**Moonquake Classification using Transformer Model**

**Project Overview**

This project aims to classify moonquake events based on seismic data collected during lunar exploration missions. A Transformer-based deep learning model is used to process and classify velocity time series data from moonquake seismic recordings.

The dataset consists of CSV files containing velocity measurements, which are paired with corresponding moonquake types from a catalog. The goal is to accurately classify different types of moonquakes using machine learning.

**Table of Contents**

1. [Installation](#installation)
2. [Dataset](#dataset)
3. [Usage](#usage)
4. [Model Architecture](#model-architecture)
5. [Data Preprocessing](#data-preprocessing)
6. [Training & Evaluation](#training--evaluation)
7. [Results](#results)
8. [Future Work](#future-work)

**Installation**

To run this project, clone the repository and install the required dependencies.

Clone the repository

git clone https://github.com/----/moonquake-classification.git

cd moonquake-classification

Install dependencies

Use pip to install the required Python packages.

pip install -r requirements.txt

**Required packages**:

* PyTorch
* Numpy
* Pandas
* Scikit-learn
* Matplotlib
* SciPy

**Dataset**

The dataset consists of seismic velocity measurements from moonquake events. Each data point is stored in a CSV file with the following columns:

* **Velocity (m/s)**: The seismic velocity at specific time intervals.
* **Time (sec)**: The corresponding time for each velocity measurement.

The dataset also includes a catalog file that maps the CSV files to their corresponding moonquake types. The available moonquake types are listed as class labels for classification.

## Usage

Step 1: Prepare the Dataset

Ensure that the seismic data CSV files and the catalog file are placed in the appropriate directory.

Step 2: Train the Model

Run the main.py script to train the Transformer-based model.

python main.py

The script will:

* Load the data
* Extract features from each CSV file
* Train the model
* Evaluate the model's performance on the test set

Step 3: View Results

After training, the model’s performance will be evaluated, and a classification report (precision, recall, F1-score) will be printed.

## Model Architecture

This project uses a **Transformer** architecture to classify moonquake events. The key components of the model include:

* **Embedding Layer**: Converts the input data into the model’s internal representation.
* **Positional Encoding**: Adds positional information to help the model learn the temporal relationships in the data.
* **Transformer Encoder**: The main component for learning dependencies in the input sequence using multi-head attention and feed-forward networks.
* **Fully Connected Layer**: Maps the model output to the target space (moonquake classes).

## Data Preprocessing

The dataset undergoes several preprocessing steps:

1. **Feature Extraction**: Features are extracted from the seismic data, including statistical, frequency-domain, and spectral features.
2. **Standardization**: Features are standardized using StandardScaler to normalize the data.
3. **Label Encoding**: The moonquake types (class labels) are encoded using LabelEncoder for compatibility with the classification model.

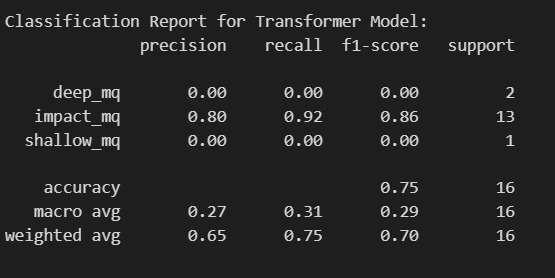
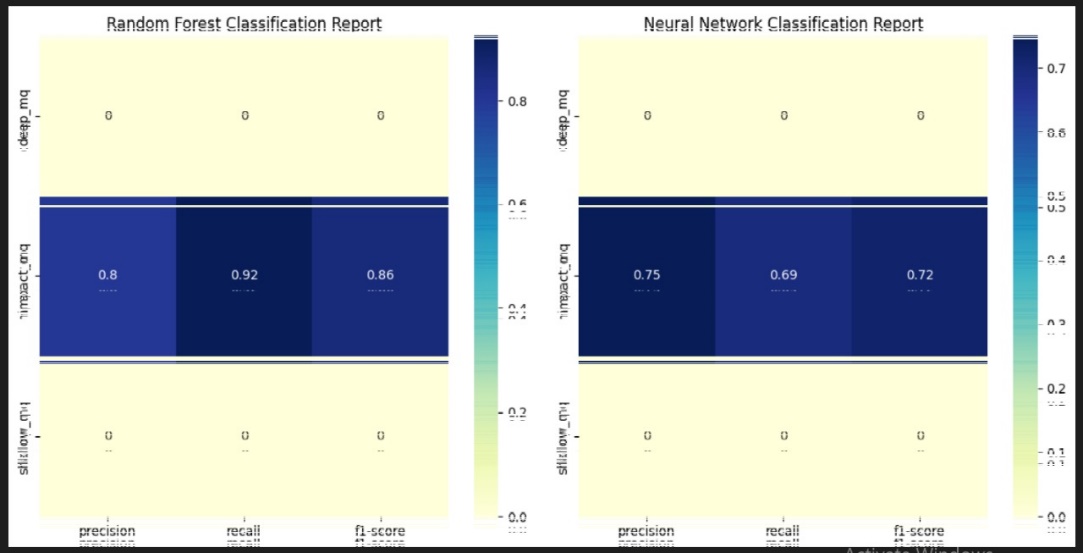
## Training & Evaluation

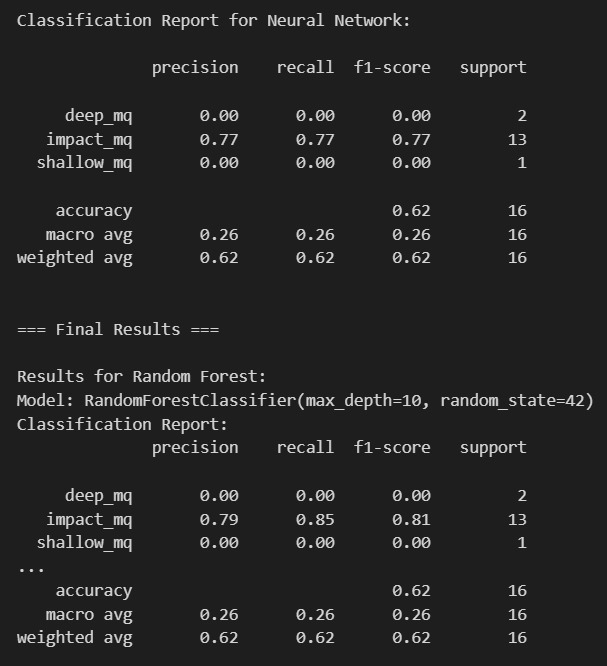
The model is trained using:

* **Optimizer**: Adam
* **Loss Function**: Cross-Entropy Loss

During training, the model undergoes backpropagation to minimize the loss, and its performance is evaluated using the classification report after training.

## Results

The model's performance is assessed based on precision, recall, F1-score, and accuracy for each moonquake class. The classification report gives insights into the model’s effectiveness at identifying different moonquake types.

 **Future Work**

**Hyperparameter Tuning**: Further tuning of the Transformer architecture (e.g., number of layers, model size) could improve performance.

**Data Augmentation**: Applying data augmentation techniques (e.g., noise addition, time shifts) could help improve the model’s generalization ability.

**Model Optimization**: Exploring model pruning or quantization to reduce the model size and improve inference speed.