Illinois Institute of Technology

CS584: Machine Learning Final Project

Final PROJECT

Classification of Harmful Brain Activities in EEG Signals with Deep Learning

Student Name	CWID
Varshith Chandramukhi	A20563861
Nikhitha Chintakuntla	A20561244
Nandini Singireddy	A20561245

Table of Contents

Classification of Harmful Brain Activities in EEG Signals with Deep Learning

1. Introduction

Physicians utilize several instruments, such as tongue depressors and stethoscopes, to treat their patients. When a patient is in critical condition, doctors utilize electroencephalography (EEG) to look for seizures and other brain activity that may lead to brain damage. EEG monitoring is now done entirely by manual by trained neurologists. Although very useful, this labor-intensive procedure is a significant bottleneck. In addition to being time-consuming, manual review of EEG recordings can be costly, prone to errors caused by weariness, and have reliability problems across reviewers—even experts—when done by various people.

Our project is focused on automating EEG analysis to aid doctors and brain researchers with detecting seizures and other types of brain activity that can cause brain damage. This would help in more accurate and quicker treatment. We have focused on six patterns of interest in EEG signals for this project namely : seizure (SZ), generalized periodic discharges (GPD), lateralized periodic discharges (LPD), lateralized rhythmic delta activity (LRDA), generalized rhythmic delta activity (GRDA), or “other”.

Over a decade the Deep Neural Networks have shown significant impact in vision applications such as image classification, segmentation, object detection, etc. We deployed State-of-the-art neural networks and discussed the performance in the coming sections. Figure 1. Gives a pictorial overview of our project.

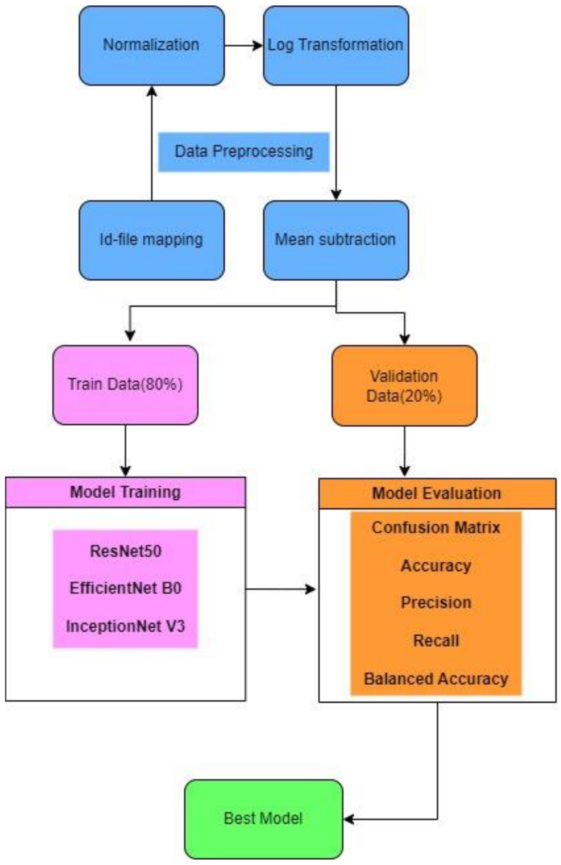


Figure 1. Proposed architecture.

2.1. Dataset Used:

We used the dataset introduced in the Harvard Medical School Research Code Competition which focuses on detecting and classifying seizures and other types of harmful brain activity. There are six patterns of interest for this competition: seizure (SZ), generalized periodic discharges (GPD), lateralized periodic discharges (LPD), lateralized rhythmic delta activity (LRDA), generalized rhythmic delta activity (GRDA), or “other”.

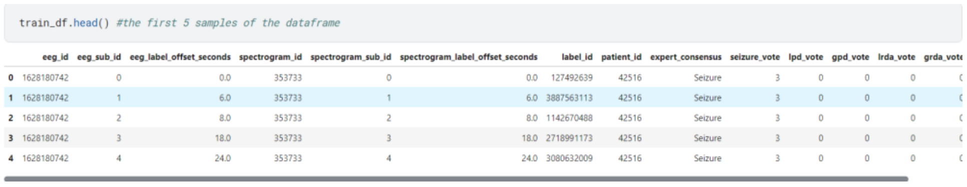
The EEG segments used in this competition have been annotated, or classified, by a group of experts. In some cases experts completely agree about the correct label. On other cases the experts disagree. call segments where there are high levels of agreement “idealized” patterns. Cases where ~1/2 of experts give a label as “other” and ~1/2 give one of the remaining five labels, we call “proto patterns”. Cases where experts are approximately split between 2 of the 5 named patterns, we call “edge cases”.

Train set:

Metadata for the train set. The expert annotators reviewed 50 second long EEG samples plus matched spectrograms covering 10 a minute window centered at the same time and labeled the central 10 seconds.

There are three files belonging to Train set:

1. Train.csv has the information about:
    - i. eeg_id - A unique identifier for the entire EEG recording.
    - ii. eeg_sub_id - An ID for the specific 50 second long subsample this row's labels apply to.
    - iii. eeg_label_offset_seconds - The time between the beginning of the consolidated EEG and this subsample.
    - iv. spectrogram_id - A unique identifier for the entire EEG recording.
    - v. spectrogram_sub_id - An ID for the specific 10 minute subsample this row's labels apply to.
    - vi. spectogram_label_offset_seconds - The time between the beginning of the consolidated spectrogram and this subsample.
    - vii. label_id - An ID for this set of labels.
    - viii. patient_id - An ID for the patient who donated the data.
    - ix. expert_consensus - The consensus annotator label. Provided for convenience only.
    - x. [seizure/lpd/gpd/lrda/grda/other]_vote - The count of annotator votes for a given brain activity class. The full names of the activity classes are as follows: lpd: lateralized periodic discharges, gpd: generalized periodic discharges, lrd: lateralized rhythmic delta activity, and grda: generalized rhythmic delta activity .

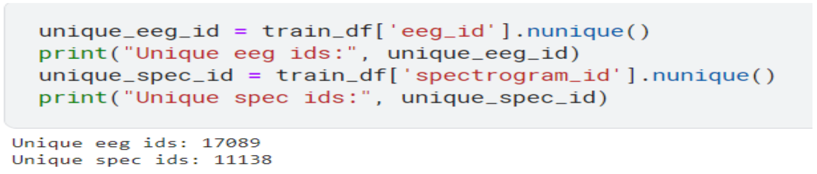


```
train_df.head() #the first 5 samples of the dataframe
```

	eeg_id	eeg_sub_id	eeg_label_offset_seconds	spectrogram_id	spectrogram_sub_id	spectrogram_label_offset_seconds	label_id	patient_id	expert_consensus	seizure_vote	lpd_vote	gpd_vote	lrda_vote	grda_vote
0	1628180742	0	0.0	353733	0	0.0	127492639	42516	Seizure	3	0	0	0	
1	1628180742	1	6.0	353733	1	6.0	3887563113	42516	Seizure	3	0	0	0	
2	1628180742	2	8.0	353733	2	8.0	1142670488	42516	Seizure	3	0	0	0	
3	1628180742	3	18.0	353733	3	18.0	2718991173	42516	Seizure	3	0	0	0	
4	1628180742	4	24.0	353733	4	24.0	3080632009	42516	Seizure	3	0	0	0	

Figure 2. Overview of train.csv data.

There are a total of 106800 observations in the train.csv, but there are only 11138 unique spectrogram_ids and 17089 unique eeg ids. Many of these samples overlapped and have been consolidated.



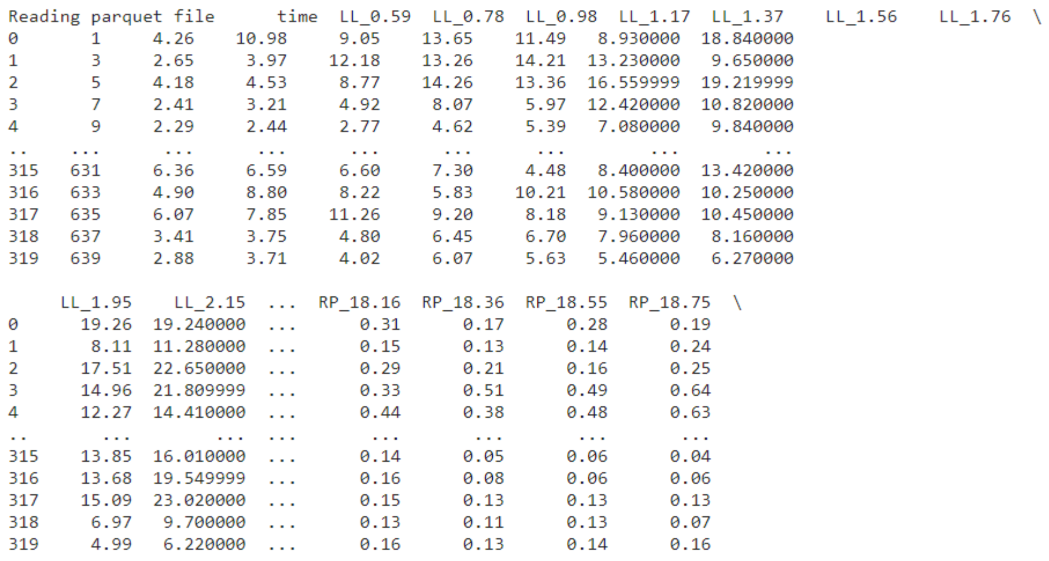
```
unique_eeg_id = train_df['eeg_id'].nunique()
print("Unique eeg ids:", unique_eeg_id)
unique_spec_id = train_df['spectrogram_id'].nunique()
print("Unique spec ids:", unique_spec_id)
```

```
Unique eeg ids: 17089
Unique spec ids: 11138
```

Figure 3. Number of unique eeg and spectrogram ids.

Understanding parquet files:

Both Spectrogram and EEG signals are in parquet format. Figure 4. Gives an overview of how the files look like. We understand the rows represent the time and the columns represent magnitude or intensity of the signal at different frequencies. So, the data is in Time X Frequency format. But, in general, spectrograms are typically arranged in Frequency X Time format. So, we performed transpose for further analysis.

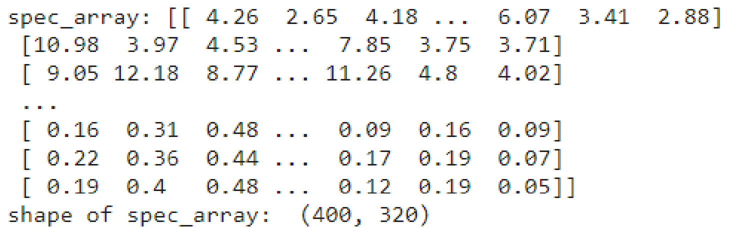


```
Reading parquet file      time  LL_0.59  LL_0.78  LL_0.98  LL_1.17  LL_1.37    LL_1.56    LL_1.76  \
0        1     4.26    10.98     9.05    13.65    11.49   8.930000  18.840000
1        3     2.65     3.97    12.18    13.26    14.21  13.230000   9.650000
2        5     4.18     4.53     8.77    14.26    13.36  16.559999  19.219999
3        7     2.41     3.21     4.92     8.07     5.97  12.420000  10.820000
4        9     2.29     2.44     2.77     4.62     5.39   7.080000   9.840000
..     ...      ...      ...      ...      ...      ...        ...        ...
315    631     6.36     6.59     6.60     7.30     4.48   8.400000  13.420000
316    633     4.90     8.80     8.22     5.83    10.21  10.580000  10.250000
317    635     6.07     7.85    11.26     9.20     8.18   9.130000  10.450000
318    637     3.41     3.75     4.80     6.45     6.70   7.960000   8.160000
319    639     2.88     3.71     4.02     6.07     5.63   5.460000   6.270000

     LL_1.95    LL_2.15  ...  RP_18.16  RP_18.36  RP_18.55  RP_18.75  \
0      19.26  19.240000  ...      0.31      0.17      0.28      0.19
1       8.11  11.280000  ...      0.15      0.13      0.14      0.24
2      17.51  22.650000  ...      0.29      0.21      0.16      0.25
3      14.96  21.809999  ...      0.33      0.51      0.49      0.64
4      12.27  14.410000  ...      0.44      0.38      0.48      0.63
..       ...        ...  ...       ...       ...       ...       ...
315    13.85  16.010000  ...      0.14      0.05      0.06      0.04
316    13.68  19.549999  ...      0.16      0.08      0.06      0.06
317    15.09  23.020000  ...      0.15      0.13      0.13      0.13
318     6.97   9.700000  ...      0.13      0.11      0.13      0.07
319     4.99   6.220000  ...      0.16      0.13      0.14      0.16
```

Figure 4. Reading Parquet file.

For further data preprocessing, we have converted the file type from parquet to numpy array.
Now the shape of the array (400, 320) where 400 and 320 denote frequency and time.



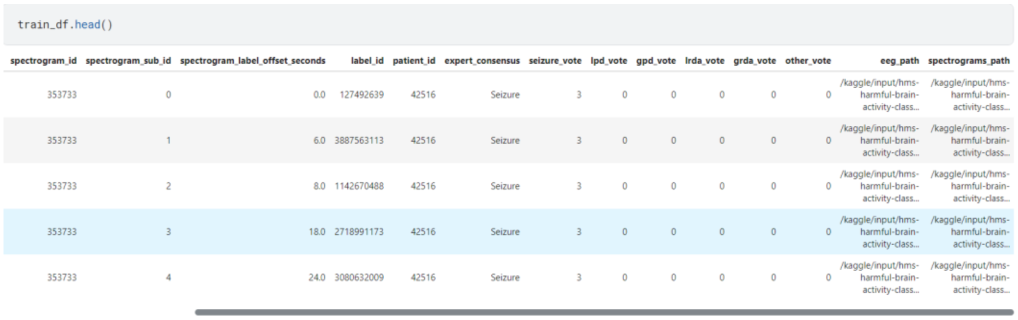
```
spec_array: [[ 4.26  2.65  4.18 ...  6.07  3.41  2.88]
 [10.98  3.97  4.53 ...  7.85  3.75  3.71]
 [ 9.05 12.18  8.77 ... 11.26  4.8   4.02]
 ...
 [ 0.16  0.31  0.48 ...  0.09  0.16  0.09]
 [ 0.22  0.36  0.44 ...  0.17  0.19  0.07]
 [ 0.19  0.4   0.48 ...  0.12  0.19  0.05]]
shape of spec_array:  (400, 320)
```

Figure 5. Spectrograms in numpy array format.

2.2. Data Preprocessing:

a. Id-file mapping.

The spectrogram and eeg files are stored in different parent folders with their respective ids in the file names. We have extracted the file paths using glob library and created id mapping function to map the ids with file names and saved it in the dataframe.



```
train_df.head()
```

spectrogram_id	spectrogram_sub_id	spectrogram_label_offset_seconds	label_id	patient_id	expert_consensus	seizure_vote	lpd_vote	gpd_vote	lrda_vote	grda_vote	other_vote	eeg_path	spectrograms_path
353733	0	0.0	127492639	42516	Seizure	3	0	0	0	0	0	/kaggle/input/hms-harmful-brain-activity-class...	/kaggle/input/hms-harmful-brain-activity-class...
353733	1	6.0	3887563113	42516	Seizure	3	0	0	0	0	0	/kaggle/input/hms-harmful-brain-activity-class...	/kaggle/input/hms-harmful-brain-activity-class...
353733	2	8.0	1142670488	42516	Seizure	3	0	0	0	0	0	/kaggle/input/hms-harmful-brain-activity-class...	/kaggle/input/hms-harmful-brain-activity-class...
353733	3	18.0	2718991173	42516	Seizure	3	0	0	0	0	0	/kaggle/input/hms-harmful-brain-activity-class...	/kaggle/input/hms-harmful-brain-activity-class...
353733	4	24.0	3080632009	42516	Seizure	3	0	0	0	0	0	/kaggle/input/hms-harmful-brain-activity-class...	/kaggle/input/hms-harmful-brain-activity-class...

Figure 6. Dataframe with extracted spectrogram and eeg file paths.

b. Normalization

From Figure 5 we can see the values of the pixels are not in the range of (0, 1). Considering the size of the data, we performed normalization to reduce the stress on the computation. This also ensures that all input features have a similar scale, which can improve the performance and stability of deep learning models

c. Log transformation

This boosts the contrast while lessening the impact of anomalies. It facilitates improved spectrogram feature visualization.

d. Mean subtraction

This centers the data around zero which helps in improving the stability of model by reducing the bias.

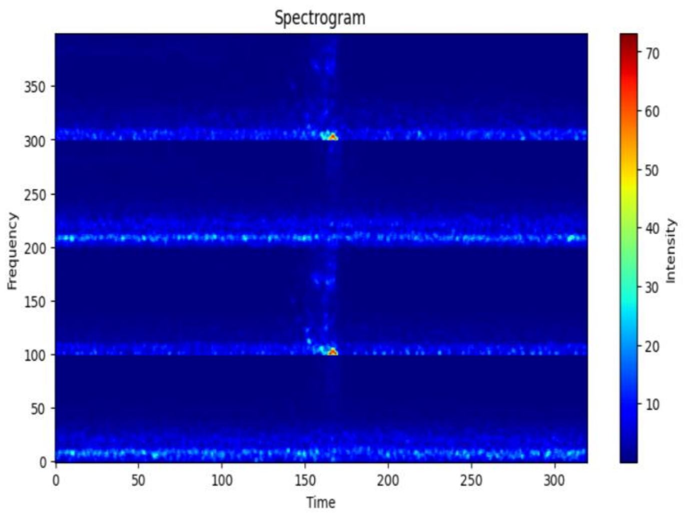


Figure 7. Visual representation of spectrogram before preprocessing

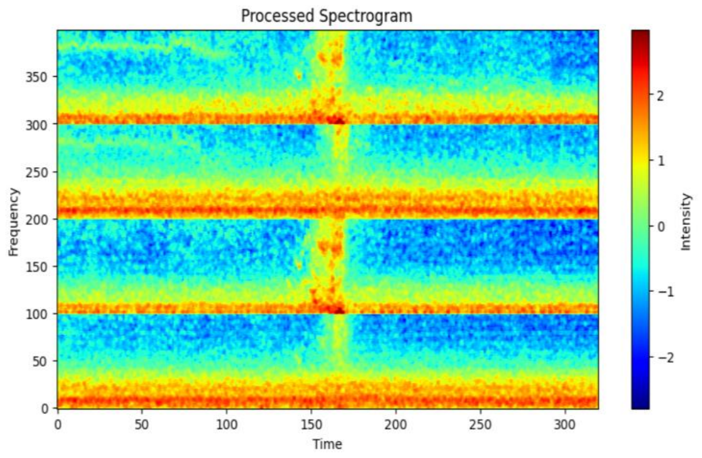


Figure 8. Visual representation of spectrogram after preprocessing.

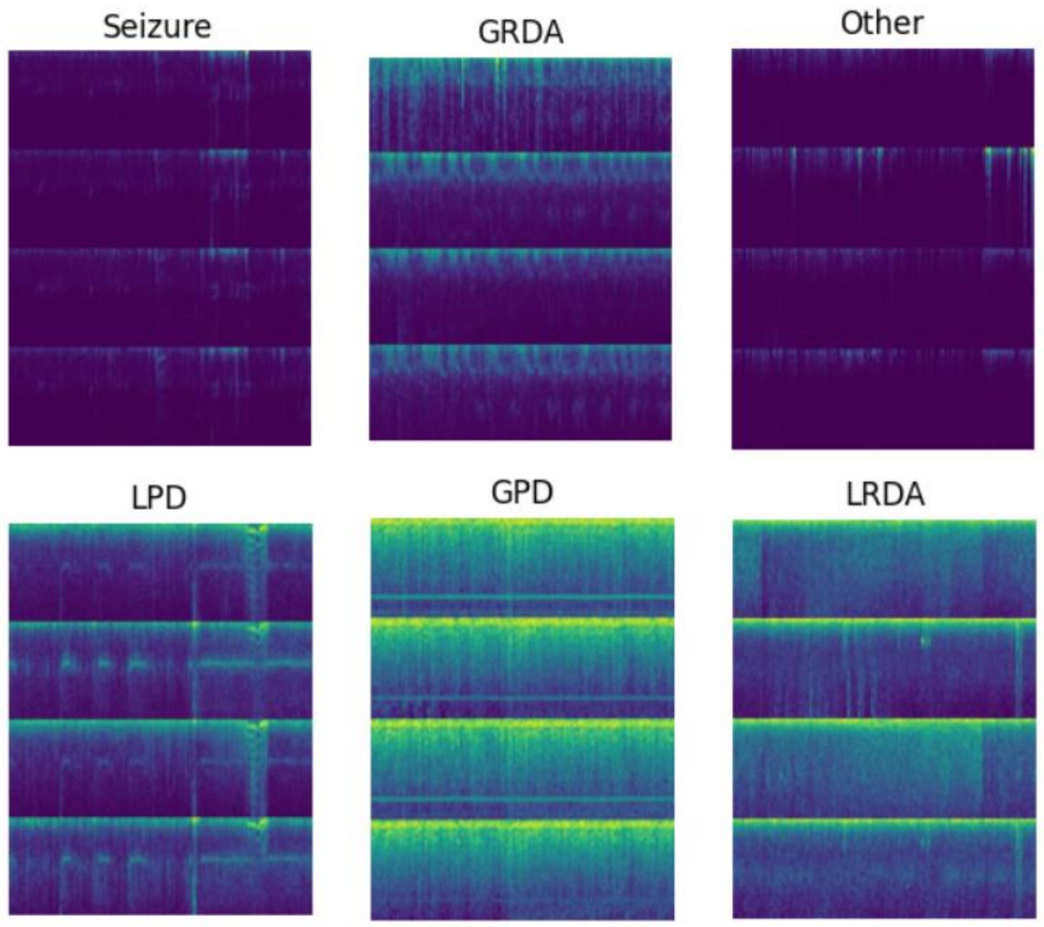
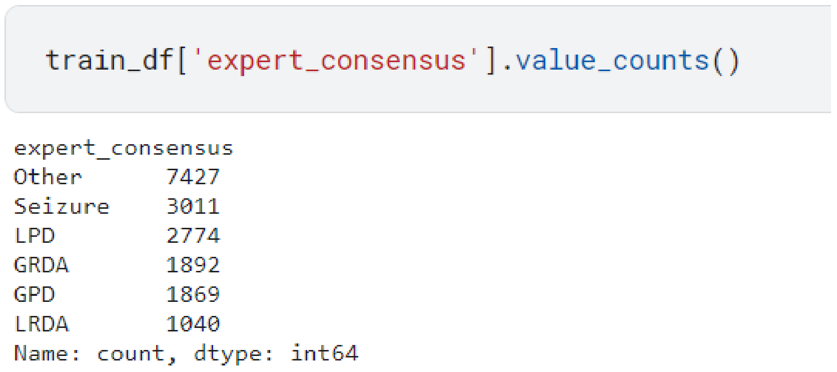


Figure 9. Visual representation of spectrograms from each class after preprocessing.

Spectrograms provide a visual representation of the frequency content of EEG signals over time. They capture important frequency patterns that are often crucial for analyzing and interpreting brain activity. Considering the size of the dataset, Hence we have considered only spectrograms to classify the brain disorder. There are total of 18013 unique combinations of eeg_id, spectrogram_id, and expert_consensus. We have constructed the dataset with all possible unique combinations to prevent any loss of information. Figure 10 gives overview of each class distribution in the dataset which clearly indicates the **dataset is unbalanced.** We choose appropriate performance metrics to avoid any bias caused by unbalanced dataset.

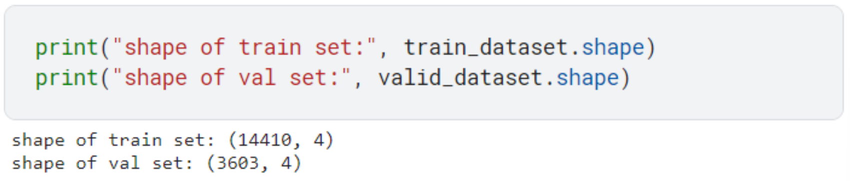


```
train_df['expert_consensus'].value_counts()
```

```
expert_consensus
Other      7427
Seizure    3011
LPD        2774
GRDA       1892
GPD        1869
LRDA       1040
Name: count, dtype: int64
```

Figure 10. Overview of individual class distribution in the dataset.

Train – Validation Split: We split the dataset into train and validation with 80:20. The size of train and validation sets can be observed in Figure 11.



```
print("shape of train set:", train_dataset.shape)
print("shape of val set:", valid_dataset.shape)
```

```
shape of train set: (14410, 4)
shape of val set: (3603, 4)
```

Figure 11. Size of train and validation data.

3. Model training

3.1. Model Selection:

a. ResNet

Deep neural networks, well the word itself has “deep” which explains how deep it could be sometimes. In huge depth neural networks, the back propagation of errors faces challenges while training which is called as **Vanishing gradient**. This challenge arises due to the diminishing magnitude of gradients as they traverse through numerous layers during back propagation. Essentially, the gradients become almost negligible, so they are useless for changing the weights of the early layers. As a result, these layers are unable to acquire meaningful representations, which hinders the network's overall convergence.

To encounter this problem ResNet [1] has come with a technique called Skip Connections. Skip connections are intentionally inserted to promote the flow of gradients across the network. They have deployed skip connections in various stages to enable the network to efficiently learn both local and global features effectively.

b. EfficientNet

The compound scaling method in EfficientNet[2] is a strategy for scaling up a base network architecture to achieve improved performance without a significant increase in **computational cost**. This method involves scaling the network in three key dimensions: depth, width, and resolution.

a. Depth scaling involves increasing the depth of the network by adding more layers or repeating base blocks. Deeper networks can capture more complex features, but they are also more computationally expensive.
b. Width scaling refers to increasing the number of channels in each layer of the network. This allows the network to capture a greater variety of features at each spatial location.
c. Resolution scaling involves increasing the input image resolution. Higher-resolution images contain more detailed information, which can help the network learn complex patterns.

By carefully balancing these three scaling dimensions, EfficientNet achieves better performance compared to traditional methods that only scale one dimension (such as depth or width) while keeping the others fixed.

c. **InceptionNet**

One of the main challenges in designing deep neural networks is the **trade-off between depth and width**. Deeper networks can capture more complex features but are computationally expensive, while wider networks are more efficient but may not capture as much complex information.
The Inception[3] architecture addresses this trade-off by using inception modules. These modules allow the network to have both depth and width by using parallel convolutions with different kernel sizes. This enables the network to capture features at multiple scales, from fine details to more global patterns, without significantly increasing the number of parameters or computational cost.

We have used the above 3 pre-trained networks which are trained on the State-of-the-art ImageNet dataset. We used the weights generated from training on ImageNet dataset as building blocks, since training such huge depth models completely on our dataset could cause over-fitting. All three models—the custom EfficientNet, InceptionNet, and ResNet—have an input size of 256 × 256 pixels in order to promote consistency and efficient model training. The dataset has been preprocessed and loaded with ease using the ImageDataGenerator. TensorFlow was used as the backend, and the Keras library was utilized to build the deep learning models. We used pre-trained neural networks from tensorflow.keras.applications. For training the deep learning models we took help of Kaggle and its GPU (GPU P100) support.

3.2. Hyper-parameters Used.

Hyper-parameter	**Parameter value**
Loss function	Categorical cross entropy
Optimizer	Adam
Dropout %	30
Batch size	32
Learning Rate	0.0001 – 0.00001 (Dynamic)
Activation	Relu(CNN layers), Softmax(output layer)
Image shape	(256, 256, 3)

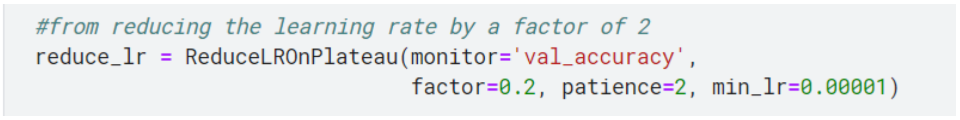
Table 1. Hyper-parameters used in the training phase.

The choice of batch size plays a crucial role in neural network training, as it dictates the number of samples processed tin each iteration before the model updates its weights. The model becomes more susceptible to individual data points with smaller batch sizes, which raises the possibility of overfitting. However, a large batch size necessitates strong GPU support. We chose a reasonable batch size of 32 in order to achieve balance and meet the training requirements of the models that were presented.

3.3 Model implementation.

The following techniques were implemented while training the deep learning models.

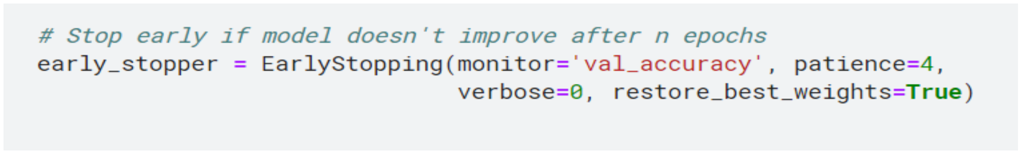
a. **ReduceLRonPlateau:** It is a technique used during training to adjust the learning rate when a metric has stopped improving. It helps to fine-tune the model by reducing the learning rate when the model's performance plateaus, allowing it to potentially escape local minima and converge more effectively**.**



```
#from reducing the learning rate by a factor of 2
reduce_lr = ReduceLROnPlateau(monitor='val_accuracy',
                              factor=0.2, patience=2, min_lr=0.00001)
```

Here we monitored validation accuracy, and with patience = 2, the learning rate reduces by a factor of 0.2 and the minimum learning rate is set to be 10^-5.

b. **Early Stopping**: It is a technique used during training to prevent overfitting. It stops training the model when a monitored metric has stopped improving, thus preventing the model from learning noise in the training data and improving generalization to unseen data.



```
# Stop early if model doesn't improve after n epochs
early_stopper = EarlyStopping(monitor='val_accuracy', patience=4,
                              verbose=0, restore_best_weights=True)
```

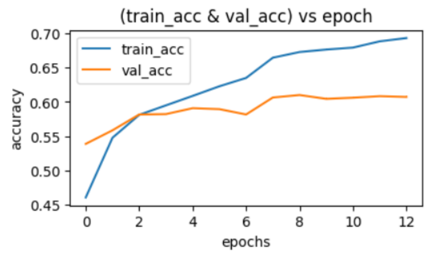
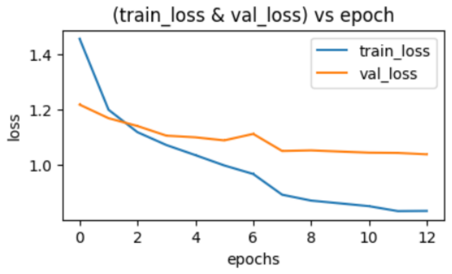
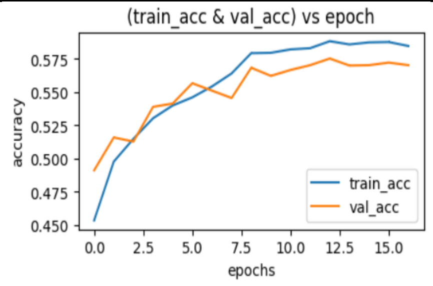
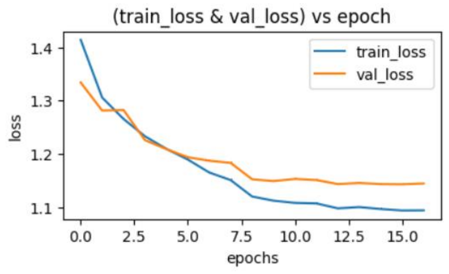
Here we monitor the validation accuracy and if it doesn't improve for 4(patience =4) epochs then training stops.

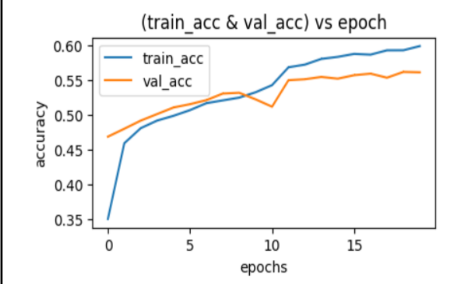
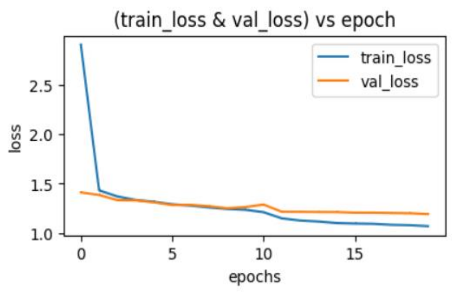
c. **Dropout layers:** Dropout layers are a type of regularization technique used to reduce overfitting in neural networks. During training, dropout randomly sets a fraction of input units to zero, which helps to prevent units from co-adapting too much, leading to a more robust network. We set the value of dropout to be 30% which randomly drops 30% of parameters in that particular layer.

4. Results and observations

Table 2 provides an overview of the accuracy and loss progression for both the training and validation sets. In ResNet50, the train and validation curves move apart with each epoch being trained on. This indicates the model is being overfit to the train data but unable to improve its performance to validation data. The training has stopped after 14th epoch which explains absence of improvement in accuracy from 10th epoch. Early stopping is rightly used here. Whereas in EfficientNet B0, both train and validation go along together which tells there is no overfitting to the train data but the model has stopped improving after 12-13 epochs. Coming to InceptionNet, the model has trained all 20 epochs which explains there is improvement throughout the training phase and no presence of overfitting.

Table 2. Training accuracy and loss of Deep CNN models over 20 epochs.

S.No	Model	Accuracy	Loss
1	ResNet 50		
2	EfficientNet B0		

3	**InceptionNet V3**		

In evaluating the performance of our Deep Convolutional Neural Network (DCNN) models, various performance metrics were employed and are thoroughly examined in this section.

a. Confusion matrix

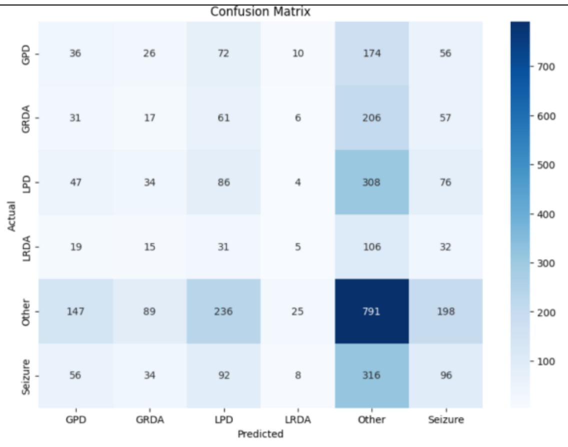
The confusion matrix is a visual representation that helps assess the performance of a machine learning model by presenting the counts of true positive (TP), true negative (TN), false positive (FP), and false negative (FN) predictions.

TP (True Positive): The model correctly predicts the positive class. TN (True Negative): The model correctly predicts the negative class. FP (False Positive): The model incorrectly predicts the positive class. FN (False Negative): The model incorrectly predicts the negative class.

Confusion Matrix	**Actual 0**	**Actual 1**
Predicted 0	A	B
Predicted 1	C	D

A = True Negatives, B = False Negatives, C = False Positives, D = True Positives

Table 3. Confusion matrix of the Deep CNN models for the validation sets.

Type of dataset and model	Confusion matrix
Custom model (Butterfly dataset)	

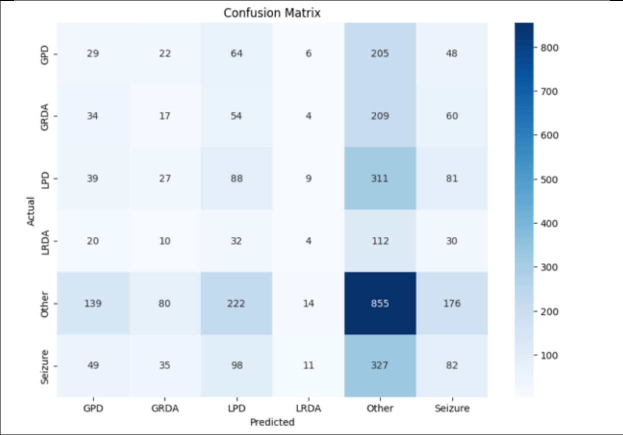
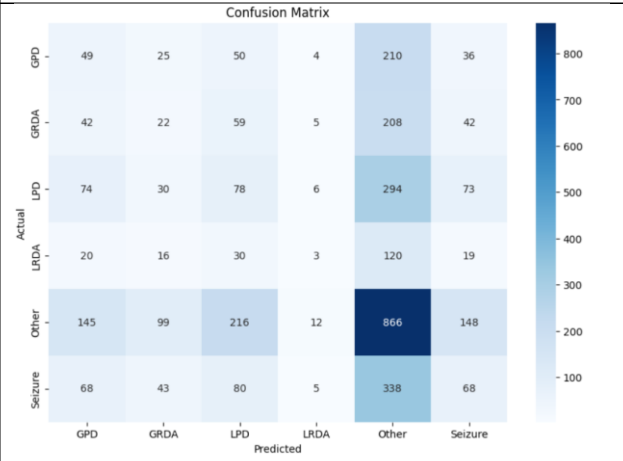
Custom model (Facial Expression Recognition)	
ResNet –50 [Fine-tuned] (Butterfly dataset)	

Table 3 explains the effect of having unbalanced dataset. Since the size of the class "other" is too big compare to other classes, the model tend to falsely predicts all the remaining classes as "other". Hence we have explored other metrics to have the correct performance analysis.

b. Accuracy

Accuracy in machine learning measures how well a model performs by gauging its correct predictions against the total predictions available in the dataset.

c. Precision

Precision is a metric that evaluates the accuracy of a model's predictions for the positive class. It is calculated by dividing the number of true positives (correctly predicted positive instances) by the sum of true positives and false positives (incorrectly predicted positive instances). Precision values range from 0 to 1, where a higher value indicates better precision. A precision score of 1 means that all predicted positive instances are correct, while a score of 0 indicates that none of the predicted positive instances are correct.

d. Recall

Recall gauges the actual true positives among all the true samples in the dataset. Precision, which varies between 0 and 1, indicates better precision with higher values.

e. Balanced accuracy

It calculates the average accuracy of each class, giving each class equal weight regardless of its size. This metric is particularly useful when the classes are skewed, ensuring that the evaluation is not biased towards the majority class.

Table 4. Performance analysis of DCNNs.

Neural Network	**Accuracy(%)**	**Precision(%)**	**Recall(%)**	**Balanced Accuracy(%)**
ResNet50	**61.004**	17.14	17.23	16.86
EfficientNet B0	57.535	17.21	17.59	16.86
InceptionNet V3	56.14	**18.45**	**17.81**	**17.33**

Table 4. gives us the complete understanding of how the models have performed and how unbalanced dataset has effected the deep learning models. In case of accuracy, ResNet50 has given the best results with accuracy of 61.004% and InceptionNet V3 has given the least accuracy of 56.14%. But in the rest of the metrics InceptionNetV3 has proven to be the best model compared to rest of them. Though InceptionNet too has failed to give significant results, but it gave us understanding that metrics such as **precision, recall and balanced accuracy can be used in the presence of unbalanced datased**.

The unbalanced datasets led to biased results for some of the DCNN models. Hence balancing the dataset by applying different data augmentation techniques could provide more robust models.

References

[1] He, K., Zhang, X., Ren, S. and Sun, J., 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

[2] Tan, M. and Le, Q., 2019, May. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning* (pp. 6105-6114). PMLR.

[3] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A., 2015. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1-9).