Technical Report

Segmenting tree trunks from 3D point clouds with the use of deep learning

Version 1.0

August 17, 2019

Contents

1 Introduction

Within the forest industry, forest inventory is a large and critical step when it comes to planning, caring and valuing the resources within the forest. The most common approach today is to scan the forest from above, so called aerial laser scanning, with a LIDAR scanner attached to a helicopter or airplane. The data collected this way can be used to estimate timber volume, tree heights and terrain information [1]. Another less common procedure is to scan the forest in a similar way but on ground level, terrestrial laser scanning, where the scanners are either mounted on tripods or a mobile backpack setup. This procedure yields a lot more detailed and dense data of the actual trees, especially of the tree trunks.

This report concerns an experiment performed in order to see whether it is possible to utilize a neural network based on deep learning (DL) to classify tree trunks from the point cloud data collected from terrestrial laser scanning. The first step was to collect a suitable amount of training data, for which a method was developed to minimize manual work. The next step was to look into possible structures and algorithms for a neural network.

2 Method

Throughout the project, a model has been developed and adjusted based on the response and results along the way. Overall, the process consists of two major steps, where the first one is to generate training data and the second is to train a neural network with the generated training data.

2.1 Data set

The raw data that was provided was a collection of *.las* files from terrential laser scanning covering an area of roughly 1 hectare each. The points in these files were in the coordinate system called *Sweref99TM*, which is a national reference coordinate system in Sweden[2]. The type of forest that was scanned were mostly spruce, but also some which contains mixed coniferous forest. The files for the spruce forest typically consists of roughly 800 000 000 point records, while the mixed coniferous forest files consists of around 600 000 000 point records.

2.2 Softwares

Throughout the project, multiple softwares were used. A brief description of each program and how they were used will be listed in this section.

2.2.1 Tree trunk classification tool

A large asset for this project is a tree trunk classification tool developed by Kenneth Olofsson [3]. This particular program consists of multiple features which were used frequently during the project:

- reduce - Since the point clouds were so incredibly dense, it was both necessary and reasonable to reduce the amount of points by a set amount. The function takes in an integer corresponding to the reduce rate. With a reduce rate of n, every n:th point is kept and the rest are removed.
- trafotrans - transforms the data into another coordinate system. Since our data was in the coordinate system Sweref99TM the coordinates have very high values which are not compatible with the other functions which needs coordinates close to (0, 0, 0). Trafotrans transforms the coordinates of the data a desired amount for each coordinate. In our case they were transformed such that the minimum coordinates were located at (0, 0, 0).
- cuttiles - Creates tiles of the original file into the desired size. The function creates a new LAS file for each tile. We used a tile size of 10 meters.
- geotiffground - Creates a ground file of a LAS file. The ground file is needed for the classification function.

- tlsclassify - Classifies tree trunks in a LAS file. Input is the LAS file and its corresponding ground file created by geotiffground.

2.2.2 Lastools

For further handling of the LAS files, LAStools was used[6]. LAStools allowed us to merge files together, which was done after the individual tiles were classified and reduced, in order to load and process larger areas into the VR software PointCloud XR[7].

- merge - Merges las files into one.
- duplicate - Removes duplicates of points in a las file. Removes in a certain way making it possible to control which point it removes (in case you want to keep the ones with a specific class).

2.2.3 PointCloud XR

To further increase the quality of the training data, it was polished in a virtual reality (VR) software called PointCloud XR. This software made it possible to classify misclassified points manually. A more clear description of how this was done will be given further on.

2.3 Preprocessing of data

The data processing procedure that was developed consists of multiple steps. An overview of the workflow can be seen in figure 1.

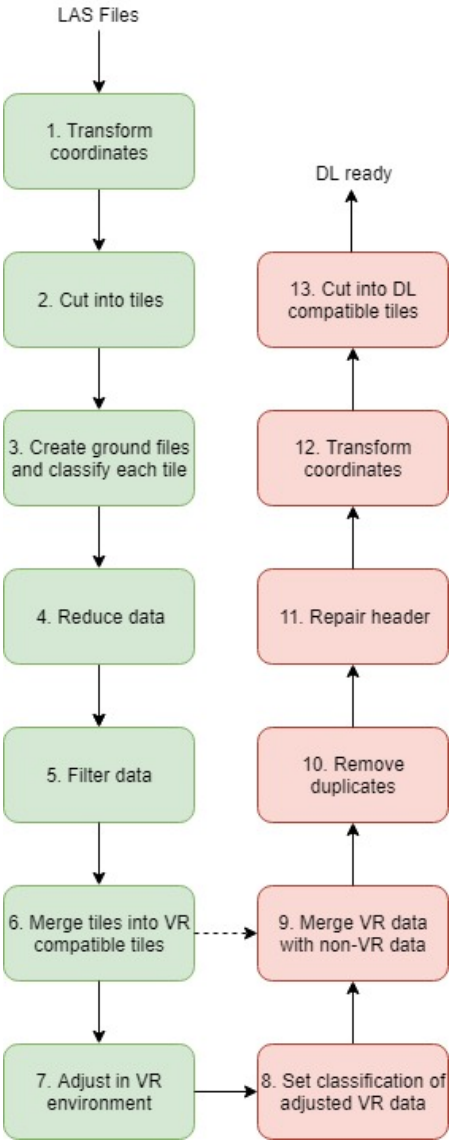

Figure 1 – A flowchart over the process to prepare raw pointcloud data for a neural network.

A more thorough description of each step:

1. **Transform coordinates -** This step is done because of the fact that the raw data is in coordinate system Sweref99TM, which means that the values for each coordinate point is a very large number. Some of the programs used in the process can not handle large numbers, which is why the first step is to transform the coordinates to have a minimum coordinate (0,0,0).
2. **Cut into tiles -** This step is done because the pre-classification process requires

smaller files, and a tile size of 10m x 10m was used.

3. **Create ground files and classify each tile -** The ground file is necessary for the classification to work. The classification of tree trunks in this step is rather rough, but it correctly classifies a large portion of the file.

4. **Reduce data -** Removes 14 out of 15 points to make computation times faster for up coming computations and in training.

5. **Filter data -** After the crude classification in the previous step, the data is filtered. The filter loops trough each tree trunk classified point and converts non-tree trunk neighbours of theses points into tree trunk classified points. The purpose is to ensure that as much tree trunk as possible is correctly classified even if it will incorrectly classify branches and ground. This is done in a exaggerated way (large radius to be categorized as a neighbouring point) to ensure that it captures as much of the tree trunk as possible. To minimize amounts incorrectly classified points the distance to be categorized as a neigbour was higher in z-direction (vertically) since non-tree trunk points mostly are located beside the tree trunk points and not above or below.

6. **Merges tiles into VR compatible sized files -** The VR software can, after some testing, load .las files with roughly the size 120 MB at a time, so a script was created to easily combine tiles and still keep the file small enough.

7. **Adjust in VR environment -** The files are loaded into the VR software, and the first thing to do is directly export the file. This is because the VR software transforms the coordinates, and to be able to calculate the transformation a non-modified file is needed. After this is done, the following procedure is performed:

   - Mark the points that are wrongly classified.
   - Use the "select inverse" functionality.
   - Use the "Delete selected", which means that the point records that are left are misclassified points since we selected the inverse.

   The point cloud that remains after performing the steps above is then exported as a new .las file.

8. **Set classification of adjusted VR data -** The points that make up the exported file with misclassified points are set to the same class as non-tree trunk, i.e. 0.

9. **Merge VR data with non-VR data -** The file from above step is merged with the original file.

10. **Remove duplicates -** The duplicate points in the merged file is removed, and due to the order the files are merged, the misclassified points will as a result be replaced with the ones from the adjusted data.

11. **Repair header -** The previous procedure destroys the header of the las file for some reason, so it is repaired.

12. **Transform coordinates -** From the originally exported VR file, the shift is calculated relative the untouched file and the merged file is thus shifted back to its original position.

13. **Cut into DL compatible tiles -** The files that are adjusted are then cut into tiles of size 10m x 10m again to feed into a Deep learning network.

2.4 Constructing and training the neural network

Mainly two different networks were used in this project, PointNet part segmentation network [5] and one network published by PhD Candidate at the University of Maine's School of Forest Resources, Elias Ayrey [4].

2.5 Elias Ayrey based network

The network based on the work of Elias Ayrey is basically a typical convolutional neural network which utilizes maxpooling. This network gives voxel-level predictions, which means that the number of inputs is equal to the number of voxels, and the output consists of predictions for each voxel. This was briefly tested and basically used as is, with a few modifications to make it easier to use. A helper script was also created to be able to extract the prediction of the voxels and properly classify the file that is tested. Since this network was just briefly tested, we will not go into any details to how it is set up exactly.

2.6 PointNet

PointNets part segmentation networks segments a point cloud into desired categories i.e. segmenting the different parts of a car (roof, windows, tires). In our case we tried this with two categories, tree trunk and non-tree trunk points. The special thing with PointNet is that it does not use voxels as most 3D neural networks does but instead feeds in the point cloud directly. This is done due to the fact that more properties of the 3D data is preserved when not applying a 3D voxel grid.

The pointNet implementation we looked at was adapted to using Keras[8]. For this implementation the las files had to change format into pts files and seg files and every file had to be reduced to below 2048 points (limit of the networks number of points per file). For each tile a pts file was created which stored the coordinates of the las files. For each tile one seg file was created for each category (in our case 2) and contained a one or a zero for each point, where one stands for that the point is from that category and a zero the opposite category. These files were further processed into two H5PY files which is fed into the network for training. The H5PY is good for storing huge amounts of data

and is still compatible with Numpy operations[9]. An overview of the network can be observed in figure 2.

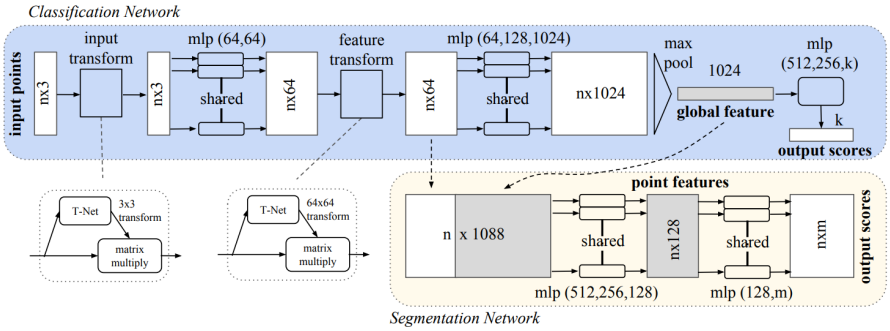

Figure 2 – The architecture of the network. The blue part is only the classification network and the yellow part is the segmentation network which is an extension of the classification network. The segmentation Network uses global and local features from the classification network to make its predictions. For more information see [5].

We will not go into more detail on how the network works since it was quickly tested in this project and not used or explored to any greater extent.

3 Results

A method for retrieving training data was achieved using two scripts and a VR software. The Original data is reduced by a factor of 15. A comparison of the data can be seen in figure 3.

Figure 3 – (a) How original data looks with no reduction. (b) How the data looks after 14 out of 15 points are removed (not the same data file).

We can see that the main features of the trees are not lost after the reduction. This reduction is needed to be able to achieve reasonable training times for the networks.

For the network based on Elias Ayrey's work, the accuracy during training looked quite promising. Sadly, these were not recorded, but they converged towards roughly 80%. However, this was probably due to immense overfitting, since when trying out the network on new data, we got the following result:

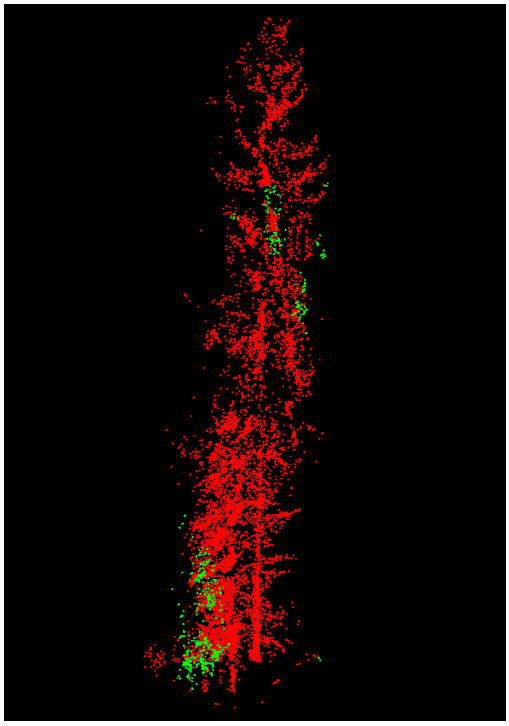

Figure 4 – Classification of a point cloud using Elias Ayrey's work. Green points corresponds to tree trunks and red point corresponds to non-tree trunk.

Ideally, the points that make up tree trunk should be green and the remaining points, i.e. those who makes up everything else, should be red. Clearly, this is not the case, since the points seem randomly classified.

When training the PointNet an accuracy of around 90% was achieved with a loss of around 0.3. The result of the classification can be observed in figure 5.

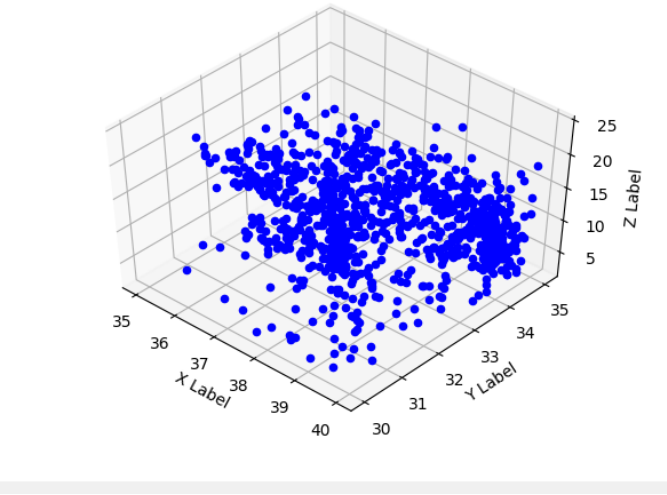

Figure 5 – A visualization of the classification made by the trained PointNet.

4 Discussion

4.1 Preprocessing of data

The workflow of data processing has been developed and adjusted throughout the entirety of the project. The method that was developed and used in the end feels like it has several strengths when it comes to retrieve high quality training data to use for a neural network. These strengths lies in both the ease of generating more training data, but also the actual quality of the data, in that it does not lose much valuable information. The basic idea has always revolved around as little manual work as possible, i.e having to manually adjust the misclassified points in the virtual reality software, which we feel like we have kept to a minimum.

The big bottleneck for retrieving data is the manual work that has to be done in the VR environment. We reduced the amount of points needed to be removed in VR by applying filters that changed the class of neighbouring points and a filter which changed the class for all points a certain amount above the ground model of the data. After this process the biggest problem were leafs, needles and branches. In figure 6 we can see the incorrectly classified points which needs to be removed with the VR software.

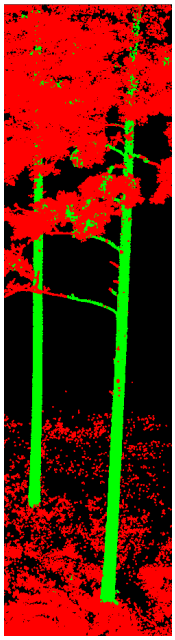

Figure 6 – How the classification looks before VR adjustments. Green points represent tree trunk classified points.

One alternative would be to ignore these errors to be able to skip the time consuming VR-process. Doing this would make it possible to continuously produce big amounts

of training data around the clock with several computers. This would be a trade-off between quality and quantity of the training data. The resulting network might find more trees but with more noise in each segmented tree and depending of how much the extra incorrect classified biomass affects the result this might be a reasonable strategy.

4.2 Constructing and training neural network

For the convolutional neural network based on Elias Ayrey's work, the data had to be voxelized before it was fed into the network. This meant that the resolution that could realistically be used is rather poor relative to how detailed the original data is. One can thus not conclude that it is the fact that we lacked training data that caused such poor result and we suspect we would still get such poor results even if we had more training data because of low resolution. This means that we probably want to increase the resolution by either increasing the number of voxels, i.e. reducing their size, or reducing the size of the tiles. However, reducing the size of the tiles would most likely make it so that a lot of the trees are cut in half, so the large picture would be lost.

As seen in figure 5 we can see that results are very bad for the PointNet. It looks like the points are all classified into one class. We think that the network classifies all points as non tree trunk since that will give a very high accuracy due to the fact that so few points are actually tree trunk. This visualisation is the one provided from the keras implementation of PointNet and we do not really understand how the visualization is generated and the results does not look like the original point cloud. Something is probably not working or it only shows a part of the point cloud. Due to time limitations this was not explored more however it was discovered that the network might need RGB info from the training data.

4.3 Possible improvements

Since this was a research based project there are several areas with improvement potentials. The biggest improvement could be done with the neural networks since too little time was spent on developing them due to the fact that the preprocessing part was more time consuming than anticipated. Since we thought we had found a neural network that could be used with just a few smaller adjustments, we spent more time on the preprocessing part than we should have. This means that the preprocessing procedure, however, is rather well thought-out and the deep learning aspect is lacking.
Some concrete improvements we have thought about during the project are the following:

- Further improve the preprocessing procedure to remove the VR adjusting part resulting in faster collection of training data.
- Put more time into investigate the networks to achieve usable results.
- Use smaller voxels, i.e. bigger resolution, which would lead to a longer training time but we suspect it will achieve better classification.

A final note is that it would probably be preferable to construct a classifier which initially segments entire trees out of a point cloud, and afterwards classifies each point within that segment as tree trunk or biomass. This would allow a much better resolution, since each tree segment will be a lot smaller than the 10m x 10m tiles that were used in this project.

References

[1] Skogsstyrelsen. *Laserskannade Skogliga grunddata*. https://www.skogsstyrelsen.se/mer-om-skog/laserskannade-skogsdata/ downloaded 2019-06-08

[2] Lantmäteriet. *SWEREF 99, projektioner*. https://www.lantmateriet.se/sv/Kartor-och-geografisk-information/gps-geodesi-och-swepos/Referenssystem/Tvadimensionella-system/SWEREF-99-projektioner/ , downloaded 2019-06-09.

[3] Kenneth Olofsson and Johan Holmgren. *Single Tree Stem Profile Detection Using Terrestrial Laser Scanner Data, Flatness Saliency Features and Curvature Properties*. Forests 7, no. 9: 207, 2016.

[4] Elias Ayrey's Github. https://github.com/Eayrey/3D-Convolutional-Neural-Networks-with-LiDAR, downloaded 2019-06-9.

[5] Qi, Charles R and Su, Hao and Mo, Kaichun and Guibas, Leonidas J. *PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation*. arXiv preprint arXiv:1612.00593, 2016.

[6] Martin Isenburg. *LAStools*. https://rapidlasso.com/contact/, downloaded 2019-08-16

[7] Ljungbergslaboratoriet *PointCloud XR*. http://www.rslab.se/pointcloud-xr/, downloaded 2019-08-16

[8] Keras. *Keras: The Python Deep Learning library*. https://keras.io/, downloaded 2019-08-16

[9] Numpy. *Numpy*. https://www.numpy.org/, downloaded 2019-08-16