Image Classification Using Machine Learning Algorithms in Google Earth Engine Environment

Paul TEDORESCU, Simona-Nicoleta VOICU
National Institute for Research & Development in Informatics - ICI Bucharest
paul.teodorescu@ici.ro, simona.voicu@ici.ro

In this paper it is proposed to classify an image using a Machine Learning approach inside the Google Earth Engine platform and Jupyter Notebook: feeding the computer with training data (in our case being points/pixels having a label which represent the land-cover type), it will learn to recognize the type of pixel through a model built on the technique called supervised learning. Even if the reader is not so familiar with the GEE environment and with GEE data structures and data types (like image, image Collections, features, feature Collections, geometry etc.), we'll try to guide him step by step in this modern exercise of building a machine learning model which, having the intelligence to guess the land-cover for each pixel, it will finally create a thematic map, very useful for scientist and specialists. This is actually what it's called Artificial Intelligence and the model built here can be re-used with new data, new images.
Keywords: *Machine Learning, Google Earth Engine, Image classification, Jupyter Notebook, Python*
DOI: 10.24818/issn14531305/25.3.2021.01

1 Introduction

Information on the type of land covering a geographical area plays a vital role in many aspects of life, from science to economics and from economics to politics. Accurate and timely information on the type of land cover is in high demand.

There are two ways to detect land cover types from a satellite image: we can interpret the land cover visually (for example green color as vegetation, brown as soil or blue as water) or we can apply Artificial Intelligence techniques to geospatial data (Machine Learning) to classify the land cover automatically. Classification in Machine Learning and statistics is a learning approach: the computer learns from the incoming data and makes predictions with new data. In image classification, the model built by research teams in their geospatial data processing work, will learn to identify land cover type for each pixel, helping them to create thematic maps and to discover changes over time for a specific geographical area.

Google has assembled a huge amount of Earth observation data from satellites like Landsat 8, Landsat 7, Landsat 5, Sentinel, MODIS, SRTM and made them available in the cloud through Google Earth Engine (GEE). More than that, with his API's and planetary analysis functions, GEE is a powerful tool to classify images. In this article we'll show how to classify an image (a geographic zone from Romania) using Google Earth Engine, it's satellite maps and Machine Learning algorithms to classify and detect „cover-land" information (the terrain type of a surface chosen by us) and also using Jupyter Notebook with Python programming language. With this material, in which machine learning techniques have been used, it is hoped that researchers will be able to advance more easily in their work of processing satellite data.

2 Materials and Methods

Google Earth Engine allows the analysis and visualization of geospatial data. Geospatial (spatial) data are data about objects (people, roads) or events (earthquakes, floods), which can be located geographically by coordinates. There are two types of spatial data classified as follows:

- vector data (points, lines, polygons are used and represent cities, roads, buildings). They are used to store the outline of objects;

- raster data: stores the contents of objects. For example: photos taken from an airplane or satellite. These dates are files that store information in arrays of pixels; each pixel contains information about a city, a place, a forest, etc.). Raster images can be found in the form: TIFF, JPEG, GIF, PNG. The platform is efficient in storing images captured by satellites. These satellites are: MODIS (moderate resolution imaging spectroradiometer), LANDSAT and SENTINEL.

The images captured with the help of satellites are useful in forest analysis, deforestation tracking, water-covered areas, land use change, land cover, land health assessment, etc. Moderate Resolution Imaging Spectroradiometer (MODIS) is an instrument equipped with sensors that acquire images since 1999, daily images, surface reflectance adjusted BRDF for 16 days, reflection factor, is the proportion of light reflected on the surface a material, by-products such as indications of vegetation or snow cover [1]. Google Earth Engine organizes geospatial information to be universally accessible and useful, and makes it available for analysis. It is worth mentioning that data and images can be imported from third parties into the Earth Engine for analysis. Any analysis performed in the Earth Engine can be used by third party tools. The catalog with available data sets includes:

- Landsat catalog (USGS / NASA);
- MODIS data sets;
- Sentinel-1 date;
- precipitation data;
- sea surface temperature data;
- data on climate, altitude and altitude.

Users can upload their own data to Earth Engine for analysis (raster data or vector data: GeoTIFF or shape files). When we talk about satellites, we refer to Remote Sensing images (collecting information about objects at a distance, without direct contact, using sensors). There are two ways to extract information from satellite images of the terrain:

- Maps are made with variables such as biomass, LAI (Leaf Area Index), tree canopy (these variables change depending on the season, e.g.: LAI will be lower in winter than in summer);
- Maps are made with variables such as land cover, burned areas, floods, forests and a distinction is made between agriculture, forest, water areas, and this is called IMAGE CLASSIFICATION or THEMATIC REMOTE SENSING (thematic RS).

Image classification is the most widely used type of ML and remote sensing. Image classification is an automatic approach to classifying raster (satellite) and vector images. For GIS and remote sensing systems, supervised and unsupervised image classification is used [2].

In the land cover classification studies of the last decade, results have been obtained with better accuracy when satellite images from a time series have been used, series of images from the same area taken at different times (time series satellite images) versus using images taken at one time. Recently, the availability of Google Earth Engine (GEE), which is a cloud-based computing platform, has gained attention for remote sensing-based applications: acquisition of information about a phenomenon or object without physical contact with that object. Temporal aggregation methods derived from time series images are widely applied. For example, using „mean" or „median" values from a time-series of images (temporal series) is better than using only one image from that time series. As will be shown below, in GEE, many techniques simply use and select as many images as possible, regardless of how many of these images (annual, monthly, or seasonal) could affect the accuracy / precision of the classification. In the supervised classification, it starts with a set of training data, which are actually points on the map (with their spectral values) at which it is known what type of terrain it represents [3]. Romania (and regions in Romania) will be chosen as the study area. As for data, it will be used LANDSAT satellites data and MODIS satellite data. LANDSAT satellites have the optimal ground resolution and spectral bands to efficiently track land use and to document land change due to climate change, urbanization, drought, wildfire, biomass changes.

MODIS satellites (Moderate Resolution Imaging Spectroradiometer) provides complete daily coverage of the earth. The purpose of GEE is to classify the pixels of an image (process called "image classification"), meaning to identify and portray the type of land cover on the ground. This research will explain how to build a predictive model which will guess the type of land as geographic information, such as water, urban area, forest on a geographical region chosen by the user from a Google map. The technique used is Machine Learning. Image classification is perhaps the most important part of digital image analysis. In order to better deepen the research topic, we tested the generation of maps on various areas in Romania (Bucharest or the Carpathian Mountains area) using the Jupyter Notebook platform that uses the Python language. Jupyter Notebook is a platform similar to Google Earth Engine, which uses Python as a programming language compared to JavaScript. This has allowed the documentation, data visualization and storage to be much easier to use. Jupyter Notebook allows data cleaning, statistical modeling, ML model training and data visualization. Jupyter allows users to view code results online without dependence on other parts of the code. In the notebook, each cell of the code can be checked at any time to draw an output. Because of this, Jupyter helps print the output line, which becomes extremely useful for the exploratory information analysis (EDA) process. The Figure 1 summarizes the 3 components of this research: a huge catalog of satellite images and geospatial data sets (owned by Google), a Machine Learning algorithm to guess or predict the "cover land" information and the final application written in JavaScript/Phyton inside the code editor provided by Google Earth Engine/Jupyter Notebook.

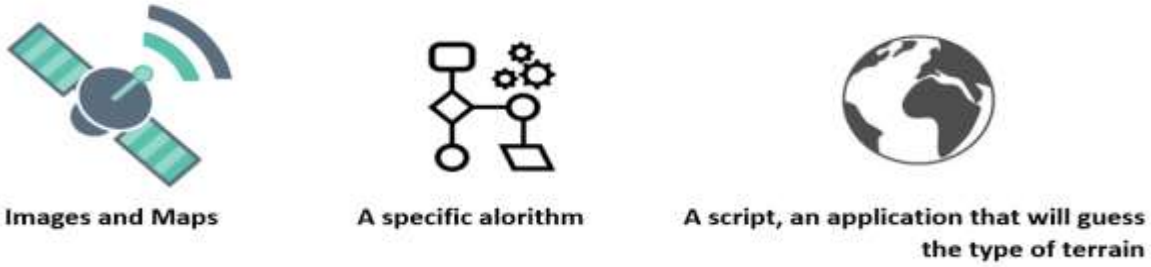

Fig. 1. Components of research

The goal is to build a model, also called „a classifier". This model will learn to identify the type of terrain, only by feeding it with data already labeled. The data from which the model will learn contain an additional information that designates the type of terrain. In the "world" of artificial intelligence, it is said "data is labeled" meaning that the data is grouped into a specific category or class. The type of terrain is actually the label of that point on the map and represents the class to which it belongs. Labels are nothing more than consecutive integers. For example, the number zero can denote the type of land "water" and the number "1" can denote the type of land "vegetation" [4].

This model which was trained and was taught to recognize the label will be used with new data from the region to be classified and it will identify which class/category/type of terrain the new data will fall into. The model finally will provide new information, a new attribute for each pixel on the map: terrain type (the pixel class). Every class (or type of terrain) will have an associated color from the Annual International Geosphere-Biosphere Program (IGBP). Knowing that geospatial datasets contain one or more layers called bands, for training data we'll select data from the first MODIS band called LC-Type1 band. The reason why the first Modis band was chosen is because it contains that additional attribute

called "label" which is in fact the type of terrain or landcover type [5].

3 Google Earth Engine using JavaScript

The model which will be built in JavaScript, will be a classifier and it will be called as such: the Classifier. It contains in fact an algorithm, a Machine Learning algorithm which will categorize the type of terrain (where that pixel, from the Google map, resides). The Classifier will initially be trained on MODIS data (used as training data) in order to learn how to distinguish the type of terrain. After that we'll apply the Classifier on a Google Maps region chosen by the user, such as Banat, Transylvania or a hand-drawn polygon. To be sure it's working well, we'll take new data (called validation data) and we'll put the classifier back to work, we'll display the classified image (where each type of terrain is colored differently), we'll calculate the accuracy of the classification and we'll print the error matrix.
Inside GEE editor will be chosen Landsat 8 satellite images, with the mention that any other source image can be used, as can be seen in image. Writing Landsat 8 in the search box, GEE is listing all the Landsat 8 datasets. The *USGS Landsat 8 Collection 1 Tier 1 and Real-Time data TOA Reflectance* collection is chosen and GEE will generate some variable shown in the Figure 2. This variable can also be manually written using a special ID of this data set [6]. For the geographical area to be classified, either it will be chosen to manually draw a polygon (an area from Romania, case in which GEE it will generate a variable) or it will be added code to generate a variable with the coordinates of the region/polygon called Region Romania. This image collection is filtered by date (the summer of 2019), by region and another new filter for the clouds. We'll remove images with high-cloud cover using:

```
sort('CLOUD_COVER')
filterMetadata("CLOUD_COVER",
"less_than", 1)
```

Because images are taken at different times, atmospheric conditions can change spectral values. To reduce this effect, the first least cloudy image is chosen from the collection, using the *first ()* function:

```
   var landsat = ee.Image(ee.ImageCollec-
tion('LANDSAT/LC08/C01/T1_RT_TOA')
   .filterDate('2019-06-01', '2019-09-
30')
   .filterBounds(Region_Romania)
      .sort('CLOUD_COVER')
      .first());
```

After *this* filtering operation, to view the resulting image, the *addLayer* function is used (which adds an EE object as a map layer) having the following parameters:

- the resulting image after applying the filters;
- the Landsat spectral bands we decided to be displayed: we will choose only 3 bands B4, B3 and B2 (blue, green, red);
- the chosen name of this layer to be displayed: „*Landsat with 3 bands, RGB"*.

```
Map.addLayer(land-
sat,{bands:["B4","B3","B2"]},"Landsat
with 3 bands, RGB")
```

To calculate the cloudiness score it is necessary to clean the cloud images, using the command:

```
var cloudScore = ee.Algorithms.Land-
sat.simpleCloudScore (landsat).select
('cloud');
```

We will mask the clouds with a cloud index higher than 50. This score (which is in the range of [0.100]) is randomly selected at 50 (as a *cloudiness* index), but can have any value. Then it is a need to apply a reducer (a GEE technique) for the best possible masking in the cloud. The *Reduction* function (which acts on all the bands of an image and on all the pixels) will receive Landsat data as input and will aggregate pixels on all bands of an image. The reducer will choose a minimum of the spectral value for each pixel and for each band, passing through the desired bands. The result of these operations will generate the variable "input". Basically, this variable contains a desired image as clean as possible:

```
var input = landsat.updateMask(land-
sat.mask().re-
duce('min').and(cloudScore.lte(50)));
```

For training data (which is actually a collection of features, a *featureCollection*), one of the properties (or attribute) is the *class* label. Labels are actually consecutive integers. For example, the number zero can be assigned to the label „water", and the number one can be assigned to the label "vegetation". The training data is imported from a good source called MODIS. As it was done with Landsat data, a Modis data ID is also used and a data set from January 2019 is chosen, the year from which we also chose Landsat data. It is ideal to find out the specific day, month and year of the least cloudy map used for Landsat to be able to use the same day of the year for MODIS data. By using:

```
ee.Date(image.get('sys-
tem:time_start')).format('YYYY-MM-
dd').getInfo())
```

it was displayed the date of the least clouded Landsat map: "2019_08_14 ". So, that day it will be specified in our code:

```
var modis = ee.Im-
age("MODIS/006/MCD12Q1/2019_08_14").se-
lect('Land_Cover_Type_1');
```

From this data set, only the first band called "LC_Type1" or "Land-Cover_Type_1" is chosen because it offers the class or label. The chosen band "LC_Type1" has 17 classes, which will be used for classification, i.e. for recognizing the types of terrain after the classifier has learned to do so with training data. With the Modis data set (contained in the "modis" variable), the function *addBands* is used to add the additional band containing 17 labels. The GEE documentation helps the user to use libraries and functions, as can be seen in Figure 2.

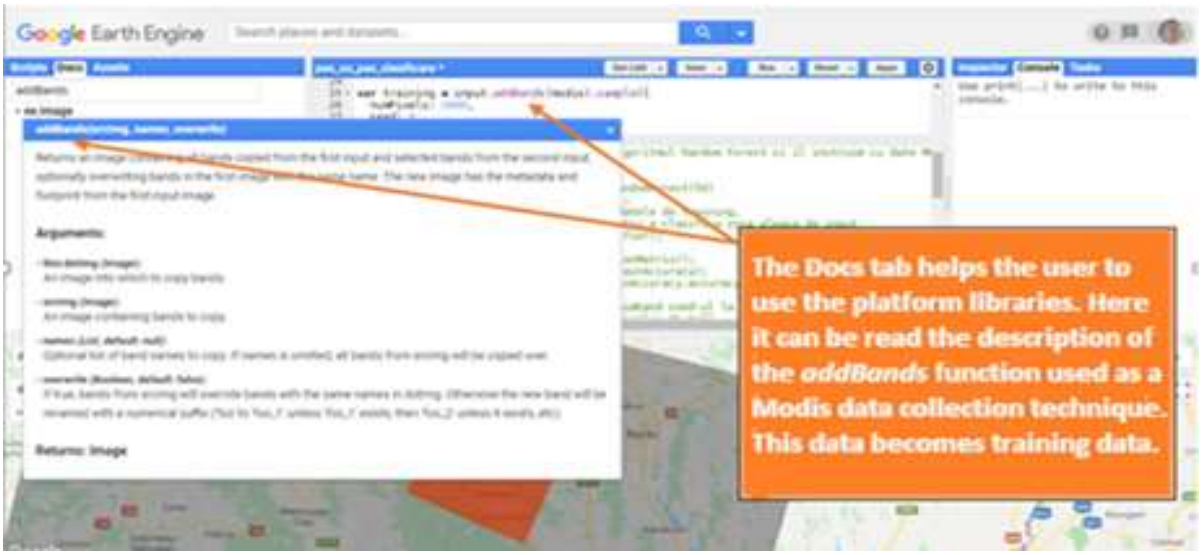

Fig. 2. GEE functions

Therefore, using the GEE platform, it ensures that, to all the information that is brought by the Landsat data (contained in the "input" variable), an additional information is added: the class or the type of terrain brought by the MODIS data set. A sample of 5,000 points of data is collected from the Landsat and MODIS image.

```
var training = input.addBands(modis).sam-
ple({  numPixels: 5000,     seed: 0});
```

After the printing operation on the GEE console, it can be seen how this operation supplemented the Landsat data information with another information (from MODIS) that helps the classifier to learn the class from where that pixel belongs.

It is worth mentioning that GEE has another technique for adding that information containing the type of terrain (class) from the MODIS data set: using *sampleRegions* function. Like *addBands* function, this one also converts each pixel into a specific GEE data structure

called feature and returns a collection of features (*featureCollections*). *Feature* - as a GEE data type - is an abstraction made by Google Earth to be able to succeed in data processing. *Features* are actually objects. They are GeoJason data (a special format for encoding geographic data structures, a format that is able to describe *Geometry* data types such as point, line, polygon) and can be seen as a list of properties among which, the most important one, is Geometry. These features can also be viewed as a row in a table/matrix where one of the columns is Geometry [7].
Returning to the sampleRegions operation: practically this technique overlaps points (an overlay process) from two images: one from Landsat and one from MODIS:

```
var training = input.select(bands).sam-
pleRegions({
  collections: modis,
 properties: ['Land_Cover_Type_1'],
scale: 30});
```

Note that the *feature collection* is specified and also the property containing the class: the name of the Modis band which contains all the classes called "Land_Cover_Type_1".
The training data sample is now ready. With this data, the model (the classifier) is trained and therefore „taught" to recognize the type of terrain for each pixel. The user can choose any of classifiers presented by GEE (each having its own classification algorithm), as seen in Figure 3.

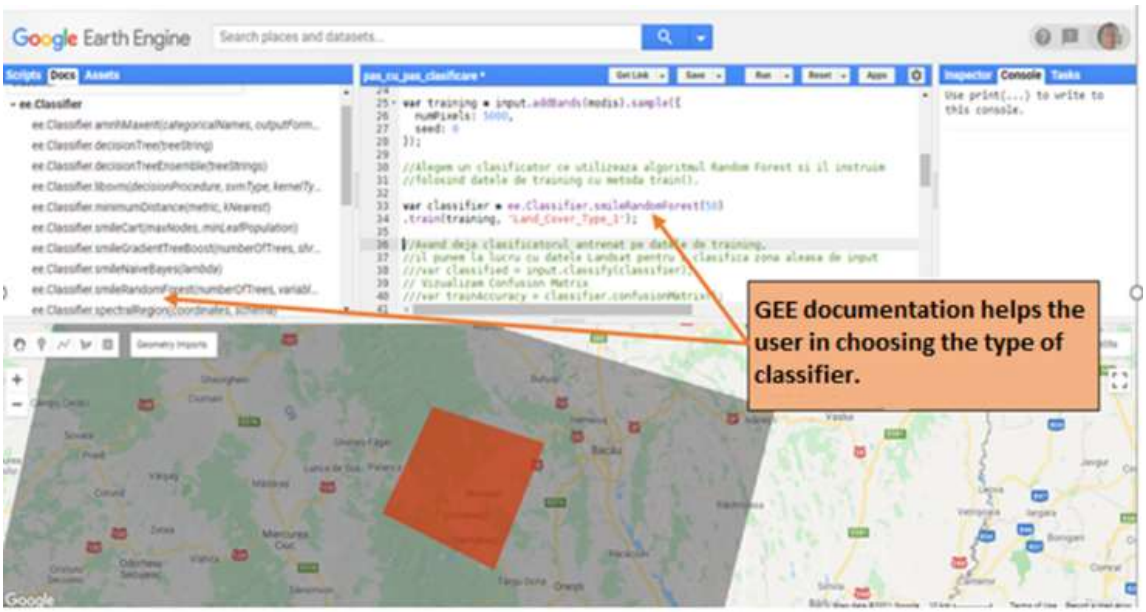

Fig. 3. The type of classifier: *smileRandomforest*

By selecting the classifier that works with the *RandomForest* algorithm, with 50 trees, we train the classifier with the training data:

```
var classifier = ee.Classifier.smileRan-
domForest (50) .train (training,
'Land_Cover_Type_1');
```

The result obtained (after the training and learning operation of the classifier) will be contained in the *classifier* variable. This is also the name of the classifier ready to classify any other data, dots or pixels. The model is fed with new data of the image desired to be classified, contained in the variable "input":

```
var classified = input.classify (classi-
fier);
```

This line of code actually classifies the desired image, using a trained classifier with 5,000 MODIS data.To verify the accuracy of the classifier is necessary to see the Confusion Matrix, also known as "the error matrix". This matrix reflects the difference between reality (on the ground) and the predictions made by the classifier.

```
var trainAccuracy = classifier.confusion-
Matrix();
```

A new data collection will be used, as was explained at the beginning of this research. With this new data set, the classifier should be relaunched. In Machine Learning, this is called „the validation operation". Most of the time, it is a good practice to use new sets of data, called „test data" and/or „validation data". Actually, is better to use both sets of data, test data and validation data. In this case it was used only validation data. The new data is randomly generated by changing the seed to „1" in the sampling operation and then filter to remove null pixels, using the B1 Coastal Aerosol band:

```
var validation = input.addBands(modis).sample({
numPixels: 5000,
seed: 1
 })
.filter (ee.Filter.neq('B1',null));
```

The validation data is now classified with:

```
var validated = validation.classify(classifier);
```

And the error matrix is once again being calculated, this time on the validation data:

```
 var testAccuracy = validated.errorMatrix('Land_Cover_Type_1', 'classification');
```

Display the results:

```
print('Validation error ,atric: ', testAccuracy);
```

To display a colored map of the selected region, it is necessary to define a color palette. For this, we intend to use the same set of 17 colors that were used by the research program called International Geosphere-Biosphere Programme (IGBP) dedicated to studying the phenomenon of global change. This is because Modis data contains the same number of classes or labels: 17 classes, each class being a type of terrain found on the ground [8].
So, having defined a color palette, in our code we use the variable *igpbPalette* (called like that because it works with the IGPB standard colors) and display two layers: a map layer "Landsat" and a layer called „thematic map classified" which is the result of a prediction made by the trained classifier (trained using MODIS source):

```
Map.centerObject(Region_Romania,10);
Map.addLayer(input, {bands: ['B3', 'B2', 'B1'], max: 0.4}, 'landsat');
Map.addLayer(classified, {palette: igbpPalette, min: 0, max: 17}, 'harta tematica-harta clasificata')
```

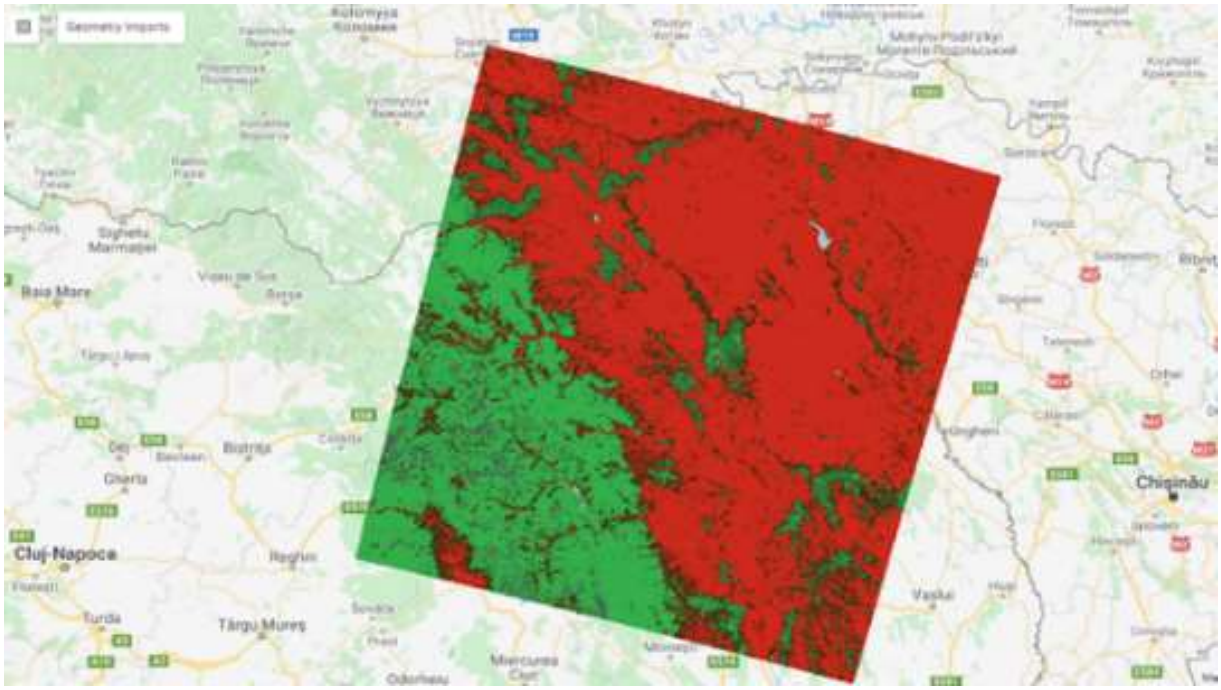

Fig. 4. The results of the classification process using a Machine Learning algorithm

From the above classified map, the researcher can easily "read" the type of terrain (the class) by associating the map colors with the IGBP color palette.
For example:

- the RED color is urban and built-up area
- the BLUE color is water
- the GREEN color is evergreen broadleaf forest or mixed forest
- the GREY color is barren land

4 Jupyter Notebook using Python

For researchers there is a way to work with the entire arsenal offered by Google Earth Engine but not necessarily on the native platform and not necessarily with JavaScript language. Instead, Jupyter platform use the Python language which can do the same work in a different way. Both JavaScript and Python are useful, easy to learn and good for writing short scripts, but Python aims much higher through its ability to work with objects (object-oriented programming language) and, in addition, it is the language that lends itself perfect in the IT areas of Data Analytics, Artificial Intelligence and Machine Learning. Given that the classification process is part of ML jobs, it would be more natural to use Python language. This chapter explains the technique to make the transition from the GEE framework to a Python environment (such as Jupyter Notebook) where users can still use the GEE strengths. To set up the Python environment that allows interaction with Google Earth Engine, a special package called *geemap* is required. *Geemap* package offers the entire Python ecosystem, with its various libraries and special tools for exploring Google Earth Engine [9].
Users who want to migrate to Python ecosystem, should follow the following steps:

1.A new environment called gee is created in Command prompt

```
conda create –n gee python
```

2. The new environment is activated

```
conda activate gee
```

3. From within gee environment, install geemap

```
conda install –c conda – forge geemap
```

4. The default loading of the Jupyter extension is also expected
5. Launch Jupyter Notebook

```
jupyter notebook
```

6. Jupyter opens automatically and write down the first command

```
import geemap
```

And follow the instructions to authorize access needed by Earth Engine.
The Python programmers can write code in Python to succeed in the classification exercise done above. In this new demo it was selected a region around Bucharest (the center of the image having 26.053 and 44.452 as coordinates), with a buffer zone of 10.000 meters. To have an eye-catching display, it was used the near-infrared band (B5), as was explained below:

```
region=ee.Geometry.Point([26.053,44.452]).buffer(10000)
image = ee.ImageCollection('LANDSAT/LC08/C01/T1_SR') \
                    .filterBounds(region) \
                    .filterDate('2019-06-01', '2019-09-30') \

.sort('CLOUD_COVER') \
                    .first() \
                    .select('B[1-7]')
vis_params = {
                    'min': 0,
                    'max': 3000,
                    'bands': ['B5', 'B4', 'B3']}
Map.centerObject(region, 8)
Map.addLayer(image, vis_params, "Landsat-8")
```

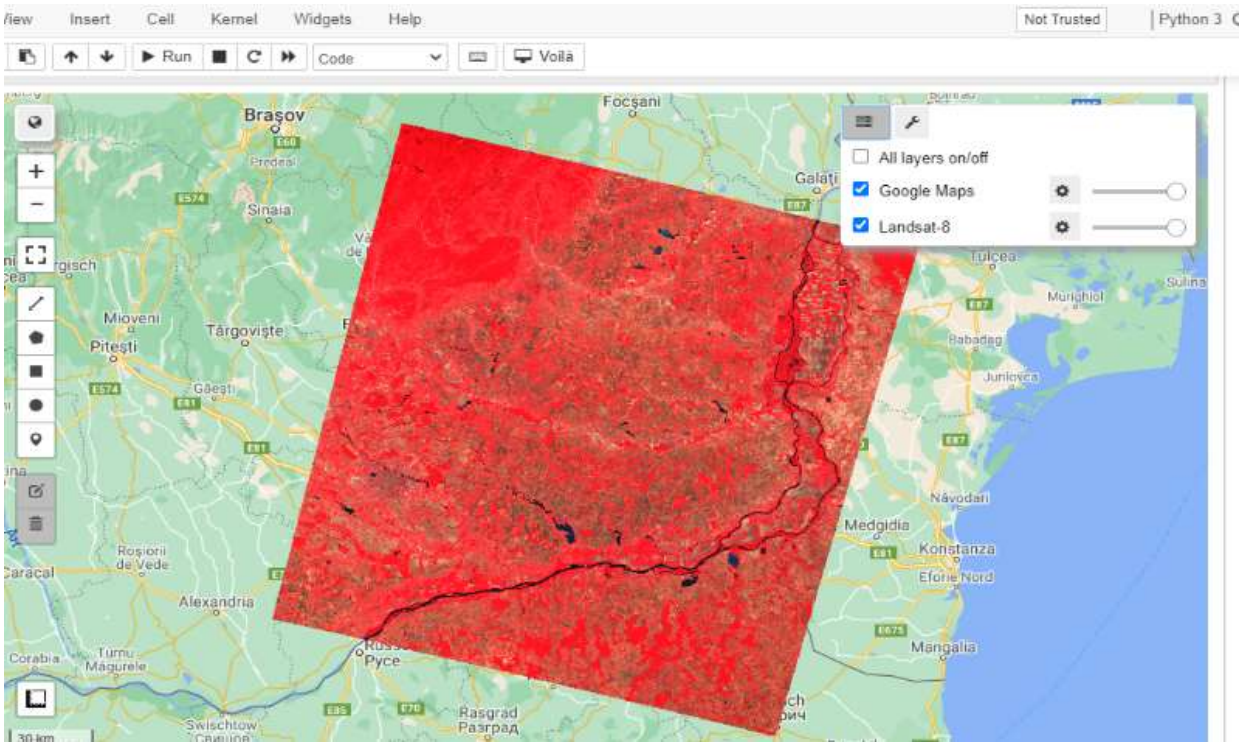

Fig. 5. Selected area of Romania to be classified, in a false-color composite

The image displayed (Figure 5) is in, what is called, a *false-color* composites. Band B5 it was used for near-infrared range, together with B4 and B3 bands (red and green colors). The selection of these bands was made to allow the user to visualize the wavelengths the human eye does not see (near the infrared range). The use of bands, such as near infrared, increases spectral separation and can enhance the interpretability of data. In contrast to a *true-color* image, a *false-color* image is not rendering in natural colors in order to ease the detection of features that are not easy to be seen. The use of near infrared band for the detection of vegetation is very popular method in satellite images. As was the case with the above demonstration of the classification algorithm from within GEE environment (see the previous chapter), the Modis Land Cover Type Yearly Global data are used as the training data. From Modis data, which are already labeled, a sample is extracted based on some criteria. A new layer called "training" and consisting of 5000 Modis points/pixels (black dots on the map as seen in Figure 6), will be displayed on the screen.

This is actually a GEE technique for the formation of training data, by using a method that overlaps the Modis points (with *sampleRegion* function) with the image to be classified (see Figure 6) [10]. In this way, to all the data provided by the chosen image, it will be added that information given by Modis (from the variable “points”) that specifies the class or type of terrain of that pixel. The line of code to do that is:

```
training = image.select(bands).sampleRegions(**{
  'collection': points,
   'properties': [label],
   'scale': 30})
```

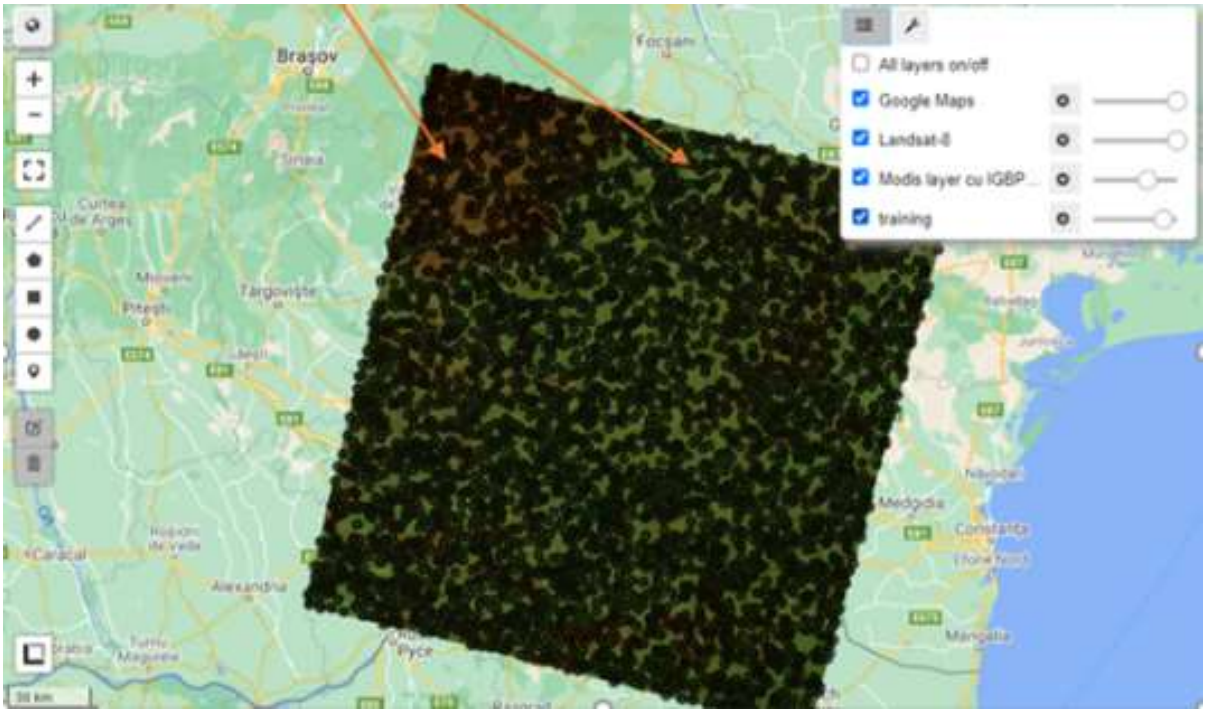

Fig. 6. The overlay of Modis points with the image to be classified

A simple line of code is enough to describe the first Modis point (which is a point, a *Geometry* type) and to demonstrate that MODIS data contains an additional information (needed for classification) called the "class" or "label". This info actually describes the type of terrain of each point contained in the "points" variable:

```
print(points.first().getInfo())
The result is shown below:
{'type': 'Feature', 'geometry': {'type':
'Point',                    'coordinates':
[27.491495166136097,
44.83389043165135]}, 'id': '0', 'proper-
ties': {'LC_Type1': 12}}
```

Here, the description of the first Modis point shows that it resides on the type of land having number 12, which is croplands: at least 60% of area is cultivated cropland. After feeding the model (the *classifier*) with training data in order to learn to predict the type of land of every pixel, the researcher is using it to guess the labels of new data, the data coming from the region of his choice. Because Modis data has 17 classes, to make the demonstration more interesting and colorful, we are using a set of 17 colors (a color for each class) which is the same set used by the International Geosphere-Biosphere Program. The result is a thematic map which is actually a prediction made by the trained classifier (trained using MODIS source, Figure 7).

Fig. 7. The final display of the classified image in IGBP colors

5 Conclusion

Thematic maps, like the one built in this research, helps to simplify and clarify the message of the map. Classification and map generalization matters in every attempt to simplify the real world when raw data brings confusion. However, we need to be cautious because we may create false patterns which will demolish the actual geographic phenomena we are trying to present. The classification process is grouping together similar observations and puts apart those observations that are substantially different. Finding the ideal number of classes is challenging. In the above exercise, intentionally it was used more classes than a usual map classification (it was done for all the classes offered by MODIS data and all 17 colors of International Geosphere-Biosphere Program) but, in order to be safe, is better to make a thematic map with 3-7 data classes. The more colors are used, the harder is to read maps so the risk of map reading errors increases.

This material presented how GEE knows how to process data using its spectacular toolset and functions and wanted to teach researchers how to move to a Python platform to do the same things but with a different language. The benefits of Artificial Intelligence (specifically Machine Learning) shown here, for the task of a map classification, are only a small fraction of the wide range of possibilities and strengths of AI in satellite data processing.

References

[1] J. Schreier, G. Ghazaeyan and O. Dubovyk, "Crop-specific phenomapping by fusing Landsat and Sentinel data with MODIS time series" in *European Journal of Remote Sensing*, Volume 54, 2021, pp. 48-55.

[2] N. Gorelick, M. Hancher, M. Dixon, S. IIyushchenko, D. Thau and R. Moree, "Google Earth Engine: Planetary-scale geospatial analysis for everyone" in Remote Sensing of Environment, Volume 202, 2017, pp. 19-23.

[3] T. Noi Phan, V. Kuch and L. W. Lehnet, "Land Cover Classification using Google Earth Engine and Random Forest Classifier—The Role of Image Composition" in Remote Sensing Journal, Volume 12, 2020, pp. 4-15.

[4] K. Vos, K.D. Splinter, M.D. Harley, J.A. Simmons, I.L. Turner, "A Google Earth Engine-enabled Python toolkit to extract

shorelines from publicly available satellite imagery" in Environmental Modelling& Software Journal, Volume 122, 2019, pp. 3-6.
[5] H. Tamiminia, B. Salehi, M. Mahdianpari, L.J.Quackenbush, S. Adeli and B. Brisco "Google Earth Engine for geo-big data applications: A meta-analysis and systematic review", in *ISPRS Journal of Photogrammetry and Remote Sensing*, Volume 164, 2020, pp. 154-160.
[6] L. Giglio, L. Boschetti, D.P. Roy, M.L Humber, C.O. Justice,"The Collection 6 MODIS burned area mapping algorithm and produc", in Remote Sensing of Environment Journal, Volume 217, 2018, pp. 72-80.
[7] R. Camdem, "An Introduction to GeoJSON", 2019, Available*:* https://developer.here.com/blog/an-introduction-to-geojson.
[8] IGBP, "Vegetation, Water, Humans and the Climate", 2004, Available: http://www.igbp.net/publications/igbpbookseries/igbpbookseries/vegetationwaterhumansandtheclimate2004.5.1b8ae20512db692f2a680007479.html.
[9] Q. Wu, "geemap: A Python package for interactive mapping with Google Earth Engine", in *The Journal of Open Source Software*, Volume 5, 2020, pp. 3-9.
[10] G.S Bhunia, P.K Shit, D. Sengupta,"Free-Open Access Geospatial Data and Tools for Forest Resources Management", in *Spatial Modeling in Forest Resources Management* , 2020, pp. 15-20.

Paul TEODORESCU An engineer by training with a solid background in computer science, has worked in IT field in Romania and Canada. Specializing in databases, PL/SQL, Oracle, Warehousing, Business Intelligence, Artificial Intelligence (Machine Learning, Artificial Neural Networks, Natural Language Processing), he lived and worked for 11 years in Canada. He is currently working at Computer Science Research Institute in Bucharest - ICI - as a research scientist and is involved in Artificial Intelligence and NLP projects.

Simona -Nicoleta VOICU has graduated the „Alexandru Ioan Cuza" University, Iași, Faculty of Philosophy and Social - Political Sciences, Specialization Communication and Public Relations, with a master's degree in Public Relations and Advertising. She has over 7 years of experience in Communication and Social Media, being involved in various projects in the central administration, but also in the private sector. Since 2017 he carries out the activity within the National Institute for Research - Development in Informatics - ICI Bucharest, and lately she undertakes research activities within the Communication, Digital Applications and Systems Department, Communication Infrastructures Service.