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

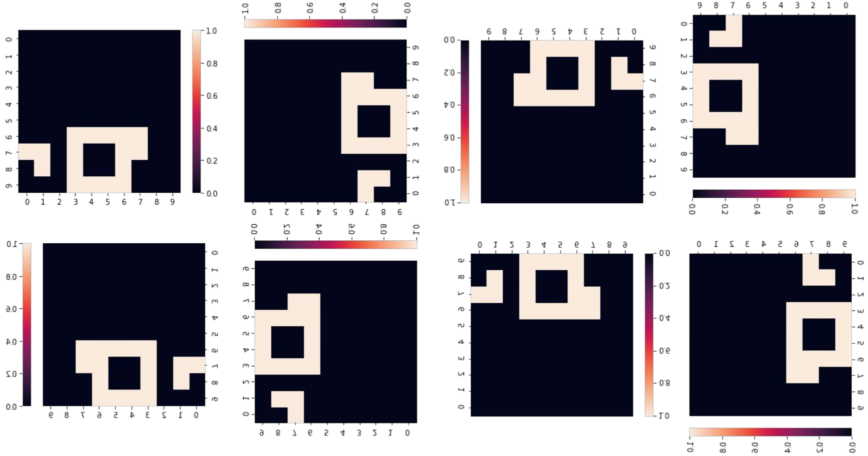
Resources & requirements

- 2x NVidia V100 32GB
  - 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

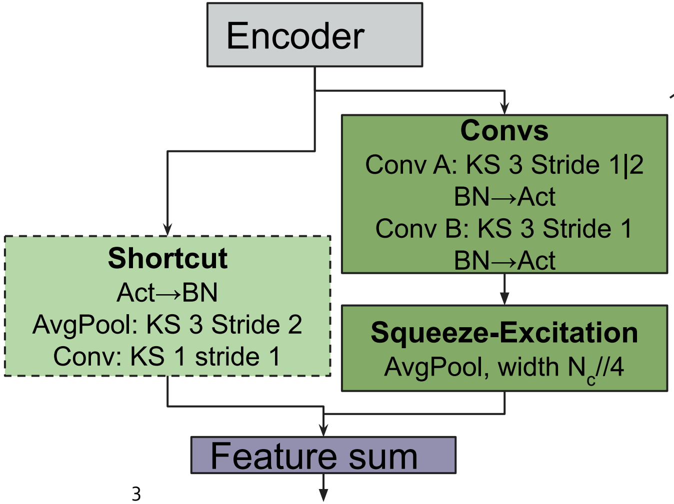
DATA AUGMENTATION

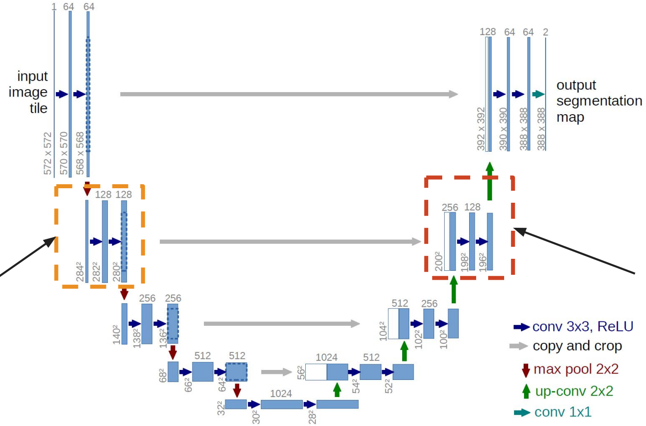
- Volumes display symmetries:
  - 90* rotational symmetry about the center in the XY plane
  - Flips in the X and Y plane
- Every sample can be transformed into 7 extra samples
  - All valid, but different enough to be useful
- Train-time data augmentation:
  - Apply random aug to each sample in batch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

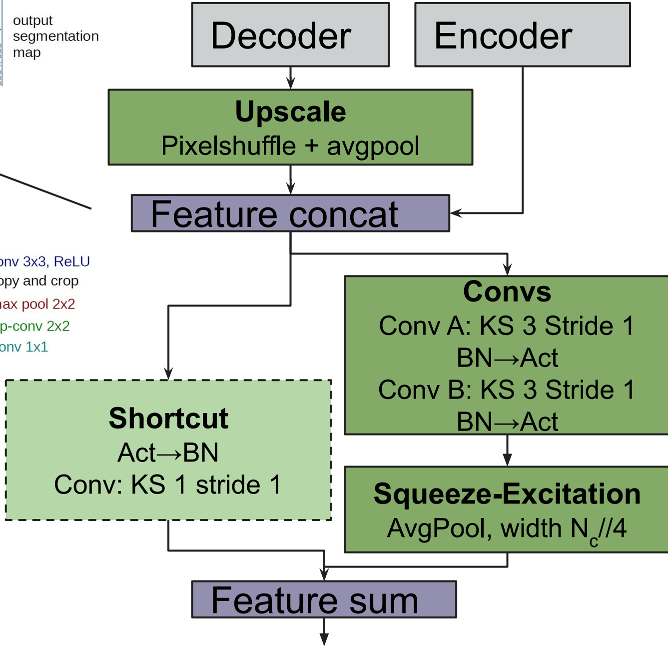


UNET ARCHITECTURE

- 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)

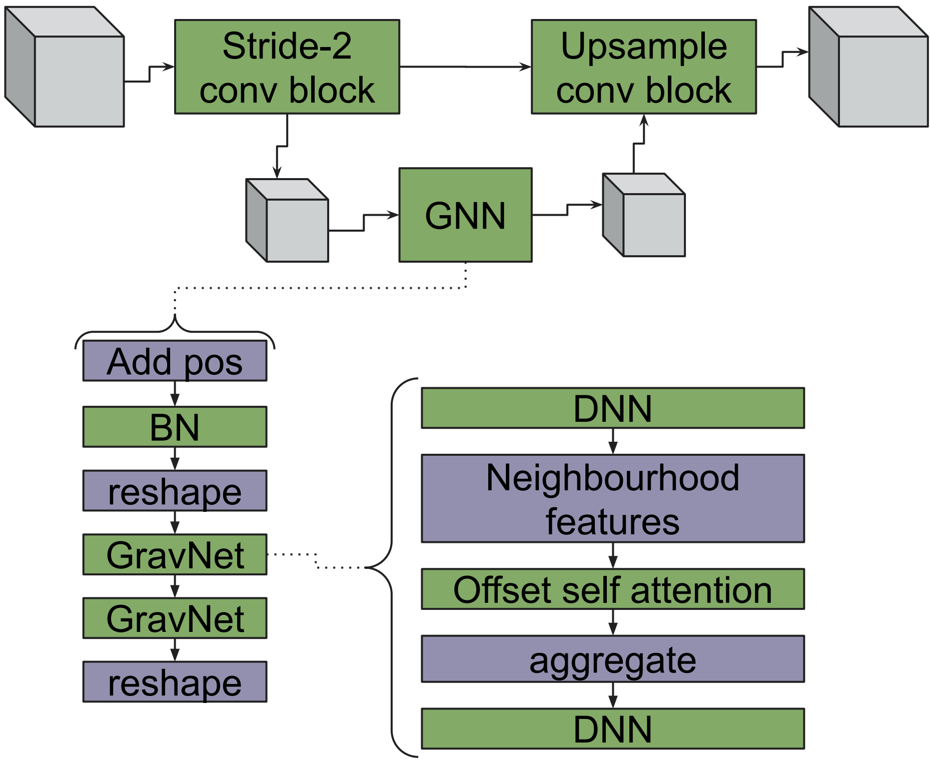






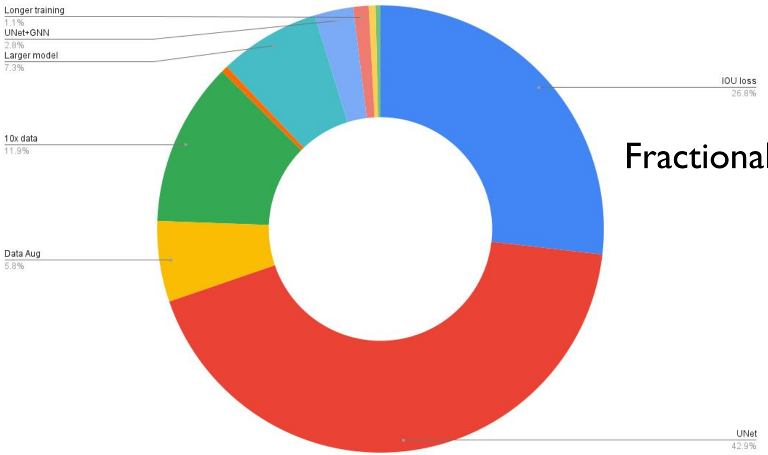
UNET+GNN

- 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



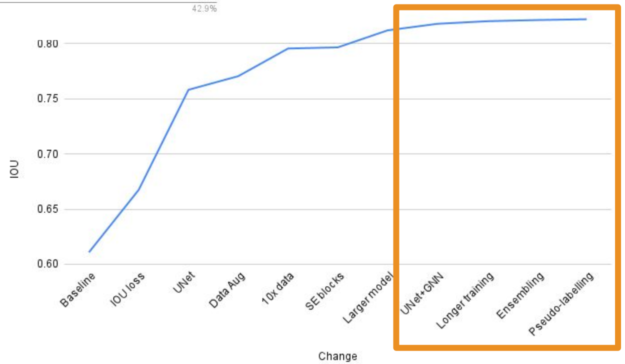
IMPROVEMENT SUMMARY

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



Fractional improvement sources

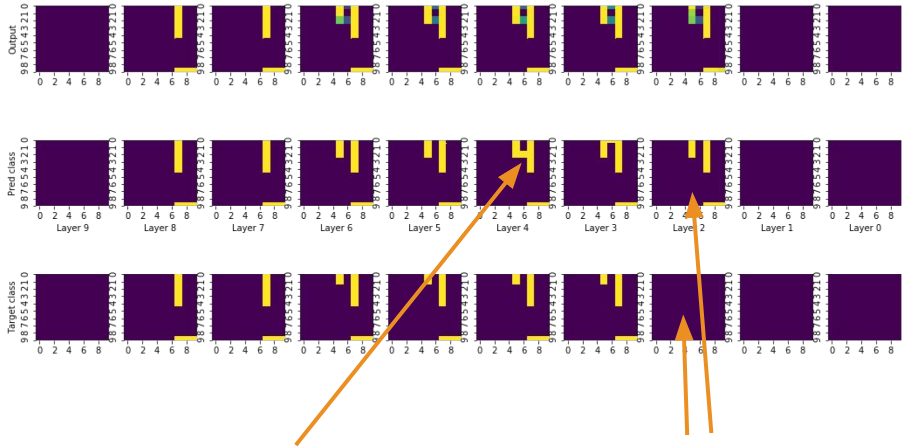
Overkill



N.B. Pseudo-labelling score is public IOU (no validation data available)

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