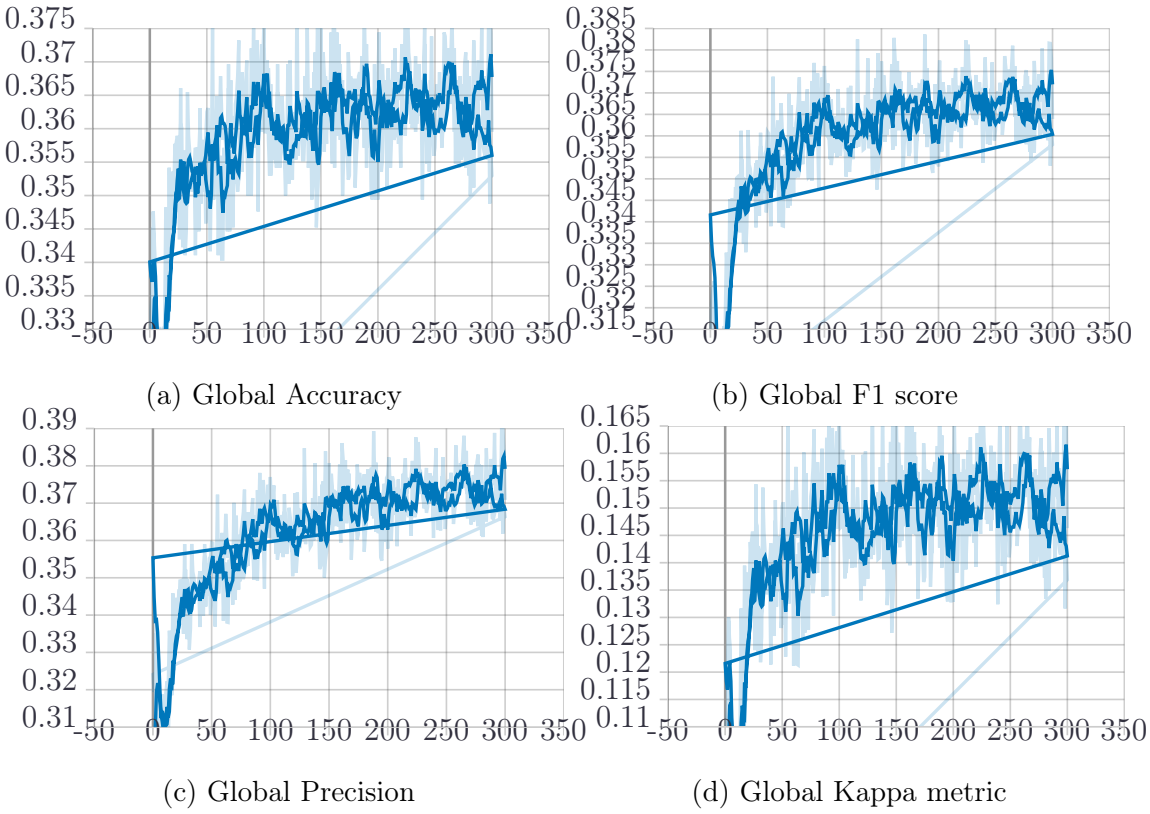
Accuracy performance of this class is calculated to be 60 %

Precision is calculated to be 52 %

The ability of the model to find relevant cases “Recall” is calculated to be 60%.

F1 score is calculated to be 50.5%.

6.1.5 Global results



(a) Global Accuracy

(b) Global F1 score

(c) Global Precision

(d) Global Kappa metric

Figure 6.5: DNN global results

Global Performance Metrics of this Model using a dataset over 20 subjects are calculated to be as follows:

Accuracy performance of this class is calculated to be 37 %

Precision is calculated to be 28 %

The ability of the model to find relevant cases “Kappa Metric” is calculated to be 16%.

F1 score is calculated to be 37%.

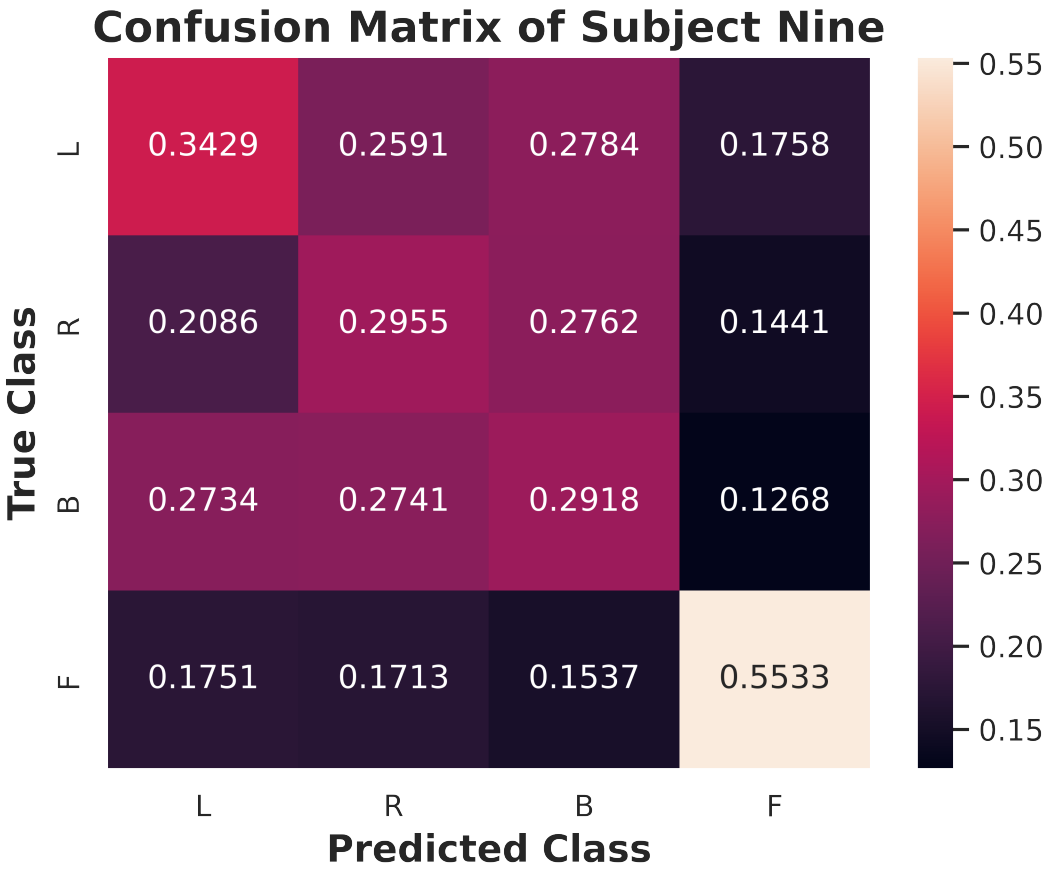


Figure 6.6: DNN confusion matrix

This confusion matrix shows the percentage of the correct classification for each class.

The fist class is classified correctly by 34.2 %

The second class is classified correctly by 29.5 %

The third class is classified correctly by 29.1 %

The fourth class is classified correctly by 55.3 %

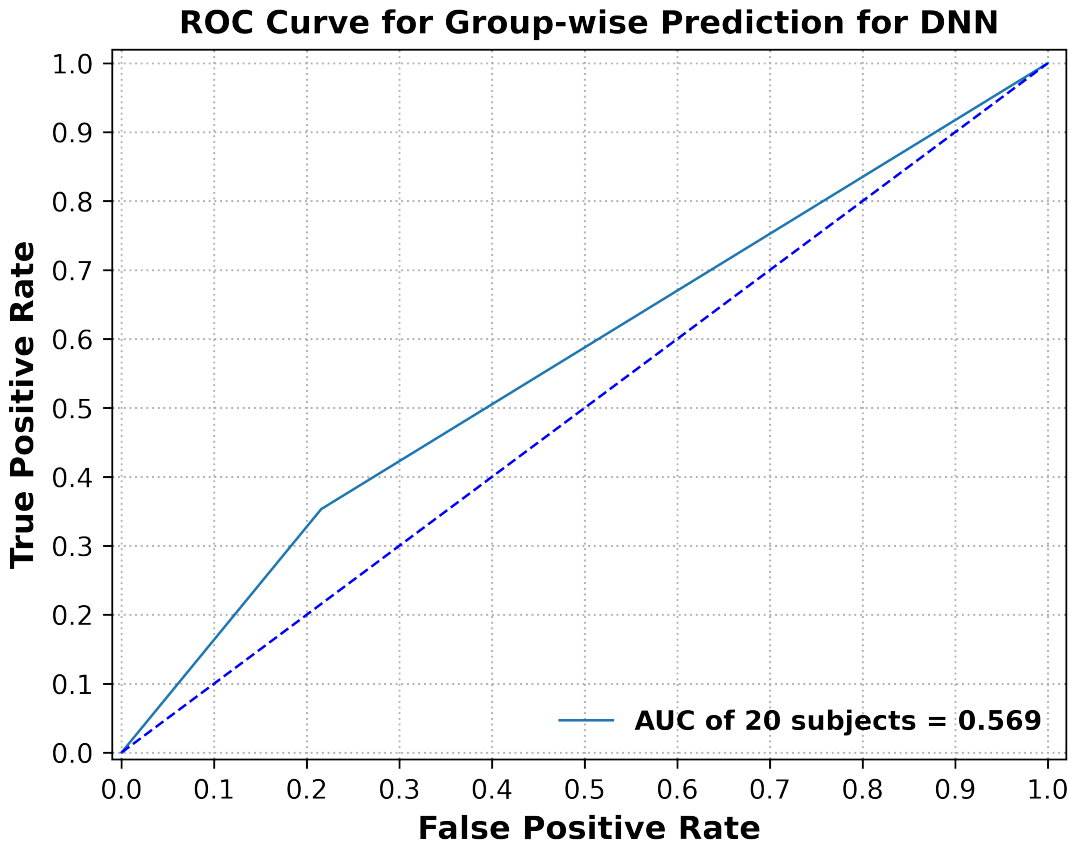


Figure 6.7: DNN ROC curve

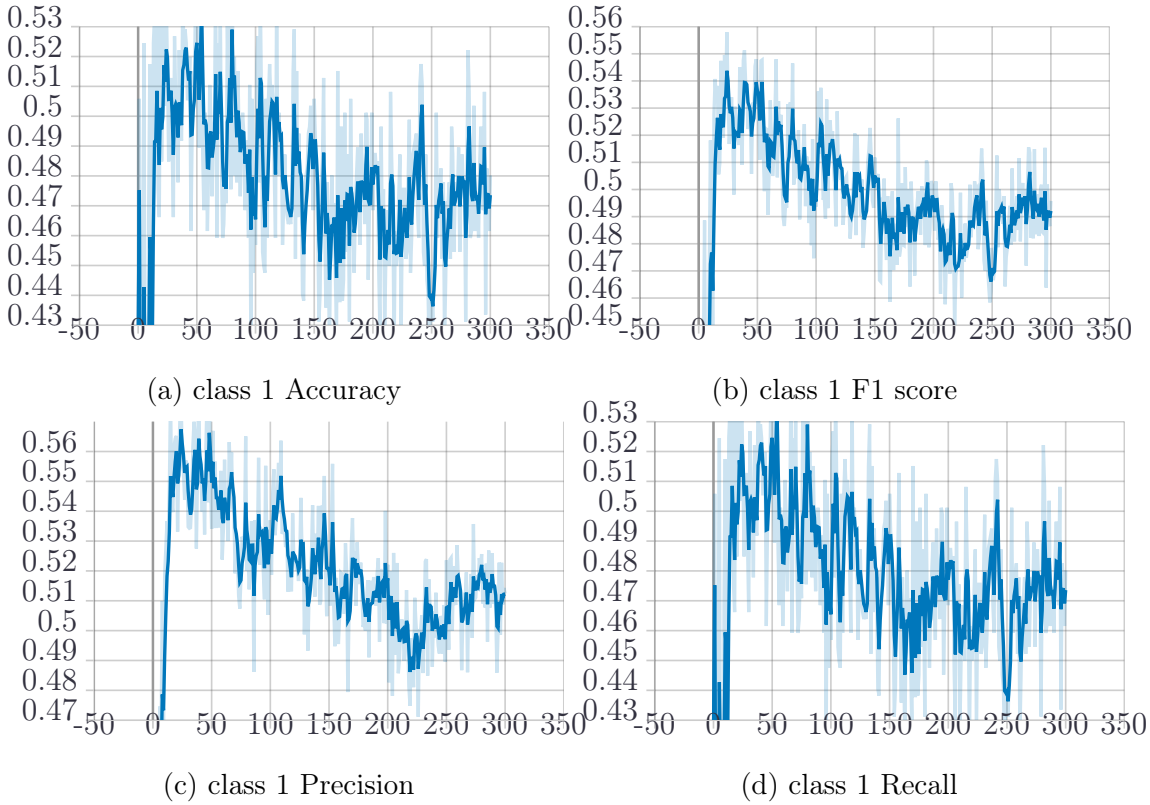
The ROC curve shows the trade-off between sensitivity (or TPR) and specificity (1 – FPR). Classifiers that give curves closer to the top-left corner indicate a better performance. As a baseline, a random classifier is expected to give points lying along the diagonal (FPR = TPR). The closer the curve comes to the 45-degree diagonal of the ROC space, the less accurate the test.

The area under curve (AUC) equals 0.569 unit of area

6.2 Convolutional Neural Network —CNN

This section include the detailed results of the Convolutional Neural Network CNN which include the evaluation metrics for the four classes. Accuracy,F1 Score,Precision & Recall will be used to evaluate each class separately . The same four Metrics mentioned above as well as the Confusion Matrix & The receiver Operating Characteristics Curve"ROC Curve".

6.2.1 Results of the first class



(a) class 1 Accuracy

(b) class 1 F1 score

(c) class 1 Precision

(d) class 1 Recall

Figure 6.8: CNN first class results

Performance Metrics for this class are calculated to be as follows:

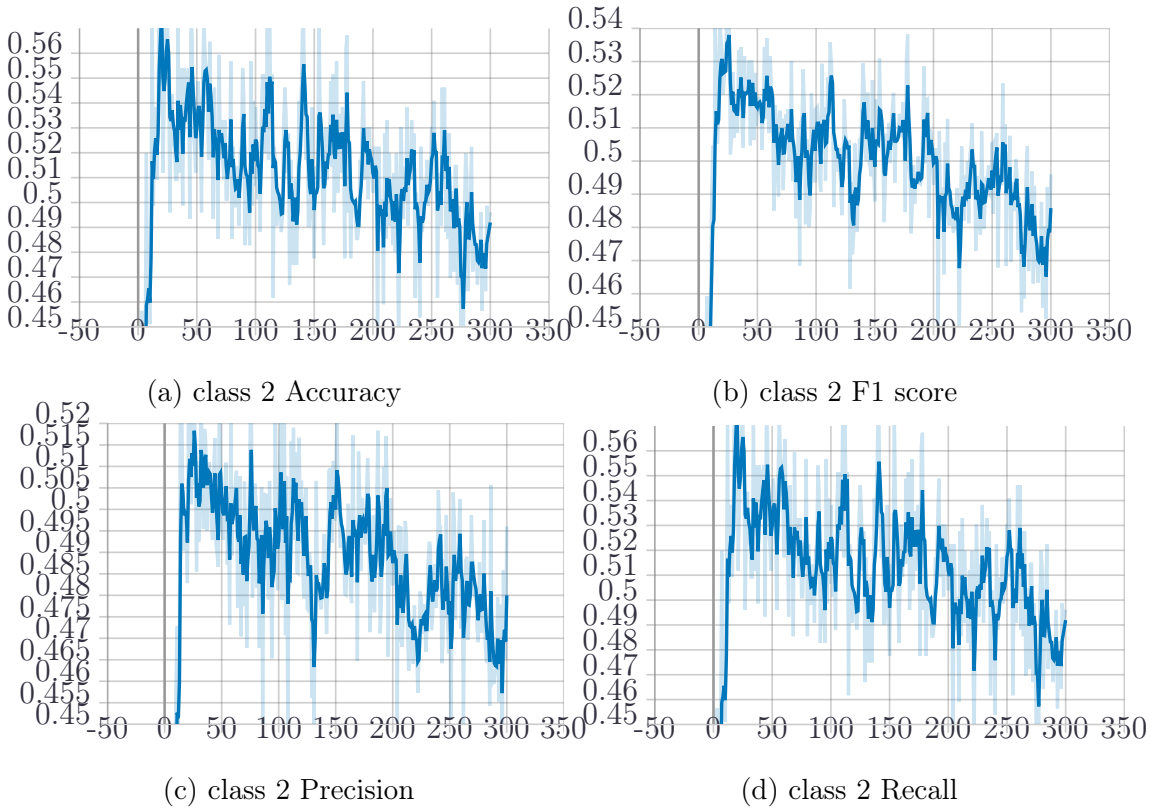
Accuracy performance of this class is calculated to be 53%

Precision is calculated to be 56 %

The ability of the model to find relevant cases “Recall” is calculated to be 53%.

F1 score is calculated to be 55%.

6.2.2 Results of the second class



(a) class 2 Accuracy

(b) class 2 F1 score

(c) class 2 Precision

(d) class 2 Recall

Figure 6.9: CNN second class results

Performance Metrics for this class are calculated to be as follows:

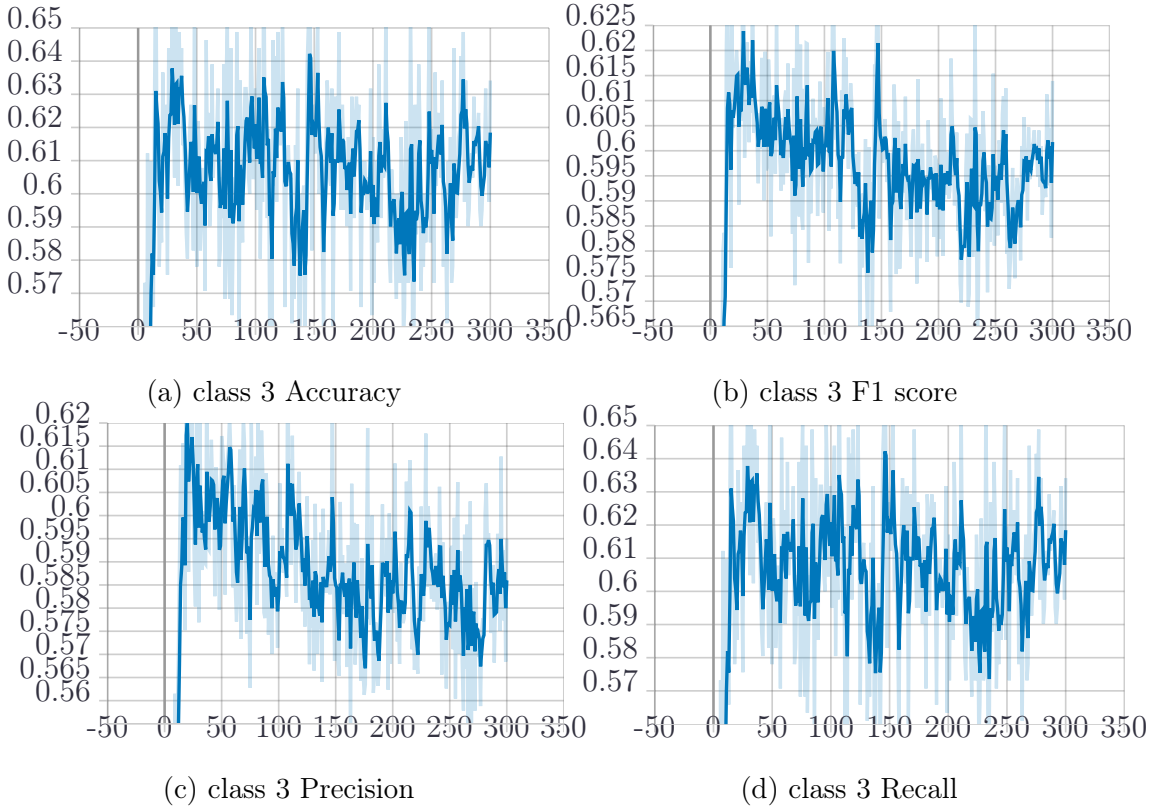
Accuracy performance of this class is calculated to be 56%

Precision is calculated to be 52 %

The ability of the model to find relevant cases “Recall” is calculated to be 56%.

F1 score is calculated to be 54%.

6.2.3 Results of the third class



(a) class 3 Accuracy

(b) class 3 F1 score

(c) class 3 Precision

(d) class 3 Recall

Figure 6.10: CNN third class results

Performance Metrics for this class are calculated to be as follows:

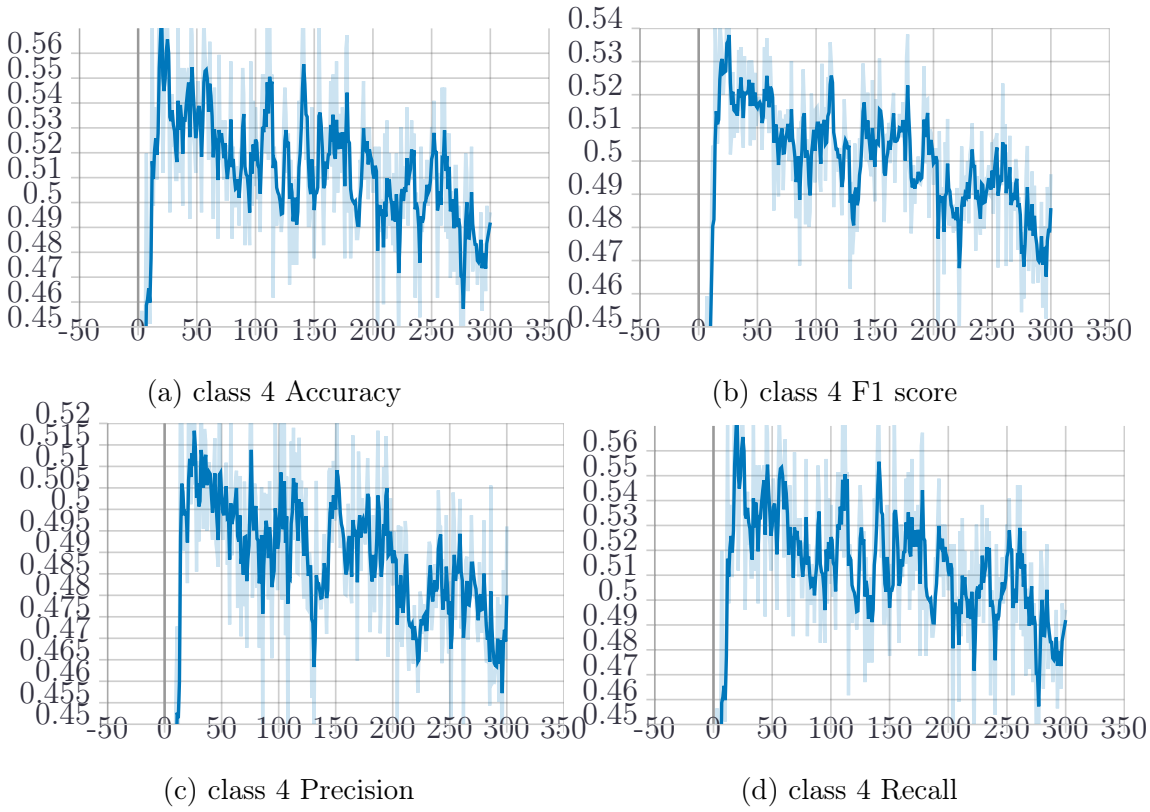
Accuracy performance of this class is calculated to be 64%

Precision is calculated to be 61 %

The ability of the model to find relevant cases "Recall" is calculated to be 64%.

F1 score is calculated to be 62.5%.

6.2.4 Results of the fourth class



(a) class 4 Accuracy

(b) class 4 F1 score

(c) class 4 Precision

(d) class 4 Recall

Figure 6.11: CNN fourth class results

Performance Metrics for this class are calculated to be as follows:

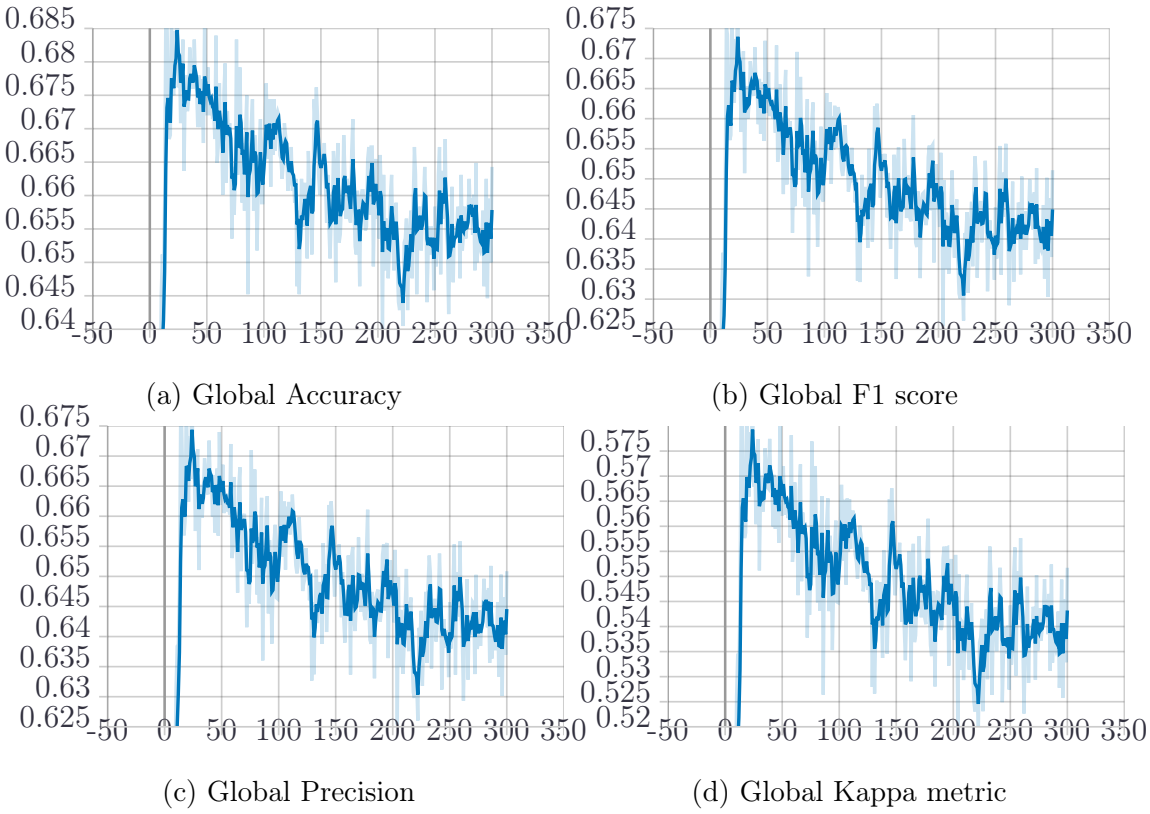
Accuracy performance of this class is calculated to be 56%

Precision is calculated to be 53 %

The ability of the model to find relevant cases “Recall” is calculated to be 56%.

F1 score is calculated to be 54%.

6.2.5 Global results



(a) Global Accuracy

(b) Global F1 score

(c) Global Precision

(d) Global Kappa metric

Figure 6.12: CNN global results

Global Performance Metrics of this Model using a dataset over 20 subjects are calculated to be as follows:

Accuracy performance of this class is calculated to be 68.5 %

Precision is calculated to be 67.5 %.

The ability of the model to find relevant cases “Kappa Metric” is calculated to be 57.5 %.

F1 score is calculated to be 67.5 %.

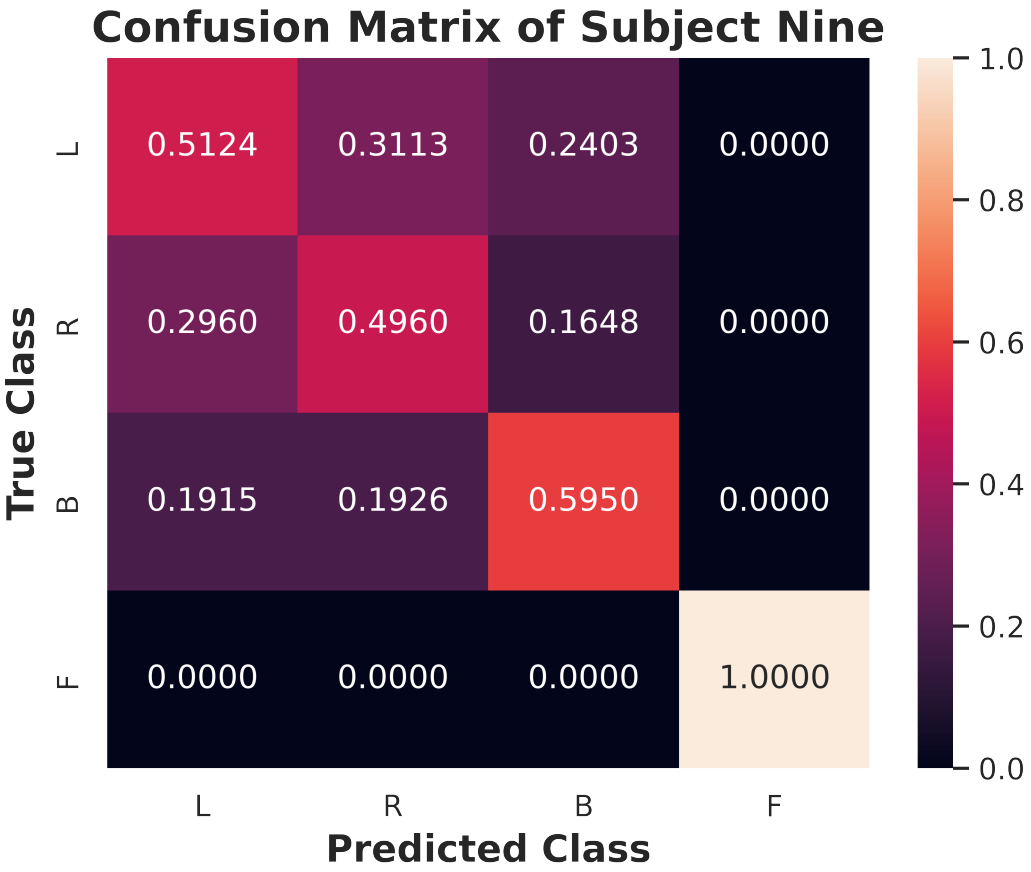


Figure 6.13: CNN confusion matrix

This confusion matrix shows the percentage of the correct classification for each class.

The fist class is classified correctly by 51.2 %

The second class is classified correctly by 49.6 %

The third class is classified correctly by 59.5 %

The fourth class is classified correctly by 100 %

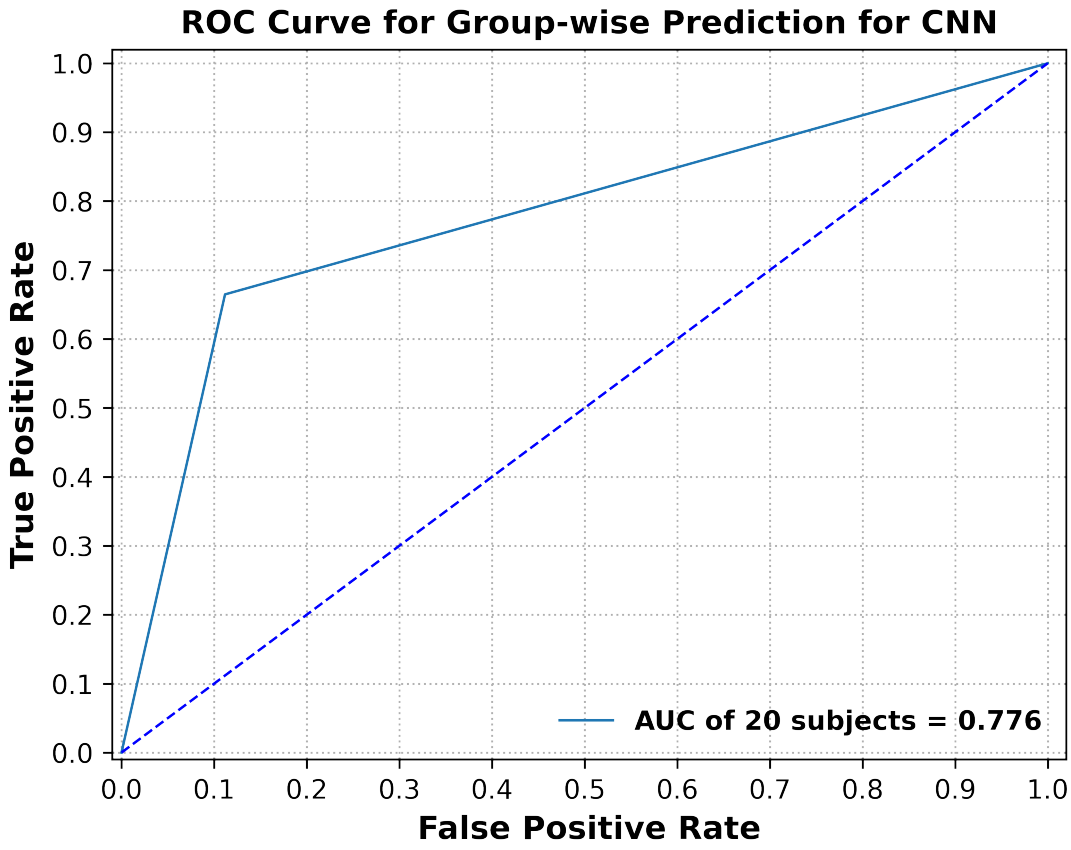


Figure 6.14: CNN ROC curve

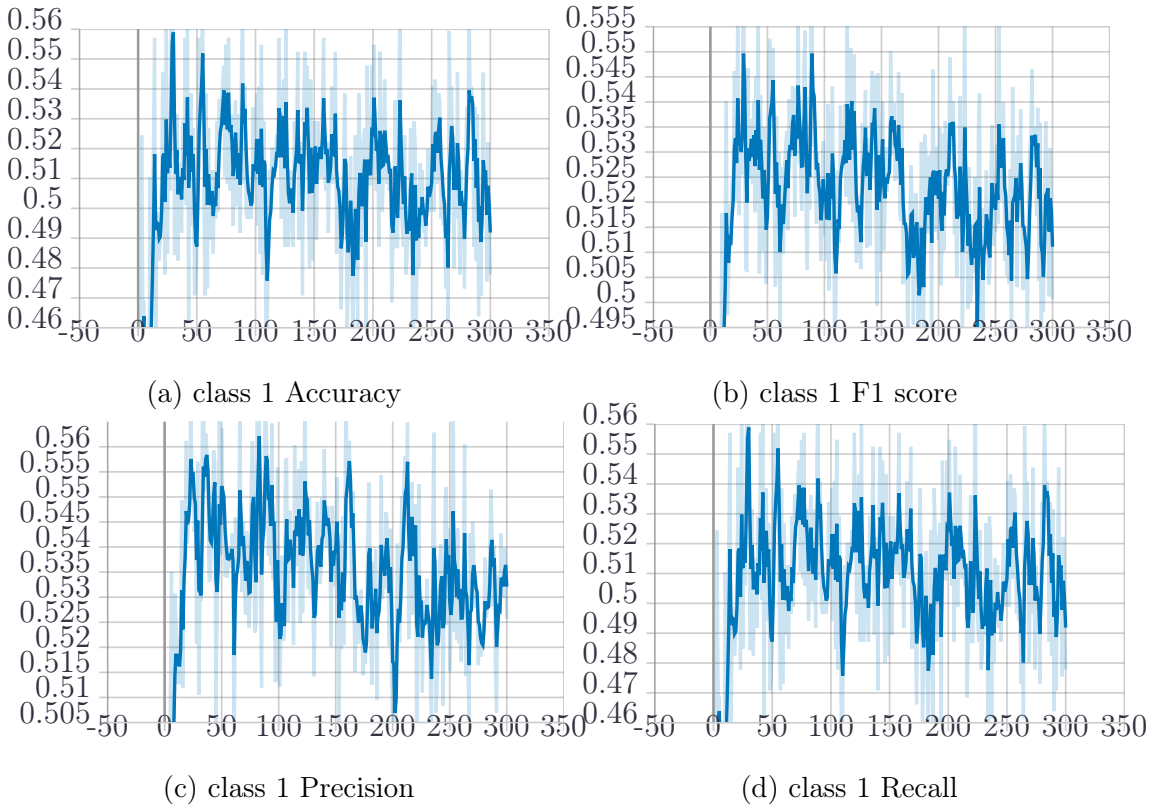
The ROC curve shows the trade-off between sensitivity (or TPR) and specificity (1 – FPR). Classifiers that give curves closer to the top-left corner indicate a better performance. As a baseline, a random classifier is expected to give points lying along the diagonal (FPR = TPR). The closer the curve comes to the 45-degree diagonal of the ROC space, the less accurate the test.

The area under curve (AUC) equals 0.776 unit of area.

6.3 Fully Convolutional Neural Network—FCN

This section include the detailed results of the Fully Convolutional Neural Network FCN which include the evaluation metrics for the four classes. Accuracy,F1 Score,Precision & Recall will be used to evaluate each class separately . The same four Metrics mentioned above as well as the Confusion Matrix & The receiver Operating Characteristics Curve"ROC Curve".

6.3.1 Results of the first class



(a) class 1 Accuracy

(b) class 1 F1 score

(c) class 1 Precision

(d) class 1 Recall

Figure 6.15: FCN first class results

Performance Metrics for this class are calculated to be as follows:

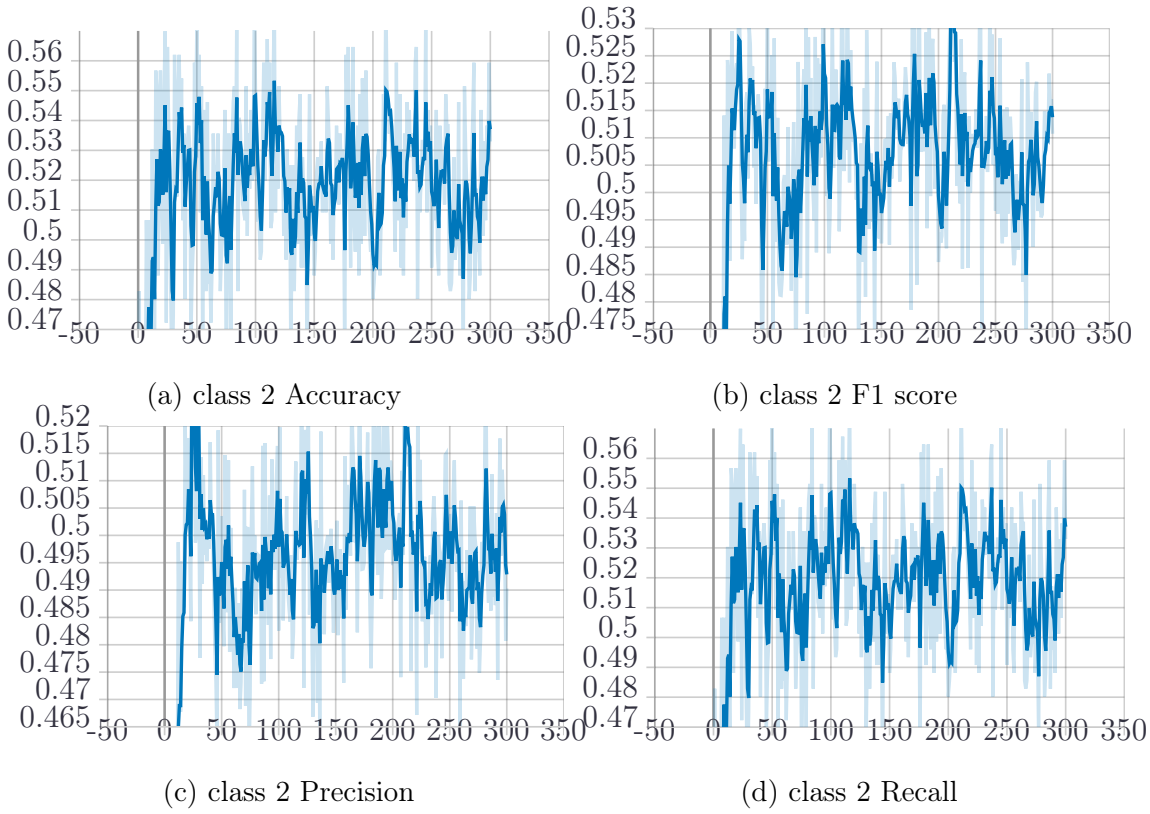
Accuracy performance of this class is calculated to be 56%

Precision is calculated to be 55 %

The ability of the model to find relevant cases “Recall” is calculated to be 56%.

F1 score is calculated to be 56%.

6.3.2 Results of the second class



(a) class 2 Accuracy

(b) class 2 F1 score

(c) class 2 Precision

(d) class 2 Recall

Figure 6.16: FCN second class results

Performance Metrics for this class are calculated to be as follows:

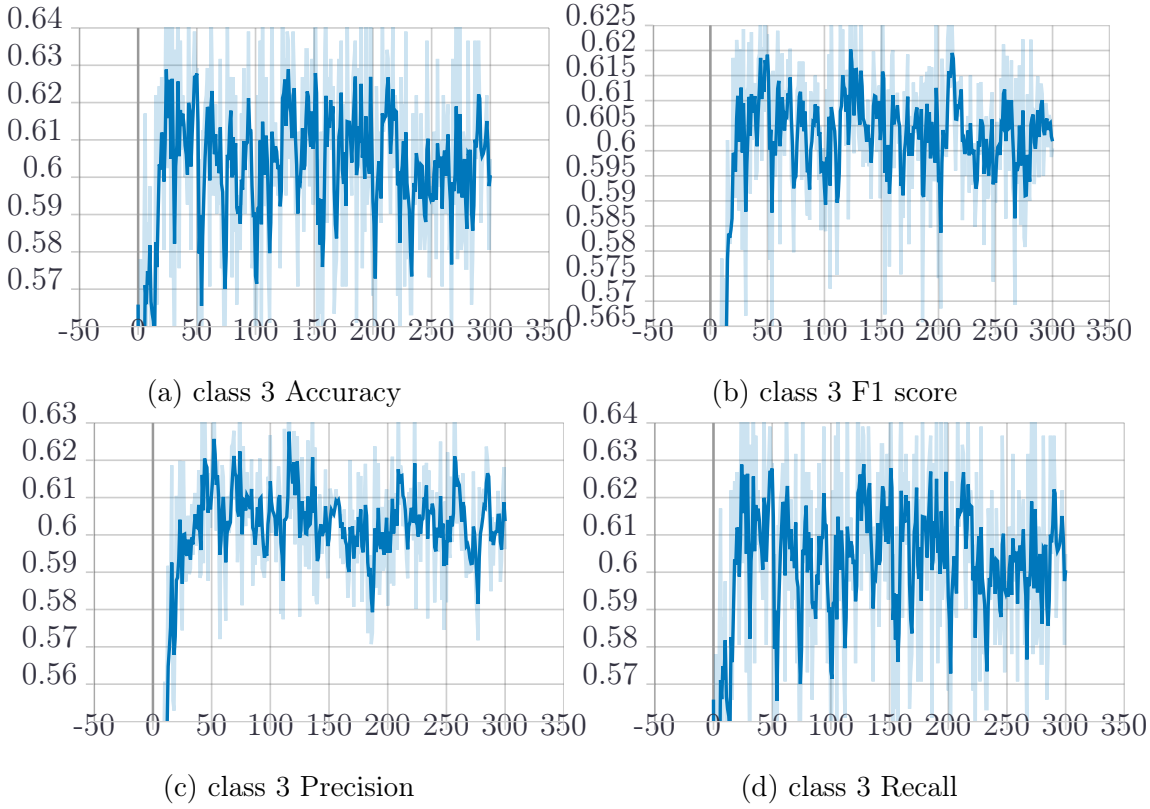
Accuracy performance of this class is calculated to be 55%

Precision is calculated to be 53 %

The ability of the model to find relevant cases "Recall" is calculated to be 56%.

F1 score is calculated to be 53%.

6.3.3 Results of the third class



(a) class 3 Accuracy

(b) class 3 F1 score

(c) class 3 Precision

(d) class 3 Recall

Figure 6.17: FCN third class results

Performance Metrics for this class are calculated to be as follows:

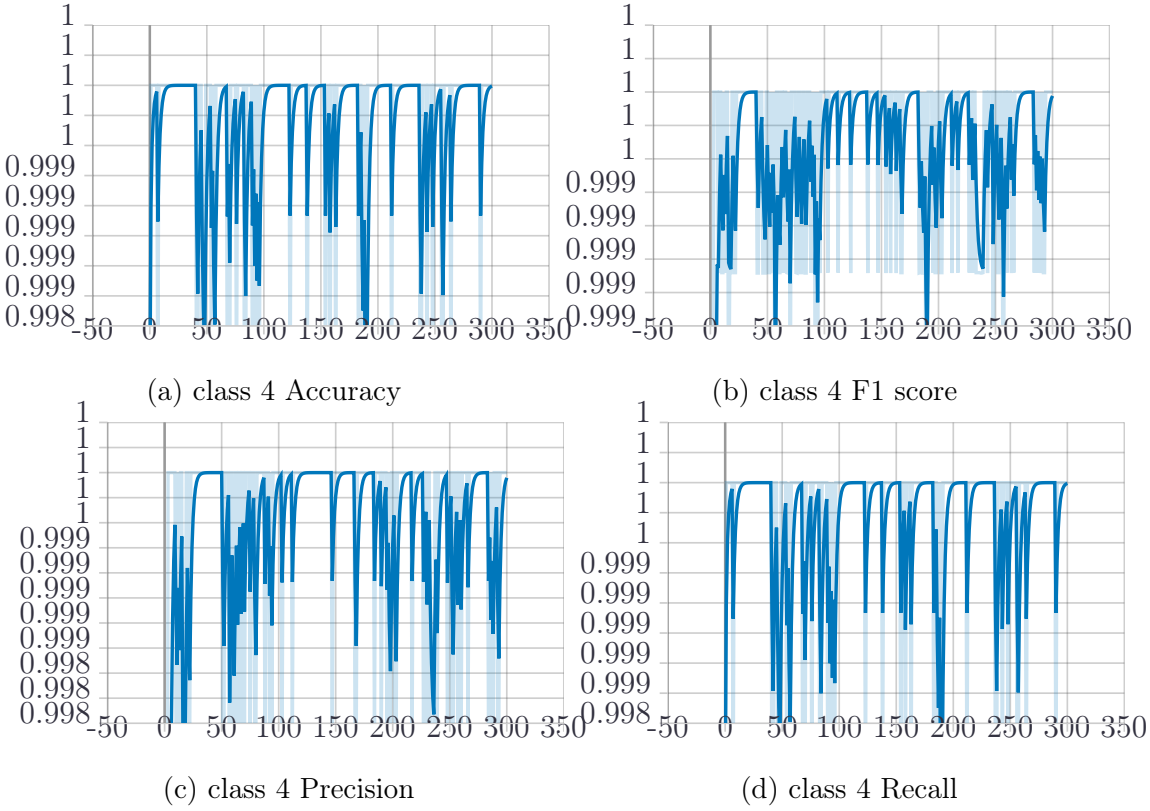
Accuracy performance of this class is calculated to be 63%

Precision is calculated to be 63 %

The ability of the model to find relevant cases “Recall” is calculated to be 62%.

F1 score is calculated to be 62.5%.

6.3.4 Results of the fourth class



(a) class 4 Accuracy

(b) class 4 F1 score

(c) class 4 Precision

(d) class 4 Recall

Figure 6.18: FCN fourth class results

Performance Metrics for this class are calculated to be as follows:

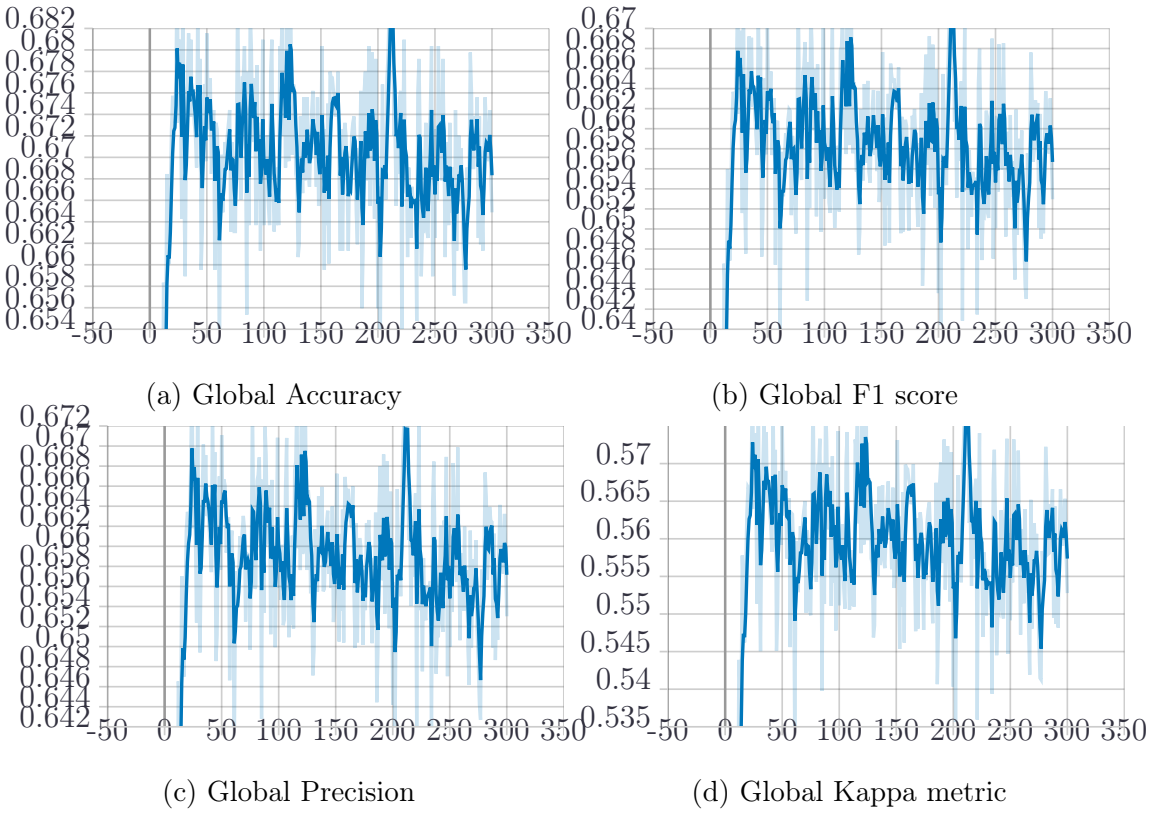
Accuracy performance of this class is calculated to be 100%

Precision is calculated to be 100 %

The ability of the model to find relevant cases “Recall” is calculated to be 100%.

F1 score is calculated to be 100%.

6.3.5 Global results



(a) Global Accuracy

(b) Global F1 score

(c) Global Precision

(d) Global Kappa metric

Figure 6.19: FCN global results

Global Performance Metrics of this Model using a dataset over 20 subjects are calculated to be as follows:

Accuracy performance of this class is calculated to be 68.5 %

Precision is calculated to be 67.2 %.

The ability of the model to find relevant cases “Kappa Metric” is calculated to be 57.5 %.

F1 score is calculated to be 67.5 %.

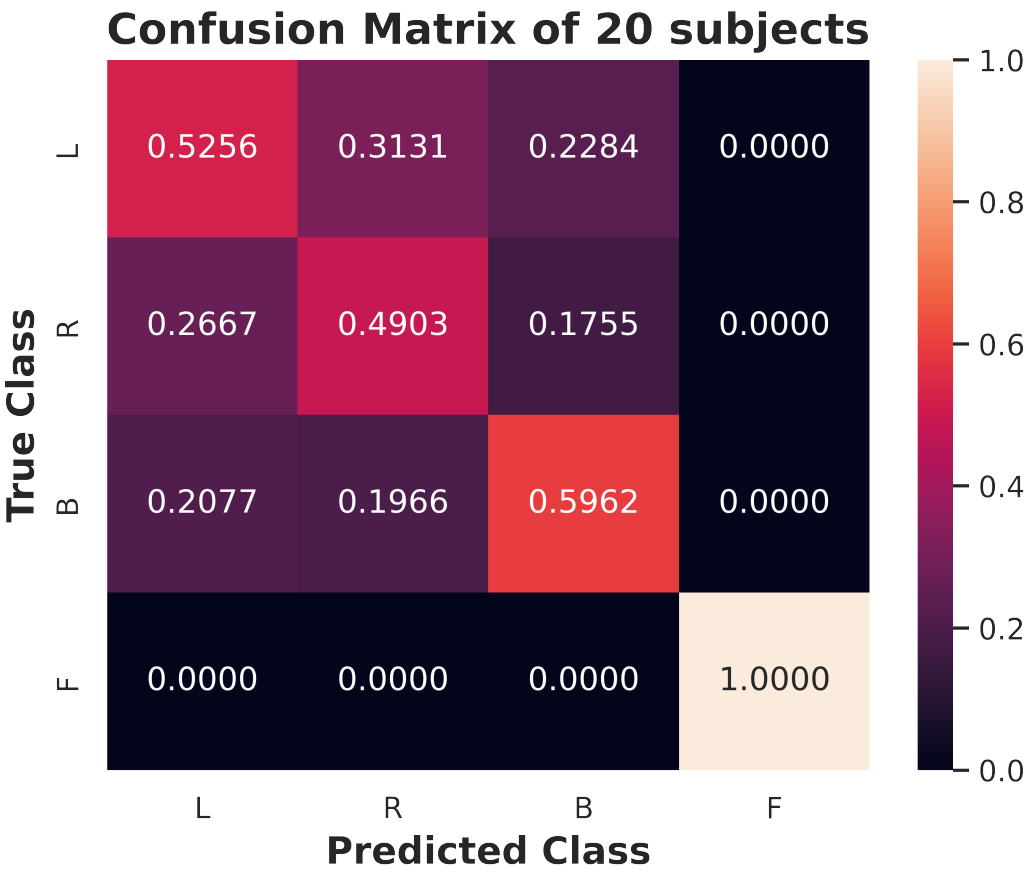


Figure 6.20: FCN confusion matrix

This confusion matrix shows the percentage of the correct classification for each class.

The fist class is classified correctly by 52.5 %

The second class is classified correctly by 49.3 %

The third class is classified correctly by 59.6 %

The fourth class is classified correctly by 100 %

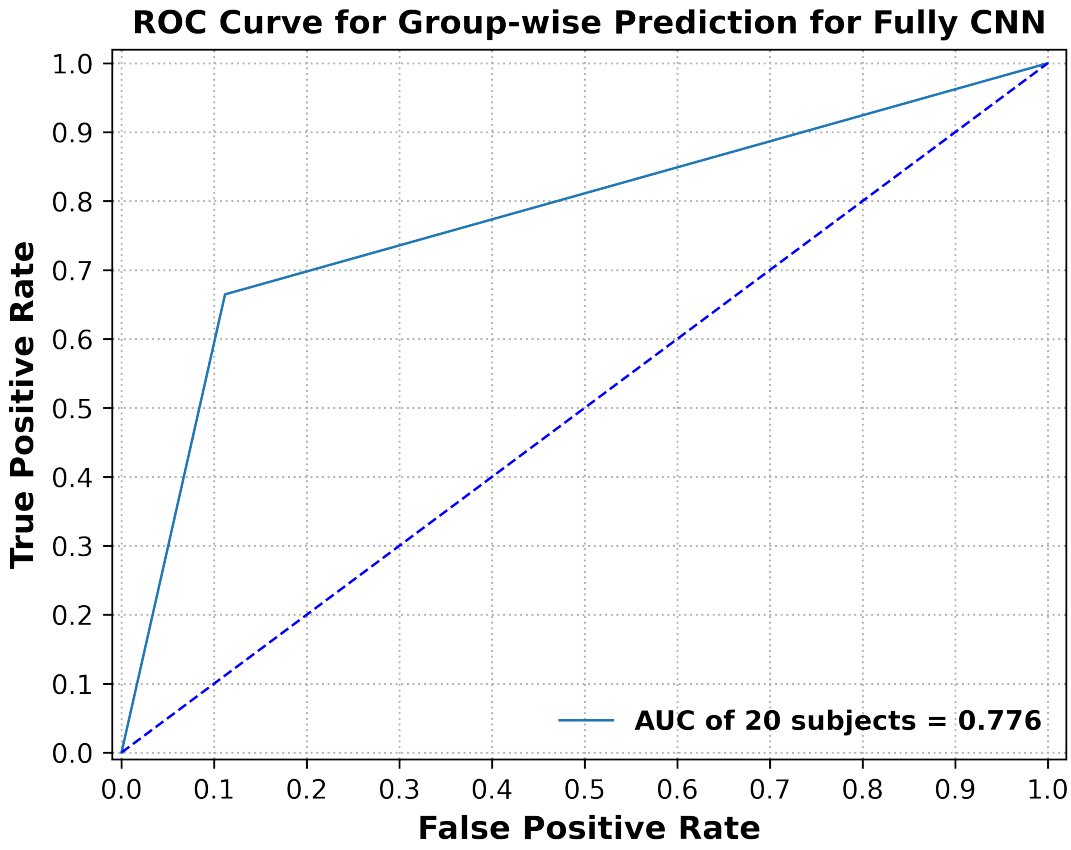


Figure 6.21: FCN ROC curve

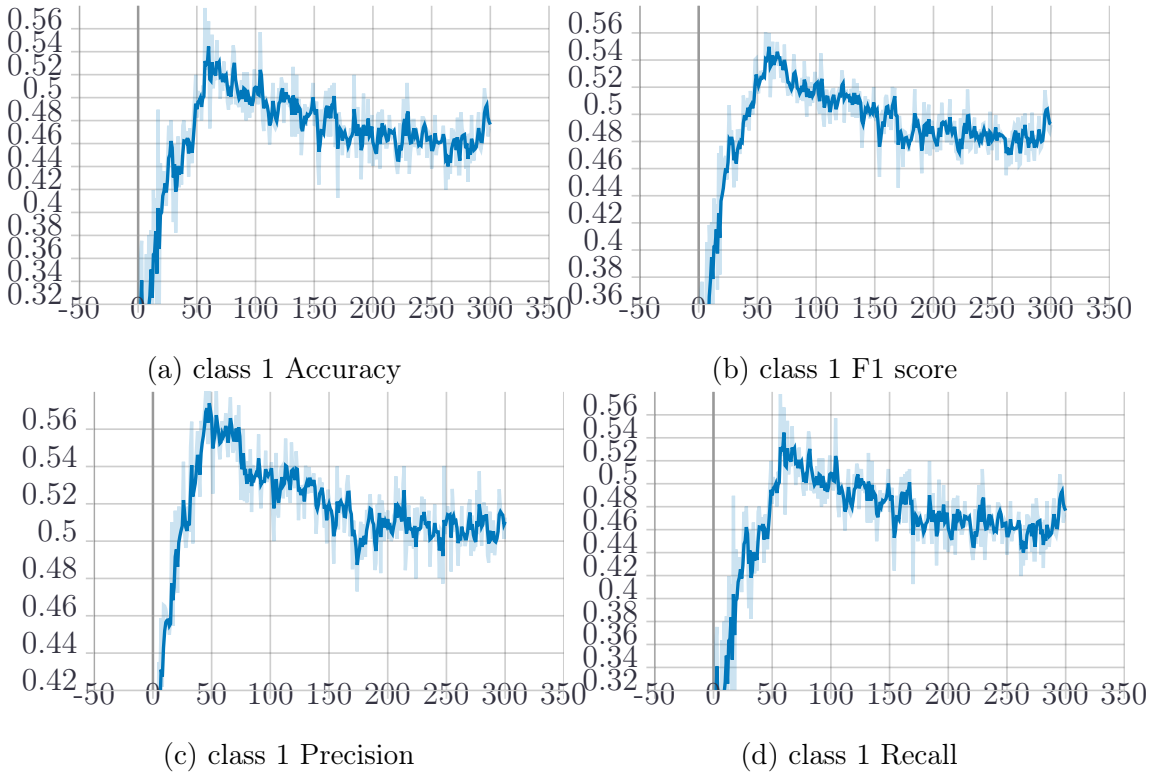
The ROC curve shows the trade-off between sensitivity (or TPR) and specificity (1 – FPR). Classifiers that give curves closer to the top-left corner indicate a better performance. As a baseline, a random classifier is expected to give points lying along the diagonal (FPR = TPR). The closer the curve comes to the 45-degree diagonal of the ROC space, the less accurate the test.

The area under curve (AUC) equals 0.776 unit of area.

6.4 Residual Neural Network—ResNet

This section include the detailed results of the Residual Neural Network ResNet which include the evaluation metrics for the four classes. Accuracy,F1 Score,Precision & Recall will be used to evaluate each class separately . The same four Metrics mentioned above as well as the Confusion Matrix & The receiver Operating Characteristics Curve"ROC Curve".

6.4.1 Results of the first class



(a) class 1 Accuracy

(b) class 1 F1 score

(c) class 1 Precision

(d) class 1 Recall

Figure 6.22: ResNet first class results

Performance Metrics for this class are calculated to be as follows:

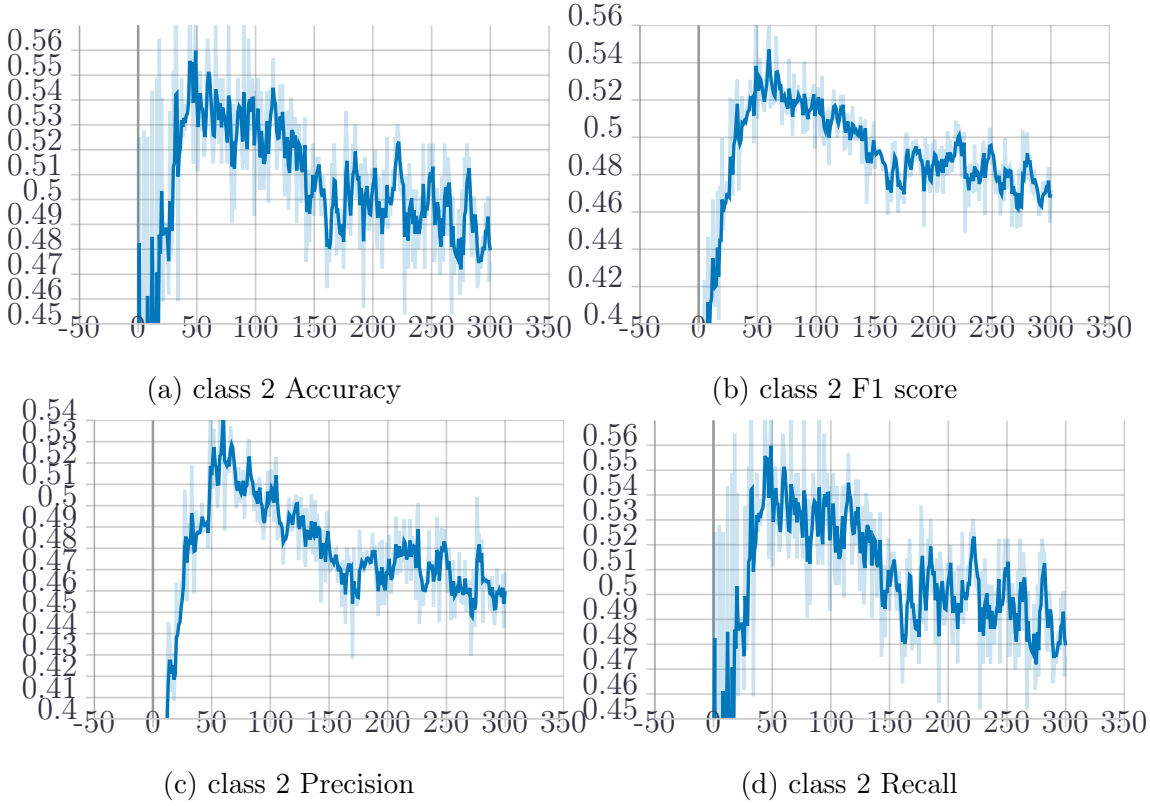
Accuracy performance of this class is calculated to be 54%

Precision is calculated to be 56 %

The ability of the model to find relevant cases “Recall” is calculated to be 54%.

F1 score is calculated to be 54%.

6.4.2 Results of the second class



(a) class 2 Accuracy

(b) class 2 F1 score

(c) class 2 Precision

(d) class 2 Recall

Figure 6.23: ResNet second class results

Performance Metrics for this class are calculated to be as follows:

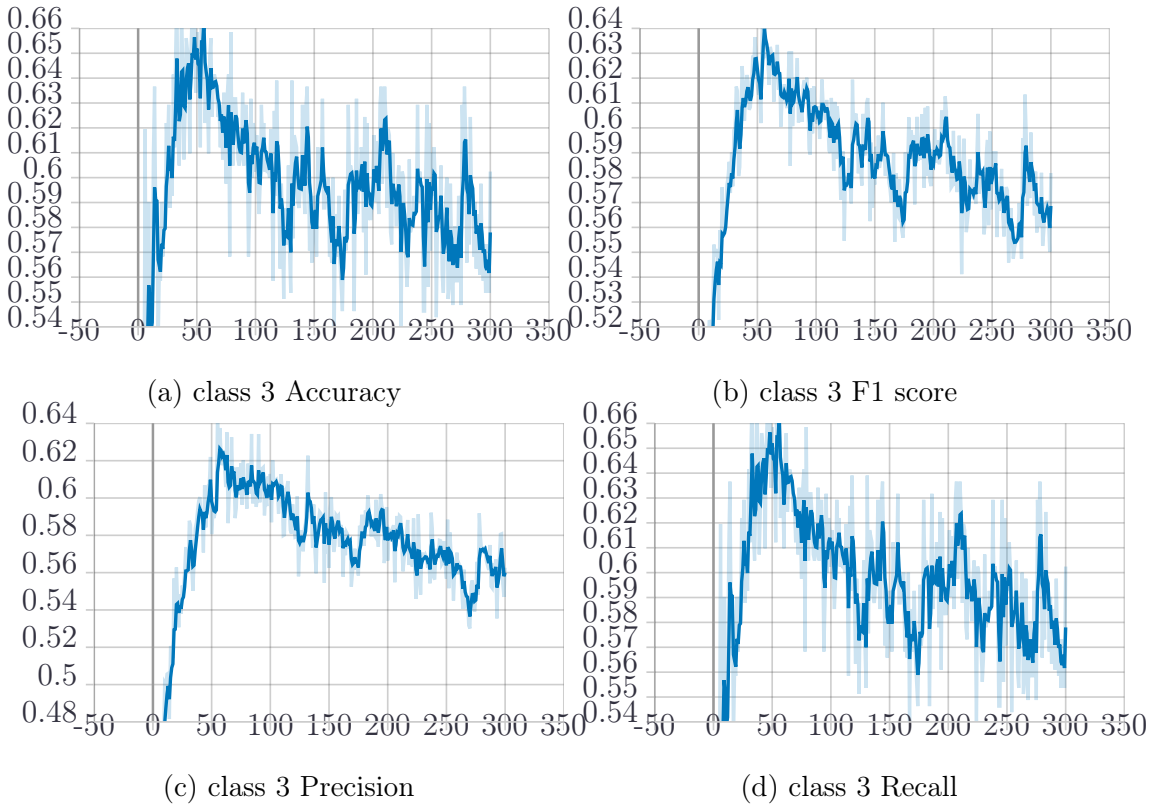
Accuracy performance of this class is calculated to be 55%

Precision is calculated to be 53 %

The ability of the model to find relevant cases “Recall” is calculated to be 55%.

F1 score is calculated to be 54%.

6.4.3 Results of the third class



(a) class 3 Accuracy

(b) class 3 F1 score

(c) class 3 Precision

(d) class 3 Recall

Figure 6.24: ResNet third class results

Performance Metrics for this class are calculated to be as follows:

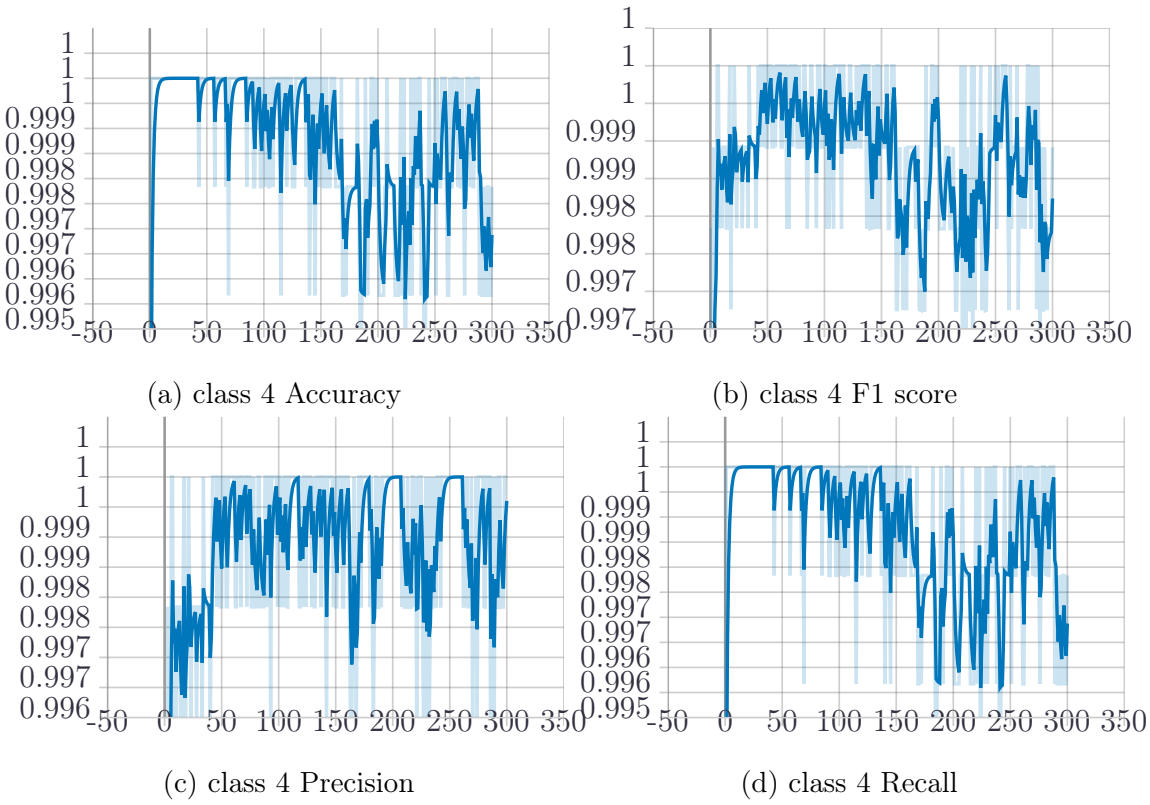
Accuracy performance of this class is calculated to be 63%

Precision is calculated to be 65 %

The ability of the model to find relevant cases “Recall” is calculated to be 62%.

F1 score is calculated to be 65%.

6.4.4 Results of the fourth class



(a) class 4 Accuracy

(b) class 4 F1 score

(c) class 4 Precision

(d) class 4 Recall

Figure 6.25: ResNet fourth class results

Performance Metrics for this class are calculated to be as follows:

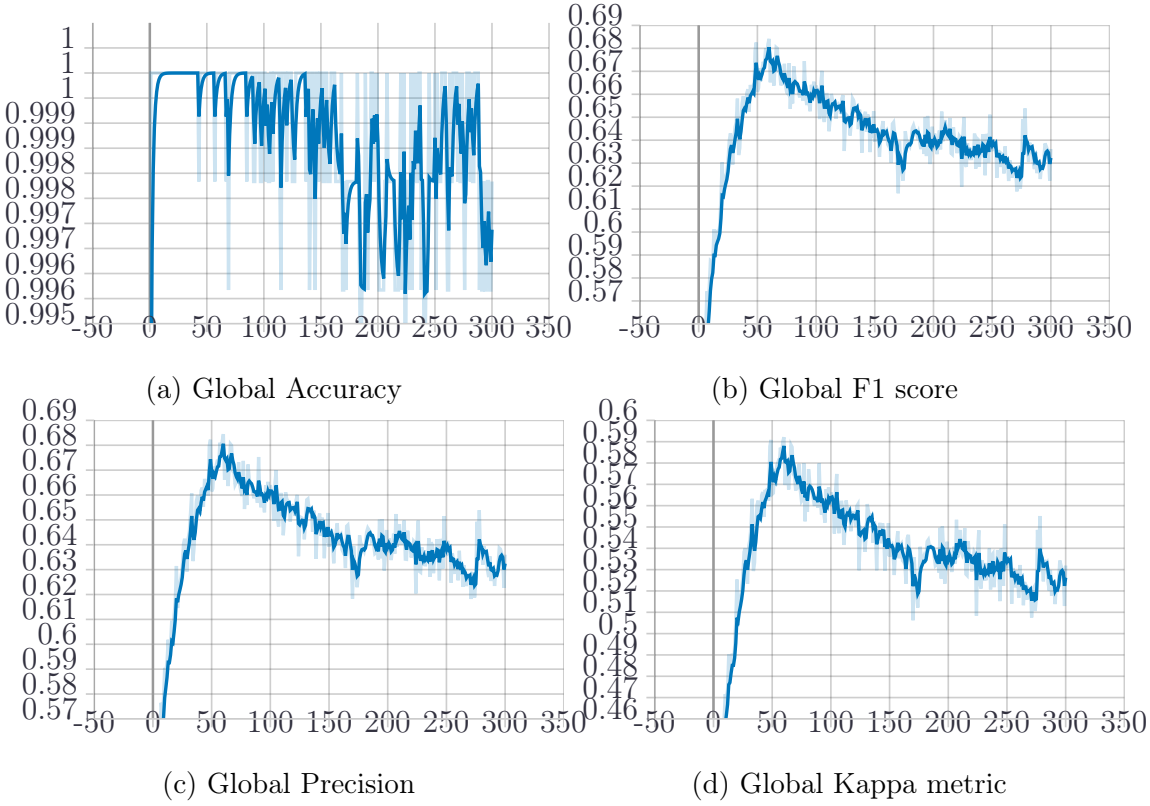
Accuracy performance of this class is calculated to be 100%

Precision is calculated to be 100 %

The ability of the model to find relevant cases “Recall” is calculated to be 100%.

F1 score is calculated to be 100%.

6.4.5 Global results



(a) Global Accuracy

(b) Global F1 score

(c) Global Precision

(d) Global Kappa metric

Figure 6.26: ResNet global results

Global Performance Metrics of this Model using a dataset over 20 subjects are calculated to be as follows:

Accuracy performance of this class is calculated to be 68 %

Precision is calculated to be 68 %.

The ability of the model to find relevant cases “Kappa Metric” is calculated to be 59 %.

F1 score is calculated to be 58 %.

Figure 6.27: ResNet confusion matrix

This confusion matrix shows the percentage of the correct classification for each class.

The fist class is classified correctly by 51.2 %

The second class is classified correctly by 46.8 %

The third class is classified correctly by 56.2 %

The fourth class is classified correctly by 100 %

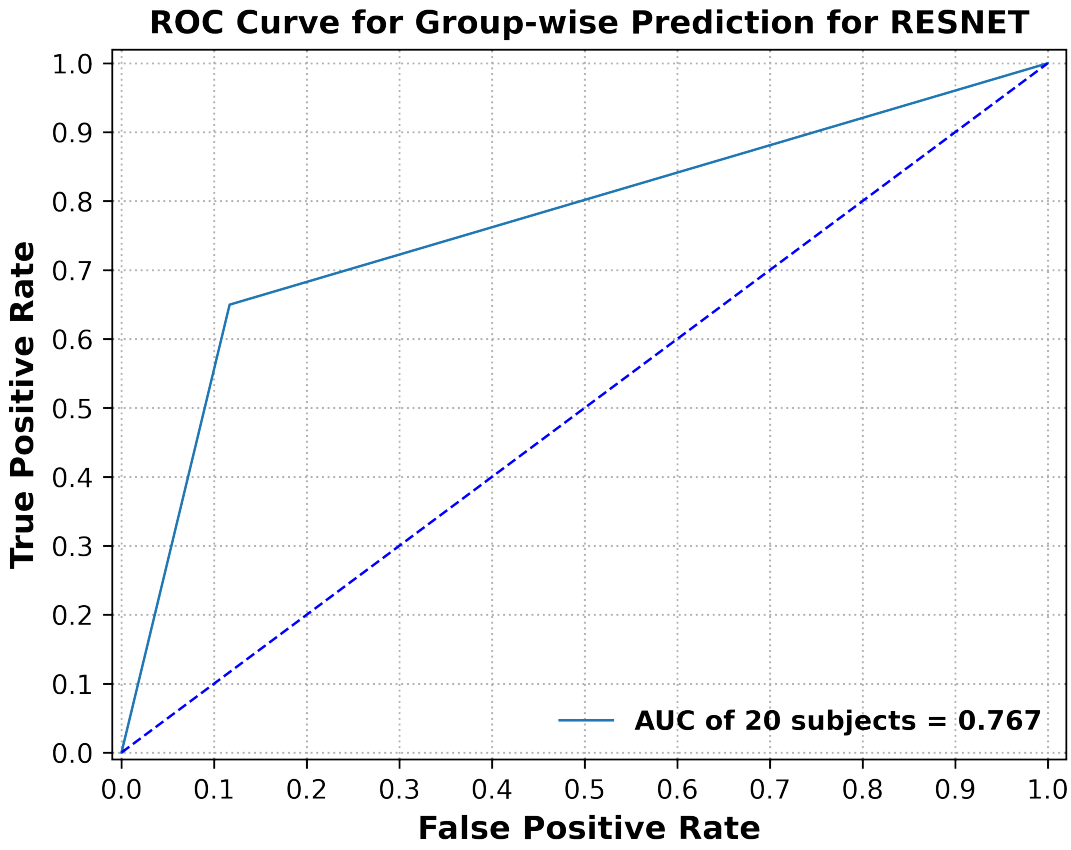


Figure 6.28: ResNet ROC curve

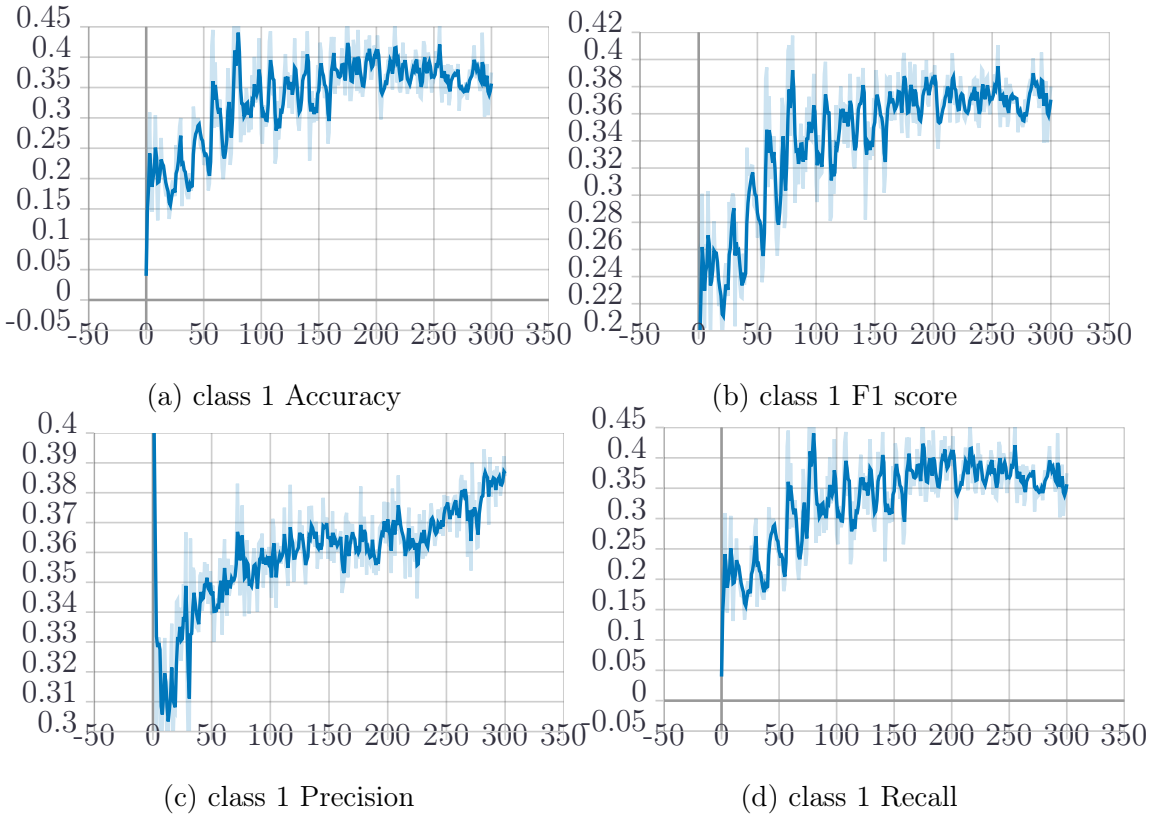
The ROC curve shows the trade-off between sensitivity (or TPR) and specificity (1 – FPR). Classifiers that give curves closer to the top-left corner indicate a better performance. As a baseline, a random classifier is expected to give points lying along the diagonal (FPR = TPR). The closer the curve comes to the 45-degree diagonal of the ROC space, the less accurate the test.

The area under curve (AUC) equals 0.766 unit of area.

6.5 Recurrent Neural Network —RNN

This section include the detailed results of the Recurrent Neural Network RNN which include the evaluation metrics for the four classes. Accuracy,F1 Score,Precision & Recall will be used to evaluate each class separately . The same four Metrics mentioned above as well as the Confusion Matrix & The receiver Operating Characteristics Curve"ROC Curve".

6.5.1 Results of the first class



(a) class 1 Accuracy

(b) class 1 F1 score

(c) class 1 Precision

(d) class 1 Recall

Figure 6.29: RNN first class results

Performance Metrics for this class are calculated to be as follows:

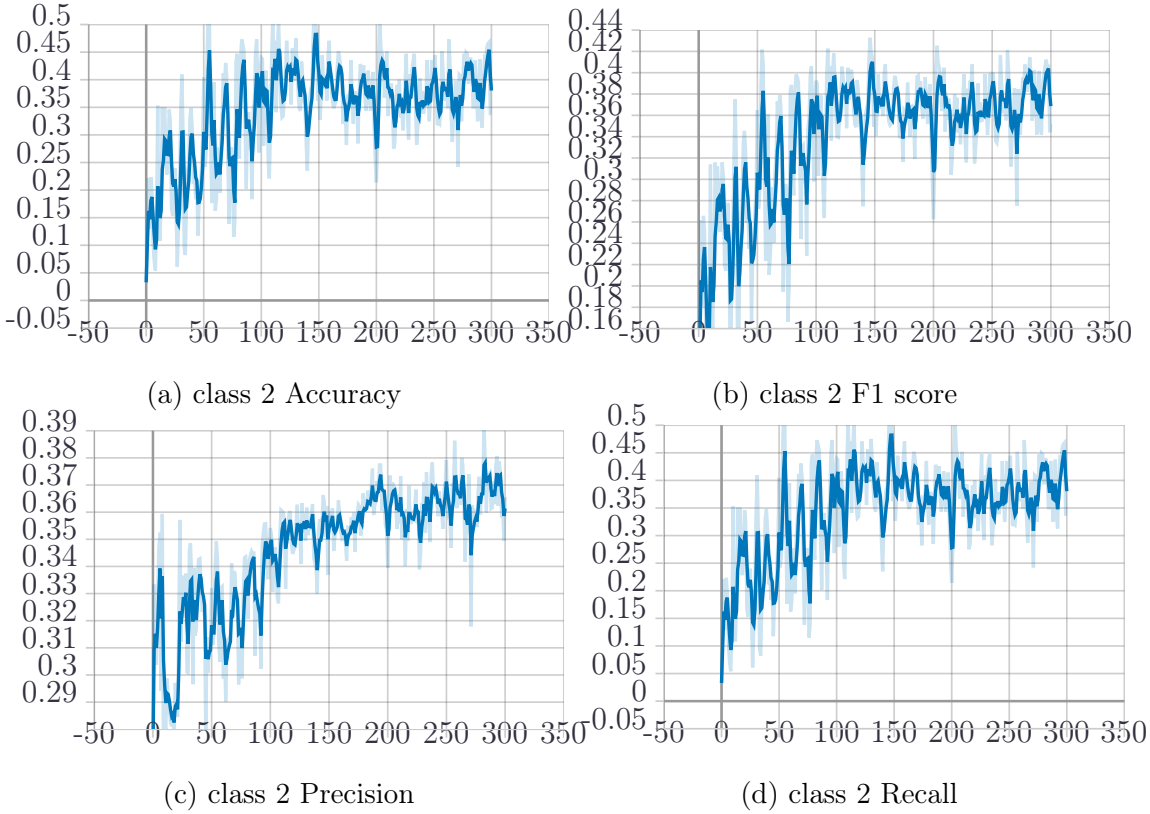
Accuracy performance of this class is calculated to be 41%

Precision is calculated to be 39 %

The ability of the model to find relevant cases “Recall” is calculated to be 40%.

F1 score is calculated to be 38%.

6.5.2 Results of the second class



(a) class 2 Accuracy

(b) class 2 F1 score

(c) class 2 Precision

(d) class 2 Recall

Figure 6.30: RNN second class results

Performance Metrics for this class are calculated to be as follows:

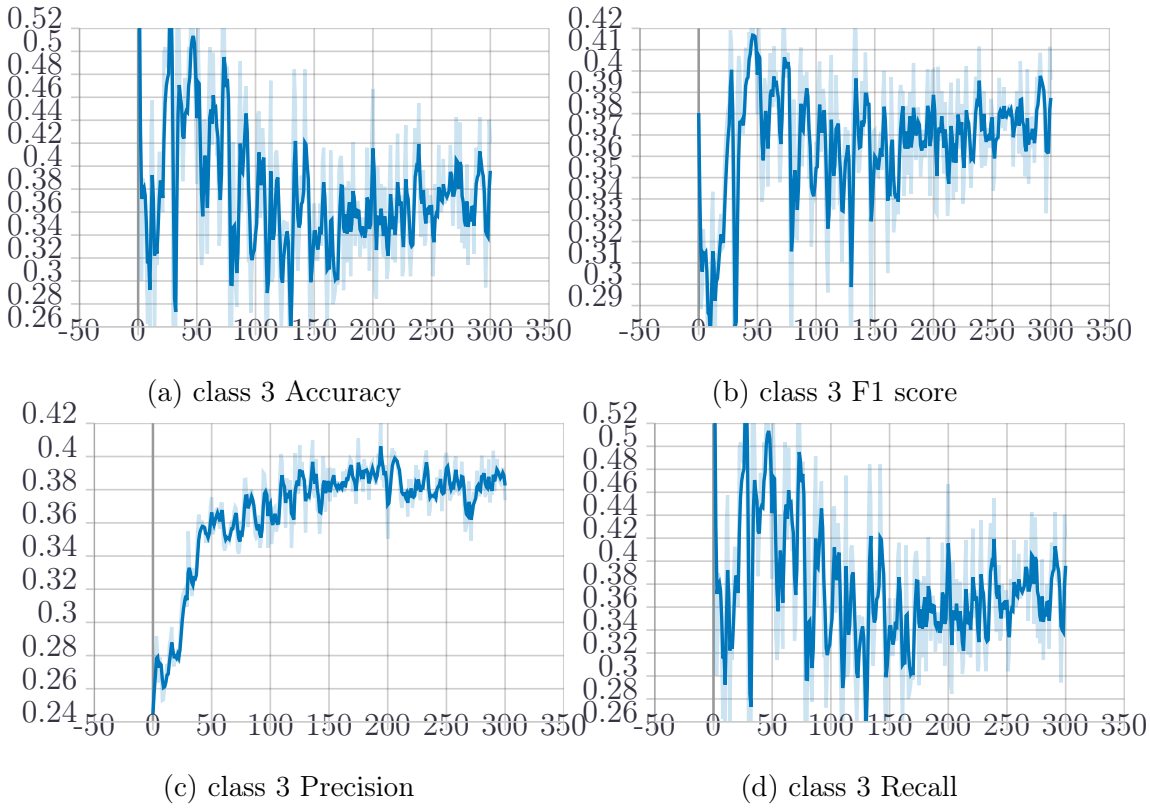
Accuracy performance of this class is calculated to be 40%

Precision is calculated to be 37 %

The ability of the model to find relevant cases “Recall” is calculated to be 40%.

F1 score is calculated to be 42%.

6.5.3 Results of the third class



(a) class 3 Accuracy

(b) class 3 F1 score

(c) class 3 Precision

(d) class 3 Recall

Figure 6.31: RNN third class results

Performance Metrics for this class are calculated to be as follows:

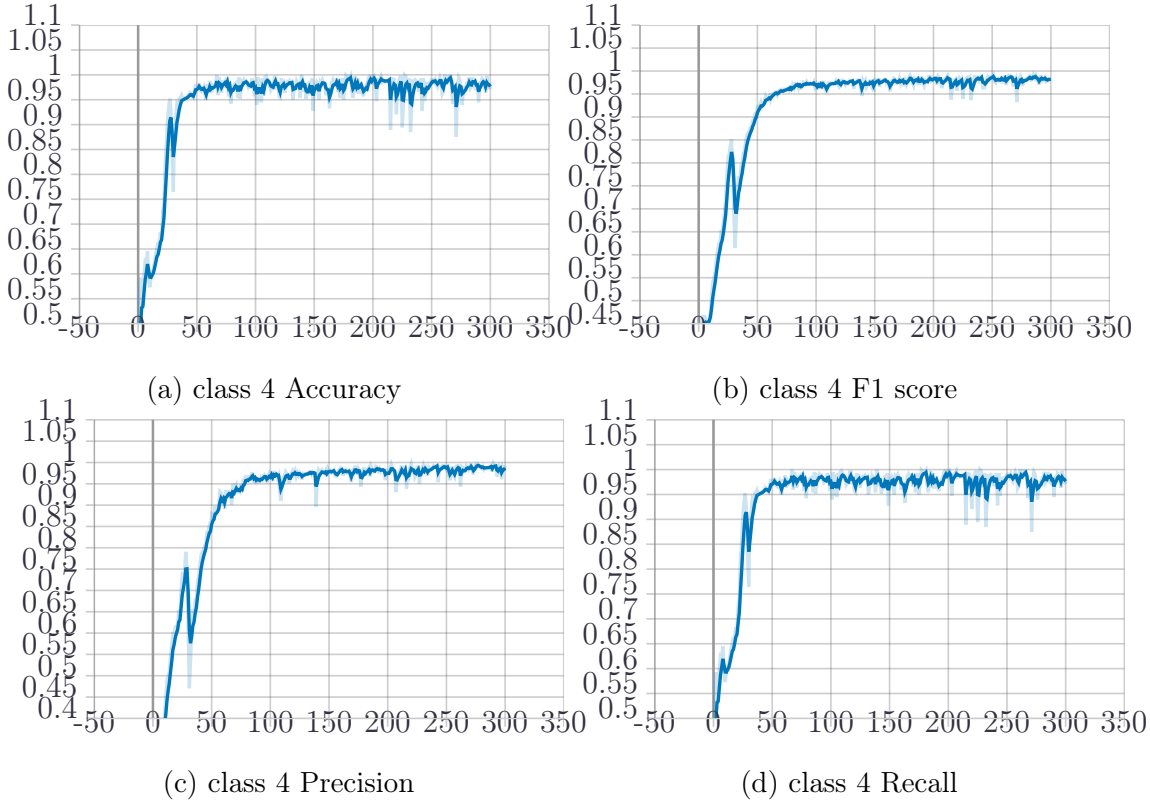
Accuracy performance of this class is calculated to be 51%

Precision is calculated to be 39 %

The ability of the model to find relevant cases “Recall” is calculated to be 51%.

F1 score is calculated to be 41 %.

6.5.4 Results of the fourth class



(a) class 4 Accuracy

(b) class 4 F1 score

(c) class 4 Precision

(d) class 4 Recall

Figure 6.32: RNN fourth class results

Performance Metrics for this class are calculated to be as follows:

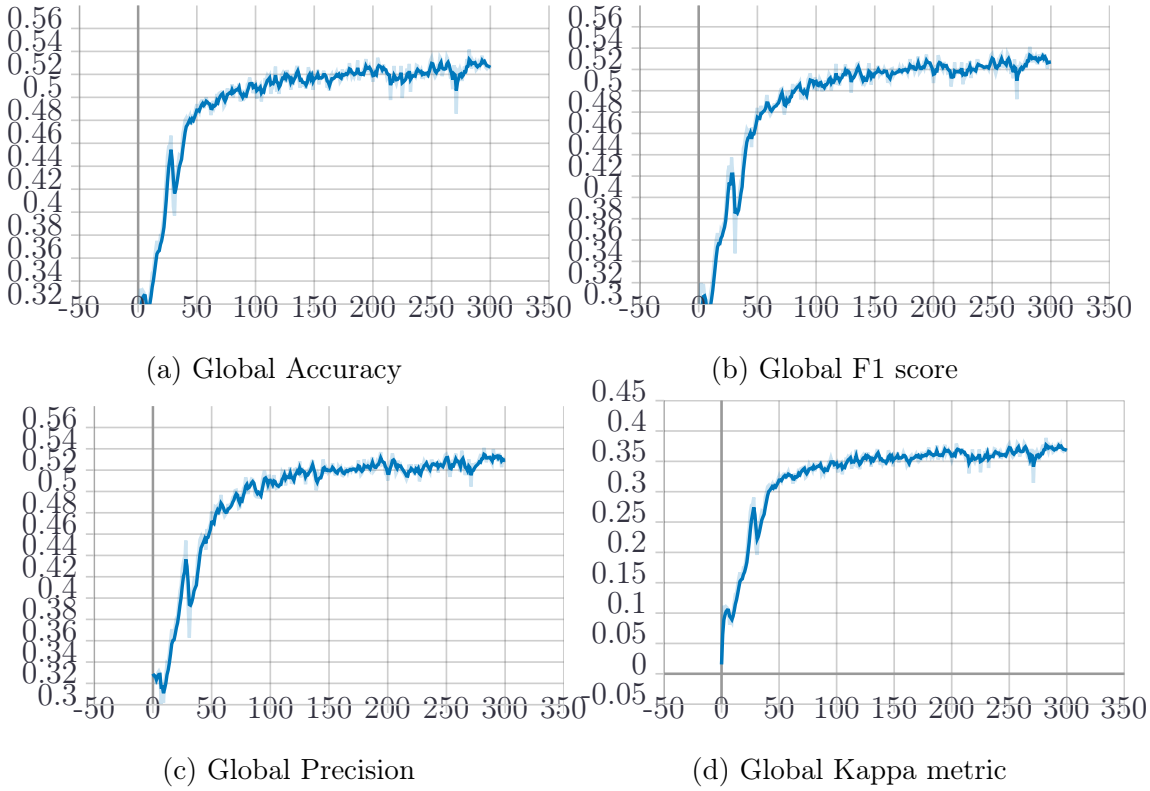
Accuracy performance of this class is calculated to be 95%

Precision is calculated to be 95 %

The ability of the model to find relevant cases “Recall” is calculated to be 95%.

F1 score is calculated to be 95%.

6.5.5 Global results



(a) Global Accuracy

(b) Global F1 score

(c) Global Precision

(d) Global Kappa metric

Figure 6.33: RNN global results

Global Performance Metrics of this Model using a dataset over 20 subjects are calculated to be as follows:

Accuracy performance of this class is calculated to be 54 %

Precision is calculated to be 53 %.

The ability of the model to find relevant cases “Kappa Metric” is calculated to be 38 %.

F1 score is calculated to be 53 %.

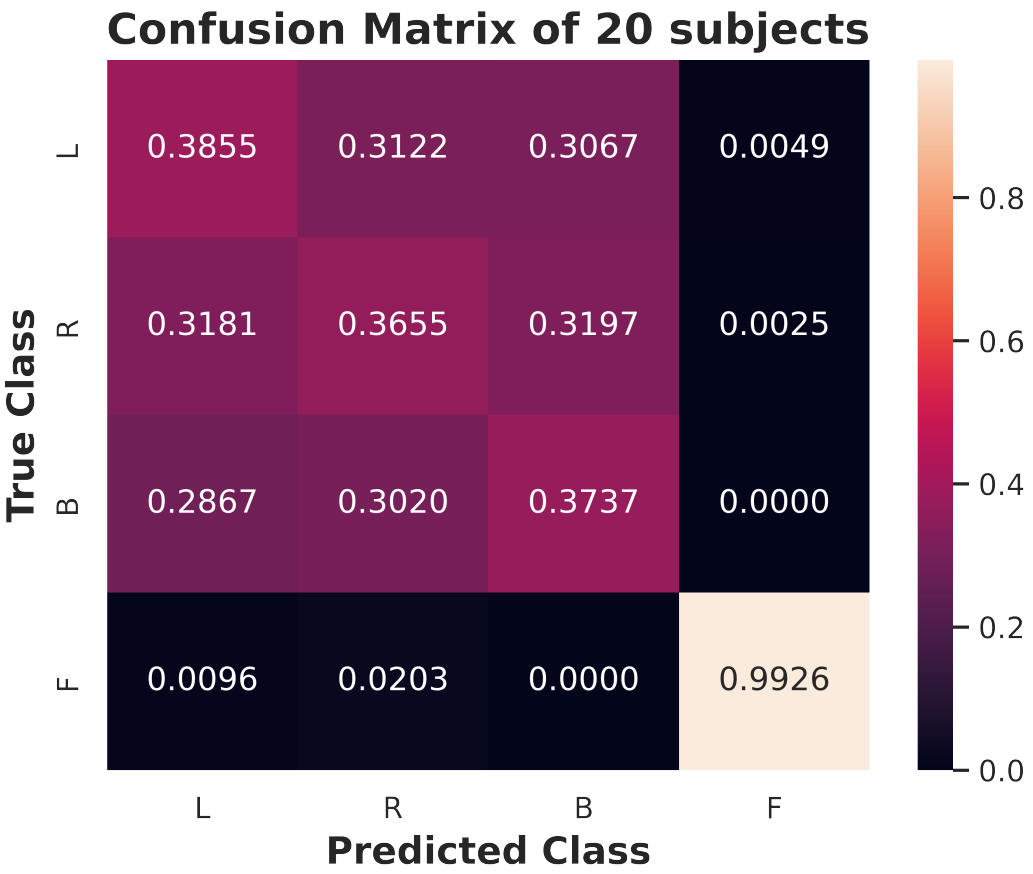


Figure 6.34: RNN confusion matrix

This confusion matrix shows the percentage of the correct classification for each class.

The fist class is classified correctly by 38.5 %

The second class is classified correctly by 36.5 %

The third class is classified correctly by 37.3 %

The fourth class is classified correctly by 99.2 %

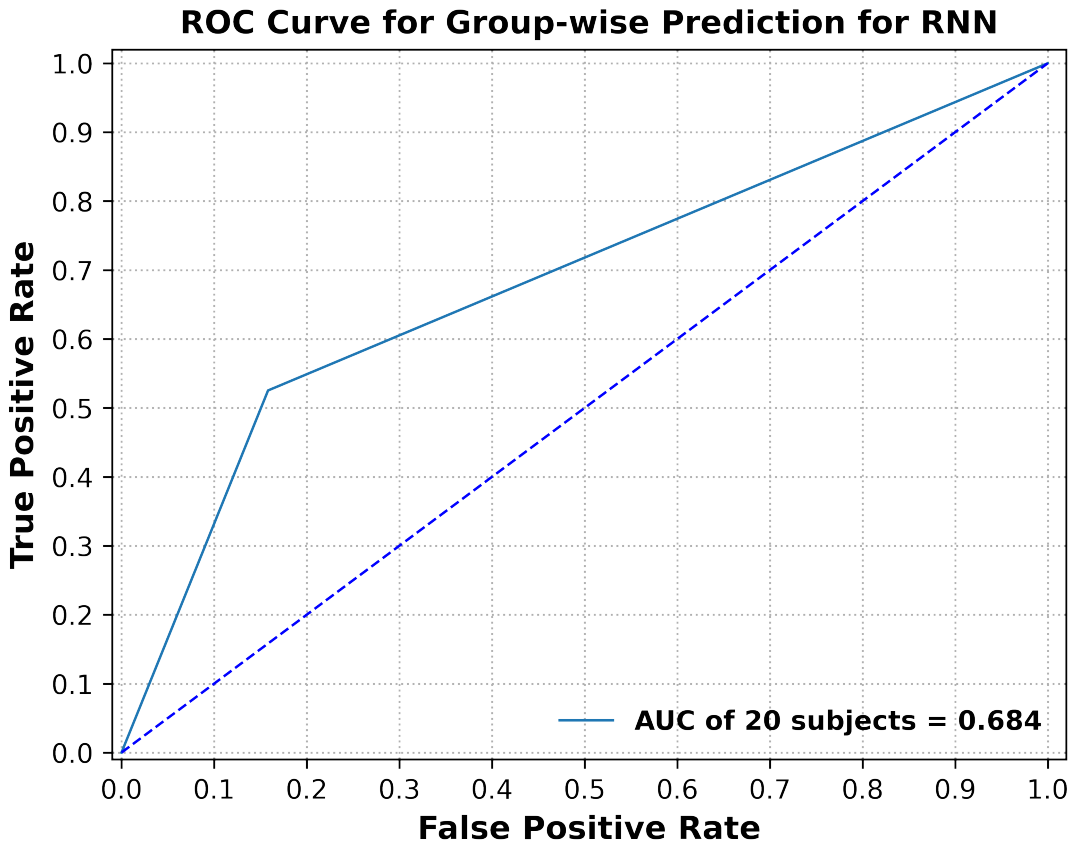


Figure 6.35: RNN ROC curve

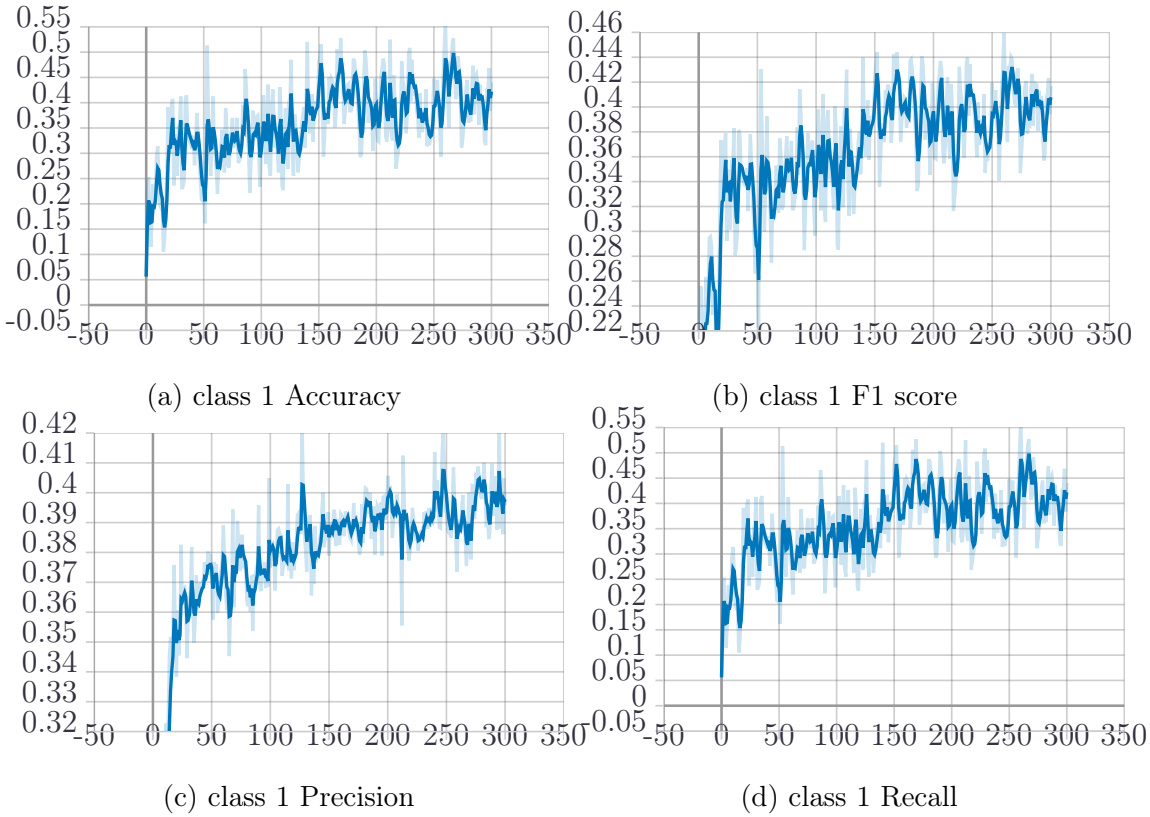
The ROC curve shows the trade-off between sensitivity (or TPR) and specificity (1 – FPR). Classifiers that give curves closer to the top-left corner indicate a better performance. As a baseline, a random classifier is expected to give points lying along the diagonal (FPR = TPR). The closer the curve comes to the 45-degree diagonal of the ROC space, the less accurate the test.

The area under curve (AUC) equals 0.68 unit of area.

6.6 Recurrent Neural Network With Attention

This section include the detailed results of the Recurrent Neural Network with attention RNN which include the evaluation metrics for the four classes. Accuracy,F1 Score,Precision & Recall will be used to evaluate each class separately . The same four Metrics mentioned above as well as the Confusion Matrix & The receiver Operating Characteristics Curve"ROC Curve".

6.6.1 Results of the first class



(a) class 1 Accuracy

(b) class 1 F1 score

(c) class 1 Precision

(d) class 1 Recall

Figure 6.36: RNN with Attention first class results

Performance Metrics for this class are calculated to be as follows:

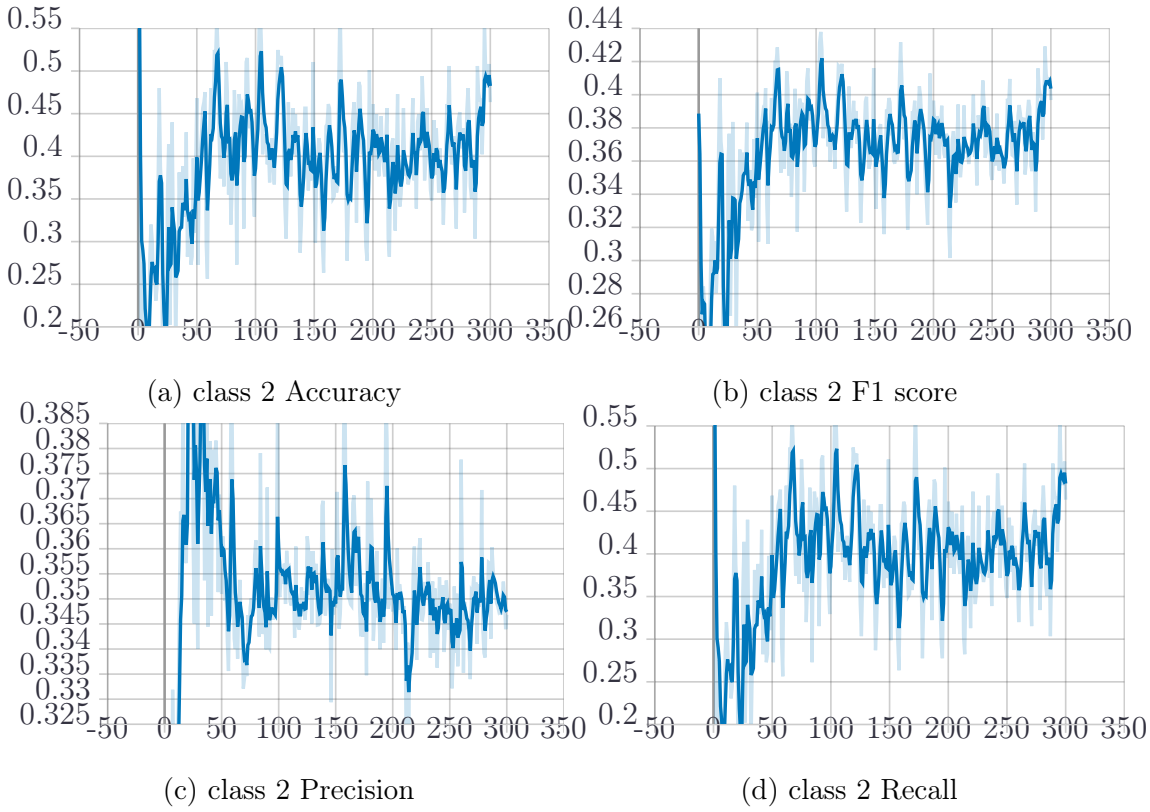
Accuracy performance of this class is calculated to be 51%

Precision is calculated to be 41 %

The ability of the model to find relevant cases “Recall” is calculated to be 50%.

F1 score is calculated to be 42%.

6.6.2 Results of the second class



(a) class 2 Accuracy

(b) class 2 F1 score

(c) class 2 Precision

(d) class 2 Recall

Figure 6.37: RNN with Attention second class results

Performance Metrics for this class are calculated to be as follows:

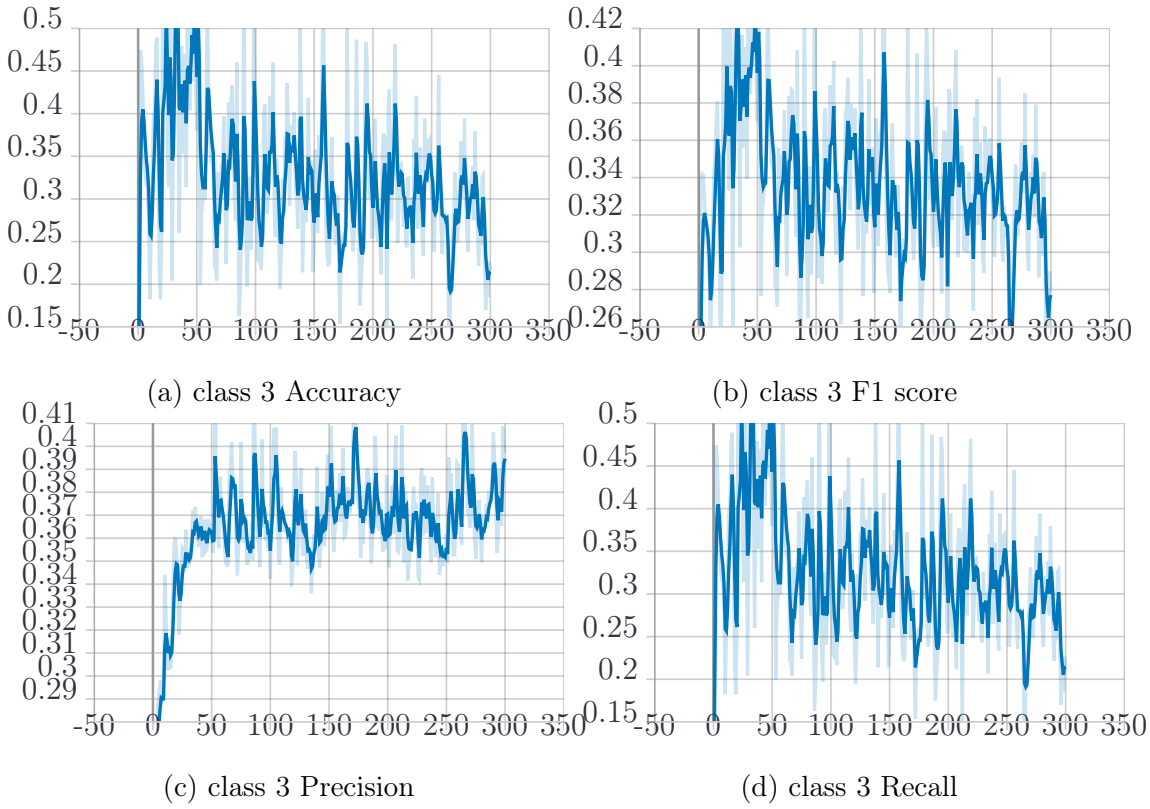
Accuracy performance of this class is calculated to be 55%

Precision is calculated to be 39 %

The ability of the model to find relevant cases “Recall” is calculated to be 52%.

F1 score is calculated to be 45%.

6.6.3 Results of the third class



(a) class 3 Accuracy

(b) class 3 F1 score

(c) class 3 Precision

(d) class 3 Recall

Figure 6.38: RNN with Attention third class results

Performance Metrics for this class are calculated to be as follows:

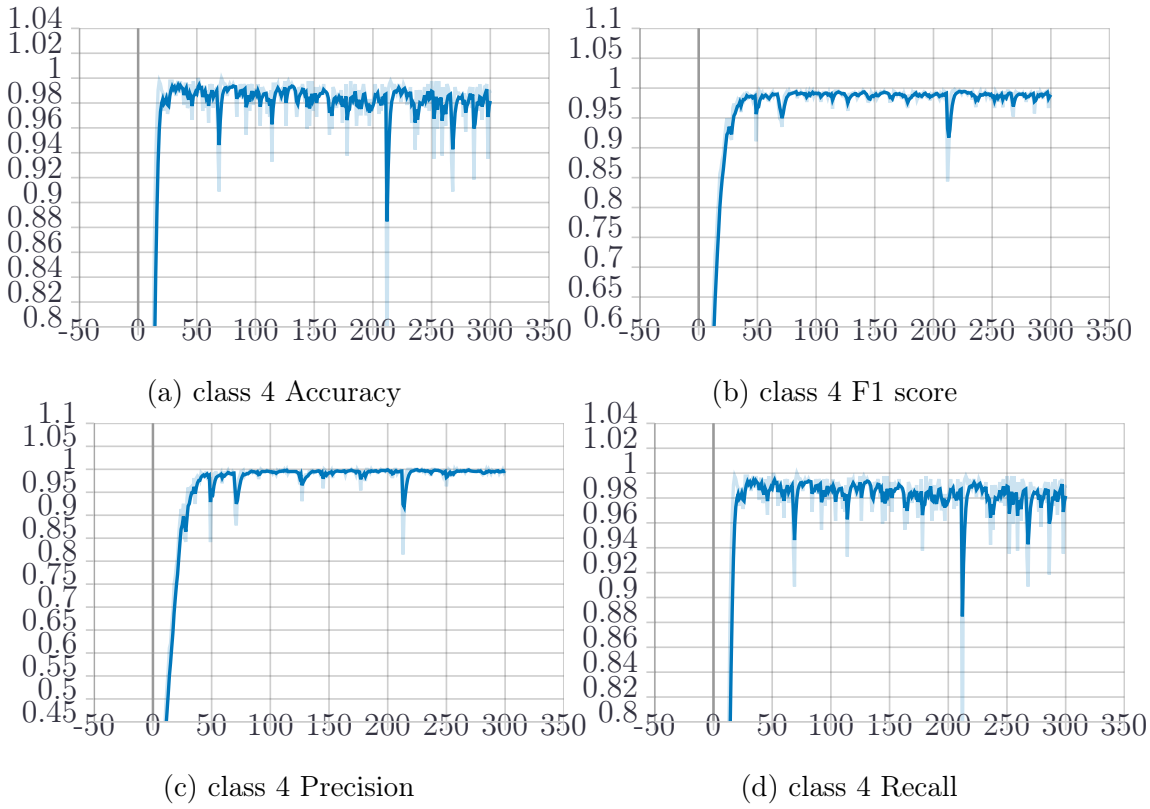
Accuracy performance of this class is calculated to be 50%

Precision is calculated to be 40 %

The ability of the model to find relevant cases “Recall” is calculated to be 50%.

F1 score is calculated to be 45 %.

6.6.4 Results of the fourth class



(a) class 4 Accuracy

(b) class 4 F1 score

(c) class 4 Precision

(d) class 4 Recall

Figure 6.39: RNN with Attention fourth class results

Performance Metrics for this class are calculated to be as follows:

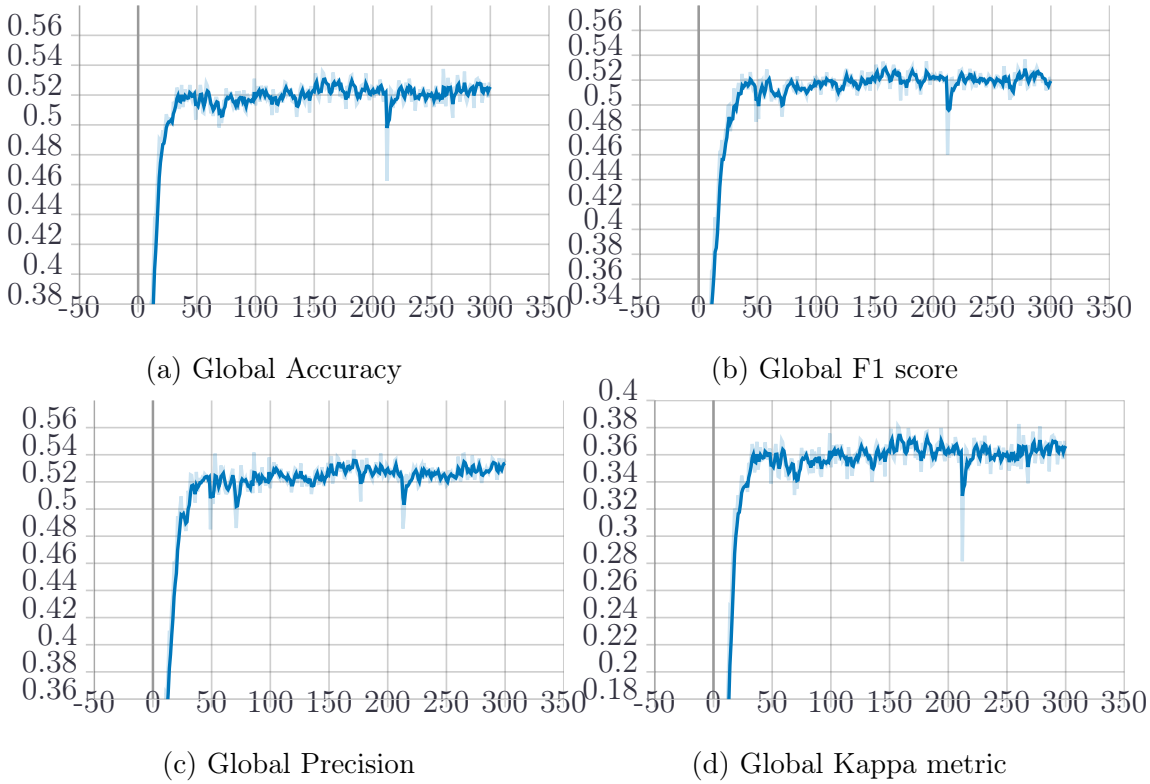
Accuracy performance of this class is calculated to be 95%

Precision is calculated to be 95 %

The ability of the model to find relevant cases “Recall” is calculated to be 95%.

F1 score is calculated to be 95%.

6.6.5 Global results



(a) Global Accuracy

(b) Global F1 score

(c) Global Precision

(d) Global Kappa metric

Figure 6.40: RNN with Attention global results

Global Performance Metrics of this Model using a dataset over 20 subjects are calculated to be as follows:

Accuracy performance of this class is calculated to be 54 %

Precision is calculated to be 54 %.

The ability of the model to find relevant cases “Kappa Metric” is calculated to be 38 %.

F1 score is calculated to be 52 %.

Figure 6.41: RNN with Attention confusion matrix

This confusion matrix shows the percentage of the correct classification for each class.

The fist class is classified correctly by 39.5 %

The second class is classified correctly by 34.5 %

The third class is classified correctly by 39.7 %

The fourth class is classified correctly by 99.7 %

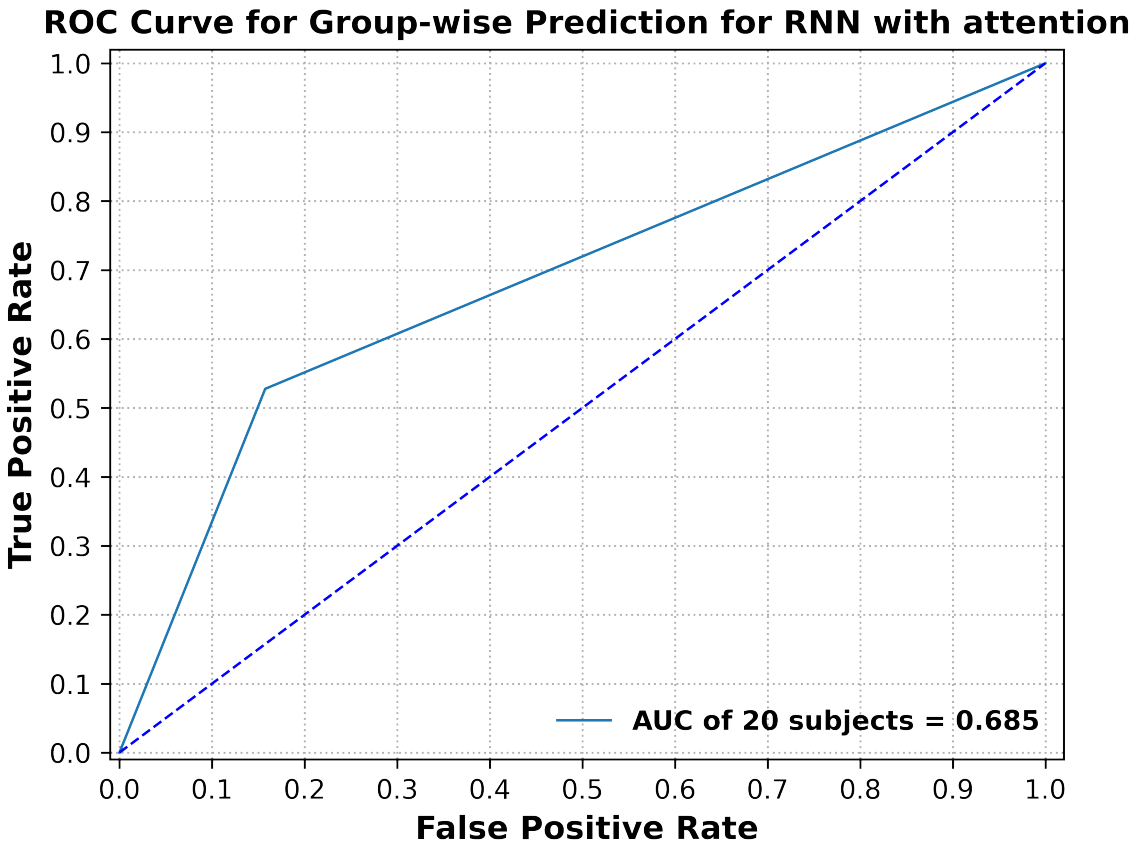


Figure 6.42: RNN with Attention ROC curve

The ROC curve shows the trade-off between sensitivity (or TPR) and specificity (1 – FPR). Classifiers that give curves closer to the top-left corner indicate a better performance. As a baseline, a random classifier is expected to give points lying along the diagonal (FPR = TPR). The closer the curve comes to the 45-degree diagonal of the ROC space, the less accurate the test.

The area under curve (AUC) equals 0.685 unit of area.

Chapter 7

Conclusion and Future Work

best performance metrics

cnn, fcn, resnet

Chapter 8

Future Work

Contents

Our future work is mainly focused on three main aspects:

- increasing the number of subjects in the dataset
- locating and utilizing an EEG reading headset with sufficient number of electrodes
- converting our model along with the bionic arm into a real-time processing mobile device

8.1 increasing number of subject

In the preprocessing phase of the data which utilized the Matlab signal processing software, the maximum array size was achieved even after using maximum available specs at the time"16 GB RAM", which lead to not being able to utilize more of the dataset.

As stated before our dataset is comprised of 109 subjects however due to GPU limitations only 20 subjects were able to be used in training and since normally the increase in training data leads to a better fitting of the model"better performance based on our metrics".

If we are able to increase the training data to include all 109 subject or even add on to the dataset beyond the 109. that would hopefully drastically increase our model's performance

8.2 Utilizing an EEG reading Headset

since our goal is ultimately being able to use this technology in real-time we have to use a device that is able to acquire the EEG signal also in real-time;therefore we need to implement this functionality. one device was used in the efforts of achieving this goal"Neurosky",However the performance of the Device was far from satisfactory as this device only has one electrode & it's only intended for specific more detectable tasks"Attention,relaxation".

8.3 Mobile Embedded System

in order for our product to be of effective daily use , it needs to be able to run "classify new data" on the go regardless of the environment. in order to achieve that it was proposed that we process data on a mobile RASPBERRY PI/NVIDIA JETSON microprocessor in order to be able to achieve the mobility functionality.

Bibliography