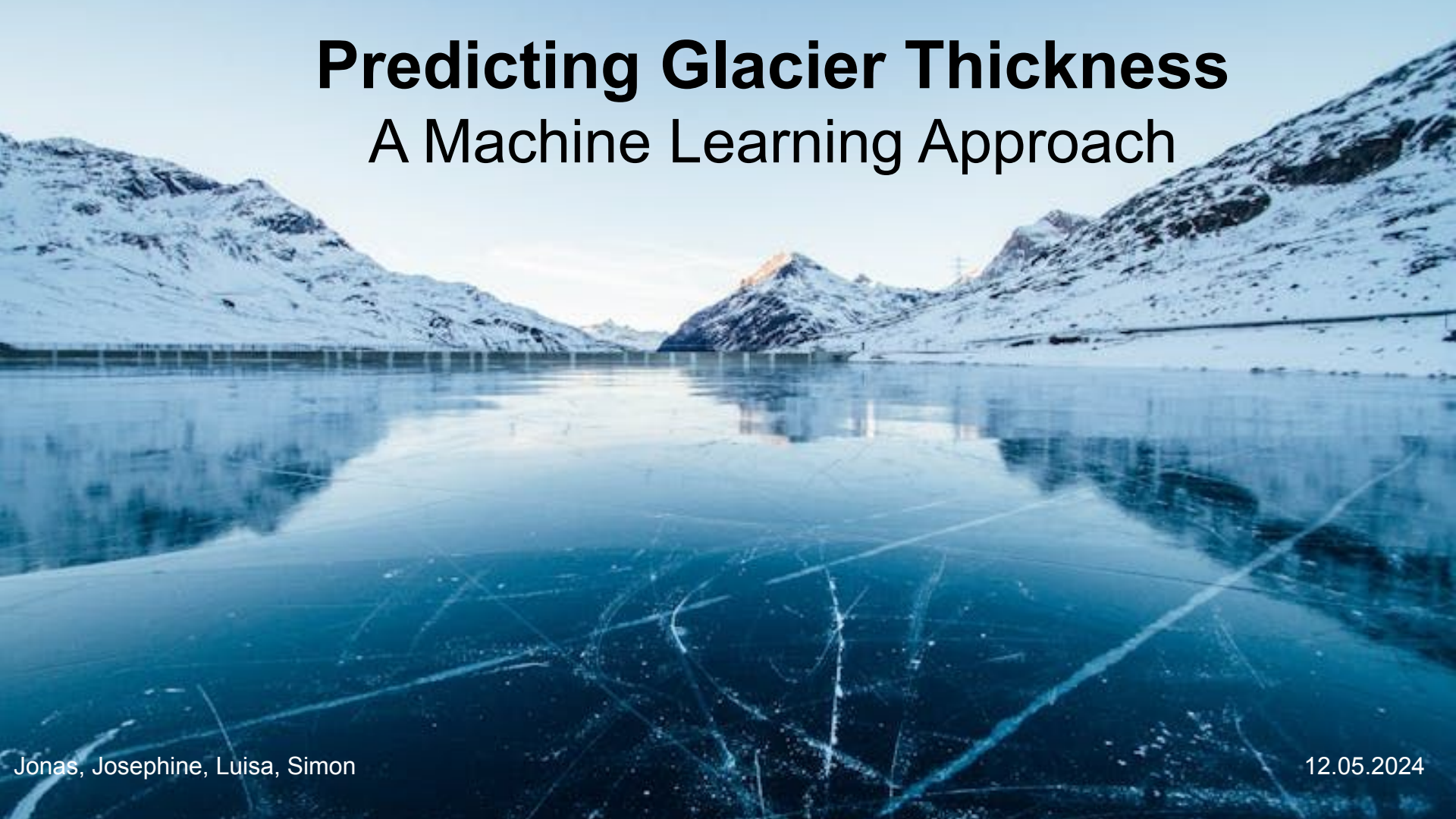
Predicting Glacier Thickness

A Machine Learning Approach



Jonas, Josephine, Luisa, Simon

12.05.2024

Overview

- Introduction, motivation and goal
- Data
- Method: Machine Learning algorithms
  - preprocessing + SHAP
  - BDT + NN
  - CNN
- Results
- Conclusion
- Future directions

Glacier definition: a large mass of ice that moves slowly

Goal of this project

Determine thickness of glaciers

Beat the initial model

Implement a convolutional neural network

Construct a model that achieves a minimum MAE

Motivation: Importance of predicting glacier thickness

Melting glaciers contribute significantly to global sea level rise

Glaciers provide essential freshwater resources

Data Collection

Sources of data (satellite images, field measurements, features from various sources)

data	20x20 grid	100x100 grid	original dataset	Glaciers (Images)
original amount of measurements/images	81.290 (75 MB)	337.367 (309 MB)	3.767.954 (3.5 GB)	2.321
amount of measurements after replacing 0 m thickness (pre-processing)	73.111	317.015	3.662.321	2.101
amount of measurement of thickness >= 1000 m	194 (0.3 %)*	1.183 (0.3 %)*	34.879 (0.9 %)*	25

* percentage of the amount of measurements after replacing 0 m thickness

Features

local and per-glacier features + calculation of own features - example:

$$\text{slope}_{res} = \sqrt{\text{slope}_{lat}^2 + \text{slope}_{lon}^2}$$

	RGIId	slope_lat	slope_lon	vx	vy	THICKNESS	Area	dist_from_border_km_geom	TermType
0	RGI60-11.00638	-0.462330	-0.813588	1.288977	-5.4352	6.0	0.082	0.026833	0.0
1	RGI60-11.00638	-0.450002	-0.752270	1.288977	-5.4352	11.0	0.082	0.018446	0.0
2	RGI60-11.00638	-0.459016	-0.814325	1.288977	-5.4352	6.0	0.082	0.028935	0.0
3	RGI60-11.00638	-0.460448	-0.814099	1.288977	-5.4352	6.0	0.082	0.027870	0.0
4	RGI60-11.00638	-0.441938	-0.748874	1.288977	-5.4352	11.0	0.082	0.019278	0.0

Data Pre-Processing Approaches

1. Look at the data:
   - No NaN-Values in our chosen features
   - 0 m thicknesses?

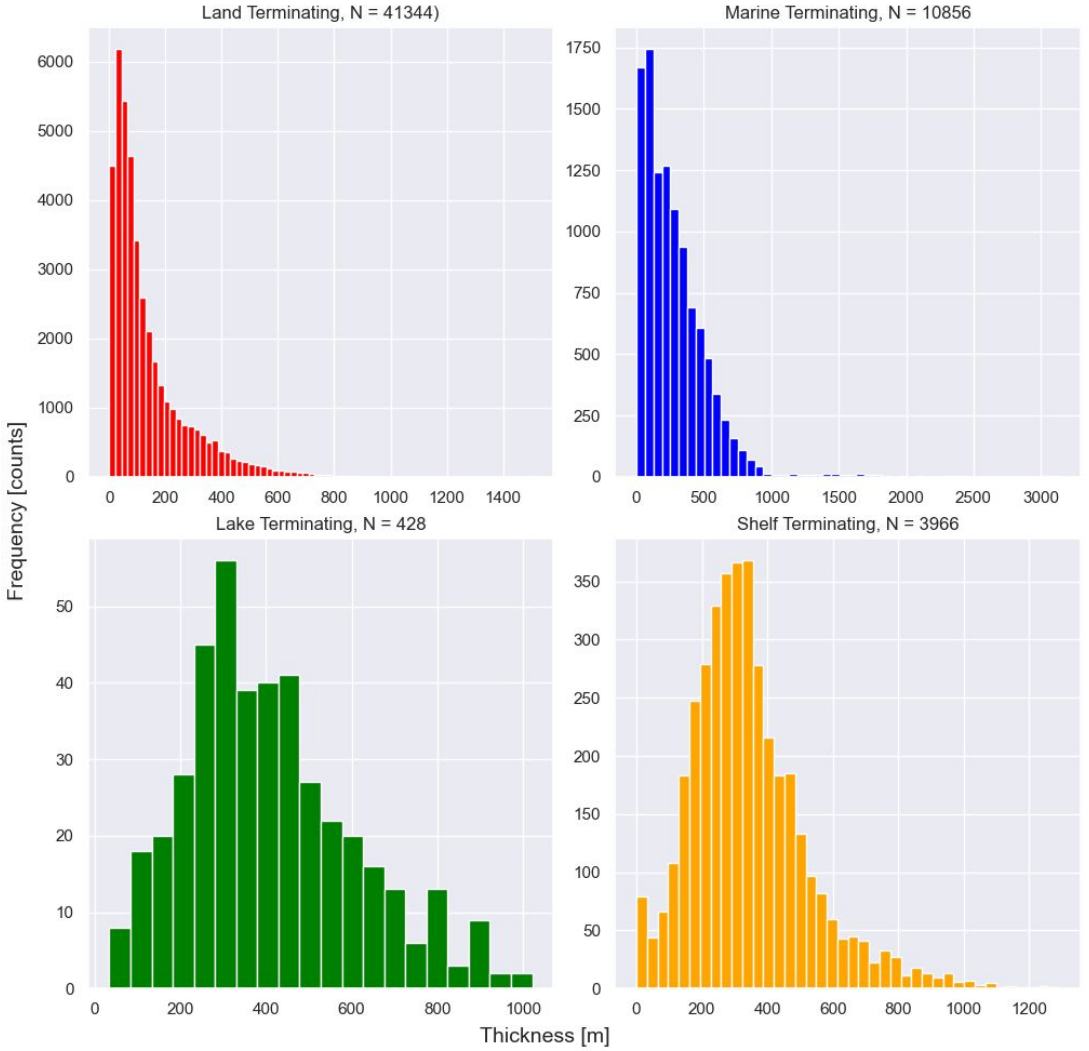
2. Handle 0 m thicknesses:
   2.1 remove all measurements below 1 m thicknesses
   2.2 replace 0 m thicknesses by calculating the mean values of the models

3. Images:
   - masking the colour to get array of 1 & 0

Thickness Distribution

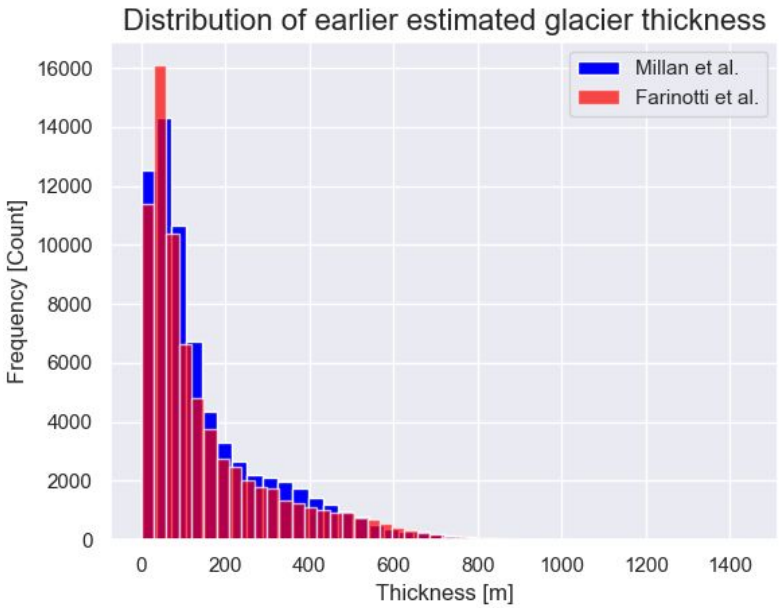
- Depending on the termination type
- A lot of 0 m thicknesses
- Two approaches:
  1. Remove all zeros
  2. Estimate from other studies

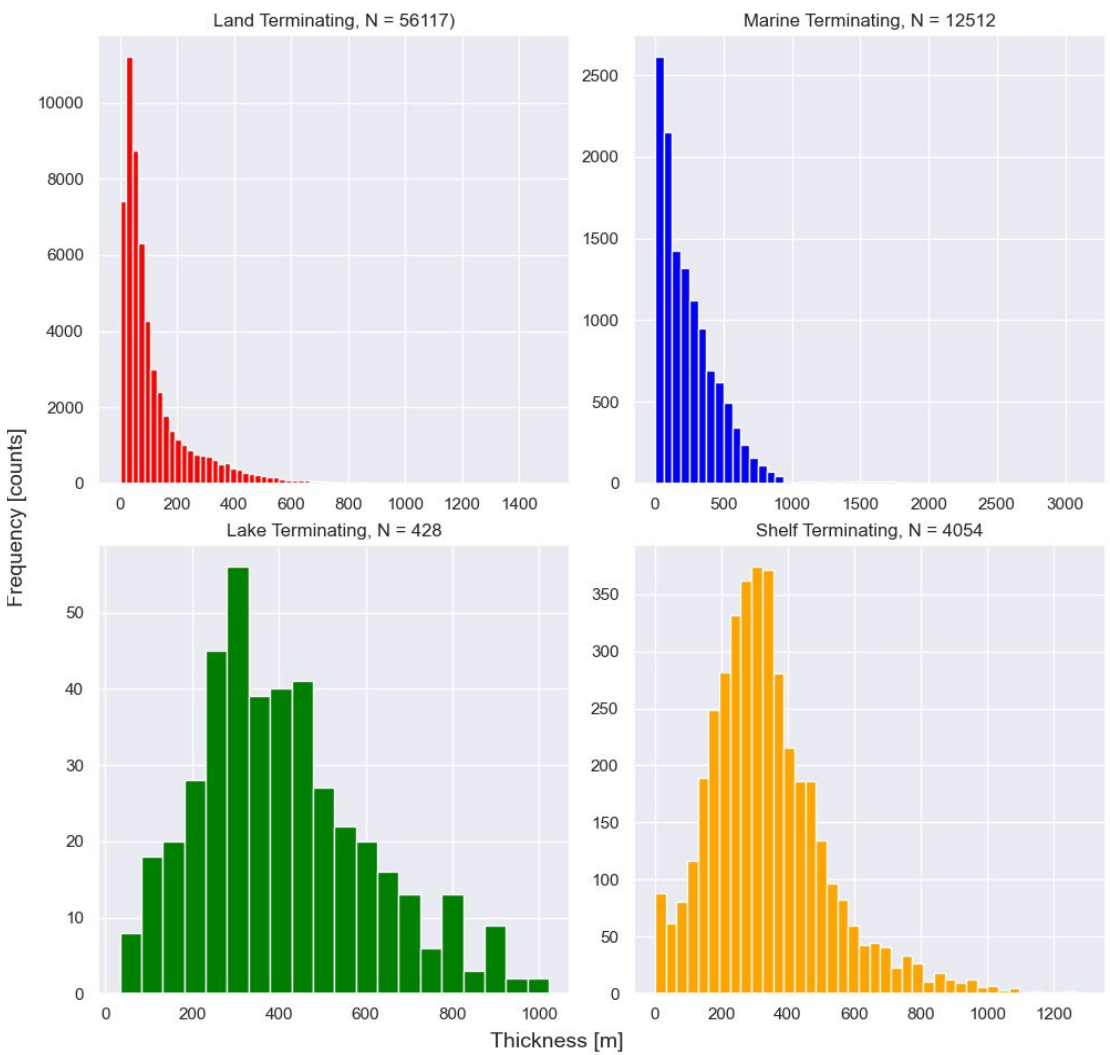
THICKNESS	ESTIMATION_IS_TRUE
1512.21	0
32.33	1
6.89	1
532.01	0



Estimated 0 Thickness

- replace 0 m thickness with the mean value from the models





Machine Learning Methods for Glacier Thickness Predictions

Regression BDT

GOAL: Regional training into global prediction

Simple initial model (Niccolo)

Alternative to LightGBM

Hyperparameter Optimization

Regression Neural Network

GOAL: Beyond BDT. Additional information from images

Alternative to both BDT

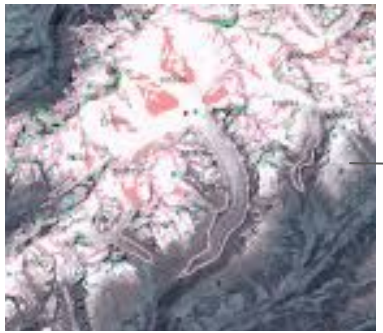
RandomSearchCV for Hyperparameter Optimization

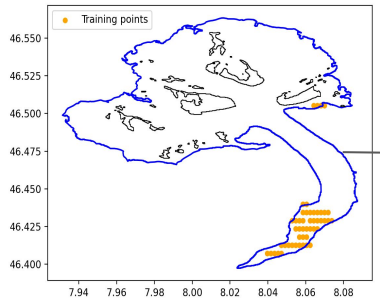
Loss function: Mean Absolute Error (MAE)

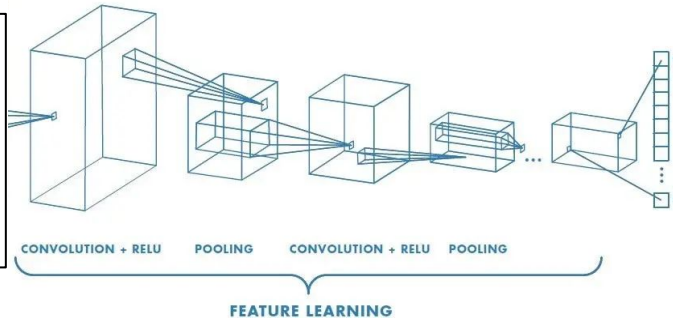
CNN Approaches

- Motivation:

  Obtain additional information on the shape of the glaciers and link it to the tabular data. Is there an improvement in the prediction of thickness?
- Train a CNN with our images (size 64x64)
- Use a pre-trained CNN (VGG16)

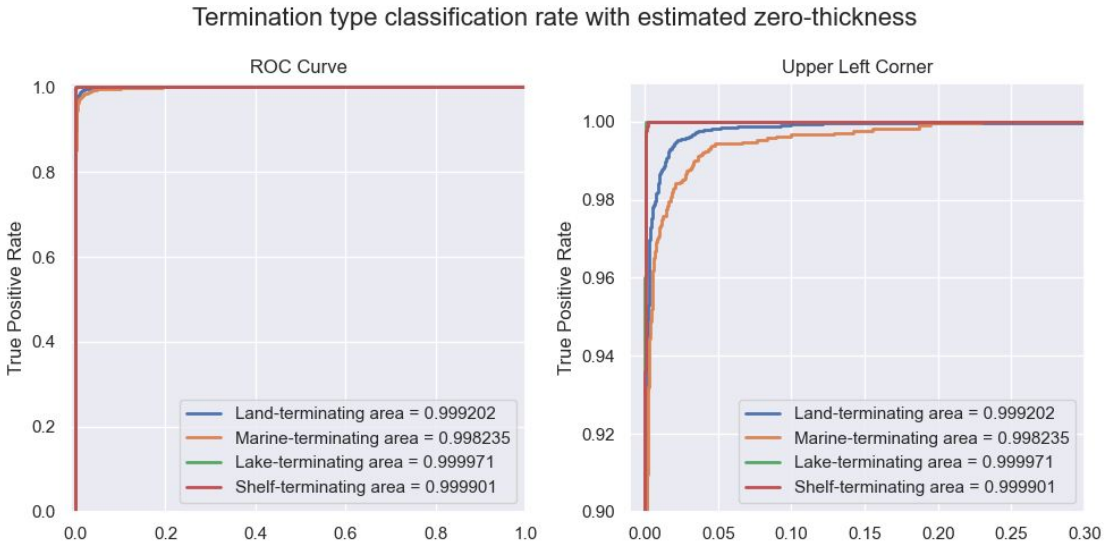




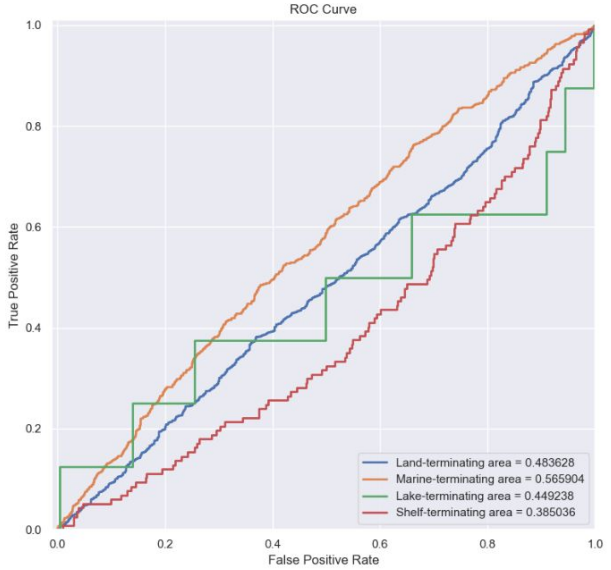


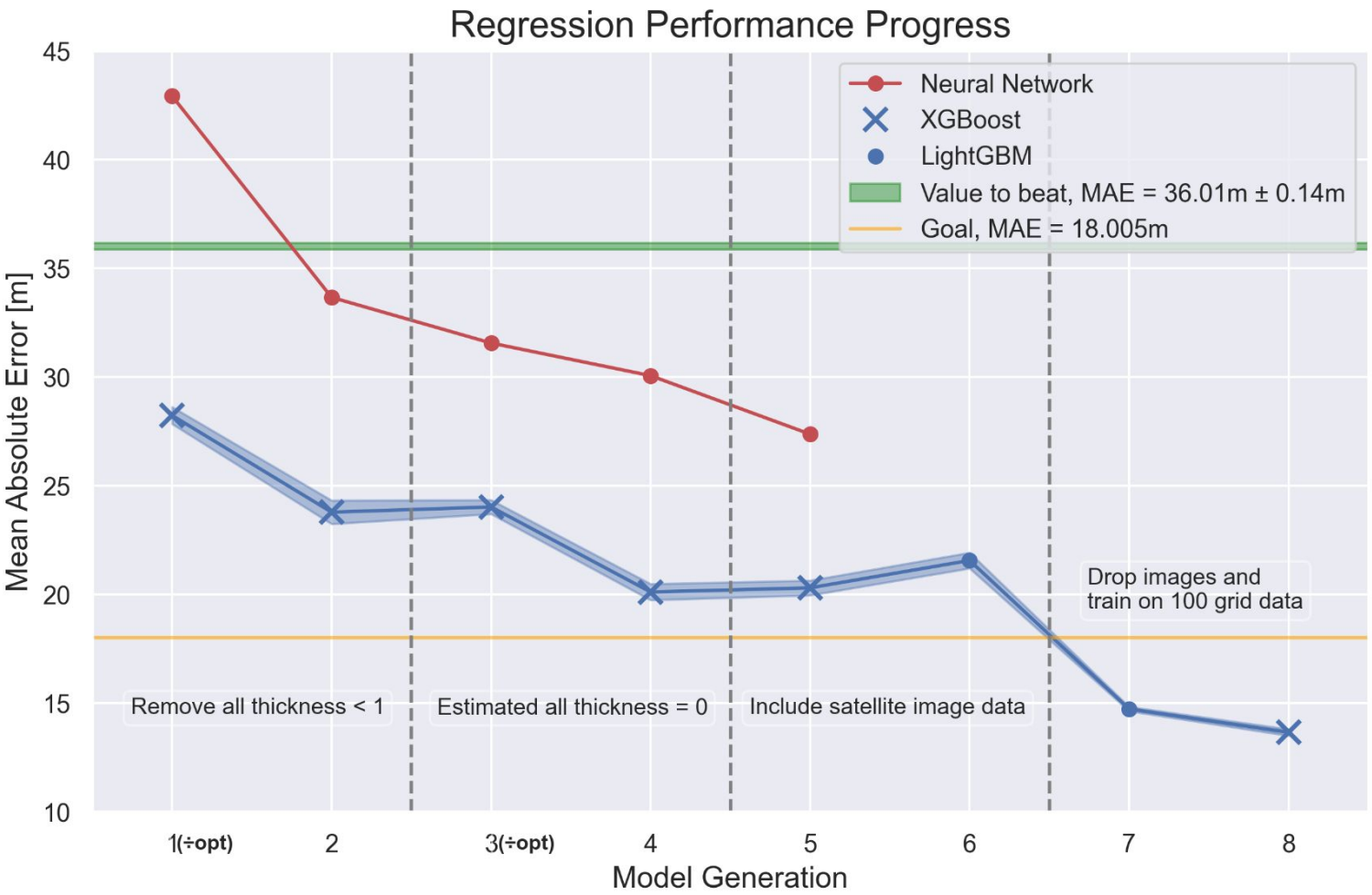
Side-Quest: Classification Termination Type

NN **without** pictures



NN with **only** pictures



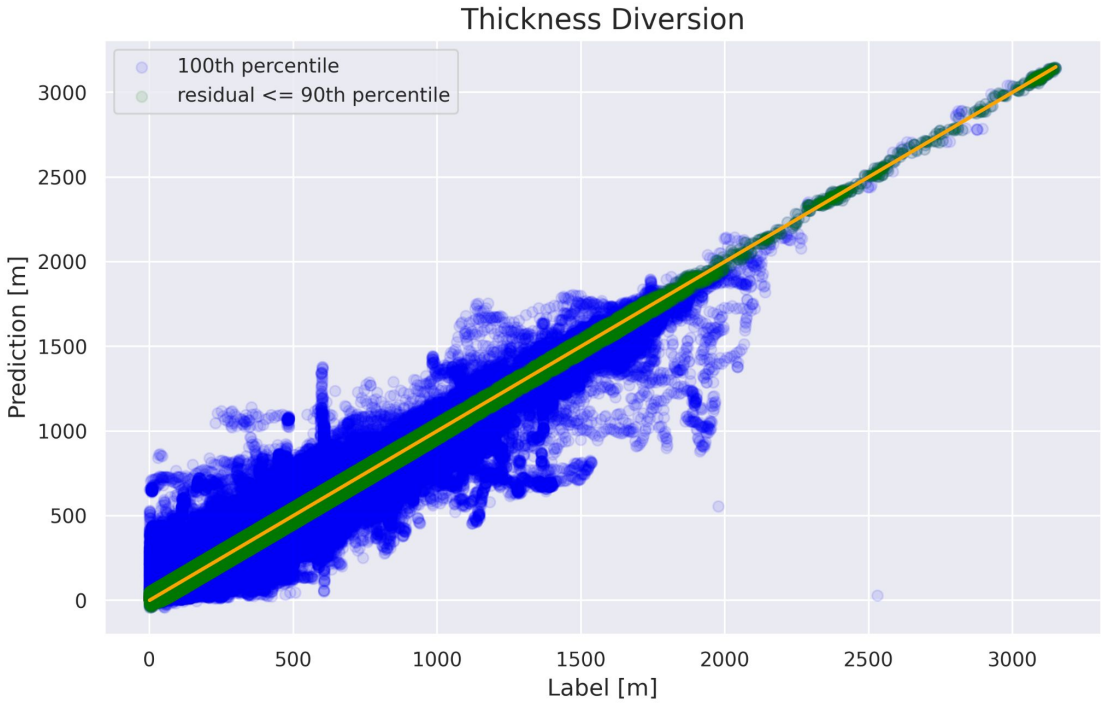

Regression Performance Progress
Neural Network
XGBoost
LightGBM
Value to beat, MAE = 36.01m ± 0.14m
Goal, MAE = 18.005m
Mean Absolute Error [m]
45
40
35
30
25
20
15
10
Remove all thickness < 1
Estimated all thickness = 0
Include satellite image data
Drop images and
train on 100 grid data
1(÷opt)
2
3(÷opt)
4
5
6
7
8
Model Generation

Prediction of the Original Data

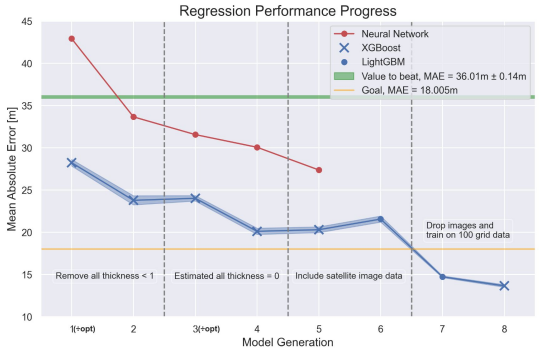
Trained by our best model.

Mae Cross-Validation: 13.65 +- 0.07 m

- data: 100x100 grid
- pre-processing: replace 0 m thicknesses by mean values of the models
- method: XGBRegressor + Optuna

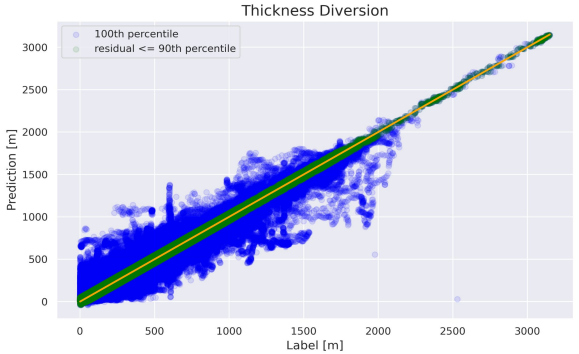


Conclusion



- CNNs don’t improve the prediction.
- XGBoost win the competition.
- It makes sense to include the interpolated thicknesses of the models in the approach (replace 0 m thickness).

- 0 m thicknesses
- CNN feature extraction



- Prediction of large thicknesses surprisingly good.
- Underprediction as often as overprediction.

Future Prospects

What we can influence:

- Work with correctly scaled satellite images.
- Add location information to satellite pictures.
- Add additional features: Temperature - could it help?

What we can’t influence:

- Take more data so that we don’t have the unbalanced dataset of a lot of low thicknesses.
- Steady state system.

Appendix

Workload distribution

All work was done equally in the group

- When typing we took turns on the keyboard

Structure of the Appendix

1. Read Me - Python Codes in github
2. CNN
   - VGG16 (2 slides)
   - Simple homemade CNN (2 slides)
3. XGBRegressor
   - descriptions/thoughts (3 slides)
   - 20x20 grid (6 slides)
   - 100x100 grid (2 slides)
   - deep thicknesses (2 slides)
   - original data prediction (1 slide)
4. LightGBM
5. Classification
6. NN
7. Clustering

Read Me - Python Codes in github

‘XGBRegressor.ipynb’ - XGBoost regression model + optimization
‘Optuna_lightGBM.ipynb’ - LightGBM regression model
‘CNN_VGG16.ipynb’ - pre-trained CNN VGG16 feature extraction
‘Clustering_PCA.ipynb’ + ‘Clustering_tsne.ipynb’ - the clustering plots
‘CNN_classification.ipynb’ - NN + CNN & CNN + BDT - regression (+classification)

CNN

Pre-Trained CNN VGG16

Motivation: See if a trained CNN with many images from other areas is better at extracting features or if it is overtrained. Code: CNN-VGG16.ipynb

1. Pre-Processing: 64x64 Grayscale Glacier Images to 64x64 RGB Images
2. Extract the features with the following architecture (very complex):

Layer (type)	Output Shape	Param #
input_4 (InputLayer)	[(None, 64, 64, 3)]	0
block1_conv1 (Conv2D)	(None, 64, 64, 64)	1792
block1_conv2 (Conv2D)	(None, 64, 64, 64)	36928
block1_pool (MaxPooling2D)	(None, 32, 32, 64)	0
block2_conv1 (Conv2D)	(None, 32, 32, 128)	73856
block2_conv2 (Conv2D)	(None, 32, 32, 128)	147584
block2_pool (MaxPooling2D)	(None, 16, 16, 128)	0
block3_conv1 (Conv2D)	(None, 16, 16, 256)	295168
block3_conv2 (Conv2D)	(None, 16, 16, 256)	590080
block3_conv3 (Conv2D)	(None, 16, 16, 256)	590080
block3_pool (MaxPooling2D)	(None, 8, 8, 256)	0
block4_conv1 (Conv2D)	(None, 8, 8, 512)	1180160
block4_conv2 (Conv2D)	(None, 8, 8, 512)	2359808
block4_conv3 (Conv2D)	(None, 8, 8, 512)	2359808
block4_pool (MaxPooling2D)	(None, 4, 4, 512)	0
block5_conv1 (Conv2D)	(None, 4, 4, 512)	2359808
block5_conv2 (Conv2D)	(None, 4, 4, 512)	2359808
block5_conv3 (Conv2D)	(None, 4, 4, 512)	2359808
block5_pool (MaxPooling2D)	(None, 2, 2, 512)	0

Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0

Pre-Trained CNN VGG16 - Thoughts

- Too many layers:
  CNNs learn more complex/abstract features, but we only need to recognize edges and a few CNN layers would be sufficient for that

- Overfitting:
  The more complex, the more likely overfitting is. Therefore, it might not work well for the unseen glacier images

- Easy to apply:
  We just need to create the RGB images and specify which should be our output layer.

CNN trained

Use rotated mirrored pictures for training.

Layer (type)	Output Shape	Param #	Connected to
input_16 (InputLayer)	[(None, 64, 64, 1)]	0	[]
conv2d_56 (Conv2D)	(None, 62, 62, 32)	320	['input_16[0][0]']
max_pooling2d_56 (MaxPooling2D)	(None, 31, 31, 32)	0	['conv2d_56[0][0]']
conv2d_57 (Conv2D)	(None, 29, 29, 64)	18496	['max_pooling2d_56[0][0]']
max_pooling2d_57 (MaxPooling2D)	(None, 14, 14, 64)	0	['conv2d_57[0][0]']
conv2d_58 (Conv2D)	(None, 12, 12, 128)	73856	['max_pooling2d_57[0][0]']
max_pooling2d_58 (MaxPooling2D)	(None, 6, 6, 128)	0	['conv2d_58[0][0]']
conv2d_59 (Conv2D)	(None, 4, 4, 256)	295168	['max_pooling2d_58[0][0]']
max_pooling2d_59 (MaxPooling2D)	(None, 2, 2, 256)	0	['conv2d_59[0][0]']
flatten_14 (Flatten)	(None, 1024)	0	['max_pooling2d_59[0][0]']
dense_22 (Dense)	(None, 64)	65600	['flatten_14[0][0]']
input_17 (InputLayer)	[(None, 68)]	0	[]
concatenate_3 (Concatenate)	(None, 132)	0	['dense_22[0][0]', 'input_17[0][0]']
dense_23 (Dense)	(None, 132)	17556	['concatenate_3[0][0]']
dense_24 (Dense)	(None, 145)	19285	['dense_23[0][0]']
dense_26 (Dense)	(None, 96)	14016	['dense_24[0][0]']
dense_27 (Dense)	(None, 1)	97	['dense_26[0][0]']

Total params: 504,394
Trainable params: 504,394
Non-trainable params: 0

CNN trained

Import the weights from previous model

This is then fed into the BDT

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, 64, 64, 1)]       0

conv2d_4 (Conv2D)            (None, 62, 62, 32)        320

max_pooling2d_4 (MaxPooling  (None, 31, 31, 32)        0
2D)

conv2d_5 (Conv2D)            (None, 29, 29, 64)        18496

max_pooling2d_5 (MaxPooling  (None, 14, 14, 64)        0
2D)

conv2d_6 (Conv2D)            (None, 12, 12, 128)       73856

max_pooling2d_6 (MaxPooling  (None, 6, 6, 128)         0
2D)

conv2d_7 (Conv2D)            (None, 4, 4, 256)         295168

max_pooling2d_7 (MaxPooling  (None, 2, 2, 256)         0
2D)

flatten_1 (Flatten)          (None, 1024)              0

=================================================================
Total params: 387,840
Trainable params: 387,840
Non-trainable params: 0
_________________________________________________________________
```

XGBoost

XGBRegressor

Motivation: Can this method improve our regression after the good results in the initial project? Can it beat lightGBM?
Code: XGBRegressor.ipynb

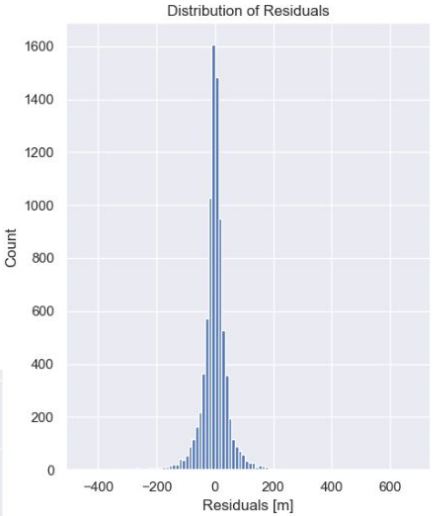
Run the training dataset of 20x20 grid:
(first without hyperparameter optimization, second Optuna, trials = 50, early-stopping always include, rounds = 5)

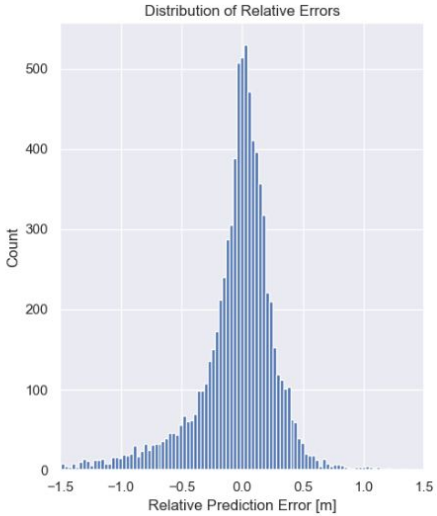
- remove any thickness less than 1 m (a)
- replace each 0 m thickness with an average of the two models and remove everything less than or equal to 0 m (b)
- Add the information from the satellite images
  - from VGG16 (c)
  - from the self-trained CNN (d) - at the end we only did it with lightGBM

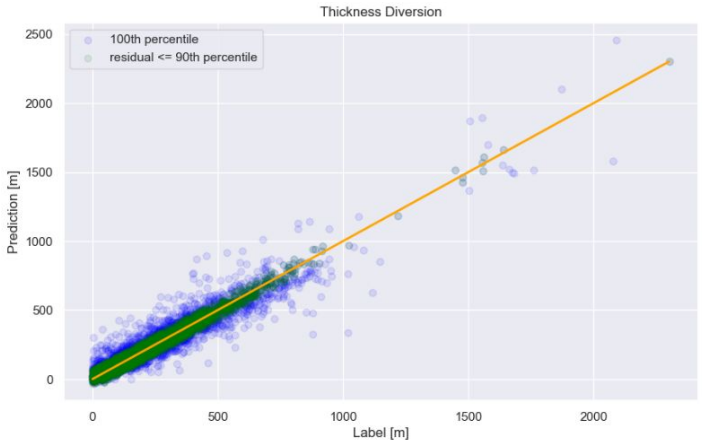
Use the best regression approach from the 20x20 grid dataset and run it on the training dataset of 100x100 grid.

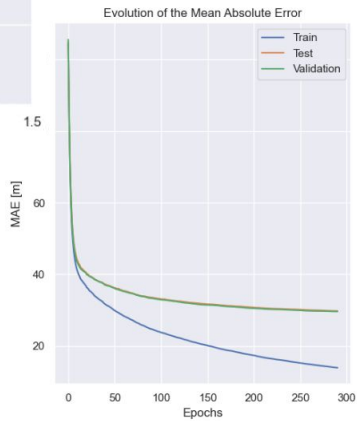
XGBRegressor - 20x20 grid (a) without Hyperparameter Optimization

Training Mae: 14.03 m
Test Mae: 29.78 m
Valid Mae: 29.63 m
Mae Cross-Validation: 28.85 +- 0.24 m



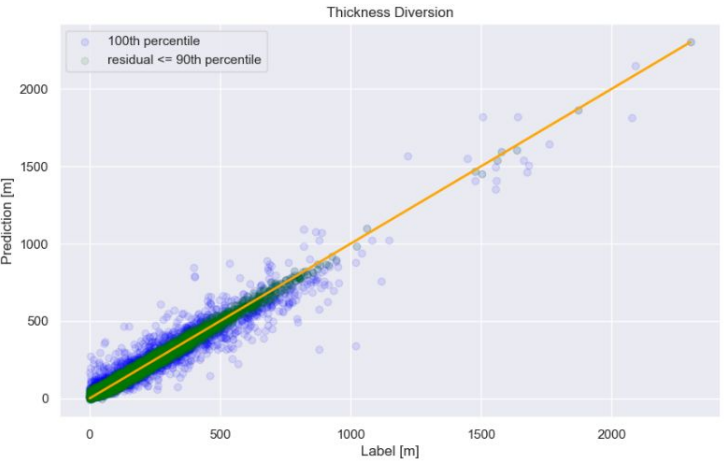


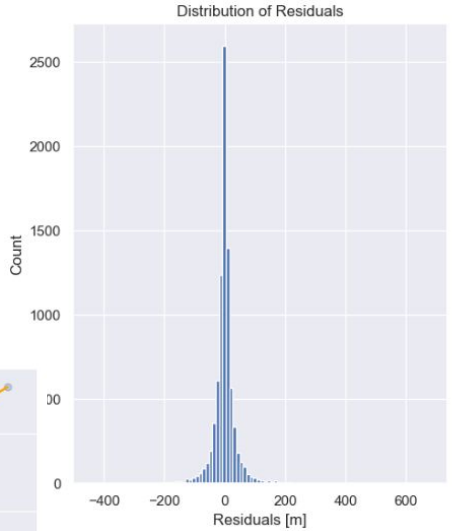


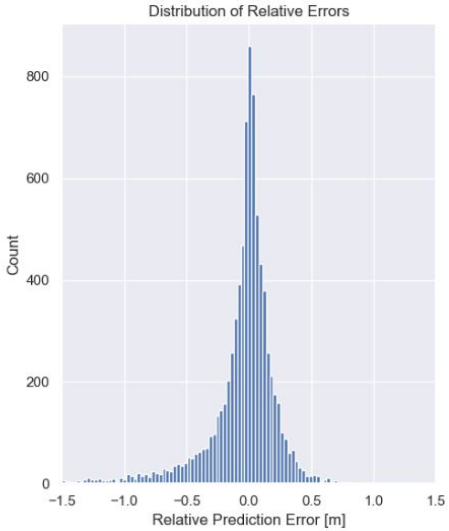


XGBRegressor - 20x20 grid (a) with Hyperparameter Optimization

Training Mae: 2.93 m
Test Mae: 23.24 m
Valid Mae: 23.17 m
Mae Cross-Validation: 23.92 +- 0.17 m

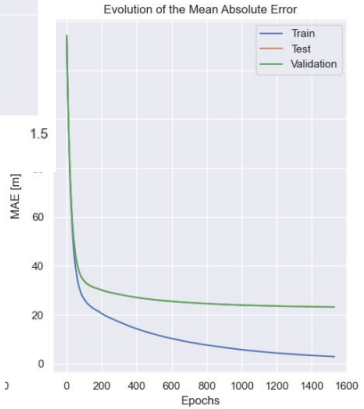






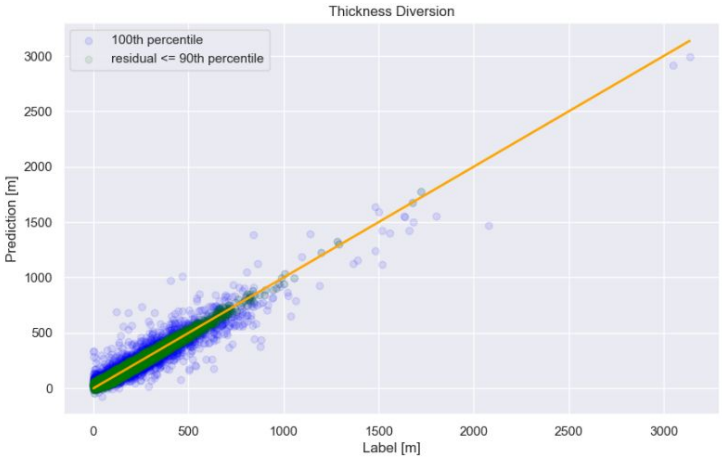
Best Hyperparameters:

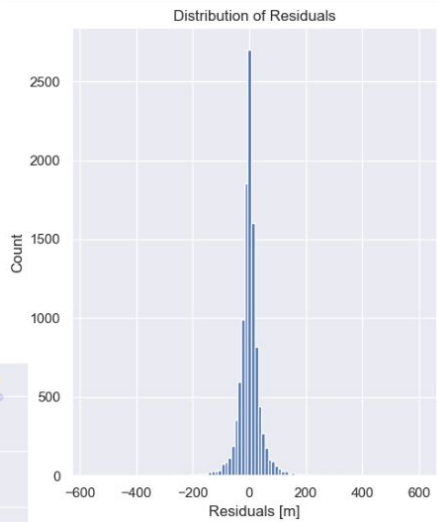
{‘max_depth’: 10, ‘learning_rate’: 0.037, ‘min_child_weight’: 1, ‘subsample’: 0.0.506, ‘gamma’: 3.934, ‘alpha’: 0.263, ‘lambda’: 7.783}

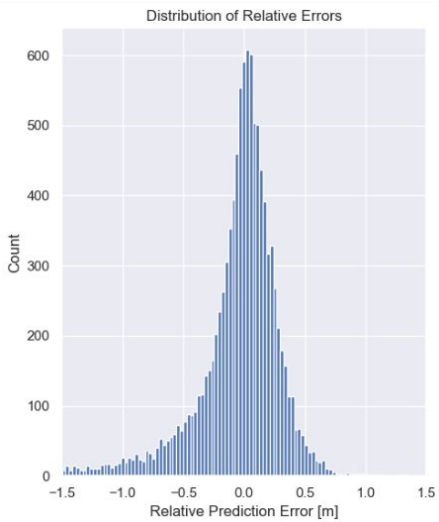


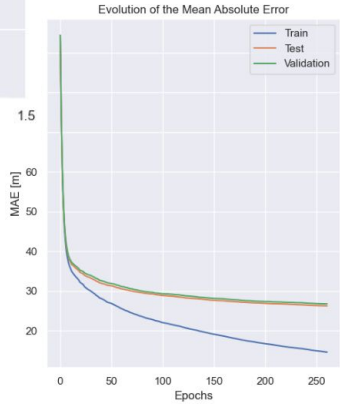
XGBRegressor - 20x20 grid (b) without Hyperparameter Optimization

Training Mae: 14.80 m
Test Mae: 26.32 m
Valid Mae: 26.79 m
Mae Cross-Validation: 25.15 +- 0.16 m



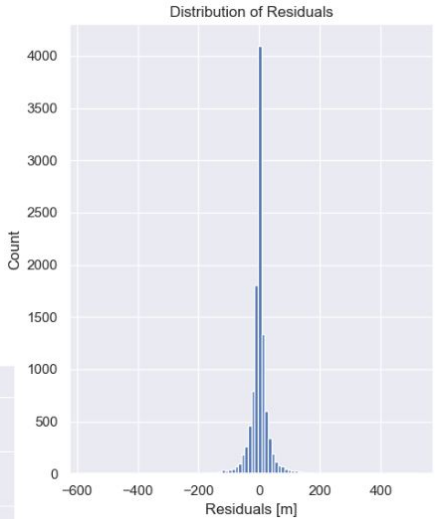


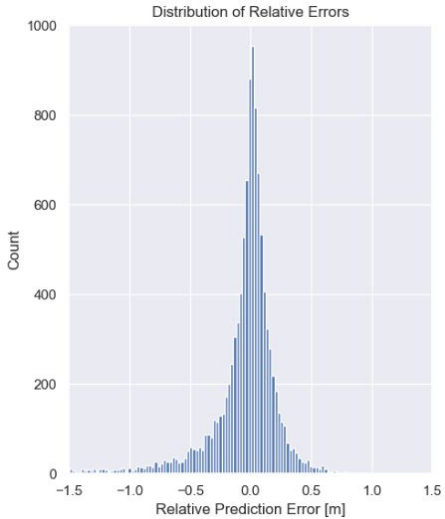


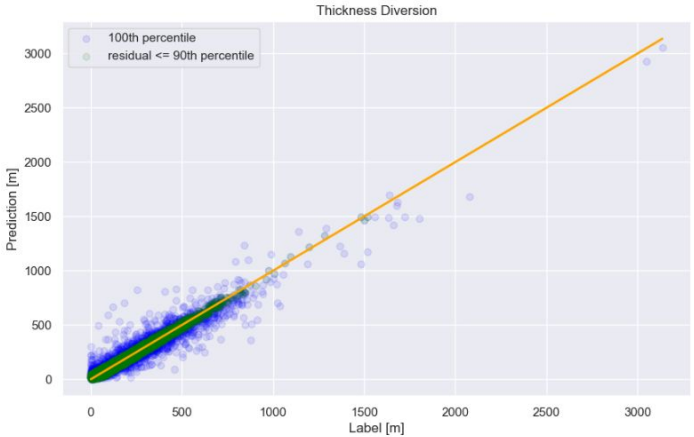


XGBRegressor - 20x20 grid (b) with Hyperparameter Optimization

Training Mae: 1.23 m
Test Mae: 20.29 m
Valid Mae: 20.54 m
Mae Cross-Validation: 21.19 +- 0.10 m

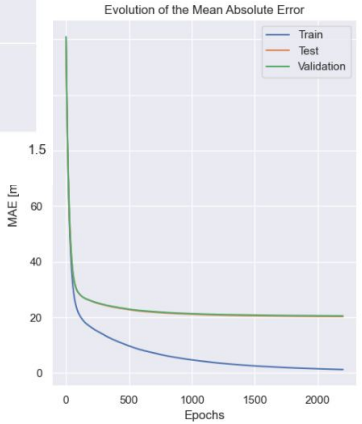






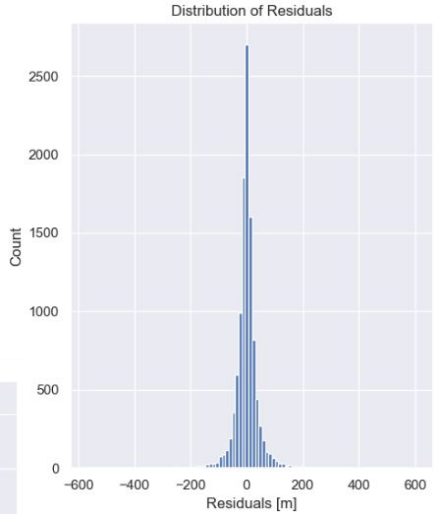
Best Hyperparameters:

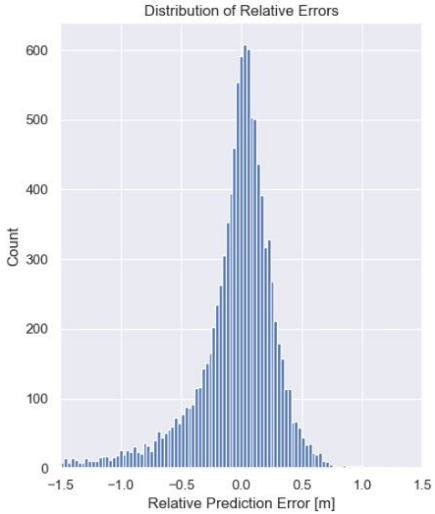
{‘max_depth’: 10, ‘learning_rate’: 0.037, ‘min_child_weight’: 2, ‘subsample’: 0.814, ‘gamma’: 2.414, ‘alpha’: 0.749, ‘lambda’: 5.964}

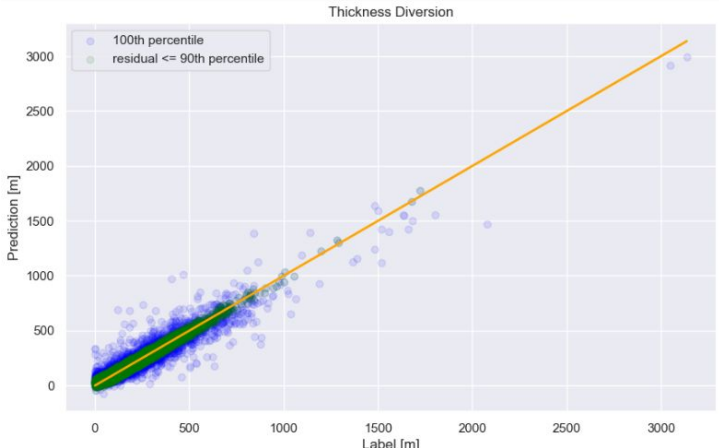


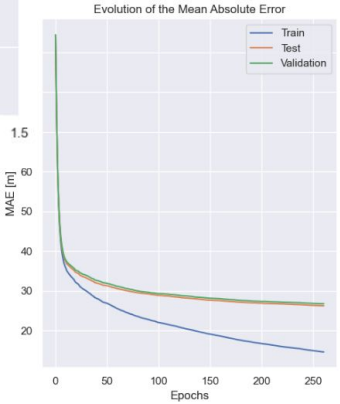
XGBRegressor - 20x20 grid (c) without Hyperparameter Optimization

Training Mae: 14.80 m
Test Mae: 26.32 m
Valid Mae: 26.79 m
Mae Cross-Validation: 25.15 +- 0.16 m



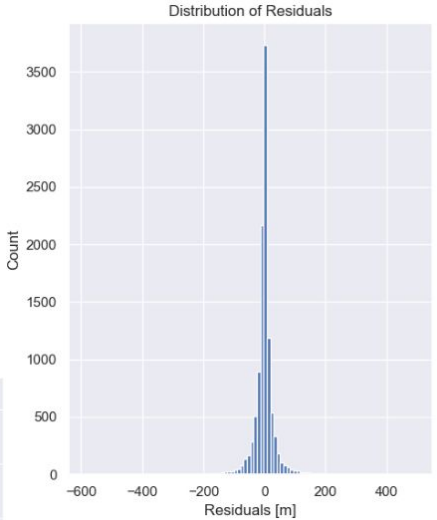


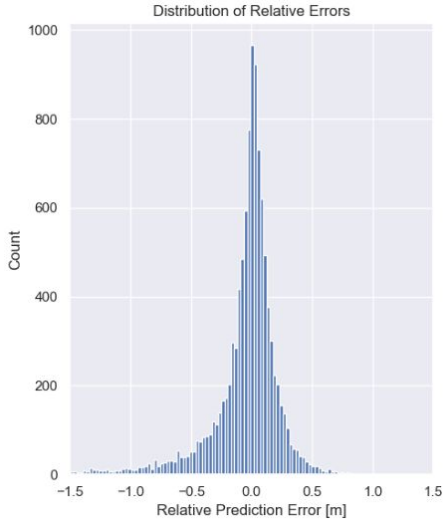


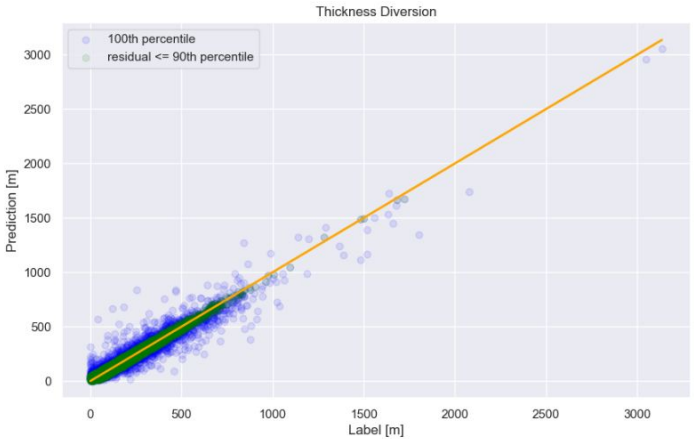


XGBRegressor - 20x20 grid (c) with Hyperparameter Optimization

Training Mae: 2.52 m
Test Mae: 20.65 m
Valid Mae: 20.81 m
Mae Cross-Validation: 21.21 +- 0.13 m

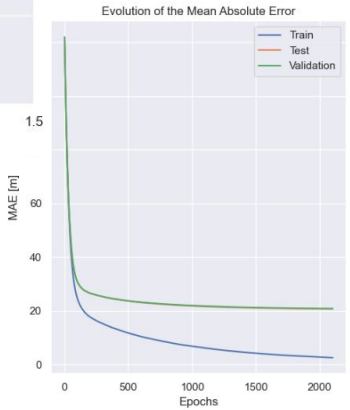






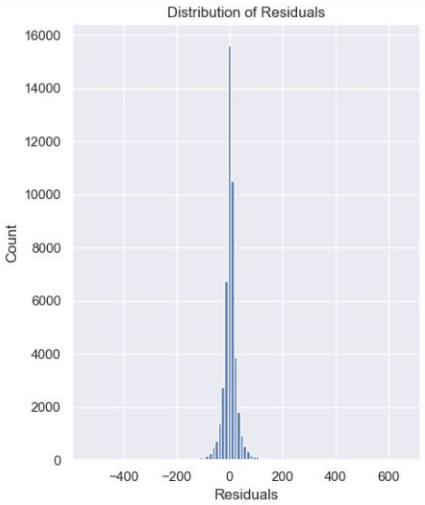
Best Hyperparameters:

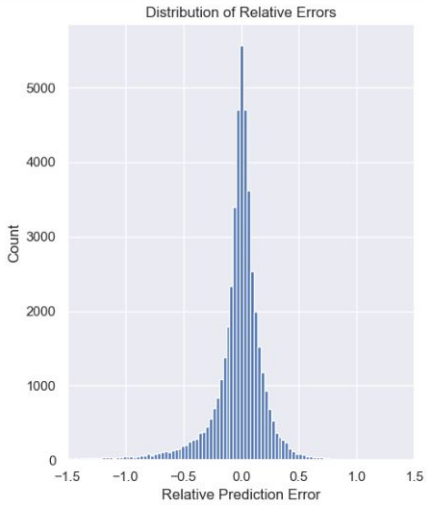
{‘max_depth’: 10, ‘learning_rate’: 0.028, ‘min_child_weight’: 5, ‘subsample’: 0.820, ‘gamma’: 0.440, ‘alpha’: 0.139, ‘lambda’: 3.007}

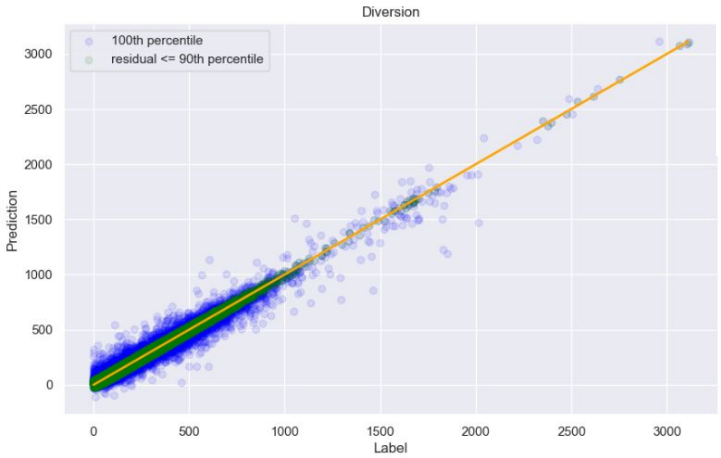


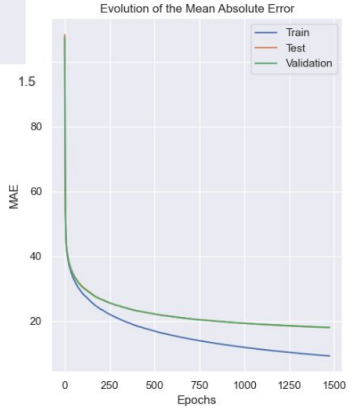
XGBRegressor - 100x100 grid without Hyperparameter Optimization

Training Mae: 9.21 m
Test Mae: 17.99 m
Valid Mae: 18.02 m
Mae Cross-Validation: 17.19 +- 0.06 m



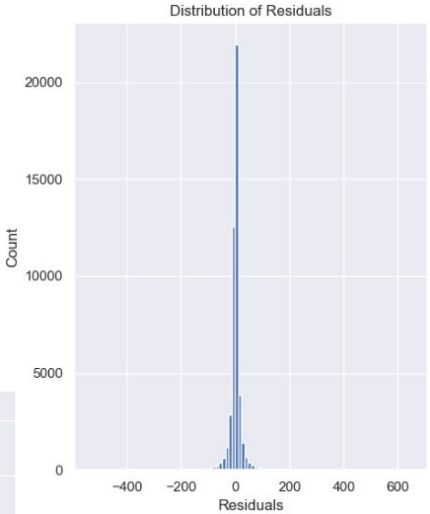


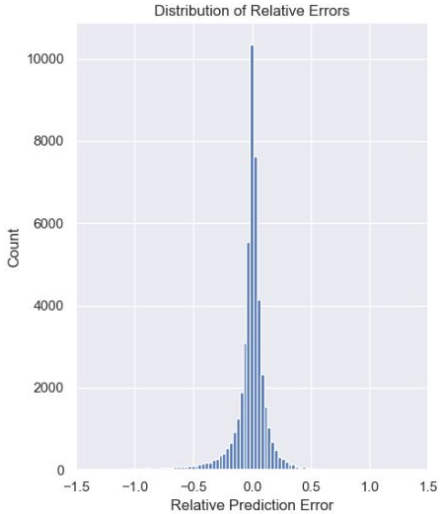


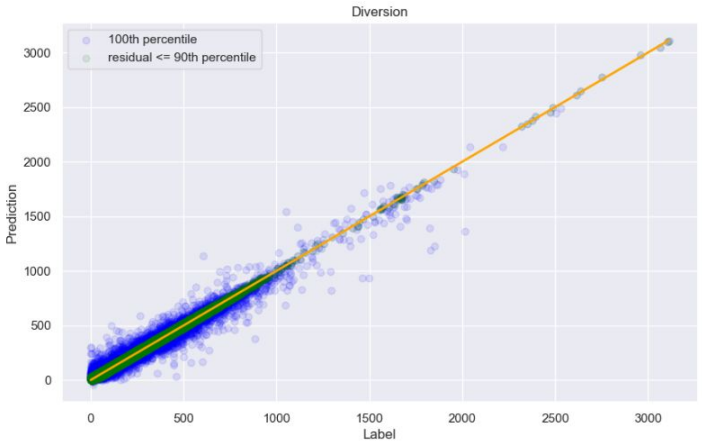


XGBRegressor - 100x100 grid with Hyperparameter Optimization

Training Mae: 1.23 m
Test Mae: 12.34 m
Valid Mae: 12.45 m
Mae Cross-Validation: 13.65 +- 0.07 m

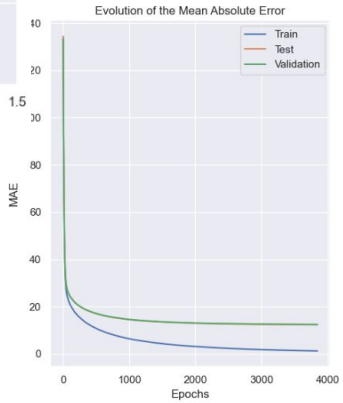






Best Hyperparameters:

{‘max_depth’: 10, ‘learning_rate’: 0.071, ‘min_child_weight’: 8, ‘subsample’: 0.808, ‘gamma’: 2.303, ‘alpha’: 0.698, ‘lambda’: 5.009}



XGBRegressor - Conclusion

- Adding information from satellite images does not improve the prediction.

  This may be because our features also come from satellite images and the new information is not as dominant. Also, the Shap values for the features show that the most important features are the additional calculated slopes and only area and distance to boundary are features that could also be helpful from the satellite images. But we have already included them without the features from the images. So no new important features were added. And there is also no information of the measurement in the image when we train the CNN.
- No noise or distortion of the regression due to information from satellite images despite temporal discrepancy.
- The approach of replacing the 0 m thickness with the mean values from the models is successful. Leads to a much better result.
- No overtraining.

XGBRegressor - Deep Thicknesses

- Use the metadata19.csv dataset to train on deep thicknesses (>= 1000 m):
- pre-processing:

    - looking for NaN-values
    - replace each 0 m thickness with an average of the two models and remove everything less than or equal to 0 m
    - use the same features as for the 20x20 trainin

→ amount of datapoints/measurements: 34879

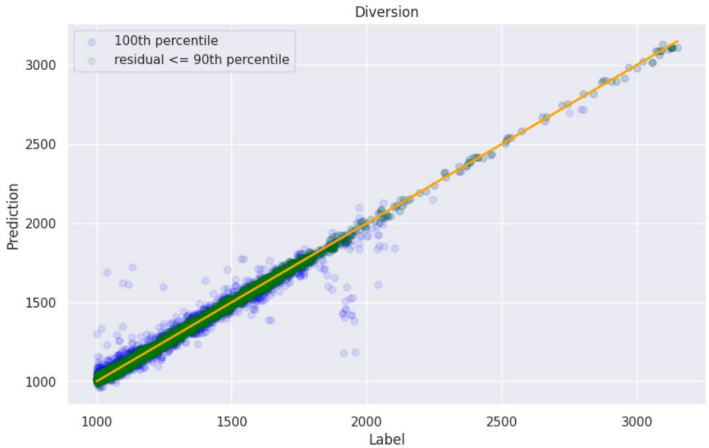
- first without hyperparameter optimization, second Optuna, trials = 50, early-stopping always included, rounds = 5

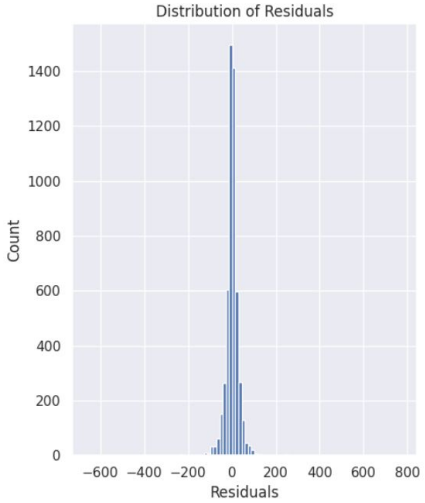
XGBRegressor - Deep Thicknesses without Hyperparameter Optimization

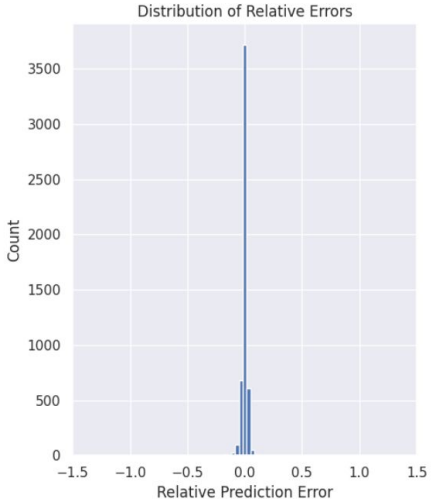
Training Mae: 16.48 m
Test Mae: 21.39 m
Valid Mae: 21.04 m
Mae Cross-Validation: 14.23 +- 0.24 m
Training Time: 1.74 s

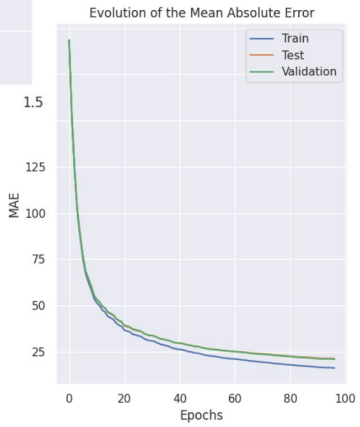
WHAT?

Cross-Validation is so small!!!







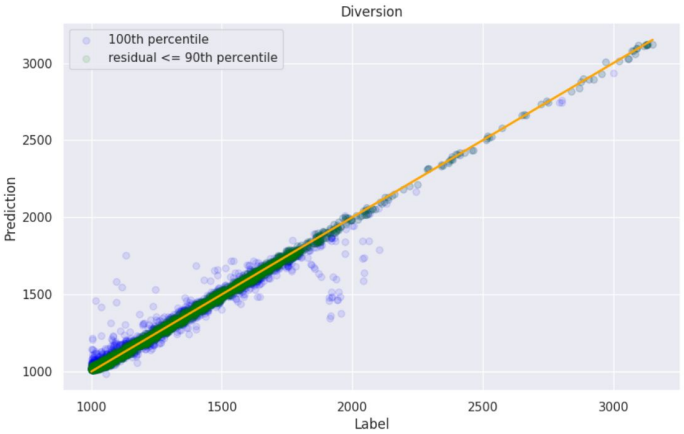


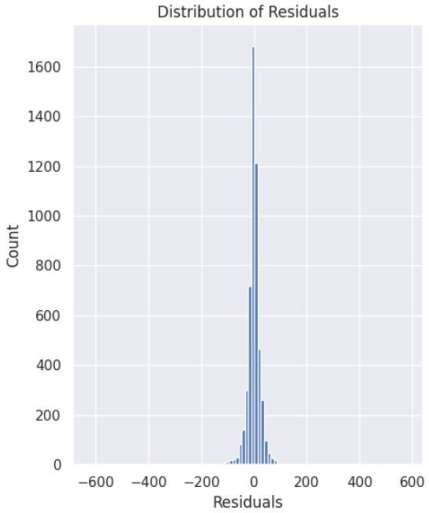
XGBRegressor - Deep Thicknesses with Hyperparameter Optimization

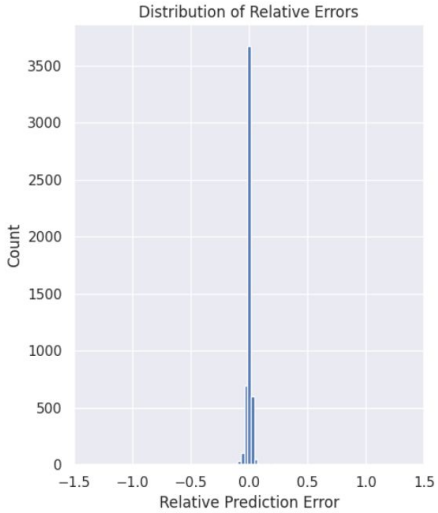
Training Mae: 14.41 m
Test Mae: 18.72 m
Valid Mae: 18.44 m
Mae Cross-Validation: 11.94 +- 0.15 m
Training Time: 4.37 s

WHAT?

Cross-Validation is so small!!!

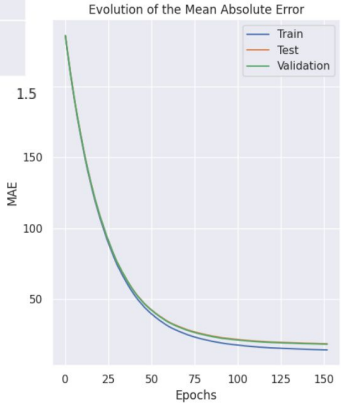






Best Hyperparameters:

{‘max_depth’: 10, ‘learning_rate’: 0.045, ‘min_child_weight’: 8, ‘subsample’: 0.829, ‘gamma’: 3.914, ‘alpha’: 0.004, ‘lambda’: 2.862}

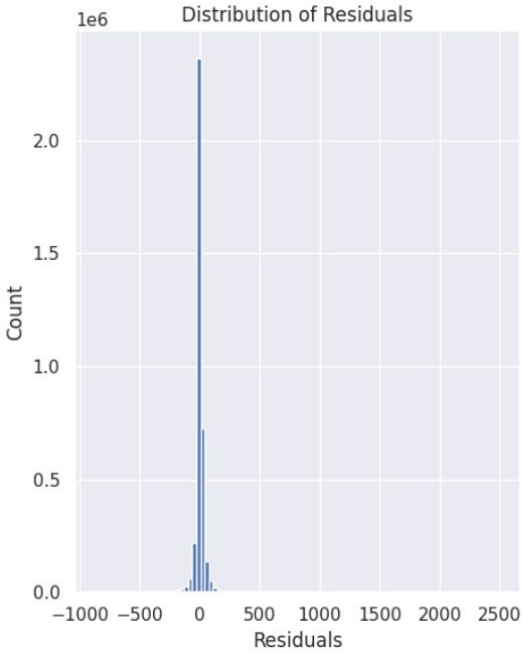


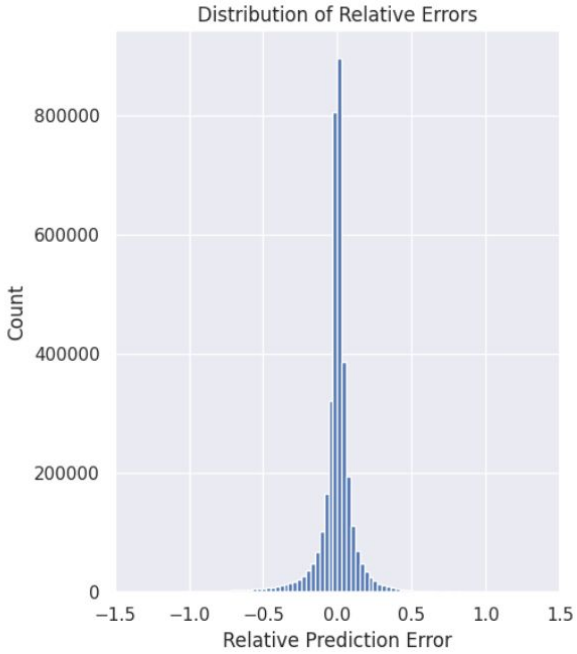
Predict the Original Data - Additional Plots

Trained by our best model.

Mae Cross-Validation: 13.65 +- 0.07 m

- data: 100x100 grid
- pre-processing: replace 0 m thicknesses by mean values of the models
- method: XGBRegressor + Optuna





LightGBM

LightGBMRegressor

Motivation: Can the simply model from Niccolo be improved? This is evaluated based on the MAE

Code (b,c): Optuna_lightGBM.ipynb. (a) CNN_classification.ipynb

(a, b) Run the training dataset of 20x20 grid:

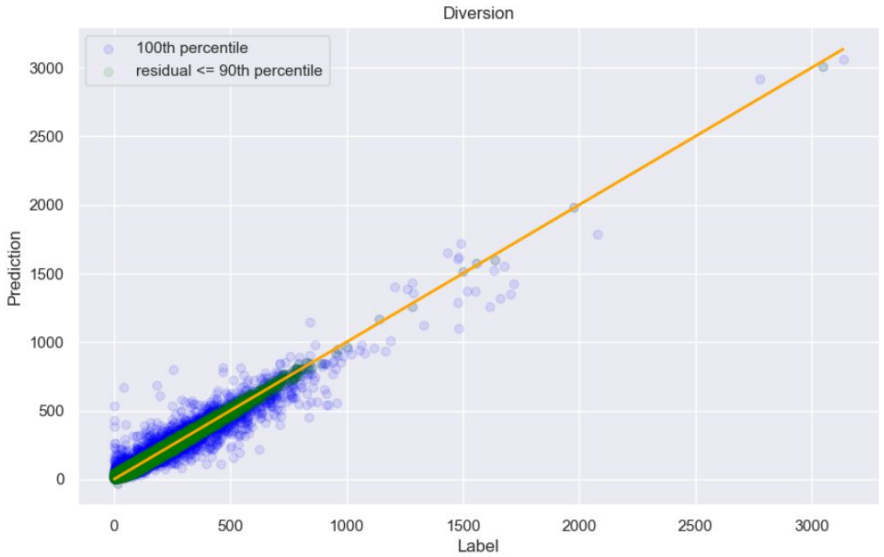
(a) includes information from images. (b) is without picture information.

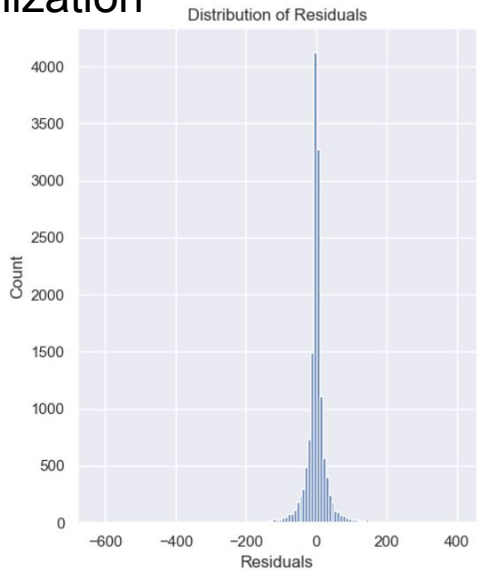
(c) Run the training dataset of 100x100 grid:

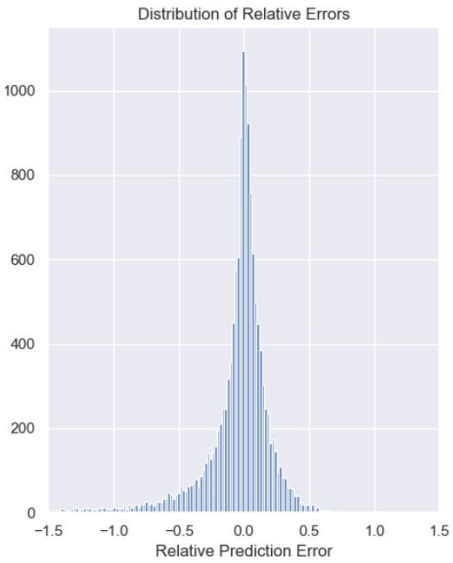
(c) is without the picture information

LightGBM - 20x20 grid (a) (I am killing myself (with pictures)) with Hyperparameter Optimization

Test Mae: 19.78 m
Mae Cross-Validation:
21.57 +- 0.16 m

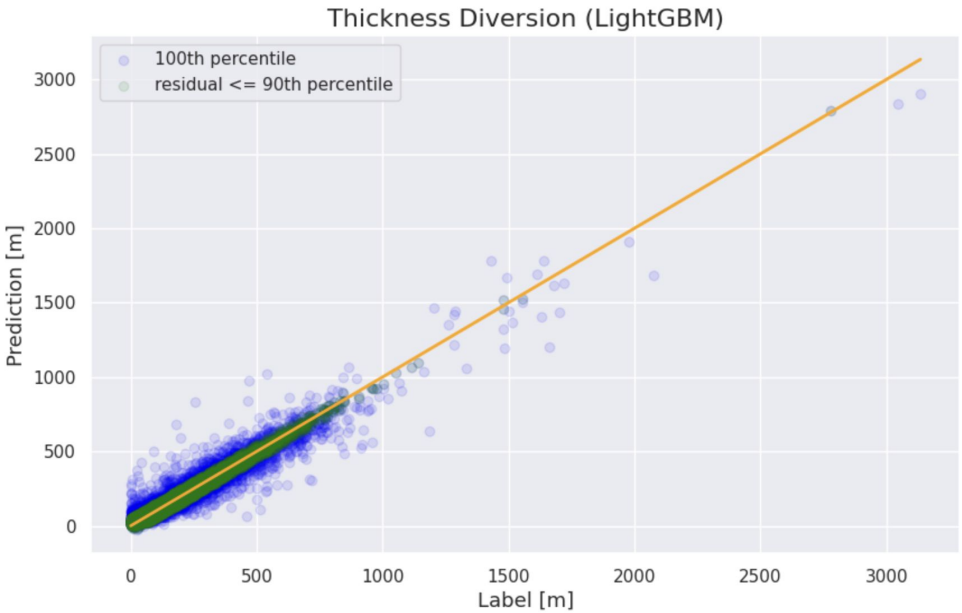


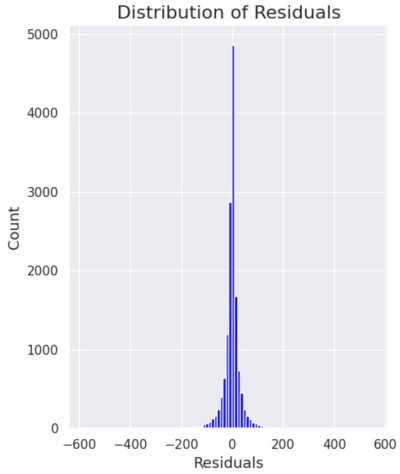


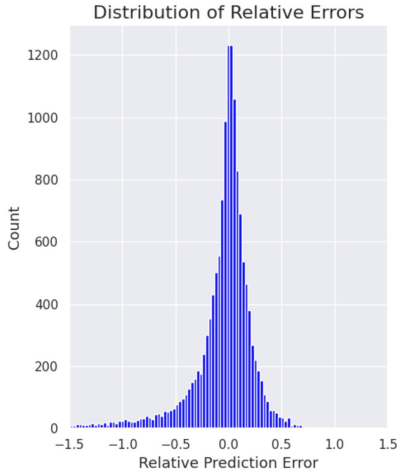


LightGBM - 20x20 grid (b) with Hyperparameter Optimization

Training Mae: 4.5092 m
Test Mae: 21.5049 m
Mae Cross-Validation:
22.9377 +- 0.130765



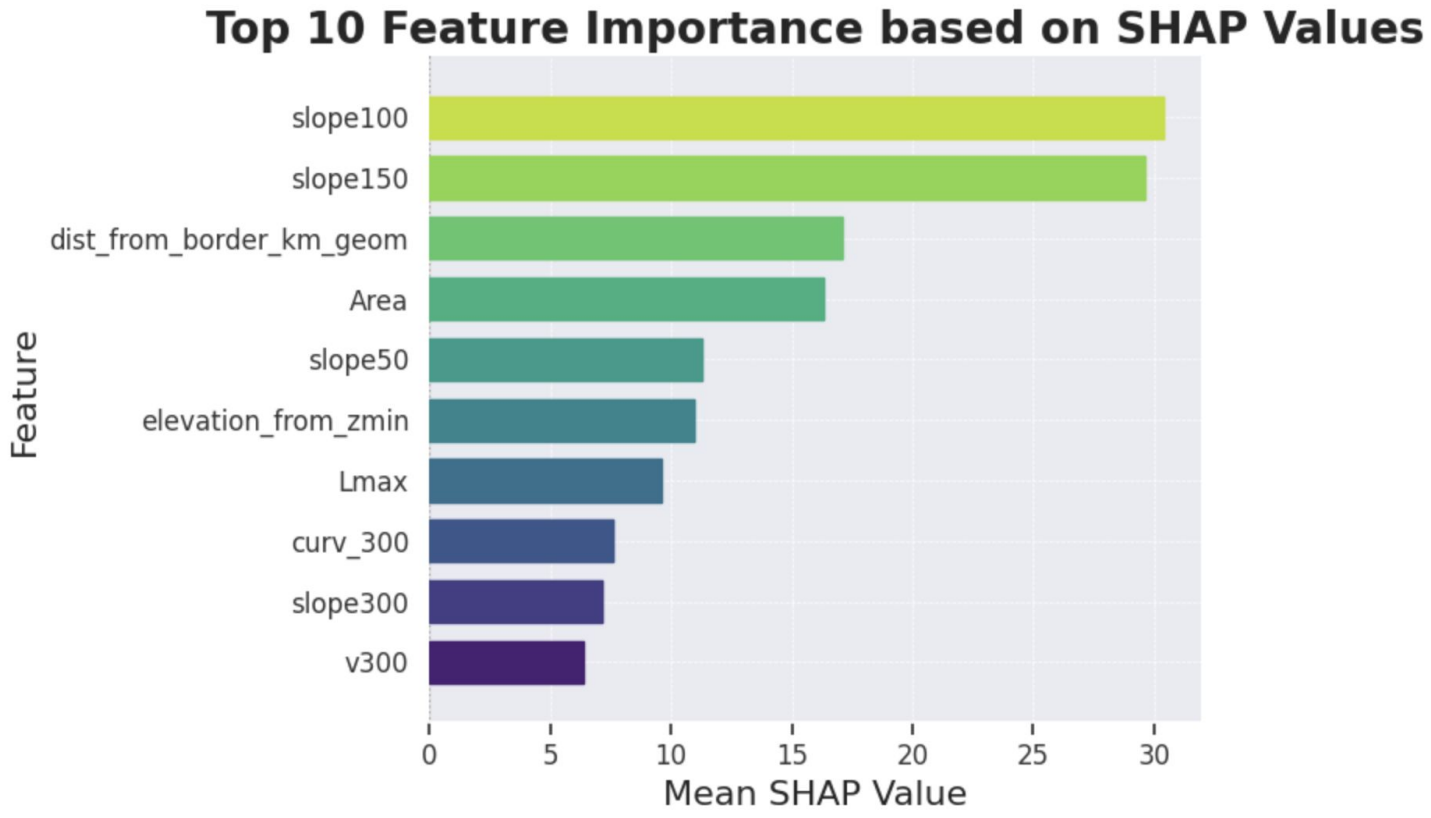




Best Hyperparameters:

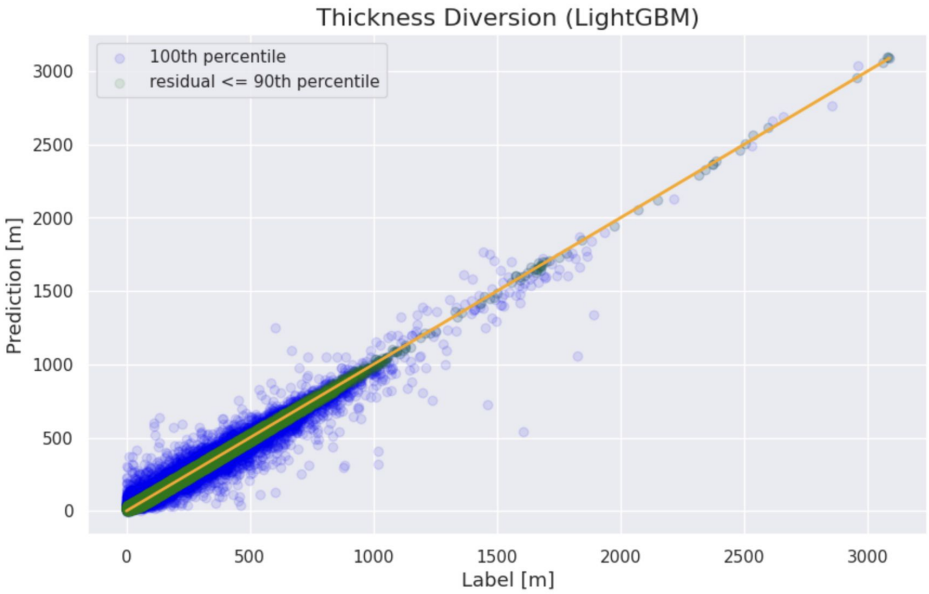
{'lambda_l1': 0.2696406338164187, 'lambda_l2': 0.4162002567105561, 'num_leaves': 1626, 'feature_fraction': 0.8293387459215363, 'bagging_fraction': 0.95562296523139, 'bagging_freq': 6, 'min_child_samples': 19}

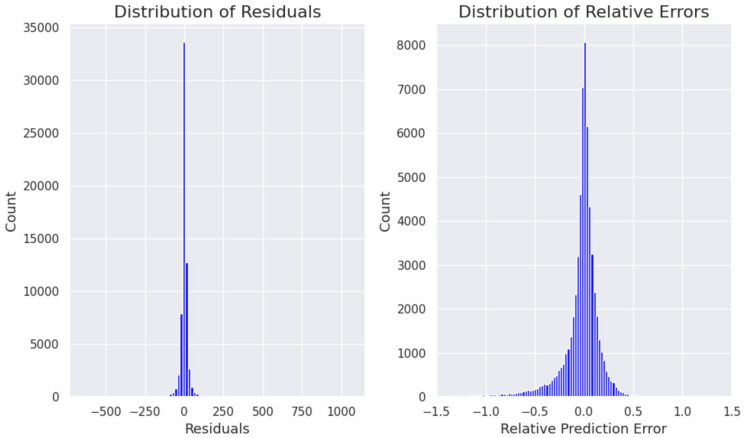
Shap Values for LightGBM- 20x20 grid (b) with Hyperparameter Optimization



LightGBM - 100x100 grid (c) with Hyperparameter Optimization

Training Mae: 6.9748 m
Test Mae: 14.6949 m
Mae Cross-Validation:
15.7155 +- 0.0840

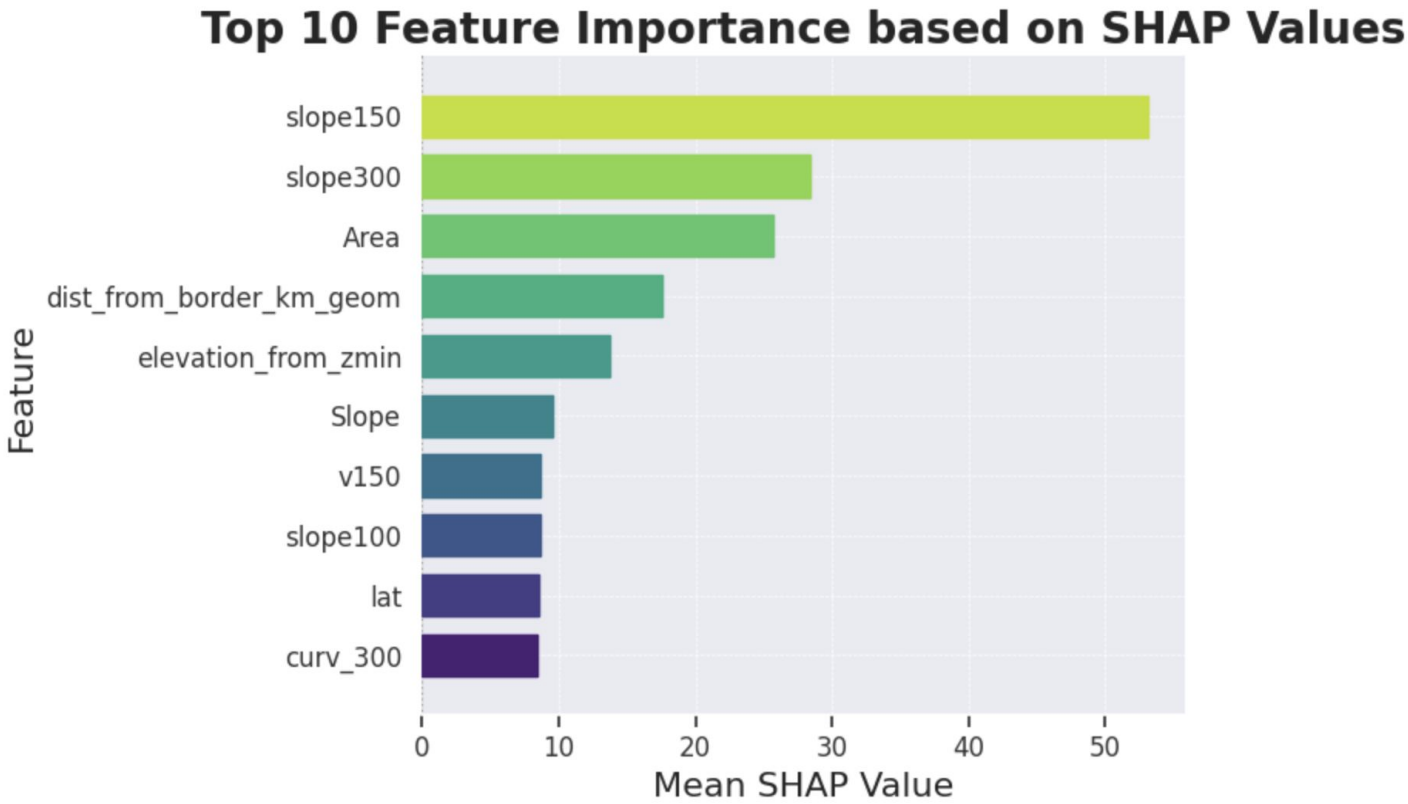




Best Hyperparameters:

{'lambda_l1': 0.2513239013767374, 'lambda_l2': 0.05357125688831004, 'num_leaves': 2040, 'feature_fraction': 0.9448131174359503, 'bagging_fraction': 0.9843654991192715, 'bagging_freq': 4, 'min_child_samples': 5}

Shap Values for LightGBM - 100x100 grid (c) with Hyperparameter Optimization



LightGBMRegressor - Conclusion

It works well, but in the end XGB beats it with a lower MAE.

Predict the Original Data with Niccolo's Simple Model

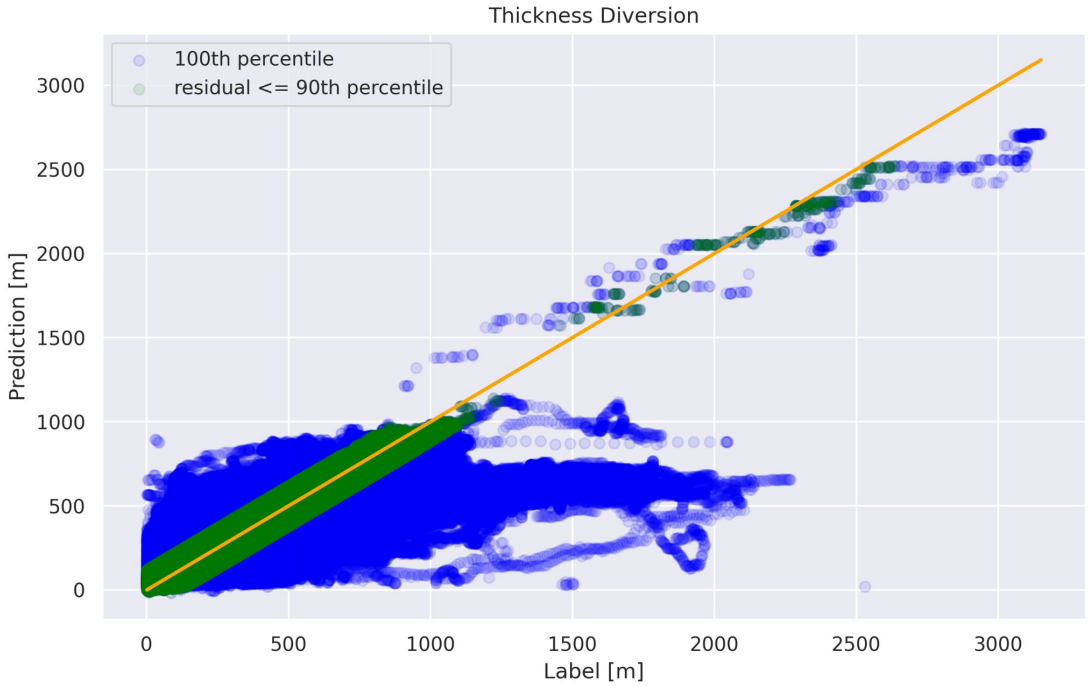
Trained by our best model.

Mae Cross-Validation: 13.65 +- 0.07 m

- data: 20x20 grid
- pre-processing: remove thicknesses <1 m
- method: lightGBM

Conclusion:

Many underprediction of thicknesses for thicknesses lower than 2000 m.



Features

Included Features from the Dataset

- latitude (POINT_LAT)
- Randolph Glacier Inventory Region (RGI)
- Minimum, maximum and mean glacier evaluation (Zmin, Zmax, Zmed)
- Glacier maximum length (Lmax)
- Glacier type (Form)
- Terminus type (TermType - Land-Terminating, Marine-Terminating….)
- Glacier mean aspect + additional aspect (Aspect, aspect_50, aspect_300, aspect_gfat)
- Glacier mean mass balance from Hugonnet et al. (2021) (dmdtda_hugo)
- Interpolated elevation (elevation)
- Curvature of the elevation map (curv_50, curv_300, curv_gfa)
- Mass balance from a mix of different methods (smb)
- Closest distance of the point to any glacier border (dist_from_border_km_geom)

Additional Calculated Features

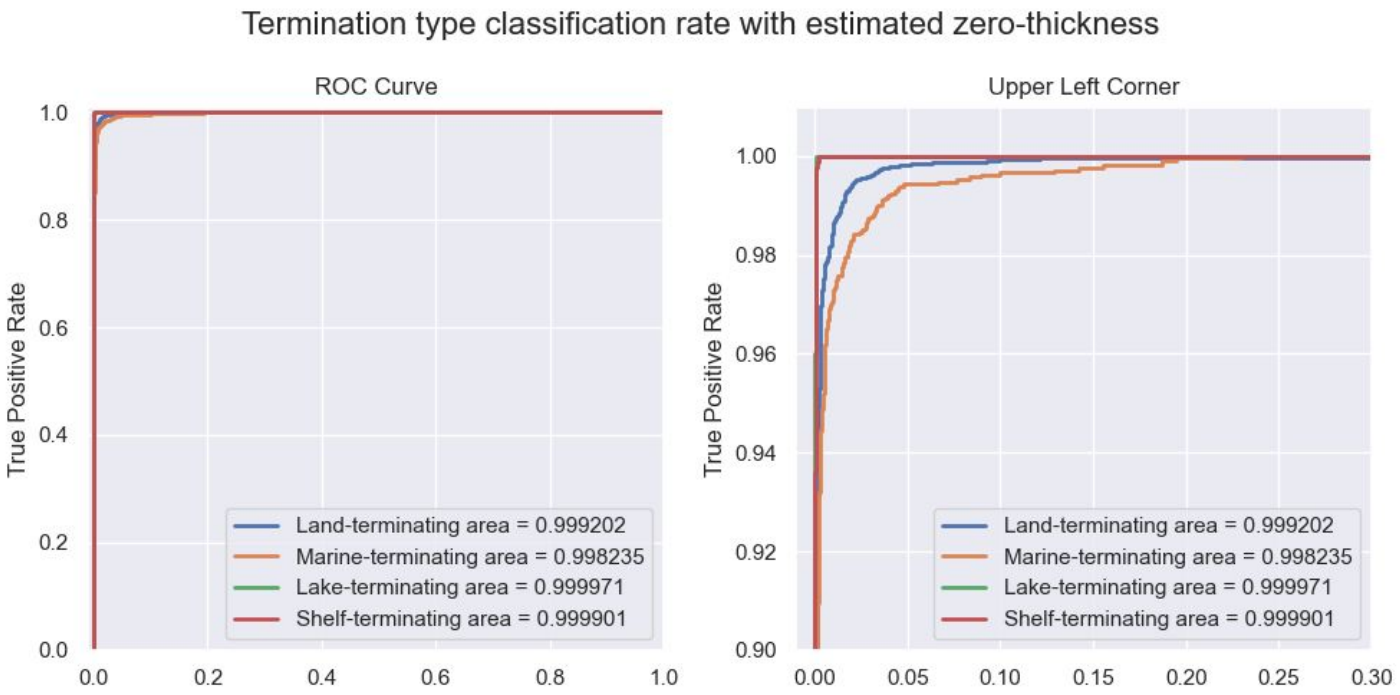
- Modified elevation: Elevation - Minimum elevation of the glacier
  data['elevation_from_zmin'] = data['elevation'] - data['Zmin']

- Slopes depending on latitude and longitude:
  data['slope50'] = np.sqrt(data['slope_lon_gf50']**2 + data['slope_lat_gf50']**2) - example

- Smoothed Velocity depending on latitude and longitude:
  data['v50'] = np.sqrt(data['vx_gf50']**2 + data['vy_gf50']**2) - example

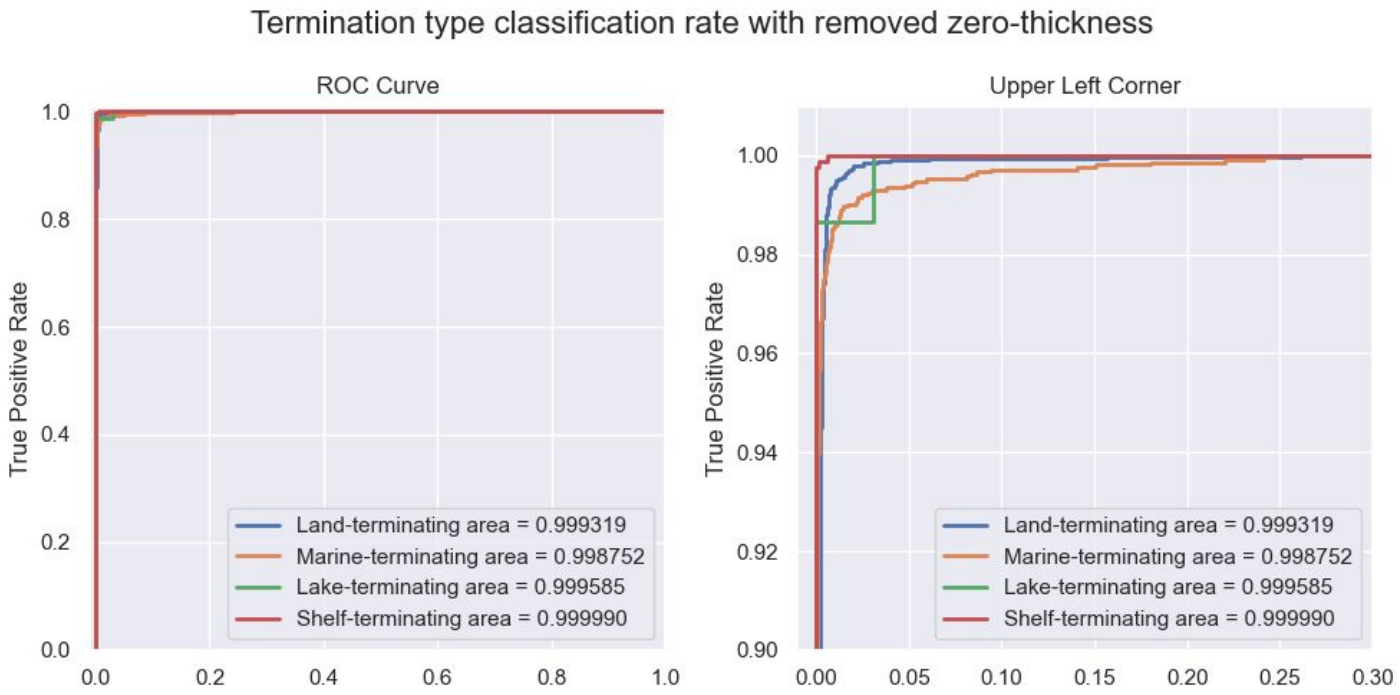
Classification

Classification of Termination Type, tabular data



Hyperparameters found with random search:
{'activation': 'tanh', 'alpha': 0.01882786639507904, 'early_stopping': True, 'hidden_layer_sizes': (147, 119, 38), 'learning_rate': 'constant', 'solver': 'adam'}

Classification of Termination Type, tabular data

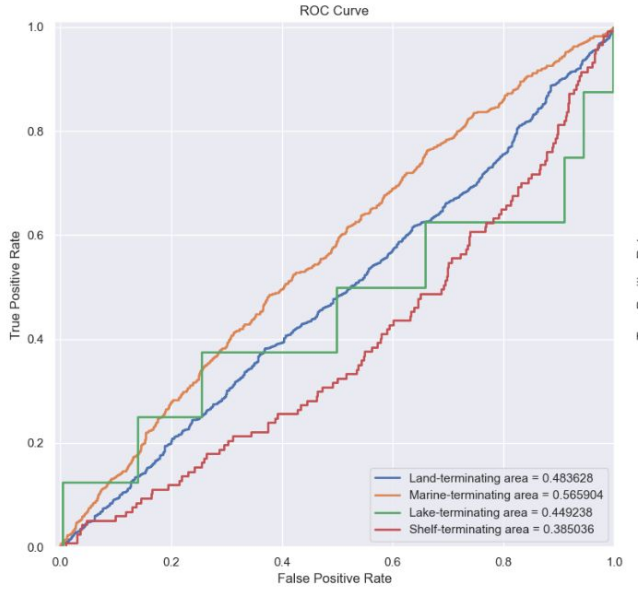


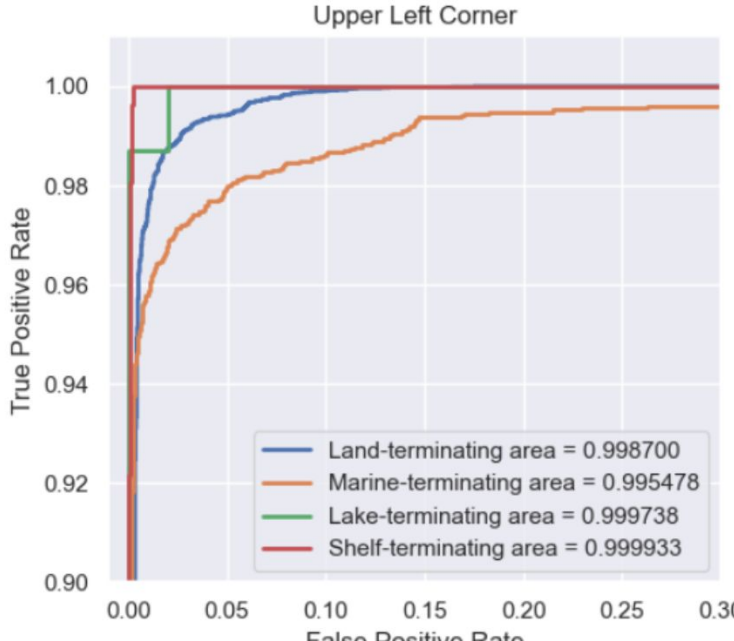
Hyperparameters found with random search:
{'activation': 'relu', 'alpha': 0.030062355171428815, 'early_stopping': True, 'hidden_layer_sizes': (101, 122, 28), 'learning_rate': 'adaptive', 'solver': 'adam'}

ROC curve for CNN

Key finding: Pictures dont add anything meaningfull

CNN + NN is worse than BDT comparable to NN



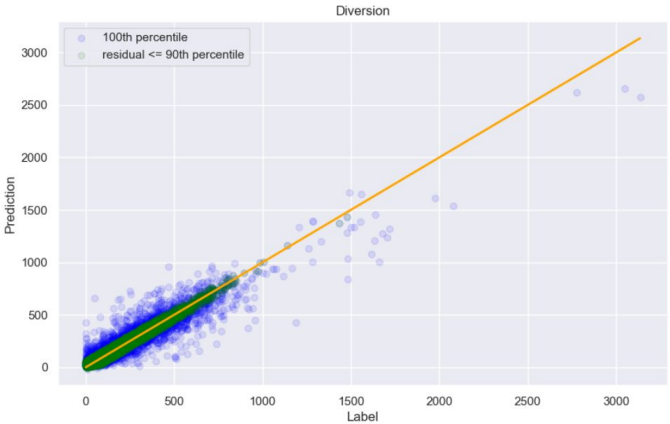


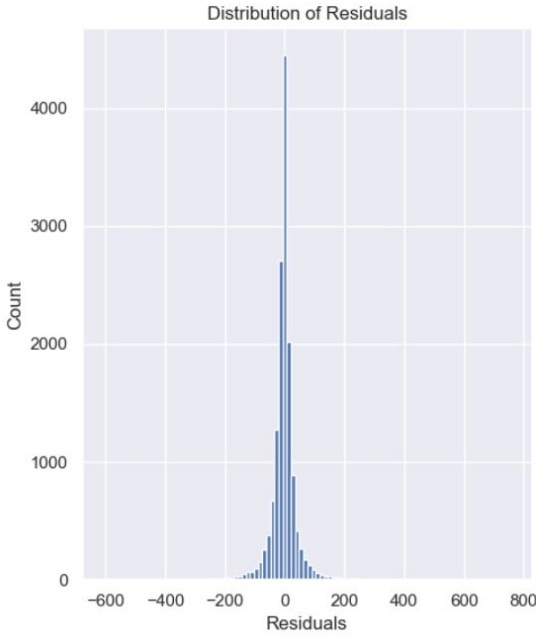
Neural Network

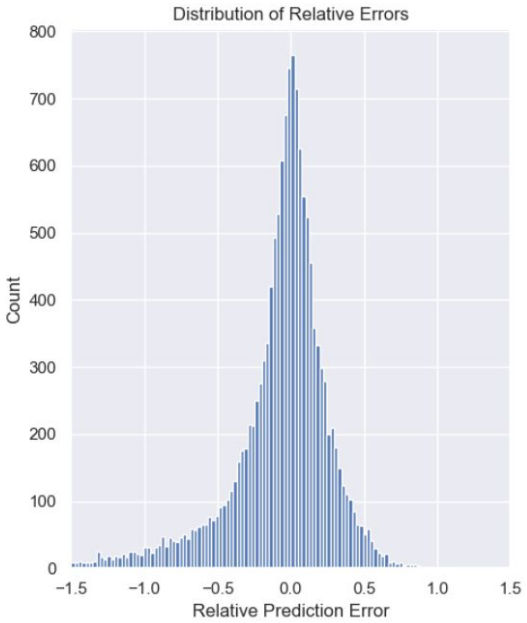
Tensorflow grid 20 data n0 **with** pictures

Test Mae: 27.37 m
Training Time: ~16920 s or 282 minutes :)

And years of my future lifespan (:



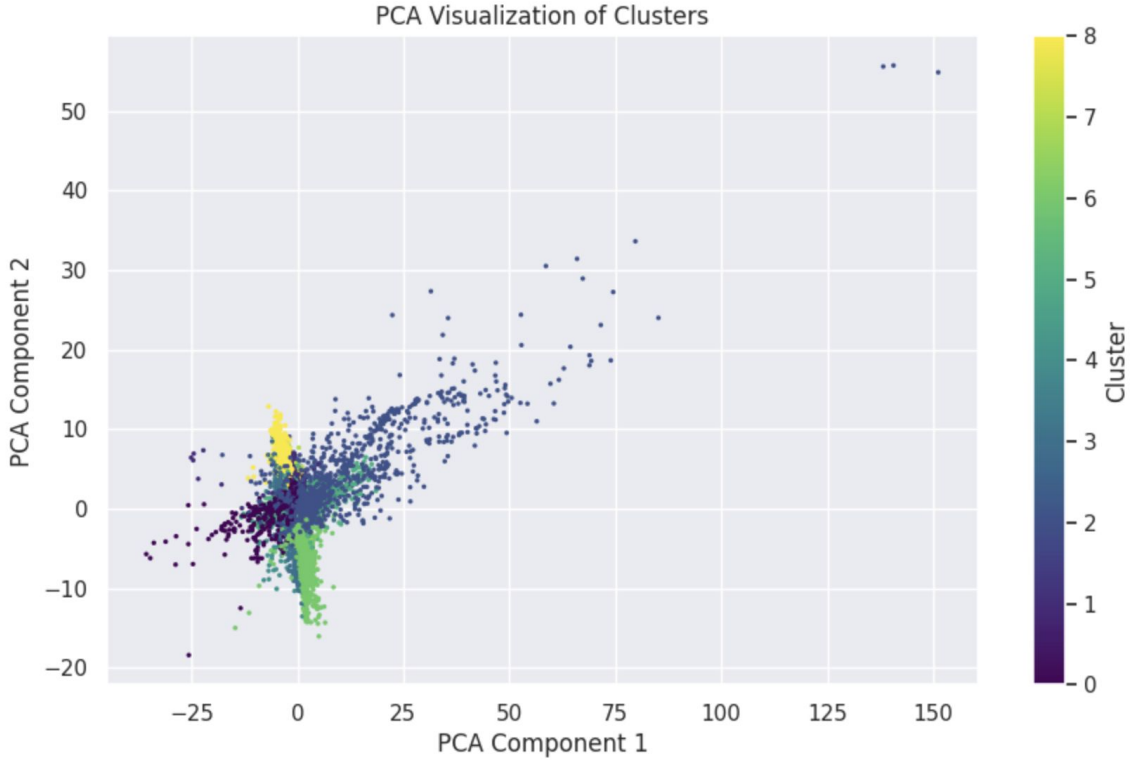




Clustering

Clustering

MinibatchKmeans + PCA



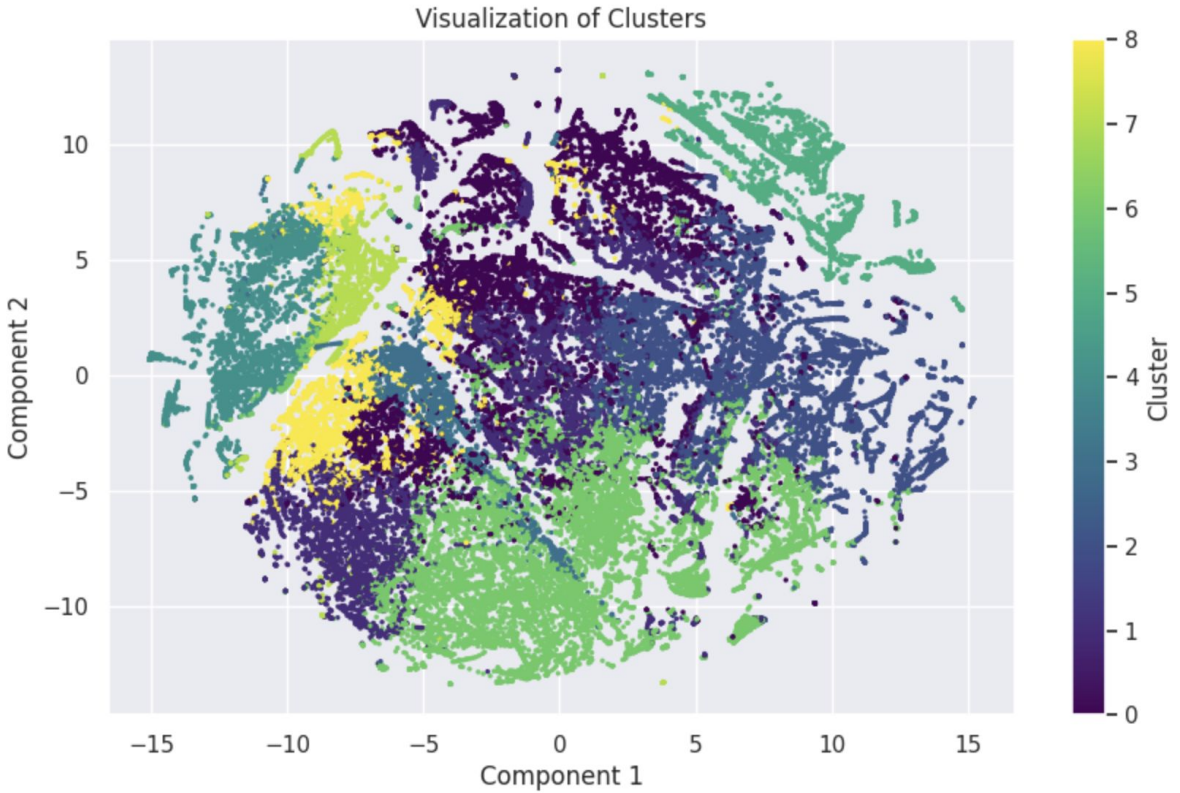
Step 1: Preprocess the data by scaling with StandardScaler()

Step 2: Clustering with MiniBatchKMeans. #cluster = 9

Step 3: For visualization purposes dimension reduction with PCA

Clustering

MinibatchKmeans + t-sne



Step 1: Preprocess the data by scaling with StandardScaler()

Step 2: Clustering with MiniBatchKMeans. #cluster = 9

Step 3: For visualization purposes dimension reduction with t-sne

Clustering conclusion

As learned from lectures in machine learning. Try throw a clustering at the data. This is the result of that with different method for visual representation.

We did not use this result in the end.