

**VSB - Technical University of Ostrava**  
**Faculty of University Study Programmes**  
**IT4Innovations**

**Application of Machine Learning Algorithms for forest monitoring  
by Satellite SAR data**

**Aplikace algoritmů strojového učení pro sledování lesů  
prostřednictvím družicových radarových dat**

## Diploma Thesis Assignment

Student: **Sweety Wadhwa**

Study Programme: N2658 Computational Sciences

Study Branch: 2612T078 Computational Sciences

Title: **Application of machine learning algorithms for forest monitoring by satellite SAR data**  
**Aplikace algoritmů strojového učení pro sledování lesů prostřednictvím družicových radarových dat**

The thesis language: English

### Description:

The student will learn about processing satellite Synthetic Aperture Radar (SAR) images and prepare a methodology based on using novel computational approaches to fulfill tasks related to monitoring of forest damages using Sentinel-1 dual polarization data.

In particular, the student will find and further develop algorithms suitable for identification of forest damage (e.g. windfall) and deforestation, e.g. Non-Local Mean filtering or application of convolutional neural networks. This consists in SAR data preprocessing (coregistration, radiometric corrections etc.) and specific multitemporal speckle filtering, especially focusing on edge identification and algorithms for detection of polygons that show temporal changes.

The methodology should be applied to identify forest segments with trees fallen by extreme winds in the Czech Republic.

Student will develop scripts for automatization of the process in the environment of Linux OS.

### References:

C. Danilla, 2016: Convolutional neural networks for contextual denoising and classification of SAR images, MSc. thesis, University of Twente, Enschede, 68 p.

Weijiang Zhao, Loïc Denis, Charles-Alban Deledalle, Henri Maître, Jean-Marie Nicolas, et al., Ratiobased multi-temporal SAR images denoising. 2018. <hal-01791355>

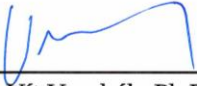
Maître, H. et al. Processing of Synthetic Aperture Radar Images. Paris: Ecole Nationale Supérieure des Télécommunications, 2008, p. 408. isbn: 9-781848210240.

Extent and terms of a thesis are specified in directions for its elaboration that are opened to the public on the web sites of the faculty.


Supervisor:  **Ing. Milan Lazecký, Ph.D.**

Date of issue: 9-7-2018

Date of submission: 30.4.2019

  
doc. Mgr. Vít Vondrák, Ph.D.  
Head of Department



  
Ing. Zdeňka Chmelíková, Ph.D.  
Vice-rectress for Study Affairs

## Declaration

I declare that this thesis written by myself. I have listed all the sources and publications from which I drew.

Ostrava, April 30, 2019



I would like to thank my supervisor Ing. Milan Lazecký, Ph.D. for his continued support and providing me the guidance necessary to work on this project. Without Ing. Milan help I was unable to do this. I would like to thank my advisor Ing. Jan Martinovic, Ph.D. and Ing. Michal Podhoranyi, Ph.D. for teaching me core skills needed to succeed and reviewing my project.

## **Abstract**

Hurricanes cannot be controlled by humans, and are increasing in number day by day. Hurricanes are responsible for large-scale loss of life and assets. They appear within a very short time, and are unstoppable by people once they have started. Therefore, for effective risk management, damage should be assessed after the disaster. Satellite radar images (SAR) have advantages because the radar sensor can work in all-weather conditions, is not affected by clouds, so the use of SAR imagery is useful in identifying damage and loss of assets. In our project, we selected the Jeseník area, because a hurricane occurred on March 17, 2018, and there were substantial losses in forest areas in particular where there are many homes. Sentinel 1 (S1) images have been used, some from the pre-disaster period and others from the post-disaster period. Backscatter values are analyzed in both images. It is expected that the difference between post-disaster images will be greater than the pre-disaster images. (In case of extensive damage). After applying the segmentation algorithm, we find out the segmentation of different area. The results show a polygon for damages detected by SAR images.

## **Keywords**

Edge detection, Forest Segmentation, Detect forest windfall area, Disaster Management, SAR.

## **Abstrakt**

Hurikány nemohou být kontrolovány člověkem a každým dnem se zvyšuje jejich počet. Hurikány jsou zodpovědné za rozsáhlé ztráty na životech a majetku. Objevují se ve velmi krátkém čase a nemohou být zastaveny lidmi. Proto by pro účinné řízení rizik měla být posouzena míra škod po katastrofě. Družicové radarové snímky (SAR) mají výhody, protože radarový senzor může pracovat za všech povětrnostních podmínek, není ovlivněn oblačností, takže použití snímků SAR je užitečné při identifikaci poškození a ztráty majetku. V našem projektu jsme vybrali oblast Jeseník, protože 17. března 2018 došlo k hurikánu a v lesních oblastech došlo k velkým ztrátám, zejména v blízkosti obydlených oblastí. Byly použity snímky Sentinel 1 (S1), některé z období před katastrofou a jiné z období po katastrofě. Hodnoty zpětného odrazu jsou analyzovány na obou snímcích. Očekává se, že rozdíl mezi obrazy po katastrofě bude větší než obrazy období před katastrofou. (v případě, že dojde k rozsáhlému poškození). Po použití segmentačního algoritmu zjišťujeme segmentaci různých oblastí. Výsledky ukazují polygon pro poškození detekované obrazy SAR.

## **Klíčová slova**

Detekce hran, Segmentace lesů, Detekce lesního poškození větrnými poryvy, Řízení katastrof, SAR

# Table of Contents

List of used abbreviations .....	8
List of Figures .....	9
List of Tables.....	10
1 Introduction .....	11
2 Study Area and Data .....	14
2.1 Study Area.....	14
2.2 Data .....	15
3 Tools and libraries used.....	18
3.1 Python .....	18
3.2 Anaconda.....	18
3.3 Jupyter Notebook .....	18
3.4 OS.....	18
3.5 OpenCV-Python.....	18
3.6 NumPy.....	19
3.7 Matplotlib Tutorial: Python Plotting .....	19
3.8 GDAL/OGR in Python.....	19
3.9 SNAP.....	20
4 Methodology .....	21
4.1 Data pre-processing.....	21
4.1.1 Pre-step.....	21
4.1.2 Averaging descending and ascending tracks for before and after the event: .....	22
4.1.3 Merging the before and after images:.....	25
4.2 Forest Segmentation.....	26
4.2.1 Pre-step:.....	27
4.2.2 Denoising Filter.....	29
4.2.3 Thresholding: .....	32
4.2.4 Edge Detection .....	35
4.2.5 Morphological Transformation .....	39
4.3 Vectorization / Polygon Shape file.....	43
4.4 Testing Result.....	46
4.4.1 Test 1 .....	46
4.4.2 Conclusion for Test 1 .....	50
4.4.3 Test 2.....	50

4.4.4 Conclusion for Test 2 .....	52
5 Application for edge detection .....	53
6 Conclusions .....	54
7 Future work .....	55
8 References .....	56
9 Appendix .....	59
A1. Histogram equalization.....	59
A2. Non-Local Mean Filter .....	59
A3. Simple Thresholding .....	59
A4. Canny Edge Detection.....	60

## List of used abbreviations

SAR	Synthetic Aperture Radar
S1	Sentinel 1
NL Mean filter	Non-Local Mean filter
PCA	Principal Component Analysis
GDAL	Geospatial Data Abstraction Library



## List of Figures

Figure 1: Process flow of the project .....	12
Figure 2 Study area, Jeseniky Area, Czech Republic (Google Map).....	14
Figure 3: Descending Pass Orbits 22 and 124.....	15
Figure 4: Ascending pass orbits 73 and 175.....	15
Figure 5: Average images from SENTINEL-1 images: before and after the hurricane for both pass orbits .....	16
Figure 6: Calibration Process .....	21
Figure 7: VH Intensity Maps for Sentinel-1 Images .....	24
Figure 8: Finding the difference between after and before the event using PCA with component 3.....	26
Figure 9: Process flow for the forest segmentation.....	27
Figure 10: Histogram of input data .....	28
Figure 11: Left-hand side image without equalization and right-hand side image output of histogram equalization. ....	28
Figure 12: Left-hand side histogram of without equalization and right-hand side histogram for equalization image .....	29
Figure 13: Output of NL Means denoising with different parameter for without equalization input data...31	
Figure 14: Output for Simple thresholding with different methods .....	33
Figure 15: working of THRESH_TOZERO simple thresholding.....	33
Figure 16: THRESH_TOZERO output image with threshold_value = 10 and max_value = 255 .....	34
Figure 17: Output for the edge detection algorithm .....	35
Figure 18: Canny Edge Detection Output for min_value = 100 and max_value = 450 .....	37
Figure 19: Canny Edge Detection Output for min_value = 150 and max_value = 700 .....	37
Figure 20: Canny Edge Detection Output for min_value = 210 and max_value = 950 .....	38
Figure 21: Canny Edge Detection Output for min_value = 250 and max_value = 1000 .....	39
Figure 22: Dilation Output with kernel (1, 10) .....	41
Figure 23: Erosion output with kernel (1, 10) .....	41
Figure 24: Dilation Output with kernel (5, 10) .....	42
Figure 25: Erosion output with kernel (5, 10).....	42
Figure 26: Dilation Output with kernel (10, 10) .....	43
Figure 27: Erosion output with kernel (1, 10).....	43
Figure 28 : Syntax of GDAL_polygonize method [24] .....	44
Figure 29: Shape file output .....	46
Figure 30: Output of NL Means denoising .....	48
Figure 31: Output of THRESH_TOZERO thresholding.....	48
Figure 32: Output of Canny Edge Detection.....	49
Figure 33: Output for Dilation .....	49
Figure 34: Output for Erosion .....	50
Figure 35: Result from Ing. Marek Mlcousek Jesenik area. ....	51
Figure 36: Our final result for Jesenik area.....	51
Figure 37: Our result overlaid on newest ortophotomap. Affected forest segments are visible. Our result is fitting these areas very well.....	52
Figure 38: Application for setting all the parameter .....	53

## List of Tables

Table 1: Features of Sentinel-1 Data.....	17
Table 2: Non-local means denoising parameter details for without equalization input data (here src =PCA Component 3 output image and dst = none) .....	29
Table 3: Simple Thresholding Methods with different parameter (here input_image = NL Mean denoising output image) .....	32
Table 4 : THRESH_TOZERO thresholding with different parameter (here input_image = Denoising output image and thresholding_name = cv2.THRESH_TOZERO).....	34
Table 5: Different parameter detail for Canny Edge Detection (here Input_image = THRESH_TOZERO output image) .....	36
Table 6: Morphological Transformation test for Dilation and Erosion with different size of kernel .....	40
Table 7: Non-local means denoising (here src =PCA Component 3 output image and dst = none).....	47
Table 8: THRESH_TOZERO thresholding (here input_image = Denoising output image and thresholding_name = cv2.THRESH_TOZERO).....	47
Table 9: Canny Edge Detection (here Input_image = THRESH_TOZERO output image) .....	47
Table 10: Size of kernel for Dilation and Erosion operation .....	47

# 1 Introduction

This project based on find out the damaged forest, after heavy wind or storm. Storm with high windstorm is cause of damage forest. It is natural phenomenon, but due to that heavy storm forest and economic is damage. Damage forest can increase the fungal infection with high wind [1]. We need to clean the damaged forest immediately but due to limited manpower and resources, we are using the different approach to clean the forest. Bark beetle dispersion are the good approach to protect undamaged tree and we can save the economic losses [2]. Recently, the events of extreme storms have influenced Central Europe regularly, changing climate very frequently [4]. For example, Europe and Asia, is suffered by Ips typographus.

We can get the rough estimation of damaged area with field survey and air imagery. But both method are costly and time consuming and we cannot reached all the areas due to security reasons.

Remote sensing is good and fast method as compare with above both methods, for large scale area.

We can choose the different method as per size of area. For example, optical sensor, airborne laser scanning (ALS), or synthetic aperture radar (SAR). Air imagery with both spaces and individual spatial resolutions has been used to find the wind-throw.

“Study on wind-throws were obtained by Elatawneh et al. [6] and Einzmann et al. [7] based on Rapideye imagery with a spatial resolution of 5 m and change detection techniques.”

A general critical reduction of optical imagery is dependent on its daylight and cloud-free conditions, which is not expected to occur immediately after the occurrence of a storm. Airborne laser scanning data is based on approaches using promising and even more beneficial results. A significant benefit of ALS data is the feasibility of detecting single wind through trees.

Synthetic Aperture Radar (SAR) is good approach to find the affected area. It is good method and works well in daylight and cloud cover. Because resultant is good of SAR data, so nowadays uses is more. SAR backscatter is depend on object and some sensor property, such as frequency: X-, C-, L-band. SAR data will works horizontal as well as vertical, For example co-polarized horizontal-horizontal (HH) And Vertical-Vertical (VV), and cross-polarized vertical-horizontal (VH) and horizontal-vertical (HV). C-band, is used for high frequency and L-band is used for low frequency.

The effect of polarization is slightly different, but cross-polarized (VH or HV) backscatter usually shows high relation with biomass compared to co-polarized channels. Object properties that can affect backscatter from forested areas and we can divided into two categories.

In the first category, wood temperature is most important. The second, will depend size, orientation, and the overall pattern of the trees as a whole, but their branches and leaves [13].

“Apart from this, Ahern et al. [14] and Rüetschi et al. [15] reported a different effect of phonological changes in the leaves on backscatter in wide forests.”

In October 2017 and March 2018, Hurricane happened in many places in the Czech Republic, especially in Moravia and North Bohemia. The Czech Republic faced very differently frosty morning on 17 March 2018. Šerák was very cold, where the recorded temperature was 17.9 degrees Celsius below Zero. The country is also troubled by heavy winds, especially in Moravia and North Bohemia. In the evening, power lines failed in fifteen places mainly in Moravia and East Bohemia. There were about two thousand houses without electricity. Additionally, due to the sudden cooling before start of the spring, the temperature

recodes statistics to 35 places. In addition to the Labská bouda chalet, the Jeseník, Rokytnice v Orlických Horách, Ostrava-Poruba, Žabovřesky, and Brno Libuš stations report overcoming the previous low temperatures (Ceska televize, 2018).

For this project, we choose the Jeseník area, focusing on the March 2018 hurricane. For this area, we have SAR images with VH channel. The test images about vegetation and leaves. We are following below steps to get shapefile with disaster area.

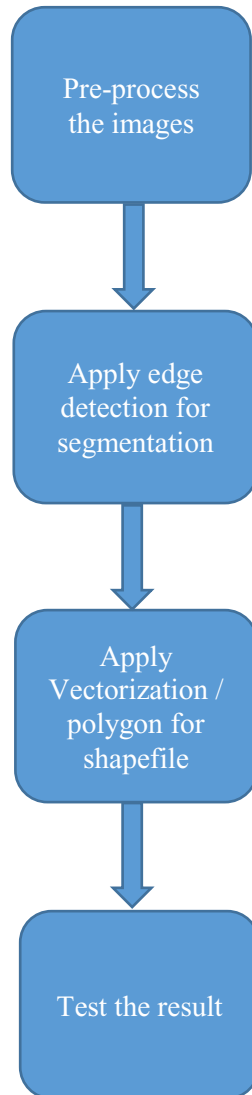


Figure 1: Process flow of the project

- Data pre-processing
- Forest Segmentation
- Vectorization / Polygon Shapefile
- Testing Result

In the First step, data preprocessing, we are detecting the difference between pre and post-hurricane images. In this step input file is GRD images and we are getting output in tiff format.

In the second step, the output of the first step we are taking as an input for this step. In this step are finding segmentation of changes which happened due to the storm. Before finding Segmentation, we are removing noise and doing thresholding of data.

In the third step, we are getting shapefile for the segmented image. The final output is a shapefile format.

In the last step, we are checking the result with Ing. Marek Mlcousek on-site verification mapping.

## 2 Study Area and Data

### 2.1 Study Area

In this project, the Jeseniky area in the Czech Republic is selected as a test area (Figure 2). The Jeseniky Mountain is unique for its rich prosperity, which is primarily due to the elevation and polymorphism of the landscape; Climate and oil structure are other important factors. Most of the areas are in cold regions, and the mountain ranges are the coldest in the country. At the study area, the average annual temperature is 0.9°C. [18]. 17 March 2018, due to heavy wind in the Jeseniky area, a lot of forest area was damaged, and due to the fallen trees, power supply was failed. Many houses were without power. The temperature reached minus 17 degrees.

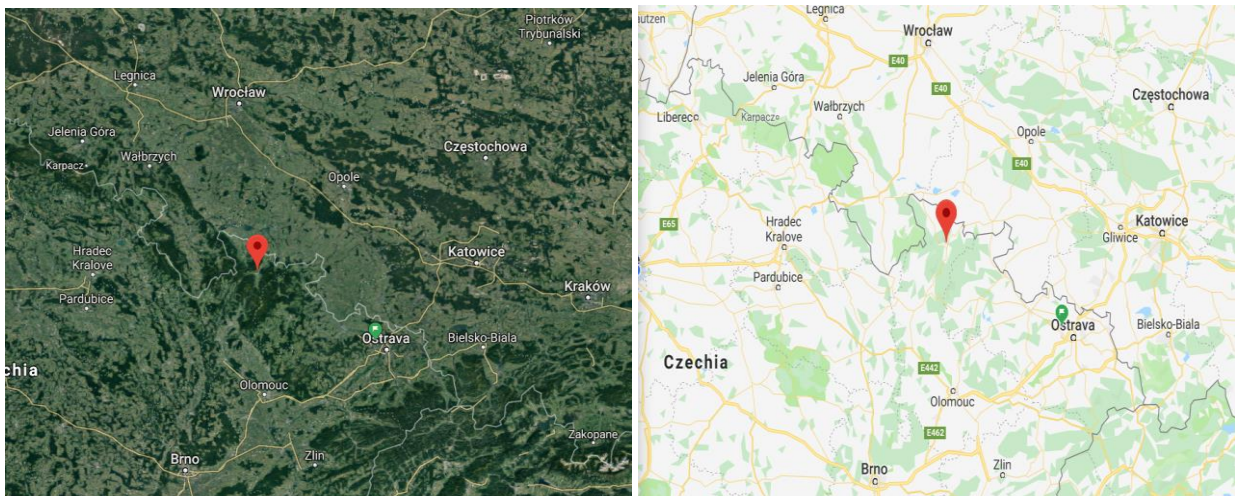


Figure 2 Study area, Jeseniky Area, Czech Republic (Google Map)

## 2.2 Data



Figure 3: Descending Pass Orbits 22 and 124



Figure 4: Ascending pass orbits 73 and 175

Sentinel-1 was launched on April 3, 2014. It was fully operational and provided data on a regular basis. The data was acquired in every twelve days from Sentinel [19]. To detect wind throws, we used Sentinel-1 C-band SAR data acquired at VH polarization. Ten S-1 acquisitions before the event and ten after the storm were used for study areas. For the study area, we chose 10 pre and post-storm VH images, respectively. Both orbits included S-1 acquisitions from both ascending and descending pass directions. This can also be stated for the pre and post-storm timespans of the study area. We used a maximum of ten acquisitions after the storm event, as we decided that further latency (more than four weeks) would not conform to the fast response framework and would increase the probability of contamination from already started clean-up in the forest. Moreover, after approx. Four weeks, the chances of being able to acquire cloud-free optical imagery, which can offer spatial resolution and spectral information superior to SAR data, substantially increase.



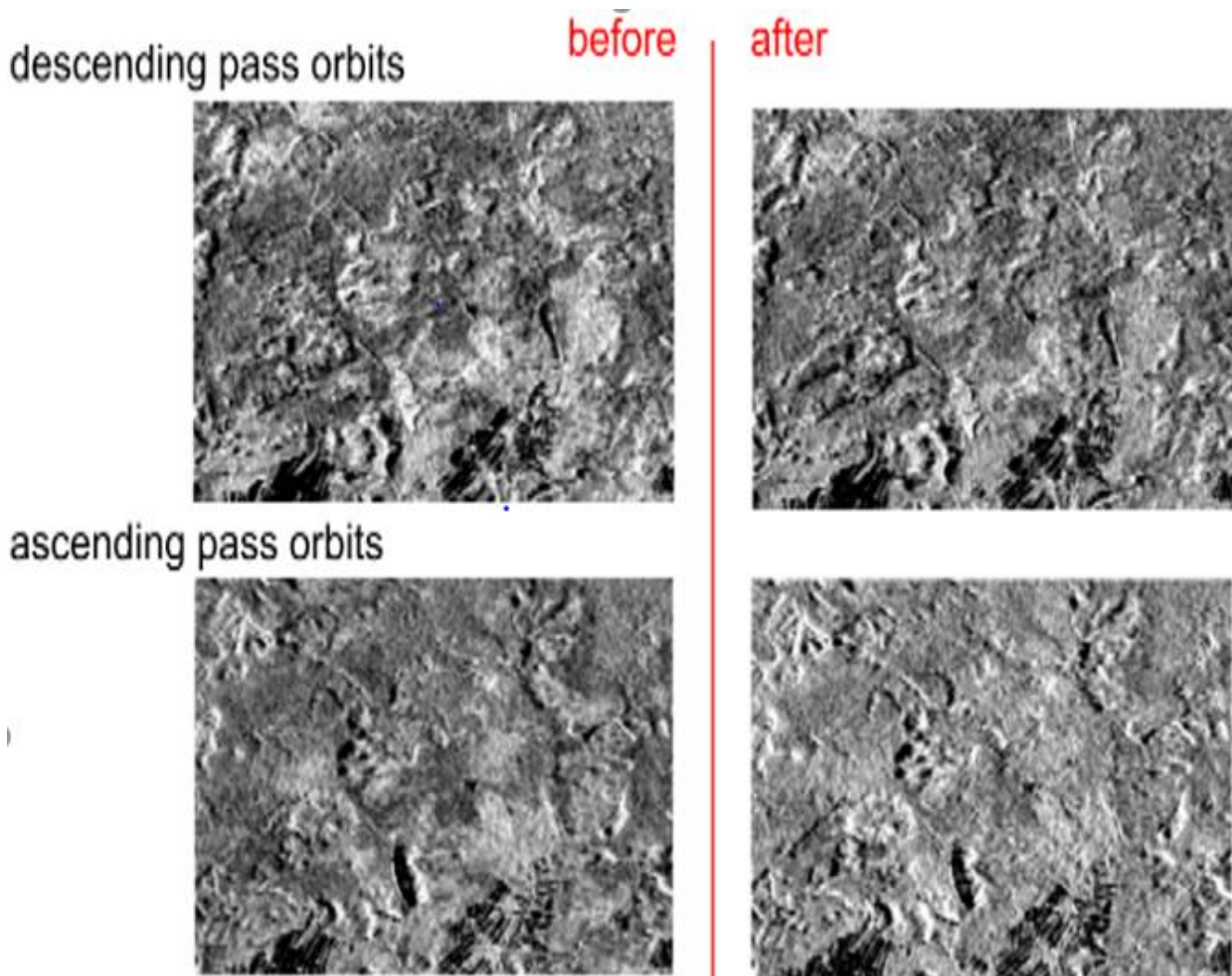


Figure 5: Average images from SENTINEL-1 images: before and after the hurricane for both pass orbits

For this project, Sentinel-1 images dated between January 17, 2018, and May 17, 2018, were used as basic data to determine fields which were affected by the tornado. These images were demonstrated in Figure 5. In Figure 5 first row for average VH intensity maps from 2 descending passes - averages of 10 Sentinel-1 images per pass before and 10 images per pass after the wind calamity and second row: average VH intensity maps from 2 ascending passes - averages of 10 Sentinel-1 images per pass before and 10 images per pass after the wind calamity

The SAR data were obtained in Interferometric Wide Swath Mode (IW) and GRD file format. Information about used SAR images is provided in Table 1.



<b>Date range:</b>	17.01.2018 (pre-disaster) 17.05.2018 (post-disaster)
<b>Instrument:</b>	SAR-C
<b>Polarisation:</b>	VH
<b>Pass Orbits Direction:</b>	Descending Ascending
<b>No. of Images Used</b>	10 Images(Pre-disaster) 10 Images(Post-disaster)

Table 1: Features of Sentinel-1 Data

## 3 Tools and libraries used

### 3.1 Python

Python is a powerful scripting language and is very useful for solving statistical problems involving machine learning algorithms. It has various utility functions which help in pre-processing. Processing is fast, and it is supported on almost all platforms. Integration with C++ and other image libraries is very easy, and it has in-built functions and libraries to store and manipulate data of all types. It provides the pandas and numpy framework which helps in manipulation of data as per our need. A good feature set can be created using the numpy arrays which can have n-dimensional data.

### 3.2 Anaconda

Anaconda is an open-source tool. It is the easiest way to work in Python /R. It is available for all the operating system like Linux and Windows. It has many features [20].

- Easy to download Python / R package.
- Manage environments with Conda and manage libraries.
- Data performance with Dask, NumPy, pandas, and Numba and analyze with scalability.
- Show results with Matplotlib, Datashader, Bokeh, and Holoviews.
- Scikit-learn, TensorFlow, and Theano are useful for developing and train machine learning and deep learning algorithm.

### 3.3 Jupyter Notebook

Jupyter Notebook is an integrated development environment. We can integrate with python very easily, and all the libraries can be used in our Implementation. It is interactive, although some complex computations require time to complete. Plots and images are displayed instantly. Most of the libraries like Dlib, OpenCV, Scikit-learn can be used easily.

### 3.4 OS

The OS module in Python gives a method for utilizing the working framework subordinate usefulness. The capacities that the OS module furnishes enables you to interface with the hidden working framework that Python is running on – be that Windows, Mac or Linux. We can discover vital data about your area or about the procedure.

### 3.5 OpenCV-Python

OpenCV-Python can be defined as a library of Python bindings that was fundamentally designed to solve computer vision problems. Python was started by Guido van Rossum for general purpose programming, and it became very popular in a short period of time due to its code readability and simplicity. Python provides the users to implement their ideas in decreased lines of code without affecting the readability. Python is slower compared to C/C++, but python can be used with C/C++ as an extension. As a result, a

computationally intensive code can be written in C/C++, and it can be made into Python wrappers which can be used as Python modules. It gives us two benefits: 1st, the code is fast as the C/C++ original version code (As it's the original C++ code running in the background). 2nd, Python is easier to code compared to C/C++. OpenCV-Python is a wrapped Python for the original OpenCV C++ implementation. NumPy is made used in OpenCV-Python, where library for numerical operations is highly optimized with MATLAB style syntax. To and from NumPy arrays all OpenCV array structures are converted. This makes it easier to integrate with other libraries that make use of NumPy such as SciPy and Matplotlib. OpenCV is useful for image processing.

### 3.6 NumPy

The fundamental package for computing numbers with Python is this. It contains many things:

- A powerful N-dimensional array object.
- For broadcasting functions, we can use.
- Tools available for integrating C/C++ and FORTRAN language.
- Useful linear in Algebra, Fourier transforms, and in random number problems.

NumPy can also be used with N-dimensional array. We can define different data-types. This is quickly integrated with a large type of databases.

### 3.7 Matplotlib Tutorial: Python Plotting

Humans are visual creatures, to understand things better we need to see things in a visualized manner. However, the steps to present analyses, insights or results may be a tough task, we might get an idea where to begin, or we might have a predefined format in mind, but still questions may arise like, "Is this the proper way to visualize the insights that we want to bring to the audience?"

Matplotlib is a library used for plotting in Python programming language, and NumPy is the numerical extension of Python. Using general-purpose GUI toolkits like Tkinter, wxPython, Qt or GTK+ it provides object-oriented API for embedding plots into applications. To closely resemble MATLAB there is a procedural PyLab interface based on a state machine like (OpenGL), though its use is discouraged. SciPy makes use of Matplotlib.

### 3.8 GDAL/OGR in Python

This Python bundle and expansions are various apparatuses for programming and controlling the GDAL\_ Geospatial Information Deliberation Library. It is two libraries - GDAL for controlling geospatial raster information and OGR for controlling geospatial vector information - yet we allude to the whole bundle as the GDAL library for the motivations behind this record.

The GDAL venture (principally Even Rouault) keeps up SWIG created Python ties for GDAL and OGR. As a rule, the classes and methods, for the most part, coordinate those of the GDAL and OGR C++ classes. There is no Python explicit reference documentation, yet the 'GDAL Programming interface Tutorial'\_ adds Python models.

### **3.9 SNAP**

The SNAP tool is used for Earth Observation processing and analysis due to the following technological innovations: Extensibility, Portability, Modular Rich Client Platform, Generic EO Data Abstraction, Tiled Memory Management, and a Graph Processing Framework.

## 4 Methodology

A static approach for forest segmentation using image processing is used in this project. The focus is on extracting segment and vectorization, using python, image processing libraries and using Geospatial Data Abstraction (GDAL) Library. Our implementation is divided below parts.

- Data pre-processing
- Forest Segmentation
- Vectorization / Polygon Shapefile
- Testing Result

### 4.1 Data pre-processing

Generally, this implementation takes GRD images before and after the event from both ascending and descending Sentinel-1 tracks, we did some preprocessing and prepared averages of images before the event and after the event. Then it merges both descending and ascending average images together (Figure 6) and finally it does a PCA analysis, choosing component 3 image is the most effective for identification of tiny radiometric changes due to wind-fallen trees (**Error! Reference source not found.**). We performed the following steps:

- Pre-step
- Averaging descending and ascending tracks for before and after the event
- Merging the before and after images

#### 4.1.1 Pre-step

Download the Sentinel-1 GRD files: In this step, we are downloading the data. We used the openSARTool kit and SNAP tool. For downloading the data we are using two parameters one is a start date, and another one is the end date, around ten images before and ten after the event, in this case between 2018-01-17 and 2018-05-17 because the event happened on 17th March 2018 and revisit time of Sentinel-1 is six days. This download step was performed twice because we are using both ascending and descending passes. After downloading the data, we applied some SNAP filtering algorithm:

1. **Calibrating the data:** Perform radiometric calibration to get VH bands.



Figure 6: Calibration Process

2. **Radiometric Terrain Flattening:** Perform radiometric calibration for removal of topography related signal.
3. **Multi-temporal Speckle Filter:** Perform a multi-temporal filter to reduce the salt and pepper noise. We used Refined Lee filter because it will preserve the edges.

Below are lines of code which we used for this step. This code was reused to our purpose.[25]

```
In [ ]: # standard Libs
import os, glob

# ost Libs
from ost.s1 import refine, grd2ardBatch, ts, grd2ard

# the main project directory
prjDir = '/home/wad0002/DATA/processing/2018/Jesenik_windfall/asc'

# this is where we downloaded the original scenes
dwnDir = '{}/download'.format(prjDir)
# this folder will be used for the inventory shape files
invDir = '{}/inventory'.format(prjDir)
# this folder will be used for the processed data
prcDir = '{}/processed'.format(prjDir)

tmpDir = '{}tmp'.format(prjDir)
```

After following the Sentinel-1 Data Inventory.ipynb, the data will be downloaded to \$prjDir/download/SAR/GRD. There will be a specific shapefile in invDir:

```
In [ ]: # choose the orbit and pol info with most msoaics based (this comes from the data inventory)
mosaicKey = 'ASCENDING_VH'

invFile = glob.glob('{}/*_{}.shp'.format(invDir, mosaicKey))[0]
```

```
In [ ]: # processing parameters
outResolution = 10 # resolution of the output product in meters
lsMap = True # layover/shadow mask generation (Boolean)
spkFlt = False # speckle filtering on single images(Boolean)

# 3 different output product types are available
# - RTC: radiometrically corrected for slope
# - GTCgamma: geometrically terrain corrected and compensated for ellipsoid curvature
# - GTCsigma: geometrically terrain corrected
outPrdType = 'RTC'

mtSpkFlt = True # multi-temporal speckle filtering on time-series (Boolean)
toDB = True # conversion of backscatter to decibel (recommended)
dType = 'uint16' # conversion of final products to float32, uint8 or uint16 datatype
```

```
In [ ]: #to compute the ARD (analysis-ready data)
grd2ardBatch.grd2ArdBatch(invFile, dwnDir, prcDir, tmpDir, outResolution, outPrdType, lsMap, spkFlt)
```

```
In [ ]: #to prepare data for time series (and perform multitemporal filtering)
grd2ardBatch.ard2Ts(invFile, prcDir, tmpDir, mtSpkFlt, toDB, lsMap, dType)
```

#### 4.1.2 Averaging descending and ascending tracks for before and after the event:

Now data is ready for both ascending and descending passes. For this step, we used Orfeo Toolbox. We did averages before and after the event. **In Error! Reference source not found.** first row, for average VH intensity maps from 2 descending passes - averages of 10 Sentinel-1 images per pass before and ten images per pass after the wind calamity. Second row, average VH intensity maps from 2 ascending passes - averages of 10 Sentinel one image per pass before and ten images per pass after the wind calamity. Third row, image after averaging the both ascending and descending passes we get one image for before and one for after. We removed the residual topographic signal from both sets of images. Below is the line of code which we used to perform this step. Below are lines of code which we used for this step. This code was reused to our purpose.[25]

```

event_date=20180317
path=/home/wad0002/DATA/processing/2018/Jesenik_windfall

for pass in asc desc; do
cd $path/$pass/pro*/*/T*

#renaming to something simpler
for x in `ls *VH*tif`; do mv $x 20`echo $x | cut -d '.' -f2`_VH.tif; done

for pol in VH; do
#getting before and after images
for image in `ls *$pol.tif`; do
datum=`echo $image | cut -d '_' -f1`;
if [ $datum -lt $event_date ]; then
echo $image >> before_$pol.txt;
else
echo $image >> after_$pol.txt;
fi;
done
done

#preparing the averaging: first attempt using gdal_calc failed due to uint16. Therefore using OTB:
for filelist in `ls *.txt`; do
calc="( ";
nofiles=`cat $filelist | wc -l`; n=0; command="";
for file in `cat $filelist`; do
let n=n+1;
calc=$calc"im"$n"b1+";
command=$command" "$file;
done;
calc=`echo $calc | rev | cut -c 2- | rev`" ) / "$nofiles;
echo otbcli_BandMath -il $command -exp \'$calc\' -out `echo $filelist | sed 's/txt/tif/'` uint16 >>run.sh
done

#running the averaging
cat run.sh
sh run.sh
rm run.sh

done
^^^

```

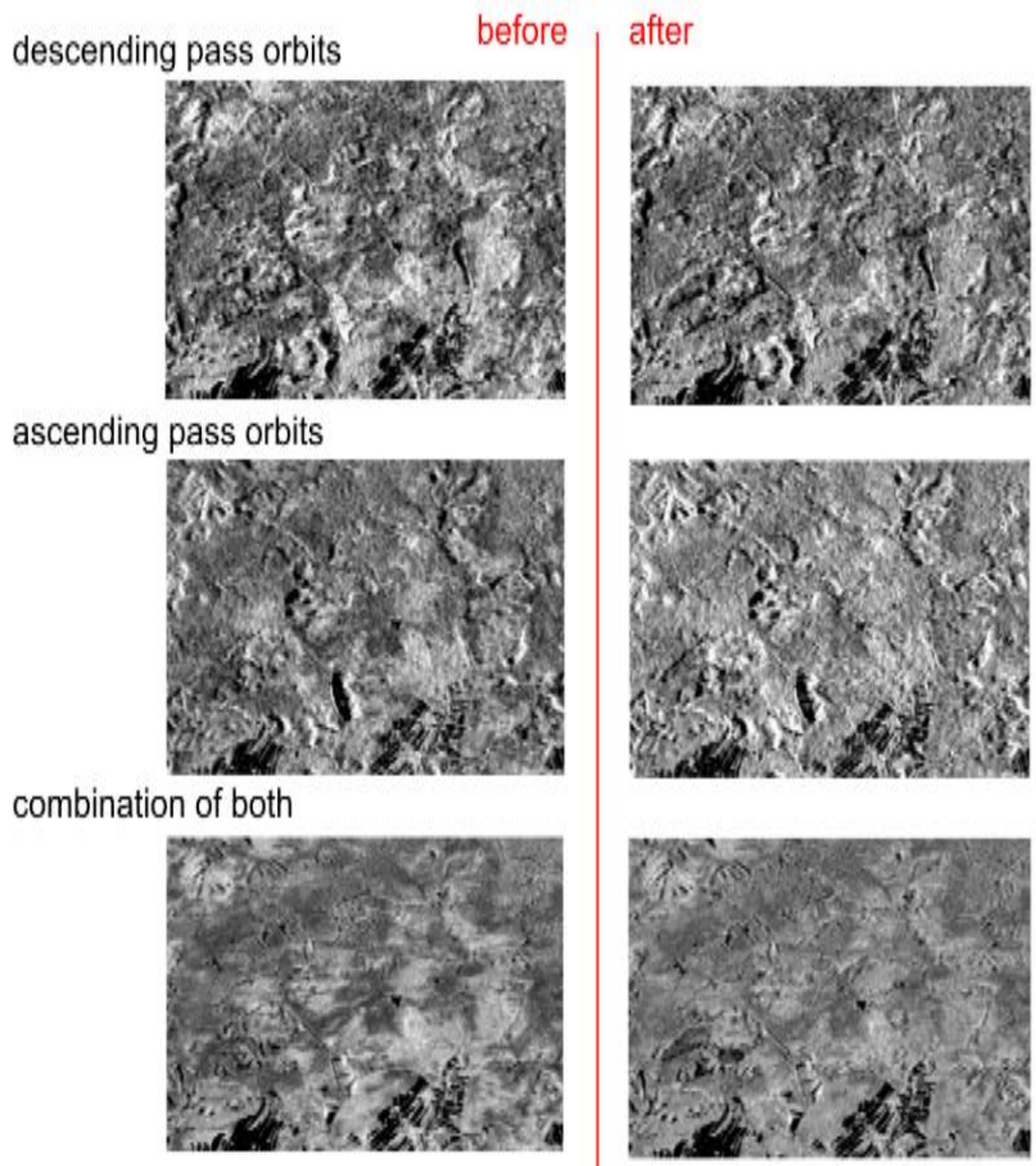


Figure 7: VH Intensity Maps for Sentinel-1 Images



### 4.1.3 Merging the before and after images:

In this step, we used OTB functionalities within the bash shell script. We merged before and after image and tried to get one image with that. For this, we used the Principle Component Analysis (PCA) algorithm with component value 3. Below code, we used for this step and

Figure 8 shows the output of this step. Below are lines of code which we used for this step. This code was reused to our purpose.[25]

```
path=/home/wad0002/DATA/processing/2018/Jesenik_windfall
cd $path
mkdir combined

#merge them and do the average
for pol in VH; do
for time in after before; do
gpt Collocate -Smaster='ls $path/asc/pro*/*/T*/$time'_.$pol.tif' -Slave='ls $path/desc/pro*/*/T*/$time'_.$pol.tif' -t
combined/tmp_$time'_.$pol.dim
otbcli_BandMath -il combined/tmp_$time'_.$pol.data/band_1_M.img combined/tmp_$time'_.$pol.data/band_1_S.img -exp '( im1b1+im2b1
) / 2' -out combined/$time'_.$pol.tif uint16
done
done

#do the PCA analysis
cd combined
gdalbuildvrt -separate /vsistdout/ after_VH.tif before_VH.tif | gdal_translate /vsistdin/ -of ENVI -ot UInt16 merged_all.img
otbcli_DimensionalityReduction -in merged_all.img -out pca.tif -method pca
gdal_translate -b 3 pca.tif changes_forest.tif

##do the same but only with VH
#gdalbuildvrt -separate /vsistdout/ after_VH.tif before_VH.tif | gdal_translate /vsistdin/ -of ENVI -ot UInt16 merged_VH.img
#gpt PCA -PcomponentCount=3 -t pca_VH.dim -Ssource=merged_VH.hdr
#gdal_translate -ot UInt16 pca_VH.data/component_2.img changes_forest_VH.tif
#cd ..
...
---
```

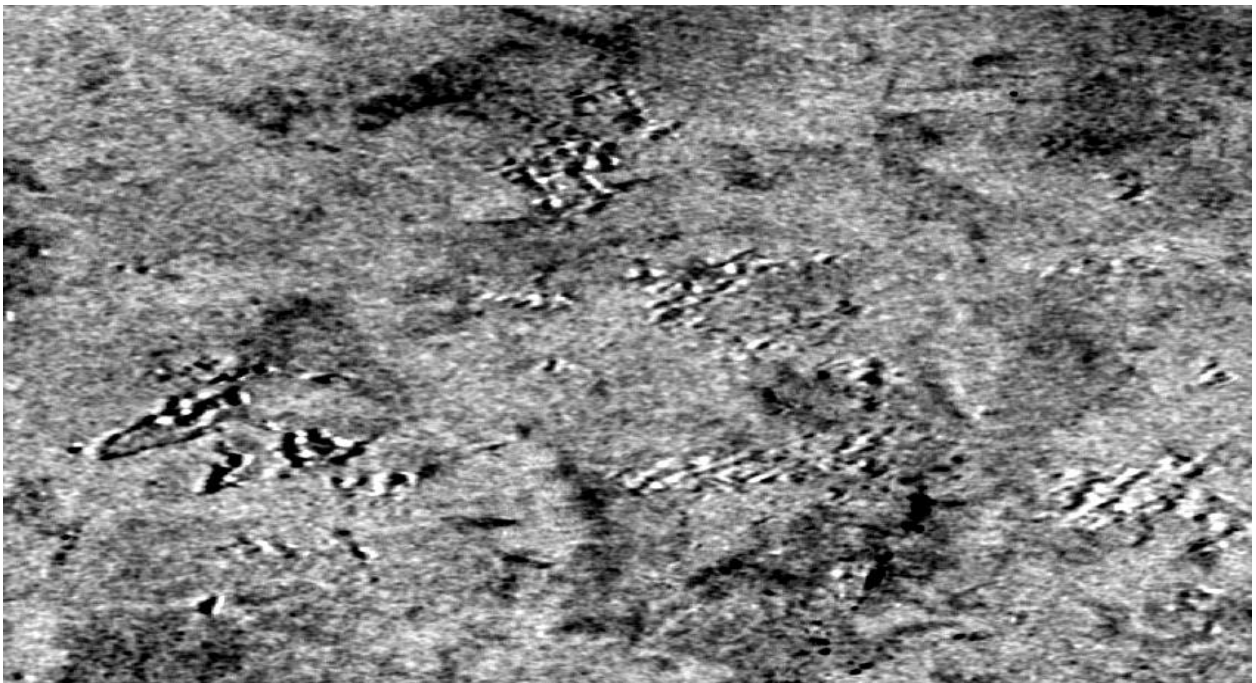


Figure 8: Finding the difference between after and before the event using PCA with component 3

## 4.2 Forest Segmentation

The aim of this step, get the part of the forest which was affected by heavy wind. To find the segment of the forest we used edge detection algorithms. Because input image (Figure 8) was noisy so, before using edge detection algorithm we applied denoising and threshold algorithm on the input image. After detecting the edges we tried to make a smooth shape for segments. We removed white noise and increased the segment area using morphological transformation. For that we performed below steps (Figure 9)

- Pre-step
- Denoising filter
- Thresholding
- Edge Detection
- Morphological Transformation

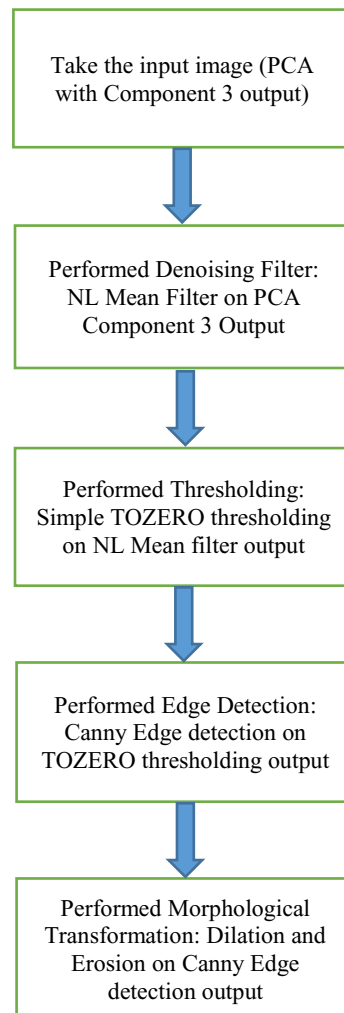


Figure 9: Process flow for the forest segmentation

#### 4.2.1 Pre-step:

In pre-step, we loaded all the required library files and read the input image. After reading the data, we found some basic information for the data. To improve the contrast of the image we used equalization function and compared both the results.

1. **Loading Library:** Before starting the process we imported all the required libraries. Section 3 (Tools and libraries used) is about more details and purpose of about used libraries.

```
In [1]: #Loading the Packages
import cv2
import gdal
import numpy as np
import matplotlib.pyplot as plt
from osgeo import gdal, ogr, osr
import os
```

2. **Read the data:** For reading the input image in a grayscale mode, we used imread function of OpenCV. We used the tiff image as an input. The first parameter of the function defines the name of the source image and the second parameter is defining the scale of the image (grayscale/color). With imshow function, we displayed the input image.

```
In [2]: #Loading Image and converting it to grayscale Image
img = cv2.imread("Input_Img/component_3 (002).tif", 0)
cv2.imshow("/notebooks/Original Image",img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

For checking the pixel size and type of input data we used type and shape function. After used this function we got the image properties. Below output is showing image properties.

```
In [3]: #Type and Shape of Image
Type_of_Image = type(img)
Shape_of_Image = img.shape
print("Image Type: ",Type_of_Image)
print("Image Shape: ",Shape_of_Image)

Image Type: <class 'numpy.ndarray'>
Image Shape: (587, 846)
```

In Figure 10, we plot image histogram. The histogram is a plotting pixel value (ranging from 0 to 255) in X-axis and the corresponding number of pixels in the input data on Y-axis. Figure 10 is showing, from 120 to 170, pixel values, the number of pixels is high, and from 0 to 50 and from 210 to 255, it is low. 0-pixel value defining the black color in the data and 255 is for white color.

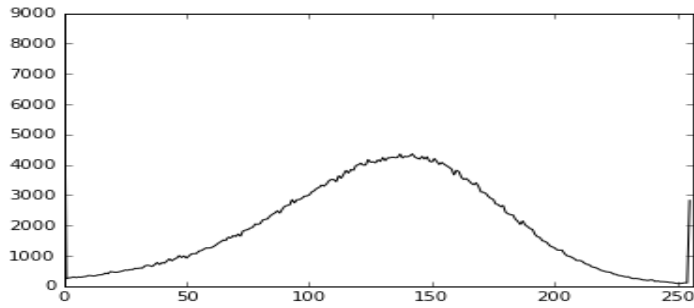


Figure 10: Histogram of input data

3. **Histogram equalization:** To improve the contrast of input image we tried equalization method. Equalization method is useful for satellite images. The appendix (A1. Histogram equalization) is containing more details about this function. In
- 4.
5. Figure 11 right-hand side image showing the output of the used function. As we can easily see that difference between both the images. We tested the edge detection algorithm with both input images.

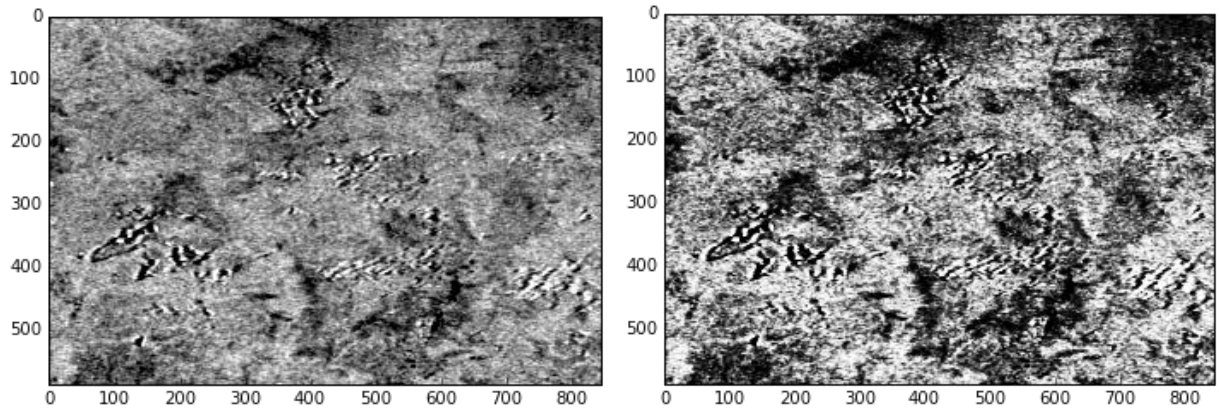


Figure 11: Left-hand side image without equalization and right-hand side image output of histogram equalization.

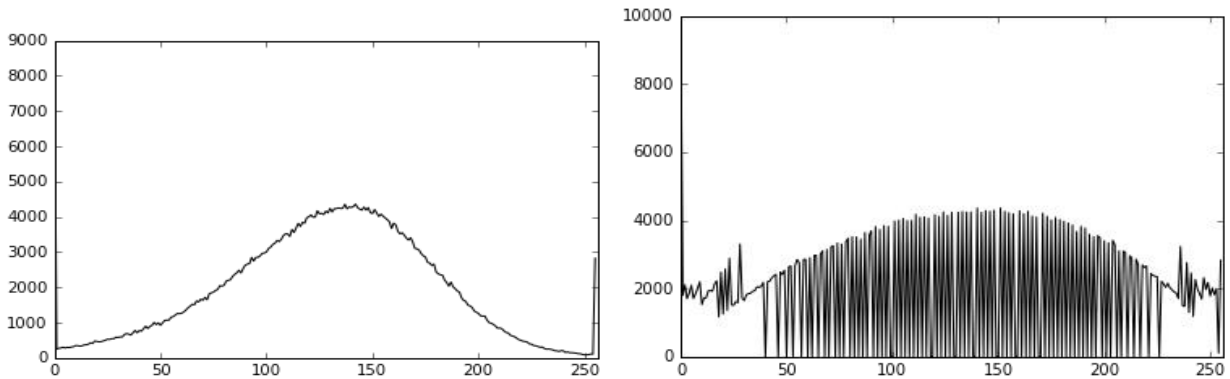


Figure 12: Left-hand side histogram of without equalization and right-hand side histogram for equalization image

Figure 12 is showing the color contrast for every pixel. In the left-hand side, histogram is containing the low black and white contrast as compared to the right-hand histogram. In left-hand side histogram slightly increasing the color contrast from 0 to 4000 and in right-hand side histogram color contrast, for all pixel value lying between 2000 to 3500. It is starting itself from 2000.

#### 4.2.2 Denoising Filter

For removing the noise from data, we used a denoising filter. Many denoising filters are available in python with OpenCV library. We tried Gaussian Blurring, Bilateral Filtering, Median Blurring, and Non-local Means denoising algorithm. After comparing the result for all denoising algorithm, Non-local Mean denoising filter produced a good result. This filter is available for grayscale images as well as colored images. Our input data is contained the grayscale image as an input, so we used a filter for grayscale. Below is the syntax for non-local mean denoising filter for the grayscale image. Details about this filter can be found in the appendix (A2. Non-Local Mean Filter).

**cv2.fastNlMeansDenoising(src[, dst[, h[, templateWindowSize[, searchWindowSize]]]) [21]**

We performed this filter for both the images (We performed this filter for both the images (

Figure 11) with different parameters. Below table (Table 2) is containing some parameters details which we used for finding the de-noised image. First 2 columns contain the input and output image details, and the third column is responsible for removing the noise so for the third column we used different-different values and tried to find out suitable parameter. In First row we used suggested parameter  $h = 10$ ,  $templateWindowSize = 7$  and  $searchWindowSize = 21$  and Figure 13(a) showing the result for same.

<b>h</b>	<b>templateWindowSize</b>	<b>searchWindowSize</b>
10	7	21
20	7	21
40	7	21
20	51	21
10	19	31
20	7	41

Table 2: Non-local means denoising parameter details for without equalization input data (here src =PCA Component 3 output image and dst = none)

As per our test, In Table 2, in the first three rows, we changed only h value and tried to find out the result. We observed after increasing the h value from 20, we are losing our data information and because our data contained a lot of noise, so h =10 is not good value as shown in Figure 13(a). So as per comparison we observed h = 20 is a suitable parameter with our data and in the last three rows, we changed templateWindowSize and searchWindowSize. But as per result and comparison we find out for templateWindowSize = 7 is good and for searchWindowSize = 21 is suitable parameter.

Figure 13(a) shows the output of when h = 20, templateWindowSize = 7 and searchWindowSize = 21 and we observed this output is still containing the noise.

Figure 13(b) shows the output of when h = 20, templateWindowSize = 7 and searchWindowSize = 21 and as per comparison and observation this is the best output for our input data.

Figure 13(c) shows the output of when h = 40, templateWindowSize = 7 and searchWindowSize = 21 then we observed we are missing the actual data. So we avoided this result.

Figure 13(d) shows the output of when h = 20, templateWindowSize=51, searchWindowSize=21. We observed output is same as second row output ( Figure 13(b)).

Figure 13(e) shows the output of when h = 10, templateWindowSize=19, searchWindowSize=31. We observed output is same as first row output ( Figure 13(a)).

Figure 13(f) represents the output of when h = 20, templateWindowSize=7, searchWindowSize=41. We observed output is same as first row output ( Figure 13(b)).

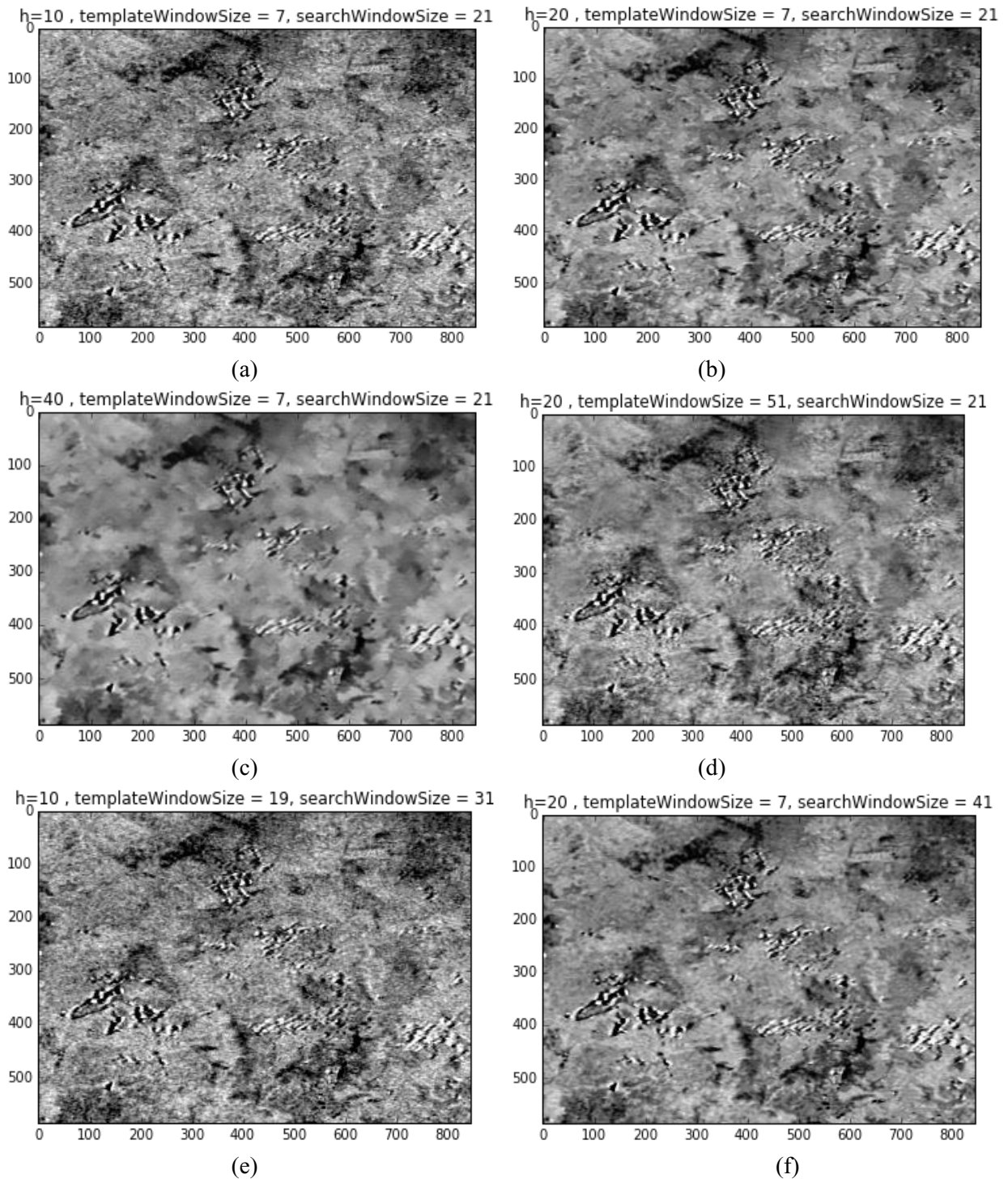


Figure 13: Output of NL Means denoising with different parameter for without equalization input data

The Same test we performed for Histogram equalization input image. But we did not get any good result. After comparison both output, we selected input data without equalization.

After comparison, we get good result when  $h = 20$ , `templateWindowSize = 7` and `searchWindowSize = 21`.

Figure 13(b) shows the output of same.

### 4.2.3 Thresholding:

After denoising filter, we performed thresholding on previous step output. For segmentation, thresholding is the simplest way, for finding the binary image, we used the thresholding on a grayscale image. In python, OpenCV library is providing different types of thresholding methods like Simple Thresholding, Adaptive Thresholding, Otsu's Binarization methods. In our project, we used a simple thresholding method to get the image segment. Below is the syntax of simple thresholding. The appendix (A3. Simple Thresholding) is containing a piece of more information about simple thresholding.

In simple thresholding, the input image should be a grayscale image. Threshold value based on our requirement. Max value should be a max pixel value, i.e. 255 and the last parameter defined which thresholding method we want to use. In our project, we performed different methods for simple thresholding.

`cv2.threshold(input_image,threshold_value,max_value,cv2.thresholding_name)`

threshold_value	max_value	cv2.thresholding_name
10	255	cv.THRESH_BINARY
10	255	cv2.THRESH_BINARY_INV
10	255	cv2.THRESH_TRUNC
10	255	cv2.THRESH_TOZERO
10	255	cv2.THRESH_TOZERO_INV

Table 3: Simple Thresholding Methods with different parameter (here input\_image = NL Mean denoising output image)

Above Table 3, showing the parameter details which we used to get the threshold image and Figure 14 is showing the resultant image for used parameters.



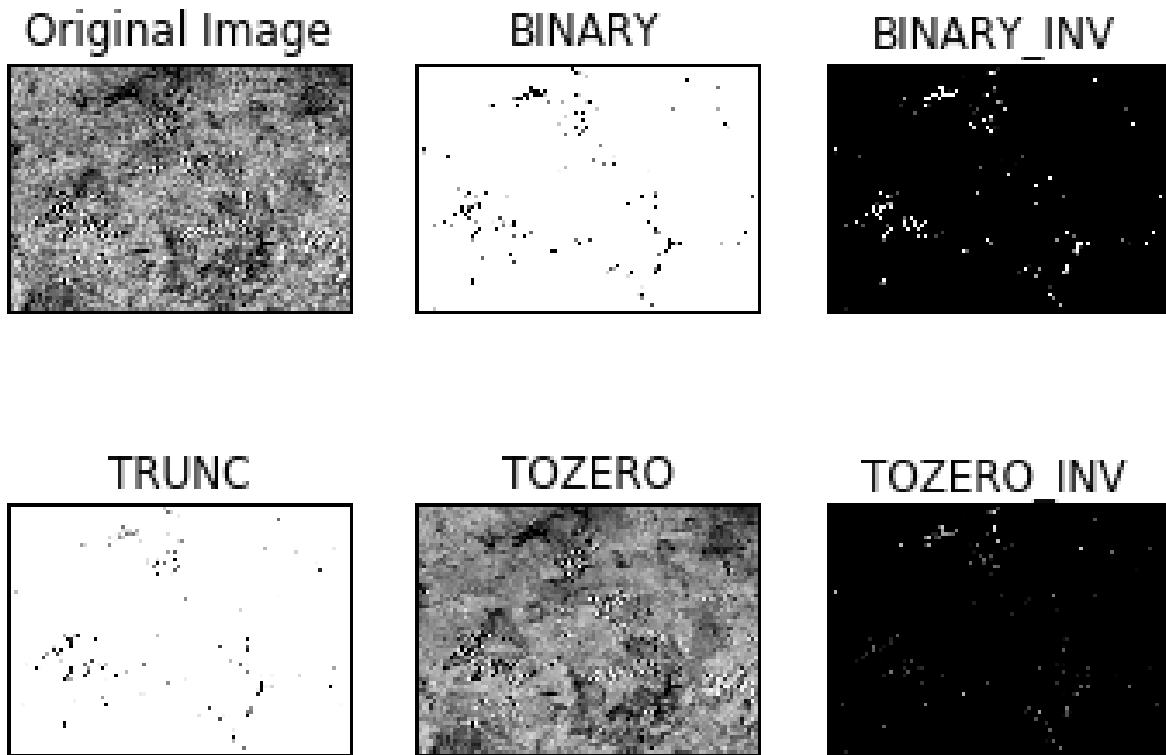


Figure 14: Output for Simple thresholding with different methods

After performed all the different type of thresholding and comparison of result we selected cv2.THRESH\_TOZERO thresholding.

Figure 15 is showing a brief working of a tozero threshold. x and y are representing the pixel size. As per algorithm, if the input image pixel size is greater than the threshold value, then the output of thresholding is the same as the input image. Alternatively, else if the pixel size is smaller than it will replace with black color.

```

if (input_image(x,y) > threshold_value)
    output_image = input_image
else
    output_image = 0

```

Figure 15: working of THRESH\_TOZERO simple thresholding

Table 4 is showing some parameter which we tested for TOZERO thresholding. After compared all the results, we get good when threshold\_value = 10 and max\_value = 255. Figure 16 shows the output for same.

threshold_value	max_value
10	255
5	255
15	200

Table 4 : THRESH\_TOZERO thresholding with different parameter (here input\_image = Denoising output image and thresholding\_name = cv2.THRESH\_TOZERO).

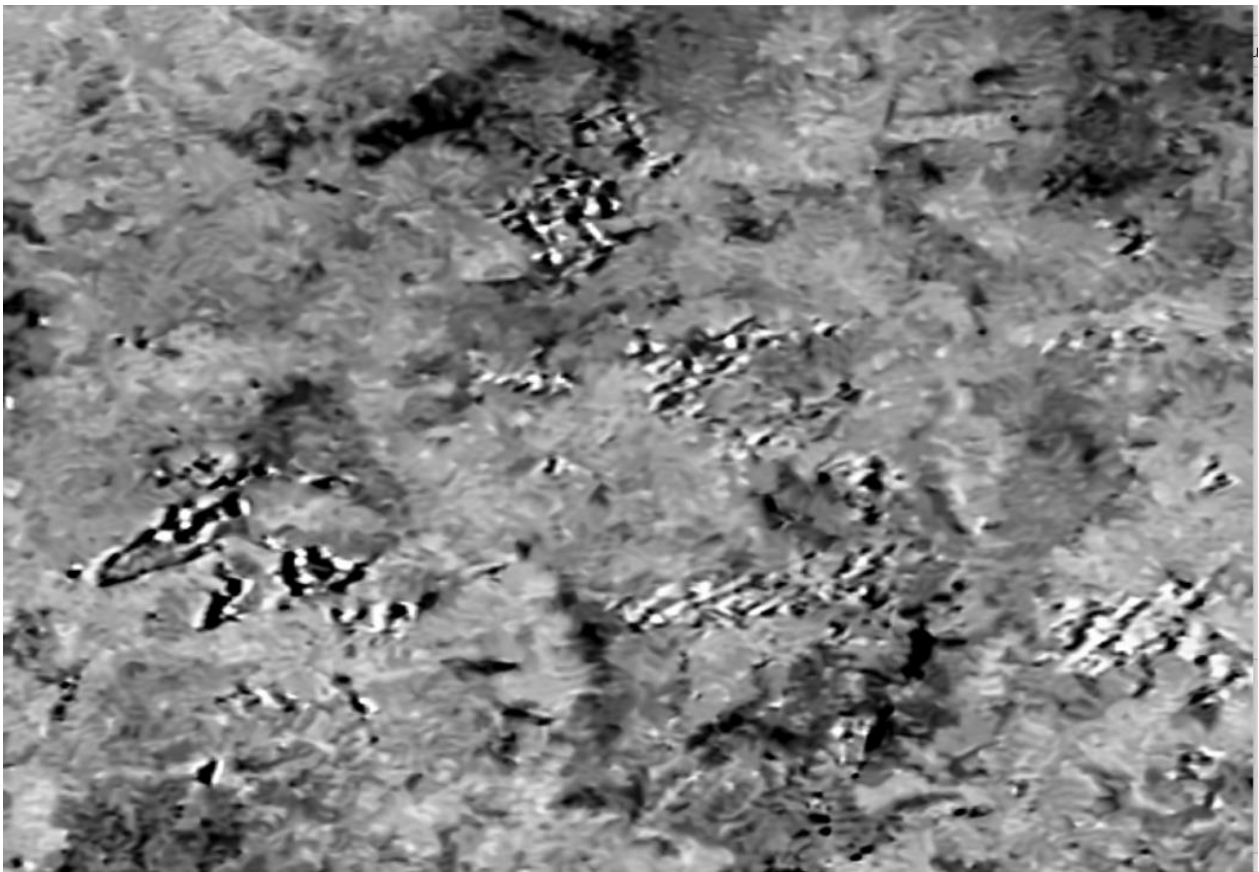


Figure 16: THRESH\_TOZERO output image with threshold\_value = 10 and max\_value = 255

#### 4.2.4 Edge Detection

After finding the segment of the image we performed edge detection algorithm for finding the edges of the area. In python, OpenCV library has many algorithms to detect the edges. We performed Laplacian, Sobelx, Sobely and Canny edge detection methods. Figure 17 is showing the output for the edge detection method. After comparing the result, canny edge detection, detect the part which we expected. However, with the result of the remaining edge detection algorithm, we did not get any good result.

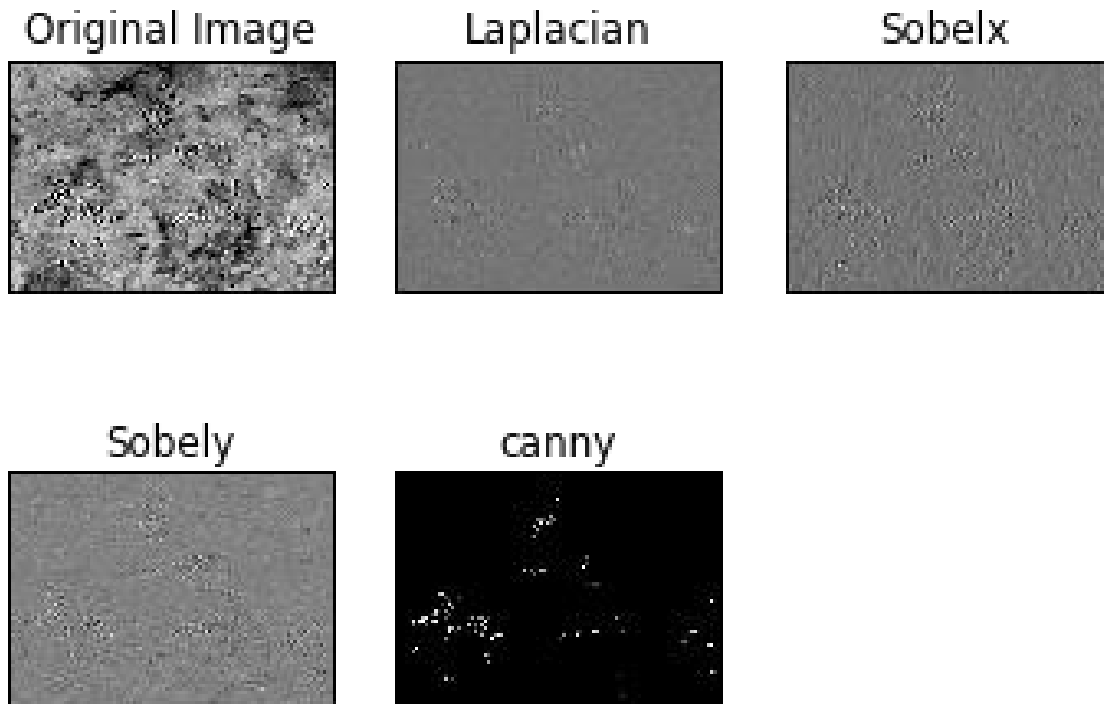


Figure 17: Output for the edge detection algorithm

Canny edge detection works well compared to remaining algorithms. Multiple stages are there to detect the edges. Below is the name for stages: [22]

- Noise Reduction
- Finding Intensity Gradient of the Image
- Non-maximum Suppression
- Hysteresis Thresholding

Below is the syntax for a canny algorithm. For the canny algorithm, we need to pass three parameters. The first parameter is for an input image. The second parameter is showing the minimum value of edges and the third parameter for a maximum number of edges that we want to include in our result. Last parameter responsible for finding gradient magnitude. If it is true, then we need to define the values of magnitude, but by default it is false. In our project, we are not passing L2gradient values. The appendix is containing more information about the canny method.

**cv2.Canny(input\_Imgage,min\_value,max\_value,L2gradient)**

We applied the canny method with different values.

Table 5 is showing some values which we tested on our input. We tested this method with different L2gradient values, but after observing the result, we used default L2gradient. For the experiment, we slowly increased the min\_value parameter, but for max\_value we increased parameter value by a large amount.

<b>min_value</b>	<b>max_value</b>
100	450
150	700
210	950
250	1000

Table 5: Different parameter detail for Canny Edge Detection (here Input\_image = THRESH\_TOZERO output image)

In Figure 18, after getting the result, we compared with input image and we observed with min\_value = 100 and max\_value = 450, getting output is containing many unwanted areas of forest. Because in this test we chose edge value very small, so we decide to increase the number of parameter values. In this test due to the small value of parameter, the output is showing extra area which we can easily notice on top left near to pixel value x=100 and y = 200 and bottom right near to pixel value x = 800 and y = 100. So we performed a new test with different values.

Figure 19, is showing a new output image with changed parameters. This time we choose min\_value = 150 and max\_value = 750 and performed our test. Because our interest to find out the big area as well as a small area of the forest also if something happened at the time of heavy wind. That is why we increased min\_value with 50 and max\_value with 250. After getting the result we compared with the input image and we observed with min\_value = 150 and max\_value = 700, getting output is containing a still unwanted area of forest, so we decide to increase the number of parameter values again. In this test, we can easily see that we are near to result but some area on the top right is unwanted.

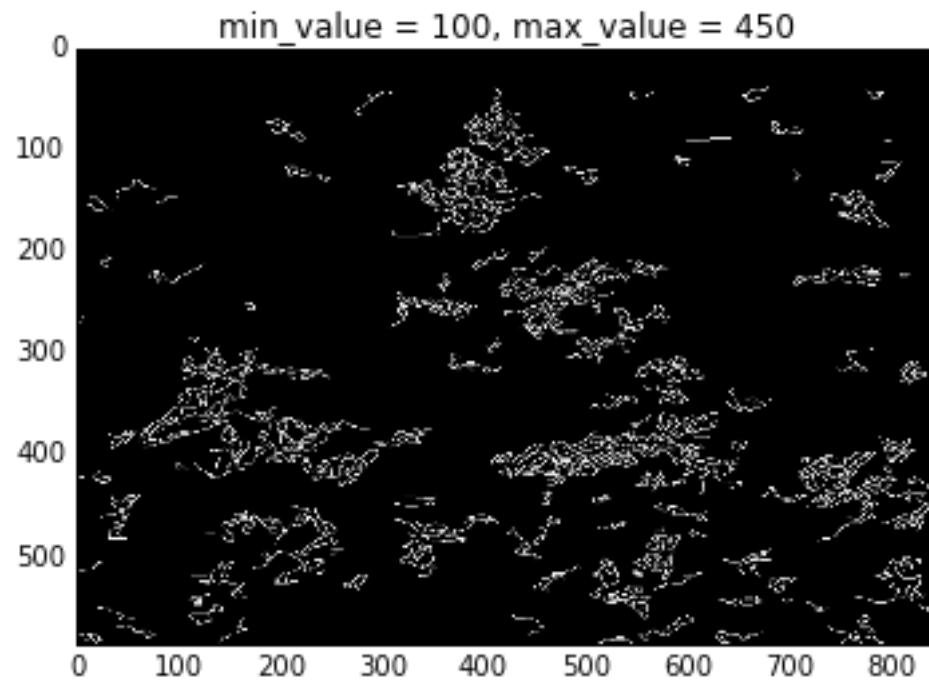


Figure 18: Canny Edge Detection Output for min\_value = 100 and max\_value = 450

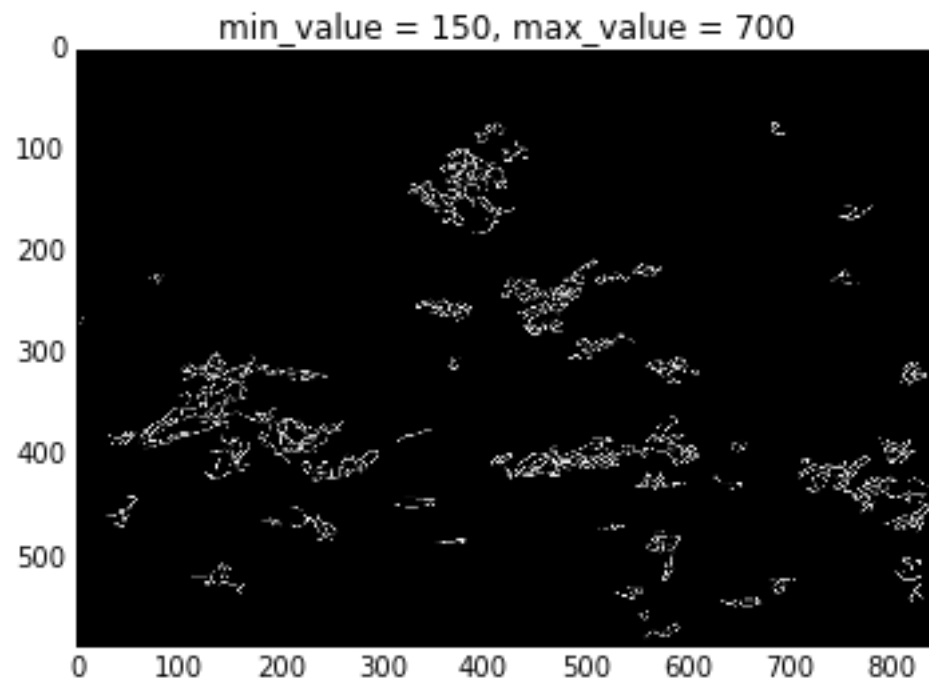


Figure 19: Canny Edge Detection Output for min\_value = 150 and max\_value = 700

After getting a result of Figure 19, we slowly increased both the parameters with the small number of differences and after comparing all results we observed with `min_value = 210` and `max_value = 950` we are getting wanted area of forest. Figure 20, is the output of the same. We observed we are getting a large affected area as well as small area also as per our expectation. Bottom left near pixel value `x = 110` and `y = 450` area is small area and bottom left between `x = 100` and `y = 400` area is largest area of forest. And at last, we performed one new test.

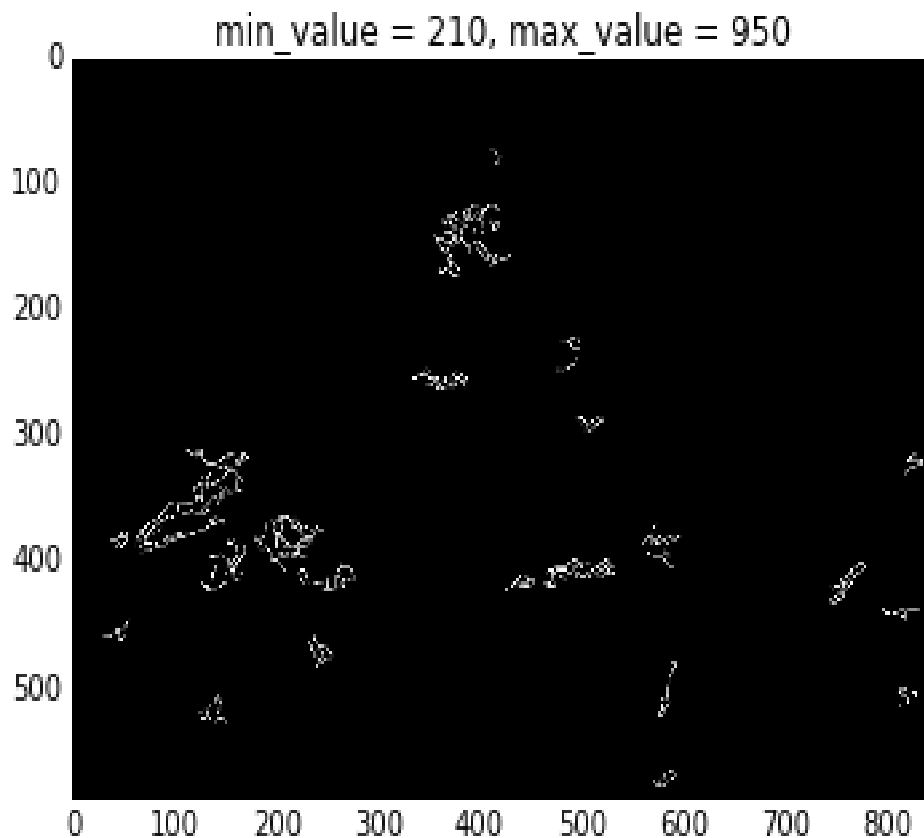


Figure 20: Canny Edge Detection Output for `min_value = 210` and `max_value = 950`

Figure 21, is showing the output of a new test. In this test, we increased `min_value` with 40 and `max_value` with 50. After comparison, we observed we are losing our target area also if now we are increasing the value of our parameter. In top left near to pixel value `x = 400` and `y = 390`, detected area was wanted area. But in a new test we are losing that part.

So as per different result, we decided to test 3 with `main_value = 210` and `max_value = 950` is producing a good result. So for the further test, we selected this value and we moved with this output.

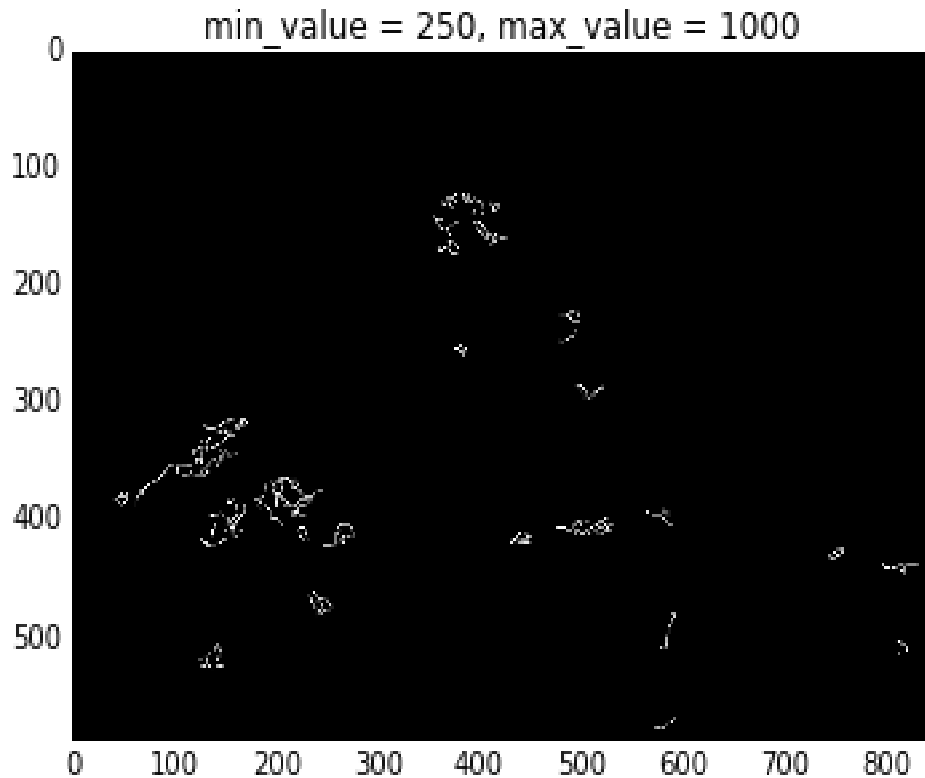


Figure 21: Canny Edge Detection Output for min\_value = 250 and max\_value = 1000

#### 4.2.5 Morphological Transformation

After edge detection method we used morphological transformation for finding the smooth area of detected edges. In python, the OpenCV library provide the morphological transformation. The different operator is available for this transformation like Erosion, Dilation, Opening, Closing, and Morphological Gradient. But in our project, we used 2 operator Erosion and Dilation.

We got the edges but it was not smooth, it was broken, so dilation was useful for joining broken parts of the area and also useful to increase the object area. But our main purpose to join the broken parts of edge and Erosion is useful to remove the white noise from input image [23]. For both the method we need a kernel and as per kernel size, both operators will work. Below is the syntax of dilation and erosion.

**cv.dilate(input\_image,kernel,no\_of\_iterations)**

**cv.erode(input\_image,kernel,no\_of\_iterations)**

For both the operator we need to mention 3 arguments. The first argument is for input data, the second argument we are providing kernel value and size and third argument we are using for no of iterations. In our project, first we are performing dilation on the output of edge detection and after getting the output of dilation, we are performing erosion. For Erosion, the output of dilation is the input of erosion. We performed both operators with multiple iterations. But after result comparison, we are using only one iteration for both operators. We used the different kernel for both the operator.

Table 6 is showing some values for the kernel which we tested in our project. Below is the syntax for the kernel.

**np.ones((x\_size,y\_size),np.uint8)**

Using the above syntax we defined our kernel for both the operator. In python, with Numpy library we can create a kernel with one value. Kernel size depends on x\_size and y\_size. In our project, we created a kernel with (1, 10), (5, 10), (10, 10) and (1, 10) (

Table 6). If we are going to define kernel with (10, 10), then we are generating a matrix with 10 rows and 10 columns with 1 value.

<b>1<sup>st</sup> test</b>		
	x_size	y_size
Kernel value for Dilation	1	10
Kernel value for Erosion	1	10
<b>2<sup>nd</sup> test</b>		
Kernel value for Dilation	5	10
Kernel value for Erosion	5	10
<b>3<sup>rd</sup> test</b>		
Kernel value for Dilation	10	10
Kernel value for Erosion	1	10

Table 6: Morphological Transformation test for Dilation and Erosion with different size of kernel

In the first test we defined (1, 10) kernel, and after getting the result we observed area is broken, so for the best result, we performed next test. Below

Figure 22 is showing the resultant image for dilation and Figure 23, is for erosion

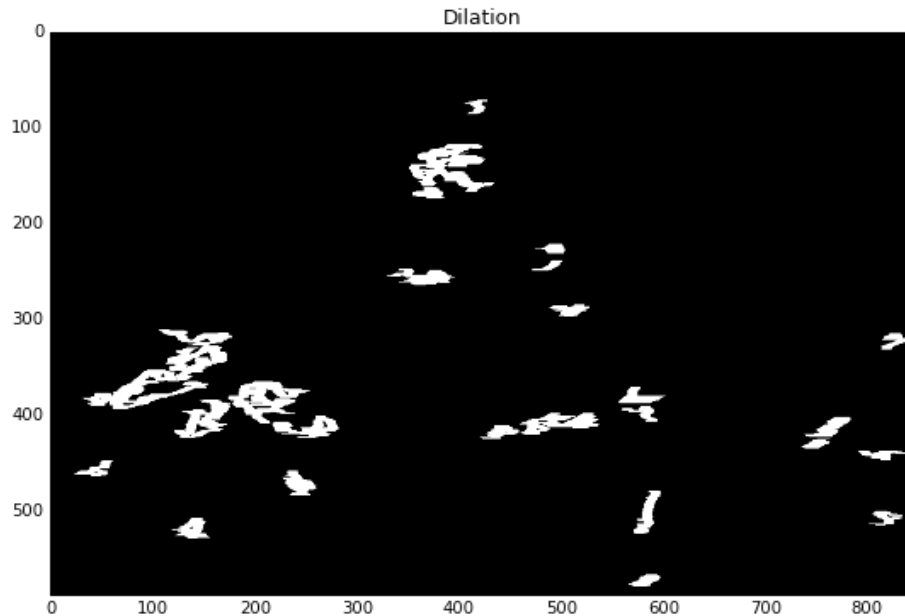




Figure 22: Dilation Output with kernel (1, 10)

In the next test we defined (5, 10) kernel for both operators and after getting the result we observed that area is still not smooth, so for a smooth area, we performed next test. Below Figure 24 is showing the resultant image for dilation and Figure 25, is for erosion.

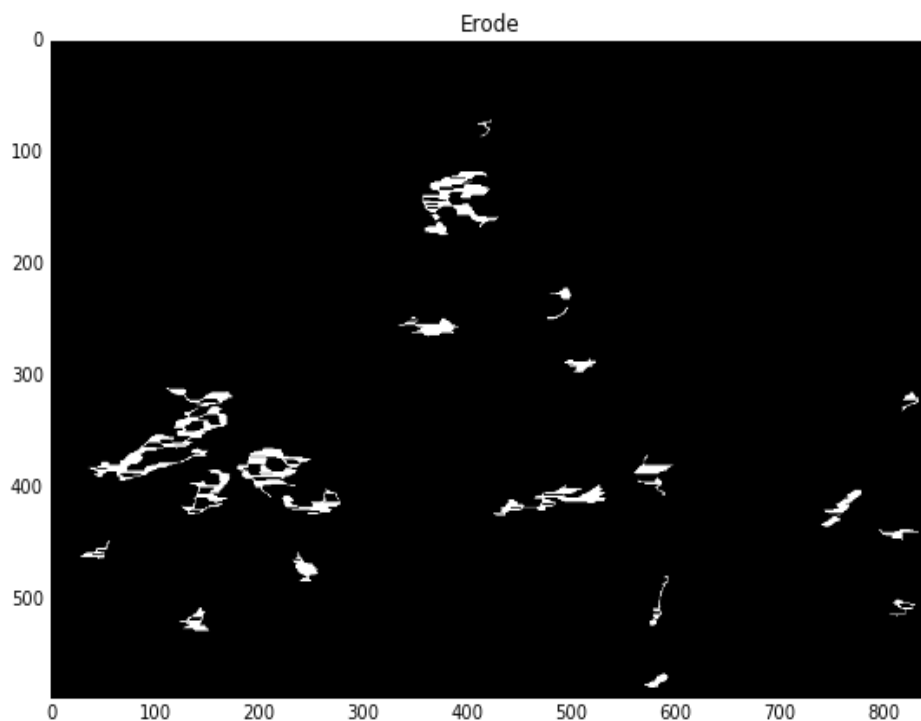


Figure 23: Erosion output with kernel (1, 10)

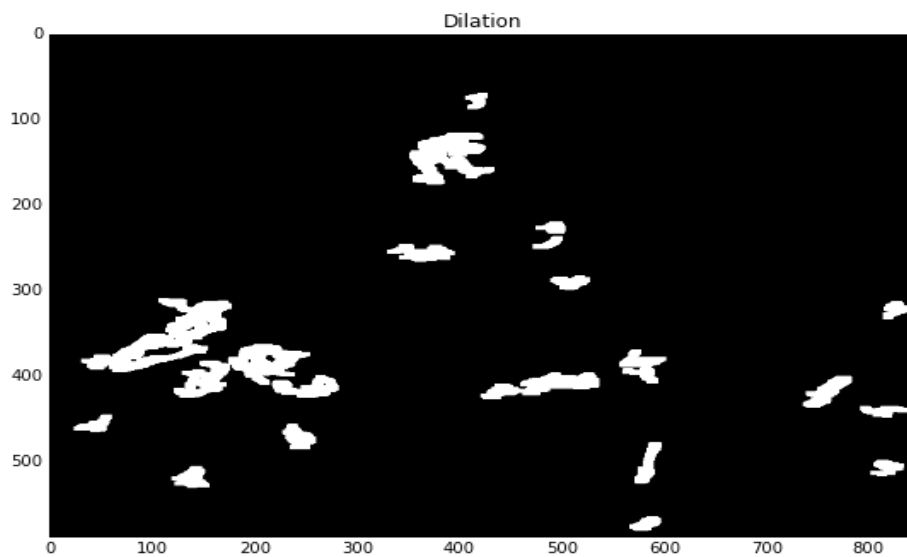


Figure 24: Dilation Output with kernel (5, 10)

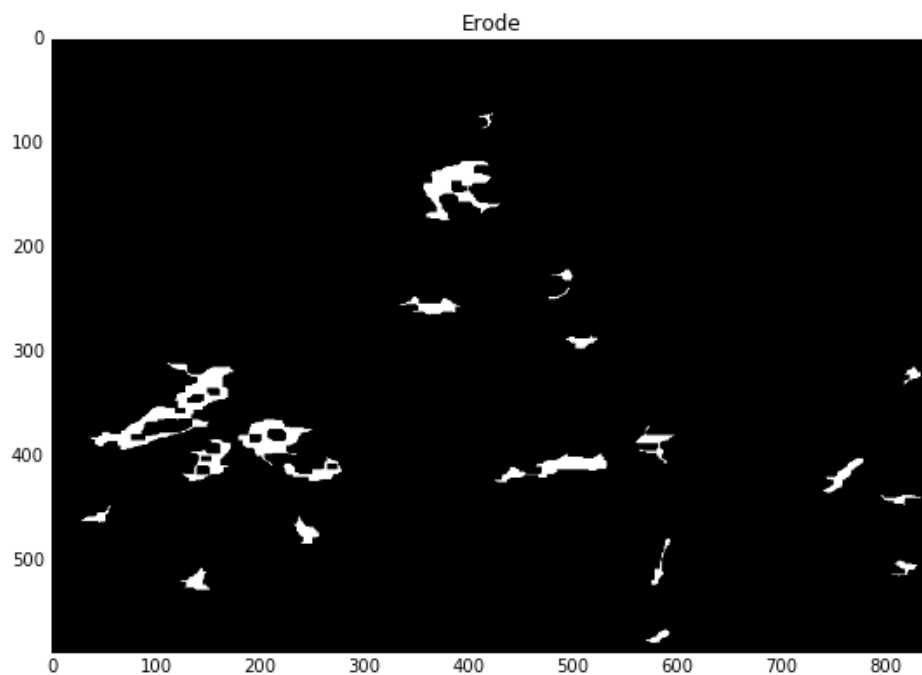


Figure 25: Erosion output with kernel (5, 10)

In the last, test we defined (10, 10) kernel for dilation and (1, 10) kernel for erosion.

After getting the results and comparing with all the previous result we observed this is a good result and we are getting the smooth object. Below

Figure 26 is showing the resultant image for dilation and Figure 27, is for erosion. This step was the last step for segment detection, and Figure 27 is the final result for this detecting the segments of the forest. After this step, we were able to say this area was affected due to heavy wind.

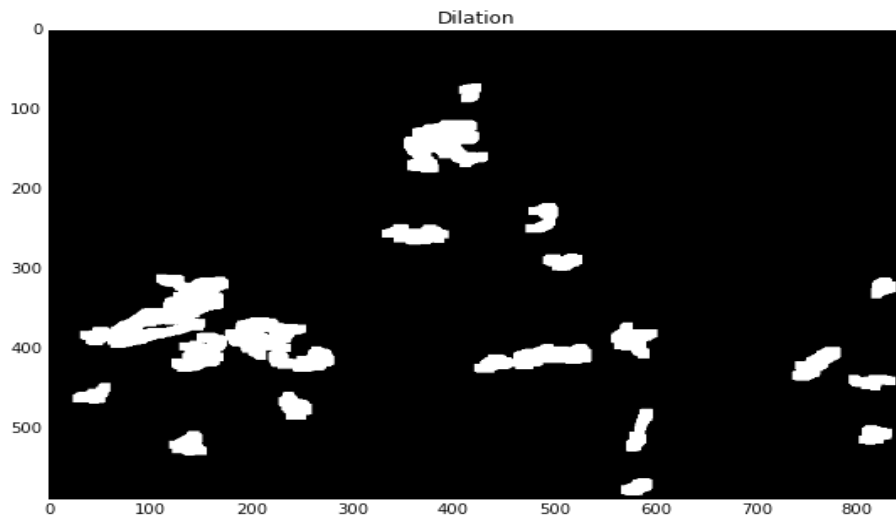


Figure 26: Dilation Output with kernel (10, 10)

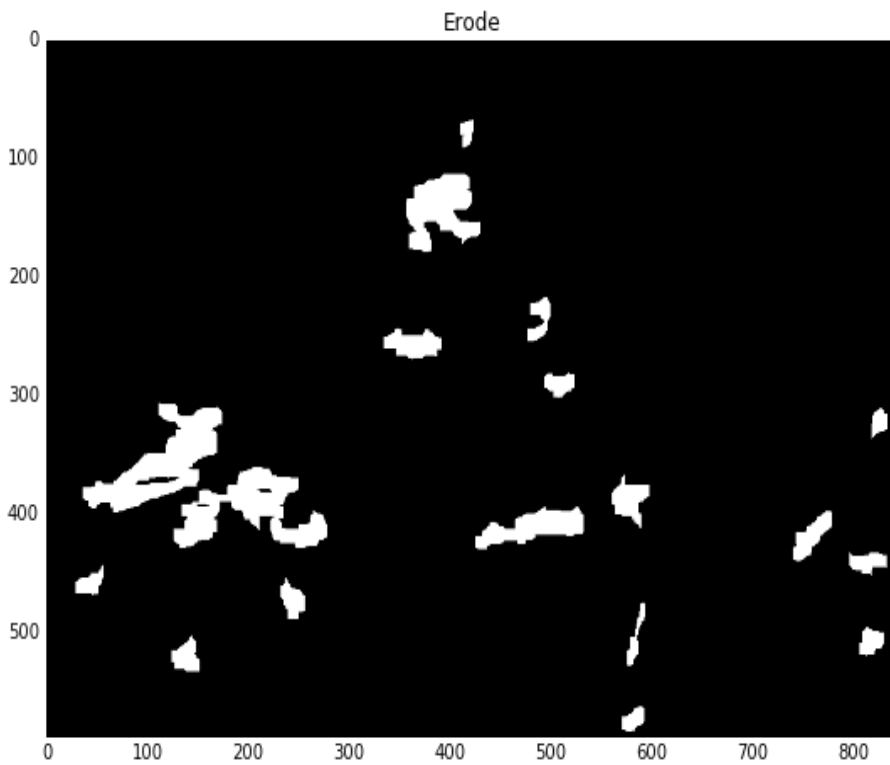


Figure 27: Erosion output with kernel (1, 10)

#### 4.3 Vectorization / Polygon Shape file

The aim of this step is to get the polygon shapefile and visualize the result for previous step output. In this step we are using the python, Geospatial Data Abstraction Library (GDAL) and OGR library. GDAL used for raster data and OGR used for vectorization.

GDAL library is providing many functions, but for our project we used GDAL.Polygonize function. Erosion output is input for this step and after this step, we got the shapefile. Below is the syntax of this method (Figure 28).

```
gdal_polygonize.py [-s] [-nomask] [-mask filename] raster_file [-b band][-q] [-f ogr_format] out_file  
[layer] [fieldname]
```

Figure 28 : Syntax of GDAL\_polygonize method [24]

Below lines of code, we performed to get shapefile from tiff input data. After performing this function we got below four files for output.

- DBF file
- PRJ file
- SHP file
- SHX file

Figure 29, is showing the output of shapefile.

```

In [29]: #Creting Shape files from raster data
def polygonize(rasterTemp,outShp):
    sourceRaster = gdal.Open(rasterTemp)
    band = sourceRaster.GetRasterBand(1)
    driver = ogr.GetDriverByName("ESRI Shapefile")
    # If shapefile already exist, delete it
    if os.path.exists(outShp):
        driver.DeleteDataSource(outShp)
        outDatasource = driver.CreateDataSource(outShp)
    # get proj from raster
    srs = osr.SpatialReference()
    srs.ImportFromWkt( sourceRaster.GetProjectionRef() )
    # create layer with proj
    outLayer = outDatasource.CreateLayer(outShp,srs)
    # Add class column (1,2,..) to shapefile
    newField = ogr.FieldDefn('Class', ogr.OFTInteger)
    outLayer.CreateField(newField)
    gdal.Polygonize(band, None,outLayer, 0,[],callback=None)
    outDatasource.Destroy()
    sourceRaster=None
    band=None
    ioShpFile = ogr.Open(outShp, update = 1)
    lyr = ioShpFile.GetLayerByIndex(0)
    lyr.ResetReading()

    for i in lyr:
        lyr.SetFeature(i)
        # if area is less than inMinSize or if it isn't forest, remove polygon
        if i.GetField('Class')!=1:
            lyr.DeleteFeature(i.GetFID())
    ioShpFile.Destroy()
    return outShp

```



Figure 29: Shape file output

#### 4.4 Testing Result

##### 4.4.1 Test 1

In this test, we performed forest segmentation (4.2) with different parameter for NL denoising filter and compared with final result (Figure 27: Erosion output with kernel (1, 10)).

**Step 1: Denoising filter:** We applied NL Mean filter this following parameters (Table 7) and Figure 30 shows the output for same.

<b>h</b>	<b>templateWindowSize</b>	<b>searchWindowSize</b>
40	7	21

Table 7: Non-local means denoising (here src =PCA Component 3 output image and dst = none)

**Step 2: Thresholding:** We applied Simple THRESH\_TOZERO with following parameter (Table 8) and Figure 31 shows the output for same.

<b>threshold_value</b>	<b>max_value</b>
10	255

Table 8: THRESH\_TOZERO thresholding (here input\_image = Denoising output image and thresholding\_name = cv2.THRESH\_TOZERO).

**Step 3: Edge Detection:** We applied Canny Edge Detection with following parameter (Table 9) and Figure 32 shows the output for same.

<b>min_value</b>	<b>max_value</b>
210	950

Table 9: Canny Edge Detection (here Input\_image = THRESH\_TOZERO output image)

*Step 4: Morphological Transformation: Apply Dilation and Erosion with following kernel (Table 10) and*

*Figure 33 shows the output for Dilation and*

Figure 34 shows output for Erosion operation.

	<b>x_size</b>	<b>y_size</b>
Kernel value for Dilation	10	10
Kernel value for Erosion	1	10

Table 10: Size of kernel for Dilation and Erosion operation

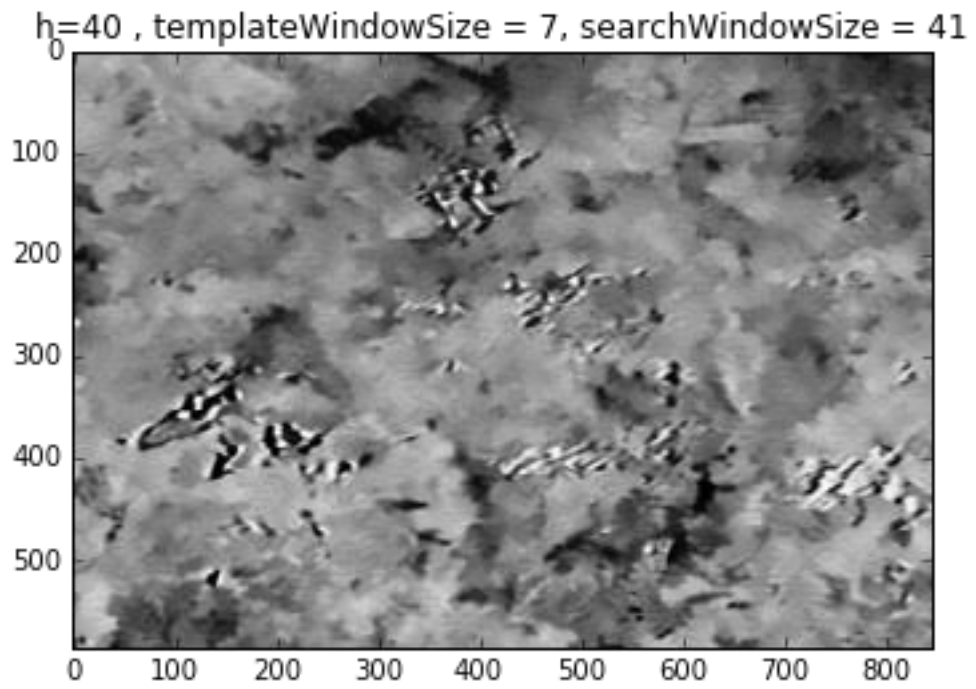


Figure 30: Output of NL Means denoising

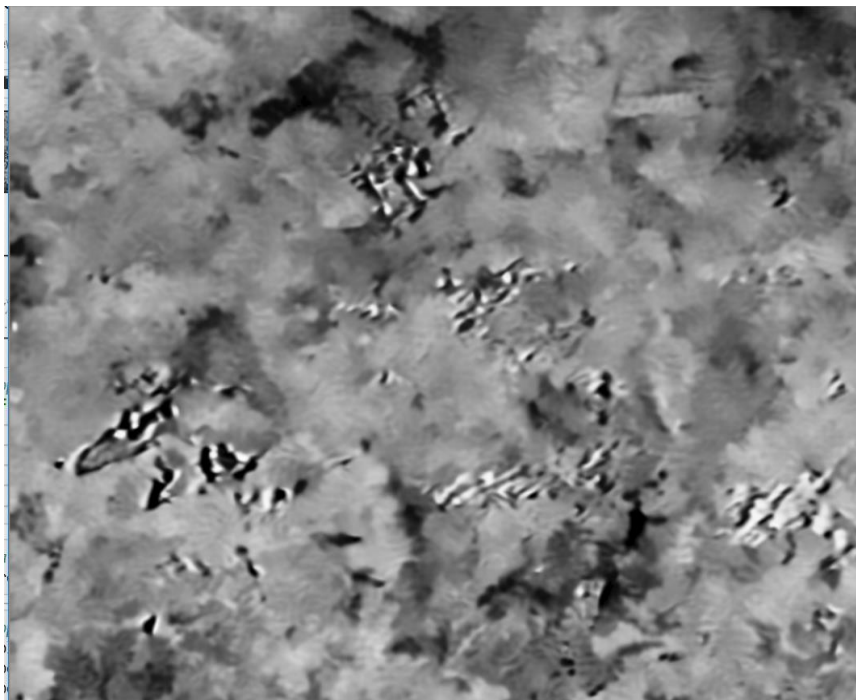


Figure 31: Output of THRESH\_TOZERO thresholding



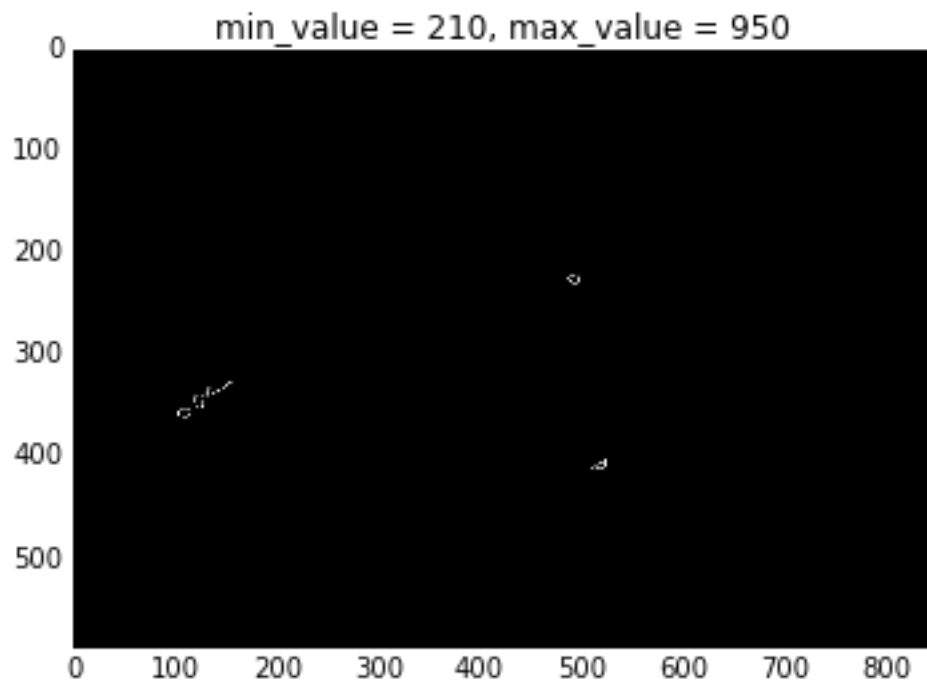


Figure 32: Output of Canny Edge Detection

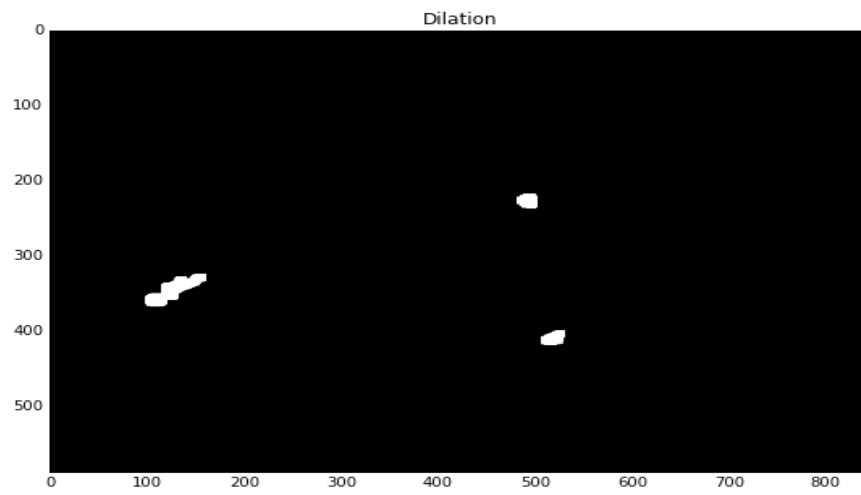


Figure 33: Output for Dilation

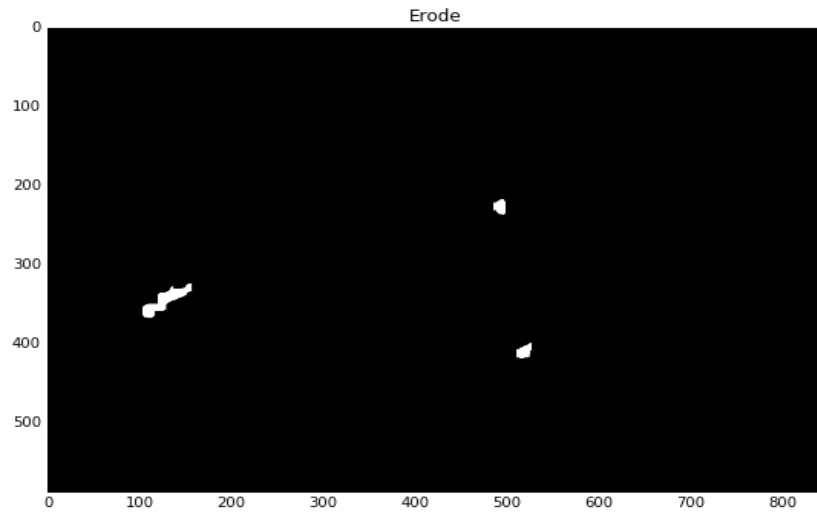


Figure 34: Output for Erosion

#### 4.4.2 Conclusion for Test 1

After getting result from test 1, we compared our final result (Figure 27: Erosion output with kernel (1, 10)) and test1 result (Figure 34: Output for Erosion). We concluded, if we are changing the parameter values then we did not get good result. In Figure 34: Output for Erosion, we can clearly see that forest segments are not good and it is not matching with Figure 35: Result from Ing. Marek Mlcousek Jesenik area.

#### 4.4.3 Test 2

The test aimed to verify our result. We wanted to ensure that which output we get from our implementation, that is correct and not. So for that, we compared our result with Ing. Marek Mlcousek from Frydek-Mistek result. Ing. Marek Mlcousek is working in the forestry department, and he did site verification for two areas and below output image (Figure 35) is showing the two wind-fallen forest segments on Jesenik area, which Marek Mlcousek got. Because this was the actual result, so we compare this two segment with our output, and both segments are present in our result. Figure 36, is showing the result which is matching with the actual result. Figure 30 is showing our result after vectorization of erode output.

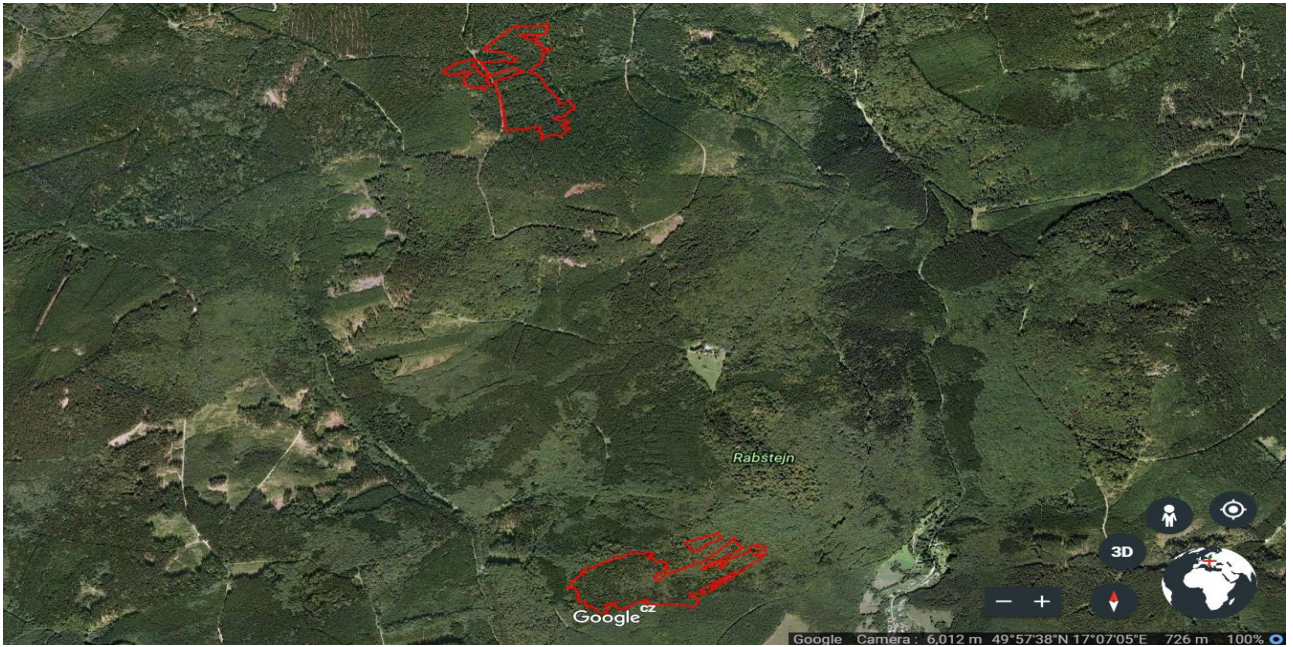


Figure 35: Result from Ing. Marek Mlcousek Jeseník area.

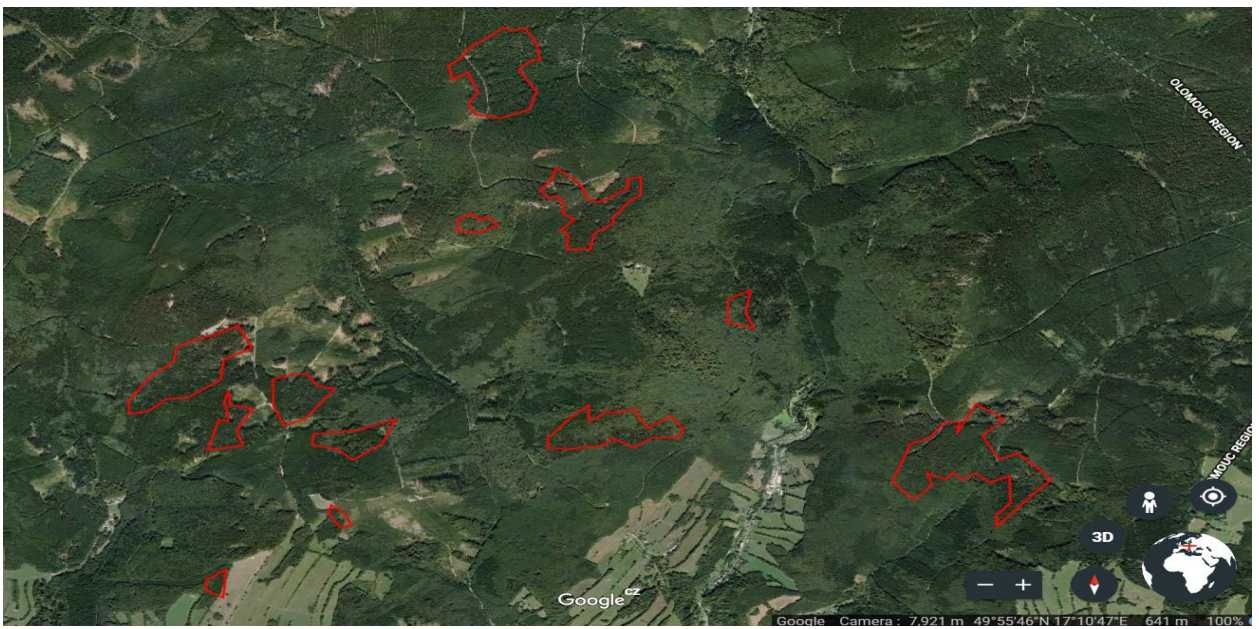


Figure 36: Our final result for Jeseník area





Figure 37: Our result overlaid on newest orthophotomap. Affected forest segments are visible. Our result is fitting these areas very well.

#### 4.4.4 Conclusion for Test 2

Figure 36, is showing the result is matching with the actual result (Figure 35).

## 5 Application for edge detection

The aim of the step is to generate a user-friendly application for selecting the parameters for thresholding, Canny edge detection, and NL Mean filter.

For this implementation, we used OpenCV library in python. OpenCV providing a trackbar method. With trackbar, we can get the position and we can able to create our track bar also. For our application we used createTrackbar() function. With this function, we were able to manage the values with this button. Figure 38, is showing the output for this application.

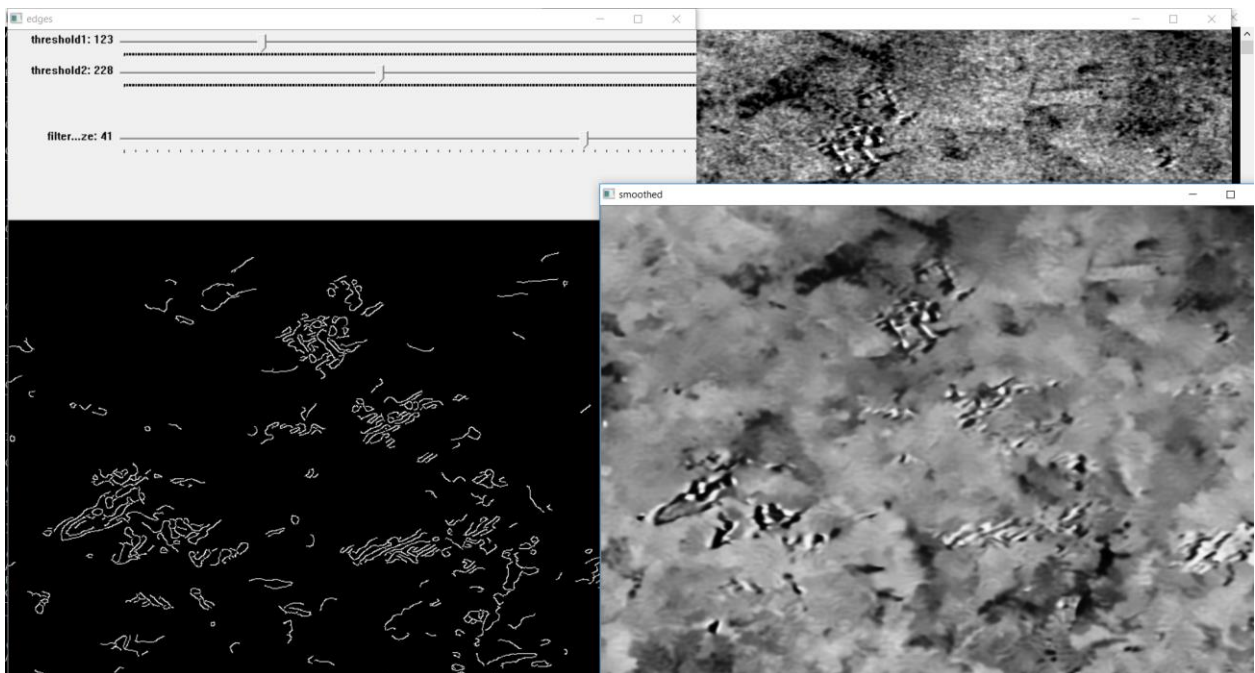


Figure 38: Application for setting all the parameter

## 6 Conclusions

In this project, we find out the area of forest which is affected by heavy wind. We found the damaged area and made segments of the object. We divided our implementation into three parts:

- Data pre-processing
- Forest Segmentation
- Vectorization / Polygon Shapefile
- Testing Result

In the data pre-processing part, SAR data was used for the assessment of damages caused by the heavy wind in the forest. The multispectral image was also used to identify the areas which were affected by the wind. The results from SAR and multispectral image both correlate, so SAR image can be an alternative for detection of hurricane damages with its advantage of operability in all weather conditions. After dealing with SAR images, PCA was used to find out the difference.

For forest segmentation, OpenCV library was used. openCV contained the necessary methods for image processing. OpenCV library was very useful for research work on images.

For visualizing the result, GDAL library was instrumental.

The result from Marek Mlcousek side was very useful for us. After comparing the results, we could verify our implementation.

## 7 Future work

- For future work, to improve the results and improve the accuracy we can use experiment the different algorithm. After studied research paper, we believe K-mean will be helpful for finding a good result.
- Because our data is satellite images so we can integrate the ArcGIS with python, it will be very useful for visualizing good results and saving time.
- The future, to reduce the computational time and speed up, we can use the NVIDIA GPU programming for PCA implementation. GPUs have been shown to increase the performance of machine learning algorithms by up to 100 times because they are highly efficient at parallel processing. All the matrix multiplications, which are integral to neural network models, can be performed simultaneously to gain drastic speedup in performance.
- The storm is a very common problem nowadays, so in future, with the full implementation, we can make as a user-friendly application and can offer it to the forest department. If this full implementation is user-friendly then the normal user can easily access the application and find out the result.
- In the future, we can integrate big data platforms with python for storing the input and output result.

## 8 References

- [1] McCarthy, J.K.; Hood, I.A.; Kimberley, M.O.; Didham, R.K.; Bakys, R.; Fleet, K.R.; Brownlie, R.K.; Flint, H.J.; Brockerhoff, E.G. Effects of season and region on sapstain and wood degrade following simulated storm damage in *Pinus radiata* plantations. *For. Ecol. Manag.* 2012, 277, 81–89. [CrossRef]
- [2] Økland, B.; Berryman, A. Resource dynamic plays a key role in regional fluctuations of the spruce bark beetles *Ips typographus*. *Agric. For. Entomol.* 2004, 6, 141–146. [CrossRef]  
8.1.1.1.1
- [3] Schwarz, M.; Steinmeier, C.; Holecz, F.; Stebler, O.; Wagner, H. Detection of Windthrow in Mountainous Regions with Different Remote Sensing Data and Classification Methods. *Scand. J. For. Res.* 2003, 18, 525–536. [CrossRef]
- [4] Usbeck, T.; Wohlgemuth, T.; Dobbertin, M.; Pfister, C.; Bürgi, A.; Rebetez, M. Increasing storm damage to forests in Switzerland from 1858 to 2007. *Agric. For. Meteorol.* 2010, 150, 47–55. [CrossRef]
- [5] Dyukarev, E.A.; Pologova, N.N.; Golovatskaya, E.A.; Dyukarev, A.G. Forest cover disturbances in the South Taiga of West Siberia. *Environ. Res. Lett.* 2011, 6. [CrossRef]
- [6] Elatawneh, A.; Wallner, A.; Manakos, I.; Schneider, T.; Knoke, T. Forest cover database updates using multi-seasonal rapideye data-storm event assessment in the Bavarian Forest National Park. *Forests* 2014, 5, 1284–1303.  
8.1.1.1.2
- [7] Einzmann, K.; Immitzer, M.; Böck, S.; Bauer, O.; Schmitt, A.; Atzberger, C. Windthrow Detection in European Forests with Very High-Resolution Optical Data. *Forests* 2017, 8, 21.  
8.1.1.1.3
- [8] Honkavaara, E.; Litkey, P.; Nurminen, K. Automatic Storm Damage Detection in Forests Using High-Altitude Photogrammetric Imagery. *Remote Sens.* 2013, 5, 1405–1424.
- [9] Kellndorfer, J.M.; Pierce, L.E.; Dobson, M.C.; Ulaby, F.T. Toward consistent regional-to-global-scale vegetation characterization using orbital SAR systems. *IEEE Trans. Geosci. Remote Sens.* 1998, 36, 1396–1411



- [10] Way, J.; Parist, J.; Kasischke, E.; Slaughter, C.; Viereck, L.; Christensen, N.; Dobson, M.C.; Ulaby, F.; Richards, J.; Milne, A.; et al. The effect of changing environmental conditions on microwave signatures of forest ecosystems: Preliminary results of the March 1988 Alaskan aircraft SAR experiment. *Int. J. Remote Sens.* 1990, 11, 1119–1144.  
8.1.1.1.4
- [11] Proisy, C.; Mougin, E.; Dufrière, E.; Le Dantec, V. Monitoring seasonal changes of a mixed temperate forest using ERS SAR observations. *IEEE Trans. Geosci. Remote Sens.* 2000, 38, 540–552.  
8.1.1.1.5
- [12] Koskinen, J.T.; Pulliainen, J.T.; Hallikainen, M.T. The Use of ERS-1 SAR Data in Snow Melt Monitoring. *IEEE Trans. Geosci. Remote Sens.* 1997, 35, 601–610.  
8.1.1.1.6  
8.1.1.1.7
- [13] Westman, W.E.; Paris, J.F. Detecting forest structure and biomass with C-band multipolarization Radar: Physical model and field tests. *Remote Sens. Environ.* 1987, 22, 249–269.  
8.1.1.1.8
- [14] Ahern, F.J.; Leckie, D.J.; Drieman, J.A. Seasonal changes in relative C-band backscatter of northern forest cover types. *IEEE Trans. Geosci. Remote Sens.* 1993, 31, 668–680.  
8.1.1.1.9  
8.1.1.1.10
- [15] Rüetschi, M.; Schaepman, M.E.; Small, D. Using multitemporal Sentinel-1 C-band backscatter to monitor phenology and classify deciduous and coniferous forests in northern Switzerland. *Remote Sens.* 2018, 10, 55.  
8.1.1.1.11
- [16] Green, R.M. The sensitivity of SAR backscatter to forest windthrow gaps. *Int. J. Remote Sens.* 1998, 19, 2419–2425.  
8.1.1.1.12
- [17] Fransson, J.E.S.; Walter, F.; Blennow, K.; Gustavsson, A.; Ulander, L.M.H. Detection of storm-damaged forested areas using airborne CARABAS-II VHF SAR image data. *IEEE Trans. Geosci. Remote Sens.* 2002, 40, 2170–2175.
- [18] Czech.cz. (2019). Jeseníky Mountains. [online] Available at: <http://www.czech.cz/en/66897-jeseniky-mountains> [Accessed 29 Apr. 2019].
- [19] Sentinel.esa.int. (2019). Sentinel-1 - Missions - Sentinel Online. [online] Available at: <https://sentinel.esa.int/web/sentinel/missions/sentinel-1> [Accessed 29 Apr. 2019].
- [20] Anaconda. (2019). Anaconda Python/R Distribution - Anaconda. [online] Available at: <https://www.anaconda.com/distribution/> [Accessed 29 Apr. 2019].
- [21] Docs.opencv.org. (2019). Denoising — OpenCV 3.0.0-dev documentation. [online] Available at: <https://docs.opencv.org/3.0-alpha/modules/photo/doc/denoising.html> [Accessed 27 Apr. 2019].

- [22] Docs.opencv.org. (2019). OpenCV: Canny Edge Detection. [online] Available at: [https://docs.opencv.org/3.1.0/da/d22/tutorial\\_py\\_canny.html](https://docs.opencv.org/3.1.0/da/d22/tutorial_py_canny.html) [Accessed 28 Apr. 2019].
- [23] Docs.opencv.org. (2019). OpenCV: Morphological Transformations. [online] Available at: [https://docs.opencv.org/trunk/d9/d61/tutorial\\_py\\_morphological\\_ops.html](https://docs.opencv.org/trunk/d9/d61/tutorial_py_morphological_ops.html) [Accessed 28 Apr. 2019].
- [24] Gdal.org. (2019). GDAL: gdal\_polygonize.py. [online] Available at: [https://www.gdal.org/gdal\\_polygonize.html](https://www.gdal.org/gdal_polygonize.html) [Accessed 28 Apr. 2019].
- [25] GitHub. (2019). BuddyVolly/SepalOstJupyter. [online] Available at: <https://github.com/BuddyVolly/SepalOstJupyter> [Accessed 30 Apr. 2019].

## 9 Appendix

### A1. Histogram equalization

The histogram equalization process is an image processing method to adjust the contrast of an image by modifying the image's histogram. The purpose behind this process is that histograms with large peaks correspond to images with low contrast where the background and the foreground are both dark and both light. Hence histogram equalization stretches the peak across the whole range of values leading to an improvement in the global contrast of an image. It is usually applied to grayscale images and it tends to produce unrealistic effects, but it is highly used where high contrast is needed such as in medical or satellite images.

### A2. Non-Local Mean Filter

Noise is containing a random variable with 0 mean. Consider a noisy pixel,  $p$  is equal to the summation of  $p_0$  and  $n$  where  $p_0$  is containing the original value of pixel and  $n$  is the noise in that pixel. We can take a large number of same pixels (say  $N$ ) from different images and compute their average. Lastly, we should get  $p$  equal to  $p_0$  since mean of noise is 0.

OpenCV provides four methods:

`cv.fastNlMeansDenoising()` – useful for single grayscale images

`cv.fastNlMeansDenoisingColored()` – useful for color image.

`cv.fastNlMeansDenoisingMulti()` – useful for grayscale image sequence captured in short period of time.

`cv.fastNlMeansDenoisingColoredMulti()` - useful for color image sequence captured in short period of time.

Same Arguments for all 4 methods:

- `h`: parameter deciding filter strength. Higher `h` value removes noise better but removes details of the image also. (10 is ok)
- `hForColorComponents`: same as `h`, but for color images only. (normally same as `h`)
- `templateWindowSize`: should be odd.
- `searchWindowSize`: should be odd.

### A3. Simple Thresholding

In simple thresholding If the pixel value is greater than a threshold value, it is equal to 1 value (white), else it is assigned 0 (black). The function used is `cv.threshold()`. The first parameter is the input image, which should be a grayscale image. The second parameter is the threshold value which is used to classify the pixel values. The third parameter is the max value which represents the value to be given if the pixel value is more than (sometimes less than) the threshold value. OpenCV provides different styles of thresholding and it is decided by the fourth parameter of the function. Different types are:

- `cv.THRESH_BINARY`:

- `cv.THRESH_BINARY_INV`: Inverse binary thresholding is just the opposite of binary thresholding. The output pixel is set to zero if the same input pixel is greater than the threshold, and to the max value, if the input pixel is less than the threshold.
- `cv.THRESH_TRUNC`: In this, the output pixel is set to the threshold if the input pixel value is greater than the threshold. Else it is set to the input pixel value. Max value is not considered.
- `cv.THRESH_TOZERO`: In this, the output pixel value is set to the same input pixel value if the input pixel value is larger the threshold. Otherwise, it is set to 0. Max value is not considered.
- `cv.THRESH_TOZERO_INV`: In this thresholding, output pixel is set to be 0 value if the input pixel value is larger than the threshold. Else it is set to the same as an input pixel value. Max value is considered.

#### **A4. Canny Edge Detection**

Canny Edge Detection Method By applying edge detection method the data volume in the image can significantly be reduced without affecting the structural properties which will be later used for image processing. There are several procedures available for edge detection. We have considered a Canny edge detection method with regards to the following criteria:

- **Detection:** The probability of detecting real edge points should be maximized while the probability of falsely detecting non-edge points should be minimized. This corresponds to maximizing the signal-to-noise ratio.
  - **Localization:** The detected edges should be very close to real edges. There will be a minimum gap between the real edge and detected Edge.
  - **A number of responses:** One real edge should not result in more than one detected edge. We have chosen a Canny edge detector which is optimal for step edges. Canny edge detection algorithm runs mainly in below steps:
    1. **Smoothing:** By applying the Gaussian filter, smoothing of an image is done to reduce noise.
    2. **Finding gradients:** The edges have been marked where the gradients of the image are having large magnitudes.
    3. **Non-maximum suppression:** Only local maxima have been marked as edges.
    4. **Double thresholding:** Potential and actual edges are determined by thresholding.
    5. **Edge tracking by hysteresis:** Edges that are not connected with strong edges have been suppressed.
- Smoothing Images will contain some amount of noise. As noise can change the result in finding edges, we have to reduce the noise.