National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Nagalakshmi Ramakrishna
Student ID: x23183829

School of Computing
National College of Ireland

Supervisor: Christian Horn

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Nagalakshmi Ramakrishna
Student ID:	x23183829
Programme:	Data Analytics
Year:	2024-2025
Module:	MSc Research Project
Supervisor:	Christian Horn
Submission Due Date:	12/12/2024
Project Title:	PhaseNet and EfficientNet-B0 for Phase Detection and Arrival Time Prediction
Word Count:	1420
Page Count:	

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Nagalakshmi
Date:	12/12/2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on the computer.	

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Nagalakshmi Ramakrishna
x23183829
MSc Data Analytics

Introduction

This configuration manual provides comprehensive details for implementing the research project "PhaseNet and EfficientNet-B0 for Phase Detection and Arrival Time Prediction." It includes system requirements, setup instructions, execution procedures, and operational guidelines. The project evaluates deep learning models to classify seismic phases and predict P-wave and S-wave arrival times using the INSTANCE dataset.

1. System Configurations

1.1 Hardware Requirements

- **Processor:** 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz 2.80 GHz
- **RAM:** 12.0 GB
- **System type:** 64-bit operating system, x64-based processor
- **Storage:** 500 GB
- **GPU:** NVIDIA GeForce MX450, 8 GB

1.2 Software Requirement

- **Primary Tool**: Jupyter Notebook
- **Programming Language**: Python 3.8+

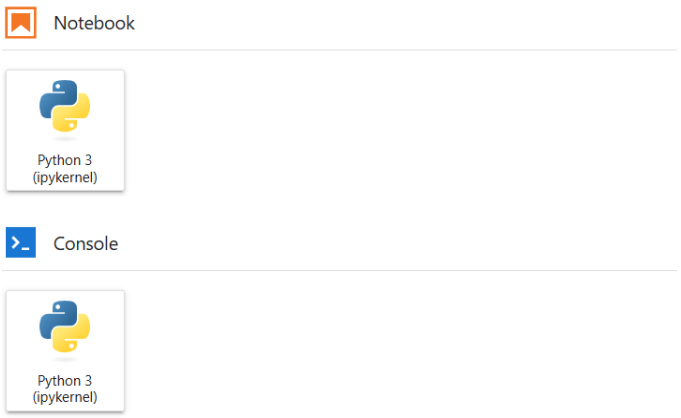


Figure1: JupyterLab's Launcher

Key Libraries:

- NumPy, Pandas, TensorFlow, Keras, Matplotlib, Seaborn.

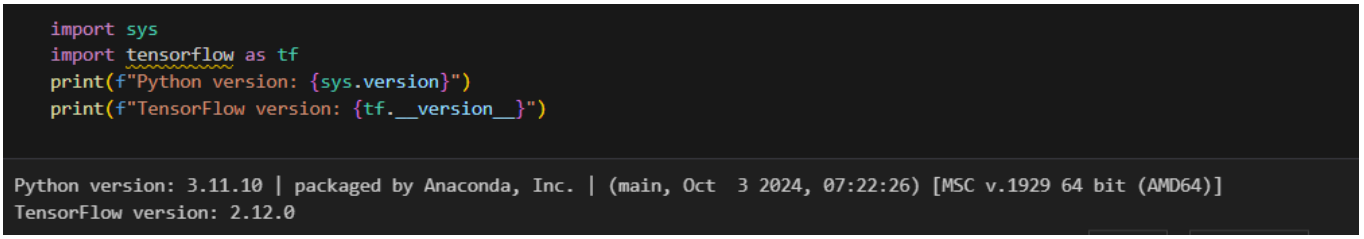


Figure2: Python and TensorFlow Version

- H5py for seismic data

Required Version to run both the file is shown in Fig. 2.

Tensor flow :- 2.12

Python :-> 3.11.10

2. Data Collection

The **INSTANCE dataset** was selected for this project. It is a well-known repository of seismic waveform data and metadata curated for machine learning tasks.

Key Features:

1. **Data Volume and Composition**:
   - Earthquake Data: 54,008 events with 1,159,249 three-channel waveforms.
   - Noise Data: 132,330 three-channel waveforms.

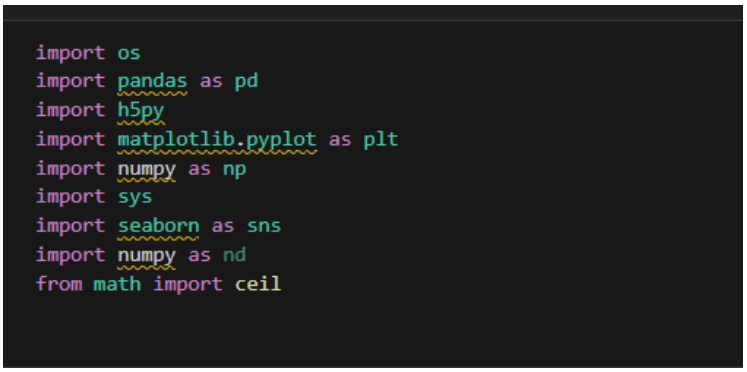
Data Type	Format	Size
Waveforms	HDF5	1,159,249
Metadata	CSV	54,008 events, 132,330 noise samples

2. **Metadata**:
   - Includes 115 attributes per event (e.g., station, trace, source).
3. **Geographic Diversity**:
   - Covers 19 networks and 620 seismic stations.
4. **Data Format**:
   - Waveform Data: HDF5 format.
   - Metadata: CSV format.

3.Execution Order

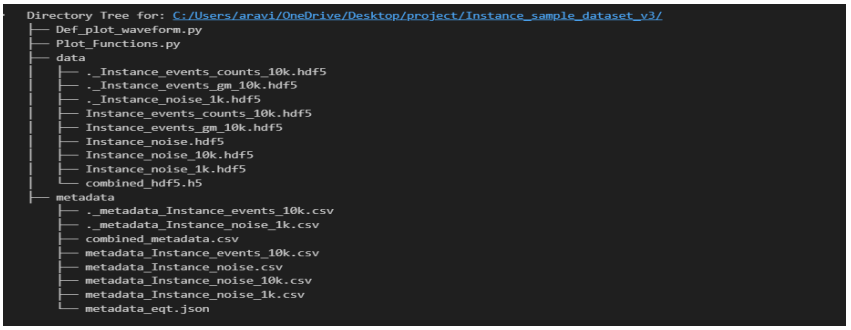
3.1 Exploratory Data Analysis (EDA):

1. Open and run the Thesis EDA.ipynb file.



```
import os
import pandas as pd
import h5py
import matplotlib.pyplot as plt
import numpy as np
import sys
import seaborn as sns
import numpy as nd
from math import ceil
```

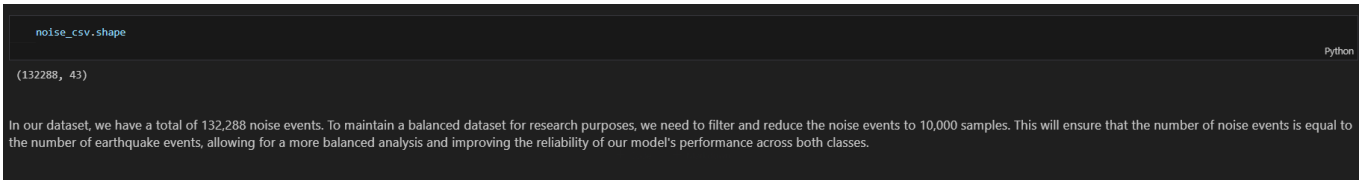
Figure3: Importing libraries



```
Directory Tree for: C:/Users/aravi/OneDrive/Desktop/project/Instance_sample_dataset_v3/
├── Def_plot_waveform.py
├── Plot_Functions.py
├── data
│   ├── ._Instance_events_counts_10k.hdf5
│   ├── ._Instance_events_gm_10k.hdf5
│   ├── ._Instance_noise_1k.hdf5
│   ├── Instance_events_counts_10k.hdf5
│   ├── Instance_events_gm_10k.hdf5
│   ├── Instance_noise.hdf5
│   ├── Instance_noise_10k.hdf5
│   ├── Instance_noise_1k.hdf5
│   └── combined_hdf5.h5
├── metadata
    ├── ._metadata_Instance_events_10k.csv
    ├── ._metadata_Instance_noise_1k.csv
    ├── combined_metadata.csv
    ├── metadata_Instance_events_10k.csv
    ├── metadata_Instance_noise.csv
    ├── metadata_Instance_noise_10k.csv
    ├── metadata_Instance_noise_1k.csv
    └── metadata_eqt.json
```

Figure4: Directory tree

2. Perform initial data preprocessing, including:
    - Data cleaning



```
noise_csv.shape
```

Python

```
(132288, 43)
```

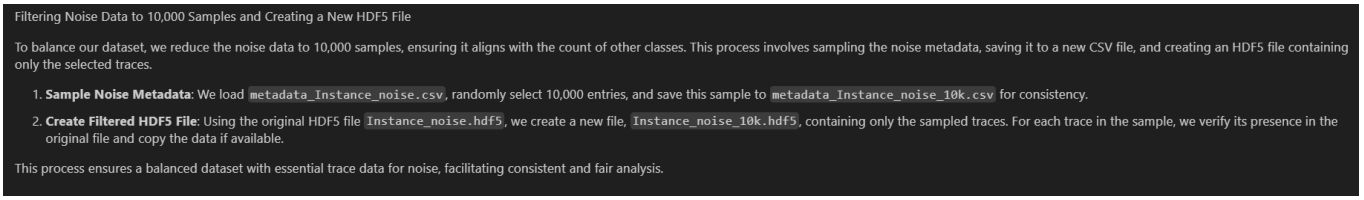
In our dataset, we have a total of 132,288 noise events. To maintain a balanced dataset for research purposes, we need to filter and reduce the noise events to 10,000 samples. This will ensure that the number of noise events is equal to the number of earthquake events, allowing for a more balanced analysis and improving the reliability of our model's performance across both classes.

Figure5: Noise data events

- Data transformation
- Visualization

Execute the following tasks:

- Load waveform and metadata files.
- Clean and preprocess the data (e.g., handle missing values, normalize waveforms).



Filtering Noise Data to 10,000 Samples and Creating a New HDF5 File

To balance our dataset, we reduce the noise data to 10,000 samples, ensuring it aligns with the count of other classes. This process involves sampling the noise metadata, saving it to a new CSV file, and creating an HDF5 file containing only the selected traces.

1. **Sample Noise Metadata**: We load `metadata_Instance_noise.csv`, randomly select 10,000 entries, and save this sample to `metadata_Instance_noise_10k.csv` for consistency.
2. **Create Filtered HDF5 File**: Using the original HDF5 file `Instance_noise.hdf5`, we create a new file, `Instance_noise_10k.hdf5`, containing only the sampled traces. For each trace in the sample, we verify its presence in the original file and copy the data if available.

This process ensures a balanced dataset with essential trace data for noise, facilitating consistent and fair analysis.

Figure6: filtering Noise data

- Generate visualizations like spectrograms and statistical distributions.

3. Generate preprocessed datasets and visualizations for the modeling phase.
4. Save the cleaned dataset for modeling.

3.2 Model Development and Training:

1. Open and run the Thesis ML.ipynb file.
2. Implement the PhaseNet and EfficientNet-B0 models.
3. Train and evaluate the models using the preprocessed data.

Step-by-Step Instructions

1. Launch Jupyter Notebook.

2. Open the Thesis EDA.ipynb file and execute all cells sequentially:

- Preprocess and visualize the data.

File	Type	Description	Shape
`instance.events.counts10k.hdf5`	Waveform Data	10,000 seismic events	10,000 × 3 channels
`instance.noise1k.hdf5`	Waveform Data	1,000 noise instances	1,000 × 3 channels
`metadata.instance.events10k.csv`	Metadata	Event metadata with 115 fields	10,000 × 115 columns
`metadata.instance.noise1k.csv`	Metadata	Noise metadata with 43 fields	1,000 × 43 columns

Figure7: Data Structure

- Verify that preprocessed data is saved correctly.

```
Folder structure for: C:/Users/aravi/OneDrive/Desktop/project/Instance_sample_dataset_v3/
|-- data
    |-- Instance_events_counts_10k.hdf5
    |-- Instance_events_gm_10k.hdf5
    |-- Instance_noise.hdf5
    |-- Instance_noise_10k.hdf5
    |-- Instance_noise_1k.hdf5
|-- Def_plot_waveform.py
|-- metadata
    |-- metadata_Instance_events_10k.csv
    |-- metadata_Instance_noise.csv
    |-- metadata_Instance_noise_10k.csv
    |-- metadata_Instance_noise_1k.csv
|-- Plot_Functions.py
```

Figure8: Folder structure

```
dataset = "C:/Users/aravi/OneDrive/Desktop/project/Instance_sample_dataset_v3/"
metadata_path = os.path.join(dataset, "metadata")
Image_path = os.path.join(dataset, "data")
output_dir = os.path.join(dataset, "output")

events_metaname=os.path.join(metadata_path,'metadata_Instance_events_10k.csv') # Either Counts and GroundMotion
noise_metaname=os.path.join(metadata_path,'metadata_Instance_noise_1k.csv')

# HDF5
events_hdfname=os.path.join(Image_path,'Instance_events_counts_10k.hdf5')
gm_hdfname=os.path.join(Image_path,'Instance_events_gm_10k.hdf5')
```

Figure9: Defining Path & Import

3. Loading Data event

```
# Define the paths for input data and output directories
event_csv_path = os.path.join(metadata_path,'metadata_Instance_events_10k.csv')
event_image_path = os.path.join(Image_path,'Instance_events_counts_10k.hdf5')
image_save_path =  os.path.join(dataset, "output")
```

Figure10: Output directories

4. Execute cells to train and evaluate models.
5. Monitor outputs, including training metrics, confusion matrices, and scatter plots.

4. Project Development

The project workflow is divided into three main stages:

1. **Data Preparation**: Data preprocessing, selection, and balancing.
2. **Modeling**: Model implementation using TensorFlow and Keras, including hyperparameter tuning.
3. **Evaluation**: Assessing model performance using metrics such as MAE, precision, recall, F1-score, and accuracy.

4.1 Data Preparation

1. **Importing Datasets**:
    - Waveform data imported in HDF5 format using the h5py library.
    - Metadata processed using NumPy for efficient numerical computations.

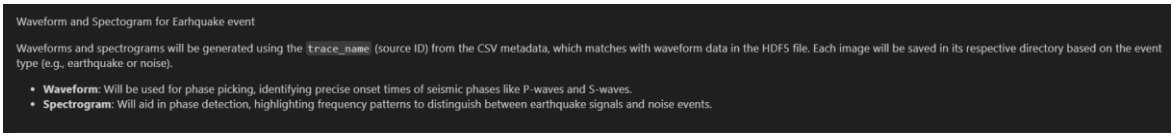


Figure11: waveform and Spectrogram event

2. **Data Cleaning:**
    - Removal of missing or corrupted entries.
    - Verification of consistency between waveform files and metadata.

3. **Data Transformation:**
    - Normalize waveform data for consistency.
    - Generate spectrograms using Short-Time Fourier Transform (STFT).
4. **Balancing Dataset:**
    - Additional noise samples introduced to address class imbalance.

5. **Phase Detection:**
   Phase detection is a critical task in seismic data analysis that involves classifying input signals into distinct categories such as Noise, P-wave, and S-wave. The PhaseNet model is utilized for this purpose, leveraging convolutional layers to extract features from spectrogram images.

```
import os
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
%matplotlib inline
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import (accuracy_score, precision_score, recall_score,
                             mean_squared_error, f1_score, confusion_matrix, ConfusionMatrixDisplay)
from tensorflow.python.keras.callbacks import TensorBoard, EarlyStopping, ModelCheckpoint
from sklearn import metrics
import PIL
```

Figure12: Confusion matrix

6. EfficientNet for Phase Detection

Checkpoint Path

- The `checkpoint path` is defined to save the model weights during training. This ensures that the best-performing model is saved for later use.

```
# Checkpoint path
checkpoint_path = os.path.join(os.getcwd(), 'checkpoint', 'EfficientNetClassification', 'checkpoint.ckpt')

# Early Stopping
early_stopping_callback = EarlyStopping(
    monitor="val_loss",
    min_delta=0.001,
    patience=5,
    verbose=1,
    mode="min",
    restore_best_weights=True
)

# Model Checkpoint
model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_path,
    save_weights_only=True,
    monitor='val_loss',
    verbose=1,
    save_best_only=True
)

# Combine Callbacks
callbacks = [early_stopping_callback, model_checkpoint_callback]
```

Figure13: EfficientNet for Phase Detection

7. Splitting Data:

Dataset divided into 60% training, 20% validation, and 20% testing subsets.

Spliting the data for classification

```
splitfolders.ratio(
    os.path.join(image_save_path, "Spectrogram"),
    output=os.path.join(image_save_path,"phase_detection"),
    seed=1337,
    ratio=(0.6, 0.2, 0.2)  # 60% for training, 20% for validation, 20% for testing
)
```

```
Copying files: 20000 files [02:24, 138.42 files/s]
```

Figure14: Splitting data

8. Confusion matrix for EfficientNet

This section describes the evaluation process for the classification model, including metrics calculation, confusion matrix visualization, and accuracy history plotting.The key performance metrics such as

accuracy, precision, recall, and F1-score are computed to assess the classification performance.

```
# Create confusion matrix
print('Building confusion matrix')
cm = confusion_matrix(test.classes, predicted_classes)  # compare target values to predicted values and show confusion matrix
print(cm)
accuracy = accuracy_score(test.classes, predicted_classes)
precision = precision_score(test.classes, predicted_classes)
recall = recall_score(test.classes, predicted_classes)
f1score = f1_score(test.classes, predicted_classes)
print(f'The accuracy of the model is {accuracy}, the precision is {precision}, the recall is {recall}, and the F1 score is {f1score}')

# Plot confusion matrix
plt.style.use('default')
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Noise', 'Earthquake'])
disp.plot(cmap='Blues', values_format='')
plt.title('Classification Effecient net)')
plt.tight_layout()
plt.show()

# Plot accuracy history
plt.style.use('ggplot')
fig, ax = plt.subplots(figsize=(7, 7))
ax.plot(history.history['accuracy'])
ax.plot(history.history['val_accuracy'])
ax.set_title('Model Accuracy')
ax.set_ylabel('Accuracy')
ax.set_xlabel('Epoch')
ax.legend(['train', 'test'])
plt.show()
```

Figure15: EfficientNet Confusion Matrix

9. Phase Regression:

This section highlights the libraries and modules imported for data handling, model building, evaluation, and visualization in the project.

```
import os
import random
import numpy as np
import pandas as pd
import seaborn as sns
from PIL import Image
import tensorflow as tf
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, mean_absolute_error, mean_squared_error, r2_score

from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.losses import MeanAbsoluteError, MeanAbsolutePercentageError
from tensorflow.keras import layers, models, Model
from tensorflow.keras.models import Sequential
from tensorflow.keras.applications import EfficientNetB0
from tensorflow.keras.utils import plot_model
from tensorflow.keras.optimizers import Adam
from tensorflow.python.keras.callbacks import TensorBoard, EarlyStopping, ModelCheckpoint, History
```

Figure16: Phase regression

10. Data Preparation for Regression

This section describes the preparation of data for the regression task, where the model predicts the P-wave and S-wave arrival times. The code snippet provides a step to verify the output directory structure for organizing the data required for regression tasks.

The directory contains three main categories of data:

- **phase detection**: Contains labeled data for phase classification (Noise, P-wave, S-wave).
- **Spectrogram**: Contains frequency-based representations of seismic signals.
- **Waveform**: Raw waveform data for regression tasks.

```
path = image_save_path


# List files in the output directory
dataset_path = os.listdir(path)
print(dataset_path)
```

```
['phase_detection', 'Spectrogram', 'Waveform']
```

Figure17: Output directory

11. EfficientNet for Phase Regression

EfficientNet is utilized to predict P-wave and S-wave arrival times, a regression task where Mean Absolute Percentage Error (MAPE) is monitored to optimize performance. The following implementation employs callbacks for Early Stopping and Model Checkpointing to enhance training efficiency and save the best model configuration.

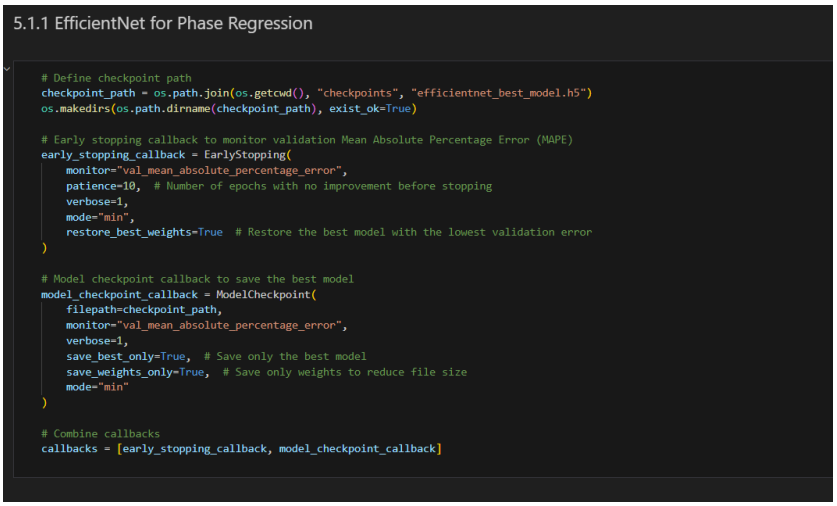


Figure18: EffiecientNet for Phase regression

12. PhaseNet for Phase Regression

The PhaseNet model is designed for regression tasks, specifically predicting P-wave or S-wave arrival times from spectrogram data. This section describes the architecture and implementation of a convolutional neural network (CNN)-based PhaseNet model for regression.

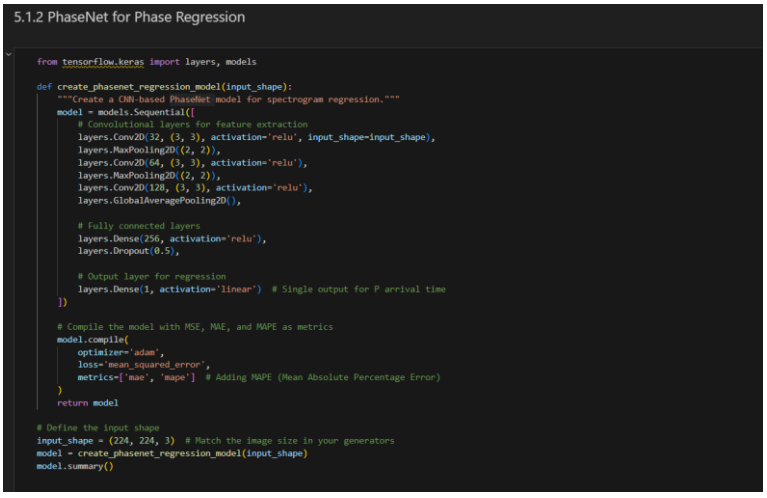


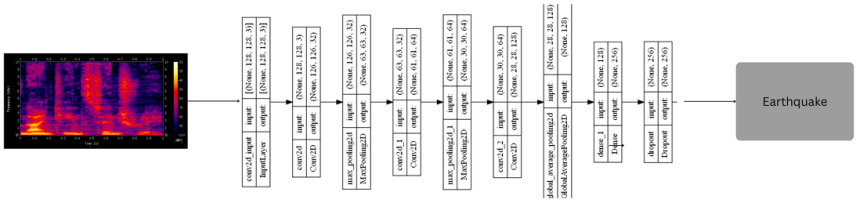
Figure19:PhaseNet for Phase regression

5. Model Development

Models Used

1. **PhaseNet**:

- **Purpose**: Seismic phase detection (noise, P-wave, S-wave classification).
- **Architecture**:
  - Convolutional layers for feature extraction.
  - Pooling layers for dimensionality reduction.
  - Fully connected layers for classification.

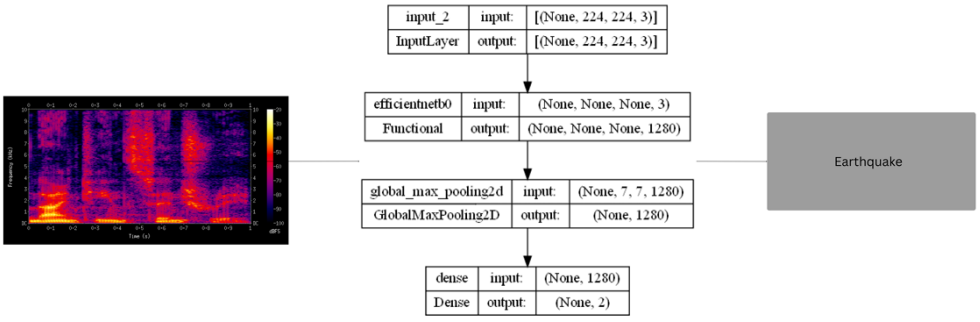


Phase Net Classification Architecture

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

2. **EfficientNet-B0**:

- **Purpose**: Predicting P and S wave arrival times (regression).
- **Architecture**:
  - Pre-trained EfficientNet-B0 with fine-tuning.
  - Fully connected layers for regression tasks.
- **Loss Function**: Mean Absolute Error (MAE)
- **Optimizer**: Adam



EfficientNet-B0-based architecture classification

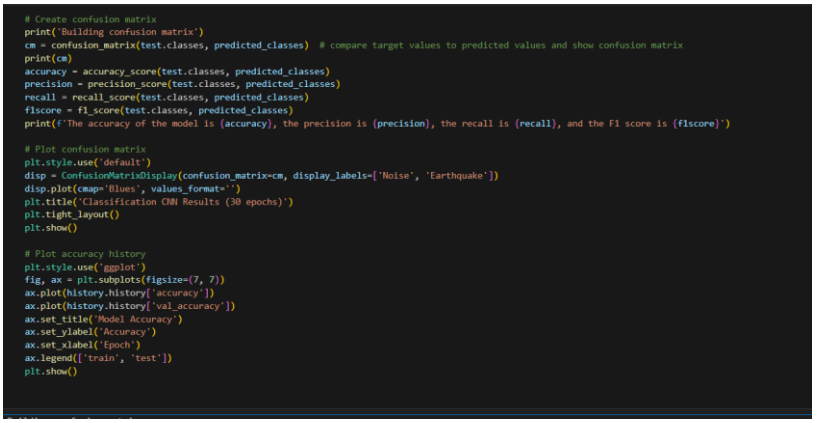
6. Model Training

1. **Data Input and Preprocessing**:
   - Spectrograms used as input for **PhaseNet**.
   - Raw waveforms used as input for **EfficientNet-B0**.

2. **Training Process**:
   - Batch size: 32
   - Epochs: 50 (with early stopping to prevent overfitting).
3. **Evaluation Metrics**:
   - **PhaseNet**: Precision, Recall, F1-score, Accuracy.
   - **EfficientNet-B0**: MAE, Root Mean Squared Error (RMSE).

7. Evaluation

PhaseNet:
PhaseNet achieved exceptional performance, with training accuracy.



```
# Create confusion matrix
print('Building confusion matrix')
cm = confusion_matrix(test.classes, predicted_classes)  # compare target values to predicted values and show confusion matrix
print(cm)
accuracy = accuracy_score(test.classes, predicted_classes)
precision = precision_score(test.classes, predicted_classes)
recall = recall_score(test.classes, predicted_classes)
f1score = f1_score(test.classes, predicted_classes)
print(f'The accuracy of the model is {accuracy}, the precision is {precision}, the recall is {recall}, and the F1 score is {f1score}')

# Plot confusion matrix
plt.style.use('default')
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Noise', 'Earthquake'])
disp.plot(cmap='Blues', values_format='')
plt.title('Classification CNN Results (30 epochs)')
plt.tight_layout()
plt.show()

# Plot accuracy history
plt.style.use('ggplot')
fig, ax = plt.subplots(figsize=(7, 7))
ax.plot(history.history['accuracy'])
ax.plot(history.history['val_accuracy'])
ax.set_title('Model Accuracy')
ax.set_ylabel('Accuracy')
ax.set_xlabel('Epoch')
ax.legend(['train', 'test'])
plt.show()
```

EfficientNet-B0:
EfficientNet-B0, a highly efficient convolutional neural network, was employed for image classification tasks. EfficientNet is a family of models ranging from B0 to B7, designed to balance performance and computational efficiency.

```
# Create confusion matrix
print('Building confusion matrix')
cm = confusion_matrix(test.classes, predicted_classes)  # compare target values to predicted values and show confusion matrix
print(cm)
accuracy = accuracy_score(test.classes, predicted_classes)
precision = precision_score(test.classes, predicted_classes)
recall = recall_score(test.classes, predicted_classes)
f1score = f1_score(test.classes, predicted_classes)
print(f'The accuracy of the model is {accuracy}, the precision is {precision}, the recall is {recall}, and the F1 score is {f1score}')

# Plot confusion matrix
plt.style.use('default')
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Noise', 'Earthquake'])
disp.plot(cmap='Blues', values_format='')
plt.title('Classification Effecient net)')
plt.tight_layout()
plt.show()

# Plot accuracy history
plt.style.use('ggplot')
fig, ax = plt.subplots(figsize=(7, 7))
ax.plot(history.history['accuracy'])
ax.plot(history.history['val_accuracy'])
ax.set_title('Model Accuracy')
ax.set_ylabel('Accuracy')
ax.set_xlabel('Epoch')
ax.legend(['train', 'test'])
plt.show()
```

8. Evaluation Metrics

PhaseNet Evaluation:

- Accuracy: Overall classification performance.
- F1-Score: Harmonic mean of precision and recall.

EfficientNet-B0 Evaluation:

- MAE: Mean Absolute Error for arrival time prediction.
- RMSE: Root Mean Squared Error

8. Conclusion

This manual documents the process of configuring and executing deep learning-based seismic analysis. **PhaseNet** excels in classifying seismic signals, while **EfficientNet-B0** provides precise arrival time predictions. The implementation leverages cutting-edge machine learning techniques, ensuring robust and accurate results.

9. References

1. Zhu, W. & Beroza, G. C. (2019). PhaseNet: A deep-neural-network-based seismic phase picker.
2. Tan, M. & Le, Q. (2019). EfficientNet: Rethinking model scaling for convolutional neural networks.
3. TensorFlow Documentation: https://www.tensorflow.org
4. INSTANCE Dataset Documentation: https://instance.readthedocs.io