Neural Network Analysis of MRI Scans for FND Diagnosis

Mathematics & Computer Science

Samiel H. Azmaien

The Gwinnett School of Mathematics, Science, and Technology

970 McElvaney Lane, Lawrenceville, GA 30044

January 9, 2024

Table of Contents

Acknowledgement of Major Assistance

This research was conducted at both my school and home. I would like to acknowledge that the entirety of this study was carried out independently, without direct assistance from others. I credit the open-source community and pertinent studies in the field of neurological illnesses for their contributions to my background research.

Introduction

Functional Neurological Disorder (FND) has projected misdiagnosis rates of around 50 cases per 1,000 (*Misdiagnosed - FND Hope International*, 2019). This high rate is due to the lack of an efficient diagnostic method for this condition. Currently, the primary method involves using video telemetry electroencephalography tests to measure the brain's electrical activity for extended periods, lasting up to a day, which is very time-inefficient (FND Action, 2023). This inefficiency stems from the difficulty in identifying another neurological condition that better explains the initial symptoms in hindsight (Walzl et al., 2019). Integrating neural network analysis of patient MRI data with clinical history trials could potentially alleviate the difficulties in diagnosing FND. By achieving this, there will be less pressure to evaluate patients quickly due to the inability to describe the condition clearly. It will also assist users of the program in understanding the diagnosis despite lacking training in neuropsychiatric principles (Perez et al., 2020). Continuously refining a machine learning algorithm for FND diagnosis can lead to more accurate and timely diagnoses. This, in turn, could lay the groundwork for future explorations into AI-driven diagnostics in neurology and medicine.

To address the current misdiagnosis issues with FND, the proposed solution involved using a logistic regression model for general neurological disorder classification and a neural network specifically for FND diagnosis. To diagnose FND with the highest accuracy, an

algorithm was created that utilized ROC curve plotting, which illustrated the diagnostic ability of a binary classifier system as its discrimination threshold was varied (*Receiver Operating Characteristic (ROC)*, n.d.). This metric is represented on a scale from 0 to 1, where a score of 1 indicates an ideal, flawless diagnostic test, and a score of 0.5 suggests a test whose accuracy is no better than random chance (*Classification: ROC Curve and AUC*, n.d.). This quantification served as a crucial indicator of the diagnostic method's performance. By applying this multifaceted model, an accurate diagnosis for FND could be achieved.

Methods

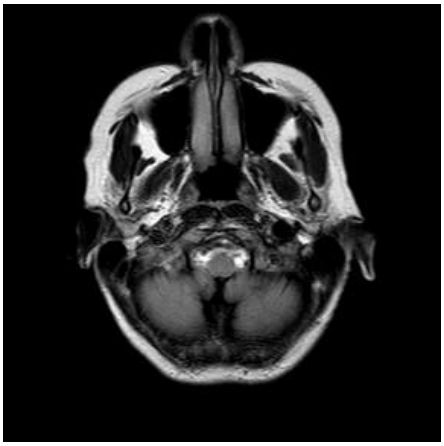


Figure 1: MRI Scan of a patient diagnosed with FND for the training phase (*OSF*, n.d.).

This image is part of a larger dataset of MRI scans from patients diagnosed with FND and a control group without FND. The dataset was divided into training (70%), validation (15%), and test sets (15%). The MRI scans were then pre-processed to extract the relevant features, which included noise reduction, normalization (where a MinMaxScaler was utilized - a process that adjusts the scale of the features), and segmentation (GeeksforGeeks, 2023). A convolutional neural network (CNN) is a type of neural network that utilized multiple layers and pooling steps to classify data or images and this was used to automatically extract the most relevant features from the MRI scans and clinical history trials that could characterize FND (*What are Convolutional Neural Networks? | IBM*, n.d.).

```python
def filter_data(data, high_pass_cutoff=0.5, low_pass_cutoff=35):
    # Apply high-pass filter if the data length exceeds the threshold.
    if len(data) > 8001:
        # Design a high-pass filter
        high_pass_coeffs = signal.butter(N=4, Wn=high_pass_cutoff, btype='high', analog=False)
        # Pad the signal to compensate for the filter delay
        padded_signal = npy.pad(data[:, 1], (0, sum(len(coef) for coef in high_pass_coeffs) - 2), mode='constant')
        # Apply the high-pass filter
        data[:, 1] = signal.filtfilt(*high_pass_coeffs, padded_signal)[len(high_pass_coeffs[0]) - 1 : -len(high_pass_coeffs[1]) + 1]

    # Apply low-pass filter to the (now high-pass-filtered) signal
    low_pass_coeffs = signal.butter(N=4, Wn=low_pass_cutoff, btype='low', analog=False)
    data[:, 1] = signal.filtfilt(*low_pass_coeffs, data[:, 1])
    return data
```

Figure 2: Preprocessing the MRI data using signal processing and feature engineering

The extracted features, along with the clinical history attributes inputted by the user, were employed to train a neural network to predict the FND diagnosis. This model was composed of multiple dense layers, culminating in a sigmoid activation tailored for binary classification tasks. The use of TensorFlow and Keras facilitated the implementation of this model, allowing for a nuanced architecture that could capture complex patterns in the data.

```
def classify_FND(train_data, train_labels, test_data, test_labels, validation_data, validation_labels):
    neural_network = Sequential([
        Dense(units=64, activation='relu', input_dim=train_data.shape[1]),
        Dense(units=64, activation='relu'),
        Flatten(),
        Dense(units=1, activation='sigmoid')
    ])
    # Compile the model with loss and optimizer
    neural_network.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    # Train the model with the training data and validation data
    neural_network.fit(x=train_data, y=train_labels, validation_data=(validation_data, validation_labels), epochs=10, verbose=0)
    # Predict probabilities on the test set
    predictions_prob = neural_network.predict(test_data).flatten()
    # Convert probabilities to binary predictions
    predictions_binary = (predictions_prob > 0.5).astype(int)
    # Perform and display the t-test
    print("Performing t-test between actual labels and predicted probabilities...")
    perform_t_test(test_labels, predictions_prob)
    # Perform and display the ROC curve analysis
    print("Displaying ROC curve...")
    display_roc_curve(test_labels, predictions_prob, "FND Neural Network")
    report = classification_report(test_labels, predictions_prob, output_dict=True)
    display_confusion_matrix(test_labels, predictions_prob, "FND Neural Network")
    display_classification_report(test_labels, predictions_prob, "FND Neural Network")
    display_heatmaps(report, "FND Neural Network")
    # Visualize activation patterns for the first hidden layer
    visualize_activation_patterns(neural_network, 'dense_30', test_data)
    return test_labels, predictions_prob
```

Figure 3: Sequential model with two densely connected hidden layers of 64 neurons

Performance metrics were then computed such as accuracy, precision, recall, F1 score, and area under the AUC-ROC curve for a comprehensive evaluation. The model was then readjusted for the validation image set. FND diagnostic statements were then produced and compared to the original MRI data, indicating if the patient was diagnosed or not.

Results

General Classification Model

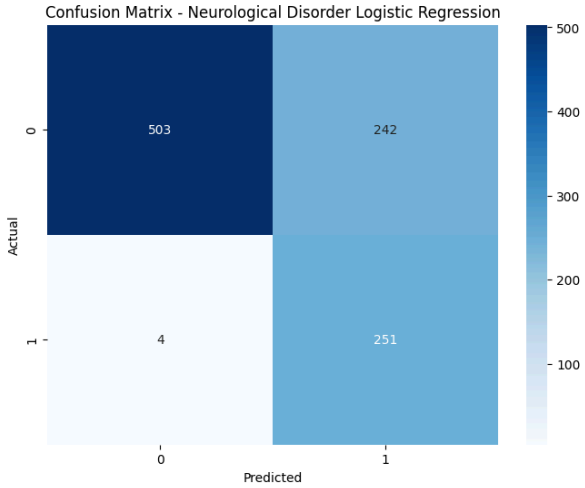


Figure 4: Confusion matrix for general neurological disorder classification

Figure 4 presents the prediction and measured summaries of the general classification model, with 503 true negatives, 242 false positives, 4 false negatives, and 251 true positives. The values of true positives (TP) and true negatives (TN) were fairly maximized, showing a 75.4% accuracy, as depicted in the heatmap below.

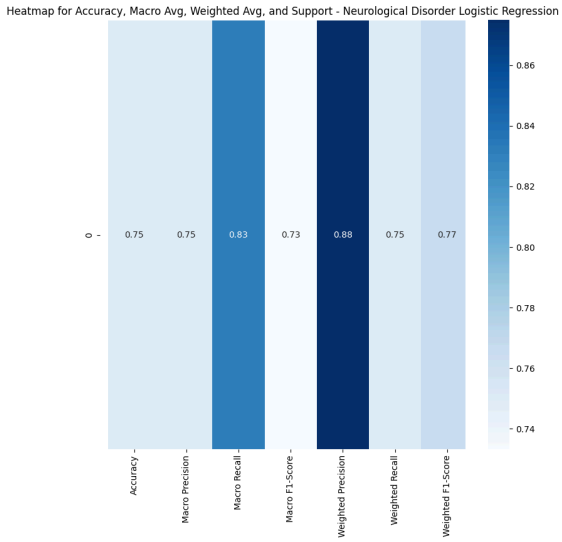


Figure 5: Heatmap for accuracy and F1-score of general neurological disorder classification

The F1-score metric combines accuracy and precision to provide a single measure of the model's robustness. Weighted metrics consider the imbalance in the dataset, offering a nuanced view of the model's performance across different classes, as seen in Figure 5.

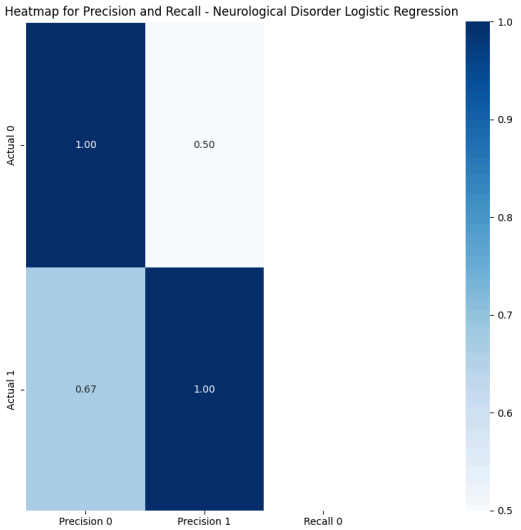


Figure 6: Heatmap for precision and recall for general neurological disorder classification

The general classification model demonstrated perfect precision for both classes, as indicated by a score of 1.00 in Figure 6. Every instance predicted by the model to be positive (Precision 0) or negative (Precision 1) was correct, with no false positives. The recall for the negative class (Actual 0) also reached a score of 1.00, which means the model successfully identified all negative cases without any misses. However, the recall for the positive class (Actual 1) is 0.67, implying that the model misses about a third of the actual positive cases. This may be due to overfitting to the negative class during the training phase, resulting in less effective generalization for positive cases.

FND Neural Network

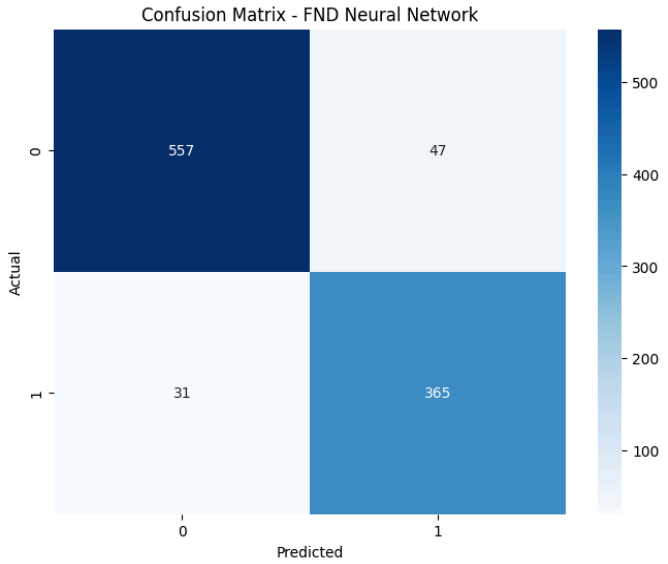


Figure 7: Confusion matrix for neural network analysis of FND

Figure 7 presents the prediction and measured summaries of the FND neural network, with 557 true negatives, 47 false positives, 31 false negatives, and 365 true positives. The values of true positives (TP) and true negatives (TN) were greatly maximized, showing a 92% accuracy, as depicted in the heatmap below.

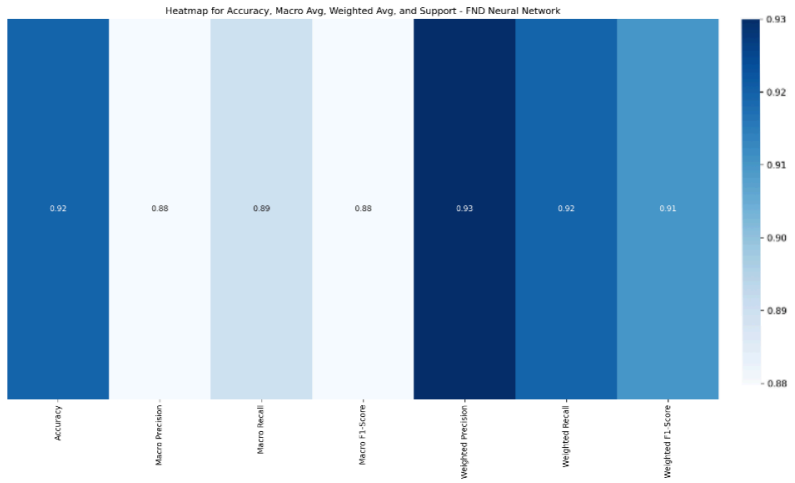


Figure 8: Heatmap for accuracy and F1 score of neural network analysis of FND

Figure 8 shows that the neural network achieved a 92% accuracy in diagnosing FND. While this provided a statistically significant result regarding the model's capabilities, accuracy alone does not offer information about the model's performance on individual cases, such as the balance between false positives and false negatives. It is necessary to compare this alongside other metrics, such as precision and recall. As demonstrated in Figure 9 below, the recall for the negative class (Actual 0) is 0.33, which is quite low, indicating that the model only identified 33% of the actual negative cases.

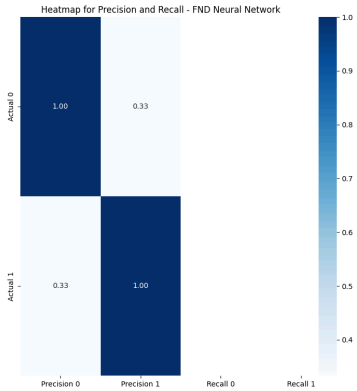
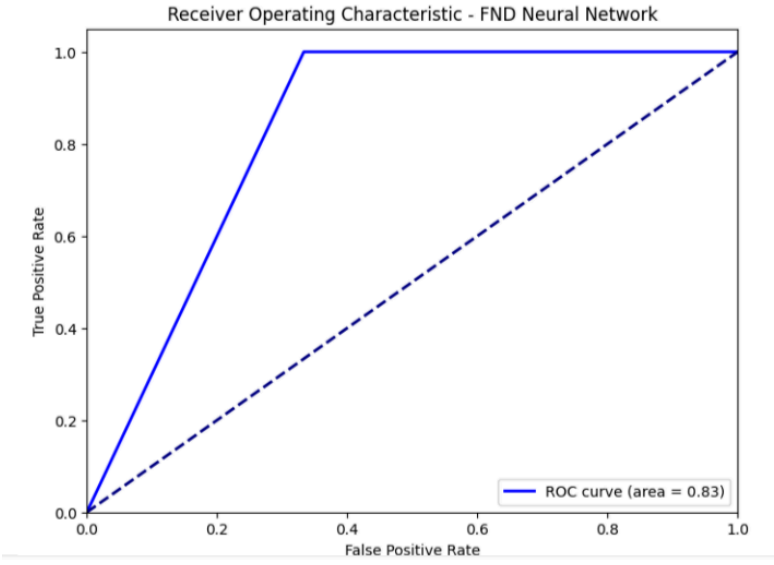


Figure 9: Heatmap for precision and recall for general neurological disorder classification



Graph 1: Matplotlib program that prints out a ROC curve analysis

Graph 1 presents the ROC curve for the neural network's performance, with the AUC being 0.83. This indicated an above-average discriminative power to differentiate patients with and without FND. The integration of this analysis with a diagnostic statement would allow for more accurate diagnoses of FND in patients.

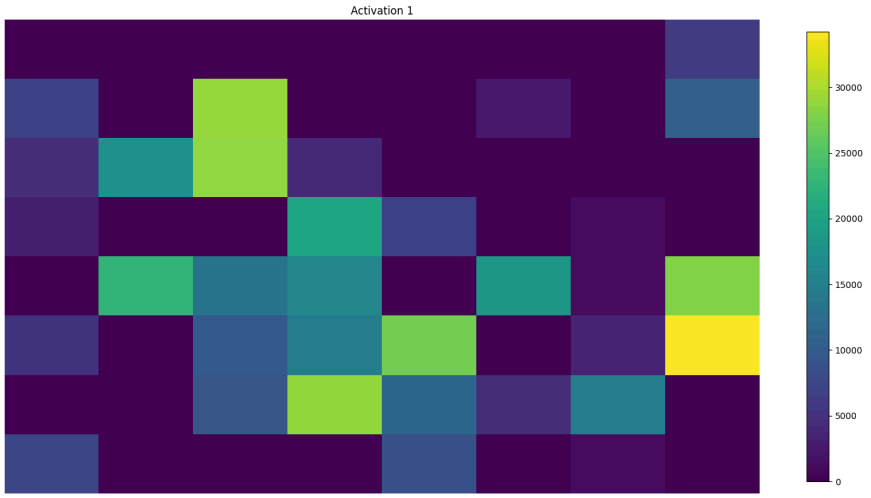
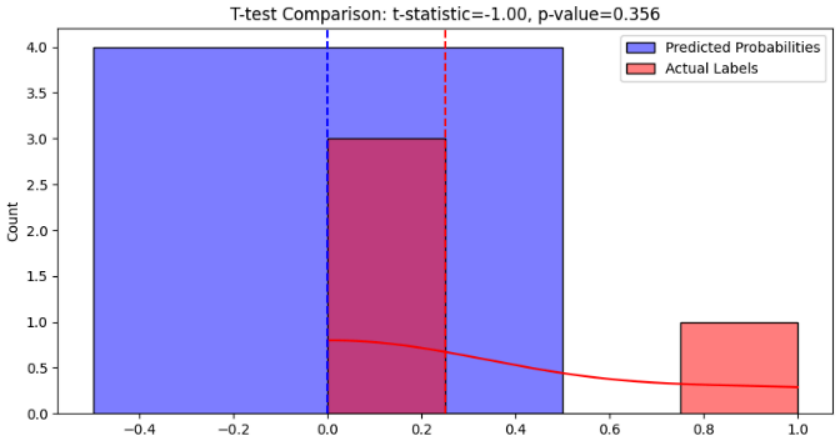


Figure 10: Heatmap for neuron activation on the first hidden layer - 'dense-30'

Figure 10 shows a heatmap that depicts the intensity of activations across a grid of neurons in response to features extracted from MRI scans. This illustrates the steps in the neural network's process of learning to differentiate FND cases from non-cases.



Graph 2: T-test for comparing group means

In Graph 2 above, a t-test was performed to compare the means of the predictions from the neural network against the actual labels, assuming they are continuous, or against another set of predictions. The distributions of the y_pred and y_true labels were then plotted with a kernel density estimate to visualize that there was no statistical difference in the means.

```
def perform_t_test(y_true, y_pred_probs):
    # Calculate the t-statistic and the p-value
    t_stat, p_val = ttest_ind(y_pred_probs, y_true)
    print(f"T-test result: t-statistic={t_stat:.2f}, p-value={p_val:.3f}")
    # Plot the histogram of predicted probabilities and actual labels
    mpl.figure(figsize=(10, 6))
    sbn.histplot(y_pred_probs, color="blue", label="Predicted Probabilities", kde=False)
    sbn.histplot(y_true, color="red", label="Actual Labels", kde=False, bins=[-0.5, 0.5, 1.5])
    # Plot the means of both distributions
    mpl.axvline(npy.mean(y_pred_probs), color="blue", linestyle='--')
    mpl.axvline(npy.mean(y_true), color="red", linestyle='--')
    # Annotate the t-statistic and p-value in the plot
    mpl.text(0.5, max(mpl.ylim()), f't-statistic={t_stat:.2f}', ha='center', va='bottom', color="blue")
    mpl.text(0.5, max(mpl.ylim()), f'p-value={p_val:.3f}', ha='center', va='top', color="red")
    # Add the legend, title, and show the plot
    mpl.legend()
    mpl.title('T-test Comparison')
    mpl.xlabel('Value')
    mpl.ylabel('Frequency')
    mpl.show()
```

Figure 11: Function for performing the t-test with KDE to visualize difference in means

Discussion and Conclusions

Classification of general neurological disorders was completed as it aided in the process for the neural network to classify Functional Neurological Disorder. This general classification model served as a baseline comparison to compare the performance of models focused on more specific conditions with a broader model. The model was also able to properly differentiate between FND and a negative/positive state for other neurological conditions based on the general model being present, which was crucial in achieving high accuracy for the FND model. The dataset used for the general classification also aided in fine-tuning feature engineering as the general model allowed the FND model to already have a context of the relevant features to identify.

After completing all three stages of training, validating, and testing the data, a high accuracy score of 92% was reached in diagnosing patients with FND. The considerably higher accuracy of the FND neural network than that of the general classification model might be attributed to continuous refinement and optimization through the validation process. The dataset obtained for the general classification model was also separate from the FND model, which may explain why the data is skewed. Another factor to consider is that an overlap in neurological conditions would make it more challenging for the logistic regression model to distinguish between them accurately. The relatively small batch size of 1,000 images among 10-epoch intervals may have led to an inflated accuracy score. Likely, with a larger batch size, the model would initially have a smaller accuracy.

While the political and social implications of utilizing artificial intelligence in the medical industry have been considered, this study aimed to provide a new perspective on a fairly accurate and optimized integration of neural networks into patient diagnoses. However, an

immediate conversion to using artificial intelligence is not ready yet. Further experimentation still needs to be conducted before the safety of patients is entrusted to computational models. Despite this, the usage of machine learning, particularly neural networks, holds significant potential for improving the diagnosis of not only Functional Neurological Disorder but also other hard-to-treat conditions as well.

References

Classification: ROC curve and AUC. (n.d.). Google for Developers. https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc

FND Action. (2023, October 18). *Diagnosis - FND Action*. https://www.fndaction.org.uk/diagnosis/

GeeksforGeeks. (2023, April 24). *StandardScaler, MinMaxScaler, and RobustScaler techniques - ML*. https://www.geeksforgeeks.org/standardscaler-minmaxscaler-and-robustscaler-techniques-ml/

Misdiagnosed - FND Hope International. (2019, December 16). FND Hope International. https://fndhope.org/fnd-guide/diagnosis/misdiagnosed/

OSF. (n.d.). https://osf.io/

Perez, D. L., Hunt, A. R., Sharma, N., Flaherty, A. W., Caplan, D., & Schmahmann, J. D. (2020). Cautionary notes on diagnosing Functional Neurologic Disorder as a neurologist-in-training. *Neurology Clinical Practice*, *10*(6), 484–487. https://doi.org/10.1212/cpj.0000000000000779

Receiver operating characteristic (ROC). (n.d.). Scikit-learn. https://scikit-learn.org/1.1/auto_examples/model_selection/plot_roc.html

Walzl, D., Carson, A., & Stone, J. (2019). The misdiagnosis of functional disorders as other neurological conditions. *Journal of Neurology*, *266*(8), 2018–2026. https://doi.org/10.1007/s00415-019-09356-3

What are convolutional neural networks? | IBM. (n.d.). https://www.ibm.com/topics/convolutional-neural-networks

Appendix

```python
import os
import pandas as pds
import numpy as npy
import matplotlib.pyplot as mpl
import seaborn as sbn
from scipy import signal
from scipy.stats import ttest_ind
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import (
    confusion_matrix,
    classification_report,
    roc_auc_score,
    roc_curve,
    precision_recall_fscore_support,
)
from sklearn.linear_model import LogisticRegression
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
import tensorflow as tf
def read_data(file):
    data = []
    df = pds.read_csv(file)
    data.append(df)
    return pds.concat(data)
```

```python
def filter_data(data, high_pass_cutoff=0.5, low_pass_cutoff=35):
    # Apply high-pass filter if the data length exceeds the threshold.
    if len(data) > 8001:
        # Design a high-pass filter
        high_pass_coeffs = signal.butter(N=4, Wn=high_pass_cutoff, btype='high', analog=False)
        # Pad the signal to compensate for the filter delay
        padded_signal = npy.pad(data[:, 1], (0, sum(len(coef) for coef in high_pass_coeffs) - 2), mode='constant')
        # Apply the high-pass filter
        data[:, 1] = signal.filtfilt(*high_pass_coeffs, padded_signal)[len(high_pass_coeffs[0]) - 1 : -len(high_pass_coeffs[1]) + 1]

    # Apply low-pass filter to the (now high-pass-filtered) signal
    low_pass_coeffs = signal.butter(N=4, Wn=low_pass_cutoff, btype='low', analog=False)
    data[:, 1] = signal.filtfilt(*low_pass_coeffs, data[:, 1])
    return data
def partition_dataset(features, target, test_ratio=0.2, validation_ratio=0.25):
    X_train, X_temp, y_train, y_temp = train_test_split(features, target, test_size=test_ratio, random_state=42)
    X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=validation_ratio, random_state=42)
    return X_train, X_val, X_test, y_train, y_val, y_test
def display_confusion_matrix(y_true, y_pred, model_name):
    cm = confusion_matrix(y_true, y_pred)
    mpl.figure(figsize=(8, 6))
    sbn.heatmap(cm, annot=True, fmt="d", cmap="Blues")
    mpl.title(f"Confusion Matrix - {model_name}")
    mpl.ylabel('Actual')
    mpl.xlabel('Predicted')
    mpl.show()
```

```python
def display_classification_report(y_true, y_pred, model_name):
    report = classification_report(y_true, y_pred, output_dict=True)
    df_report = pds.DataFrame(report).round(2)
    print(f"Classification Report - {model_name}:\n", df_report)
def display_heatmaps(report, model_name):
    precision_recall = npy.array([report['0']['precision'], report['1']['precision'], report['0']['recall'], report['1']['recall']])
    accuracy_macro_avg_weighted_avg = npy.array([report['accuracy'], report['macro avg']['precision'], report['macro avg']['recall'],
                                                 report['macro avg']['f1-score'], report['weighted avg']['precision'],
                                                 report['weighted avg']['recall'], report['weighted avg']['f1-score']])
    fig, axes = mpl.subplots(1, 2, figsize=(15, 6))
    sbn.heatmap(precision_recall.reshape(2, 2), annot=True, cmap="Blues", xticklabels=["Precision 0", "Precision 1", "Recall 0", "Recall 1"],
                yticklabels=["Actual 0", "Actual 1"], fmt=".2f", ax=axes[0])
    axes[0].set_title(f"Heatmap for Precision and Recall - {model_name}")
    xticklabels = ["Accuracy", "Macro Precision", "Macro Recall", "Macro F1-Score", "Weighted Precision", "Weighted Recall", "Weighted F1-Score"]
    sbn.heatmap(accuracy_macro_avg_weighted_avg.reshape(1, -1), annot=True, cmap="Blues", xticklabels=xticklabels, fmt=".2f", ax=axes[1])
    axes[1].set_title(f"Heatmap for Accuracy, Macro Avg, Weighted Avg - {model_name}")
    mpl.show()
def visualize_activation_patterns(model, layer_name, input_data):
    # Get the intermediate layer output
    intermediate_layer_model = tf.keras.models.Model(inputs=model.input, outputs=model.get_layer(layer_name).output)
    activations = intermediate_layer_model.predict(input_data)
    # Ensure activations are 4D
    if activations.ndim == 2:
        # Add dimensions to make it 4D for individual activations
        activations = activations.reshape(activations.shape[0], activations.shape[1], 1, 1)
    # Calculate the number of grid rows and columns
    num_activations = activations.shape[-1]
    num_cols = int(npy.ceil(npy.sqrt(num_activations)))
    num_rows = int(npy.ceil(num_activations / num_cols))
    # Set up the figure
    fig, axes = mpl.subplots(num_rows, num_cols, figsize=(num_cols * 3, num_rows * 3))
```

```
    fig, axes = mpl.subplots(num_rows, num_cols, figsize=(num_cols * 3, num_rows * 3))
    fig.subplots_adjust(hspace=0.5, wspace=0.5)
    # Ensure axes is an array even if there is only one subplot
    if num_activations == 1:
        axes = npy.array([axes])
    # Flatten axes array
    axes_flat = axes.flatten()
    # Plot each activation
    for i in range(num_activations):
        ax = axes_flat[i]
        activation = activations[0, :, :, i].squeeze()
        if activation.ndim == 1:
            activation = activation.reshape(int(npy.sqrt(activation.shape[0])), -1)
        im = ax.imshow(activation, cmap='viridis', interpolation='nearest', aspect='auto')
        ax.set_title(f'Activation {i + 1}')
        ax.axis('off')
    # Hide any unused axes
    for i in range(num_activations, len(axes_flat)):
        axes_flat[i].axis('off')
    # Add color bar
    fig.colorbar(im, ax=axes.ravel().tolist(), shrink=0.95)
    mpl.show()
def classify_neurological_disorder(training_features, training_labels, testing_features, testing_labels):
   # Normalize the feature space
    feature_scaler = StandardScaler()
    normalized_training_features = feature_scaler.fit_transform(training_features)
    normalized_testing_features = feature_scaler.transform(testing_features)
    # Initialize and train the logistic regression model
    disorder_predictor = LogisticRegression()
    disorder_predictor.fit(normalized_training_features, training_labels)
```

```
    disorder_predictor.fit(normalized_training_features, training_labels)
    # Predict the disorders using the testing set
    disorder_predictions = disorder_predictor.predict(normalized_testing_features)
    # Compile a report on the model's performance
    performance_report = classification_report(testing_labels, disorder_predictions, output_dict=True)
    # Visualizations
    model_title = "Logistic Regression - Neurological Disorder Prediction"
    display_confusion_matrix(testing_labels, disorder_predictions, model_title)
    display_classification_report(testing_labels, disorder_predictions, model_title)
    display_heatmaps(performance_report, model_title)
def perform_t_test(y_true, y_pred_probs):
    # Calculate the t-statistic and the p-value
    t_stat, p_val = ttest_ind(y_pred_probs, y_true)
    print(f"T-test result: t-statistic={t_stat:.2f}, p-value={p_val:.3f}")
    # Plot the histogram of predicted probabilities and actual labels
    mpl.figure(figsize=(10, 6))
    sbn.histplot(y_pred_probs, color="blue", label="Predicted Probabilities", kde=False)
    sbn.histplot(y_true, color="red", label="Actual Labels", kde=False, bins=[-0.5, 0.5, 1.5])
    # Plot the means of both distributions
    mpl.axvline(npy.mean(y_pred_probs), color="blue", linestyle='--')
    mpl.axvline(npy.mean(y_true), color="red", linestyle='--')
    # Annotate the t-statistic and p-value in the plot
    mpl.text(0.5, max(mpl.ylim()), f't-statistic={t_stat:.2f}', ha='center', va='bottom', color="blue")
    mpl.text(0.5, max(mpl.ylim()), f'p-value={p_val:.3f}', ha='center', va='top', color="red")
    # Add the legend, title, and show the plot
    mpl.legend()
    mpl.title('T-test Comparison')
    mpl.xlabel('Value')
    mpl.ylabel('Frequency')
    mpl.show()
```

```
def display_roc_curve(y_true, y_pred_probs, model_name):
    fpr, tpr, thresholds = roc_curve(y_true, y_pred_probs)
    roc_auc = roc_auc_score(y_true, y_pred_probs)
    mpl.figure(figsize=(8, 6))
    mpl.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (area = {roc_auc:.2f})')
    mpl.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    mpl.xlim([0.0, 1.0])
    mpl.ylim([0.0, 1.05])
    mpl.xlabel('False Positive Rate')
    mpl.ylabel('True Positive Rate')
    mpl.title(f'Receiver Operating Characteristic - {model_name}')
    mpl.legend(loc="lower right")
    mpl.show()
def classify_FND(train_data, train_labels, test_data, test_labels, validation_data, validation_labels):
    neural_network = Sequential([
        Dense(units=64, activation='relu', input_dim=train_data.shape[1]),
        Dense(units=64, activation='relu'),
        Flatten(),
        Dense(units=1, activation='sigmoid')
    ])
    # Compile the model with loss and optimizer
    neural_network.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    # Train the model with the training data and validation data
    neural_network.fit(x=train_data, y=train_labels, validation_data=(validation_data, validation_labels), epochs=10, verbose=0)
    # Predict probabilities on the test set
    predictions_prob = neural_network.predict(test_data).flatten()
    # Convert probabilities to binary predictions
    predictions_binary = (predictions_prob > 0.5).astype(int)
    # Perform and display the t-test
    print("Performing t-test between actual labels and predicted probabilities...")
    perform_t_test(test_labels, predictions_prob)
```

```
    # Perform and display the ROC curve analysis
    print("Displaying ROC curve...")
    display_roc_curve(test_labels, predictions_prob, "FND Neural Network")
    report = classification_report(test_labels, predictions_prob, output_dict=True)
    display_confusion_matrix(test_labels, predictions_prob, "FND Neural Network")
    display_classification_report(test_labels, predictions_prob, "FND Neural Network")
    display_heatmaps(report, "FND Neural Network")
    # Visualize activation patterns for the first hidden layer
    visualize_activation_patterns(neural_network, 'dense_30', test_data)
    return test_labels, predictions_prob
def FND_diagnosis_statement(y_test, y_pred_proba):
    roc_auc = roc_auc_score(y_test, y_pred_proba)
    diagnosis = "The patient is diagnosed with FND." if roc_auc >= 0.7 else "The patient is not diagnosed with FND."
    return diagnosis, roc_auc
def main():
    file = "C:\\Users\\unite\\OneDrive\\Desktop\\research\\Neurological_Disorder_Classification_and_FND_Diagnosis\\csv"
    data = read_data(file)
    filtered_data = filter_data(data)
    labels = npy.array([0 if i < len(filtered_data) / 2 else 1 for i in range(len(filtered_data))])
    data_npy = filtered_data.to_numpy()
    X_train, X_test, X_val, y_train, y_test, y_val = partition_dataset(data_npy, labels)
    print("Classifying neurological disorder:")
    classify_neurological_disorder(X_train, y_train, X_test, y_test)
    print("Classifying FND with Neural Network:")
    y_test_nn, y_pred_nn = classify_FND(X_train, y_train, X_test, y_test, X_val, y_val)
    diagnosis, roc_auc = FND_diagnosis_statement(y_test_nn, y_pred_nn)
    print("\nDiagnostic Statement:")
    print(diagnosis)
    print("ROC AUC: {:.2f}".format(roc_auc))
if __name__ == '__main__':
    main()
```

Seven screenshots of the source code

	cjv	cnr	efc	fber	fwhm_avg	fwhm_x	fwhm_y	fwhm_z	icvs_csf
0	0.544640	1.433259	0.533018	312.469818	4.69318	4.31190	4.95596	4.83732	0.202715
1	0.520758	1.365295	0.469442	455.972321	4.47883	3.92999	4.83537	4.72794	0.189376
2	0.566076	1.081764	0.551494	189.937942	4.44582	4.00216	4.71632	4.65543	0.186211
3	0.510835	1.158584	0.549823	311.487579	4.11390	3.64717	4.37933	4.35910	0.184841
4	0.502329	1.462283	0.494155	467.448486	4.34930	3.97723	4.54172	4.55467	0.210685
5	0.493695	1.191643	0.556112	363.441620	4.46163	4.03945	4.72513	4.65312	0.189888
6	0.505619	1.332918	0.499637	426.528381	4.07487	3.67361	4.30145	4.28187	0.209245
7	0.490931	1.371683	0.510032	467.261414	4.37741	3.96808	4.56423	4.63131	0.172815
8	0.498223	1.237100	0.564277	347.639465	4.69108	4.25320	4.96795	4.88568	0.211027
9	0.488832	1.418375	0.499681	437.298492	4.55611	4.20175	4.74029	4.74841	0.201233
10	0.495040	1.548957	0.519538	409.969574	4.83673	4.38932	5.09884	5.05575	0.222823
11	0.523161	1.100955	0.534372	335.485199	4.28939	3.91534	4.45629	4.52317	0.210903
12	0.512244	1.246711	0.529643	432.946747	4.25624	3.80504	4.41868	4.58591	0.199496
13	0.521190	1.237732	0.563403	315.654266	4.30408	3.85299	4.60949	4.48941	0.187361
14	0.498360	1.387963	0.519481	499.128265	4.31720	3.85612	4.60601	4.53034	0.197523
15	0.489214	1.450789	0.519918	418.035523	4.53886	4.10191	4.70540	4.84459	0.192933
16	0.489214	1.450789	0.519918	415.371277	4.53886	4.10191	4.70540	4.84459	0.192933

icvs_gm	...	summary_p05_wm	summary_p95_bg	summary_p95_csf	summary_p95_gm	summary_p95_wm	summary_stdv_bg	summary_stdv_csf	summary_stdv_gm	summary_stdv_wm	wm2max
0.417046	...	15.840013	75	26.000000	48	78	18.436827	11.309881	13.321481	20.531166	0.613208
0.434557	...	16.100008	71	27.000000	47	73	16.587288	11.104458	13.000434	19.296249	0.598131
0.441928	...	15.000018	76	30.000000	52	80	17.812222	13.755800	14.117490	21.511131	0.589286
0.468063	...	13.940014	69	25.000000	45	72	17.240479	14.126932	12.717019	19.448999	0.491935
0.422141	...	17.640015	75	28.000000	50	78	17.797934	11.542548	14.101040	19.720718	0.644231
0.446176	...	16.920015	74	28.000000	49	77	17.494446	13.060905	13.802983	19.708481	0.573913
0.429012	...	18.870016	79	30.000000	54	83	19.240841	14.267403	15.216043	20.937077	0.612069
0.457950	...	15.580012	70	26.909986	46	72	16.467045	14.362858	12.941573	18.670832	0.563636
0.429619	...	16.800014	75	27.000000	51	78	18.162132	12.039571	14.183859	20.114748	0.515625
0.443540	...	20.350018	89	32.799980	59	93	20.985428	13.275007	16.513474	24.082800	0.620155
0.419827	...	18.900023	80	27.599983	52	83	19.377707	10.746583	14.520782	21.171226	0.657407
0.401749	...	17.020014	71	29.000000	50	74	16.431540	10.991287	14.142323	18.448402	0.529412
0.422811	...	18.480019	77	30.000000	53	80	17.990843	13.604383	14.768683	20.034895	0.591304
0.436799	...	15.640016	71	26.000000	48	74	17.199450	11.917496	13.529204	19.039455	0.588785
0.429657	...	19.440018	82	30.000000	55	85	19.547876	13.121756	15.527175	21.612547	0.574803
0.455695	...	18.300022	83	30.000000	55	87	19.949436	14.291942	15.255460	22.771635	0.614754
0.455695	...	18.300022	83	30.000000	55	87	19.949436	14.291942	15.255460	22.771635	0.614754

Example .csv data file from the dataset folder