National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Arunprasath Ramakrishnan Arularasan
Student ID: 20207417

School of Computing
National College of Ireland

Supervisor: Dr. Vladimir Milosavljevic

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Arunprasath Ramakrishnan Arularasan
Student ID:	20207417
Programme:	Data Analytics **Year:** 2021-2022
Module:	MSc Research Project
Lecturer:	Dr. Vladimir Milosavljevic
Submission Due Date:	August 15, 2022
Project Title:	Seismic Phase Detection & Picking using EfficientNet
Word Count:	1526 **Page Count:** 12

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: Arunprasath Ramakrishnan Arularasan

Date: August 15, 2022

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 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

Arunprasath Ramakrishnan Arularasan
Student ID: 20207417

1 Introduction

The configuration manual discusses the model-building steps of the research "Seismic Phase Detection & Picking using EfficientNet". The research was done using Jupyter Notebook on a local machine and Python. The system configuration used for the research is shown in Table 1. The latest version of TensorFlow was used to access the EfficientNet model. The Keras Image generator was used for reading the images. The "Matplotlib" library was used for the visualisations and the "sklearn" library was used for the evaluation metrics.

Table 1 System Configuration

Operating System	Windows 10
RAM	16 GB
Processor	AMD Ryzen 7 5800H
Graphic Card	NVIDIA RTX 3060

2 Data pre-processing

The Italian Seismic Dataset (INSTANCE) was used for this research. The datasets consist of raw waveform signals that need to be converted into waveform and spectrogram plots for the research. The signals are stored as hdf5 files and the metadata for the signals is stored as CSV. The h5py library is used for reading the HDF5 files and the plots of waveform were visualised using the matplotlib library. The parallel, delayed function in the joblib library is used for multiprocessing.

```
import pandas as pd
import h5py
import numpy as np
import matplotlib.pyplot as plt
import time
import timeit
from joblib import Parallel,delayed
```

Figure 1 Required Modules

```
eq1_csv_path = r'E:\Research_Proj\Dataset\metadata\metadata_Instance_events_10k.csv'
eq1_sig_path = r'E:\Research_Proj\Dataset\data\Instance_events_gm_10k.hdf5'
eq1_sig_path = r'E:\Research_Proj\Dataset\data\Instance_events_counts_10k.hdf5'
```

```
earthquakes_1 = pd.read_csv(eq1_csv_path)
earthquakes_1
```

Figure 2: Reading the Metadata

```
chunk_name = earthquakes_1
data_start = 0
data_end = 10000
data_interval = 2000
eqpath = eq1_sig_path
img_save_path = r"E:\\Research_Proj\\Dataset\\output\\"
```

```
eqlist = chunk_name['trace_name'].to_list()
#eqlist = np.random.choice(eqlist,20,replace=False) # for random sample of signals

starts = list(np.linspace(data_start,data_end-data_interval,int((data_end-data_start)/data_interval)))
ends = list(np.linspace(data_interval,data_end,int((data_end-data_start)/data_interval)))
set = str(chunk_name)
```

Figure 3: Creating Data intervals

```
import os

os.makedirs(img_save_path+'Waveform/Earthquake/',exist_ok=True)
os.makedirs(img_save_path+'Waveform/Nosie/',exist_ok=True)
os.makedirs(img_save_path+'Spectrogram/Earthquake/',exist_ok=True)
os.makedirs(img_save_path+'Spectrogram/Noise/',exist_ok=True)
```

Figure 4: Creating Folders and Sub-folders for Plots

The paths to the HDF5 files were given and the metadata file was read using the “pandas” library as shown in Fig2. For this research, 10000 earthquake signals were chosen and the data interval was chosen as 2000. The name of all the traces from the metadata was stored in a separate list. Separate subdirectories are created as shown in Fig 4 for Waveform and Spectrogram plots with the subfolders as Earthquake and Noise. Fig 5 shows the code block that converts the waveform signals into their respective waveform and spectrogram plots. The traces are read in batches of 2000 using for loop. The traces list containing the 10,000 earthquake waveforms is then read from the HDF5 files. The waveforms are then transposed using NumPy. The waveforms are then plotted for the 120s with the labels and the axis removed. The waveform is then saved to its path. The spectrogram is plotted using the specgram function in matplotlib. The colormap chosen for the spectrogram was plasma as EfficientNet requires images in 3 channels with the sampling frequency set at 100Hz. The plots are then stored in the respective directories. Since the processing of 10,000 files consumes a lot of time, the parallel and delayed functions were used for multiprocessing the batches of 2000 images. This allows for faster conversion of signals to waveform and spectrogram plots. Both the waveform and the spectrogram plots of a particular trace were stored with their names from the metadata file.

```
for n in range(0,len(starts)):
    def make_images(i):
    # retrieving selected waveforms from the hdf5 file:
        try:
            dtfl = h5py.File(path, 'r')
            dataset = dtfl.get('data/'+str(traces[i]))
            data1 = np.array(dataset)
            data=data1.T

            fig, ax = plt.subplots(figsize=(6,2))
            ax.plot(np.linspace(0,120,12000),data[:,2],color='k',linewidth=1)
            ax.set_xlim([0,120])
            ax.axis('off')
            plt.gca().set_axis_off()
            plt.subplots_adjust(top = 1, bottom = 0, right = 1, left = 0,
                        hspace = 0, wspace = 0)
            plt.margins(0,0)
            #plt.show()
            plt.savefig(img_save_path+"Waveform\\Earthquake\\"+traces[i]+'.png',bbox_inches='tight',dpi=100)
            plt.close()
            fig, ax = plt.subplots(figsize=(3,2))
            ax.specgram(data[:,2],Fs=100,NFFT=256,cmap='plasma',vmin=-10,vmax=25);
            ax.set_xlim([0,120])
            ax.axis('off')
            plt.gca().set_axis_off()
            plt.subplots_adjust(top = 1, bottom = 0, right = 1, left = 0,
                        hspace = 0, wspace = 0)
            plt.margins(0,0)
            print("here",1)
            plt.savefig(img_save_path+"Spectrogram\\Earthquake\\"+traces[i]+'.png',bbox_inches='tight',
                        transparent = False,pad_inches=0,dpi=50)
            plt.close()


        except:
            print('String index out of range')

    traces = eqlist[int(starts[n]):int(ends[n])]
    path = eqpath
    print(path)
    # create images for selected data
    start = time.time()
    print(start)
    Parallel(n_jobs=-2)(delayed(make_images)(i) for i in range(0,len(traces)))
    end = time.time()
    print(f'Took {end-start} s')
```

Figure 5: Converting Signals to Spectrogram and Waveform

For the processing of the noise waves the same code snipped was used with the file's paths changed for the HDF5 file and the CSV file as shown in Fig 6. The noise waves were processed in a similar way to the earthquake signals. The waveform of the noise and the spectrogram plots were stored in their respective directories.

```
noise_csv_path=r'E:\Research_Proj\Dataset\metadata\metadata_Instance_noise.csv'
noise_sig_path = r'E:\Research_Proj\Dataset\data\Instance_noise.hdf5'
```

```
noise_csv = pd.read_csv(noise_csv_path)
noise_csv
```

Figure 6: Processing Noise Waves

The spectrogram plots for the phase classification model were trained using Keras Image Generator which requires a separate folder structure for the train, test, and validation. The "splitfolders" library was used for converting it into that folder structure in the ratio of 60% for the training data and 20% each for the test and validation data.

```
import splitfolders
splitfolders.ratio('E:\\Research_Proj\\Dataset\\output\\Spectrogram\\', output="E:\\Research_Proj\\Dataset\\output",
                   seed=1337, ratio=(.8, 0.1,0.1))
```

Figure 7: Creating the Folder structure

3 Phase Classification model

For the phase classification model, the libraries shown in Fig 8 were used. The Keras Image Data Generator was used for processing the images from the train, test, and validation folders. The *"flow_from_directory"* function takes the directory path and reads the images from both classes. Since the EfficientNet-B0 model was used for the research, the images were resized into the shape (224,224). The classes were specified to ensure that Noise was encoded as 0 and earthquake was encoded as 1. The images were read in batches of 64 and since this is a classification model the class mode was set as categorical.

```
import os
import numpy as np
import pandas as pd
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
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from tensorflow.python.keras.callbacks import TensorBoard, EarlyStopping, ModelCheckpoint
from sklearn import metrics
```

Figure 8: Libraries for the Phase Classification model

Fig 10 shows the callback functions used while training the model. The callback functions used were the early stopping callback and the model checkpoint callback. The early stopping callback was set to stop after 10 epochs if the improvements were worse than 1%. The model checkpoint callback was used to save the model weight after every epoch to continue the training from the last saved state. Fig 11 shows the EfficientNet architecture used for the Phase Classification model. The EfficientNet B0 was loaded from TensorFlow Keras applications. The layers of the EfficientNet were set to be not trainable. The Global max pooling layer and dense layer were used to get the output from the EfficientNet architecture to classify the signals. The "SoftMax" function was used to dense the output into two probabilities for the Earthquake class and the Noise Class. The model was trained with the train generator and the val generator was used for validation and the model was trained for 50 epochs. Fig 12 shows the code used for retrieving the "Accuracy vs Loss" curve from the history variable. Fig 13 shows the evaluation of the model on the test dataset. Fig 14 shows the calculation of evaluation metrics like the accuracy, F1 score, recall, and precision from the "sklearn" metrics. The confusion matrix is plotted using the "ConfusionMatrixDisplay" and the plots are saved. The accuracy history for the training and test dataset for every epoch is plotted using the history variable. The false positive rate, true positive rate, and AUC are calculated using the sklearn metrics function. These variables are used to plot the Receiver Operating Characteristics (ROC) in Fig 15.

Fig 16 shows the CNN architecture that was used for comparing the performance of the EfficientNet. The model was then compiled with the same parameters and trained with the train generator. The model was evaluated similarly to the EfficientNet. For finding the

interpretability of the model, the code in Fig 17 was used to display random image samples with the predicted labels. The same code was reused with the path of the train, test, and validation directories for training with the STEAD dataset.

```
IMG_SIZE = (224, 224)
CLASSES = ['Noise' , 'Earthquake']
path_to_dir="C:/Users/Arunprasath/Downloads/Final_code/"
train_dataset_path = 'E:/Research_Proj/Dataset/output5/train/'
test_dataset_path = 'E:/Research_Proj/Dataset/output5/test/'
val_dataset_path = 'E:/Research_Proj/Dataset/output5/validation/'
```

```
train_gen = ImageDataGenerator()
train = train_gen.flow_from_directory(train_dataset_path,
                                      batch_size=64,
                                      target_size=(224, 224),
                                      class_mode='categorical',
                                      classes=CLASSES,
                                      shuffle=True,
                                      seed=42)
# Validation dataset
val_gen = ImageDataGenerator()
val = val_gen.flow_from_directory(val_dataset_path,
                                  batch_size=64,
                                  target_size=(224, 224),
                                  class_mode='categorical',
                                  classes=CLASSES,
                                  shuffle=True,
                                  seed=42)
# Testing dataset
test_gen = ImageDataGenerator()
test = test_gen.flow_from_directory(test_dataset_path,
                                  batch_size=64,
                                  target_size=(224, 224),
                                  class_mode='categorical',
                                  classes=CLASSES,
                                  shuffle=False,
                                  seed=42)
```

```
Found 12000 images belonging to 2 classes.
Found 4000 images belonging to 2 classes.
Found 4000 images belonging to 2 classes.
```

Figure 9: Creating the Image Data Generator

```
checkpoint_path = path_to_dir+'checkpoint/EfficientNetClassification/checkpoint.ckpt'
checkpoint_path
```

```
'C:/Users/Arunprasath/Downloads/Final_code/checkpoint/EfficientNetClassification/checkpoint.ckpt'
```

```
early_stopping_callback = EarlyStopping(
    monitor="loss",
    min_delta=1,  # model should improve by at least 1%
    patience=10,  # amount of epochs  with improvements worse than 1% until the model stops
    verbose=1,
    mode="min",
    restore_best_weights=True,  # restore the best model with the lowest validation error
)

model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_path,
    save_weights_only=True,
    monitor='accuracy',
    save_freq='epoch',
    verbose=1,
    save_best_only=False)

callbacks=[early_stopping_callback,model_checkpoint_callback]
```

Figure 10: Callback functions

```
base_model = tf.keras.applications.EfficientNetB0(include_top=False)
base_model.trainable = False

inputs = tf.keras.layers.Input(shape=IMG_SIZE+(3,))
x = base_model(inputs, training=False)

x = tf.keras.layers.GlobalMaxPooling2D()(x)
outputs = tf.keras.layers.Dense(units=2, activation='softmax')(x)
model = tf.keras.Model(inputs, outputs)
model.compile(optimizer='adam', metrics=['accuracy'], loss=tf.keras.losses.CategoricalCrossentropy())

history = model.fit(train,validation_data=val, callbacks=callbacks,epochs=50)
```

Figure 11: EfficientNet Architecture

```
pd.DataFrame(history.history).plot()
plt.title('Accuracy Vs Loss')
```

```
Text(0.5, 1.0, 'Accuracy Vs Loss')
```

Figure 12: History plot

```
print('Evaluating model on test dataset')
test_loss, test_acc = model.evaluate(test, verbose=1) # get model evaluation metrics
print("\nTest data, accuracy: {:5.2f}%".format(100*test_acc))

print('Finding predicted classes and probabilities to build confusion matrix')
predicted_classes = np.argmax(model.predict(test),axis=-1) # predict the class of each image
predicted_probs = model.predict(test) # predict the probability of each image belonging to a class
```

```
Evaluating model on test dataset
63/63 [==============================] - 135s 2s/step - loss: 0.2966 - accuracy: 0.8813

Test data, accuracy: 88.13%
Finding predicted classes and probabilities to build confusion matrix
63/63 [==============================] - 120s 2s/step
63/63 [==============================] - 104s 2s/step
```

Figure 13: Evaluating Test dataset

```
# 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 {f1sc


# plot confusion matrix
plt.style.use('default')
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Noise','Earthquake'])
disp.plot(cmap='Blues', values_format='')
plt.title(f'Classification CNN Results (30 epochs)')
plt.tight_layout()
plt.savefig(path_to_dir+'\\plots\\Eff_INS_confusion_matrix.png')
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.savefig(path_to_dir+'\\plots\\Eff_INS_model_accuracy.png')
plt.show()
```

Figure 14: Evaluation metrics

```
fpr, tpr, _ = metrics.roc_curve(test.classes,predicted_classes)
auc = metrics.roc_auc_score(test.classes,predicted_classes)

#create ROC curve
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.title("ROC Curve")
plt.show()
```

Figure 15: ROC Curve

```
model1 = tf.keras.Sequential([
  tf.keras.layers.Conv2D(16, 3, padding='same', activation='relu', input_shape=(224, 224, 3)),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Conv2D(32, 3, padding='same', activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Conv2D(64, 3, padding='same', activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(2)
])
model1.compile(optimizer='adam',metrics=['accuracy'],loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True))
```

```
print(model1.summary())
```

```
history1 = model1.fit(train,validation_data=val, callbacks=callbacks,epochs=50)
```

Figure 16: CNN architecture

```
import random

fnames = test.filenames
y_pred = np.argmax(model2.predict(test),axis=-1)## fnames is all the filenames/samples used in testing

errors = np.where(y_pred != test.classes)[0] ## misclassifications done on the test data where y_pred is the predicted values

wrong_pred=[]
wrong_labels=[]
for i in errors:
    wrong_labels.append(y_pred[i])
    wrong_pred.append(fnames[i])

right_labels=["Noise "if x==1 else "Earthquake" for x in wrong_labels]
wrong_lab=["Noise "if x==0 else "Earthquake" for x in wrong_labels]

plt.figure(1, figsize=(15, 9))
plt.axis('off')

n = 0
for i in range(16):
    n += 1
    random_no = random.randint(0,len(wrong_labels))
    imgs = plt.imread(test_dataset_path+wrong_pred[random_no])
    plt.title(f"Actual: {right_labels[random_no]}, Predicted: {wrong_lab[random_no]}")
    plt.subplot(4, 4, n)
    plt.axis('off')
    plt.imshow(imgs)

plt.show()
```

Figure 17: Displaying wrong predictions

4 Phase Regression model

The libraries used for the Phase regression model are shown in Fig 18. The labels for the arrival times for the trace are retrieved by reading the metadata file as shown in Fig 19. Since the regression model can't use the folder structure like the classification model, the path of the files is stored in a separate dataframe as shown in Fig 20 along with their corresponding labels from the metadata file.

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

Figure 18: Libraries for Phase Regression

```
dir_path=r'C:/Users/Arunprasath/Downloads/Final_code/'
path='E:/Research_Proj/Dataset/output3/'
metadata_path='E:\Research_Proj\Dataset\metadata\metadata_Instance_events_10k.csv'
dataset_path = os.listdir(path)
dataset_path
```

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

```
df_metadata_eq=pd.read_csv(metadata_path)
df_metadata_eq
```

Figure 19: Reading the Metadata

```
class_labels = []
i=0
for data in dataset_path:
    all_classes = os.listdir(path+data+'//')
    print(all_classes)
    for item in all_classes:
        all_files = os.listdir(path+data+'//'+item)
        print(data,item,len(all_files))
        for room in all_files:
            if data=="Waveform":
                p_arrival=0
                s_arrival=0
                if item=="Earthquake":
                    p_arrival=df_metadata_eq.loc[df_metadata_eq['trace_name'] == room[0:-4], 'trace_P_arrival_sample'].iloc[0]
                    s_arrival=df_metadata_eq.loc[df_metadata_eq['trace_name'] == room[0:-4], 'trace_S_arrival_sample'].iloc[0]
                    class_labels.append((item,p_arrival,s_arrival,str(path+data+'/'+item+"/"+room)))
            i+=1
            print(f"{i} processed")
df = pd.DataFrame(data=class_labels, columns=['Labels','P_arrival_time','S_arrival_time' ,'Path'])
print(df.head())
print(df.tail())
```

Figure 20: Creating dataframe with image path and arrival times

The dataframe is then split into train, test, and validation and passed to the Image data generators for creating their respective generators. The Image generator takes the path of the image as “x_col” and the “y_col” as the arrival time of the P/S waves. The class mode was set as raw as it is a regression model. Fig 22 shows the calculation of the baseline MAPE and MSE for the model by predicting the mean of the arrival times.

```
from sklearn.model_selection import train_test_split

train, val = train_test_split(df, test_size=0.2, random_state=1)  # split the data with a validation size o 20%
train, test = train_test_split(train, test_size=0.125, random_state=1)
```

```
train_generator = ImageDataGenerator(
        rescale=1.0 / 255
    )
validation_generator = ImageDataGenerator(
    rescale=1.0 / 255
)
test_generator = ImageDataGenerator(rescale=1.0 / 255)


train_generator = train_generator.flow_from_dataframe(
    dataframe=train,
    x_col="Path",
    y_col="P_arrival_time",
    class_mode="raw",
    target_size=(224, 224),
    batch_size=128,
)

validation_generator = validation_generator.flow_from_dataframe(
    dataframe=val,
    x_col="Path",
    y_col="P_arrival_time",
    class_mode="raw",
    target_size=(224, 224),
    batch_size=128,
)
test_generator = test_generator.flow_from_dataframe(
    dataframe=test,
    x_col="Path",
    y_col="P_arrival_time",
    class_mode="raw",
    target_size=(224, 224),
    batch_size=128,
)
```

```
Found 7000 validated image filenames.
Found 2000 validated image filenames.
Found 1000 validated image filenames.
```

Figure 21: Creating Image Data Generator

```
y_hat = train["P_arrival_time"].mean()
val["y_hat"] = y_hat
mae = MeanAbsoluteError()
mae = mae(val["P_arrival_time"], val["y_hat"]).numpy()
mape = MeanAbsolutePercentageError()
mape = mape(val["P_arrival_time"], val["y_hat"]).numpy()

print(mae)
print("mean baseline MAPE: ", mape)
```

```
358.65880342857145
mean baseline MAPE:  15.789500012133509
```

Figure 22: Baseline MAE & MAPE

Fig 23 shows the callback functions used for the model. The early stopping callback and the model checkpoint callback were aused for the model with the validation MAPE being the monitor variable. Fig 25 shows the EfficientNet regression architecture used and Fig 25 shows the and training of the model with “rectified adam” as the optimizer.

```
early_stopping_callback = EarlyStopping(
    monitor="val_mean_absolute_percentage_error",
    min_delta=1,  # model should improve by at least 1%
    patience=10,  # amount of epochs  with improvements worse than 1% until the model stops
    verbose=2,
    mode="min",
    restore_best_weights=True,  # restore the best model with the lowest validation error
)

model_checkpoint_callback = ModelCheckpoint(
    dir_path+ "EffNetRegInstance-P" + ".h5",
    monitor="val_mean_absolute_percentage_error",
    verbose=0,
    save_best_only=True,  # save the best model
    mode="min",
    save_freq="epoch",  # save every epoch
)

callbacks=[early_stopping_callback,model_checkpoint_callback]
```

Figure 23: Callback functions

```
inputs = layers.Input(
    shape=(224, 224, 3)
)

model = EfficientNetB0(include_top=False, input_tensor=inputs)
# Freeze the pretrained weights
model.trainable = False

# Rebuild top
x = layers.GlobalAveragePooling2D()(model.output)
x = layers.BatchNormalization()(x)
top_dropout_rate = 0.4
x = layers.Dropout(top_dropout_rate)(x)
outputs = layers.Dense(1)(x)

# Compile
model = keras.Model(inputs, outputs, name="EfficientNet")
```

Figure 24: EfficientNet Regression architecture

```
import tensorflow_addons as tfa

model.summary()
radam = tfa.optimizers.RectifiedAdam(learning_rate=0.01)
ranger = tfa.optimizers.Lookahead(radam, sync_period=6, slow_step_size=0.5)
optimizer = ranger

model.compile(
    optimizer=optimizer, loss="mean_absolute_error", metrics=[MeanAbsoluteError(), MeanAbsolutePercentageError()]
)
history = model.fit(
    train_generator,
    epochs=50,
    validation_data=validation_generator,callbacks=callbacks,
    workers=2
)
```

Figure 25: Compiling the model

The model was evaluated using the train generator as shown in Fig 26. Fig 27 shows how the true picks were classified for the model. If a predicted value lied within a range of 150 (1.5s), it was considered a true pick. The CNN Regression architecture is shown in Fig 28. The model was compiled similarly to the EfficientNet architecture and the model's performance was evaluated using the test dataset. The MAPE plots for every epoch were visualised for both the architectures using the code snippet in Fig 29. The baseline MAPE was used as a reference line. The same code was used for S waves with the variable name changed and the STEAD dataset.

```
print('Evaluating model on test dataset')
test_loss = model.evaluate(test_generator, verbose=1)
print(f'Test data loss: {test_loss}')

print('Getting predictions')
predicted = model.predict(test_generator)
```

Figure 26: Evaluating the Test data

```
pred_val=predicted.flatten()
TP=0
for act, pre in zip(test_generator.labels,pred_val):
    if abs(act-pre)<=150:
        TP+=1
print(TP)
```

```
486
```

Figure 27: Finding True Picks

```
model1 = models.Sequential()
model1.add(layers.Conv2D(32, (3, 3), activation="relu", input_shape=(224, 224, 3)))
model1.add(layers.MaxPooling2D((2, 2)))
model1.add(layers.Conv2D(64, (3, 3), activation="relu"))
model1.add(layers.MaxPooling2D((2, 2)))
model1.add(layers.Conv2D(64, (3, 3), activation="relu"))

model1.add(layers.Flatten())
model1.add(layers.Dense(64, activation="relu"))
model1.add(layers.Dense(1))
```

```
model1.summary()
```

```
model1.compile(optimizer=optimizer, loss="mean_absolute_error", metrics=[MeanAbsoluteError(), MeanAbsolutePercentageError()])
```

```
history1 = model1.fit(
    train_generator,
    epochs=50,
    validation_data=validation_generator,
    callbacks=callbacks,
    workers=2, # adjust this according to the number of CPU cores of your machine
)
```

Figure 28: CNN Regression Architecture

```
dict1 = {
    "MAPE":  history.history["mean_absolute_error"],
    "type": "training",
    "model": "eff_net",
}
dict2 = {
    "MAPE": history.history["mean_absolute_error"],
    "type": "validation",
    "model": "eff_net",
}
dict3 = {
    "MAPE": history1.history["mean_absolute_error"],
    "type": "training",
    "model": "CNN_net",
}
dict4 = {
    "MAPE": history1.history["mean_absolute_error"],
    "type": "validation",
    "model": "CNN_net",
}
s1 = pd.DataFrame(dict1)
s2 = pd.DataFrame(dict2)
s3 = pd.DataFrame(dict3)
s4 = pd.DataFrame(dict4)
df = pd.concat([ s1,s2,s3, s4], axis=0).reset_index()
grid = sns.relplot(data=df, x=df["index"], y="MAPE", hue="model", col="type", kind="line", legend=False)
grid.set(ylim=(310, 500))  # set the y-axis limit
for ax in grid.axes.flat:
    ax.axhline(
        y=mae, color="lightcoral", linestyle="dashed"
    )  # add a mean baseline horizontal bar to each plot
    ax.set(xlabel="Epoch")
labels = ["eff_net","CNN" ,"mean_baseline"]  # custom labels for the plot

plt.legend(labels=labels)
plt.savefig("training_validation_CNN_INS_P.png")
plt.show()
```

Figure 29: Comparison of MAPE over epochs