Exasens Dataset Classification Report

This report provides a comprehensive overview of the Deep Learning project implemented in the provided Jupyter Notebook. The project focuses on developing a neural network model to classify patient diagnoses using the Exasens dataset.

1. Project Objective

The primary objective of this project is to build and evaluate a deep learning model capable of predicting a patient's 'Diagnosis' based on various physiological and psychological attributes present in the Exasens dataset. The goal is to achieve a robust classification model that can accurately distinguish between different diagnostic categories.

2. Data Source and Initial Exploration

The project utilizes the Exasens.csv dataset. While the notebook does not explicitly show initial data exploration steps (like data.info(), data.describe(), or data.head()), it immediately proceeds with data loading and cleaning.

3. Data Preprocessing Steps

Thorough data preprocessing is crucial for the performance of any machine learning model, especially neural networks. The notebook performs the following key preprocessing steps:

- **Data Loading**: The dataset is loaded into a pandas DataFrame:
  import pandas as pd

  data = pd.read_csv('/content/drive/MyDrive/datasets/Exasens.csv')

- **Column Dropping**: The 'ID' column, which is likely a unique identifier and not relevant for classification, is removed:

  data_cleaned = data.drop(columns=['ID'])

- **Handling Missing Values**: Rows containing missing values in critical columns ('Imagery_part_min', 'Imagery_part_avg', 'Real_part_min', 'Real_part_avg') are removed to ensure data integrity:

  ```
  data_cleaned = data_cleaned.dropna(subset=['Imagery_part_min',
  'Imagery_part_avg', 'Real_part_min', 'Real_part_avg'])
  ```

- **Feature and Target Separation**: The dataset is split into features (X) and the target variable (y):

  ```
  X = data_cleaned.drop(columns=['Diagnosis'])
  y = data_cleaned['Diagnosis']
  ```

- **Target Variable Encoding**: The categorical 'Diagnosis' target variable is converted into numerical labels using LabelEncoder. This is a necessary step before one-hot encoding:

  ```
  from sklearn.preprocessing import LabelEncoder
  label_encoder = LabelEncoder()
  y_encoded = label_encoder.fit_transform(y)
  ```

- **Feature Scaling**: All numerical features in X are scaled using StandardScaler. This process transforms the data such that it has a mean of 0 and a standard deviation of 1, which helps neural networks converge faster and perform better:

  ```
  from sklearn.preprocessing import StandardScaler
  scaler = StandardScaler()
  X_scaled = scaler.fit_transform(X)
  ```

- **One-Hot Encoding of Target**: The numerical target labels are converted into a one-hot encoded format. This is required for multi-class classification problems when using categorical_crossentropy as the loss function:

  ```
  from tensorflow.keras.utils import to_categorical
  y_one_hot = to_categorical(y_encoded)
  ```

- **Data Splitting**: The scaled features and one-hot encoded targets are split into training and testing sets. A common split ratio of 80% for training and 20% for testing is used:

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_one_hot, test_size=0.2, random_state=42)

4. Model Architecture

The deep learning model is a sequential neural network built using TensorFlow/Keras. Its architecture is defined as follows:

- **Input Layer**: A Dense layer with 32 neurons and a 'relu' (Rectified Linear Unit) activation function. The input_shape is set to the number of features in the dataset.
- **Hidden Layer**: Another Dense layer with 16 neurons, also using 'relu' activation. This layer helps the model learn more complex patterns from the data.
- **Output Layer**: A Dense layer with 4 neurons (corresponding to the 4 unique diagnosis classes) and a 'softmax' activation function. The 'softmax' activation ensures that the output probabilities for each class sum up to 1, making it suitable for multi-class classification.

The model summary would look like this:

```
Model: "sequential"
_________________________________________________________________
Layer (type)             Output Shape            Param #
=================================================================
dense (Dense)            (None, 32)              XXX
                                (where XXX is (num_features * 32) + 32)
dense_1 (Dense)          (None, 16)              YYY
                                (where YYY is (32 * 16) + 16)
dense_2 (Dense)          (None, 4)               ZZZ
                                (where ZZZ is (16 * 4) + 4)
=================================================================
Total params: XXX + YYY + ZZZ
Trainable params: XXX + YYY + ZZZ
Non-trainable params: 0
_________________________________________________________________
```

5. Model Compilation and Training

The model is compiled with specific parameters suitable for this classification task:

- **Optimizer**: The 'adam' optimizer is chosen, known for its efficiency and good performance in a wide range of deep learning tasks.
- **Loss Function**: categorical_crossentropy is used as the loss function. This is the standard choice for multi-class classification problems where the target labels are one-hot encoded.
- **Metrics**: 'accuracy' is monitored during training to assess the model's performance.

The training process is configured as follows:

- **Epochs**: The model is trained for 100 epochs, meaning the entire dataset is passed forward and backward through the neural network 100 times.
- **Batch Size**: A batch size of 8 is used, indicating that the model updates its weights after processing every 8 samples.
- **Validation Split**: A validation split of 0.2 (20%) is applied to the training data. This portion is used to monitor the model's performance on unseen data during training, helping to detect overfitting.

6. Model Evaluation

After training, the model's performance is evaluated on the dedicated test set, which comprises 20% of the original dataset and has not been seen by the model during training.

- **Test Accuracy**: The evaluation on the test set yields an accuracy of approximately **68.75%**.

This accuracy indicates the proportion of correctly classified instances in the test set. While 68.75% is a reasonable starting point, further improvements could be explored through hyperparameter tuning, more complex architectures, or additional feature engineering.

7. Conclusion and Future Work

The notebook successfully demonstrates the end-to-end process of building and evaluating a deep learning model for classification. The model achieves a test accuracy of approximately 68.75% on the Exasens dataset.

Potential areas for future work include:

- **Hyperparameter Tuning**: Experimenting with different numbers of layers, neurons per layer, activation functions, optimizers, learning rates, and batch sizes.
- **Regularization Techniques**: Implementing techniques like dropout or L1/L2 regularization to prevent overfitting.
- **More Complex Architectures**: Exploring convolutional neural networks (CNNs) if there's a spatial or temporal relationship in the data (though less likely for this tabular dataset), or recurrent neural networks (RNNs) if there's sequential data.
- **Ensemble Methods**: Combining multiple models to potentially improve overall performance.
- **Detailed Error Analysis**: Investigating misclassified samples to understand common patterns of errors and identify areas for improvement.
- **Class Imbalance Handling**: If the 'Diagnosis' classes are imbalanced, techniques like oversampling, undersampling, or using weighted loss functions could be applied.