REI204M: High Performance Computing
Industry Course Project
Orb - Forest Assessment

Emil Atli Þórhallsson
Samuel Caspar Gerbers

11. May 2025

GitHub Repository:
All code, data processing, and analysis scripts used in this report are available at:
`https://github.com/EmilAtli/Orb-project`

Introduction

With this project the goal is to see how we can translate the efforts from Meta in their project to predict tree heights based on high resolution satellite data [Tolan et al., 2024] to new data. In the paper the focuse was on Los Angeles, USA and Sao Paulo, Brazil and we want to see if the model can also be used for Iceland's terrain. For that purpose we have tree data from Iceland, that we use for testing. A success of the project would mean, that the process of measuring trees in Iceland to watch the tree population can be made immensely easier and less time intensive.

Running the Model

Since the project is using a model using CUDA, making use of a high performance system will be a huge help. In the beginning we didn't have access to a system so we used Google Colab with its NVIDIA Tesla T4 runtime, you can see the procedure in the ORB.ipynb Jupyter notebook in the zip or on our Github. Later we switched to using the Elja system to run the project.

Preparing the environment for Elja

We start by accessing the system using our ssh key:

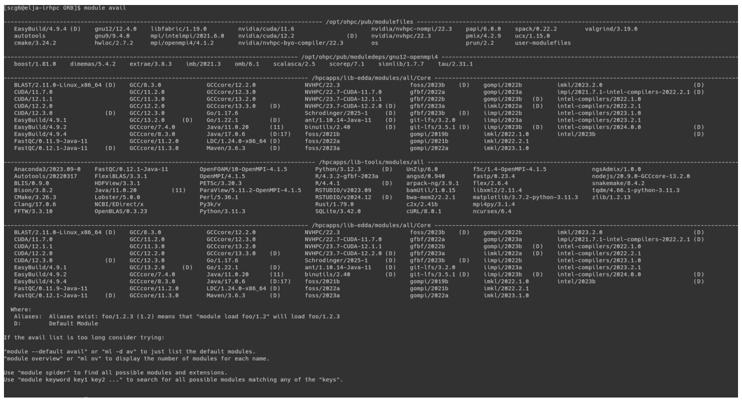
```
ssh -i ~/.ssh/elja scg6@elja.hi.is
```

Then we set up the folder structure:

```
mkdir ORB
cd ORB
```

Now we are on the login node and that is where we prepare the virtual environment for Python to be able to run the script.

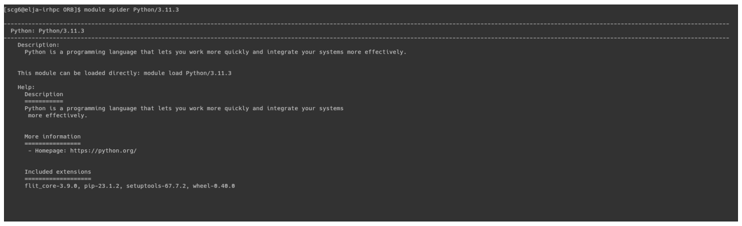
First to make sure everything is consistent we also want to use the specific Python version that we want to use for the project which is Python 3.11 and with `module avail` in the system we can see that there is even a module with Python 3.11.3 which we can use:



```
[scg6@elja-irhpc ORB]$ module avail

------------------------------------------------------------------------------ /opt/ohpc/pub/modulefiles ------------------------------------------------------------------------------
   EasyBuild/4.9.4 (D)    gnu12/12.4.0    libfabric/1.19.0         nvidia/cuda/11.6                        nvidia/nvhpc-nompi/22.3    papi/6.0.0    spack/0.22.2        valgrind/3.19.0
   autotools              gnu9/9.4.0      mpi/intelmpi/2021.6.0    nvidia/cuda/12.2                 (D)    nvidia/nvhpc/22.3          pmix/4.2.9    ucx/1.15.0
   cmake/3.24.2           hwloc/2.7.2     mpi/openmpi4/4.1.2       nvidia/nvhpc-byo-compiler/22.3          os                         prun/2.2      user-modulefiles

---------------------------------------------------------------------------- /opt/ohpc/pub/moduledeps/gnu12-openmpi4 ----------------------------------------------------------------------------
   boost/1.81.0    dimemas/5.4.2    extrae/3.8.3    imb/2021.3    omb/6.1    scalasca/2.5    scorep/7.1    sionlib/1.7.7    tau/2.31.1

------------------------------------------------------------------------------- /hpcapps/lib-edda/modules/all/Core -------------------------------------------------------------------------------
   BLAST/2.11.0-Linux_x86_64 (D)    GCC/8.3.0              GCCcore/12.2.0             NVHPC/22.3                     foss/2023b    (D)    gompi/2022b          imkl/2023.2.0                                (D)
   CUDA/11.7.0                      GCC/11.2.0             GCCcore/12.3.0             NVHPC/22.7-CUDA-11.7.0         gfbf/2022a           gompi/2023a          impi/2021.7.1-intel-compilers-2022.2.1 (D)
   CUDA/12.1.1                      GCC/11.3.0             GCCcore/13.2.0             NVHPC/23.7-CUDA-12.1.1         gfbf/2022b           gompi/2023b   (D)    intel-compilers/2022.1.0
   CUDA/12.2.0                      GCC/12.2.0             GCCcore/13.3.0    (D)      NVHPC/23.7-CUDA-12.2.0 (D)     gfbf/2023a           iimkl/2022a   (D)    intel-compilers/2022.2.1
   CUDA/12.3.0               (D)    GCC/12.3.0             Go/1.17.6                  Schrodinger/2025-1     (D)     gfbf/2023b    (D)    iimpi/2022b          intel-compilers/2023.1.0
   EasyBuild/4.9.1                  GCC/13.2.0     (D)     Go/1.22.1         (D)      ant/1.10.14-Java-11    (D)     git-lfs/3.2.0        iimpi/2023a          intel-compilers/2023.2.1
   EasyBuild/4.9.2                  GCCcore/7.4.0          Java/11.0.20      (11)     binutils/2.40          (D)     git-lfs/3.5.1 (D)    iimpi/2023b   (D)    intel-compilers/2024.0.0                     (D)
   EasyBuild/4.9.4                  GCCcore/8.3.0          Java/17.0.6       (D:17)   foss/2021b                     gompi/2019b          imkl/2022.1.0        intel/2023b                                  (D)
   FastQC/0.11.9-Java-11            GCCcore/11.2.0         LDC/1.24.0-x86_64 (D)      foss/2022a                     gompi/2021b          imkl/2022.2.1
   FastQC/0.12.1-Java-11     (D)    GCCcore/11.3.0         Maven/3.6.3       (D)      foss/2023a                     gompi/2022a          imkl/2023.1.0

-------------------------------------------------------------------------------- /hpcapps/lib-tools/modules/all --------------------------------------------------------------------------------
   Anaconda3/2023.09-0    FastQC/0.12.1-Java-11         OpenFOAM/10-OpenMPI-4.1.5        Python/3.12.3      (D)    UnZip/6.0         f5c/1.4-OpenMPI-4.1.5               ngsAdmix/1.0.0
   Autotools/20220317     FlexiBLAS/3.3.1               OpenMPI/4.1.5                    R/4.3.2-gfbf-2023a        angsd/0.940       fastp/0.23.4                        nodejs/20.9.0-GCCcore-13.2.0
   BLIS/0.9.0             HDFView/3.3.1                 PETSc/3.20.3                     R/4.4.1            (D)    arpack-ng/3.9.1   flex/2.6.4                          snakemake/8.4.2
   Bison/3.8.2            Java/11.0.20          (11)    ParaView/5.11.2-OpenMPI-4.1.5    RSTUDIO/v2023.09          bamUtil/1.0.15    libxml2/2.11.4                      tqdm/4.66.1-python-3.11.3
   CMake/3.26.3           Lobster/5.0.0                 Perl/5.36.1                      RSTUDIO/v2024.12   (D)    bwa-mem2/2.2.1    matplotlib/3.7.2-python-3.11.3      zlib/1.2.13
   Clang/17.0.6           NCBI/EDirect/x                Py3k/v                           Rust/1.79.0               c2x/2.41b         mpi4py/3.1.4
   FFTW/3.3.10            OpenBLAS/0.3.23               Python/3.11.3                    SQLite/3.42.0             cURL/8.0.1        ncurses/6.4

------------------------------------------------------------------------------- /hpcapps/lib-edda/modules/all/Core -------------------------------------------------------------------------------
   BLAST/2.11.0-Linux_x86_64 (D)    GCC/8.3.0              GCCcore/12.2.0             NVHPC/22.3                     foss/2023b    (D)    gompi/2022b          imkl/2023.2.0                                (D)
   CUDA/11.7.0                      GCC/11.2.0             GCCcore/12.3.0             NVHPC/22.7-CUDA-11.7.0         gfbf/2022a           gompi/2023a          impi/2021.7.1-intel-compilers-2022.2.1 (D)
   CUDA/12.1.1                      GCC/11.3.0             GCCcore/13.2.0             NVHPC/23.7-CUDA-12.1.1         gfbf/2022b           gompi/2023b   (D)    intel-compilers/2022.1.0
   CUDA/12.2.0                      GCC/12.2.0             GCCcore/13.3.0    (D)      NVHPC/23.7-CUDA-12.2.0 (D)     gfbf/2023a           iimkl/2022a   (D)    intel-compilers/2022.2.1
   CUDA/12.3.0               (D)    GCC/12.3.0             Go/1.17.6                  Schrodinger/2025-1     (D)     gfbf/2023b    (D)    iimpi/2022b          intel-compilers/2023.1.0
   EasyBuild/4.9.1                  GCC/13.2.0     (D)     Go/1.22.1         (D)      ant/1.10.14-Java-11    (D)     git-lfs/3.2.0        iimpi/2023a          intel-compilers/2023.2.1
   EasyBuild/4.9.2                  GCCcore/7.4.0          Java/11.0.20      (11)     binutils/2.40          (D)     git-lfs/3.5.1 (D)    iimpi/2023b   (D)    intel-compilers/2024.0.0                     (D)
   EasyBuild/4.9.4                  GCCcore/8.3.0          Java/17.0.6       (D:17)   foss/2021b                     gompi/2019b          imkl/2022.1.0        intel/2023b                                  (D)
   FastQC/0.11.9-Java-11            GCCcore/11.2.0         LDC/1.24.0-x86_64 (D)      foss/2022a                     gompi/2021b          imkl/2022.2.1
   FastQC/0.12.1-Java-11     (D)    GCCcore/11.3.0         Maven/3.6.3       (D)      foss/2023a                     gompi/2022a          imkl/2023.1.0

  Where:
   Aliases:  Aliases exist: foo/1.2.3 (1.2) means that "module load foo/1.2" will load foo/1.2.3
   D:        Default Module

If the avail list is too long consider trying:

"module --default avail" or "ml -d av" to just list the default modules.
"module overview" or "ml ov" to display the number of modules for each name.

Use "module spider" to find all possible modules and extensions.
Use "module keyword key1 key2 ..." to search for all possible modules matching any of the "keys".
```

Figure 1: Result of module avail

Using `module spider` for the Module we can see that we can directly load the module:



```
[scg6@elja-irhpc ORB]$ module spider Python/3.11.3

--------------------------------------------------------------------------------------------------------------------------------------------
  Python: Python/3.11.3
--------------------------------------------------------------------------------------------------------------------------------------------
    Description:
      Python is a programming language that lets you work more quickly and integrate your systems more effectively.


    This module can be loaded directly: module load Python/3.11.3

    Help:
      Description
      ===========
      Python is a programming language that lets you work more quickly and integrate your systems
       more effectively.


      More information
      ================
       - Homepage: https://python.org/


      Included extensions
      ===================
      flit_core-3.9.0, pip-23.1.2, setuptools-67.7.2, wheel-0.40.0
```

Figure 2: Result of module spider

After we got the correct version of Python we can start setting up the virtual environment with all the required libraries with the correct version:

```
python3 -m venv orb_env
source orb_env/bin/activate
pip install pip==24.0
pip install numpy==1.26.4 pytorch-lightning==1.7 torch==2.0.1
    torchvision==0.15.2 torchmetrics==0.11.4 pillow==11.0.0 pandas
    ==2.2.3 matplotlib==3.9.4
pip install awscli
pip install transformers==4.38.2
```

Now that we have the correct environment lets set up the fork of the git project from Meta that we got from Jón from ORB and as well get the required data for the project over aws:

```
git clone https://github.com/jonarnar/HighResCanopyHeight.git
cd HighResCanopyHeight
aws s3 --no-sign-request cp --recursive s3://dataforgood-fb-data/
    forests/v1/models/ .
unzip data.zip
rm data.zip
```

When we wanted to run the program we had a problem with pytorch needing to have a compiler, which it didn't have but we could fix it by adding a dummy compiler to the inference.py script inside the git repository. To add it you can run the following python script or add the block in the python script manually:

```python
import fileinput
# Inserts a dummy compiler, since torch v2.0.1 doesn't have a
    compiler.

# block to insert.
block = '''if not hasattr(torch, "compiler"):
    class DummyCompiler:
        @staticmethod
        def disable(*args, **kwargs):
            def decorator(func):
                return func
            return decorator
    torch.compiler = DummyCompiler()
'''

# Open inference.py for in-place editing.
with fileinput.input('./inference.py', inplace=True) as file:
    for line_number, line in enumerate(file, 1):
        print(line, end='')
        # After line 8 (which is "import torch"), block is inserted
    .
        if line_number == 8:
            print(block, end='')
```

So to execute the script you can simply create a file using vim and then copy the data there and execute the file:

```
vim fix_inference.py
# paste the code above inside, save it and run the script:
python3 fix_inference.py
```

Now we have to prepare the data we want to use for the inference so the tree data that we were provided. I still had it on my local machine so we can use scp to copy it from my local machine to the remote HPC system, so we execute the following command from our local machine:

```
scp -i ~/.ssh/elja trees.zip scg6@elja.hi.is:/jotunn/home/scg6/ORB/
    HighResCanopyHeight
```

After that we also have to prepare the csv file that correctly uses the new data set, for that purpose we made a short python script that we executed on our local machine after unzipping the tree data there. So we:

```
unzip trees.zip
```

And then we can use the following make_csv.py script to create the new csv:

```python
import os
import csv

file_list = sorted(os.listdir('./output_images/'))

with open("data.csv", "w", newline="") as csvfile:
    writer = csv.writer(csvfile)
    writer.writerow(["", "index", "image"])  # header with leading
    comma
    for idx, filename in enumerate(file_list):
        index = int(filename[:-4])
        writer.writerow([idx, index, filename])
```

Now we also want to send this csv file to Elja:

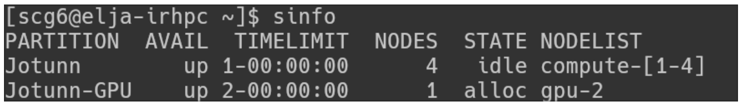
```
scp -i ~/.ssh/elja data.csv scg6@elja.hi.is:/jotunn/home/scg6/ORB/
    HighResCanopyHeight
```

Finally before executing the the programm we also need to unzip the trees.zip file on Elja:

```
unzip trees.zip
```

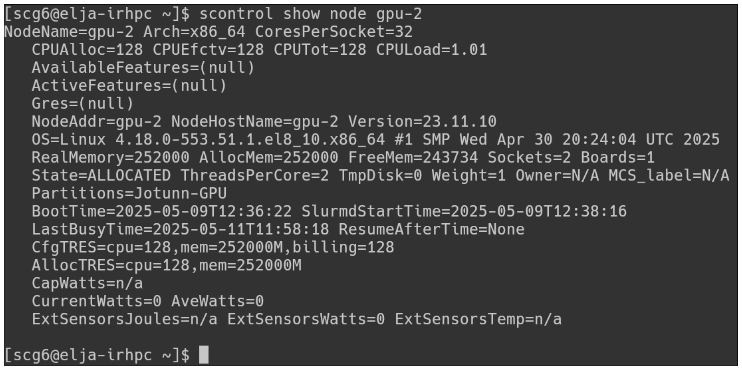
Running the model on Elja

Now everything is prepared to actually execute the model on Elja, to make sure we use the right partition and we understand how it is structured to write the batch script to submit the job we used `sinfo` and `scontrol show node`:



```
[scg6@elja-irhpc ~]$ sinfo
PARTITION  AVAIL  TIMELIMIT  NODES  STATE NODELIST
Jotunn        up 1-00:00:00      4   idle compute-[1-4]
Jotunn-GPU    up 2-00:00:00      1  alloc gpu-2
```

Figure 3: Result of sinfo



```
[scg6@elja-irhpc ~]$ scontrol show node gpu-2
NodeName=gpu-2 Arch=x86_64 CoresPerSocket=32
   CPUAlloc=128 CPUEfctv=128 CPUTot=128 CPULoad=1.01
   AvailableFeatures=(null)
   ActiveFeatures=(null)
   Gres=(null)
   NodeAddr=gpu-2 NodeHostName=gpu-2 Version=23.11.10
   OS=Linux 4.18.0-553.51.1.el8_10.x86_64 #1 SMP Wed Apr 30 20:24:04 UTC 2025
   RealMemory=252000 AllocMem=252000 FreeMem=243734 Sockets=2 Boards=1
   State=ALLOCATED ThreadsPerCore=2 TmpDisk=0 Weight=1 Owner=N/A MCS_label=N/A
   Partitions=Jotunn-GPU
   BootTime=2025-05-09T12:36:22 SlurmdStartTime=2025-05-09T12:38:16
   LastBusyTime=2025-05-11T11:58:18 ResumeAfterTime=None
   CfgTRES=cpu=128,mem=252000M,billing=128
   AllocTRES=cpu=128,mem=252000M
   CapWatts=n/a
   CurrentWatts=0 AveWatts=0
   ExtSensorsJoules=n/a ExtSensorsWatts=0 ExtSensorsTemp=n/a

[scg6@elja-irhpc ~]$
```

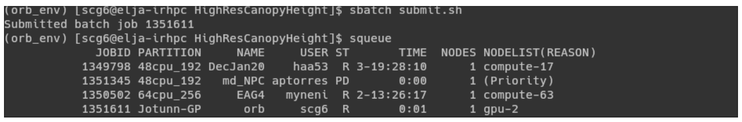
Figure 4: Result of scontrol show node

Using that info we can write our batch script submit.sh to submit the job, here we can then simply activate the python environment we created before:

```
#!/bin/bash
#SBATCH -J orb
#SBATCH --partition=Jotunn-GPU
```

```
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=4
#SBATCH --mem=32G
#SBATCH --time=01:00:00
#SBATCH --mail-user=scg6@hi.is
#SBATCH --mail-type=end

module load Python/3.11.3
source ../orb_env/bin/activate

python inference.py --csv ./data.csv --image_dir ./output_images/
    --name output_inference
```

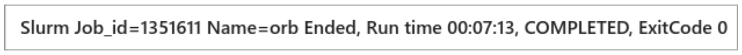
Then we can actually submit it and see it in the queue:



```
(orb_env) [scg6@elja-irhpc HighResCanopyHeight]$ sbatch submit.sh
Submitted batch job 1351611
(orb_env) [scg6@elja-irhpc HighResCanopyHeight]$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
           1349798 48cpu_192 DecJan20    haa53  R 3-19:28:10      1 compute-17
           1351345 48cpu_192   md_NPC aptorres PD       0:00      1 (Priority)
           1350502 64cpu_256     EAG4   myneni  R 2-13:26:17      1 compute-63
           1351611 Jotunn-GP      orb     scg6  R       0:01      1 gpu-2
```

Figure 5: Submitting the job

Since we specified we get a notification at the end of the job via mail, we get get the runtime from there:



Slurm Job_id=1351611 Name=orb Ended, Run time 00:07:13, COMPLETED, ExitCode 0

Figure 6: finished job

It only took 07:13 to complete the job, showing the how useful it is to use a HPC system like Elja.

Since a lot of of the trees are in the same areas, they are combined in one picture instead of having individual pictures for each tree. Now we have all the resulting trees in the output_inference folder, which we can zip and then send back to our local machine to do the analysis of the data there:

```
(on elja) zip -r output_inference.zip output_inference/
(on local machine) scp -i ~/.ssh/elja scg6@elja.hi.is:/jotunn/home/
    scg6/ORB/HighResCanopyHeight/output_inference.zip /home/samu/
    Documents/Iceland/Uni/Orb/submit/
```

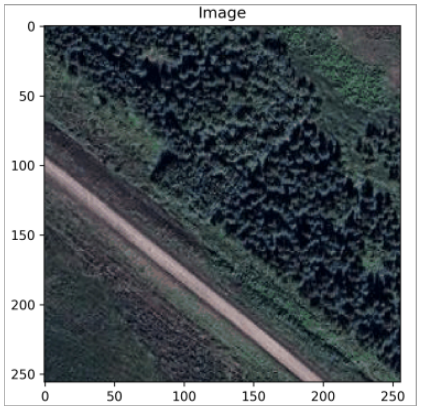
Combining the data

The zip trees.zip contains the output.gpkg file which has all the information about the trees including the position of the tree in the image and also the gorund truth height of the tree. Using this file we can easily combine the ground truth with the predictions of the model.

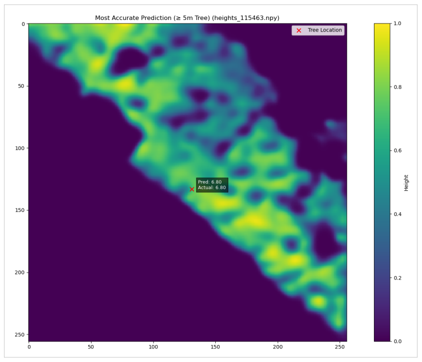
To combine it we can simply add a column describing the predicted height, this can be done with a simple python script by checking, the location of the height map that was generated by the model for the height that we predicted for each tree and then adding that number to the row of the tree:

```python
import geopandas as gpd
import numpy as np
import os
from scipy.ndimage import map_coordinates

input_gpkg = "./output.gpkg"
input_layer = "output"
npy_folder = "./inference/"
output_gpkg = "./output_with_predictions.gpkg"
output_layer = "predicted_output"

gdf = gpd.read_file(input_gpkg, layer=input_layer)

def get_precise_height(image_name, px, py):
    npy_name = "heights_" + image_name.replace(".png", ".npy")
    npy_path = os.path.join(npy_folder, npy_name)
    if not os.path.exists(npy_path):
        return None
    arr = np.load(npy_path)
    if 0 <= px < arr.shape[1] and 0 <= py < arr.shape[0]:
        coords = np.array([[py], [px]])
        return float(map_coordinates(arr, coords, order=1, mode='
    nearest')[0])
    return None

gdf["predicted_height"] = gdf.apply(
    lambda row: get_precise_height(row["image"], row["px"], row["py
    "]),
    axis=1
)

gdf.to_file(output_gpkg, layer=output_layer, driver="GPKG")
print("done")
```

And we can also look at some specific examples of our prediction here is one example where it performed good and one where it performed bad:

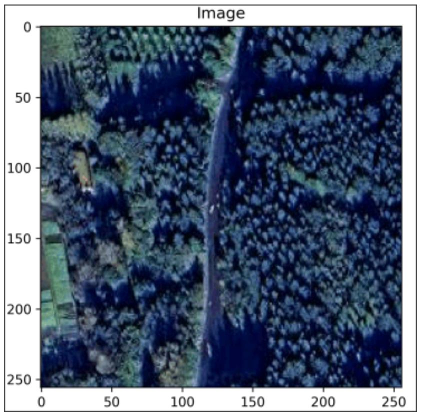


(a) Satellite Picture

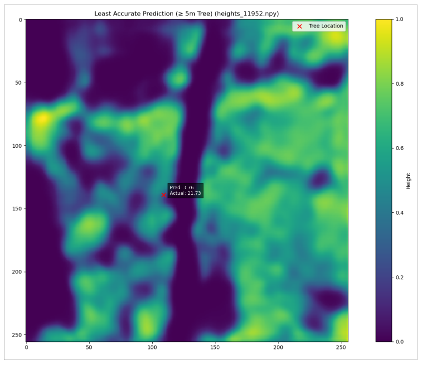


(b) Prediction

Figure 7: Good performing sample



(a) Satellite Picture



(b) Prediction

Figure 8: Bad performing sample

Assessing the Performance

Linear Regression

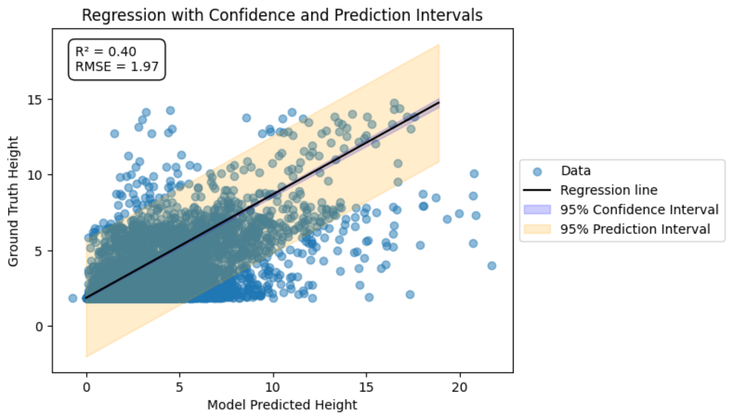


Figure 9: Result of Linear Regression

The figure above shows the result of applying a linear regression to compare the predicted tree heights from the model with the ground truth measurements from the entire dataset. Each dot represents an individual tree, with the predicted height on the x-axis and the actual measured height on the y-axis.

The black line is the fitted regression line, showing the general trend between predictions and reality. The blue shaded area around the line represents the 95% confidence interval, which tells us how certain the model is about the average trend. The wider orange band shows the 95% prediction interval, which indicates the expected range for individual predictions.

We calculated an R^2 score of 0.40, meaning the model explains about 40% of the variance in the data. The root mean squared error (RMSE) is 1.97 meters, which means that, on average, the model's predictions are off by nearly 2 meters. Based on this result, the model shows a general trend but struggles to make precise predictions for individual trees when using the full dataset.

Table 1: Regression results for each species, sorted by R^2 score.

Species	R^2	RMSE
Hvítgreni	0.92	0.58
Gráölur	0.86	1.27
Reynir	0.83	0.38

Continued on next page

Continued from previous page

Species	R^2	RMSE
Broddgreni	0.70	1.57
Lindifura	0.60	0.57
Hengibirki	0.53	1.66
Rússa- og síberíulerki	0.51	1.90
Sitkagreni	0.51	1.70
Selja	0.50	1.22
Stafafura	0.49	1.78
Sitkagreni og -bastarður	0.47	2.07
Rússalerki	0.45	1.65
Alaskaösp	0.42	3.06
Rauðgreni	0.41	2.41
Gulvíðir	0.38	1.22
Reyniviður	0.37	2.02
Sitkaölur	0.30	0.75
Bergfura	0.29	1.55
Ilmbjörk	0.26	1.51
Fjallaþöll	0.25	0.24
Evrópulerki	0.25	0.59
Blágreni	0.17	1.19
Skógarfura	0.15	1.91
Víðir	0.13	1.48
Viðja	0.10	1.75
Alaskavíðir	0.10	1.47
Ilmbjörk - sjálfsáð	0.09	1.35
Jörfavíðir	0.07	0.45
Fjallaþinur	0.00	0.91

Table 1 shows how the model performed on each tree species, based on R^2 and RMSE. Looking at the results, it's clear that the model works best for *Hvítgreni*, *Gráölur*, and *Reynir*, which all have R^2 values over 0.8. This means the model does a pretty good job of predicting tree heights for these species. The RMSE for these species is also quite low, which adds to our confidence in the results.

Further down the list, the R^2 values drop off quite a bit. For many of the other species, the model struggles to make reliable predictions. Some of the R^2 scores are even close to zero, which basically means the model isn't doing much better than random guessing for those species.

It's also worth mentioning that a few species were skipped in the analysis because there was only one sample available. These species were *Silfurreynir*, *Álmur*, *Douglasgreni*, *Kjarrölur*, *Fura*, *Heggur*, and *Fjallafura*. With just one data point, it isn't possible to fit a meaningful regression model, so these were left out.

All in all, it seems like the model shows potential for certain species, but there's still a lot of room for improvement if we want it to work well across the board.

Improved Sampling with Neighborhood Maximum

In the previous analysis, the predicted height for each tree was sampled directly at the pixel location provided in the ground truth data. However, this assumes that the provided pixel coordinates perfectly align with the actual treetop in the prediction map, which is often not the case. To account for this, we updated the method to search for the highest predicted value in a local neighborhood around each tree location.

After testing different window sizes, ranging from 3x3 to 21x21 pixels, we found that using a 21x21 window (covering approximately 10.5 meters by 10.5 meters) produced the best results. This window size likely captures the actual treetop even if the provided coordinates are slightly off. The improved regression analysis using this method achieved an R^2 score of 0.48 and an RMSE of 1.84 meters, slightly better than the previous approach.

Figure 10 shows the updated regression result with the improved sampling method. The results suggest that allowing the model to search for the local maximum in the area around each tree provides a more reliable comparison to the ground truth.

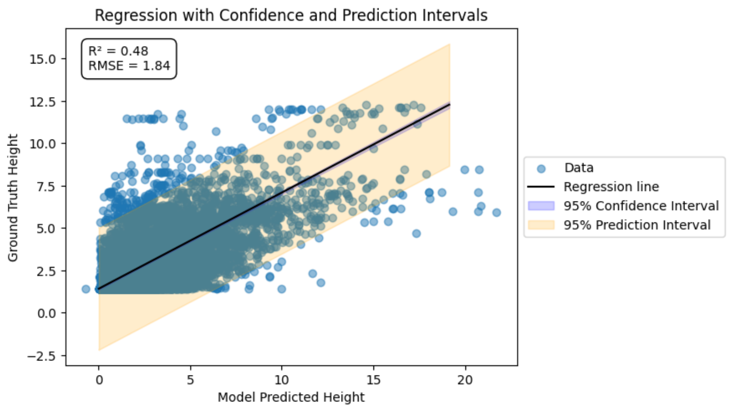


Figure 10: Linear regression result using the maximum value in a 21x21 pixel window around each tree location.

Table 2: Regression results per species after applying neighborhood maximum sampling (21x21 window).

Species	R^2	RMSE
Lindifura	0.92	0.25
Hvítgreni	0.89	0.65
Reynir	0.83	0.38
Rússalerki	0.66	1.30
Rússa- og síberíulerki	0.62	1.67
Alaskaösp	0.57	2.63
Stafafura	0.55	1.68
Gulvíðir	0.54	1.04
Rauðgreni	0.54	2.14
Sitkagreni og -bastarður	0.51	2.00
Gráölur	0.50	2.37
Víðir	0.49	1.13
Sitkagreni	0.49	1.74
Sitkaölur	0.47	0.66
Reyniviður	0.41	1.94
Ilmbjörk	0.36	1.39
Broddgreni	0.35	2.33
Bergfura	0.34	1.51
Hengibirki	0.29	2.03
Evrópulerki	0.24	0.59
Viðja	0.19	1.66
Skógarfura	0.15	1.91
Ilmbjörk - sjálfsáð	0.11	1.33
Jörfavíðir	0.09	0.45
Fjallaþöll	0.07	0.27
Alaskavíðir	0.06	1.50
Blágreni	0.05	1.27
Fjallaþinur	0.02	0.91
Selja	0.01	1.71

Table 2 shows the updated regression results after applying the neighborhood maximum sampling method using a 21x21 pixel window. Compared to the previous analysis, which relied on single-pixel sampling, this approach generally improved the R^2 scores for most species. Notably, *Lindifura* now achieves the highest R^2 of 0.92 with a low RMSE of 0.25 meters. This is a clear improvement over the earlier results, where *Hvítgreni* had the best performance.

Another interesting thing is that several species, such as *Rússalerki*, *Rússa- og síberíulerki*, and *Alaskaösp*, show higher R^2 values than before, suggesting that the neighborhood search helps the model better capture the true tree heights for these species.

However, the improvements are not universal. Some species, especially those with lower R^2 values in the earlier table, remain challenging for the model. While the neighborhood maximum method provides a clear overall benefit, it does not fully resolve the limitations for all species.

Overall, looking at both tables side by side, it seems pretty clear that letting the model search for the highest value in the area around each tree helps. This is especially true for species where the exact tree location might not line up perfectly with the prediction map.

Distribution of Predictions and Errors

To get a better understanding of how our model performs across different parts of Iceland, we visualized the spatial distribution of both the predicted tree heights and the ground truth heights. This also allowed us to generate an error map, showing how far off the predictions are compared to the actual measured tree heights.

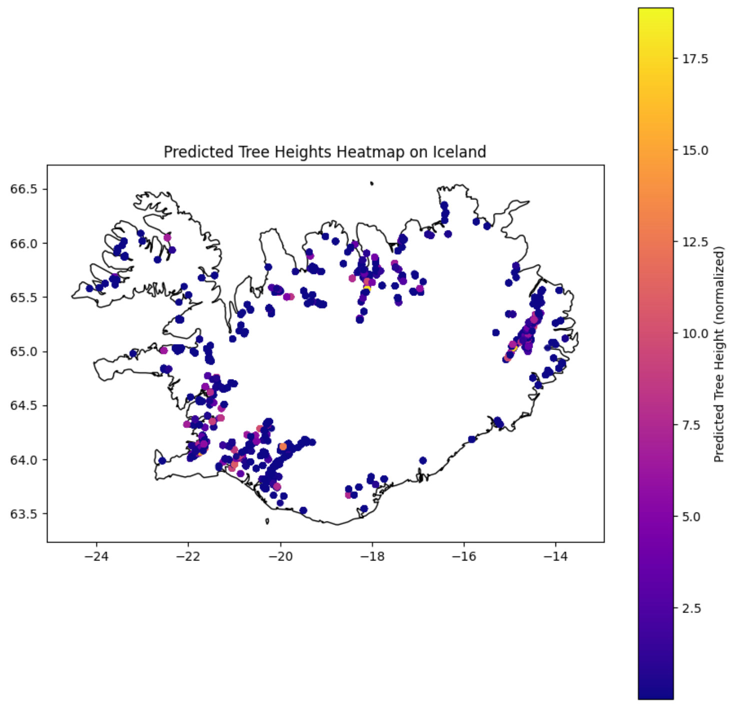

Predicted Tree Heights Heatmap on Iceland
66.5
66.0
65.5
65.0
64.5
64.0
63.5
−24
−22
−20
−18
−16
−14
17.5
15.0
12.5
10.0
7.5
5.0
2.5
Predicted Tree Height (normalized)

Figure 11: Predicted tree heights plotted on Iceland's outline.

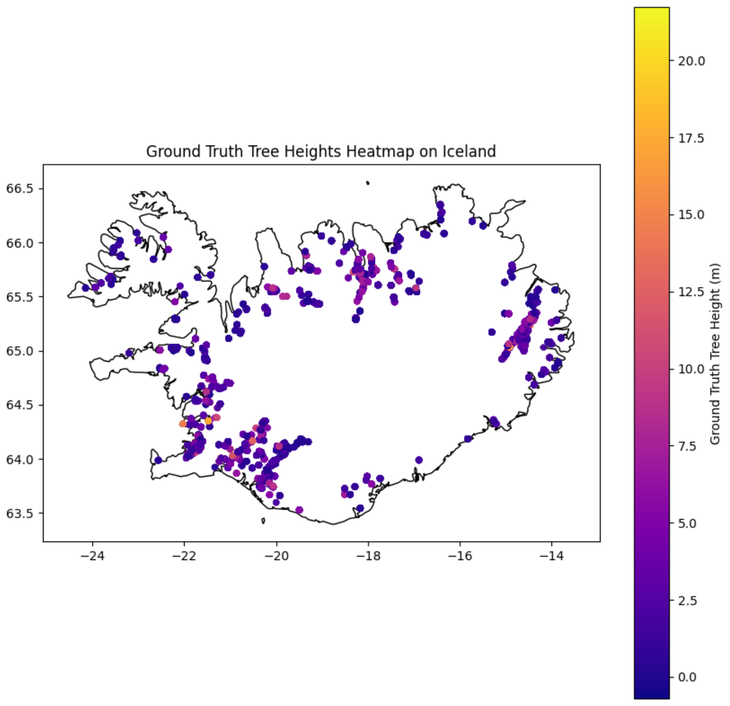


Figure 12: Ground truth tree heights plotted on Iceland's outline.

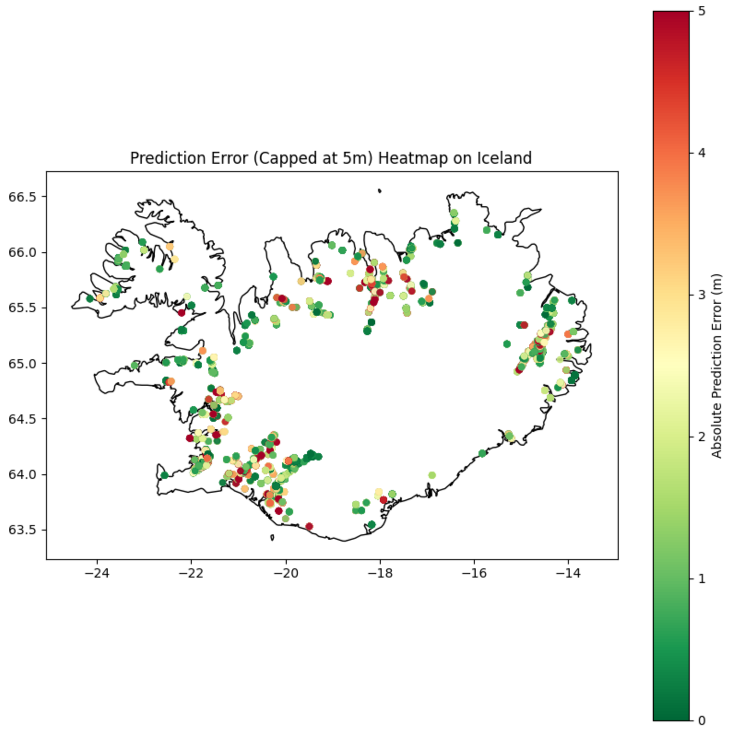

Prediction Error (Capped at 5m) Heatmap on Iceland
66.5
66.0
65.5
65.0
64.5
64.0
63.5
−24
−22
−20
−18
−16
−14
Absolute Prediction Error (m)
0
1
2
3
4
5

Figure 13: Prediction error heatmap, showing how far off the predictions are compared to ground truth. Green areas indicate good alignment, while red areas show larger errors. The error scale is capped at 5 meters to highlight the typical error range.

Looking at these maps, it becomes clear that while the general distribution of predicted and actual tree heights matches reasonably well, there are localized areas with higher errors. The error map shows that although the model performs well in many areas (green dots), there are clusters where the predictions are off by several meters (yellow to red dots). This spatial overview helps us identify regions where the model might need improvement or where the input data might be less reliable.

References

[Tolan et al., 2024] Tolan, J., Yang, H.-I., Nosarzewski, B., Couairon, G., Vo, H. V., Brandt, J., Spore, J., Majumdar, S., Haziza, D., Vamaraju, J., Moutakanni, T., Bojanowski, P., Johns, T., White, B., Tiecke, T., and Couprie, C. (2024). Very high resolution canopy height maps from rgb imagery using self-supervised vision transformer and convolutional decoder trained on aerial lidar. *Remote Sensing of Environment*, 300:113888.