

Machine Learning Engineer Nanodegree

Capstone Project

Francisco Javier Proboste Cárdenas
October 31th, 2019

I. Definition

Project Overview

This Project is framed in the field of Environmental Sciences and Conservancy. In particular, it is part of the "remote sensing approaches" to monitor nature dynamics. These techniques have been developed for 50 years, mainly in the context of forestry planning by using Landsat images (Cheng et al, 2017). The main source of information in this field comes from satellite images in the visible spectrum or other frequencies (Latifi and Heurich, 2019). At the beginning, experts performed most of the analysis manually (Aldrich, 1979), but nowadays image recognition techniques provide unprecedented possibilities for massive analysis of the ecosystem. Deep Learning is particularly one of the most promising tools in the development of this field (Khan et al, 2016; Ma et al, 2019).

Problem Statement

In this capstone project I want to tackle a very particular problem in the forest remote sensing field. Nowadays we are facing massive ecosystem degradation over all the earth. One of most damaged are forests. Our cities, food industry and other extractive human activities are destroying large extensions of boreal, template and tropical forests. These forests sustain most of the terrestrial life diversity and also provide oxygen and other environmental services to the entire earth ecosystem. A lot has been talked about where forests are being cut down (Indonesian due to Oil Palm, Amazonian due to cattle rising, etc.), however we don't know much about places where the forest

is growing. Knowing about places that are actually recovering could give us clues about how to recover other places, and how should society behave in order to sustain the life diversity on earth. One of the main approaches to forest quantification are based on satellite images analysis, something that has been studied for decades by geographers in a subfield known as "remote sensing". There has been a recent rebirth in the development of this field due to the possibilities that offers the use of Deep Learning in image recognition. This is something very recent and on the edge of the field (Khan et al, 2016; Ma et al, 2019). In addition, it is important to mention here that most of the earth satellite images captured by NASA or ESO satellites are open to the public through different repositories. These images and metadata associated are used to train the models in this project.

So, in conclusion, we propose to build a supervised classification algorithm to classify the percentage of forest in a given satellite image; which enable to recognize if there is forest recovery over a series of satellite images over time.

Metrics

In terms of the machine-learning field, this is merely a classification problem. The service will use deep learning and transfer learning to train an image classification tool that will classify (discretely in amounts of 5% batches) the amount of forest of a satellite image, and then will analyze if that discrete indicator of amount has increased over time, in a time ordered series of images of one same portion of the earth surface. We will label the images with the percentage of forest cover over the total surface. That data is obtained from the Earth Engine (Google) API for the year 2000 (https://developers.google.com/earth-engine/tutorial_forest_02), which has a Python wrapper that we integrated to the project.

AS this is a supervised classification problem, we proposed to use the ROC Curve to evaluate performance. Actually we use one ROC curve for each batch of forest percentage category to predict, this means having twenty different ROC curves. We can compute the area below the curve for each one (AUC), and then average them to get one testing measurement index to compare the different models. As a business rule we have defined that false negatives are worst than false positives, so the "Area under

the ROC Curve" will be a very good general approach to measure the performance of the different models.

II. Analysis

Data Exploration

As exposed before, this solution focuses on classifying the percentage of forest cover present in a satellite image of the earth surface. In order to complete this task, we used two main data sources:

1. Visible spectrum satellite images of the earth surface
2. The forest cover of each image to label them in one of the twenty different percentage batches

The visible spectrum satellite images are simple png images retrieved from Google Earth software. The images are obtained as screenshots of Google Earth GUI as can bee seen in Figure 1.

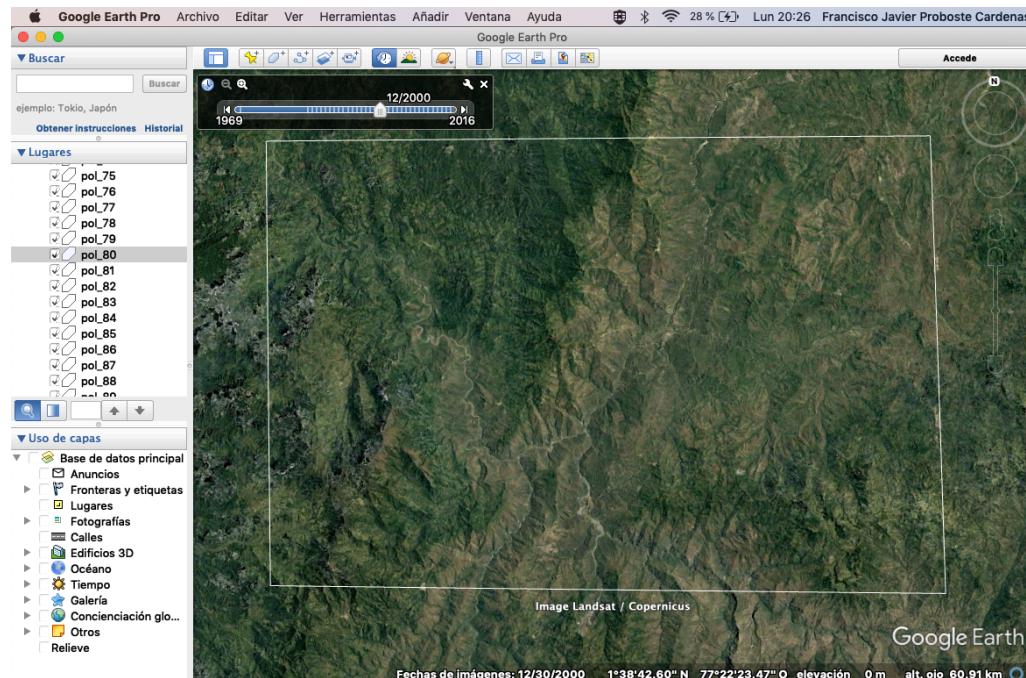


Figure 1: Input png visible spectrum obtaining method.

The image is selected from the year "2000" layer available for Landsat/Copernicus satellites. Each image is stored as a png file with a name composed by "pol_" + correlative number as shown in **Figure 2**. This name is used as a key to identify each image.

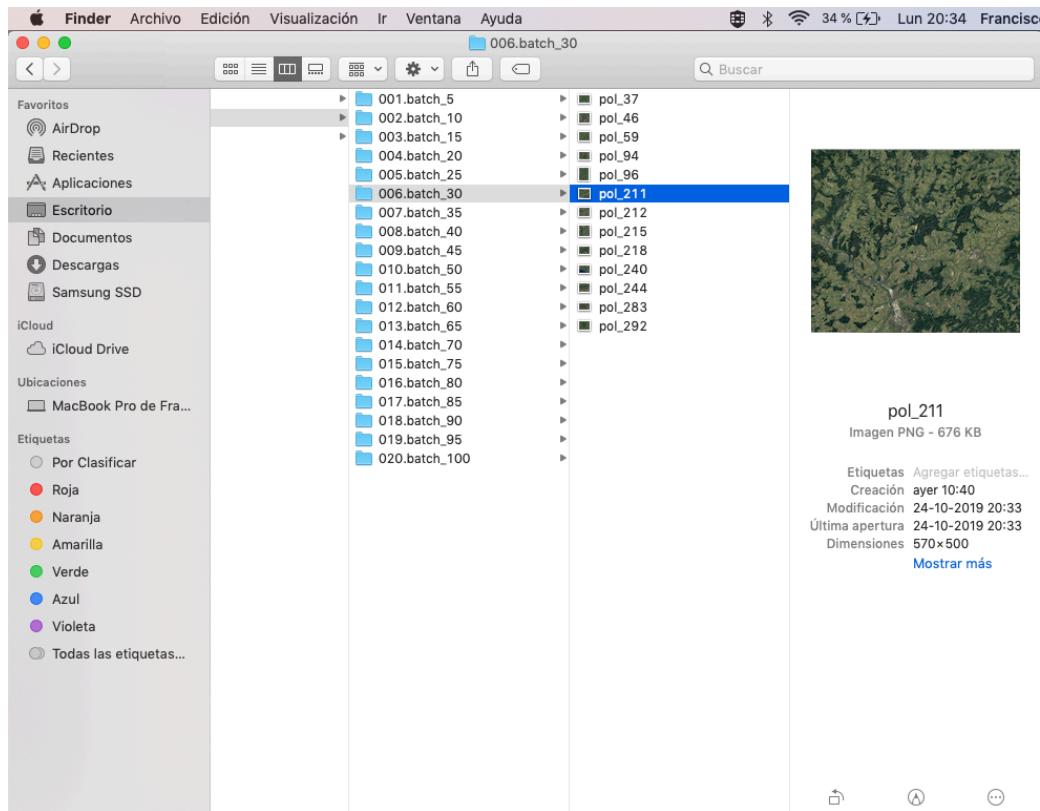


Figure 2: Image labeling and the classification folders

Each image is afterwards loaded to the project using "sklearn.datasets.loadfiles" that represent each png image as a matrix of pixels containing a red/green/blue tuple.

The second source of information is the forest cover of each image. As explained before, this information is used to label each picture in one of the twenty different percentage batches that start in 5% and end up in 100% as can be seen in the classification folder shown in **Figure 2**.

This key information to enable our supervised classification algorithm is obtained from legacy Remote Sensing Techniques applied to satellite images of the year 2000 all over earth. That information is accessible through the Google Cloud Solution Earth Engine, which provides a python API to fetch that information directly from a python script. The specific source of that information inside the Earth Engine cloud software is the "treecover2000" (see Figure 3) of the repo called "Hansen et al. Global Forest Change Data". More info related could be found through the next link: https://developers.google.com/earth-engine/tutorial_forest_02.

The screenshot shows a web browser displaying the Google Earth Engine tutorial page at https://developers.google.com/earth-engine/tutorial_forest_02. The left sidebar contains a navigation menu with sections like API Tutorials, Introduction to JavaScript for Earth Engine, The Earth Engine API, Global Forest Change (which is expanded), Introduction, Introduction to Hansen et al. Global Forest Change Data (which is selected and highlighted in blue), Quantifying Forest Change, Charting Yearly Forest Loss, Introduction to Forest Monitoring for Action (FORMA) data, Global Surface Water, Video Tutorials, and Community Tutorials. The main content area is titled 'The bands in the Global Forest Change data are:' and lists a table of bands. The table has columns for 'Band Name', 'Description', and 'Range'. The bands listed are:

Band Name	Description	Range
treecover2000	Percentage of tree cover in the pixel.	0 - 100
loss	1 if loss ever happen during the study period.	0 or 1
gain	1 if gain ever happen during the study period.	0 or 1
lossyear	The year loss occurred, one-indexed from year 2001, or zero if no loss occurred.	0 - 12
first_b30	The Landsat 7 red band built from the first valid pixels in 2000 (or older if there were no valid pixels in 2000).	0 - 255
first_b40	The Landsat 7 near infrared band built from the first valid pixels in 2000.	0 - 255
first_b50	The first Landsat 7 short wave infrared band built from the first valid pixels in 2000.	0 - 255
first_b70	The second Landsat 7 short wave infrared band built from the first valid pixels in 2000.	0 - 255
last_b30	The Landsat 7 red band built from the latest valid pixels in 2012.	0 - 255
last_b40	The Landsat 7 near infrared band built from the latest valid pixels in 2012.	0 - 255
last_b50	The first Landsat 7 short wave infrared band built from the latest valid pixels 2012.	0 - 255

Figure 3: Available "bands" in the Global Forest Change repository of Earth Engine

More information about how we retrieved this labeling information will be presented in the "Implementation" Section of this report.

Algorithms and Techniques

The algorithm chosen to create a forest cover classifier is Convolutional Neural Network (CNN). As studied along the Machine Learning Engineer Nanodegree, CNN show to be one of the most successful techniques for image recognition and classification. They amplify the power of neural network adding pooling layers to change the dimensions of the input data as represented in Figure 4. This feature is particularly useful for image recognition as it allows making better interpretation of the information of one pixel and its surrounding neighbors. Another argument to select this technique is that along the literature review we performed about Remote Sensing in general, shows that CNN promise important improvements in the field, as can be seen in Ma et al, 2019.

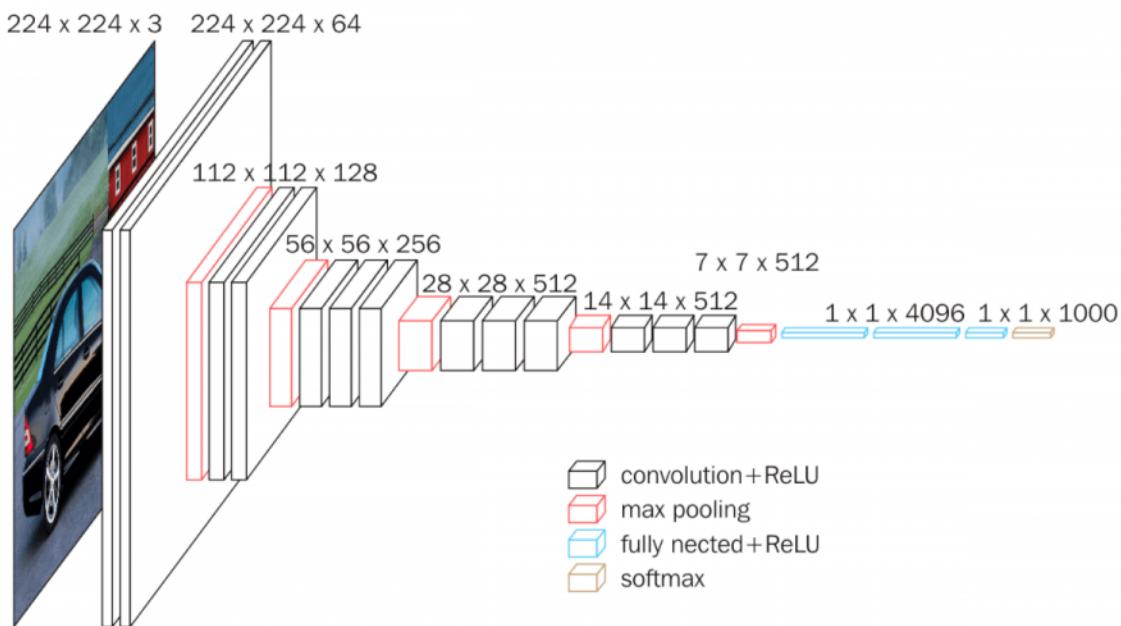


Figure 4: Representation of VGG16 Convolutional Neural Network. <https://neurohive.io/en/popular-networks/vgg16/>

For the implementation of the algorithm we designed one CNN from scratch and then tried other three models using Transfer Learning. Transfer Learning has shown to be an excellent option for image classification tasks when not too many training images are available.

In particular, we chosen these three base models for transfer learning:

1. VGG16
2. InceptionResNetV2
3. DenseNet201

For each of these models we trained “bottleneck features” using our training and validation images as input. Then a second sequential small neural network is trained using the bottleneck features as input and one of the 20 categories as output. A representation of that second network can be seen in [Figure 5](#).

Model: "sequential_8"		
Layer (type)	Output Shape	Param #
global_average_pooling2d_7 ((None, 1536)		0
dense_7 (Dense)	(None, 20)	30740
<hr/>		
Total params: 30,740		
Trainable params: 30,740		
Non-trainable params: 0		

Figure 5: Sequential network feeded with the transfer learning bottleneck features.

Benchmark

As benchmark, we will use the Australian tool for ground cover analysis called “Vegmachine” (<https://vegmachine.net/#>). This tool is an effort of different institutions in Australia to “to summarize decades of change in Australia’s grazing lands”. It is free to use and provides the evolution of the “green cover” since 1991. This greencover can be compared to our Forest Cover category classification. Which might not indicate exactly the same kind of vegetation, but is a proxy. The Vegmachine web GUI allows us to create polygons over the surface of Australia and analyze its ground cover evolution as shown in [Figure 6](#).

So, Vegmachine helps us to compare our Deep Network Forest Cover with a tool that has been fitted by geographers for the Australian Subcontinent.

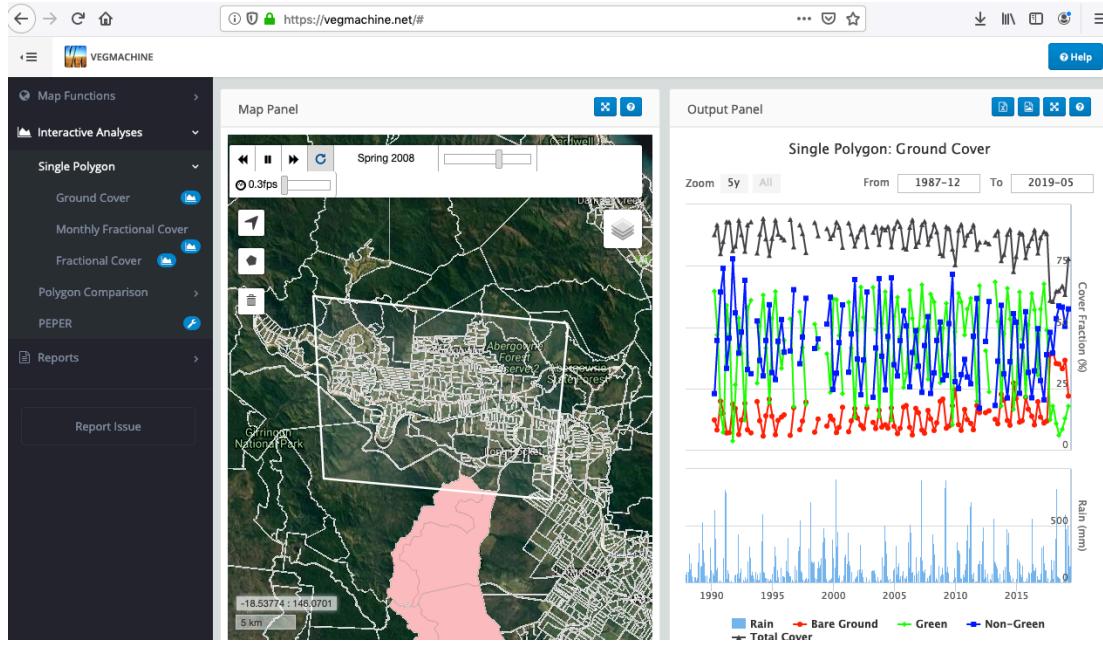


Figure 6: Vegmachine Web GUI

III. Methodology

Data Preprocessing

The first step of the data preprocessing was the labeling. As exposed before, Forest Cover from Earth Engine API is retrieved to label each picture in one of the twenty different percentage batches that start in 5% and end up in 100%. These batches can be seen in the classification folder shown in Figure 2.

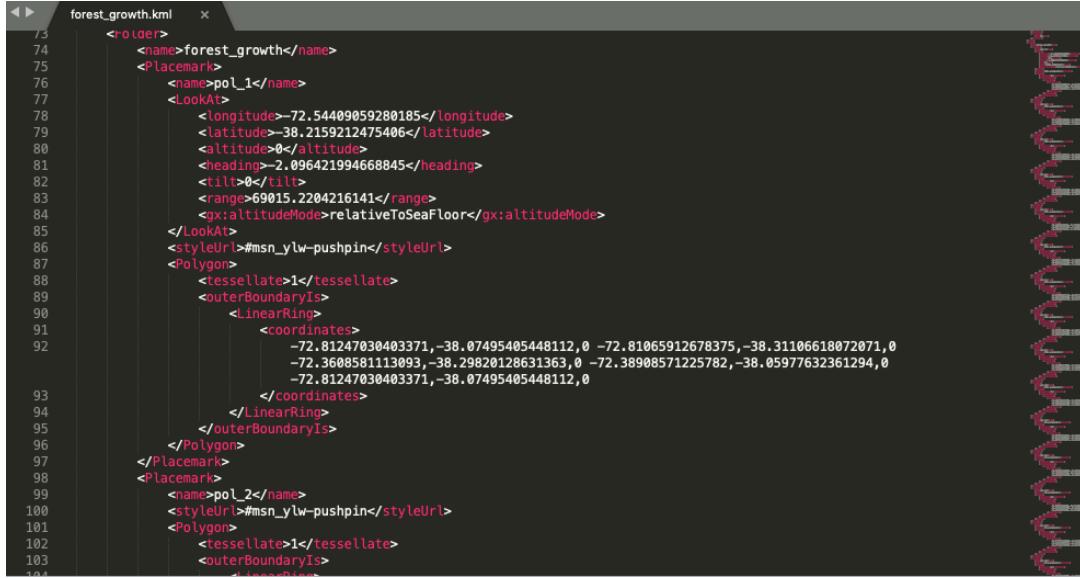
That information is accessible through the Google Cloud Solution called Earth Engine, which provides a python API to fetch that information directly from a python script. The specific source of that information inside the Earth Engine cloud software is the "treecover2000" band (see Figure 3) of the repo called "Hansen et al. Global Forest Change Data". In order to get the percentage of forest cover from that source it is needed to provide a polygon with coordinates as input. That georeferenced polygon indicates the API, which is the portion of the earth surface that we are inspecting. We manually built one polygon through the Google Earth Software for each of the images retrieved. These polygons, and their respective images where captured from the Americas, Europe, Africa, Australia and South East Asia, representing a well balanced mix of different forests

of the earth. Figure 7 shows the polygons associated to images taken from South America.



Figure 7: Polygon construction for the labeling process.

After manually building those polygons, we saved them in a KML file to be processed afterwards. That KML file has some kind of JSON dictionary for each of the polygons containing the name given in Google Earth, and all their four corner geographical coordinates. Figure 8 shows the look of the structure of the KML file.



```

<?xml version='1.0' encoding='UTF-8'?>
<Folder>
  <name>forest_growth</name>
  <Placemark>
    <name>pol_1</name>
    <LookAt>
      <longitude>-72.54409059280185</longitude>
      <latitude>-38.21592212475406</latitude>
      <altitude>0</altitude>
      <heading>-2.096421994668845</heading>
      <tilt>0</tilt>
      <range>69015.2204216141</range>
      <gx:altitudeMode>relativeToSeaFloor</gx:altitudeMode>
    </LookAt>
    <styleUrl>#msn_ylw-pushpin</styleUrl>
    <Polygon>
      <tessellate>1</tessellate>
      <outerBoundaryIs>
        <LinearRing>
          <coordinates>
            -72.81247030403371,-38.07495405448112,0 -72.81065912678375,-38.31106618072071,0
            -72.3608581113093,-38.29820128631363,0 -72.38908571225782,-38.05977632361294,0
            -72.81247030403371,-38.07495405448112,0
          </coordinates>
        </LinearRing>
      </outerBoundaryIs>
    </Polygon>
  </Placemark>
  <Placemark>
    <name>pol_2</name>
    <styleUrl>#msn_ylw-pushpin</styleUrl>
    <Polygon>
      <tessellate>1</tessellate>
      <outerBoundaryIs>

```

Figure 8: KML file containing polygon data for each image.

As manually querying the forest cover information for each of the 400 polygons would get too much time, an automatic KML parser was built to get a tuple with the name and coordinates for each polygon (`read_polygons` in `data_loading.py`). Then, that list of tuples is passed to the function called “`get_earth_engine_green_cover`” in `data_loading.py`, which establishes connection with the Earth Engine API, and then fetches the forest cover for each of the polygons. After these two steps, we get the forest cover to label all our images. With that information, images are classified into one of the twenty folders that represent each batch of forest cover percentage.

The other step in the preprocessing is the image resizing. This process is simpler than the one explained before. In this case, we use the “image” module of `keras.preprocessing`, as `keras` is the selected library to develop our deep learning networks. The size of the images chosen to be used by the CNN models is 224 by 224 pixels. This transformation is performed just after data loading in the “`path_to_tensor`” function of `preprocess.py`.

With these two steps done, images and label data is ready to train our CNN networks.

Implementation

The deep learning library chosen to develop our model is Keras. As exposed before we developed two kind of models: one made from

scratch designing the layers from zero, and secondly using transfer learning to enhance a final classification model.

The transfer learning models used are:

1. VGG16
2. InceptionResNetV2
3. DenseNet201

All this model construction, training and tuning was developed in the Jupyter Notebook in the script “main_train.ipynb”, which has some markdown text to explain the procedure and the calling of other python functions developed for the project.

The first model made from scratch is a Keras sequential model, and is composed by a series of three set of convolutional and max pooling layers pairs. After these three pairs we add a global average pooling layer and a fully connected layer to represent a final output of 20 classification categories. This architecture can be observed in **Figure 9**.

Model: "sequential_9"		
Layer (type)	Output Shape	Param #
conv2d_210 (Conv2D)	(None, 224, 224, 16)	208
max_pooling2d_11 (MaxPooling)	(None, 112, 112, 16)	0
conv2d_211 (Conv2D)	(None, 112, 112, 32)	2080
max_pooling2d_12 (MaxPooling)	(None, 56, 56, 32)	0
conv2d_212 (Conv2D)	(None, 56, 56, 64)	8256
max_pooling2d_13 (MaxPooling)	(None, 28, 28, 64)	0
global_average_pooling2d_8 ((None, 64)	0
dense_8 (Dense)	(None, 20)	1300
<hr/>		
Total params: 11,844		
Trainable params: 11,844		
Non-trainable params: 0		

Figure 9: Convolutional Neural Network Architecture for Forest Cover Classification

On the other hand, the transfer learning models are developed training “bottleneck features” using the images on the base

models (VGG16, InceptionResNetV2 and XXX), available in “keras.applications” module. Three sets of bottleneck features are trained for train, validation and testing images as shown in **Figure 10**. The bottleneck features are obtained by applying the already trained base models on the images, by the keras.model.predict function.

Part 2 - Get VGG16 bottleneck features with our data

```
In [31]: # Now we do prediction over our training and validation images
vgg16_bottleneck_features_train = np.asarray([vgg16_first_model.predict(np.expand_dims(tensor, axis=0)) for tensor in train_data])
vgg16_bottleneck_features_valid = np.asarray([vgg16_first_model.predict(np.expand_dims(tensor, axis=0)) for tensor in validation_data])
vgg16_bottleneck_features_test = np.asarray([vgg16_first_model.predict(np.expand_dims(tensor, axis=0)) for tensor in test_data])
```

Figure 10: Bottleneck Features Training

The output of the predict function over the images is stored as numpy arrays that can be afterwards used as input for our second part of the transfer learning model. This second stage of the model consist on a max pooling layer with a shape that fits with the bottleneck features output, and a dense layer that enables this second part of the model as a fully connected classifier that funnels the data flow into the 20 desired categories. A representation of VGG16 as base model and then a sequential fully connected layer scheme can be observed in **Figure 11**.

This second stage part of the model is trained for each of the three base models using the bottleneck features of training and validation images. Finally the output of the trained models can be evaluated using the test images and bottleneck features.

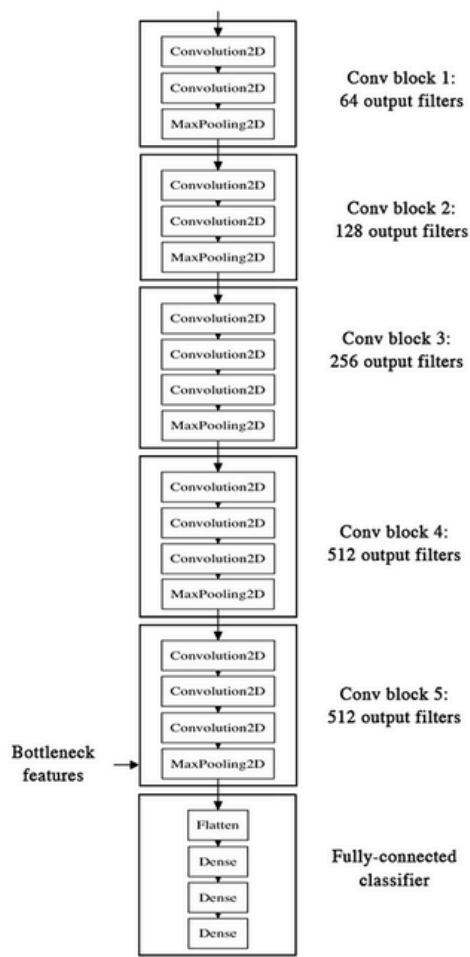


Figure 11: VGG16 Transfer learning model example.

Source: <https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html>

As this is a classification model, we decided to use the ROC Curve and its AUC area as validation metric. This is not a binary classification though, so we needed to implement an adaptation of the ROC Curve to a multiclass classification using "micro average ROC Curve" and "macro average ROC Curve". The idea behind this adaptations is simple. We just built one ROC Curve for each of the categories label. Then we compute the AUC for each of the "micro" curves and then average the twenty different indicators as a "macro" AUC that represents the whole model.

This is the whole classification model built upon Convolutional Neural Networks.

Refinement

The refinement process started with the first implementation of the models exposed in the previous sections. These model triggered results that were able to be enhanced with some tuning in the model. In particular, we decided to try the models with different "batch sizes" and "epochs". These two parameters enabled some result optimization that will be exposed in the results section of this report.

IV. Results

Model Evaluation and Validation

As stated in previous sections, the evaluation metric is the ROC Curve and its area under the curve called AUC. As learned from the Program classes The "Receiver Operating Characteristic" (ROC) Curve show the relation between the test cases rate that show to be False Positives and the True Positives. As this is multiclass classification problem, we need to adapt the traditional binary classification scheme of the ROC Curve. In this case, "micro average ROC Curve" and "macro average ROC Curve" are used. The idea behind this adaptation is simple. A specific ROC Curve is built for each of the categories label. Then we compute an averaged ROC Curve and its AUC and use it as a representative of the whole model.

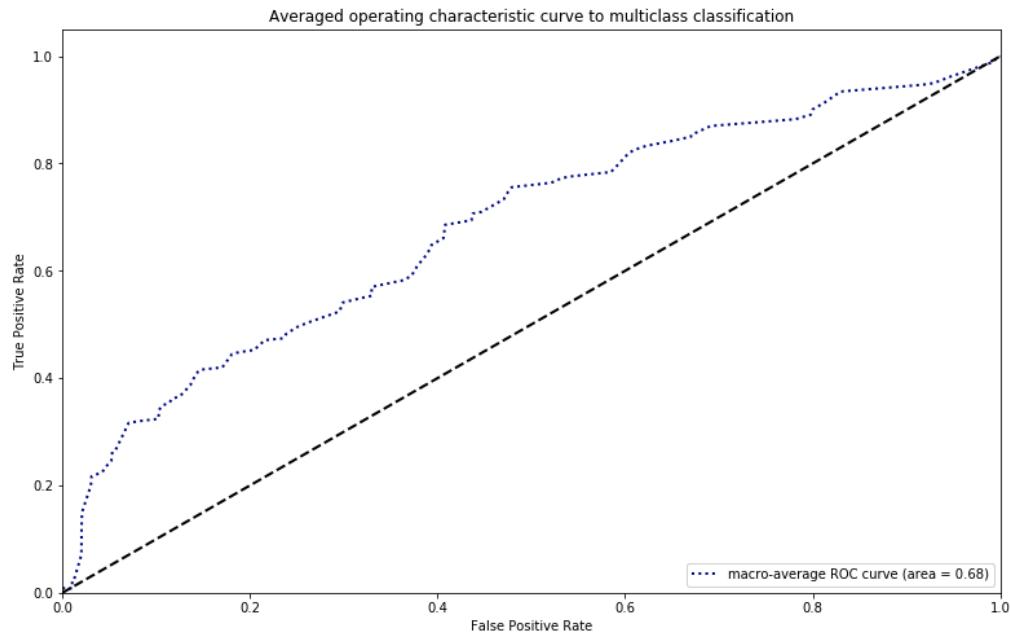


Figure 12: Macro-Average ROC Curve for the Scratch Model

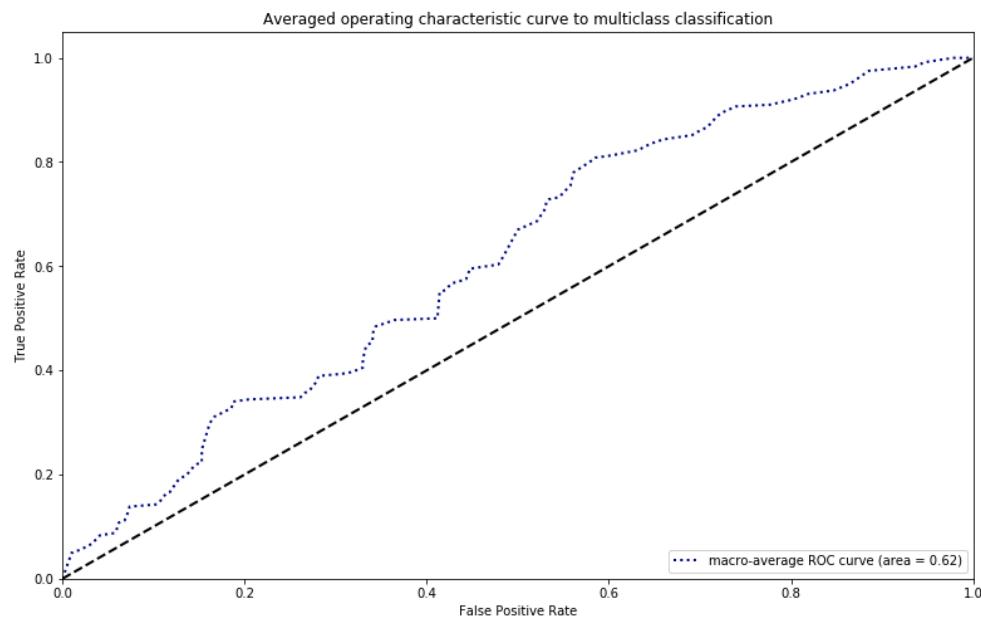


Figure 13: Macro-Average ROC Curve for the VGG16 Model

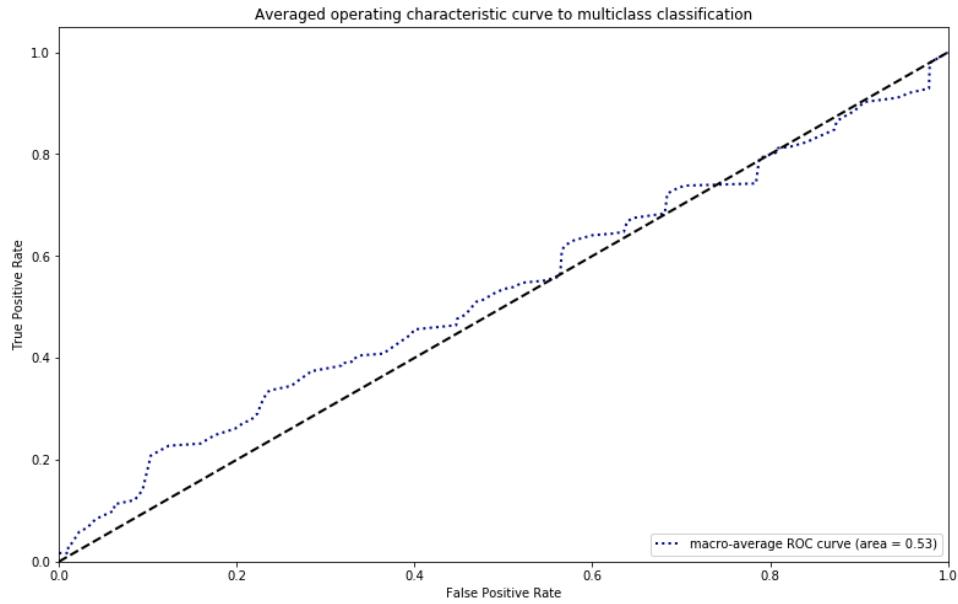


Figure 14: Macro-Average ROC Curve for the Inception Resnet V2 Model

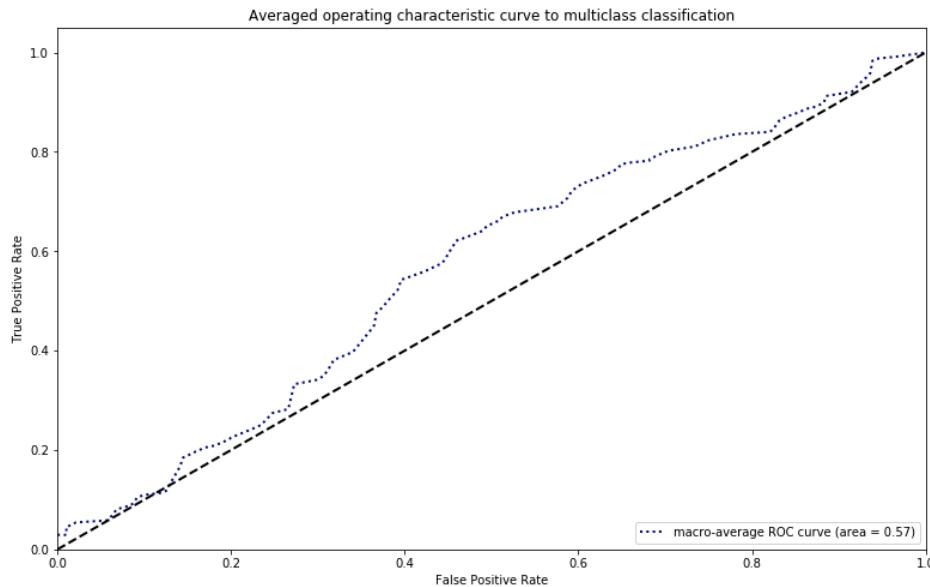


Figure 15: Macro-Average ROC Curve for the DenseNet201 Model

As seen in the previous figures, the Model that presents the highest AUC indicator is the CNN Model made from scratch. This fact surprises me as I thought transfer learning would improve the results form the first scratch model, as is documented in different sources that indicate the usefulness of using transfer

learning. One of the reasons behind this surprising result might be that all these three transfer-learning models were trained using the Imagenet Image files repository that does not contain geographical pictures. This image repository is focused on objects, animals, humans, etc. and thus, its training focuses more on understanding the shape of one element in relation to its surroundings. This is something not very clear in satellite forest images.

Justification

For the benchmark comparison, we selected an area in Australia (close to the Gold Coast, see [Figure 16](#)) where we can apply the comparison source called "Vegmachine". As exposed before, Vegmachine is an Australian tool to monitor their territory using Satellite Remote Sensing Techniques.

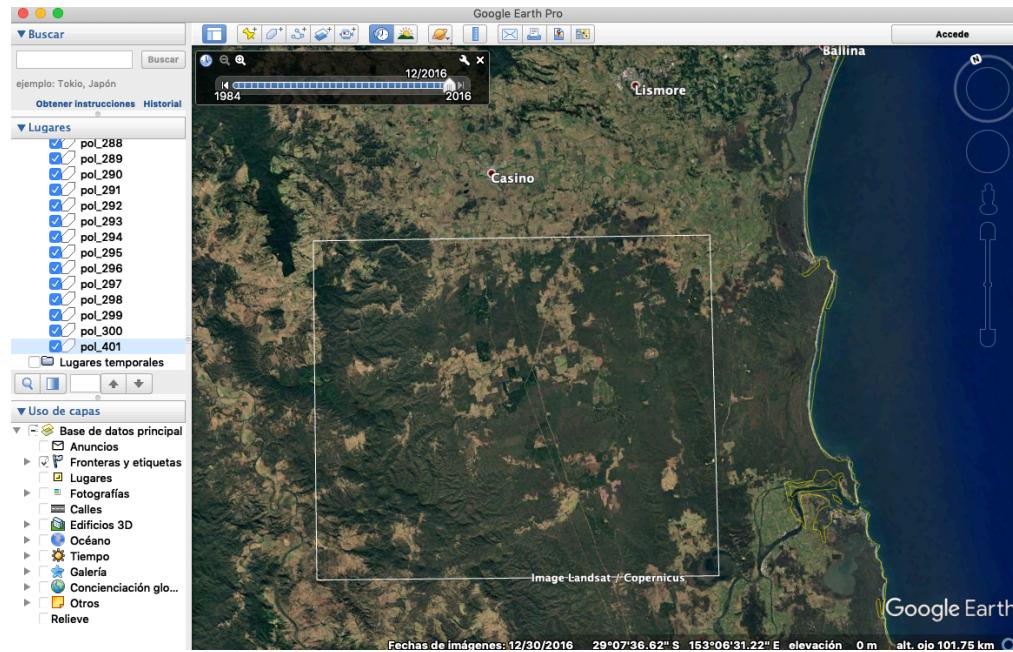


Figure 16: Area selected for benchmark comparison in Google Earth to be processed in the best Model.

Applying our model to six images of this area each 5 years starting in 1991 (+1996, 2001, 2006, 2011 and 2016), the results show that there is no significant change in the forest cover. If we check the results shown in [Figure 17](#), the batches are always #9,

that means that the image is predicted to have 50% of forest cover, with no variance among batches for all the series.

Finally, lets load the benchmark images and predict their forest cover batches from the scratch model, to compare that info with the benchmark

```
In [4]: benchmark_tensor= preprocess.paths_to_tensor(data_dict['benchmark_files'])
100%|██████████| 6/6 [00:00<00:00, 45.80it/s]

In [12]: # Get forest batch prediction
scratch_model_predictions = [np.argmax(model.predict(np.expand_dims(tensor, axis=0))) for tensor in benchmark_tensor]

In [13]: scratch_model_predictions
Out[13]: [9, 9, 9, 9, 9, 9]
```

Figure 17: Benchmark images Scratch Model result

On the other hand, Vegmachines has information about green cover just starting in 2014, what makes very poor the comparison. The green cover in Vegmachine shows to be slightly less than 50% from 2014 to 2017, and after that point it goes down to less than 25% (wild fires maybe?). This can be observed through the green line in Figure 18. The only comparison we can make is that the image processed in our model for 2016, shows a percentage of forest relatively similar to the one exposed in Vegmachine. A better Benchmark could be very useful, but this forest cover information is too difficult to find for the granularity of data we are working with.

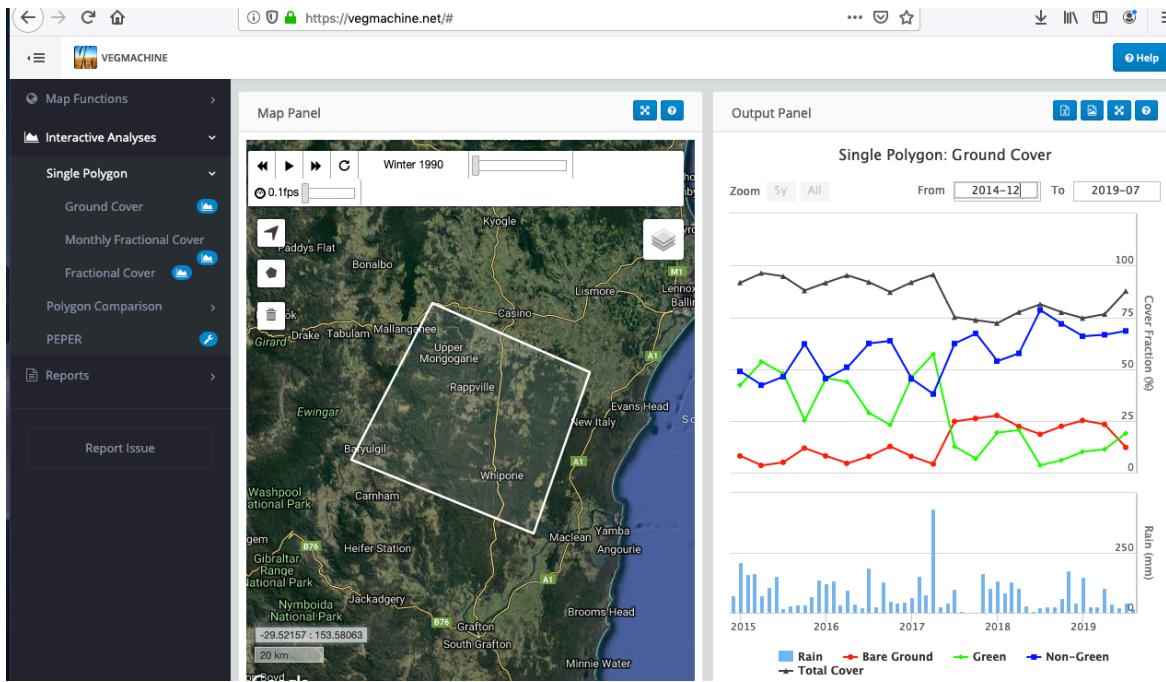


Figure 18: Vegmachine algorithm over Benchmark polygon

V. Conclusion

Free-Form Visualization

Multiclass classification poses significant difficulties in relation to binary classification problems. It seems that the data required for this kind of problem must be way more abundant to create relevant and useful tools. This multiclass classification is in my opinion the most complex and important feature of this project. Figure 19 synthesizes some of that complexity by showing the ROC Curves of all the 20 categories for the best model (Scratch Model).

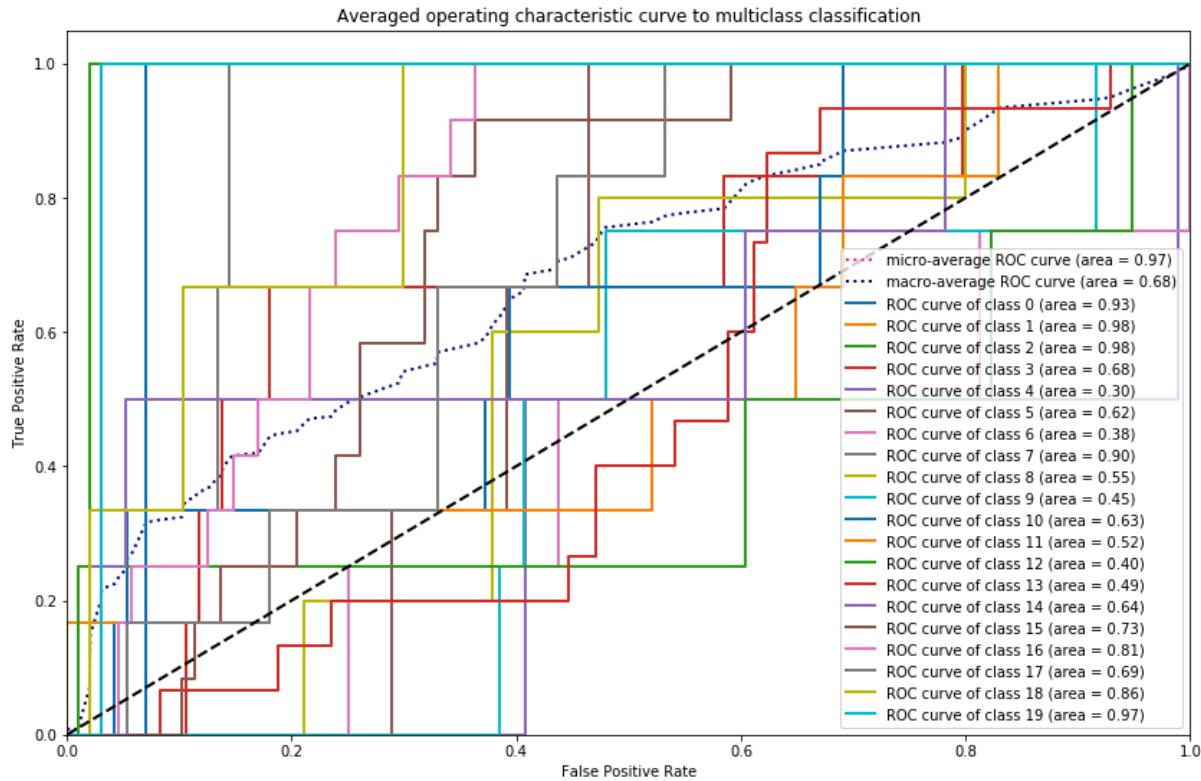


Figure 19: ROC Curves for all the 20 categories in the Scratch Model

Reflection

The solution developed in this project is better than simple random choice. We started by researching the tools necessary to retrieve information from satellite repositories, and had to learn how to interact with Google Cloud for that. Then, it was necessary to make an exhaustive work designing the images to be analyzed and its polygons to get a forest cover label. All this data preparation effort was probably the most difficult part of the project as needed abundant research and learning about new tools I didn't know nor learnt through the Nanodegree. Afterwards the CNN models were developed. This part was fun as I already felt confortable with what I learned from that section of the Nanodegree and the Dog-Breed Classifier Project. The best model got an AUC rate of almost 70% that is not insignificant, showing there is effectively some intelligence in all the training and the models. This is far from the results I expected though. As read from the academic literature about the promises of deep learning over the remote sensing field, I expect we can do much better.

Improvement

I think that a software like the one developed in this project could be promising, specially thinking in the benefits image classification has shown over diverse fields. Certainly, there is the need of exploring some different models, in my opinion, however, the largest opportunity to enhance this solution is by labeling a massive amount of images that could reflect the rich diversity of satellite image possibilities: water mixed with forest, different colors among vegetation, ice, rocks, mountains, cities and so many other odd figures of the geography that make the earth the beautiful place it is.

Another structural difference that should be explored is trying to predict the forest cover as a continuum number between 0 and 100. I though this could be something less promising as it might be difficult to tackle with deep learning, but, perhaps there is other machine learning techniques that could leverage better results for this problem in a continuum spectrum of possibilities.

Academic References

Aldrich Robert C. (1979). Remote Sensing of Wildland Resources: A State-of-the-Art Review. General Technical Report RM-7.1Rocky Mountain Forest and Range Experiment Station. Forest Service, USDA Forest Service

G. Cheng, J. Han and X. Lu, "Remote Sensing Image Scene Classification: Benchmark and State of the Art," in Proceedings of the IEEE, vol. 105, no. 10, pp. 1865-1883, Oct. 2017.doi: 10.1109/JPROC.2017.2675998

Michael A. Wulder, Nicholas C. Coops, David P. Roy, Joanne C. White & Txomin Hermosilla (2018) Land cover 2.0, International Journal of Remote Sensing, 39:12, 4254-4284, DOI: 10.1080/01431161.2018.1452075

Latifi, Hooman & Heurich, Marco. (2019). Multi-Scale Remote Sensing-Assisted Forest Inventory: A Glimpse of the State-of-the-Art and Future Prospects. Remote Sensing. 11. 1260. 10.3390/rs1111260.

Lei Ma, Yu Liu, Xueliang Zhang, Yuanxin Ye, Gaofei Yin, Brian Alan Johnson,

Deep learning in remote sensing applications: A meta-analysis and review, ISPRS Journal of Photogrammetry and Remote Sensing, Volume 152, 2019, 166–177, ISSN 0924-2716,
<https://doi.org/10.1016/j.isprsjprs.2019.04.015>.