Hybrid CNN/GNN Network

Jake Calcutt

Introduction

Brett and I have worked to design a new Deep Learning ROI-finding network

- With architecture suggestions from Gemini

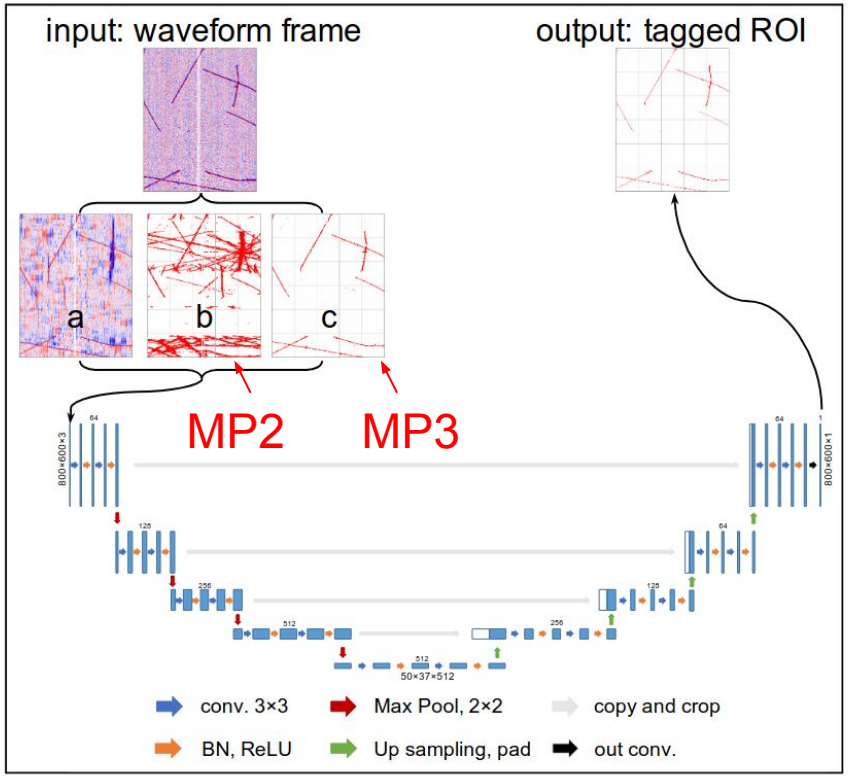
Hybrid CNN/GNN

- CNN/UNet: Finds readout representations in each 2D channel-vs-tick view
  - 1 UNet per plane
- GNN: shares information between the planes using geometric information
  - Crossing pattern of readout elements
- Relies only on deconvolved + filtered waveforms
  - No initial (manual) ROI-finding needed

Geometric Info in DNNROI – What is currently done?

As a reminder: we do currently encode geometric information in DNNROI

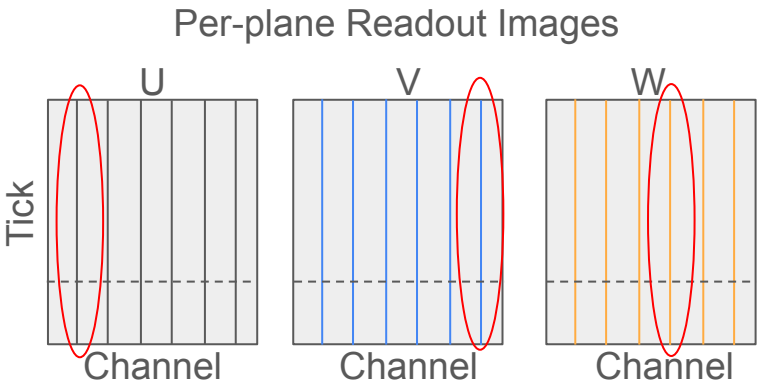
- For some target plane (U or V), find its ROIs manually
  - Look for ROIs on 2 other planes. For a given tick:
    - If crossing wires from other views had an ROI and target plane had ROI → MP3
    - If target plane had no ROI but the other views did → MP2
  - DNNROI then serves to refine inefficiencies & impurities from traditional ROI finding

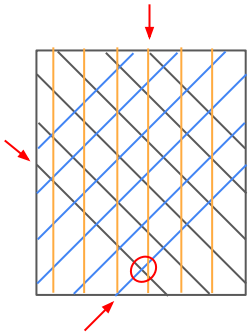


Geometric Info in DNNROI cont.

For a given time sample go from electronics channels → readout segments on a given anode face

- Find a crossing trio & determine status per previous slide definition





Room for improvement

DNNROI relies on traditional ROI finding:

1. Needs dedicated tuning even for ‘normal detector operation’
2. Has a lot of hardcoded assumptions in current SigProc implementation
    a. Assumes specific set of filters for U/V but not W
    b. No MP2/MP3 finding for W plane, etc.
        i. This is important for PDHD APA1
3. Current implementation could be faster

End-to-end DL approach could help:

1. Instead of ROI tuning + DL training → Just DL training
2. Opportunity to make more general
3. Optimize model implementation to reduce processing time
    a. Caveat: not guaranteed

Additional Benefits – PDHD APA1

PDHD APA1's W plane has low S/N – DNNROI would be useful. However:

- Lots of hardcoded ROI finding operations not applied to W plane
- Different filters might need to be applied
- MP2/MP3 finding not done for W planes

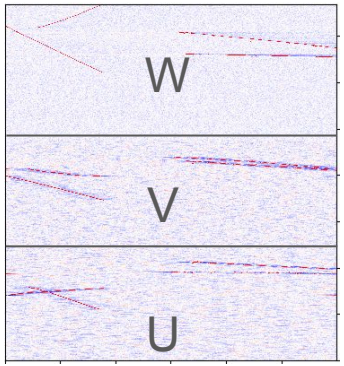
Perhaps we can achieve good efficiency and purity for W plane ROIs

- This would be very beneficial for higher-level reconstruction within APA1

New Model Design

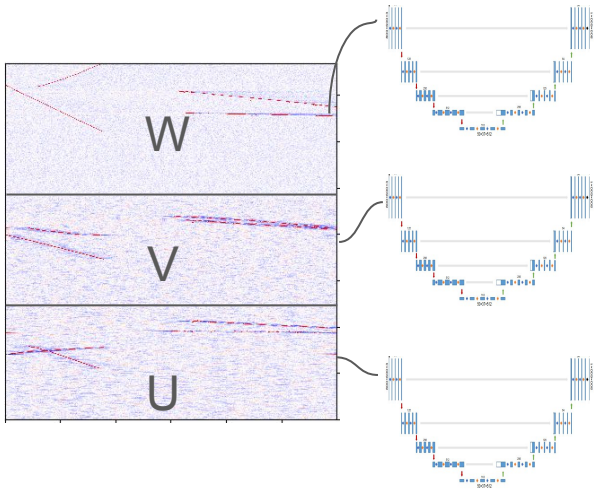
New Model Design

Deconvolved & filtered readout in each view



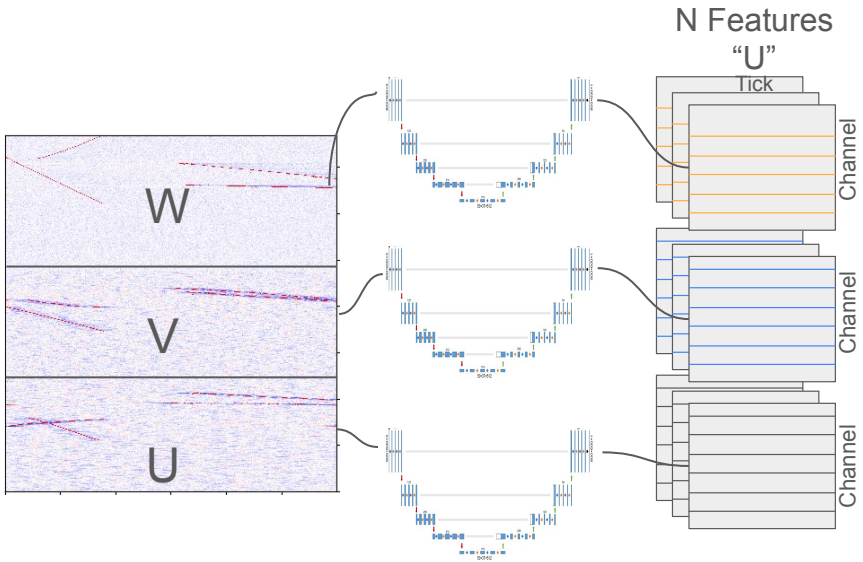
New Model Design

3 UNets (1 per view)

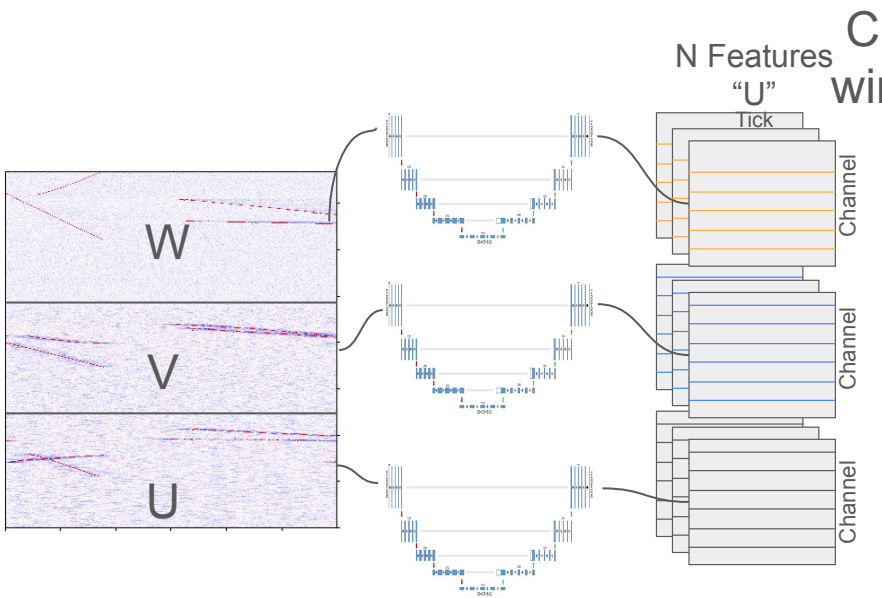


New Model Design

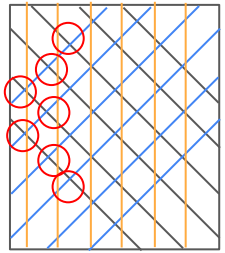
Results in N
Features from UNets



New Model Design

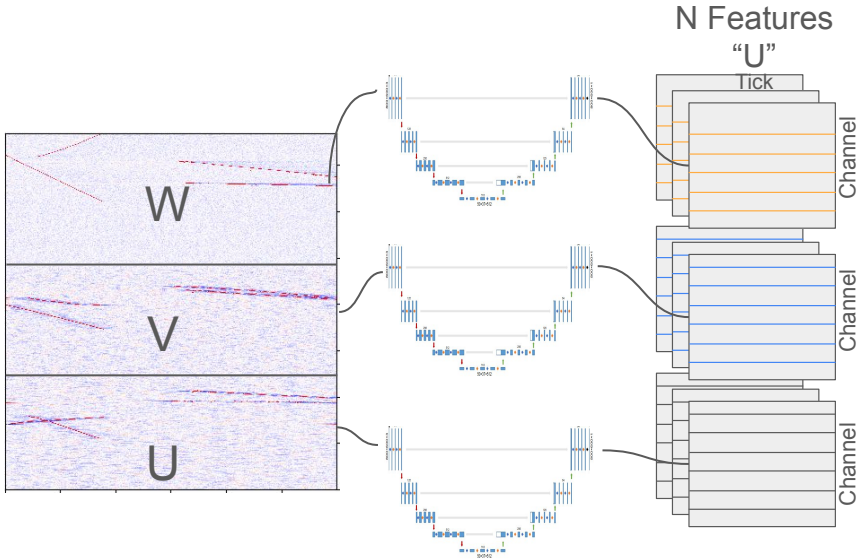


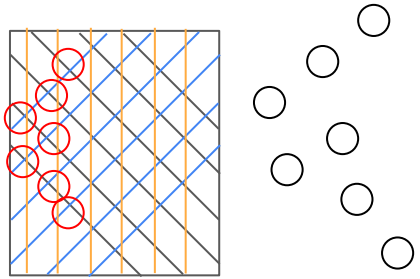
Crossing triplets of wires/strips for each tick



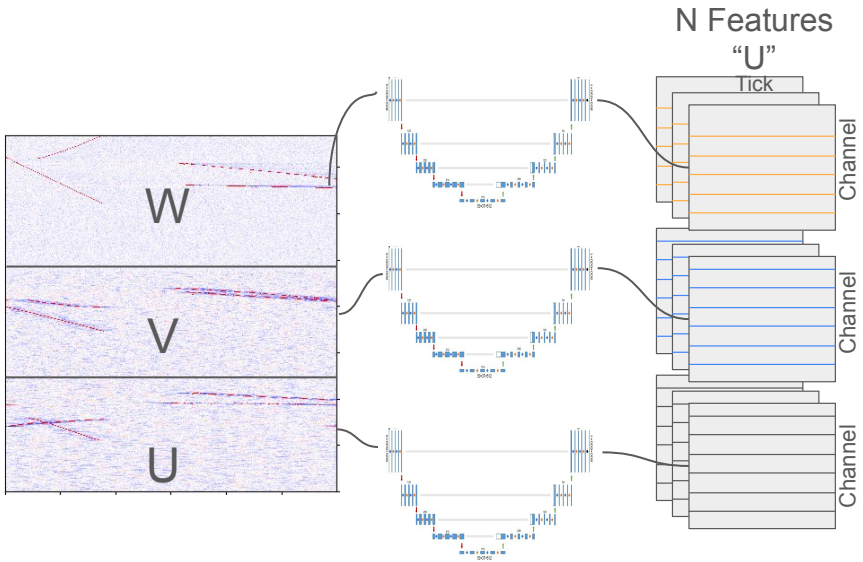
New Model Design

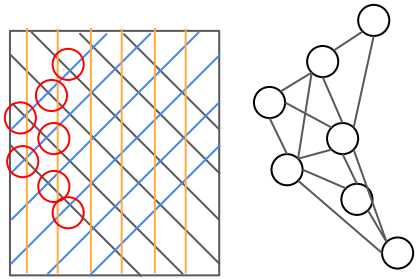
Defines nodes of a graph (concatenate features from each wire)





New Model Design

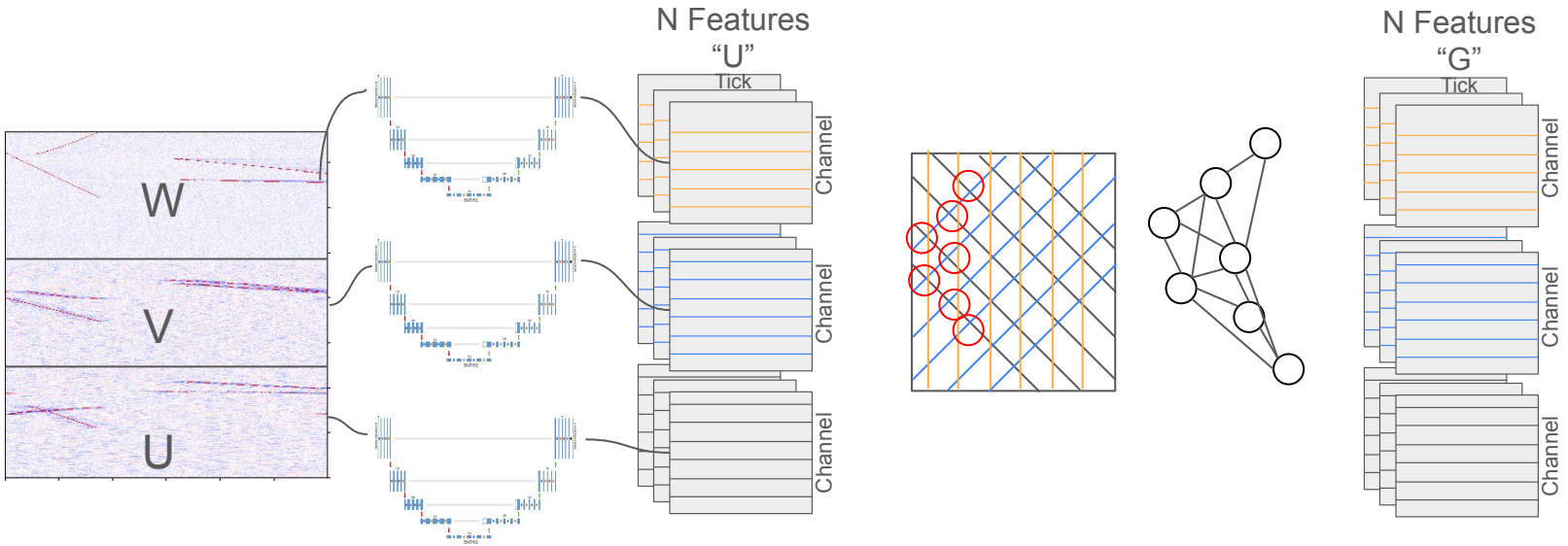




Determine edges from nearest neighbors

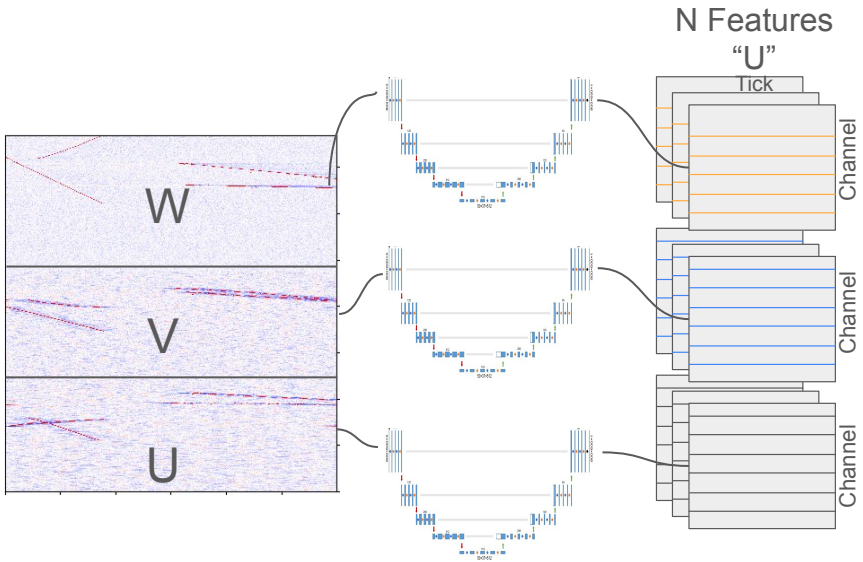
New Model Design

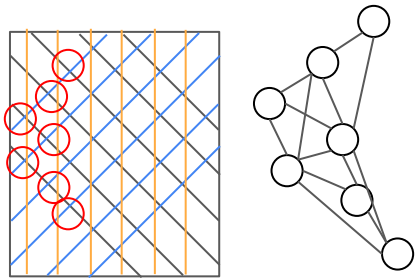
Pass through GNN
→ Geometric
Information Sharing

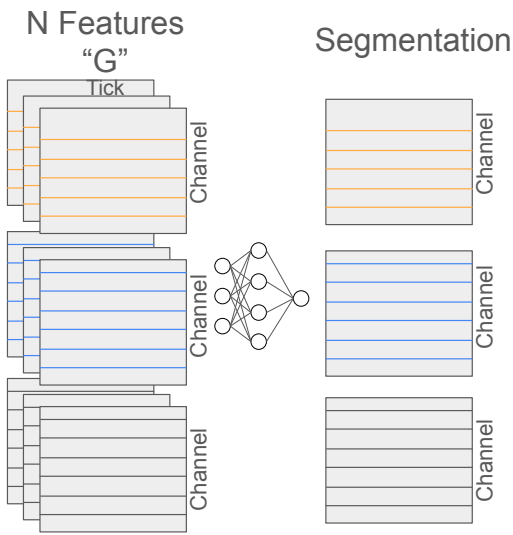


New Model Design

Pass through MLP (applied to channels) to turn into logits → Compare to true ROIs





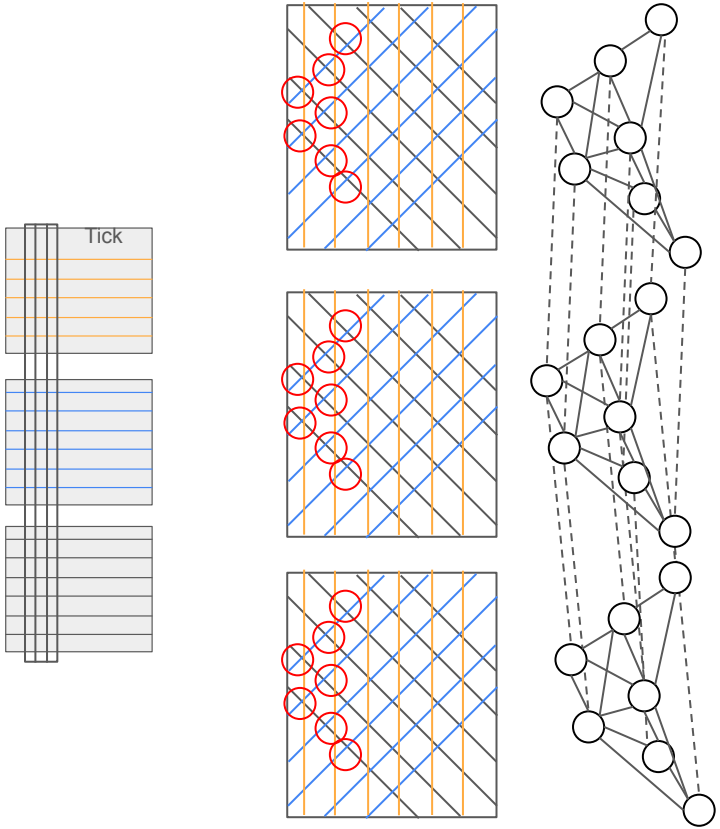


Sliding Time Window

A suggestion from Gemini was to encode temporal (third spatial dimension) information into the graph by using a ‘sliding window’

Connect nodes representing the same wire crossings at different points in time

- Messages will be passed between nodes in time during training, giving the target subgraph (middle nodes) local temporal info
- Size of window is a hyper parameter



Hyperparameters & Setups

There are many setups we can test/optimize over:

1. Input images: Can we provide several filtered images (Loose LF, Tight LF, etc.) to achieve better performance?
2. Number of UNet output features (No need to make them the same between plane types)
3. GNN Setup:
    a. Number of hidden channels
    b. Number of message passes
    c. What to include for edge features (distance between nodes, etc.)
    d. General GNN architecture (currently using GATConv – Could also try GATv2Conv? GraphSAGE?)

Preliminary tests

Test setup

- Input: Loose-LF images
- 4 UNet features
- GNN:
  - 16 hidden channels
  - 4 message passes
  - 8 output channels
- Time-window width: 1 tick
- Output MLP: 1 layer (8→1)

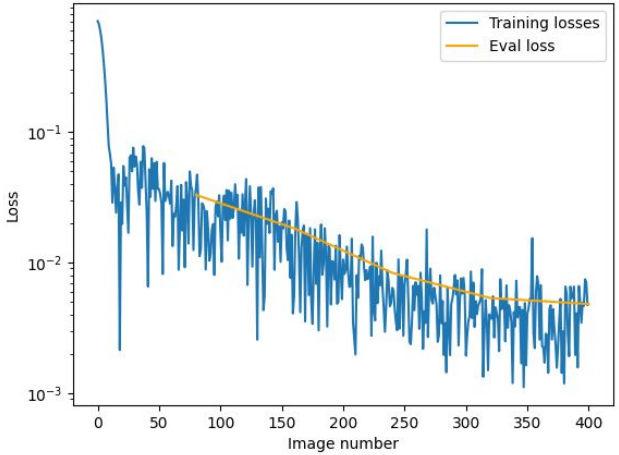
Trained on 80 (test on 20) simulated (cosmic gen + g4) PDVD images

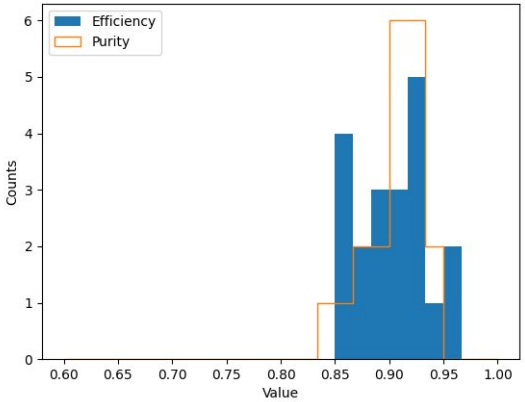
90+-5% efficiency & purity

- Relative to all real pixels in each event, need to check vs angle relative to strips

Each epoch took (45 min)

5 epochs (one epoch every 80 images)





Cropped from a full image

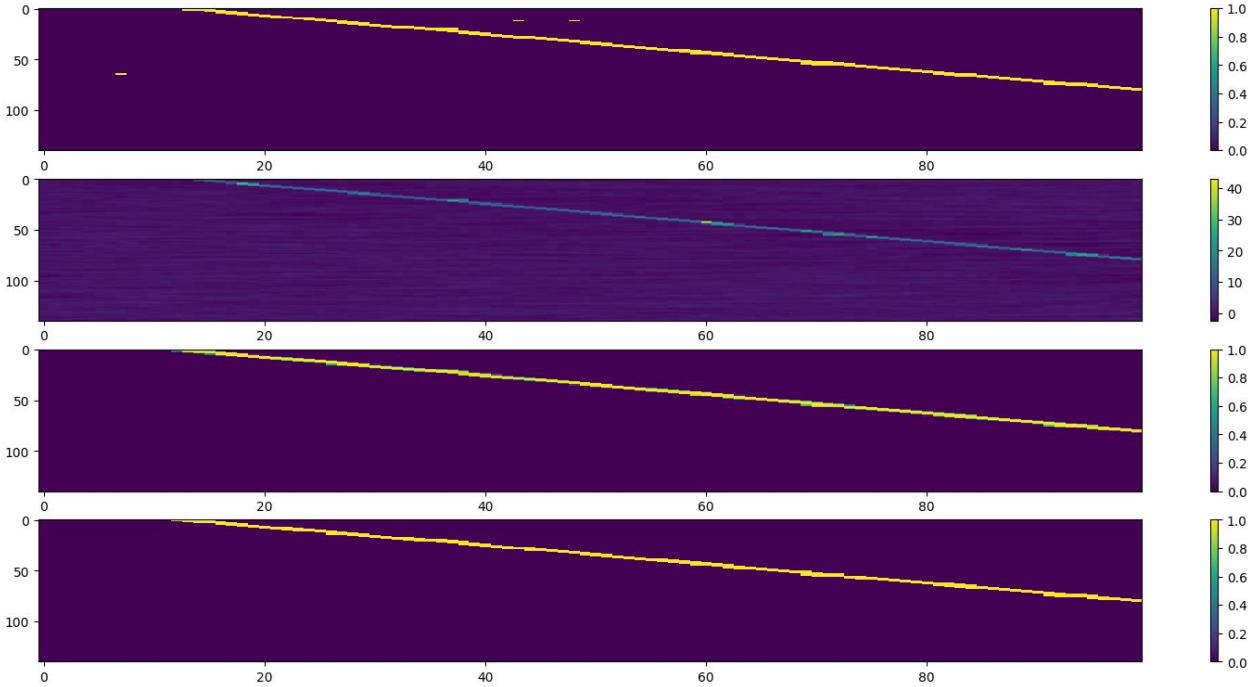
Inference example 1

Truth

Input

Output

Output > 0.5



Cropped from a full image

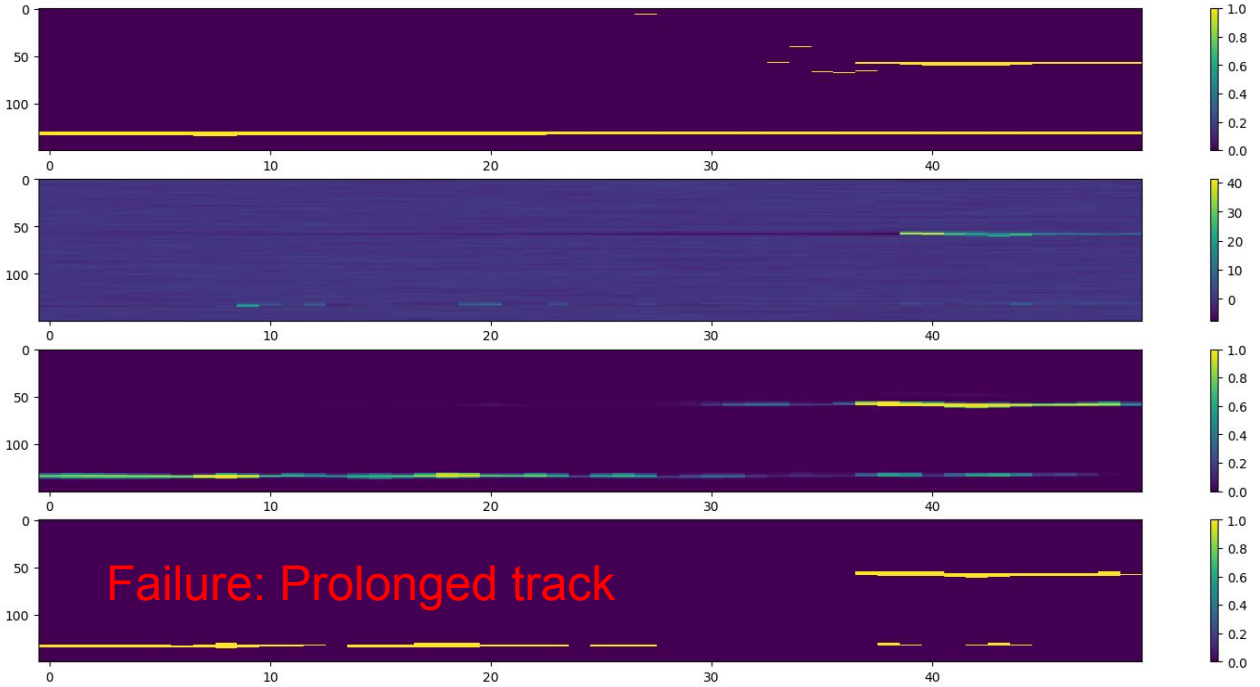
Inference example 2

Truth

Input

Output

Output > 0.5



Cropped from a full image

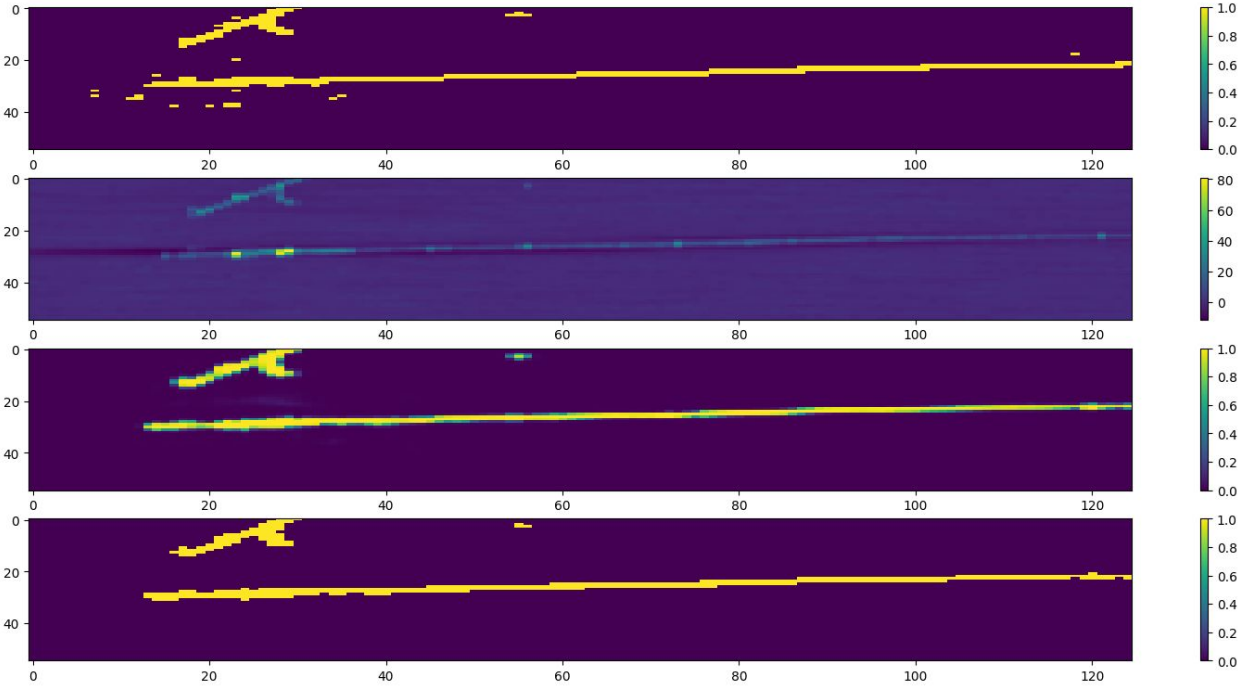
Inference example 3

Truth

Input

Output

Output > 0.5

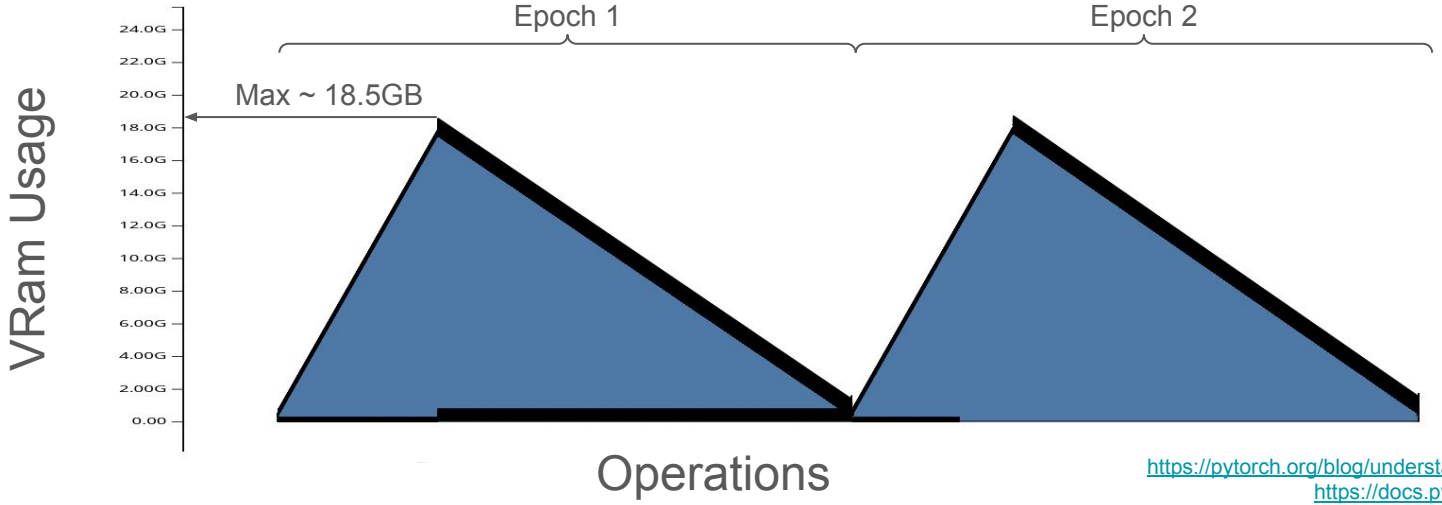


Resource Utilization

The setup presented above takes ~44 mins per epoch to train

And the memory utilization is considerably high for training (18.5GB VRam Max)

- Driven by calculating gradients →Inference is not as bad



https://pytorch.org/blog/understanding-gpu-memory-1/
https://docs.pytorch.org/memory_viz

Summary/Next Steps

Presented new CNN/GNN network for end-to-end ROI finding from deconvolved & filtered images alone

- Graph portion encodes geometric information from known wire/strip crossing pattern
- Preliminary results are promising: achieving ~90% efficiency with 80 input images trained over 5 epochs

Next steps

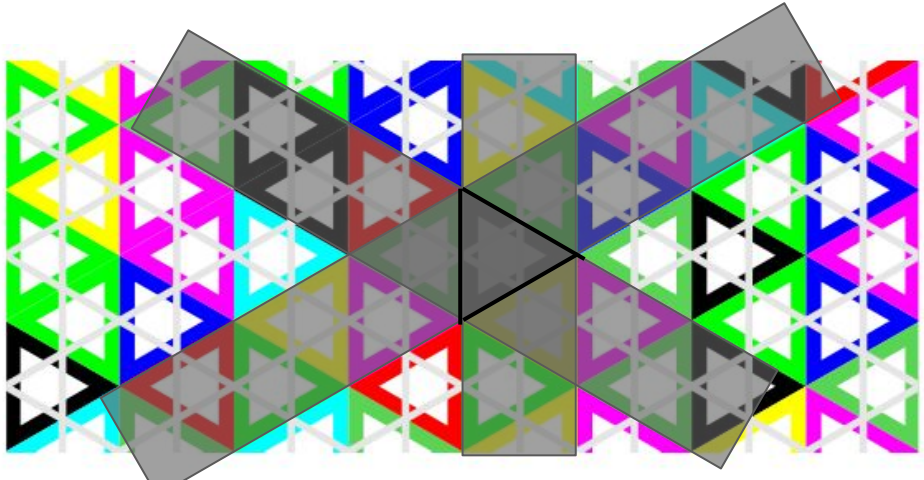
- Refining architecture for performance & resource utilization optimization
- Consider possible extensions
  - Can we add additional loss terms based on charge in 3D space?
    - Note: want to avoid predicting charge as output – this would just be to improve tomographic performance

Backup

How are nodes defined

Using the concept of cells from Wire Cell Imaging

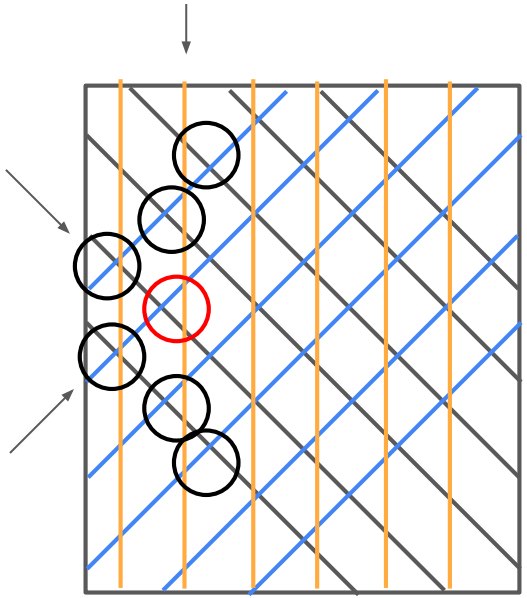
- Pitch-wide rectangles centered on wires/strips: Below is an idealized setup (symmetric angles in microboone)
  - For DUNE-style anodes, we have several shapes/areas of cells
    - Use this information in node features (WIP)

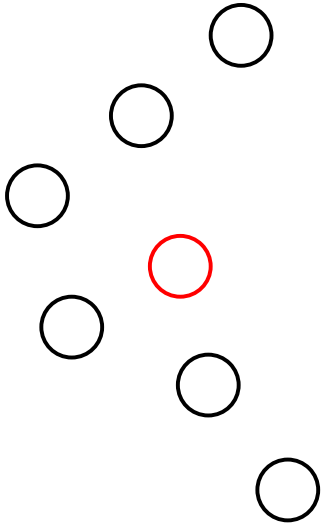


How are edges defined

Current implementation. On either face of an anode, for a given node, start at its wires

Increment +- N in each plane

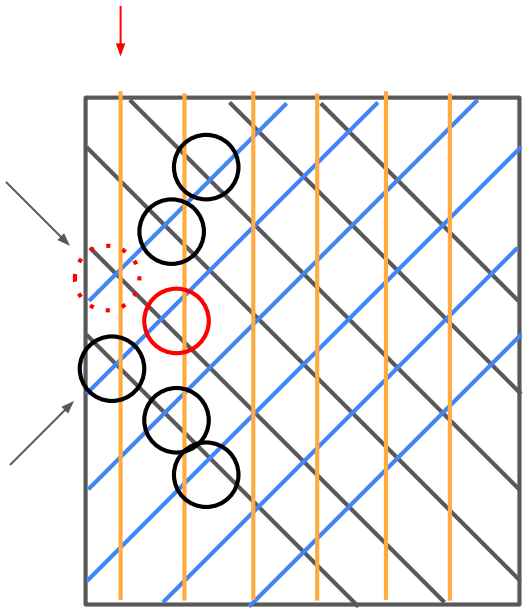


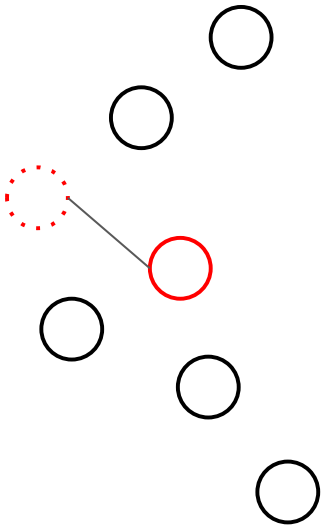


How are edges defined

Current implementation. On either face of an anode, for a given node, start at its wires

Increment +- N in each plane

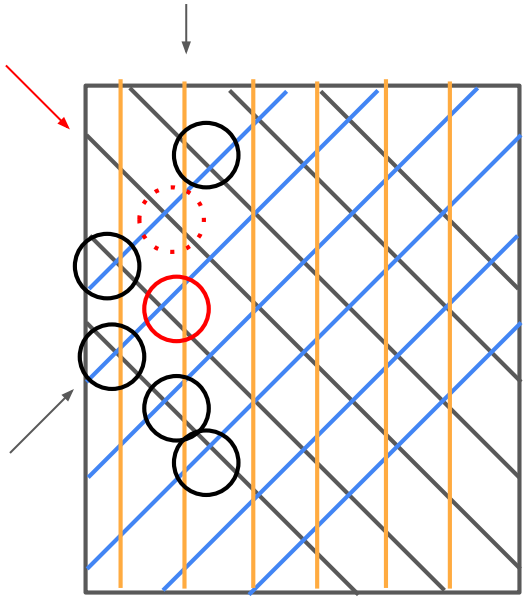


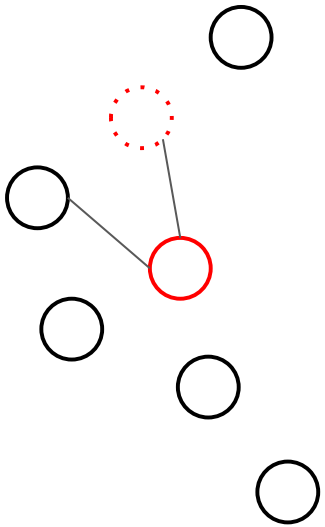


How are edges defined

Current implementation. On either face of an anode, for a given node, start at its wires

Increment +- N in each plane

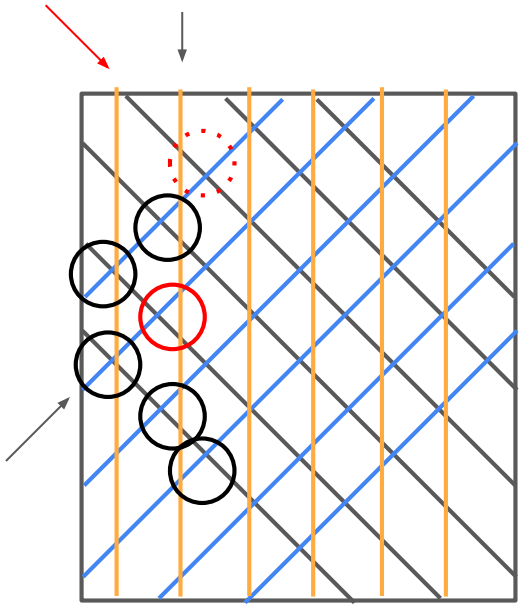


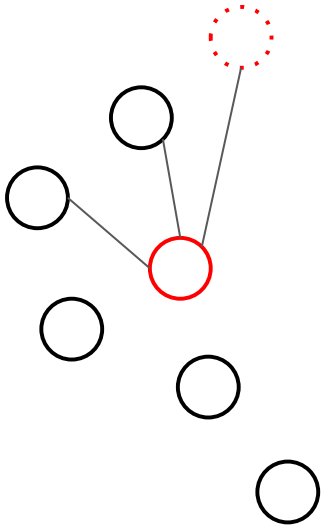


How are edges defined

Current implementation. On either face of an anode, for a given node, start at its wires

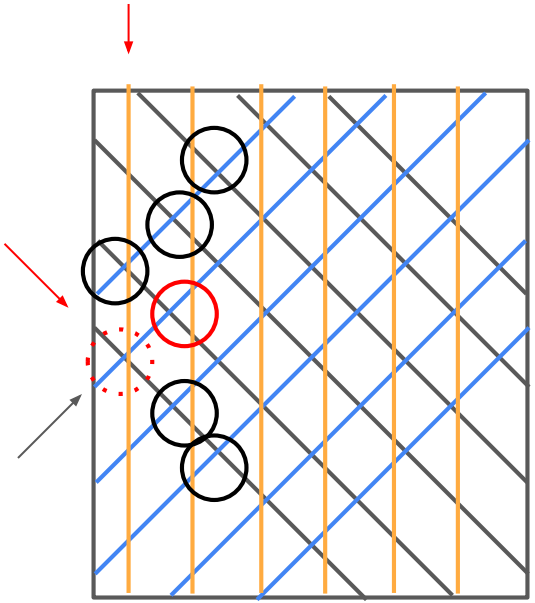
Increment +- N in each plane

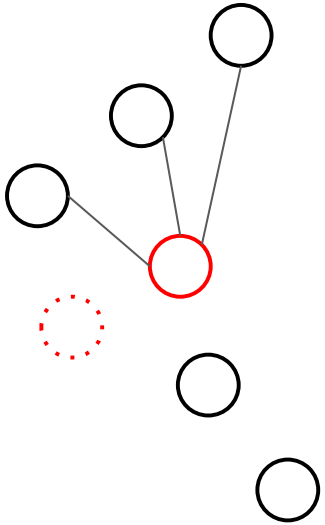




How are edges defined

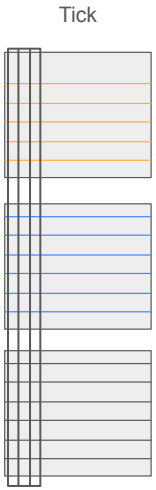
I just realized this would require a shift in 2 wire planes at once, so this is one deficiency in this algorithm

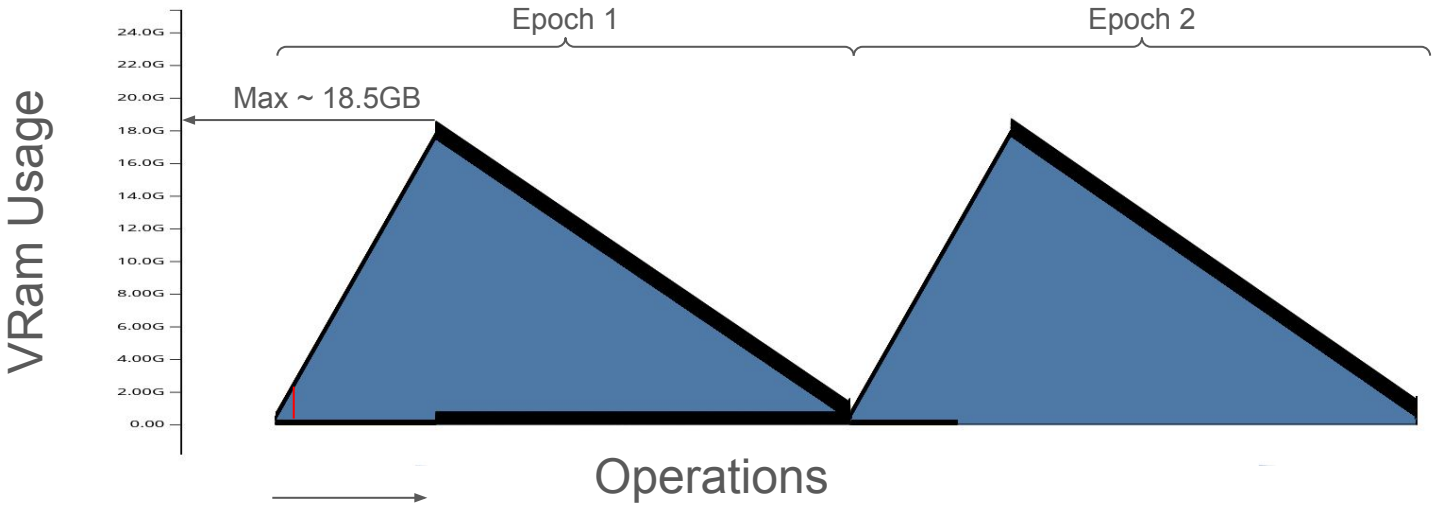




Resource Utilization – Understanding Memory Profile

Pyramid shape comes from sliding window

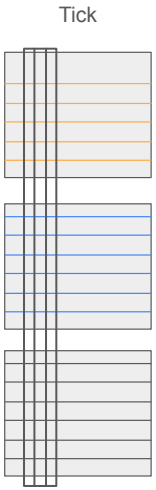


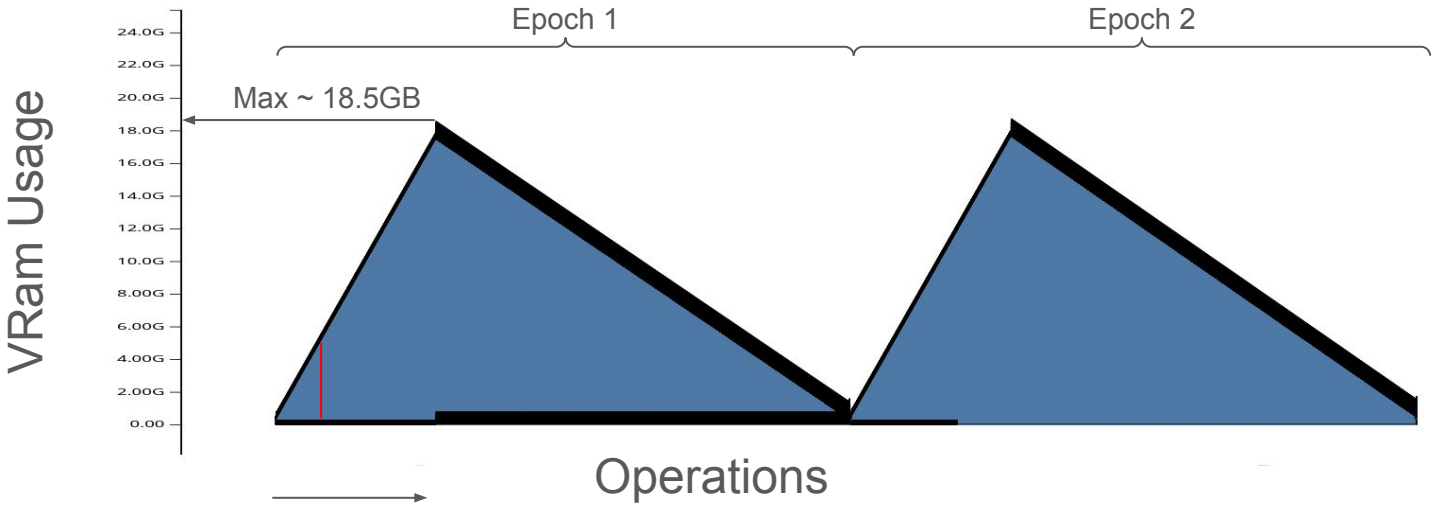


Forward pass – accumulating gradient activations

Resource Utilization – Understanding Memory Profile

Pyramid shape comes from sliding window

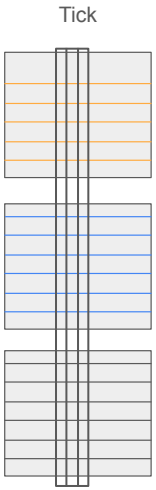


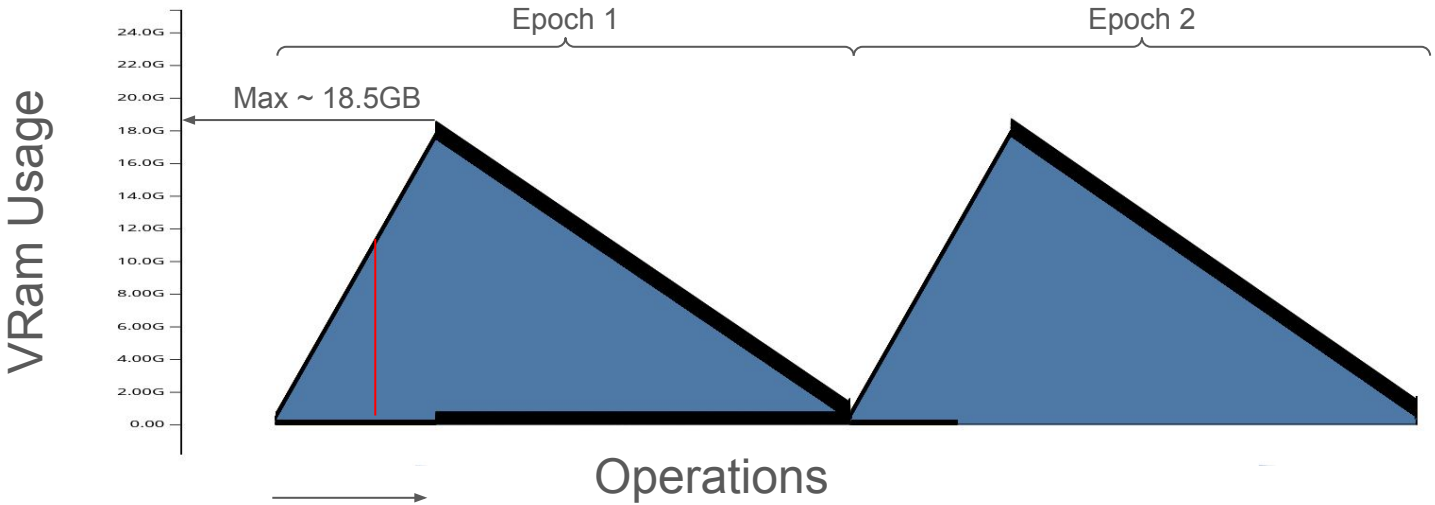


Forward pass – accumulating gradient activations

Resource Utilization – Understanding Memory Profile

Pyramid shape comes from sliding window

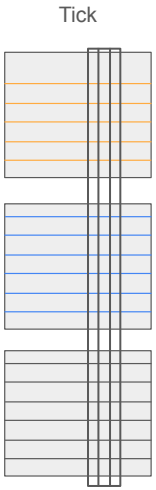


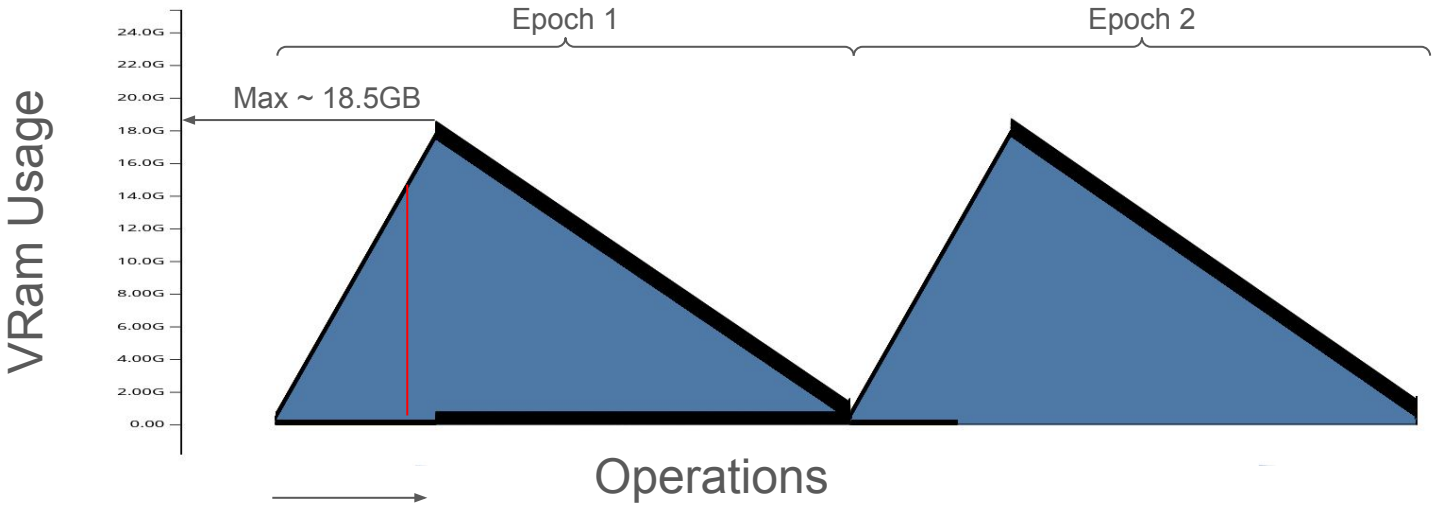


Forward pass – accumulating gradient activations

Resource Utilization – Understanding Memory Profile

Pyramid shape comes from sliding window

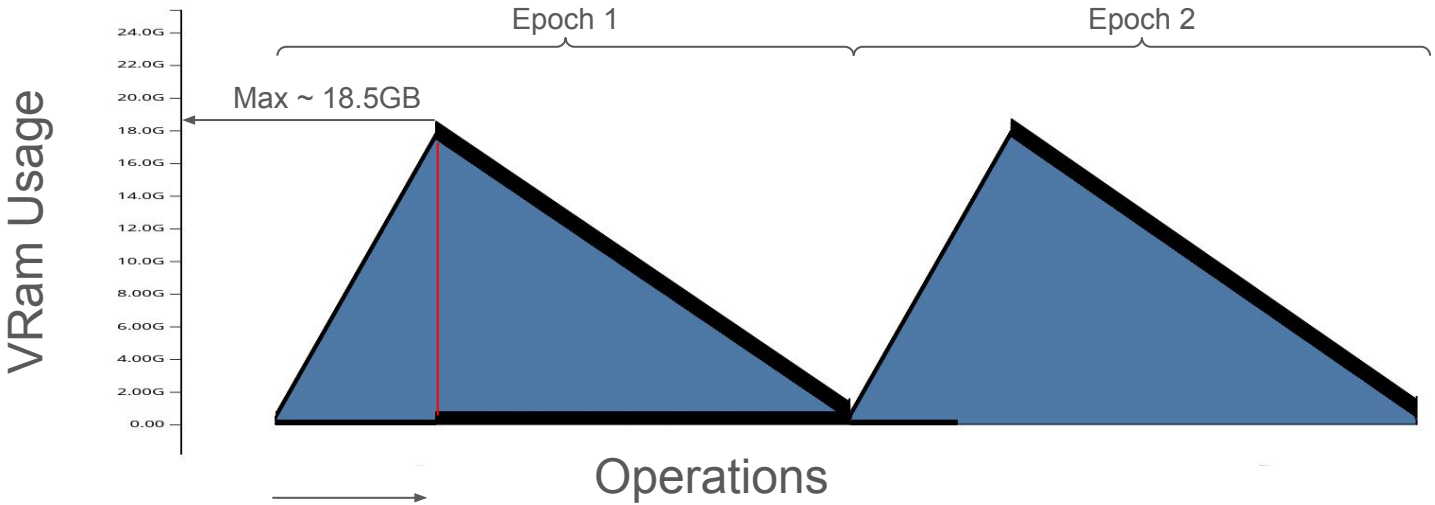




Forward pass – accumulating gradient activations

Resource Utilization – Understanding Memory Profile

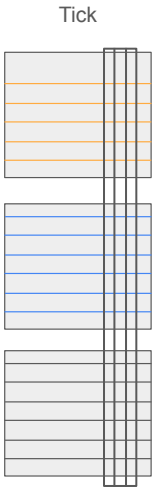
Pyramid shape comes from sliding window

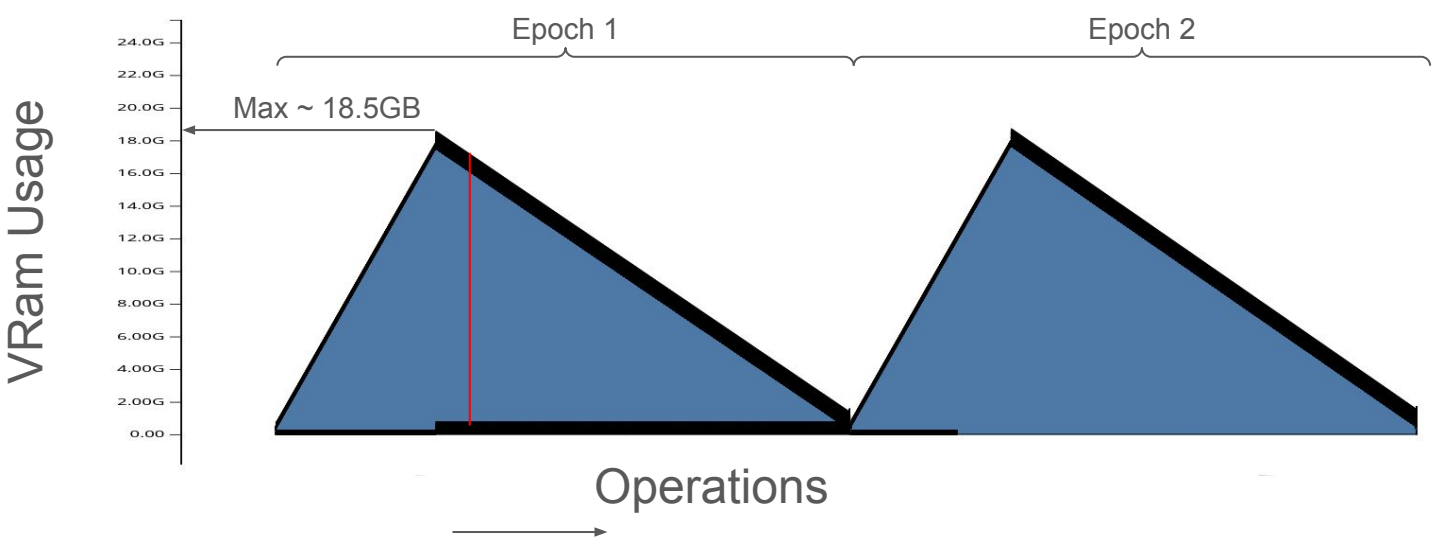


Forward pass – accumulating gradient activations

Resource Utilization – Understanding Memory Profile

Pyramid shape comes from sliding window

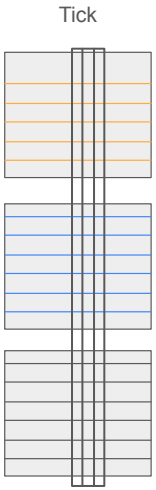


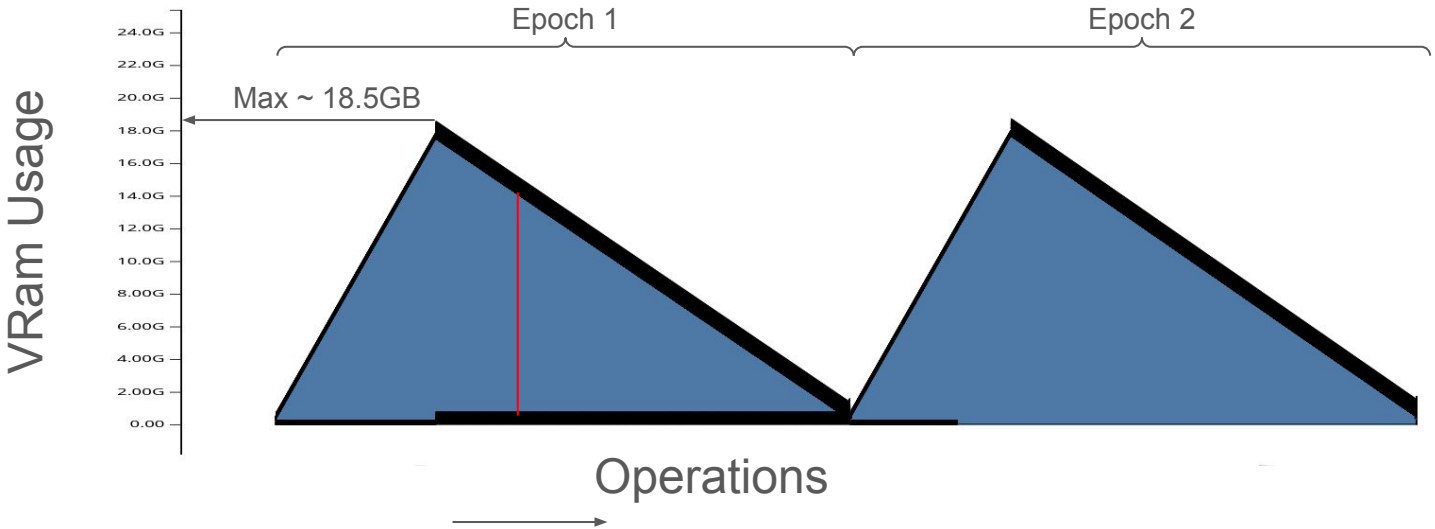


Backward pass – Removes unneeded activations sequentially

Resource Utilization – Understanding Memory Profile

Pyramid shape comes from sliding window

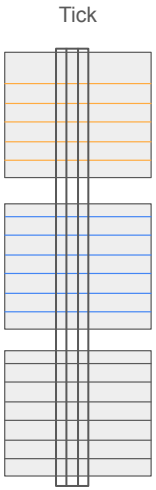


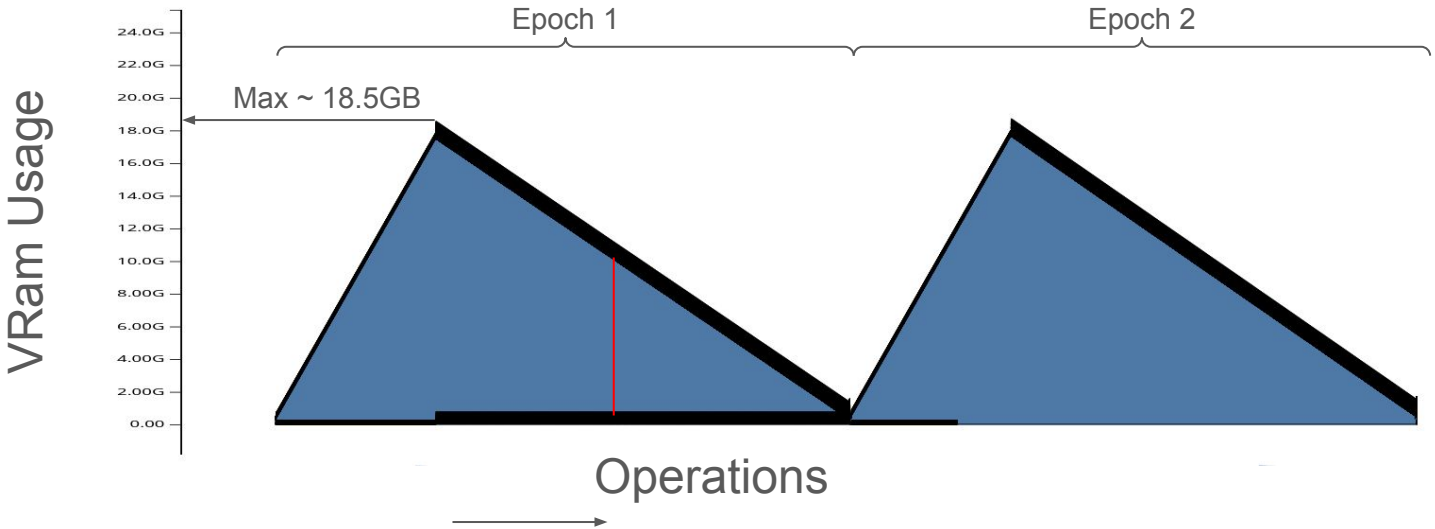


Backward pass – Removes unneeded activations sequentially

Resource Utilization – Understanding Memory Profile

Pyramid shape comes from sliding window

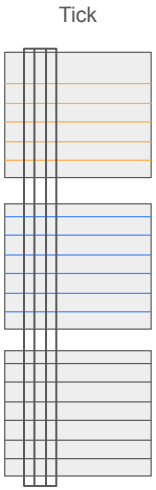


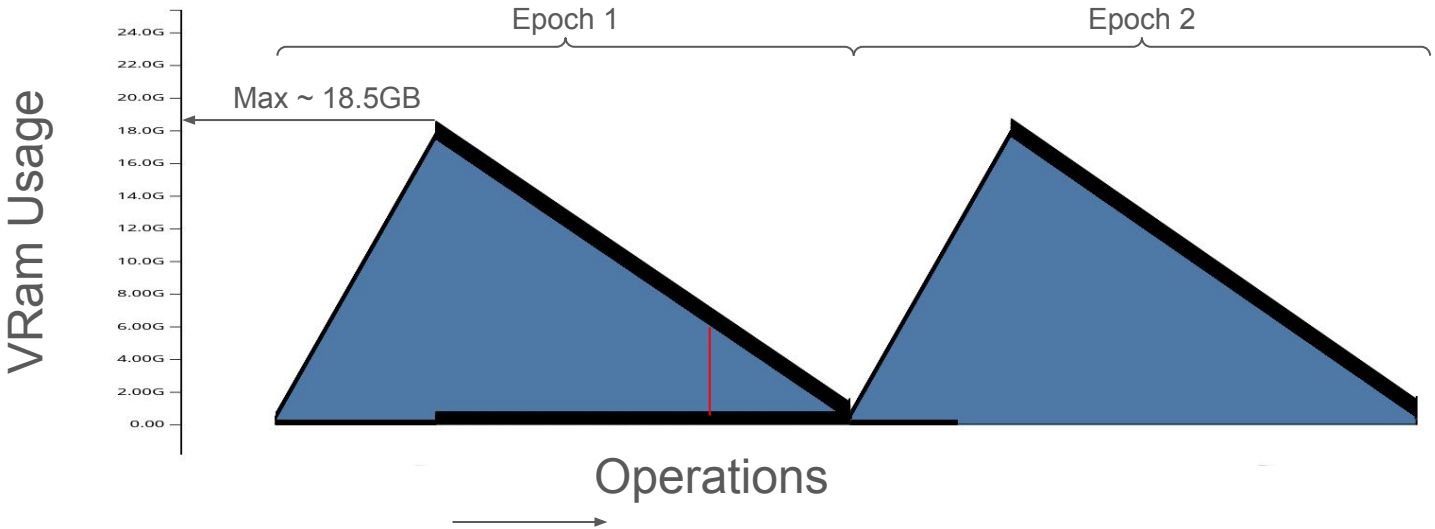


Backward pass – Removes unneeded activations sequentially

Resource Utilization – Understanding Memory Profile

Pyramid shape comes from sliding window

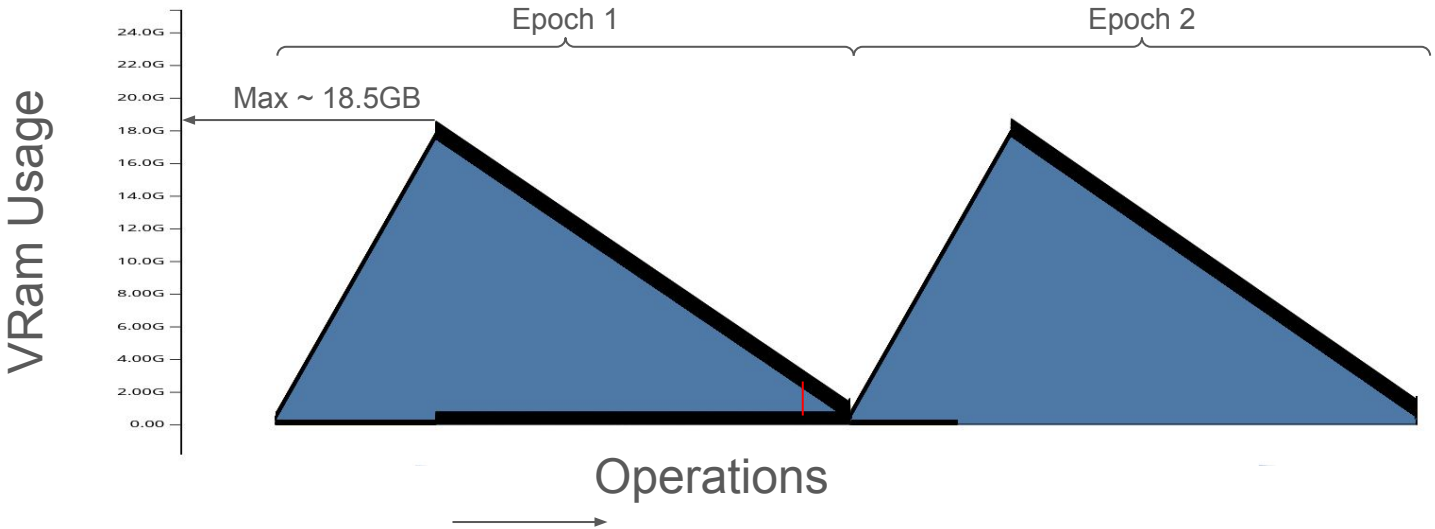




Backward pass – Removes unneeded activations sequentially

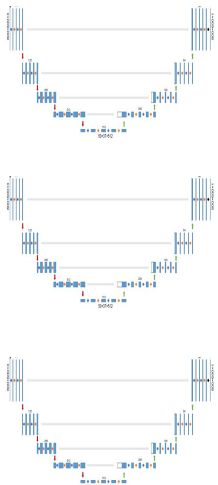
Resource Utilization – Understanding Memory Profile

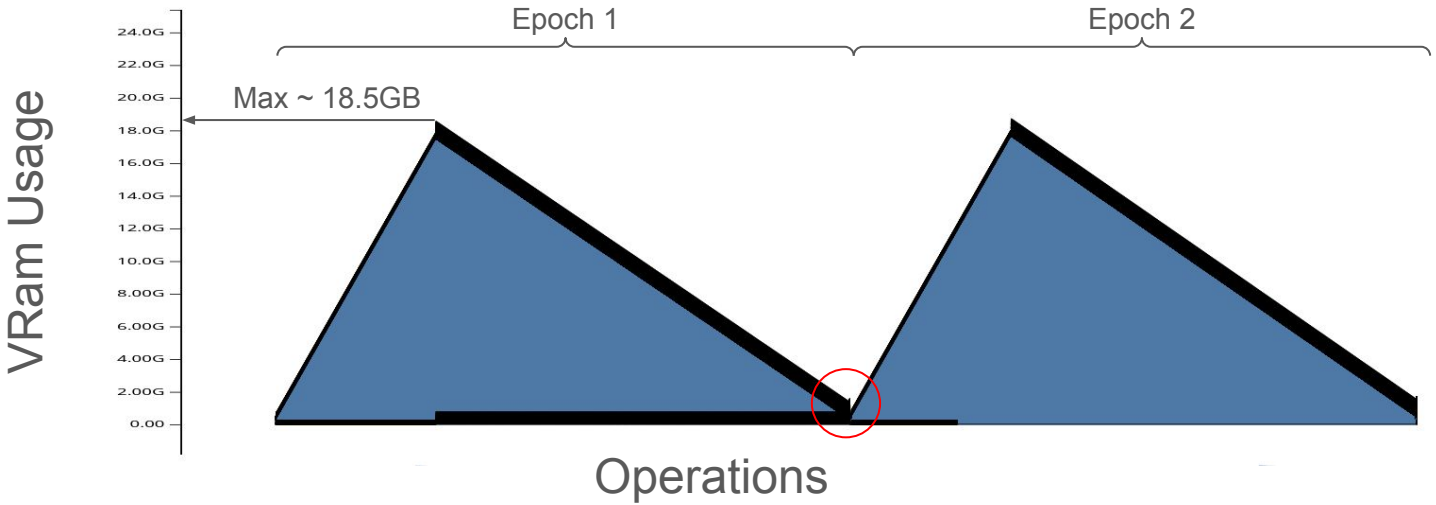
Pyramid shape comes from sliding window



Backward pass – Removes unneeded activations sequentially

Resource Utilization – Understanding Memory Profile





Tiny blip → UNets

- Using gradient checkpointing (recalculates gradients when needed)