# Introduction to deep learning for LArTPCs

Andy Chappell and Leigh Whitehead

31/10/2024

9th UK LArTPC Software and Analysis Workshop

# Introduction

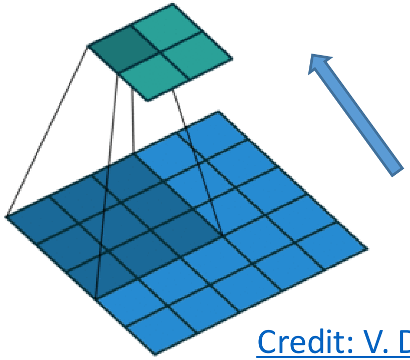


- So far, we've looked predominantly at general deep learning concepts
- Now we'll look at some more specific architectures and application to LArTPCs
  - Convolutions, activations, normalisation and ResNets
  - Introduction to semantic segmentation
  - Pandora's vertex finding network in DUNE

# Convolution and transpose convolution

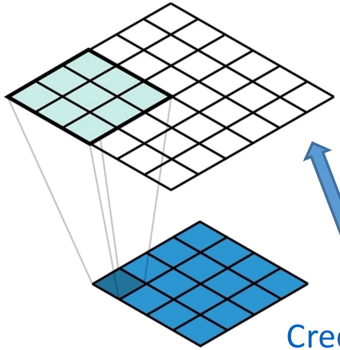


Down-sample



Credit: V. Dumoulin & F. Visin

Up-sample



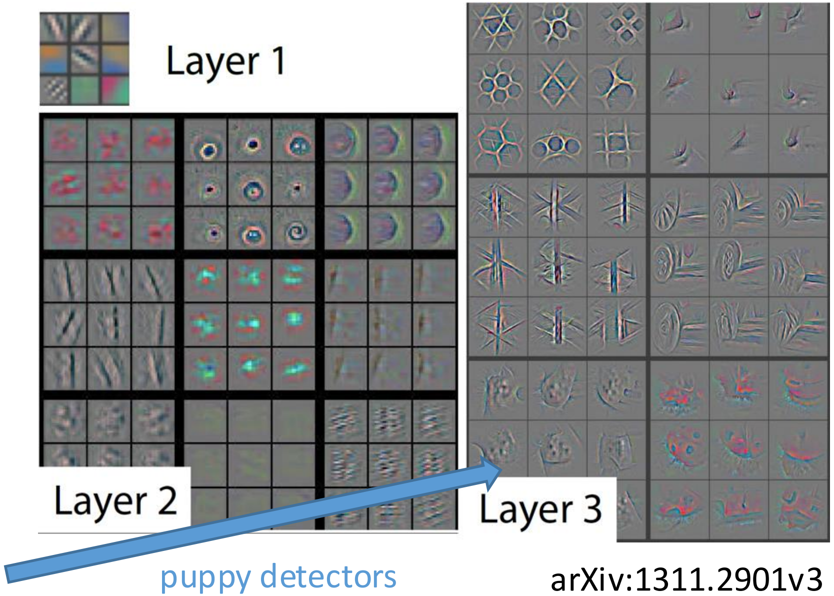
Credit: T. Lane

- Multiple input pixels map to one output pixel
- Each layer increases number of kernels to build more complex features
- Stride 2 (sliding the convolution filter 2 pixels) down-samples to reduce computational overhead

- Each input pixel maps to multiple output pixels
- Effective stride 1/2 up-samples to return to original image size
- Higher-resolution activations from down-sampling layer can then be added to the up-sampled images

# What are these features?

- ResNet18 is one of many pretrained networks available through, e.g., PyTorch torchvision
  - Pretrained on [ImageNet]() – a database of millions of photographs, comprising 1000 classes
  - 11,178,051 trainable parameters (this is the smallest ResNet)
- It's possible to look at which learned convolutional kernels are activated by given inputs
  - Networks learn features at different scales
  - Shallow layers learn the most primitive structures
  - Deep layers learn the most abstract features
  - Those early layers are still relevant in neutrino interactions




puppy detectors

arXiv:1311.2901v3

# Normalisation and activation function



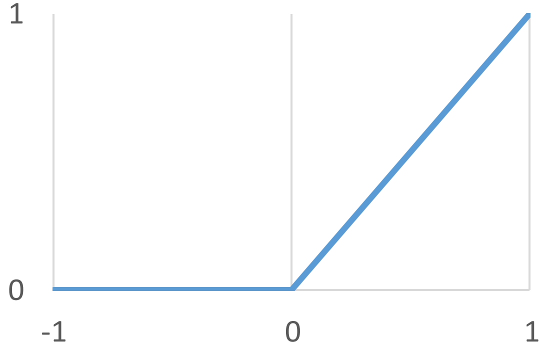
- Why normalise?
  - Input data changes with each batch/epoch, so input distribution can vary and these variations build up deep in the network
  - Small/large gradients can vanish/explode as they are multiplied in deep networks
  - Batch normalisation ensures each batch has zero mean and unit variance, giving consistent input distributions and avoiding gradient problems, but also scales and shifts to avoid loss of representational power
- Why ReLU?
  - Non-linear activations have high representational power
  - It's fast. Simple gradient calculation (0 or 1)
  - Doesn't squash activations with repeated activation (unlike sigmoid)

## Batch normalisation

$$\hat{x}_i = \frac{x_i - \mu_B}{\sigma_B}$$

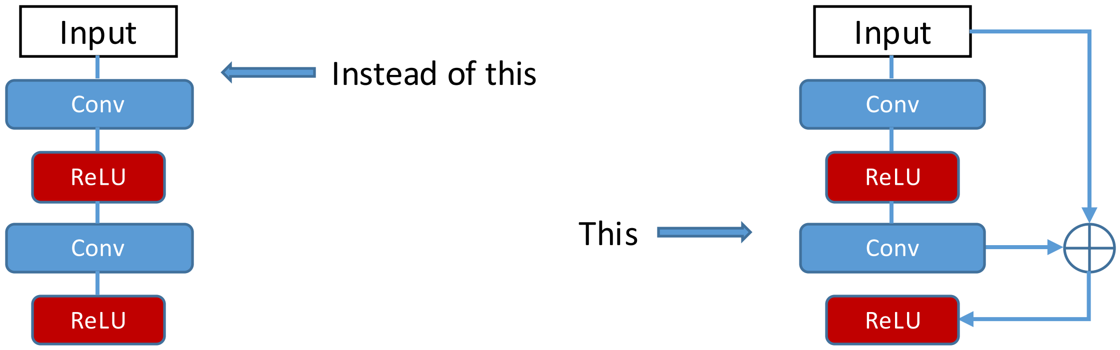
$$y_i = \gamma \hat{x}_i + \beta$$

## Rectified Linear Unit



# ResNet

- We'll now return to take a brief look at probably the most famous CNN for classification, ResNet
- ResNet was introduced in 2015
  - There are a lot of neat ideas introduced in this paper, but the key one is the introduction of the residual (the Res in ResNet) shortcut connection
  - This innovation allowed networks to get much deeper and still train effectively



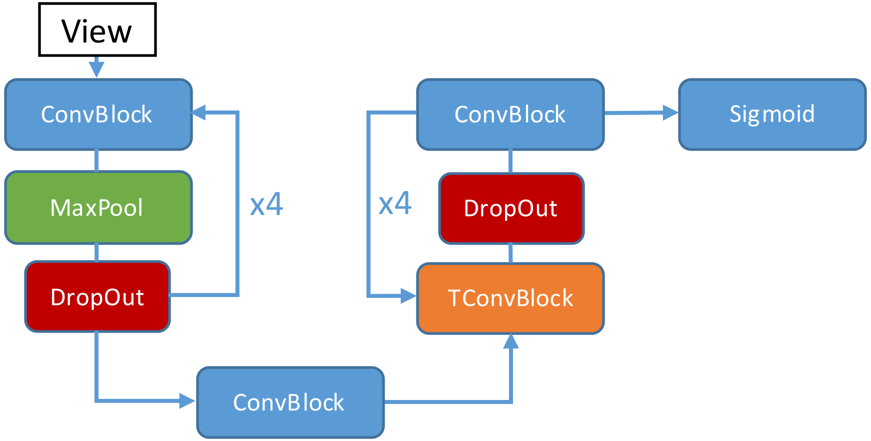

- Instead of learning the mapping from input to output, you learn the residuals that get you from input to output
- e.g, if the optimal mapping is the identity, it's easier to push the residuals to zero than to relearn the identity

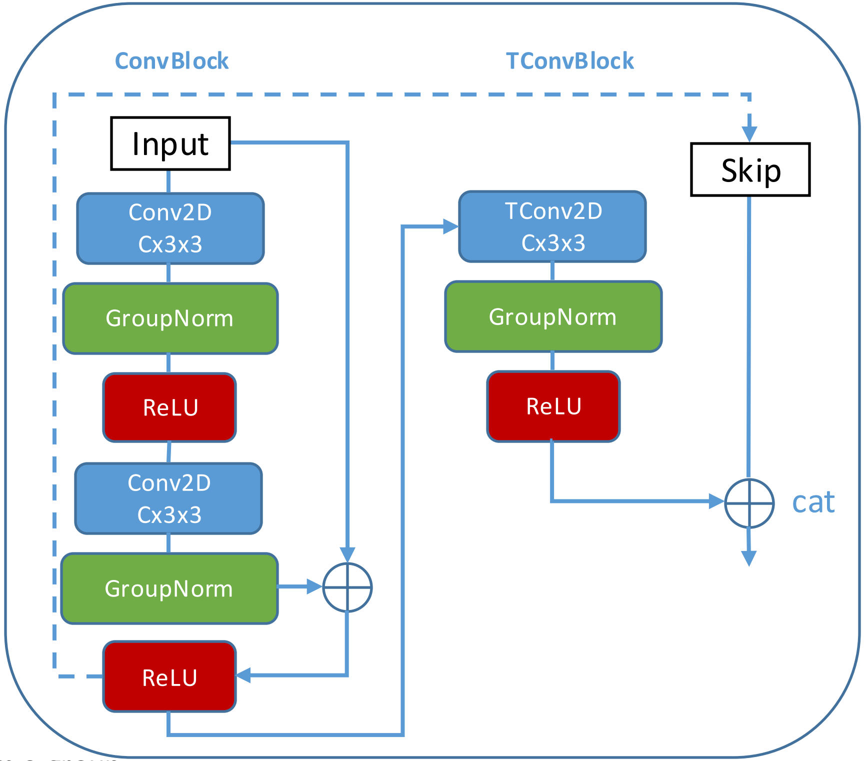
# Semantic segmentation

# U-Nets for semantic segmentation



- U-Net concept introduced in 2015 for biomedical image segmentation
- The name comes from the conceptual structure of the network



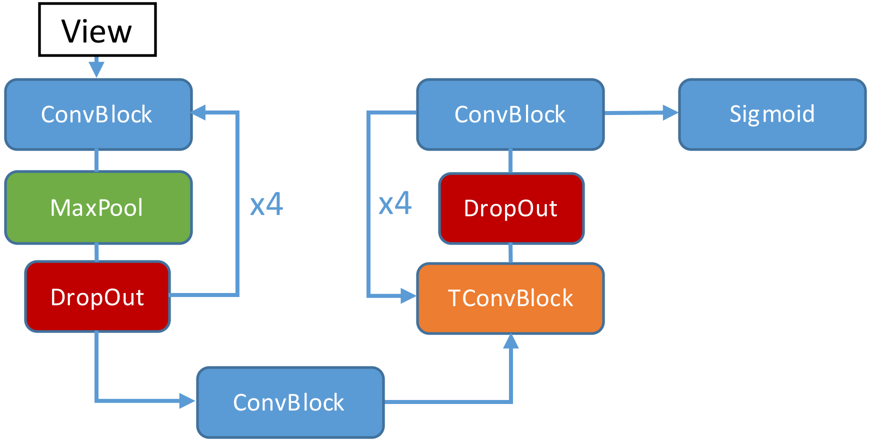



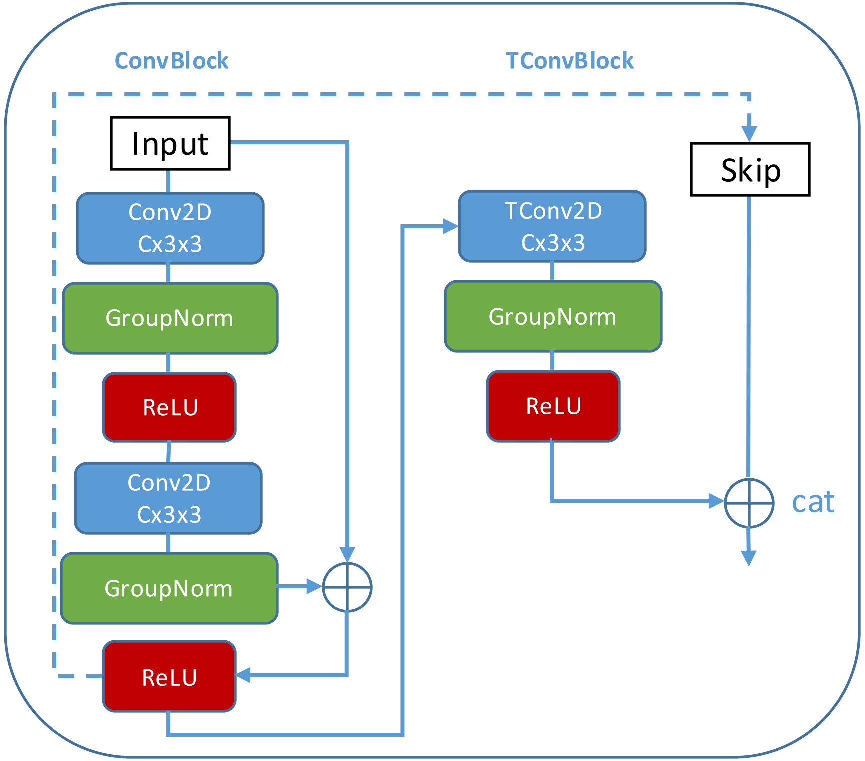

Pooling merges neighbouring pixels: MaxPool picks the largest pixel from a group
DropOut randomly turns off weights during training to reduce over-fitting

# U-Nets for semantic segmentation



- Down-sampling and feature extraction is performed via a Convolutional Neural Network (CNN) in the left arm of the U
- Result of each intermediate convolution block is retained for use in skip connections

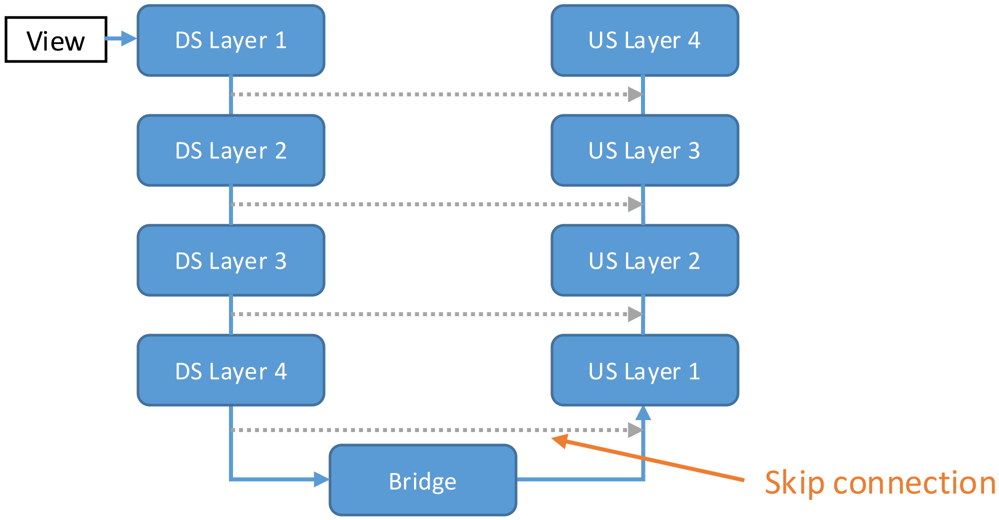


# What are skip connections?



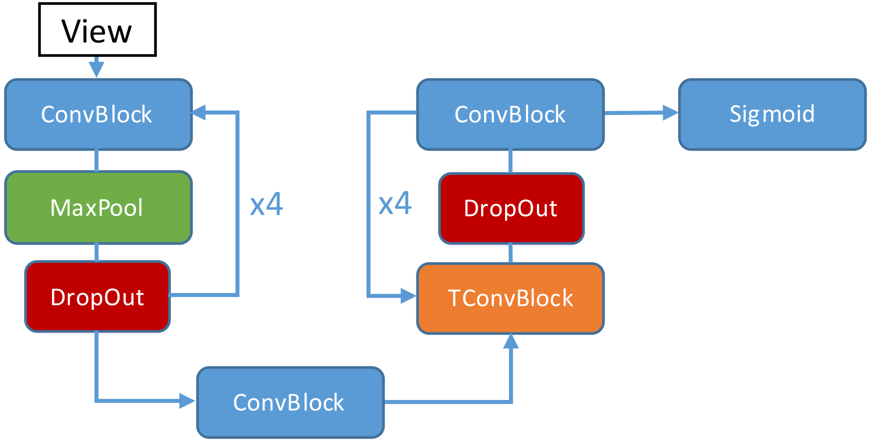
- The final output of a U-Net needs to be the same size as the original input.
- Repeatedly down-sampling means we have to get back to high resolution from very low resolution
- Skip connections provide a means to augment up-sampled images with higher-resolution activations from earlier network layers

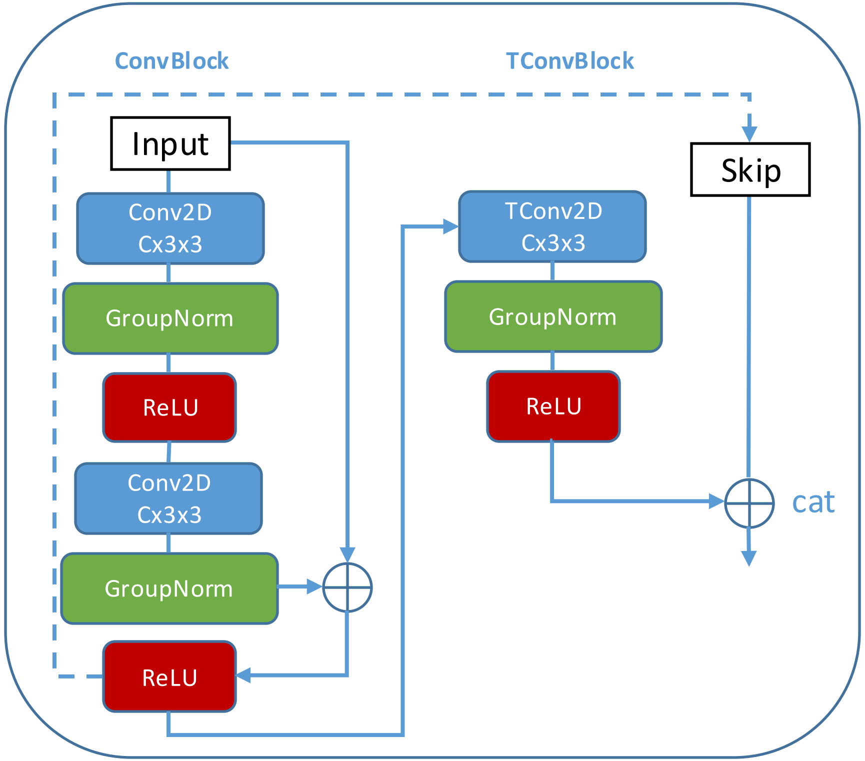


# U-Nets for semantic segmentation



- Up-sampling and image augmentation is performed via transpose convolutions (discussed later) in the right arm of the U

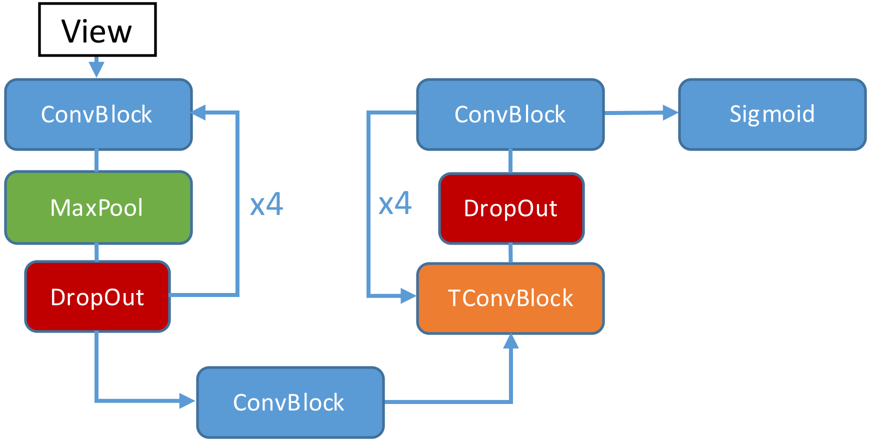


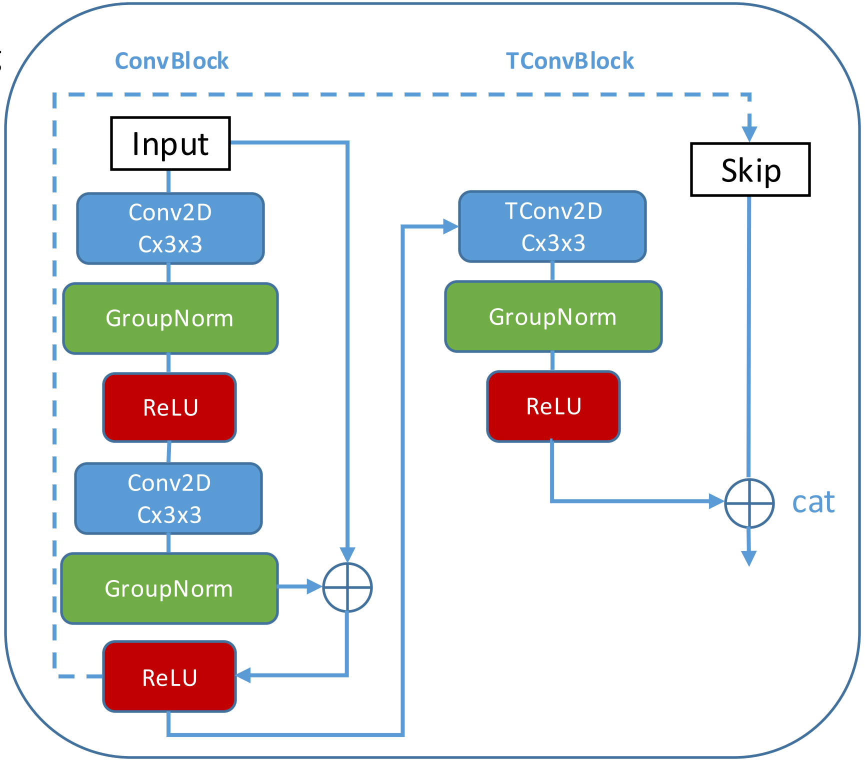




- Intermediate results from down-sampling are added to the up-sampled images via skip connections to “fill in the gaps” from up-sampling

# U-Nets for semantic segmentation

- The base of the U is known as the bridge
  - Performs additional feature extraction
  - Ensures matching tensor sizes between down-sampling and up-sampling arms

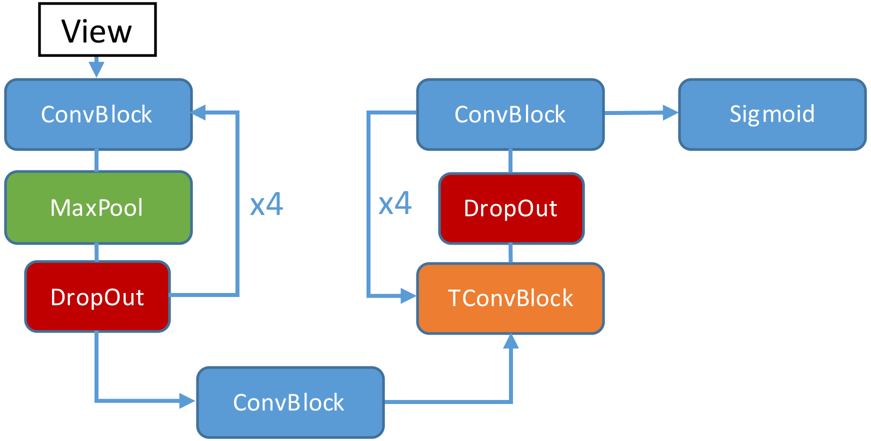


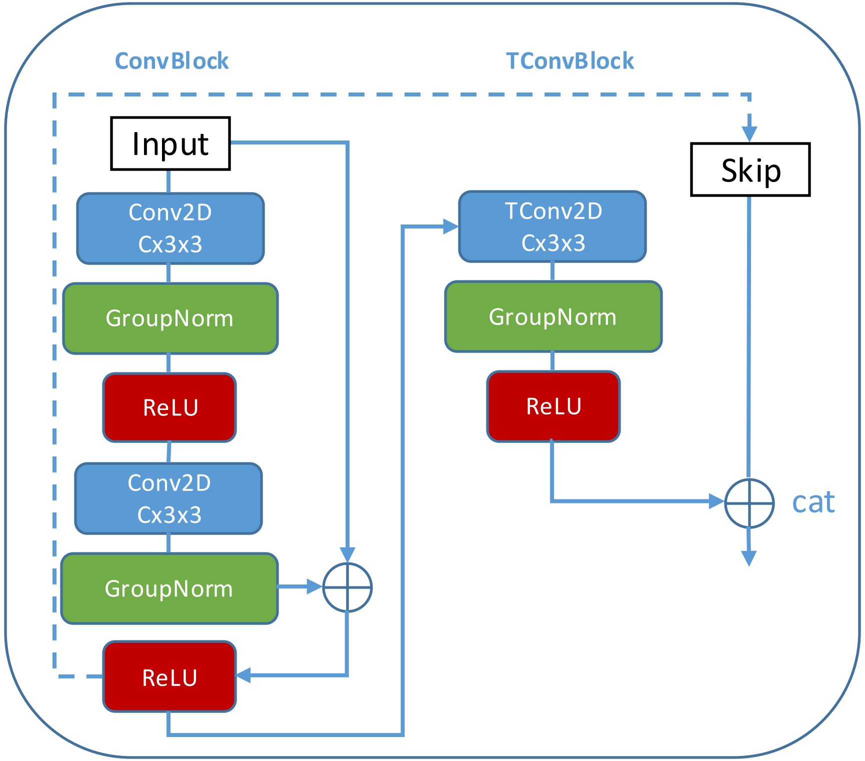



# U-Nets for semantic segmentation



- Key goal of the U structure is to classify **every pixel** from the input image
  - Track versus shower
  - Particle ID
  - …



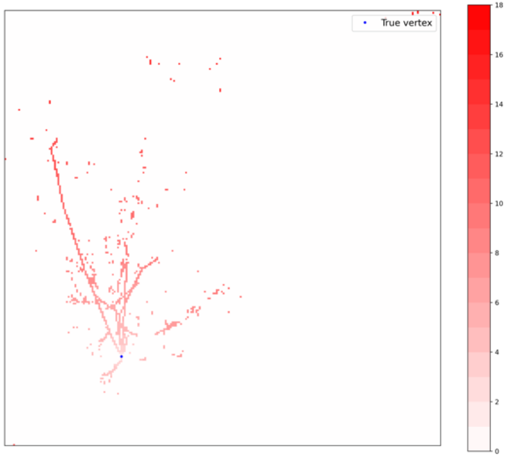



# Pandora’s DL vertex finding

# Using semantic segmentation in Pandora



- Semantic segmentation forms the basis of Pandora's vertex finding algorithm for DUNE
- Why would you use a classification network to find an interaction vertex?
  - Regression for vertex finding in LArTPCs is hard
  - You ask a network to learn a single (or small set of) target location(s) in a complex image
  - Semantic segmentation treats the whole image as a target to learn
- Classify each pixel according to its distance from the estimated vertex location
  - Adjacent pixels are obviously correlated, so context helps learning
- The network doesn't return a vertex location
- How do we extract the vertex?

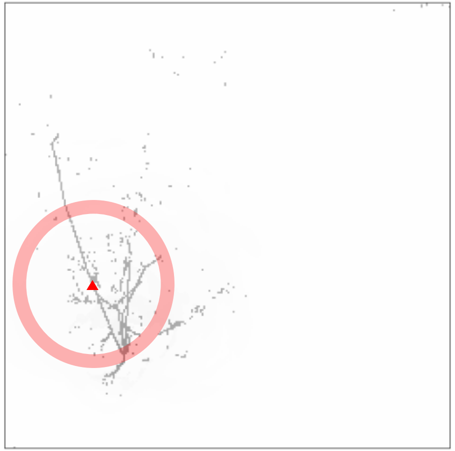



Network classification

# Mixing deep learning with 'traditional' algorithms

- We have a set of distance classes for each occupied pixel
- For each hit, convert the class to the known lower and upper distance bounds
- Draw a ring, centred on the hit with radii corresponding to those distance bounds
- Weight the pixels in the ring inversely proportional to its area
- Vertex could be anywhere within the shaded region of one ring
- Many rings for a heat map, where high weight indicates likely location

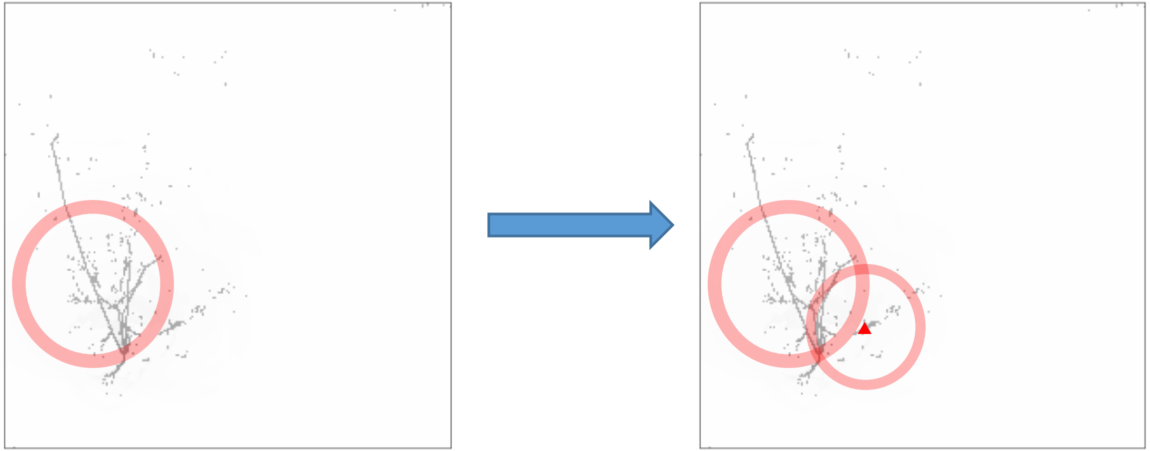
Image from a single wire plane



Heat map from one classified pixel

# Mixing deep learning with 'traditional' algorithms

- We have a set of distance classes for each occupied pixel
- For each hit, convert the class to the known lower and upper distance bounds
- Draw a ring, centred on the hit with radii corresponding to those distance bounds
- Weight the pixels in the ring inversely proportional to its area
- Vertex could be anywhere within the shaded region of one ring
- Many rings for a heat map, where high weight indicates likely location



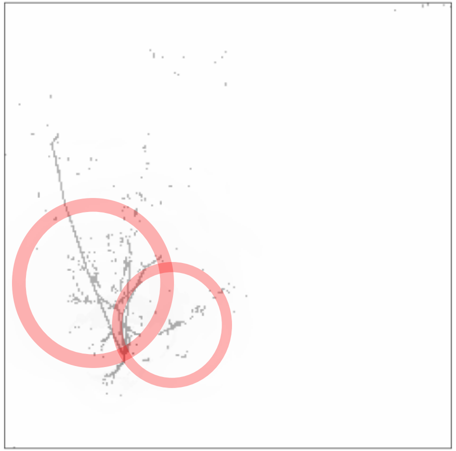
Heat map from one classified pixel

Heat map from two classified pixels

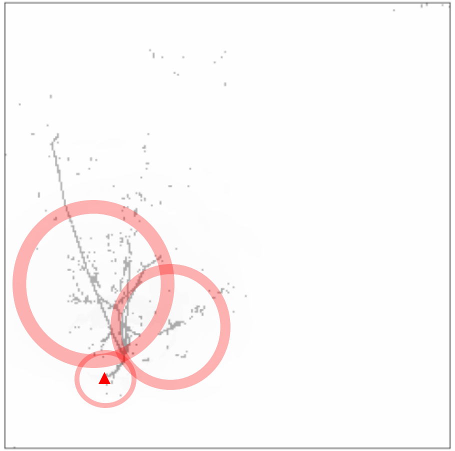
# Mixing deep learning with ‘traditional’ algorithms



- We have a set of distance classes for each occupied pixel
- For each hit, convert the class to the known lower and upper distance bounds
- Draw a ring, centred on the hit with radii corresponding to those distance bounds
- Weight the pixels in the ring inversely proportional to its area
- Vertex could be anywhere within the shaded region of one ring
- Many rings for a heat map, where high weight indicates likely location



Heat map from two classified pixels

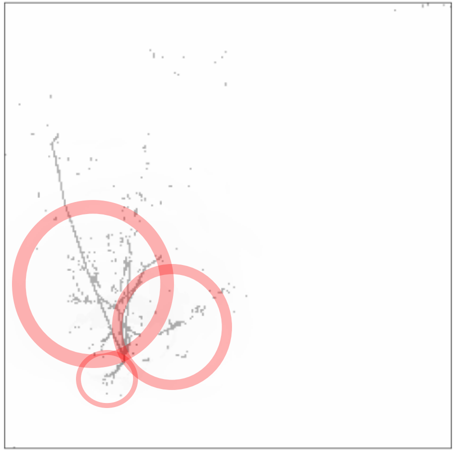


Heat map from three classified pixels

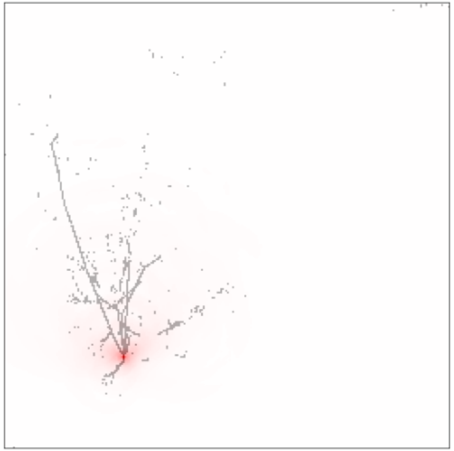
# Mixing deep learning with 'traditional' algorithms



- We have a set of distance classes for each occupied pixel
- For each hit, convert the class to the known lower and upper distance bounds
- Draw a ring, centred on the hit with radii corresponding to those distance bounds
- Weight the pixels in the ring inversely proportional to its area
- Vertex could be anywhere within the shaded region of one ring
- Many rings for a heat map, where high weight indicates likely location



Heat map from 3 classified pixels



Heat map from all classified pixels

# A brief aside on TorchScript



- Currently, LArSoft expects PyTorch networks to be C++-based and CPU-bound for inference
- This will hopefully change in time, but until it does, if you have a deep neural network you'd like to use in, Pandora, for example, you need to know about TorchScript
- Pandora's vertex finding network was trained using Python on GPUs, but you can't run that in Pandora, you need to convert it

```
device = torch.device('cpu')
model = load_model(filename, device)    # custom code to load your specific model
sm = torch.jit.script(model)
sm.save(output_filename)
```

- TorchScript can take a model defined using standard PyTorch code and convert it to a format that can be run on a CPU
- Such a network can now be used in Pandora (you'll need to manage the inputs and outputs of course, but we won't cover that today)

# Getting some practical experience

- It’s now time to build a CNN to classify some neutrino interactions…