Configuration Manual

MSc Research Project
Data Analytics

Aanchal Singh
Student ID: X19221771

School of Computing
National College of Ireland

Supervisor: Majid Latifi

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Aanchal Singh
Student ID:	X19221771
Programme:	Data Analytics
Year:	2021
Module:	MSc Research Project
Supervisor:	Majid Latifi
Submission Due Date:	16/08/2021
Project Title:	Configuration Manual
Word Count:	915
Page Count:	9

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:	Aanchal Singh
Date:	15th August 2021

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

Aanchal Singh
X19221771

1 Introduction

The purpose of this document is go through the software, hardware and system configurations which are required during the entire process flow of the research project. Below is the process which is followed to apply pre processing techniques followed by machine and deep learning algorithms.

2 System Configuration

2.1 Hardware

Processor: Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz
GPU: NVIDIA GeForce GTX 1050Ti (4GB)
RAM: 8GB
Storage: 1TB HDD
Operating system: Windows 10, 64-bit.

2.2 Software

Python using Google Collaboratory and Jupyter Notebook: Data cleaning, data pre-processing, analysis and visualization.

Microsoft Excel: Save the data from extracted FITS files

Device name	LAPTOP-1B5C81HL
Processor	Intel(R) Core(TM) i5-1035G1 CPU @ 1.00GHz 1.19 GHz
Installed RAM	8.00 GB (7.70 GB usable)
Device ID	1FF55988-2643-426C-8A61-F8E46B2F087B
Product ID	00327-35885-16033-AAOEM
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

Figure 1: System Configuration

3 Implementation

The implementation of the thesis follows a step by step approach as mentioned in Figure 2. The research is divided in three parts. Extracting the data then cleansing it for any null values followed by the next step of applying pre processing techniques of Fast Fourier Transform (FFT) and Recurrence Plot (RP). Once these are applied, machine learning algorithm, Support Vector Machine(SVM) is applied to FFT pre processed data and Convolutional Neural Network along with pre trained VGG 16 model is applied to data pre processed by RP.

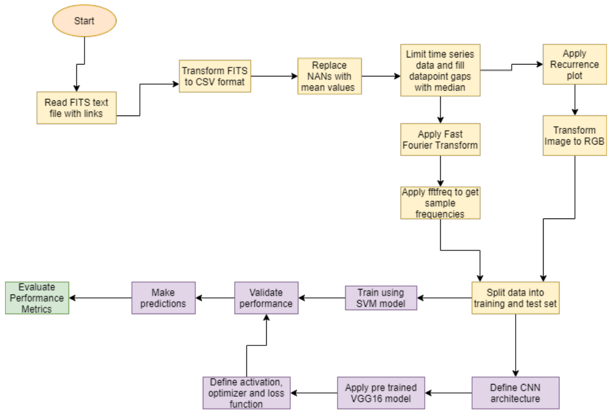

Figure 2: End to end flow of finding Exoplanets

4 Data Pre-processing

4.1 Data Collection from Mikulski Archive for Space Telescope (MAST) Portal

The data chosen for this research is from MAST portal [1] where there are different parameters for searching the data. The data is filtered on taking Exoplanet Host star data and False positive data. Figure 3 shows the different filters which can be applied to the data.

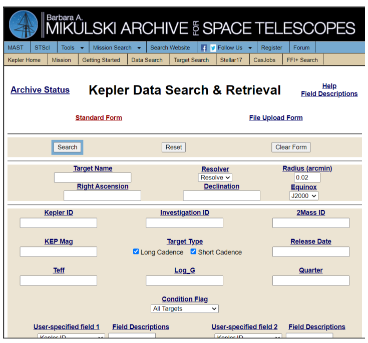

Figure 3: MAST Portal for NASA's Kepler Mission's Data

[1] Kepler Mission Data: `http://archive.stsci.edu/kepler/data_search/search.php`

4.2 FITS file

The data is in the format of URLs for the light curves. These can be accessed by astropy.io package in Python which extracts the time series data. Figure 4 is a sample of the FITS data.(Silva et al.; 2013)

```
https://archive.stsci.edu/missions/kepler/lightcurves/0008/000892772/kplr000892772-2010078095331_llc.fits
https://archive.stsci.edu/missions/kepler/lightcurves/0008/000892772/kplr000892772-2010174085026_llc.fits
https://archive.stsci.edu/missions/kepler/lightcurves/0008/000892772/kplr000892772-2010265121752_llc.fits
https://archive.stsci.edu/missions/kepler/lightcurves/0008/000892772/kplr000892772-2010355172524_llc.fits
https://archive.stsci.edu/missions/kepler/lightcurves/0008/000892772/kplr000892772-2011073133259_llc.fits
https://archive.stsci.edu/missions/kepler/lightcurves/0008/000892772/kplr000892772-2011177032512_llc.fits
https://archive.stsci.edu/missions/kepler/lightcurves/0008/000892772/kplr000892772-2011271113734_llc.fits
https://archive.stsci.edu/missions/kepler/lightcurves/0008/000892772/kplr000892772-2012004120508_llc.fits
https://archive.stsci.edu/missions/kepler/lightcurves/0008/000892772/kplr000892772-2012088054726_llc.fits
https://archive.stsci.edu/missions/kepler/lightcurves/0008/000892772/kplr000892772-2012179063303_llc.fits
https://archive.stsci.edu/missions/kepler/lightcurves/0008/000892772/kplr000892772-2012277125453_llc.fits
https://archive.stsci.edu/missions/kepler/lightcurves/0008/000892772/kplr000892772-2013011073258_llc.fits
https://archive.stsci.edu/missions/kepler/lightcurves/0008/000892772/kplr000892772-2013098041711_llc.fits
https://archive.stsci.edu/missions/kepler/lightcurves/0008/000892772/kplr000892772-2013131215648_llc.fits
https://archive.stsci.edu/missions/kepler/lightcurves/0010/001025986/kplr001025986-2009131105131_llc.fits
https://archive.stsci.edu/missions/kepler/lightcurves/0010/001025986/kplr001025986-2009166043257_llc.fits
https://archive.stsci.edu/missions/kepler/lightcurves/0010/001025986/kplr001025986-2009259160929_llc.fits
https://archive.stsci.edu/missions/kepler/lightcurves/0010/001025986/kplr001025986-2009322144938_slc.fits
https://archive.stsci.edu/missions/kepler/lightcurves/0010/001025986/kplr001025986-2009350155506_llc.fits
https://archive.stsci.edu/missions/kepler/lightcurves/0010/001025986/kplr001025986-2010078095331_llc.fits
https://archive.stsci.edu/missions/kepler/lightcurves/0010/001025986/kplr001025986-2010174085026_llc.fits
https://archive.stsci.edu/missions/kepler/lightcurves/0010/001025986/kplr001025986-2010265121752_llc.fits
https://archive.stsci.edu/missions/kepler/lightcurves/0010/001025986/kplr001025986-2010355172524_llc.fits
https://archive.stsci.edu/missions/kepler/lightcurves/0010/001025986/kplr001025986-2011073133259_llc.fits
```

Figure 4: FITS file having URLs

To do processing of data using Python libraries like astropy, Scipy, Keras, Tensorflow, CV2, etc need to be installed. These need to be imported and then used. The interface used for Python in this research id Google Collaboratory. The notebook can also be run on Anaconda's Jupyter Notebook. In below images Figure 5 and Figure 6 all libraries required for the project are imported.

```
# Importing and installing necessary modules
!pip install pyts
!pip install keras

import matplotlib.pyplot as plt
import sys
import numpy as np
np.set_printoptions(threshold=sys.maxsize)
from scipy import pi
from scipy.fft import fft, fftfreq, ifft
import pandas as pd
from scipy.spatial.distance import pdist, squareform
from matplotlib import pyplot
import matplotlib.cm as cm
from matplotlib.colors import Normalize
from pylab import rcParams

from astropy.visualization import astropy_mpl_style
plt.style.use(astropy_mpl_style)
from astropy.utils.data import get_pkg_data_filename
from astropy.table import Table
from pyts.image import RecurrencePlot
import astropy.io.fits as fits

from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.model_selection import cross_val_score
import sklearn as sk
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
from xgboost import XGBClassifier
from sklearn.metrics import precision_score, accuracy_score
```

Figure 5: Python Libraries Imported

To access these files these are read using the fits.open() and the brightness of the star which is signified by PDCSAP_FLUX feature. The NAN values are replaced by the mean flux values as below in Figure 7.

```python
from sklearn.model_selection import cross_val_score
from sklearn.metrics import precision_score, recall_score,roc_curve,auc, f1_score, roc_auc_score,confusion_matrix, accuracy_score, classification_report
from scipy import fftpack

from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input
from tensorflow.keras.utils import to_categorical
import tensorflow as tf
from tensorflow.keras import layers, models
import keras
from tensorflow import keras
from keras.models import Model
from keras.layers import LSTM, Activation, Dense, Dropout, Input, Embedding
from keras.optimizers import RMSprop
from keras.preprocessing.text import Tokenizer
from keras.preprocessing import sequence
from keras.models import Sequential
from keras.models import Sequential
from keras.layers import Flatten, Dense, Dropout, Conv2D, MaxPooling2D
from keras.optimizers import RMSprop
from keras.callbacks import EarlyStopping, ReduceLROnPlateau
from sklearn.svm import SVC

from csv import writer
import csv
from numpy import genfromtxt
from csv import reader

import cv2
from contextlib import redirect_stdout
import seaborn as sns
```

Figure 6: Python Libraries Imported

```python
# # The data is read from the links and stored in variable called Data_Exo and Data_No_Exo
# # This data is in FITS(Flexible Image Transport System) format and can be accessed using astropy lib in python.
# # This library is imported in the first cell.

# # Saving data from the links to CSV file in time series
# # Total data contains number of data sets used for testing and training.
# # Training data is 80% of Total data set whereas Testing data is reamining 20%
# # Enter split ratio for training data

# POS = pd.DataFrame()

# for i in range(DATA_TO_COPY2CSV):
#      Data_Exo = fits.open(Exo_content_list[i])
#      flux = Data_Exo[1].data['  '].byteswap().newbyteorder()
#      mean = np.nanmean(flux)
#      inds = np.where(np.isnan(flux))
#      flux[inds] = mean
#      POS = POS.append(pd.DataFrame(flux.reshape(1,flux.shape[0]), dtype='float32'))
#      print("DONE ",i+1,"/",str(DATA_TO_COPY2CSV))

# POS = POS.fillna(0)
# POS.to_csv('POS_TimeSeries.csv', mode='w', header=False, index=False)
# print("DONE POS")
# # POS
# # NEG
```

Figure 7: Cleansing data and coverting it to CSV format

4.3 Plot for Time Series Data

Using the time and flux values the initial data is plotted to check how patterns are being formed. For experiment 1, data with initial cleansing was applied to SVM. The results were satisfactory with a classification accuracy of 52 percent. To improve this accuracy this trsearch focuses on using pre processing techniques using FFT and RP.

```
def model(classifier,dtrain_x,dtrain_y,dtest_x,dtest_y):
    #fit the model
    #classifier.probability=True
    classifier.fit(dtrain_x,dtrain_y)
    predictions = classifier.decision_function(dtest_x)
    predict_acc = classifier.predict(dtest_x)
    print(predictions[:10])
    #Accuracy
    print ("\naccuracy_score :",accuracy_score(dtest_y,predict_acc))

    #Classification report
    print ("\nclassification report :\n",(classification_report(dtest_y,predict_acc)))

    #Confusion matrix
    plt.figure(figsize=(13,10))
    plt.subplot(221)
    sns.heatmap(confusion_matrix(dtest_y,predict_acc),annot=True,cmap="viridis",fmt = "d",linecolor="k",linewidths=3)
    plt.title("CONFUSION MATRIX",fontsize=20)
    return predictions

SVM_model=SVC()
predictions1 = model(SVM_model,
                     X_train_time.reshape(X_train_time.shape[0],X_train_time.shape[1]),
                     Y_train_time.reshape(Y_train_time.shape[0]),
                     X_test_time.reshape(X_test_time.shape[0],X_test_time.shape[1]),
                     Y_test_time.reshape(Y_test_time.shape[0]))
```

Figure 8: Applying SVM without any pre processing

4.4 Application of Fast Fourier Transform

The next experiment uses initially cleansed data. FFT is applied to this data by using Scipy library in Python which provides fft package. Using fftpack and fftfreq, FFT is applied in samples. A sample of 100 data points is created and iterated for all the data.

```
# # FFT analysis
X_train_fft = np.abs(fft(X_train)).astype(np.float32)
X_test_fft = np.abs(fft(X_test)).astype(np.float32)


def Plot(X, i):
  f_s = 100
  freqs = fftpack.fftfreq(len(X_train_fft[i])) * f_s

  plt.xlabel('Frequency in Hertz [Hz]')
  plt.ylabel('Frequency Domain (Spectrum) Magnitude')
  # ax.set_xlim(-f_s / 2, f_s / 2)
  plt.ylim(0, 10000)
  plt.plot(freqs, X_train_fft[i], marker=".", markersize=2)
  plt.figure()

Plot(X_train_fft[10],10)
Plot(X_train_fft[0],0)
Plot(X_train_fft[30],30)
Plot(X_train_fft[-11],-11)
Plot(X_train_fft[-21],-21)
Plot(X_train_fft[-31],-31)
```

Figure 9: FFT applied to Samples of Data

4.5 Application of Recurrence Plots

Recurrence Plots plot time series data and help in visualizing the data. To analyse the data it was first converted to 1 dimensional figure which gave different patterns of light curves. Using matplotlib and CV2 libraries in Python recurrence plots are plotted as shown in Figure 10.

```
# Reference Plots from time curves
# 1, 2, 3 recurrence plots are for postive data and 4, 5, 6 are for negative data

def rec_plot(s, eps=0.50, steps=10):
    d = pdist(s[:,None])
    d = np.floor(d/eps)
    d[d>steps] = steps
    Z = squareform(d)
    return Z


List = [10,0,30]
for _, i in enumerate(List):
  image = rec_plot(X_train[i,:2000])
  plt.figure()
  plt.imshow(image,cmap="gray")


List = [X_train.shape[0]-11, X_train.shape[0]-21, X_train.shape[0]-31]
for _, i in enumerate(List):
  image = rec_plot(X_train[i,:2000])
  plt.figure()
  plt.imshow(image,cmap="gray")
```

Figure 10: Recurrence Plot applied for 1D image

These images cannot be fed to any model or algorithm as they are in 1 Dimensional format. To make these images ready for model application it is needed that they are converted to 2 dimensional or 3 dimensional format. This is done by using stacking technique present in CV2 library, The interpolation of images are changed and made suitable for modelling.

```
# Stacking recurrence plot images (Conversion from single channel to multi channel images)

# 3 Channel image for Positive data
image_pos = rec_plot(X_train[0,:3000])
print("Shape of image before resizing",image_pos.shape)
image_pos = cv2.resize(image_pos, (HEIGHT,WIDTH), interpolation=cv2.INTER_CUBIC)
print("Shape of image before resizing",image_pos.shape)
image_pos = np.stack((image_pos, image_pos, image_pos),axis=0).reshape(HEIGHT,WIDTH,3)
plt.figure()
plt.imshow(image_pos)

# 3 Channel image for Negative data
image_neg = rec_plot(X_train[-11,:3000])
print("Shape of image before resizing",image_neg.shape)
image_neg = cv2.resize(image_neg, (HEIGHT,WIDTH), interpolation=cv2.INTER_CUBIC)
print("Shape of image before resizing",image_neg.shape)
image_neg = np.stack((image_neg, image_neg, image_neg),axis=0).reshape(HEIGHT,WIDTH,3)
plt.figure()
plt.imshow(image_neg)
```

Figure 11: Converting RP to 2 dimensional images

After this SVM is applied to FFT processed data and CNN is applied to RP processed data. CNN uses a pre trained model VGG16 with customized layers for classification of the data. All three steps were performed using Keras and Tensorflow libraries. Following

```
base_model = VGG16(weights="imagenet", include_top=False, input_shape=X_TRAIN[0].shape)

base_model.trainable = False ## Not trainable weights

## Preprocessing input
train_ds = preprocess_input(X_TRAIN)
test_ds = preprocess_input(X_TEST)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 0s 0us/step
```

```
## Loading VGG16 model
base_model = VGG16(weights="imagenet", include_top=False, input_shape=(HEIGHT, WIDTH, 3))
base_model.trainable = False ## Not trainable weights

## Preprocessing input
# train_ds = preprocess_input(X_TRAIN.reshape(X_TRAIN.shape[0],HEIGHT,WIDTH,3))
# test_ds = preprocess_input(X_TEST.reshape(X_TEST.shape[0],HEIGHT,WIDTH,3))
```

Figure 12: Create base model and Pre process input for VGG16

figures are how model is loaded for VGG16 followed by application of dense layers and model compliation by using Adam optimizer and crossentropy function.

The snippet in Figure 12 executes the base model with pre prcessing for the model. It categorises the data into positive and negative classes.

```
flatten_layer = layers.Flatten()
dense_layer_1 = layers.Dense(256, activation='relu')
dense_layer_2 = layers.Dense(180, activation='relu')
dense_layer_3 = layers.Dense(128, activation='relu')
dense_layer_4 = layers.Dense(64, activation='relu')
prediction_layer = layers.Dense(2, activation='softmax')


model = models.Sequential([
    base_model,
    flatten_layer,
    #dense_layer_1,
    #dense_layer_2,
    dense_layer_3,
    dense_layer_4,
    prediction_layer
])
```

Figure 13: Creating and applying dense layers to the model

The next code snippet in Figure 13 executes customized layers that are built for our model four dense layers and reLU activation function. A prediction layer is also used which uses the Softmax activation function. The total number of parameters are found and segregated into trainable and non trainable parameters.

```
model.compile(
    optimizer='Adam',
    loss='categorical_crossentropy',
    metrics=['accuracy'],
)


es = EarlyStopping(monitor='val_accuracy', mode='max', patience=5,  restore_best_weights=True)

history2 = model.fit(train_ds, train_labels, epochs=50, validation_split=0.2, batch_size=32, callbacks=[es])
```

Figure 14: Model compilation

Once the parameters are found, Figure 14 optimizes the model by using optimizer, cross entropy function and loss function.

The FFT processed data is applied with SVM again. The snippet below is applying SVM on the data which is processed by FFT previously.

```
def model(classifier,dtrain_x,dtrain_y,dtest_x,dtest_y):
    #fit the model
    classifier.fit(dtrain_x,dtrain_y)
    predictions = classifier.decision_function(dtest_x)
    predict_acc = classifier.predict(dtest_x)

    #Accuracy
    print ("\naccuracy_score :",accuracy_score(dtest_y,predict_acc))

    #Classification report
    print ("\nclassification report :\n",(classification_report(dtest_y,predict_acc)))

    #Confusion matrix
    plt.figure(figsize=(13,10))
    plt.subplot(221)
    sns.heatmap(confusion_matrix(dtest_y,predict_acc),annot=True,cmap="viridis",fmt = "d",linecolor="k",linewidths=3)
    plt.title("CONFUSION MATRIX",fontsize=20)
    return predictions

SVM_model=SVC()
predictions3 = model(SVM_model,
                              X_train_fft.reshape(X_train_fft.shape[0],X_train_fft.shape[1]),
                              Y_train_fft.reshape(Y_train_fft.shape[0]),
                              X_test_fft.reshape(X_test_fft.shape[0],X_test_fft.shape[1]),
                              Y_test_fft.reshape(Y_test_fft.shape[0]))
```

Figure 15: Model compilation

5 Evaluation

Once the models are built, it was compared with normal model and pre processed models. Evaluation of models was done on the basis accuracy, precision and ROC curve. Confusion matrix is also used as a parameter. Below snippet is used to create ROC curve for all the models and was used as a metrics to measure the performance.

```
# Plotting ROC curves for all models

prob=[]
for i in range(prediction3.shape[0]):
  if Y_TEST[i]==0:
    prob.append(prediction3[i][0])
  else:
    prob.append(prediction3[i][1])
prob_array = np.array(prob)
prob_array = prob_array.reshape(prob_array.shape[0],1)

false_positive_rate2, true_positive_rate2, threshold1 = roc_curve(Y_TEST,prob_array)

plt.subplots(1, figsize=(10,10))
plt.title('Receiver Operating Characteristic Curve for Recurrence Plot model')

plt.plot(false_positive_rate2, true_positive_rate2)

plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

Figure 16: Code for ROC curve for RP

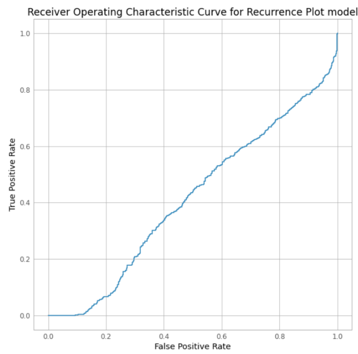

Figure 17: ROC Curve for RP

ROC curve is also created for before application of FFT and after application of FFT. This curve was created together in one graph to compare in a better way.

Using above codes ROC curves were created and in a similar way Confusion matrix and other metrics were calculated for each model.

```python
# Plotting ROC curves for all models

false_positive_rate3, true_positive_rate3, threshold3 = roc_curve(Y_test_fft, predictions3)
false_positive_rate1, true_positive_rate1, threshold1 = roc_curve(Y_test_time, predictions1)

plt.subplots(1, figsize=(10,10))
plt.title('Receiver Operating Characteristic Curve for FFT model')
plt.plot(false_positive_rate3, true_positive_rate3)
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()

plt.subplots(1, figsize=(10,10))
plt.title('Receiver Operating Characteristic Curve for FFT model and Time model combined')
plt.plot(false_positive_rate3, true_positive_rate3)
plt.plot(false_positive_rate1, true_positive_rate1)
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

Figure 18: Code for ROC curve for FFT

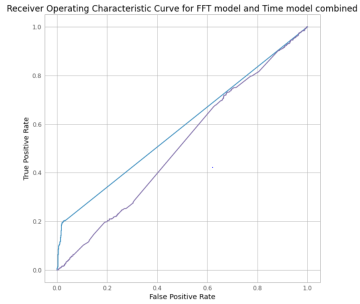

Figure 19: Combined ROC for SVM before and after FFT

References

Silva, D. F., Souza, V. M. D. and Batista, G. E. (2013). Time Series Classification Using Compression Distance of Recurrence Plots, *2013 IEEE 13th International Conference on Data Mining*, pp. 687–696. ISSN: 2374-8486.