Project 2 Report Paper:

Harmful Brain activity Classification with KerasCV

Joseph Martínez

Old Dominion University, jmart130@odu.edu

*Abstract* - The Kaggle challenge HMS - Harmful Brain Activity Classification focuses on automating the analysis of electroencephalography (EEG) signals to detect seizures and harmful brain activity in critically ill patients. Hosted by Sunstella Foundation, the competition aims to enhance EEG pattern classification accuracy, crucial for neurocritical care and drug development. Currently, manual EEG analysis is time-consuming and prone to errors. Participants are tasked with developing models to classify EEG patterns, including seizures and various discharges. Annotated EEG segments range from well-agreed idealized patterns to edge cases with disagreements among experts. Successful algorithms could expedite accurate treatments, benefiting neurology and pharmaceutical research.

*Index Terms* – EEG signals, seizure detection, pattern classification, neurocritical care.

Problem Definition

The HMS - Harmful Brain Activity Classification Kaggle competition [1] addresses the pressing need for automating the detection and classification of seizures and other harmful brain activities in critically ill patients. Presently, electroencephalography (EEG) monitoring in neurocritical care heavily relies on manual interpretation by specialized neurologists. This process is not only time-consuming but also prone to errors, leading to delays in treatment and potential misdiagnoses. Moreover, the labor-intensive nature of manual EEG analysis poses significant challenges, including high costs, fatigue-related errors, and inconsistencies between different reviewers, even when they are experts in the field.

One of the primary challenges in EEG analysis is the variability in EEG patterns, which can range from well-defined seizures to subtle abnormalities. Additionally, there can be disagreements among experts regarding the classification of EEG segments, further complicating the diagnostic process. This variability and disagreement highlight the need for automated methods capable of accurately classifying EEG patterns in real-time.

The primary objective of this competition is to develop machine learning models that can accurately classify EEG segments into specific patterns, including seizures, generalized periodic discharges, lateralized periodic discharges, and other relevant categories.

By leveraging EEG data recorded from critically ill patients

and advancing state-of-the-art machine learning techniques, participants are tasked with creating models that will not only expedite the diagnostic process but also improve the accuracy and reliability of EEG analysis, ultimately leading to better patient outcomes in neurocritical care settings.

Methods

We adapted the code *HMS-HBAC: KerasCV Starter Notebook[[1]](#footnote-1)* from by the Keras team, utilizing KerasCV[[2]](#footnote-2) [2] and Keras[[3]](#footnote-3) (JAX as preferred backend), changing the model but keeping the preprocessing, training process, and evaluation the same. Their solution presented for the HMS - Harmful Brain Activity Classification Kaggle competition employs a Deep Learning model, specifically EfficientNetV2 with spectrograms of the EEG data to classify the patterns.

KerasCV is a library that extends the capabilities of Keras, a popular deep learning framework. It provides additional functionalities and pre-implemented models for various computer vision tasks, making it easier for developers to build and experiment with deep learning models for image-related tasks [2]. The original network used the preset for EfficientNetV2, but KerasCV has a model preset library[[4]](#footnote-4) of over 20 models.

Our submission used the model MobileNetV3 instead since it generally achieves good performance on image classification tasks while maintaining efficiency. The model has a lightweight design optimized for mobile and edge devices, making it perfect for use outside of servers with big performance. With this change, we aim that with MobileNetV3, users can try and test the resulting model in mobile and smaller devices.

This section outlines the methodology, including data preprocessing, model architecture, training process, and evaluation.

## Data Preprocessing:

The data preprocessing pipeline for the classification task involves several steps to prepare the spectrogram data for model training:

1. **Convert .parquet to .npy:** The original spectrogram data is stored in .parquet files. To facilitate easier data loading and processing, the spectrogram data is converted to .npy format. This process involves reading the .parquet files, filling any NaN values with zeros, transposing the data to rearrange the dimensions from (Time, Frequency) to (Frequency, Time), and saving the data as .npy files.
2. **Data Loading**: The DataLoader reads the preprocessed .npy spectrogram files and extracts labeled subsamples using specified offset values. These offset values determine the temporal segment of the spectrogram to be used for training. The DataLoader also converts the spectrogram data into log spectrograms to enhance feature representation.
3. **Data Augmentation:** Data augmentation techniques are applied to enhance the model's ability to generalize and improve robustness. Augmentation techniques such as MixUp, frequency masking, and time masking are employed to introduce variations in the spectrogram data. MixUp interpolates between two examples to create new training samples, while frequency and time masking randomly mask out portions of the spectrogram data.
4. **Data Split:** The data is divided into training and validation sets using a Stratified Group K-Fold cross-validation strategy. This ensures that each fold contains a balanced distribution of class labels and avoids potential data leakage issues by preventing overlap of patients between the training and validation sets.
5. **Build Train & Valid Dataset:** Only the first sample for each spectrogram\_id is used to keep the dataset size manageable. The training and validation datasets are constructed using the DataLoader, specifying the paths to the spectrogram files, offsets for labeled subsamples, class labels, batch size, and other parameters. The datasets are then prepared for model training and validation.

## Model Architecture:

The model architecture utilized the original notebook employs the EfficientNetV2 B2, a convolutional neural network (CNN) model from KerasCV's collection of pretrained models. EfficientNetV2 is known for its effectiveness in various computer vision tasks due to its balanced architecture, which efficiently scales the network's depth, width, and resolution. The architecture consists of several key components (see Figure 2):

* Input Layer: Accepts input images with three color channels (RGB).
* EfficientNetV2 B2 Backbone: The backbone of the model, responsible for extracting features from input images. It consists of multiple layers that perform convolutional operations to extract hierarchical representations of input images.
* Global Average Pooling 2D Layer (avg\_pool): A pooling layer that reduces the spatial dimensions of the feature maps while retaining important information. It computes the average value of each feature map, resulting in a fixed-size output.
* Predictions Layer (predictions): The output layer of the model, which consists of a dense layer with six units, corresponding to the six classes of EEG patterns to be classified.

The total number of parameters in the model is 8,777,828, with 8,695,540 trainable parameters and 82,288 non-trainable parameters. This architecture has been pretrained on the ImageNet dataset, making it capable of extracting meaningful features from input EEG signals.

|  |  |  |
| --- | --- | --- |
| **Layer (type)** | **Output shape** | **Param #** |
| input\_layer (InputLayer) | (None, None, None, 3) | 0 |
| efficient\_net\_v2b2\_backbone  (EfficientNetV2Backbone) | (None, None, None, 1408) | 8,769,374 |
| avg\_pool  (GlobalAveragePooling2D) | (None, 1408) | 0 |
| predictions (Dense) | (None, 6) | 8,454 |

Total params: 8,777,828 (33.48 MB)

Trainable params: 8,695,540 (33.17 MB)

Non-trainable params: 82,288 (321.44 KB)

Figure I

Original model architecture

## Training Process:

Training was performed over 13 epochs with a batch size of 64 samples. A cosine learning rate scheduler was employed to adjust the learning rate during training. The training process was executed with verbosity level set to 1, providing detailed output information during each epoch.

## Evaluation:

The evaluation metric in this competition is KL Divergence:

Where *P* is the true distribution and *Q* is the predicted distribution. We used it directly as our loss function, removing the need of a third-party metric like Accuracy to evaluate our model. Therefore, valid loss can stand alone as an indicator for our evaluation. We used the implementation for KL Divergence loss in Keras:

keras.losses.KLDivergence()

Changes to Original Architecture

## Model Architecture:

In contrast to the original implementation utilizing EfficientNetV2, we opted for the utilization of MobileNetV3 for the model architecture. This substitution introduces alterations to the network's composition and parameterization, leading to distinctive performance characteristics. The modified architecture comprises the following components (See figure below):

* Input Layer: The initial layer of the network, accepting input images with three color channels (RGB).
* MobileNetV3 Large Backbone: Replacing the EfficientNetV2 backbone, the MobileNetV3 Large Backbone serves as the feature extraction component of the model. It consists of multiple layers specifically tailored to enhance efficiency and performance in mobile and edge computing environments.
* Global Average Pooling 2D Layer (avg\_pool): A pooling layer responsible for reducing the spatial dimensions of the feature maps generated by the backbone. It computes the average value of each feature map across all spatial locations, resulting in a fixed-size output.
* Predictions Layer (predictions): The output layer of the model, consisting of a dense layer with six units corresponding to the six classes of EEG patterns to be classified.

The updated architecture configuration leads to a total of 3,002,118 parameters, with 2,977,718 parameters trainable and 24,400 non-trainable parameters. This substitution aims to explore the performance characteristics of MobileNetV3 in comparison to the original EfficientNetV2 architecture, potentially offering insights into the efficacy of different network architectures for the EEG pattern classification task.

|  |  |  |
| --- | --- | --- |
| **Layer (type)** | **Output shape** | **Param #** |
| input\_layer (InputLayer) | (None, None, None, 3) | 0 |
| mobile\_net\_v3\_large\_backbone (MobileNetV3Backbone) | (None, None, None, 960) | 2,996,352 |
| avg\_pool  (GlobalAveragePooling2D) | (None, 960) | 0 |
| predictions (Dense) | (None, 6) | 5,766 |

Total params: 3,002,118 (11.45 MB)

Trainable params: 2,977,718 (11.36 MB)

Non-trainable params: 24,400 (95.31 KB)

Figure II

Modified model architecture

Results

The performance of the model trained using MobileNetV3 Large as compared to the original model based on EfficientNetV2 is summarized below.

It is evident from the results that both models exhibit a decrease in training loss and validation loss over the course of training epochs. However, the MobileNetV3 Large model demonstrates a slightly higher validation loss compared to the original model across all epochs. This suggests that the MobileNetV3 Large model may have slightly inferior performance in terms of generalization to unseen data compared to the original EfficientNetV2 model.

Further analysis could provide additional insights into the comparative performance of the two models. Additionally, experimentation with hyperparameters and model configurations may lead to improvements in the performance of both models.

The performance of the modified convolutional neural network (CNN) model in classifying foliar diseases in apple trees was evaluated based on several metrics. The obtained results are as follows:

TABLE I

Performance metrics

|  |  |
| --- | --- |
| Metric | Value |
| Accuracy  Precision  Recall  F1 Score | 0.3065  0.0939  0.3065  0.1438 |

For comparison, Table II shows the scored of the top 5 leaderboard of the competition.

TABLE II

Performance of leaderboard top 5 of the competition

|  |  |  |
| --- | --- | --- |
| # | Team | Score |
| 1  2  3  4  5 | baseline  Luminide  Data Oriented  Team Why  mknzfr | 0.88336  0.87560  0.87469  0.87287  0.86851 |

Comparing these metrics with the top performers on the Kaggle leaderboard, it is evident that the model's performance falls significantly short in terms of accuracy and other evaluation metrics. The model achieved an accuracy of 30.65%, which is substantially lower than the top leaderboard entries, which achieved accuracies above 86%.

The relatively low performance of the model can be attributed to several factors:

1. Limited Data and Reduced Training Size: Utilizing only 10% of the original dataset for training, validation, and testing might have resulted in insufficient data for the model to learn complex patterns effectively. This reduction in training data could have led to poor generalization and lower performance.

2. Simplified Model Architecture: The reduction in the number of convolutional layers and the removal of one layer from the original architecture might have resulted in a loss of representational capacity, limiting the model's ability to extract intricate features from the input images.

3. High Class Imbalance: The dataset might have exhibited significant class imbalance, with certain disease categories being overrepresented compared to others. This imbalance could have affected the model's ability to learn from minority classes, leading to lower precision and recall values.

4. Limited Computational Resources: Despite efforts to optimize the model for training on a local GPU with 8GB of RAM, the computational resources might still have been insufficient to train a complex CNN architecture effectively. This limitation could have hindered the model's capacity to learn intricate patterns from the data.

## Lessons Learned

This project highlights several key learnings:

1. Importance of Data Quantity and Quality: Adequate data quantity and diversity are essential for training robust machine learning models, particularly in complex tasks such as image classification. Future iterations of the project should focus on acquiring a larger and more diverse dataset to improve model performance.

2. Model Complexity vs. Computational Resources: Balancing model complexity with available computational resources is crucial. While simplifying the model architecture can improve training speed and resource utilization, it should be done judiciously to avoid compromising performance.

3. Continuous Iteration and Experimentation: Machine learning projects often require iterative development and experimentation. Different model architectures, hyperparameters, and training strategies should be explored systematically to identify the most effective approach.

Conclusion

In conclusion, while the modified CNN model exhibited shortcomings in achieving competitive performance levels compared to the top-performing models on the Kaggle leaderboard, the project has offered valuable insights into the intricacies of disease classification within agricultural settings. The constrained computational resources and the limited training data subset posed significant challenges in training a highly accurate model. Despite these obstacles, the project underscores the importance of continued exploration and innovation in the realm of plant pathology diagnosis.

While the ultimate goal of achieving state-of-the-art performance was not met, the project lays a foundation for future advancements in the field. Moving forward, addressing the limitations observed in this iteration, such as augmenting the dataset with a larger and more diverse collection of images, exploring more sophisticated model architectures, and leveraging advanced training techniques, can lead to the development of more robust and accurate solutions for plant pathology diagnosis. By iteratively refining methodologies and incorporating lessons learned from this endeavor, future iterations of the project hold the promise of realizing the overarching goal of enhancing disease detection and management in agricultural systems.

References

1. Jin, J., *HMS - Harmful Brain Activity Classification*. Kaggle.

2. Wood, L., et al., *KerasCV*. <https://github.com/keras-team/keras-cv>.

Author Information

**Joseph Martínez,** Modeling and Simulation Engineering Master’s student at Old Dominion University (ODU).

1. <https://www.kaggle.com/code/awsaf49/hms-hbac-kerascv-starter-notebook> [↑](#footnote-ref-1)
2. <https://github.com/keras-team/keras-cv> [↑](#footnote-ref-2)
3. <https://github.com/keras-team/keras> [↑](#footnote-ref-3)
4. <https://keras.io/api/keras_cv/models/> [↑](#footnote-ref-4)