Online Geoprocessing using Multi-Dimensional Gridded Data

Amruth Kiran [1], Prasun Kumar Gupta [1], Ashutosh Kumar Jha [1], Sameer Saran [1]

[1] Geoinformatics Department, Indian Institute of Remote Sensing, Dehradun, India - amruthkiran94@gmail.com, (prasun, akjha, sameer)@iirs.gov.in

Commission V, WG V/4

KEY WORDS: Data Cube, Geoprocessing, Python, LISS III Data, NDVI, Online Spatial Analysis.

ABSTRACT:

Traditional geoprocessing techniques often rely on the use of multiple softwares for data handling and management which consumes almost 80% of the time and requires the user to be well versed with all the intricacies of pre-processing. Therefore, there is a need to reverse the trend on analysis and data management, so as to enable scientists and researchers to focus on the science rather than data handling and pre-processing. The concept of a Data Cube which is a massive multi-dimensional array of raster or gridded data, ‘stacks’ satellite images and addresses the problems faced by traditional remote sensing practices and provides an interactive environment where datasets can be analysed with relative ease as compared to its traditional counterparts. This framework allows multi-format and multi-projection datasets spanning decades to be used in various geoprocessing techniques from simple GIS tasks such as data conversion, time series generation, and to do more complex tasks such as change detection, NDVI generation, unsupervised classification and modelling. LISS III data for the state of Uttarakhand, India was used on an interactive interface called the Jupyter Notebook where scripts written in Python allowed data to be ingested, analysed and visualised. The Data Cube framework hence proved to be a flexible and extensive development environment which can be extended to meet more complex modelling requirements.

1. INTRODUCTION

Native and offline geoprocessing is hindered by few guiding principles that govern the functionality of large scale and complex datasets that requires to be processed and analysed before being accessed by users (Hofer, 2015). Traditionally, remote sensing products goes through many step-by-step procedures before being shipped out to a client, this often takes upto 80% of the total process, with less than 20% actually utilised for analysis and development (Oliver & Woodcock, 2015). Along with data interoperability and exchange, data size limitations and processing capabilities often pose a challenge to researchers bounded by using a more traditional or Desktop-approach to geoprocessing which is limited by not only hardware but also various complexities the user must attend to facilitate the data, its management and utilization in various softwares. Considering the vast amounts of Earth Observational (EO) data generated per day, there exists a large potential for data to be unstructured and more importantly not conforming to international standards (Lewis et al., 2017). To overcome such issues and vastly improve the user accessibility and scalability of EO data, a more robust and powerful framework that adheres to various standards of interoperability and allows on-the-fly geoprocessing of large amounts of EO data is required. The Data Cube, is one such framework which works on the principle of “stacking” satellite imagery in a multi-dimensional array of gridded data which overcomes such challenges. A study by Mueller et al., (2016) consisting of over 100,000 satellite images and metadata were ortho-rectified, corrected to measurements of surface reflectance and analysed for observations of water at a resolution of 25m. A project of this scale would not be possible if traditional methods of remote sensing were applied. This framework can be extended to fit various use-cases ranging from continental-scale analysis of vegetation change, species distribution modelling and understanding climatic change over long periods of time.

2. RELATED STUDIES

2.1 Online Geoprocessing technology and its attempts worldwide

One of the most profound examples of online, scalable and interoperable geoprocessing platforms belongs to that of Google. The Google Earth Engine (GEE) is a cloud-based platform for planetary scale analysis and is built upon its powerful supercomputational capabilities of petabyte scale analysis of EO data (Gorelick et al., 2017). Housing massive data catalogues which are indexed by high-performance parallel computers, GEE allows users to quickly access and analyse data from a web-browser. This technology allows a researcher to skip various hurdles of handling EO data, such as file formats, managing databases and using geospatial data processing techniques. File-based data handling mechanisms such as Hadoop Distributed File System (HDFS) and GeoTrellis works on the principle of processing large spatial queries using distributed memory abstraction techniques which enables the collection of elements in parallel (Appel, Lahn, Buytaert, & Pebesma, 2018). An alternative method is to represent EO data as multidimensional arrays and utilise such databases for not only storage but also analyses. The EarthServer project utilises such technology for use in domain of image and sensor statistics, neuro science, OLAP and high-level computing. The RasDaMan Engine is efficient in managing potentially unlimited data volumes and adheres to OGC data and service standards for interoperability. This ensures that big EO data can be handled and analysed in a cost-efficient and scalable manner (Baumann, 1999; “EarthServer.eu,” 2018).

This paper is based on the work carried out by the Australian Geoscience Data Cube (AGDC) project from which the multi-dimensional framework is adapted. AGDC has addressed

various challenges faced by using large EO data, in particular focusing on the V’s of Big Data. (Lewis et al., 2017) Volume, Veracity, Velocity and Variety has been successfully addressed using the Data Cube framework upon which multiple studies have been carried out. Projects such as the Water Observation from Space (WOfS) (Mueller et al., 2016), Fractional Cover (Scarth, P., Röder, A., Schmidt, 2010), Normalised Difference Vegetation Index (NDVI) , Intertidal Extents Model (ITEM) (Sagar, Roberts, Bala, & Lymburner, 2017) and Surface Reflectance (SR) were carried out using the Landsat archive of Australia. The Data Cube framework was built, coupled with the computing facility of the National Computational Infrastructure (NCI) where petabyte-scale level EO data was orthocorrected, atmospherically corrected and analysed successfully.

3. STUDY AREA

The area of study in this research is the State of Uttarakhand, India which is located at the foothills of the Himalayas and often referred to as *“Devbhumi”- Land of the Gods.* The state is roughly 54,000 Km^2 with an elevation range of 600 to 7800 meters. It is an agricultural and tourism dependent state where the role geospatial data is crucial. The terrain and climate is very diverse with regions near the Himalayas experiencing heavy snowfall while the plains are dense with populated cities often experiencing heavy rainfall. The study area is shown in Figure 1.

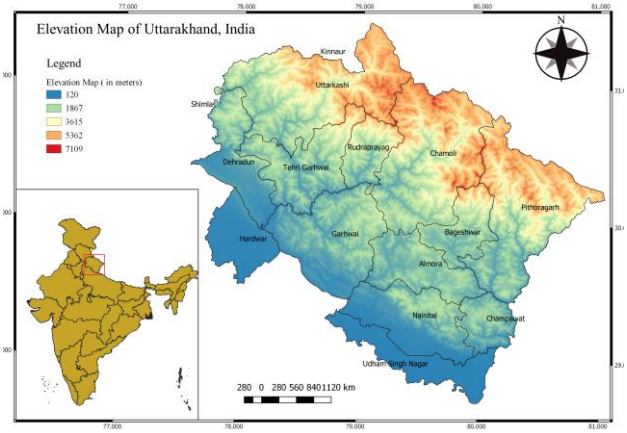

Figure 1. Study Area

4. MATERIALS AND METHODS

4.1 Materials Required

4.1.1 Hardware Requirements: The Data Cube framework was setup on a workstation PC with the latest Intel i7 6th generation processor coupled with 16 Gigabytes of RAM

4.1.2 Software Requirements: The following are the software requirements for the Data Cube framework -

- Python 3.5 and supported libraries: These are used to build the model and run analysis on all datasets, communicating with the Data Cube.
- PGadmin: Used to monitor and query data.
- Miniconda/Anaconda: Python environment to build model and setup Data Cube.
- QGIS/ArcMap: Pre-process datasets for modelling.
- HTML, PHP, JavaScript: To build front-end interface.
- OS-Windows 10 Pro and Ubuntu 14.0 LS

4.1.3 Datasets used: The datasets (as shown in Table 1) used to carry out this research was retrieved from the Indian Remote Sensing satellite (IRS) series and the Resourcesat Series, aboard which was the Linear Imaging Self Scanning (LISS) sensor. The sensor operates at a resolution of 23.5 meters with a swath of 142 kilometres. The sensor was classified into 4 optical multispectral bands covering green, red, near infrared and shortwave infrared. Pre-monsoon months between January and April were collected with almost 250 tiles covering the entire study area.

Table 1 EO Satellite Datasets and Size

Sl.No	Satellite Name	Product	Product Type	Data Size
1	IRS 1D	LISS III	Optical MSS	80 GB
2	IRS 1D	LISS III	Optical MSS	
3	Resourcesat 1	LISS III	Optical MSS	
4	Resourcesat 2	LISS III	Optical MSS	

4.2 Data Cube Setup and Execution:

The overall workflow as shown in Figure 2, from the installation of the Data Cube to execution of models is described in the following steps -

1. DC Package Installation
2. Database Initialization
3. Product Definition
4. Metadata Preparation
5. Data Indexing
6. Data Ingestion
7. Data Loading

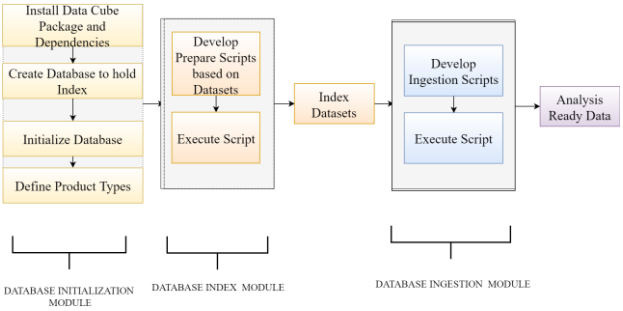

Figure 2. Data Cube Setup

4.2.1 Data Cube Package Installation: The latest version of the Data Cube package can be sourced from Git and installed along with all its dependencies a few of which are listed below -

- GDAL
- Rasterio
- Numpy
- netCDF4
- scipy
- pandas
- Matplotlib
- lxml etc.

This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-IV-5-29-2018 | © Authors 2018. CC BY 4.0 License.

Installation of Data Cube and all its supporting dependencies should be carried out on a virtual environment, so as to avoid version and library conflicts with other packages within the test system.

4.2.2 Database Initialization: A PostgreSQL database is initialised with super user permission and a schema is generated to hold all the table values. The *agdc* schema consists of 5 tables, namely -

- dataset
- dataset_location
- dataset_source
- dataset_type
- metadata_type, as well as a login table to maintain user records.

4.2.3 Product Definition: The Data Cube can handle many different types of data and for this very exact reason it is essential that the Data Cube understands the differences and nuances of each dataset and what to do with them. The product definition describes numerous variables similar to the ingestion configuration discussed before but is unique to each satellite data product. A few of the variables are listed below -

- Name
- Description
- Metadata
  - Platform
  - Instrument
  - Processing Level
  - Product Type
  - Format
- Measurements
  - Datatype
  - Nodata
  - Spectral Response

4.2.4 Metadata Preparation: A prerequisite for data index and ingestion is the meta-data preparation phase. A metadata file usually in the format of a basic text file or XML file is often accompanied with the satellite data, but this format is not readable within the Data Cube. A specific format called a Markup Language, more precisely a YAML formatted text file needs to be generated from the data in order for the Data Cube to index a dataset. The dataset meta-data generated consists of the following description variables similar to the product definition. A few of the variables are listed below -

- Unique ID
- Creation Date
- Product Type
- Platform
- Instrument
- Format
- Extent
  - Coordinates
  - From Date
  - To Date
- Grid Spatial
  - Geo Reference Points
  - Spatial Reference

This configuration file is unique to each scene of the product as the extents vary across multiple tiles. Thus there could be tens if not hundreds of dataset documents required to map each scene perfectly.

4.2.5 Database Indexing: Indexing of a dataset into a database is the process of setting and recording an instance the data and its corresponding metadata into a temporary storage in the database. This method is carried out purely for the sake of improving the speed of data access, especially when dealing with large scale datasets of Gigabyte if not Petabyte scales. Also to ensure that any changes during data manipulation and analysis does not alter the original dataset stored in the system (Lewis et al., 2017).

The Entity Relationship Diagram (ERD) for the database indexing phase is described below in Figure 3.

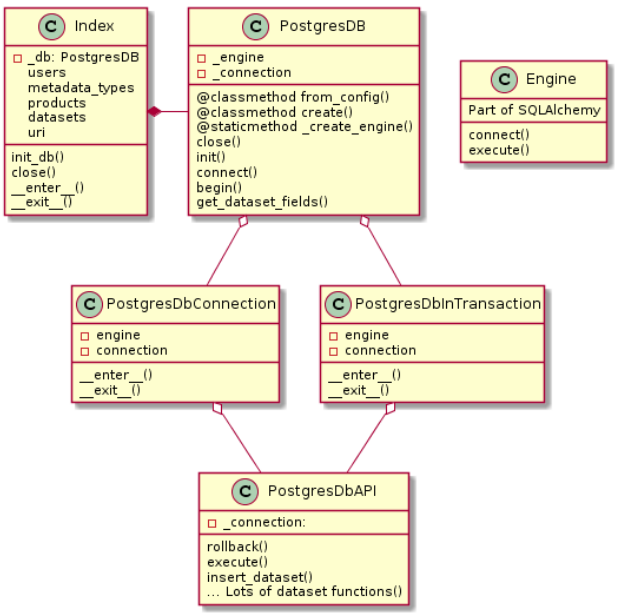

Figure 3. Database Indexing

The class PostgresDB contains the Index class which stores the reference to all users, datasets, metadata_types, products, datasets and URI and is defined by functions to initialise the database, to close the database connection as well as to enter and exit the initialisation modules. Part of the PostgresDB class is the Python specific SQLAlchemy, a database toolkit. This module is an object based relational mapper with the benefits and flexibility of SQL at complete high performance access rates and data abstraction. SQLAlchemy is the crux of the Engine class which establishes database connection as well as enables the execution of various methods of the XArray Dataset methods to carry out analysis. All changes in these database objects are only reflected in the index class and not in the stored data.

4.2.6 Data Ingestion: The process of inserting data into the Data Cube by mapping the dataset from its original form to a new storage schema is called as ingesting a dataset. The process is governed by many variables which dictate the meta-data and storage format before being written out to disk. The ingestion configuration file describes the following variables written in YAML formatted text.

- Location
- Storage driver
- CRS
- Tile_size
- Resolution

- Chunking
- Measurements

The initial step consists of ensuring the file names to be ingested matches the files described in the ingestion configuration file. The configuration files are verified after supplying root access, after which the indexed data is verified for each scene and product indexed into the Data Cube. The next step involves the testing of the grids, spatial reference and if it matches to the reference set by the ingestion configuration. By default, the Data Cube API transforms all datasets to a standard spatial reference set in the initialization phase, this ensures that none of the datasets will be mismatched and spatially unrecognized. Compliance Phase comes to an end where the default storage unit is set and a NetCDF check is accompanied for each file to maintain the common data format standard followed by the Data Cube API. NetCDF files are interoperable over the internet along with ability to store almost limitless number of dimensions and groups. Zlib data compression standard allows such complex data to be shared across the internet at a low-cost disk/bandwidth space (“netCDF4 API documentation,” 2018). Figure 4 describes the ingestion process in detail.

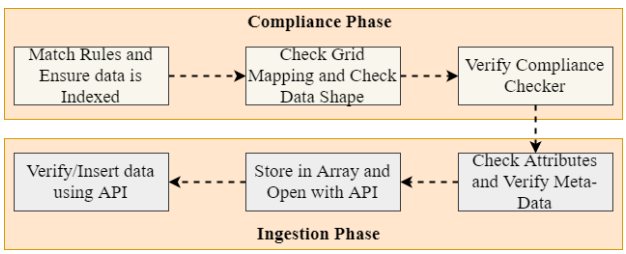

Figure 4. Data Ingestion

The ingestion phase begins with the reading of the configuration file, each attribute is read and verified to exist in the index before continuing. Now the NetCDF file metadata is cross-checked with the attributes found in the configuration file and against the original metadata (satellite order). When all the attributes, filenames and meta-data are found to be correct, the Data Cube API sets the input types, extents, time slices and dimensions along with the spatial reference onto the example NetCDF file (which is also indexed).

The final stage of ingestion is the Reprojection of each scene/tile into scaled down pixel values using the Data Cube API. This step concludes the ingestion of a scene, which is repeated for all scenes and products indexed in the Data Cube. The ingestion process ends when all the scenes have been reprojected as per configuration and each NetCDF file is written onto the disks.

4.2.7 Data Load: Loading of data once ingestion is complete is the process of query the database for the required dataset and its matching product for the indexed XArray Dataset storage format. The flowchart depicted in Figure 5 describes the process of data loading depending on the query supplied by the user.

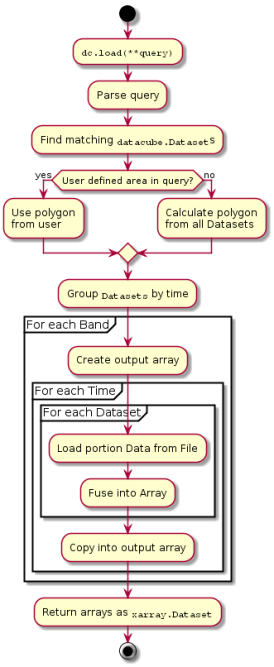

Figure 5. Data Load

Once a query is read and the polygon is calculated, all the datasets within the time-frame supplied is grouped. An output array for all of the bands is created at every time-stamp. The same process is repeated at each dataset encounter, after which the data portion is loaded from file and fused into array. Fuse operation is carried out when over-lapping regions exist in the polygons. Once all the datasets have been queried,read and fused, an output array is created to store all the datasets grouped by time and is returned as a XArray Dataset to the user.This can now be analysed and visualised using supported libraries such as Pandas and Matplotlib for various use-cases and algorithms.

5. RESULTS

This chapter describes the various results obtained after the implementation of the Data Cube and running various analysis on the environment. These operations were carried out on an interactive platform called the Jupyter Notebook which is a browser based interpreter and visualizer of live codes, equations and visualizations. The Data Cube can be remotely accessed via a web browser (Jupyter Notebook) and algorithms can be developed across multiple datasets without facing any complications related to data interoperability or compatibility. The results achieved during the course of this project have been described below depicting the following:

- Satellite Data Ingestion
- Accessing the Data Cube
- Viewing products and measurements
- Retrieving Data
- Plotting Data
- Masking
- NDVI

- Band Statistics
- Reprojection
- Change Detection
- Unsupervised classification

5.1 Ingestion of Satellite Data

As described in section 4.2.7, various scripts were written to customise different satellite data along with user requirements and definitions. Data from the Linear Imaging Self Scanner (LISS) III satellite data was acquired for the study area from 2000 to 2015, consuming over 120 gigabytes and generating over 8,000 NetCDF files used by the Data Cube index operations. The scenes were indexed and ingested successfully on the hardware as described in section 4.1.1.

5.2 Accessing the Data Cube

A database configuration file holds the key information required to connect to the Data Cube such as the database name, hostname or IP address of database, username and password authentic to each user. This ensures that only users with the right credentials who are authorised to access the Data Cube are allowed to query results out of it. A code snippet below shows how the datacube library is imported and the configuration file is accessed via Jupyter Notebook.

```
import datacube
dc=datacube.Datacube(config='path_to_file')
```

5.3 Viewing Products and Measurements

Each satellite data comes with a descriptive metadata information consisting of the imagery extents, format, platform, instrument etc. which can be viewed by the user with Jupyter Notebook. This is accessed by the Data Cube API which reads the metadata of each tile/scene and summarises band measurements and product descriptions.

```
import datacube
dc.list_measurements()
dc.list_products()
```

5.4 Retrieving Data

The Data Cube API called to load specific product types ingested along with the area of interest described by its extents. Along with the AOI, the resolution of the required tiles can also be specified, if resampling to a lower resolution is required the sampling algorithm along with its resolution in meters can be denoted within the code itself.

```
Syntax
Variable=dc.load(product='product_name',
resolution=(-x,y),x=(x1,x2),y=(y1,y2))

Code
WC=dc.load(product=LISSIII_Ingest,resolutio
n=(-
24,24),x=(77.500,81.083),y=(28.666,31.499))
```

5.5 Plotting Data

Figure 6 and 7 represents the single plots of the LISS III band 3 and band 4 imagery retrieved from the Data Cube. All the scenes pertaining to the study area was mosaicked using Python and its dependencies without relying on consumer grade softwares.

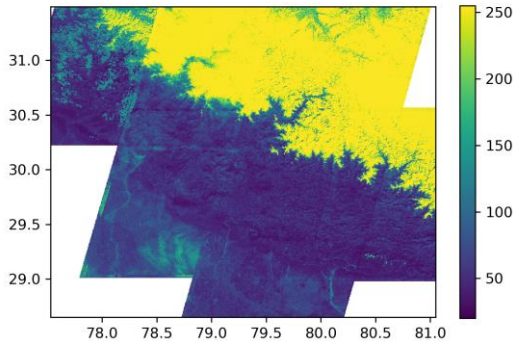

Figure 6. LISS III band 3

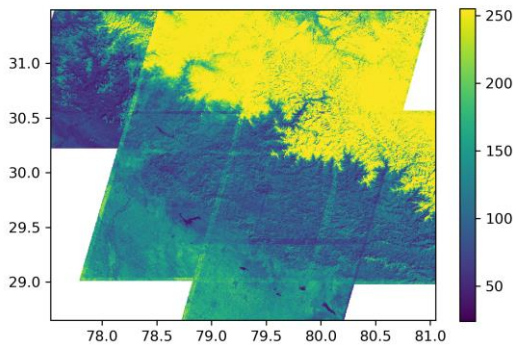

Figure 7. LISS III band 4

5.6 Masking NO_DATA

NO_DATA/Nan values in rasters often occur due to many reasons such as incomplete data retrieval and scene boundaries, these are often filled with a special value and can be filtered out using the Data Cube API, where the XArray data structure reads the typical value of -9999 as a floating type value which enables plotting to be easily visualised. Shown in Figure 8 are a few tiles of LISS III imagery from 2015 which have been masked and NO_DATA has been removed from scene boundaries.

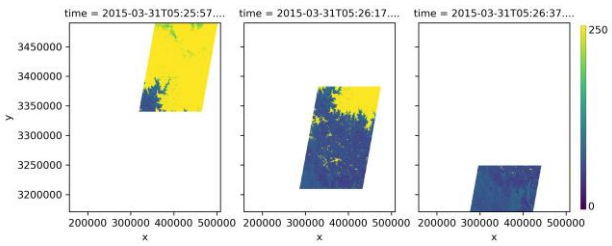

Figure 8. NO_DATA Masking

```
Syntax
Variable1=DataArray.Band.where
(DataArray.Band!=
DataArray.Band.atrs['nodata'])
```

```
DataArray.Band.plot(col='time',col_wrap=x)

Code
L3_ND_B2 = L3_2005_B2.where(L3_2005_B2 != L3_2005_B2.attrs['nodata'])
L3_ND_B2.plot(col='time', col_wrap=3)
```

5.7 NDVI Generation

The Normalised Difference Vegetation Index was calculated using the Red and NIR bands of LISS III imagery, the Data Cube API was used to call the product and the corresponding tiles within the AOI. After NO_DATA was masked, NDVI was calculated for each tile. After NDVI was calculated, the tiles were mosaicked with the mean NDVI taken into consideration.

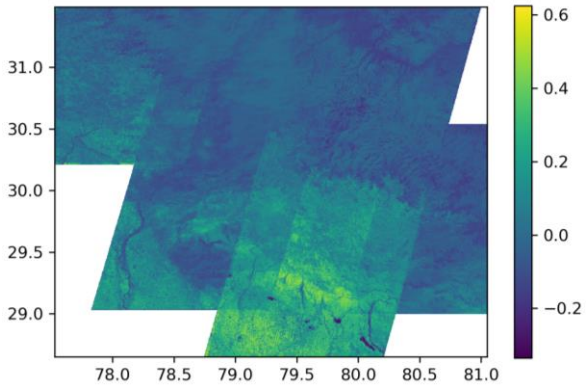

Figure 9. NDVI of year 2010

Figure 9 describes the NDVI generated for the year 2010 on the Data Cube framework. The NDVI tiles were mosaicked by considering the mean NDVI of all the tiles pertaining to the study area. This operation was carried out in under 3 minutes, operating at 70% CPU and 65% Memory utilization rates. Such tasks are hardware dependent, therefore the better the hardware configuration, faster the processing. Similarly Figure 10 depicts the reclassified NDVI values to fall into 4 distinct classes, where -1 to 0 for class 1, 0 to 0.1 for class 2, 0.1 to 0.25 for class 3 and above 0.25 for class 4 this is done to better visualize the areas of low and high/dense vegetation.

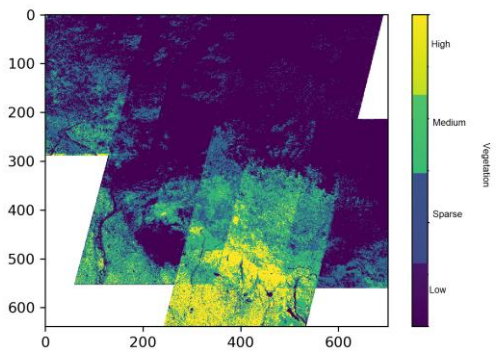

Figure 10. Reclassified NDVI of 2010

5.8 Band Statistics

Various operations such as False Colour Composite (FCC) generation, Mean, Median and histogram generation was carried out to test the efficacy of the Data Cube framework to handle the most common GIS tasks often carried out on traditional softwares. Once data was ingested into the Data Cube, the user need to worry about facing issues related to projection mismatches and the various technicality required to operate a software. The Jupyter Notebook interface allows a user to implement customised codes and algorithms to work based on the requirements of the user. This allows a use to implement the same logic on all types of datasets ingested into the Data Cube. Therefore, enabling *create once, use many* policy of operations. This not only saves the effort required in repeating tasks on multiple datasets but also allows the use to concentrate on the analysis and improving the algorithm without worrying about handling large datasets. Shown in Figure 11 is the FCC generated using the *Raster Stack* operation.

```
Syntax
Variable2=np.array([mosaic_band1,mosaic_band2,mosaic_band3],dtype=np.datatype)
Plt.imshow(np.dstack(variable1)

Code
all_bands= np.array([all_b4,all_b3,all_b2], dtype=np.int64)
plt.imshow(np.dstack(all_bands))
```

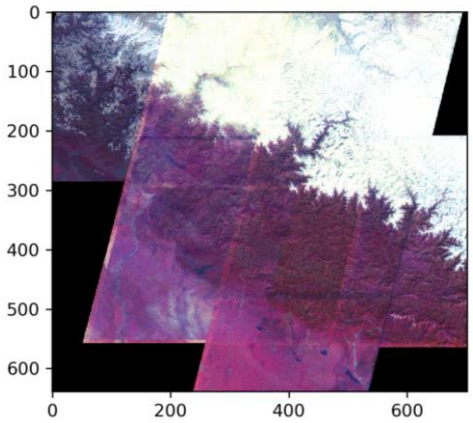

Figure 11. FCC generation of Uttarakhand

```
Syntax
Variable2=np.nanmedian(DataArray_Band,axis=x)
Plt.imshow(Variable2,extent=(x1,x2,y1,y2)

Code
import numpy as np
import matplotlib.pyplot as plt
all_b4_10=np.nanmedian(L3_ND_B4_10,axis=0)
plt.imshow(all_b4_10,extent=(77.53,81.05,28.65,31.49))
plt.colorbar()
plt.savefig('path_to_disk/name.png',dpi=300)
```

Mean and Median operations can also be carried out on the datasets ingested into the Data Cube. These allow multiple tiles of imagery within the extent to be mosaicked.

The Data Cube API along with the use of the Python Numpy library allows very quick and efficient raster band operations to be carried out on the multi-dimensional database. These are can now be save to disk directly. Shown in Figures 12 and 13 are the mosaicked median and mean outputs of the study area.

This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-IV-5-29-2018 | © Authors 2018. CC BY 4.0 License.

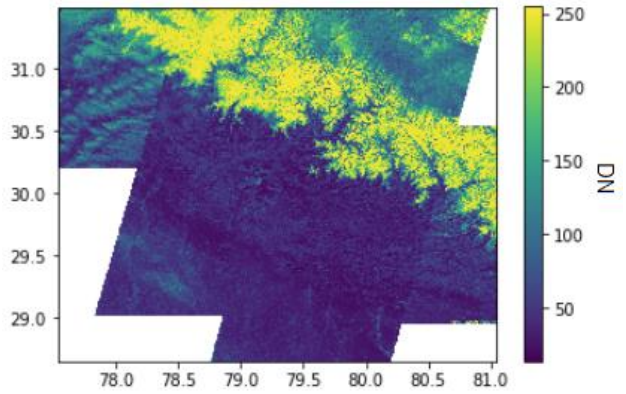

Figure 12. Median operation on mosaicked image

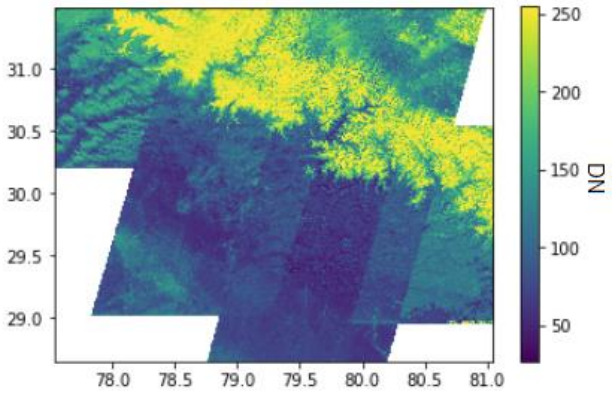

Figure 13. Mean operation on Mosaicked image

```
Syntax
Variable3=DataArray_Band.loc['year 1':'year
2']

Code
L31_2005_B4 = L31.B4.loc['2005':'2015']
L31_2005_B4.plot()
```

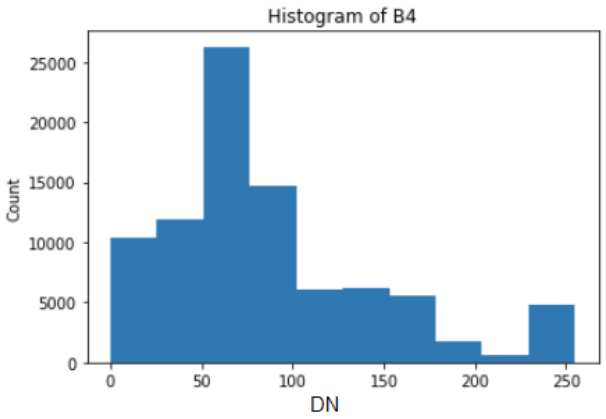

Figure 14. Histogram Generation (Band)

Histograms can also be generated as shown in Figures 14 and 15 depicting the pixel counts of band 4 and NDVI values respectively. These operations use the *matplotlib* library to read and plot the datasets operations.

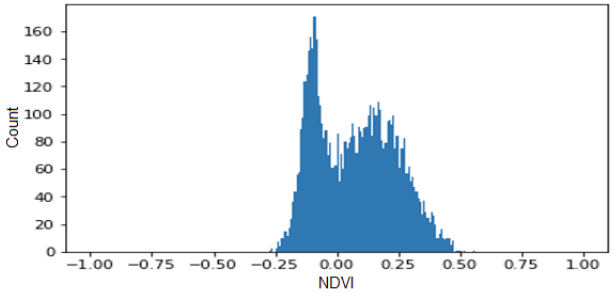

Figure 15. Histogram Generation (NDVI-2010)

5.9 Reprojection

Since the Data Cube relies heavily on the GDAL environment, all geoprocessing tasks including Reprojection, reading and writing raster as well as vector data, map algebra and classification can be carried out on the Data Cube. Shown below is an example code snippet to reproject raster data.

```
Syntax
Variable4=  dc.load(product='product_name',
resolution=(-x,y),x=(x1,x2),y=(y1,y2),
output_crs='EPSG CODE')

Code
Albers_new=dc.load(product='cartoDEM',
x=(77.500,81.083),y=(28.666,31.499),output_
crs='EPSG:3577',resolution=(-25,25))
albers_grid.elevation.shape
```

5.10 Change Detection

Using the generated NDVI values of the years 2005, 2010 and 2015 change detection was carried out to identify the migration of classes from highly vegetative regions to low and vice-versa as depicted in Figures 16 and 17.

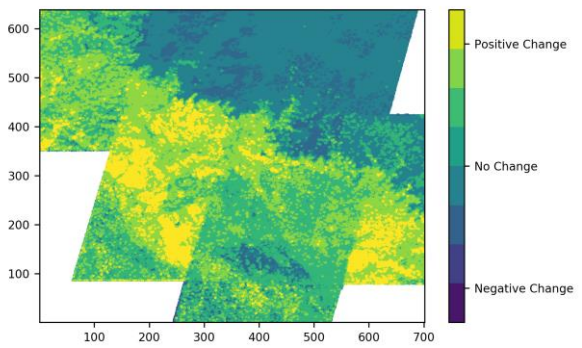

Figure 16. 2010-2005 Change Detection

```
from matplotlib import ticker
plt.contour(ndvi_diff,linestyles="solid",or
igin='upper')
plt.contourf(ndvi_diff,origin='upper')
cbar=plt.colorbar()
tick_locator = ticker.MaxNLocator(nbins=3)
cbar.locator = tick_locator
cbar.update_ticks()
cbar.ax.set_yticklabels(['Negative
Change','No Change','Positive Change'])
plt.rc('font', size=8)
plt.draw()
plt.show()
```

This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-IV-5-29-2018 | © Authors 2018. CC BY 4.0 License.

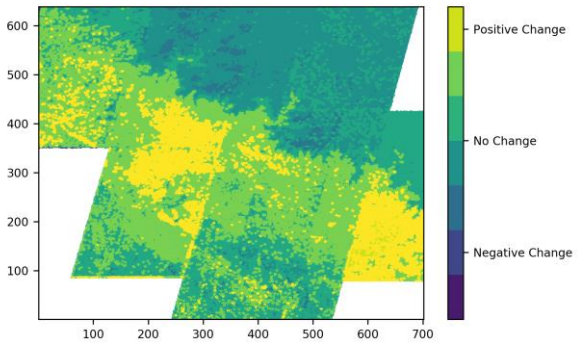

Figure 17. 2015-2005 Change Detection

5.11 Unsupervised Classification

More complex algorithms such as classification techniques can also be carried out on the Data Cube. Unsupervised classification of raster datasets was carried on the LISS III imagery with a cluster size of 5 using the MiniBatchKMeans algorithm specified by *sklearn* cluster library as shown in Figure 18.

```
lf_class2=sklearn.cluster.MiniBatchKMeans(init='k-means++',n_clusters=5,batch_size=10,n_init=10,max_no_improvement=10, verbose=0,random_state=0)
```

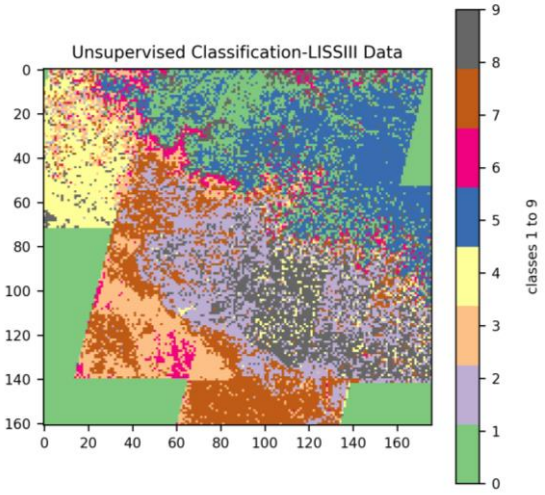

Figure 18. MiniBatchKMeans unsupervised classification

6. CONCLUSION

This study has shown that having a scalable and robust framework that can be adapted to multiple datasets without having to deal with data dependency issues, file format and projection issues. This framework paves the way to a more efficient use of large remotely sensed data which is growing immensely. Technology today has advanced to such a stage that data generation from EO satellites far outgrow the data utilization rate. As new frontiers in Machine Learning, Deep Learning, Computer Vision and Human Computer Interaction are crossed, there is a need for data to be ‘Analysis Ready’. This is imperative as data from the Sentinel and GiSAT series would further push our data handling and processing capabilities, which if not addressed properly would affect the quality of research and more importantly its impact on society. Addressing issues related to not only usability and efficiency, but also the challenges faced by Big Data is where a multi-dimensional framework of gridded data can overcome when compared to the slow and tiresome process that traditional remote sensing has been following for decades.

REFERENCES

Appel, M., Lahn, F., Buytaert, W., & Pebesma, E. (2018). Open and scalable analytics of large Earth observation datasets: From scenes to multidimensional arrays using SciDB and GDAL. *ISPRS Journal of Photogrammetry and Remote Sensing*, *138*, 47–56. https://doi.org/10.1016/j.isprsjprs.2018.01.014

Baumann, P. (1999). A Database Array Algebra for Spatio-Temporal Data and Beyond. In *NGIT ’99 Proceedings of the 4th International Workshop on Next Generation Information Technologies and Systems* (pp. 76–93). Berlin: Springer-Verlag Berlin, Heidelberg. https://doi.org/10.1007/3-540-48521-X_7

Gorelick, N., Hancher, M., Dixon, M., Ilyushchenko, S., Thau, D., & Moore, R. (2017). Google Earth Engine: Planetary-scale geospatial analysis for everyone. *Remote Sensing of Environment*, *202*, 18–27. https://doi.org/10.1016/j.rse.2017.06.031

Hofer, B. (2015). Uses of online geoprocessing technology in analyses and case studies: a systematic analysis of literature. *International Journal of Digital Earth*, *8*(11), 901–917. https://doi.org/10.1080/17538947.2014.962632

Home | EarthServer.eu. (2018). Retrieved March 21, 2018, from http://www.earthserver.eu/

Lewis, A., Oliver, S., Lymburner, L., Evans, B., Wyborn, L., Mueller, N., … Wang, L.-W. (2017). The Australian Geoscience Data Cube — Foundations and lessons learned. *Remote Sensing of Environment*, *202*, 276–292. https://doi.org/10.1016/j.rse.2017.03.015

Mueller, N., Lewis, A., Roberts, D., Ring, S., Melrose, R., Sixsmith, J., … Ip, A. (2016). Water observations from space: Mapping surface water from 25years of Landsat imagery across Australia. *Remote Sensing of Environment*, *174*, 341–352. https://doi.org/10.1016/j.rse.2015.11.003

netCDF4 API documentation. (2018). Retrieved May 10, 2018, from http://unidata.github.io/netcdf4-python/

Oliver, S., & Woodcock, R. (2015). *Australian Geoscience Data Cube Agency Report*. Melbourne. Retrieved from http://ceos.org/meetings/wgiss-40/

Sagar, S., Roberts, D., Bala, B., & Lymburner, L. (2017). Extracting the intertidal extent and topography of the Australian coastline from a 28 year time series of Landsat observations. *Remote Sensing of Environment*, *195*, 153–169. https://doi.org/10.1016/j.rse.2017.04.009

Scarth, P., Röder, A., Schmidt, M. (2010). Tracking grazing pressure and climate interaction - the role of Landsat fractional cover in time series analysis. In *15th Australasian Remote Sensing and Photogrammetry Conference (ARSPC)* (pp. 13–17), Australia https://doi.org/https://doi.org/10.6084/m9.figshare.94250.v1