

Aerial Scene Classification using Deep Transfer Learning Network and Tree-based Classifiers

Research Project Configuration Manual
M.Sc.in Data Analytics

Abdul Azadh Abdul Saleem
Student ID: x18203621@student.ncirl.ie

School of Computing
National College of Ireland

Supervisor: Dr.Hicham Rifai

**National College of Ireland
Project Submission Sheet
School of Computing**



Student Name:	Abdul Azadh Abdul Saleem
Student ID:	x18203621@student.ncirl.ie
Programme:	M.Sc.in Data Analytics
Year:	2020-21
Module:	Research Project Configuration Manual
Supervisor:	Dr.Hicham Rifai
Submission Due Date:	17/12/2020
Project Title:	Aerial Scene Classification using Deep Transfer Learning Network and Tree-based Classifiers
Word Count:	1703
Page Count:	16

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:	16th December 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):	

Aerial Scene Classification using Deep Transfer Learning Network and Tree-based Classifiers

Abdul Azadh Abdul Saleem
x18203621@student.ncirl.ie

1 Introduction

This document depicts the configuration manual of the research project titled ‘**Scene Classification of Aerial Images using Transfer Learning and Tree-based Classifiers**’ and also covers the system requirements and environmental set-up required for the successful execution of the research. This document also explains the procedure to conduct the extraction of features, then to classify them with tree-based classifiers and then testing the performance of the classifiers using the Graphical User Interface (GUI).

2 System Configuration

This section explains the system configuration required for the successful execution of all the modules of the project.

2.1 Hardware Requirement

The below Figure 1 shows the hardware specification that has been utilized for the research, but minimum requirement of 8GB RAM is enough for the successful execution of the project.

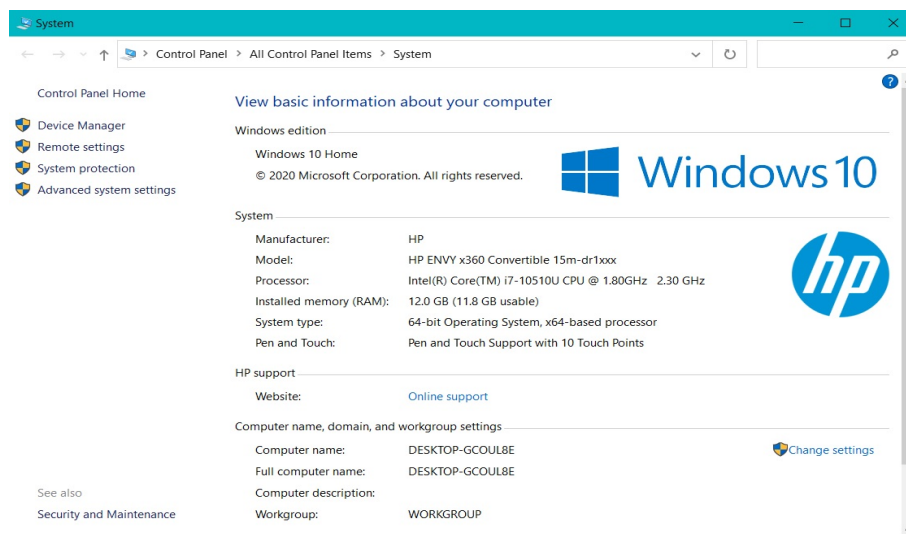


Figure 1: Hardware Specification

2.2 Software Requirement

This section all the softwares that has been utilized for the project, along with the python packages and its version required for the project.

2.2.1 Softwares

The below Table 1 provides the information about the softwares required for the execution of the research modules and the version information. The Project Interpreter for virtual

Table 1: Softwares and Version Required

Software	Version
Anaconda	4.8.4
PyCharm	2020.1.2

python environment that has been utilized for this project is Python 3.6 version as shown in the Figure 2.

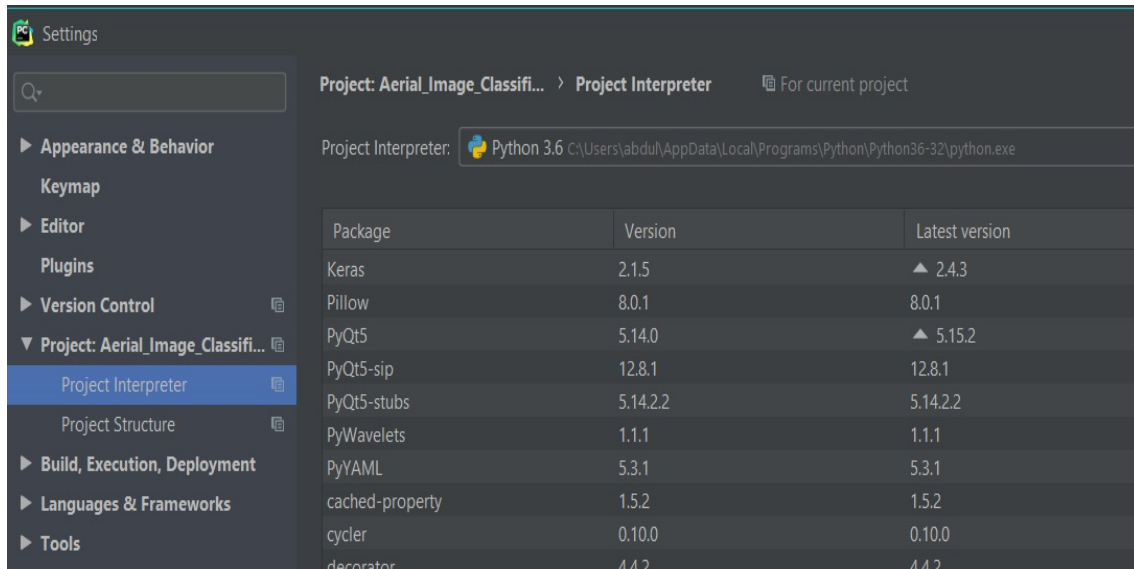


Figure 2: Settings for Project Interpreter

2.2.2 Python Libraries

The requirement of the project has been mentioned in the file named 'requirements.txt' along with the code artifacts, which would help to set-up the required packages before execution of the project. The below Table 2 shows the Python Library Packages and its version which are specified to be required in order to execute the project modules.

Table 2: Python Library Packages and Version Required

Python Library	Version	Python Library	Version
Keras	2.1.5	pandas	0.25.1
numpy	1.16.0	pyqt5	1.9.0
tensorflow	1.14.0	scikit-learn	0.21.3
scipy	1.1.0	tqdm	4.36.1
pyqt5	5.14	seaborn	0.9
h5py	2.9.0	matplotlib	3.1.1

3 Project Development

The project development of this research consist of three modules, which has to be executed in sequence in order to get the target result. The three modules of this project includes Feature Extraction, Feature Classification and User Interface for testing the classifier models.

The below Figure 3 shows the python files that has been included in the code artefacts submitted.










 assets	11/26/2020 11:01 AM	File folder	
 Data	12/15/2020 8:27 AM	File folder	
 Test	12/9/2020 12:43 PM	File folder	
 weights	11/26/2020 11:21 AM	File folder	
 feature_extractor.py	12/15/2020 8:24 AM	JetBrains PyCharm	3 KB
 gui.py	12/15/2020 7:15 AM	JetBrains PyCharm	10 KB
 requirements.txt	12/14/2020 8:51 AM	Text Document	1 KB
 trainer.py	12/13/2020 11:49 AM	JetBrains PyCharm	5 KB
 utils.py	12/15/2020 7:16 AM	JetBrains PyCharm	4 KB

Figure 3: Files included in Code Artefacts

- **assets** folder contains the font file that has been utilized for the project.
- **Data** folder is the folder were the Dataset has to be extracted and saved and after completion of feature extraction a new sub-folder called 'feature' is created in it, where the features and labels were stored.
- **Test** folder has to test images for testing
- **weights** folder has the pre-defined weights which has been utilize to pre-train the Inception V3 model.
- **feature_extractor.py** is python file which perform the function of extraction of features.
- **trainer.py** performs the process of classifying the land-use classes using Random Forest and Decision Tree.
- Execution of **gui.py** launches the Graphical User Interface, which utilize feature extraction function and trained classifier files to perform testing the performance of classifiers.

PROCEDURE TO EXECUTE THE MODULES:

The procedure to execute the python files has been summarized as follows.

1. Download and save the dataset as mentioned in Section 3.1.1.
2. Run **feature_extractor.py**.
3. Run **trainer.py**.
4. Run **gui.py**.

3.1 Feature Extraction

This module has been conducted as per the following procedure,

1. The downloading the datasets from the authorized sites.
2. Importing and pre-training the Inception V3 model.
3. Utilizing the pre-trained layers to extract deep features from each image.
4. Storing the features and labels as Hierarchical Data Format (HDF).

3.1.1 Data Acquisition

This project requires two datasets to be downloaded and to store them under the './Data/source' directory, in two different folders. The project has been completely executed with UCM dataset initially and evaluated. Later, the Aerial Image Detection (AID) dataset directory has been imported to perform the complete operation and then evaluated. Links to extract the UC Merced dataset ¹(Yang and Newsam (2010)) and AID dataset ²(Xia et al. (2017)) has been provided in the dataset.

```
if __name__ == '__main__':
    resetRandom()
    data_path = 'Data/source/UCM' # datapath for dataset
    data_save_path = 'Data/features' # datapath to save extracted features
    make_dirs(data_save_path) # creating a new directory
    feats_dict = {'inception_v3_features': []} # saving the features in variable
    labels = [] # empty array for labels
    iv3_model = get_inception_v3_model() # calling the transfer learning model for feature extraction
    for cls in CLASS_NAMES:
        for img_path in tqdm.tqdm(glob.glob(os.path.join(data_path, cls, '*.tif'))):
            desc=f'Extracting Features For Class {cls}':
            feats_dict['inception_v3_features'].append(extract_feature(img_path, iv3_model))
            labels.append(CLASS_NAMES.index(cls))
```

Figure 4: Code with Directory to import UCM dataset highlighted

The Figure 4 shows the directory of UCM dataset and the image format (.tif) to be imported to run the model.

¹<http://weegee.vision.ucmerced.edu/datasets/landuse.html>

²<https://captain-whu.github.io/AID/>

```

if __name__ == '__main__':
    resetRandom()
    data_path = 'Data/source/AID' # datapath for dataset
    data_save_path = 'Data/features' # datapath to save extracted features
    make_dirs(data_save_path) # creating a new directory
    feats_dict = {'inception_v3_features': []} # saving the features in variable
    labels = [] # empty array for labels
    iv3_model = get_inception_v3_model() # calling the transfer learning model for feature extraction
    for cls in CLASS_NAMES:
        for img_path in tqdm.tqdm(glob.glob(os.path.join(data_path, cls, '*.jpg')),
                                   desc=f'Extracting Features For Class {cls}'):
            feats_dict['inception_v3_features'].append(extract_feature(img_path, iv3_model))

```

Figure 5: Code with Directory to import AID dataset highlighted

The Figure 5 shows the directory and the image file format (.jpg) that has to be changed to import the AID dataset. The function performed by the code will be explained in section 3.1.3.

3.1.2 Importing Inception V3 model

To import the transfer learning model and to pre-train the deep layers with the pre-defined weights will be performed by the function ‘**get_inception_v3_model()**’. This function imports the Inception V3 layers and the input layer separately from the python **keras** packages. And pre-train the model with the pre-defined weight provided in the ‘./weights’ directory as shown in the Figure 6.

```

def get_inception_v3_model():
    from tensorflow.python.keras.applications.inception_v3 import InceptionV3
    from tensorflow.python.keras.models import Input
    model = InceptionV3(include_top=False,
                        weights='weights/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5',
                        input_tensor=Input(shape=(299, 299, 3)))
    model.summary()
    return model

def extract_feature(img_p, model):
    from tensorflow.python.keras.preprocessing import image
    from tensorflow.python.keras.applications.inception_v3 import preprocess_input
    size = (299, 299, 3)
    img = image.load_img(img_p, target_size=size)
    x = image.img_to_array(img)
    x = np.expand_dims(x, axis=0)
    x = preprocess_input(x)
    feature = model.predict(x)
    feat = feature.flatten()
    return feat

```

Figure 6: Functions to Import and Utilize the Inception V3 Layers

The another function ‘**extract_feature()**’ as shown in Figure 6, performs the function of importing the aerial images into the deep convolutional layers and extracting the features from each image. This function also performs the pre-processing steps before

importing the image into the Inception V3 layers and the numpy array of features extracted from the output layer is flattened in order to provide them as input to the machine learning classifiers.

The functions ‘`get_inception_v3_model()`’ and ‘`extract_feature()`’ will be utilized in both the **Feature Extraction** module and the **User Interface** module to extract the feature of the test image.

3.1.3 Extraction of Features using Inception V3 Layers

The extraction of feature from every image in every land-use class has been performed by the execution of the code in the main function shown in Figure 7.

```
if __name__ == '__main__':
    resetRandom()
    data_path = 'Data/source/UCM'                # datapath for dataset
    data_save_path = 'Data/features'             # datapath to save extracted features
    make_dirs(data_save_path)                    # creating a new directory
    feats_dict = {'inception_v3_features': []}    # saving the features in variable
    labels = []                                 # empty array for labels
    iv3_model = get_inception_v3_model()         # calling the transfer learning model for feature extraction
    for cls in CLASS_NAMES:
        for img_path in tqdm.tqdm(glob.glob(os.path.join(data_path, cls, '*.tif'))):
            desc=f'Extracting Features For Class {cls}':
                feats_dict['inception_v3_features'].append(extract_feature(img_path, iv3_model))
                labels.append(CLASS_NAMES.index(cls))
    for k, v in feats_dict.items():
        if v:
            h5f_data = h5py.File(os.path.join(data_save_path, '{0}.h5'.format(k)), 'w')
            h5f_data.create_dataset('features', data=np.array(v))
            h5f_data.close()
```

Figure 7: Extraction of Features using Inception V3 Layers

The directory to extract the data and the directory where the extracted features and labels has to be stored has been mentioned. This main function mentioned in the Fig.7 calls the ‘`get_inception_v3_model()`’ function to utilize the layers to extract the features. Every image in the every land-use classes has been made to flow through the layers in order to extract the features with reduced parameters. By utilizing the ‘`tqdm`’ package the progress of the process of extraction has been visualized in the console, with progress-bar showing the number of images extracted from each class as shown in Figure 8 . The extracted features appended on dictionary and labels on an array separately.

```
=====
activation_93[0][0]
=====
Total params: 21,802,784
Trainable params: 21,768,352
Non-trainable params: 34,432
Extracting Features For Class agricultural: 100%|
Extracting Features For Class airplane: 100%|
Extracting Features For Class baseballdiamond: 100%|
Extracting Features For Class beach: 100%|
```

100/100	[00:16<00:00,	6.23it/s]
100/100	[00:15<00:00,	6.44it/s]
100/100	[00:15<00:00,	6.37it/s]
100/100	[00:16<00:00,	6.19it/s]

Figure 8: Console displaying the progress of Feature Extraction Process

3.1.4 Storing Extracted Features

The code shown in the Figure 9 performs the operation of creating the HDF5 dataset and append the features values. The feature dataset and the labels array were later stored by creating a new directory './Data/features' in two separate file 'inception_v3_feature.h5' and 'labels.h5'.

```
for k, v in feats_dict.items():
    if v:
        h5f_data = h5py.File(os.path.join(data_save_path, '{0}.h5'.format(k)), 'w')
        h5f_data.create_dataset('features', data=np.array(v))
        h5f_data.close()

h5f_data = h5py.File(os.path.join(data_save_path, 'labels.h5'), 'w')
h5f_data.create_dataset('labels', data=np.array(labels))
h5f_data.close()
```

Figure 9: Storing Features and Labels

3.2 Feature Classification

This section covers the utilization of features and labels from previous module in Section 3.1, to train the classifiers and to evaluate the performance of the model.

3.2.1 Importing the Features and Labels

The features and labels stored in the previous module were imported and stored in two separate arrays 'fes' and 'lbs' as shown in the Figure 10.

```
# looping features keys
for feature in features.keys():
    # reloading saved features from hdf5 file
    fe_path = os.path.join(features_dir, '{0}_{1}.h5'.format(feature, 'features'))
    feats = h5py.File(fe_path, 'r')
    fes = np.array(feats.get('features'))
    # reloading saved labels from hdf5 file
    lb_path = os.path.join(features_dir, 'labels.h5')
    labels = h5py.File(lb_path, 'r')
    lbs = np.array(labels.get('labels'))
    # store to features
    features[feature]['x'] = fes
    features[feature]['y'] = lbs
```

Figure 10: Importing Features and Labels file

3.2.2 Training and Evaluating Classifiers

The below Figure 11 shows the dictionary of classifiers and the parameters used in the research.

```
classifiers = {  
    'inception_v3': [  
        DecisionTreeClassifier(max_leaf_nodes=360),  
        RandomForestClassifier(n_estimators=10),  
    ],  
}
```

Figure 11: Classifiers and Parameters.

The features and labels were split into Training and Testing data with the proportion of (80:20) as shown in the Figure 12. The models were trained with the training data using 'classifier.fit(train_x,train_y)' command.

```
# looping features dict  
for feature in list(features.keys()):  
    x = features[feature]['x']  
    y = features[feature]['y']  
    train_x, test_x, train_y, test_y = train_test_split(x, y, test_size=0.2)  
    print('[INFO] Train Data Shape :: {0} | Train Labels Shape :: {1}'.format(train_x.shape, train_y.shape))  
    print('[INFO] Test Data Shape :: {0} | Test Labels Shape :: {1}'.format(test_x.shape, test_y.shape))  
    # looping classifiers  
    for classifier in classifiers[feature]:  
        # resetting all random value generators by seed 1  
        resetRandom()  
  
        c_name = classifier.__class__.__name__  
        classifier_path = os.path.join(classifier_dir, '{0}_{1}.pkl'.format(feature, c_name))  
        print('[INFO] Feature :: {0} | Classifier :: {1}'.format(feature, c_name))  
        # fitting x and y to train classifier  
        classifier.fit(train_x, train_y)
```

Figure 12: Train and Test Split of Data and Fitting the Classifiers with Training data.

The trained classifier models were evaluated with the Training and Testing Dataset as shown in the Figure 13.

```
# TRAINING DATASET
train_pred = classifier.predict(train_x)

# storing classification report
train_res = classification_report(train_y, train_pred, output_dict=True)

print('\tTraining Accuracy :: {0} %'.format(round(train_res['accuracy'] * 100, 2)))

# storing confusion matrix
train_cm = confusion_matrix(train_y, train_pred)
print('\tTraining Confusion Matrix\n{0}'.format(train_cm))

title = f'Features :: {feature} | Classifier :: {c_name}'
plot_conf_matrix(train_cm, title, CLASS_NAMES, 'results/{0}_{1}_train.png'.format(feature, c_name))

# TEST DATASET
test_pred = classifier.predict(test_x)
# storing classification report
test_res = classification_report(test_y, test_pred, output_dict=True)
print('\tTesting Accuracy :: {0} %'.format(round(test_res['accuracy'] * 100, 2)))
# storing confusion matrix
test_cm = confusion_matrix(test_y, test_pred)
print('\tTesting Confusion Matrix\n{0}'.format(test_cm))
title = f'Features :: {feature} | Classifier :: {c_name}'
plot_conf_matrix(test_cm, title, CLASS_NAMES, 'results/{0}_{1}_test.png'.format(feature, c_name))
```

Figure 13: Evaluation of the Classifiers using Training and Testing Dataset.

3.2.3 Storing Classifiers and Results

This section covers the storage of trained classifiers and results of the evaluation of the performance of the classifiers in different directory. The trained classifier files were stored as pickle files for each classifier by creating a new directory './classifiers/', and the results of evaluation has been exported as '.csv' file and the confusion matrix as image in '.png' format and stored in './results/' directory. The Figure 14, shows the directory to import the input and to store the classifier file and results of evaluation.

```
features_dir = 'Data/features'
classifier_dir = 'classifiers'
results_dir = 'results'
os.makedirs(classifier_dir, exist_ok=True)
os.makedirs(results_dir, exist_ok=True)
```

Figure 14: Directory to import the data and store the output of the classifiers.

The Figure 15 shows the dictionary function to store the output of the evaluation of classifier.

```
# dict to store performance measures
classifier_out = {'Feature': [], 'Classifier': [], 'Accuracy': [],
                 'Precision': [], 'Recall': [], 'F1-Score': []}
```

Figure 15: Dictionary to store the results of Classifiers.

The trained classifiers were stored as pickle files which can be utilized to perform the scene classification of the test image in the User Interface module. The results were appended as the dataframe and printed on the ‘.csv’ file and the confusion matrix has been exported in ‘.png’ format. The commands in the Figure 16 shows the functions to stored the trained classifiers and results.

```
print('\tSaving Classifier {0}'.format(classifier_path))
# saving classifier to pickle file
classifier_pkl = open(classifier_path, 'wb')
pickle.dump(classifier, classifier_pkl)
classifier_pkl.close()
classifier_out['Feature'].append(feature)
classifier_out['Classifier'].append(c_name)
classifier_out['Accuracy'].append('{0}'.format(
    round(train_res['accuracy'] * 100, 2)
))
classifier_out['Precision'].append('{0}'.format(
    round(train_res['macro avg']['precision'] * 100, 2)
))
classifier_out['Recall'].append('{0}'.format(
    round(train_res['macro avg']['recall'] * 100, 2)
))
classifier_out['F1-Score'].append('{0}'.format(
    round(train_res['macro avg']['f1-score'] * 100, 2)
))

# saving performance measure to results.csv
pd.DataFrame.from_dict(classifier_out).to_csv('results/results.csv', index=False)
```

Figure 16: Commands to save the trained classifier files and results of evaluation

3.3 User Interface

The User Interface has been developed as shown in the Figure 17 to test the performance of classifiers with the test images which were randomly cropped randomly from the Google Earth³.

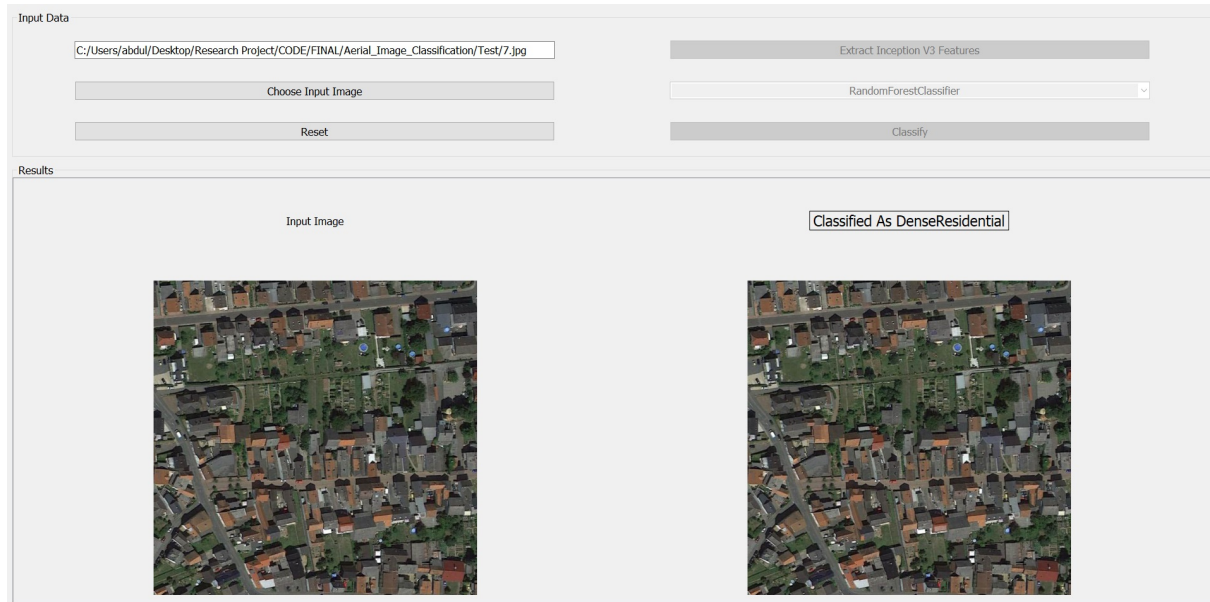


Figure 17: User Interface to test the performance of Classifier

Procedure to conduct test of scene classification in the User Interface.

1. Choose input image and import test image from './Test' folder.
2. Extract Inception V3 Features.
3. Choose any classifier.
4. Classify.

³<https://www.google.com/earth/>

This User Interface has been developed by utilizing different packages from PyQt5 GUI framework ⁴ as shown in the Figure 18.

```
from PyQt5.QtCore import Qt, QThreadPool
from PyQt5.QtGui import QFontDatabase, QFont, QFontValidator, QImage, QPixmap
from PyQt5.QtWidgets import (QDialog, QProgressBar, QApplication, QWidget, QComboBox, QVBoxLayout, QGroupBox,
                             QGridLayout, QLineEdit, QPushButton, QLabel, QFileDialog, QMessageBox, QScrollArea)

from feature_extractor import extract_feature, get_inception_v3_model
```

Figure 18: PyQt5 packages utilized to develop GUI.

This User Interface utilizes the ‘`get_inception_v3_model()`’ and ‘`extract_feature()`’ functions from the Feature Extraction Module (Section 3.1.2), to extract the features from the test image, and the classification of features has been conducted with the saved trained classifiers as mentioned in the Section 3.2.3.

Figure 19 shows the function ‘`add_image()`’ defines the operation of adding the test image to the User Interface, and to display the image using `Pixmap()` command. The configuration of the message box, which pops wherever an expected error occurs, has been provided with ‘`show_message_box`’ function.

```
def add_image(self, im_path, title, f_size=False):
    image_lb = QLabel()
    image_lb.setFixedHeight(self._image_size[0])
    image_lb.setFixedWidth(self._image_size[1])
    image_lb.setScaledContents(True)
    image_lb.setStyleSheet('padding-top: 30px;')
    qimg = QImage(im_path)
    pixmap = QPixmap.fromImage(qimg)
    image_lb.setPixmap(pixmap)
    self.grid_2.addWidget(image_lb, 1, self.index, Qt.AlignCenter)
    txt_lb = QLabel(title)
    if f_size:
        txt_lb.setFont(QFont('FiraCode', 14, 0))
        txt_lb.setStyleSheet("border: 1px solid black;")
    self.grid_2.addWidget(txt_lb, 0, self.index, Qt.AlignCenter)
    self.index += 1

def show_message_box(self, title, icon, msg):
    msg_box = QMessageBox(self)
    msg_box.setFont(QFont('FiraCode', 10, 1))
    msg_box.setWindowTitle(title)
    msg_box.setText(msg)
    msg_box.setIcon(icon)
    msg_box.setDefaultButton(QMessageBox.Ok)
    msg_box.setWindowModality(Qt.ApplicationModal)
    msg_box.exec_()
```

Figure 19: Functions in User Interface - 1.

⁴<https://pypi.org/project/PyQt5/>

Figure 20 shows the functions ‘choose_input()’ which helps in the operation of browsing the test from the local disk and adding it to the user interface. This function is also programmed to handle the exceptional case if the image was not properly added. And ‘clear_layout()’ function helps to clear the previous instance of input.

```
def choose_input(self):
    self.reset()
    _input_image_path, _ = QFileDialog.getOpenFileName(self,
                                                       caption="Choose Input Image", directory=".",
                                                       options=QFileDialog.DontUseNativeDialog,
                                                       filter="JPG Files (*.jpg);;"
                                                       "BMP Files (*.bmp);;PNG Files (*.png);;")

    if os.path.isfile(_input_image_path):
        self._input_image_path = _input_image_path
        self.ip_le.setText(self._input_image_path)
        self.add_image(self._input_image_path, 'Input Image')
        self.ex_fe_pb.setEnabled(True)
    else:
        self.show_message_box('InputImageError', QMessageBox.Critical, 'Choose valid image?')

    @staticmethod
    def clear_layout(layout):
        while layout.count() > 0:
            item = layout.takeAt(0)
            if not item:
                continue
            w = item.widget()
            if w:
                w.deleteLater()
```

Figure 20: Functions in User Interface - 2.

The functions mentioned in the Figure 21, helps to control the function of single thread which has been assigned to perform the feature extraction function, and the obtain the output.

```
def fe_extract_thread(self):
    worker = Worker(self.fe_extract_runner)
    self.thread_pool.start(worker)
    worker.signals.finished.connect(self.fe_extract_finisher)
    self.load_screen.setWindowModality(Qt.ApplicationModal)
    self.load_screen.show()

def fe_extract_runner(self, progress_callback):
    self.feature = extract_feature(self._input_image_path, self._iv3_model)
    progress_callback.emit('Done')

def fe_extract_finisher(self):
    self.cls_combo.setEnabled(True)
    self.cls_pb.setEnabled(True)
    self.ex_fe_pb.setEnabled(False)
    self.load_screen.close()
```

Figure 21: Functions of Feature Extraction in UI.

The functions mentioned in the Figure 22 defines the operation carried in the single thread established to preform Feature Classification function.

```
def classify_thread(self):
    if self.cls_combo.currentIndex() == 0:
        self.show_message_box('ClassifierError', QMessageBox.Critical, 'Please choose Classifier?')
    else:
        worker = Worker(self.classify_runner)
        self.thread_pool.start(worker)
        worker.signals.finished.connect(self.classify_finisher)
        self.load_screen.setWindowModality(Qt.ApplicationModal)
        self.load_screen.show()

def classify_runner(self, progress_callback):
    resetRandom()
    cls = self.cls_combo.currentText()
    cls_path = os.path.join('classifiers', 'inception_v3_{0}.pkl'.format(cls)) #importing the trained classifiers
    classifier = pickle.load(open(cls_path, 'rb')) #loading the classifiers to perform classification
    import numpy as np
    pred = classifier.predict(np.array(self.feature, ndmin=2))[0]
    self.class_ = CLASS_NAMES[pred]
    progress_callback.emit('Done')

def classify_finisher(self):
    self.add_image(self._input_image_path, 'Classified As {0}'.format(self.class_), f_size=True)
    self.class_ = None
    self.cls_combo.setEnabled(False)
    self.cls_pb.setEnabled(False)
    self.load_screen.close()
```

Figure 22: Functions of Feature Classification in UI.

4 Utilities

In addition to the main python files needed to execute the main functions of the project. The '**utils.py**' file, which is the utility file, has the supporting functions which supporting the flawless execution of the project. Some of the important functions mentioned programmed in the utility file are shown in the Figures 23 Figure 24.

Figure 23 shows the functions such as 'resetRandom()' which has been programmed to reset the random values assigned to different components of the tensorflow model, with seed = 1, everytime this function is executed. 'make_dir()' function is utilized wherever a new directory has to be created to store the files during the process, as mentioned in the Section 3.2.3 and Section 3.1.4. 'plot_conf_matrix()' has been configured to plot the heat map on the confusion matrix plotted to evaluate the performance of classifiers. The functions mentioned in the Figure 24, helps to control the events generated from arguments, signal from or to different functions, and slot mechanism.

```

def resetRandom():
    seed = 1
    os.environ['PYTHONHASHSEED'] = str(seed)
    random.seed(seed)
    np.random.seed(seed)
    tf.compat.v1.random.set_random_seed(seed)
    #session_conf = tf.compat.v1.ConfigProto(intra_op_parallelism_threads=1, inter_op_parallelism_threads=1)
    tf.compat.v1.set_random_seed(seed)
    #sess = tf.compat.v1.Session(graph=tf.compat.v1.get_default_graph(), config=session_conf)
    #k.set_session(sess)

def make_dirs(*dirs):
    for dir_ in dirs:
        os.makedirs(dir_, exist_ok=True)

def plot_conf_matrix(conf_mat, title, labels, path):
    plt.subplots(figsize=(19, 9))
    sbn.heatmap(conf_mat, annot=True, annot_kws={"size": 16}, linewidths=0.5, fmt='d',
                yticklabels=labels, xticklabels=labels, cmap='Blues')
    plt.xlabel('Predicted Class', labelpad=20)
    plt.ylabel('Actual Class', labelpad=20)
    plt.title(title, pad=20)
    plt.savefig(path, bbox_inches="tight")
    plt.close()
    plt.clf()

```

Figure 23: Functions in Utilities file - 1.

```

class WorkerSignals(QObject):
    finished = pyqtSignal()
    error = pyqtSignal(tuple)
    result = pyqtSignal(object)
    progress = pyqtSignal(int)

class Worker(QRunnable):
    def __init__(self, fn, *args, **kwargs):
        super(Worker, self).__init__()

        self.fn = fn
        self.args = args
        self.kwargs = kwargs
        self.signals = WorkerSignals()

        self.kwargs['progress_callback'] = self.signals.progress

    @pyqtSlot()
    def run(self):
        try:
            result = self.fn(*self.args, **self.kwargs)
        except:
            traceback.print_exc()
            exctype, value = sys.exc_info()[:2]
            self.signals.error.emit((exctype, value, traceback.format_exc()))
        else:
            self.signals.result.emit(result)
        finally:
            self.signals.finished.emit()

```

Figure 24: Functions in Utilities file - 2.

Figure 25 shows the Class folder names of the different datasets utilized for the project, which has to be commented and decommented, depending the dataset utilized to train the classifier.

```
# Class names for AID dataset
#CLASS_NAMES = ['Airport', 'BareLand', 'BaseballField', 'Beach', 'Bridge', 'Center', 'Church',
#               'Commercial', 'DenseResidential', 'Desert', 'Farmland', 'Forest', 'Industrial', 'Meadow',
#               'MediumResidential', 'Mountain', 'Park', 'Parking', 'Playground', 'Pond', 'Port',
#               'RailwayStation', 'Resort', 'River', 'School', 'SparseResidential', 'Square', 'Stadium',
#               'StorageTanks', 'Viaduct' ]

# Class names for UCM dataset
CLASS_NAMES = [
    'agricultural', 'airplane', 'baseballdiamond', 'beach', 'buildings', 'chaparral',
    'denseresidential', 'forest', 'freeway', 'golfcourse', 'harbor', 'intersection',
    'mediumresidential', 'mobilehomepark', 'overpass', 'parkinglot', 'river', 'runway',
    'sparseresidential', 'storagetanks', 'tenniscourt',
]
```

Figure 25: Class folder names of Datasets.

References

- Xia, G., Hu, J., Hu, F., Shi, B., Bai, X., Zhong, Y., Zhang, L. and Lu, X. (2017). Aid: A benchmark data set for performance evaluation of aerial scene classification, *IEEE Transactions on Geoscience and Remote Sensing* **55**(7): 3965–3981.
- Yang, Y. and Newsam, S. (2010). Bag-of-visual-words and spatial extensions for land-use classification, *Proceedings of the 18th SIGSPATIAL international conference on advances in geographic information systems*, pp. 270–279.