Nov 26, 2024

Project Report

Names: Adin Mauer (43514439), Sikher Sinha (23422355), Weifeng Ke (18879288)

Abstract

This project focuses on developing a machine learning model that can analyze and classify electroencephalogram (EEG) signals recorded from human subjects performing or imagining specific motor movements with their fists or feet. This kind of task is critical for advancing brain-computer interface (BCI) applications. The dataset used for this study is publicly available from OpenNeuro (https://openneuro.org/datasets/ds004362/versions/1.0.0), providing comprehensive EEG recordings suitable for our objectives.

We explored and delivered two machine learning methods: a Support Vector Machine (that showed limited performance), and a Convolutional Neural Networks (CNNs). We evaluated these models using metrics such as accuracy, recall, precision, and F1 scores. Our results, summarized in the 'evaluation' section, indicate that the CNN model outperforms the SVM model, achieving an accuracy of **61.05% vs 56.4%** on the classification. Despite not reaching our target accuracy of 70%-75% (from the proposal), we believe that the effort invested in building and improving the model has provided us with significant theoretical and practical knowledge, as well as valuable skills in machine learning. This report outlines the various strategies and approaches we employed to develop and improve the model.

Introduction:

Before machine learning became widespread, researchers used advanced signal processing techniques like Continuous Wavelet Transforms (CWT) and Short-Term Fourier Transforms (STFT) to analyze and classify EEG signals from multiple channels. These methods break down multi-channel EEG signals into temporal-spatial and frequency components, which help understand how brain signal frequency characteristics behave and change over time. However, the usage of these tools require manual feature engineering and may not capture all the complex patterns in the data - and this is where Machine Learning shines.

Machine learning methods do not rely on manually finding features, but are rather algorithms that can automatically discover feature patterns within the data. In our project, we tried to use support vector machines (SVM) and Convolutional Neural Networks (CNN) for classifying EEG motor activity. SVM can efficiently handle high-dimensional feature spaces and is robust to outliers, noise. These merits make SVM a powerful tool for EEG signal analysis. On the other hand, Convolutional Neural Networks (CNNs) have become powerful tools for classifying EEG signals. While CNNs aren't specifically designed to capture temporal data sequence patterns, they can still detect patterns over time and between different EEG channels. To tackle our classification problem, we started with two CNN architectures inspired by recent academic papers (see references), and used those models for inspiration to start building our own models.

Data:

Introduction

The dataset utilized in this study is obtained from a collection of **103 subjects**, 59 females, 42 males (M:F ratio ~0.71), and 2 unspecified. Ages (available for 93) average ~40.5 years, with most in their 30s or 40s. Each subject performs various motor and imagery tasks. EEG signals were recorded using **64-channel electrodes** through the **BCI 2000 system**. After preprocessing, this dataset produced **18,529 distinct labeled data instances**. Each data instance consists of **64 EEG channels** with time-domain signal data collected at a sampling rate of **160 Hz** for a duration of **3 starting** from motion initiation.

Nov 26, 2024

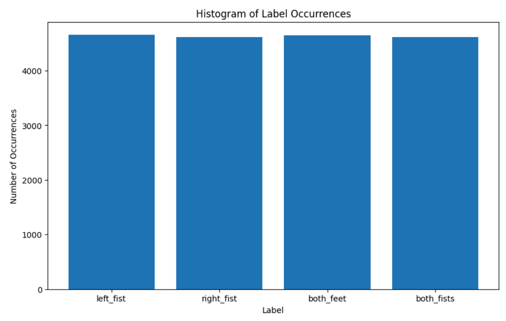
Preprocessing

The data underwent several preprocessing steps to ensure high-quality input for the models:

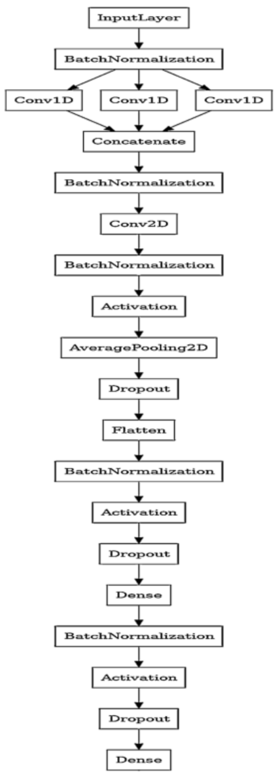
1. **Data Extraction**: The raw data, originally stored in **.set** files, were extracted using Python libraries **MNE**. The EEG signals were then converted into **numpy arrays** for easier manipulation. Each experimental run was classified into four distinct tasks based on the event labels (TaskXT0), resulting in a structured dataset where each instance corresponds to a specific task (**Left Fist (T1)**, **Right Fist (T2)**, **Both Fists (T1)**, and **Both Feet (T2)**). After data was segmented and labeled, the final dataset was stored in **HDF5** format for efficient data handling and model input. File size is **2.21 GB**.
2. **Data Filtering**: As most motor/imagery signals typically lie within the frequency bands of delta (0.5-4 Hz), theta (4-8 Hz), alpha (8-13 Hz), and beta (13-30 Hz), a **low-pass filter** was applied to eliminate frequencies above **35 Hz**, following recommendations from prior research on motor imagery. This filtering step was optional, allowing for both **filtered** and **unfiltered** versions of the dataset to be used for CNN model input.
3. **Data structure:** The input data has **18,529** data instances, each data instance consists of **64** channels and each channel consists of **480** time series data points. The data was then saved in separate training and test files (80% train, 10% validation, 10% test).

Data Characteristics

- **Data Balance**: The dataset has a relatively balanced distribution with respect to the four distinct motor/imagery tasks: Left Fist, Right Fist, Both Fists, and Both Feet. Here is the histogram of the Labelled data.
- **Data Division**: The data was[1] split into **80% training, 10% testing, and 10% validation** sets. Each subject's data was randomly assigned to these sets to ensure no overlap between training and testing samples, thereby maintaining the integrity of the evaluation process. The final dataset contains approximately **18.5k samples**.



In conclusion: the vast dataset of motor/imagery EEG data required a lot of preprocessing to be ready to be put into any deep learning models. Filtered and unfiltered versions of the processed dataset were made available for flexibility in training. The comprehensive preprocessing ensured that the data is clean, well-structured, and ready for further analysis.



Model:

The primary model we used for this project is a CNN specifically designed for analyzing EEG data. Its architecture is tailored to handle input data of size 64×480 representing 64 EEG channels and 480 time steps. The model consists of several layers that process the data step by step to identify meaningful patterns.

The first set of layers is made up of 3 parallel 1D temporal convolution layers. These layers extract features from the time domain, such as spikes, oscillations, and trends within the EEG signals. The next layer is a spatial convolution layer, which captures relationships across the 64 EEG channels. This helps the model understand how different regions of the brain interact. After the convolutions, an average pooling layer reduces the size of the

[1] Figure from (Ousama, 2024). The figure does not accurately show our CNN model.

Nov 26, 2024

feature maps, making the model computationally efficient while retaining important information.

The processed data is then passed through fully connected layers, which learn higher-level patterns and make the final classification into one of the four output classes. Dropout is used in the fully connected layers to prevent overfitting during training, and batch normalization ensures stable and faster learning.

We chose this model because it is well-suited for time-series data like EEG signals, capturing both temporal and spatial patterns. The architecture was built and fine-tuned through hyperparameter grid search, where we optimized parameters like learning rate, weight decay, batch size, and dropout rate. This process ensured that the model performed well without overfitting or underfitting. Further details about the hyperparameter optimization are provided in the discussion section. The table below outlines how the data is passed through various layers explaining how its size changes based on the kernel size, number of filters, and other parameters.

Layer	Input Dimension:	Output Dimension:	Number of Kernels:	Kernel Dimension:	Stride:	Paddings
Conv1D temporal	(num_batches, 64, 480)	(batch_size, 64, 80)	8	(1,8)	(1,6)	(0,3)
Conv1D temporal	(num_batches, 64, 480)	(batch_size, 64, 80)	8	(1,16)	(1,6)	(0,7)
Conv1D temporal	(num_batches, 64, 480)	(batch_size, 64, 80)	8	(1,32)	(1,6)	(0,15)
Concatenation	Of the 1st 3 layers	(batch_size, 24, 64, 80)				
Conv2d spatial	(batch_size, 24, 64, 80)	(batch_size, 40, 1, 80)	40	(64, 1)	(1,1)	
Avg Pool	(batch_size, 40, 1, 80)	(batch_size, 40, 1, 16)		(1,5)	(1,5)	
Flatten	(batch_size, 40, 1, 16)	(batch_size, 640)				
Dense 1	(batch_size, 640)	(batch_size, 100)				
Dropout						
Dense 2	(batch_size, 100)	(batch_size, 4)				

In terms of **optimization**, we chose to use the **“Adam” optimizer**, which is a variant of Stochastic Gradient Descent that utilizes momentum (not just simple gradient) for calculating the direction and magnitude to the next step in the cost function hyperplane. The **cost function** we used is ‘cross-validation’, as is fitting for a classification problem.

Evaluation:

Every iteration of training the CNN & SVM model, the performance was compared relative to the last best model known for each separately. If the new model outperformed the previously best-known model, then the model parameters, hyperparameters and results were overwritten. In training, 10% of the data (~1800 inputs) was set aside for testing. Post training, a confusion matrix and evaluation metrics are found from inference on the test data set.

Overall Accuracy for CNN Model: 61.03% **SVM Model: 56.2%**

CNN	Class 0:	Class 1:	Class 2:	Class 3:	Average	SVM	Class 0:	Class 1:	Class 2:	Class 3:	Average
Precision	0.7	0.68	0.59	0.51	62%		0.62	0.81	0.7	0.40	63%
Recall	0.57	0.62	0.62	0.63	58.5%		0.6	0.24	0.71	0.69	56%
F1-Score	0.63	0.65	0.61	0.57	61.5%		0.61	0.38	0.71	0.51	55%

Discussion:

Hyperparameter Optimization

One of the annoying things about deep learning is that there is no standard closed form technique for finding hyper parameters of a model architecture. While there are several techniques and tools to help with hyperparameter tuning, we chose to run a grid search on the following parameters:

- 'learning_rate': [0.001, 0.0001],
- 'batch_size': [8, 16],
- 'epochs': [20],

Nov 26, 2024

- 'weight_decay': [1e-5, 1e-4],
- 'dropout_rate': [0.5]

Although we did experiment a lot with numbers of kernels (output channels) and kernel dimensions (temporal and spatial), we did not approach these hyperparameters systematically. We may have been able to further improve the model by tuning those hyper-parameters systematically as well (kernels per layer [output channels], kernel sizes, strides, padding, etc.).

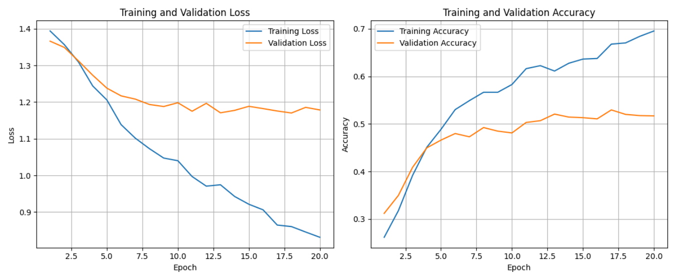
Normalization

One of the things we struggled with initially is that our model was giving us only 25% accuracy despite anything we were trying. After printing the weights being learned from the first layers we noticed quickly vanishing gradients. A quick search on the internet suggested that our data may not be normalized properly. We thought that min-max normalization of data between [0,1] would be enough, but it turns out that after normalizing our data to have mean 0 and standard deviation of 1, our results improved immediately. Normalization of the input data is extremely important and useful because it prevents the model from needing to learn the initial bias and variance which may be significantly different between models (but those data properties do not hold the information needed to identify EEG features). Additionally, we introduced batch normalization layers into the CNN, which also improved the performance of the model.

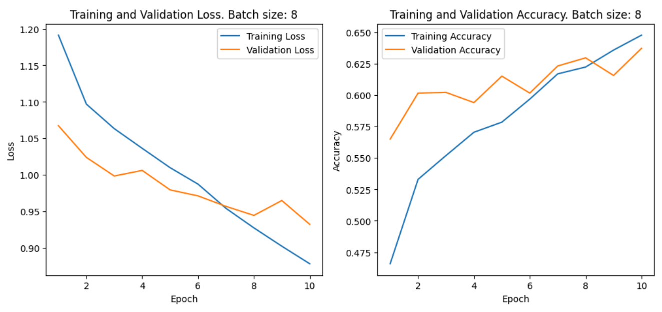
Train loss vs Validation loss during training, in between epochs:

Every epoch contains a training session on all data and a validation run on the rest. At the end of every epoch the following metrics are recorded: Training loss, training accuracy, validation loss, validation accuracy. After all epochs run the code then plots both - as shown below -
Keeping a close look at these graphs can teach us a lot about the performance of the model.

- ***Comparing training vs validation:***
  - Overfitting: If the training accuracy overshoots the validation accuracy in some earlier epoch and the training ends with a significant final overshoot, then you can say that the model is training for too many epochs and it is **overfitting** the training data. The response from this may be to limit the number of epochs or introduce stronger regularization to the model. This can also mean that the model is not yet powerful enough to learn complex features from the data.



  - Underfitting: If (as shown above) the training and validation are both similar and trending upwards at the end of the training, it is safe to say that the model may benefit from additional epochs of training.
  - Finally, the ideal picture you want to see is where the training accuracy starts lower than the validation, and over time until the end of the training the both converge to the same approximate range.



- ***Inferring about learning rate from stability:***
  - If you see in between epochs the loss/accuracy varies greatly and is unstable, that may mean that your learning rate is too high, and you are overshooting minima points in the cost function and preventing convergence.

Nov 26, 2024

References:

Tarahi, Ousama & Hamou, Soukaina & Moufassih, Mustapha & Agounad, Said & Hafida, Idrissi Azami. (2024). *Decoding Brain Signals: A Convolutional Neural Network Approach for Motor Imagery Classification.* e-Prime - Advances in Electrical Engineering, Electronics and Energy. 7. 100451. 10.1016/j.prime.2024.100451.

Zhang J, Liu D, Chen W, Pei Z, Wang J. *Deep Convolutional Neural Network for EEG-Based Motor Decoding.* Micromachines. 2022; 13(9):1485. https://doi.org/10.3390/mi13091485

The dataset is taken from the OpenNeuro website and has a GitHub repo. The dataset was created by: BCI R&D Program, Wadsworth Center, New York State Department of Health, Albany, NY.

Link to dataset: https://openneuro.org/datasets/ds004362/versions/1.0.0