

1.ABSTRACT

We know that our brain functions with various signals. These signals are generated with every activity that we do, even in our sleep. We basically capture these signals by putting electrodes on our scalp and then in the EEG it is recorded as wave patterns. Normal activity would have the usual pattern but abnormal EEG has some distinguishable features. Doctors can identify abnormal EEG from the normal ones after some observation.

Although EEGs helps us differentiate between Normal and Abnormal, it is still heavily dependent upon the examiner to give the last judgement. In order to interpret the signals captured by EEG, we need the help of an expert in this field. Our main objective is to lessen the burden on examiner and reduce the time of examination.

The final goal of this project is to automate the whole process of examination of classifying EEGs. We have now focused only on classifying an EEG into its appropriate type. This automation will help reducing the overall time required to identify the EEG and help the examiner.

2.PROBLEM STATEMENT

To classify EEG signals into Normal and Abnormal EEG Signals using various technique and compare the results.

3.INTRODUCTION

Brain is a complex as well as significant part of human. It also holds many secrets such as how the memories are stored in our brain, how different abnormalities like seizures etc. Now we have come to such point of research that we are much closer to discovering the mysteries of brain.

Among the various measures of analysis, electro-encephalogram is one the most used in the research. Normally outpatient EEG diagnosis takes about 20 min. But this duration is not always sufficient for the analysis of epilepsy or other seizure disorders.

Here comes the question-How do we capture EEG? Basically EEG is the brain waves which captured by metal disc placed on the scalp. Whenever we use our brain, brain emits signal and the metal disc, i.e electrodes, captures this signal and store them in the record as waves.

EEG reports are manually diagnosed by medically certified physicians. As it is performed manually, it may take a lot of time to analyze each of the report thoroughly. Not only this, the analysis is purely based on the subjective interpretation of the examiner. This may lead to some errors.

As we can see that there can be various issue when manually analyzing, we can try to approach this issue using automation of the process. The thought of fully automating the process is interesting. But in order to do that we have to solve the individual module separately and accurately. So in this project, we are focusing on how to filter the EEG reports into normal and abnormal cases.

We want to proceed with first pre-processing the data with various methods to learn the distinguishable features in those EEG. Then comes the training of various models. In order to do that we are hoping to use pre-trained models with traditional machine learning algorithms.

Given that deep learning is unbiased towards the features currently used in visual inspection and is able to learn from raw data, it can be an alternative to visual inspection and traditional machine learning methods for EEG analysis.

4. LITERATURE SURVEY

There are many researches that are already being done for this EEG classification.

Among these the most prominent ones are:

1. Lopez, S. (2017). Automated Identification of Abnormal EEGs. Temple University.
2. Deep Learning Enabled Automatic Abnormal EEG Identification. by Subhrajit Roy_{_1}, Isabell Kiral-Kornek₁, and Stefan Harrer₁, IEEE Senior Member
3. Automated detection of abnormal EEG signals using localized wavelet filter banks.
Manish Sharma ^{a,*}, Sohamkumar Patel ^a, U. Rajendra Acharya
4. A new feature extraction and classification mechanisms For EEG signal processing. by Hemant Choubey₁ · Alpana Pandey₁

1.

For the first paper, we can see that it was first introduced by the Temple university Hospital. As the EEG classification by Human Examiner takes a lot of time, they first proposed that automation of this process would be much beneficial in the medical science.

Their main approach was to implement a classifier based on Hidden Markov Model (HMM) , Convolution Neural Network – Multiple Layer Perceptron (CNN-MLP), KNN and Random Forest Ensemble Classifier.

The performance is as follows in their research:

Table 1. Summary of the performance for all the evaluated systems

System Description	Error (%)
kNN (k=20)	41.8%
RF (Nt=50)	31.7%
PCA-HMM #GM = 3 #HMM States = 3)	25.6%
GMM-HMM (#GM = 3 #HMM States = 3)	17.0%

2.

For the Deep Learning based Automatic Abnormal EEG Identification, they have introduced an approach of using deep neural network in order to classify the data into normal and abnormal. They have used some pre-processing techniques in order to extract features. They didn't use hand-engineering techniques to extract the feature but learn the features.

They have used some deep learning techniques to learn the features. We have also introduced this technique in our project.

They have pre-processed the data into three types –

- Direct using the time series wave signal
- Converting the signal into visual spectrogram
- Converting the signal into GAF

Now after processing the data, they have introduced some classifier models to predict the EEG Signals such as

- Logistic Regression
- MLP Classifier
- 1-D CNN
- 2-D CNN
- 1-D CNN-RNN
- TCNN-RNN

The result obtained by their research work is given as:

Pre. proc.	Algo.	Log. Reg.		MLP		1D-CNN		2D-CNN		1D-CNN-RNN		TCNN-RNN	
		Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
Time-series	83.72%	49.09%	69.64%	54.15%	82.04%	76.90%	N/A	N/A	99.16%	82.27%	N/A	N/A	
Spectrograms	N/A	N/A	N/A	N/A	N/A	N/A	86.31%	70.39%	N/A	N/A	95.22%	71.48%	
GAF	N/A	N/A	N/A	N/A	N/A	N/A	79.46%	68.61%	N/A	N/A	92.55%	67.02%	

TABLE I: Comparison of performance for combination of pre-processing techniques and learning algorithms.

3.

For the next paper, they have introduced another research approach as novel stop-band energy (SBE) minimized orthogonal wavelet filter bank. After using this method they have used SVM classifier to classify the EEG signals into abnormal. They have used unorthodox method to implement this classifier.

4.

For the next paper, they have proposed a method of extracting the features using Masking and Check-in based Feature Extraction Technique (MCFET) and an integrated K-Means with K-Nearest Neighbour classification algorithms to classify the EEG Signals into Normal and Abnormal Signals.

For this method of extracting, first mask is generated , then the check in function is calculated, upper and lower envelops are calculated. Finally the features are extracted from this.

After obtaining this feature, they have used knn classifier to classify the signals into normal and abnormal EEG Signal.

5.1.PROPOSED METHODOLOGY

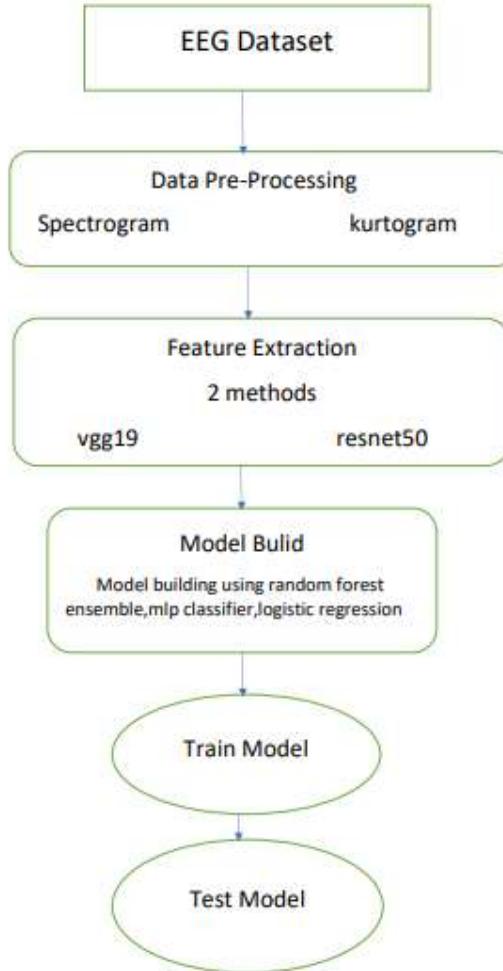
To classify the EEG wave signals into two classes, i.e. normal and abnormal EEG signal, we introduced a three-step architecture.

In the first step, we saw that there are several nodes that captures the brain signal and plot it into EEG. In a single session performed by the examiner, we can see that one patient has about 30 to 40 such nodes associated with his EEG report. To store the EEG report there is a specific data format used. Its name is EDF. Now we have raw data in EDF format and to make use of this data, we have process it according to our needs. So, the first step will be Data Pre-processing.

In the second step, we have to extract features from the processed data. In order to do that we intend to use some techniques and take the best technique after we get the desired result.

In the final step, we have to build some classifier model and train it using the features that we extracted from the previous step. After that we have to test the classifier using the test data that we have and get the desired accuracy we need in order to classify the EEG signals into Normal and Abnormal EEG signals.

5.2.BLOCK DIAGRAM



In this block diagram, we can see the basic Block Diagram of our work-flow.

5.2.1.

First Block of our diagram is the dataset and for this block of our work-flow, we are mainly focusing on how the data is situated as EDF format. How this data is then converted as matlab data array.

5.2.2.

Second Block of our diagram is the data pre-processing. In this flow of our work-flow, we basically pre-processed the data into suitable data to obtain

temporal frequency spectrum. Our main intuition was that the data is basically signal with temporal axis. So, temporal frequency spectrum would be appropriate for data pre-processing.

5.2.3.

Third block of our block diagram is feature extraction. In this stage, we used pre-trained model to extract features. in that case we used 2 models, i.e VGG19 and ResNet50 model. In convolution neural network, we see that in each convolution layer the model tries to obtain new features from the image and tries to recognise more complex feature. Using that intuition, we used these 2 pre-trained models to extract features from our dataset.

5.2.4.

In the fourth block, we built our main classifier model that would classify our data into Normal and Abnormal EEG signals. There are many binary classifier that are available. We used random forest ensemble, logistic regression and MLP classifier.

5.2.5.

In the next step, we used our extracted feature vector to train our 3 models.

5.2.6.

After that, we tested our test data to obtain how much accuracy our model can gain. This part concludes our project.

6.1.DATASET

In order to implement our project, we needed a dataset of EEG signals and their classification of Normal and Abnormal. For this we got the “Temple University Hospital EEG Data Corpus” that was provided by our respected mentor, Dr. Rishav Singh.

In this dataset, the EEG signals are obtained by taking a EEG session of about 15 minutes of several patients and observing the brain signals with the help of metal discs placed on the scalps of the patients. These signals are thereafter stored in the data format of EDF. These EDF data format consists the name of the nodes from where the signals are coming from, the sampling rate of the signal, signals in form of wave and etc.

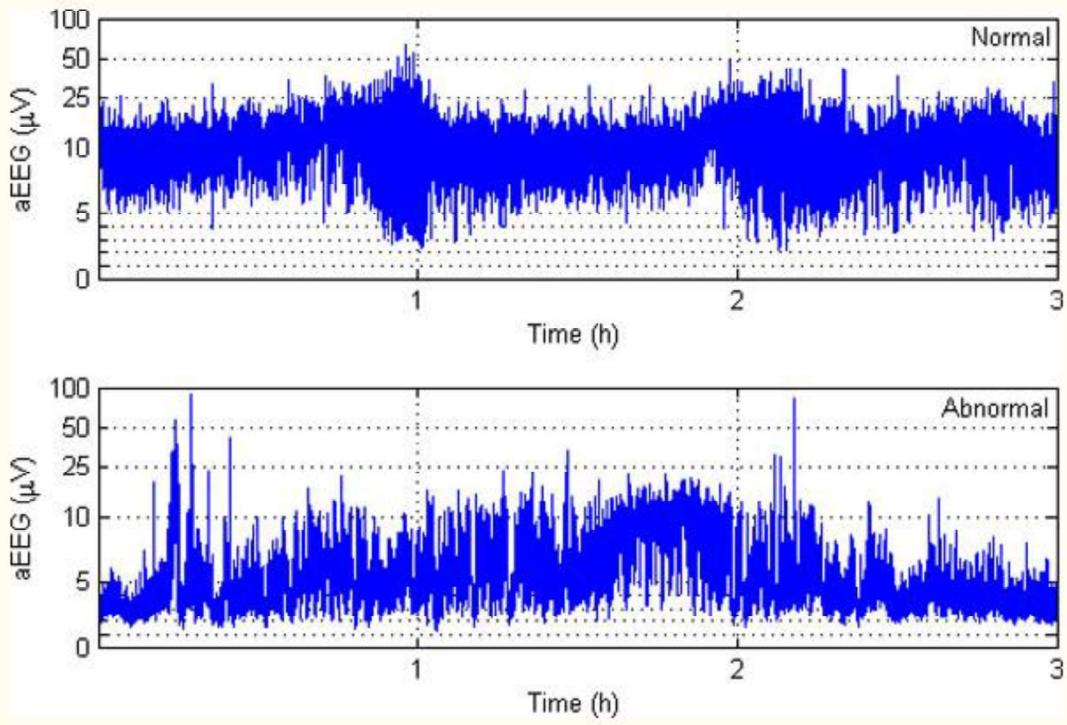
The distribution of the dataset is given below:

Table 1. File statistics for the full evaluation set.

Evaluation					
Description	Files	Patients	hours		
Abnormal	126	46.4%	105	41.5%	48.9
Normal	150	53.6%	148	58.5%	55.4
Total	276	100.0%	253	100.0%	104.4

Table 2. File statistics for the full training set.

Training					
Description	Files	Patients	Hours		
Abnormal	1346	50.2%	899	42.1%	546.4
Normal	1371	49.8%	1239	58.0%	518.3
Total	2717	100.0%	2138	100.0%	1064.7



Here we can see an example of EEG signal. In the above figure one of them is normal and another one is abnormal. From visually analysing the figure, we are able to see that the normal one is having a uniform band of voltage that holds from 5-25.

But for the abnormal, we can see that it is having a band of non-uniform voltage. It is also having some spikes.

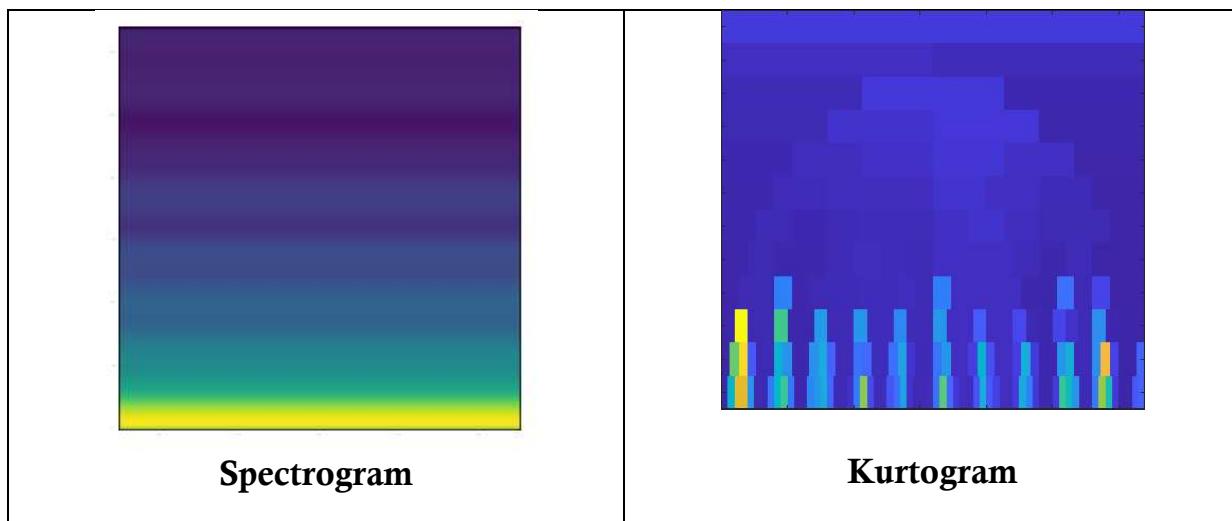
6.2.DATA PRE-PROCESSING

First of all, the dataset is in the form of EDF. So, to get the process this data into our useful data, we converted this EDF data into matlab array data. Now we have a matrix of data on which we can apply several pre-processing techniques.

The dataset has many nodes from which 21 signals of each data-point are picked and for this our accuracy increased by 10%.

There are several pre-processing techniques that are available. As we wanted to use CNN based feature extraction in the next step, we had to use pre-processing techniques that would convert the signals into temporal frequency spectrum. Some of these techniques are spectrogram and kurtograms.

We used both of these techniques to pre-process the data into spectrograms and kurtograms and stored them for the next step. This temporal frequency spectrum would be helpful in order to extract the features.



In order to read the dataset in python, we have used “pyedflib” library in python. After reading the data in python, we obtained the signal in array and used that array to save in matlab array using the “scipy.io” and savemat() function.

```
import pyedflib
from scipy.io import savemat

def saveM(sigbufs,j):
    sigbuf = sigbufs.flatten('F')
    dic = {"a":sigbuf}
    print('data'+str(j)+'.mat')
    filename = dname+'/'+stype+'/'+data'+str(j)+'.mat'
    savemat(filename,dic)
```

After obtaining the matlab data array, we used matlab software to convert the array signal into spectrogram and kurtogram.

In order to obtain the spectrogram and kurtogram took a lot of time, one whole day to be precise. I think the time took such a long time as the processing of a single data was done at a time. For this reason, for converting 3k data points individually took such a long time.

6.2.1. SPECTROGRAM

The spectrogram contains a compromise between time resolution and frequency resolution: the large window provides less local processing at a time and more discrimination at times. The window receives a piece of signal, where the spectral features are almost invariant

the found parts move the timer window at a certain interval. The spectrogram is defined as the size of $S(m, k)$, represented as $A(m, k)$, as a show in equation

$$A(m,k) = \frac{1}{N} |S(m,k)|^2.$$

Spectrogram adjustment can be improved to change the window length; The long window offers the best solution for frequency, but the worst time fix. A shorter window, however, offers better timing but better resolution resolution. Good visibility in the spectrogram depends on choosing the right window length and spacing. Figure 2 shows the signal spectrogram, which is a multi-signal signal display. Spectrum programming is usually as follows: x-axis represents time, y-axis represents frequency, and the third dimension is the amplitude (visual content) of the frequency, which is encoded. This three-dimensional data can also create a 3D structure, where power is represented as height on z-axis but a 2D chart provides better understanding.

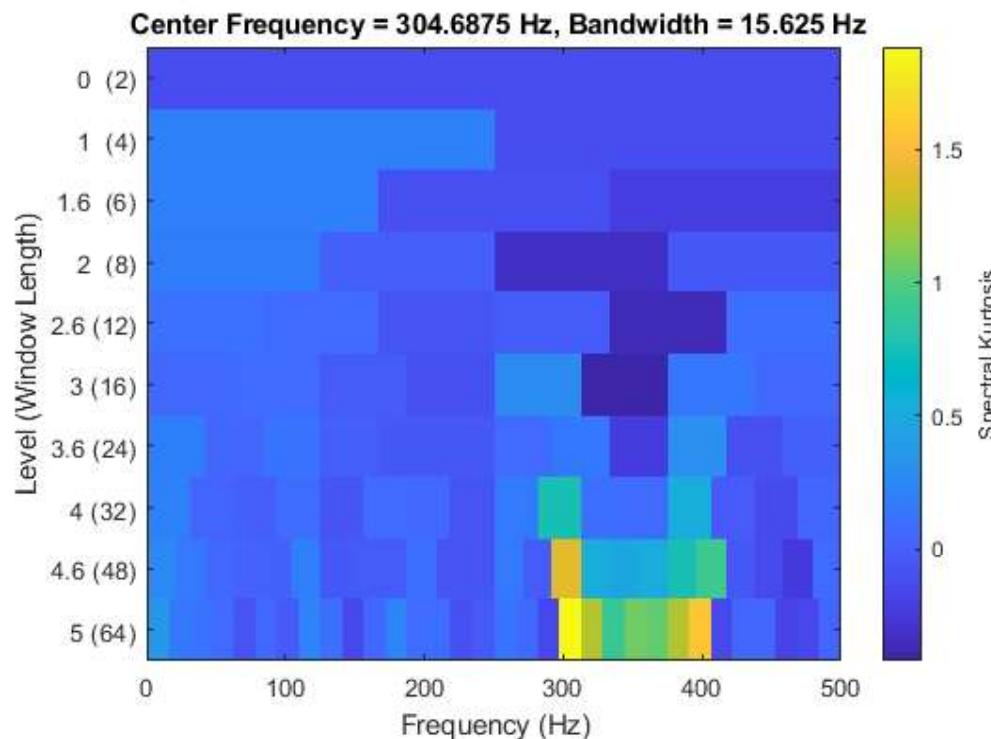


Spectrogram of
An EEG signal

6.2.2. KURTOGRAM

A kurtogram is a spectral analysis tool used to detect irregularities in the signal area. It can be used effectively to find a suitable filter to remove the feature element from the vibration damp, because error signal-induced signal times can be considered static. However, the efficiency of the kurtogram decreases when the signal is collected in a structure that operates under various speed conditions.

A kurtogram is a newly developed spectral analysis tool for the fourth order to detect and mark non-signal objects. The paradigm is based on the premise that each temporal type is associated with the correct dyad {frequency (frequency / frequency) dyad {} which increases its kurtosis, hence its detection. However, a complete overhaul of all aircraft () is a daunting task that is incomparable to online industrial applications.



7.FEATURE EXTRACTION

For the second step of our project, we intended to use CNN based pre-trained model for extracting the features from the spectrograms and kurtograms. For our project we used VGG19 and RESNET50 pre-trained model.

VGG19 pre-trained on “Imagenet” dataset consists of 8 convolution layers and with fully-connected layers total 11 layers. In the case of VGG19, we used “block4_conv1” layer output as the extracted feature. After obtaining the feature, we flattened the feature and made it into a feature vector and will use it in the next step as training data.

ResNet50 consists of 5 stages each with a convolution block and an identity block. Each Convolution block has 3 convolution layer and each identity block also has 3 convolution layers. For our project, we used some of the layers such as “conv3_block2_add”.

After extracting the features, we flattened the output layer and got the desired featured vector. Now we shall use this to train our classifier model to get the accuracy.

7.1. VGG19:

For the VGG19 model, the architecture is given as:

Model: "vgg19"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 256, 256, 3)]	0
block1_conv1 (Conv2D)	(None, 256, 256, 64)	1792
block1_conv2 (Conv2D)	(None, 256, 256, 64)	36928
block1_pool (MaxPooling2D)	(None, 128, 128, 64)	0
block2_conv1 (Conv2D)	(None, 128, 128, 128)	73856
block2_conv2 (Conv2D)	(None, 128, 128, 128)	147584
block2_pool (MaxPooling2D)	(None, 64, 64, 128)	0
block3_conv1 (Conv2D)	(None, 64, 64, 256)	295168
block3_conv2 (Conv2D)	(None, 64, 64, 256)	590080
block3_conv3 (Conv2D)	(None, 64, 64, 256)	590080
block3_conv4 (Conv2D)	(None, 64, 64, 256)	590080
block3_pool (MaxPooling2D)	(None, 32, 32, 256)	0
block4_conv1 (Conv2D)	(None, 32, 32, 512)	1180160
block4_conv2 (Conv2D)	(None, 32, 32, 512)	2359808
block4_conv3 (Conv2D)	(None, 32, 32, 512)	2359808
block4_conv4 (Conv2D)	(None, 32, 32, 512)	2359808
block4_pool (MaxPooling2D)	(None, 16, 16, 512)	0
block5_conv1 (Conv2D)	(None, 16, 16, 512)	2359808
block5_conv2 (Conv2D)	(None, 16, 16, 512)	2359808
block5_conv3 (Conv2D)	(None, 16, 16, 512)	2359808
block5_conv4 (Conv2D)	(None, 16, 16, 512)	2359808
block5_pool (MaxPooling2D)	(None, 8, 8, 512)	0

Total params: 20,024,384

Trainable params: 20,024,384

Non-trainable params: 0

From this model, we took the output layer “block4_conv1” and used those to extract the features.

We can see that we have used adam optimizer as optimizer to extract features and used the categorical cross-entropy as a loss function. The pre-trained model is using the “imagenet” as weights.

The ultimately the output is stored in bottleneck_train1 and later we have flattened this bottleneck_train1 and made it into a feature vector.

In this way, we obtain the features from VGG19 model.

```
layers_list = ["block4_conv1"]
for layer_feature in layers_list:

    optimizer = Adam(lr=0.0001)

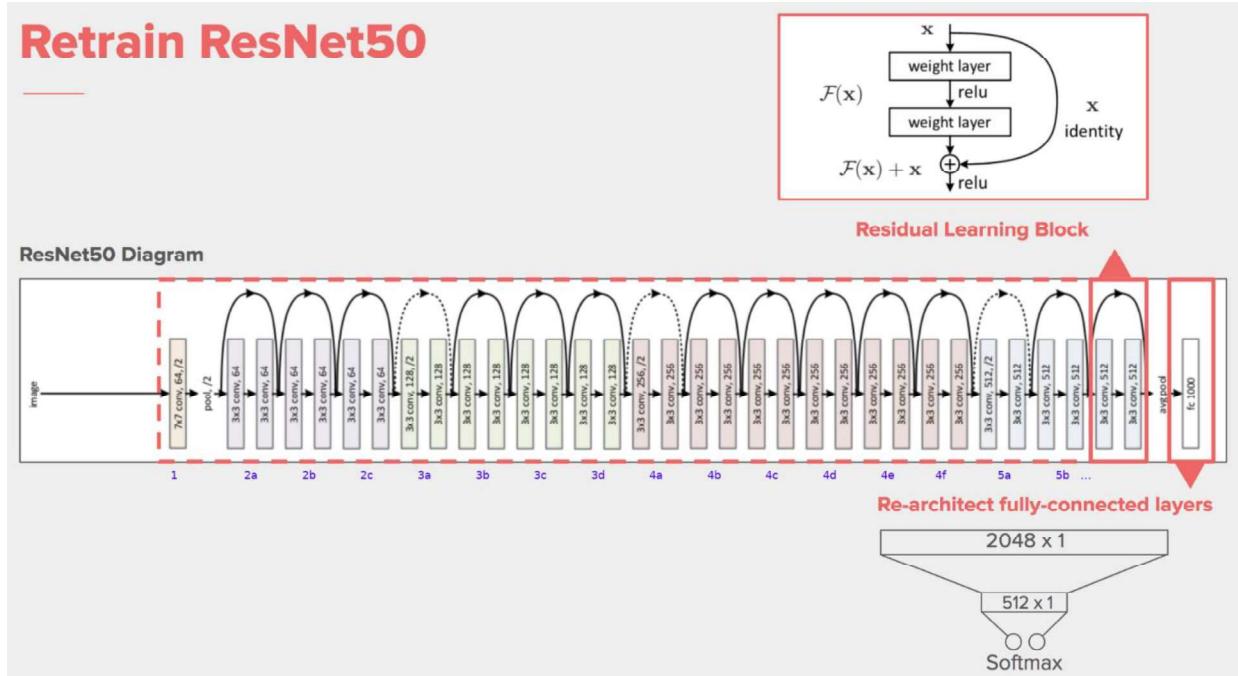
    arg_model= Model(inputs=model_vgg19.input, outputs=model_vgg19.get_layer(layer_feature).output)
    arg_model.compile(loss='categorical_crossentropy',
                      optimizer=optimizer,
                      metrics=['accuracy'])

    from keras.applications.vgg19 import preprocess_input
    bottleneck_test1=arg_model.predict(preprocess_input(x_test),batch_size=50,verbose=1)
    bottleneck_train1 = np.zeros((x_train.shape[0],32, 32, 512))
    for k in range(9):
        n = x_train.shape[0]
        bottleneck_train1[(n*k)//10:(n*(k+1))//10]=arg_model.predict(preprocess_input(x_train),batch_size=50,verbose=1)

    print(bottleneck_test1.shape)
    print(bottleneck_train1.shape)
```

7.2. ResNet50:

For the ResNet50 model, the architecture is as follows:



From this model, we have extracted “conv3_block2_add” output layer.

We can see that we have used adam optimizer as optimizer to extract features and used the categorical cross-entropy as a loss function. The pre-trained model is using the “imagenet” as weights.

The ultimately the output is stored in bottleneck_train1 and later we have flattened this bottleneck_train1 and made it into a feature vector.

In this way, we obtain the features from ResNet50 model.

```
layers_list = ["conv3_block2_add"]
for layer_feature in layers_list:
    #downloading model for transfer learning

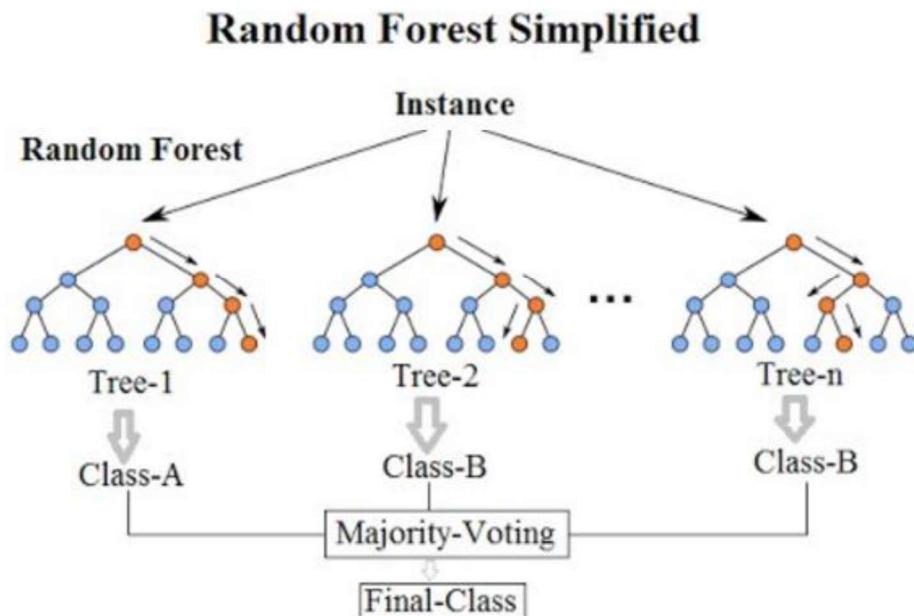
    optimizer = Adam(lr=0.0001)

    arg_model= Model(inputs=model.input, outputs=model.get_layer(layer_feature).output)
    arg_model.compile(loss='categorical_crossentropy',
                      optimizer=optimizer,
                      metrics=['accuracy'])
    bottleneck_train1=arg_model.predict(new_x_train,batch_size=8,verbose=1) #calculating bottleneck features
    #bottleneck_val1=arg_model.predict(preprocess_input(X_val),batch_size=50,verbose=1)
    bottleneck_test1=arg_model.predict(new_x_test,batch_size=8,verbose=1)
    print(bottleneck_test1.shape)
    print(bottleneck_train1.shape)
```

8.MODEL BUILD

For our classifier model, we used several models. These models are mainly Random Forest Classifier, Logistic Regression and MLP Classifier.

8.1. Random Forest Classifier:

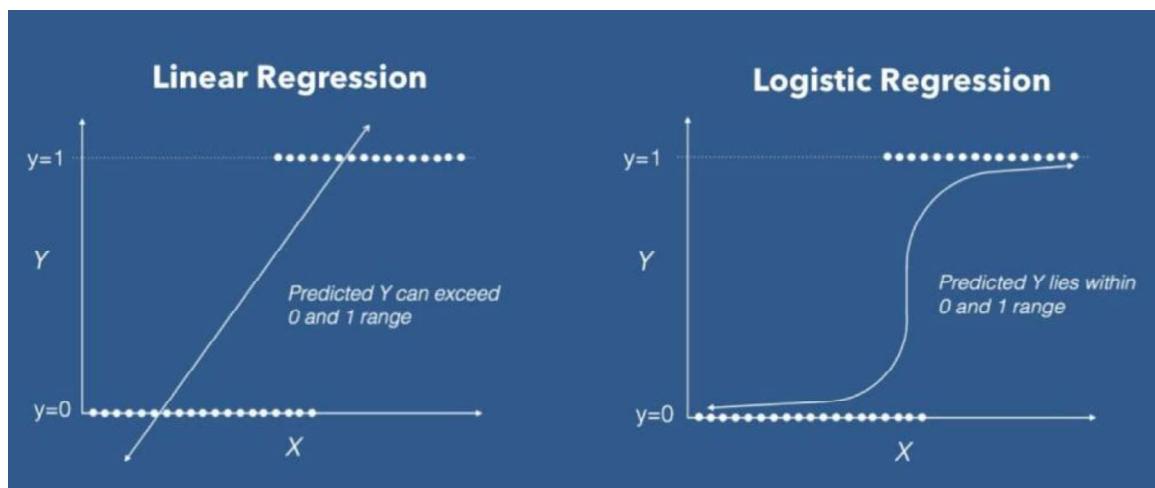


When we try to train a bunch of individual models in parallel with each having a random subset of the actual dataset and later taking the best out of the outputs generated by those individual models, it is called Bagging. Random Forest Ensemble Classifier is implemented upon Bagging. Random forest classifier uses an ensemble learning method for classification which uses multiple decision trees during training phase and outputs average prediction of individual trees. This classifier generates forests with random amount of trees. Normal decision tree algorithms are rule based and are solely based on some set of rules for prediction on data set. In contrast to this, random forest classifiers instead of using gini index or information gain for calculation of root node, it finds the root node and splits the features randomly. The following Figure

illustrates a general working of the classifier. x represents the input to the classifier. Random decision trees are generated namely tree1, tree2, ..., treeB, whose corresponding outputs are k_1, k_2, \dots, k_B . Majority Voting is taken and class k is selected from k_1, k_2, \dots, k_B . Output of the classifier is class k with majority of the votes.

In this, we train n decision trees with n randomly generated subset of the actual dataset. These n decision trees would give n output and using the majority vote concept we get the final output.

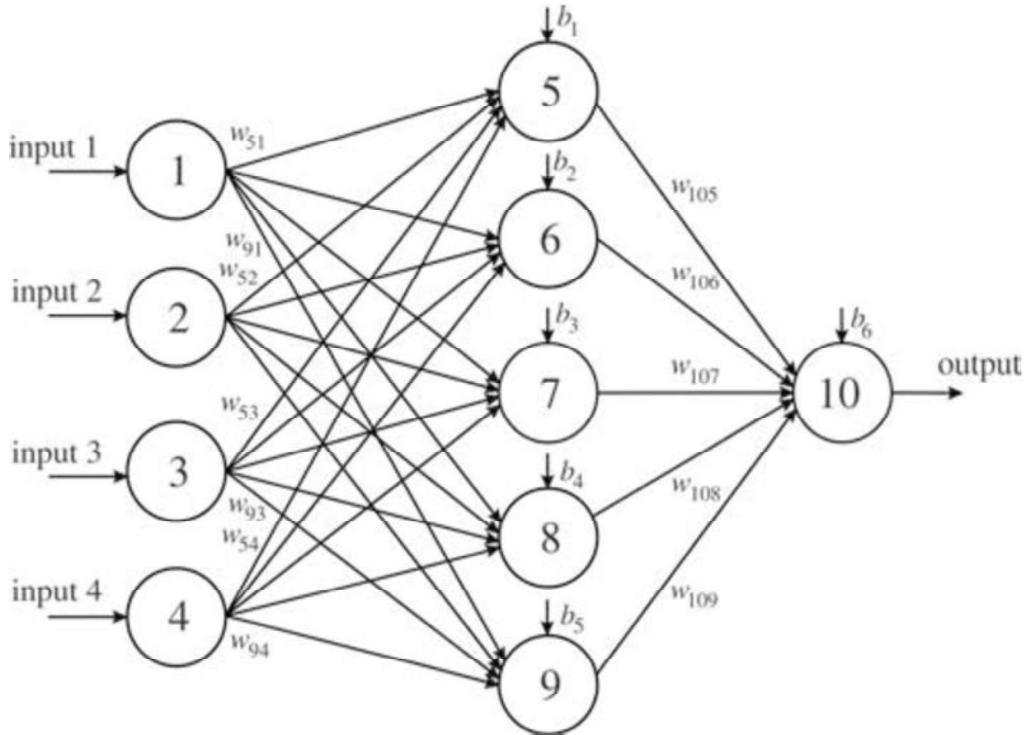
8.2. Logistic Regression:



For the logistic regression, we try to minimize the binary cross-entropy by using back-propagation algorithm. In this, we use sigmoid output function on $(W^T X + b)$ and get the value of y in range of $(0,1)$. By using logistic regression, thus we can get a binary classifier.

For linear regression, the predicted y becomes large for large values of x . for that reason, we cannot classify using linear regression. But in logistic regression, as we can see in the figure, it uses sigmoid activation function to get the value of predicted y in range of $(0,1)$.

8.3. MLP Classifier:



For the MLP classifier, we have neural network and some hidden layers. By using back-propagation we can train the parameters and get the final output with binary output layer. In this classifier, first we have an input layer. In the last we have output layer. Between these two layers there exist some hidden layers as we can see in the figure.

Input layer has the same shape of our feature vector. For the output layer, we have nodes same as the number of our output classes. In our case we have two output classes, i.e Normal and Abnormal.

9. TRAINING THE MODEL

For the training of the models, we used the extracted feature vector and feed it into the several model that we had built. We built and trained our model in google collab using 25 GB Ram and 6 GB Graphics Memory. For training, we used two feature vectors. One obtained from VGG19 and another one from ResNet50.

9.1. Random Forest Classifier:

```
from sklearn.ensemble import RandomForestClassifier  
clf = RandomForestClassifier(max_depth=5000, random_state=0)  
rftrain = bottleneck_train  
clf.fit(rftrain,y_train1)
```

For the random forest ensemble classifier, we used maximum depth of 5000 and trained it till it reached completion. It took about 20 to 30 min with the help of our GPU.

In the above figure, we can see that we have used sklearn library to implement our Random Forest Classifier. The model is named as clf and we have fitted the model with our extracted features.

9.2. Logistic Regression:

```
from sklearn.linear_model import LogisticRegression

# Parameters setting for the model
classifier_LR = LogisticRegression(C=1.0, max_iter=100, solver='sag', verbose=1)

#Train the model
classifier_LR.fit(bottleneck_train, y_train1)
```

For the Logistic Regression, we used iteration of 100 to 200 to train our model. It took about 30 min.

In the above figure, we are able to see that we have used sklearn to implement our logistic regression. The model that we are using is having max iteration of 100. The solver that the model is using is sag. It converges faster for large dataset and also handles L2 or no penalty. For that reason, we have used sag as our solver. After that we have fitted that model with our feature vector.

9.3. MLP Classifier:

```
from sklearn.neural_network import MLPClassifier

# Parameters setting for the model
classifier_LR = MLPClassifier(random_state=1, max_iter=100, verbose=True)

# Train the model
classifier_LR.fit(bottleneck_train, y_train1)
```

For the MLP Classifier, we used iteration of 100 to train our model. It took about 30 min to train.

In the above figure, we can see that the MLPClassifier model is implemented using sklearn library. After that it is having some parameters such as max iteration and so on. After that we have fitted the model using our extracted feature vector.

10. TESTING THE MODEL

For testing the test data, we had to go through with the same feature extraction that we had done for the training data. After we extracted the features from the test data, we used our several trained models to predict the output and compare the various test accuracy.

10.1. Random Forest Classifier:

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(max_depth=5000, random_state=0)
rftrain = bottleneck_train
clf.fit(rftrain,y_train1)
rftest = bottleneck_test
classifier_rf_pred = clf.predict(rftest)
```

In the above figure, we can see that for random forest classifier after fitting, we have used our test data extracted vector to predict if it is normal and abnormal. After that we have stored it into classifier_rf_pred array for generating the accuracy for later.

10.2. Logistic Regression:

```
from sklearn.linear_model import LogisticRegression

# Parameters setting for the model
classifier_LR = LogisticRegression(C=1.0, max_iter=100, solver='sag', verbose=1)

# Train the model
classifier_LR.fit(bottleneck_train, y_train1)

# Predict the values on test data
logistic_regression_pred = classifier_LR.predict(bottleneck_test)
```

In the above figure, we can see that for Logistic Regression after fitting, we have used our test data extracted vector to predict if it is normal and abnormal. After that we have stored it into logistic_regression_pred array for generating the accuracy for later.

10.3. MLP Classifier:

```
from sklearn.neural_network import MLPClassifier

# Parameters setting for the model
classifier_LR = MLPClassifier(random_state=1, max_iter=100, verbose=True)

# Train the model
classifier_LR.fit(bottleneck_train, y_train1)

# Predict the values on test data
MLP_pred = classifier_LR.predict(bottleneck_test)
```

In the above figure, we can see that for MLP Classifier after fitting, we have used our test data extracted vector to predict if it is normal and abnormal. After that we have stored it into MLP_pred array for generating the accuracy for later.

11.RESULT

The several accuracies obtained by our models are given below:

11.1. VGG19 model based feature vector:

11.1.1. Random Forest Classifier:

```
print(str(sum(classifier_rf_pred==y_test1)/rftest.shape[0]*100)+"% Accuracy")
```

63.67%

In this case, we are able to see that the accuracy obtained is 63.67%.

11.1.2. Logistic Regression:

```
print(str(sum(logistic_regression_pred==y_test1)/bottleneck_test.shape[0]*100)+"% Accuracy")
```

60.34%

In this case, we are able to see that the accuracy obtained is 60.34%.

11.1.3. MLP Classifier:

```
print(str(sum(MLP_pred==y_test1)/bottleneck_test.shape[0]*100)+"% Accuracy")
```

61.27%

In this case, we are able to see that the accuracy obtained is 61.27%.

To summarise the accuracies for all the models, the beneath table is given:

Model	Accuracy
Random Forest Ensemble	63.67%
Logistic Regression	60.34%
MLP Classifier	61.27%

11.2. ResNet50 model based feature vector:

11.2.1. Random Forest Classifier:

```
print(str(sum(classifier_rf_pred==y_test1)/rftest.shape[0]*100)+"% Accuracy")
```

61.34%

In this case, we are able to see that the accuracy obtained is 61.34%.

11.2.2. Logistic Regression:

```
print(str(sum(logistic_regression_pred==y_test1)/bottleneck_test.shape[0]*100)+"% Accuracy")
```

59.08%

In this case, we are able to see that the accuracy obtained is 59.08%.

11.2.3. MLP Classifier:

```
print(str(sum(MLP_pred==y_test1)/bottleneck_test.shape[0]*100)+"% Accuracy")
```

60.25%

In this case, we are able to see that the accuracy obtained is 60.25%.

To summarise the accuracies for all the model the beneath table is given:

Model	Accuracy
Random Forest Ensemble	61.34%
Logistic Regression	59.08%
MLP Classifier	60.25%

From this above tables, we can see that VGG19 is better than ResNet50 to extract features from our Dataset. And the best model that gave the best accuracy is Random Forest Ensemble.

12.CONCLUSION

The division of deep learning has been used successfully in many EEG activities, the burden of mental functioning, goals on the stage of sleep, the power associated with the event, and the functions of perceiving emotions. The design of these depths network studies differ significantly from input construction and network construction.

Many public databases are analysed in many studies, which have allowed us to directly compare the performance of categories based on their design. Generally, CNN's, RNN's, and DBN outperform other types of depth networks, such as SAEs and MLPNN's. In addition, CNN's best done using signal values or (spectrogram) images as input, and DBN has done very well when using it signal values or features are listed as input.

We also discussed in-depth network recommendations for each type of work. This recommendation diagram is provided with hope that it will guide the delivery of deeper education in EEG datasets in future research. Hybrid designs that include convolutional layers with repetitive or limited layers Boltzmann's equipment showed promise with precision in stages and transmission learning compared to conventional designs.

We commend an in-depth study of these compounds, especially the number and layout of various layers including RBM's, repetitive layers, convolutional layers, and layers that are fully connected. Without network design, We also encourage further research to compare how deep networks translate raw EEG and denoised raw, as this has not yet happened.

13.FUTURE SCOPE

From our project, we can see that the various models have tried to minimize the error rate and the best error rate is 36%. But it is still not sufficient though it reaches one of the state of the art accuracy. For further improvement, we can use other pre-processing techniques and make use of the other nodes that we didn't include in our project.

Also the temporal part of the several spectrums are much larger meaning that we are dealing with a session of above 15 min. Within this 15 min, the seizure where the EEG signal gets deviated from being normal happens for a small interval of time. For that we could check in which part the abnormality begins most of the times and try to check only that period of time and use our model to classify.