Predicting ice thickness on Antarctica

Markus, Philip, Adian, Vasileios, Jens GR

Problem Statement

- Predict ice-thickness across Antarctica
- Assess severity of potential rise in water levels

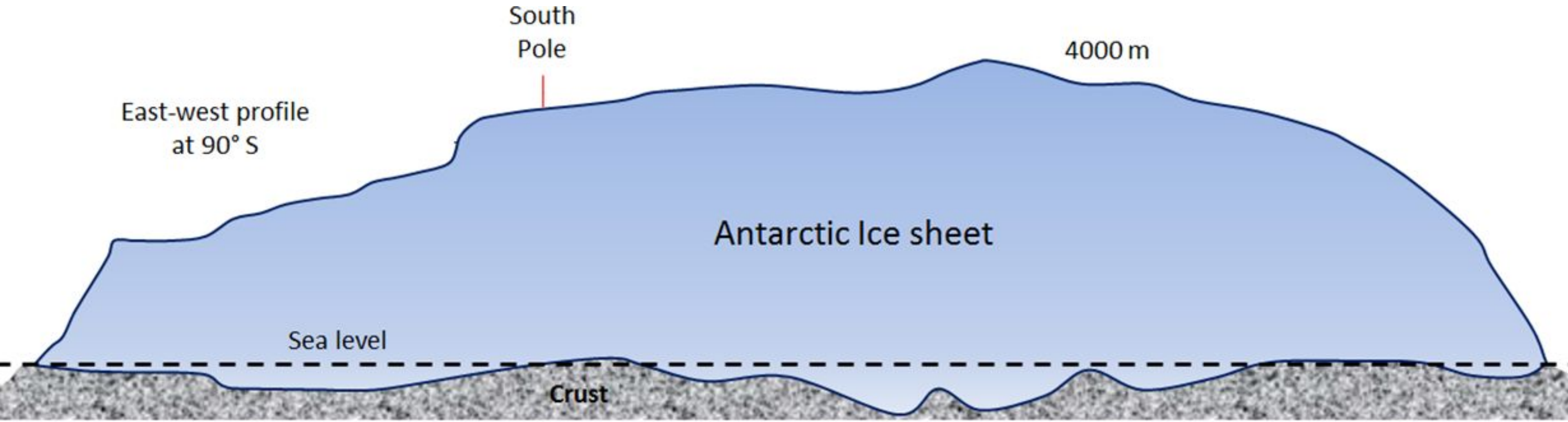


Image source: https://opentextbc.ca/physicalgeologyh5p/chapter/types-of-glaciers/

Data

True:

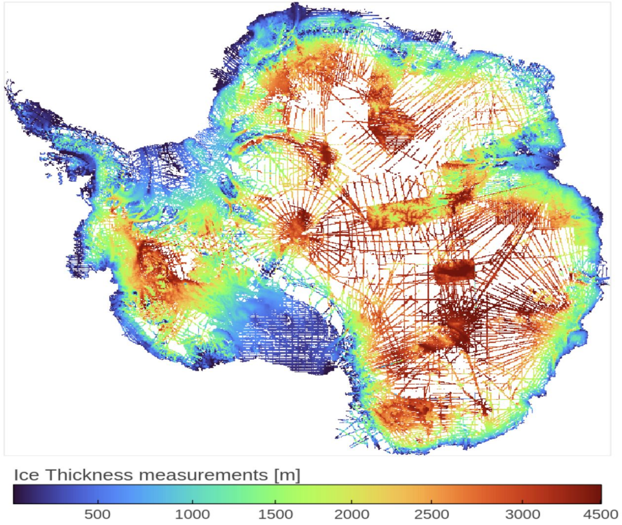
- Ice thickness: 80 million measurements with ground penetrating radar carried on airplanes (tabular). No -999.0 and no NaNs.

Features:

- ‘z’ (surface elevation)
- “v_x’ and ‘v_y’ (ice velocity)
- ‘temp’ (annual mean temperature)
- ‘smb’ (surface mass balance)
- ‘s’ (surface slope)
- ‘East’ og ‘North’ (coordinates)

Data comes primarily from satellite imagery and climate models stored as maps in netcdf files.

Interpolated to our 80 million ice thickness measurements.



Data

Mistakes along a few flight tracks:

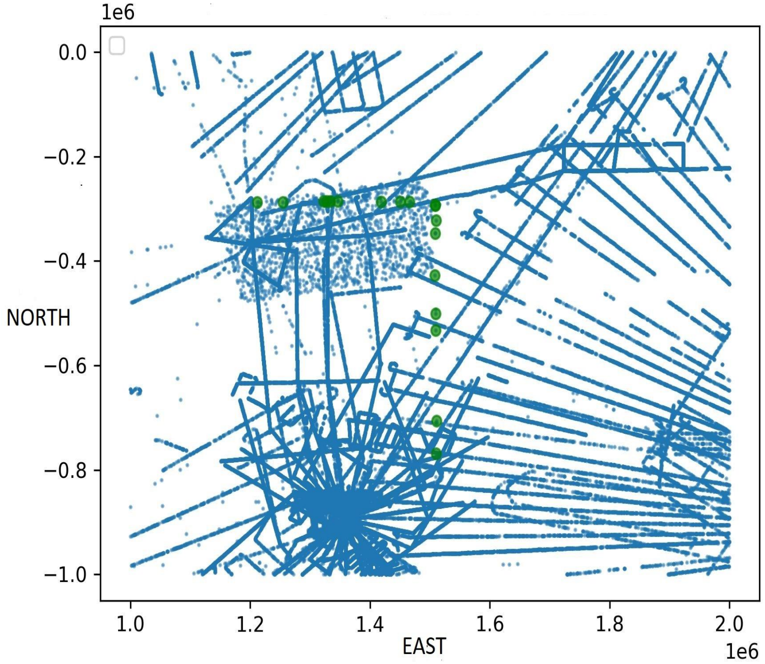
- Removing these measurements does not improve model performance significantly.

Duplicates:

- There are 2.8 million duplicates in the dataset.
- Most of them have very similar ice thickness.
- Taking the average of the duplicates does not improve model performance significantly.

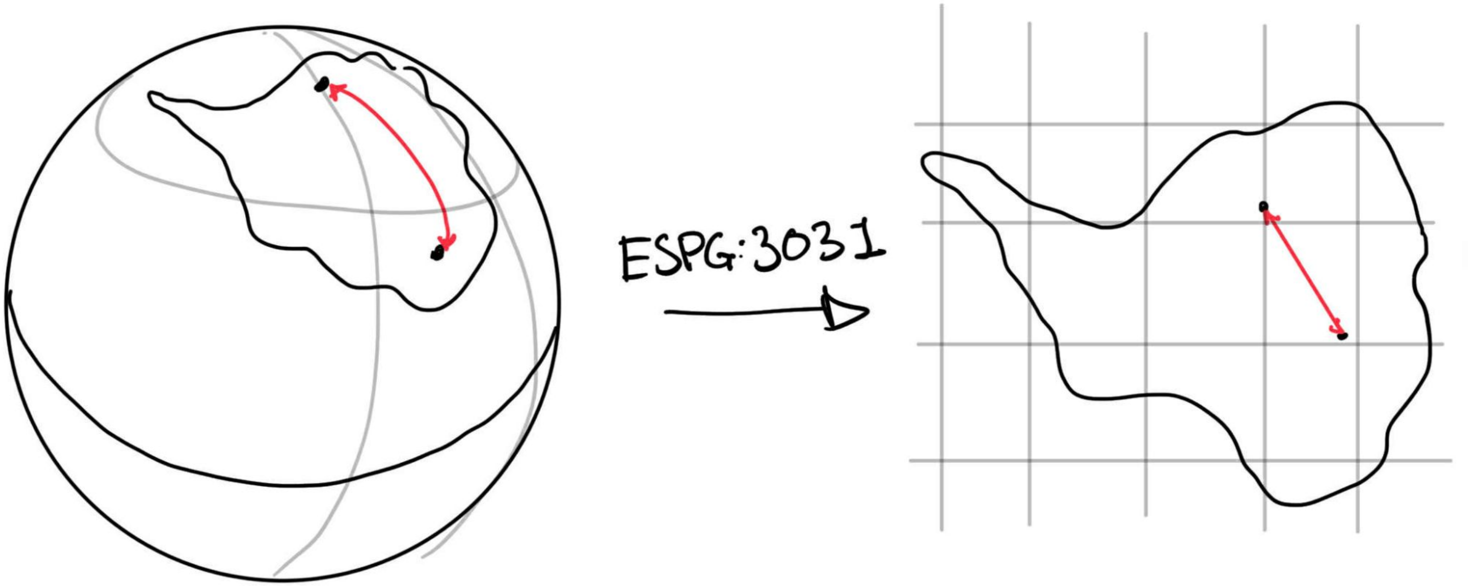
South East Antarctica

At green points ice thickness depth is about 1000m less than surroundings.



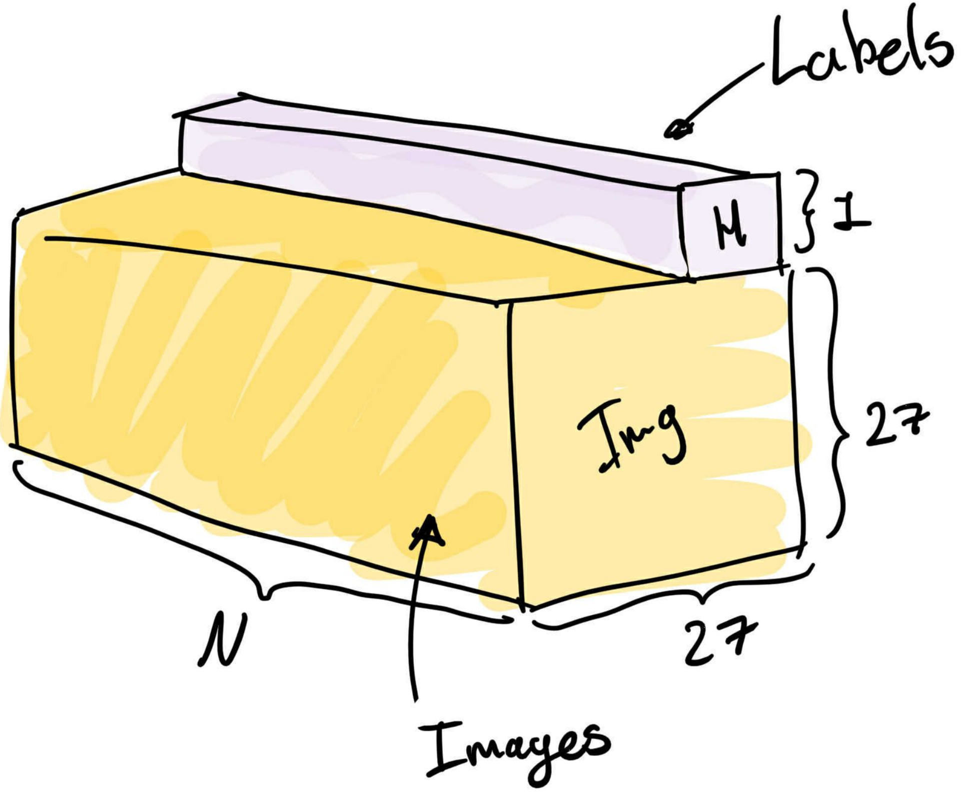
Python for Geospatial data.

- **3031 Projection**
- (LON,LAT) -> (EAST,NORTH)



Python for Geospatial data.

- **xarray**
  - Can store non-convex array shapes



► Dimensions: (**sample**: 189326, **x**: 27, **y**: 27)

▼ Coordinates:

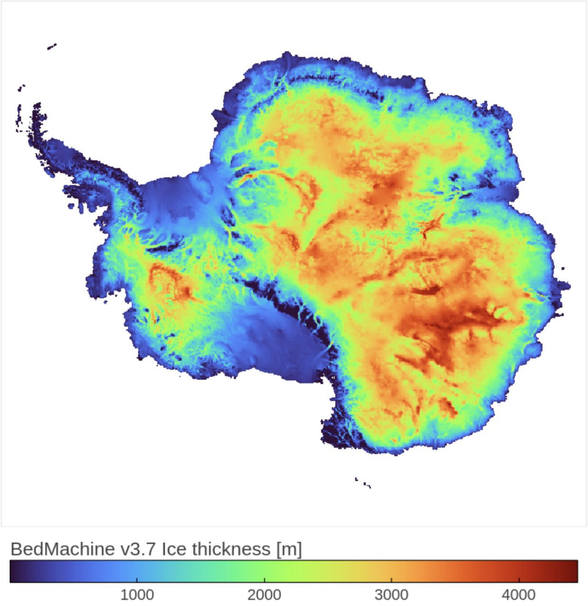
sample	(sample)	int32	0 1 2 3 ... 189323 189324 189325
x	(x)	int32	0 1 2 3 4 5 6 ... 21 22 23 24 25 26
y	(y)	int32	0 1 2 3 4 5 6 ... 21 22 23 24 25 26

▼ Data variables:

images	(sample, x, y)	float32	...
labels	(sample)	float64	...
vx	(sample)	float64	...
vy	(sample)	float64	...
v	(sample)	float64	...
smb	(sample)	float64	...
z	(sample)	float64	...
s	(sample)	float64	...
temp	(sample)	float64	...

Models and loss functions

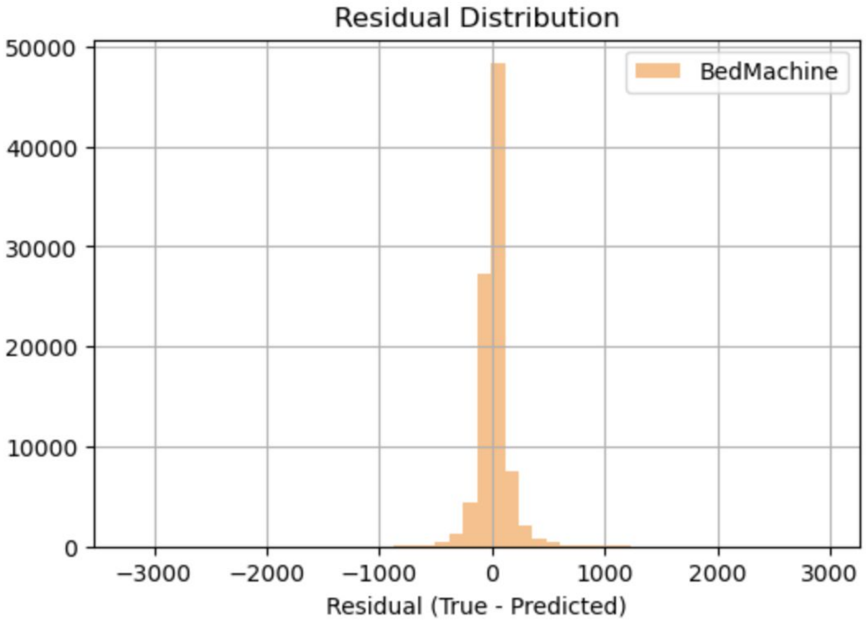
- BedMachine (For comparison)
- Tree based models
  - LightGBM
  - Decision Tree regressor
- Tabular NN
- CNN
  - One for each type of image
  - Collected model for all types

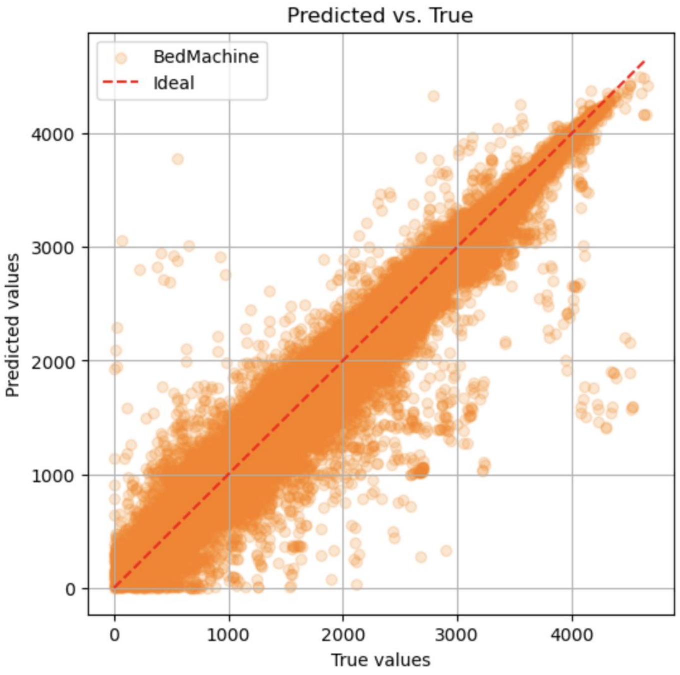


- Choice of loss function (MAE) (Why not MAPE?)

BedMachine v3.7

- State of the art glaciological model
- MAE: 81.97

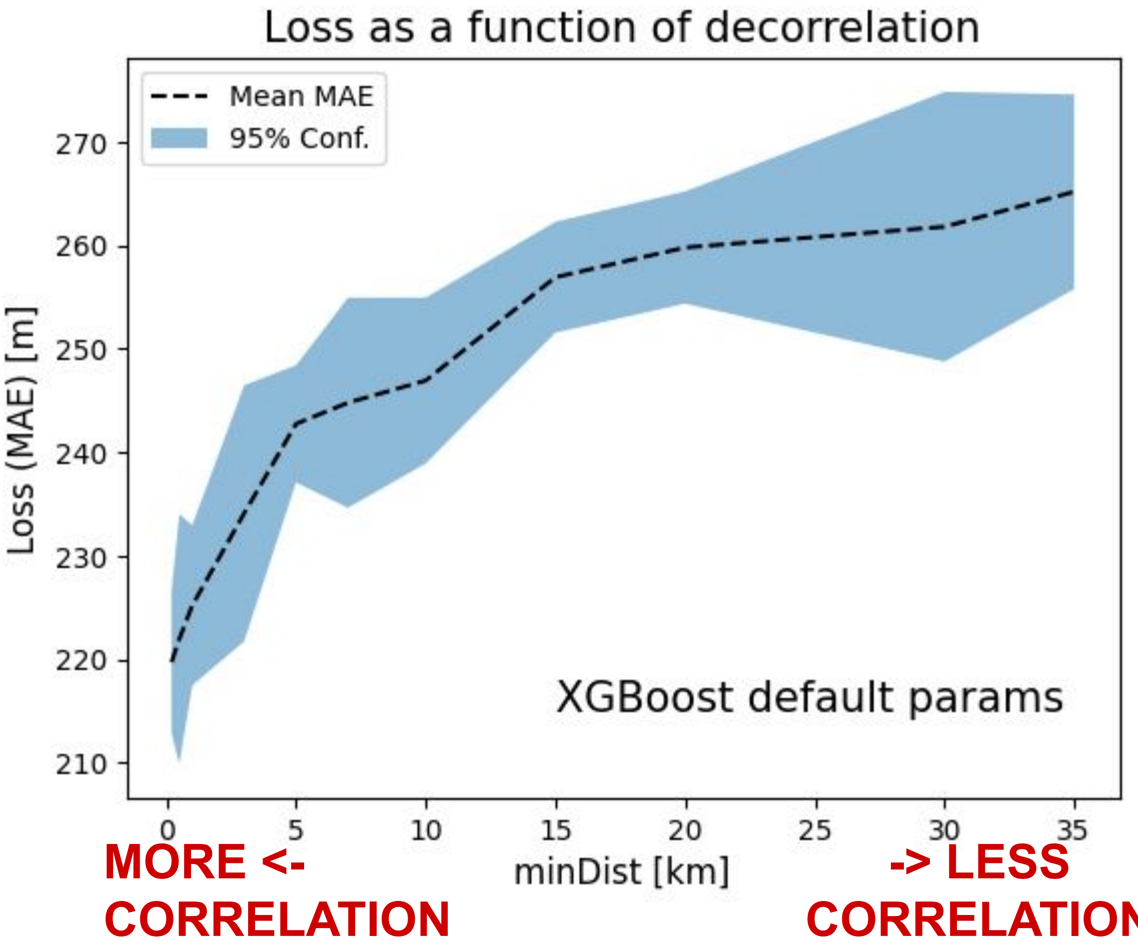


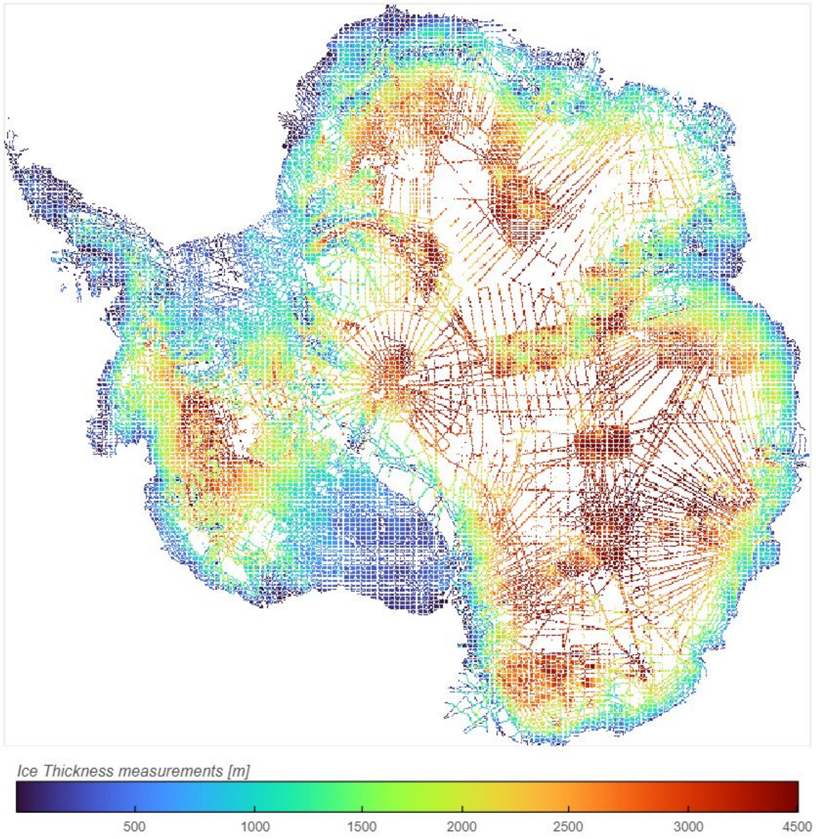


Our data is correlated

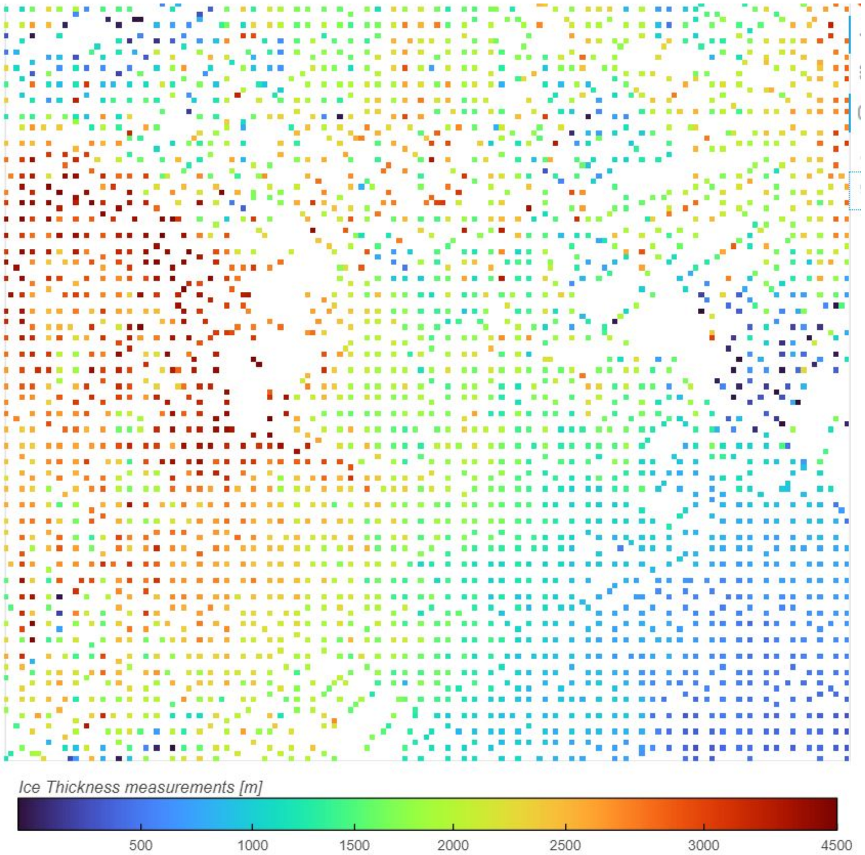
- Correlation loss

- How do we deal with it?

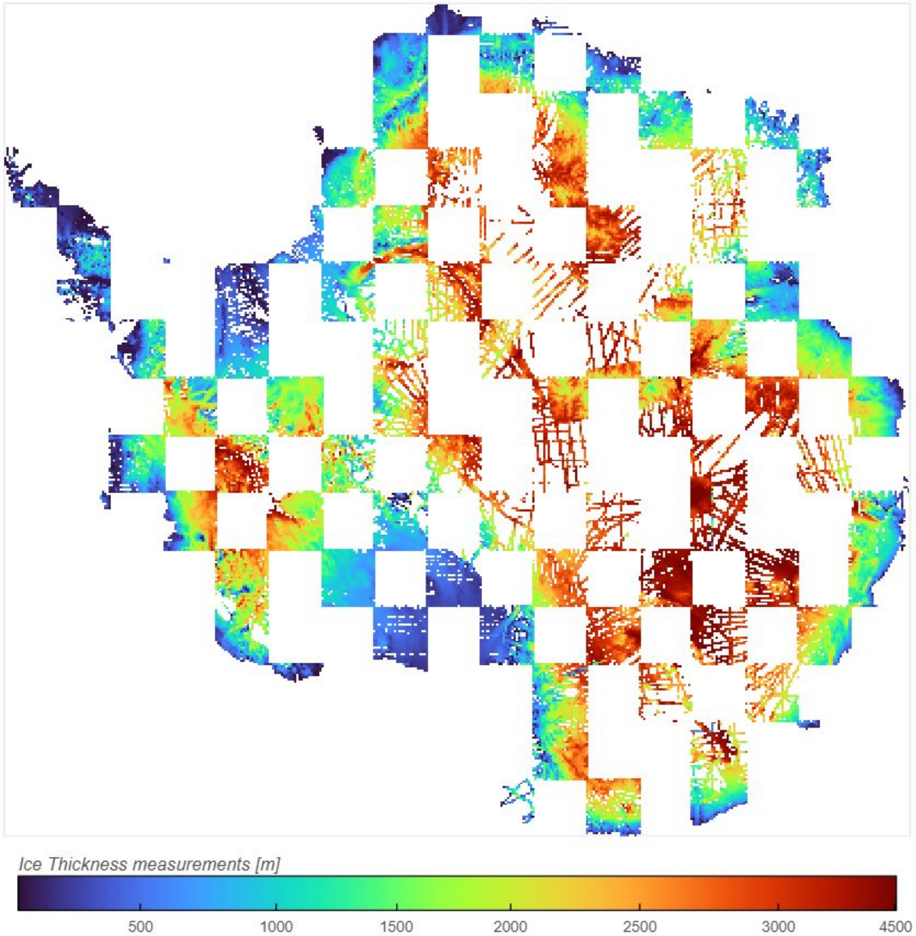


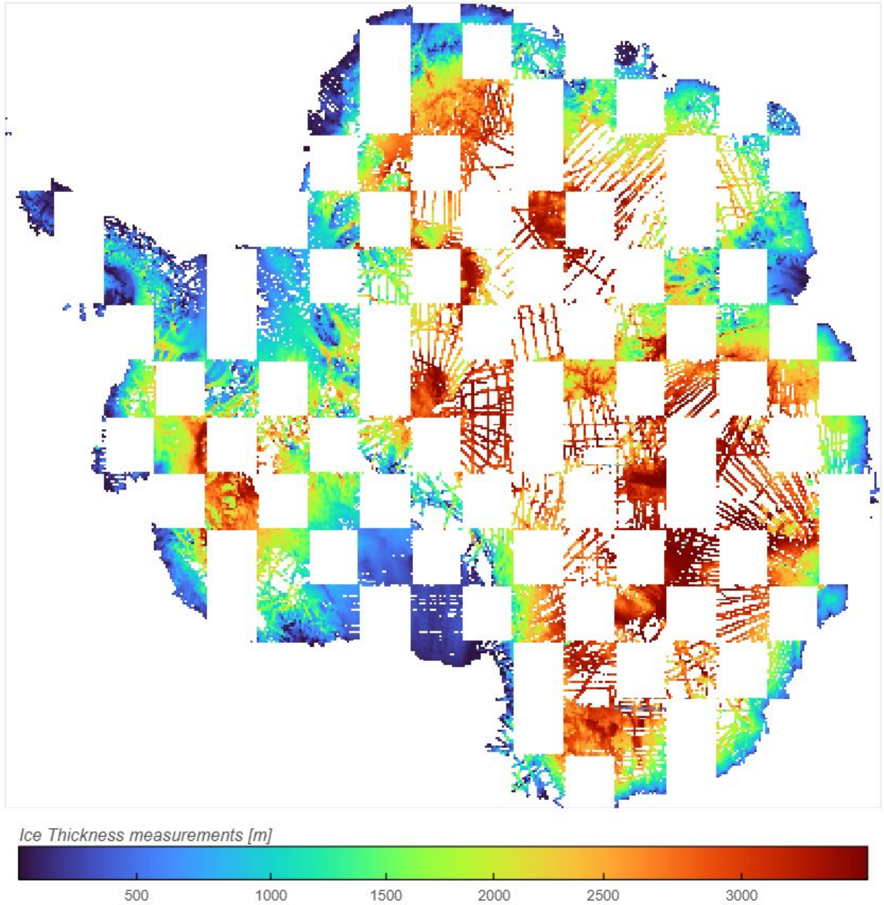


minDist = 10km



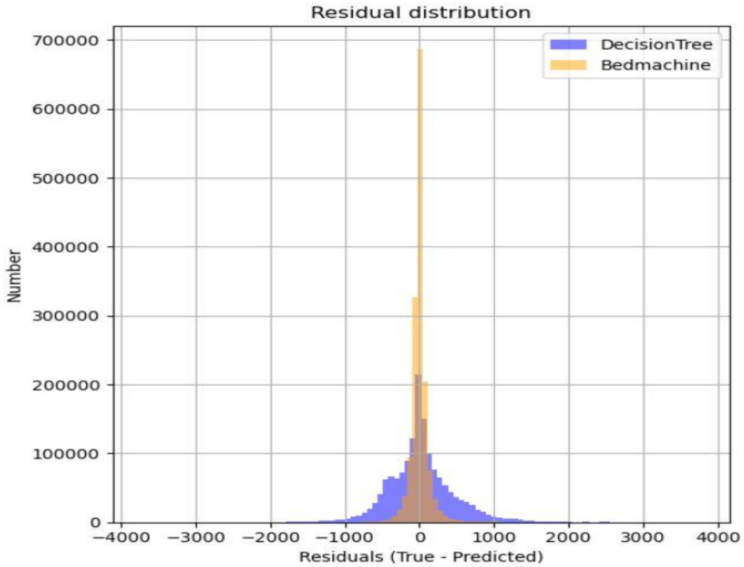
minDist = 20km

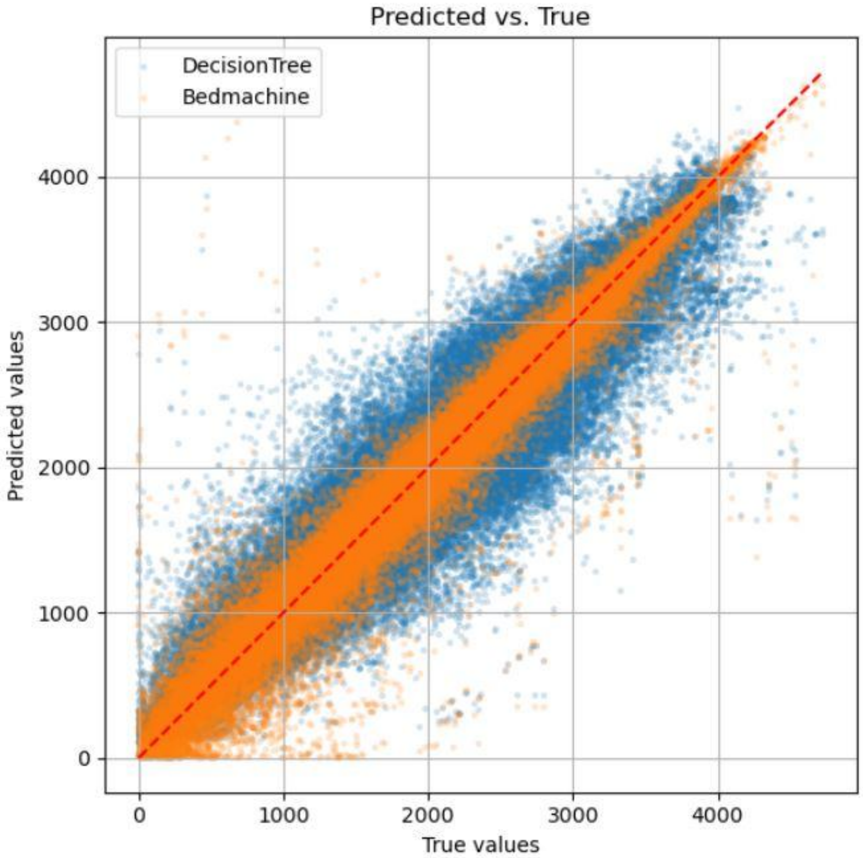

Ice Thickness measurements [m]
500
1000
1500
2000
2500
3000
4500


Ice Thickness measurements [m]
500
1000
1500
2000
2500
3000

DecisionTreeRegressor

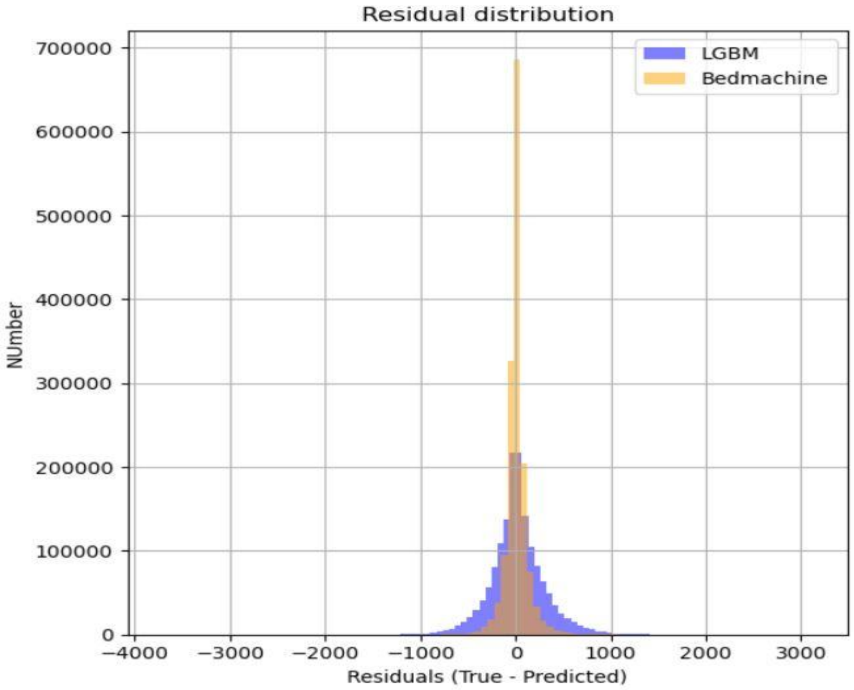
MAE = 320.66 [m]

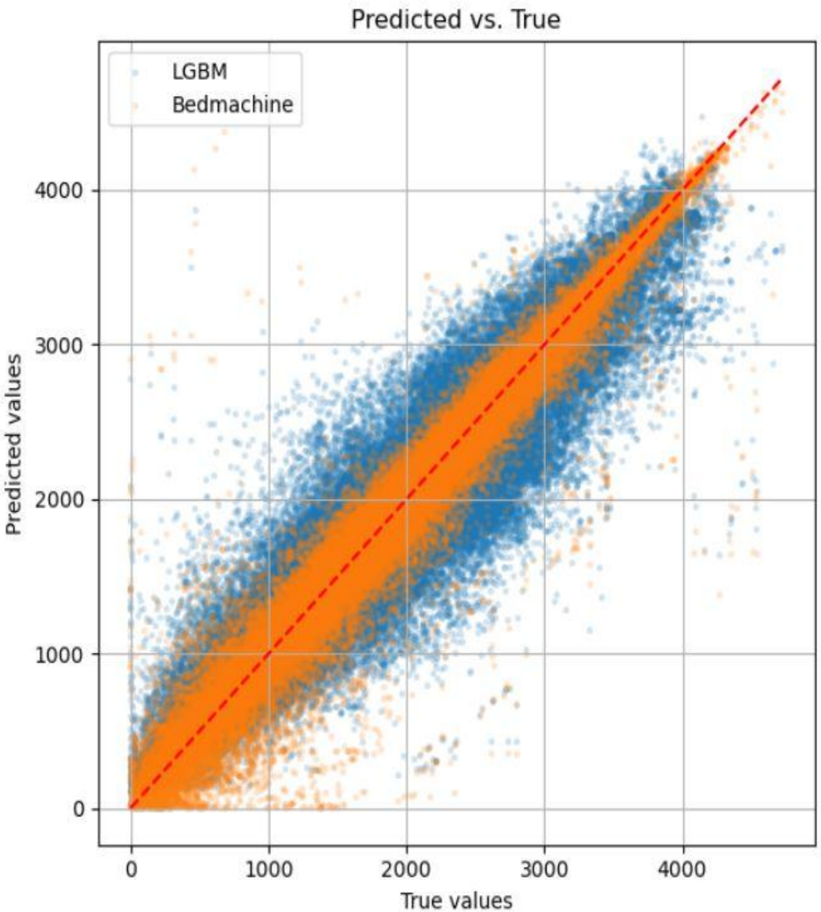




LGBM (with optimized hyperparameters)

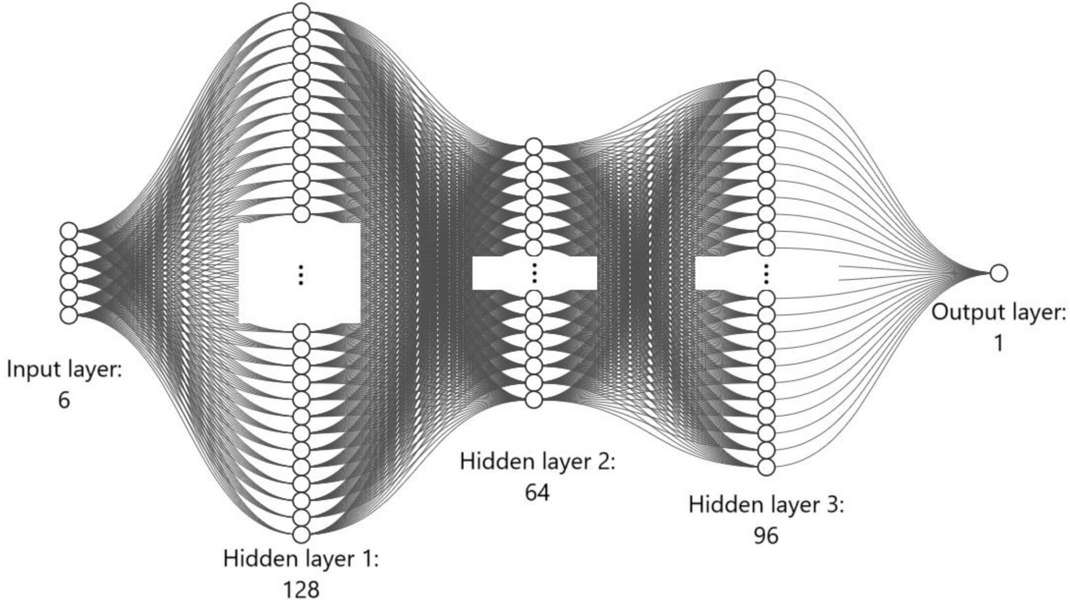
MAE = 204.80 [m]





Tabular NN

Neural Network Architecture:



+Dropout layers

Input:

Normalize with MinMax scaler

Variable: Distance to nearest Mountain, with and without.

- 10,000 candidates
- Radius of 100 kilometers
- Difference of more than 200 meters

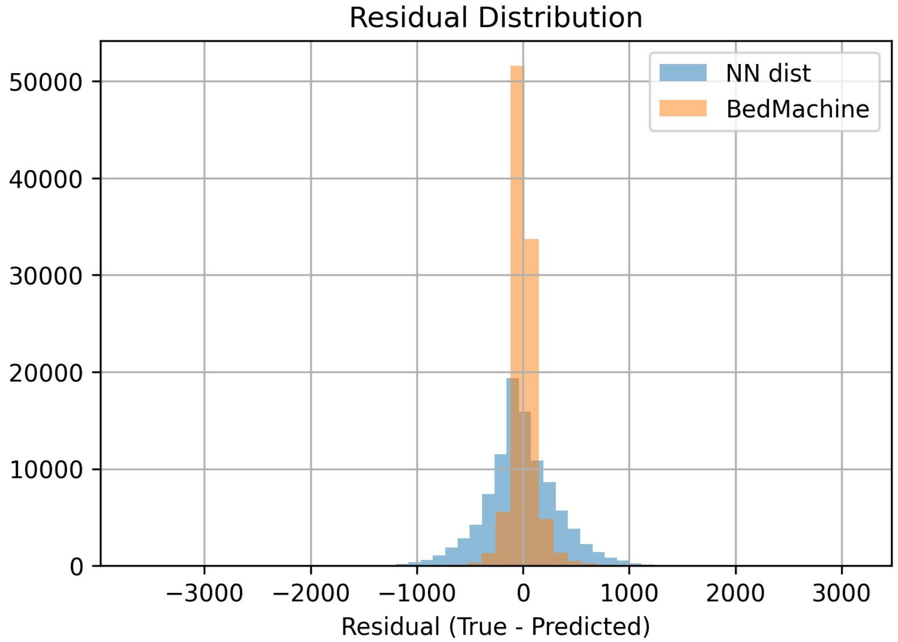
Hyperparameter Optimization:

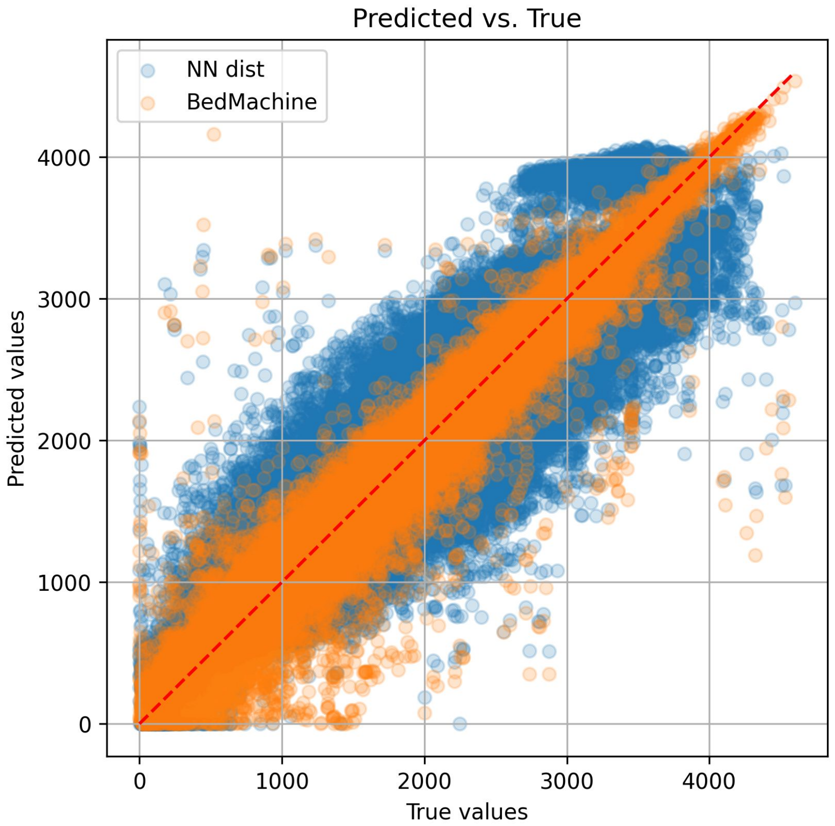
- Random Search
- Validation on uncorrelated data

Tabular NN

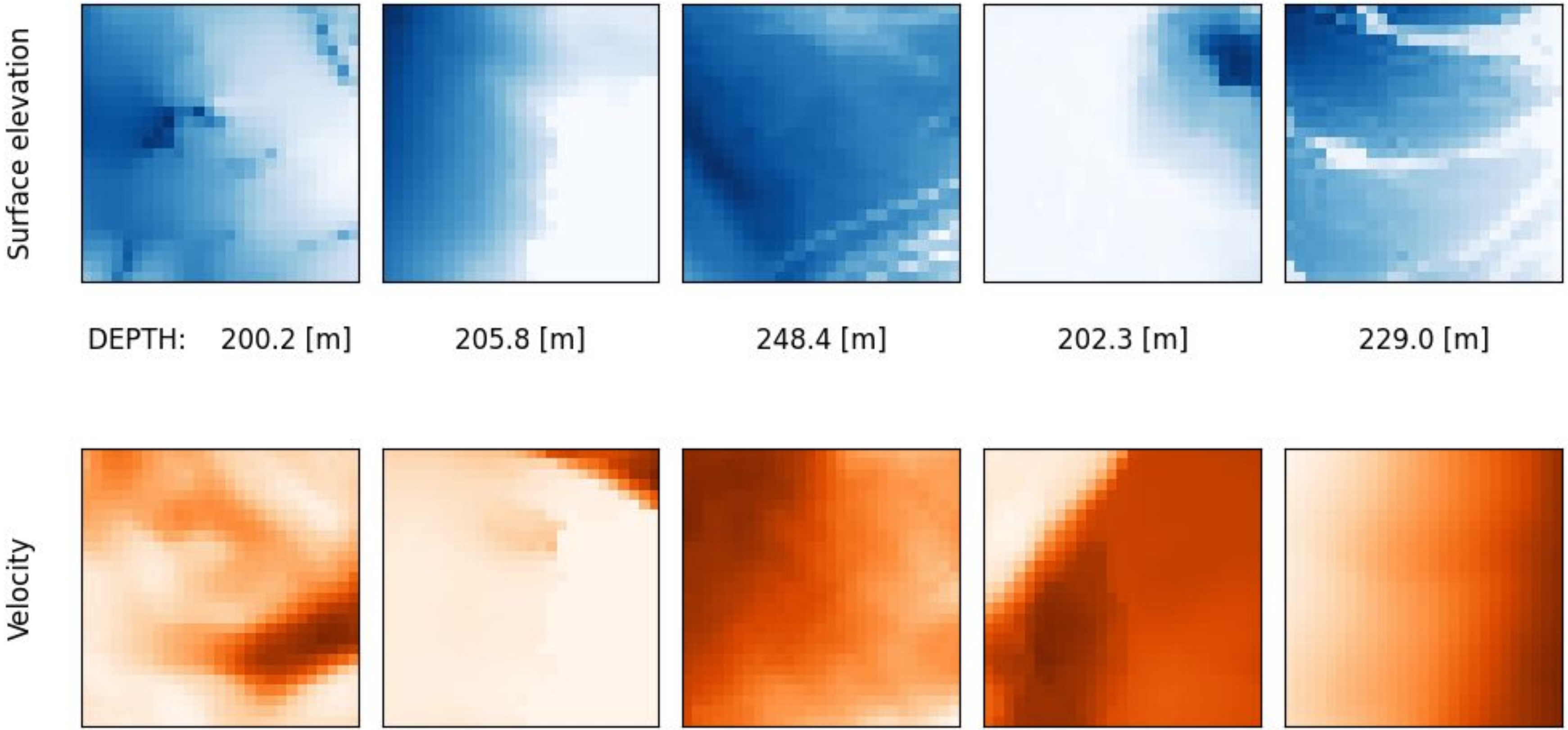
MAE (Coordinates): 246 [m]

MAE (Distance): 255 [m]





Data preprocessing for CNN's



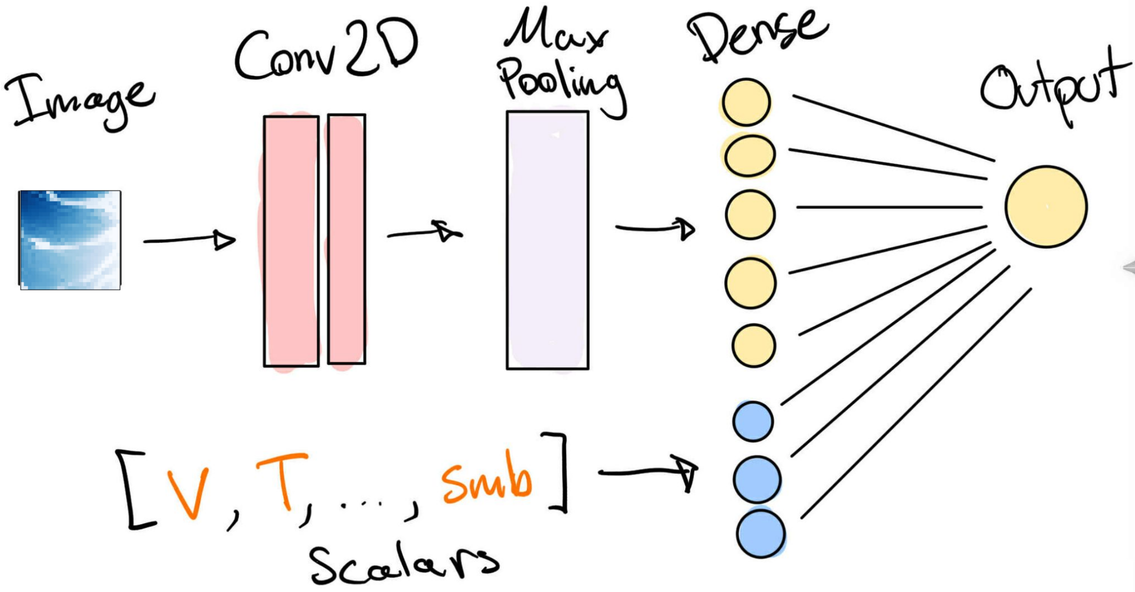
CNN architecture and random search

Conv 2d layers optimise over (1-3)

- 3x3 filters optimise over (32-128)
- 2x2 max pooling
- activation: ReLU

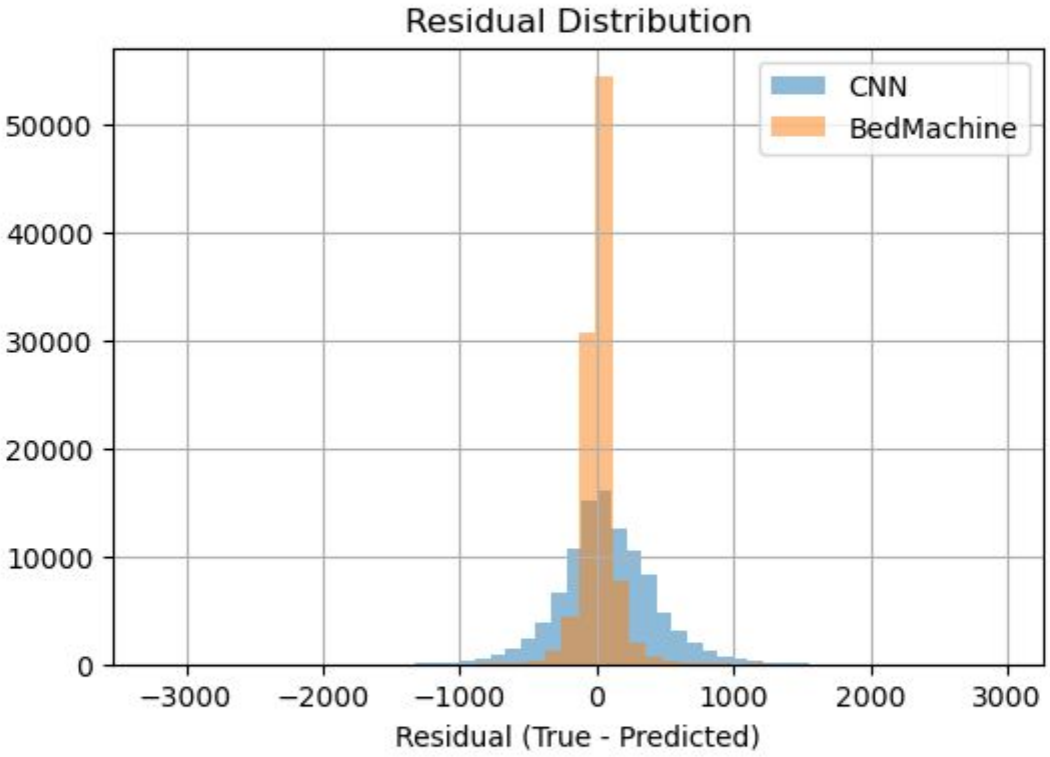
Dense layers optimise over (1-3)

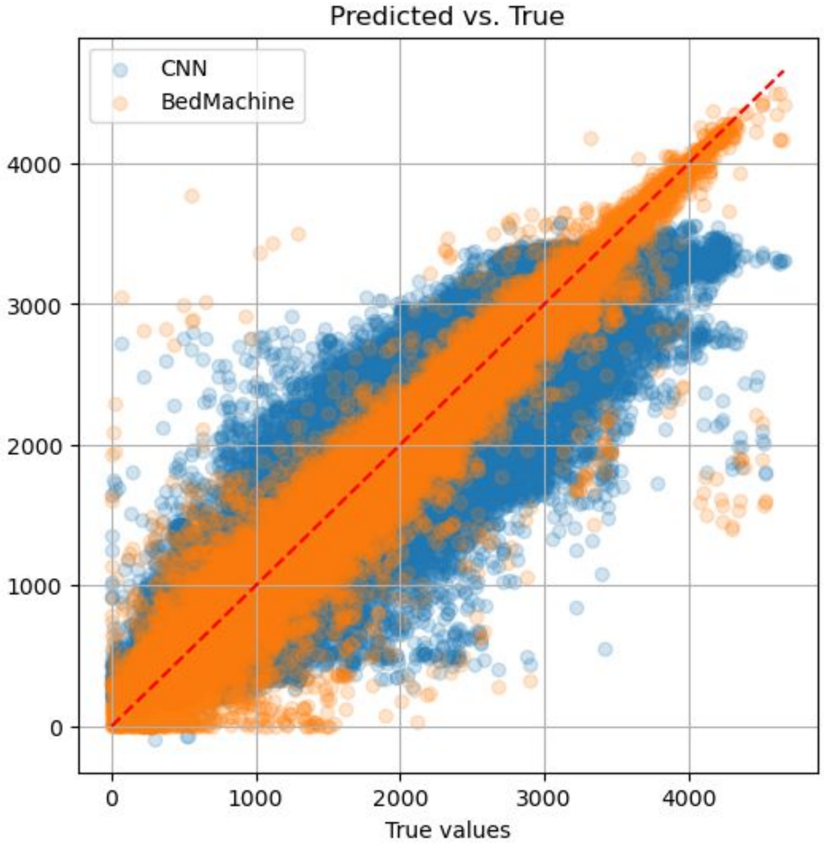
- Units (optimise over 64-256)
- Dropout lvl (optimise over 0.3-0.7)
- activation: ReLU



CNN - Surface Mass Balance

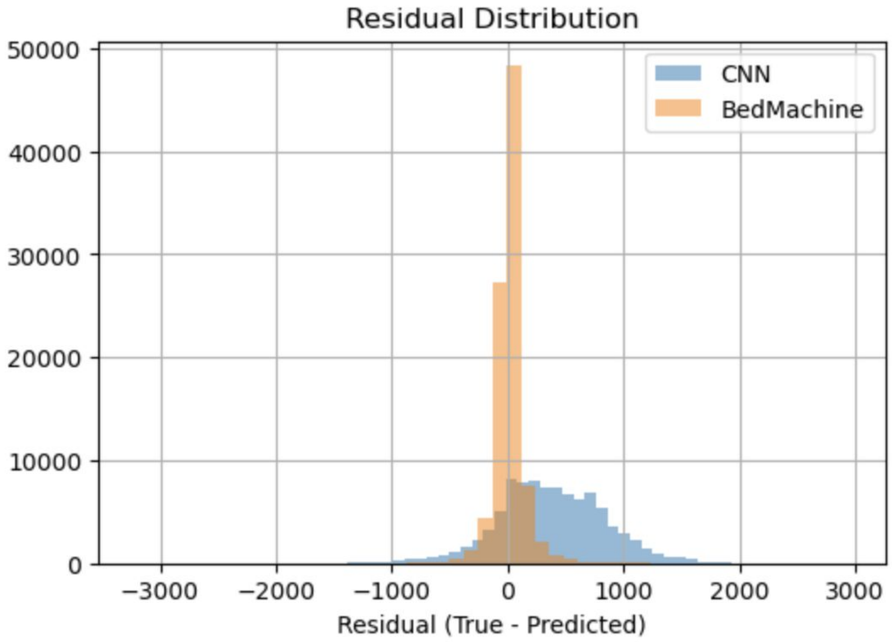
MAE = 268 [m]





CNN - Temperature

- Resolution: ca. 2605 by 2605 meters
- MAE: 475.01 [m]



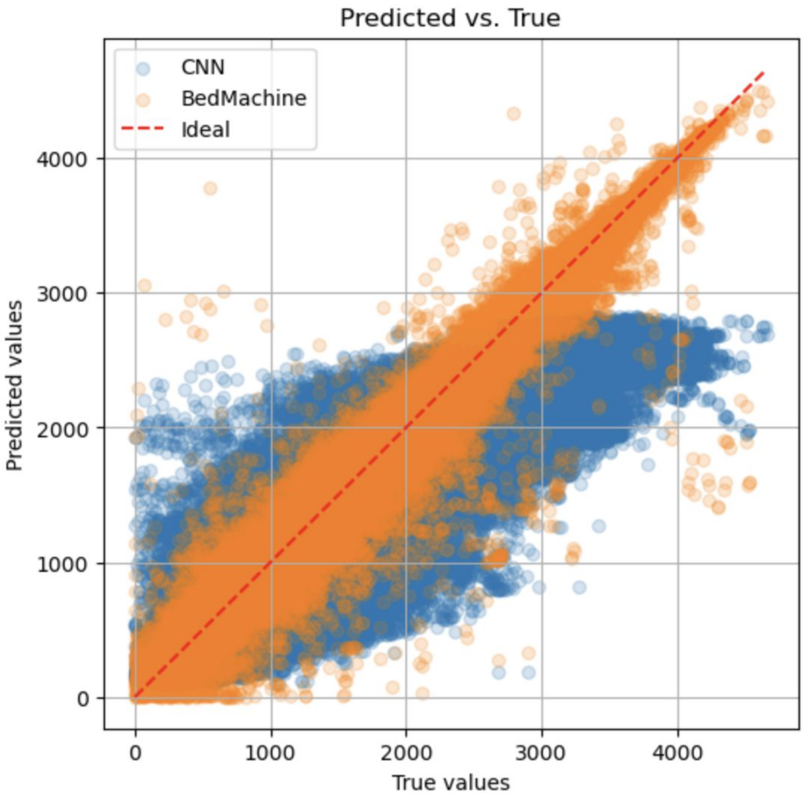
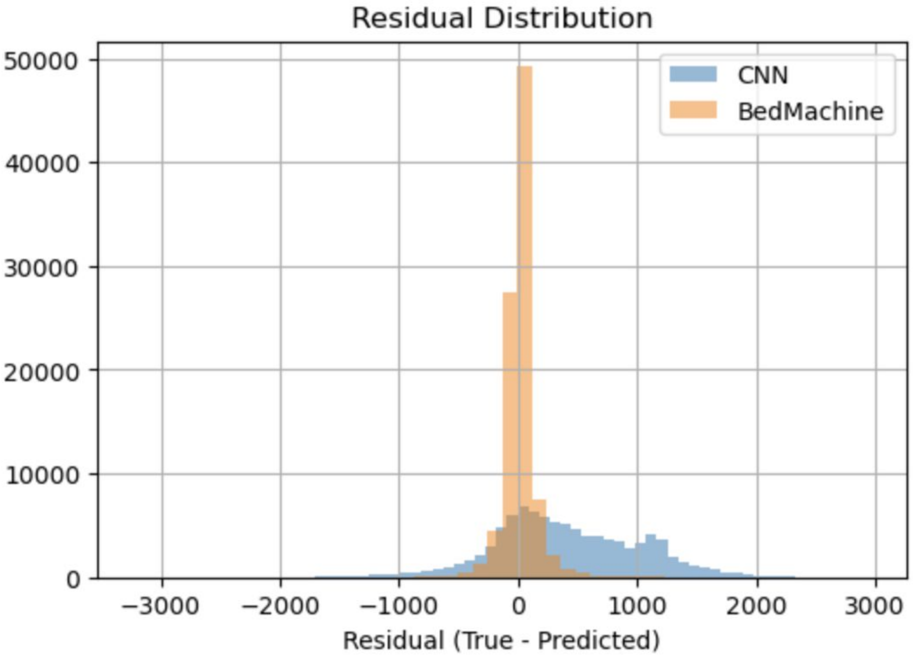


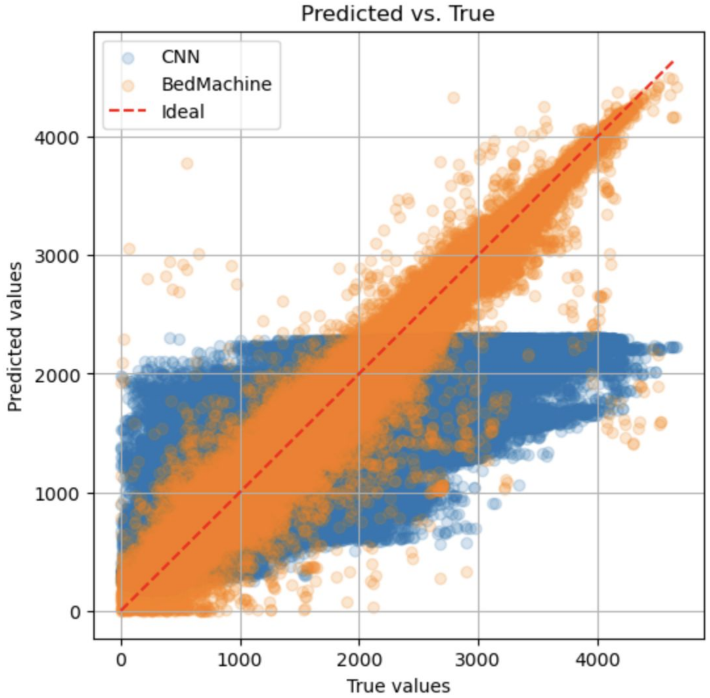
Image source: https://makeagif.com/gif/awesome-glacier-iceberg-time-lapse-moving-colliding-falling-floating-melting-ice-_7v1UJ

CNN - Velocity x

- Resolution: 450 by 450 meters
- MAE: 568.92 [m]

Direction x →





CNN - Velocity y

- Resolution: 450 by 450 meters
- MAE: 519.7 [m]

Direction y

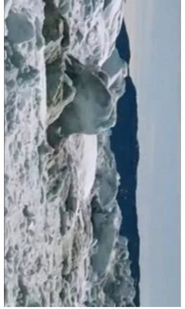
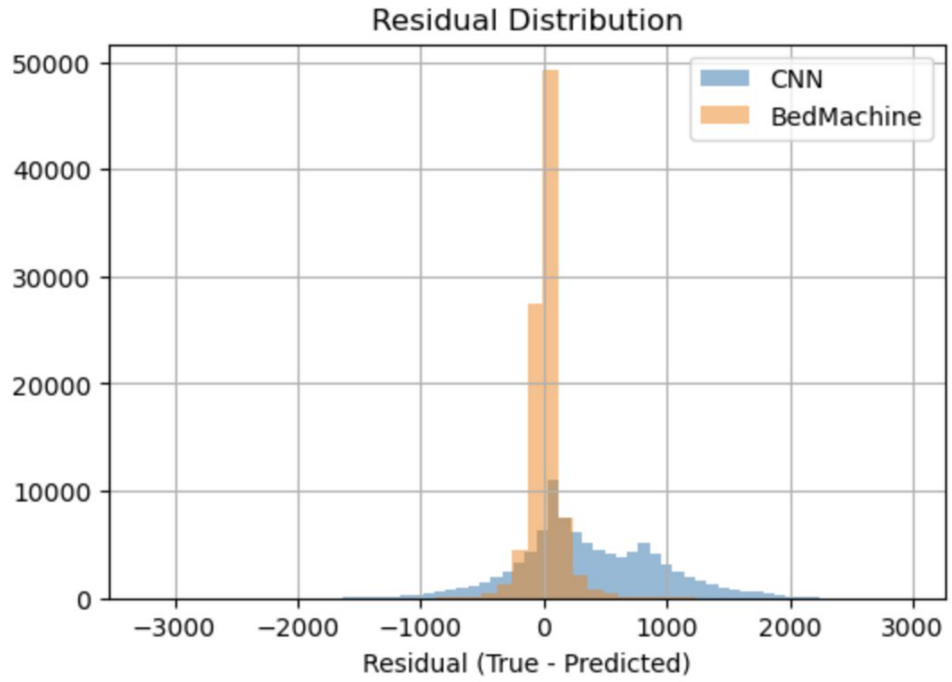
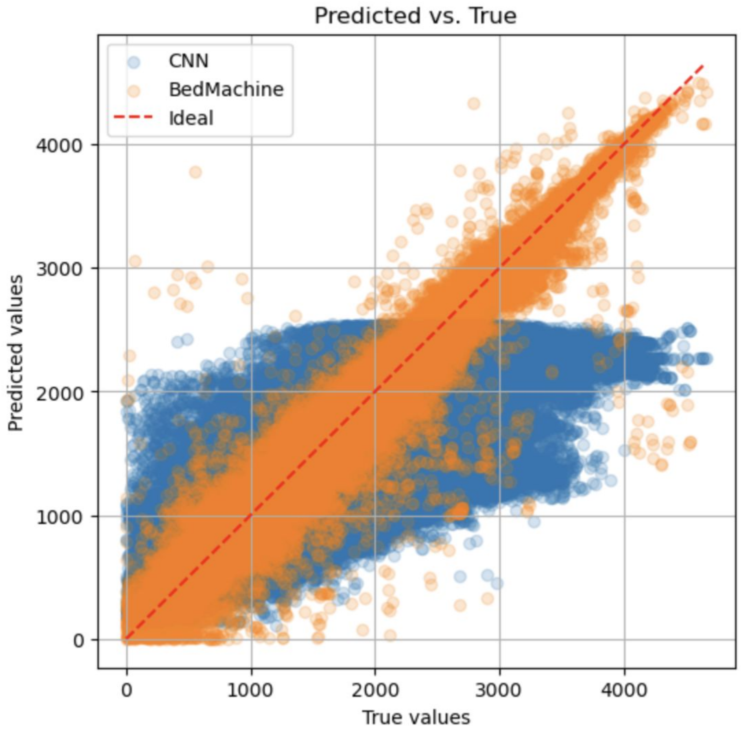


Image source: https://makeagif.com/gif/awesome-glacier-iceberg-time-lapse-moving-colliding-falling-floating-melting-ice-_7v1UJ



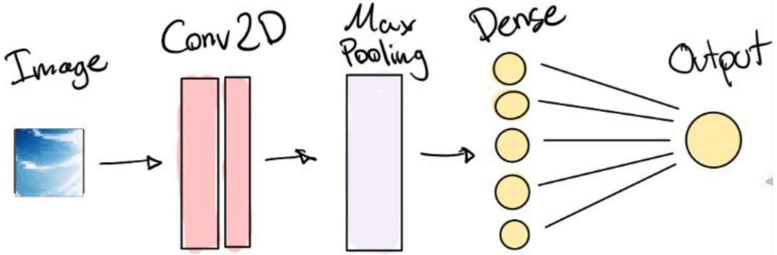


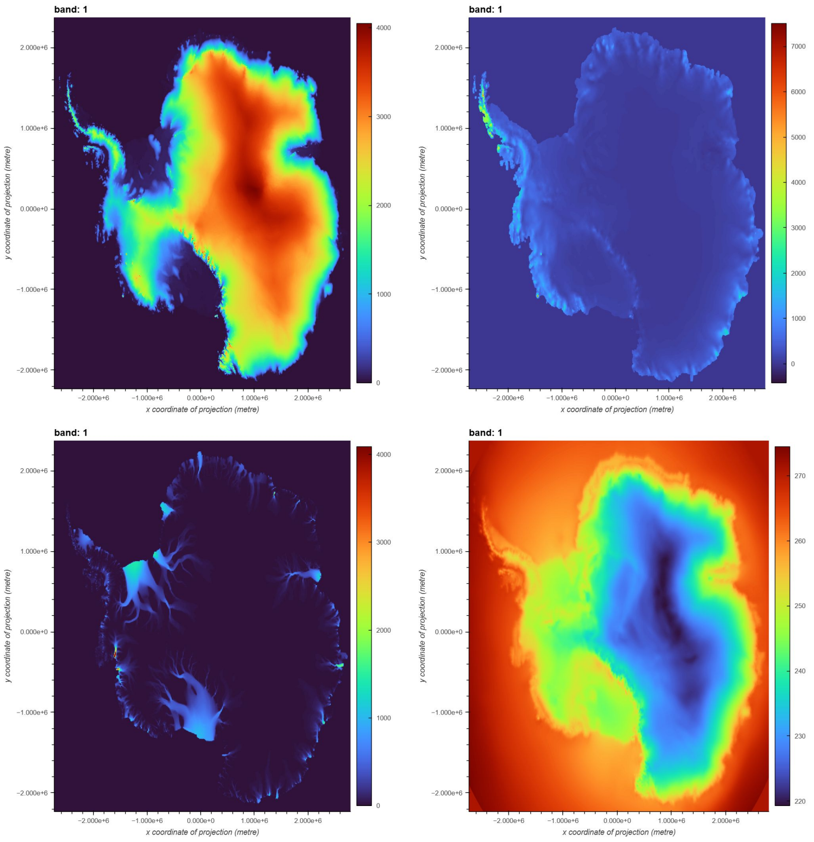
CNN all images

Preprocessing:

- Universal Raster
- Reprojection using bilinear resampling method:
- Averaging ice thickness.

CNN architecture (Images only)

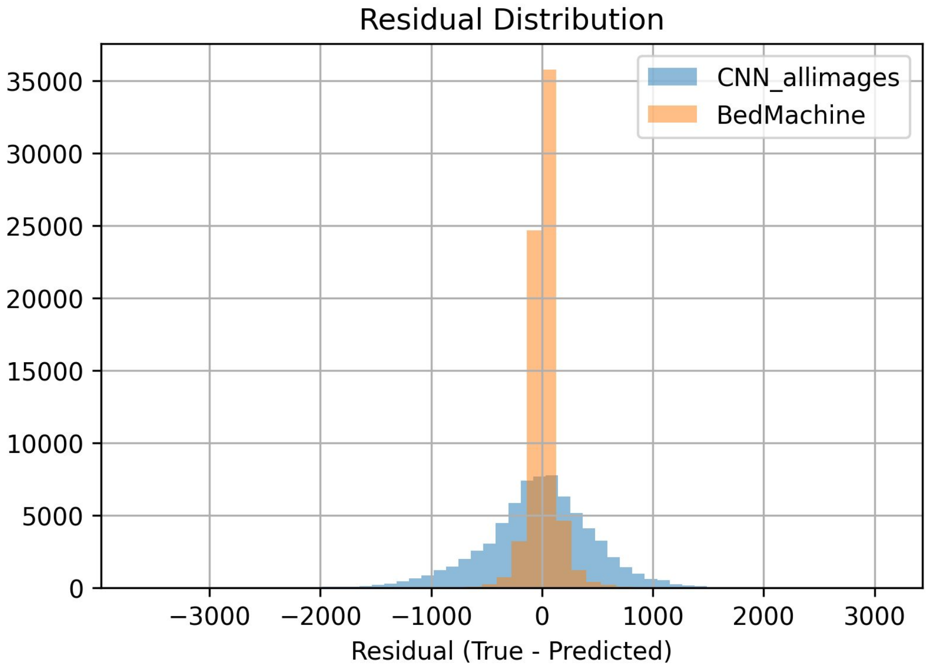


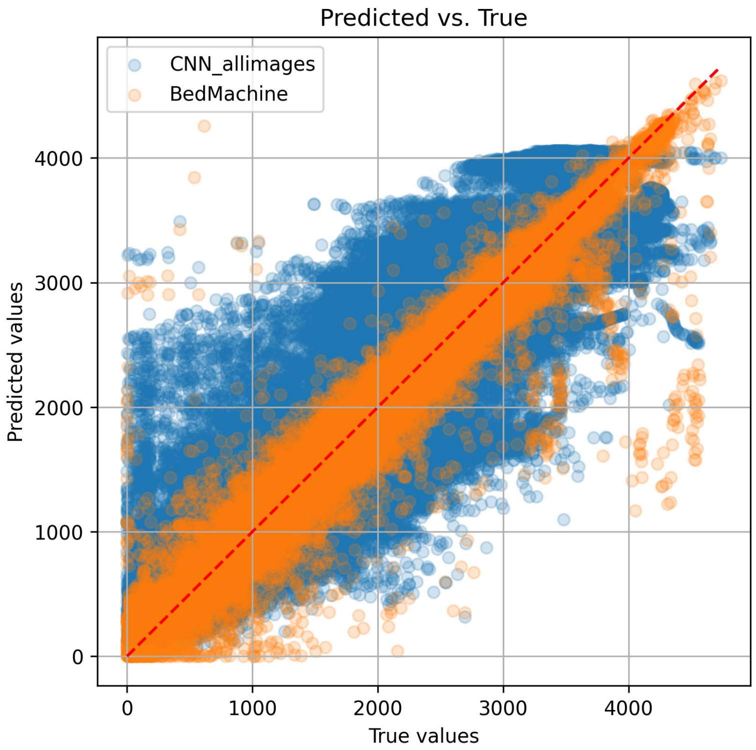


CNN all images

CNN MAE: 382 [m]

BedMachine MAE: 80 [m]





Did the images help?

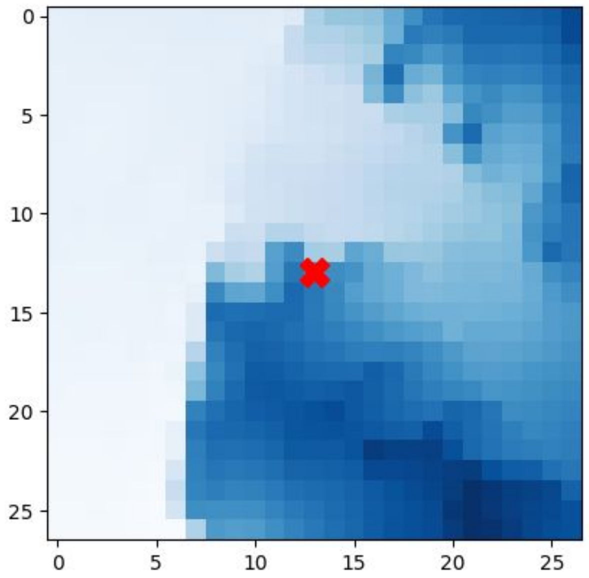
Baseline MAE: 291.8[m]

MAE after permuting images: 540.6[m]

Almost double loss!

BUT:

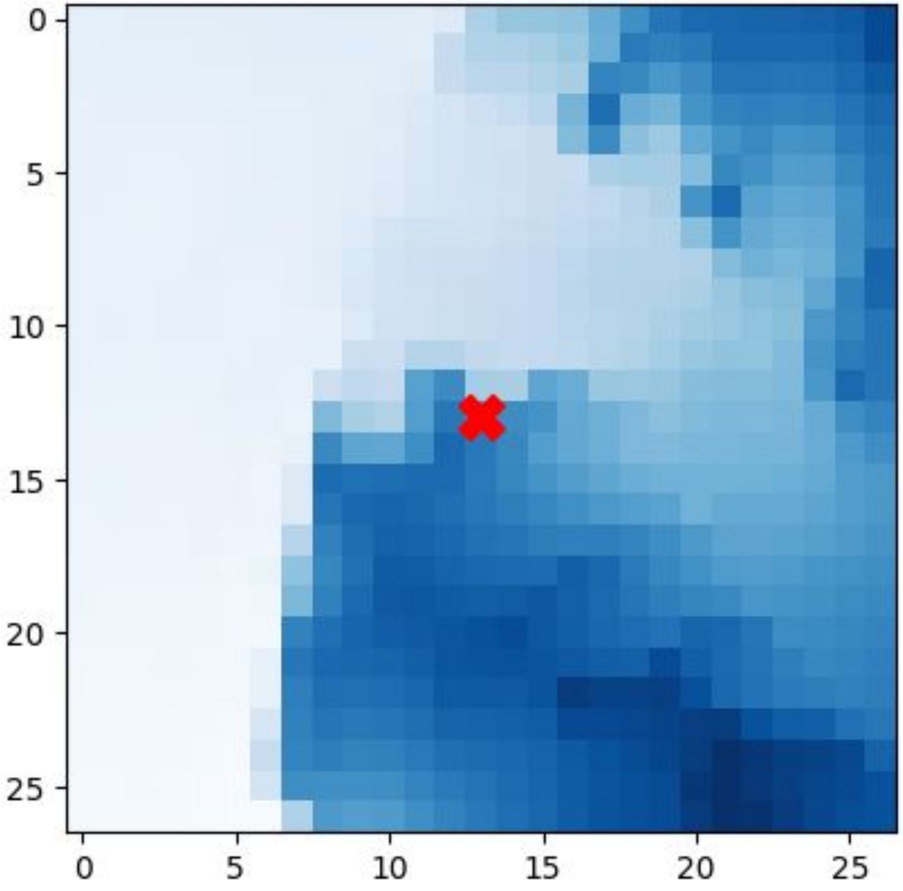
Did it use the surrounding area or just the center pixel?



Possible Workarounds

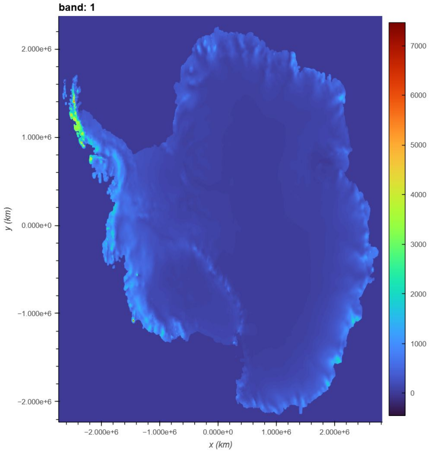
- Intentionally corrupting images, instead of permutation

- Remove actual elevation information from the images (only keep relative geospatial features)



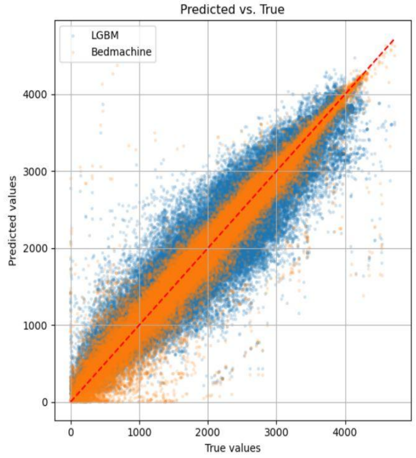
Model Summary

- More complex → Larger MAE
- Images contain useful information
- Most important image:
  SMB

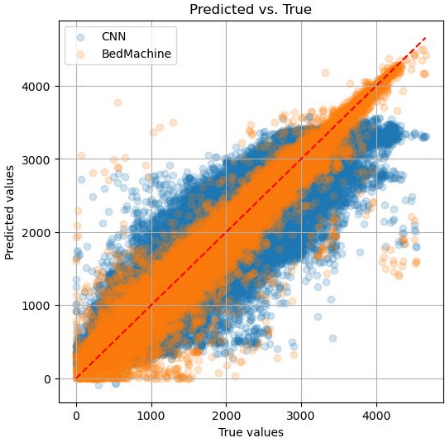


Surface elevation CNN also performed well though.

Best Model: LightGBM
MAE: 205 [m]



Best CNN Model: SMB
MAE: 268 [m]



Potential future steps and considerations

- Other means of nan handling e.g. closest value, knn other
- train on more data with better hardware
- Try different image and filter sizes
- optimise on epochs and batch sizes
- Include bedmachine as a feature as an ensemble method
- Scientific ML

$$\nabla \cdot (H\vec{v}) \approx smb$$

Appendix

Duplicates

Among the 80 million measurements of thickness by airplane there are 2.754.608 duplicates.

We took the mean value of thickness at these coordinates.

On uncorrelated data the MAE of LGBM actually increased from 204.80 to 206.09!

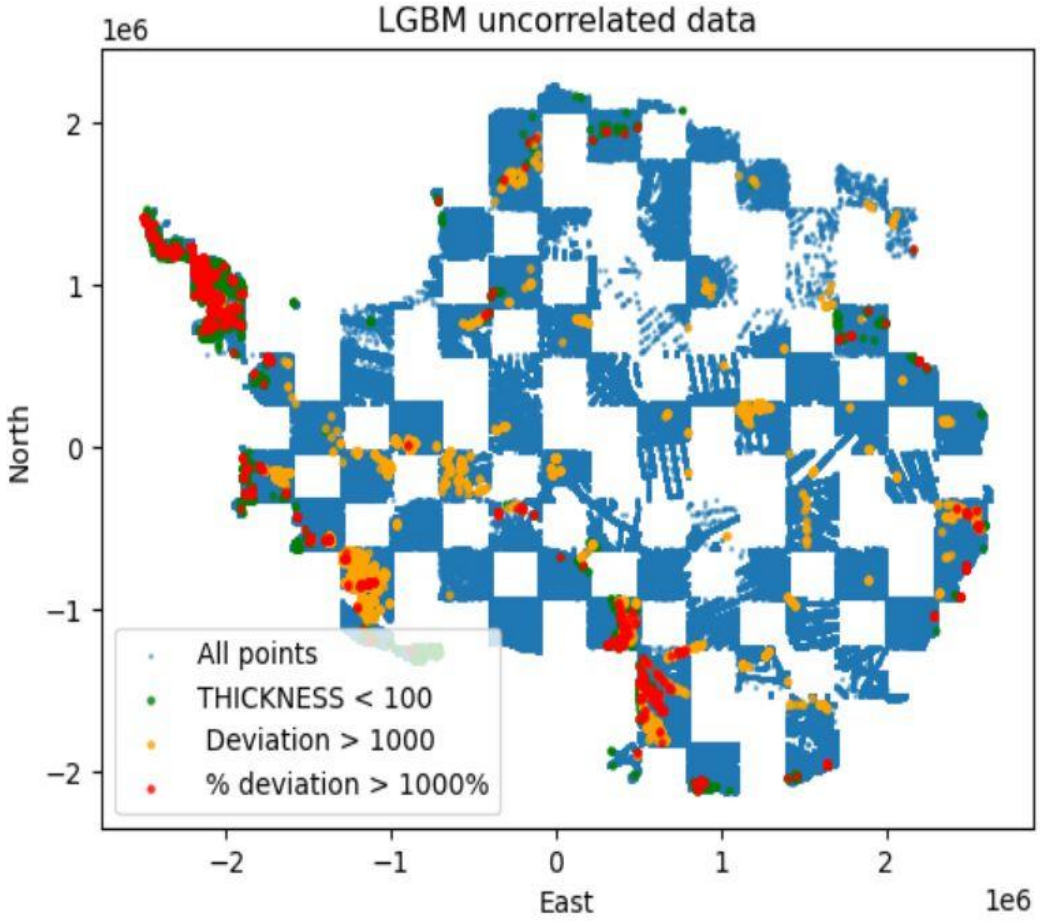
```
print(f"Antal dubletter (samme EAST & NORTH): {num_duplicates}")

coord_counts = df.groupby(['EAST', 'NORTH']).size().reset_index(name='count')
duplicate_stats = coord_counts['count'].value_counts().sort_index()

print(duplicate_stats)
```

```
Antal dubletter (samme EAST & NORTH): 2754608
1       75519114
2        1499771
3          26340
4          48011
5          13111
          ...
187            1
273            1
491            1
1439           1
2390           1
Name: count, Length: 160, dtype: int64
```

LGBM

Uncorrelated data

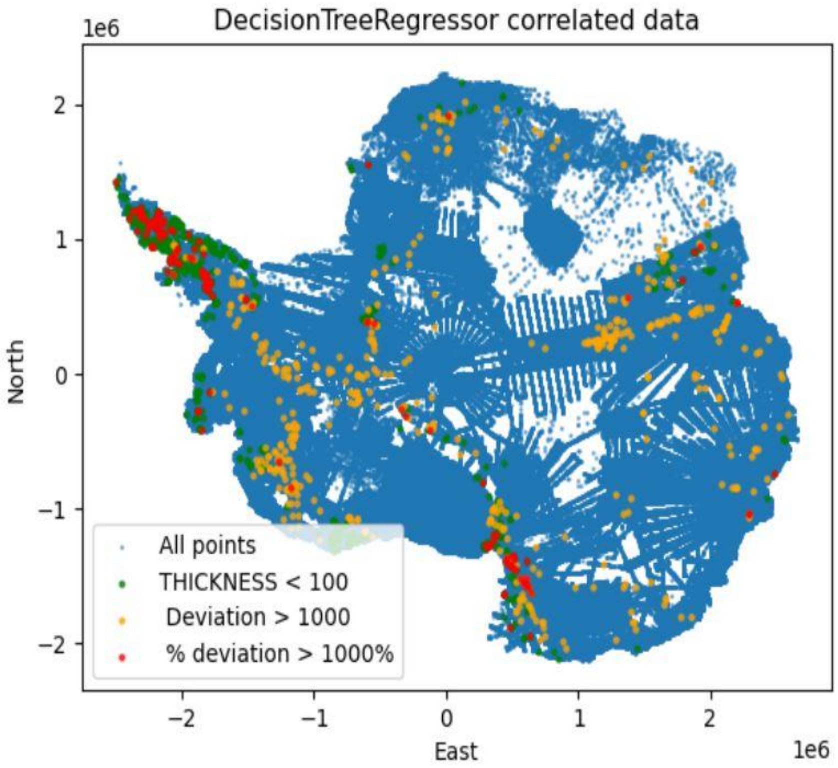
MAE = 204.80



DecisionTreeRegressor (correlated data)

Training and test data from the whole of Antarctica are split randomly with train_test_split from sklearn.model_selection.

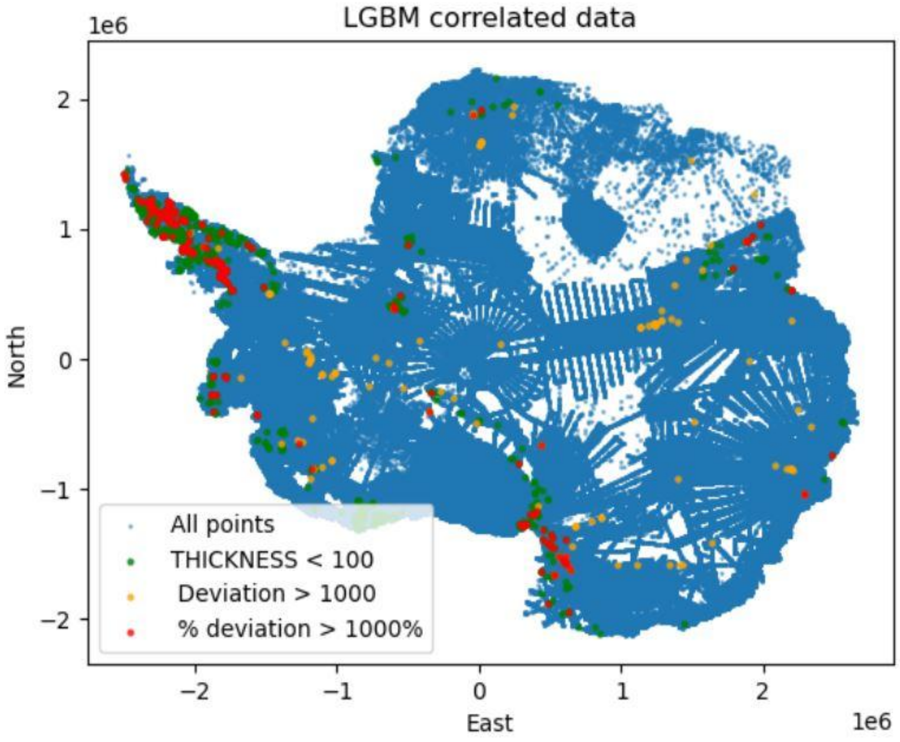
MAE = 87.73



LGBM (correlated data)

Training and test data from the whole of Antarctica are split randomly with train_test_split from sklearn.model_selection.

MAE = 124.46

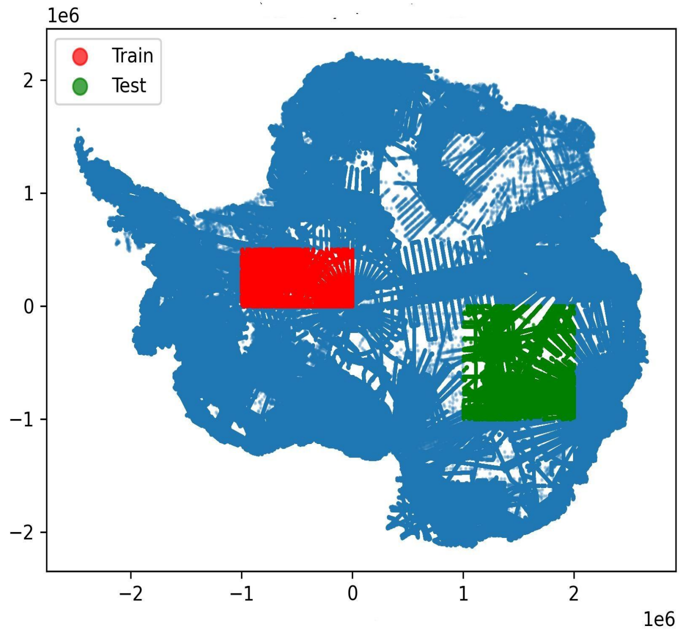


Trees

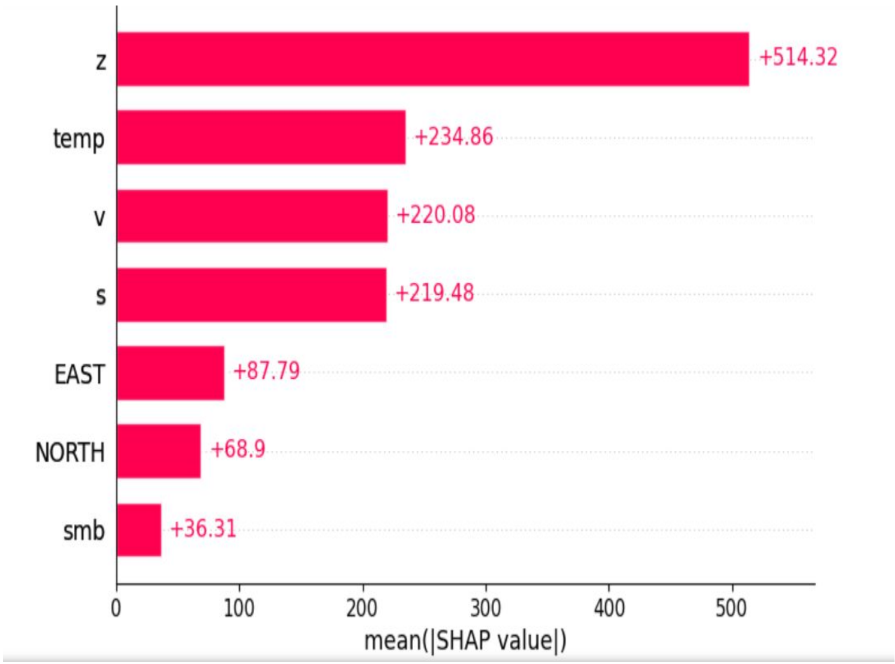
DecisionTreeRegressor and LGBM with very distant train and test area.

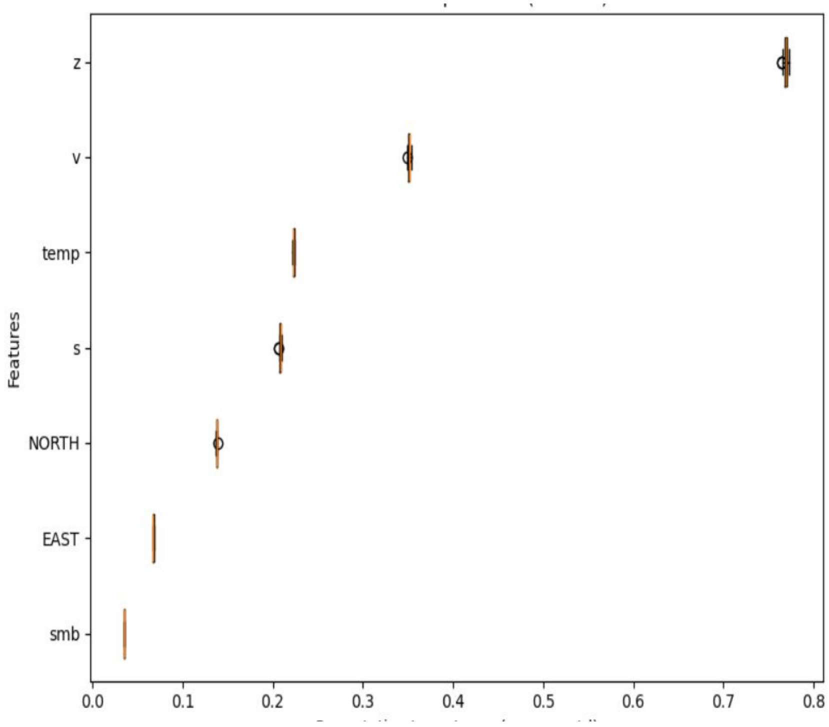
The features are very different for the train and test area.

MAE is about 1000 for both models.



LGBM (input feature ranking - correlated data)





LGBM (hyperparameter optimization - correlated data)

Gridsearch, randomsearch and Bayesian optimization has been performed.

Best parameters are shown to the right.

```
# Parametre
params = {
    'objective': 'regression',
    'metric': 'rmse',
    'boosting_type': 'gbdt',
    'learning_rate': 0.1528,
    'min_child_samples': 27,
    'n_estimators': 262,
    'num_leaves': 84,
    'verbosity': -1
}
```

```
-------------- Duplicate handling ------------------
We experienced a lot of measurements on the same (east,north) coordinates
To examine the amount of duplicates and how much the measurements differentiated
We took random points according to 10% of the 30m training data (3m)

We got:
Unique (EAST, NORTH) pairs that have duplicates: 8,369
no. observations with duplicated (EAST,NORTH) pairs 20,749

We made a summary metric with this procedure:
by (east,north)
Thick_range = max(THICK) - min(THICK)

by (east, north)
THICK_range_ration = Thick_range / mean(THICK)

no. (east, north) pairs with THICK_range_ration > 1%  : 2,317
no. (east, north) pairs with THICK_range_ration > 2.5%: 1,259

Most duplicates can be sorted by removing those with THICK_range_ration larger than 1%
and then taking the median observation for those under.
```

Tabular NN with distance variable

Hyper parameters:

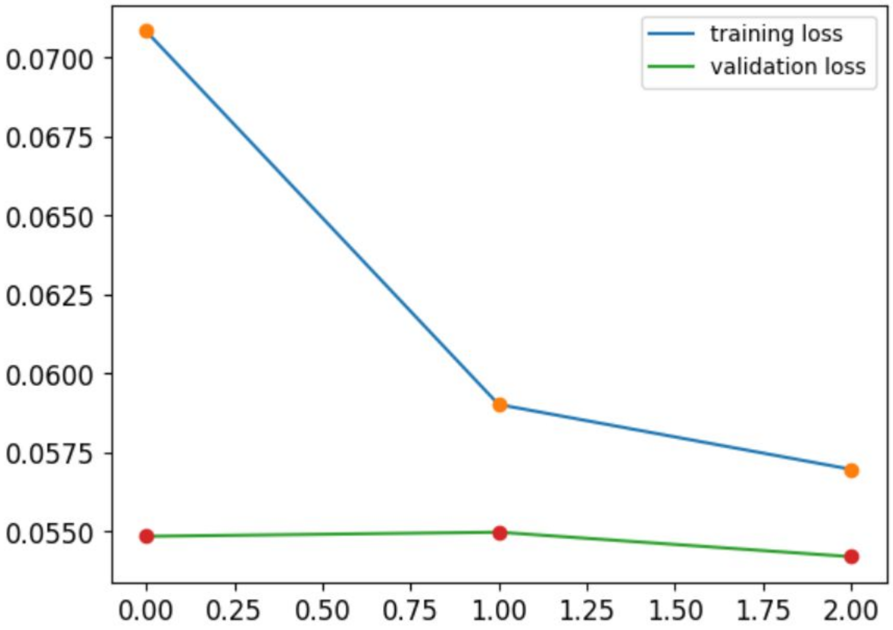
```
{'num_dense_layers': 3, 'dense_units_0': 128, 'dropout_0': 0.4, 'dense_units_1': 64, 'dropout_1': 0.2, 'dense_units_2': 96, 'dropout_2': 0.4}
```

Training, Loss over Epoch Graph:

Batch Size: 320

Trained on 1.5 M data points.

Adam optimizer used.

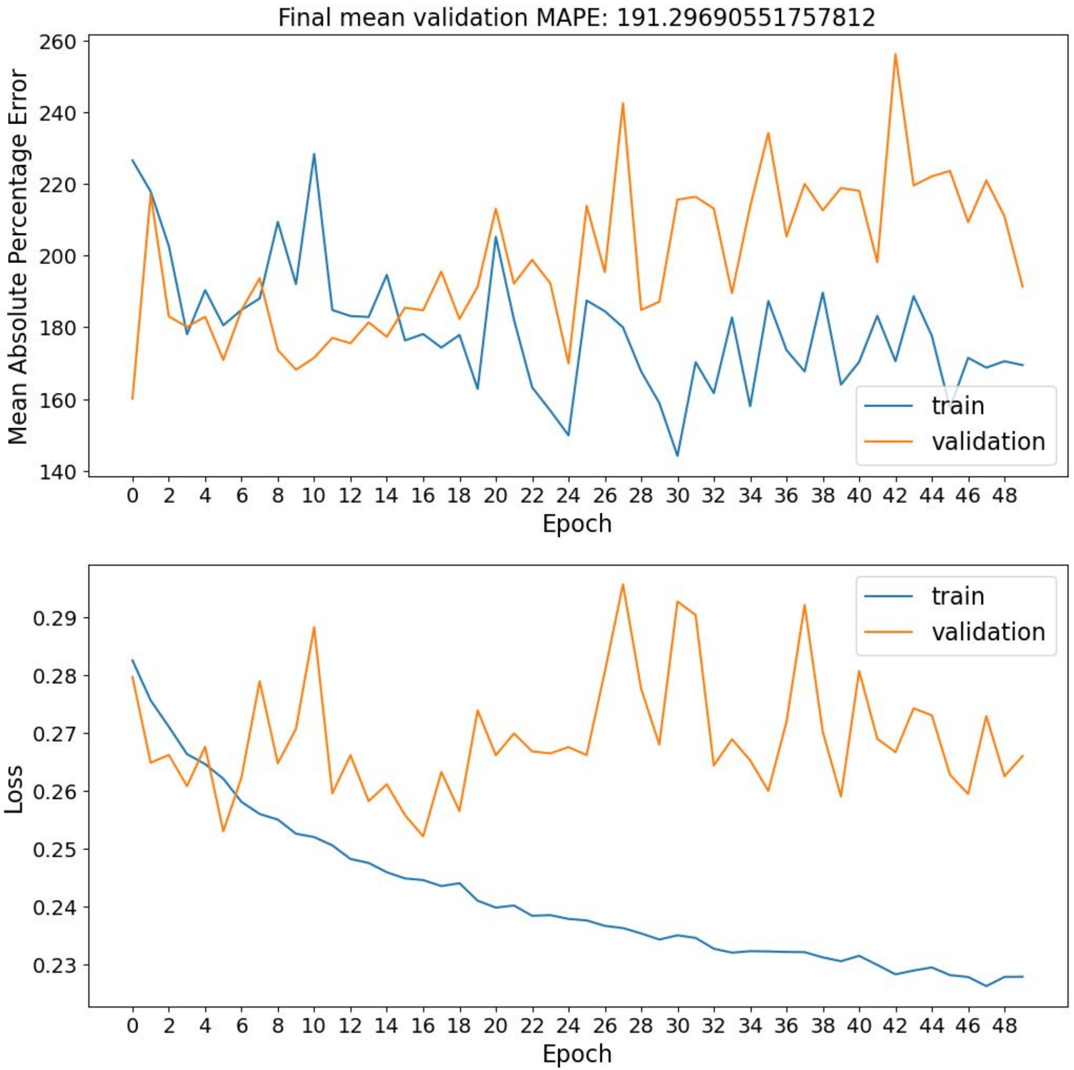


CNN-SMB

with smb as

image variable

training loss



CNN-SMB

with smb as

image variable

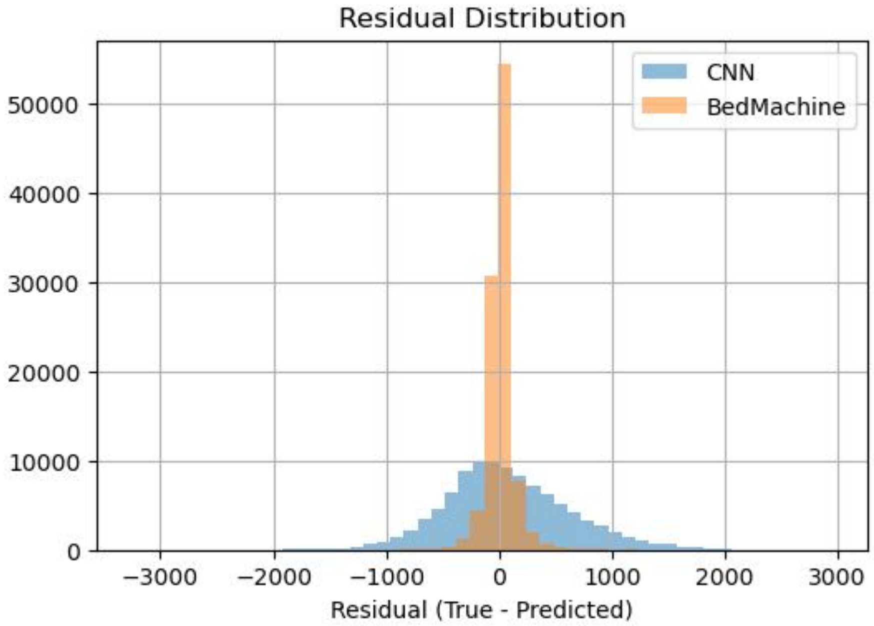
Description of

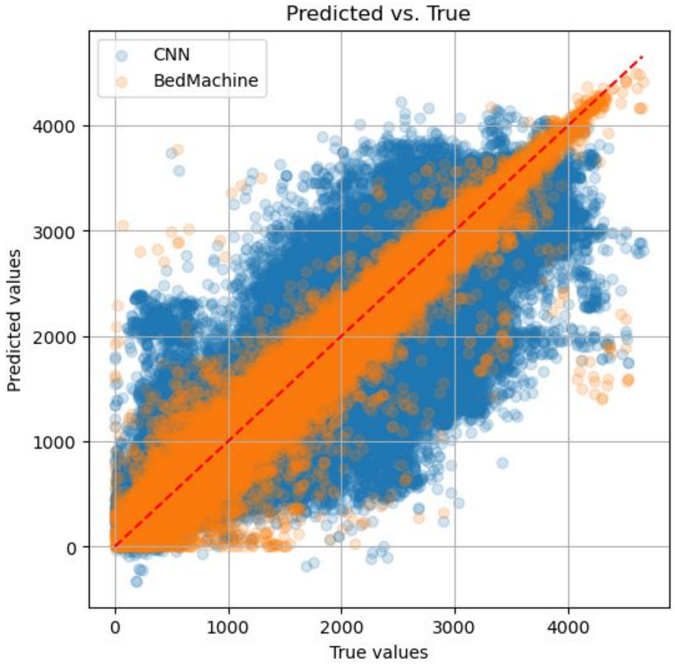
tuning

```
Standard scaled scalar features, images and target (mean=0, std=1)

HP optimization procedure:
Random search
over HP space common for other CNN's
50 trials with 10 epochs (early stopping on 3 according to mae)
bachsize = 512
HP time elapsed: 10h 16m 40s

Optimised HP:
'num_conv_layers': 2,
'filters_0': 96,
'num_dense_layers': 3,
'dense_units_0': 128,
'dropout_0': 0.3,
'dense_units_1': 192,
'dropout_1': 0.3,
'dense_units_2': 192,
'dropout_2': 0.6000000000000001,
'filters_1': 32


then trained with 50 epochs batchsize = 64
approx 2 hours training.
```

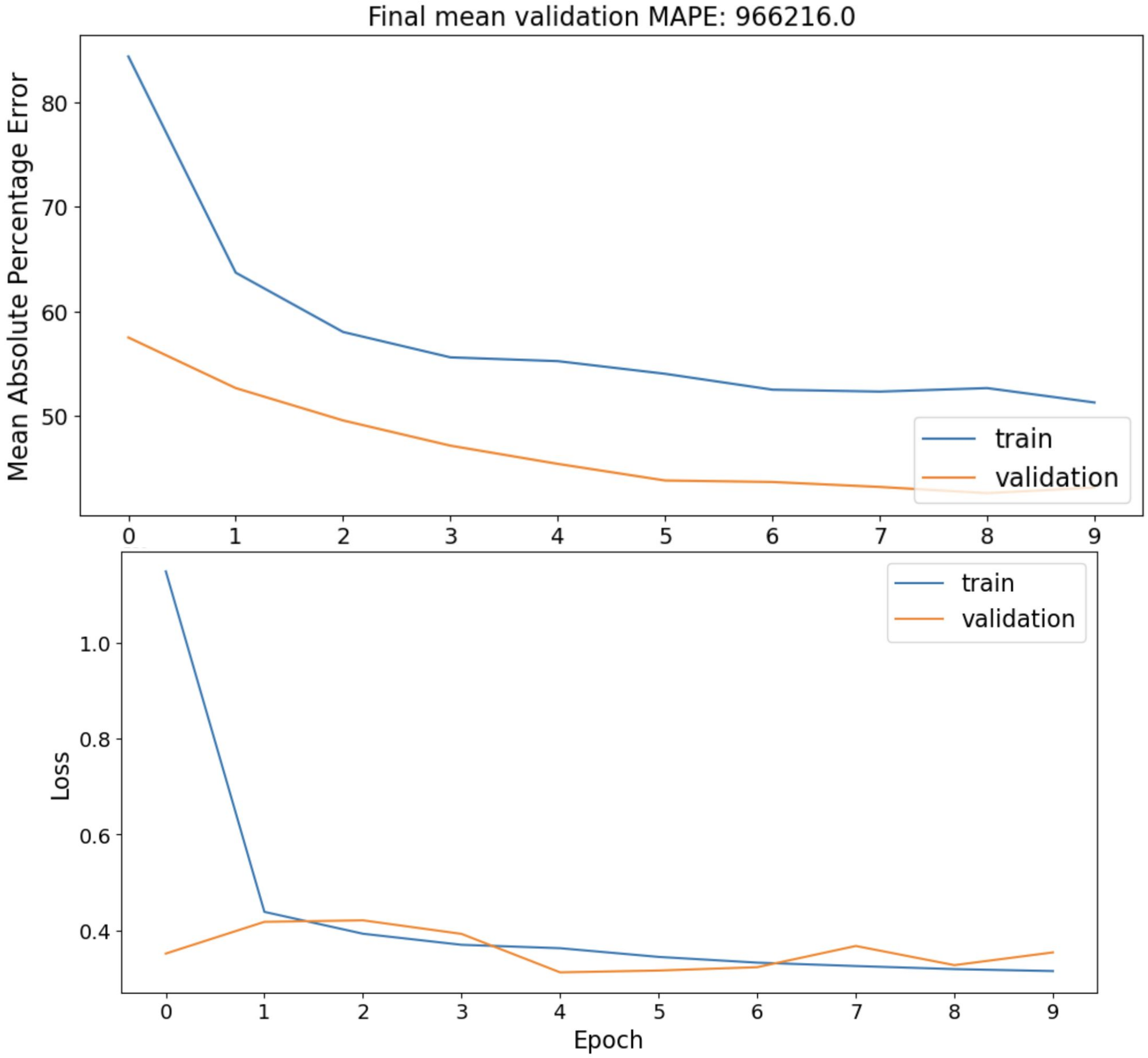
CNN-SMB trained on binned training data so thickness was uniformly distributed to see if it could capture larger values better. It did not help overall:

MAE = 438





CNN - temperature



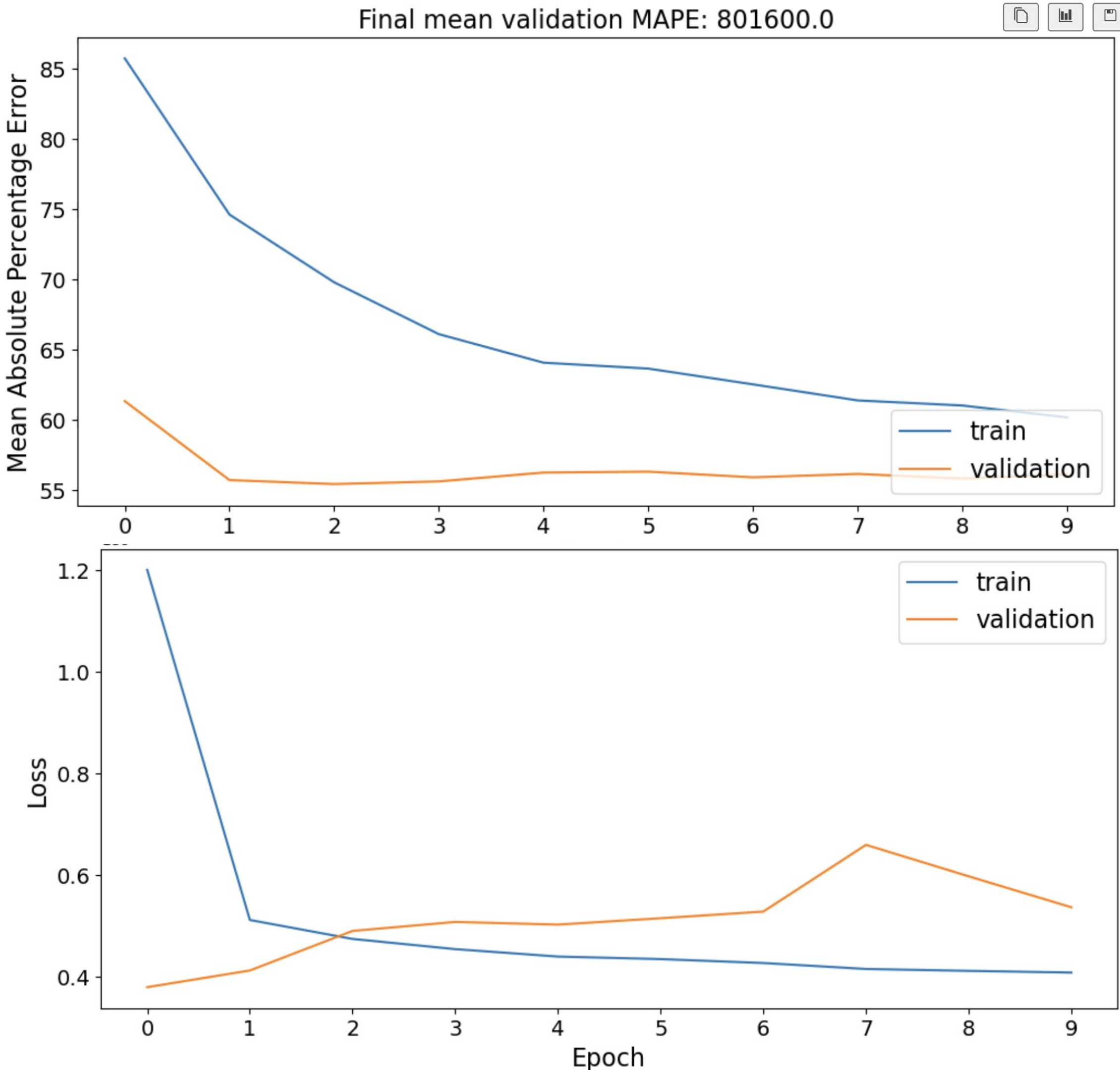
CNN - Temperature

Standard scaled scalar features, images and target (mean=0, std=1)

HP optimization procedure:
Random search
over HP space common for other CNN's
10 trials with 10 epochs
bachsize = 512

Optimised HP:
'num_conv_layers': 1,
'filters_0': 96,
'num_dense_layers': 3,
'dense_units_0': 256,
'dropout_0': 0.3,
'dense_units_1': 64,
'dropout_1': 0.5,
'dense_units_2': 128,
'dropout_2': 0.6000000000000001,
'filters_1': 64
'filters_2': 96

CNN - Velocity x



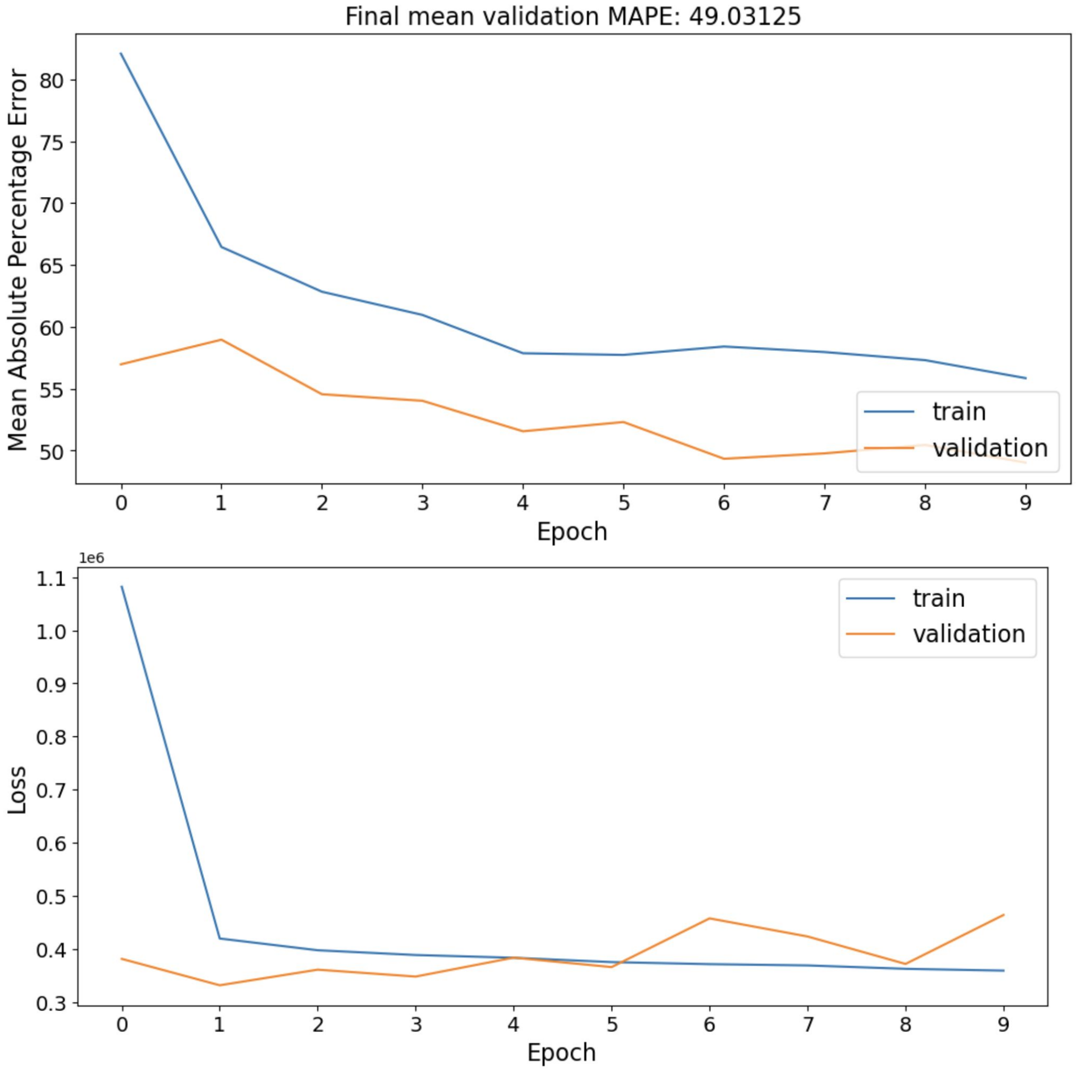
CNN - Velocity x

Standard scaled scalar features, images and target (mean=0, std=1)

HP optimization procedure:
Random search
over HP space common for other CNN's
10 trials with 10 epochs
bachsize = 512

Optimised HP:
'num_conv_layers': 3,
'filters_0': 64,
'num_dense_layers': 2,
'dense_units_0': 256,
'dropout_0': 0. 6000000000000001,
'dense_units_1': 64,
'dropout_1': 0.4,
'dense_units_2': 192,
'dropout_2': 0.3,
'filters_1': 64
'filters_2': 32

CNN - Velocity y



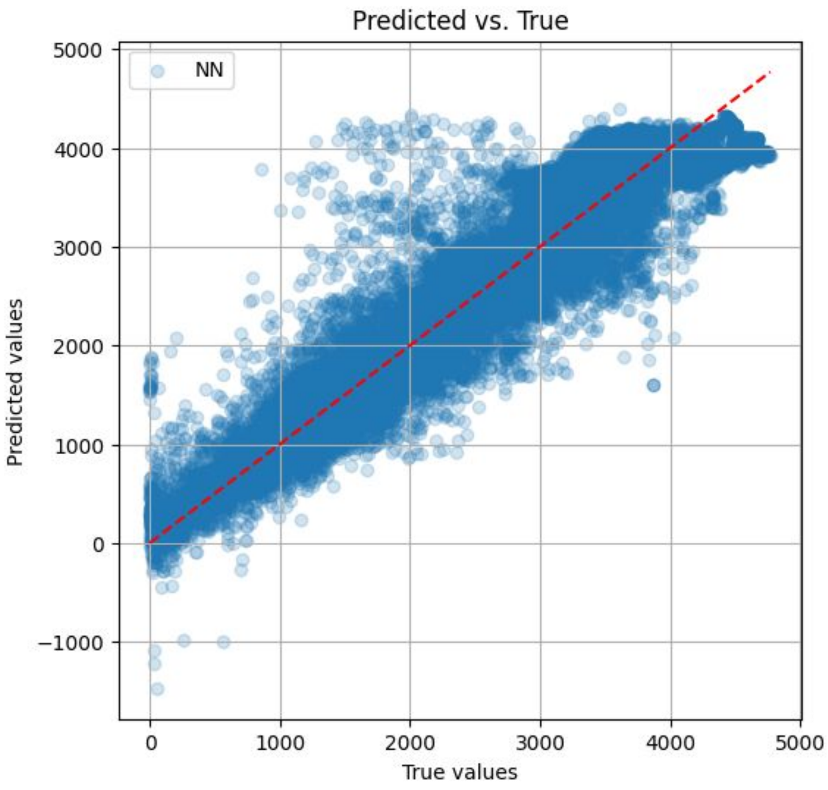
CNN - Velocity y

Standard scaled scalar features, images and target (mean=0, std=1)

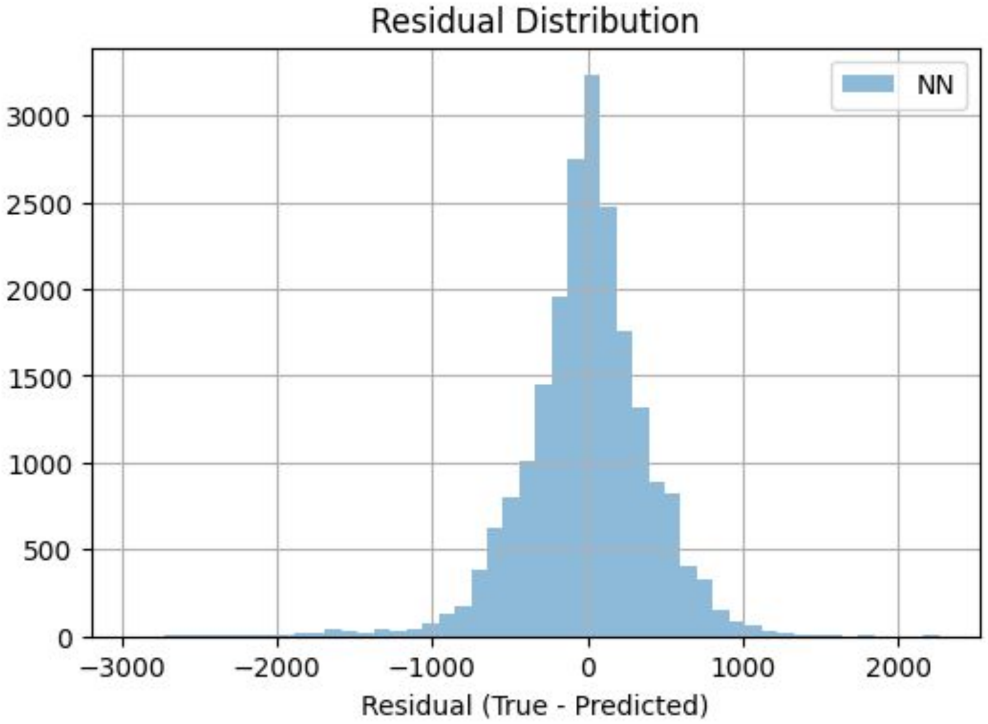
HP optimization procedure:
Random search
over HP space common for other CNN's
10 trials with 10 epochs
bachsize = 512

Optimised HP:
'num_conv_layers': 1,
'filters_0': 32,
'num_dense_layers': 3,
'dense_units_0': 192,
'dropout_0': 0. 3,
'dense_units_1': 64,
'dropout_1': 0.3,
'dense_units_2': 64,
'dropout_2': 0.3,

Silas Surface Elevation CNN - Bayesian Search Tuned



MAE = 292 [m]



CNN all images

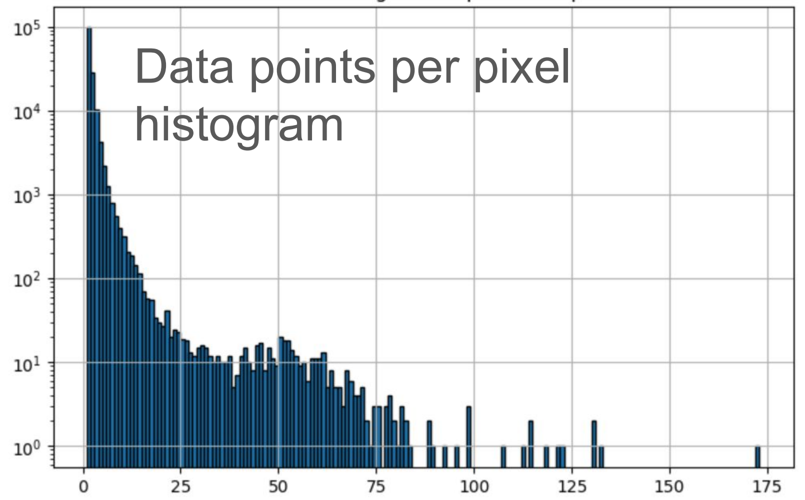
Batch size: 256

Optimizer: Adam

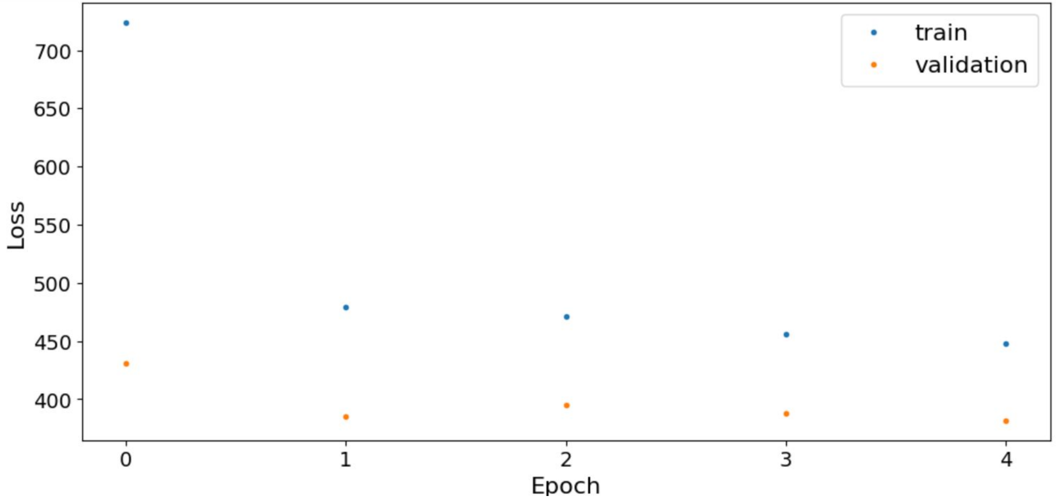
Image size: 27 x 27 pixels

Number of images trained on:

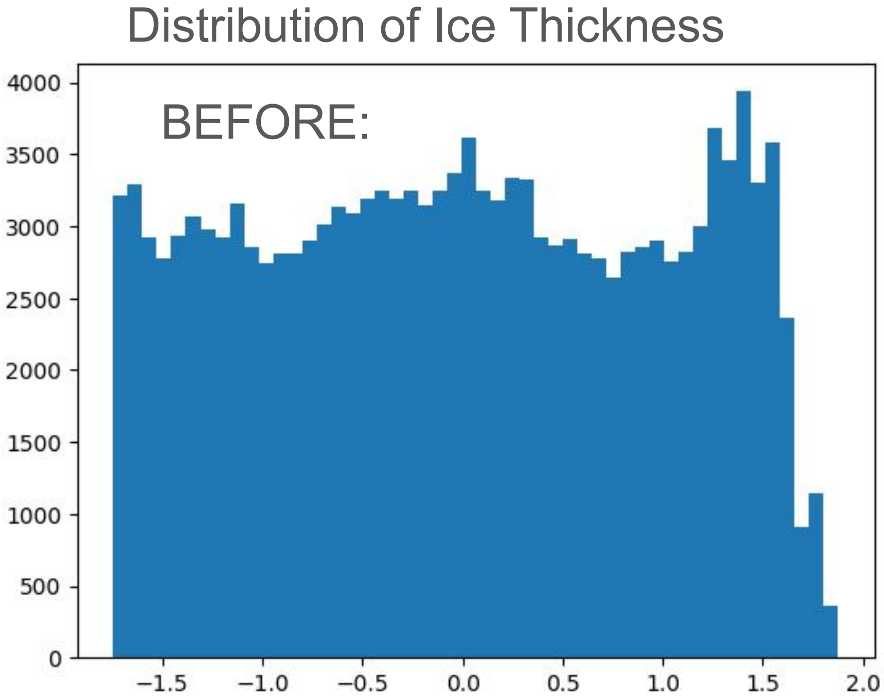
4*76651

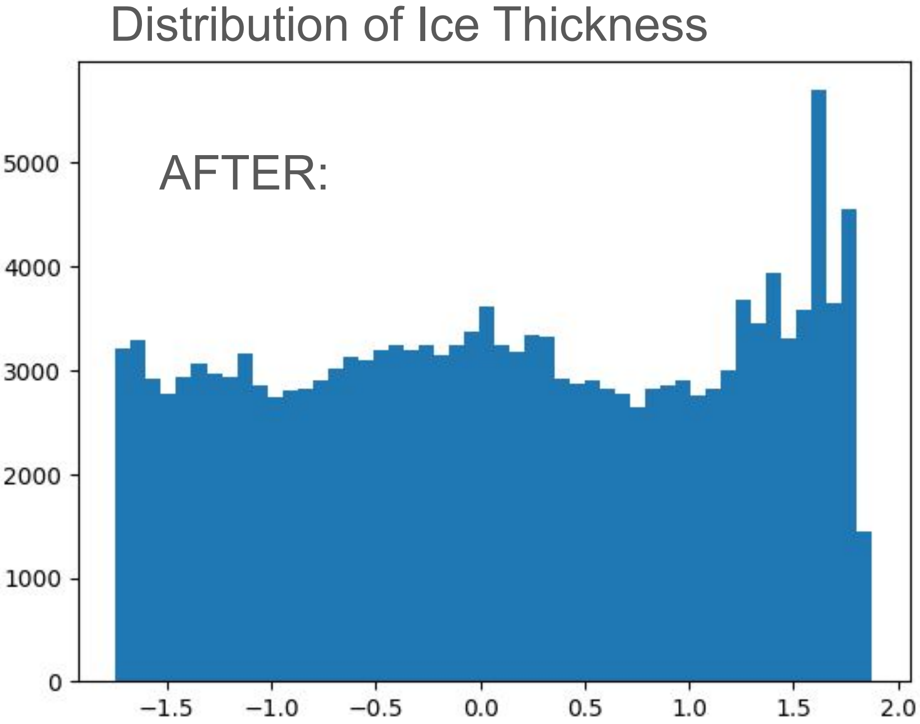


Training, Loss over Epoch graph:



Data Augmentation for Deep Ice





Failed PINN attempt:

We didn't spend a lot of time on it and scaling variables is hard because it messes with the physics equation. But Extremely interesting!

```
def physics_loss(model, x_colloc, smb):
    with tf.GradientTape(persistent=True) as tape:
        tape.watch(x_colloc)
        H, u, v = model(x_colloc)

    # Calculate the divergence of the velocity field
    div_u = tape.gradient(u, x_colloc)[:, 0:1]\
        + tape.gradient(v, x_colloc)[:, 1:2]

    del tape

    # Calculate the residual of the mass conservation equation
    residual = H * div_u - smb

    return tf.reduce_mean(tf.square(residual))
```