**A PROJECT REPORT**

**on**

**STOCK MARKET PREDICTION USING REINFORCEMENT LEARNING**

**Submitted to**

**KIIT Deemed to be University**

**In Partial Fulfilment of the Requirement for the Award of**

**BACHELOR’S DEGREE IN**

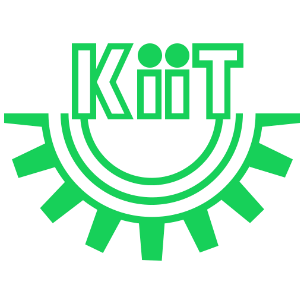
**TECHNOLOGY**

**BY**

|  |  |
| --- | --- |
| **YATHARTH JAIN** | 21051918 |
| **SAHIL KUMAR** | 21051923 |
| **ESHAAN MODH** | 21051991 |
| **SAGNIK SEN** | 21051922 |
| **KOUSHIK CHAKRABORTY** | 21051902 |
| **DEEPANSHU** | 2105 |

**UNDER THE GUIDANCE OF**

**Dr. CHITTRANJAN PRADHAN**



**SCHOOL OF COMPUTER ENGINEERING**

**KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY**

**BHUBANESWAR, ODISHA - 751024**

**May 2020**

KIIT Deemed to be University

School of Computer Engineering

Bhubaneswar, ODISHA 751024



CERTIFICATE

This is to certify that the project entitled

STOCK MARKET PREDICTION USING REINFORCEMENT LEARNING

submitted by

|  |  |
| --- | --- |
| **YATHARTH JAIN** | 21051918 |
| **SAHIL KUMAR** | 21051923 |
| **ESHAAN MODH** | 21051991 |
| **SAGNIK SEN** | 21051922 |
| **KOUSHIK CHAKRABORTY** | 21051902 |
| **DEEPANSHU** | 2105 |

is a record of bonafide work carried out by them, in the partial fulfilment of the requirement for the award of Degree of Bachelor of Engineering (Computer Science & Engineering OR Information Technology) at KIIT Deemed to be university, Bhubaneswar. This work is done during year 2024-2025, under our guidance.

Date: / /

Dr. CHITTRANJAN PRADHAN

Project Guide

**Acknowledgements**

We are profoundly grateful to Dr. **Chittranjan Pradhan** of **Affiliation** for his expert guidance and continuous encouragement throughout to see that this project rights its target since its commencement to its completion. .....................

Yatharth Jain

Sahil Kumar

Eshaan Modh

Sagnik Sen

Koushik Chakraborty

Deepanshu

**ABSTRACT**

This project is focused on the development of a machine learning model to detect and classify harmful brain activity, including seizures, using electroencephalography (EEG) signals. These signals are recorded from critically ill patients in a hospital setting. The current standard for EEG monitoring involves manual analysis by specialized neurologists. This process, while invaluable, is labor-intensive, time-consuming, and prone to errors due to fatigue and variability between different reviewers.

The goal of this project is to automate the process of EEG analysis, thereby reducing the time and cost associated with manual review, and potentially increasing the accuracy and reliability of the results. This could lead to faster and more accurate treatments for patients, and could also aid researchers in the development of drugs to treat and prevent seizures.

The machine learning model will be trained to recognize six specific patterns of interest within the EEG signals: seizure (SZ), generalized periodic discharges (GPD), lateralized periodic discharges (LPD), lateralized rhythmic delta activity (LRDA), generalized rhythmic delta activity (GRDA), and "other". These patterns have been annotated by a group of experts, providing a labeled dataset for model training.

The success of this project could lead to significant advancements in neurocritical care, epilepsy treatment, and drug development, and could potentially transform the way EEG monitoring is performed.

Contents

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | Introduction | | | 6 |
| 2 | Basic Concepts/ Literature Review | | | 7 |
|  | 2.1 | Electroencephalography (EEG) Signals | | 7 |
|  | 2.2 | Machine Learning Techniques | | 7 |
|  | 2.3 | Keras and KerasCV | | 7 |
|  | 2.4 | Harmful Brain Activity Classification | | 8 |
|  | 2.5 | Integration of KerasCV and Keras | | 8 |
|  | 2.6 | Challenges and Future Directions | | 8 |
| 3 | Problem Statement / Requirement Specifications | | | 9 |
|  | 3.1 | Project Planning | | 11 |
|  | 3.2 | Project Analysis (SRS)................. | | 12 |
|  | 3.3 | System Design ………………….. | | 13 |
|  |  | 3.3.1 | Design Constraints …… | 13 |
|  |  | 3.3.2 | System Architecture (UML) / Block Diagram … | 13 |
| 4 | Implementation | | | 15 |
|  | 4.1 | Methodology / Proposal ........................... | | 15 |
|  | 4.2 | Testing / Verification Plan ……………. | | 21 |
|  | 4.3 | Result Analysis / Screenshots …………. | | 21 |
| 5 | Standard Adopted | | | 23 |
|  | 5.1 | Design Standards . . . . . . . . . . . . . . . | | 23 |
|  | 5.2 | Coding Standards . . . . . . . . . . . . . . | | 23 |
|  | 5.3 | Testing Standards . . . . . . . . . . . . . . . | | 24 |
| 6 | Conclusion and Future Scope | | | 25 |
|  | 6.1 | Conclusion ……………………….. | | 25 |
|  | 6.2 | Future Scope ………………………. | | 26 |
| References | | | | 28 |
| Plagiarism Report | | | | 29 |

Chapter 1

Introduction

The Harmful Brain Activity Classification (HMS) framework integrated with KerasCV and Keras addresses a critical need in the field of neuroscience and clinical neurology. Currently, there exists a pressing demand for accurate and efficient methods to detect and classify harmful brain activity, particularly in conditions such as epilepsy, neurodegenerative diseases, and psychiatric disorders. Traditional diagnostic approaches often rely on subjective interpretation rather than objective criteria of electroencephalography (EEG) signals by clinicians, leading to degraded accuracy in identification of brain seizures and abnormalities and delayed results and responses. Consequently, there is an urgent need for data-driven solutions that can identify and classify harmful brain activity in real-time in a reliable manner facilitating prompt treatment plans and intervention.

Despite advancements in neuroimaging technology and machine learning algorithms, several gaps persist in current available solutions for harmful brain activity classification. Firstly, many existing models lack robustness and generalizability, as they are trained and evaluated on limited datasets that may not adequately represent the diversity of neurological conditions and individual variations in brain activity. Additionally, the interpretability of these models remains a challenge, hindering their adoption in clinical practice where transparent decision-making is paramount. Moreover, the computational complexity of some algorithms restricts their scalability and flexibility, limiting their utility in clinical settings where timely interventions are crucial. Addressing these gaps requires the development of an integrated framework like HMS, which combines state-of-the-art machine learning techniques with KerasCV and Keras to deliver accurate, interpretable, and scalable solutions for harmful brain activity classification.

Chapter 2

Basic Concepts/ Literature Review

In recent years, there has been a surge in research focusing on machine learning techniques for analyzing EEG signals and classifying harmful brain activity. Studies have explored the application of convolutional neural networks (CNNs), recurrent neural networks (RNNs), and hybrid architectures to extract relevant features from EEG data and distinguish between normal and pathological brain patterns. Additionally, researchers have investigated the integration of advanced signal processing methods, such as wavelet transforms and time-frequency analysis, to enhance the discriminative power of machine learning models.

The literature also highlights the importance of robust and diverse datasets for training and evaluating harmful brain activity classification models. Datasets encompassing EEG recordings from patients with various neurological disorders, as well as healthy controls, are essential for ensuring the generalizability and reliability of classification algorithms. Moreover, efforts have been made to develop standardized benchmark datasets and evaluation protocols to facilitate comparison and reproducibility across different studies.

2.1 Electroencephalography (EEG) Signals:

    EEG signals are essential in capturing electrical activity in the brain, providing valuable insights into neurological conditions. EEG is non-invasive and cost-effective in nature, making it pivotal for diagnosing various brain abnormalities.

2.2 Machine Learning Techniques:

    The development of accurate classification models depend on the intersection of machine learning and neuroscience facilitates. Research focuses on convolutional neural networks (CNNs), recurrent neural networks (RNNs), and hybrid architectures to process EEG data in more effective manner.

2.3 Keras and KerasCV:

    Keras, a high-level neural networks API, simplifies the creation of deep learning models with TensorFlow backend.

    KerasCV extends Keras with computer vision utilities, providing a streamlined framework for image processing tasks.

2.4 Harmful Brain Activity Classification:

    The primary objective is to identify and classify harmful brain patterns, crucial for diagnosing conditions like epilepsy and neurodegenerative diseases. Current research emphasizes the need for accurate, interpretable, and scalable classification methods.

2.5 Integration of KerasCV and Keras:

Leveraging KerasCV and Keras, the HMS framework enhances the efficiency and accuracy of harmful brain activity classification. This integration allows for seamless development and deployment of machine learning models tailored to EEG data analysis.

2.6 Challenges and Future Directions:

    Challenges include the interpretability of models, scalability of computational approaches, and real-time deployment in clinical settings. Future research should focus on interdisciplinary collaboration to address these challenges and advance the field of harmful brain activity classification.

Chapter 3

Problem Statement

Critically ill patients in intensive care units (ICUs) are susceptible to seizures and other harmful brain activity patterns that significantly impact their health outcomes. Early and accurate detection is crucial for timely intervention and improved patient care. However, current methods often rely on visually analyzing electroencephalography (EEG) data, a time-consuming and subjective process prone to human error.

**Project Goal**

Develop a robust and automated system for classifying seizures and other patterns of harmful brain activity in critically ill patients using Keras and KerasCV for the deep learning aspects. This system should:

* Achieve high accuracy in classifying harmful brain activity patterns.
* Be computationally efficient for near-real-time analysis within ICU resource constraints.
* Utilize Keras and KerasCV to leverage their image classification capabilities for EEG data represented as spectrograms.

**Requirements Specifications**

**Functional Requirements**

* **Data Input:** The system should ingest continuous EEG data streams from standard ICU monitoring equipment.
* **Data Preprocessing:** The system should preprocess the EEG data to remove noise, artifacts, and baseline shifts.
* **Feature Extraction:** The system should transform preprocessed EEG data into spectrograms, which are visual representations of the frequency content over time, suitable for image classification techniques.
* **Classification:** The system should employ a deep learning model built with Keras and KerasCV to classify spectrograms into different categories, including:
* Seizure
* Generalized periodic discharges (GPDs)
* Lateralized periodic discharges (LPDs)
* Lateralized rhythmic delta activity (LRDA)
* Generalized rhythmic delta activity (GRDA)
* Normal background activity
* **Output:** The system should provide near-real-time classification results, including:
* Class labels for each EEG segment
* Confidence scores for the assigned labels
* Visualizations or summaries of the classified activity (optional)
* **Alerting:** The system could optionally generate alerts for healthcare professionals when harmful brain activity is detected. These alerts should be configurable based on severity levels and user preferences.

**Non-Functional Requirements**

* **Accuracy:** The system should achieve a high level of classification accuracy on a benchmark dataset of labeled EEG data. A target accuracy of at least 90% for seizure detection and 85% for other harmful brain activity patterns is a common goal.
* **Performance:** The system should be computationally efficient enough to process near-real-time EEG streams within the constraints of ICU environments. Keras and KerasCV offer potential optimizations for efficient image classification.
* **Interpretability:** While achieving high accuracy is crucial, some level of interpretability is valuable for healthcare professionals to understand the reasoning behind the system's classification decisions. Techniques like Class Activation Maps (CAMs) might be explored for interpreting the model's focus areas within spectrograms.
* **Usability:** The user interface (UI) for the system should be intuitive and easy for healthcare professionals to use, even those without extensive machine learning expertise.
* **Security:** The system should adhere to relevant healthcare data security regulations to protect patient privacy.
* **Scalability:** The system should be scalable to accommodate a growing number of patients and data streams.

**3.1 Project Planning**

**Project Management Methodology:**

* Consider using a structured methodology like Agile or Waterfall, depending on project size, team composition, and tolerance for uncertainty.

**Development Phases:**

* **Phase 1: Data Acquisition and Exploration (1-2 months)**
* Gather a representative dataset of EEG recordings from critically ill patients with labeled harmful brain activity events.
* Explore the dataset to understand the spectrograms' characteristics for different brain activity patterns.
* **Phase 2: Feature Engineering (1-2 months)**
* Develop algorithms to convert raw EEG data into well-formed spectrograms suitable for deep learning models.
* Explore data augmentation techniques to artificially increase dataset size and improve model generalization.
* **Phase 3: Model Selection, Training, and Validation (2-3 months)**
* Experiment with different deep learning architectures using Keras and KerasCV, potentially including:
* Convolutional Neural Networks (CNNs) for their ability to learn spatiotemporal features from spectrograms.
* Recurrent Neural Networks (RNNs) or their variants (LSTMs, GRUs) to handle sequential dependencies within the data.
* Employ appropriate regularization techniques (e.g., dropout, weight decay) to prevent overfitting.
* Continuously validate the model's performance on a separate holdout test set.
* **Phase 4: System Integration and Testing (1-2 months)**
* Integrate the trained model and preprocessing

**3.2 Project Analysis**

**Feasibility Analysis:**

* **Technical Feasibility:** Using Keras and KerasCV for deep learning classification of EEG spectrograms is a viable approach. Deep learning has shown promising results in this domain, and Keras simplifies model building and training. However, achieving high accuracy and real-time performance requires careful data preparation, model selection, and optimization for resource-constrained ICU environments.
* **Economic Feasibility:** While initial investment in data acquisition, computational resources (potentially GPUs), and software development exists, the potential benefits of improved patient care and reduced healthcare costs can outweigh the investment. Open-source libraries like Keras and KerasCV help mitigate software development costs.
* **Operational Feasibility:** Integrating the system into existing ICU workflows and ensuring user adoption by healthcare professionals are critical considerations. Training and support are crucial for effective utilization. Partnering with hospital IT staff can streamline integration.

**Risk Analysis:**

* **Data Availability:** Access to a high-quality, labeled dataset of ICU EEG recordings with diverse harmful brain activity patterns is essential. Collaboration with hospitals or research institutions may be necessary.
* **Model Generalizability:** The model's performance might not generalize well to unseen patient data with different underlying conditions or recording equipment. Continuous validation and potential retraining might be required.
* **Computational Constraints:** Real-time processing demands efficient algorithms and hardware. Consider model optimization techniques, potentially using techniques like quantization or pruning, to reduce computational footprint. Explore lightweight model architectures suitable for deployment in resource-constrained environments.
* **Acceptance by Healthcare Professionals:** Trust and acceptance are crucial. The system's interpretability and ability to integrate seamlessly into existing workflows will be major factors. Explainable AI techniques can be explored to enhance interpretability.

**3.3 System Design**

**3.3.1 Design Constraints**

* **Real-Time Performance:** The system must process continuous EEG streams and provide classification results with minimal latency to enable timely intervention.
* **Computational Resources:** ICU environments might have limited computing power. Optimize the system for efficient execution on available hardware, potentially leveraging hardware acceleration with GPUs if feasible.
* **Data Security:** Patient EEG data is highly sensitive. Ensure adherence to relevant healthcare privacy regulations (e.g., HIPAA). Implement secure data storage and transmission protocols.
* **Interpretability:** While achieving high accuracy is crucial, some level of interpretability is valuable for healthcare professionals to understand the reasoning behind the system's classification decisions.
* **Usability:** The user interface should be intuitive and easy for healthcare professionals to use, even with limited technical expertise.

**3.3.2 System Architecture (Block Diagram)**

Spectrogram Generation

Preprocessing

Module

Data Acquisition

Module

Feature Extraction Module

M Module Module

Classification Module

Feature Augmentation

Visualization Module

User Interface

Alerting-Presentation Module

**Description of Modules:**

1. **Data Acquisition Module:**

* Interfaces with ICU monitoring equipment to receive a continuous stream of EEG data.
* Ensures data integrity and synchronization with timestamps.

1. **Preprocessing Module:**

* Removes noise, artifacts, and baseline shifts from the EEG data using techniques like filtering and baseline correction.
* May involve segmentation of the data into smaller epochs for further processing.

1. **Spectrogram Generation Module:**

* Converts preprocessed EEG data into spectrograms using appropriate time-frequency analysis techniques (e.g., Short-Time Fourier Transform).
* Standardizes the format and resolution of the generated spectrograms.

1. **Classification Module (Keras Deep Learning Model):**

* Employs a Keras-based deep learning model to classify the spectrograms into different categories of harmful brain activity patterns and normal background activity.
* Convolutional Neural Networks (CNNs) are a suitable choice due to their ability to learn spatiotemporal features from spectrograms. Explore architectures like VGG16, Inception, or EfficientNet that can be adapted for this task.

Chapter 4

Implementation

In this section, we will discuss the actual implementation of our project. We will cover the technologies used, the structure of the code, and the key functions and classes implemented.

4.1 Methodology OR Proposal

The Objective of this project is to classify seizures and other patterns of harmful brain activity in critically ill patients. We use a pre-trained image classifier and fine-tune it to best fit our use case. For this we:

* ***Import the necessary libraries***
  + We begin by importing necessary libraries such as TensorFlow, Keras, pandas, numpy, tqdm, joblib, and matplotlib.
* ***Create a config class to store hyper parameters and tuning data***
  + A configuration class is defined to set various parameters for the model such as the batch size, number of epochs, learning rate scheduler mode, number of classes, image size, and class names.
* ***Set up paths for the script to find training testing and validation datasets***
  + The script sets up the paths for the training and testing datasets. It also creates directories for storing spectrogram data.
* ***Data preprocessing***
  + We converts the EEG spectrograms from parquet to .npy format to facilitate easier data loading. This is done using a function process\_spec which reads a parquet file, fills any NaN values with 0, transposes the data, and saves it as a npy file. This function is applied to all the records in the training dataset in parallel using joblib.
* ***Data Loader***

The Data Loader is responsible for loading, preprocessing, and augmenting the spectrogram data for the machine learning model.

* **build\_augmenter**: This function creates a list of augmentations to be applied to the spectrogram data. It uses the MixUp augmentation and two types of RandomCutout augmentations for frequency-masking and time-masking. The augmentations are applied randomly with a 50% chance.
* **build\_decoder**: This function creates a function to decode the spectrogram data from .npy files. It reads the file, reshapes the data, extracts a subsample based on the offset value, applies padding if necessary, converts the data to log spectrogram, normalizes the data, and converts the mono channel signal to a 3-channel signal. If labels are provided, it also one-hot encodes the labels.
* **build\_dataset**: This function creates a TensorFlow Dataset from the provided paths, offsets, and labels. It applies the decode function to the data, caches the data if specified, shuffles the data, batches the data, applies the augment function if specified, and prefetches the data for performance.

The Data Loader is designed to handle large datasets that cannot fit into memory all at once. It reads the data in small batches, applies necessary preprocessing and augmentation, and feeds the data to the model in a way that is efficient and conducive to training a deep learning model.

* ***Data Splitting***

This module is responsible for splitting the data into training and validation sets in a stratified and grouped manner. This DataLoader first reads `npy` spectrogram files and extracts labeled subsamples using specified `offset` values. Then, it converts the spectrogram data into `log spectrogram` and applies the popular signal augmentation `MixUp`.

* **Import StratifiedGroupKFold**: This is a cross-validator provided by scikit-learn. It provides train/test indices to split data in train/test sets. This cross-validation object is a variation of StratifiedKFold that returns stratified folds. The folds are made by preserving the percentage of samples for each class.
* **Initialize StratifiedGroupKFold**: The StratifiedGroupKFold object is initialized with 5 splits, shuffle set to True, and a random state for reproducibility.
* **Initialize fold column**: A new column 'fold' is added to the DataFrame df and initialized with -1. This column will be used to store the fold number for each record.
* **Split the data**: The split method of the StratifiedGroupKFold object is used to split the DataFrame into training and validation sets. The 'y' parameter is the target variable (class label), and the 'groups' parameter is the patient ID. This ensures that records from the same patient do not appear in both the training and validation sets, preventing data leakage.
* **Assign fold numbers**: The fold number is assigned to the records in the validation set.
* **Count records in each fold**: The number of records for each class in each fold is counted. This is used to check that the class distribution is roughly equal in each fold.

This methodology ensures that the training and validation sets are stratified (have the same distribution of class labels) and grouped (records from the same patient are not split between the training and validation sets). This helps to prevent data leakage and maintain a balanced class distribution during model training and validation.

* ***Build the Training and validation datasets***

This module is responsible for creating the training and validation datasets for a machine learning model.

* + **Sample from full data**: The groupby method is used to group the DataFrame df by 'spectrogram\_id'. The head(1) method is then used to select the first record for each 'spectrogram\_id'. This is done to keep the dataset size manageable.
  + **Create training and validation DataFrames**: The training DataFrame contains all records not in the current fold (as specified by Config.fold), and the validation DataFrame contains all records in the current fold.
  + **Print the number of records in the training and validation sets:** This is done for informational purposes.
  + **Extract paths, offsets, and labels**: The paths to the spectrogram files, the offsets of the labels in the spectrograms, and the class labels are extracted from the training and validation DataFrames.
  + **Create training and validation datasets**: The build\_dataset function is called to create the training and validation datasets from the paths, offsets, and labels. The datasets are configured to repeat, shuffle, augment, and cache the data. The training dataset is set to repeat and shuffle the data and apply augmentations, while the validation dataset is not

This ensures that the training and validation datasets are created in a way that is conducive to training a deep learning model. The data is shuffled and augmented to prevent overfitting and improve the model's ability to generalize to new data.

* ***Loss Metric***

The loss metric used in this Python script is Kullback-Leibler (KL) Divergence. KL Divergence is a measure of how one probability distribution is different from a second, reference probability distribution.

In the context of this script:

P is the true distribution (actual labels).

Q is the predicted distribution (predictions from the model).

KL Divergence is differentiable, which makes it suitable as a loss function for training machine learning models using gradient-based optimization methods.

The advantage of using KL Divergence as a loss function is that it directly measures the difference between the predicted and true distributions, so there's no need for a separate metric like accuracy for evaluation. The valid\_loss (validation loss) can serve as an *evaluation metric.*

The line ***LOSS = keras.losses.KLDivergence()*** initializes the KL Divergence loss function using Keras.

* ***Model Building***

In this module we build and compile the machine learning model for this project.

* + **Build Classifier:** *keras\_cv.models.ImageClassifier.from\_preset* function is used to create an image classifier model from a preset configuration. The preset is specified by *Config.preset* and the number of classes is specified by *Config.numClasses*. In this case, the preset is *EfficientNetV2 B2*, which is a variant of the *EfficientNet* model that has been pre-trained on a large dataset.
  + **Compile the model**: The compile method is used to configure the learning process before training the model. The optimizer is set to Adam with a learning rate of 1e-4, and the loss function is set to LOSS, which was defined earlier in the script (not shown in the selected code).
  + **Model Summary**: The summary method is used to print a summary of the model's architecture, including the number of layers, the output shape of each layer, and the number of parameters in each layer.

This ensures that the model is properly configured for training. The use of a pre-trained model helps to improve the model's performance by leveraging knowledge learned from a large dataset. The Adam optimizer and the specified loss function are commonly used in deep learning for image classification tasks.

* ***Learning Rate Schedule***

We need to ensure that the learning rate is adjusted in a structured way during model training, which can help to improve the efficiency of the training process and the performance of the final model.

* + **Define LR schedule parameters**: The parameters for the learning rate schedule are defined, including the start, maximum, and minimum learning rates (lr\_start, lr\_max, lr\_min), the number of epochs for the learning rate ramp-up and sustain periods (lr\_ramp\_ep, lr\_sus\_ep), and the decay rate (lr\_decay).
  + **Define LR update function (lrfn)**: This function calculates the learning rate for a given epoch based on the schedule parameters and the current epoch number. The learning rate is calculated differently depending on the mode ('exp', 'step', or 'cos').
  + **Create LR callback**: The keras.callbacks.LearningRateScheduler function is used to create a callback that updates the learning rate according to the lrfn function. This callback is used during model training to adjust the learning rate at the end of each epoch.
  + **Use LR callback**: The get\_lr\_callback function is called with the batch size, learning rate mode, and plot option specified in the Config object. The resulting callback (lr\_cb) is used in the model training process.
* ***Model Checkpointing***

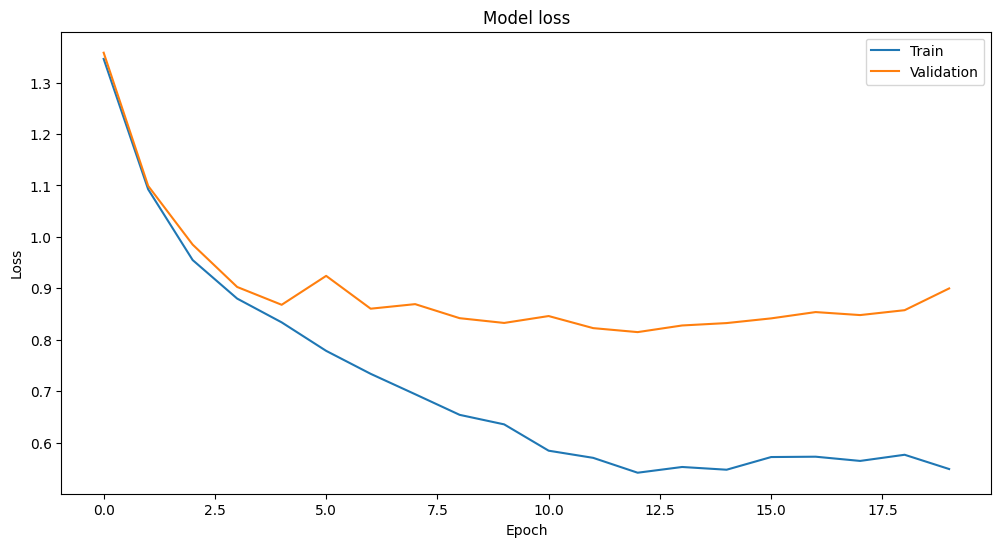
This is to ensure that the best version of the model is saved during training. This is useful because the model's performance may start to degrade after a certain point due to overfitting. By saving the model whenever it improves, you can ensure that you have the best version of the model, even if the training process continues past the point of peak performance.

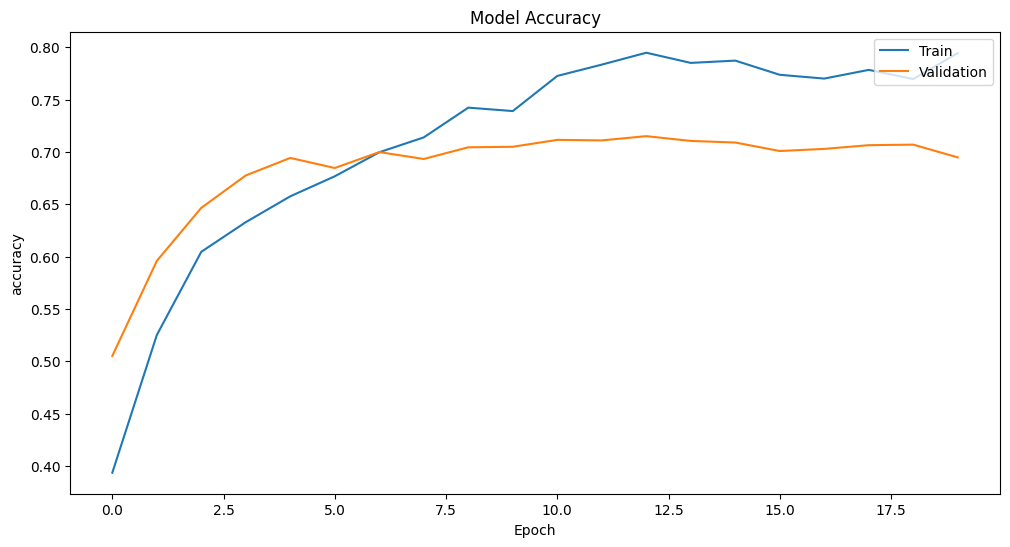
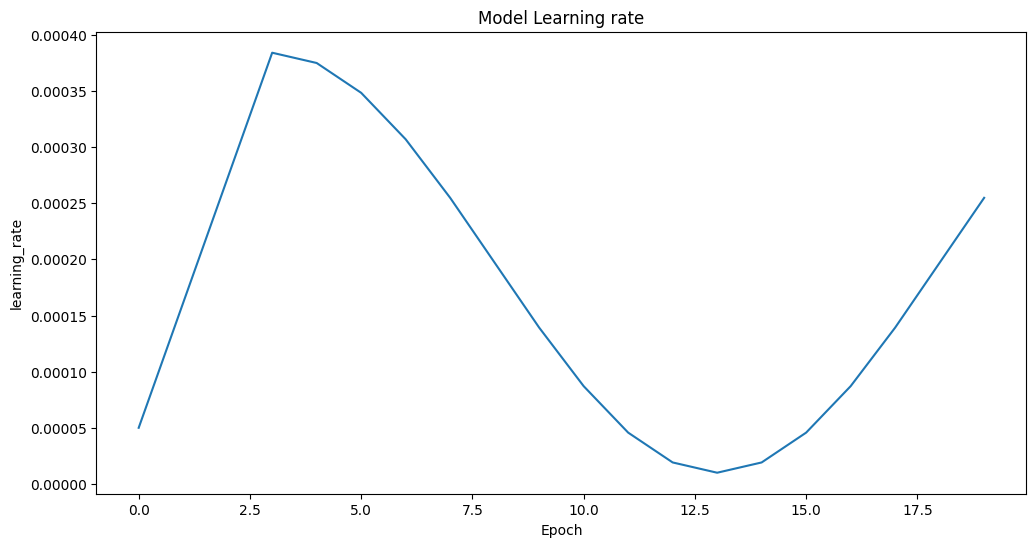
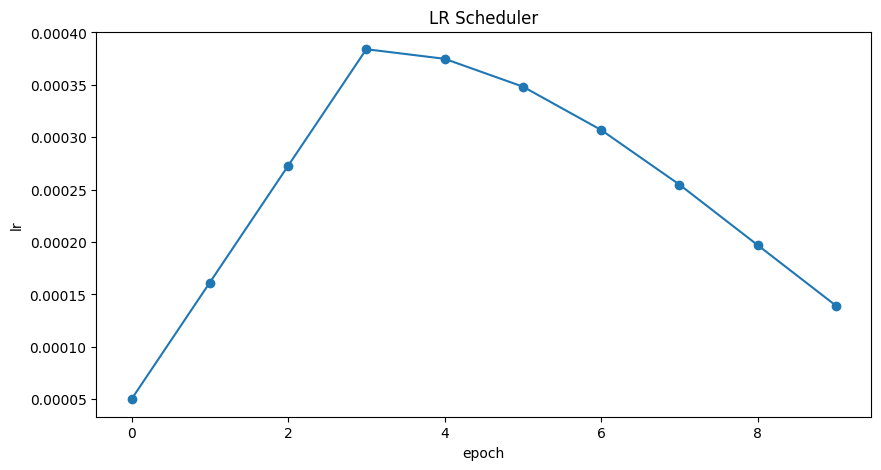
* + **Create ModelCheckpoint callback**: The *keras.callbacks.ModelCheckpoint* function is used to create a callback that saves the model at regular intervals during training. The parameters specify that the model should be saved to a file named "best\_model.keras", that the model should only be saved when its validation loss (val\_loss) improves, that the entire model (not just the weights) should be saved, and that 'improvement' is defined as a decrease in val\_loss (mode='min').
* ***Training***
  + The *fit* method is used to train the model for a specified number of epochs (Config.epochs). The training data (train\_dataset), validation data (valid\_dataset), and callbacks (callbacks) are passed as arguments. The steps\_per\_epoch and validation\_steps parameters are set to ensure that each epoch goes through the entire dataset.
* ***Building test data and Predictions***
  + **Build Test Dataset**: The *build\_dataset* function is used to create the test dataset from the paths specified in test\_df.spec2\_path.values. The batch size is set to the minimum of *Config.batchSize* and the length of *test\_df*. The dataset is not repeated, shuffled, cached, or augmented, as these operations are typically only done during training.
  + **Make Predictions**: The predict method of the model is used to make predictions on the *test\_ds* dataset. The predictions are stored in the preds variable.

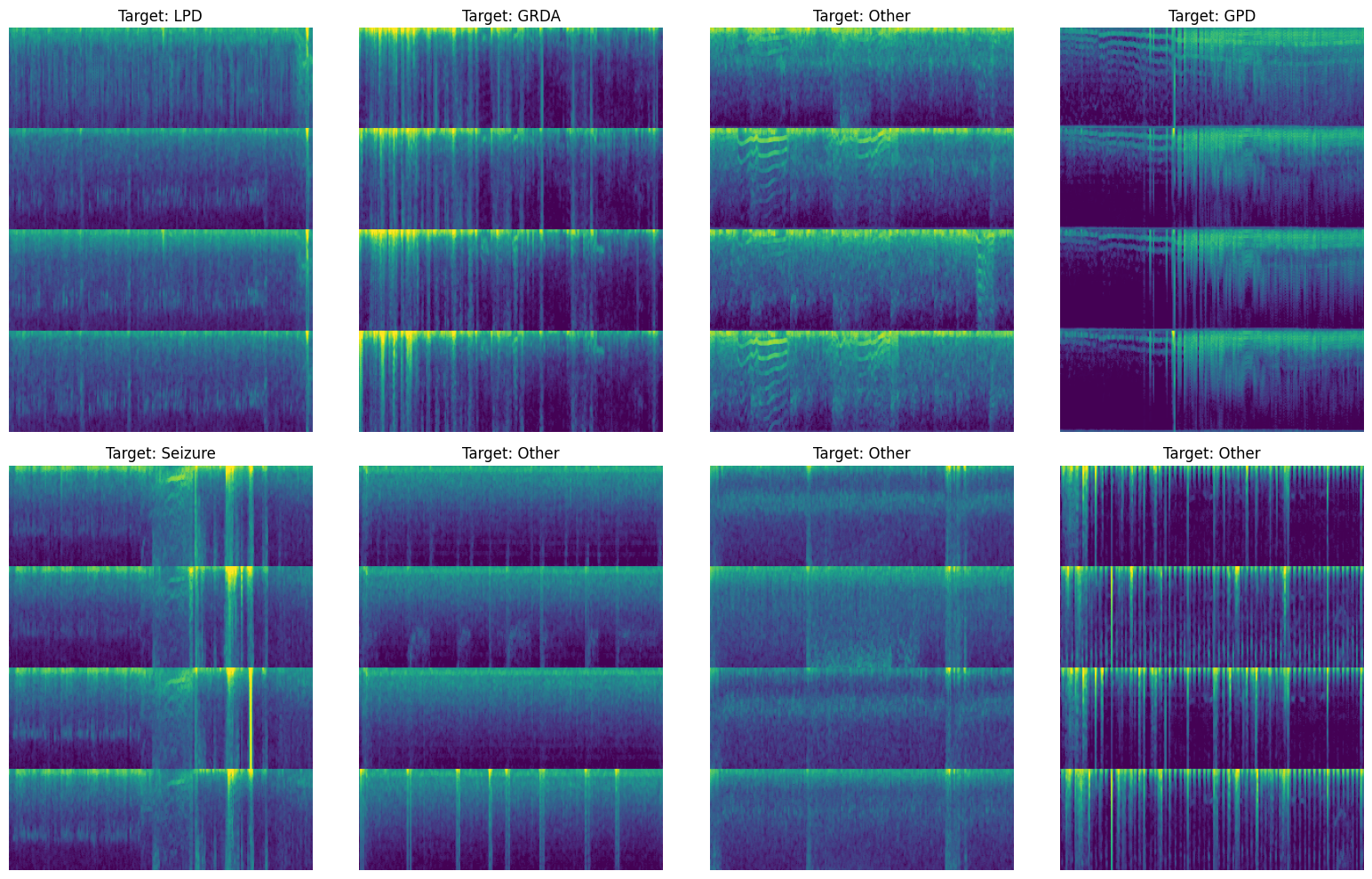
4.2 Testing and verification plan

* ***Unit Testing***: Write unit tests for each function in the script. This includes *build\_dataset, get\_lr\_callback*, and any other helper functions. These tests should verify that each function behaves as expected when given a variety of inputs.
* ***Integration Testing***: Test the script as a whole to ensure that all parts work together correctly. This includes testing the training and prediction processes to ensure that they complete without errors and produce reasonable results.
* ***Performance Testing***: Monitor the performance of the model during training and prediction. This includes tracking the loss and accuracy during training, and checking the speed of prediction on the test dataset.
* ***Validation Testing***: Compare the model's predictions on the validation dataset to the true labels to measure its performance. This can be done using metrics such as accuracy, precision, recall, and F1 score.
* ***Test Dataset Evaluation***: After training the model, evaluate its performance on the test dataset. This will give an unbiased estimate of the model's performance on new, unseen data.

4.3 Result Analysis





Chapter 5

Standards Adopted

5.1 Design Standards:

In the development of the Harmful Brain Activity Classification system, adherence to established design standards is crucial for ensuring consistency, maintainability, and scalability. The following design standards were adopted:

* IEEE Standards: IEEE provides comprehensive guidelines for software and system design. Adhering to IEEE standards ensures compliance with industry best practices and promotes interoperability.
* ISO Standards: ISO standards offer internationally recognized frameworks for quality management and process improvement. Integration of ISO standards enhances the reliability and effectiveness of the design process.
* UML Diagrams: Unified Modeling Language (UML) diagrams serve as a standardized notation for visualizing, specifying, constructing, and documenting software systems. Utilizing UML diagrams facilitates clear communication and understanding of system architecture and design.

5.2 Coding Standards:

Maintaining consistent coding standards is essential for promoting readability, collaboration, and maintainability of the source code. The following coding standards were adhered to during the development of the Harmful Brain Activity Classification system:

* Conciseness: Write code that is as comapact as possible without sacrificing clarity or functionality.
* Naming Conventions: Use meaningful and descriptive names for variables, functions, and classes to enhance code comprehensibility.
* Code Organization: Segment blocks of code into logical sections and paragraphs, making it easier to navigate and understand the codebase.
* Indentation: Utilize consistent indentation to denote the beginning and end of control structures, improving code readability and structure.
* Modularity: Avoid lengthy functions and aim for modular code design where each function performs a single, well-defined task.
* Documentation: Document code extensively using comments and documentation tools to aid comprehension and future maintenance efforts.

5.3 Testing Standards:

Quality assurance and testing play a crucial role in ensuring the reliability, accuracy, and performance of the Harmful Brain Activity Classification system. The following standards were followed for testing and verification:

* ISO/IEC 25000: ISO/IEC 25000 series provides standards for software product quality requirements and evaluation. Adherence to ISO/IEC 25000 standards ensures systematic evaluation and validation of the system's functionality and performance.
* IEEE 829: IEEE 829 standardizes software test documentation, including test plans, test cases, and test reports. Following IEEE 829 facilitates comprehensive test coverage and documentation, aiding in the identification and resolution of defects.
* ISTQB: The International Software Testing Qualifications Board (ISTQB) provides a standardized approach to software testing methodologies and practices. Adhering to ISTQB principles ensures systematic and rigorous testing processes, enhancing the overall quality and reliability of the system.

By aligning with established design, coding, and testing standards, the Harmful Brain Activity Classification system maintains high levels of quality, reliability, and effectiveness, contributing to its successful development and deployment in clinical and research environments.

Chapter 6

Conclusion and Future scope

­­­­­­­­­­6.1 Conclusion

In this project, we tackled the task of classifying seizures and other patterns of harmful brain activity in critically ill patients using deep learning techniques. We leveraged spectrogram data from EEG recordings and employed EfficientNetV2, a state-of-the-art neural network architecture, for the classification task. Through the implementation of various components such as data loading, preprocessing, model building, training, inference, and submission, we developed a robust pipeline for the classification task.

Key achievements of the project include:

* Efficient data loading and preprocessing using tf.data to handle large datasets efficiently.
* Utilization of KerasCV and Keras libraries for model building, which allowed us to seamlessly switch between different deep learning frameworks such as TensorFlow, PyTorch, and JAX.
* Incorporation of data augmentation techniques such as MixUp and random cutout to enhance the model's generalization ability.
* Implementation of a well-structured learning rate scheduler to optimize the training process.
* Training of the model using KL Divergence loss, which is directly applicable as the evaluation metric, eliminating the need for separate evaluation metrics.
  1. Future Scope

The future scope of this project could encompass several areas:

* **Technology Upgrades**: The performance of the model could be enhanced by leveraging new and more powerful processors and accelerators. The use of different programming languages, such as C, could also be explored for potential performance benefits. The integration of advanced technologies like brain chips, such as Neurolink, could also be considered to further improve the accuracy and efficiency of seizure detection and classification.
* **Performance Improvements**: The model could be fine-tuned to improve its accuracy and efficiency. Techniques such as efficient parallelization of data loading could be employed to speed up the training and prediction processes.
* **Scalability**: The model could be deployed on the cloud to cater to a larger number of users without compromising on performance. This would also allow for easy scaling up or down based on demand.
* **Security Enhancements**: Measures could be implemented to improve data security, such as enhanced data encryption, abstraction, audit trails, and informed consent mechanisms. This would ensure the privacy and confidentiality of patient data.
* **User Interface Improvements**: The user interface could be enhanced to provide better data visualization, real-time feedback, and aesthetic improvements. This would improve the user experience and make the system more intuitive to use.
* **Integration with Other Services**: The model could be integrated with other similar models, such as heart disease predictors, to provide a more comprehensive health assessment. This could revolutionize medical diagnosis and significantly improve the quality of healthcare.
* **Real-time Monitoring**: The system could be enhanced to support real-time monitoring of EEG signals. This would allow for immediate detection and classification of seizures, enabling faster intervention and treatment.
* **Personalized Care**: The model could be further developed to provide personalized predictions based on individual patient characteristics. This would improve the accuracy of the predictions and provide more personalized care for each patient.
* **Research and Development**: The system could be used as a platform for further research and development in the field of neurology. The data collected could be used to gain new insights into brain activity and the causes of seizures, potentially leading to the development of new treatments and therapies.

**References:**

* <https://www.kaggle.com/competitions/hms-harmful-brain-activity-classification/overview>
* <https://www.mdpi.com/1424-8220/23/13/6001>
* <https://bme.unc.edu/wp-content/uploads/sites/917/2022/11/EEG_explanation.pdf>
* <https://iopscience.iop.org/article/10.1088/1741-2552/ab260c>
* <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8314009/>
* <https://inria.hal.science/hal-01846433/document>
* <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4677116/>