

Configuration Manual

Research Project
MSc in Data Analytics

Nandhavarman Jeevarathinam

Student ID: x18186459

School of Computing
National College of Ireland

Supervisor: Dr. Paul Stynes

National College of Ireland
Project Submission Sheet
School of Computing



Student Name:	Nandhavarman Jeevarathinam
Student ID:	x18186459
Programme:	MSc in Data Analytics
Year:	2020
Module:	Research Project
Supervisor:	Dr. Paul Stynes
Submission Due Date:	30/09/2020
Project Title:	Configuration Manual
Word Count:	899
Page Count:	7

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:	
Date:	28th September 2020

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	<input type="checkbox"/>
Attach a Moodle submission receipt of the online project submission , to each project (including multiple copies).	<input type="checkbox"/>
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.	<input type="checkbox"/>

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

Nandhavarman Jeevarathinam
x18186459

1 Introduction

This document will discuss the hardware, software requirement and system configuration needed for to carry out this research project. Below are the steps that need to be followed to create the deep learning model developed in this research project.

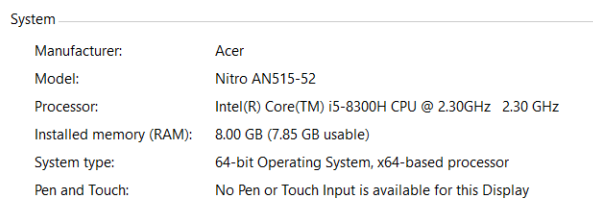
2 System Configuration

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 Spyder IDE: Data cleaning, data pre-processing, analysis and visualization.
Microsoft Excel: Used for saving of data.



System	
Manufacturer:	Acer
Model:	Nitro AN515-52
Processor:	Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz 2.30 GHz
Installed memory (RAM):	8.00 GB (7.85 GB usable)
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

This research project is implemented using the below steps as mentioned in the flow chart. The whole research can be divided into two main parts.

- Data pre-processing.
- Classification task.

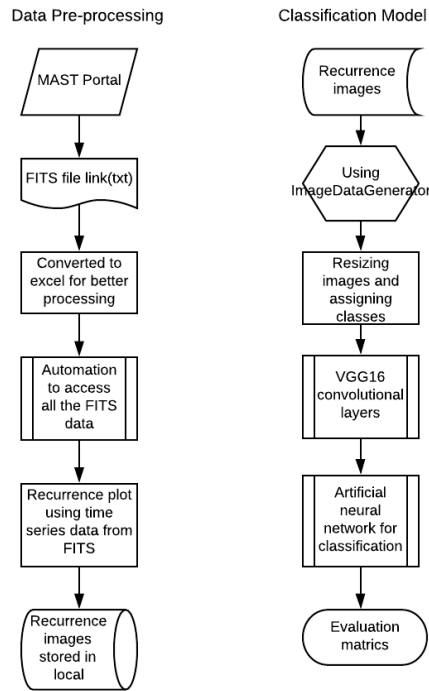


Figure 2: Implementation workflow

4 Data Pre-processing

4.1 Mikulski Archive for Space Telescope (MAST) Portal

Figure 3: Mast portal

Using the MAST portal, the data is chosen using different search parameters provided by the portal. The links to the light curves are downloaded as a text file to the local machine.

4.2 FITS link file

The link file downloaded from the MAST portal has all the URLs to download the light curves. These links will be accessed using astropy.io package in python to extract the time series data.

Links
https://archive.stsci.edu/missions/kepler/lightcurves/0014/001433410/kplr001433410-2012277125453_llc.fits
https://archive.stsci.edu/missions/kepler/lightcurves/0014/001433410/kplr001433410-2013011073258_llc.fits
https://archive.stsci.edu/missions/kepler/lightcurves/0015/001572353/kplr001572353-2009166043257_llc.fits
https://archive.stsci.edu/missions/kepler/lightcurves/0015/001572353/kplr001572353-2009259160929_llc.fits
https://archive.stsci.edu/missions/kepler/lightcurves/0015/001572353/kplr001572353-2009350155506_llc.fits
https://archive.stsci.edu/missions/kepler/lightcurves/0015/001572353/kplr001572353-2010078095331_llc.fits
https://archive.stsci.edu/missions/kepler/lightcurves/0015/001572353/kplr001572353-2010174085026_llc.fits
https://archive.stsci.edu/missions/kepler/lightcurves/0015/001572353/kplr001572353-2010265121752_llc.fits
https://archive.stsci.edu/missions/kepler/lightcurves/0015/001572353/kplr001572353-2010355172524_llc.fits
https://archive.stsci.edu/missions/kepler/lightcurves/0015/001572353/kplr001572353-2011073133259_llc.fits

Figure 4: FITS link file

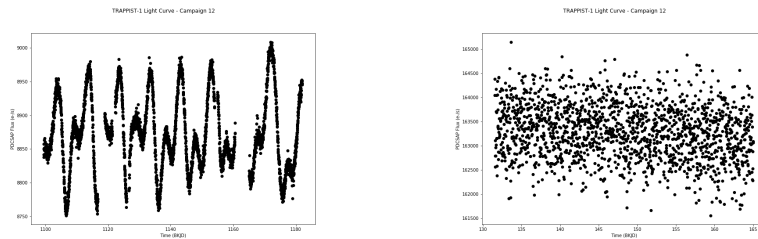
4.3 Accessing FITS data

```
for link in range(len(file1)):
    fits_file = file1['Links'][link]
    with fits.open(fits_file, mode="readonly") as hdulist:
        k2_bjds = hdulist[1].data['TIME'].byteswap().newbyteorder()
        sap_fluxes = hdulist[1].data['SAP_FLUX'].byteswap().newbyteorder()
        pdcsap_fluxes = hdulist[1].data['PDCSAP_FLUX'].byteswap().newbyteorder()
        data = {'Time': k2_bjds, 'flux': pdcsap_fluxes}
        struct_data = pd.DataFrame(data)
        struct_data = struct_data.dropna()
```

Figure 5: Code for accessing time series data

The FITS link file is loaded into the Python IDE using pandas package. Then astropy.io package is used to access each and every link as in research work [1]. Only 'TIME', 'SAP FLUX' and 'PDCSAP FLUX' data is extracted from the FITS data. Then they are stored in a dataframe for plotting the time series.

4.4 Plotting time series for experiment 1



(a) Positive case

(b) Negative case

Figure 6: Light curves

Using the 'TIME' and 'PDCSAP FLUX' data of each and every light curve, the time series graph are plotted and saved into the local machine. For experiment 1, these time series images are given as input to the VGG16 convolutional network.

4.5 Recurrence plot for experiment 2 and 3

The time series data which was stored dataframe is converted to recurrence plot data using the below algorithm in figure 7. For creating a recurrence plot, sklearn and numpy package from python are needed. Recurrence plots are created from the time series data extracted from the FITS file. It is used in this research work as it provides better pattern recognition for the convolutional neural network [1]. A custom function is made for creating the recurrence plot using the time series data. The recurrence plot custom function is added in the code artifact.

```
def recurrence_plot(s, eps=None, steps=None):
    if eps==None: eps=5
    if steps==None: steps=5
    d = sk.metrics.pairwise.pairwise_distances(s)
    d = np.floor(d / eps)
    d[d > steps] = steps
    #Z = squareform(d)
    return d
```

Figure 7: Code for recurrence plot

The recurrence plot is then used as a input for VGG16 convolutional architecture in experiment 2. VGG16 architecture recognises the pattern better than feeding the time series plot as a input.

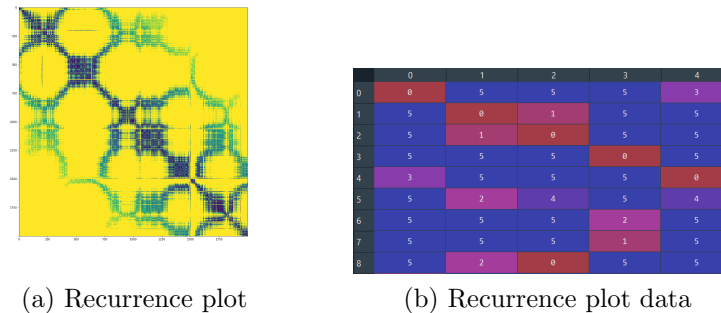


Figure 8: Recurrence plot

Then recurrence plot is plotted and stored to our local machine using the below code.

```
struct_data = struct_data.dropna()
struct_data = struct_data[0:2000]
plt.figure(figsize=(15,15))
plt.imshow(recurrence_plot(struct_data))
plt.savefig('C:/Users/nandh/Desktop/ETTS/Thesis/train_3/Class 2/input'+f"{link:2d}"+' .png')
```

Figure 9: Saving to local machine

5 Classification model for all experiments

5.1 Packages required

```
import keras
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Flatten, Dense, Dropout
from keras.optimizers import RMSprop
from keras.callbacks import EarlyStopping, ReduceLRonPlateau
import matplotlib.pyplot as plt
```

Figure 10: Packages required

Figure 10 shows the packages used to implement the VGG16 architecture for image classification.

5.2 Image Augmentation

The main motive of using image augmentation is to modify the original image by resizing, rotating images, zooming etc to design more new images. With this concept, the classification model will have more images or features to capture than before which will increase the exposure to unseen data.

```
train_datagen = ImageDataGenerator(
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    rescale=1/255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest',
    validation_split=0.2
)
test_datagen = ImageDataGenerator(
    rescale=1/255,
)
```

Figure 11: Image Augmentation

Addition of image augmentation is done with TensorFlow image data generator. When different type of augmentation are done, the original image data will be unaffected by this functionality.

5.3 Resizing the input size

```
batch_size = 4

train_generator = train_datagen.flow_from_directory(
    "C:/Users/nandh/Desktop/FITS/Thesis/train",
    target_size=(512, 512),
    batch_size=batch_size,
    class_mode='binary',
    subset='training',
)
validation_generator = train_datagen.flow_from_directory(
    "C:/Users/nandh/Desktop/FITS/Thesis/test",
    target_size=(512, 512),
    batch_size=batch_size,
    class_mode='binary',
    subset='validation'
)
```

Figure 12: Resizing and assigning classes

Using this code snippet, the test and train data are separated. While assigning the data, the input images can be resized as well. In our methodology, the images are resized to 512 x 512 size.

5.4 VGG16 architecture

The VGG16 architecture is a in built convolutional neural network of keras package. We are taking the input layer as (512,512,3) as we resized the images to 512 x 512. For the weights parameter, 'imagenet' is chosen as it performs well with image data.

```

prior = keras.applications.VGG16(
    include_top=False,
    weights='imagenet',
    input_shape=(512, 512, 3)
)
model = Sequential()
model.add(prior)
model.add(Flatten())
model.add(Dense(256, activation='relu', name='Dense_Intermediate'))
model.add(Dense(128, activation='relu', name='Dense_Intermediate1'))
model.add(Dense(64, activation='relu', name='Dense_Intermediate2'))
model.add(Dropout(0.1, name='Dropout_Regularization'))
model.add(Dense(1, activation='sigmoid', name='Output'))
model.summary()

```

Figure 13: VGG16 architecture

Then for the neural network, a sequential model is created from keras using three hidden layers with 256, 128 and 64 nodes, a dropout layer and an output layer with sigmoid activation function.

```

for cnn_block_layer in model.layers[0].layers:
    cnn_block_layer.trainable = False
model.layers[0].trainable = False

```

Figure 14: Weights for CNN nodes

The code snippet in figure 14 is used to freeze the VGG16 model from changing any weights. It is used with default weights.

```

# Compile the model. I found that RMSprop with the default learning
# weight worked fine.
model.compile(
    optimizer=RMSprop(),
    loss='binary_crossentropy',
    metrics=['accuracy']
)

import os
labels_count = dict()
for img_class in [ic for ic in os.listdir('C:/Users/nandh/Desktop/FTIS/Thesis/train/') if ic[0] != '.']:
    labels_count[img_class] = len(os.listdir('C:/Users/nandh/Desktop/FTIS/Thesis/train/' + img_class))
total_count = sum(labels_count.values())
class_weights = {cls: total_count / count for cls, count in
    enumerate(labels_count.values())}

```

Figure 15: Model compiler and calculating total count and class weights

Model is compiled using RMSprop and 'binary_crossentropy'. Using OS package, the image in each classes are counted which is used to calculate total count and class weights. These parameters will be used while training the module.

```

history = model.fit_generator(
    train_generator,
    steps_per_epoch=len(train_generator.filesnames) // batch_size,
    epochs=20,
    validation_data=validation_generator,
    validation_steps=len(train_generator.filesnames) // batch_size,
    class_weight=class_weights,
    callbacks=[
        EarlyStopping(patience=3, restore_best_weights=True),
        ReduceLROnPlateau(patience=2)
    ]
)

```

Figure 16: Training the model

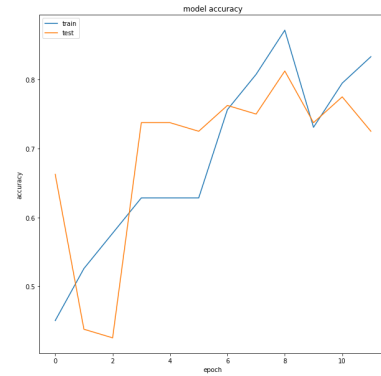
The steps per epoch and validation steps are dynamic depending upon the count of training data used. Number of epoch is 20 where if there is no improvement for three continuous epochs then the model stops training using callbacks functionality.

5.5 Evaluation

The model trained is evaluated using the following snippet for all three different experiments.


```
# summarize history for accuracy
plt.figure(figsize=(10,10))
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

(a) Model accuracy code

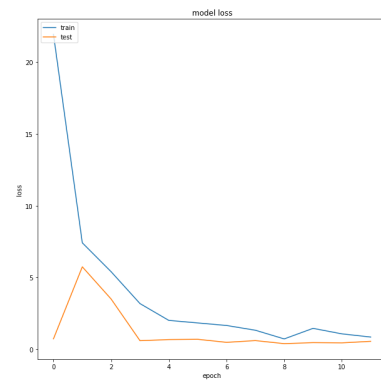


(b) Model accuracy plot

Figure 17: Model loss evaluation

```
# summarize history for loss
plt.figure(figsize=(10,10))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

(a) Model loss code



(b) Model loss plot

Figure 18: Model loss evaluation

Using the above code from figure 17, the train and test accuracy for all the epochs are plotted for comparison between experiments.

Using the above code snippet from figure 18, the model loss plot is plotted for every epoch. Both train and test loss is plotted.

References

- [1] Silva, Diego Alves de Souza, Vinícius Batista, Gustavo. (2013). Time Series Classification Using Compression Distance of Recurrence Plots. Proceedings - IEEE International Conference on Data Mining, ICDM. 687-696. 10.1109/ICDM.2013.128.