Statistics for Particle Physicists
LPC 2021, Fermilab

Lecture 15: Introduction to Machine Learning – Part 2

Harrison B. Prosper

Department of Physics, Florida State University

29 November, 2021

Outline

Recap

- Last time, we introduced decision trees and noted that it is possible to average over the outputs of many decision trees to form highly effective models $f(x, \theta)$ for classification.

- We also considered the simplest class of neural network model, namely, deep feed forward neural networks.

- We also drew attention to an important mathematical result, namely, that the quantities that determine what a machine model approximates is the loss function and the probability distribution of the data $p(y, x)\, dx\, dy$. The details of the model are important only insofar as

Outline

Today, we shall review two neural network models:

- Convolutional neural networks (CNN) – models for processing data in 2D and 3D arrays, such as images.
- Graph neural networks (GNN) – models for processing data structured as graphs, that is, vertices connected by lines.

Outline

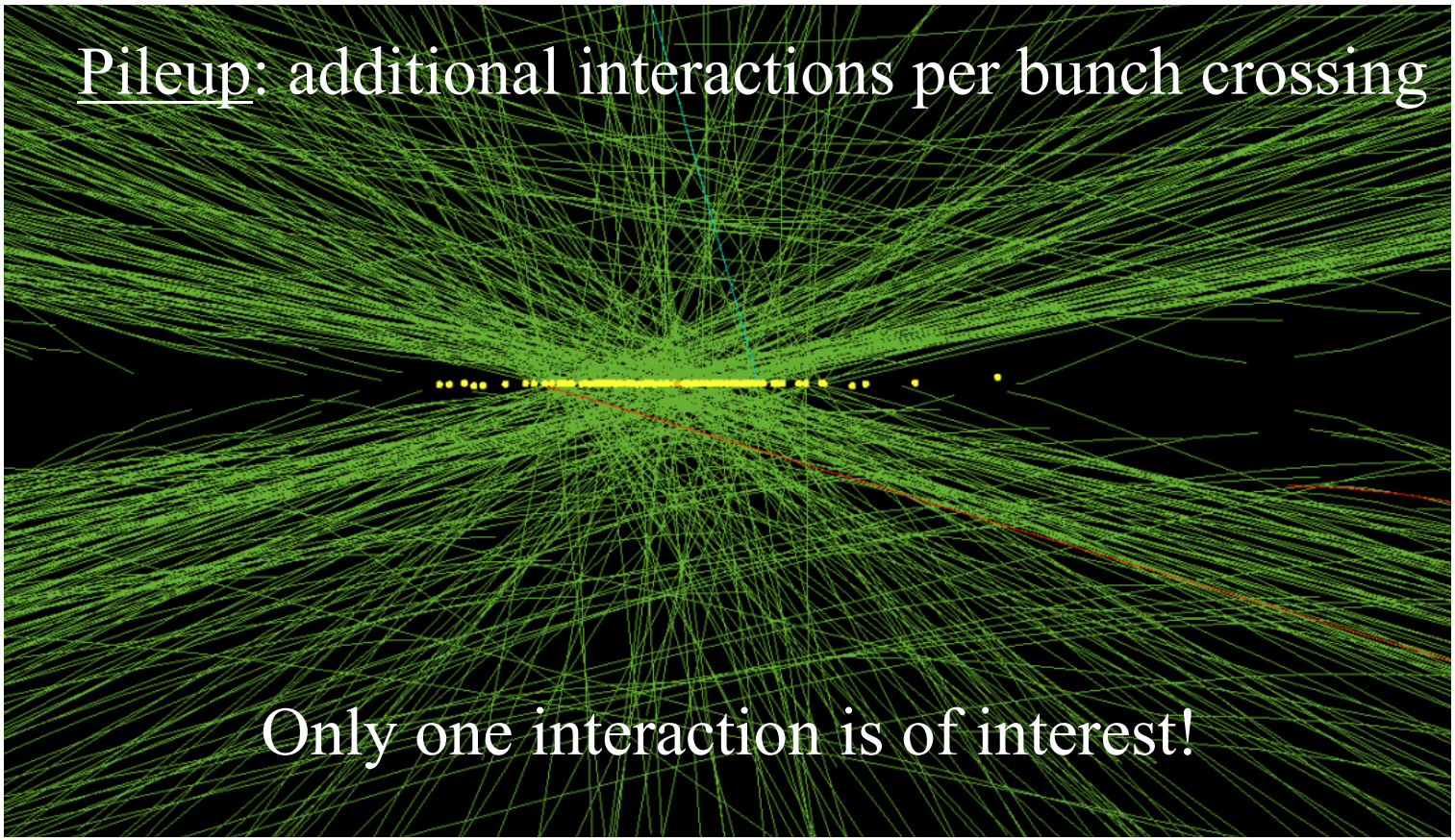
A few milestones in the development of convolutional networks:

- 1980 Kunihiko Fukushima invents the neocognitron that is able to perform character recognition.
- 1998 Yan LeCun developed LeNet, which is able to recognize handwritten zip code digits.
- 2012 Alex Krizhevsky introduced AlexNet, which produces state of the art results on the image database ImageNet Dataset[1].
- 2015 Izhar Wallach, Michael Dzamba, and Abraham Heifets (Atomwise, Inc.) introduced AtomNet for drug discovery using the 3D structure of molecules.

Deep convolutional neural networks (DCNN) are now the standard machine learning model for image recognition in many applications from self-driving cars to jet classification in particle physics.

[1] ImageNet, https://deepai.org/dataset/imagenet

A interesting application of DCNNs in particle physics was published in 2017 by Patrick T. Komiske, Eric M. Metodiev, Benjamin P. Nachman, Matthew D. Schwartz[2], namely, the removal of pileup.



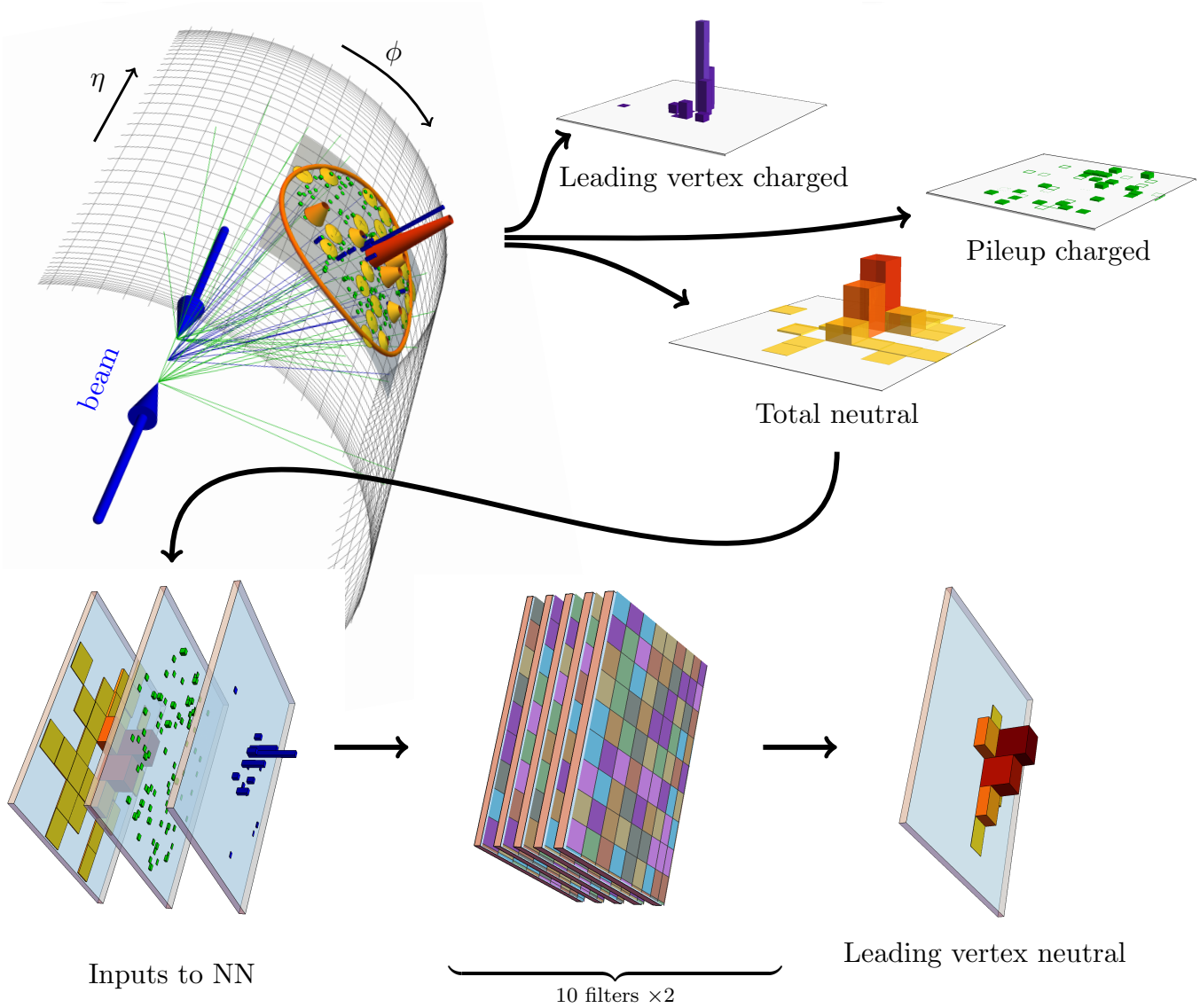
[2]Metodiev, Komiske, Nachman, Schwarz, *Pileup Mitigation with Machine Learning (PUMML)*, JHEP 12 (2017) 051, arXiv:1707.08600.

The basic idea is to treat a jet as a 36×36 pixel 3-color image in the (η, ϕ)-plane, where each color corresponds to be different category of particle:

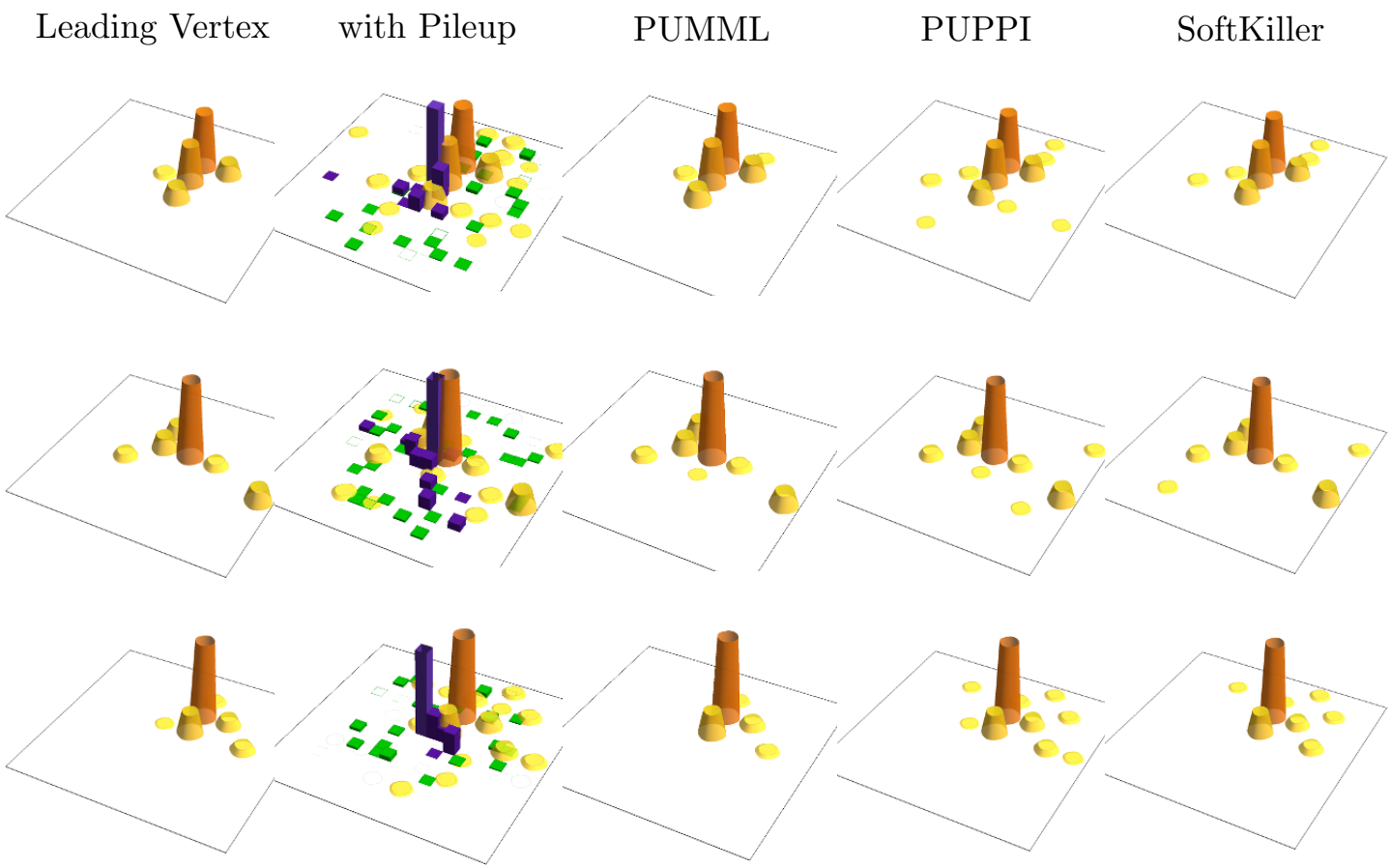
1. Red: transverse momenta (p_T) of all neutral particles.
2. Green: transverse momenta of charged particles from pileup interactions.
3. Blue: transverse momenta of charged particles from the primary vertex.

The output of the DCNN is an image of the neutral particles from the primary vertex.

The jet, with contamination from pileup removed, is formed by combining the charged and neutral particles from the primary vertex.

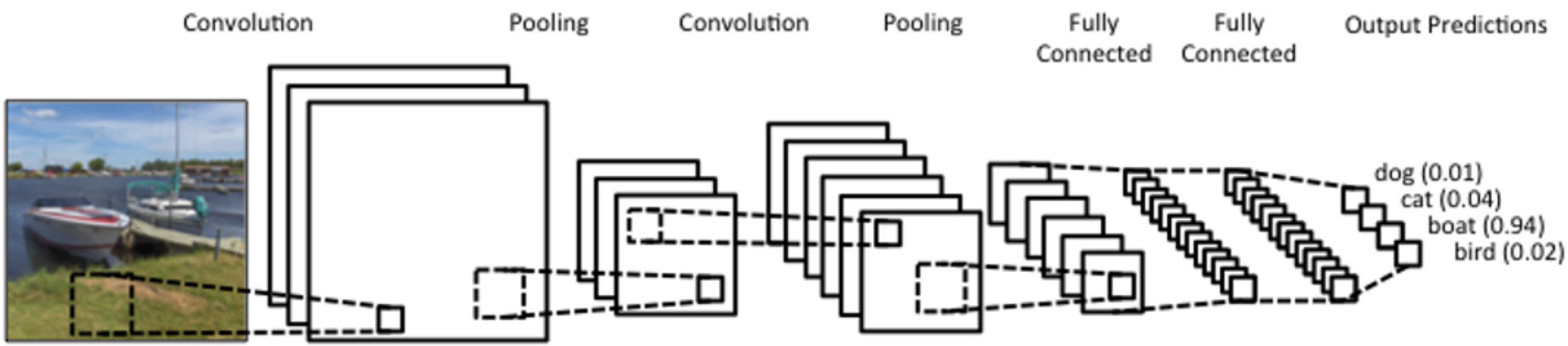

η
ϕ
beam
Leading vertex charged
Pileup charged
Total neutral
Inputs to NN
10 filters ×2
Leading vertex neutral

Results



A convolutional neural neural network comprises three types of processing element:

1. Convolution layers
2. Pooling (down-sampling) layers
3. A fully connected feed-forward network



Example (MNIST)

Consider the task of classifying the MNIST 28×28-pixel images of handwritten digits.

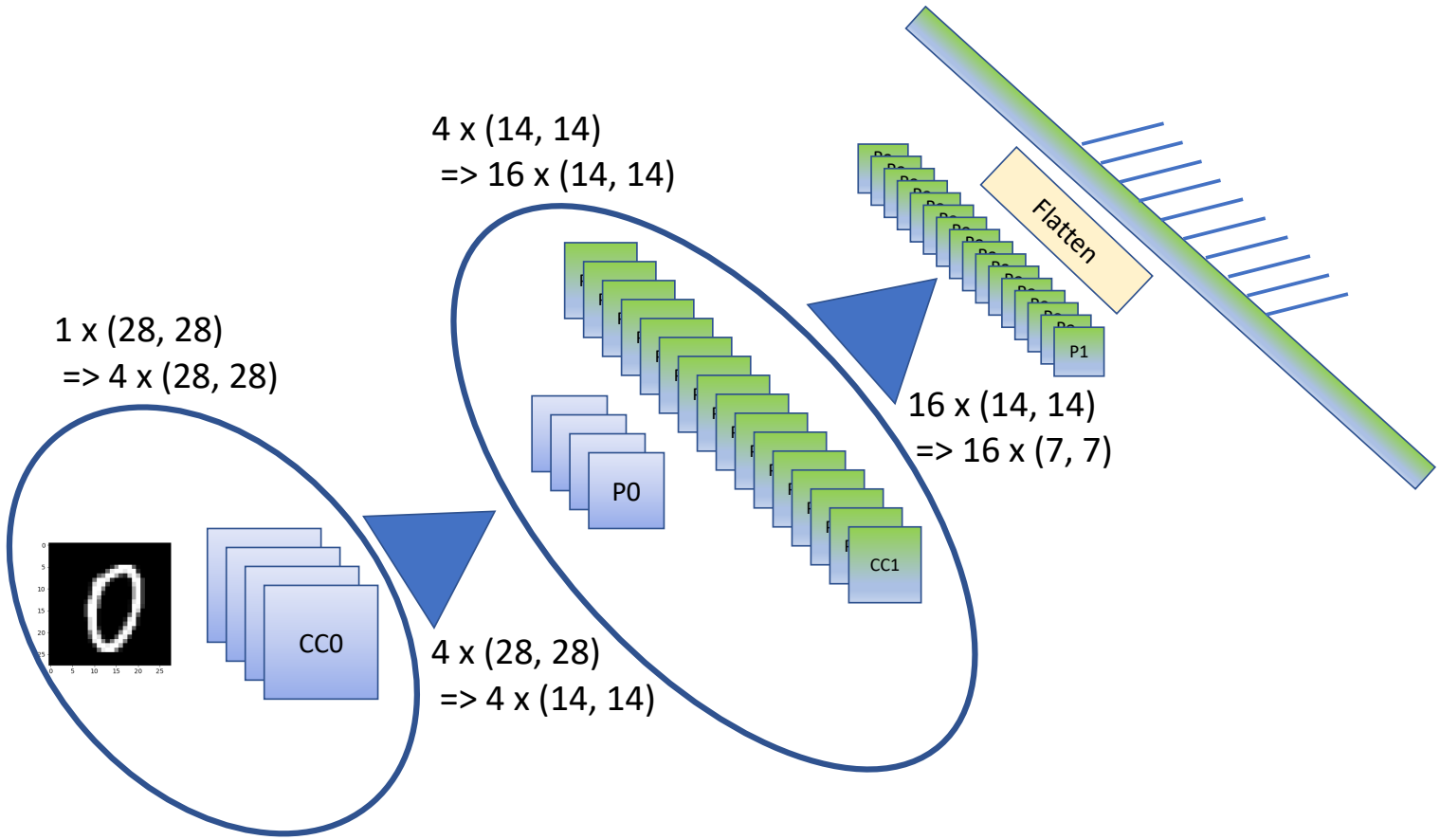
Unlike the jet images in the pileup application, the images here are single-channel gray scale.

Each image is zero padded with a 2-pixel border so that the images are of size 32×32.

As we shall see this is done so that the first convolutional layer produces filtered images that are of the same size as the originals, namely, 28×28-pixel images.

Example (MNIST)

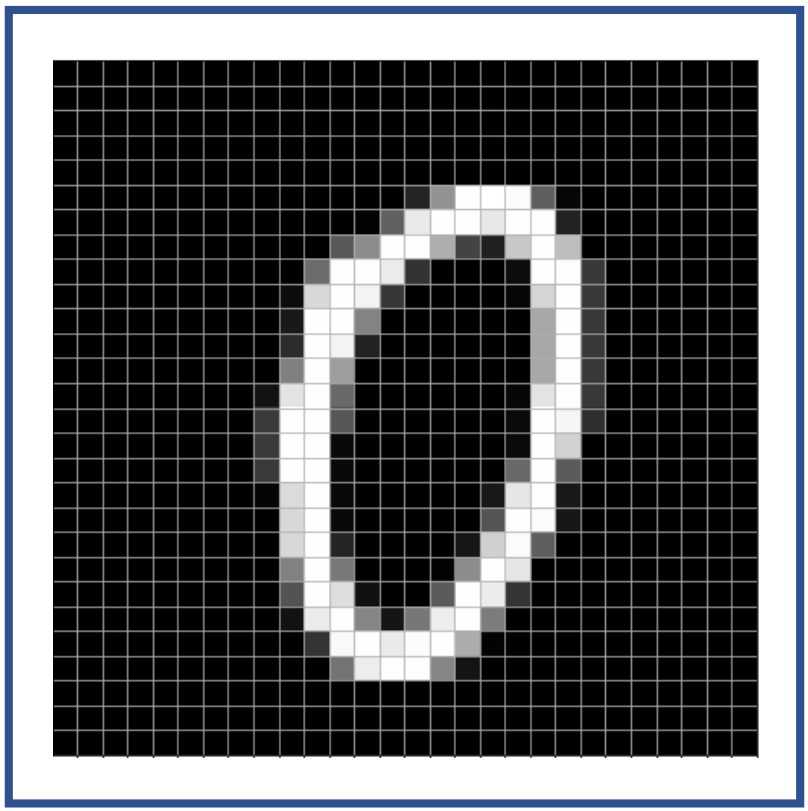
We'll classify the MNIST digits using the following model:



$$\mathbf{y} = \text{softmax}(\mathbf{b}_2 + \mathbf{w}_2 \text{flatten}(\text{relu}(\text{maxpool}(\mathbf{b}_1 + \text{cc}(\mathbf{w}_1, \text{relu}(\text{maxpool}(\mathbf{b}_0 + \text{cc}(\mathbf{w}_0, \mathbf{x}))))))))$$

Example (MNIST)

Zero padding

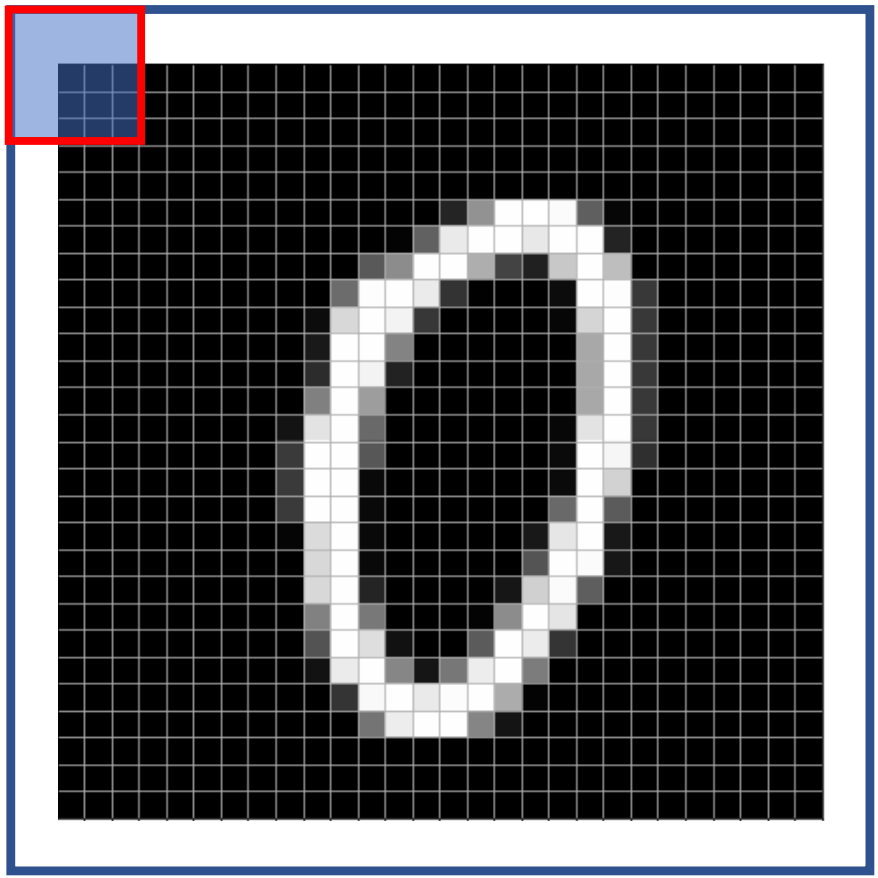


In this example, each $(28, 28)$-pixel image is zero-padded with a 2-pixel border.

We do this so that a $(5, 5)$ filter convolved with the image results in an output image of the same size as the original image.

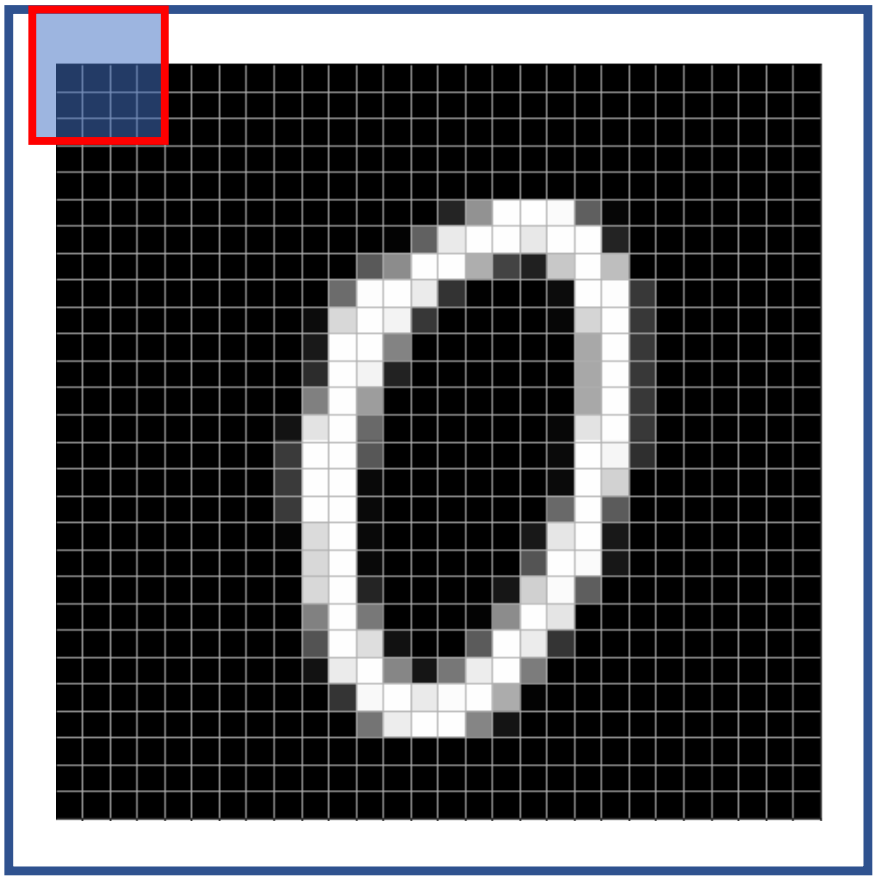
Example (MNIST)

Convolution – pixel (0, 0)



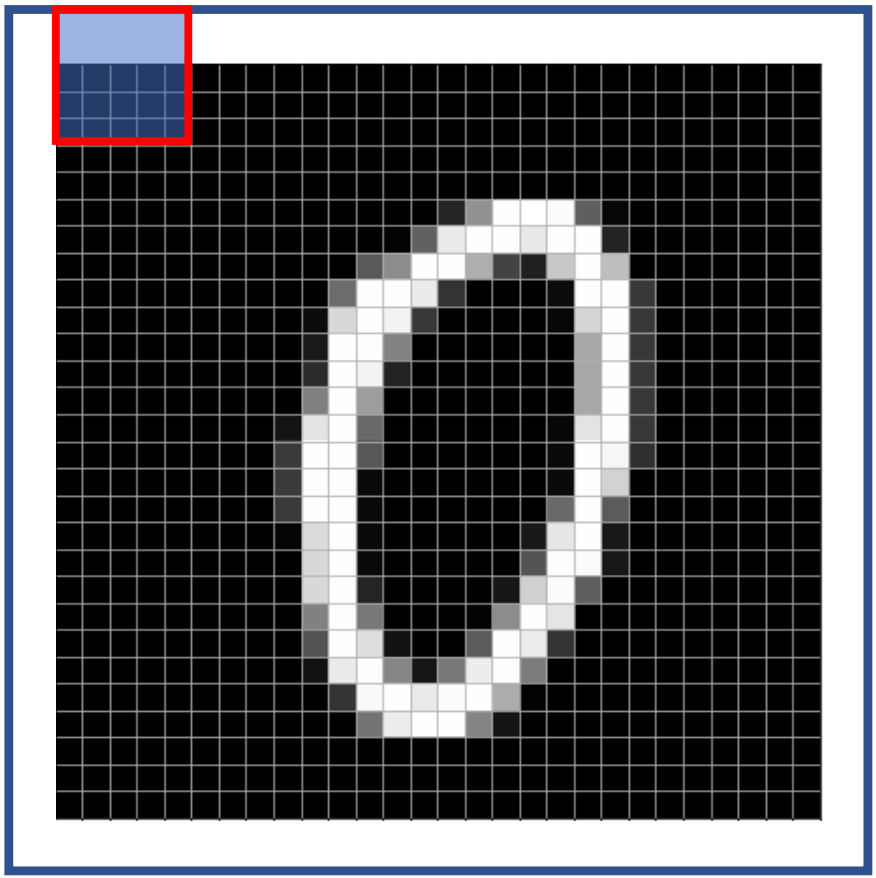
Example (MNIST)

Convolution – pixel (0, 1)



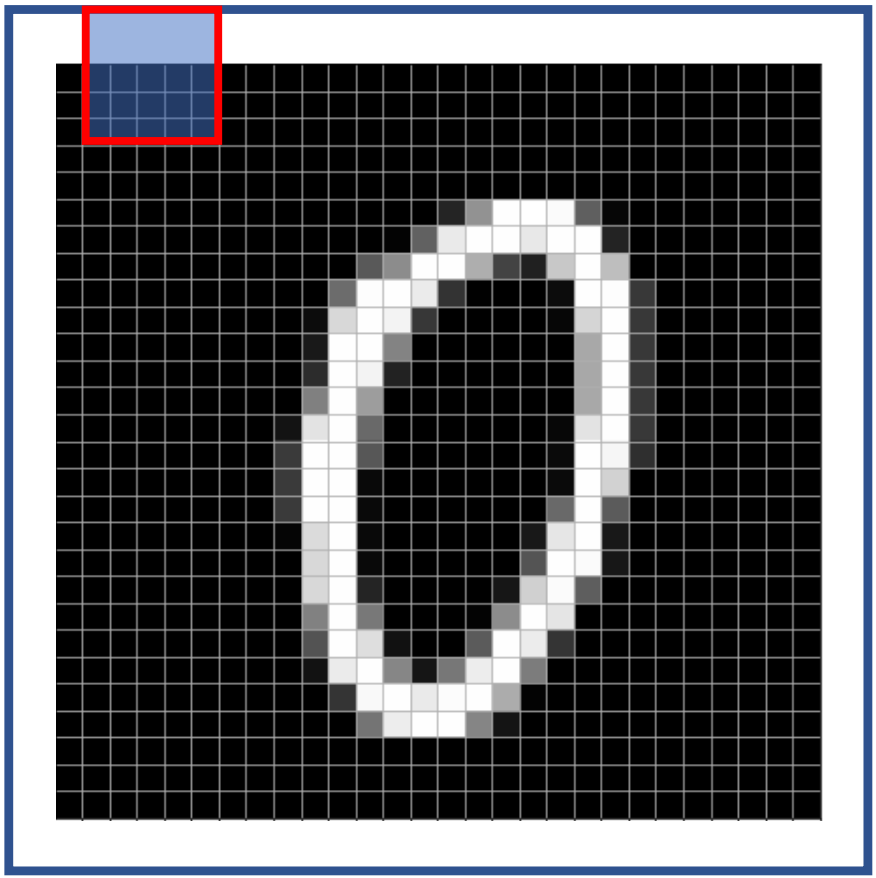
Example (MNIST)

Convolution – pixel (0, 2)



Example (MNIST)

Convolution – pixel (0, 3)



Example (MNIST)

Steps in detail:

1. Convolution layer – convolve input image $\mathbf{x}$ with a 4D kernel $\mathbf{w}_0$ comprising $(5, 5)$ filters each labeled by an output channel $k = 1, \cdots, K$ with $K = 4$ and an input channel $c = 1, \cdots C$, where for an MNIST image $C = 1$. For each input channel c and pixel (i, j) compute,

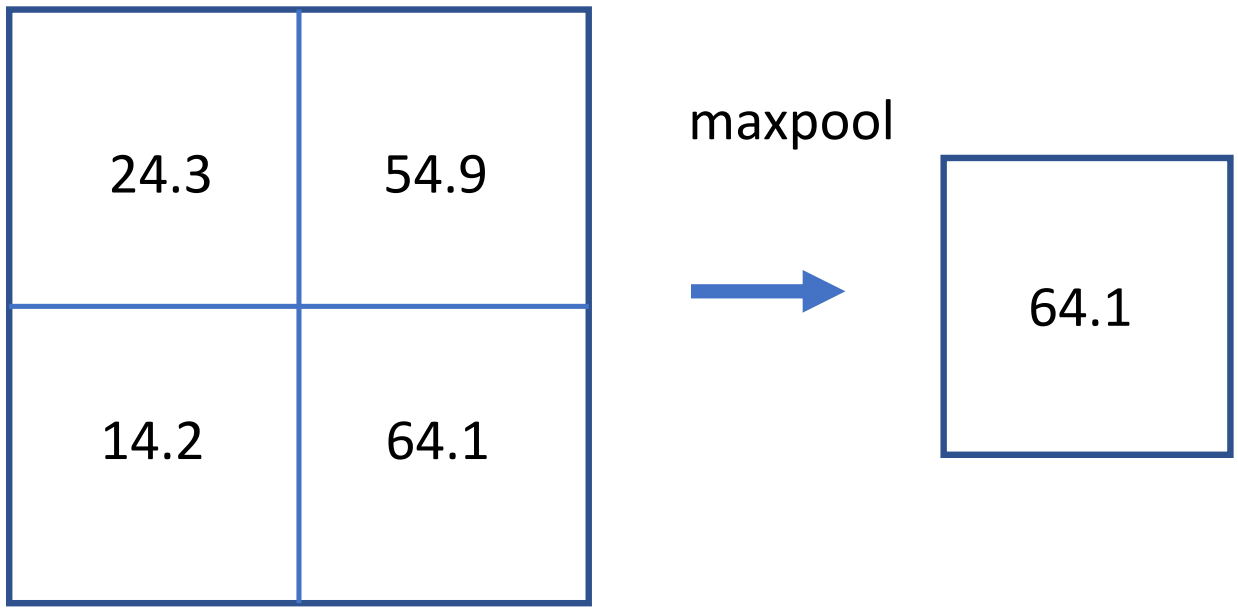
$$(\mathbf{b}_0)_k + \sum_{c=1}^{C} \sum_{r=-2}^{2} \sum_{s=-2}^{2} \mathbf{x}_{c,\, i+r,\, j+s} \, (\mathbf{w}_0)_{kcrs}.$$

   This results in $K = 4$ output images of size $(28, 28)$ pixels.

   Value of a given pixel (i, j) in an output image: compute the Hadamard product between the filter labeled by k, c and the array of $(5, 5)$ pixels it shadows in the input channel c, sum the matrix elements of the resulting $(5, 5)$ matrix and sum these sums across all input channels, c, and add an offset $(\mathbf{b}_0)_k$ to the sum.

Example (MNIST)

2. Maxpool layer – down-sample each of the output images from the previous step using a $(2, 2)$ window that breaks each of the 4 $(28, 28)$-pixel output images into a $(14, 14)$-pixel image. The value of each pixel in the down-sampled image is taken to be the maximum value within the associated $(2, 2)$ window:



Alternatively, compute the average value of the 4 pixels.

Example (MNIST)

3. Nonlinear layer – A non-linear function (in this example a ReLU) is applied element-wise to the 4 down-sampled images, which become the input channels for the next set of convolutions.

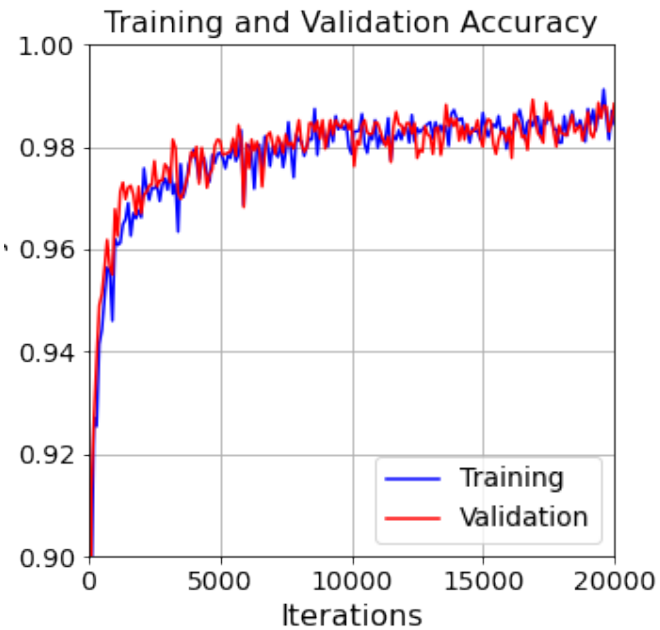
4. Repeat steps 1., 2., and 3., except that this time we have 4 $(14, 14)$-pixel input channels and the convolution layer outputs 16 channels of the same size, which is followed by a down-sampling to 16 $(7, 7)$-pixel channels.

5. Feed-forward layer – the pixels of the 16 $(7, 7)$ images are flattened to the 784 inputs of a single-layer fully-connected feed-forward neural network with 10 outputs, one for each digit, that are subject to the `softmax` function:

$$\texttt{softmax}(\mathbf{x})_m = \frac{\exp(x_m)}{\sum_n \exp(x_n)}.$$

Example (MNIST)

```
model details
  conv0.weight        torch.Size([4, 1, 5, 5])
  conv0.bias          torch.Size([4])
  conv1.weight        torch.Size([16, 4, 5, 5])
  conv1.bias          torch.Size([16])
  linear0.weight      torch.Size([10, 784])
  linear0.bias        torch.Size([10])
```

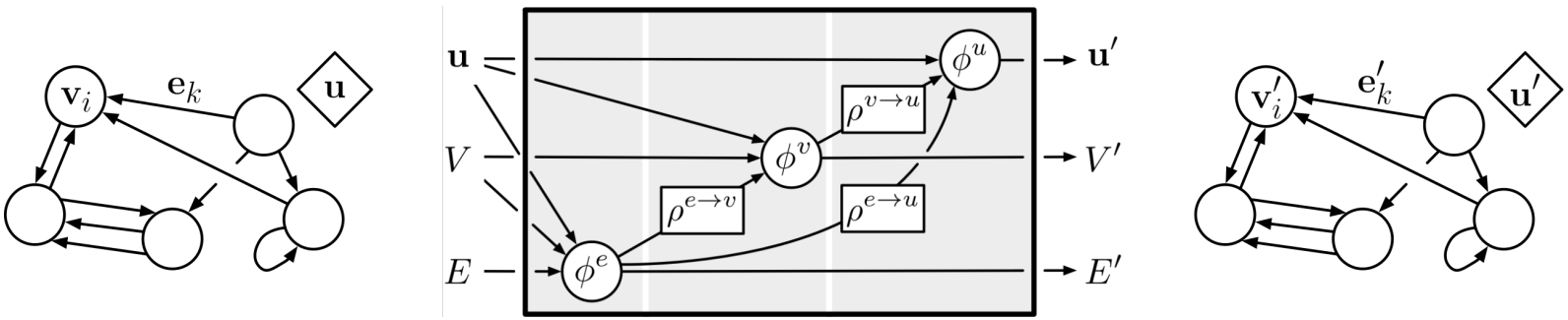


Training details:

- training sample size: 50,000
- learning rate: 1.0×10^{-3}
- batch size: 20
- optimizer: `Adam`

Outline

A graph is a structure consisting of vertices connected by edges. A graph neural network models this structure:



[3]

Each vertex (or node) i has vertex attribute $\mathbf{v}_i$ and each edge (i.e., line or lines joining nodes) k has edge attributes $\mathbf{e}_k$.

A graph enters a graph processor on the left and exits the processor on the right with altered attributes.

[3] https://github.com/deepmind/graph_nets

Consider a recent paper [4] by the IceCube Collaboration on event classification using graph networks.

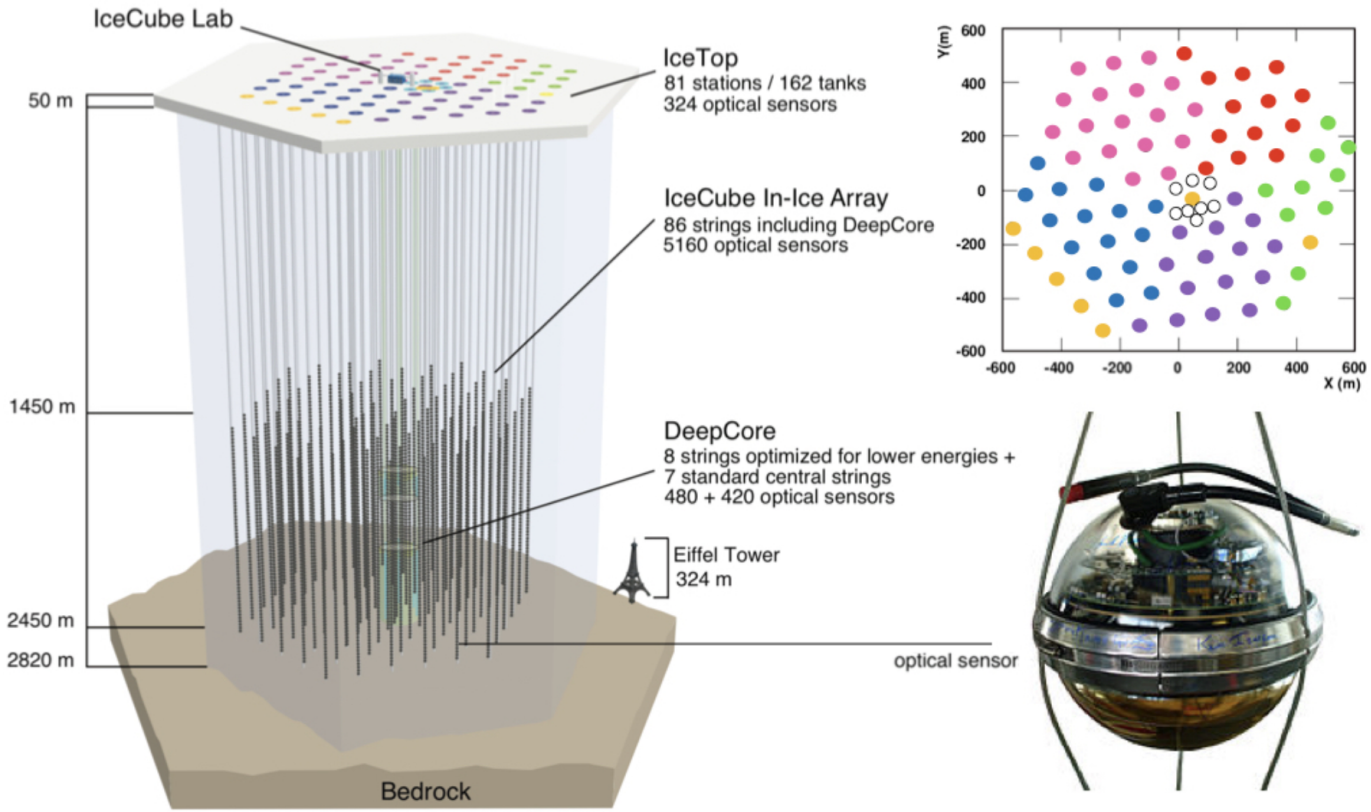
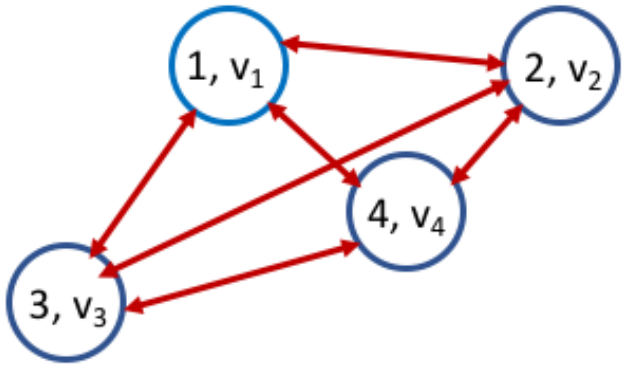


Fig. 1. The IceCube Neutrino Observatory with the in-ice array, its sub-

[4] N. Choma *et al.* `IceCube collaboration`, Graph Neural Networks for IceCube Signal Classification, arXiv:1809.06166v1

IceCube models the signals from n digital optical modules (DOMs) as the vertices of a graph in which each vertex is associated with a d-dimensional (row-wise) vector of attributes $\mathbf{v}_i = (x_1, \cdots, x_d)_i$. Three of the components are the spatial coordinates x, y, z of the DOM.

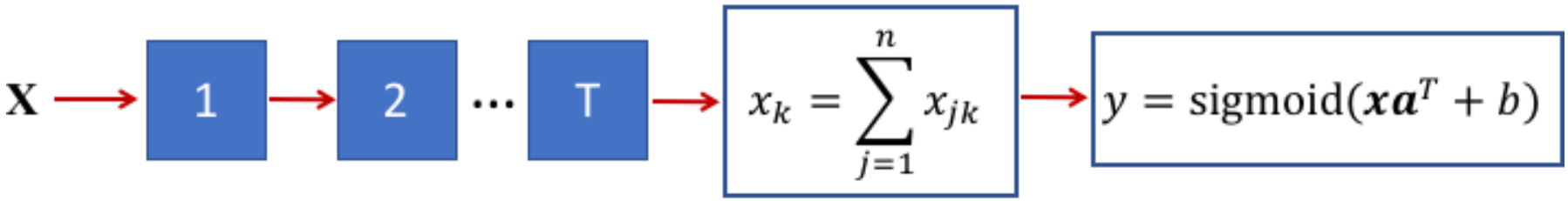


An $n \times n$ adjacency matrix, $\mathbf{A}(\sigma)$, which depends on a tunable parameter σ, models the edges and their associated (1D) edge attributes.

The vectors are concatenated vertically (denoted by $[a; b; ...]$, while $[a, b, ...]$ denotes horizontal concatenation) into an $n \times d$ matrix

$$\mathbf{X} = [\mathbf{v}_1; \cdots; \mathbf{v}_n] \tag{1}$$

The IceCube classifier recursively modifies the $n \times d$-dimensional matrix $\mathbf{X}$ using a chain of structurally identical graph processors, followed by a map of the final version of the matrix $\mathbf{X}$ to a d-dimensional vector $\mathbf{x}$ using a map that is permutation invariant with respect to the vertices and invariant with respect to the number of vertices—here a simple sum over the rows of $\mathbf{X}$. $\mathbf{x}$ is then mapped to the scalar $0 \leq y \leq 1$.



Each graph processor $(1, \cdots, T)$ is parametrized with its own $2d \times d/2$-dimensional weight matrix $\mathbf{w}$ and scalar bias b and maps the graph defined by $(\mathbf{X}, \mathbf{A})$ to the graph defined by $(\mathbf{X}', \mathbf{A})$ using

$$\mathbf{X}' = [\mathrm{ReLU}(\mathbf{Y}), \mathbf{Y}], \text{ where } \mathbf{Y} = [\mathbf{AX}, \mathbf{X}]\,\mathbf{w} + b\mathbf{u}, \tag{2}$$

ReLU is applied element-wise, and $\mathbf{u}$ is an $n \times d/2$ matrix of ones.

Performance[5]

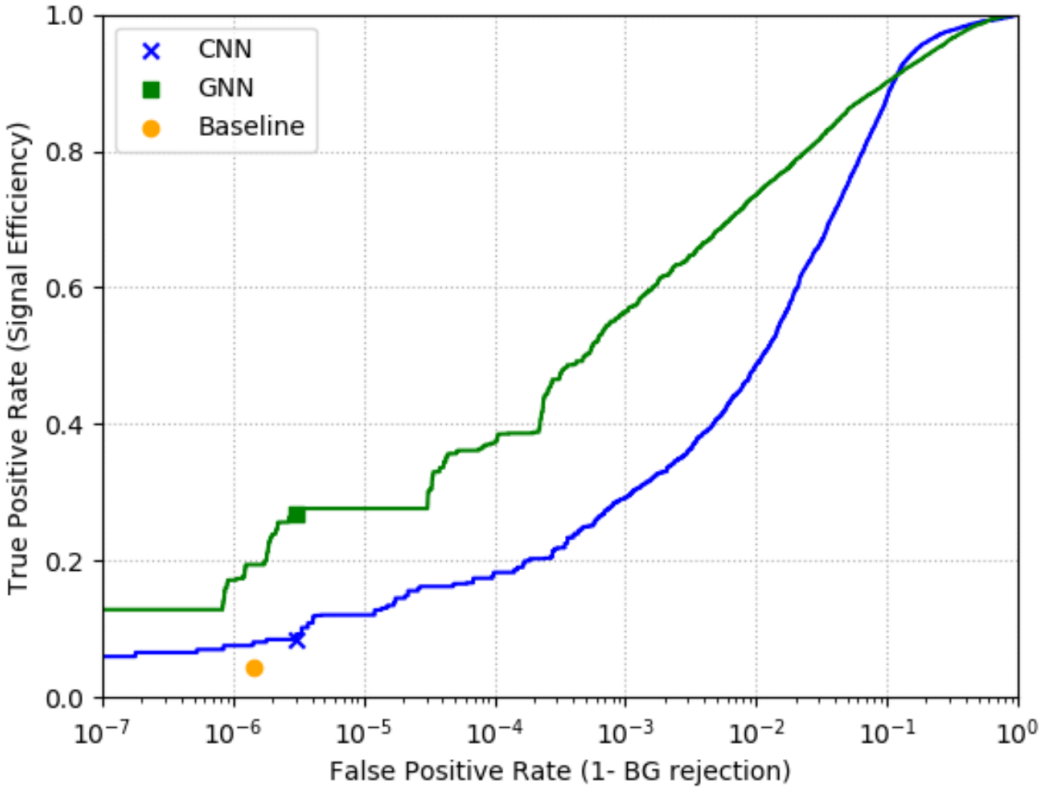


Fig. 3. Receiver operating characteristic curve for various methods considered in this paper. The green square and blue X indicate the evaluation point for the GNN and CNN, respectively.

[5] N. Choma *et al.* `IceCube collaboration`, Graph Neural Networks for IceCube Signal Classification, arXiv:1809.06166v1

Outline

- Deep convolutional neural networks (DCNN) are the model of choice today for processing 2D and 3D data such as images.
- The models work well even for data that are not intrinsically images, for example, classifying sentences.
- For data in which the order of elements is not relevant or for which the number of elements can vary, graph neural networks have been shown to give excellent performance.