Configuration Manual

MSc Research Project
Data Analytics

Preety Kumari
Student ID: x18128556

School of Computing
National College of Ireland

Supervisor: Prof. Noel Cosgrave

Student Name:	Preety Kumari
Student ID:	x18128556
Programme:	Data Analytics
Year:	2018
Module:	MSc Research Project
Supervisor:	Prof. Noel Cosgrave
Submission Due Date:	20/12/2018
Project Title:	Configuration Manual
Word Count:	1448
Page Count:	7

Signature:	
Date:	11th December 2019

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.	☐

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

Contents

Configuration Manual

Preety Kumari
x18128556

1 Introduction

This configuration manual helps the user's to understand and follow the steps involved in implementing the hybrid models proposed for land use and land cover scene classification. This manual also helps the user's to understand the internal details of the different artefacts developed during this project. System configuration, development environment artefacts, process input and output as well as steps involved in experimentation are explained in this manual.

Only the technical aspects of the developed application is specified in the manual as this is supplementary to the dissertation thesis report submitted on the topic - "Hybrid Model Combining Residual Neural Networks and SVM for Land Use and Land Cover Scene Classification". For details regarding concepts and functionalities of the models, the users are requested to refer the project report.

2 System Configuration

This section provides a general overview of the system with its environment that was used to perform the experiments in this research. It describes the hardware and software configurations that was used for performing the analysis. Note that changes in hardware and software configurations may lead to different classification results.

2.1 Hardware

- Operating System: Windows 10, 64-bit
- RAM: 16 GB
- Storage: 512 GB SSD
- Processor: Intel(R) Core(TM) i5 CPU @1.80 Ghz Processor

2.2 Software

- Anaconda: 4.7.12, an enterprise-ready, scalable and secure platform that empowers programmers to collaborate and deploy their projects. It comes with Python distribution, hence Python installation is not required explicitly. It also provides python development environment such as Jupyter Notebook, JupyterLab, Spyder, etc.
- Spyder: 3.3.6, python development environment used in this research.

- Python: 3.7.5
- Keras: 2.3.1

3 Prerequisites

The following steps should be carried out before moving forward with the experiments.

1. All softwares mentioned in the Section 2.2 should be installed and environment should be set.
2. Dataset should be downloaded, unzipped and stored in some directory in the current working directory.

   Data Source: `http://www.escience.cn/people/JunweiHan/NWPU-RESISC45.html` contains one zipfile consisting of directories containing input images for each class.

4 Application Artefacts

Some variables are common across all the models. These are explained below:

1. base_path - Path to base directory that will contain train and test data after splitting. Default value set to ”dataset”.
2. output_path - Path to directory that will contain extracted bottleneck features and their corresponding labels. Default value set to ”output”.
3. class_names - Contains list of LULC class names.

4.1 Argument Parser

Source Code: argument_parser.py

Purpose: Parser to parse command line arguments for feature extraction and classification.

Available Methods:

1. classifier_args() to parse command line arguments used for classification.
2. feature_extractor_args() to parse the command line arguments necessary for bottleneck feature extraction.

Output: Returns final command line argument list after parsing.

4.2 FC Layers Classifier

Source Code: fc_classifier.py

Purpose: To return fully-connected layers model with initial parameters if no hyper-parameter tuning is to be performed or set of parameters to perform hyper-parameter tuning.

Available Methods:

1. create_base_model() to create and return model with some initial parameters.
2. create_model() to create and return model with parameter grid for hyper-parameter tuning.

Output: Returns model and set of parameters for hyper-parameter tuning.

4.3 SVM Classifier

Source Code: svm_classifier.py

Purpose: To return support vector machine model with initial parameters if no hyper-parameter tuning is to be performed or set of parameters to perform hyper-parameter tuning.

Available Methods:

1. create_best_model() to create and return model with some initial parameters.
2. create_model() to create and return model with parameter grid for hyper-parameter tuning.

Output: Returns model and set of parameters for hyper-parameter tuning.

4.4 Data Splitting

Source Code: split_data.py

Purpose: To split input images into train and test set.

Command Line Arguments:

- --input_path : Path to input data directory. User specific, hence mandatory.

Process Steps:

1. Removes and then creates base_path if already exists. Creates train and test sub-directories inside base_path with individual class names.
2. Reads images from directories present in class_names names from input_path.
3. Shuffles the all read images per class and splits them into train and test set in the ratio of 70%-30%.

4. Finally, copies files from input_path to train and test sub-directories for the respective classes.

Output: Directory structure as shown in the Figure 1 will be created for base_path.

4.5 Bottleneck Feature Extraction

Source Code: bottleneck_feature_extractor.py

Purpose: To extract bottleneck features from dense layers of pre-trained ResNet-152 model.

Command Line Arguments:

- --input_path : Path to input data directory. User specific, hence mandatory.

Process Steps:

1. Splits input data by calling split_data.py internally.
2. Applies data augmentation and pre-processing for all input samples.
3. Builds a compiled ResNet-152 model by removing the last dense layer for feature extraction.
4. Extracts bottleneck features from ResNet-152 model with their corresponding labels.
5. Saves features and labels in the output_path.

Output: Files containing features and labels will be created in the output_path. Files are saved with the following naming convention:

Features - [mode]_features_resnet152.h5

Labels - [mode]_labels_resnet152.h5

Where mode can either be "train" or "test".

Display: The name of each image file as it is processed is shown.

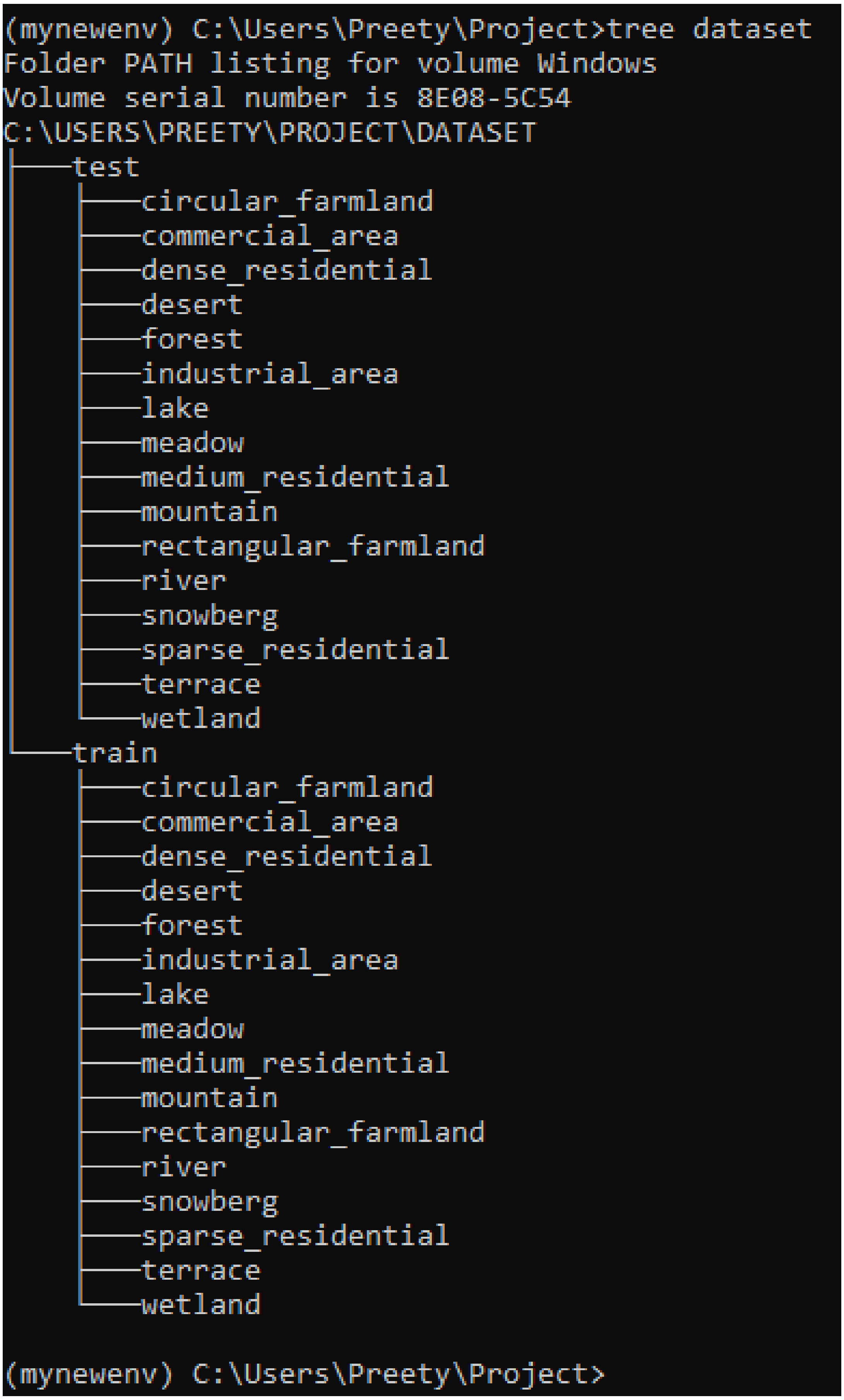
4.6 Base Classifier

Source Code: base_classifier.py

Purpose: To run both types of classifies with initial parameters.

Command Line Arguments:

- --type : "svm" to perform classification using SVM model. Absence of this argument will mean default classifier which is "fc".



```
(mynewenv) C:\Users\Preety\Project>tree dataset
Folder PATH listing for volume Windows
Volume serial number is 8E08-5C54
C:\USERS\PREETY\PROJECT\DATASET
├───test
│   ├───circular_farmland
│   ├───commercial_area
│   ├───dense_residential
│   ├───desert
│   ├───forest
│   ├───industrial_area
│   ├───lake
│   ├───meadow
│   ├───medium_residential
│   ├───mountain
│   ├───rectangular_farmland
│   ├───river
│   ├───snowberg
│   ├───sparse_residential
│   ├───terrace
│   └───wetland
└───train
    ├───circular_farmland
    ├───commercial_area
    ├───dense_residential
    ├───desert
    ├───forest
    ├───industrial_area
    ├───lake
    ├───meadow
    ├───medium_residential
    ├───mountain
    ├───rectangular_farmland
    ├───river
    ├───snowberg
    ├───sparse_residential
    ├───terrace
    └───wetland

(mynewenv) C:\Users\Preety\Project>
```

Figure 1: Directory structure created after splitting data

Process Steps:

1. Parse command line arguments using argument_parser.py
2. Reads features and labels from output_path using the load_data function.
3. Create model using either fc_classifier.py or svm_classifier.py.
4. Train the model on the training data depending upon the value of "--type" parameter.
5. Evaluate model using test data.
6. All the necessary metrics are stored in the output_path.

Output: File containing classification metrics for the model will be created in the output_path with the naming convention: results_base_[classifier].txt, where classifier can either be "fc" or "svm".

4.7 Tuned Classifier

Source Code: tuned_classifier.py

Purpose: To run both types of classifies by applying hyper-parameter tuning.

Command Line Arguments:

- --type : "svm" to perform classification using SVM model. Absence of this argument will mean default classifier which is "fc".

Process Steps:

1. Parse command line arguments using argument_parser.py
2. Reads features and labels from output_path using the load_data function.
3. Create model depending upon the value of "--type" argument with some set of parameters using either fc_classifier.py or svm_classifier.py.
4. Perform random search on the model to determine the best set of hyper-parameters for the model.
5. Train the tuned model on the training data and evaluate using test data.
6. All the necessary metrics are stored in the output_path.

Output: File containing classification metrics for the model will be created in the output_path with the naming convention: results_base_[classifier].txt, where classifier can either be "fc" or "svm".

5 Commands for performing experiments

Each experiment is divided into two steps namely, feature extraction and classification. Example commands for both the steps are specified below.

1. **Feature extraction:** Here ”./RESISC45/” is the path to local directory where downloaded data is stored.

   ```
   python bottleneck_feature_extractor.py –input_path ”./RESISC45/”
   ```

2. **Classification:** Example 1 will run fully-connected (FC) layers classifier with initial parameters which means no hyper-parameter tuning. Example 2 will use SVM for classification after applying hyper-parameter tuning.

   ```
   Example 1: python base_classifier.py
   Example 2: python tuned_classifier.py –type ”svm”
   ```

6 Best Practices

1. Always keep a backup of raw data, each version of the artefacts developed and outputs generated by the program.

2. Since actions such as deletion of old directories with specific names and generation of new ones is carried out during the execution of the program. It is recommended to perform these experiments in a new empty directory, which will prevent deletion of wrong files.

7 Future Work

Changes to improve the usability of the developed artefacts can be taken up in future. Following code related tasks are recommended for future work:

1. Merging of base_classifier.py and tuned_classifier.py is recommended by supporting command line options to run the program with or without hyper-parameter optimization.

2. Different decision function for SVM should be tried to improve the overall accuracy of the model.

3. Pre-processing and feature extraction code is currently merged. This should be segregated, as they are completely different activities.

4. Hard coding of base_path and output_path should be removed by implementing the support to specify these as command line arguments.