DATA CHALLENGE SUMMARY AND RESULTS

Pierre Auclair, Florian Bury, Borja S. González, Philipp Munkes, Giles Strong

2nd MODE Workshop on Differentiable Programming

OAC, Greece - 15/09/22

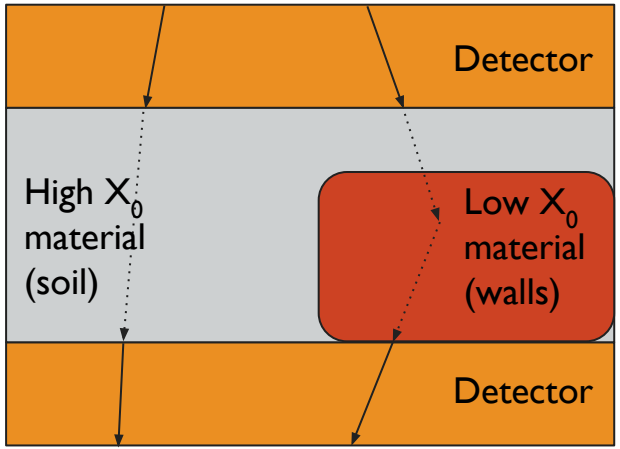
INTRODUCTION TO THE TASK

- Britain's oldest city, Colchester, was founded by the Romans around 40AD as a barracks on the site of a Celtic stronghold
- Throughout the 2nd & 3rd centuries, the city expanded, eventually becoming a *colonia* -- an extension of the city of Rome
- Roman landmarks remain to this day, with more discoveries still being made
- Whilst performing some construction, workers uncovered indications of ruined walls in a site that was previously thought to be empty
- You have been brought in to scan the site using *muon tomography* and to map out the locations of the walls to help aid the archeologists
  - You have been provided with a set of simulated data on which to develop a suitable inference algorithm
- Task dates: 01/08 - 04/09

Example of Roman walls in Colchester, credits Maria, CC-BY-SA 3.0

TOMOGRAPHY VIA MULTIPLE SCATTERING

- Muon tomography allows us to infer the material composition of unknown volumes of space
- Cosmic muons are scattered by materials in the volume according to their radiation-length (X_0 [m]) of the material
  - Radiation-length = average distance between scatterings
- By using detectors, we can measure muons above and below volume
  - The changes in trajectory provide information on material composition



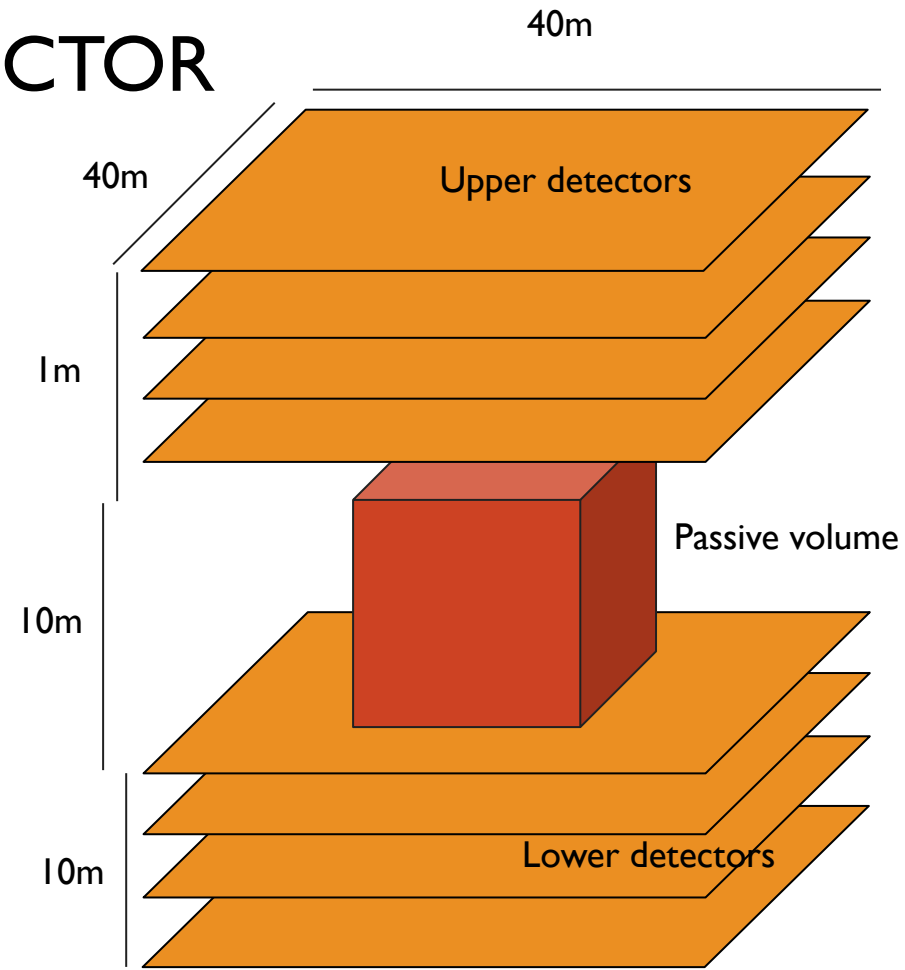
High X_0 = low scattering

Low X_0 = high scattering

X_0 = average distance between scatterings

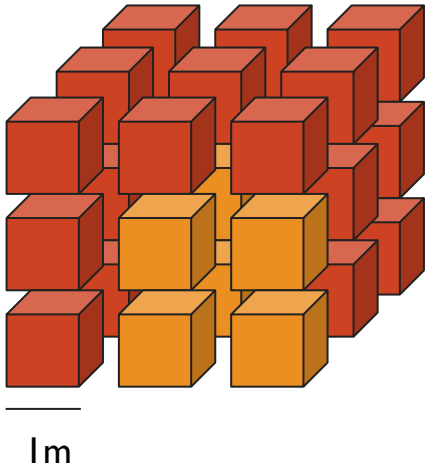
DETECTOR

- The detector setup used here is quite unrealistic
  - It assumes that a detector panel may be placed underground, directly under the passive volume
  - The simulated detector is very large (40x40m), but this is for simulation convenience; a smaller detector could be used and placed in several spots to create a combined scan
- The detector consists of two layers (1m height), placed above and below the volume
  - Each layer contains 4 equally spaced panels
  - Each panel records muon positions with an xy resolution of 0.1mm



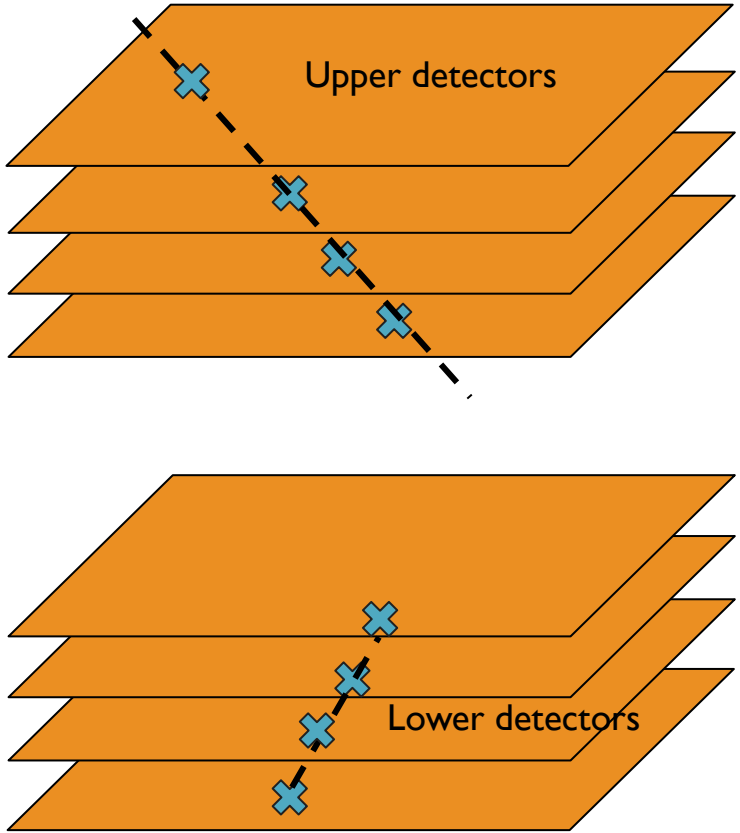
PASSIVE VOLUME

- The passive volume is a 10x10x10m cube
  - It is subdivided into 1000 voxels, each 1x1x1m in size
- Each voxel can either be soil (X0=~0.26m) or wall (X0=~0.08m)
  - The amount of muon scattering depends on the voxel X0, and scales as sqrt(distance/X0)/momentum
  - Muon momentum will always be 1GeV, but the distance depends on the incoming angle



TRACK FITTING

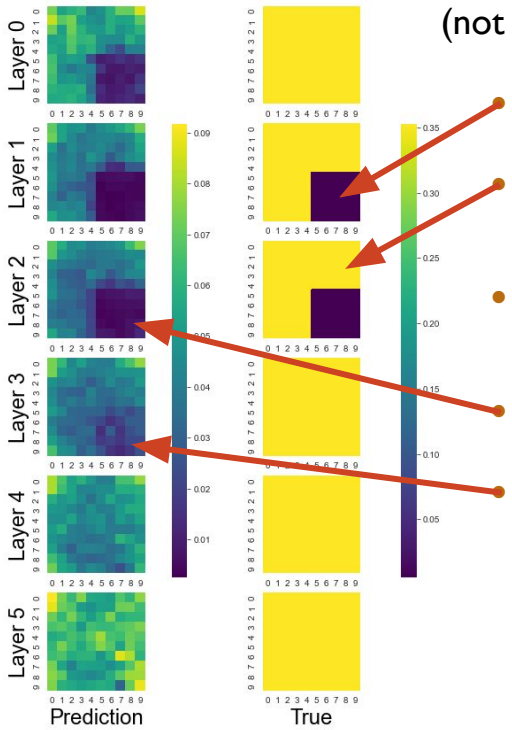
- Fit linear trajectories to the hits
- Can then compute variables about the muon scattering, e.g.:
  - Incoming and outgoing angles
  - Changes in trajectory



PoCA INFERENCE

- PoCA method assigns the entirety of the muon scattering to a single voxel
  - The voxel is chosen by extrapolating trajectories inside the passive volume to find the Point of Closest Approach
  - X0 predicted by inverting the scattering model to get X0 as a function of total scattering, and then averaging over many muons
  - We made a slight modification: X0 predictions are applied to every voxel, but in an average weighted by the probability of the scattering having occurred there
    - Computed using the uncertainty on the PoCA location
    - This provides a dense set of voxelwise X0 predictions
- PoCA is a very standard algorithm in tomography, but produces biased and blurred results

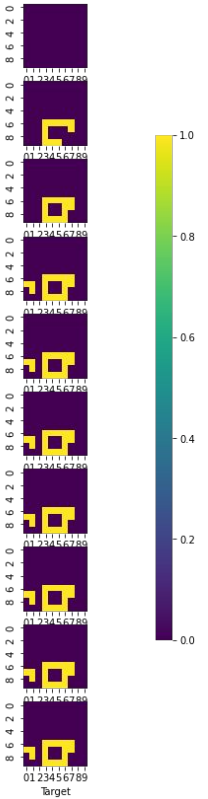
PoCA Example
(not related to out dataset)



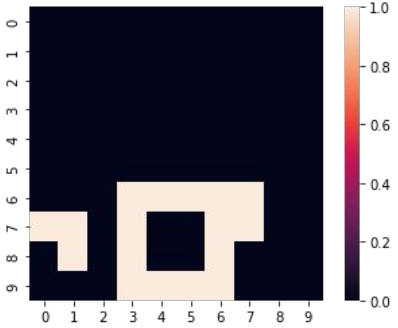
- Block of lead (X_0=0.005612m)
- Surrounded by beryllium (X_0=0.3528m)
- Predictions highly biased to underestimate X_0
- Lead block clearly visible
- but high z uncertainty in scatter location causes ‘ghosting’ above and below

TARGET

- The passive volumes are randomly generated to simulate stone walls buried underground
  - Each volume contains at least one wall, surrounded by soil
- All walls in a given sample begin on the same z position, but can vary in height
  - The starting z position of the walls can vary between samples



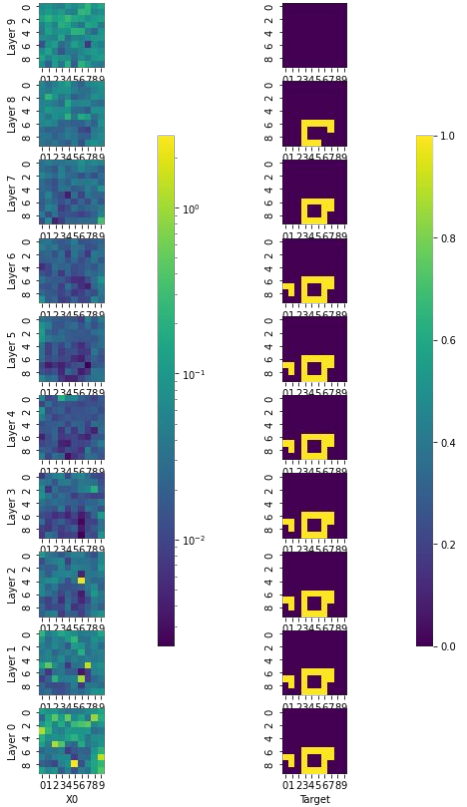
XY slices in layers of z



Top-down image

DATASET

- A sample is created by scanning a newly generated passive volume:
  - 10,000 muons are recorded
    - Incoming angle and initial xy position can vary
    - Momentum is fixed at 1GeV
  - The scanning process uses the PoCA inference method described on slide 9
  - This results in a biased PoCA image of voxelwise X0 predictions (float32)
  - The target is a map of the wall voxels (int, 0 = soil, 1 = wall)
- Approximately 100k labelled samples are provided along with 30k unlabelled testing samples
  - Your task is to provide predictions on the unlabelled sample



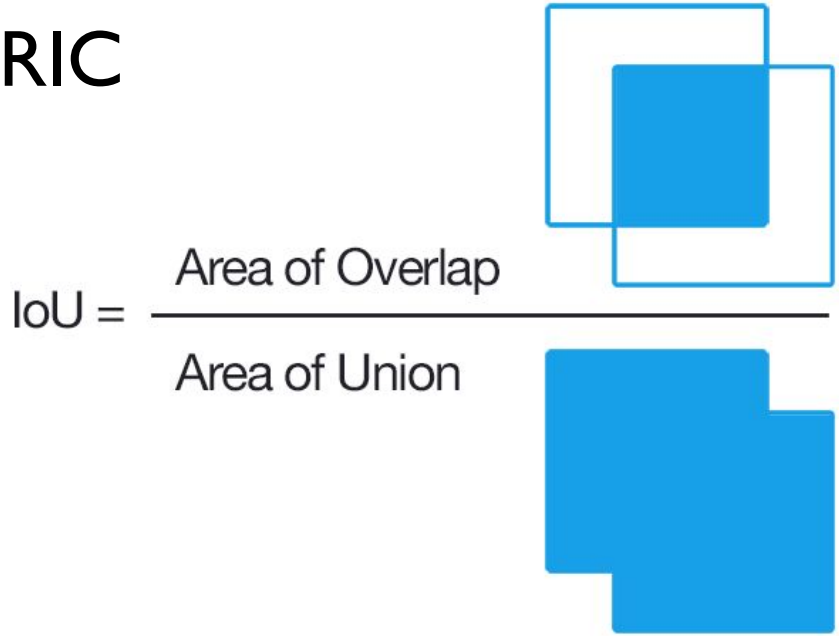
Example sample:

Left = voxelwise X0 predictions

Right = map of voxels which contain wall

METRIC

- Both predictions and targets will be a voxelwise 0 or 1
- Performance evaluated using the "intersection-over-union" (IOU) metric
  - Intersection is number of wall voxels correctly predicted to be wall
  - Union is the sum of the number of voxels that are either predicted to be wall or are actually wall
- Behaviour:
  - Predicting all walls gets high intersection but also high union = low IOU
  - Predicting no walls gets zero intersection = zero IOU
  - Accurate wall predictions get high intersection and low union = high IOU
- IOU is between 0 and 1, and higher values mean better performance
- Can be used as a loss "Soft IOU": replace predictions with raw sigmoid output



$$\mathcal{L} = \frac{y \cdot \hat{y}}{\sum [y + \hat{y}] - (y \cdot \hat{y})}$$

MODE Workshop
Data Challenge

Author: Borja S. González (LIP/IST)
Second MODE Workshop on Differentiable Programming for Experiment Design
September 2022

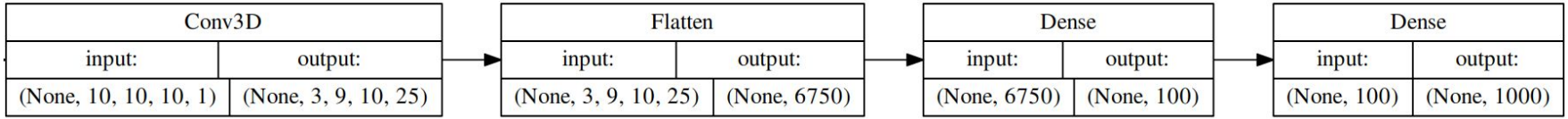
Overview

Approach

- **Main idea:** convolve over the 3d data in order to extract information learnable by a dense layer.
- **Custom simple 3d CNN**.
- **Input:** Raw data.
- **Output:** Sigmoid activation function.
- **Optimiser:** Adam.
- **Loss function:** Mean squared error.
- **Very simple architecture** due to very limited computing resources.

Resources and software used

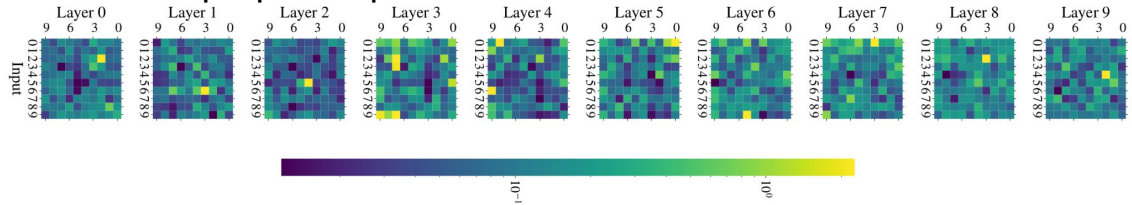
- Keras + Python framework.
- No GPU available.
- Although using a very simple CNN, it took between 3 and 10 hours (depending on the complexity) training the model using 80k training samples (20k for validation).
- Optimised NN configuration using a subset of 20k samples in order to save time.
- No data augmentation due to limited resources.



Code available at: https://github.com/borja-sg/mode_diffprog_22_challenge_borja_solution

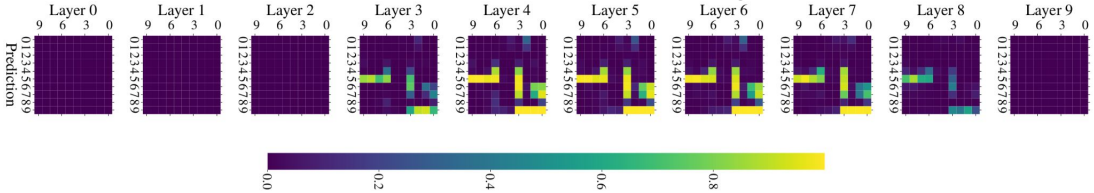
Optimisation of the NN output and final result

- Step 1: Read and prepare input for the neural network.



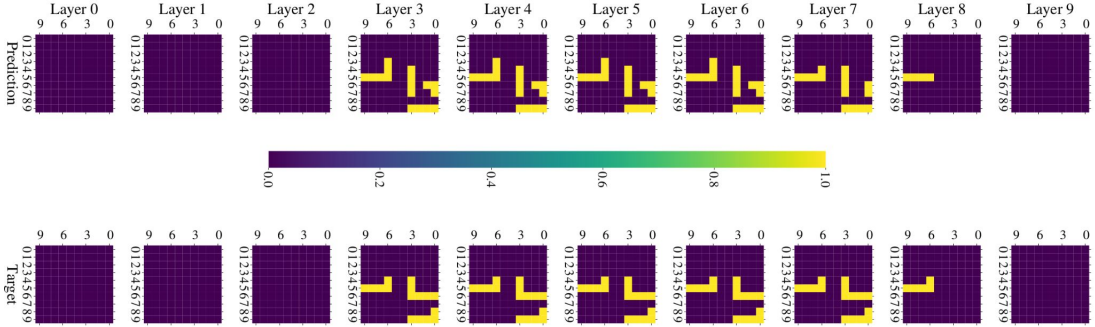
Input

- Step 2: Get a value from 0 to 1 for each voxel using the 3d CNN.



CNN prediction

- Step 3: Use optimised thresholds of each layer to determine whether it is soil or wall. The thresholds were computed by maximising the IOU of the training data set.



Final prediction

Target

Conclusions

- The CNN performance is highly dependent on the filter size.
- Architectures based on 2d CNNs performed worse.
- I tried a couple of normalisation approaches, but performance was lower after a few epochs.
- The model did not improve much after increasing its complexity and training set size.
- The optimisation of the network was limited by the long time it takes to train it without a GPU (even taking only a portion of the training data).

Result:

	Name	ID	Public IOU	Private IOU
0	BORJA_GONZALEZ	01	0.662763	0.665857

Philipp Munkes

Mode22 diffprog challenge

Short Summary

- I wanted to use this challenge as a way to improve my knowledge about Deep Learning
- I used Pytorch and started experimenting with various model designs
- I noticed that all of my (simple) models ran into an accuracy ceiling in the high 50% and low 60% region
- A colleague made the suggestion of voting on individual voxels with the output of several different models, which I implemented

Code repo: https://github.com/PMunkes/mode_diffprog_22_challenge_Philipp_solution

Overview

Approach

- Treat input data as a 1000-element vector
- Use 5 differently structured models
- Train models to produce output vectors similar to reference
- Enhance the training data set via rotations and flips

Resources

- 1xRX 6700 XT for training
- 1xR7 5800H for development
- Model Parameter counts between 2M and 4M
- Trained for 2000 Epochs.
- Pytorch
- MSE as loss function

Design of the models

- Different structures to ensure that the models train on different features
- Trained using MSE, as I could not get a SoftIOU loss-function to train the network

Model	Lin. layers	Non-lin layers	Structure
1	(1000,1280)|(1280,1000)	Tanh ReLu	LTLR
2	(1000,1000)|(1000,1000)	Tanh ReLU	LTLR
3	(1000,1000)|(1000,1000)	Tanh 2x ReLU	TLRLR
4	(1000,1100)|(1100,1000)| (1000,1000)|(1000,1000)	Tanh 3xReLU	LTLRLRLR
5	(1000,1000)|(1000,1000)| (1000,1000)|(1000,1000)	2x Tanh 2x ReLU	LTLRLTLR

Performance limitations

- Non-ideal loss function
- Limited number of models in the final vote
- Limited model complexity
- Neural Networks are potentially a sub-optimal solution

⇒Inexperience and time/processing power constraints

Public/private IOU: 0.692/0.692

MODE Data Challenge

Pierre AUCLAIR September 7, 2022

Cosmology, Universe and Relativity at Louvain (CURL)
Institute of Mathematics and Physics
Louvain University, Louvain-la-Neuve, Belgium

Approach

The problem

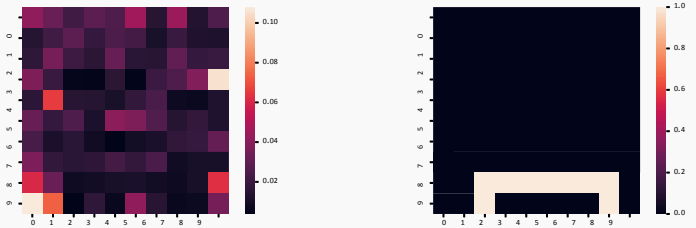
Your task is to take 3D predictions for the radiation-length of cosmic muons in voxelised volumes, and convert them to a 3D map of the locations of stone walls buried underground: a voxelwise classification of the voxel material, 0 = soil, 1 = wall.

First remarks:

- We expect that the voxels with walls should have a lower X_0 than the voxels with soil
- We have a lot of training data $\approx 100K$ samples
- Strong similarities with the problem of blind[a] image denoising

$=\Rightarrow$ A well-known strategy for $2D$ images is to use a Denoising Convolutional Neural Network (DnCNN)

[a] Unknown noise



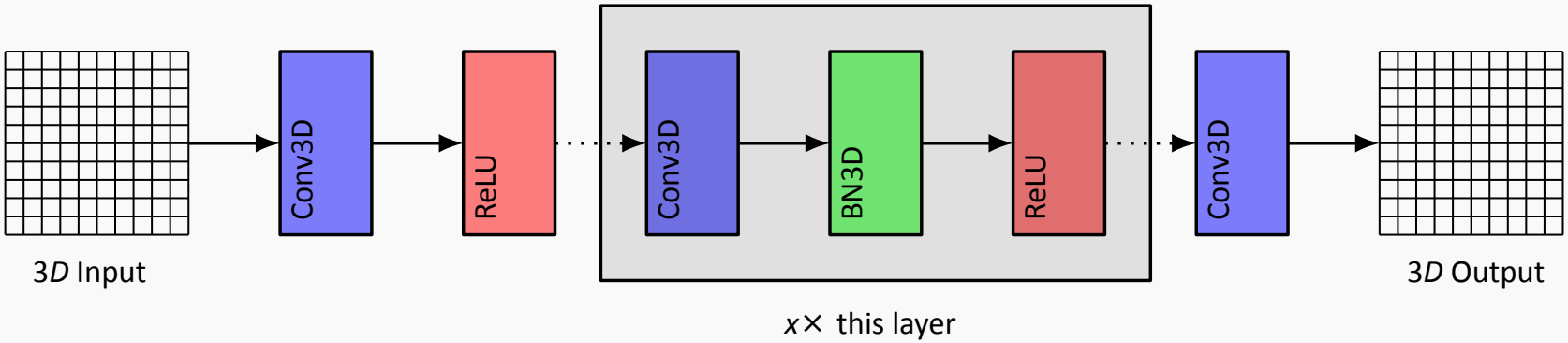
(a) Input **(b)** Target

Figure 1: Slice across one of the given test samples

Architecture of the DnCNN

Implemented using the PyTorch module

- Conv3D: applies a 3D convolution over an input signal
  Parameters: kernel size $k = 3$ or 5, number of features $f \lesssim 10$
- BN: Batch normalization
- ReLU: Applies the rectified linear unit function element-wise
- Tested for $x \lesssim 10$ layers



Training

- Loss function: SoftIOU
  - IOU: Intersection over union is a measure of the similarities between two sets of binaries
  - SoftIOU: variation of IOU but for sets of numbers $\in [0, 1]$

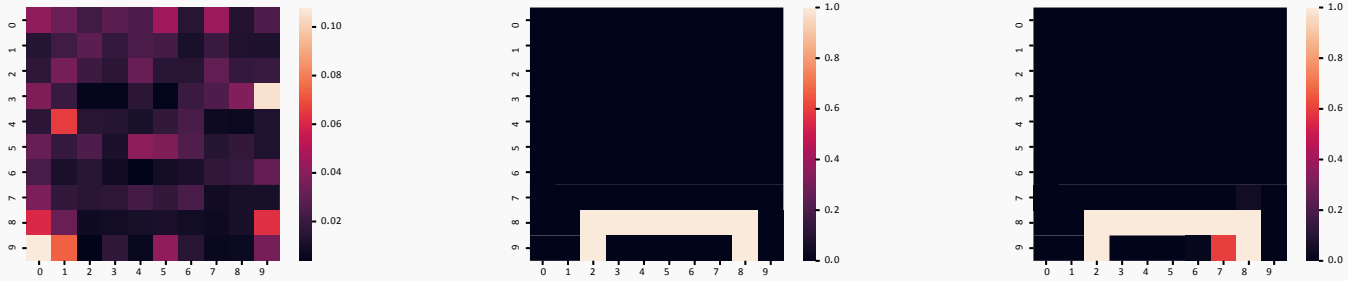
```
def loss_fn(output, target):

    smooth = 1e-17
    guess = torch.sigmoid(output)

    intersection = torch.sum(guess * target)
    union = torch.sum(guess + target) - intersection

    return -(intersection + smooth) / (union + smooth)
```

- Optimization algorithm: the Adam algorithm converges rather quickly
- Machine: training and evaluations were performed on my laptop's CPU

Results and comments

Data samples present a number of symmetries (Hinted by Giles Strong on Slack):

- Mirror symmetry along x and $y-$axis and symmetry between x and y
- This can be used to multiply by 8 the size of the dataset.
  However, the size of the dataset was not a limit in this case
- This can be used to have 8 independent predictions to average over. Increases slightly the final *IOU* of O 10^{-2} .



(a) Input

(b) Target

(c) DnCNN's prediction

Figure 2: Slice across one of the given test samples

Public/private IOU: 0.773/0.772

MODE workshop

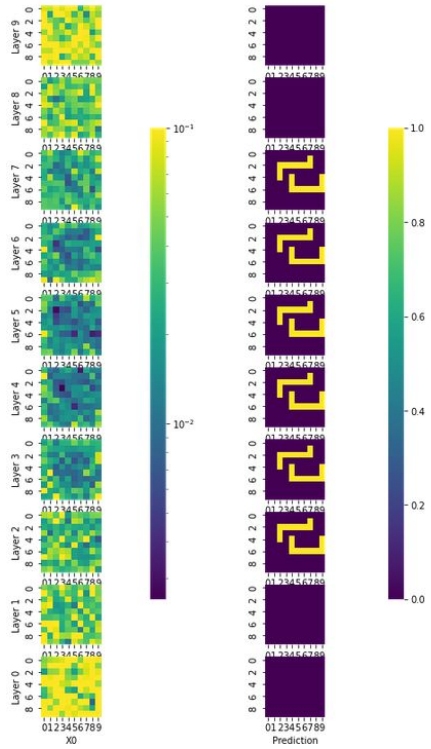
Challenge resolution

Florian Bury

https://github.com/FlorianBury/mode_diffprog_22_challenge/blob/main/training.ipynb

Speaking

The setup



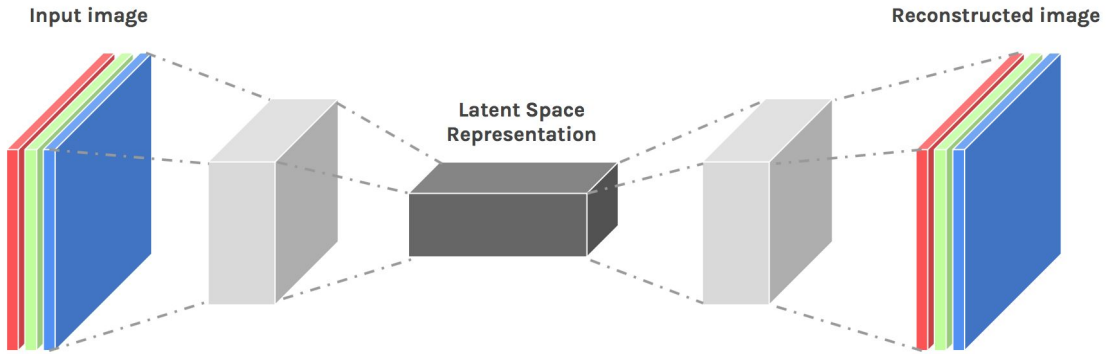
Observations :

- 2D distributions
- Different layers
- Noise and depth-dependent bias

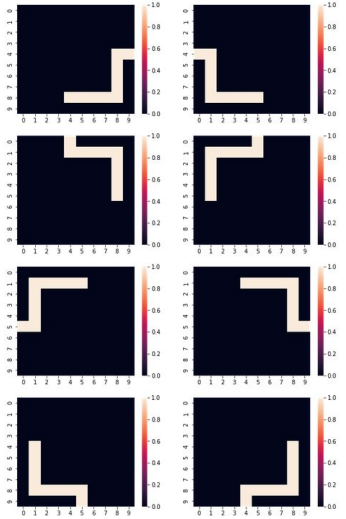
Convolutional Neural Network (CNN) :

- 2D Image processing
- 3 channels (RGB)
- Correlation between channels

⇒ Chosen architecture : autoencoder-like CNN with 10 input channels



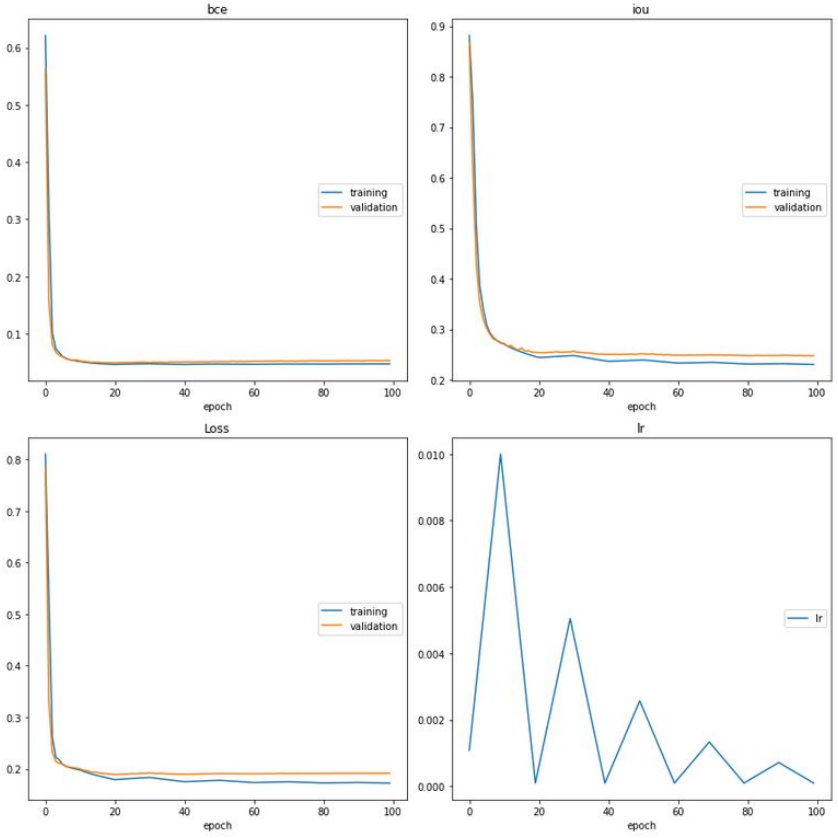
Preparations



- Data splitting :
  - Training : 80% (training of the CNN) with batches of 512 samples (trained on GPU)
  - Validation : 20% (model selection, threshold optimization)
  - Test : unlabelled set for the challenge
- Data augmentation :
  - Methods :
    - Rotations
    - Vertical/horizontal flips
  - Procedure :
    - Training set : apply rotation x hflip ⟶ factor 8 more data for training
    - Validation + test set : average over 8 transformations to smooth out potential bias
- Preprocessing : (x-μ)/σ per layer
- Loss functions :
  - Binary cross-entropy : not optimal
  - IOU loss : $\frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$ ⟶ Jaccard distance $J_1(y, \hat{y}) = 1 - \frac{(y \cdot \hat{y}) + \varepsilon}{(y + \hat{y} - y \cdot \hat{y}) + \varepsilon}$
  - Jaccard power loss : $J_p(y, \hat{y}, p) = 1 - \frac{(y \cdot \hat{y}) + \varepsilon}{(y^p + \hat{y}^p - y \cdot \hat{y}) + \varepsilon}$ (paper) ⟶ $1 \leq p \leq 2$ (used 1.75)

    ⟶ Penalizes more the worst predictions ⟶ more focus on hard predictions and better convergence

Training



- Training done on pytorch (in Jupyter notebook)
- Binary cross-entropy : start increasing as the model improves (not optimal after some point)
- IOU and loss (Power Jaccard p=1.75) :
  - Decreasing, not overfitting
  - Sometimes loss increases but IOU improves

  → “Harder” samples
- Used cyclic LR with decay rate to avoid learning rate optimization and local minima
- Best model :

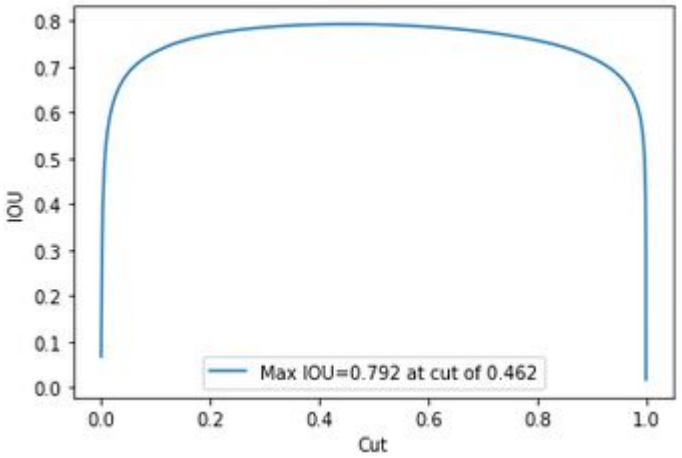
```
CNN(
  (encoder): Encoder(
    (layers): ModuleList(
      (0): Conv2d(10, 50, kernel_size=(3, 3), stride=(1, 1))
      (1): SELU(inplace=True)
      (2): BatchNorm2d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (3): Conv2d(50, 100, kernel_size=(2, 2), stride=(1, 1))
      (4): SELU(inplace=True)
      (5): BatchNorm2d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (6): Conv2d(100, 150, kernel_size=(2, 2), stride=(1, 1))
      (7): SELU(inplace=True)
      (8): BatchNorm2d(150, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (9): Conv2d(150, 200, kernel_size=(1, 1), stride=(1, 1))
      (10): SELU(inplace=True)
      (11): BatchNorm2d(200, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (decoder): Decoder(
    (layers): ModuleList(
      (0): ConvTranspose2d(200, 150, kernel_size=(1, 1), stride=(1, 1))
      (1): SELU(inplace=True)
      (2): BatchNorm2d(150, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (3): ConvTranspose2d(150, 100, kernel_size=(2, 2), stride=(1, 1))
      (4): SELU(inplace=True)
      (5): BatchNorm2d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (6): ConvTranspose2d(100, 50, kernel_size=(2, 2), stride=(1, 1))
      (7): SELU(inplace=True)
      (8): BatchNorm2d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (9): ConvTranspose2d(50, 10, kernel_size=(3, 3), stride=(1, 1))
      (10): BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (output_acti): Sigmoid()
)
```

Latent space dimension : 6x6

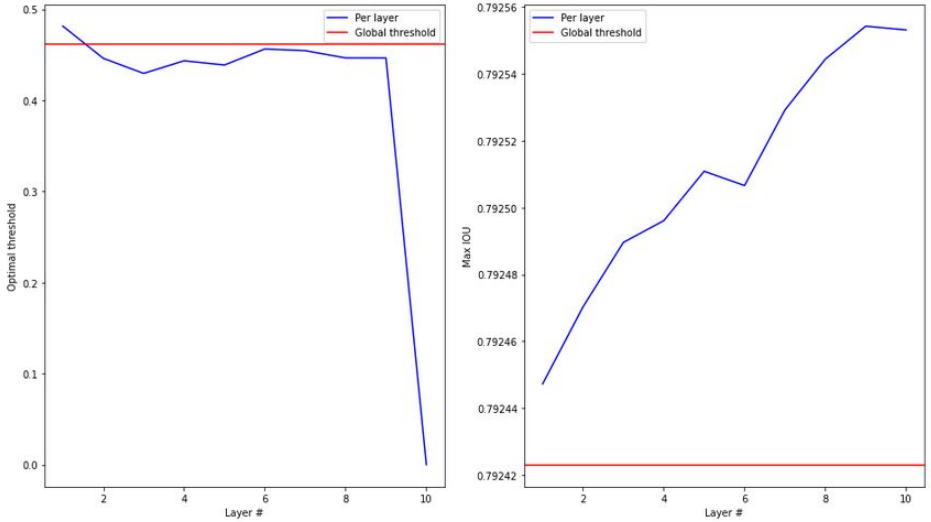
Threshold optimization

From predictions in [0,1] → wall (1) / soil (0)

First option : apply a global cut on all layers

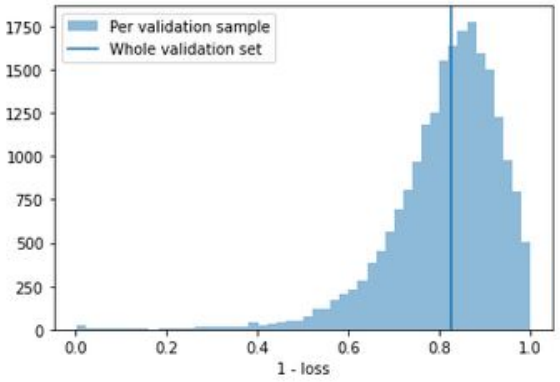


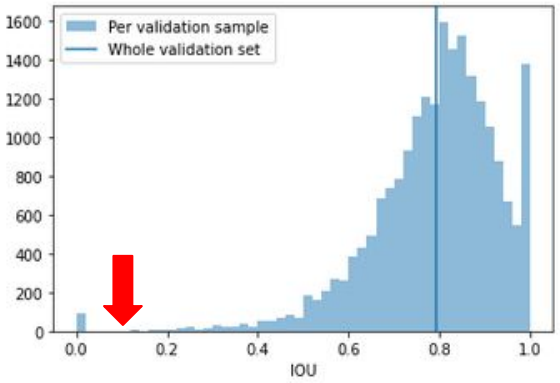
Second option : start from global threshold and optimize per layer individually

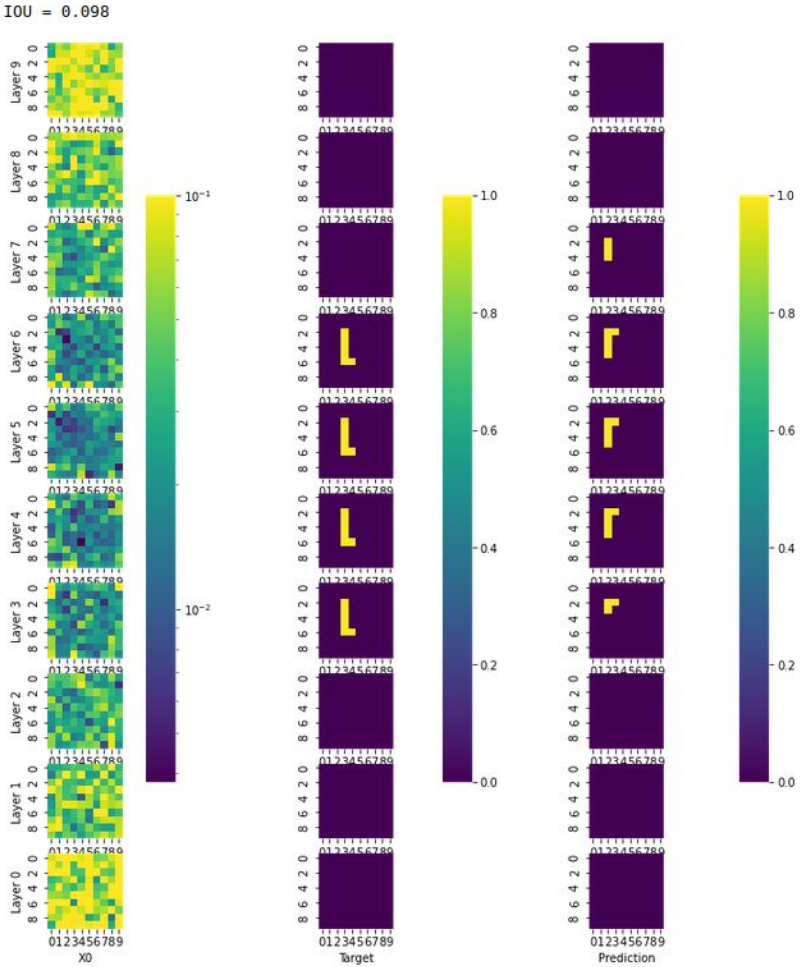


Very marginal gains in IOU
(Last layer is never populated by walls)

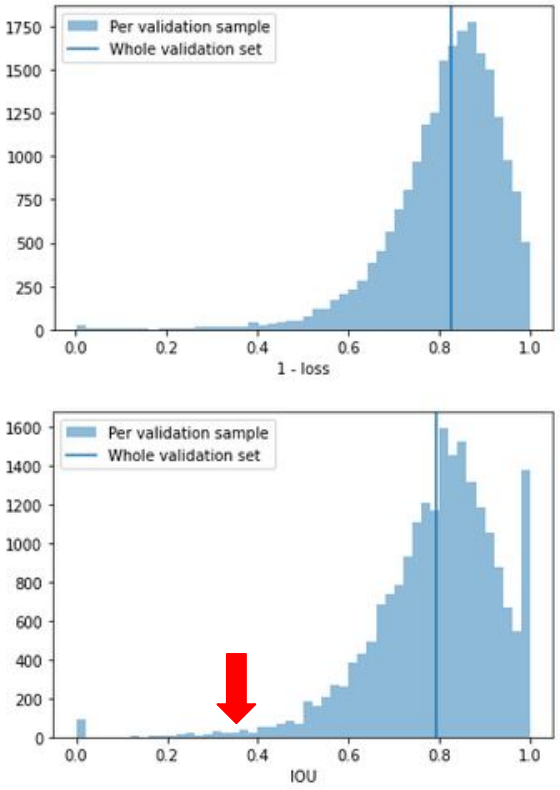
Threshold optimization

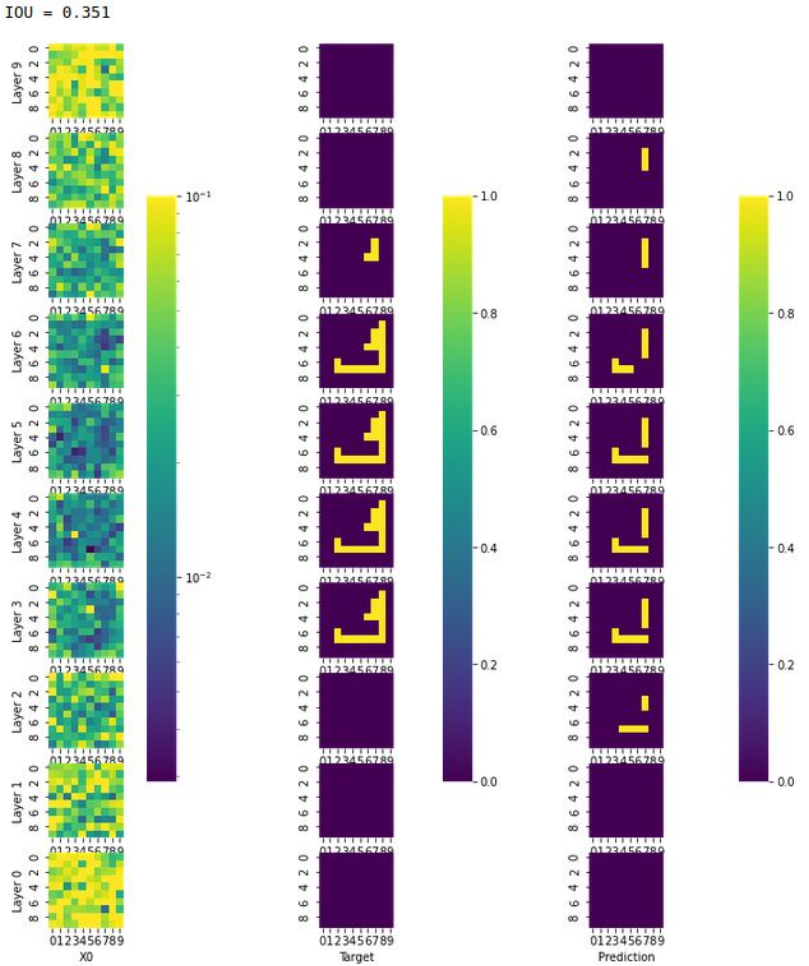




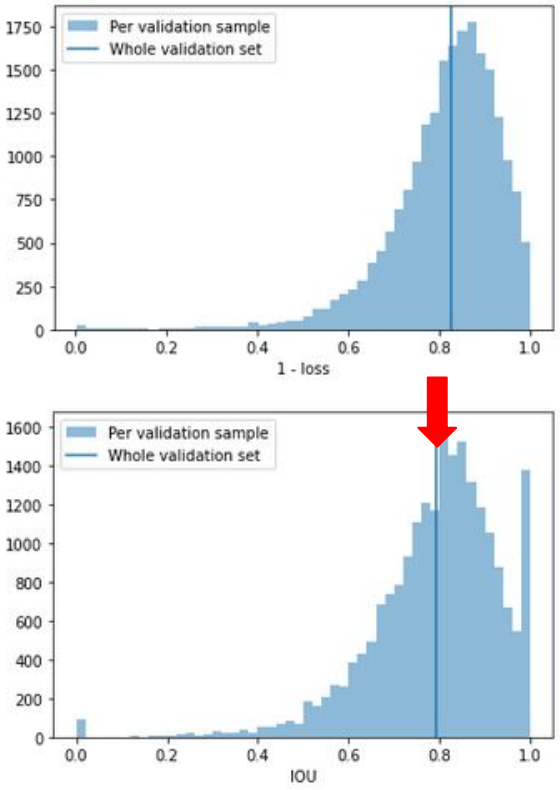


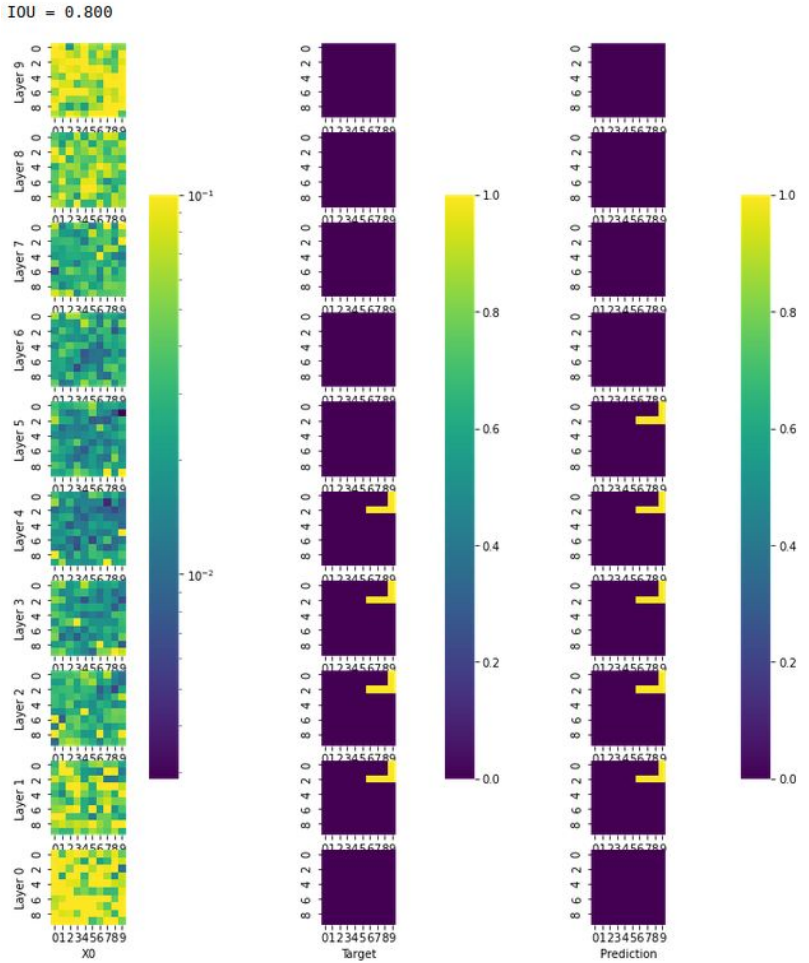
Threshold optimization



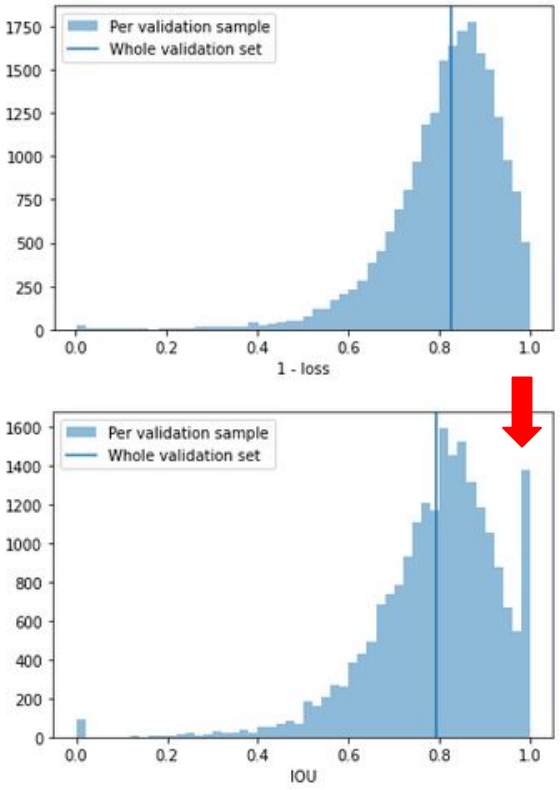


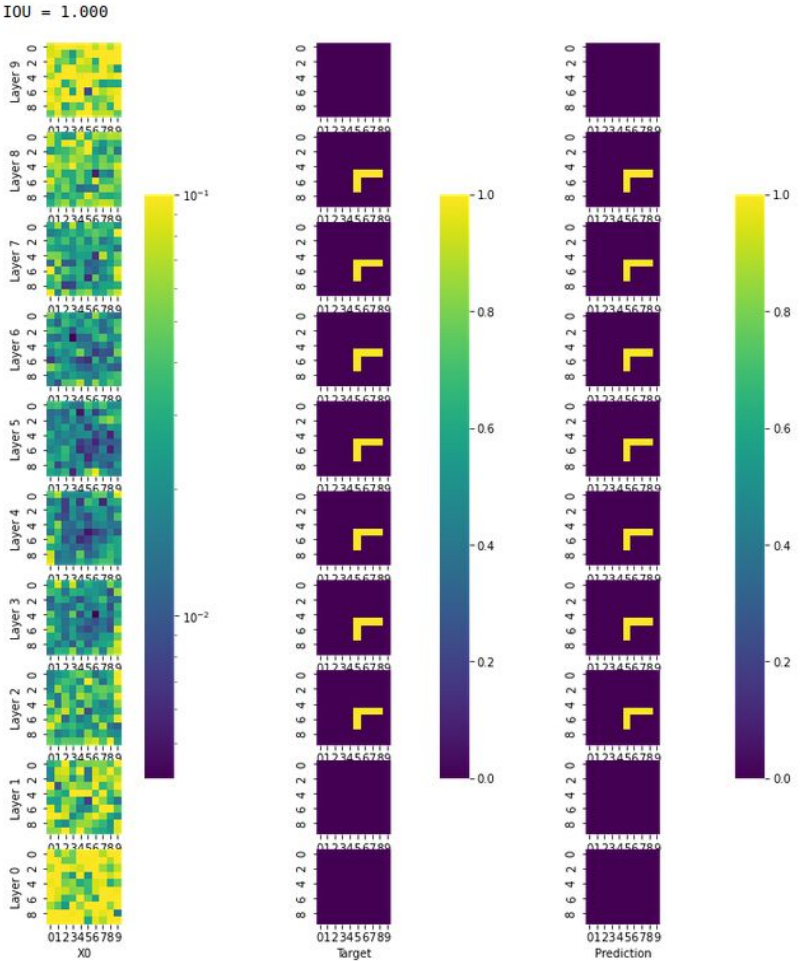
Threshold optimization



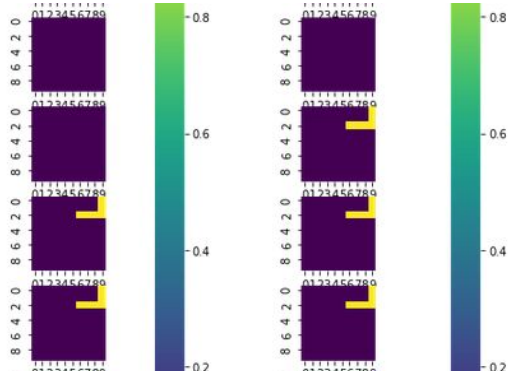


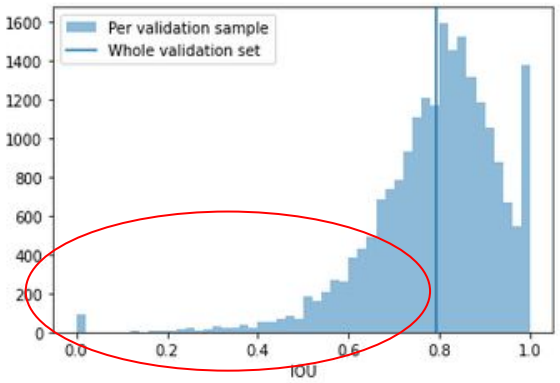
Threshold optimization





Future (potential) developments





Current issues :

- Low IOU samples (especially with small wall/soil ratio)
- Wrong wall height prediction
- Public/private IOU: 0.793/0.793

Architecture :

- CNN : more layer correlation
- Graph neural networks
- Other ?

Training :

- Weighted IOU (per-sample / per-layer)
  → Give more importance to
  - samples with low IOU
  - samples with wrong layer predictions
- Boosted training (similar to Adaboost M2 for example) : training set /batch sampling, weighted samples

DATA CHALLENGE ATTEMPT SUMMARY

Giles Strong

(N.B. Organiser -- solution unranked)

OVERVIEW

Approach

- Treat as semantic image segmentation problem in 3D space
- UNet-style model with a GravNet centre
- Soft-IOU loss
  - Compute IOU on un-thresholded predictions
- Adam optimiser with OneCycle schedule
- Train/test-time data augmentation
- Model ensembling
  - Train multiple models & average predictions
- Pseudo-labelling of test sample
  - Predict test data, treat as true targets & train larger model on train+test data

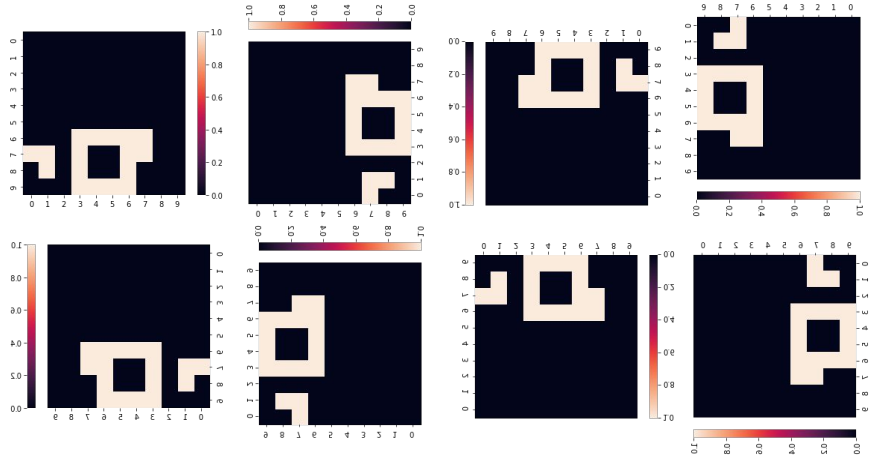
Resources & requirements

- 2x NVidia V100 32GB (CloudVeneto)
  - Models only need ~10GB
- Model parameter count = 2,695,606
- Train-time: 200 epochs ~7.5h per model (10 models across two GPUs),
- Inference time: ~0.027s/event (10x ensemble with data augmentation)
- PyTorch + LUMIN
- N.B. long training + ensembling is overkill: >0.8 IOU reachable in <20m training with single UNet model

Link to code: https://github.com/GilesStrong/mode_diffprog_22_challenge_giles_solution

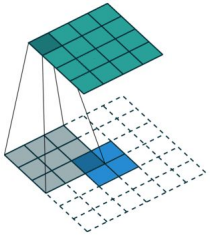
DATA AUGMENTATION

- Volumes display symmetries:
  - 90* rotational symmetry about the center in the XY plane
  - Flips in the X and Y plane
- Every sample can be transformed into 7 extra samples
  - All valid, but different enough to be useful
- Train-time data augmentation:
  - Apply random aug to each sample in batch during training ⇒ effectively increases training dataset
- Test-time data aug
  - For each testing sample take mean of predictions on all 8 versions



UNET ARCHITECTURE

- Output size must equal input size
- Two problems: where and what
  - Stride-1 CNNs must learn both simultaneously: difficult since voxels must also *communicate* across the volume
- UNet has an encoder to learn *what*, and a decoder to learn *where*
  - Originally used for medical imaging
- Encoder reduces image size but increases number of features
  - Allows distant voxels to more quickly communicate
  - Volume areas compressed as features rather than voxels
  - More features is greater variety in semantic meaning
- Decoder upscales compressed representation
  - Upscale via either:
    - transpose convolutions = opposite of convolution
    - Pixelshuffle = learn hi-res pixels as features in low res and reshape
  - Skip connections prevent information loss due to compression

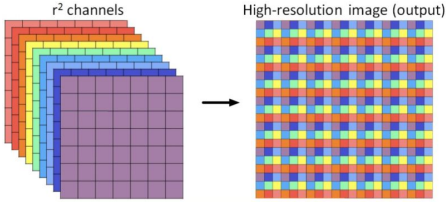


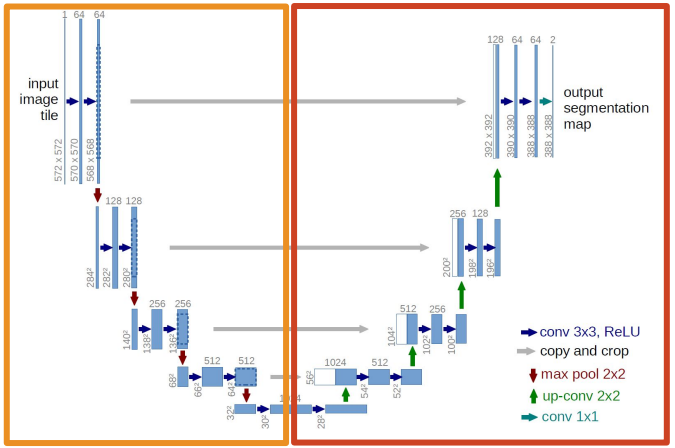
Transpose convolution

Vincent Dumoulin, Francesco Visin - A guide to convolution arithmetic for deep learning

PixelShuffle - upscale to 4x features then reshape to 0.5x features at double resolution

image source unknown





Source

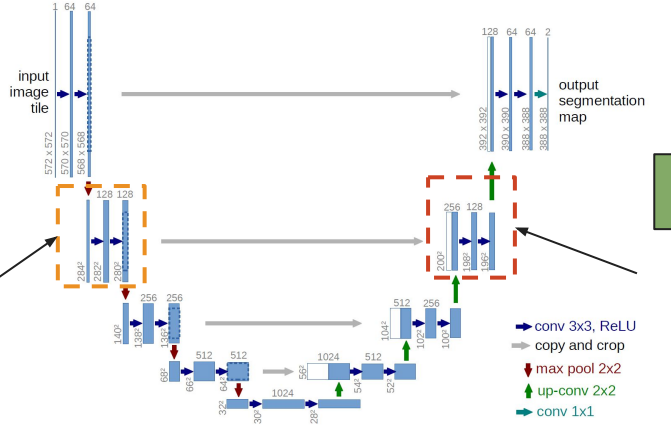
Encoder: each block halves image size and doubles features count

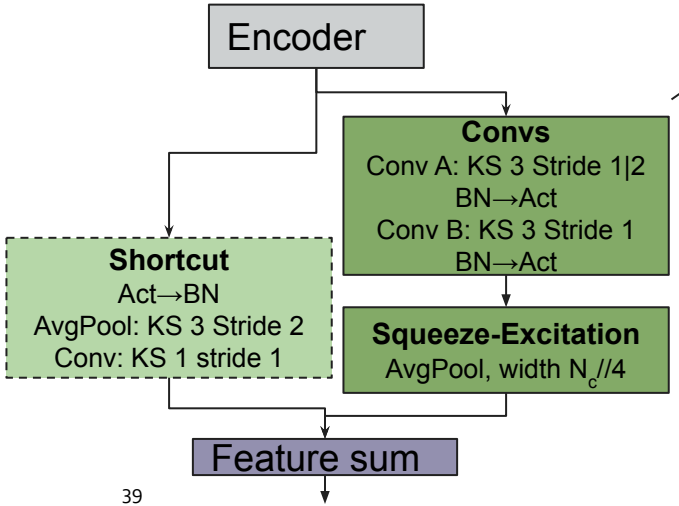
Decoder: each block doubles image size and halves features count
Features from same encoder level are concatenated to the upscaled decoder features

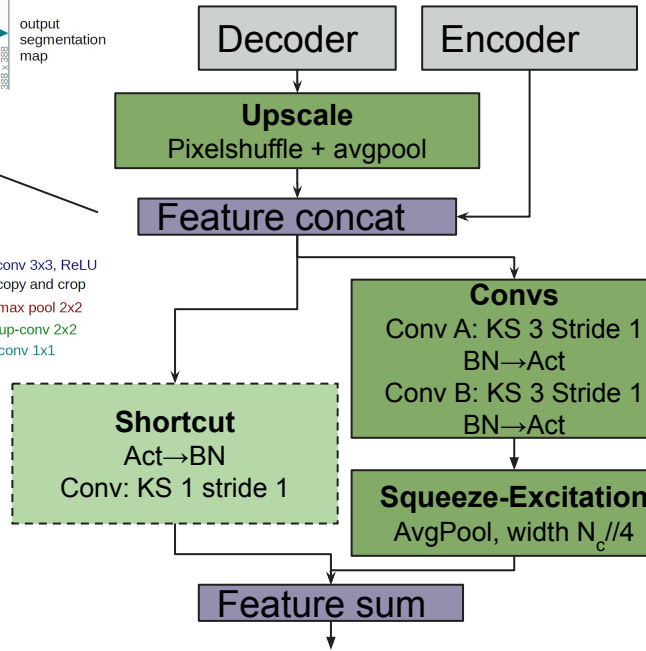
UNET ARCHITECTURE

- Built custom 3D UNet
  - Blocks are based on pre-activation ResNet, with squeeze-excitation
  - When downsampling, used avg-pool conv shortcut and Conv A is stride 2 with double the number of output features
  - When upsampling, used blurred pixelshuffle with half number of output features. Conv A (and shortcut) also halves the channel count since encoder features are concatenated with upscale output
- 6 down/up sampling stages: 10→5→3→2→3→5→10
  - Two residual blocks per sample stage
- Swish activation function (sigmoid output)

Original UNet diagram (2D) source

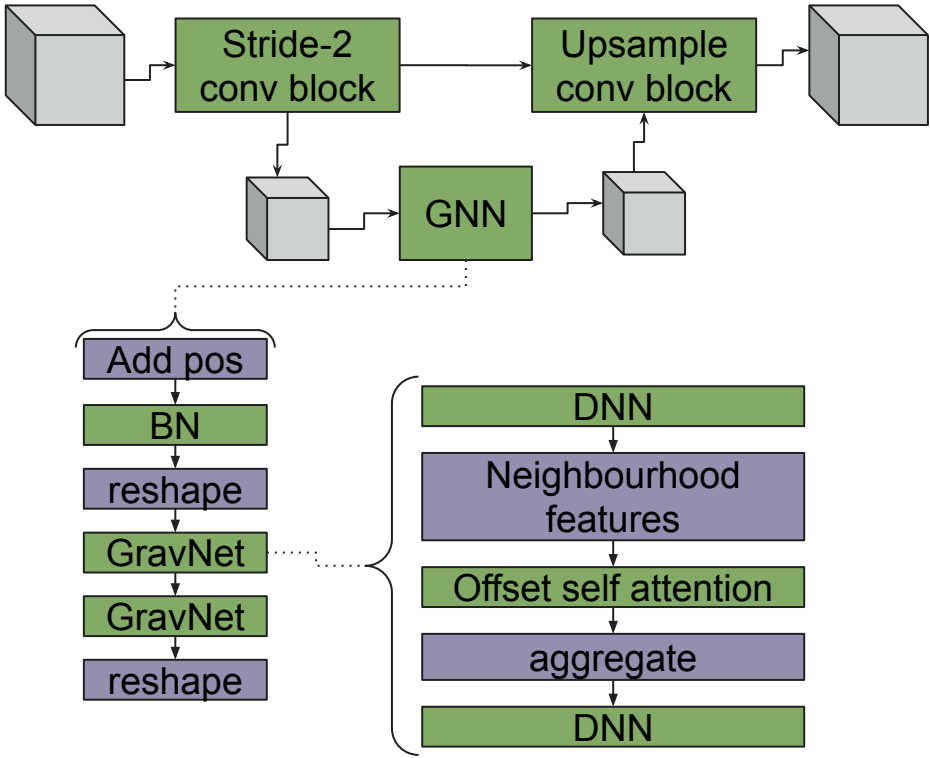






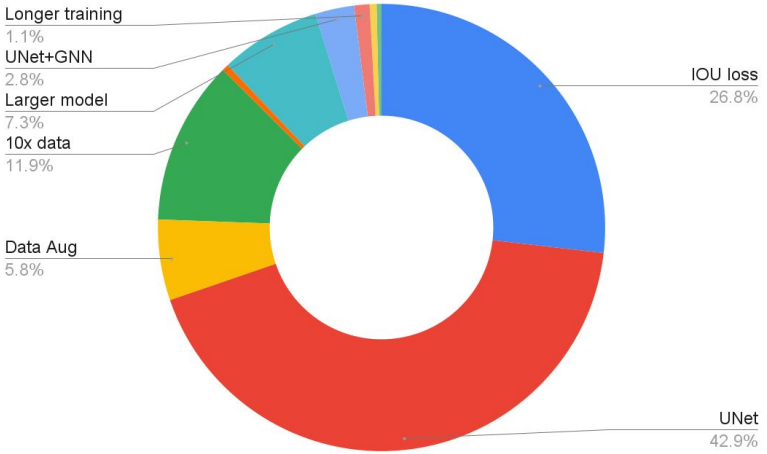
UNET+GNN

- CNNs naturally restrict communication between voxels to within the kernel size
  - Distant communication requires multiple layers
- Instead consider voxels as nodes and learn which other nodes they need to communicate
  - Voxel position can be encoded as features
  - New features can be learnt via information exchange between voxels
- GravNet learns to build graphs in latent space
  - No need to specify graph edges *a priori*
- But 1000 voxels exceeds memory:
  - Downsample volume with first block of UNet (125 voxels)
  - Pass through GNN
  - Upsample volume from latent representation using last upsample block of UNet (back to 1000 voxels)
  - Convert to class predictions with output block of UNet
- Offset self-attention used to weight graph neighbours prior to aggregation



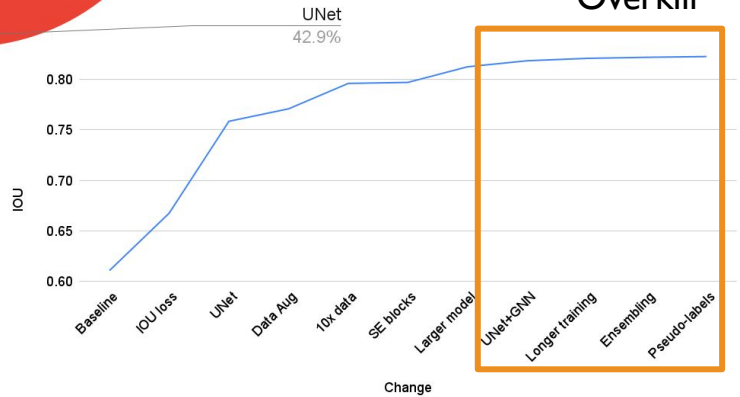
IMPROVEMENT SUMMARY

- Baseline: stride-1 BCE CNN, validation IOU = 0.611
- Final model: UNet+GNN ensemble+pseudo-labels
  - IOU = 0.822 public, 0.824 private
- Not a complete ablation study
  - Each change includes previous changes
  - Making changes in different order may give different improvement fractions
  - IOU may well saturate



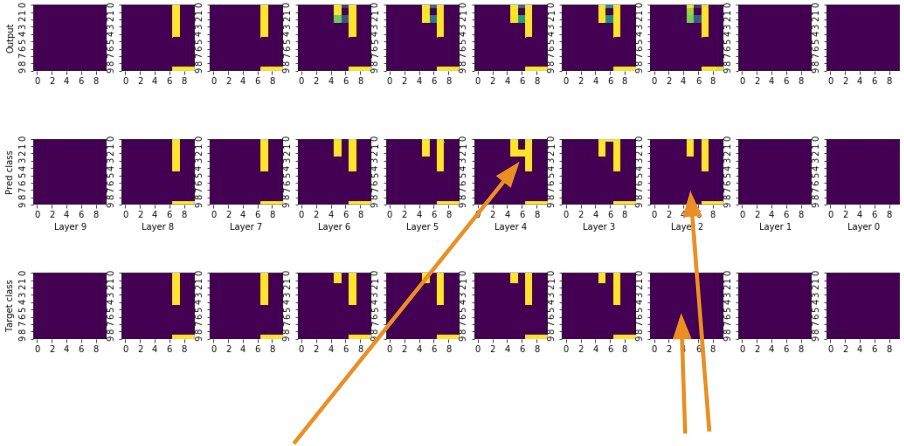
improvement sources

Overkill



LIMITATIONS

- GNN limited by memory (*only* had 32GB)
  - Downsampling + upsampling constrains power of GNN
  - Could potentially have either:
    - GNN with layers as nodes
    - Or downsample only in XY
  - Use of custom CUDA kernels might have mitigated memory overhead
- Increasing dataset size, model complexity, and train time doesn't have too much impact
  - Limitation is input features?
- Move from UNet to UNet+GNN = only minor improvement
  - Both have the same output block
  - Tried to improve output block with self attention, learnable layer rescalings, or transformer, but nothing helped



Model doesn't learn that all walls start on the same level (can encode as inductive bias?)

Model sometimes adds in / misses out whole layers in z

CLOSING

SUMMARY

- Congratulations to all the participants!
- A very diverse set of approaches and solutions:
  - 3D CNN → DNN
  - Fully flat DNN
  - Fully 3D CNN
  - Fully 2D CNN autoencoder
  - 3D CNN → GNN → 3D CNN
  - Custom losses, data augmentation, LR scheduling, ensembling
- Several participants cited hardware as a limitation: Institutes **need** to invest and have dedicated GPUs available for members to use

	Name	ID	Public IOU	Private IOU
0	FLORIAN_BURY	26	0.792526	0.793065
1	PIERRE_AUCLAIR	58	0.772798	0.771851
3	PHILIPP_MUNKES	04	0.691648	0.692056
2	BORJA_GONZALEZ	01	0.662763	0.665857

- Eligible participants ranked by private IOU (higher=better).
- Prizes:
  - 1st prize: Florian Bury - a tablet
  - 2nd prize: Pierre Auclair - a copy of "Evaluating Derivatives", a seminal textbook by Andreas Griewank and Andrea Walther, kindly offered by SIAM Publications
  - 3rd prize: Philipp Munkes - a copy of Pattern Recognition and Machine Learning, a seminal textbook by Christopher M. Bishop, kindly offered by Springer Nature