Alcoholism Classification using Deep Learning

Sneha Bahl

Research School of Computer Science, Australian National University
u7006861@anu.edu.au

Abstract. In this paper, we are using deep learning technique for the binary image classification task to detect if a subject is alcoholic or non-alcoholic. We have the EEG signal data as images along with their labels. We build various models consisting of different depths ad architectures. The main idea used is Convolutional Neural Networks (CNN) layers, which have been proved to work very efficiently for image classification tasks. We first built a base model and then further improved it by adding more convolution layers, batch-norm and dropout layers. We tested various model architectures and got to a testing accuracy of 97% while the base model could only reach to 81%.

Keywords: Deep learning, EEG, binary classification, Convolutional Neural Networks

1 Introduction

EEG data is being used a lot these days for classification [1, 7, 10] in areas like Neuromarketing, studying epilepsy, sleeping patterns etc. EEG basically records electrical activity in the brain by using sensors on the subject's scalp. This helps get a truer response than the subjects just filling up a survey or verbally responding to prompts as that leads to a bias and modifications in what one actually feels. EEG helps track changes in the response signals within fractions of a second. In this paper, we are trying to solve a binary classification problem on the UCI EEG dataset [10-11, 13]. We wish to classify a subject/person as being alcoholic or non-alcoholic based on the EEG signal images we have. We plan to first build a base model to solve the problem and then try to build a deep network using convolutional layers.

We explored the dataset to find the distribution of data between classes and visualize what the images looked like. We performed an analysis on the data and applied normalization and fixed the class-imbalance using data augmentation techniques on the class with low samples. We built a base model with Convolutional layers [3, 5, 10] and modified it to get a deeper model and added layers like batch normalization and dropout to improve the performance of the model. The insight behind each such layer is discussed in the Method section.

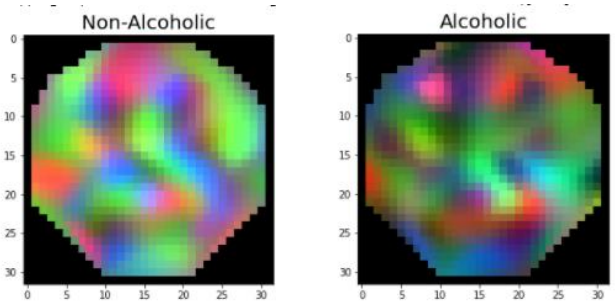
The next section covers the details about the dataset along with the pre-processing techniques used. The third section covers the method which includes the idea behind using Convolutional neural networks (CNNs) and other important techniques which we used to build our models. The fourth section presents the results and discusses the insights. The final section concludes all findings and mentions the future scope of this work.

2 Dataset

The data we are using is the preprocessed UCI EEG dataset [10-11, 13]. EEG signals contain information about the electrical activity of the brain. The dataset was collected by recording the subject's brain activity response to different stimuli. The goal of data collection was to be able to identify alcoholism in different subjects. Given the nature of the problem, it's a great approach to use EEG signals, since one might really be reserved to express if he is alcoholic or not. So, we are using this data to build classification model using the EEG signals to find if someone is alcoholic or non-alcoholic. We are using the preprocessed form of the data which contains the EEG images, generated by a lot of operations and processing on the signal. It also has variables like the subject id, trial number, stimuli, the label etc. The label is 1 for an alcoholic subject and 0 for non-alcoholic.

2.1 Visualization

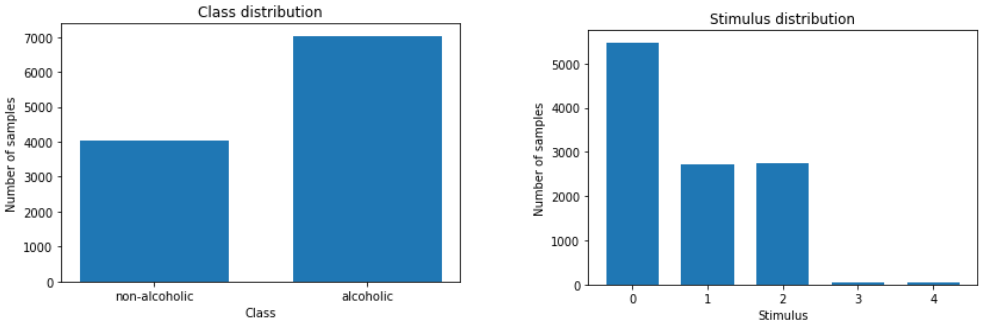
To get a deeper understanding of the dataset, we took a look at the class distribution and the images. The dataset contains 11057 RGB images of size (32 X 32 X 3). The values of the pixels ranged in (-1.85, 18.55). Below we can see one image from each class.



a. b.

Fig 1: *a.* Sample image for non-alcoholic class. *b.* Sample image for alcoholic class.

We can notice from the above that the non-alcohloic images show relatively morebrighter activtions than the alcoholic ones. Next, we found that the class distribution was not even. Around 65% of the data belonged the the 'alcoholic' class whereas only 35% of the data belonged to the 'non-alcoholic' class. We can also see that the samples are not evernly diftributed for various stimuli too. The maximum samples are for stimuli 1 and the minimum for stimuli 3 and 4. If using the stimuli variable we can group the samples for stumuli 3 and 4 into a single new stimuli or even club the samples for stimuli 2, 3 and 4 together.

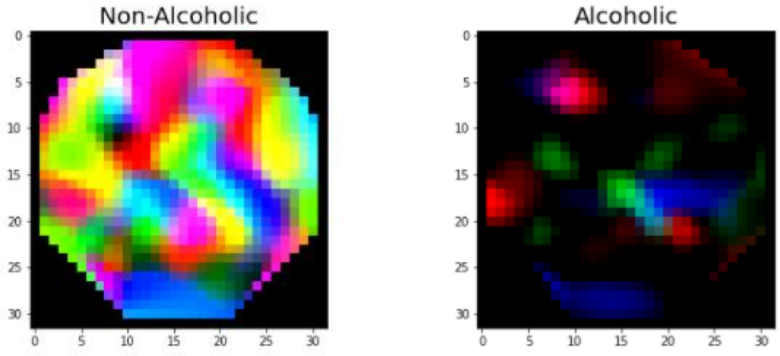


a. b.

Fig 2*: a.* The distribution of samples based on the class. ***b.*** The distribution of the samples based on the stimluli

2.2 Pre-processing

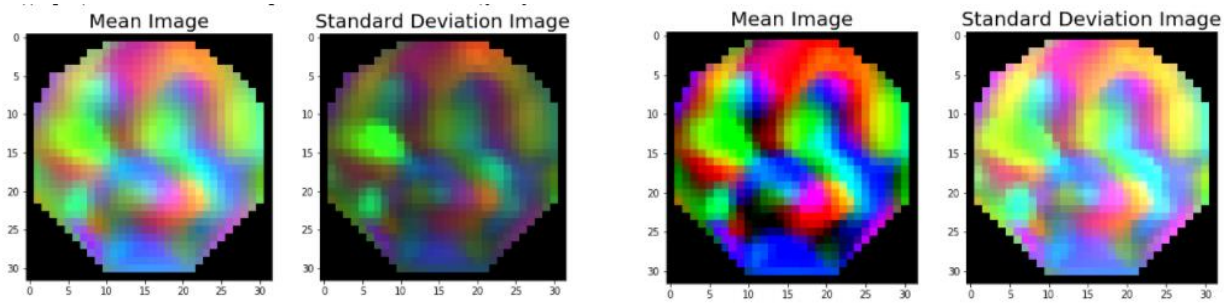
We decided not to use, the variables – 'trialnum', 'y_stimulus' and 'subjectid' based on the analysis in [2], since they had very low correlation to the label – 'y_alcoholic. The next pre-processing step that we performed was normalizing the image data. We normalized all the images using StandardScaler from the sklearn library. Below we can see the normalized images for the original images we saw above.



a. b.

Fig 3: *a.* Sample image from *Normalized* data for non-alcoholic class. *b.* Sample image from *Normalized* data for alcoholic class.

We can notice from Figure 4, that the images are less noisy and more clear after normalization. We also the mean and standard deviation of the image data before and after normalization visualized and can be seen below. We again observed sharper and less noisier data.



a. b.

Fig 4: *a.* Mean and Standard Deviation for *Original* data. *B.* Mean and Standard Deviation for *Normalized* data.

The next issue to deal with was the class imbalance problem. To deal with this, we generated around 3000 more images for the minority class i.e. the non-alcoholic class, with slight transformations on the original data. We used the ColorJitter transform from torchvision, to make changes to the lighting of the images. We set brightness factor to 0.2 and contrast to 0.6. This made very slight changes to the pixel colours in the images. The reason we chose only this particular transform was, on the basis of preliminary analysis which showed that each area is has a specific colour which is dominant and the sample only differ in the intensity, brightness and darkness of these colours. Using transforms like rotation, or random cropping could have led to bad data. These new generated images are added to the original ones and the whole set is shuffled. The new distribution can be seen below.

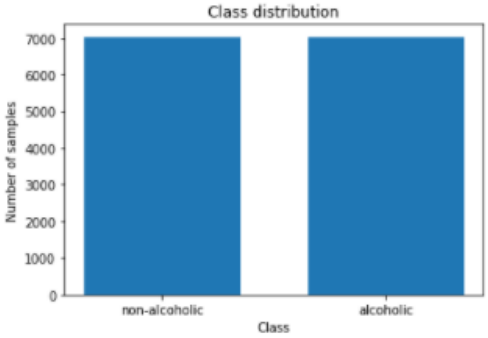


Fig 5: The distribution of samples based on the class after adding samples for minority class

3 Method

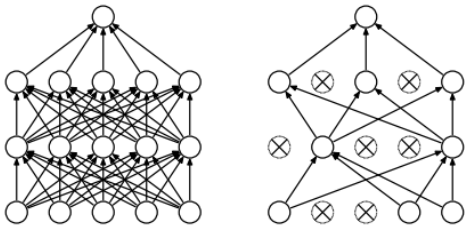
Deep Learning models learn much quicker as compared to artificial neural networks. These models we built converged within 30 epochs whereas models like in this [2] run for over 500 epochs to reach to a similar accuracy. Deep learning models provide results for complicated problems which simple models and traditional machine learning algorithms cannot but at the expense of computational resources.

Convolution Neural Networks- A convolutional neural network or CNN is a modified version of the multi-layer perceptron. The multi-layer perceptron is fully connected whereas in CNNs all the neurons of a layer are not connected to all the neurons of the next layer. This technique helps reduce the overfitting of a model. The convolution layer performs a convolution operation which is basically the dot product of the layer's input matrix along with the kernel. It is usually followed by an activation function like ReLU (Rectified Linear Unit). The LeNet5 [5] model started the typical and standard structure CNNs have that is stacked convolutional layer sometimes followed by maxpooling and normalization which is finally followed by some fully-connected layers. CNNs weren't used a lot due to their computationally intense nature but they came into more use after we were able to use them on GPUs which made the training speed 20 times faster.

These convolution operations generate feature maps which are inputted to the next layer. These feature maps are basically the learnings of the layer. The initial layers are known to learn simpler features like shape and edges. As the

network gets deeper, the model learns more detailed features like shape of nose and ear etc. in the example of face detection/classification. The hyperparameters in this layer like kernel size, padding stride etc. are very important since modifying them can result in good and poor feature and pattern learning. The use of Convolutional layers proved to be a breakthrough in imagery tasks and work around it gave many models like Vgg-16, AlexNet [3], Inception [8] and ResNeXt-50 [9] etc.

Dropout – Dropout [6] is a regularization technique which helps prevent overfitting in deep neural networks. The idea is to randomly drop some nodes along the connections during training. This helps in the network "memorizing" less and generalizing better. We can mention the probability of how a unit or node being dropped while training. This gives much better results than other regularization methods and significantly reduces overfitting. Below is an example of the dropout technique.



(a) Standard Neural Net (b) After applying dropout.

: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

Fig 6: Dropout. The image above has been taken from the original paper [6]

Batch Normalization- Normalizing the data not only improves the learning of the model but also helps speed up the learning process. Batch normalization works on batches of data and during training instead of the whole set and as a pre-processing step. It helps normalize the data over each channel of the input data. Using it within the network, helps normalize the data in every layer. This results in a much faster convergence of the model.

3.1 Architectures

We have tried 3 models and tested them with different hyperparameters. Given below is the detailed architecture of each of these models.

3.1.1 Base Model

Given below is the structure of the base model. Each layer is applied on the input tensor in a Sequential manner.

```
# Conv Layers
    Conv2d(in_channels =3, out_channels = 10, kernel_size=5),
    Max_pool2d(kernel_size = 2),
    ReLU(),
    Conv2d(10, 20, kernel_size=5),
    Max_pool2d(kernel_size = 2),
    ReLU(),
# Fully connected Layer
    Linear(500, 50),
    ReLU(),
    Dropout(),
    Linear(50, 2)
```

We also apply a log_softmax on the output from the final layer. This is why we use the Negative Log-Likelihood loss (NLL loss) during training. Using NLL loss helps us understand the model performance better too. The lower the loss the better the model and is known to work well for classification tasks.

3.1.2 Improved Model, M2 and M3

Model 2 and Model 3 have a very similar architecture. The only difference between them is the number of Convolutional blocks. For the ease of understanding, we'll just mention Model 3 and will specify which part we need to remove to get Model 2.

```
   # Conv Layer block 1
      Conv2d(in_channels=3, out_channels=32, kernel_size=3, padding=1),
      BatchNorm2d(32),
      ReLU(inplace=True),
      Conv2d(in_channels=32, out_channels=64, kernel_size=3, padding=1),
      ReLU(inplace=True),
      MaxPool2d(kernel_size=2, stride=2),

   # Conv Layer block 2
      Conv2d(in_channels=64, out_channels=128, kernel_size=3, padding=1),
      BatchNorm2d(128),
      ReLU(inplace=True),
      Conv2d(in_channels=128, out_channels=128, kernel_size=3, padding=1),
      ReLU(inplace=True),
      MaxPool2d(kernel_size=2, stride=2),
      Dropout2d(p=0.3),

   # Conv Layer block 3
      Conv2d(in_channels=128, out_channels=256, kernel_size=3, padding=1),
      BatchNorm2d(256),
      ReLU(inplace=True),
      Conv2d(in_channels=256, out_channels=256, kernel_size=3, padding=1),
      ReLU(inplace=True),
      MaxPool2d(kernel_size=2, stride=2),
      Dropout2d(p=0.3),

###### Comment this Conv 4 block for model 2/M2, but keep it for model 3(M3)
    # Conv Layer block 4
      Conv2d(in_channels=256, out_channels=512, kernel_size=3, padding=1),
      BatchNorm2d(512),
      ReLU(inplace=True),
      Conv2d(in_channels=512, out_channels=512, kernel_size=3, padding=1),
      ReLU(inplace=True),
      MaxPool2d(kernel_size=2, stride=2),

    # Fully connected layers

      Dropout(p=0.3),
     # uncomment the below layer for model 2 and comment the next one
      #Linear(4096, 1024, bias=True),
      Linear(2048, 1024, bias=True),
      ReLU(inplace=True),
      Linear(1024, 512, bias=True),
      ReLU(inplace=True),
      Dropout(p=0.3),
      Linear(512, 2)
```

Here also we apply a log_softmax on the output from the final layer. All the above layers are applied sequentially. The Dropout layers help with the overfitting problem. The details about training and hyperparameter settings are discussed for each model in the next section.

3.2 Training and Hyperparameters

We broadly have 3 models, the base model, M1 and 2 deeper and improved models M2 and M3. The detailed architectures for all three models were discussed above. We have split the data for both models into 70:15:15 corresponding to training data : testing : validation data. The data is also divided into batches using the Data Loader for all the three subsets. The batch size for training is 64 and 128 for both testing and validation. The performance metrics are an average of the performance over different batches. Given below in Figure 7, are the sample images from a batch of training data.

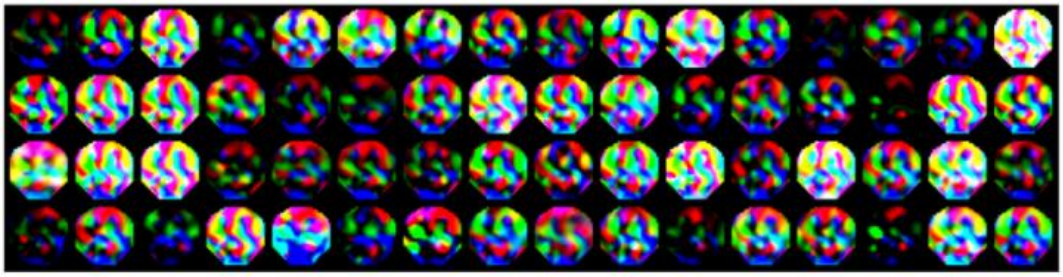


Fig 7- A batch from the training dataset

We also tried training the models with all 3 forms of the dataset namely, original, original-normalized and balanced-normalized. We are using the NLL loss for all the models along with the SGD optimizer. We also experimented with the Adam optimizer but that didn't seem to give us good results. We tested for different number of epochs and found that the models start overfitting after 30-35 epochs. So decided to move ahead with 30 epochs for the models.

4 Results and Discussion

This section discusses the results for the 3 models for different hyperparameters. All the models were trained for 30 epochs and tested and validated on parts of the dataset which were kept separate from the training data. Instructions on how to understand a few parameters like kernel size dropout etc. has been mentioned before each table.

4.1 Base Model, M1

The base model was tested with original normalized data and balanced data. We tested for various hyperparameters, some of which have not been mentioned below due to poor results. There are 2 Conv2d layers in this model. The kernel size mentioned below consists of the size for each layer. So for example, for the first row below, both the Conv2d layers have a kernel of shape of 5 X 5.

Table 1. Results for the base model, M1.

Data	Kernel size in Conv layers	Training Loss ↓	Training Accuracy ↑	Validation Accuracy ↑	Testing Accuracy ↑
Norm-original	(5,5)	0.5522	72%	73%	73%
Balanced	(5,5)	**0.4019**	**81%**	**80%**	**81%**
Norm-original	(3,3)	0.5798	72%	70%	68%
Balanced	(3,3)	0.4115	78%	78%	79%

We can see from the Table 1, the balancing in data provides a major improvement in the base model. Un-normalized data gave worse results with a test accuracy of around 69% which shows that normalization helps improve the results. In Figure 8 and 9 below, we can see the graphs for acuuracies and losses for the best base model.

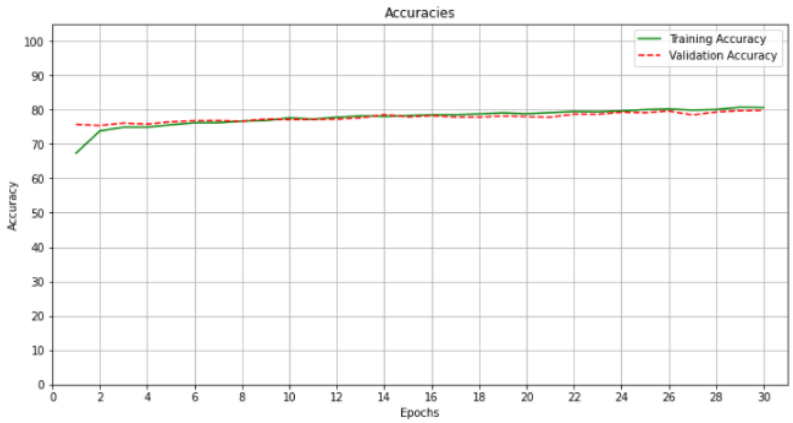


Fig 8: Training and validation accuracies for best Base model

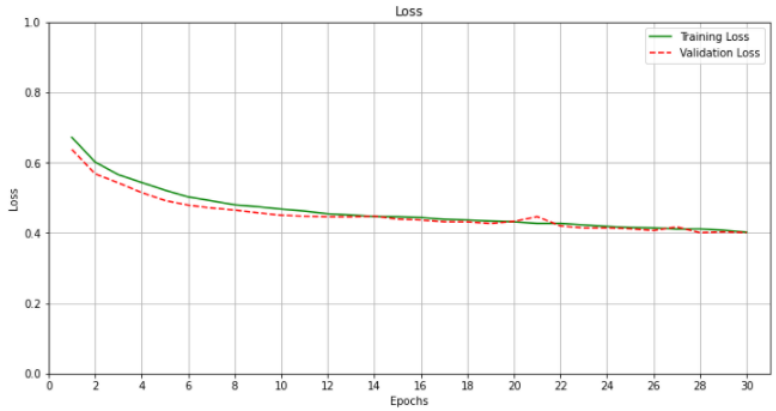


Fig 9: Training and validation loss for best Base model

We can notice from the graphs above, that this base model in not overfitting. This basic model with 2 Conv2d layers will help us compare the effects of adding different layers etc. to the network. The base model has just one dropout layer and no batch normalization applied to layers.

4.2 Improved Model, M2

Moving on the model 2, which contains 3 Convolutional blocks. Each Convolutional blocks contains a Conv2d layer followed by a BatchNorm2d layer, followed by a ReLU layer and then another Conv2d layer which is followed by a ReLU layer and finally a MaxPool2d layer. The dropout percentage has been set to 0.5 for all the dropout layers in this model. These Conv blocks are followed by a dropout layer, which is followed by 2 blocks, each having a fully connected layer and a ReLU layer. Finally we have a dropout layer and then our last fully connected layer.

. There are 3 Convolutional blocks (Conv) in this model. The kernel size mentioned below consists of the size for each block. So for example, the first row below, both the Conv2d layers in the first block have a kernel shape of 5 X 5, the same for the second block but both the layers in the third block, have a kernel of shape 3 X 3. The padding also follows the same logic. The model has 3 Linear/ Fully connected layers. In this model we have tested by keeping the bias either False or True for all three layers. The fully connected layers learn weights for a mapping of all neurons from one layer to another. It follows a learning equation x = w*y +b, where w is the weight and b is the bias. When the bias is set to True, the fully connected layer learns the bias term too, else it doesn't.

Table 2. Results for the improved model, M2.

Data	Kernel size in Conv block	Padding in Conv block	Bias in linear layer	Training Loss ↓	Training Accuracy ↑	Validation Accuracy ↑	Testing Accuracy ↑
Norm-original	(5,5,3)	(2,2,1)	False	0.0037	100%	90%	99%
Balanced	(5,5,3)	(2,2,1)	False	0.2730	86%	84%	87%
Norm-original	(3,3,3)	(1,1,1)	True	0.088	97%	89%	100%
Balanced	(3,3,3)	(1,1,1)	True	**0.2851**	**87%**	**87%**	**89%**

From Table 2, we can see that the unbalanced data overfits. We get high Training accuracy but low validation accuracies. We see that this issue is resolved when using the balanced dataset. We also notice that a kernel size of 3 works better for this model as compared to 5. We choose the last variant of the model to be the best since it doesn't overfit and gives the best performance. Getting a good testing accuracy while a poor validation accuracy, could be due to having uneven/unequal data distributions in the train, test and validation data. Figure 10 and 11 below, show the trend of the accuracies and losses with each epoch.

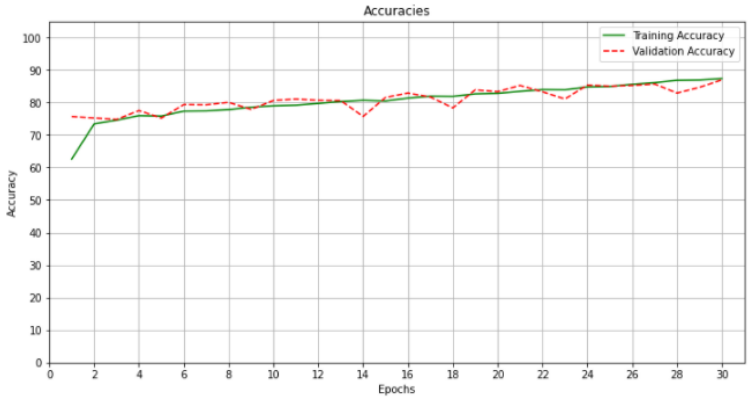


Fig 10: Training and validation accuracies for best model M2

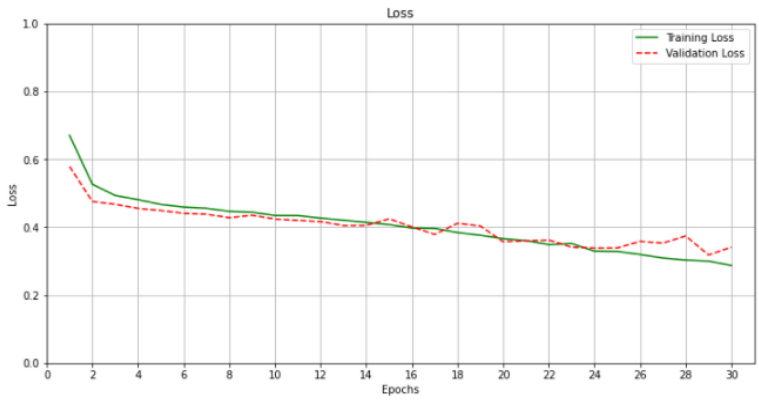


Fig 11: Training and validation loss for best model M2

4.3 Another Improved Model, M3

Model 3 or M3 contains 4 Convolutional blocks instead of 3 like in M2. The fully connected structure is also similar. More emphasis was put into Dropout for this model. Using dropout layers helps in fixing the overfitting problem. We tested with different dropout factors/probabilities.

.

There are 4 Convolutional blocks (Conv) in this model. The kernel size mentioned below consists of the size for each block. So for example, the first row below, both the Conv2d layers in the first block have a kernel shape of 5 X 5,

the same for all the 4 blocks. The padding also follows the same logic. The model has 3 Linear/ Fully connected layers. In this model we have tested different combinations of True and False bias. The bias settings for each layer is shown in order, for example, for the 5th row, the first 2 fully-connected layers have bias set as True the third has it False.

Table 3. Results for the improved model, M3.

Data	Kernel size in Conv2d layer	Padding in Conv2d layer	Dropout factor	Bias in linear layers	Training Loss ↓	Training Accuracy ↑	Validation Accuracy ↑	Testing Accuracy ↑
Norm-original	(5,5,5,5)	(2,2,2,2)	(0.5,0.5,0.5,0.5)	F,F,F	0.0126	100%	82%	92%
Balanced	(5,5,5,5)	(2,2,2,2)	(0.5,0.5,0.5,0.5)	F,F,F	0.0832	96%	86%	94%
Balanced	(3,3,3,3)	(1,1,1,1)	(0.5,0.5,0.5,0.5)	F,F,F	0.4897	86%	84%	87%
Balanced	(3,3,3,3)	(1,1,1,1)	(0.3,0.3,0.2,0.2)	F,F,F	0.2051	92%	88%	91%
Balanced	(3,3,3,3)	(1,1,1,1)	(0.3,0.3,0.3,0.3)	T,T,F	**0.1649**	**93%**	**90%**	**97%**
Balanced	(3,3,3,3)	(1,1,1,1)	(0.3,0.3,0.3,0.3)	T,T,T	0.2073	93%	89%	96%

We get better results than model 2, but we also notice cases of model overfitting, in both unbalanced and balanced cases. This shows that, fixing class imbalance improves the performance and helps avoid overfitting, like in the base model, but there is still some issue which causes such results. One thing that we can think of which I mentioned earlier too is, that we have a good training and testing accuracy but a poor validation accuracy. This shows strong signs towards having unbalanced/uneven train, test and validation sets. The test set seems to be more similar to the train set than the validation set.

We can try to use techniques like data augmentation to increase the number of images, which could lead to a better and even random split. We can also try to use K-fold cross-validation to help with this non-generalization problem.

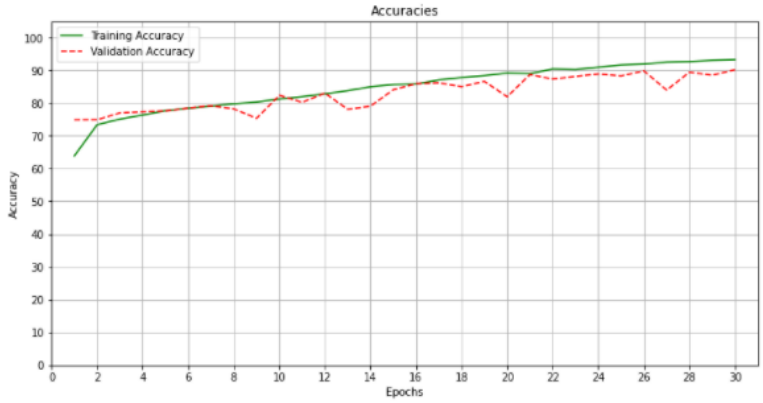


Fig 12: Training and validation accuracies for best model M3

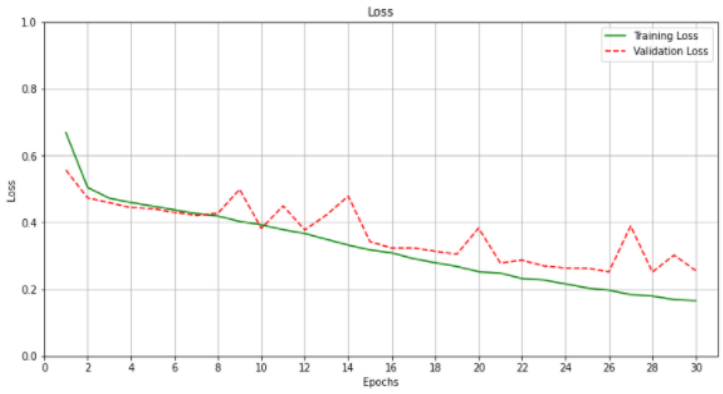


Fig 13: Training and validation loss for best model M3

Figures 12 and 13, help us see that while the selected hyperparameters for the third model, do not give a very high accuracy (95+), but they give a decent model which is still a little far from overfitting.

We can further test with more deeper models using more Convolutional blocks like VGG16 or 19 which are known to work good for image classification tasks with different hyperparameters but we couldn't try out more due to low computational capacity.

5 Conclusion and Future Works

Using convolutional networks for images works really well, converges very soon and is computationally much faster than normal artificial neural networks. Using techniques like max pooling, dropout and batch normalization help build a better model and prevents overfitting. Class imbalance creates a false picture of a model working well and ends up overfitting when tested with a different set of data. The distribution within train, test and validation is very important too. Uneven distribution could result in weird and incorrect results leading to a wrong estimation of the performance of the model. This issue could be solved using the K-fold cross validation method.

We could experiment more to find better hyperparameters and get a better and more generalized model. We can also include data augmentation techniques on the complete dataset to increase the data and reduce overfitting of the model. We can try using different networks like Deconvolutional Neural Networks [10, 12] or Autoencoders [10]. We can use our model in more classification tasks. We can modify it to cater multi-class classification by just changing the last fully-connected layer.

References

1. Afrakhteh, S., Mosavi, MR., Khishe, M.: Accurate Classification of EEG Signals Using Neural Networks Trained by Hybrid Population-physic-based Algorithm. Int. J. Autom. Comput. 17, pp 108–122 (2020). https://doi.org/10.1007/s11633-018-1158-3
2. Bahl S.: Alcoholism Classification using Neural Networks, ABCs ANU Bio-inspired Computing conference vol. 4 (2021)
3. Krizhevsky A., Sutskever I., Hinton Geoffrey E,: ImageNet Classification with Deep Convolutional Neural Networks, In: Pereira F., Burges C. J. C., Bottou L., Weinberger K. Q. (eds.), Advances in Neural Information Processing Systems, vol. 25, Curran Associates, Inc. (2012)
4. Kumar N.: Batch Normalization and Dropout in Neural Networks with Pytorch, https://towardsdatascience.com/batch-normalization-and-dropout-in-neural-networks-explained-with-pytorch-47d7a8459bcd
5. LeCun Y., Bottou L., Bengio Y., Haffner P.: Gradient-Based Learning Applied to Document Recognition, Proceedings of the IEEE, 86(11):2278-2324 (1998)
6. Srivastava N., Hinton G., Krizhevsky A., Sutskever I., Salakhutdinov R.: Dropout: A Simple Way to Prevent Neural Networks from Overfitting, Journal of Machine Learning Research, vol. 15, pp. 1929-1958, http://jmlr.org/papers/v15/srivastava14a.html
7. Subasi A., Erçelebi E,: Classification of EEG Signals Using Neural Network and Logistic Regression, Computer Methods and Programs in Biomedicine, vol. 78, Issue 2, pp. 87-99, (2005), ISSN 0169-2607, https://doi.org/10.1016/j.cmpb.2004.10.009.
8. Szegedy C., Liu W., Jia Y., Sermanet P., Reed S., Anguelov D., Erhan D., Vanhoucke V., Rabinovich A.: Going Deeper with Convolutions, CoRR, vol. abs/1409.4842 (2014), http://arxiv.org/abs/1409.4842
9. Xie S., Girshick R., Dollár P., Tu Z., He K.: Aggregated Residual Transformations for Deep Neural Networks, CoRR, vol. abs/1611.05431 (2016), http://arxiv.org/abs/1611.05431
10. Yue Y., Jo P., Gedeon T.: Deep Feature Learning and Visualization for EEG Recording Using Autoencoders, International Conference on Neural Information Processing, pp. 554--566, Springer (2018).
11. Yue Y., Jo P., Gedeon T.: Information-Preserving Feature Filter for Short-term EEG signals, Neurocomputing, vol. 408, pp. 91-99, Elsevier (2020)
12. Zeiler, M.D., Fergus, R.: Visualizing and understanding convolutional networks. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) ECCV 2014, Part I. LNCS, vol. 8689, pp. 818–833. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10590-1_53
13. UCI Machine Learning Repository, https://archive.ics.uci.edu/ml/datasets/eeg+database