UNIVERSITY OF CAMBRIDGE

Introduction to Deep Learning Techniques

Leigh Whitehead

26th October 2023

8th UK LArTPC Software and Analysis Workshop

Slack channel: #deep-learning

Introduction

- This lecture is designed to give an introduction to machine learning and convolutional neural networks
  - These are the most common deep learning techniques used in neutrino physics
- We have to start with the basics:
  - The simplest possible neural network
  - Image recognition and convolutional neural networks
- I will give an example of neutrino classification

Introduction

- Machine learning isn't a new field!
  - Many techniques have been in use for a long time
- The name is generally applied to any approach where a large set of data is used to train an algorithm to perform some classification task or parameter estimation
  - k-Nearest-Neighbour
  - Boosted Decision Tree
  - Artificial Neural Network (ANN)
  - Etc, etc…
- We'll consider an ANN in the following example
  - You may have seen these called Multi-Layer Perceptrons (MLPs)

A very simple example

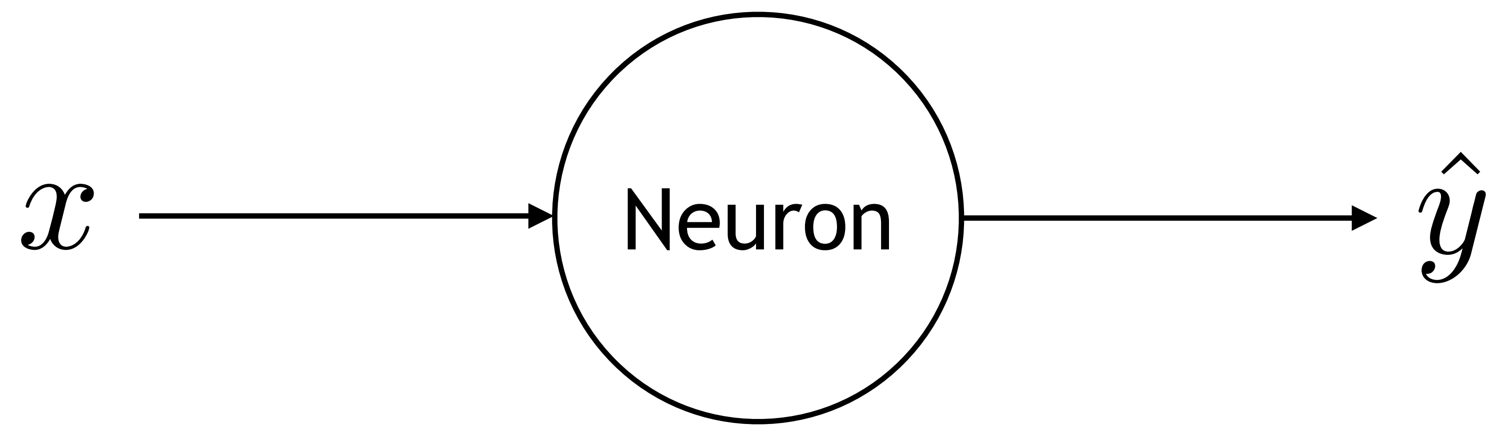
- Let’s say that we want to classify vehicles as either a car or a motorcycle using the value of a single variable
  - Define the input data as *x*, which in this case is mass
  - The target (truth) is given by *y*

Model	x (mass)	y (0=car, 1=motorcycle)
Renault Megane	1.175 tonnes	0
Yamaha YZF-R1	0.199 tonnes	1
MINI Cooper	1.360 tonnes	0
Ford C-MAX	1.550 tonnes	0
Kawasaki Ninja H2	0.240 tonnes	1

Thanks to Saúl Alonso Monsalve for this example

The architecture

- Consider the following: it corresponds to the simplest ANN we could design
  - For a given x we want to make a prediction $\hat{y}$ between 0 and 1



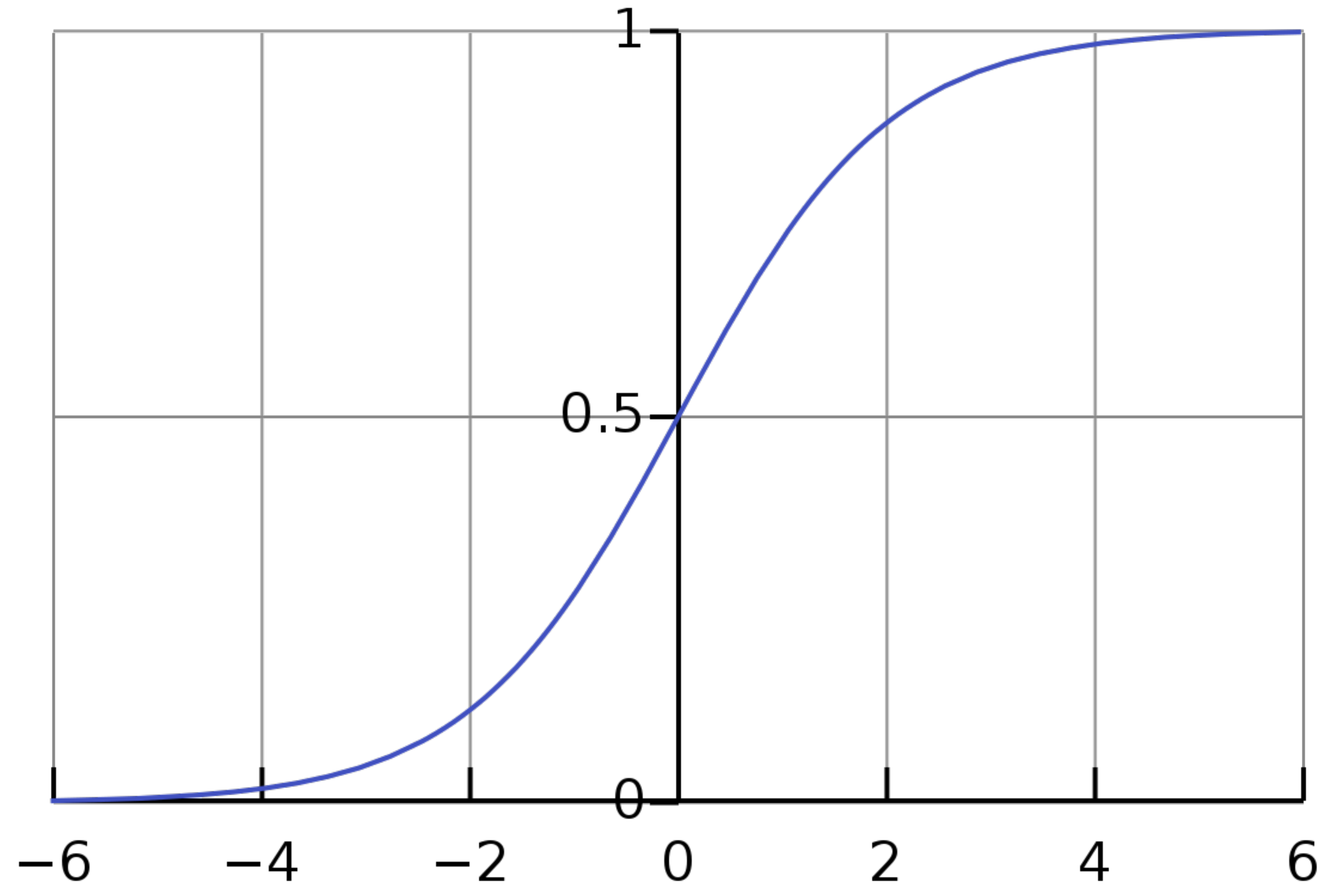
$$\hat{y} = f(wx + c)$$

NB: this single neuron ANN is just a logistic regression unit

  - Prediction depends on two other parameters
  - Common activation function choice:

$$f(z) = \mathrm{sigmoid}(z) = \frac{1}{1 + e^{-z}}$$

The sigmoid function allows us to bound our output between 0 and 1



Training the network

1. Randomly initialise variables *w* and *c* in the range (0,1)

2. Forward propagation
    1. Select a training example
    2. Calculate the prediction $\hat{y}$
    3. Calculate the loss (how close $\hat{y}$ is to y)

3. Backward propagation
    1. Compute partial derivatives of the loss
    2. Update *w* and *c*

4. Stop once we can no longer improve the loss

Repeat for the full dataset
(we call this one epoch)

Repeat as necessary for n epochs

First forward propagation

- Assume we initialised *w* = 0.5 and *c* = 0.5
- Select the first training example:

Model	x (mass)	y (0=car, 1=motorcycle)
Renault Megane	1.175 tonnes	0

$$\hat{y} = \sigma(wx + c) = \frac{1}{1 + e^{-(wx+c)}} = \frac{1}{1 + e^{-(0.5x+0.5)}} = \frac{1}{1 + e^{-1.0875}} = 0.74791066$$

- Now we need a way to compare how well we have done
  - This is where the loss function comes in

Loss functions

- Loss functions provide us with a measure of how close our predicted value $\hat{y}$ is to the true value
  - The goal is the training is to minimise the value of this loss function
- In the case for a classification problem (like this) we use the categorical cross-entropy loss
  - Since we only have two true classes, we use the binary cross-entropy loss

$$\mathcal{L}\left(y,\hat{y}\right)=-\left(y\ln\hat{y}+\left(1-y\right)\ln\left(1-\hat{y}\right)\right)$$

First forward propagation

- Assume we initialised $w = 0.5$ and $c = 0.5$
- Select the first training example:

Model	x (mass)	y (0=car, 1=motorcycle)
Renault Megane	1.175 tonnes	0

$$\hat{y} = \sigma(wx + c) = \frac{1}{1 + e^{-(wx+c)}} = \frac{1}{1 + e^{-(0.5x+0.5)}} = \frac{1}{1 + e^{-1.0875}} = 0.74791066$$

- Using our binary cross-entropy loss, we get

$$\begin{aligned}\mathcal{L}(y, \hat{y}) &= -\left(y \ln \hat{y} + (1 - y) \ln(1 - \hat{y})\right) \\ &= -(0 \times \ln(0.7479) + 1 \times \ln(1 - 0.7479)) = 1.378\end{aligned}$$

First backward propagation

- Firstly, let's simplify things as for this training example $y = 0$

$$\mathcal{L}(0,\hat{y}) = \mathcal{L}(\hat{y}) = -\ln(1-\hat{y})$$

$$\mathcal{L}(z) = -\ln\left(1 - \frac{1}{1+e^{-z}}\right) = z + \ln(1+e^{-z})$$

$$\mathcal{L}(w,c) = wx + c + \ln\left(1 + e^{-(wx+c)}\right)$$

- Now take the partial derivatives

$$\frac{\partial\mathcal{L}(w,c)}{\partial w} = x\left(1 - \frac{e^{-(wx+c)}}{1+e^{-(wx+c)}}\right) = 0.8788$$

$$\frac{\partial\mathcal{L}(w,c)}{\partial c} = 1 - \frac{e^{-(wx+c)}}{1+e^{-(wx+c)}} = 0.7479$$

First backward propagation

- Now, let's update our w and *c* values

This is a **very** important parameter. It is the **learning rate** and must be positive. Let's set it equal to 0.1 in this example.

$$w_1 = w_0 - \alpha \frac{\partial \mathcal{L}(w,c)}{\partial w}\Bigg|_{\substack{w=w_0\\c=c_0}} = 0.4121$$

$$c_1 = c_0 - \alpha \frac{\partial \mathcal{L}(w,c)}{\partial c}\Bigg|_{\substack{w=w_0\\c=c_0}} = 0.4252$$

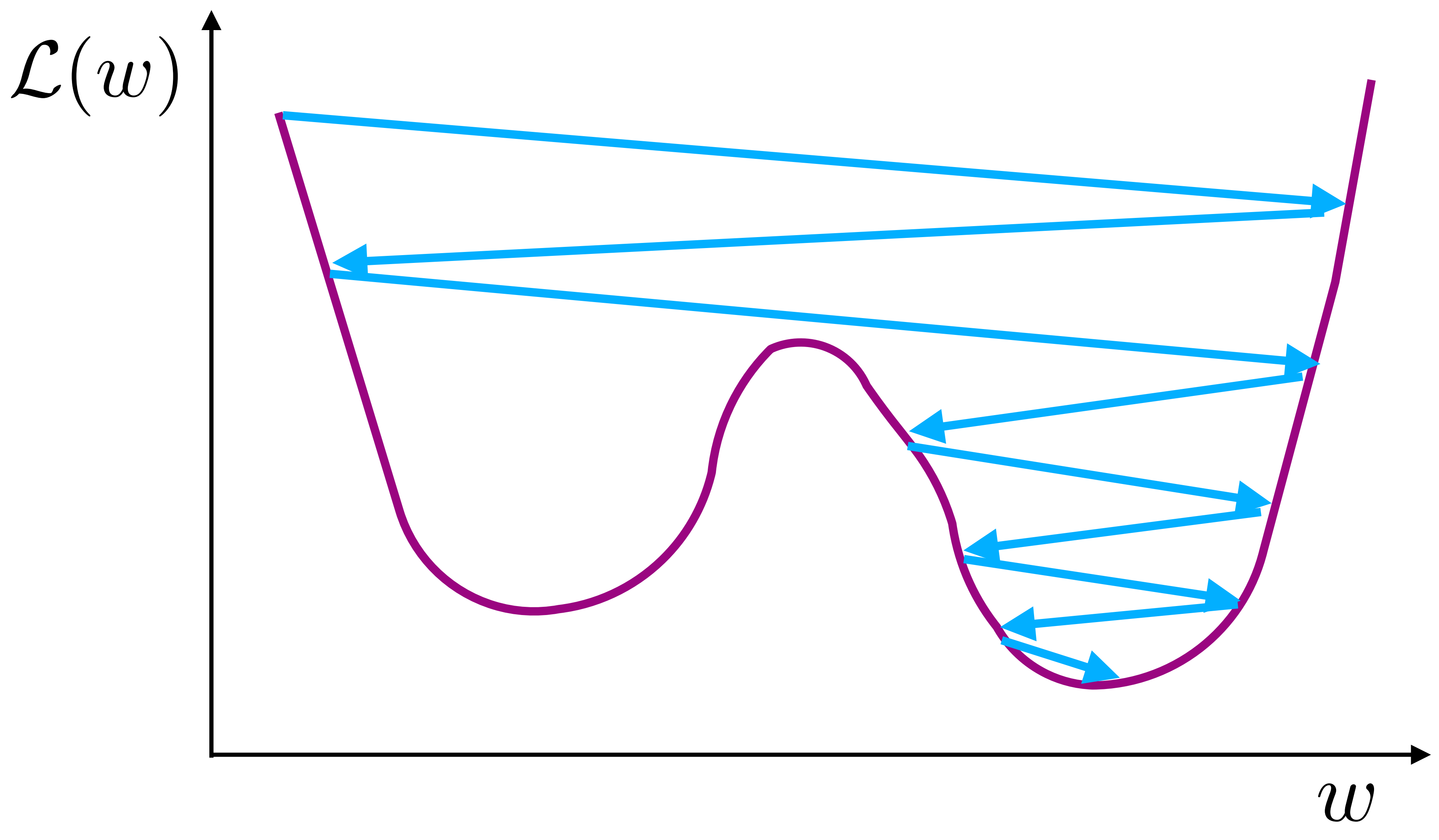
NB: these equations are for stochastic gradient descent

- Now we can compute our new prediction: $\hat{y}_1 = 0.7129$

We have gone from a prediction of 0.7479 to 0.7129 in one iteration.
Closer to our target of y = 0! Now repeat for the entire dataset!

Optimisers

- In reality we don't have to calculate these gradients ourselves
  - The optimiser does the back propagation and updates the network weights
  - Typically versions of stochastic gradient descent
  - Goal: find the global minimum of the loss function



- Some of these different algorithms try to improve on SGD
  - They use modified equations to update the weights
  - Find the global minimum
  - Converge quickly

- Some of the most used algorithms:
  - Adam, Adadelta, RMSProp, etc

Quick Aside on Learning Rate

- Now let's think about the learning rate
  - Recall that the learning rate controls the updating of the network parameters after each iteration

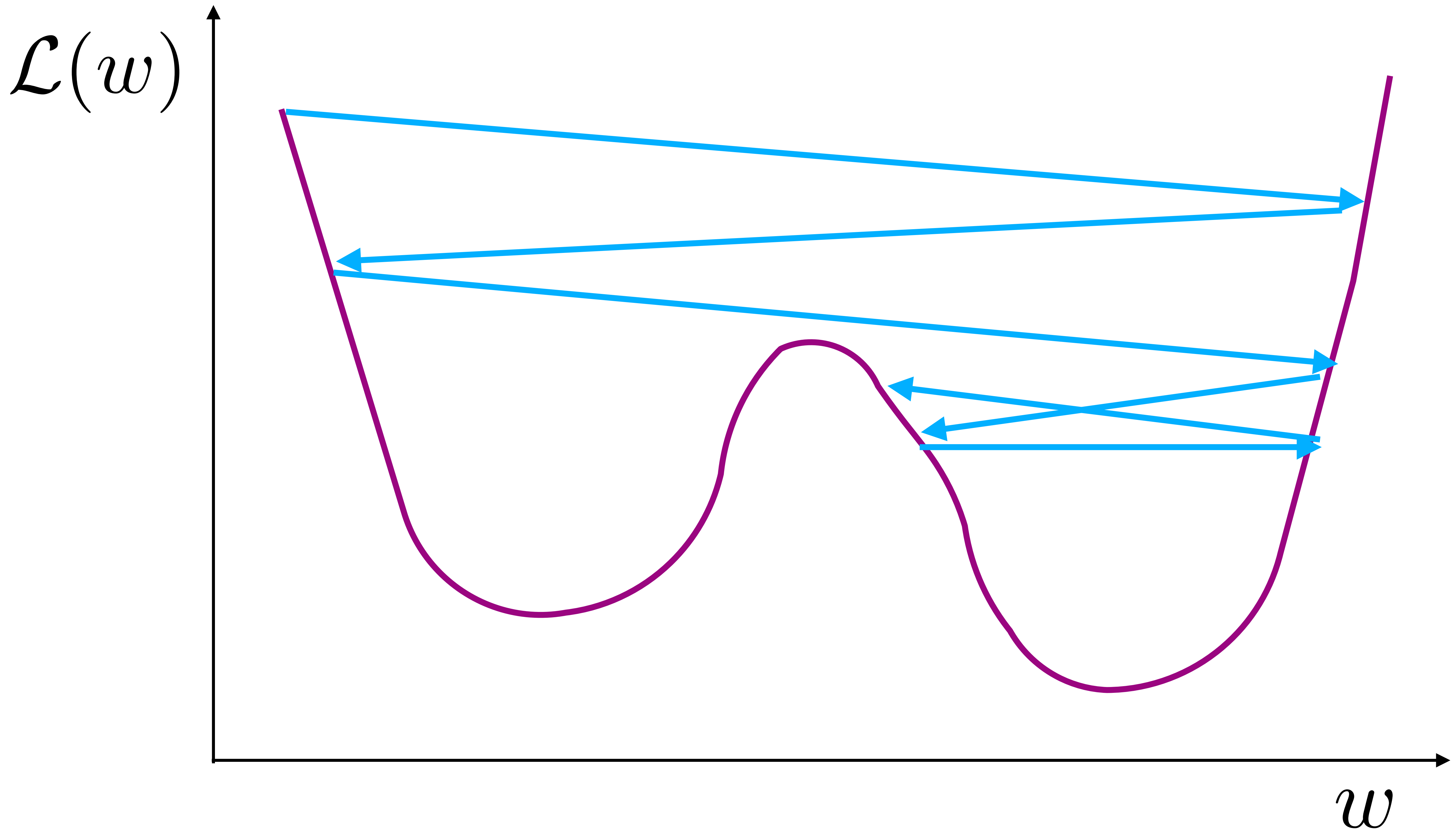
$$w_1 = w_0 - \alpha \frac{\partial \mathcal{L}(w,c)}{\partial w}\bigg|_{\substack{w=w_0 \\ c=c_0}} = 0.4121$$

$$c_1 = c_0 - \alpha \frac{\partial \mathcal{L}(w,c)}{\partial c}\bigg|_{\substack{w=w_0 \\ c=c_0}} = 0.4252$$

- The larger the learning rate, the bigger steps we take to find the minimum of the loss function

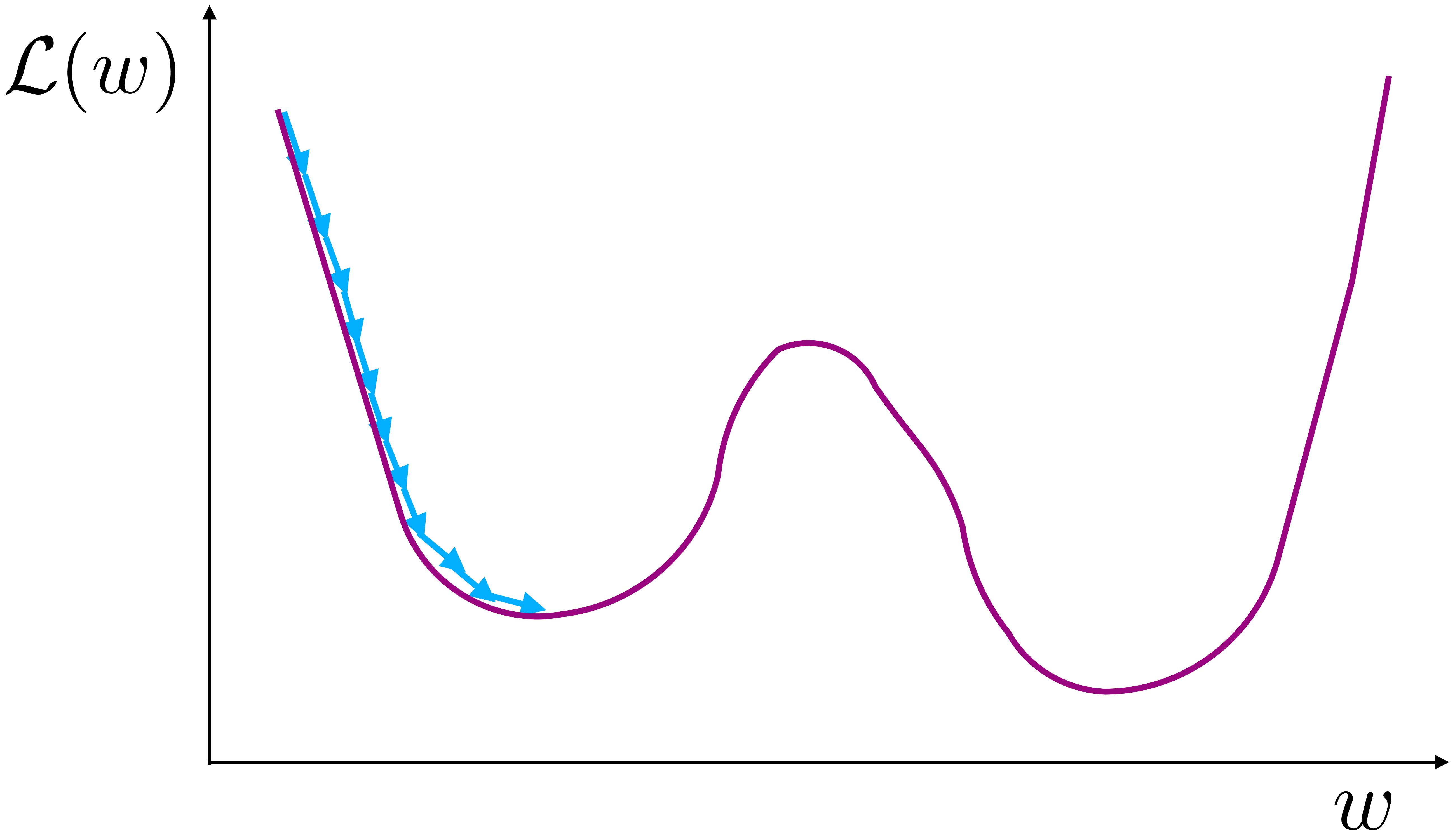
Quick Aside on Learning Rate

- We can get problems if the learning rate is:
  - Too large - can fail to converge on the minimum



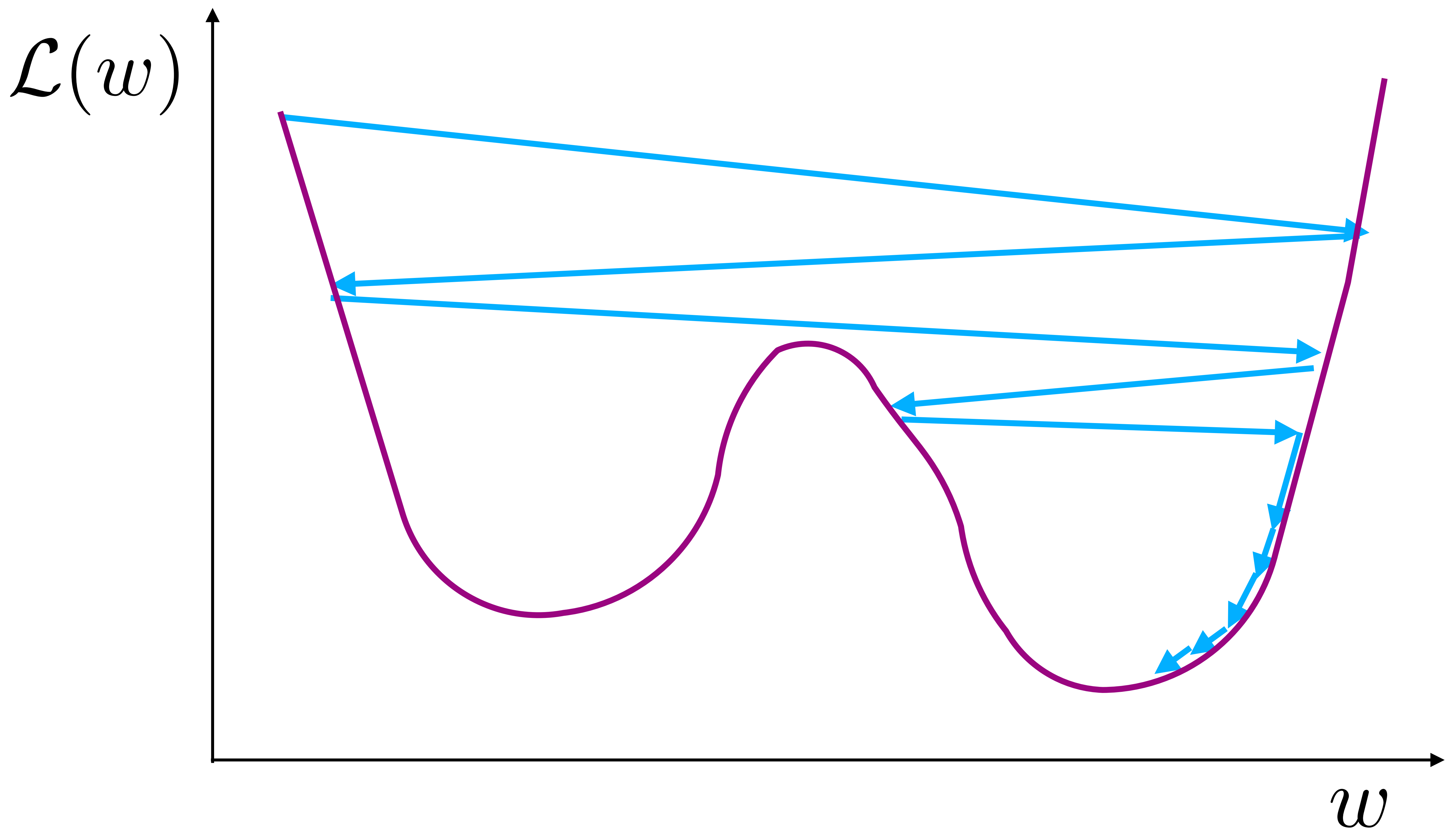
Quick Aside on Learning Rate

- We can get problems if the learning rate is:
  - Too large - can fail to converge on the minimum
  - Too small - can be very slow and find a local minimum



Quick Aside on Learning Rate

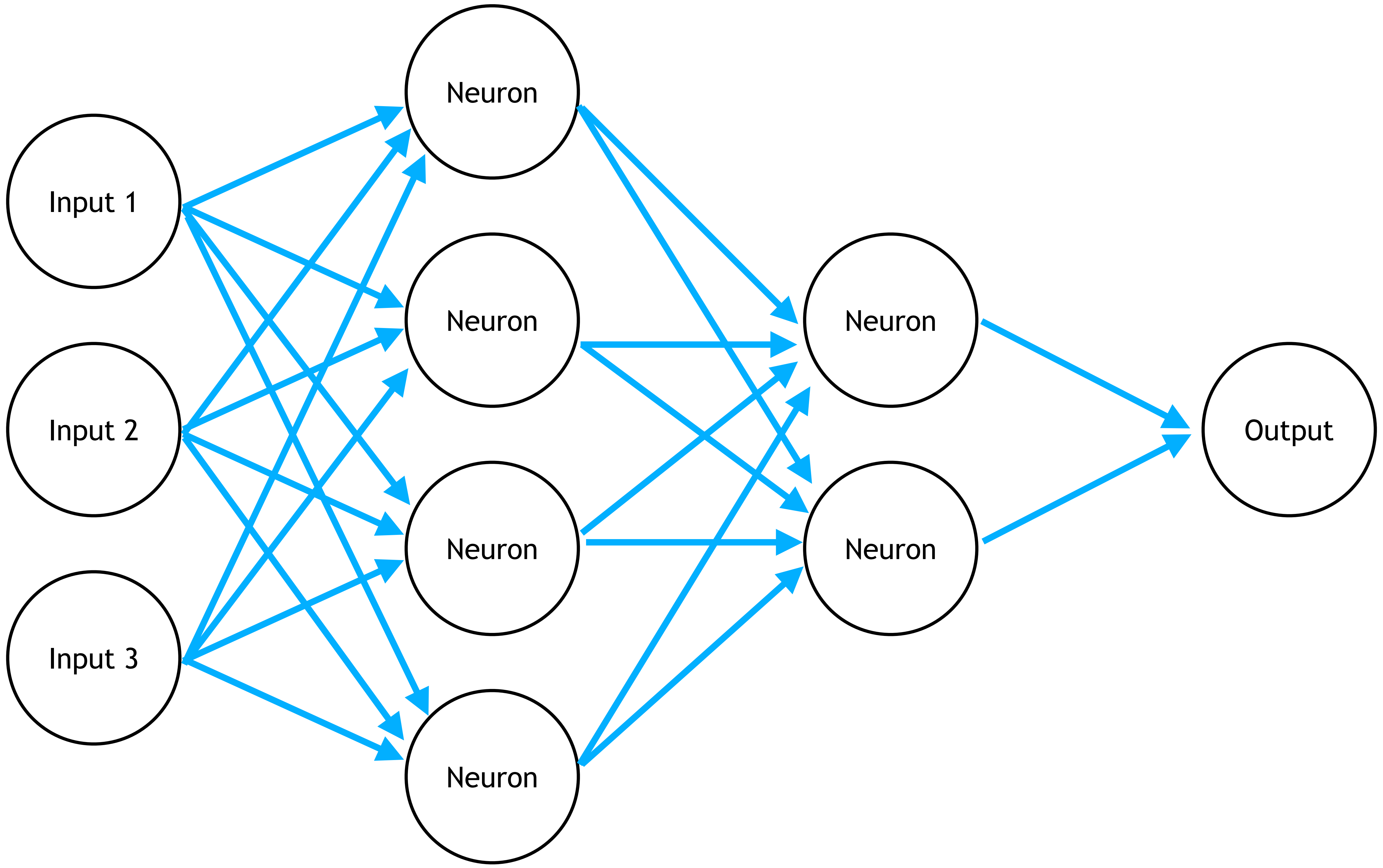
- Once thing we can try is learning rate decay
  - Start with a large rate and reduce with iterations



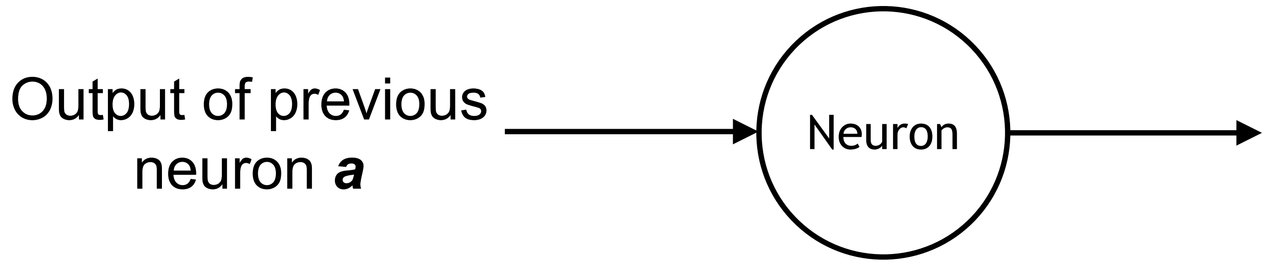
  - NB: this isn't always necessary, but something useful to know about

Going deeper

- ANNs consist of a number of neurons organised in layers



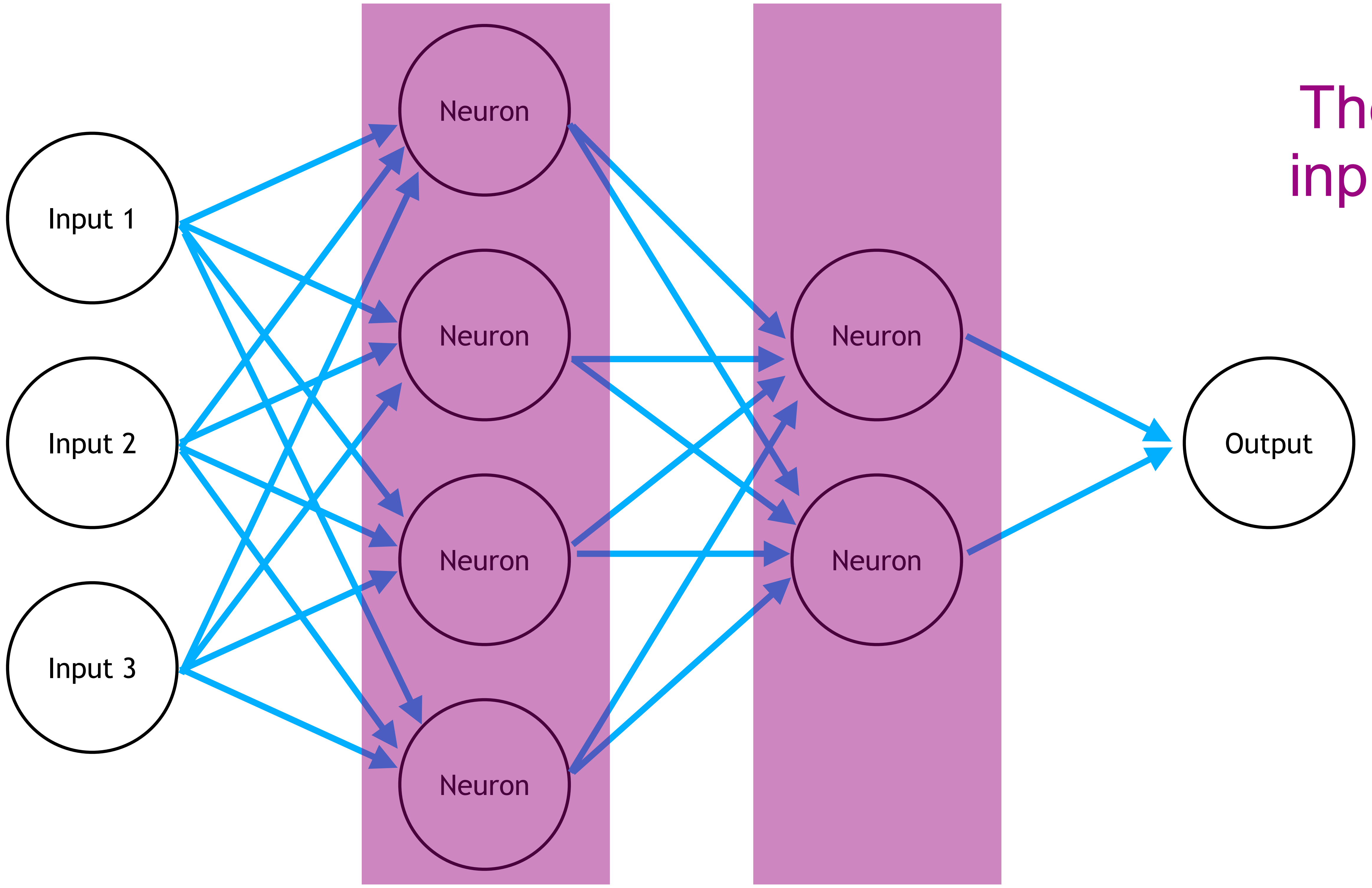
Each neuron here is just the same as in the simple example



$$f\left(w_i a + c_i\right)$$

Going deeper

- ANNs consist of a number of neurons organised in layers



The layers that are not the inputs or outputs are called **hidden layers**

- Deep Learning refers to the use of deep neural networks - networks with many hidden layers

Training networks

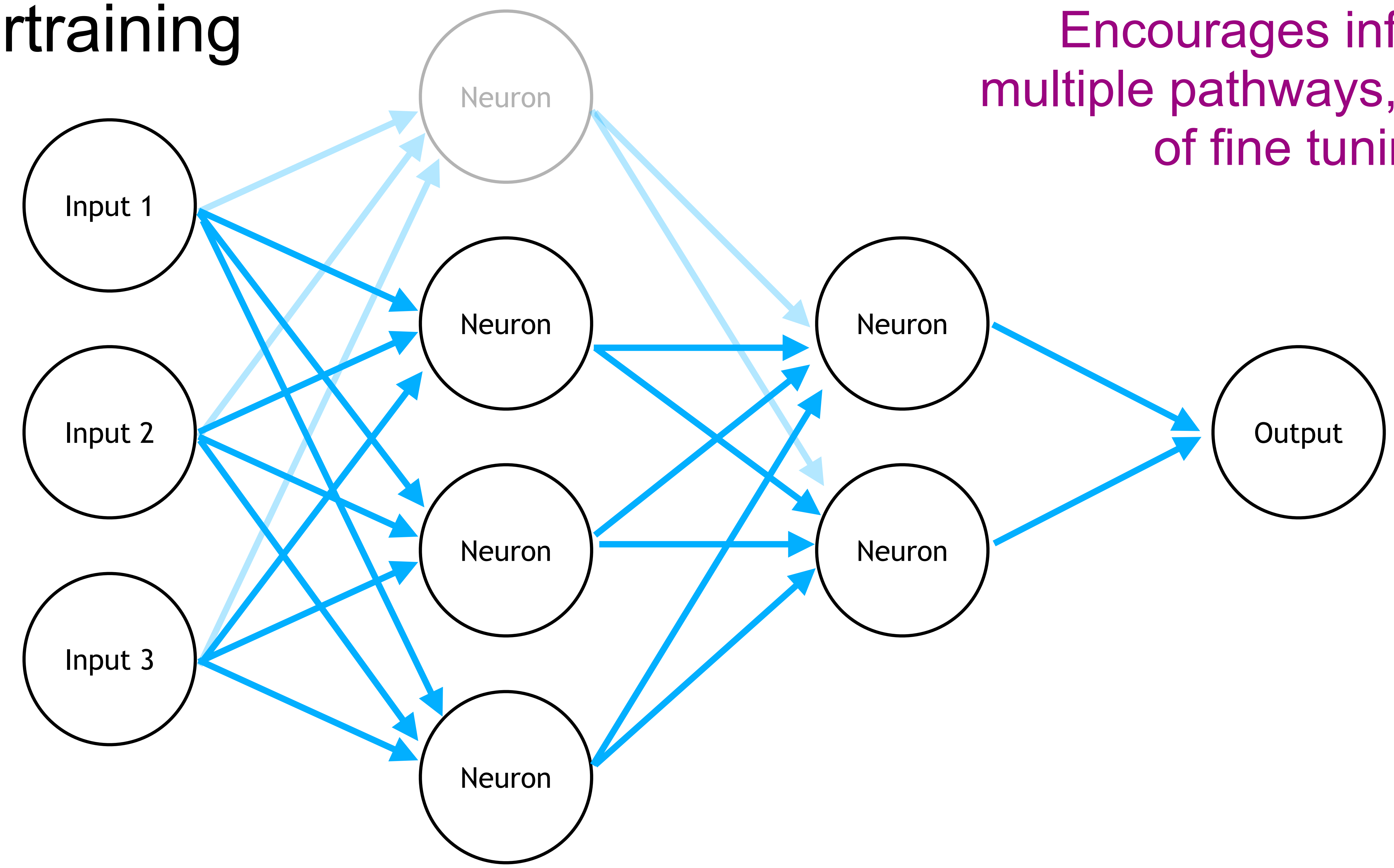
- Typically use three data samples:
- Training sample:
  - These are the events that the network learns from via the forward- and back-propagation
- Validation sample:
  - After each epoch the validation sample is used to measure the network performance
- Testing sample:
  - Once the network has finished learning, the test set provides a way to test the network generalisation

Overtraining

- A common concern is that networks can eventually learn fine details of training events that prevents generalisation to unseen events
  - This is known as overtraining
- Causes:
  - Too few training examples
  - Training set is not representative of the entire sample
  - Training for too long
- Potential solutions
  - Get more training data
  - Stop the training once the validation sample loss stops reducing
  - Look at techniques such as dropout…

Overtraining

- A common concern is that networks can eventually learn fine details of training events that prevents generalisation to unseen events
  - This is known as overtraining



Encourages information to flow through multiple pathways, hence reducing the chance of fine tuning on specific inputs

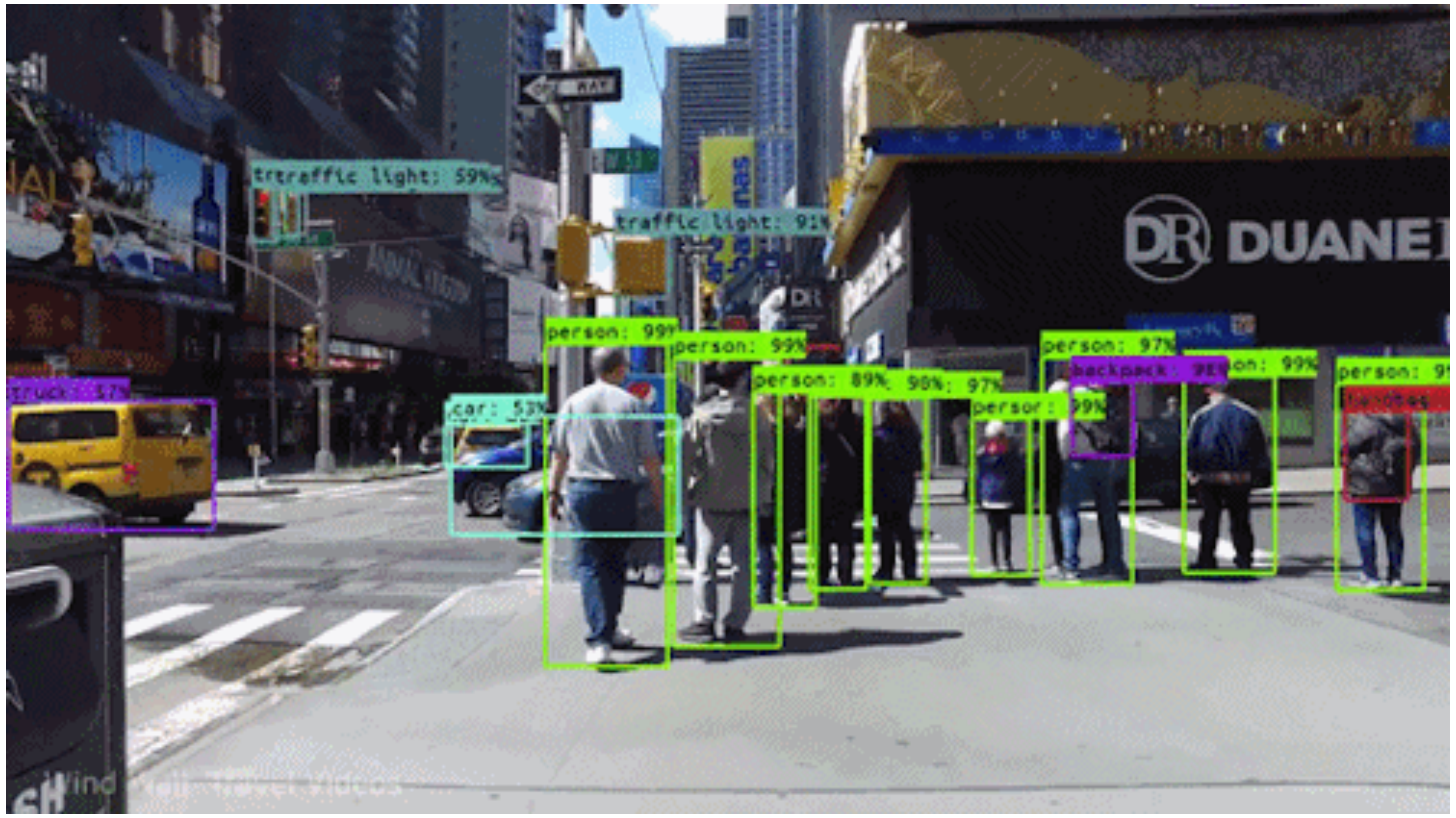
- Dropout: randomly ignore a given fraction of neurons each iteration

Image Recognition

- There has been a lot of work in the last couple of decades on automated image recognition
- There are many examples of where it is required and used
- Self driving cars are a good example
  - Need to be able to automatically recognise road signs and instructions as well as unexpected obstacles, pedestrians etc
  - The techniques have to be robust and reliable since cars can be very dangerous

Image Recognition

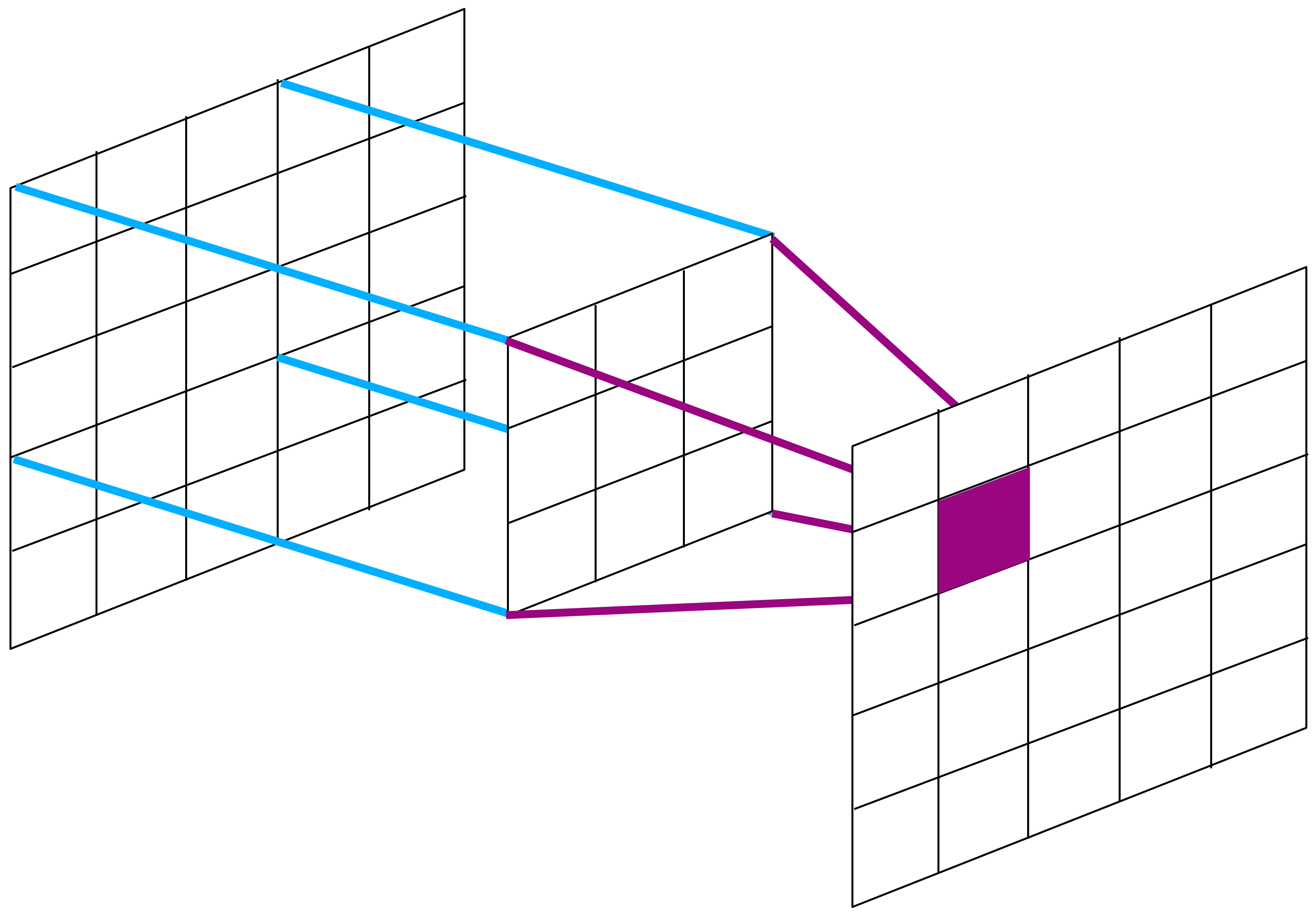
- Whichever algorithm is used, the goal is the same: to extract features from the images that allow you to classify them in some way



Picture from https://towardsdatascience.com/how-do-self-driving-cars-see-13054aee2503

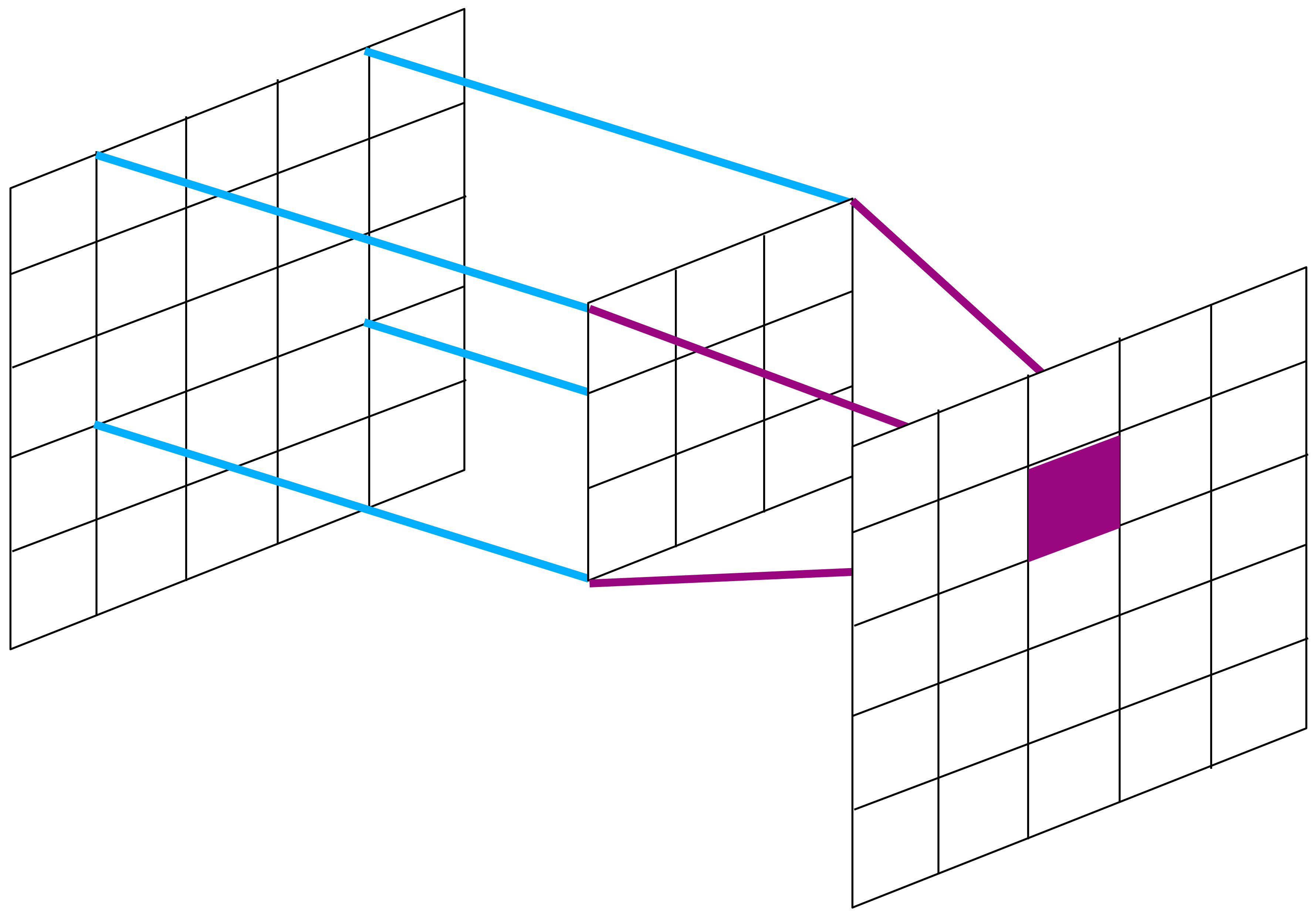
Convolutional Neural Networks

- Convolutional neural networks are designed for image recognition tasks
  - They have been the best performing class of algorithm for the last ~10 years
- Conceptually quite simple: apply filters to images to extract features
  - The filters are learned during training and not predefined
- Will use the example from DUNE neutrino event classification here



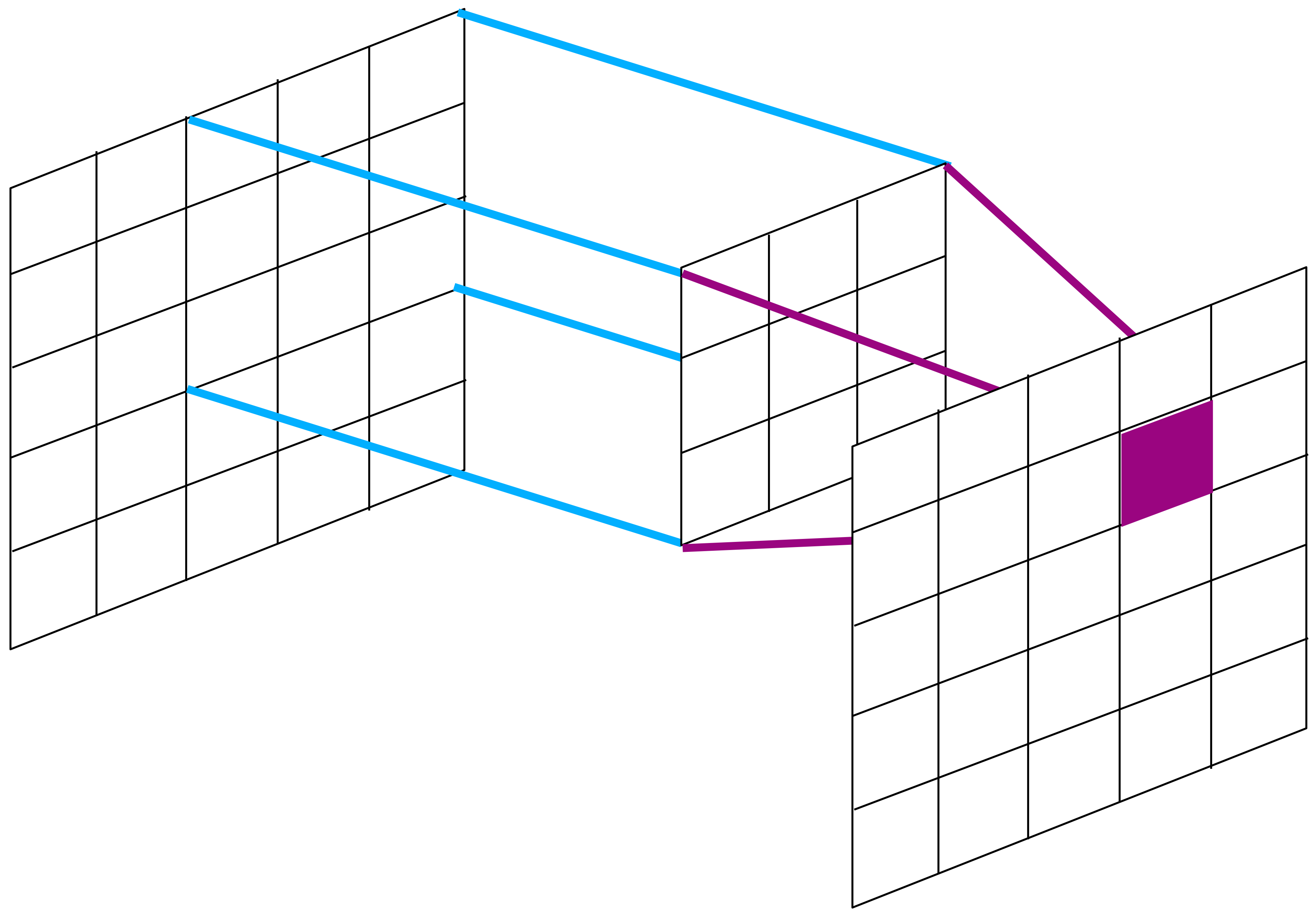
Convolutional Neural Networks

- Convolutional neural networks are designed for image recognition tasks
  - They have been the best performing class of algorithm for the last ~10 years
- Conceptually quite simple: apply filters to images to extract features
  - The filters are learned during training and not predefined
- Will use the example from DUNE neutrino event classification here



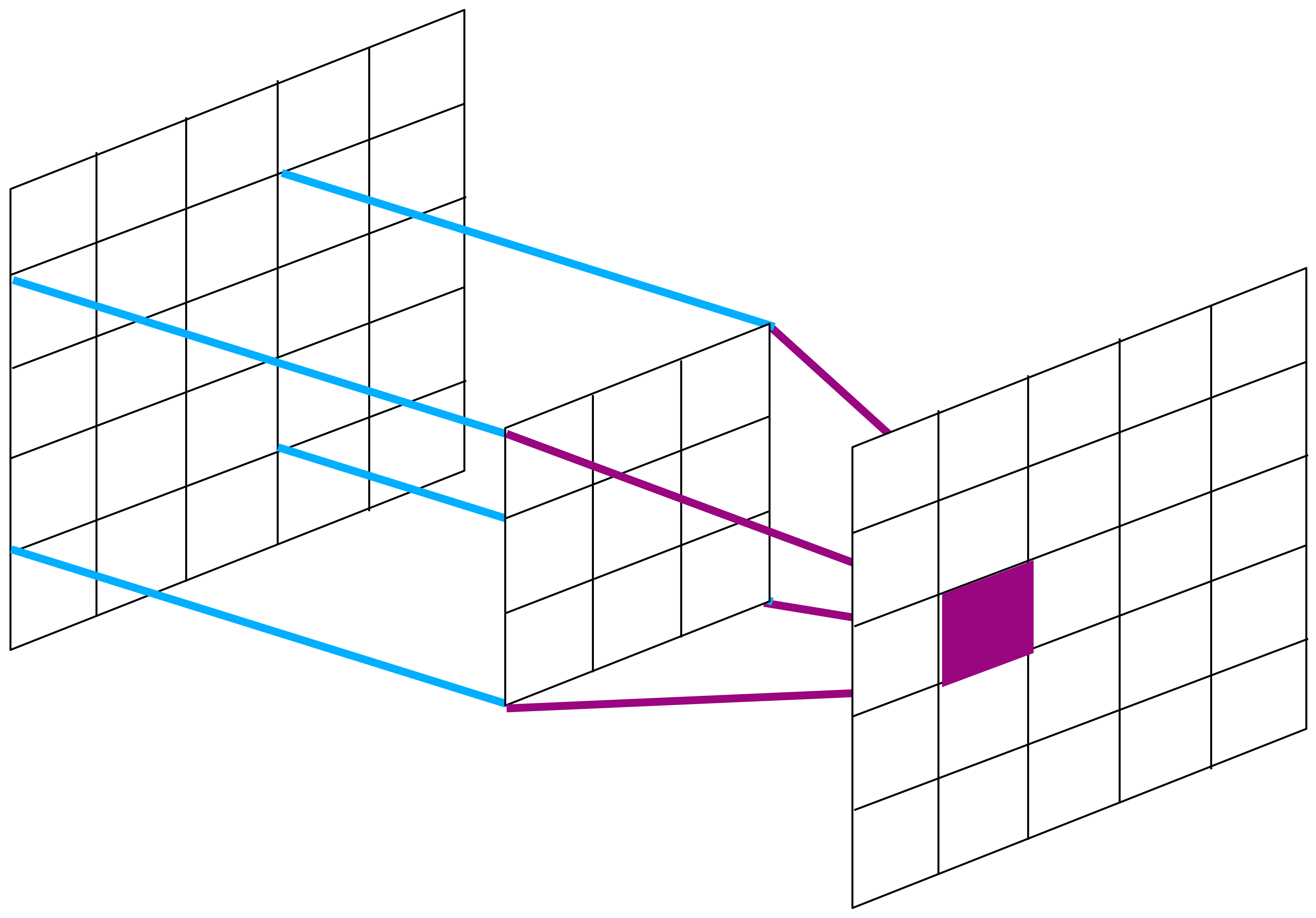
Convolutional Neural Networks

- Convolutional neural networks are designed for image recognition tasks
  - They have been the best performing class of algorithm for the last ~10 years
- Conceptually quite simple: apply filters to images to extract features
  - The filters are learned during training and not predefined
- Will use the example from DUNE neutrino event classification here



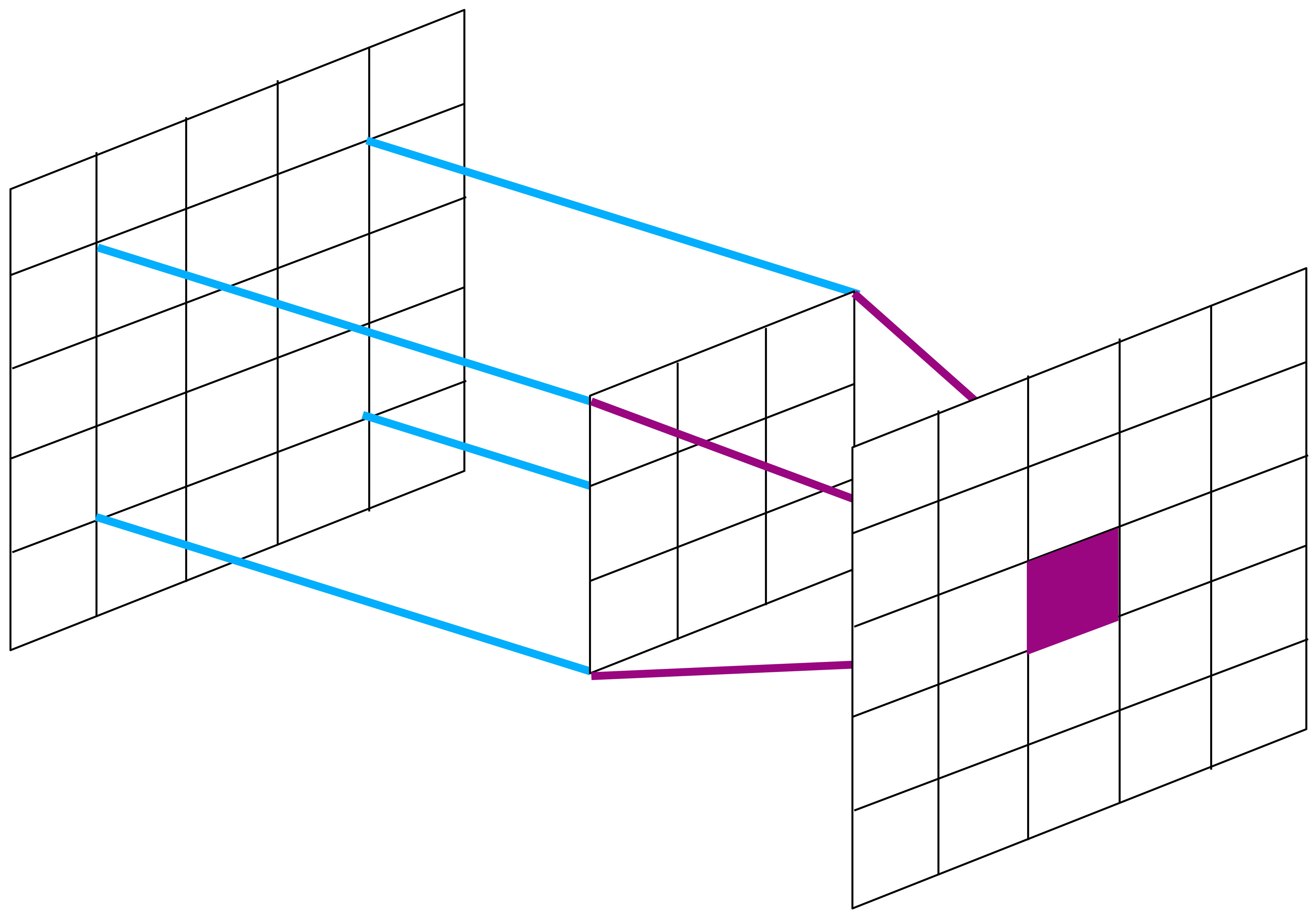
Convolutional Neural Networks

- Convolutional neural networks are designed for image recognition tasks
  - They have been the best performing class of algorithm for the last ~10 years
- Conceptually quite simple: apply filters to images to extract features
  - The filters are learned during training and not predefined
- Will use the example from DUNE neutrino event classification here



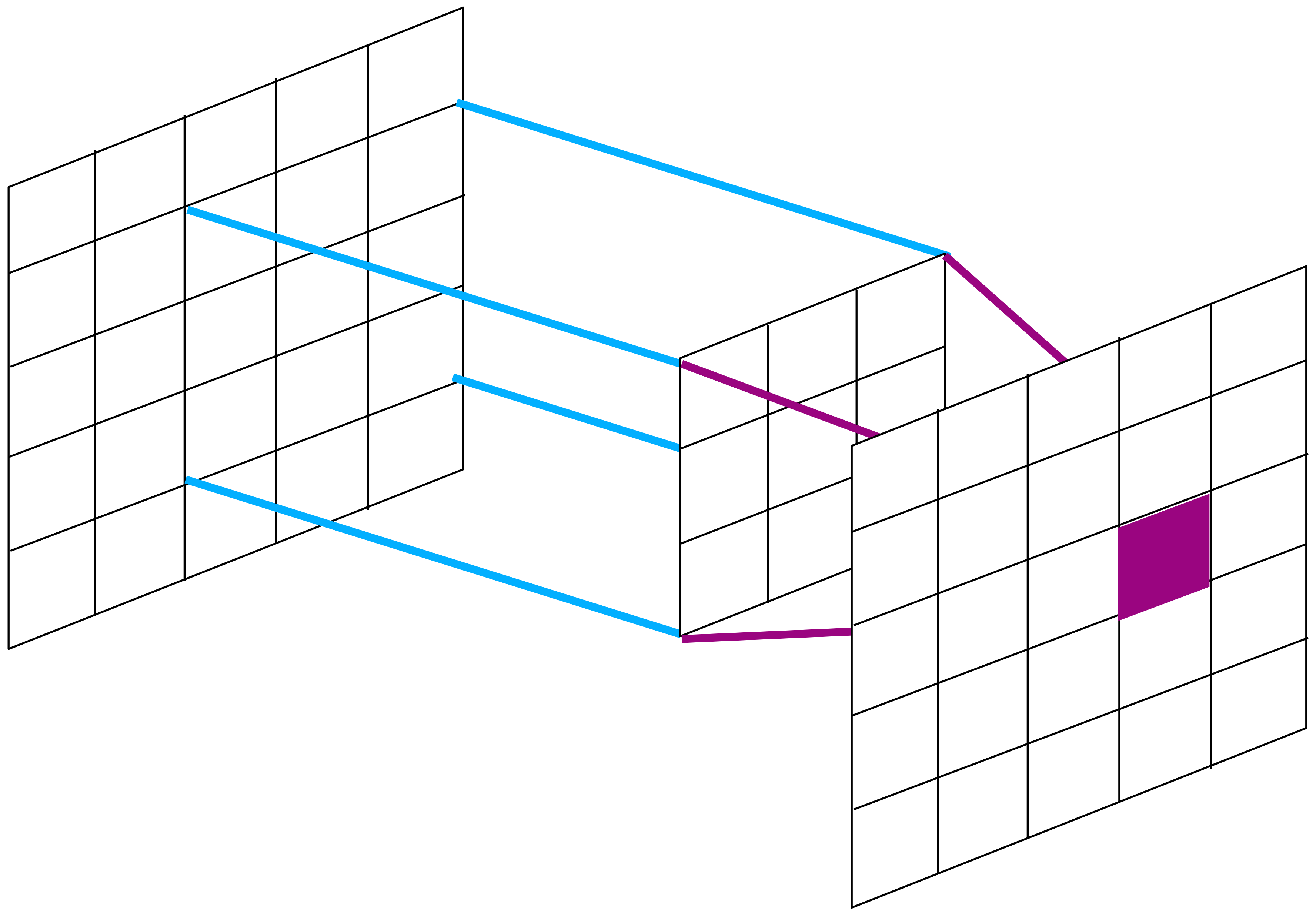
Convolutional Neural Networks

- Convolutional neural networks are designed for image recognition tasks
  - They have been the best performing class of algorithm for the last ~10 years
- Conceptually quite simple: apply filters to images to extract features
  - The filters are learned during training and not predefined
- Will use the example from DUNE neutrino event classification here



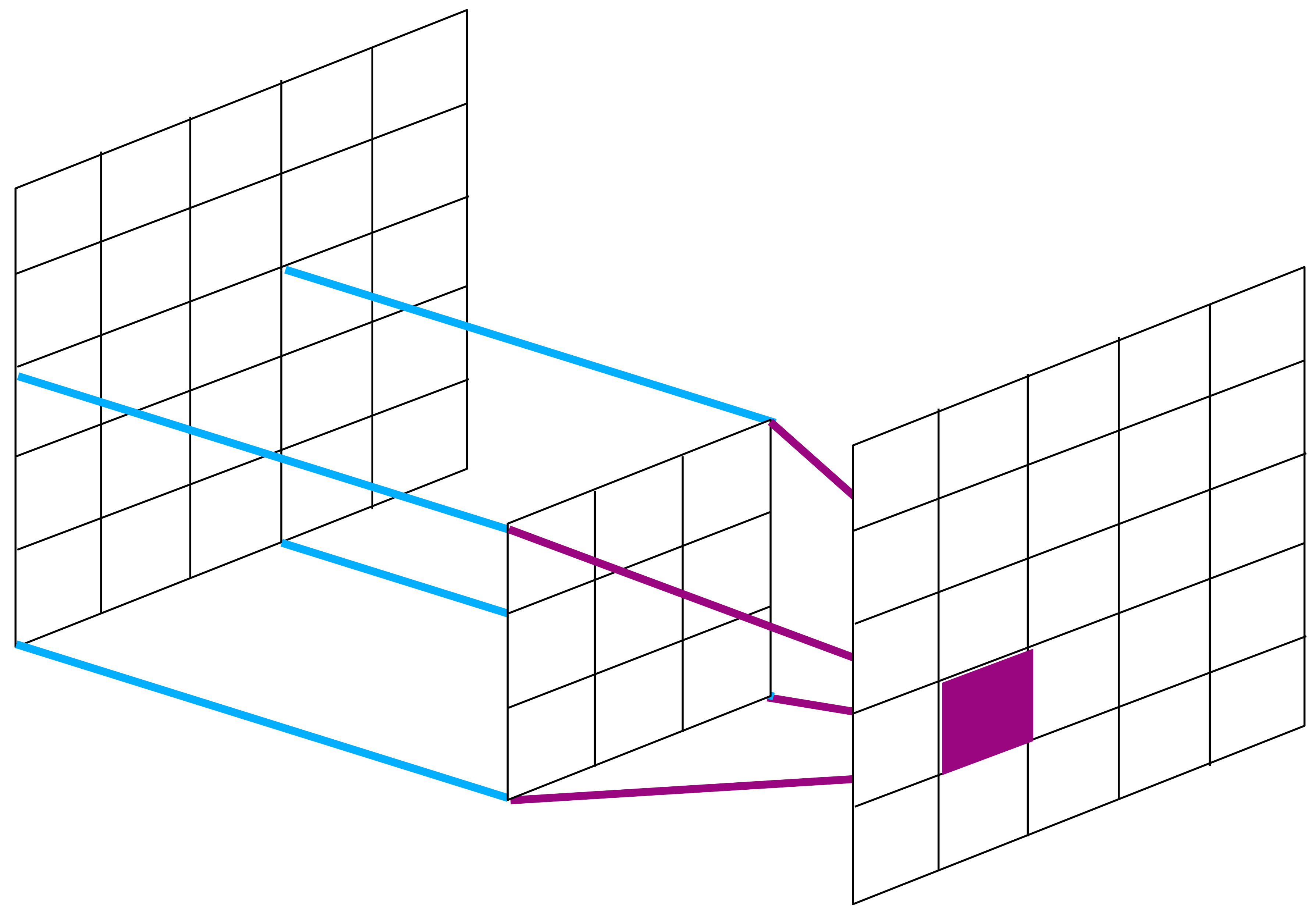
Convolutional Neural Networks

- Convolutional neural networks are designed for image recognition tasks
  - They have been the best performing class of algorithm for the last ~10 years
- Conceptually quite simple: apply filters to images to extract features
  - The filters are learned during training and not predefined
- Will use the example from DUNE neutrino event classification here



Convolutional Neural Networks

- Convolutional neural networks are designed for image recognition tasks
  - They have been the best performing class of algorithm for the last ~10 years
- Conceptually quite simple: apply filters to images to extract features
  - The filters are learned during training and not predefined
- Will use the example from DUNE neutrino event classification here

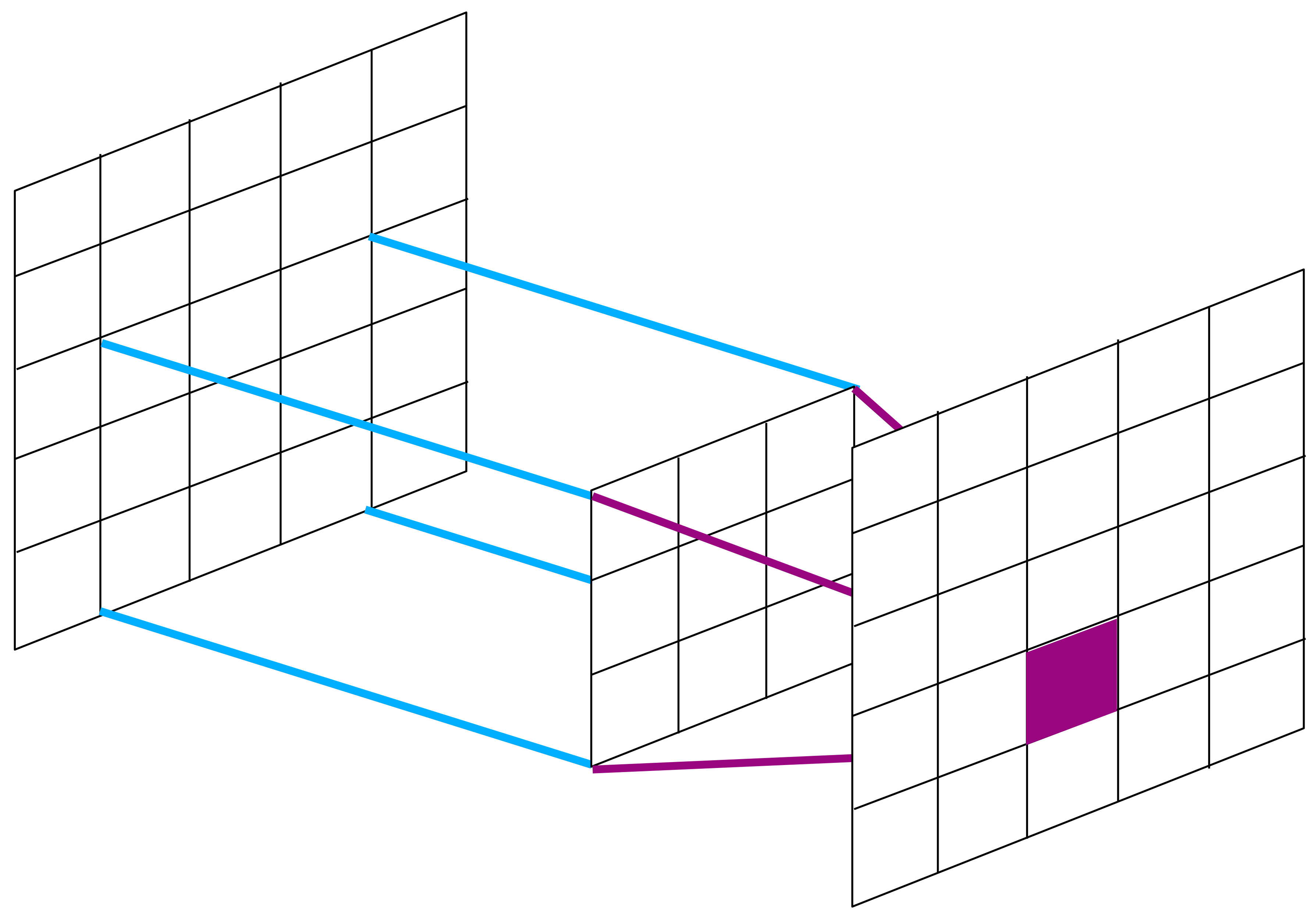


Convolutional Neural Networks

- Convolutional neural networks are designed for image recognition tasks
  - They have been the best performing class of algorithm for the last ~10 years

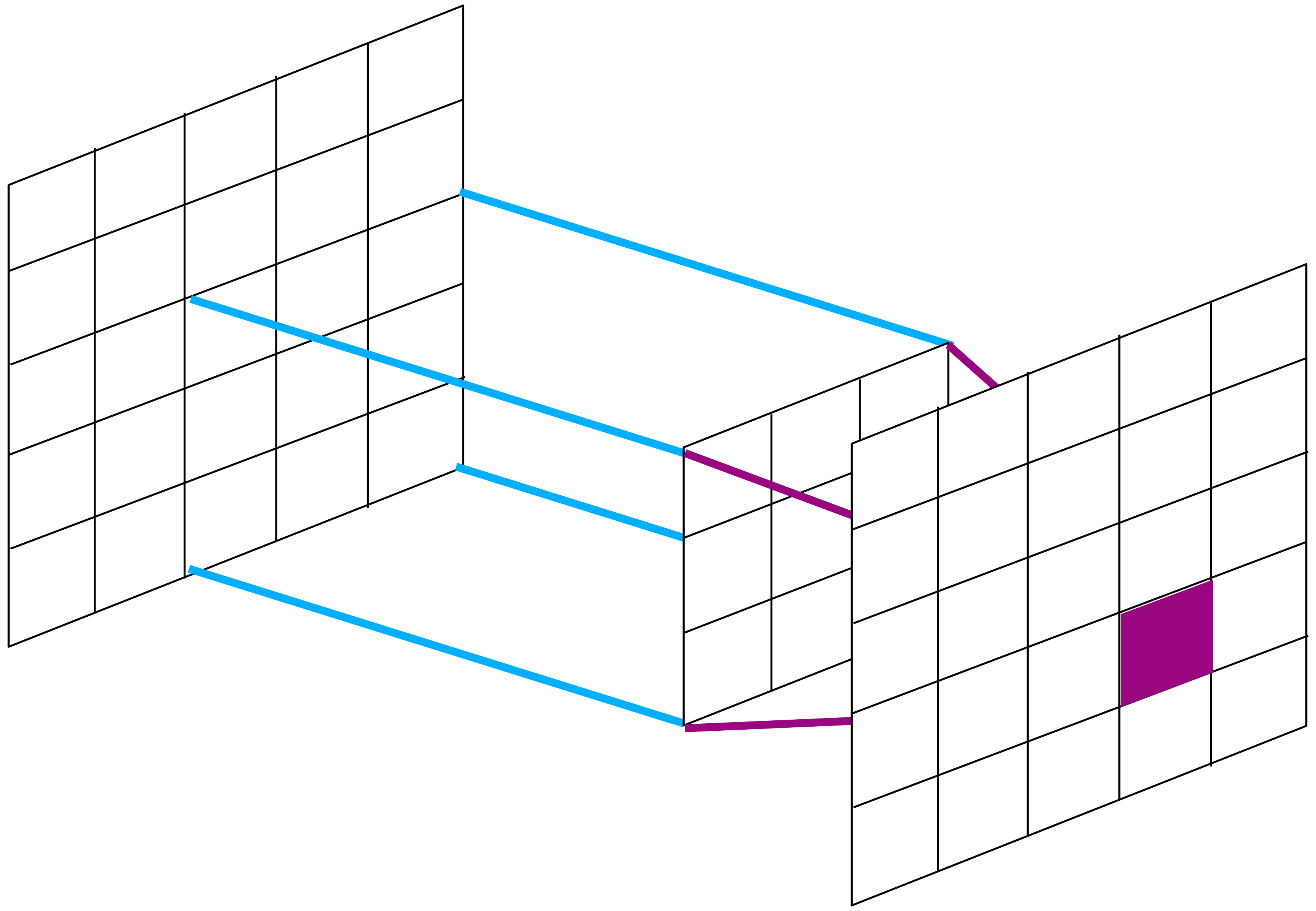
- Conceptually quite simple: apply filters to images to extract features
  - The filters are learned during training and not predefined

- Will use the example from DUNE neutrino event classification here



Convolutional Neural Networks

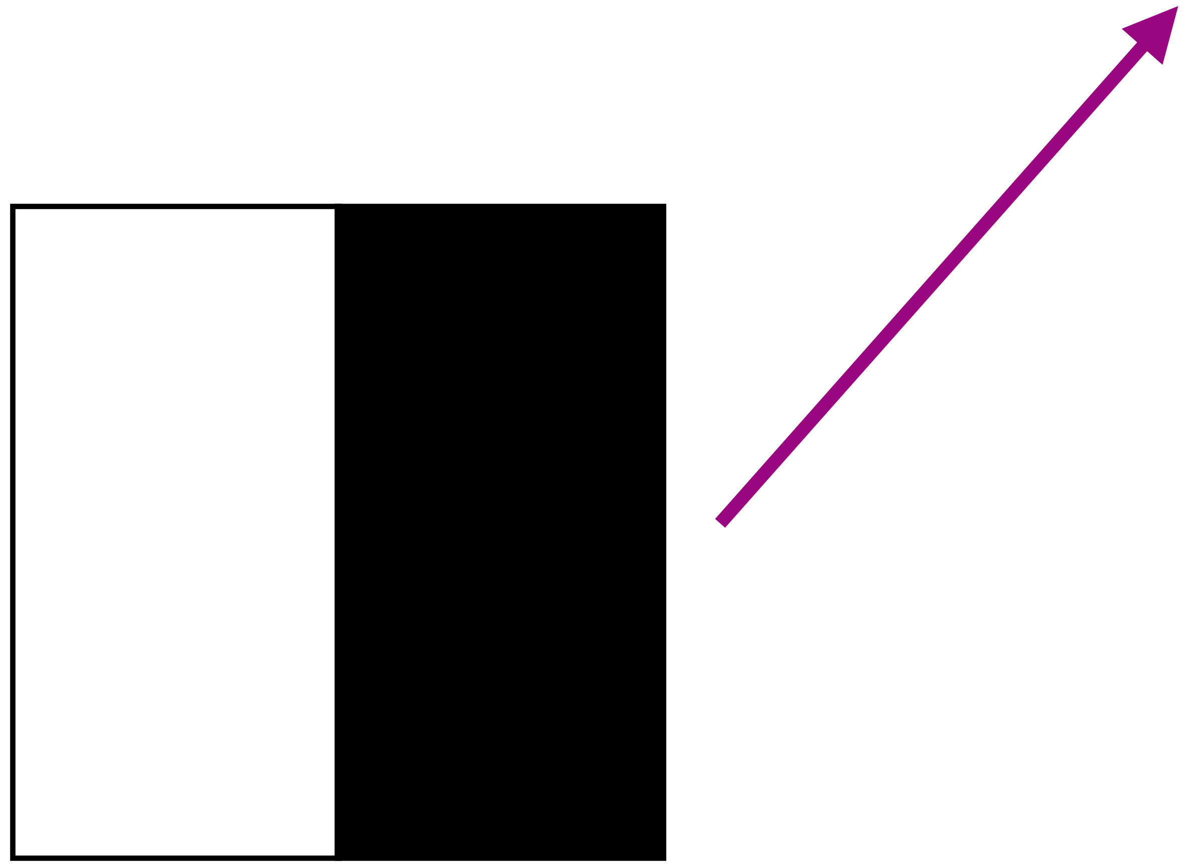
- Convolutional neural networks are designed for image recognition tasks
  - They have been the best performing class of algorithm for the last ~10 years
- Conceptually quite simple: apply filters to images to extract features
  - The filters are learned during training and not predefined
- Will use the example from DUNE neutrino event classification here



Convolutional Neural Networks

- How do we apply a convolution in 2D?
  - Slide the filter over the image and perform element-wise multiplications

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix}$$

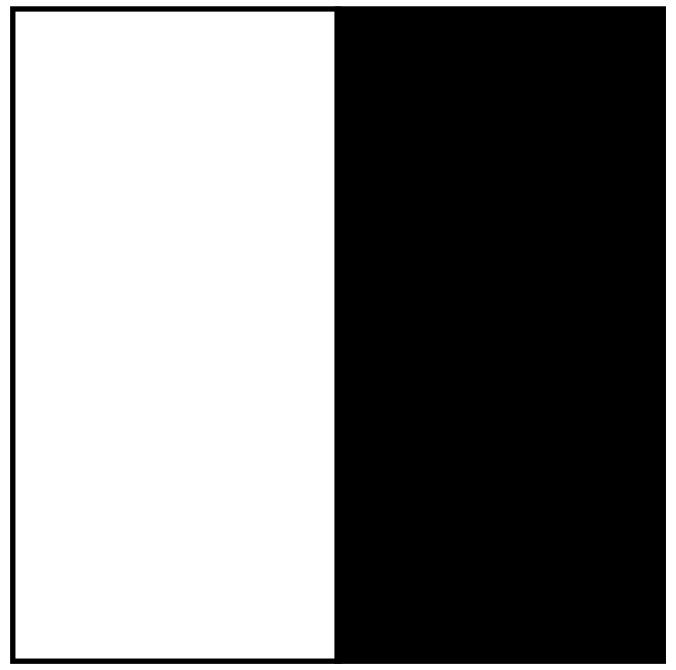


Convolutional Neural Networks

- How do we apply a convolution in 2D?
  - Slide the filter over the image and perform element-wise multiplications

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} \boxed{?} & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix}$$

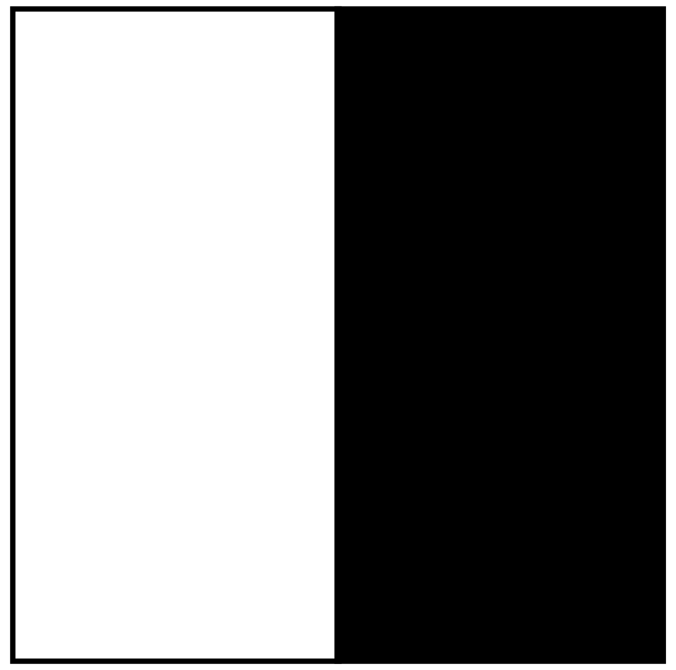
3 * (1*1 + 1*0 + -1 * 1) = 0



Convolutional Neural Networks

- How do we apply a convolution in 2D?
  - Slide the filter over the image and perform element-wise multiplications

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} \mathbf{0} & \boxed{?} & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix}$$



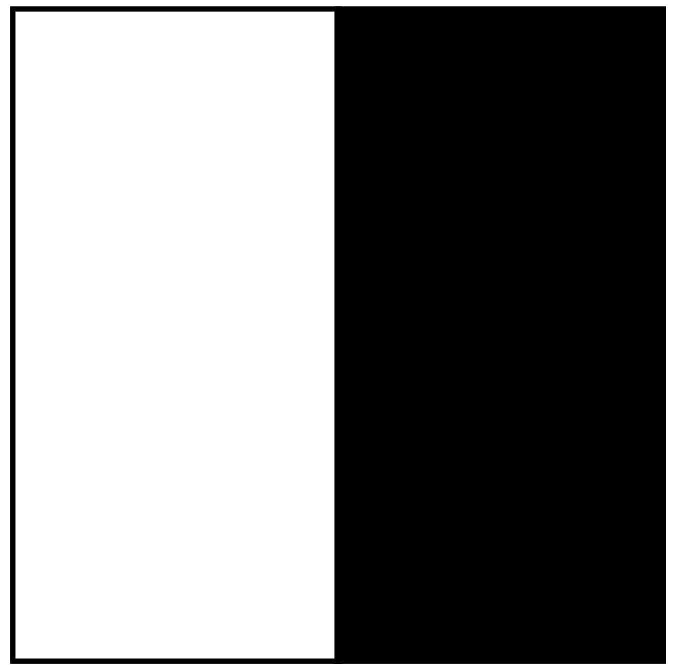
3 * (1*1 + 1*0 + 1 * -1) = 0

3 * (1*1 + 1*0 + 0 * -1) = 3

Convolutional Neural Networks

- How do we apply a convolution in 2D?
  - Slide the filter over the image and perform element-wise multiplications

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} \mathbf{0} & \mathbf{3} & \boxed{?} & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix}$$



3 * (1*1 + 1*0 + 1 * -1) = 0

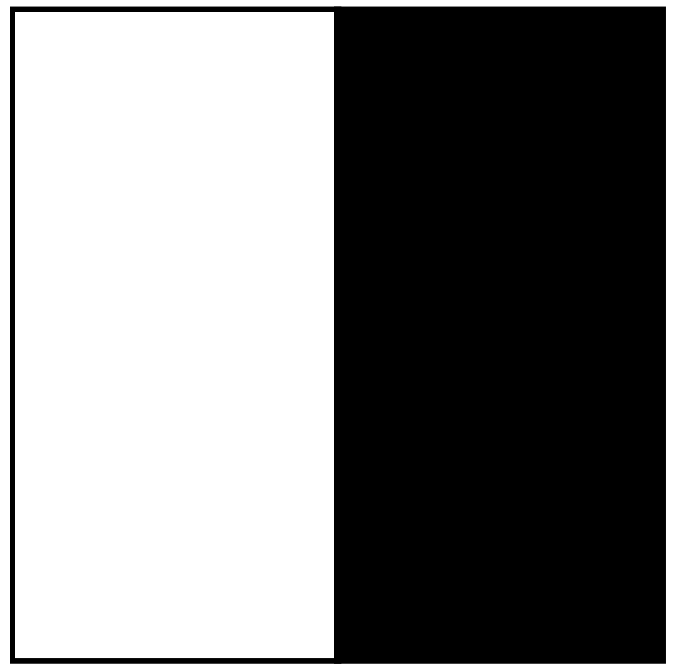
3 * (1*1 + 1*0 + 0 * -1) = 3

3 * (1*1 + 0*0 + 0 * -1) = 3

Convolutional Neural Networks

- How do we apply a convolution in 2D?
  - Slide the filter over the image and perform element-wise multiplications

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} \mathbf{0} & \mathbf{3} & \mathbf{3} & \boxed{?} \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix}$$



3 * (1*1 + 1*0 + 1 * -1) = 0

3 * (1*1 + 1*0 + 0 * -1) = 3

3 * (1*1 + 0*0 + 0 * -1) = 3

3 * (1*1 + 1*0 + 1 * -1) = 0

Convolutional Neural Networks

- How do we apply a convolution in 2D?
  - Slide the filter over the image and perform element-wise multiplications

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 3 & 3 & 0 \\ 0 & 3 & 3 & 0 \\ 0 & 3 & 3 & 0 \\ 0 & 3 & 3 & 0 \end{bmatrix}$$

  - The filter gives a response where the vertical edge in our image is
  - This filter is a vertical edge finder

Convolutional Neural Networks

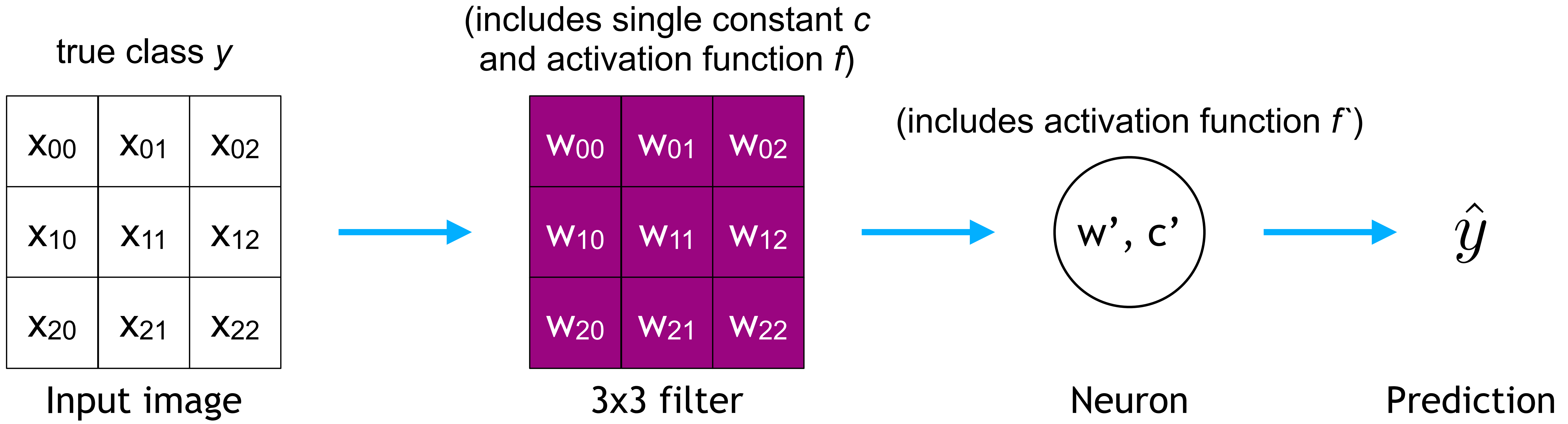
- How do we apply a convolution in 2D?
  - Slide the filter over the image and perform element-wise multiplications

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

  - Let’s check what happens with a horizontal edge finder
  - The filter produces no response since there are no horizontal edges

Convolutional Neural Networks

- Let's have a look at some maths
  - Assume we have a very simple (and unrealistic) network:

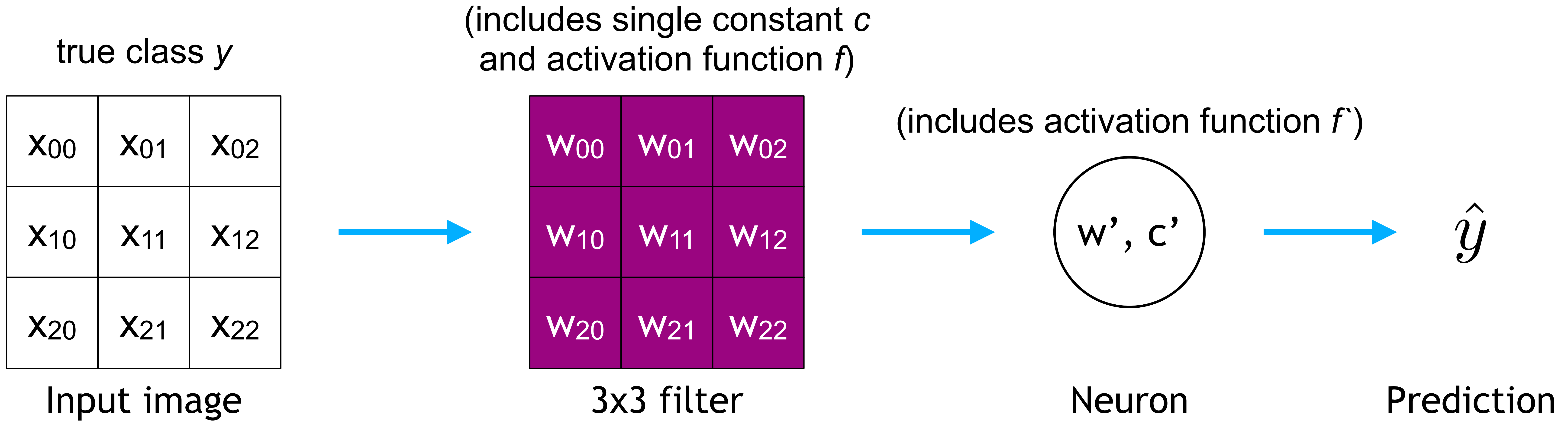


- The convolution here returns a single number since the image and filter are the same size. It is an element-wise matrix multiplication

$$a = f\left(\left[\sum_i \sum_j x_{ij} w_{ij}\right] + c\right)$$

Convolutional Neural Networks

- Let's have a look at some maths
  - Assume we have a very simple (and unrealistic) network:

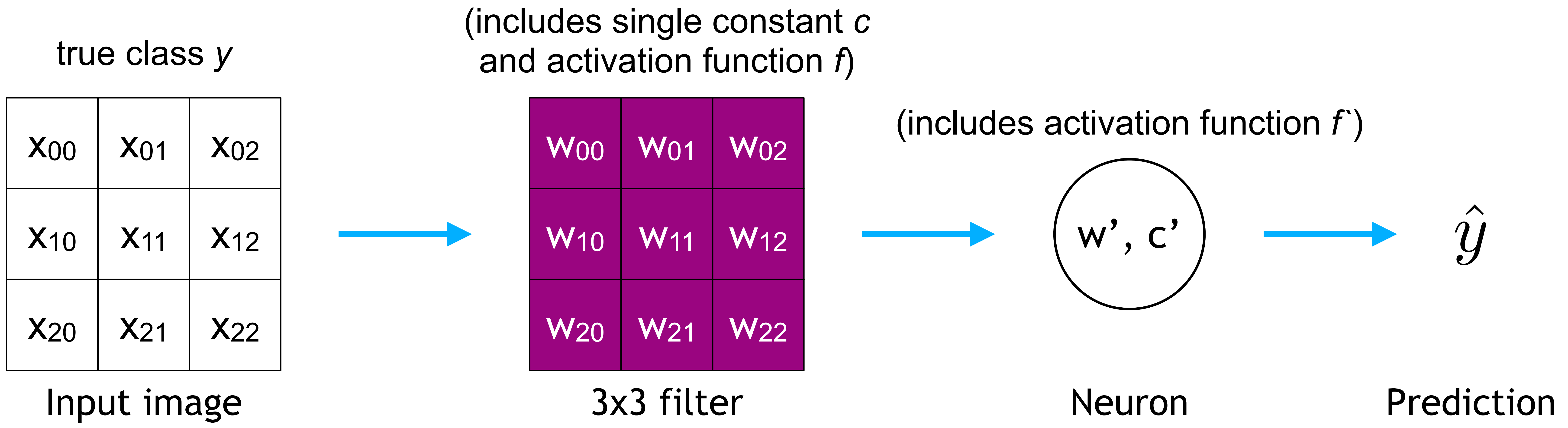


- We then propagate this activation *a* through the single neutron

$$\hat{y} = f'\left(w'a + c'\right) = f'\left(w' f\left(\left[\sum_i \sum_j x_{ij} w_{ij}\right] + c\right) + c'\right)$$

Convolutional Neural Networks

- Let's have a look at some maths
  - Assume we have a very simple (and unrealistic) network:



- Again, there is no magic here, just maths!
  - I won't write down the back propagation here, but you just need to do chain-rule differentiation from right to left
    - Once for each of the twelve parameters!

Convolutional Neural Networks

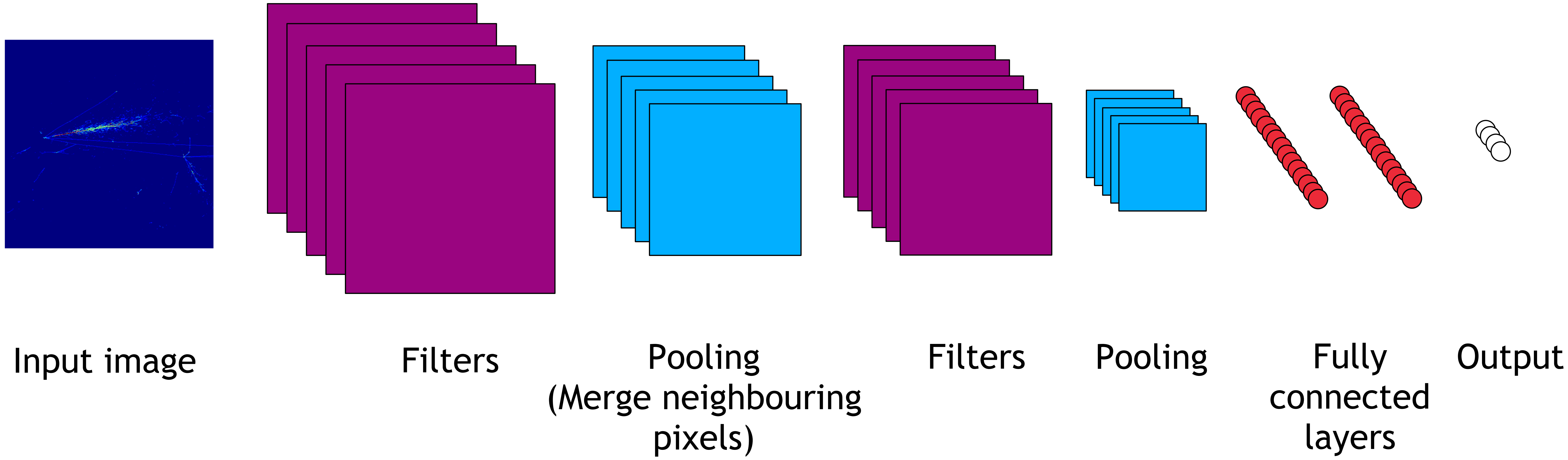
- Each element of the filter is basically like the single neuron that we saw earlier
  - So we have nine weights in a 3x3 filter plus a constant *c*

$$\begin{bmatrix} w_{00} & w_{01} & w_{02} \\ w_{10} & w_{11} & w_{12} \\ w_{20} & w_{21} & w_{22} \end{bmatrix}$$

  - These are the weights that are learned during the training
  - Thus, we do not tell the CNN which filters to use
  - It learns which filters it needs to extract the information that it needs to solve the problem

Convolutional Neural Networks

- The output from each filter then forms the basis of the next layer which can include further filters



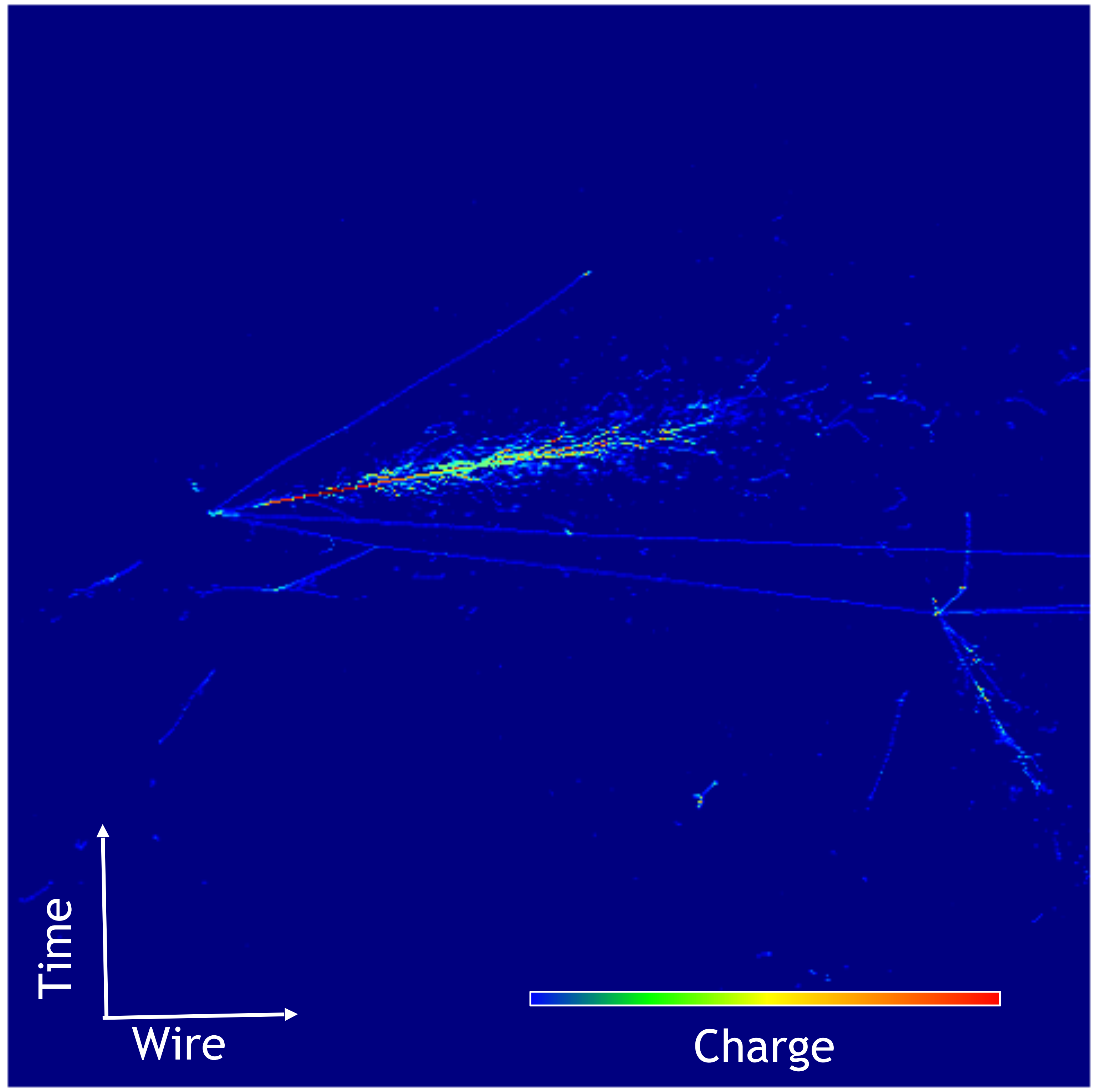
- Different architectures can be considerably more complex than the above toy example

Deep Learning in LArTPCs

- LArTPCs have fine detail of interactions and lend themselves to image recognition techniques

- Things we could classify
  - Type of neutrino that interacted
  - Individual particle types
  - Individual hits… is this pixel part of a track- or shower-like energy deposit?

- Things to measure (regression tasks)
  - The neutrino energy
  - Interaction vertex location, etc

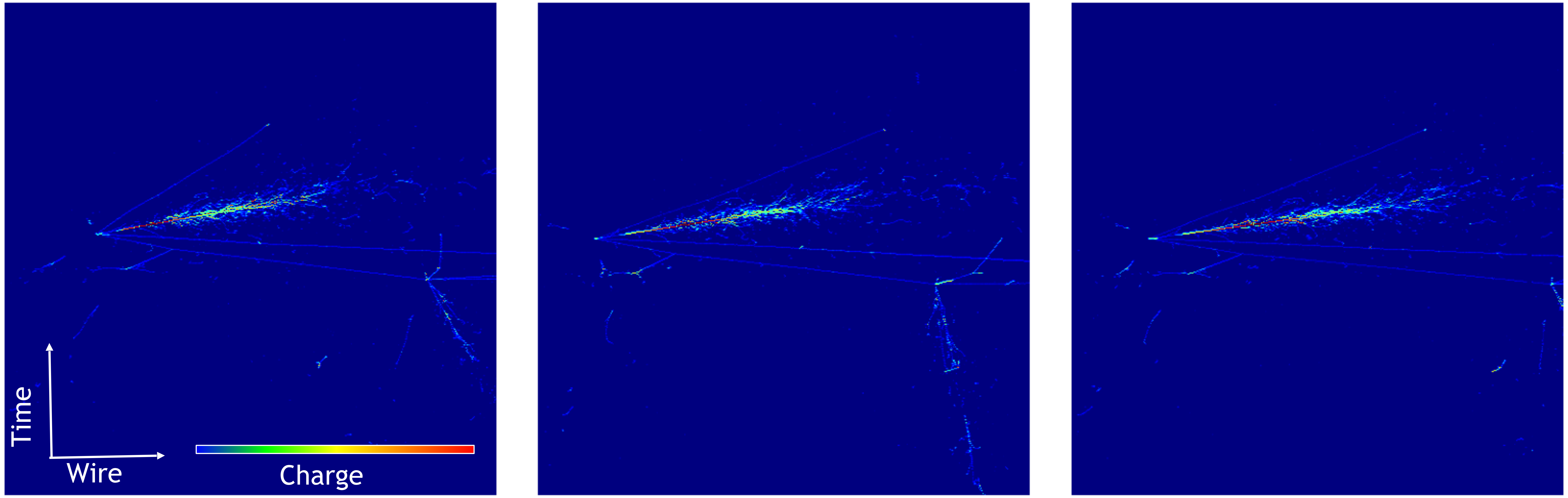


Example from the DUNE Simulation

Neutrino Interaction Images

- Build images using reconstructed hits in the (wire number, time) parameter space
- The TPC has three readout views, so we make three images (we could use one image with red / green / blue channels)

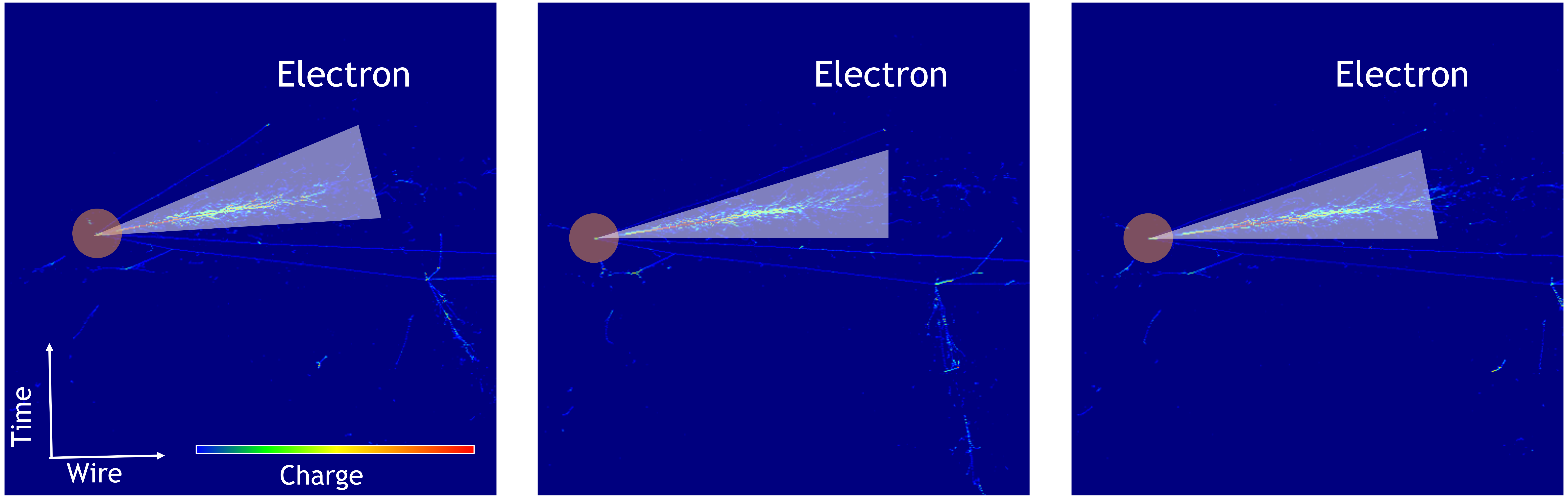
DUNE Far Detector Simulation CC ν_e interaction



Neutrino Interaction Images

- By eye you can easily see features that would help you to identify this event as an electron neutrino interaction
- We can see there is an electromagnetic shower emanating from the primary vertex

DUNE Far Detector Simulation CC ν_e interaction

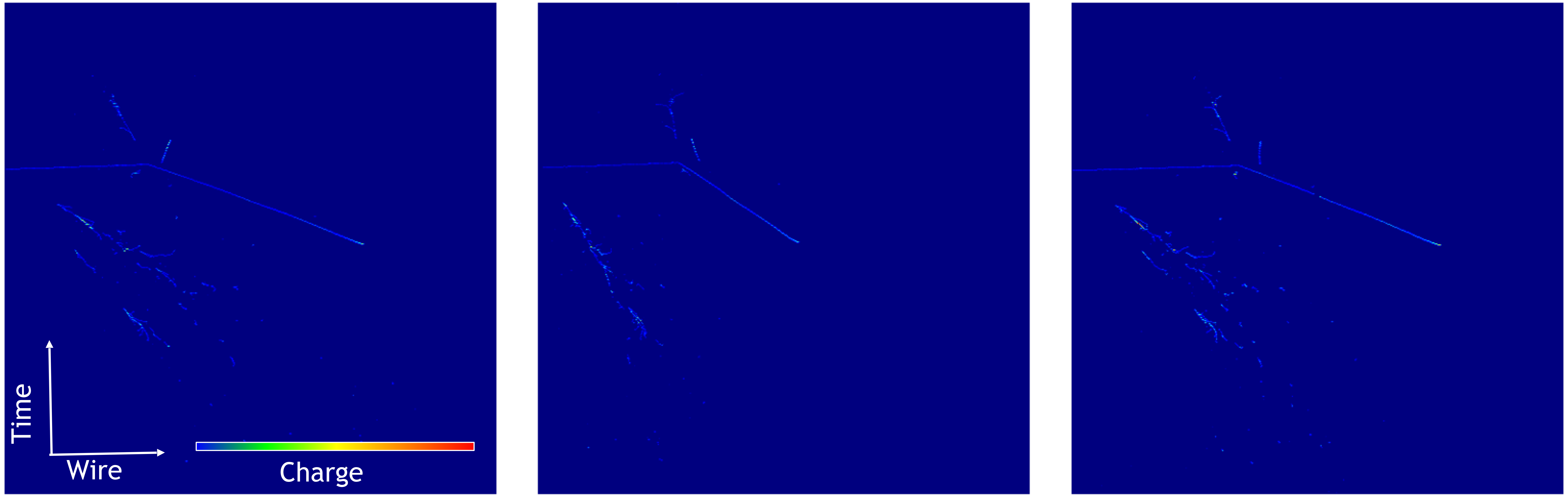


Neutrino Interaction Images

- Similarly, you can tell that this is a background interaction - a neutral current event producing a neutral pion

- We can see two electromagnetic showers not emanating from the primary vertex

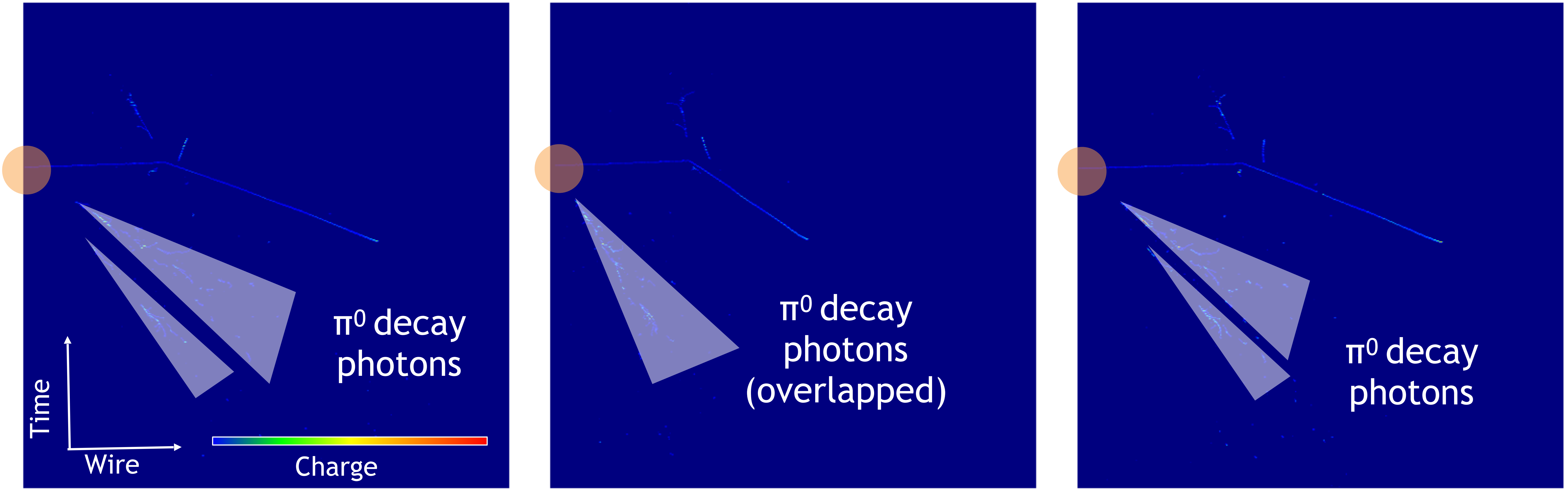
DUNE Far Detector Simulation NCπ^0 interaction



Neutrino Interaction Images

- Similarly, you can tell that this is a background interaction - a neutral current event producing a neutral pion
- We can see two electromagnetic showers not emanating from the primary vertex

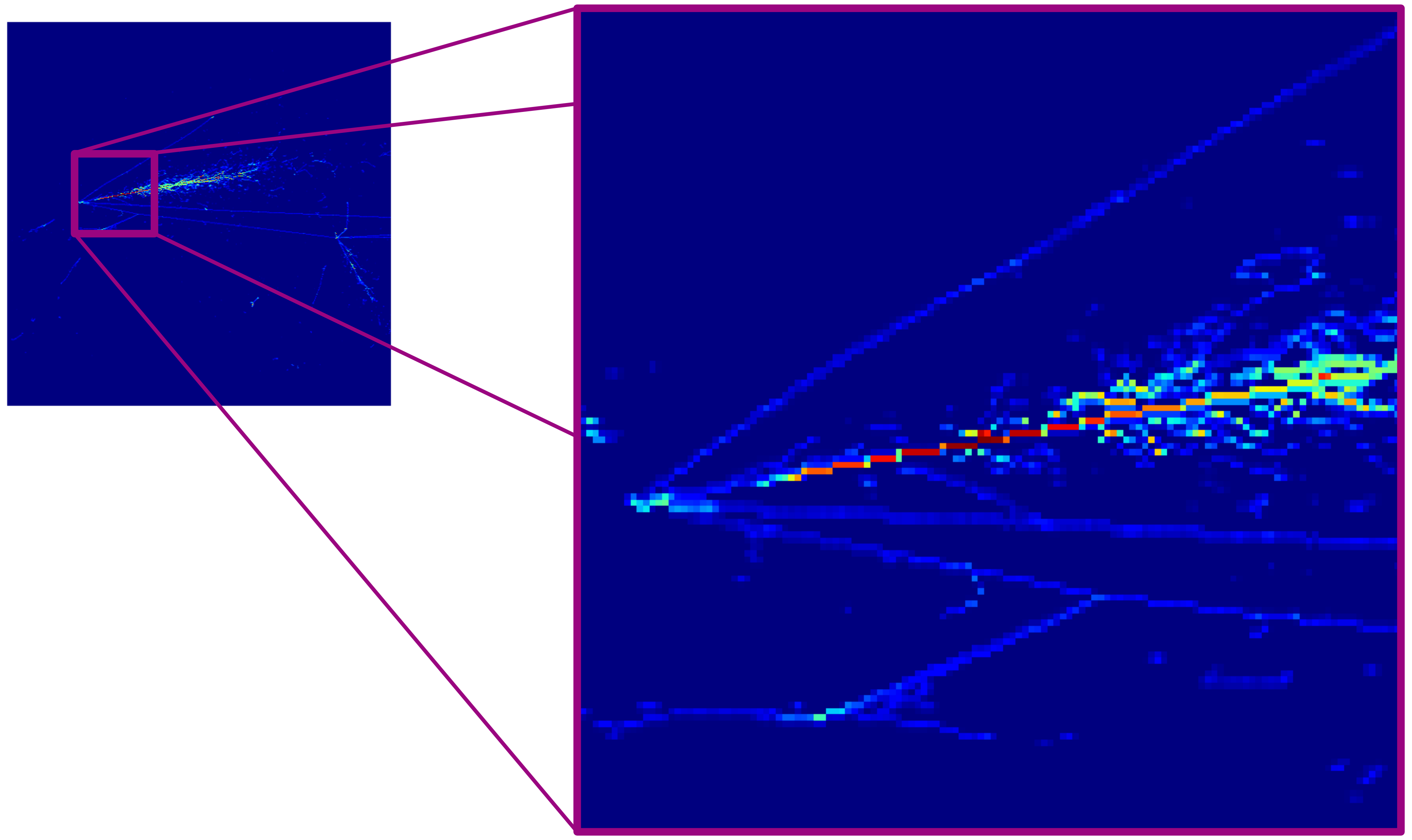
DUNE Far Detector Simulation NCπ^0 interaction



Convolutional Neural Networks

- CNNs are used to classify images by applying filters to small patches of the image (using a convolution)

- Scans over the image with a number of N x N pixel filters

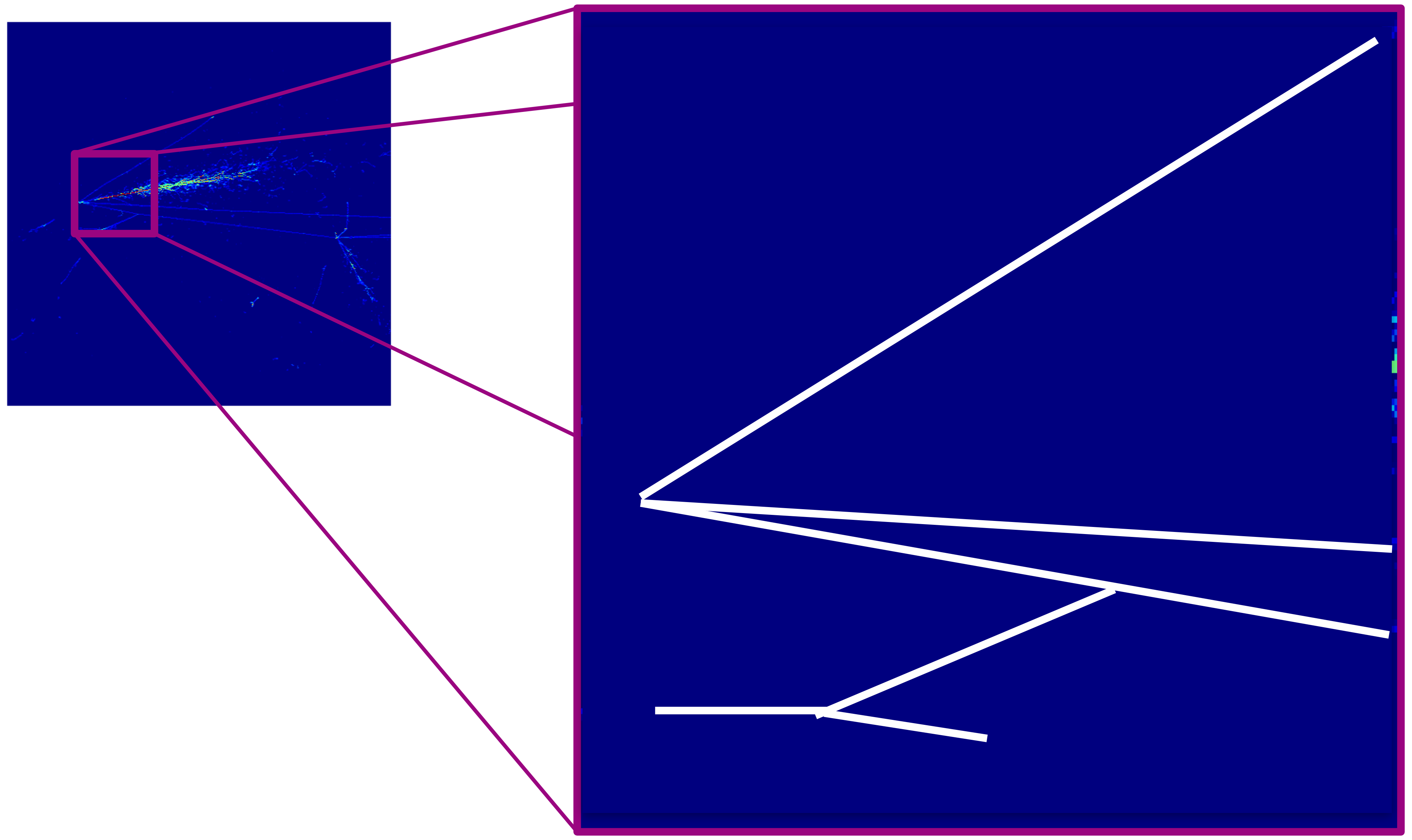


- Each filter extracts some feature from the image

NB: This is just a visual example. Filters typically have sizes of 3x3 or 7x7, much smaller than in this demonstration

Convolutional Neural Networks

- CNNs are used to classify images by applying filters to small patches of the image (using a convolution)
- Scans over the image with a number of N x N pixel filters



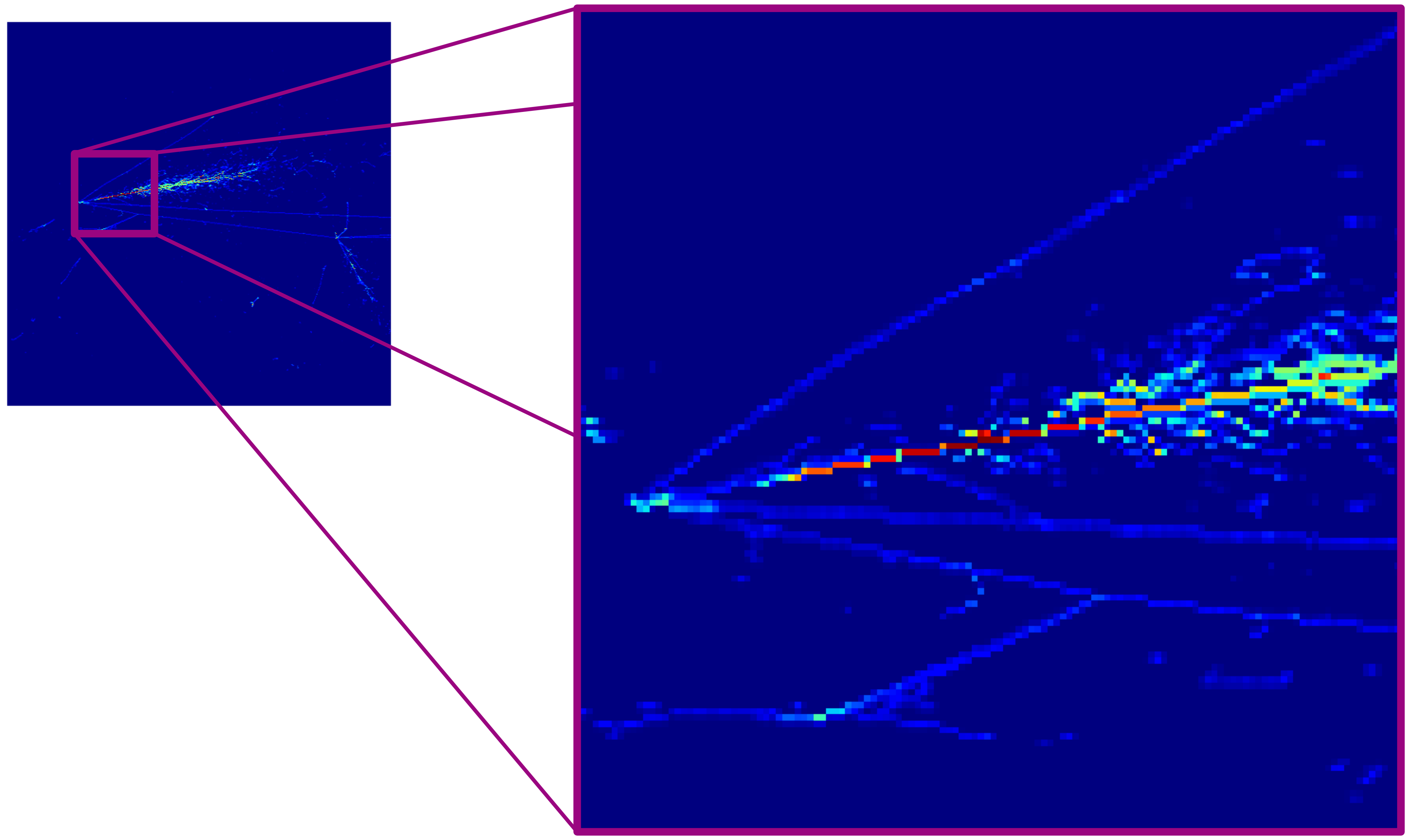
- Each filter extracts some feature from the image
- For example, filter one might find tracks

NB: This is just a visual example. Filters typically have sizes of 3x3 or 7x7, much smaller than in this demonstration

Convolutional Neural Networks

- CNNs are used to classify images by applying filters to small patches of the image (using a convolution)

- Scans over the image with a number of N x N pixel filters



- Each filter extracts some feature from the image

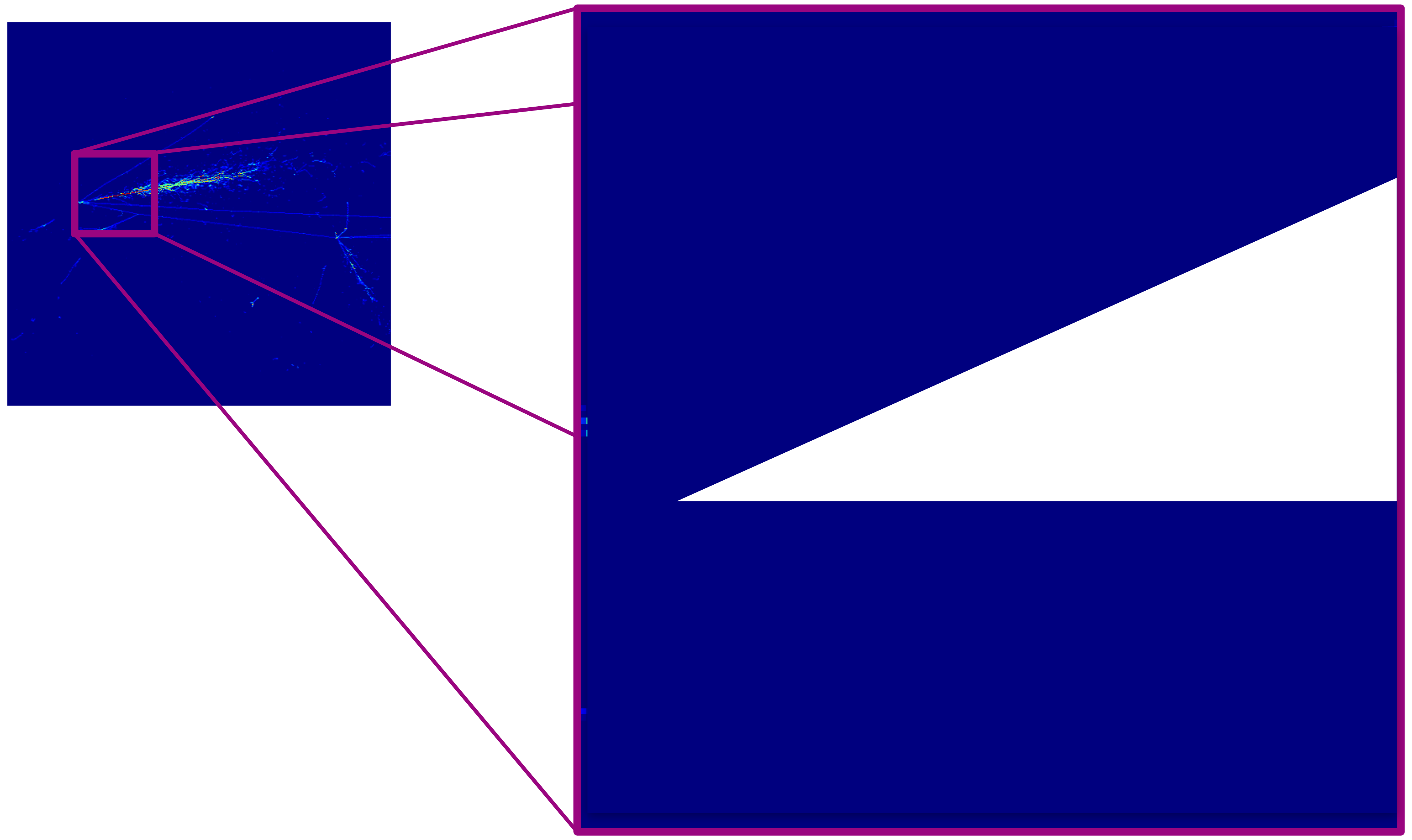
- For example, filter one might find tracks

NB: This is just a visual example. Filters typically have sizes of 3x3 or 7x7, much smaller than in this demonstration

Convolutional Neural Networks

- CNNs are used to classify images by applying filters to small patches of the image (using a convolution)

- Scans over the image with a number of N x N pixel filters



- Each filter extracts some feature from the image

- For example, filter one might find tracks

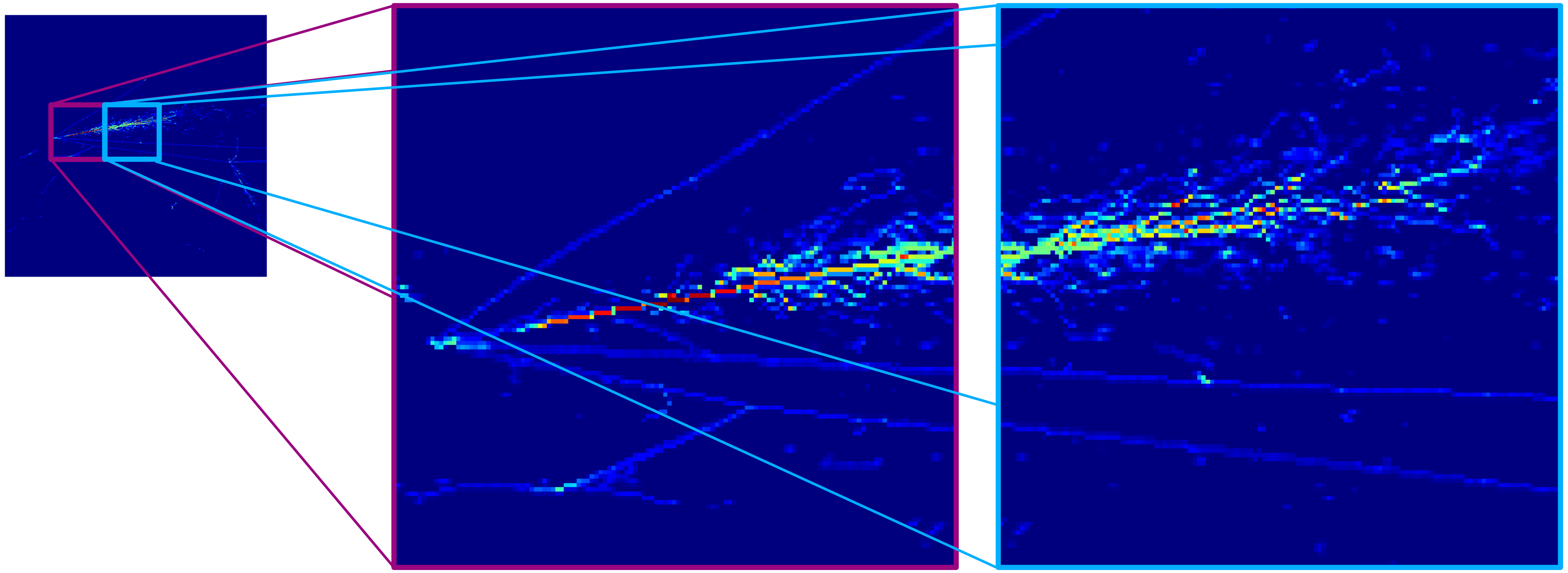
- Filter two might look for showers

NB: This is just a visual example. Filters typically have sizes of 3x3 or 7x7, much smaller than in this demonstration

Convolutional Neural Networks

- CNNs are used to classify images by applying filters to small patches of the image (using a convolution)

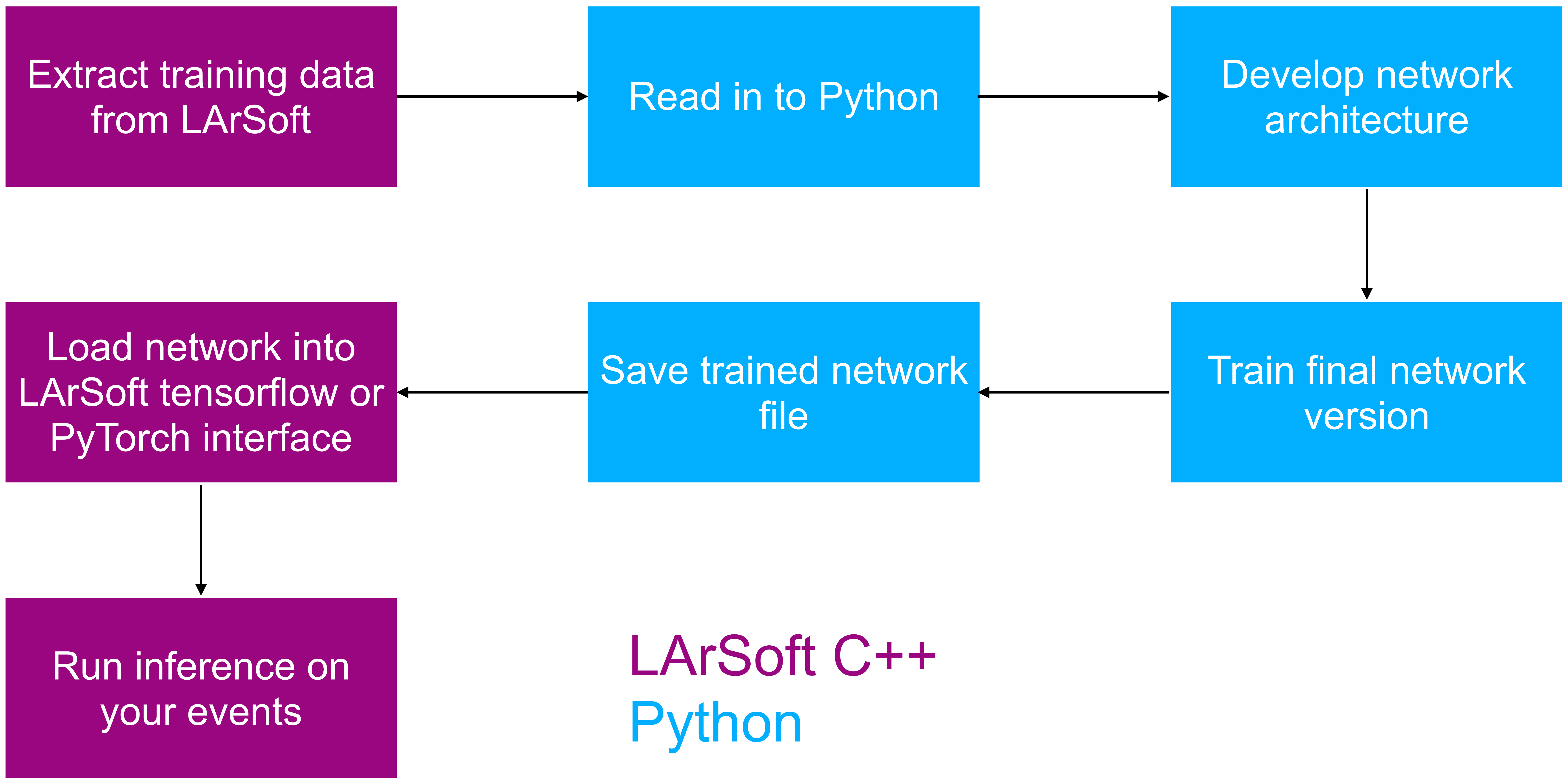
- Scans over the image with N x N pixel filters



- Then move onto the next patch of the image and repeat the process

Workflow using LArSoft

- The workflow can be a little convoluted, this is the one we use in DUNE for the CVN (a neutrino event classifier):



Workflow using LArSoft

- Write some sort of analysis module to extract the training data you require:
  - CNNs typically use 2D hits for each wire plane in the format of wire vs time images
  - One could also use natively 3D techniques such as graph neural networks and extract the 3D space points instead
- For the DUNE CVN we save this as a type of compressed file that we can easily load into python software
- Our whole development cycle takes place in python

Workflow using LArSoft

- Once we are happy with our trained network then we export the trained architecture as a tensorflow .pb file
- We wrote a C++ tensorflow interface inside LArSoft where we load this network
  - We can then pass the data (that we previously extracted) directly into tensorflow to obtain the results for each event
- An equivalent interface for PyTorch also exists
  - There is one in Pandora
- Development in python lets us do things much more quickly and in a light-weight environment

Summary

- Deep learning techniques are widespread in HEP and neutrino physics
  - Typically using CNNs that came from image recognition
- Field is rapidly advancing and taking advantage of progress in computer science
- Many other techniques becoming popular
  - Sparse CNNs
  - Graph neural networks
  - Generative Adversarial Networks
- Lots of resources available online

UNIVERSITY OF CAMBRIDGE

Backup Slides and Other Resources

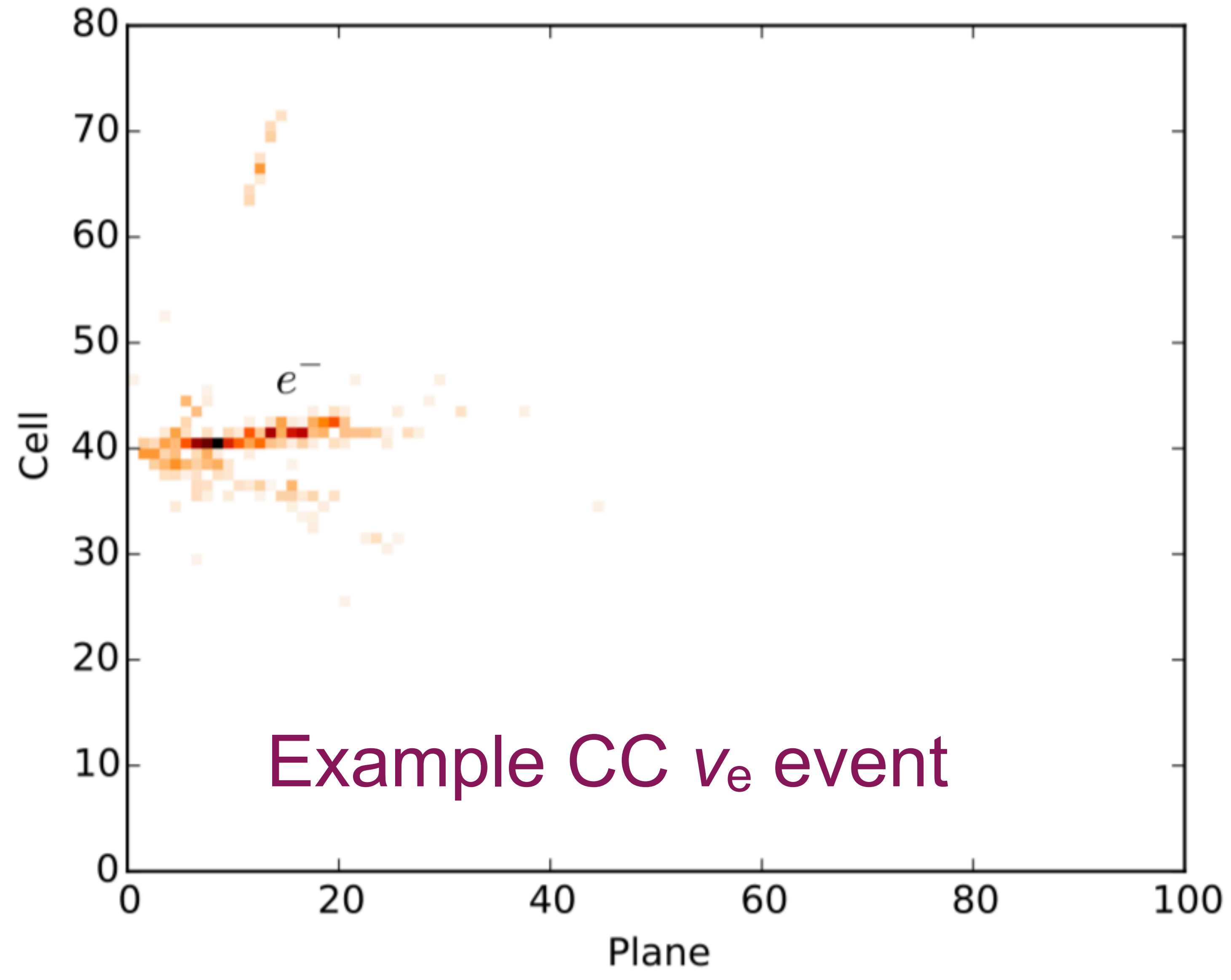
Selected CNN Highlights

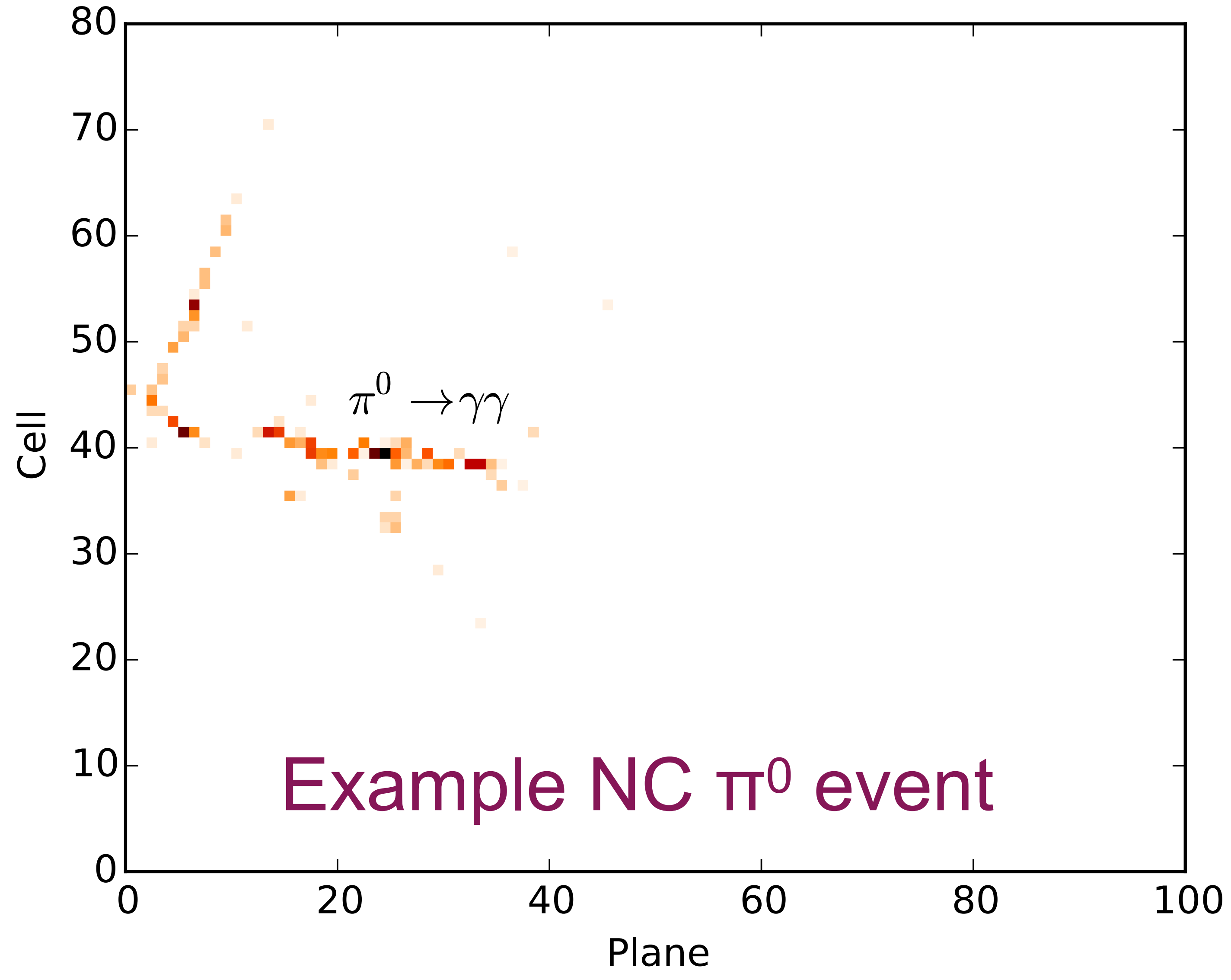
- Some examples that you can investigate:
  - NOvA
    - Neutrino ID CNN[1] was the first CNN used in neutrino physics
    - Particle identification[2]
  - MicroBooNE:
    - Example of semantic segmentation to select neutrino events[3]
    - Particle identification[4]
  - DUNE neutrino ID CNN[5]
    - Very powerful classifier based on the SE-ResNet[6,7] architecture

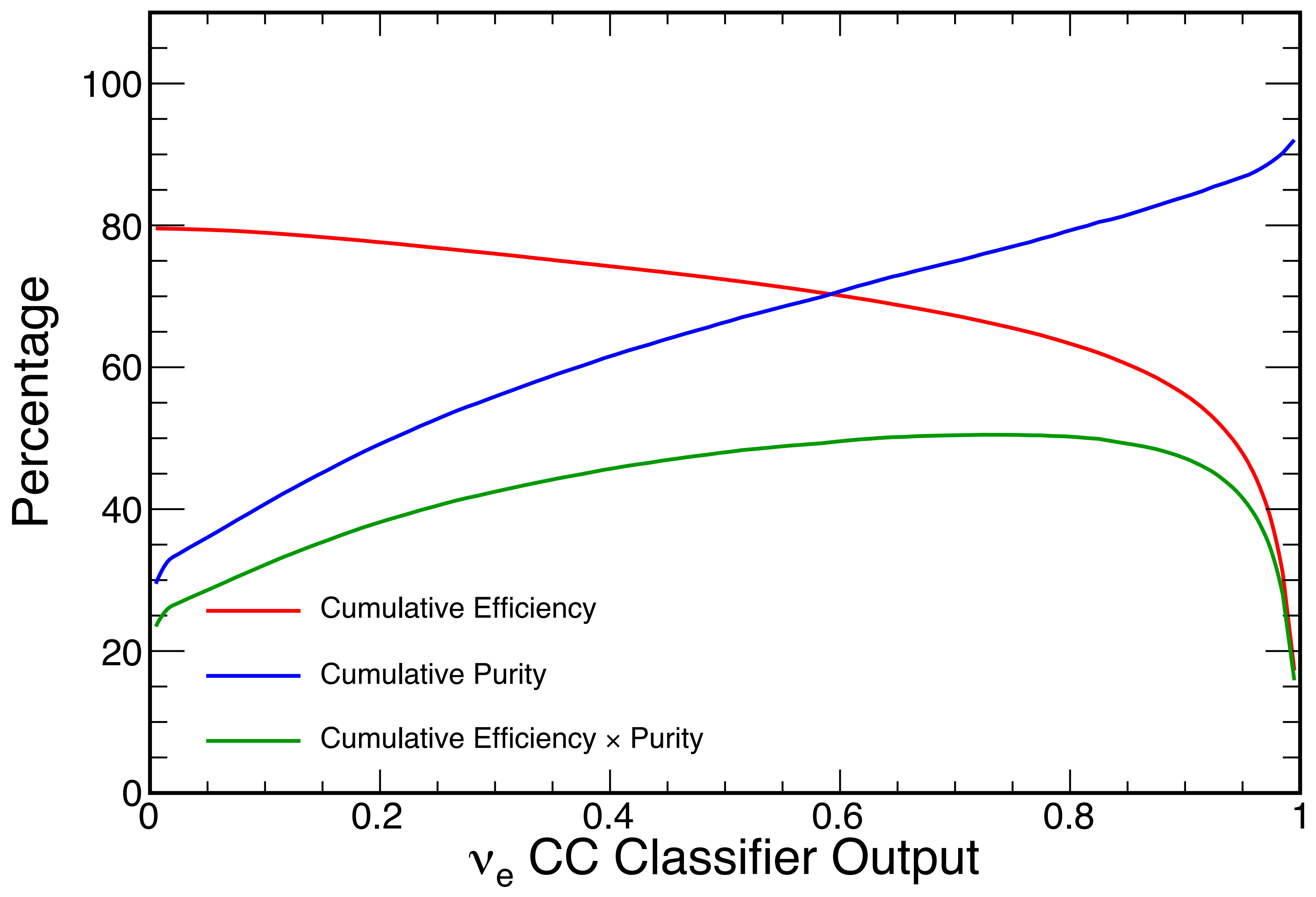
[1] NOvA Collaboration, A convolutional neural network neutrino event classifier, JINST **11** 09 P09001, 2016
[2] NOvA Collaboration, Context-enriched identification of particles with a convolutional network for neutrino events, Phys. Rev. D **100** 073005, 2019
[3] MicroBooNE Collaboration, Convolutional neural networks applied to neutrino events in a liquid argon time projection chamber, JINST **12** 03 P03011, 2017
[4] MicroBooNE Collaboration, Deep neural network for pixel-level electromagnetic particle identification in the MicroBooNE liquid argon time projection chamber, Phys. Rev. D **99** 092001, 2019
[5] DUNE Collaboration, Neutrino interaction classification with a convolutional neural network in the DUNE far detector, Phys. Rev. D **102** 092003, 2020
[6] H. Kaiming et al., Deep residual learning for image recognition, CoRR, arXiv 1512.03385, 2015
[7] J. Hu et al., Squeeze-and-Excitation Networks, arXiv 1709.01507, 2017

Selected CNN Highlights - NOvA

- Trailblazed the use of CNNs in neutrino physics
  - Scintillator detector that is less fine-grained than LArTPCs



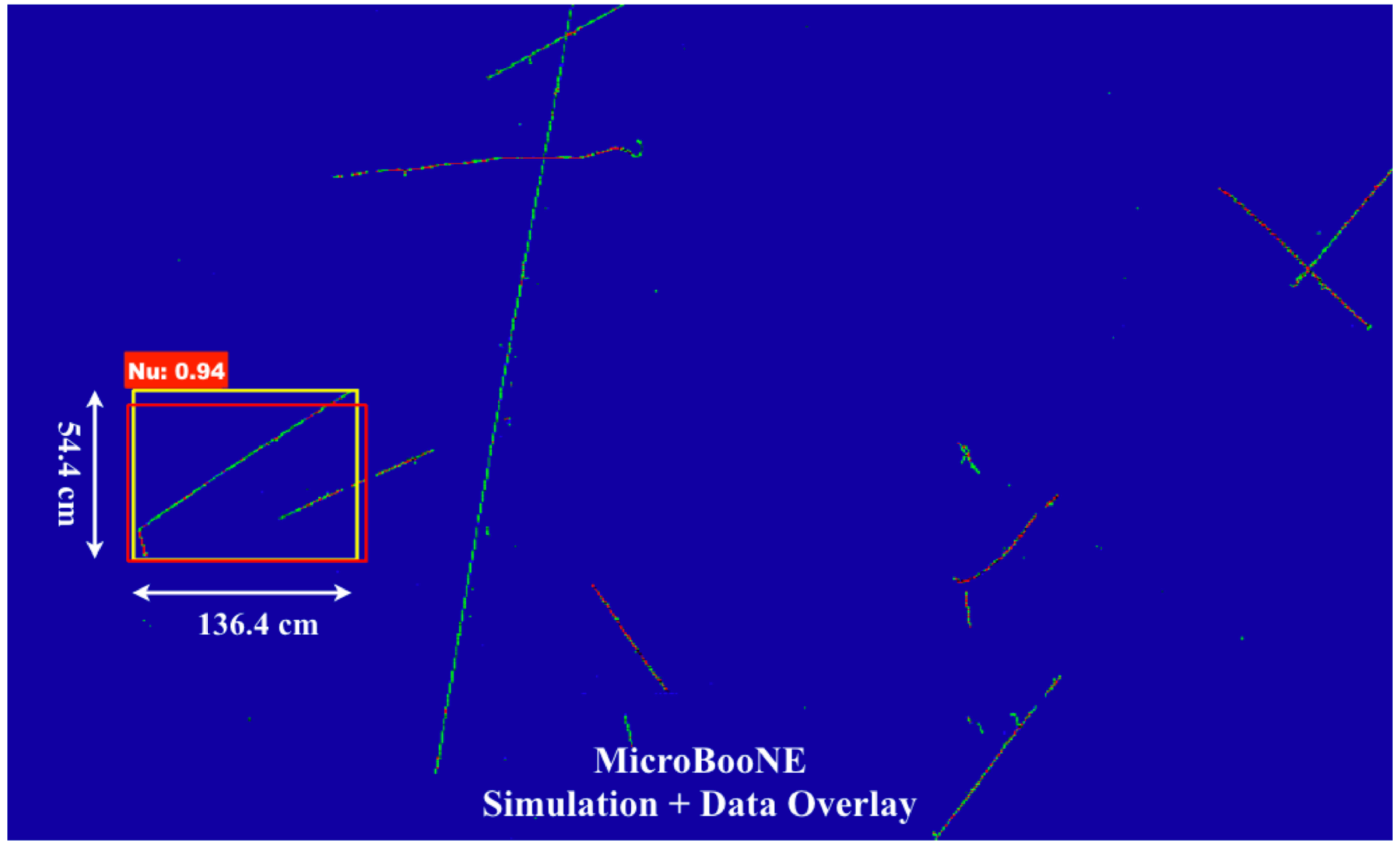


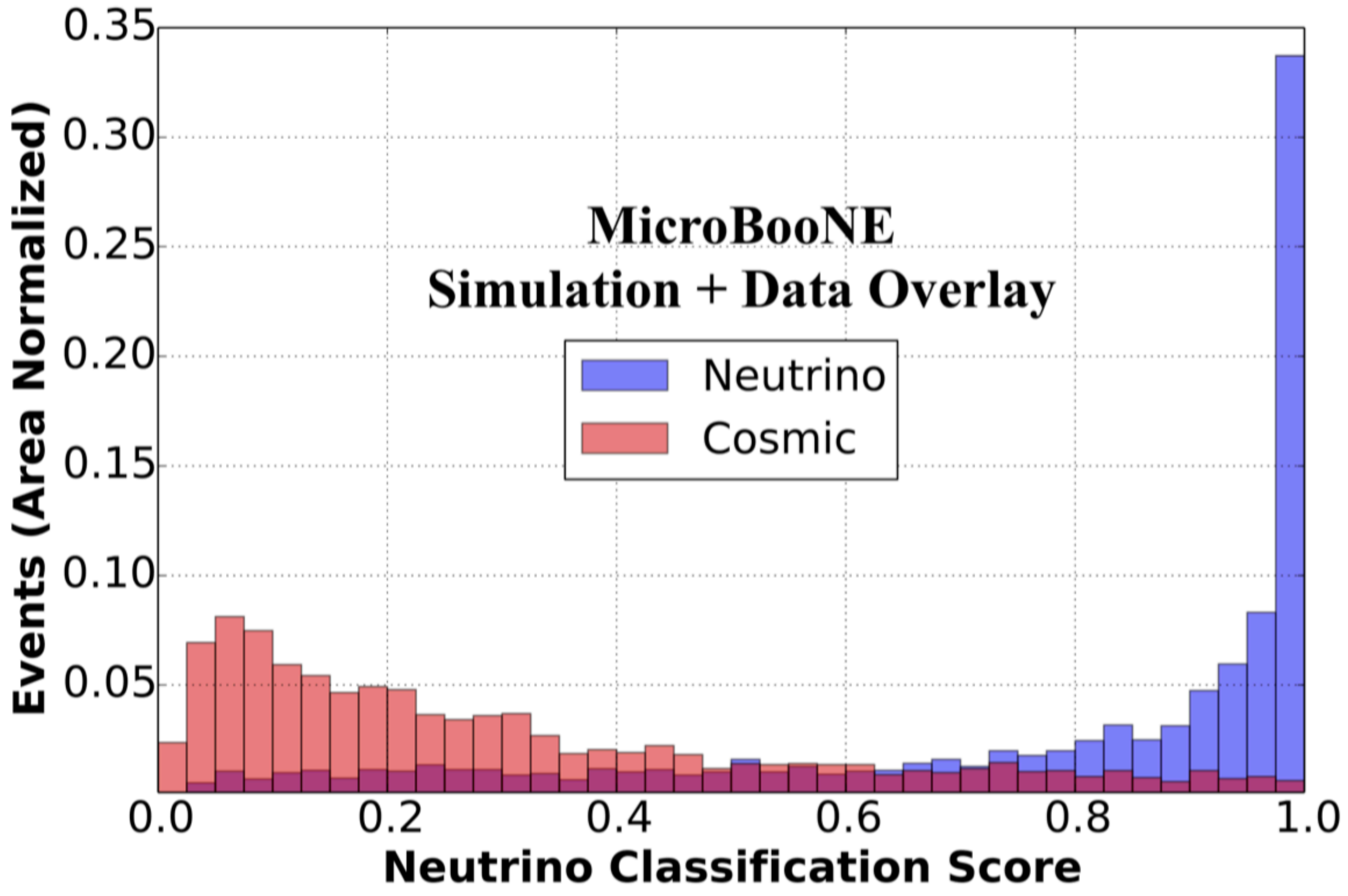


- Achieved a large performance increase (~40% in efficiency) over their traditional techniques

Selected CNN Highlights - MicroBooNE

- Use CNNs to select regions of interest (semantic segmentation) and classify the selected events
  - Selects the neutrino from within the cosmic background

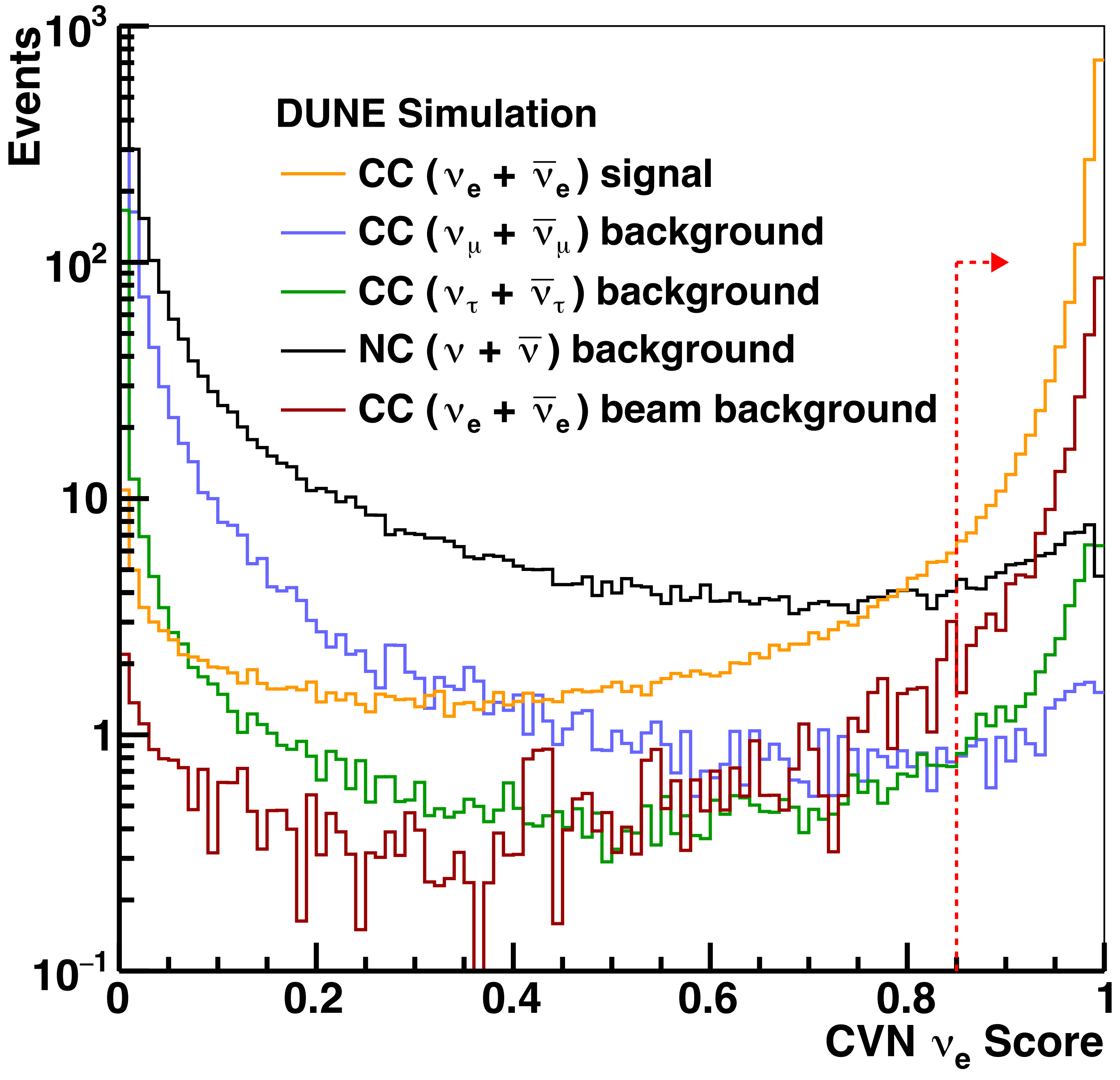


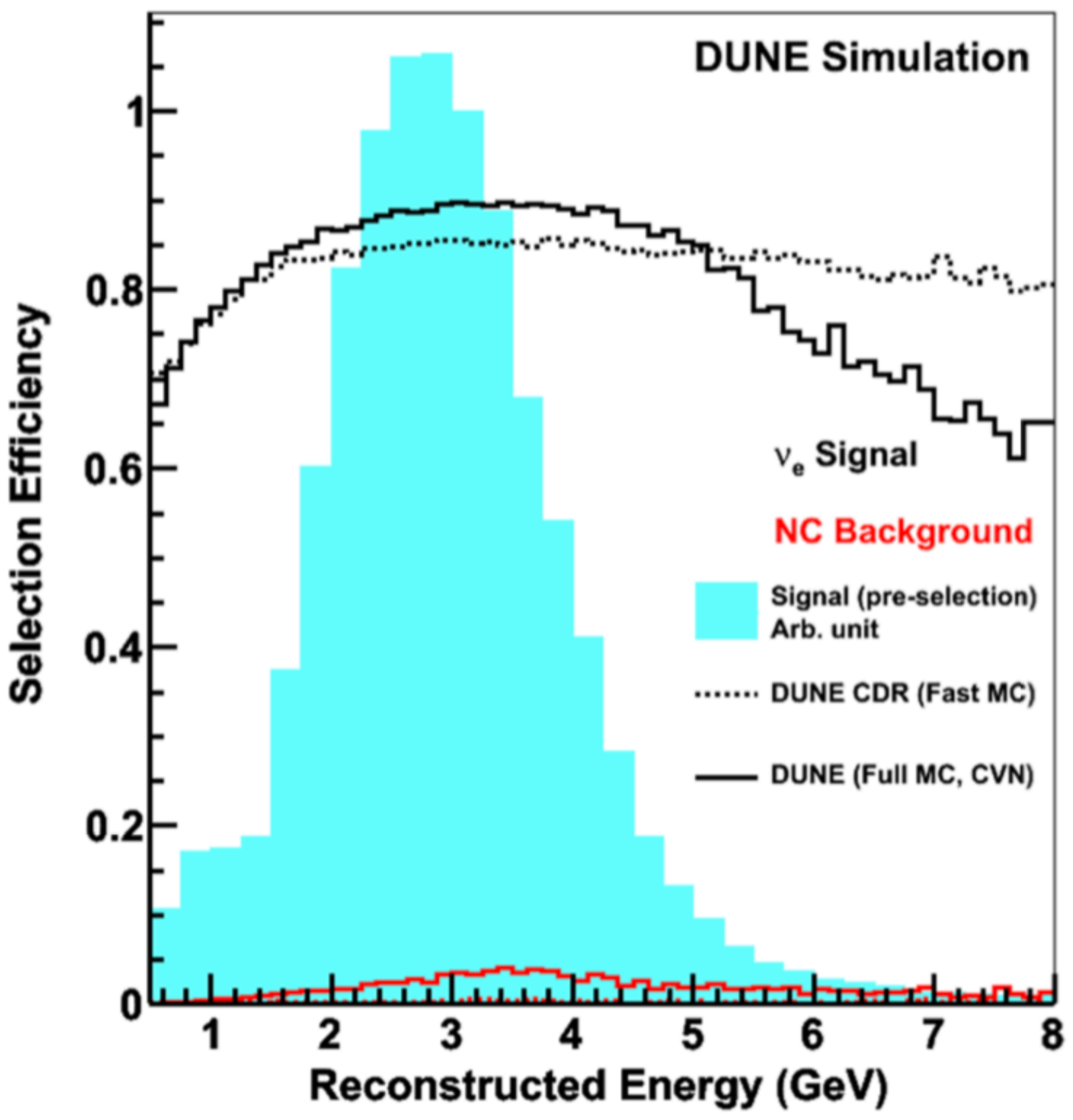


Selected CNN Highlights - DUNE

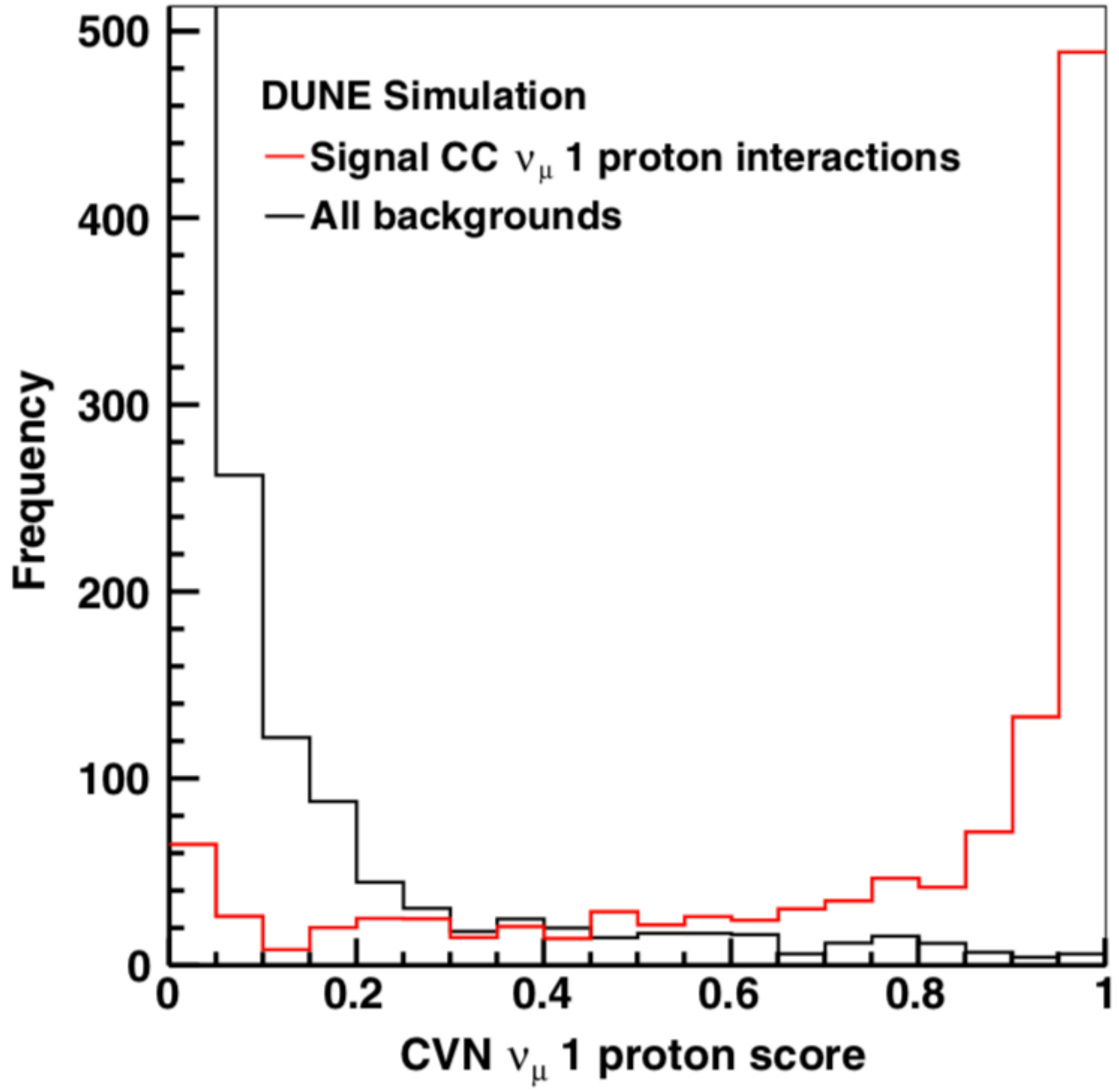
- The DUNE network has multiple outputs
  - Flavour classification and particle counting: protons, pions (charged + neutral) and neutrons

Performance of the CNN electron neutrino interaction classifier and the corresponding selection efficiency





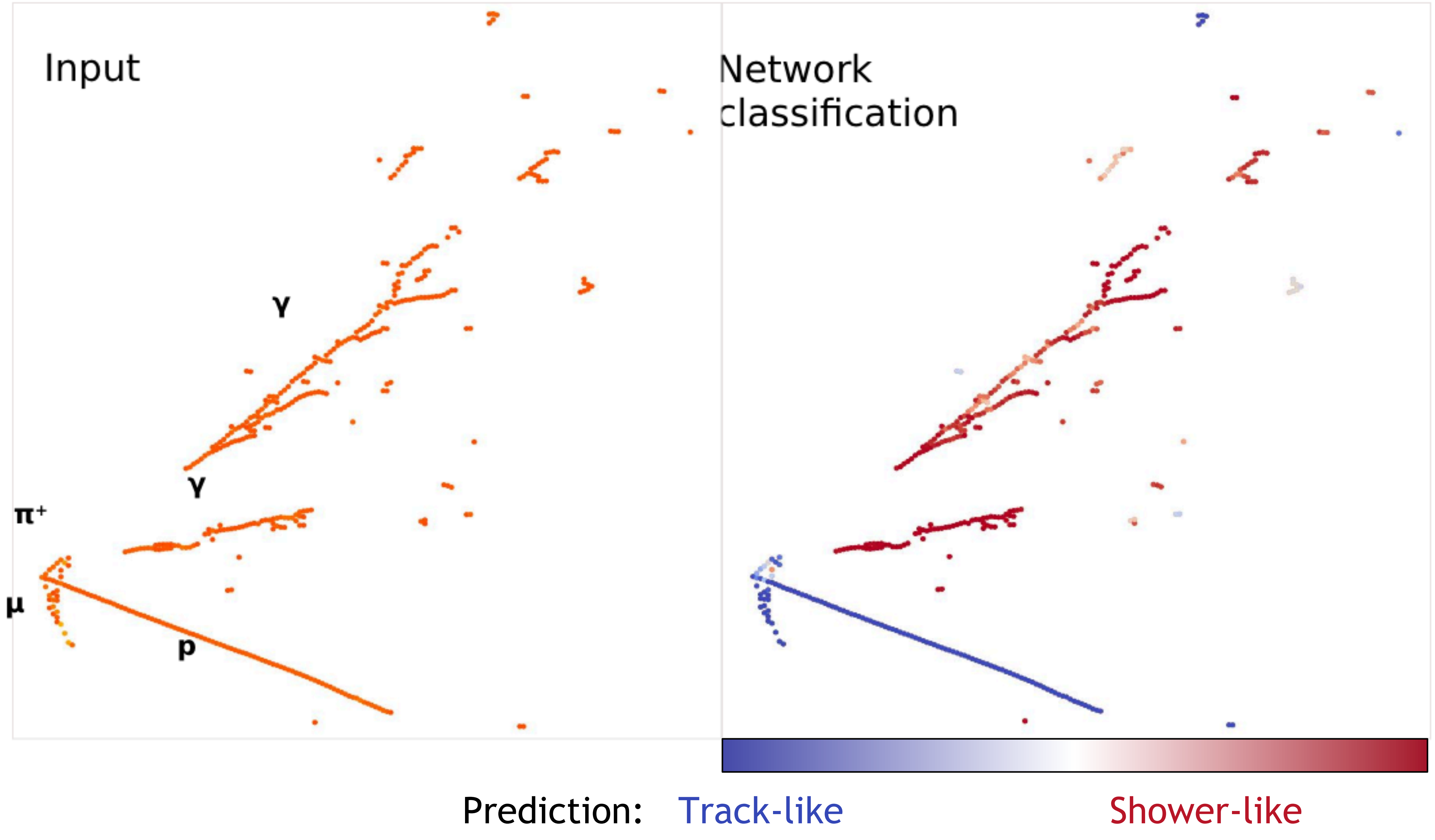
Multiply scores from different outputs for final-state selection



Selected CNN Highlights - Pandora

- Andy C. has been working on semantic segmentation to identify track- and shower-like hits in Pandora

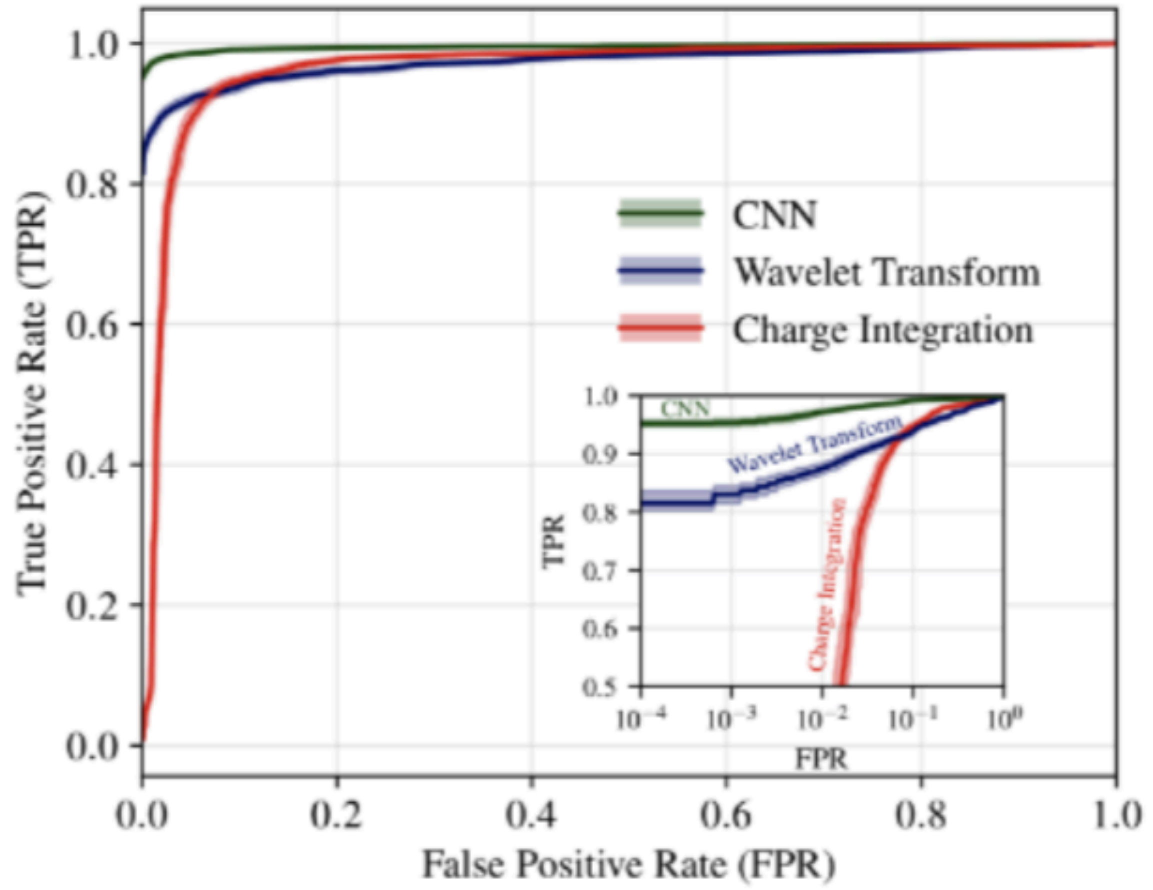
Here, the truth is:
Track-like: muon, proton, pion
Shower-like: gammas



- We can leverage deep learning in many places!

Selected CNN Highlights

- Note that CNNs don't have to be two dimensional
- I wrote a particle ID algorithm that uses 1D convolutions applied to the dE/dx profile of particles
- Other examples include signal processing and region-of-interest finding
  - Example from SoLiD: https://arxiv.org/abs/1807.06853



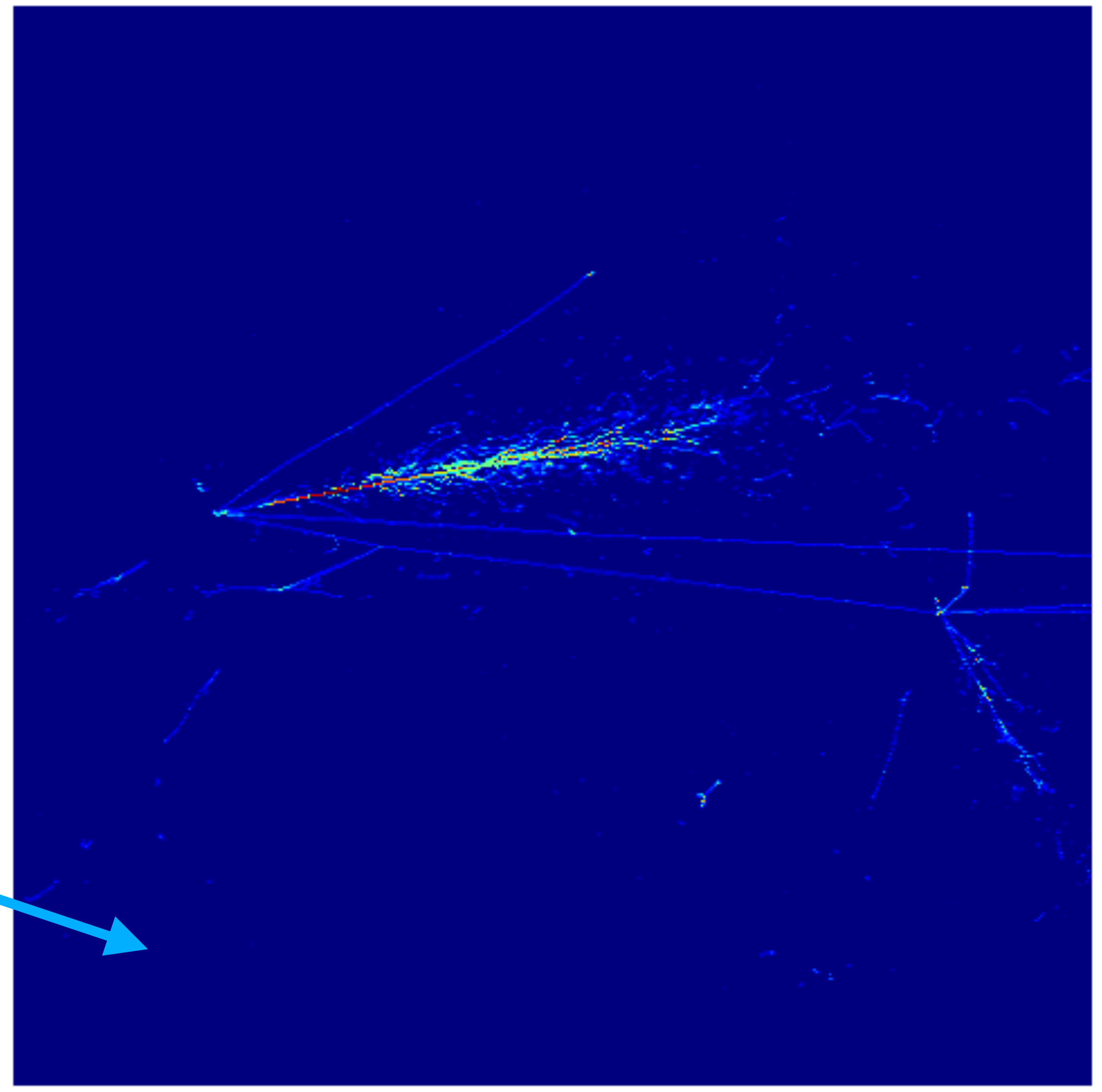
- 3D CNNs can be used for classification of video (which is just a time sequence of images)

Sparse CNNs

- The images I have shown have lots of empty pixels so computational effort is wasted.
- Sparse CNNs get around this by (cleverly) avoiding calculations on the zero value elements
  - Much more computationally efficient
  - They often work slightly better too since they avoid smearing

All dark blue pixels are empty and contain no information at all

… and there are **a lot** of them!



Graph Neural Networks

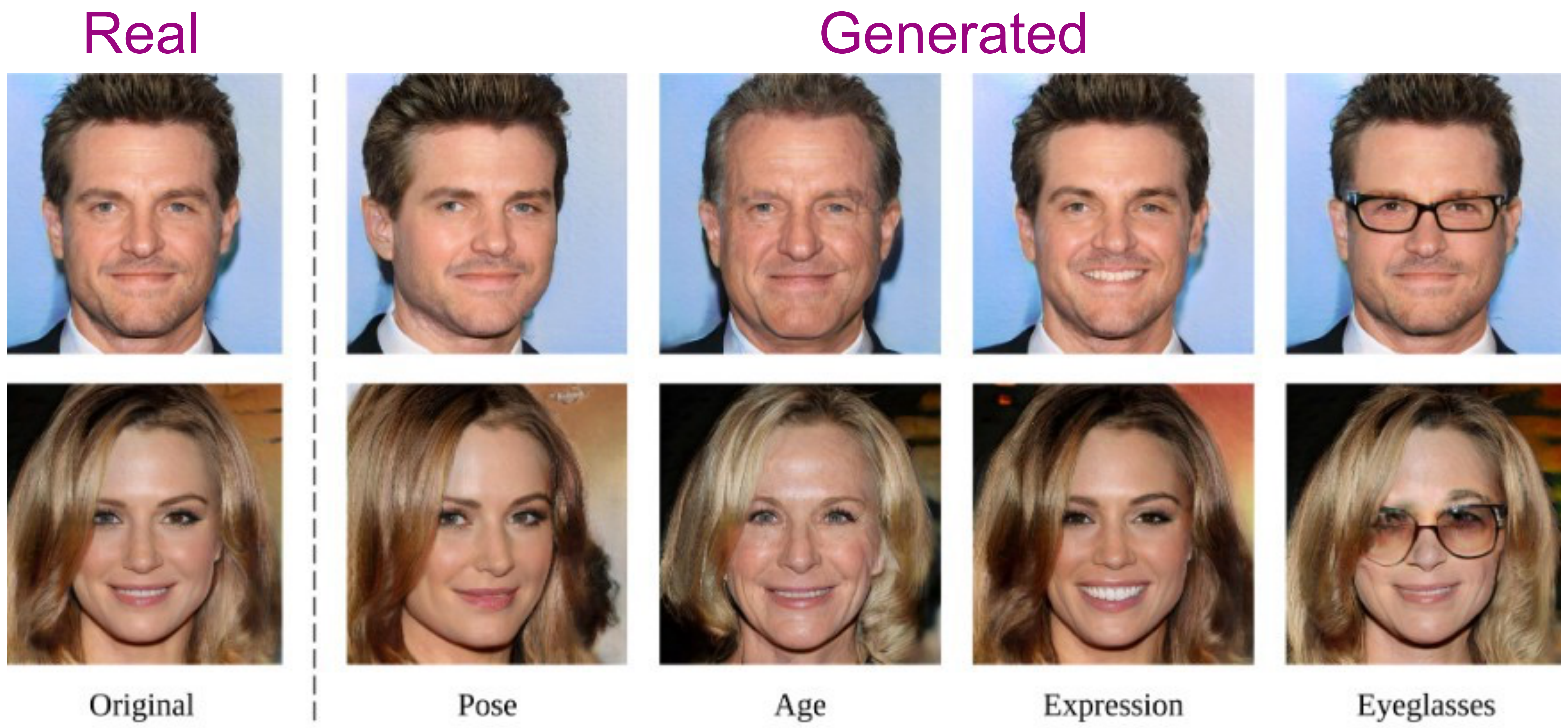
- Quite often you might find your data is difficult to format as an image
  - Likely better to use a Graph Neural Network instead of shoe-horning it into an image
- Each detector element is a node in the graph
  - Various features can be attached to nodes: charge, time, etc…
- Nodes are connected by “edges”
  - Can be defined by adjacency, or hits from the same particle etc
- IceCube used a GNN for event classification[1]
- Worked on a project to use a GNN to remove ghost hits[2]

[1] N. Choma, et al., Graph Neural Networks for IceCube Signal Classification, 2018 17th ICMLA, Orlando, FL, 2018, pp. 386-391, doi: 10.1109/ICMLA.2018.00064
[2] S. Alonso Monsalve, et al., Graph neural network for 3D classification of ambiguities and optical crosstalk in scintillator-based neutrino detectors, *Phys.Rev.D* 103 (2021) 3, 032005

Generative Adversarial Networks

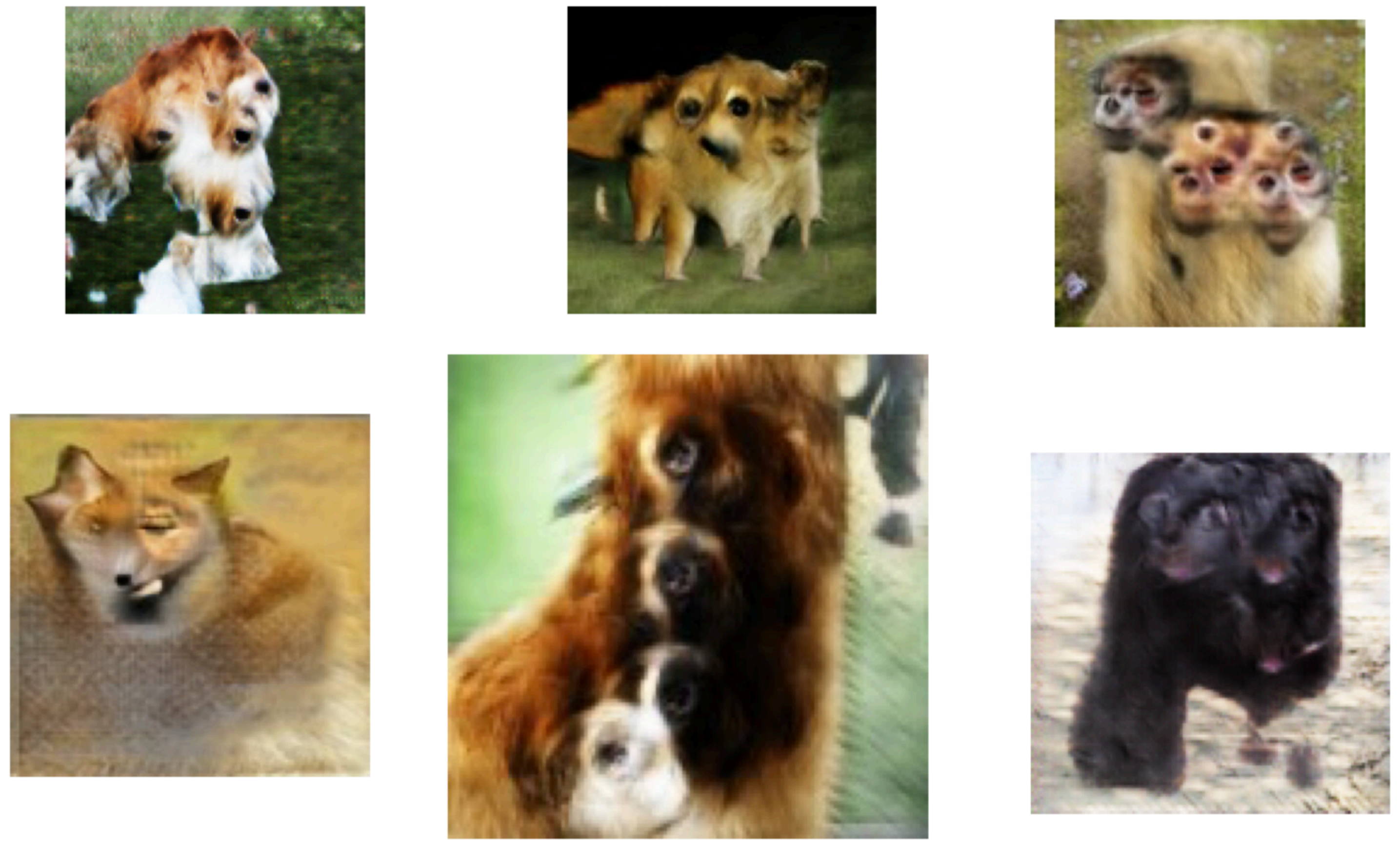
- GANs are a type of neural network composed of two different networks
  - Typically one is known as the *generator* and the other, the *discriminator*
  - Invented by Ian Goodfellow in 2014 (arXiv:1406.2661)

- They are typically used for generating images

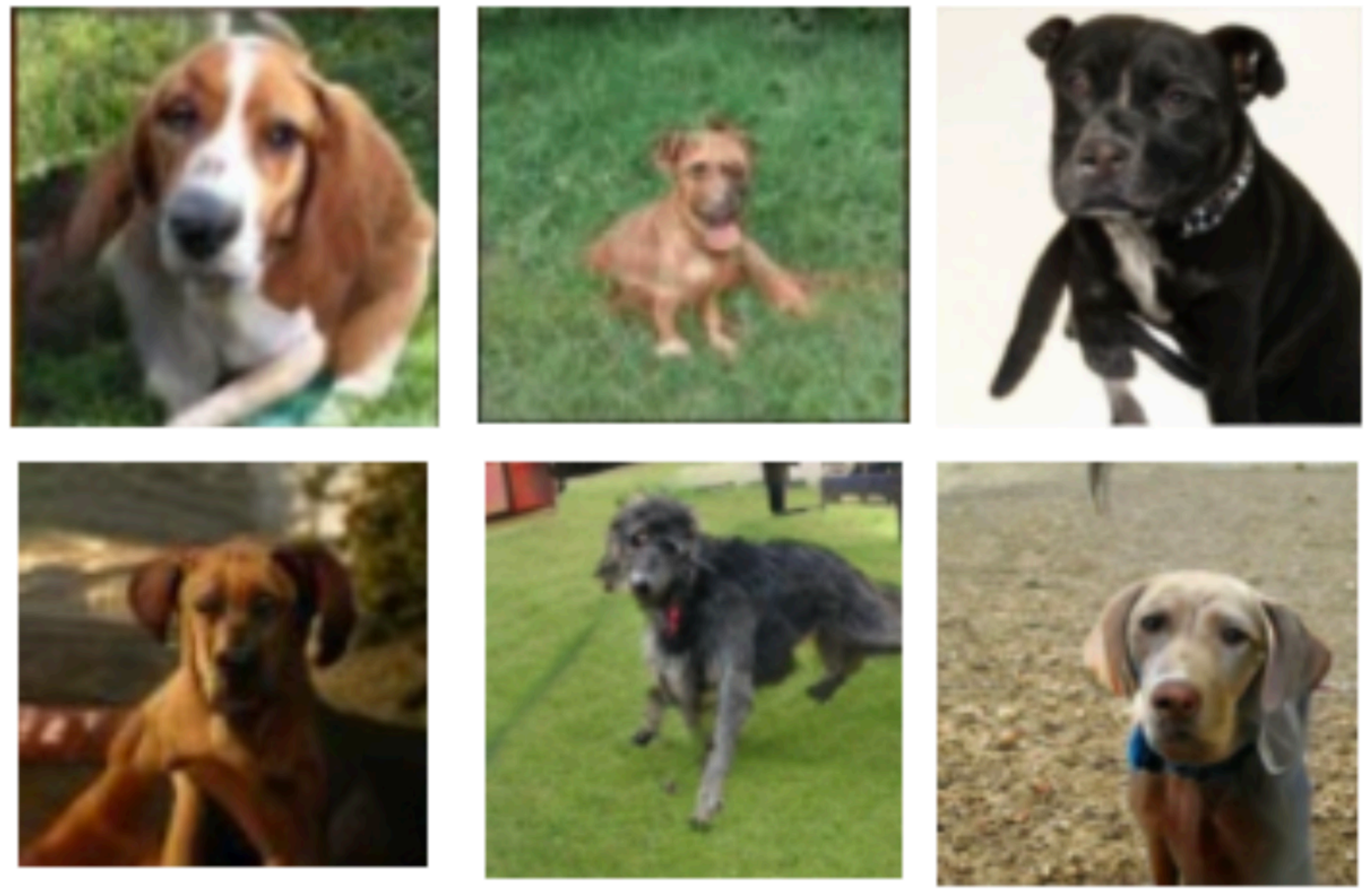


Generative Adversarial Networks

- They have come a long way in the last few years



Ian Goodfellow, NIPS 2016 Tutorial: Generative Adversarial Networks



https://www.kaggle.com/c/generative-dog-images

https://twitter.com/goodfellow_ian/status/1084973596236144640

Generative Adversarial Networks

- Simulations in HEP are generally very time consuming
  - There is a lot of appetite to make faster simulations
- Generative Adversarial Networks have two neural networks, one of which tries to trick the other. In this use case:
  - Discriminator tries to separate simulated and generated data
  - Generator tries to trick the discriminator into thinking its data are true
  - In this way, the generator learns to mimic the (complex) simulation
- Quite a few physics examples now, mostly in collider physics

Shameless plug: S. Alonso-Monsalve and L. H. Whitehead, "Image-Based Model Parameter Optimization Using Model-Assisted Generative Adversarial Networks," in *IEEE Transactions on Neural Networks and Learning Systems*, doi: 10.1109/TNNLS.2020.2969327

Transfer Learning

- Transfer learning makes use of previously trained networks
  - Allows you to fine tune a pre-trained network for your task
  - Can be useful if you don't have much data
  - The idea dates back to the early days of perceptrons[1]

- I will discuss a recent study we performed on using transfer learning in neutrino event classification

Eur. Phys. J. C (2022) 82:1099
https://doi.org/10.1140/epjc/s10052-022-11066-6

THE EUROPEAN PHYSICAL JOURNAL C

Check for updates

Regular Article - Experimental Physics

Application of transfer learning to neutrino interaction classification

Andrew Chappell[2,a], **Leigh H. Whitehead**[1,b]

[1] Department of Physics, University of Cambridge, Cambridge CB3 0HE, UK
[2] Department of Physics, University of Warwick, Coventry CV4 7AL, UK

https://link.springer.com/article/10.1140/epjc/s10052-022-11066-6

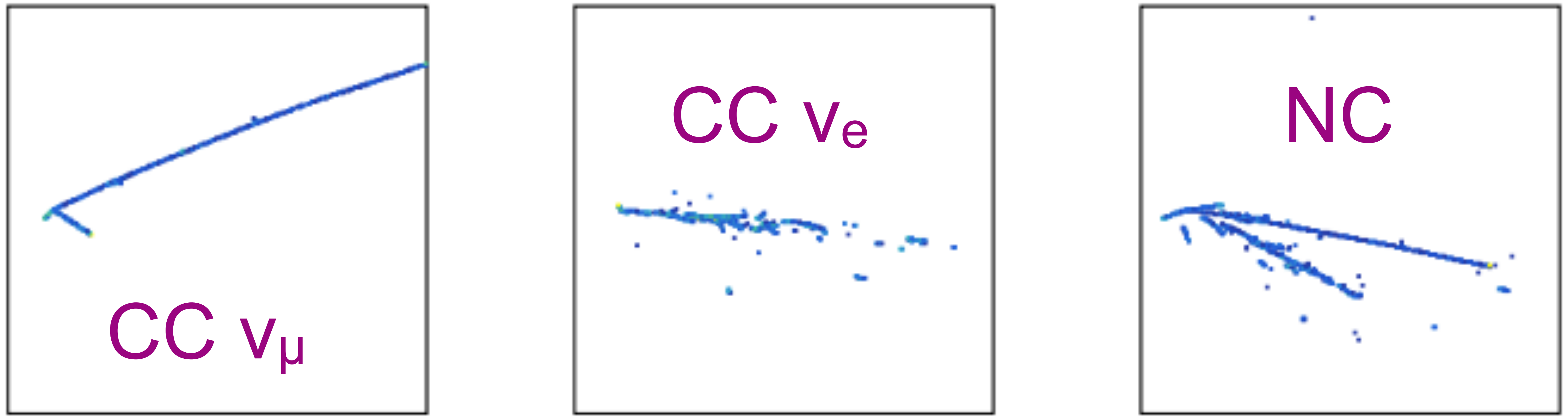
[1] S. Bozinovski, A. Fulgosi, The influence of pattern similarity and transfer learning upon the training of a base perceptron b2. In: Proceedings of Symposium Informatica, Bled, Slovenia (1976) p. 3–1215.

Transfer Learning in LArTPCs

- Can we use transfer learning to reduce the number of training examples?
  - Simulations are time consuming and GPUs need a lot of power
- Conveniently, LArTPC detectors, such as DUNE, have three readout planes
  - We get three images of a given interaction
  - Photographic images have depth three (red, green and blue channels)
- Can we use a network trained on photographs for our event classification?
  - There are plenty of networks trained on photograph-based challenges
  - Use these networks as a starting point and fine tune the weights

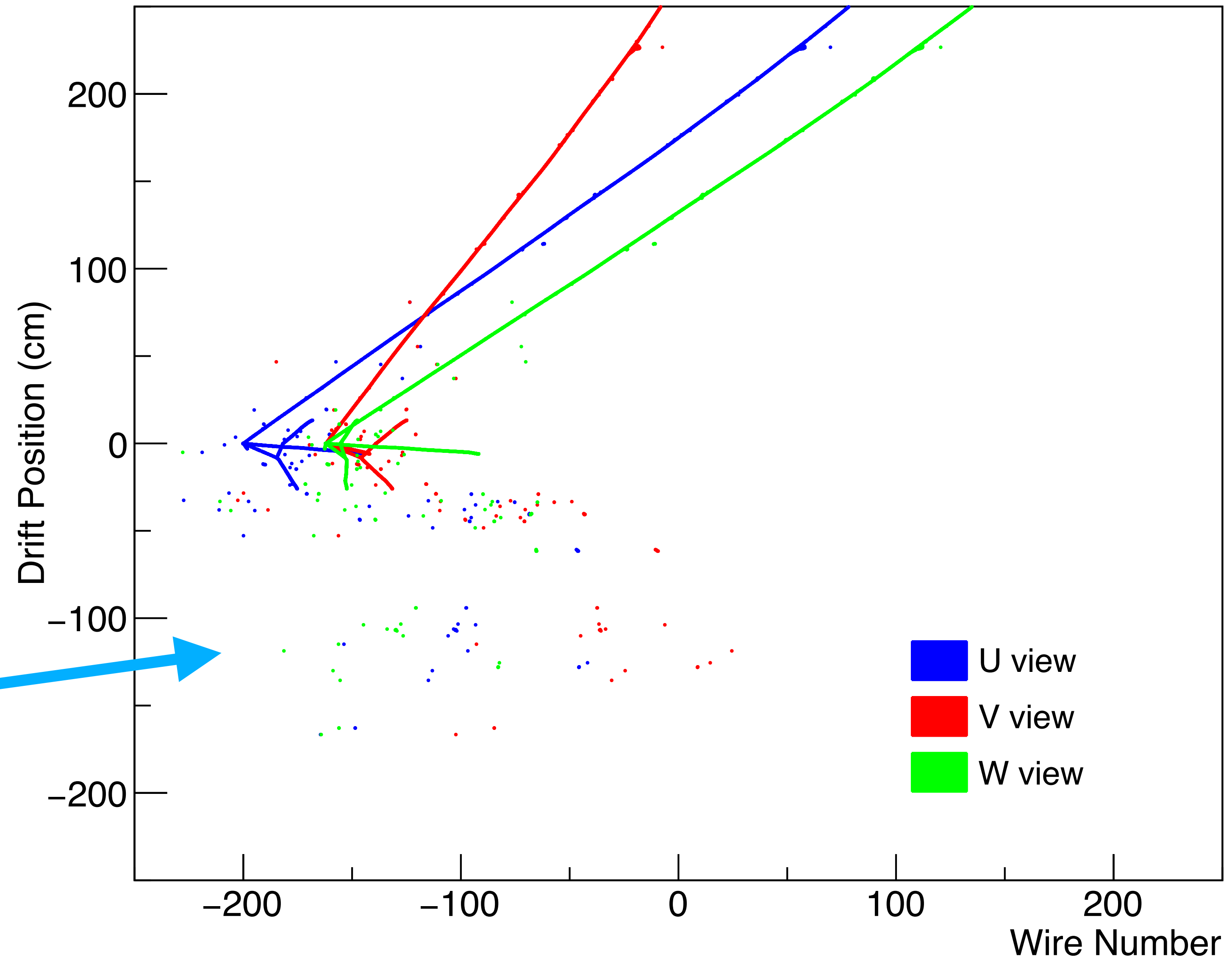
TL: Event Sample

- GENIE neutrino events:
  - CC ν_μ, CC ν_e and NC
  - 50,000 of each type



- Events passed through simple LArTPC simulation
  - Outputs three images of each event
  - Three projections of the (y,z) plane

CC ν_μ event with the three views overlaid as RGB channels



TL: Architecture and Training

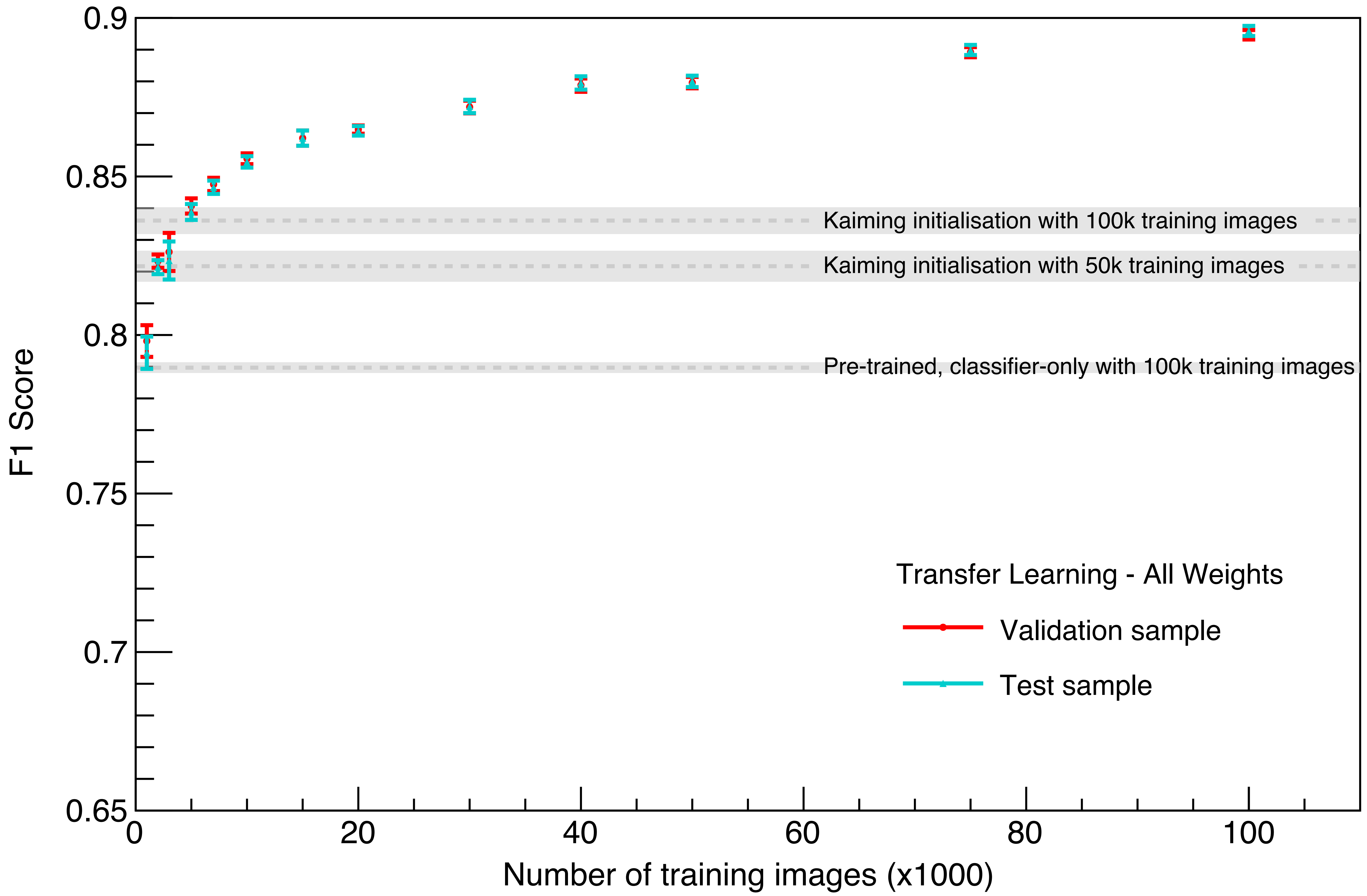
- We chose to use the Pytorch implementation of ResNet18
  - Small depth was chosen since this study involved training over 1000 networks
- The pre-trained version of ResNet18 was trained on ImageNet (224x224 pixels)
  - We had to change the final layer from 1000 to 3 classes: CC ν_μ, CC ν_e and NC
- Trained a series of networks with:
  - Kaiming (He) randomly initialised weights
  - Weights from the pre-trained ImageNet network
  - Various numbers of training events from 1,000 to 100,000
  - Trained each network 25 times to give an estimate of the uncertainty

Results: TF vs random initialisation

- Compared the F1 score from the transfer-learned networks fine-tuned with 1k to 100k images against the Kaiming-initialised network with 50k and 100k events

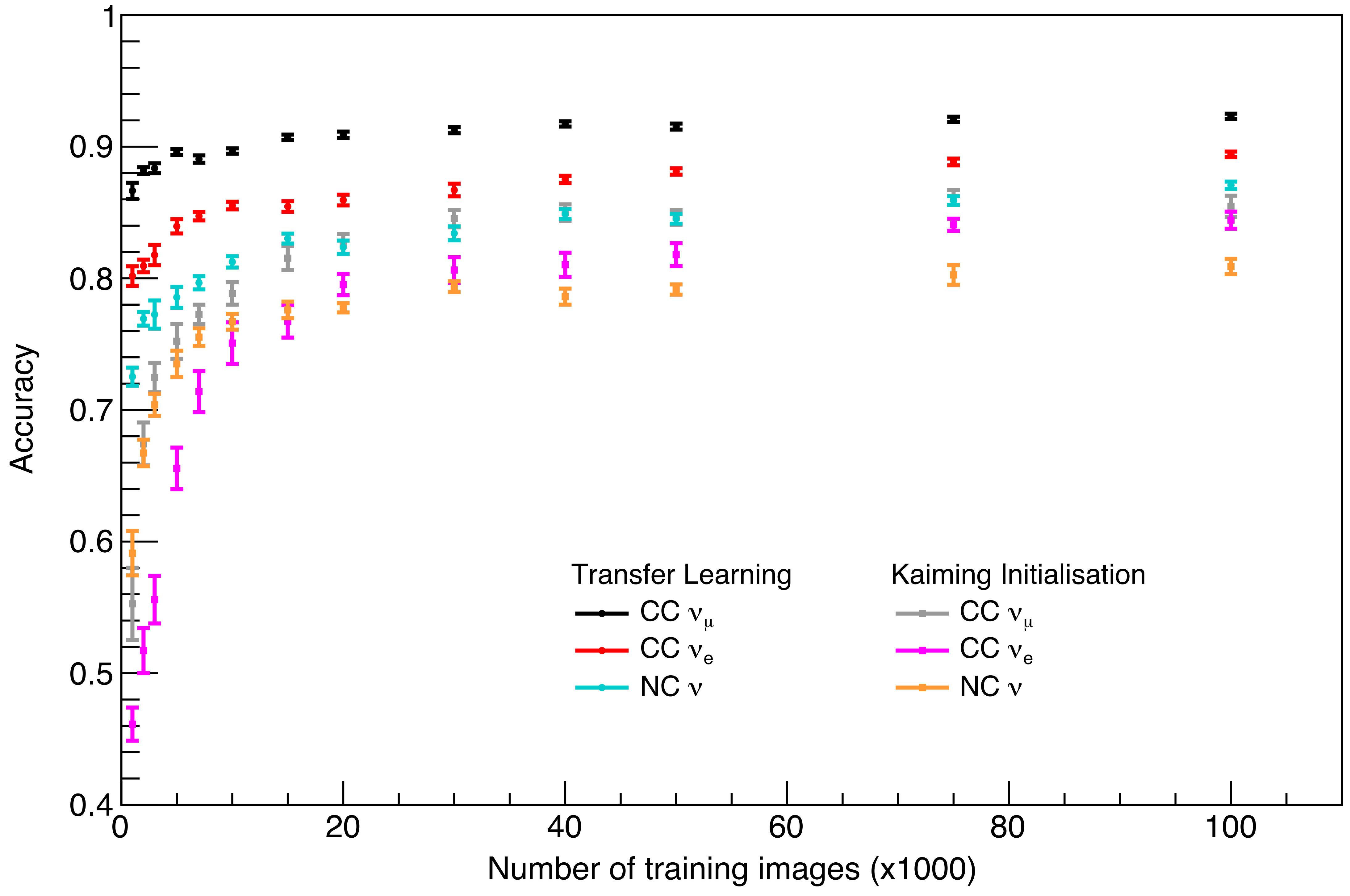
- Transfer-learned network out-performs the Kaiming-initialised network with 100k training images
  - For 7k training images and above

- Event fine-tuning just the final layer works surprising well
  - F1 score = 0.79



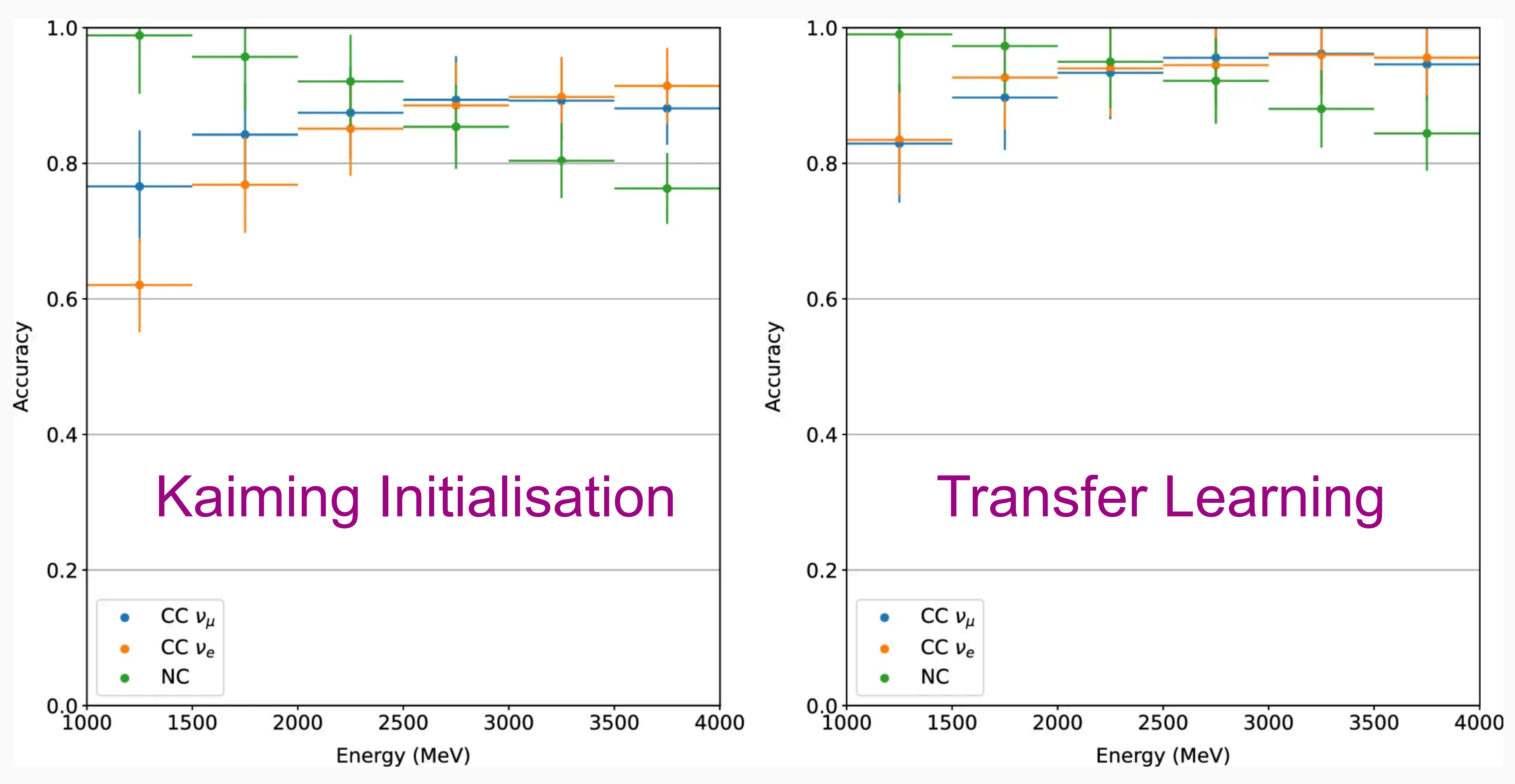
Transfer Learning in LArTPCs

- Better performance is seen in all classes
  - It wasn't just helping in specific types of events
- Performance increase seems to be across the whole sample



Transfer Learning in LArTPCs

- We also looked for potential biases between classes and a function of energy
  - See reduced bias in both cases using transfer learning
  - Plots show examples from training with 100k events



Transfer Learning in LArTPCs

- Also looked at the effect of freezing different layer weights
  - Layers 1 to 4 here correspond to the ResNet blocks
  - As a minimum we have to train the classifier (dense layer)
  - The difference between Layer 1 and All Weights is the first convolutional layer
    - No difference in performance is seen when the first layer weights can be fine-tuned
    - The initial layer feature extraction from photographic images does extract what we need for neutrino interactions!