Grasp and Lift Prediction using EEG signals

Adam Alavi
University of Pennsylvania
adam2318@seas.upenn.edu

Anshul Wadhawan
University of Pennsylvania
anshulw@seas.upenn.edu

William C Francis
University of Pennsylvania
willcf@seas.upenn.edu

Abstract

In this project, we used machine learning methods to detect hand movements like grasping, lifting or replacing an object using EEG data, which can be used to control robotic prosthetic arms. We experimented with three data pre processing techniques for EEG signals: butter low pass filtering, wavelet denoising and stacking butter low pass filters. Further, we experimented with different classification models: Support Vector Machines, Logistic Regression, Random Forest, Linear Discriminant Analysis and Quadratic Discriminant Analysis. We tried 3 different weighted ensembles of these traditional models as well. We also used a Convolutional Neural Network to fit our training data and that method ended up giving us the best testing accuracy.

1 Motivation

Many people suffer from paralysis as a result of damage to their nervous system, especially the spinal cord, or due to Amyotrophic Lateral Sclerosis (ALS). With the loss of motor functions, these individuals are forced to be dependent on others for fulfilling their daily needs and care. Unfortunately, this leads to a lower quality of life for them. Restoring a patient's ability to perform basic activities of daily life with a brain-computer interface has been a critical problem statement for the past couple of decades. Currently, the methods available to neurologically disabled people to control external prosthetic limbs include invasive methods or they are expensive or high-risk. Through this project, we aim to propose a non-invasive method that allows the neurologically disabled to control their prosthetics using Machine Learning, Deep Learning or an ensemble of both. Our model would identify when a hand is grasping, lifting, and replacing an object using EEG data that was taken from healthy subjects as they performed these activities. Studying the correlation between EEG signals and hand movements with Machine Learning would help design a BCI device that would give patients with neurological disabilities the ability to move through the world with greater autonomy and improve their quality of life.

2 Related work

The paper (Luciw et al., 2014) talks about the idea of extracting signals related to object manipulation from EEG recordings in humans seems reasonable given that even basic motor tasks engage large parts of the human cortex(Ehrsson et al., 2000). It is, however, not known how much information can actually be decoded from EEG. Specifically, it is unclear to what extent it is possible to extract signals useful for monitoring and control of manipulation tasks, for instance, to control an upper limb prosthetic device to generate a power grasp or a pinch grasp involving the thumb and index finger. While successful EEG decoding of reaching trajectories has been reported(Bradberry et al., 2010), this claim is controversial(Antelis et al., 2013).

3 Dataset

In order to collect this data set, 12 subjects performed 10 series of trials where every series contains 30 trials. During every trial, EEG brain signals were recorded from the subjects while they performed grasp-and-lift operations. The training set contains the first 8 series for each subject. The test set contains the 9th and 10th series.
For each Grasp and Lift series, there are 6 events that take place in this order: HandStart, FirstDigitTouch, BothStartLoadPhase, LiftOff, Replace and BothReleased. Each time frame is given a unique id column according to the subject, series, and frame to which it belongs. The actions are one-hot encoded and thus, the value for an action is one or

zero depending on whether the corresponding event has occurred within ±150ms (±75frames). You can find the dataset here[1]. You can also see an example trial[2] to understand the data collection process.

4 Problem Formulation

This is a supervised learning classification problem as the EEG signals in the data set are labelled with the corresponding action that was performed when the signal was captured. The task at hand is to see a new signal and classify it into one of the 6 categories so that this model can be used for prosthetic limbs in real time to capture the EEG signal from the patient's brain and perform the appropriate task. As we have time-series data in the case of EEG signals, it will certainly be beneficial to use information from the past while making a prediction for the current time frame. Another approach is to look at the entire signal over an action as one whole entity and work with convolutional neural nets to create a model capable of template matching that exploits the properties of translational invariance.

5 Methods

This section describes our proposed methodology which is separated into four sections: data collection, data visualisation, data pre-processing and the models.

5.1 Data Collection process

There are 12 subjects. Each subject records 10 series of experiments. Each series includes 6 different actions: HandStart, FirstDigitTouch, BothStartLoadPhase, LiftOff, Replace, BothReleased. Series 1 to 8 are used for training whereas series 9 and 10 are used for testing on Kaggle.

5.2 Data Visualisation

Figure 1 denotes a sample EEG signal from three different electrodes out of 32 where HandStart is denoted by the red region, FirstDigitTouch is denoted by the purple region, BothStartLoadPhase is denoted by the black region, LiftOff is denoted by the green region, Replace is denoted by the yellow region and BothReleased is denoted by the blue region.

[1] https://www.kaggle.com/c/grasp-and-lift-eeg-detection/data

[2] https://tinyurl.com/2p4b37th

5.3 Data Pre-processing

This section explains the various data pre-processing techniques like butter low pass filtering, wavelet denoising and stacking butter low pass filters which were experimented in this project.

5.3.1 Butterworth Low Pass Filter

A Butterworth filter (Zhongshen, 2007) is a type of signal processing filter designed to have a frequency response as flat as possible in the passband. Hence the Butterworth filter is also known as "maximally flat magnitude filter". The frequency response of the Butterworth filter is flat in the passband and roll-offs towards zero in the stopband. The rate of roll-off response depends on the order of the filter. In our case, we have used a low-pass filter which passes all the frequencies below a certain threshold. We need to use complex higher-order filters to achieve the characteristic near to the ideal characteristic. If you increase the order of the filter, the number of cascade stages with the filter is also increased. But in practice, we cannot achieve Butterworth's ideal frequency response because it produces excessive ripple in the passband.

5.3.2 Wavelet Denoising

Because wavelets (Dautov and Özerdem, 2018) localize features in your data to different scales, you can preserve important signal or image features while removing noise. The basic idea behind wavelet denoising, or wavelet thresholding, is that the wavelet transform leads to a sparse representation for many real-world signals and images. This means that the wavelet transform concentrates signal and image features in a few large-magnitude wavelet coefficients. Wavelet coefficients which are small in value are typically noise and you can "shrink" those coefficients or remove them without affecting the signal or image quality. After you threshold the coefficients, you reconstruct the data using the inverse wavelet transform.

5.3.3 Stacked Butter Low Pass Filters

We observed that passing the entire training data through a Butter low-pass filter improved the score on almost every model. To further increase the number of features upon which the models could train, we create new features by stacking multiple Butter Low Pass filters on the features. In our implementation, we use 10 cutoff frequencies to create 10 different Butterworth low-pass filters that

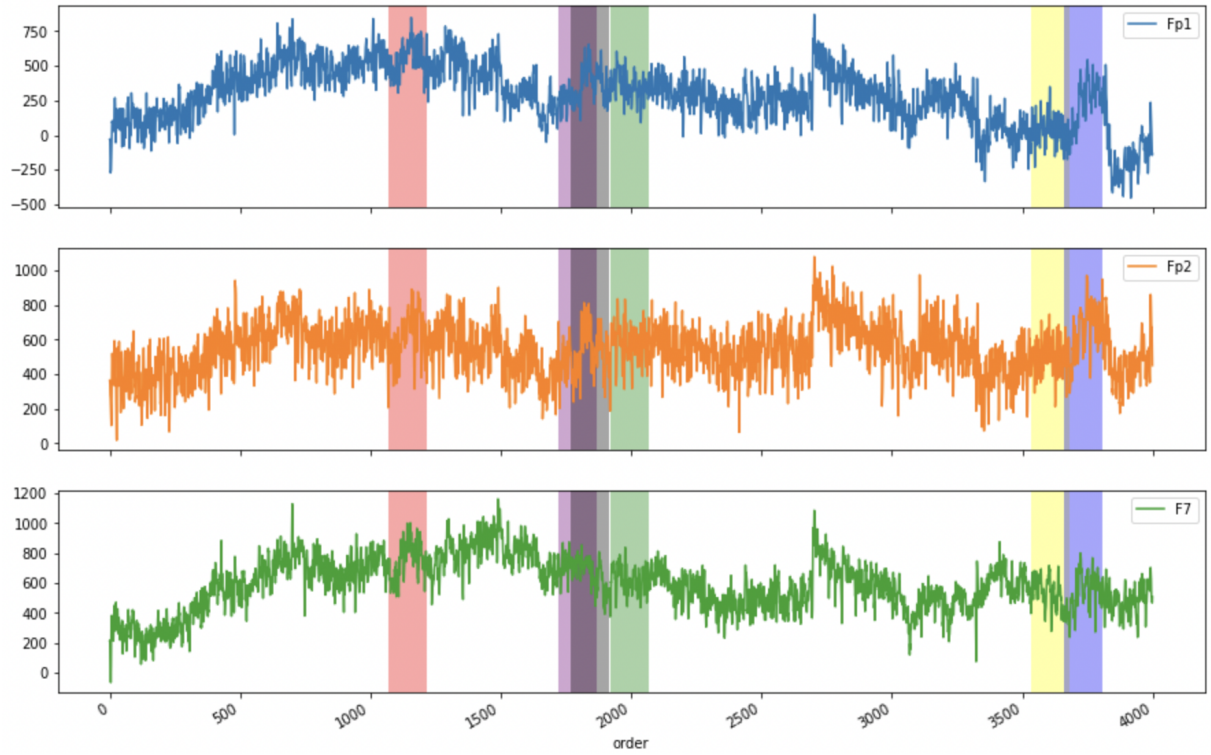

Figure 1: Data Visualisation

can be stacked together with their squares to create 20 features out of just one feature.

5.4 Models

This section puts forward different traditional machine learning models as well as deep learning based convolutional neural networks.

5.4.1 Logistic Regression

Logistic regression (Peng et al., 2002) is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. In regression analysis, logistic regression (or logit regression) is estimating the parameters of a logistic model (a form of binary regression). Mathematically, a binary logistic model has a dependent variable with two possible values, such as pass/fail which is represented by an indicator variable, where the two values are labeled ”0” and ”1”. In the logistic model, the log-odds (the logarithm of the odds) for the value labeled ”1” is a linear combination of one or more independent variables (”predictors”); the independent variables can each be a binary variable (two classes, coded by an indicator variable) or a continuous variable (any real value).

5.4.2 Linear Discriminant Analysis

Linear discriminant analysis (LDA), normal discriminant analysis (NDA), or discriminant function analysis is a generalization of Fisher’s linear discriminant, a method used in statistics and other fields, to find a linear combination of features that characterizes or separates two or more classes of objects or events. The resulting combination may be used as a linear classifier, or, more commonly, for dimensionality reduction before later classification.LDA is closely related to analysis of variance (ANOVA) and regression analysis, which also attempt to express one dependent variable as a linear combination of other features or measurements. LDA is also closely related to principal component analysis (PCA) and factor analysis in that they both look for linear combinations of variables which best explain the data. LDA explicitly attempts to model the difference between the classes of data.

5.4.3 Quadratic Discriminant Analysis

Quadratic discriminant analysis (QDA) (Ghojogh and Crowley, 2019) is closely related to linear discriminant analysis (LDA), where it is assumed that the measurements from each class are normally distributed. Unlike LDA however, in QDA there is no assumption that the covariance of each of the classes is identical. When the normality assump-

tion is true, the best possible test for the hypothesis that a given measurement is from a given class is the likelihood ratio test.

5.4.4 Random Forest Classifier

Random forests (Sarica et al., 2017) or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time. For classification tasks, the output of the random forest is the class selected by most trees. For regression tasks, the mean or average prediction of the individual trees is returned. Random decision forests correct for decision trees' habit of overfitting to their training set.

5.4.5 Support Vector Machines

Given a set of training examples, each marked as belonging to one of two categories, an SVM (Hearst et al., 1998) training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting). SVM maps training examples to points in space so as to maximise the width of the gap between the two categories. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

5.4.6 Principal Component Analysis with Random Forest

The principal components (Mishra et al., 2017) of a collection of points in a real coordinate space are a sequence of p unit vectors, where the i-th vector is the direction of a line that best fits the data while being orthogonal to the first i-1 vectors. Principal component analysis (PCA) is the process of computing the principal components and using them to perform a change of basis on the data, sometimes using only the first few principal components and ignoring the rest. Later, the shortlisted components are used as features for the random forest classifier.

5.4.7 Ensemble Modeling

Ensemble modeling is a process which outputs an outcome, either by using many different modeling algorithms or using different training data sets. The ensemble model then aggregates the prediction of each base model and results in once final prediction for the unseen data. The motivation for using ensemble models is to reduce the generalization

Hyperparameter	Value
Logistic Regression	
Penalty	L2
C	1.0
Linear Discriminant Analysis	
Solver	SVD
Quadratic Discriminant Analysis	
Regularization	0
Random Forest Classifier	
#Trees	200
Max Depth	7
Criterion	Entropy
Support Vector Machines	
C	1.0
Kernel	RBF
Degree	3
PCA with RF	
#Components	10
LDA, RF, LR Ensemble	
Weight for LDA	0.4
Weight for RF	0.25
Weight for LR	0.35
LDA, LR Ensemble	
Weight for LDA	0.5
Weight for LR	0.5
RF, LR, QDA, LDA Ensemble	
Weight for RF	0.25
Weight for LR	0.3
Weight for QDA	0.15
Weight for LDA	0.3
CNN	
Epochs	1
Learning Rate	0.002
Optimiser	Adam
Chunk Size	1000
Batch Size	1024

Table 1: Hyperparameter Values

error of the prediction. As long as the base models are diverse and independent, the prediction error of the model decreases when the ensemble approach is used.

We tried ensembling the above mentioned models and came up with these three combination of models:

- LDA + Random Forest + Logistic Regression
- LDA + Logistic Regression
- RF + Logistic Regression + QDA + LDA

5.4.8 Convolutional Neural Networks

CNNs (Gu et al., 2017) are regularized versions of multi-layer perceptrons. Multi-layer perceptrons usually mean fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The "full connectivity" of these networks make them prone to overfitting data. Typical ways of regularization, or preventing overfitting, include: penalizing parameters during training (such as weight decay) or trimming connectivity (skipped connections, dropout, etc.) CNNs take a different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble patterns of increasing complexity using smaller and simpler patterns embossed in their filters. Therefore, on a scale of connectivity and complexity, CNNs are on the lower extreme. CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns to optimize the filters (or kernels) through automated learning, whereas in traditional algorithms these filters are hand-engineered. This independence from prior knowledge and human intervention in feature extraction is a major advantage.

6 Experiments and Results

We experimented with different data preprocessing techniques like butter low pass filters, wavelet denoising, and stacking of butter low pass filters. Followed by pre-processing the data, we passed the final features to the previously mentioned machine learning models. The hyperparameter values for all the above mentioned models, that worked best for us, have been provided in Table 1. The performance of these models using stacked butter low pass filters as the data preprocessing technique, has been depicted in Table 2. The models' performances with varying data preprocessing techniques have been summarised in Figure 11.

Using Stacked Butter Low Pass for feature engineering was the biggest breakthrough in our work that improved the score of almost every model by 5 to 10%. When PCA was run on Random Forest, we noticed that the score decreased proportional to the decrease in number of components. This signified that lesser the data we give to the models, the worse they performed. This was hypothesised to be due to overfitting and was confirmed when increasing the tree depth increased training score but decreased testing score. To tackle this issue, feature engineering was first carried out on three random features. New features were engineered by stacking data that were passed through the butter low pass filter. More features were added by appending their squared terms. This gave a big bump in the test score as expected. To further improve our feature engineering technique, we chose the three most important features, namely FP1, FP2, and T7, instead of random features. Feature importance was calculated from the components after running PCA on the training data. Engineering new features based on features that most affected the predictions ensured that the newly added features contributed well to producing better predictions. Using the three most important features instead of random features improved the test scores by 2-3%

The Grasp-and-Lift Kaggle competition uses ROC AUC score on the test data to score the submissions. To benchmark our models before scoring it on the test data, we plotted the ROC AUC curves on training data and analysed how each model performed on the training data. Figures 3, 4, 5, 6, 7, 8, 9, 10, and 12, are the ROC curves for the different machine learning algorithms on the LiftOff class.

6.1 Code Walkthrough

Here, we provide a code walkthrough of the CNN implementation that we have used for our problem. The **read_csv** function handles the data import and it grabs all the $.csv$ files that are required for the training and testing loops. It imports the data from the 32 electrodes that make up the 32 features and it also gets the labels for the 6 actions that are one-hot encoded. We use the last two series of every subject for validation. The **resample_data** function is used to create chunks of data of size 1000 and this is necessary as every row in the data

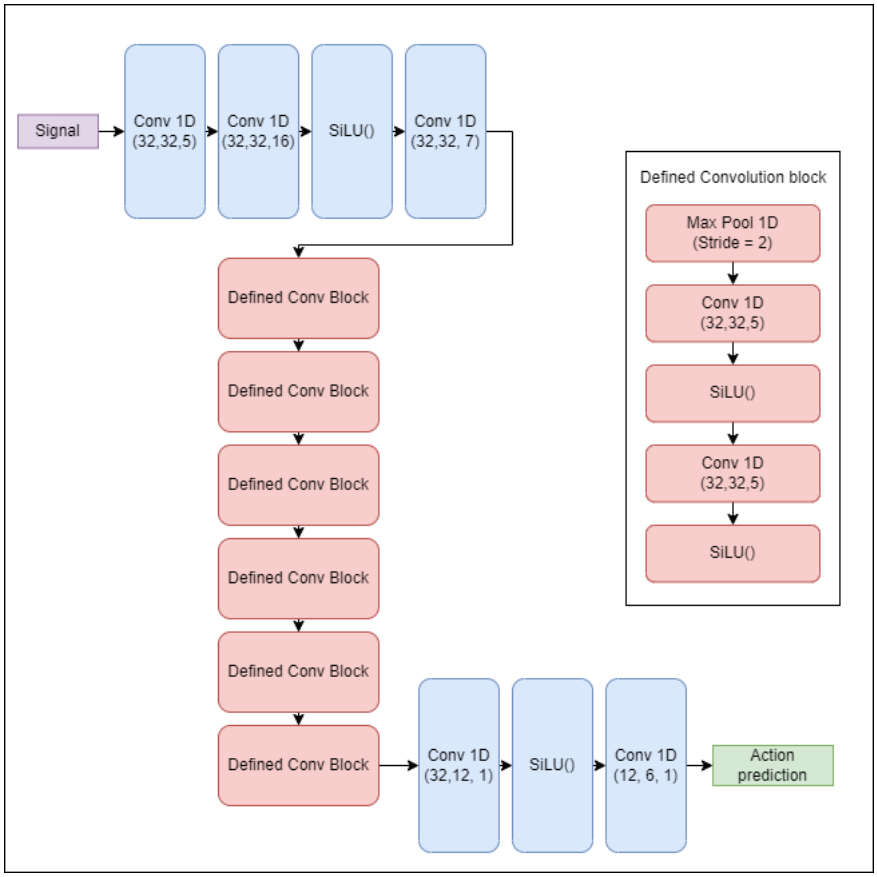

Figure 2: CNN Model Architecture

Model	Train ROC AUC	Test ROC AUC
LR	0.9048	0.8740
LDA	0.9100	0.8859
QDA	0.9023	0.7210
RF	0.9347	0.8105
SVM	0.8564	0.8263
PCA + RF	0.8067	0.6526
LDA + RF + LR	0.9601	0.8850
LDA + LR	0.9123	0.8865
RF + LR + QDA + LDA	0.9631	0.8618
CNN	0.9573	0.9235

Table 2: Models performances

captures a span of around 75ms and that means that the data in rows next to each other is highly correlated. We also discard a lot of chunks in this function as most of them do not have any event occuring in them and this really helped the model as a lot of noise was discarded in this way and it was computationally cheaper as well. We ended up eliminating almost 50% of the data by using this method. We also standardize the data in the **EEGSignalDataset** class.

We have used a deep neural network architecture as shown in Figure 2. for the problem here as the data is extremely complex and the model needs depth to extract the right features that will be useful for the classification task. The model contains 23 layers including the activation functions (Sigmoid Linear Unit), 1-D convolutions and Max pooling layers (Stride 2). The parameters of all the layers have been printed in a code block below. When it comes to hyperparameters, we have used an Adam

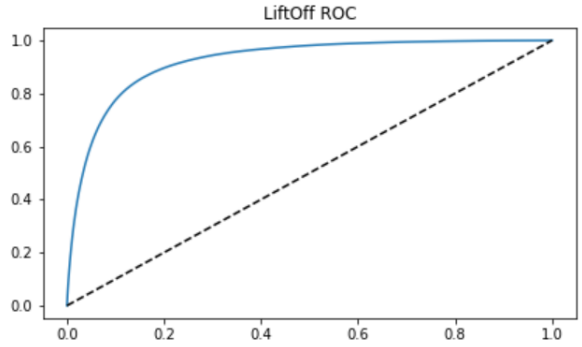

Figure 3: ROC Curve for Logistic Regression

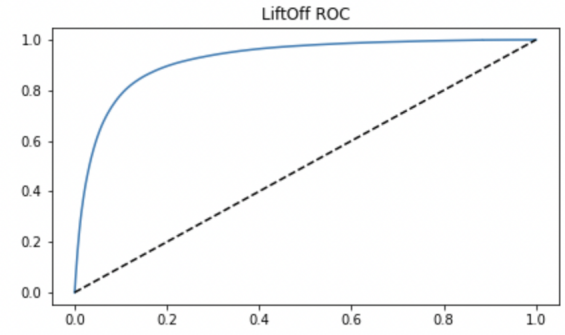

Figure 4: ROC Curve for LDA

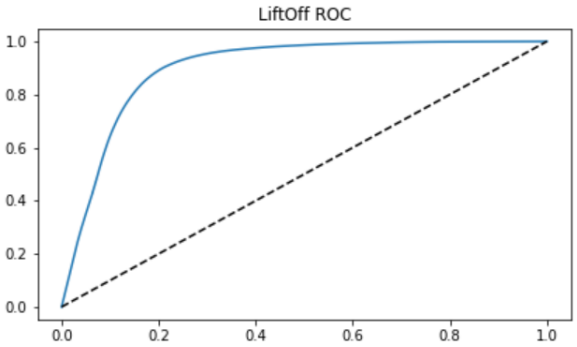

Figure 5: ROC Curve for QDA

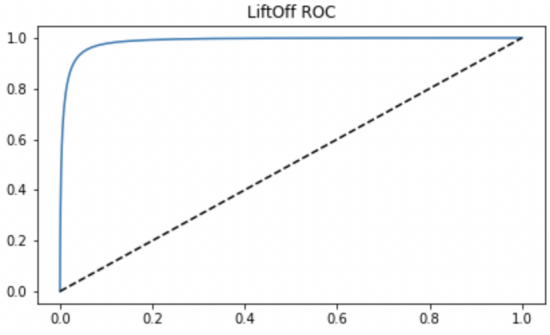

Figure 6: ROC Curve for RF

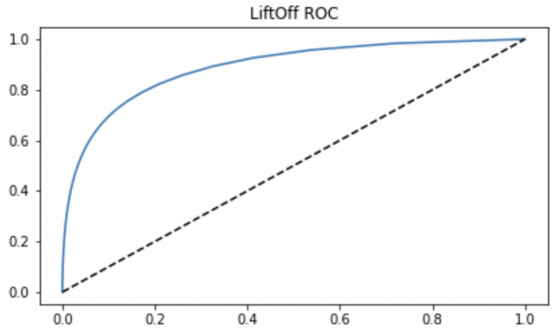

Figure 7: ROC Curve for PCA with RF

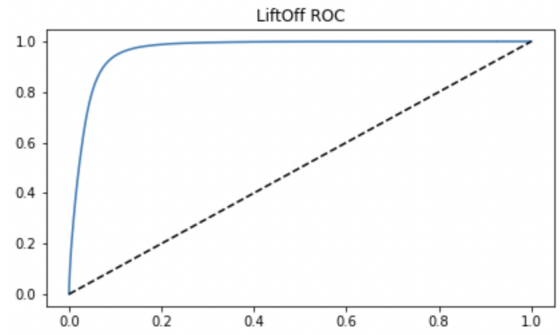

Figure 8: ROC Curve for LDA+RF+LR

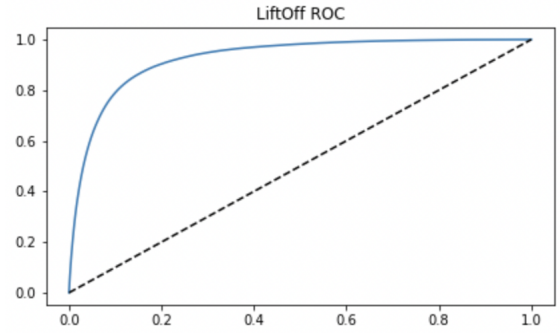

Figure 9: ROC Curve for LDA+LR

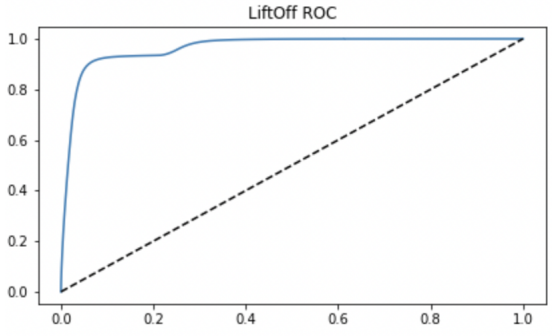

Figure 10: ROC Curve for RF+LR+QDA+LDA

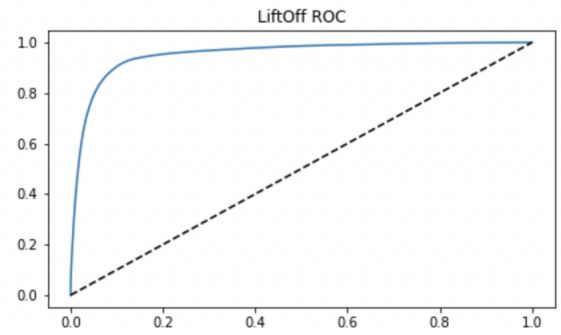

Figure 12: ROC Curve for CNN

optimizer with a $lr = 0.002$ loss and inary Cross Entropy loss. We have only trained the model for one epoch as that was sufficient due to the large size of the data. After training, we plot the ROC curves for the validation dataset and also iterate through the test data to create the $.csv$ file that is needed for a submission on Kaggle. We need to do this to check our model on the testing set as the labels for testing set are not available to us and are hidden on Kaggle. We get a final accuracy of 92.35% on the testing set.

The area under the training ROC curves indicate how well the model performed on the training data. This does not tell us how well it performed on the test data. However, by combining the training ROC scores and the testing ROC scores, we can estimate how well the models generalized and how badly some models overfitted.

7 Conclusion and Discussion

From the above experiments, we make the following conclusions:

- Stacked Lowpass filter performs better in all the models implemented. This is because it has 92 features - 32 raw features and 60 augmented features.
- SiLU activation function gives a better accuracy compared to the other activation functions in the CNN model.
- Weighted ensembling based on the accuracy of individual models gave a better score compared to the individual models. At the same time, ensembling poor models with the good models gave a poor score.
- Stacking features based on their feature importance obtained from PCA gave a better score than stacking random features.
- Deep Learning did a better job on the dataset because, in our opinion, the depth of the model enables it to learn complex features from the data that are relevant to the predictions.

The histogram in Figure 11. compares the test scores of the 31 models we implemented. The best performing model is a 22-layer CNN. This could be attributed to the fact that the data is fairly complex

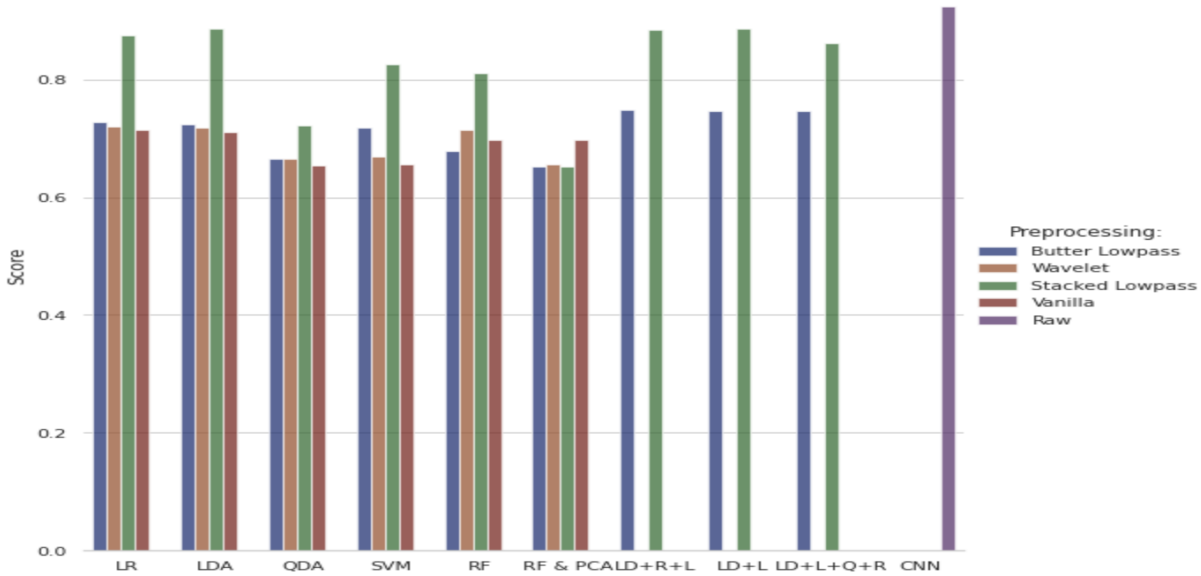

Figure 11: Graphical Comparison of Performance

and the CNN model was able to find the underlying features that contribute to the predictions. The second best set of models belong to the feature engineered models due to the increased data. We observed that the machine learning models overfit a lot when the data provided is insufficient. Thus, when extra features are generated with butter low pass filter, the models do a much better job at generalizing on the test set. We can also observe that the ensemble models give a much better performance than the individual models. This is because we have carried out weighted ensembling by giving a bigger weight for the better performing individual model. For example, logistic regression almost gets the highest weight as the model outperforms almost every other individual model.

Comparing the two pre-processing methods used, we can observe that the butter low pass filter performed slightly better than Wavelet transform. This could be because wavelet transform removes a part of the EEG signal that is pivotal in making predictions. Also considering that the models are hungry for data and are on the verge of overfitting, it is intuitive that filtering out too much data results in a poor test score.

An important observation from the histogram is that the Random Forest with PCA performs poorly. This is because the data in itself is insufficient for most machine learning models. Decreasing the dimensionality with PCA only makes it worse. It is even more peculiar that the vanilla model performs better than the ones with pre-processing. This can again be attributed to the scarcity of data and the overfitting that results from it. When use any kind of filtering technique with PCA, the training data looses the supposed "noise" from the already shrunk set of features. Shrinking the data beyond a point thus adversely affects the model and this causes PCA to fail with 10 components and even worse with PCA and signal filtering with butter lowpass and wavelet transform.

As part of future work, different pre processing techniques can evaluated with recurrent based models like RNNs-LSTMs. Further hyperparameter tuning can be done on the stacked butterpass feature engineering technique. The model can be deployed onto a prosthetic arm simulation and improvised for failing cases.

References

Javier Antelis, Luis Montesano, Ander Ramos-Murguialday, Niels Birbaumer, and Javier Minguez. 2013. On the usage of linear regression models to reconstruct limb kinematics from low frequency eeg signals. *PloS one*, 8:e61976.

Trent J. Bradberry, Rodolphe J. Gentili, and José L. Contreras-Vidal. 2010. Reconstructing three-dimensional hand movements from noninvasive electroencephalographic signals. *Journal of Neuroscience*, 30(9):3432–3437.

Çiğdem Polat Dautov and Mehmet Siraç Özerdem. 2018. Wavelet transform and signal denoising using wavelet method. In *2018 26th Signal Processing and Communications Applications Conference (SIU)*, pages 1–4.

H.Henrik Ehrsson, Oran Westling, Roland Johansson, H. Henrik, Anders Fagergren, Tomas Jonsson, S. Johansson, and Hans Forssberg. 2000. Cortical activity in precision- versus power-grip tasks: An fmri study. *J. Neurophysiol.*, 83.

Benyamin Ghojogh and Mark Crowley. 2019. Linear and quadratic discriminant analysis: Tutorial.

Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Li Wang, Gang Wang, Jianfei Cai, and Tsuhan Chen. 2017. Recent advances in convolutional neural networks.

M.A. Hearst, S.T. Dumais, E. Osuna, J. Platt, and B. Scholkopf. 1998. Support vector machines. *IEEE Intelligent Systems and their Applications*, 13(4):18–28.

Matthew Luciw, Ewa Jarocka, and Benoni Edin. 2014. Multi-channel eeg recordings during 3,936 grasp and lift trials with varying weight and friction. *Scientific data*, 1:140047.

Sidharth Mishra, Uttam Sarkar, Subhash Taraphder, Sanjoy Datta, Devi Swain, Reshma Saikhom, Sasmita Panda, and Menalsh Laishram. 2017. Principal component analysis. *International Journal of Livestock Research*, page 1.

Joanne Peng, Kuk Lee, and Gary Ingersoll. 2002. An introduction to logistic regression analysis and reporting. *Journal of Educational Research - J EDUC RES*, 96:3–14.

Alessia Sarica, Antonio Cerasa, and Aldo Quattrone. 2017. Random forest algorithm for the classification of neuroimaging data in alzheimer's disease: A systematic review. *Frontiers in Aging Neuroscience*, 9:329.

Li Zhongshen. 2007. Design and analysis of improved butterworth low pass filter. In *2007 8th International Conference on Electronic Measurement and Instruments*, pages 1–729–1–732.