Semantic Segmentation of 3D Particle Interaction Data Using Fully Convolutional DenseNet

Ji Won Park (jwp), Aldo Carranza (aldogael), Zhilin Jiang (zjiang23)

Abstract

Semantic segmentation has found applications in fields where it is necessary to localize an abstract feature by labeling each pixel in an image as belonging to one of a predefined set of classes. It most often employs a deep convolutional neural network architecture consisting of a downsampling path, which extracts high-level features from the image, and an upsampling path, which retrieves the resolution of the original image for pixel-wise classification. In this paper, we present the results of applying a semantic segmentation network to the analysis of particle interaction images derived from a novel detector technology, called the Liquid Argon Time Projection Chamber (LArTPC). Whereas it boasts of high neutrino detection efficiency and low background, the analysis of LArTPC-derived images has posed a challenge. In particular, removing cosmic ray-induced particles and reconstructing neutrino interactions from the images have required a combination of many algorithm-based frameworks. Deep neural nets, on the other hand, can provide an efficient end-to-end solution. We train a fully convolutional densenet (FC-DenseNet) [1] to perform pattern recognition in 3D – specifically, the pixel-wise classification of showers and tracks – on a simulation dataset. Our work will inform the analysis of future LArTPC experiments such as the Deep Underground Neutrino Experiment (DUNE).

1 Introduction

Deep neural networks (DNN), with their ability to extract complex high-level information from raw input data, have led to great strides in computer vision and pattern recognition. In this paper, we report on the current status of applying DNN to the analysis R&D for particle physics experiments. Specifically, we explore experiments that employ a novel detector technology, called the Liquid Argon Time Projection Chamber (LArTPC). A LArTPC consists of a volume of liquid argon in a uniform electric field and a set of wire planes at the anode end to collect the drifting ionization electrons. The induced signals from the wires can be used to reconstruct the charged particle trajectories with millimeter-scale spatial resolution and calorimetric information. See Figure 1 for an illustration.

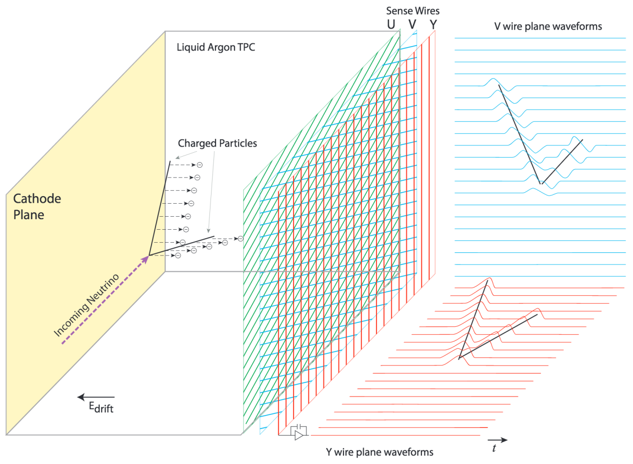

Figure 1: Schematic of LArTPC detector [14]

Analyzing data from LArTPCs poses a challenge, however. It has required a combination of many algorithm-based frameworks in removing cosmic ray-induced particles and reconstructing neutrino interactions. DNN is a promising solution to this problem as it can combine many steps of the analysis. We present the results of training a DNN for 3D pattern recognition on a simulated dataset. The task is to label each pixel of an input image as belonging to a track (a straight line) or an electromagnetic shower (a scattering burst). Such pixelwise classification is important to the bigger analysis scheme, as tracks can serve as a model for muons or protons and showers for single electrons. See Figure 2 for examples of input data and label.

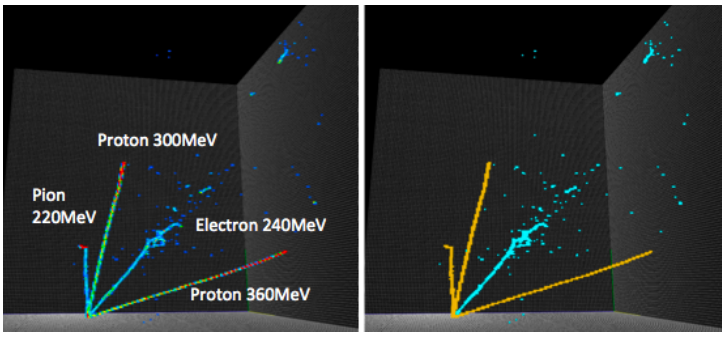

Figure 2: An example of the input data (left) with the truth identity and energy indicated next to each particle, and input label (right) with track pixels colored in yellow and shower pixels colored in cyan. These images were visualized using an open-source viewer called LArCV Viewer.[9]

The architecture we use is a fully convolutional densely connected network (FC-DenseNet) [1], which applies the idea of Fully Convolutional Networks (FCNs) [2] to a type of convolutional neural network (CNN) called Dense Convolutional Network (DenseNet) [3]. FCNs take the form of regular CNNs followed by an upsampling path, also known as a decoding path, which retrieves the original resolution of the image for pixelwise labeling. The CNN portion of FCNs then constitute the downsampling path, or the encoding path, which has the role of expanding the field of view of the algorithm and extracting the abstract features.

Our baseline model is another FCN which uses ResNet modules, which we call U-ResNet.[4][5][6] The naming comes from a public code repository we use.[7] The main difference between U-ResNet and FC-DenseNet lies in how the constituent CNN modules (i.e. ResNet and DenseNet, respectively) reinforce connections within the semantic segmentation network. The former makes use of a residual block in which feature maps are summed together.[6] The latter has dense blocks in which all the previous feature maps are concatenated iteratively. FC-DenseNet has been known to be more parameter-efficient, as keeps the old and new information separate, as opposed to giving each layer a new set of weights.[3]

2 Dataset and Features

We use supervised learning, so our input data is a pair of training data and truth labels. Our input data is 3D so we use the term "voxel" to refer to the basic unit of a 3D image, i.e. the 3D-equivalent of "pixel." The data is a simulated 3D-voxelized image of particle interactions as would be observed by a LArTPC detector. They have been generated in LArSoft[8], a software that can simulate LArTPC-derived images. We gave two constraints to the simulations. First, each image had randomized particle multiplicity of $1 \sim 4$ from a unique vertex, where the $1 \sim 4$ particles were chosen randomly from five particle classes: electron, proton, muon, pion, and gamma ray. The vertex is defined as a a single point from which particle tracks or showers begin. Second, we allowed the momentum to vary from 100MeV to 1GeV in isotropic direction. The constraints were designed so that all images have a clearly-defined vertex plus shower and/or track from which the network could learn, instead of empty voxels or cropped particles. The image resolution was 128 x 128 x 128 voxels. This corresponds to a physical size of $\sim 1\text{cm}^3$ per voxel. Each voxel of the training data contains the charge information and is therefore grayscale (one channel representing charge), whereas each voxel of the truth labels takes the value of 0 (for background), 1 (for track), and 2 (for shower). There were 9,999 images in our training set and 20 images in our test set.

The production of the simulated dataset and their visualization were possible thanks to the open-source repositories LArCV2[9] and LArCV Viewer[10]. LArCV2 is a "data processing framework for LArTPC-derived images" and was "developed to interface [the] LArTPC experiment data to deep neural network frameworks."[9]

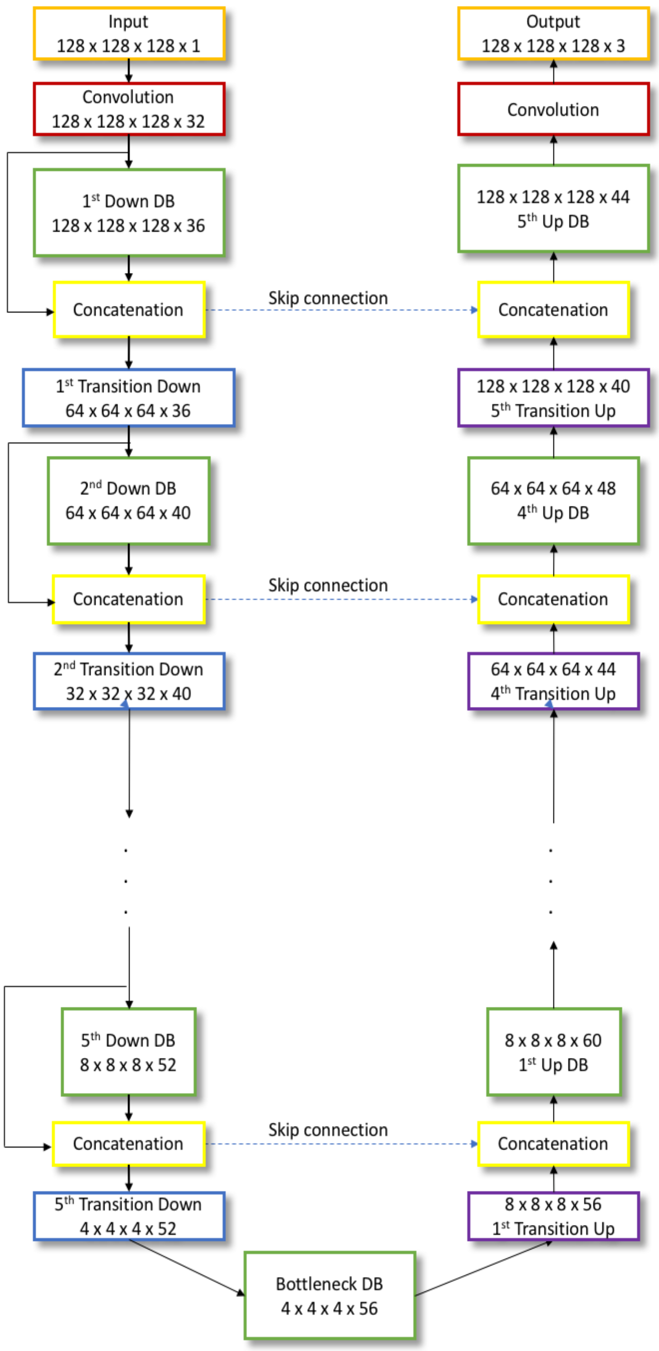

Figure 3: Architecture of Fully Convolutional DenseNet

3 Methods

3.1 Optimization Objective

We define our loss function as the sum of the log weighted softmax probability over the pixels, which is then averaged over the images in the batch, i.e.

$$\mathcal{L} = \frac{1}{N} \sum_{i}^{\text{examples}} \sum_{j}^{\text{pixels}} w_j^{(i)} softmax(z_j^{(i)}) \tag{1}$$

where N is the batch size, w_j is the weighting for each pixel, and z_j is the network output. The superscript indicates indexing across examples (images) in the training set. The weighting is not trivial, and deserves elaboration. We weight each voxel by the inverse of the number of voxels belonging to the *instance* of that voxel. For instance, if a voxel j belongs to the shortest (leftmost) track in the images on Figure 2, $w_j = 1/(\text{total number of voxels in short track})$. Note that this weighting scheme differentiates between instances of the same class. That is, a voxel in one track is weighed differently from a voxel in another track although they both belong to the same class, track.

3.2 Evaluation Metric

We define "all accuracy" as the fraction of correctly classified voxels over all voxels. But a more useful measure of the algorithm's performance is "nonzero accuracy" which computes the fraction only out of nonzero, or non-background, voxels. This is because, in a typical image, 99.99% of voxels are background so the algorithm can receive a high score simply by guessing that all voxels are background. We do not use mean IOU (intersection over union) although it is a common metric for semantic segmentation problems, because we do not have to penalize false positives. Cases where the network classifies a background voxel as track or shower present no problem, as we know which voxels are background simply by observing the charge on the training data. The reason we feed background voxels into the network at all is so that it can learn the boundaries between shower or track voxels and the background.

3.3 Network Architecture

The network architecture of FC-DenseNet consists of a downsampling path that extracts the semantic features of an image followed by an upsampling path that uses these semantic features and data from previous downsampling layers to build an image up to the resolution of the input data. Refer to Figure 3 for visualization of the network architecture of FC-DenseNet. The paths are built from modules known as *dense blocks* that each build a feature map through concatenation of iteratively generated intermediary feature maps. The feature maps then go through a *transition down* operation to reduce the spatial dimensionality of the feature maps via 1×1 convolution operations followed by 2×2 pooling operations. The entire process makes up a module whose output can be fed into another module to further reduce dimensionality of the data. In our implementation, we considered a 5-module architecture.

From the output of the final downsampling dense block, known as the *bottleneck dense block*, we must build up the spatial resolution of the data in the upsampling phase to be able to semantically segment the images at the input resolution. FC-DenseNet introduces *skip connections* between downsampling and upsampling paths to achieve this compensation of data loss from the concatenation layers. Skip connections allow us to concatenate feature maps generated in the downsampling path to build feature maps in the upsampling phase that help us recover the input spatial resolution. This concatenation is performed in the *transition up* module. After building up the spatial resolution with extra dimensions for each potential class, the output is fed into a softmax layer for classification.

3.4 Training Configurations

As is often the case with training on 3D data, the GPU memory limited our batch size. Our batch size for FC-DenseNet was 1, which was the most we fit into our GPU. We judged that this was fine since the batch size for U-ResNet, which trained successfully, was 8. We decreased learning rate by a factor of 10 when loss flattened, from the default value of 0.001 to 0.0001. We did not have a criterion for convergence and stopped training when loss flattened.

4 Results

We report the results of training U-ResNet and FC-DenseNet. Figure 4 shows the training loss and classification accuracy of the the non-background voxels over the number of iterations. Table 1 lists the training accuracy of both U-ResNet and FC-DenseNet.

Metric	U-ResNet	FC-DenseNet
Total accuracy	0.9916	0.9946
Non-background accuracy	0.9753	0.9002
Loss	0.0239	0.2009
Images consumed	17779×8 (≈14 epochs)	21299 (≈2 epochs)
Training time	15 hours	9 hours

Table 1: Final training accuracy/loss of U-ResNet trained on a Titan X (Pascal) with batch size 8 using 11.5GB of memory and final training accuracy/loss of FC-DenseNet trained on a Titan X (Pascal) with batch size 1 using 11.7GB of memory.

5 Discussion and Ongoing Work

5.1 Batch size

The batch size choice of 1 for training the FC-DenseNet was due to the fact that the number of filters increased quickly throughout the network. U-ResNet, on the other hand, did not require as many filters and could accommodate 8 images per batch. Note that the batch size is the size of our ensemble. Only exposing the algorithm to one image per iteration exposed our FC-DenseNet training to statistical fluctuations. U-ResNet updated parameters based on 8 images and thus could learn better and faster. To solve the batch size problem, we implemented minibatching to work with our data loader, available as a separate branch called "minibatch" in our GitHub code repository[11]; in a minibatch setting, gradients are accumulated (summed) over a `minibatch_size` number of images and updated only once every `minibatch_size` images. Also, we have

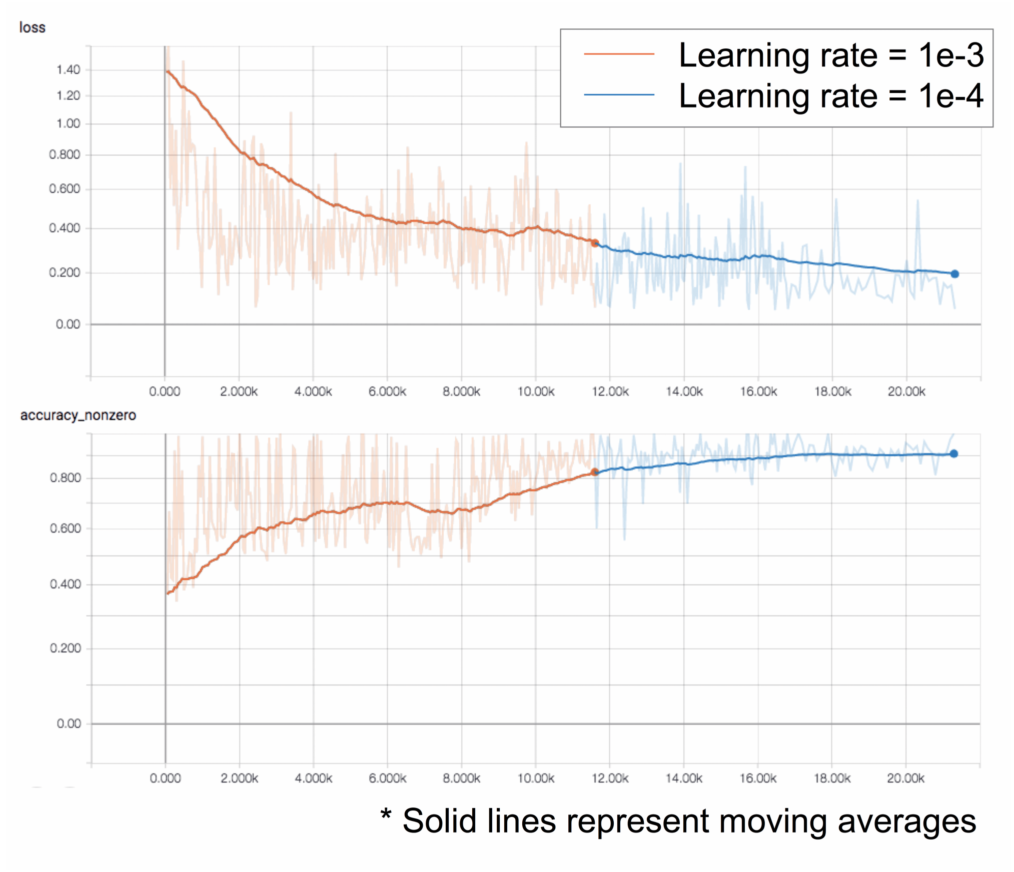

Figure 4: Training metrics

prioritized the depth of the network over the batch size, but we can theoretically (even without minibatching) accommodate up to 2 images per batch at the expense of making the network shallower. Given the GPU memory constraint, we have experimented with different depths and base number of channels and determined that the current set of hyperparameters with 32 base channels with batch size 1 allowed the network to learn best and fastest.

5.2 Learning patterns

Across the training trials we ran to obtain the optimal training configuration, we observed a consistent pattern which we report here. For U-ResNet, all accuracy shot up from the beginning to near 1 during which nonzero accuracy was near 0, and nonzero accuracy slowly increased afterward. That is, the network initially focused on learning the background pixels almost perfectly and only after switched to learning the nonzero pixels for track/shower classification. This learning pattern was observed every run across 7 training runs. For FC-DenseNet, on the other hand, all accuracy gradually increased *along with* nonzero accuracy. This pattern was observed every run except one (when the network did not learn at all) across 5 training runs. We have yet to discover exactly what difference of the architecture causes this difference in the learning pattern. Looking at the activation maps at each layer of the network did not prove helpful.

5.3 Weighting the loss

As mentioned in Section 3.2, most (99.99%) of the voxels in each image are background. This means we want to weight the non-background voxels against the background ones, so that the loss contribution from background and nonbackground voxels become comparable. Before adopting the instance-based weighting scheme introduced in Section 3.1, we experimented with other schemes, namely background vs. non-background and track vs. shower vs. background (class-based). But the current scheme (inspired by the existing U-ResNet codeset[7]) was the only one that could accommodate images where two instances of the same class, e.g. short and long tracks, had a significantly different voxel count, in which case we wanted the network to devote particular care to classify the short-track voxels correctly.

6 Conclusion and Future Work

We have reported on the results of training a FC-DenseNet on 3D simulated data of particle interactions. Although we achieve good results, we have performed poorer than our baseline model, U-ResNet. In addition, we see that determination of the hyperparameters such as the total number of layers, evolution of the number of channels through out the network, batch size, and weights for the loss requires careful consideration.

One way to alleviate the problem of small batch size is, as mentioned in Section 5.1, using minibatches. But because most of the voxels have zero value in our data, we can take advantage of sparse matrix storage and processing techniques.

In standard upsampling methods, independent kernels in the deconvolutional layers are applied to the input image to generate intermediate feature maps, so there is no direct relationship between the feature maps. Thus, when they are periodically shuffled to produce the output feature map, the values of neighboring pixels can differ significantly–which is one cause of checkerboard artifacts. The FC-DenseNet method attempts to fix this problem by adding dependencies in the feature maps generated in the dense blocks. Another method that has been proposed to solve this in the PixelDCN method presented in[12]. PixelDCN adds dependencies within the same feature map by a convolutional process on the feature map by generating a sequence of dependent feature maps and combining them to create one feature map. This allows sharing of parameters to add more spatial dependencies and smoother results. In future work, we can attempt to improve the quality of semantic segmentation of path results by incorporating the smoothing effects promised by the PixelDCL method.

7 Acknowledgement

We thank Kazuhiro Terao, Associate Scientist at the Elementary Particle Physics Group at SLAC National Accelerator Laboratory, for making the GPU resources, the U-ResNet code repository, and the 3D particle dataset available for this project.

References

[1] Jégou, Simon, et al. ”The one hundred layers tiramisu: Fully convolutional DenseNets for semantic segmentation.” arXiv preprint arXiv:1611.09326 (2016).

[2] Long, Jonathan, Evan Shelhamer, and Trevor Darrell. “Fully convolutional networks for semantic segmentation.” Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015.

[3] Huang, Gao, et al. “Densely connected convolutional networks.” arXiv preprint arXiv:1608.06993 (2016).

[4] He, Kaiming, et al. “Deep residual learning for image recognition.” Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[5] Chen, Liang-Chieh, et al. “Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs.” arXiv preprint arXiv:1606.00915 (2016).

[6] Drozdzal, Michal, et al. “The importance of skip connections in biomedical image segmentation.” International Workshop on Large-Scale Annotation of Biomedical Data and Expert Label Synthesis. Springer International Publishing, 2016.

[7] Deep Learning Physics, u-resnet, (2017), GitHub repository. Link

[8] Church, Eric D. ”LArSoft: a software package for liquid argon time projection drift chambers.” arXiv preprint arXiv:1311.6774 (2013).

[9] Deep Learning Physics, larcv (2017), GitHub repository, Link

[10] Deep Learning Physics, larcv-viewer, (2017), GitHub repository. Link

[11] jiwoncpark, fc_densenet, (2017), GitHub repository. Link

[12] Gao, Hongyang, et al. *Pixel Deconvolutional Networks.* arXiv preprint arXiv:1705.06820 (2017).

[13] Terao, Kazuhiro. (2017). Deep Learning Techniques in MicroBooNE LArTPC Detector. Retrieved from Link

[14] Abratenko, P., et al. “Determination of muon momentum in the MicroBooNE LArTPC using an improved model of multiple Coulomb scattering.” *arXiv preprint* arXiv:1703.06187 (2017).

[15] P.W.D. Charles, Project Title, (2013), GitHub repository, https://github.com/charlespwd/project-title

[16] Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. *U-net: Convolutional networks for biomedical image segmentation.* International Conference on Medical Image Computing and Computer-Assisted Intervention. Springer, Cham, 2015.

[17] Zeiler, Matthew D., et al. *Deconvolutional networks.* Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on. IEEE, 2010.

[18] Zeiler, Matthew D., Graham W. Taylor, and Rob Fergus. *Adaptive deconvolutional networks for mid and high level feature learning.* Computer Vision (ICCV), 2011 IEEE International Conference on. IEEE, 2011.

[19] Kaiming He et al. *Deep Residual Learning for Image Recognition.* The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016

[20] SimJeg, FC-Densenet, (2017), GitHub repository. Link

[21] Genty, Vic. (2016). Deep Learning MicroBooNE [Powerpoint slides]. Retrieved from Link

[22] R.Acciari et al. *Convolutional Neural Networks Applied to Neutrino Events in a Liquid Argon Time Projection Chamber.* arXiv:1611.05531 [physics.ins-det]