Antarctic ice sheet: Estimation of ice thickness

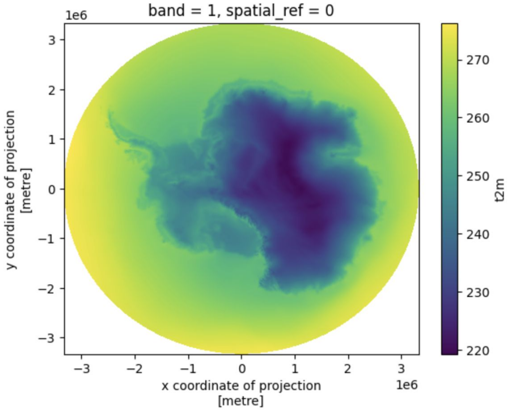
By: Annika, Henrik, Lisa & Natalie

Introduction & Data

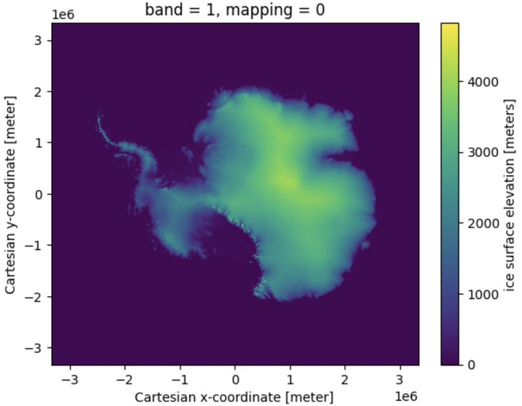
Structure of data:

- 14 rows: Longitude, latitude, thickness (target: radar measurements), geometry, east, north, vx, vy, v, ith_bm, smb, z, s, temp.
- 79,890,423 columns
- Data worked on: 70,883 → 10 km distance, between target points. Dropping geometry, longitude & latitude
- 6 Raster maps; Elevation, Slope, Temperature & Smb
    - Elevation: Coordinate system: EPSG:3031, Resolution: (500, -500), Nan values: 0
    - Slope: Coordinate system: EPSG:3031, Resolution:(500, -500), Nan values: 0
    - Temperature: Coordinate system: EPSG:4326, Resolution: (2605.16, -2605.16), Nan values: 1398181
    - Smb: Coordinate system: EPSG:3031, Resolution: (2000, -2000) , Nan values: 2911998
    - vx: Coordinate system: EPSG:3031, Resolution: (450, -450), Nan values: , Heavily dominated by 0 → Not meaningful to intreprelate.
    - vy: Coordinate system: EPSG:3031, Resolution: (450, -450), Nan values: , Heavily dominated by 0 → Not meaningful to intreprelate.

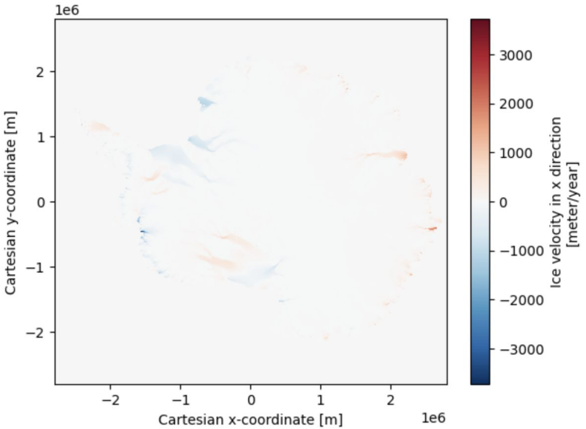
Temperature



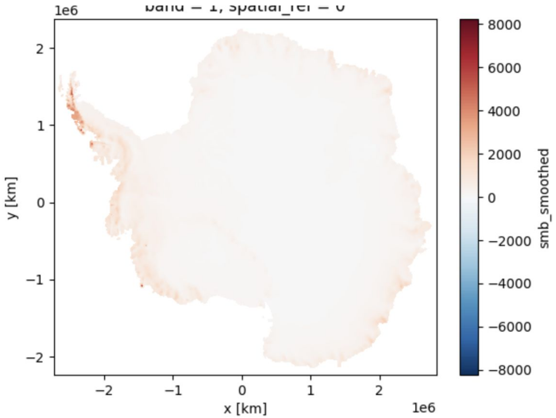
Elevation



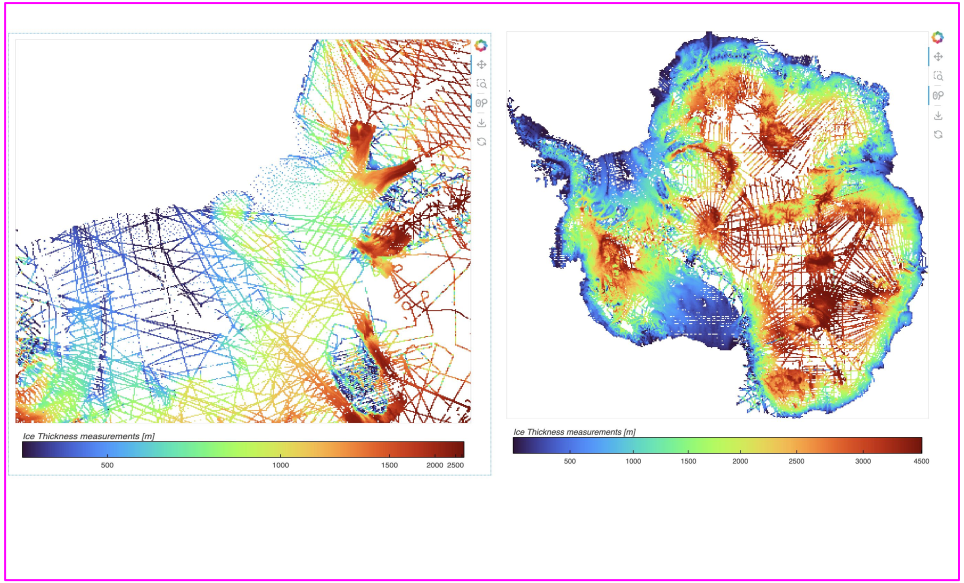
Vx



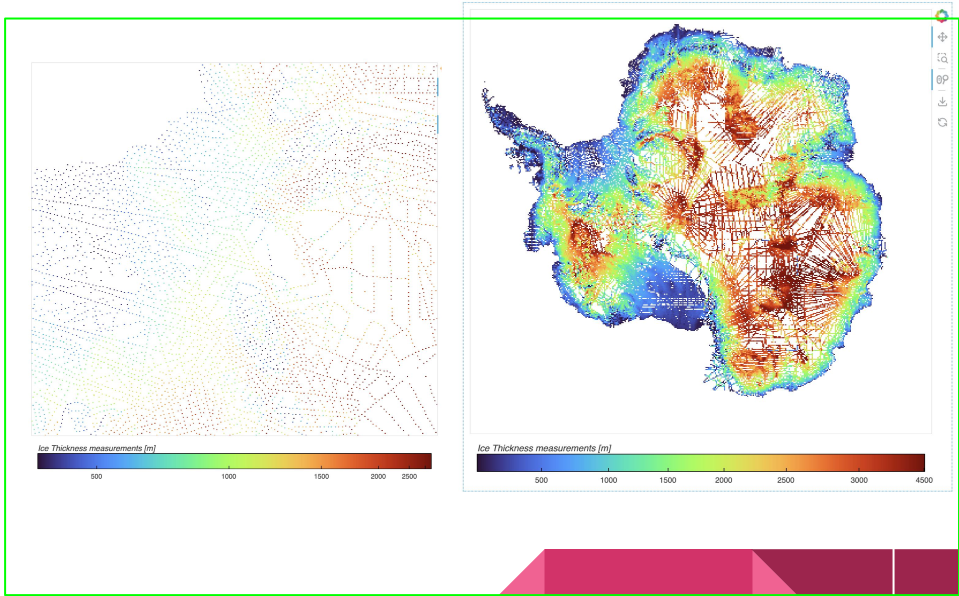
Surface mass balance



Thickness before and after requiring distance of 10 km between points



Before



After

BedMachine as baseline

BedMachine is a physics-informed data model that uses observations and theory to produce the best available map of thickness of Antarctica's ice sheet.

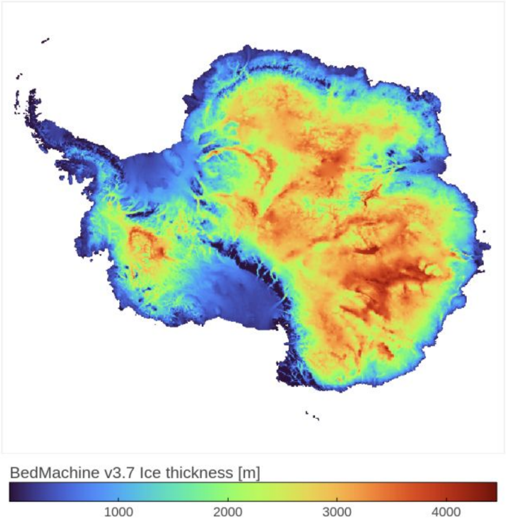
It combines:

- Airborne radar data (ice thickness measurements from plane)
- Satellite data (especially surface elevation and ice velocity)
- Climate models (surface mass balance — snowfall)
- Sea maps and gravity data (to model sea under floating ice)

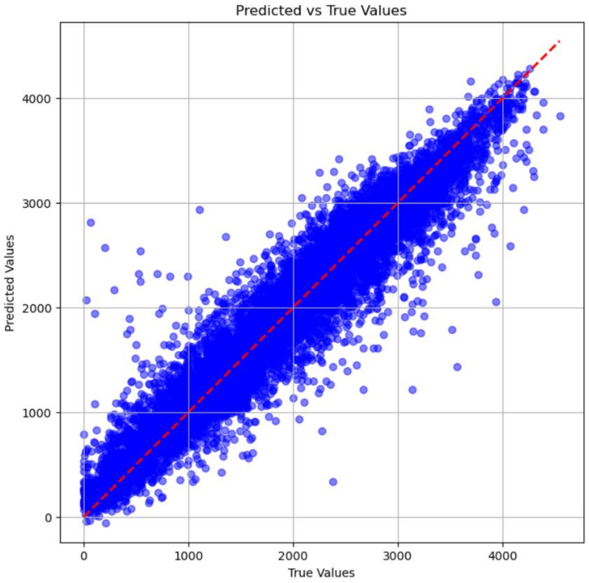
Methods:

- Calculates thickness by tracking ice flow
- Interpolates of thickness
- Uses physics of floating ice
- Estimates terrain under thick ice

The model outputs high-resolution gridded data (500 m)



Boosted decision tree




```
Final RMSE: 245.0379
Final R² Score: 0.9446
Final RMAE Score: 0.5562
```

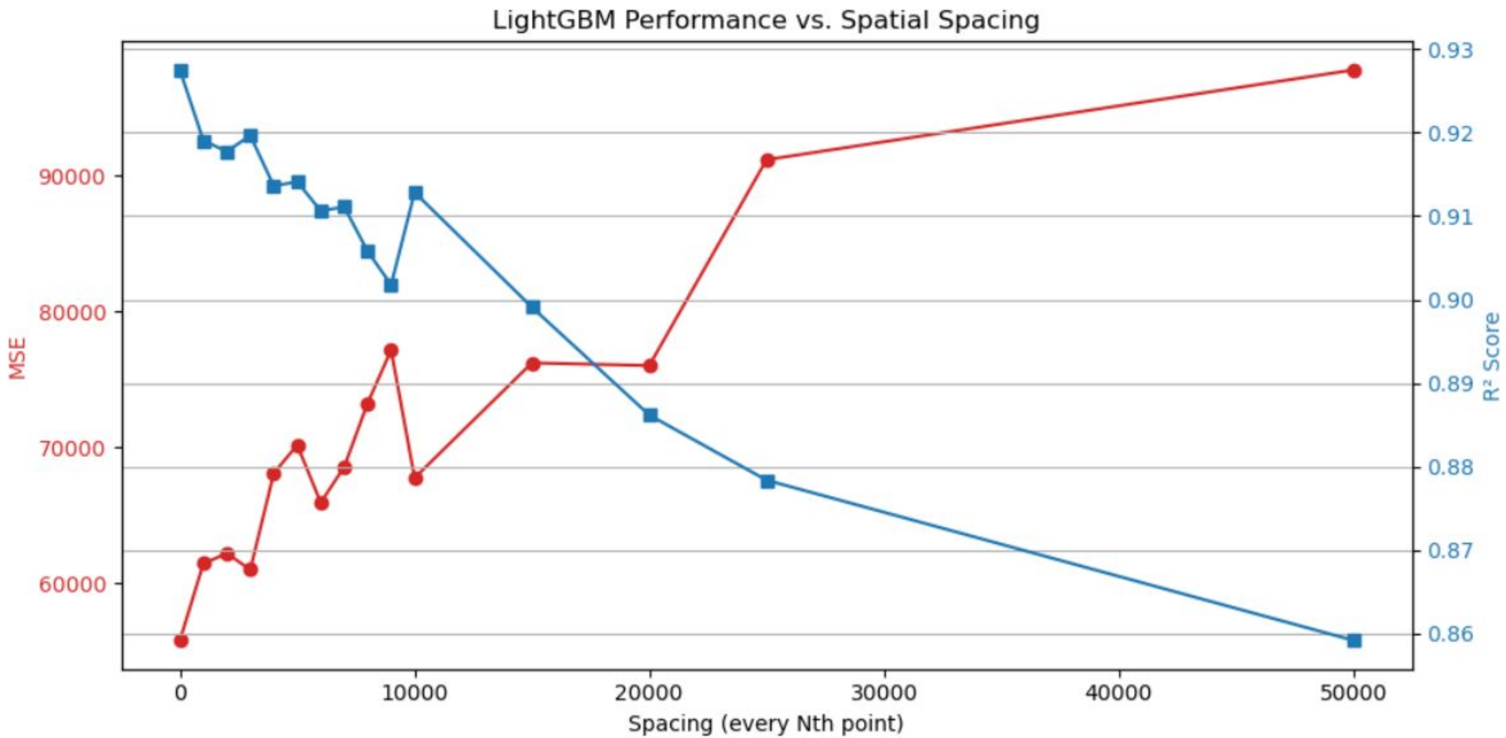
models: LightGBM

features: [East, North, s, z, temp, smb, vx, vy]

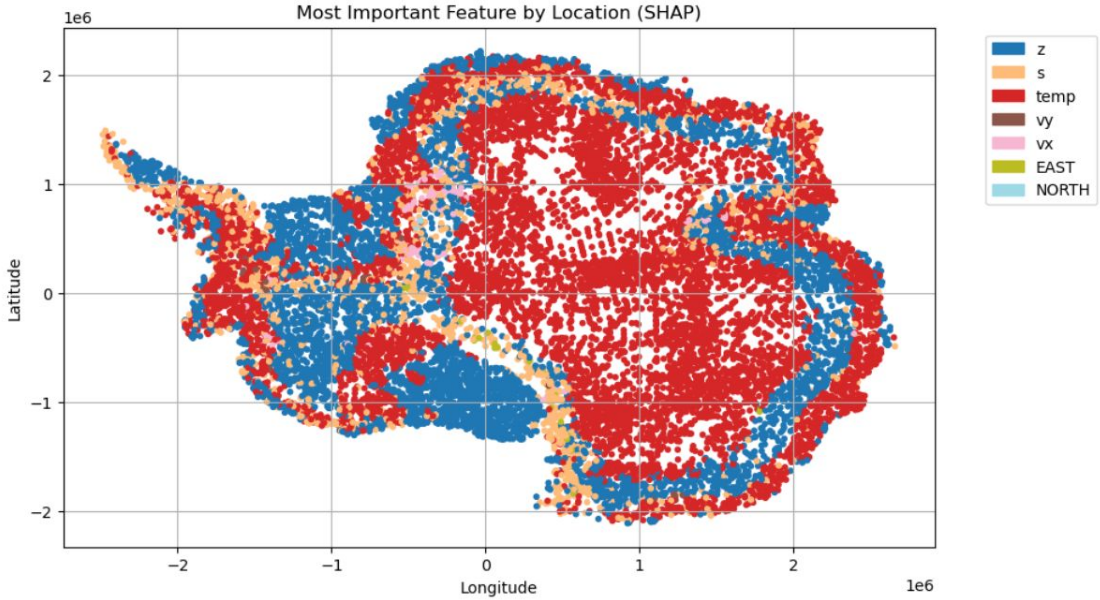
10 km spacing between points

Bayesian optimization

Does correlation matter?



Feature importance



SHAP value (impact on model output)

Understanding features and their importance could be useful for future models

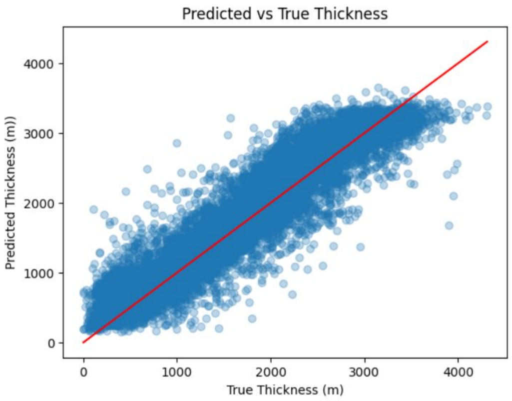
Hybrid CNN - FFNN model

- TensorFlow.keras, 3 CNN’s + tabular NN + gathering NN
- Filling from the left/right + Scaling of maps + tabular
- Validation set: Selected area, about 10-15 % of the tabular data
- Earlystopping: patience 3, Batchsize: 64, Adam optimizer
- Leaky_relu, PReLU
- Choice of Learning rate schedule: Cosine, reduce_on_plateu, exponential
- Choice of loss function: log_cosh, Huber (better for large/many outliers)

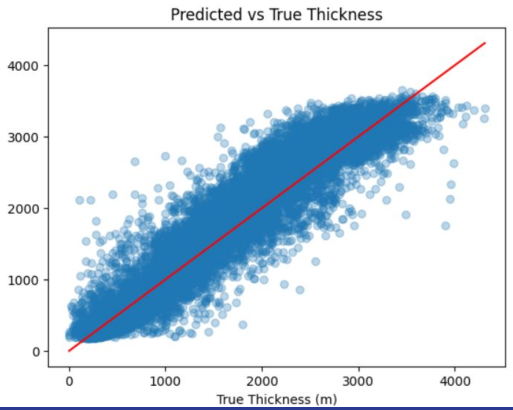
Hybrid CNN - FFNN model

- Patchsize: 14, 14, 14 (slope, elevation, VY)
- **2:** Huber(delta=0.15), LR schedule: Exp.: 5e-4, rate: 0.7, 10000 steps: MAE 248 m
- **3:** EAST/NORTH: MAE 243 m
- **1:** Exp. decay 0.9, Huber (delta=0.2): MAE 272 m
- Deeper NN's and CNN's, Wider layers, Dropout, regularization, activation function

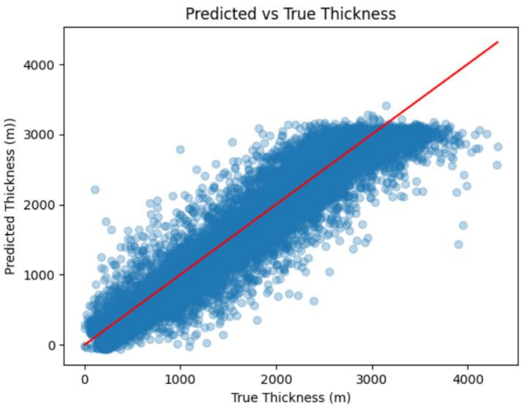
1



2



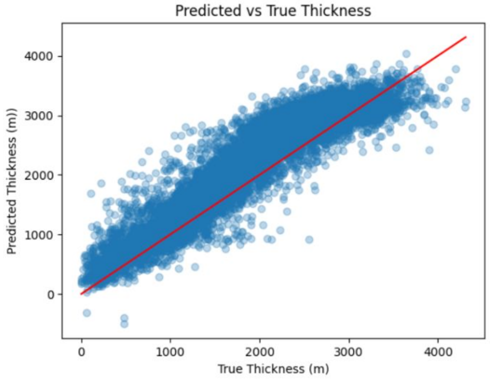
3



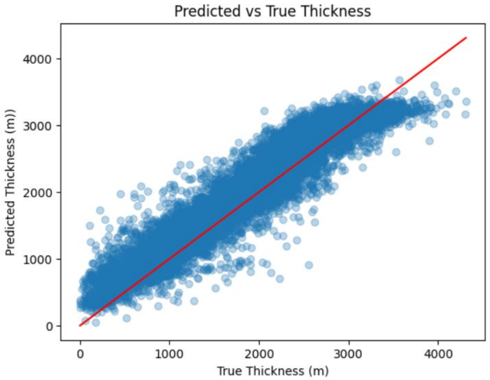
Hybrid CNN - FFNN model

- Training time: ~15 min. (no improvement with ~30 min.) (GPU speedup)
- Improvement of MAE: 400 m -> 236 m (training data ~200 m)
- Axial attention vs. transformer vs. pure CNN: No difference
- Log(y) -> worse predictions
- Patch size: 64, 64, 64: Overfitting
- MAE for maps : **VY**: 258 m, **temperature**: 298 m, **smb**: 236 m

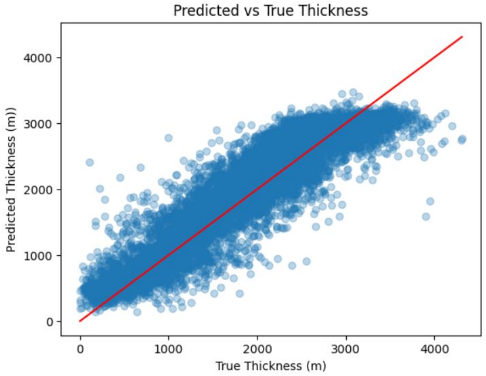
Temperature



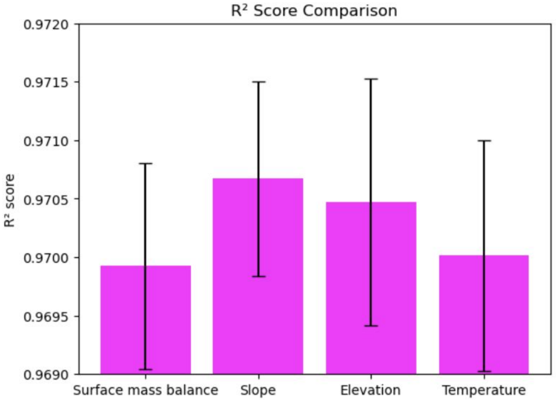
smb

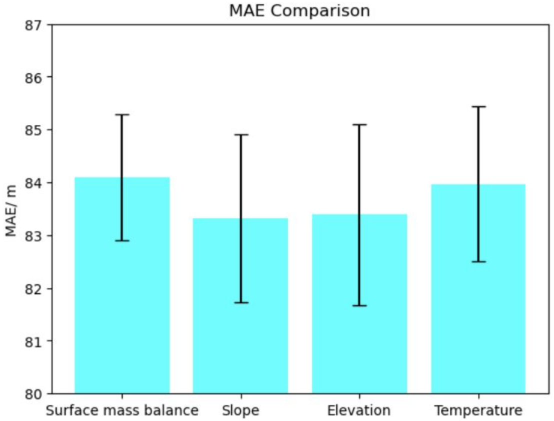


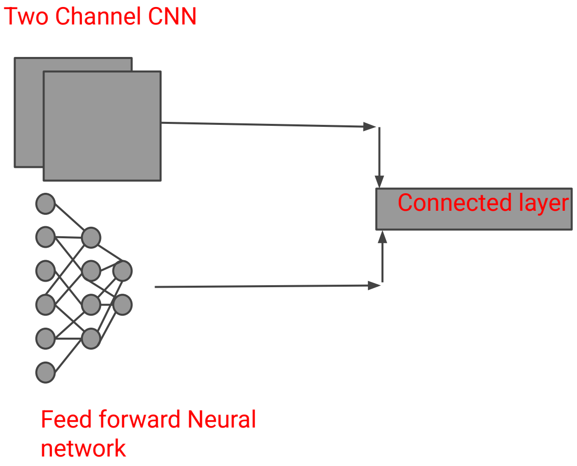
VY

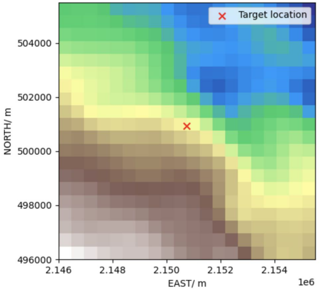


Hybrid Neural Network; FFNN+two channel CNN

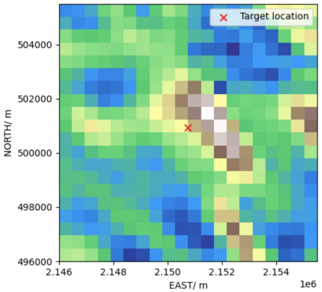








Elevation



Slope

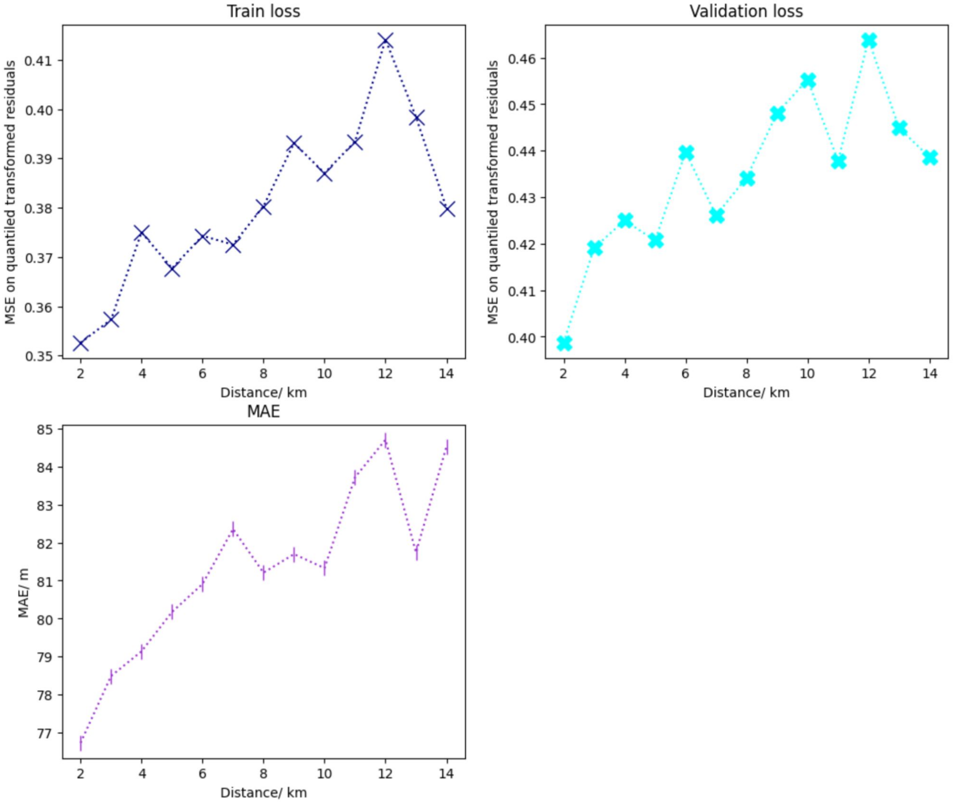
Keras model:

- Feed forward neural network: 6 layers
- CNN: 4 convolutional layers, batchnormalization, spatial dropout, maxpooling and global averaging.
- Fully connected: 6 layers, dropout
- Optimizer: Adam
- batch size: 68
- Dynamic learning rate, start: 5.52e-4

Image testing:

- 4 iterations
- Slope + elevation
- Upsampling, and projection
- Scalar features: vx, vy, smb, temp, north & east

Correlation, based on hybrid model, combining, FFNN and two channel CNN



Correlation

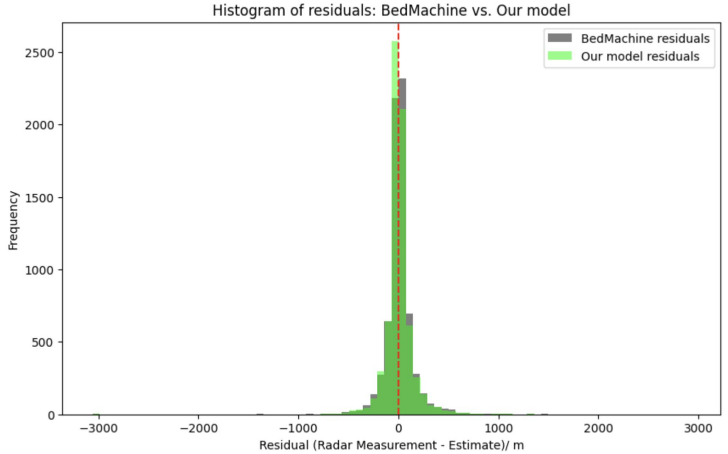
- Model memorizes instead of generalization due to spatial correlation

Optimization and final model results

- Bayesian optimization
  - 20 iterations: 7 random points and 13 iterations following.
  - learning rate, dropout, connected input layer, batch size.
- Final model, slightly better BedMachine.
  - Systematic errors

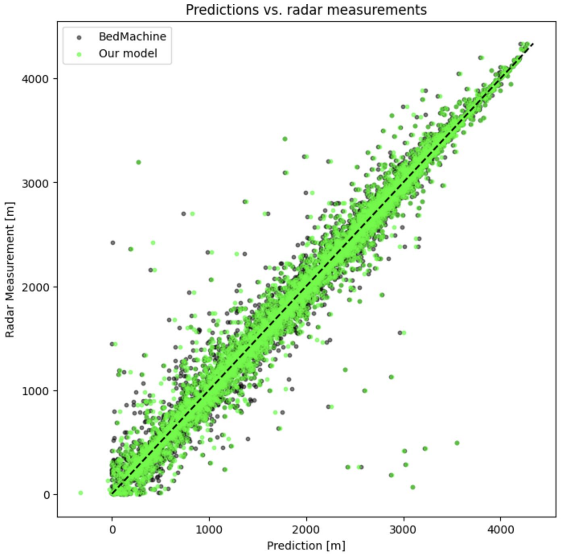
Model before optimization and not using the residuals:

- Best output from our model:
  - Relative MAE: 0.46
  - R^2: 0.93
  - MAE=182.09 m
- BedMachine:
  - Relative MAE: 0.31
  - R^2: 0.972
  - MAE=85.31 m

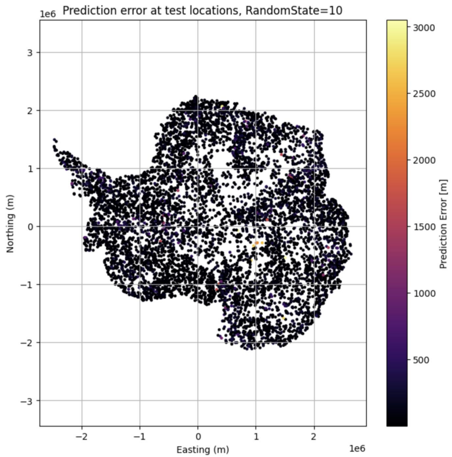


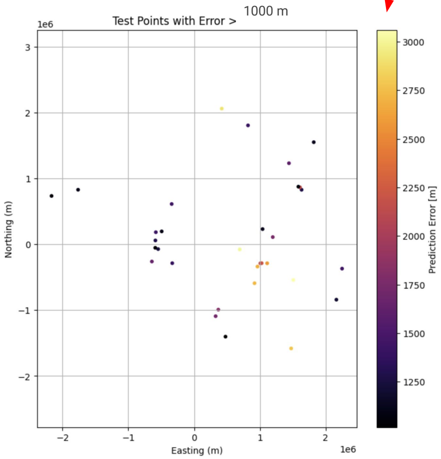
Good run	Our final model	BedMachine	Improvement
MAE/ m	85.908	91.804	6.41%
R^2	0.970	0.967	0.003
Relative MAE/ m	0.219	0.224	2.23%

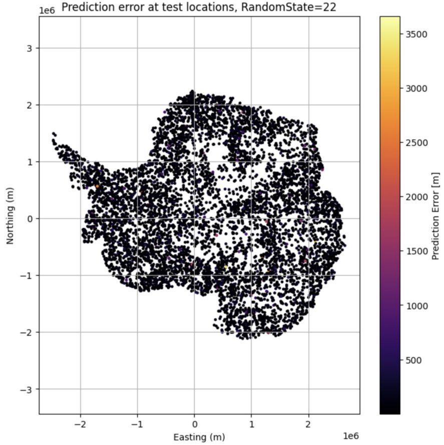
Worse run	Our final model	BedMachine	performance
MAE/ m	82.726	87.739	+5.71 %
R^2	0.968	0.966	+0.002
Relative MAE/ m	0.499	0.462	-8.01%

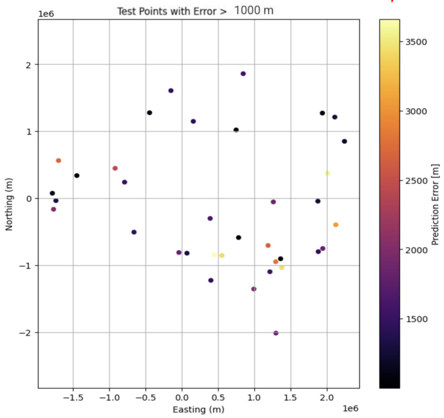


Good split vs worse







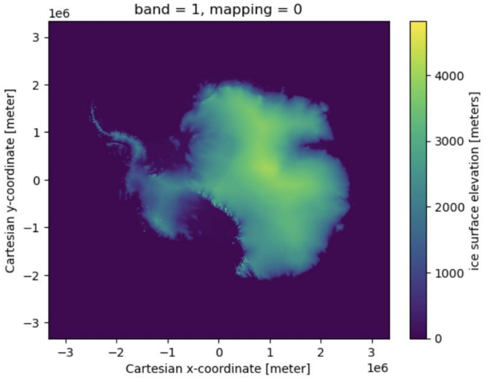


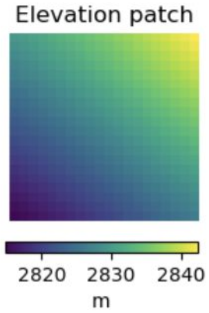
Hybrid Neural Network; FFNN+Parallel CNN’s

Model - Pytorch nn.Module CNN
Parallel CNNs - 2 conv layers
Three small MLP/FFNN

smb ——> CNN_smb + scalars ——┐
bm_z ——> CNN_z + scalars ————> Concatenate —> Fully connected layer —> Residual
Temp ——> CNN_temp + scalars —┘

Inputs:
Three raster images (Surface elevation + Surface Mass Balance + Temperature)
Scalars: *["NORTH","EAST","smb","v","s", “z”, ”temp"]*





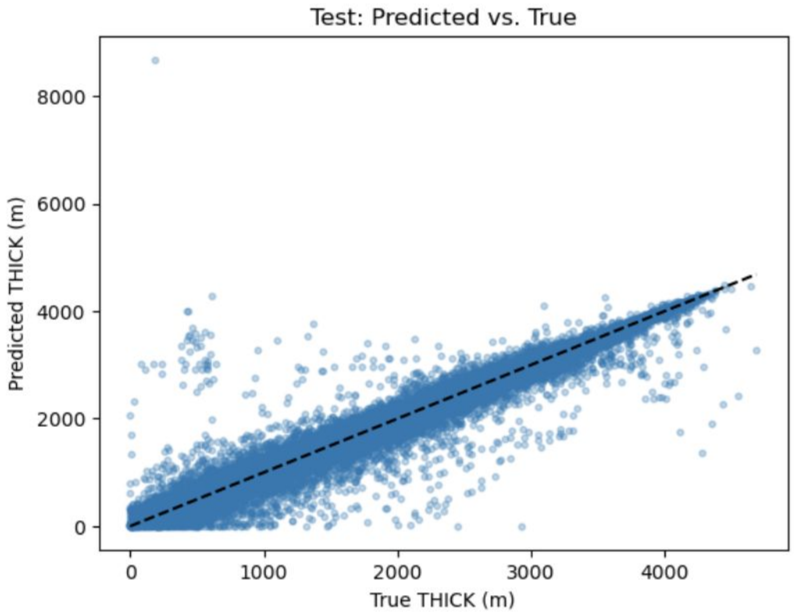
Target:
Residual in log-space

Optimizer & Loss:
SmoothL1 (Huber) with AdamW

Learning rate:
Warmup: Linear
Then: CosineAnnealingLR

Final model results on testset

Metric	Value	BedMachine	Improvement
R^2 (m)	0.977	0.977	0
MAE (m)	75.7 m	76.4 m	- 0.7 m
RMSE (m)	146.6 m	146.32 m	+ 0.28 m



Metrics shows that the errors in the BedMachine haven't improved.
BedMachine already provides a very accurate physics-based estimate, leaving little room for data-driven correction.

Testset on 10% points from data. Overall fit - most points around 1:1 line, but some extreme outliers.

Conclusion & outlook.

It is possible to improve BedMachine slightly in certain areas

- Should be investigated more in the future

Introduction of residuals → models as good or slightly better than BedMachine

  - Without residuals worse → Data limitations, few features.
  - Cleaning data

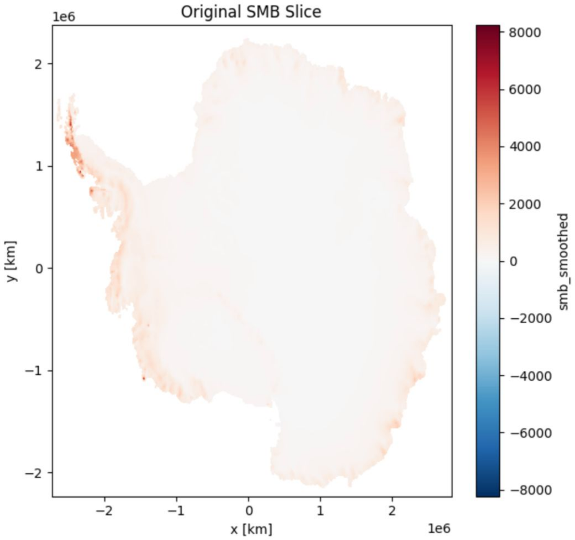
Influence of correlation.

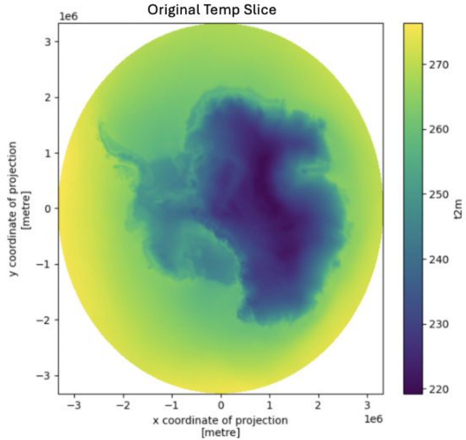
Appendix

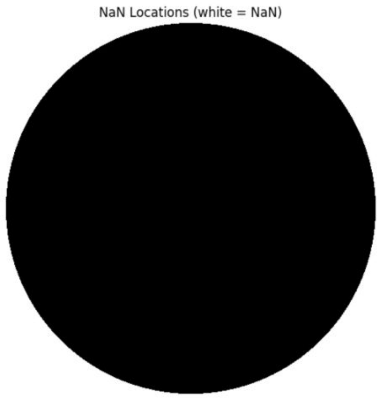
- All group members contributed equally.

Appendix (Feed forward neural network and two channel CNN)

- For image importance test, the resolution of the surface mass balance and temperature raster maps, was upsampled using lanczos (Matching the resolution of the Slope and elevation map, (500 m, 500 m) resolution, this was also used for re-projecting the temperature map, into the correct coordinate system.
    - The reason for this was mainly to get as large images (pixels x pixels) as possible, for more feature extraction for the CNN, as this was build quit deep.
- For the Surface mass balance and temperature raster maps, there were Nan values in the corners (i.e. in the ocean) and these where therefore filled with zeros (see the image to the right)

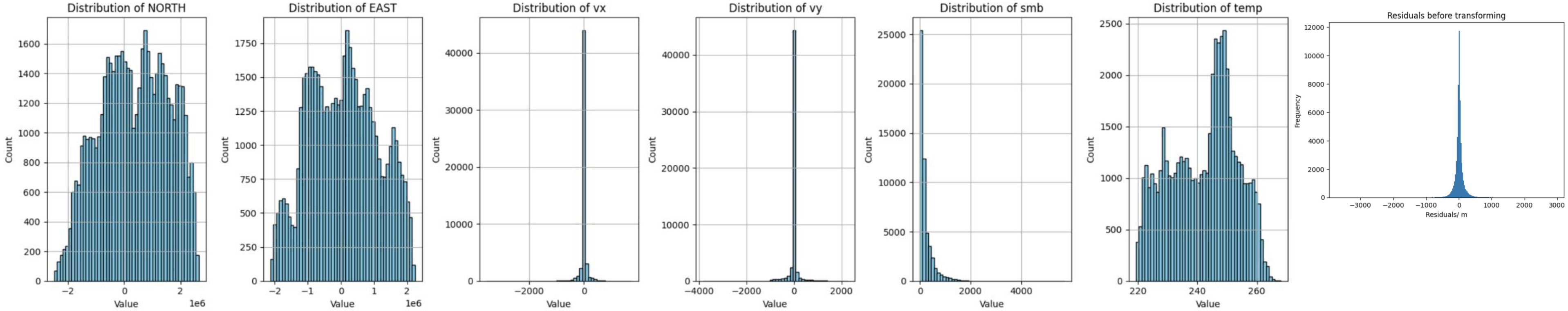




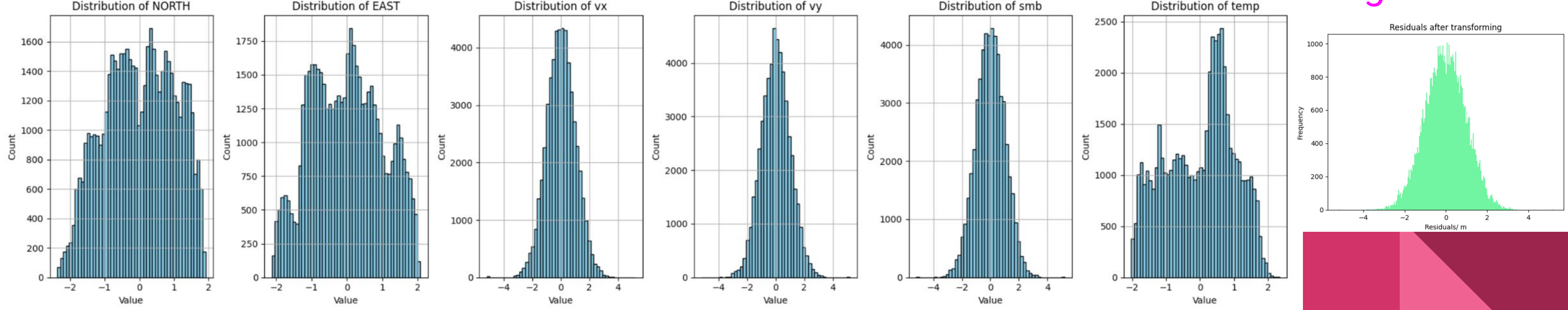


Preprocessing the data: Feed forward neural network and two channel CNN

Before   Target: Before



After   Target: after



North, East & Temperature: Standard Scaler. Vx, Vy, Smb & Target (Residuals): Quantile Transform (Normal distribution).

Appendix: Bayesian opt for FFNN+ two channel CNN

Tested:

- Learning rate, interval: 1e-4-3e-3
- Dropout for connected layer, interval: 0.05-0.3
- Spatial dropout for CNN, interval: 0.1-0.4
- Units in input layer, interval: 480-1024
- Batch size, interval: 32-128

Appendix (Feed forward neural network and two channel CNN)

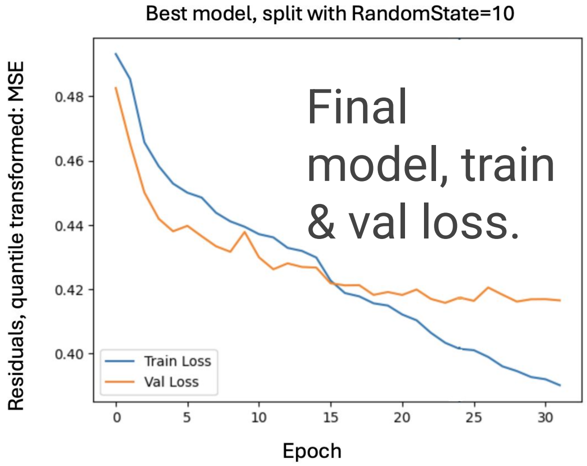
Concrete Architecture:

- Feed forward neural network:
  - Input layer(7)
  - 1. Layer: units: 128, activation function: ReLu
  - 2. Layer: units: 256, activation function: ReLu
  - 3. Layer: units: 128, activation function: ReLu
  - 4. Layer: units: 64, activation function: ReLu
  - Output layer: units: 32, activation function: ReLu
- CNN:
  - Input layer(20, 20, 2)
  - 1. Layer: units: 32, filter (3, 3), padding= same as input, l2 kernal regualization: 1e-4, activation= ReLu
    - Batchnormalization
    - Maxpooling(2, 2)
    - Spatial dropout: 0.385
  - 2. Layer: units: 64, filter (3, 3), padding=same as input, activation= ReLu
    - Batchnormalization
    - Maxpooling(2,2)
    - Spatial dropout: 0.385
  - 3. Layer: 128, filter (3, 3), padding= same as input, activation= ReLu
    - Batchnormalization
    - Maxpooling(2,2)
    - Spatial dropout: 0.385
  - 4. Layer: units: 256, filter (3,3), padding= same as input, activation= ReLu
    - Batchnormalization
    - Maxpooling(2,2)
  - output layer, Global averaging, units: 256, activation= ReLu
- Connected:
  - Input, connected scalar output and CNN output.
  - 1. Layer: units: 806, activation= ReLu
    - dropout: 0.233
  - 2. Layer: units: 256 , activation=ReLu
  - 3. Layer: units: 128, activation=ReLu
  - 4. Layer: units: 64 , activation=ReLu
  - 5. Layer: units: 32 , activation=ReLu
  - Output layer: units: 1 , activation=linear

AN IMAGE IS ALSO SHOWN ON THE NEXT SLIDE

Technical details:

- Learning rate: 5.52e-4, but dynamic, reduces when plateau is reached, patience=3
- Loss: Huber loss, delta=100
  - MSE switches to MAE when/ if criterion is meet.
- Patience=8
- Max epochs= 50
- Batch size= 68
- Optimizer=Adam
- Final testing: Train (75%), Validation (15%) & Test (10%)



Appendix (Feed forward neural network and two channel CNN)

- Image showing the architecture
  - Total params: 1,016,839 (3.88 MB)
  - Trainable params: 1,015,879 (3.88 MB)
  - Non-trainable params: 960 (3.75 KB)
- All other details are on former slide

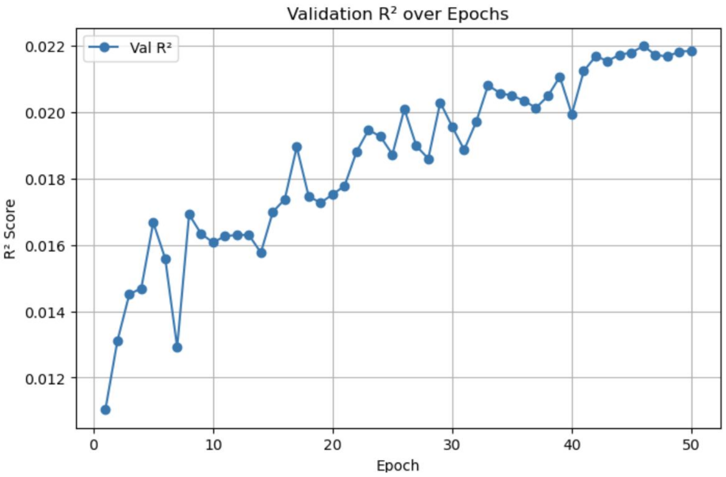
Layer (type)	Output Shape	Param #	Connected to
image_input (InputLayer)	(None, 20, 20, 2)	0	-
conv2d_4 (Conv2D)	(None, 20, 20, 32)	608	image_input[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 20, 20, 32)	128	conv2d_4[0][0]
max_pooling2d_4 (MaxPooling2D)	(None, 10, 10, 32)	0	batch_normalizat…
spatial_dropout2d_3 (SpatialDropout2D)	(None, 10, 10, 32)	0	max_pooling2d_4[…
conv2d_5 (Conv2D)	(None, 10, 10, 64)	18,496	spatial_dropout2…
batch_normalizatio… (BatchNormalizatio…	(None, 10, 10, 64)	256	conv2d_5[0][0]
max_pooling2d_5 (MaxPooling2D)	(None, 5, 5, 64)	0	batch_normalizat…
spatial_dropout2d_4 (SpatialDropout2D)	(None, 5, 5, 64)	0	max_pooling2d_5[…
conv2d_6 (Conv2D)	(None, 5, 5, 128)	73,856	spatial_dropout2…
batch_normalizatio… (BatchNormalizatio…	(None, 5, 5, 128)	512	conv2d_6[0][0]
scalar_input (InputLayer)	(None, 6)	0	-
max_pooling2d_6 (MaxPooling2D)	(None, 2, 2, 128)	0	batch_normalizat…
dense_13 (Dense)	(None, 128)	896	scalar_input[0][…
spatial_dropout2d_5 (SpatialDropout2D)	(None, 2, 2, 128)	0	max_pooling2d_6[…
dense_14 (Dense)	(None, 256)	33,024	dense_13[0][0]
conv2d_7 (Conv2D)	(None, 2, 2, 256)	295,168	spatial_dropout2…
dense_15 (Dense)	(None, 128)	32,896	dense_14[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 2, 2, 256)	1,024	conv2d_7[0][0]
dense_16 (Dense)	(None, 64)	8,256	dense_15[0][0]
max_pooling2d_7 (MaxPooling2D)	(None, 1, 1, 256)	0	batch_normalizat…
dense_17 (Dense)	(None, 32)	2,080	dense_16[0][0]
global_average_poo… (GlobalAveragePool…	(None, 256)	0	max_pooling2d_7[…
dense_18 (Dense)	(None, 32)	1,056	dense_17[0][0]
dense_19 (Dense)	(None, 256)	65,792	global_average_p…
concatenate_1 (Concatenate)	(None, 288)	0	dense_18[0][0], dense_19[0][0]
dense_20 (Dense)	(None, 806)	232,934	concatenate_1[0]…
dropout_1 (Dropout)	(None, 806)	0	dense_20[0][0]
dense_21 (Dense)	(None, 256)	206,592	dropout_1[0][0]
dense_22 (Dense)	(None, 128)	32,896	dense_21[0][0]
dense_23 (Dense)	(None, 64)	8,256	dense_22[0][0]
dense_24 (Dense)	(None, 32)	2,080	dense_23[0][0]
dense_25 (Dense)	(None, 1)	33	dense_24[0][0]

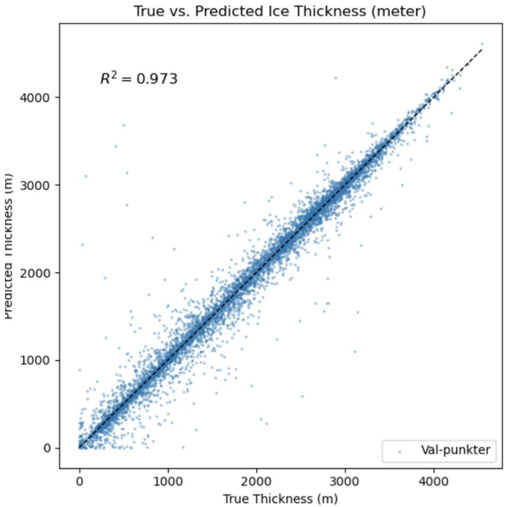
Total params: 1,016,839 (3.88 MB)

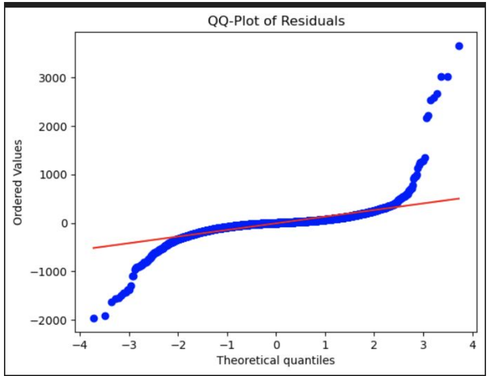
Trainable params: 1,015,879 (3.88 MB)

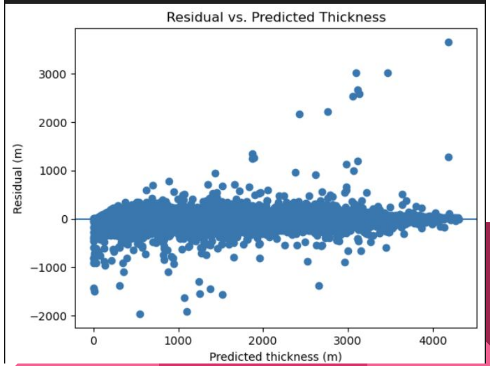
Non-trainable params: 960 (3.75 KB)

Appendix FFNNs + Parallel CNNs’ train and val curves









*In log-space

Appendix Feed forward neural network and parallel CNNs

Architecture
Input:
3 image rasters; Surface elevation, SMB, Temperatur
3 scalar input vectors: scA, scB, scC

Three CNN branches (one for each input patch):
Each CNN branch:
Conv2D (1 → 16 filters, 3×3, ReLU)
MaxPooling (2×2) → size 20→10
Conv2D (16 → 24 filters, 3×3, ReLU)
MaxPooling (2×2) → size 10→5
Flatten → output size: 600 features

Scalar branches:
scA → Linear(n_A → 32) → ReLU
scB → Linear(n_B → 32) → ReLU
scC → Linear(n_C → 32) → ReLU

Final MLP:
Concatenate all features: 600 (surface) + 600 (SMB) + 600 (temp)
+ 32 (scA) + 32 (scB) + 32 (scC)= 1896
Dense: Linear(1896 → 128) → ReLU
Dropout (p = 0.3)
Output: Linear(128 → 1), activation = linear (regression)

Technical:
Loss function: Smooth L1 (Huber) Loss, β = 0.1
Optimizer: AdamW
Learning rate:
- Warmup: Linear from 1e-5 to 1e-4 over 5 epochs
- Then: CosineAnnealingLR (T_max = 95, min_lr = 1e-6)

Weight decay: 1e-5
Gradient scaling: Mixed precision training with GradScaler
Gradient clipping: max_norm = 1.0

Train settings:
Max epochs: 100 (5 warmup + 95 cosine)
Patience = 10
Batch size: 64
Metrics tracked: MSE, RMSE, R^2 (validation)
Model checkpointing: Best model saved to resid_cnn_best.pth

Final testing: Train (80%), Validation (10%) & Test (10%)

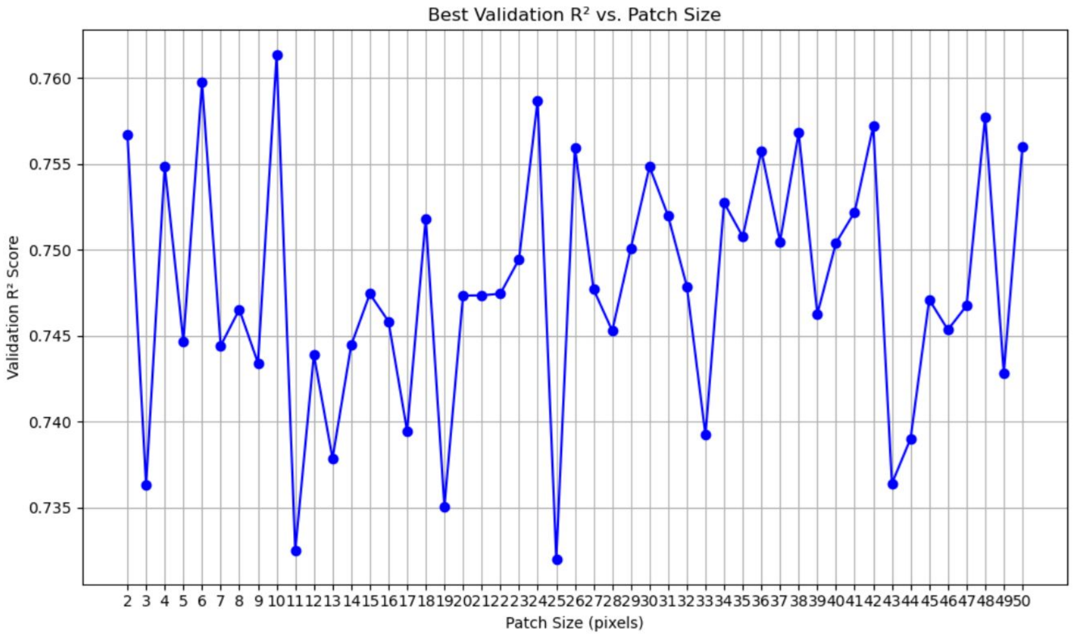
Appendix (patch size effect on CNN)

simple CNN model

appears to have no effect

Though may be a map issue

Would have expected larger patches to do better, especially for maps past 20 pixel due to overlap of the maps



Appendix CNN-FFNN

3 CNN layers:

```
    x = Conv2D(32, (3, 3), padding='same',
kernel_regularizer=regularizers.l2(1e-4))(inp)
    x = PReLU()(x)
    x = BatchNormalization()(x)
    x = Conv2D(64, (3, 3), padding='same',
kernel_regularizer=regularizers.l2(1e-4))(x)
    x = PReLU()(x)
    x = BatchNormalization()(x)
    x = Conv2D(128, (3, 3), padding='same',
kernel_regularizer=regularizers.l2(1e-3))(x)
    x = PReLU()(x)
    x = BatchNormalization()(x)
    x = Dropout(0.1)(x)
    x = Conv2D(256, (3, 3), padding='same',
kernel_regularizer=regularizers.l2(1e-3))(x)
    x = PReLU()(x)
    x = BatchNormalization()(x)
    return inp, x
```

Merging:

```
merged = Concatenate(axis=-1)([branch1, branch2,
branch3])
cnn_features =
tf.keras.layers.GlobalMaxPooling2D()(merged)
```

Tabular FFNN:

```
input_tab = Input(shape=(N_TAB_FEATURES,),
name='tabular_input')
    y =
layers.Dense(256,kernel_regularizer=regularizer
s.l2(1e-3))(input_tab)
    y = PReLU()(y)
    y = Dropout(0.1)(y)
    y = BatchNormalization()(y)
    y =
layers.Dense(128,kernel_regularizer=regularizer
s.l2(1e-3))(y)
    y = PReLU()(y)
    y = BatchNormalization()(y)
    y =
layers.Dense(64,kernel_regularizer=regularizers
.l2(1e-4))(y)
    y = PReLU()(y)
    y = BatchNormalization()(y)
    y =
layers.Dense(32,kernel_regularizer=regularizers
.l2(1e-4))(y)
    y = PReLU()(y)
    y = BatchNormalization()(y)
```

Combining to hybrid model:

```
combined =
layers.concatenate([cnn_features,
y])
    z =
layers.Dense(256,kernel_regularizer=
regularizers.l2(1e-3))(combined)
    z = PReLU()(z)
    z = Dropout(0.1)(z)
    z = BatchNormalization()(z)
    z = layers.Dense(128,
kernel_regularizer=regularizers.l2(1
e-4))(z)
    z = PReLU()(z)
    z = BatchNormalization()(z)
    z =
layers.Dense(64,kernel_regularizer=r
egularizers.l2(1e-4))(z)
    z = PReLU()(z)
    z = BatchNormalization()(z)
    z =
layers.Dense(32,kernel_regularizer=r
egularizers.l2(1e-4))(z)
    z = PReLU()(z)
    z = BatchNormalization()(z)
    z = layers.Dense(1)(z)  #Final
output
```

Appendix CNN-FFNN

- Technical details for the best model:
  - Huber loss: delta = 0.1
  - Patience: 3
  - Optimizer = Adam
  - Learning rate schedule: Exp., rate 0.6, 1500 steps, initial 5e-4
  - Maps: Temperature, Slope, Elevation
  - Patch sizes: 14, 14, 5
  - Batch size: 64
  - MAE: 236
  - Validation set: ~10000 points in the corner of the ice area
  - Max nr. epochs 50