

HEP Software R&D  
at the Frontier of  
*Data Science and Deep Learning*

Amir Farbin  
University of Texas Arlington

# Overview

- HEP Software/Computing Landscape
- LArTPC Reconstruction
- Brief Deep Learning Introduction
- Deep Learning in LArTPC Feasibility study with LArIAT
- Future frameworks
- DAG/Tensors Based frameworks.
  
- Thank you to all the people who's slides I stole...
- Lots of new ideas at Data Science @ LHC workshop.

# Processor Landscape

- Parallelism taking off:
  - Multi-core CPUs, with built in GPUs.
  - Many-core GPU/MiCs
  - OpenCL Programmable FPGAs (and ASIC?)
    - Potential to easily move algorithms from software to hardware.
- On the horizon:
  - CPU/GPU/RAM Stacking
  - CPU+FPGA on same Dye
  - Neuromorphic Chips

# HEP Software Landscape

- Shift from Fortran to C++ in late 1990s.
- *The* biggest datasets until mid 2000s... industry now leads.
- Problems:
  - Extremely complicated C++ frameworks, data structures, ...
  - Difficulty utilizing Multi-core CPUs and new co-processors (e.g. GPU/MiC)
  - Expensive: ATLAS software cost  $\sim O(250 \text{ Million})$  CHF to build over 15 years...
    - starting from scratch for Run 3 wasn't an option for ATLAS or CMS.
  - We cannot find developers to fill mission critical posts. Critical people get stuck in jobs...
  - We do not educate HEP PhDs in software... rely on talented people training themselves.
  - There is a culture that software isn't physics... but electronics and hardware are!
    - e.g. we do not support software R&D.

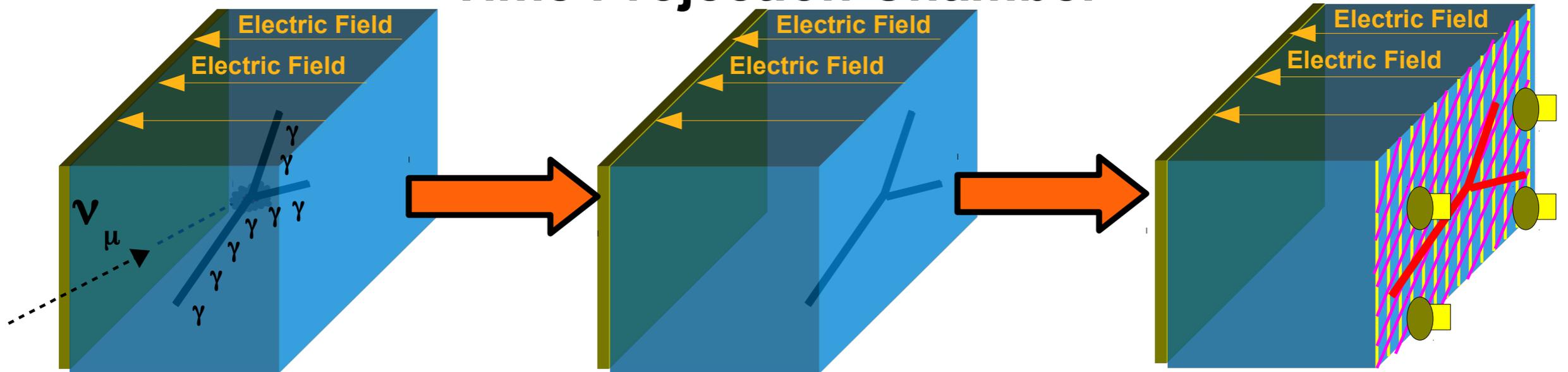
# Parallelization Problem

- *Multi-core CPUs*: we are quickly approaching 100's of cores/CPU.
  - Currently relying on Embarrassingly Parallel nature of HEP data.
    - Filling CPU cores with independent instances of software.
    - Not practical to have 4 GB/core for 100's of cores...
    - Not enough bandwidth to memory if every core needs to access different 4 GB.
  - C++ data structures make it difficult to take advantage of vectorization.
- *Many-core Co-processors (possibly within CPU dye)*: GPUs/MiCs, FPGA, (ASICs?)
  - Requires *Data Parallelization* where (for example) many events are simultaneously processed in each algorithm. HEP frameworks designed to see 1 event at a time.
  - Difficult to code. Highly sensitive to optimization and hardware. Difficult to efficiently integrate with current software. Rapidly evolving ecosystem.
  - Mostly used in specialized systems like DAQ and Trigger. No good solution for offline.

# LArTPC Reconstruction

# LArTPC

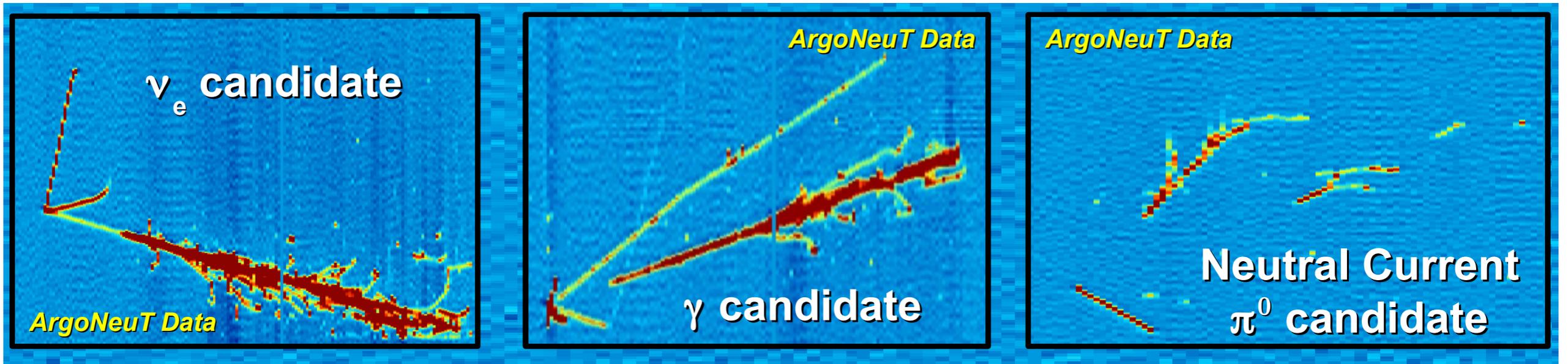
## Time Projection Chamber



Neutrino interaction in LAr produces ionization and scintillation light

Drift the ionization charge in a uniform electric field

Read out charge and light produced using precision wires and PMT's



Jonathon Asaadi

Tracking, Calorimetry, and Particle ID in same detector.

Goal  $\sim 80\%$  Neutrino Efficiency.

That's all we need

In neutrino experiments, try to determine flavor and estimate energy of incoming neutrino by looking at outgoing products of the interaction.

*Typical neutrino event*

Incoming neutrino:  
Flavor unknown  
Energy unknown

Outgoing lepton:

Flavor: CC vs. NC,  $\mu^+$  vs.  $\mu^-$ , e vs.  $\gamma$   
Energy: measure

Mesons:

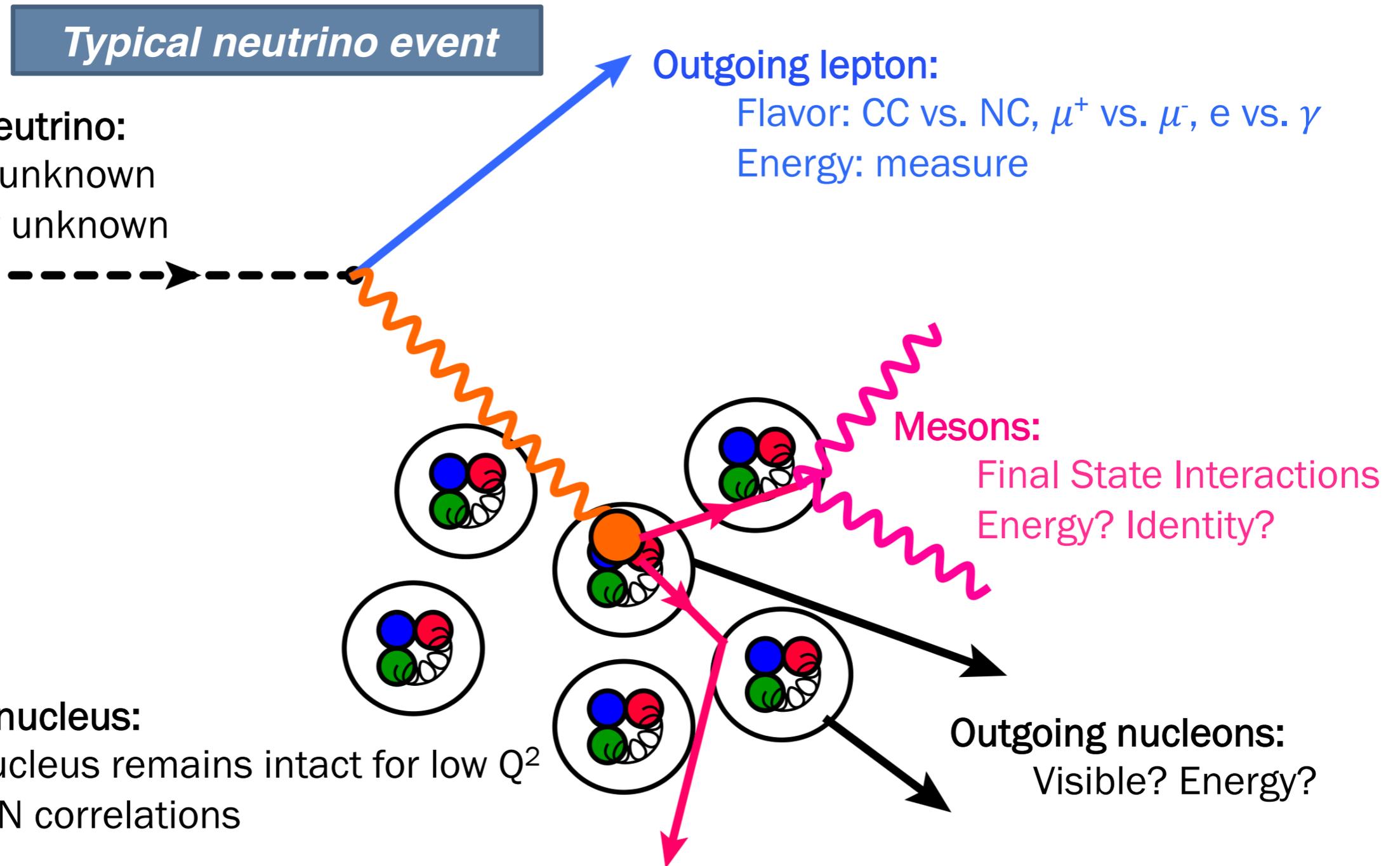
Final State Interactions  
Energy? Identity?

Target nucleus:

Nucleus remains intact for low  $Q^2$   
N-N correlations

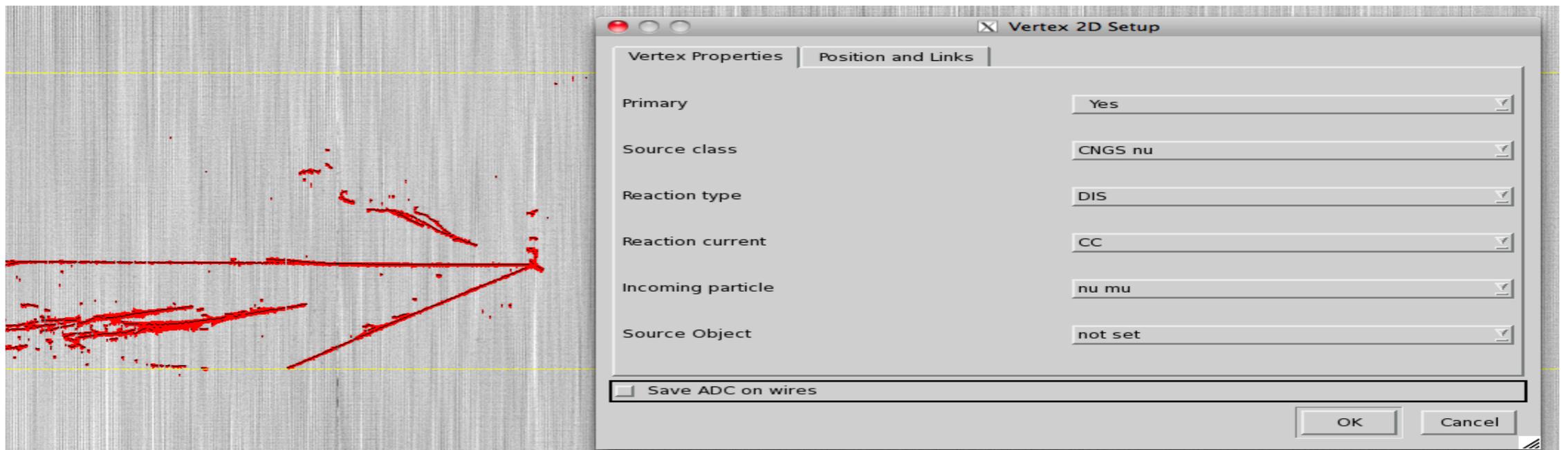
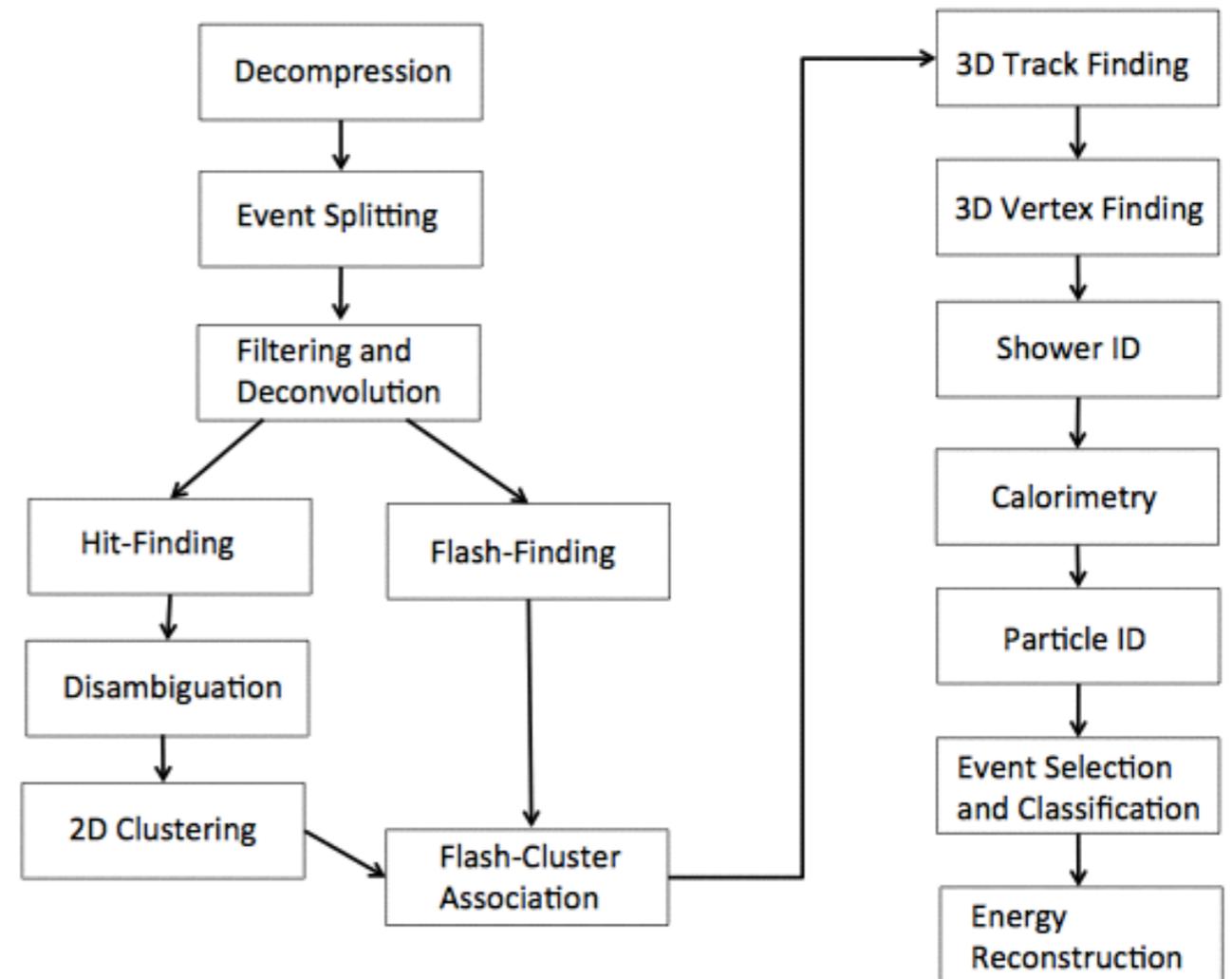
Outgoing nucleons:

Visible? Energy?

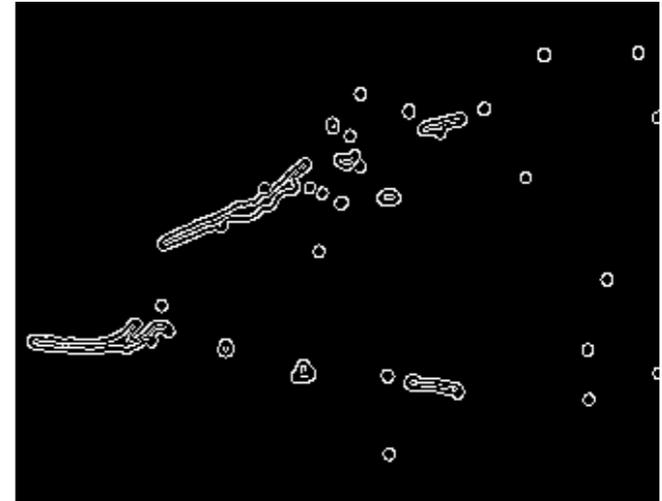
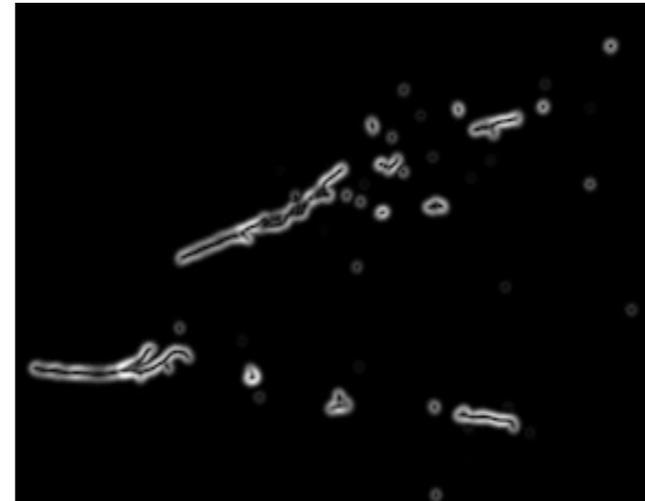
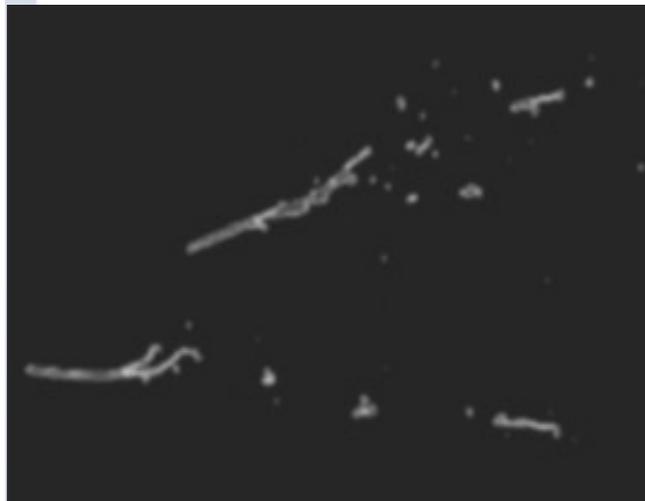


# LArTPC Reconstruction

- *Hand scans*
  - QScan: ICARUS user assisted reconstruction.
- Full automatic reconstruction is rapidly coming together
  - LArSoft
    - *pandora*
    - WireCell- tomography
- New ideas: Image processing, Deep Learning



# Image Processing



Jessica Esquivel

# Deep Learning

# Deep Learning

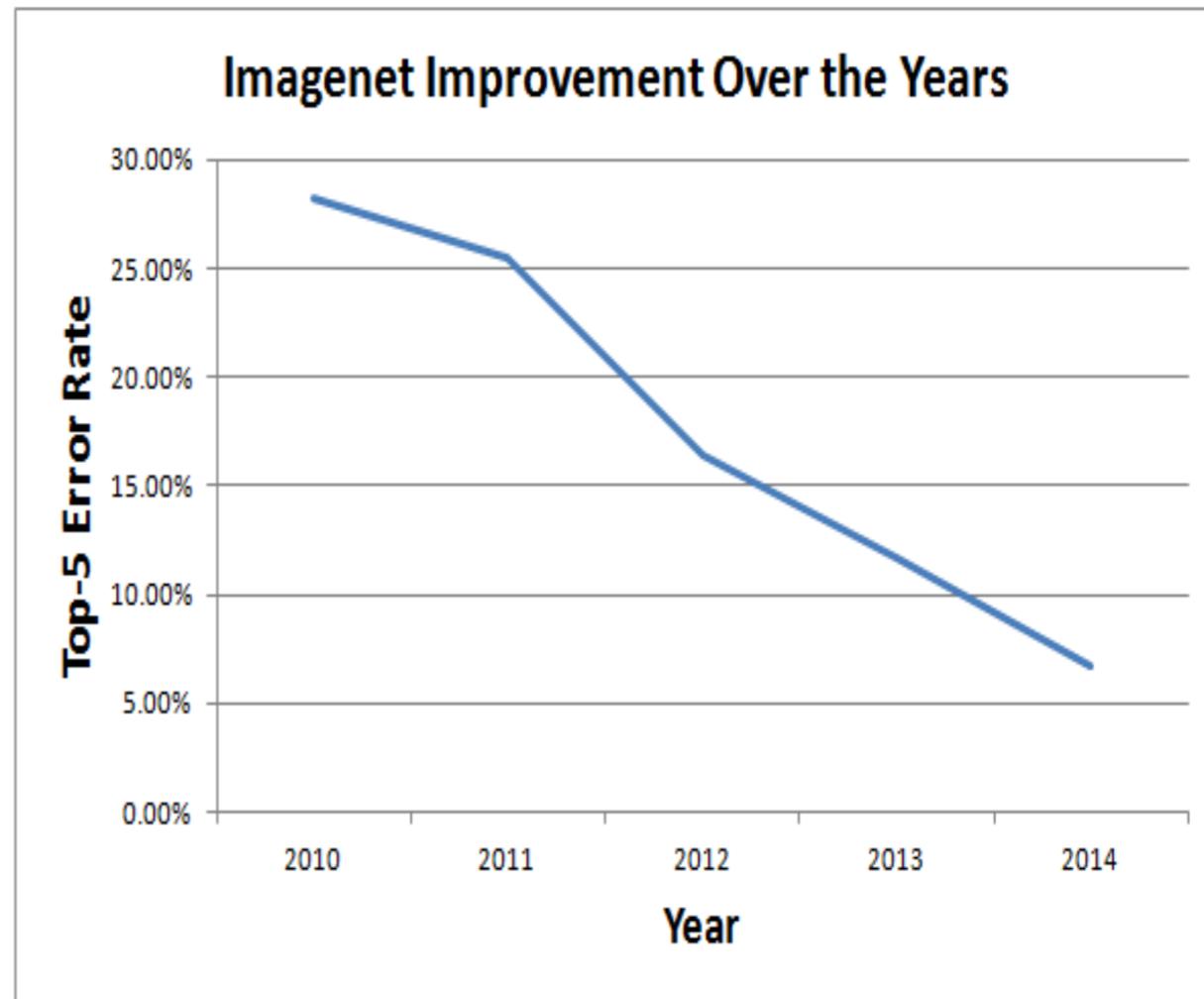
- What is it?
  - Many layer Neural Networks with large number of parameters.
- Why now? Difficulty training such big networks in the past... now:
  - Solutions to difficulties in training (vanishing gradient problem)
    - Better activation. Longer training with bigger Data sets. Unsupervised Learning.
  - Big Data provides the necessary large datasets for training
  - GPUs
- Why is deep better than shallow?
  - Eliminate *Feature Engineering*
    - Shallow networks, most of *your* time spent on developing algorithms that process raw data into the inputs to the NN.
    - Deep NNs can learn features from raw data.
  - *Unsupervised learning*: DNNs classify events without being told what are the classes.
    - The hope is that they could make sense of complicated data that we don't understand.

# Recent History

- Deep Learning feats that sparked broad interest:
  - 2012, Google 1B DNN learns to identify cats (and 20000 other types of objects) (Wired Article, paper)
    - *No feature engineering*: trained with 200x200 pixel images from YouTube
    - *Unsupervised*: the pictures were unlabeled.
    - Google cluster 16000 cores ~ \$1M. Redone with \$20k system with GPUs.
  - 2013: Deep Mind builds AI that plays ATARI (Blogpost, Nature, YouTube)

# Computer Vision - Image Classification

- **Imagenet**
- Over 1 million images, 1000 classes, different sizes, avg 482x415, color
- 16.42% Deep CNN dropout in 2012
- 6.66% 22 layer CNN (GoogLeNet) in 2014
- 4.9% (Google, Microsoft) super-human performance in 2015



Sources: Krizhevsky et al ImageNet Classification with Deep Convolutional Neural Networks, Lee et al Deeply supervised nets 2014, Szegedy et al, Going Deeper with convolutions, ILSVRC2014, Sanchez & Perronnin CVPR 2011, <http://www.clarifai.com/> Benenson, [http://rodrigob.github.io/are\\_we\\_there\\_yet/build/classification\\_datasets\\_results.html](http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html)

# Examples

stone wall [ 0.95, [web](#) ]



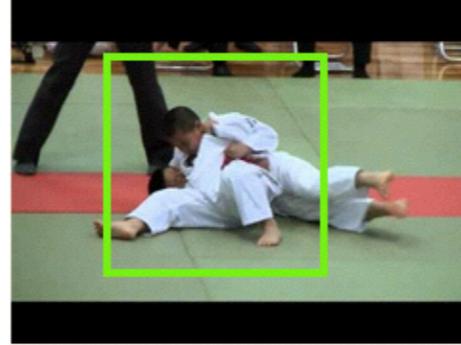
dishwasher [ 0.91, [web](#) ]



car show [ 0.99, [web](#) ]



judo [ 0.96, [web](#) ]



judo [ 0.92, [web](#) ]



judo [ 0.91, [web](#) ]



tractor [ 0.91, [web](#) ]



tractor [ 0.91, [web](#) ]

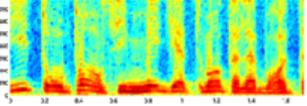


tractor [ 0.94, [web](#) ]



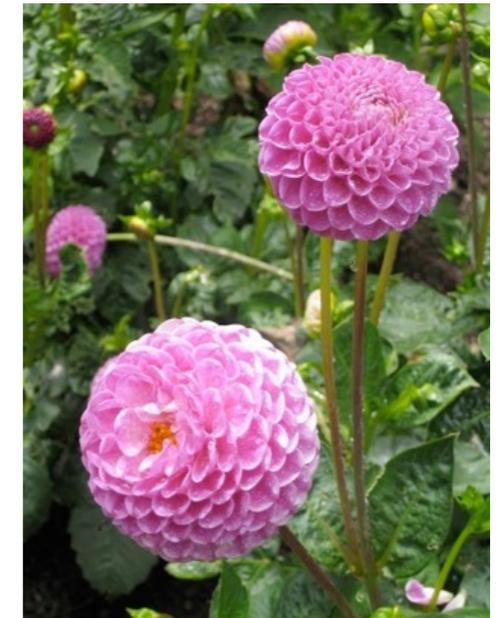


## Good Fine-grained Classification

Input	Output
Pixels: 	“ear”
Audio: 	“sh ang hai res taur aun ts”
<query, doc1, doc2>	P(doc1 preferred over doc2)
“Hello, how are you?”	“Bonjour, comment allez-vous?”



“hibiscus”



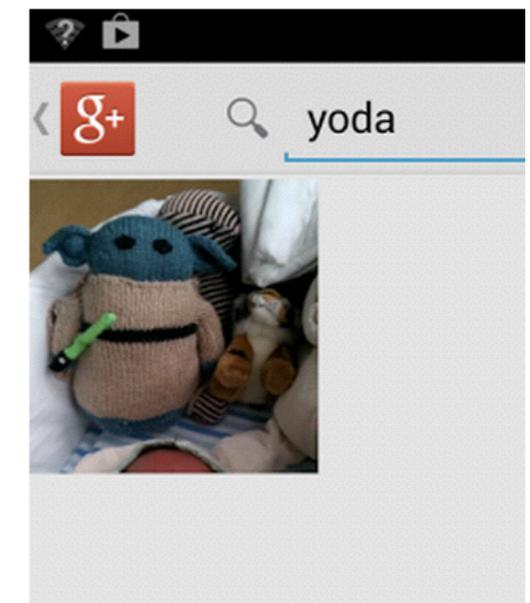
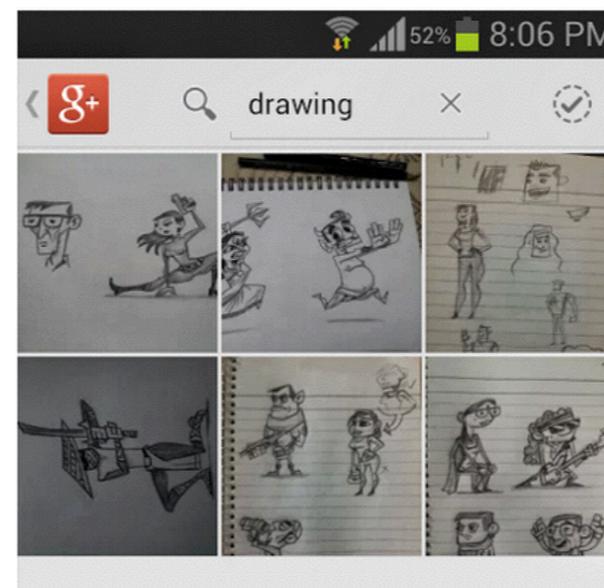
“dahlia”

## Good Generalization

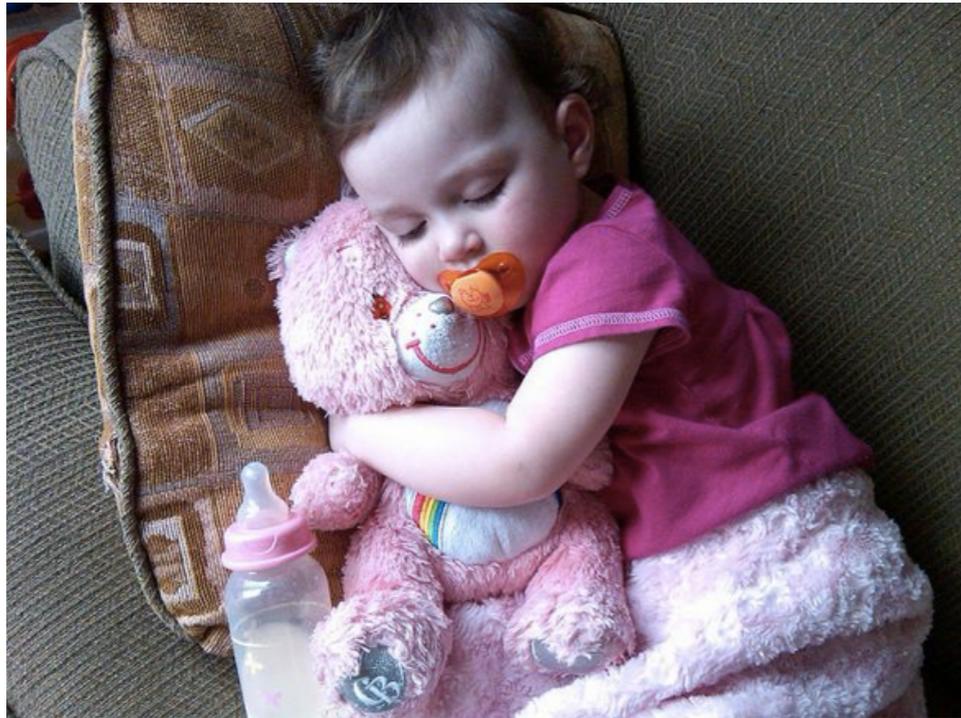
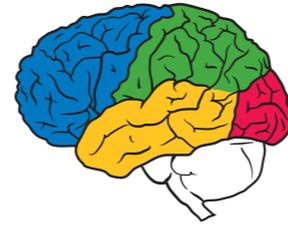


Both recognized as a  
“meal”

Google Plus photo search is awesome. Searched with keyword 'Drawing' to find all my scribbles at once :D



# Generating Image Captions from Pixels



*Human:* A young girl asleep on the sofa cuddling a stuffed bear.

*Model sample 1:* A close up of a child holding a stuffed animal.

*Model sample 2:* A baby is asleep next to a teddy bear.

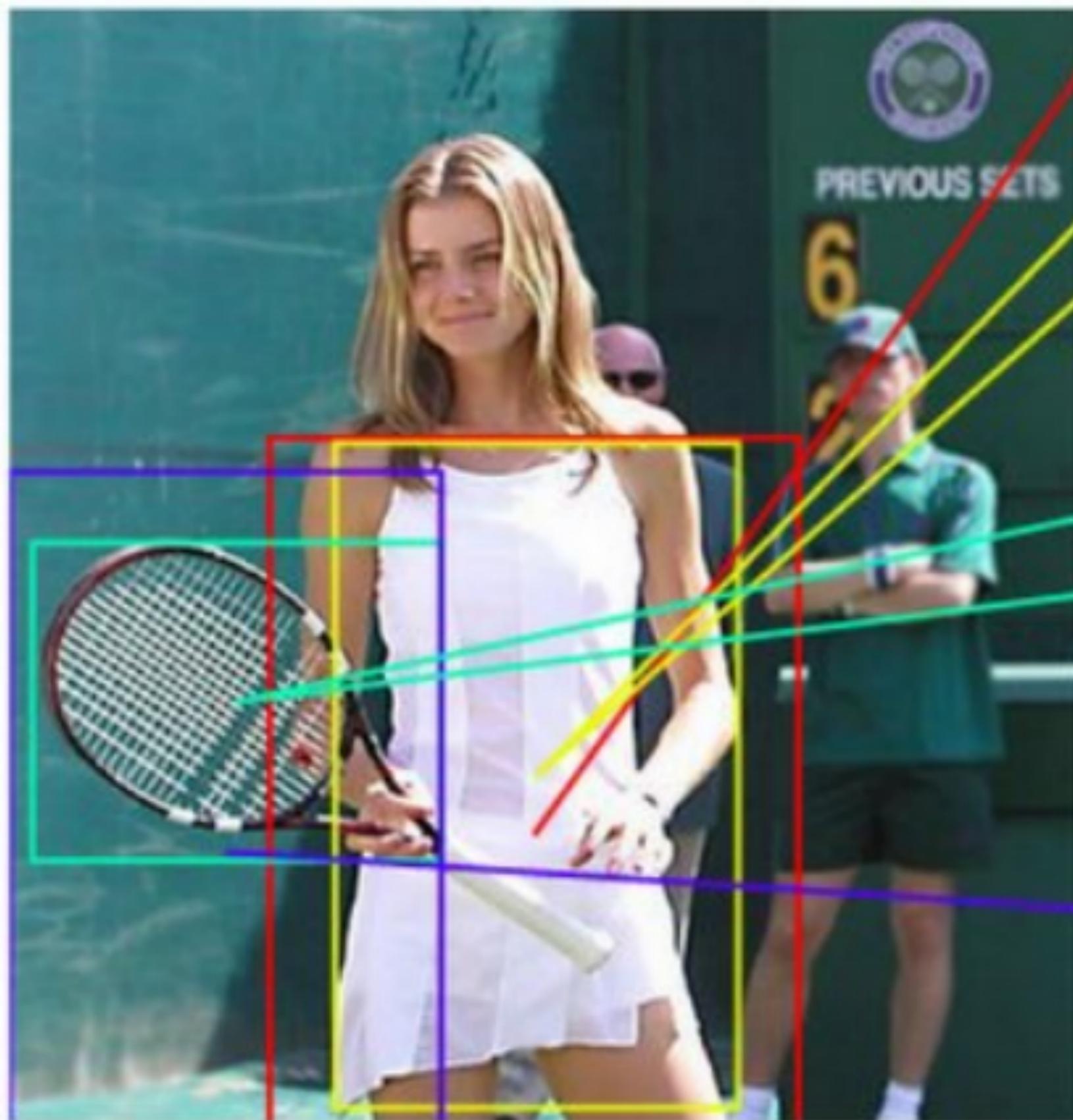
# Generating Image Captions from Pixels



*Human:* Three different types of pizza on top of a stove.

*Model sample 1:* Two pizzas sitting on top of a stove top oven.

*Model sample 2:* A pizza sitting on top of a pan on top of a stove.



1.12 woman

-0.28 in

1.23 white

1.45 dress

0.06 standing

-0.13 with

3.58 tennis

1.81 racket

0.06 two

0.05 people

-0.14 in

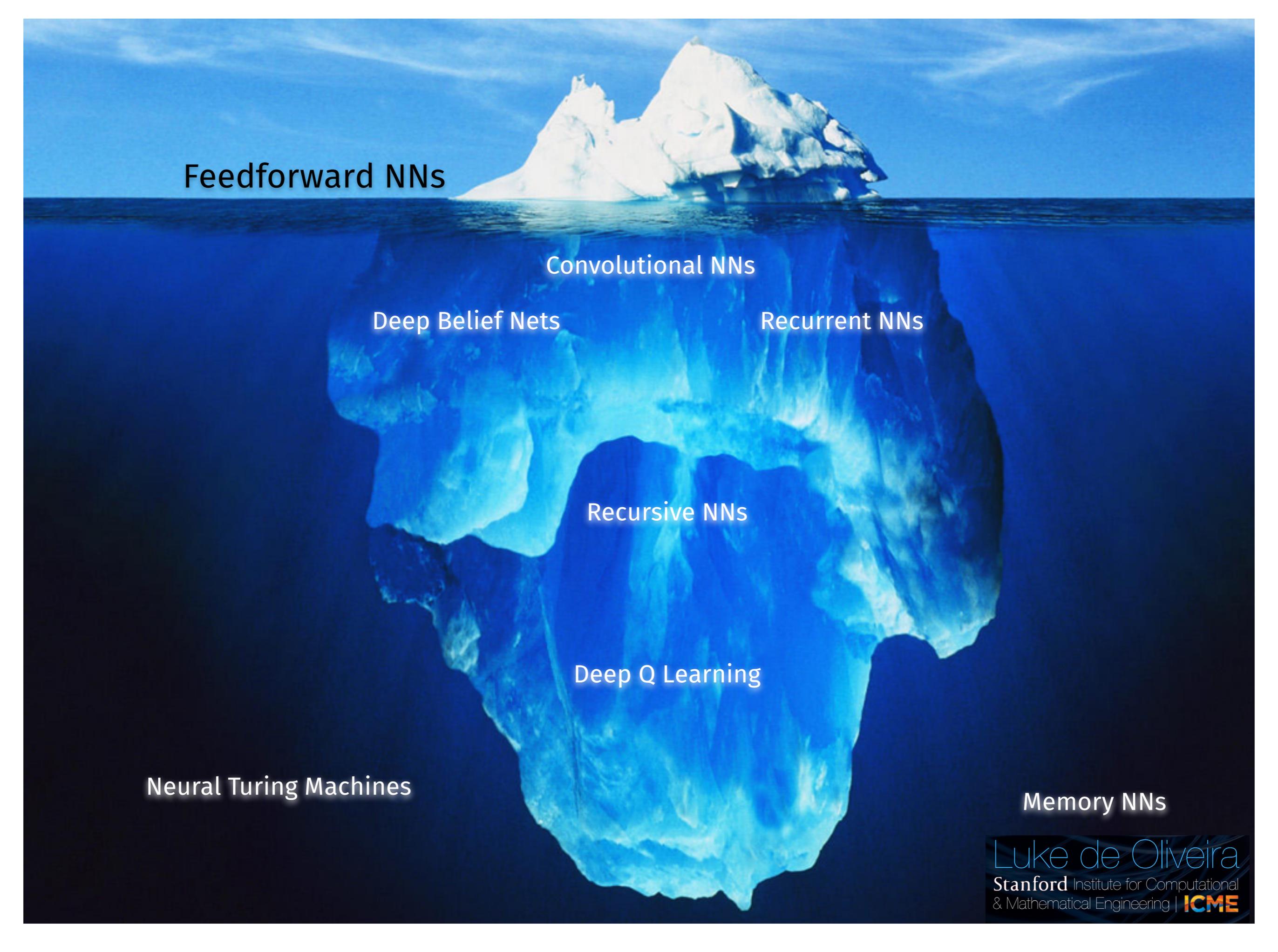
0.30 green

-0.09 behind

-0.14 her

# DNN Basics

- Representational networks: build an internal representation of input/output
- NNs can encapsulate arbitrary functions. Deeps NN are more efficient.
  - Dark Knowledge: e.g. Deep to Shallow NN.
- 2 Primary Classes of problems:
  - Classification
  - Regression
- 2 Classes of inputs:
  - Fix size input
  - Variable size inputs (typical solution: Recurrent NNs)
- Learning Approaches
  - *Supervised*: labeled data
  - *Unsupervised*: unlabeled data
  - *Reinforcement*: try, succeed/fail?

An iceberg floating in the ocean, with the visible tip above the water and a much larger, submerged part below. The visible tip is labeled 'Feedforward NNs'. The submerged part is divided into several sections, each labeled with a different type of neural network: 'Convolutional NNs' at the top of the submerged part, 'Deep Belief Nets' on the left, 'Recurrent NNs' on the right, 'Recursive NNs' in the center, 'Deep Q Learning' below that, 'Neural Turing Machines' at the bottom left, and 'Memory NNs' at the bottom right. The water is a deep blue, and the sky is a lighter blue with some clouds.

Feedforward NNs

Convolutional NNs

Deep Belief Nets

Recurrent NNs

Recursive NNs

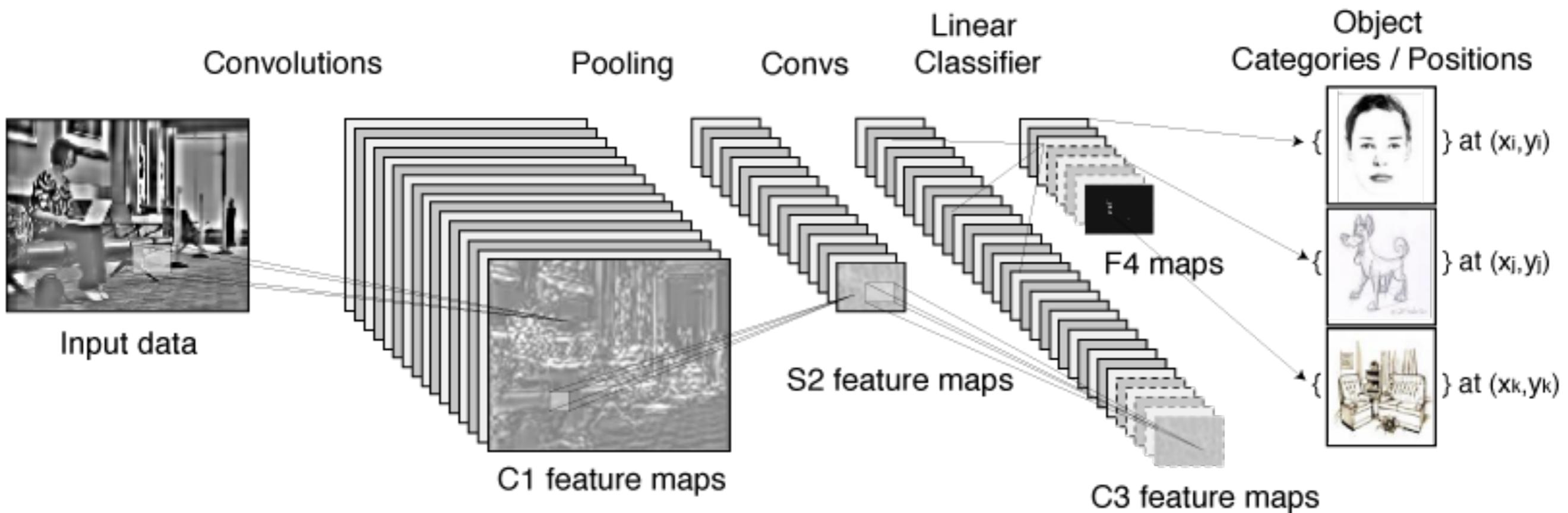
Deep Q Learning

Neural Turing Machines

Memory NNs

# Convolutional NN

- 1D: Time series, 2D: images, 3D: video



Faces



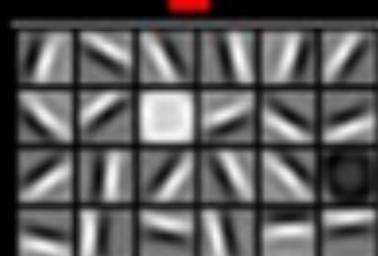
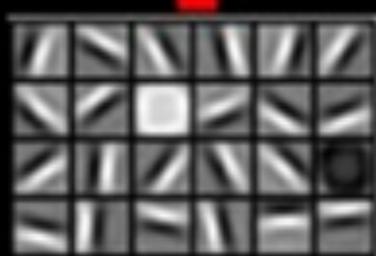
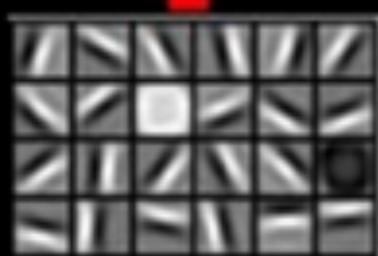
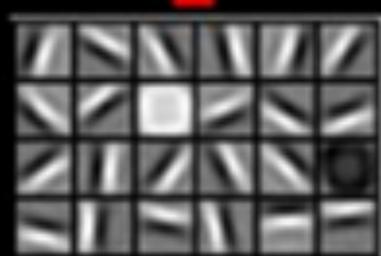
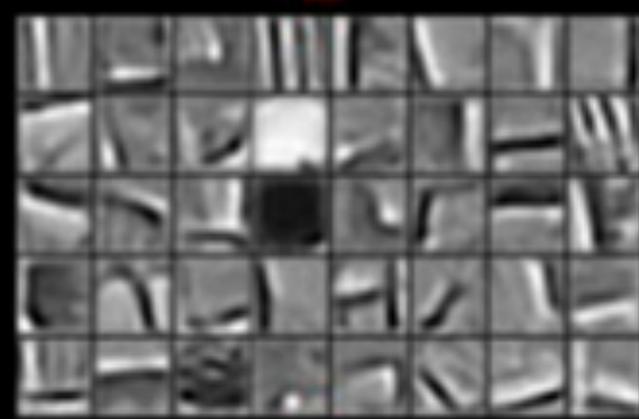
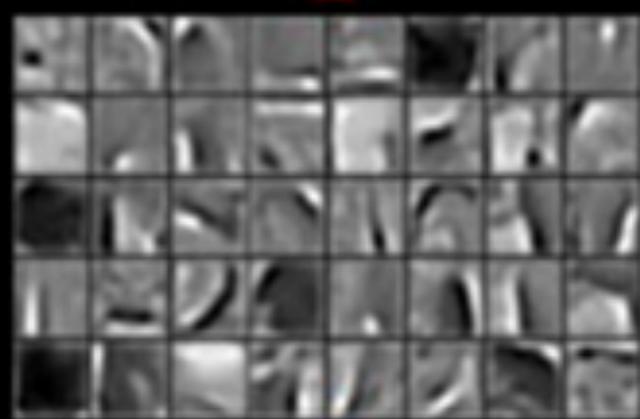
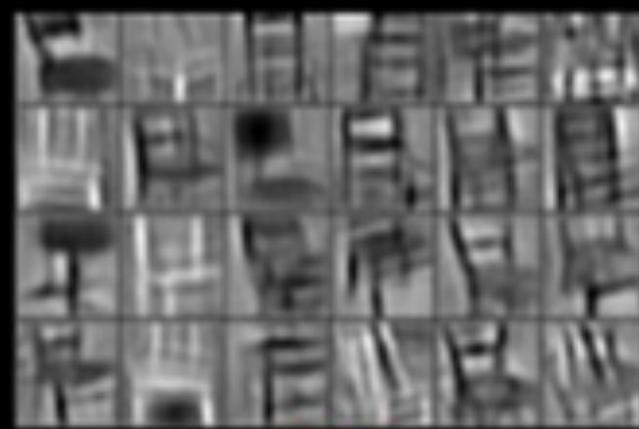
Cars



Elephants

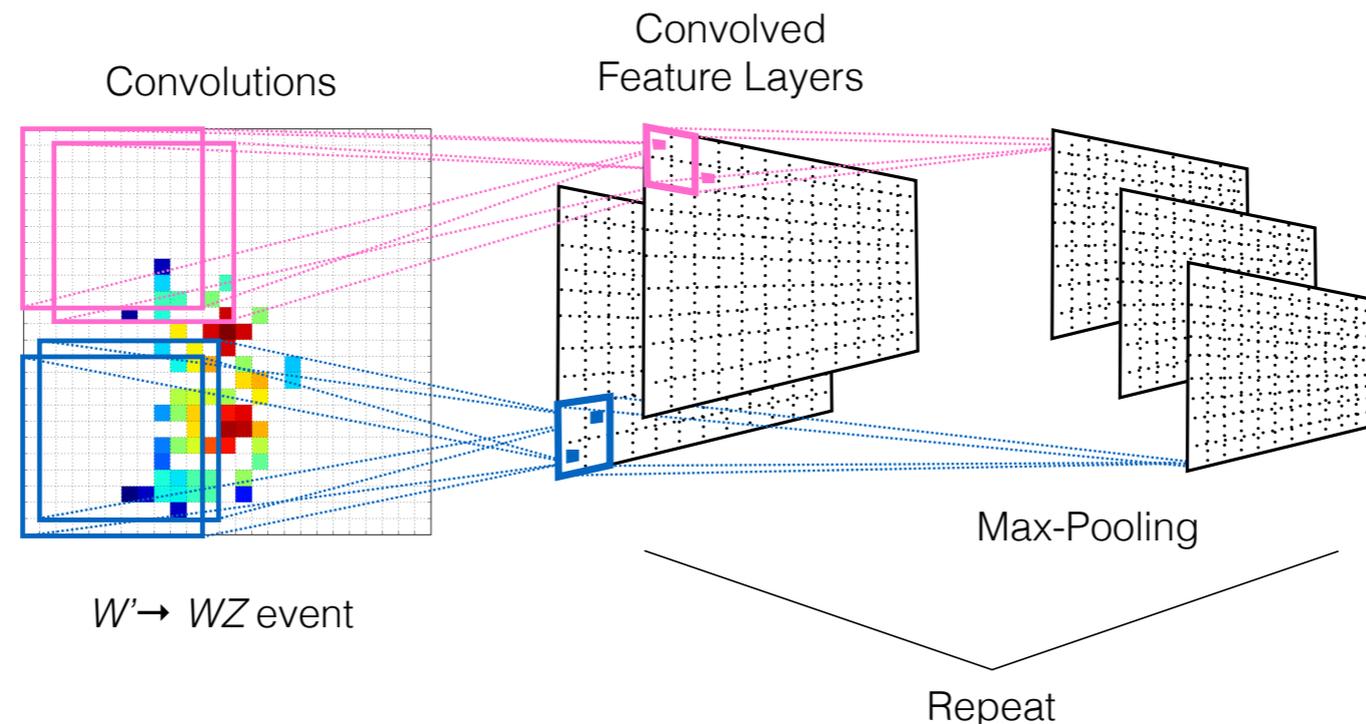


Chairs



# HEP and DNN

- D. Whiteson, P. Baldi, and P. Sadowski (UCI) showed that ([NatureCom](#), [Archive](#))
  - DNN can work better than other techniques for sig/bkg rejection for searches.
  - DNNs can reproduce physics features (e.g. masses) from four vectors.
- Proof that DNNs use “Renormalization” to recognize cats ([Wired](#))
- SLAC group is classifying jets and analyzing jet substructure ([Slides](#))
- I know of others... but through private conversations...
- Seminar at Fermilab last week on DNN in Nova (by Adam Aurisano)



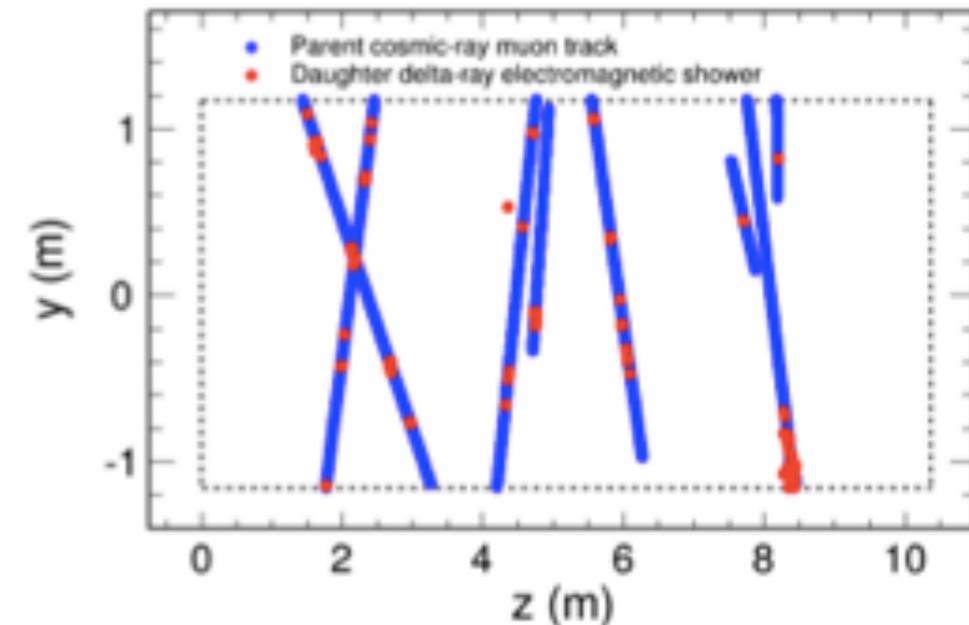
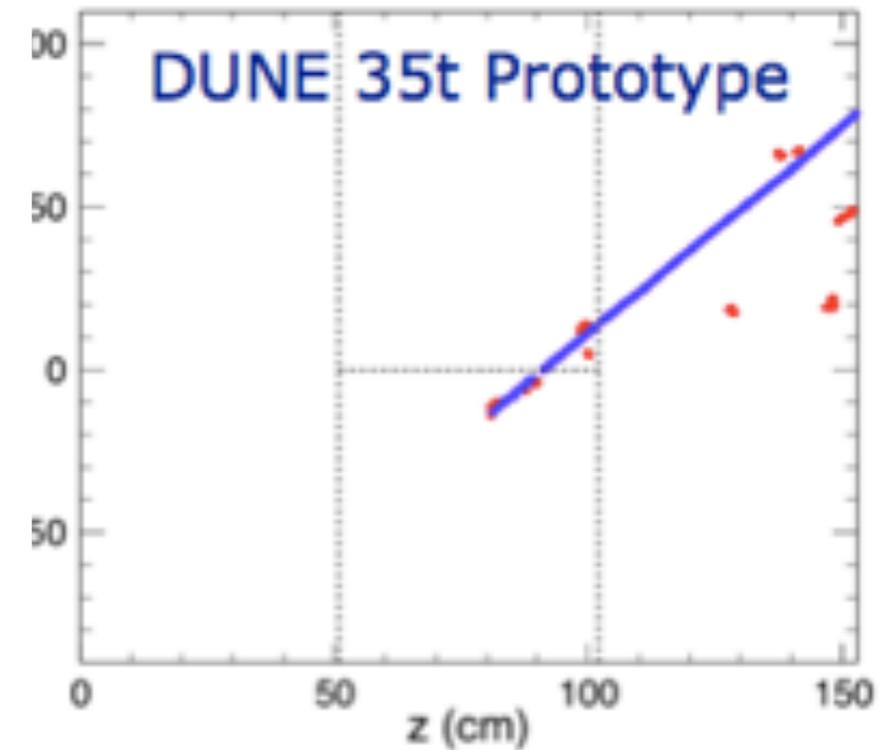
# DNN Reconstruction

# DNN Reco

- Motivations?
  - Hopefully DNN-based feature extractors *out perform* hand crafted reconstruction algorithms.
  - After training, DNN-based will likely to be much *faster* than algorithmic reconstruction. And it's already running on GPUs.
  - For LArTPC, it may be able to do something we cannot do algorithmically.
  - There is incredible value (CHF/dollars) in the fact that DNN may allow performing reconstruction *without physicists writing algorithms*.

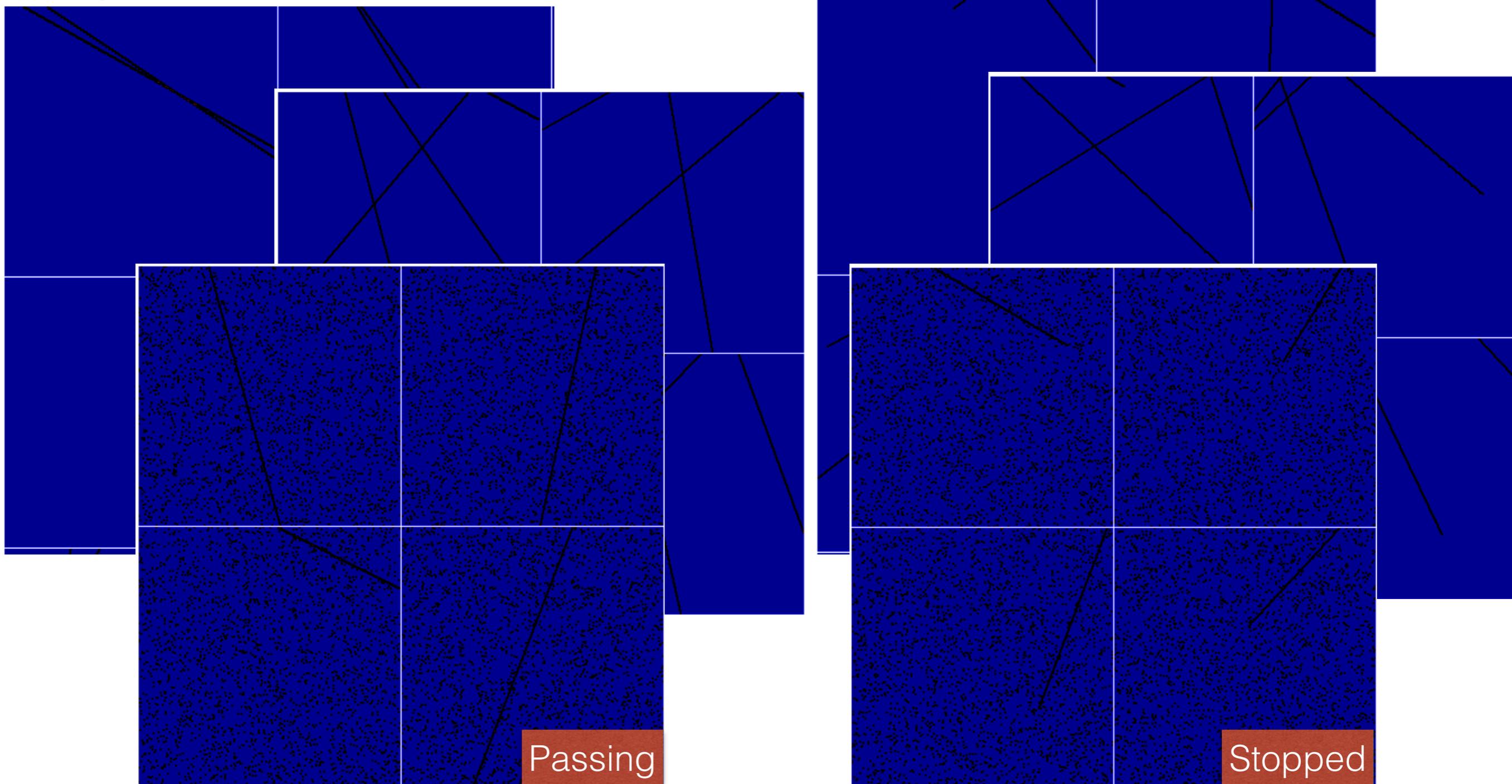
# CNN for LArTPC

- My first attempt: DUNE-35 ton
- Goal: classify raw data: Stopped muons, radiated photons, horizontal cosmics, ... are small but very useful subset of data.
- How do we make samples of these w/o full reconstruction?
- I faked some events passing/stopped muon events, fed it to NVidias DIGITS, which is a DNN image classification tool.
  - Really just a very simple proof of principle test.



# DNN Classification of “Raw” LArTPC Data

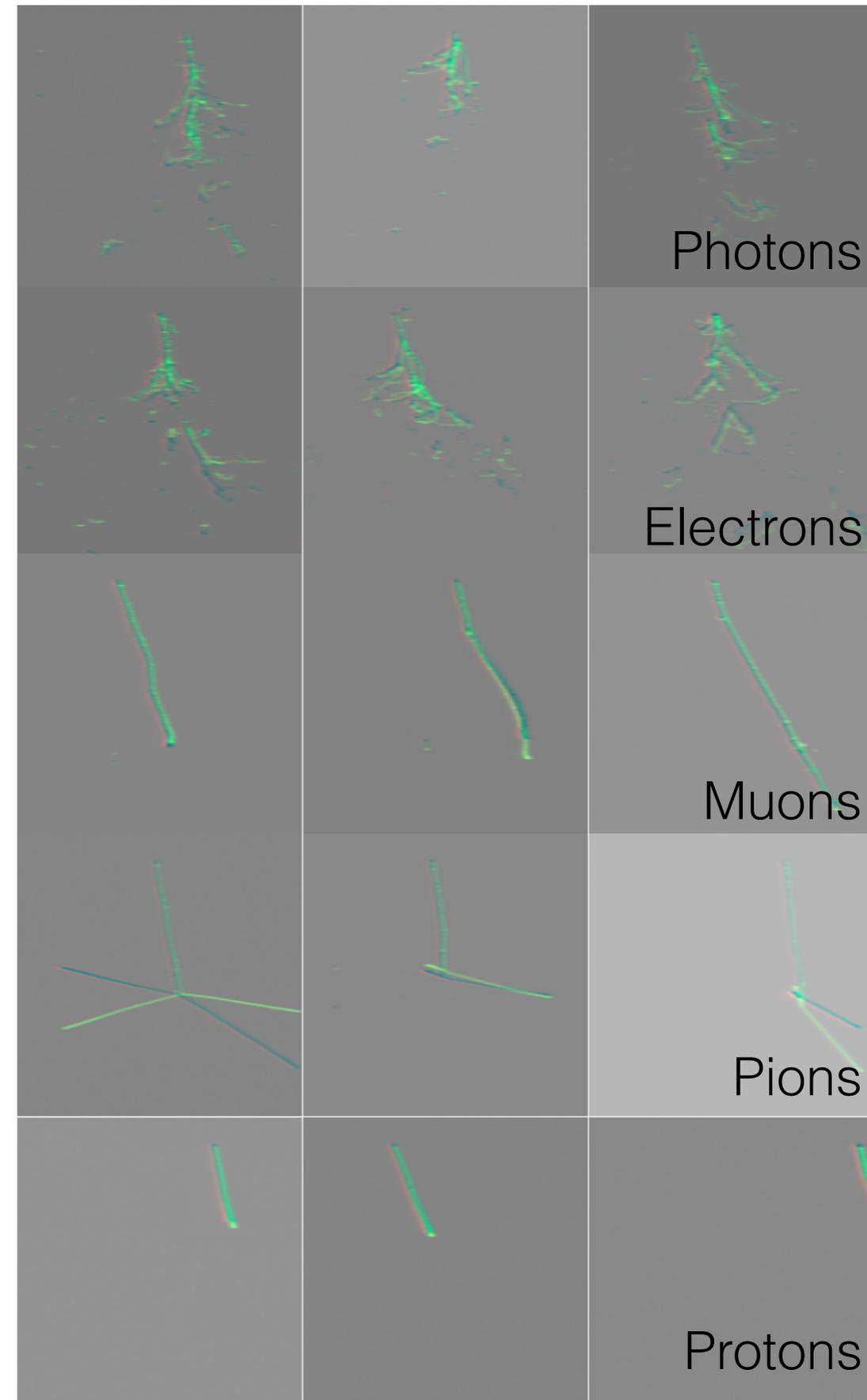
GoogleLeNet 256x256



1-4 Tracks With or without noise, DNN correctly classifies ~90-99%

# Particle ID in LArIAT

- Another proof of principle test with lots of obvious things to fix.
- LArIAT is a small detector: 2 wire planes with 240 wires each, 4096 samples.
- Generated single 500 +/- 200 MeV particles. 50k of each type.
- Used standard implementation of GoogLeNet, which takes 224x224 pixel color images (png files!).
- Reduced samples by
  - Down sample: Summing N ticks, where N=1-8.
  - Scanning for the box with maximum total charge.
  - 2 planes -> R and G color intensities.
- Very preliminary... not completely understood.



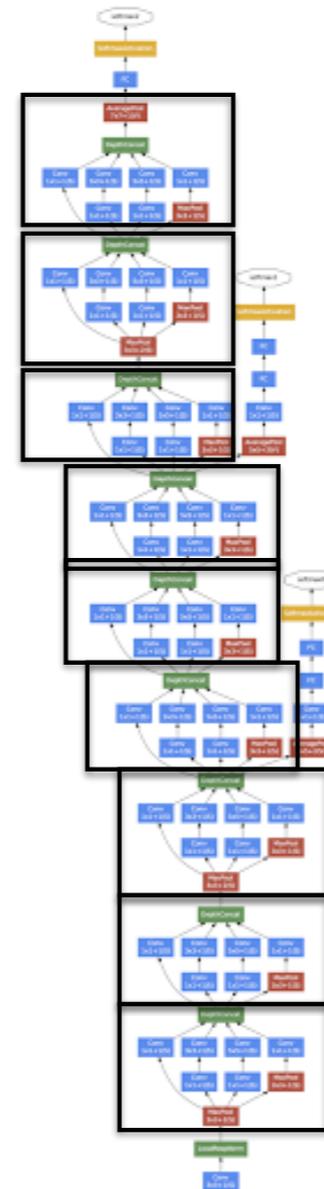
# GoogLeNet

2014-era Model for Object Recognition



 Module with 6 separate convolutional layers

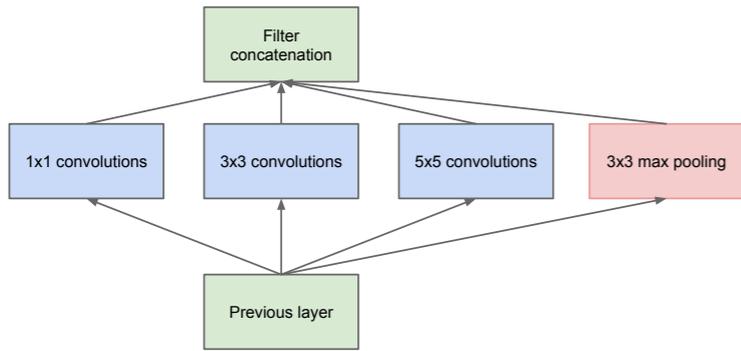
24 layers deep!



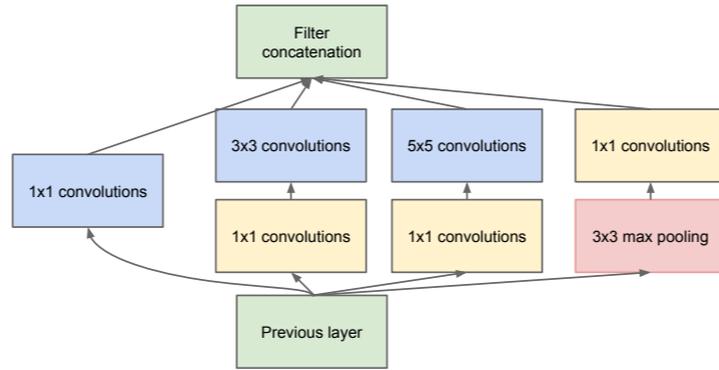
Developed by team of Google Researchers:

Won 2014 ImageNet challenge with **6.66%** top-5 error rate

<http://arxiv.org/abs/1409.4842>



(a) Inception module, naïve version



(b) Inception module with dimension reductions

*64 Convolution in First Layer*

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

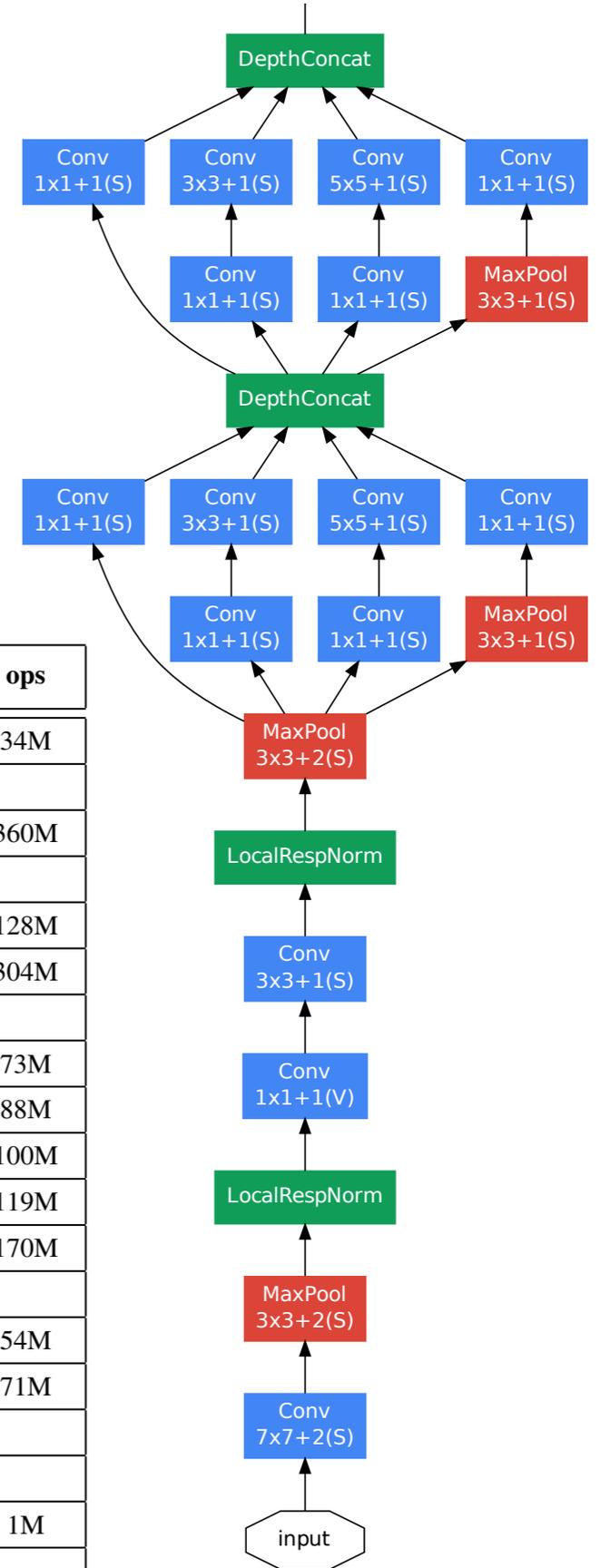
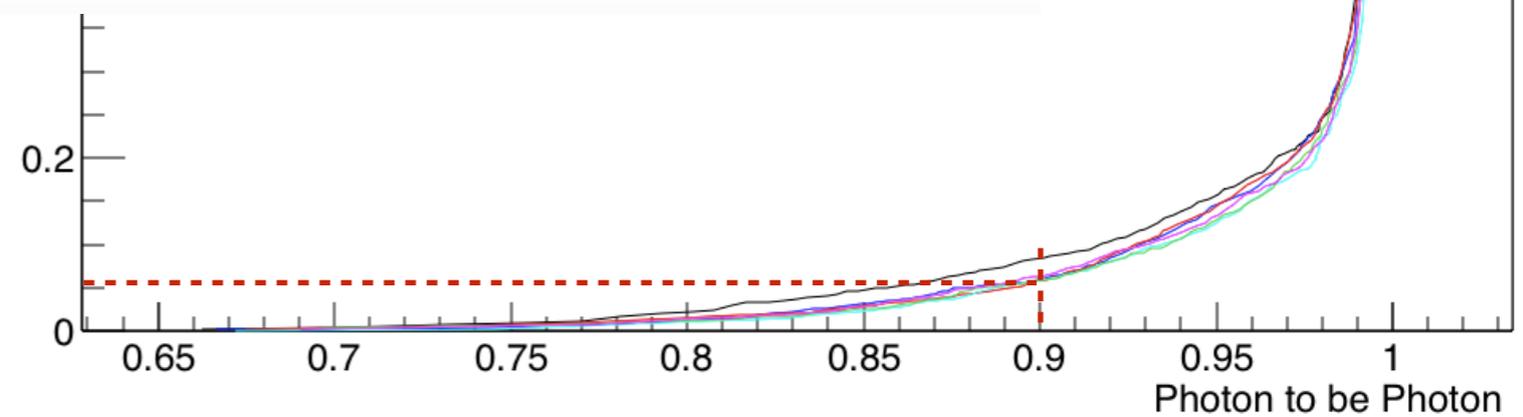
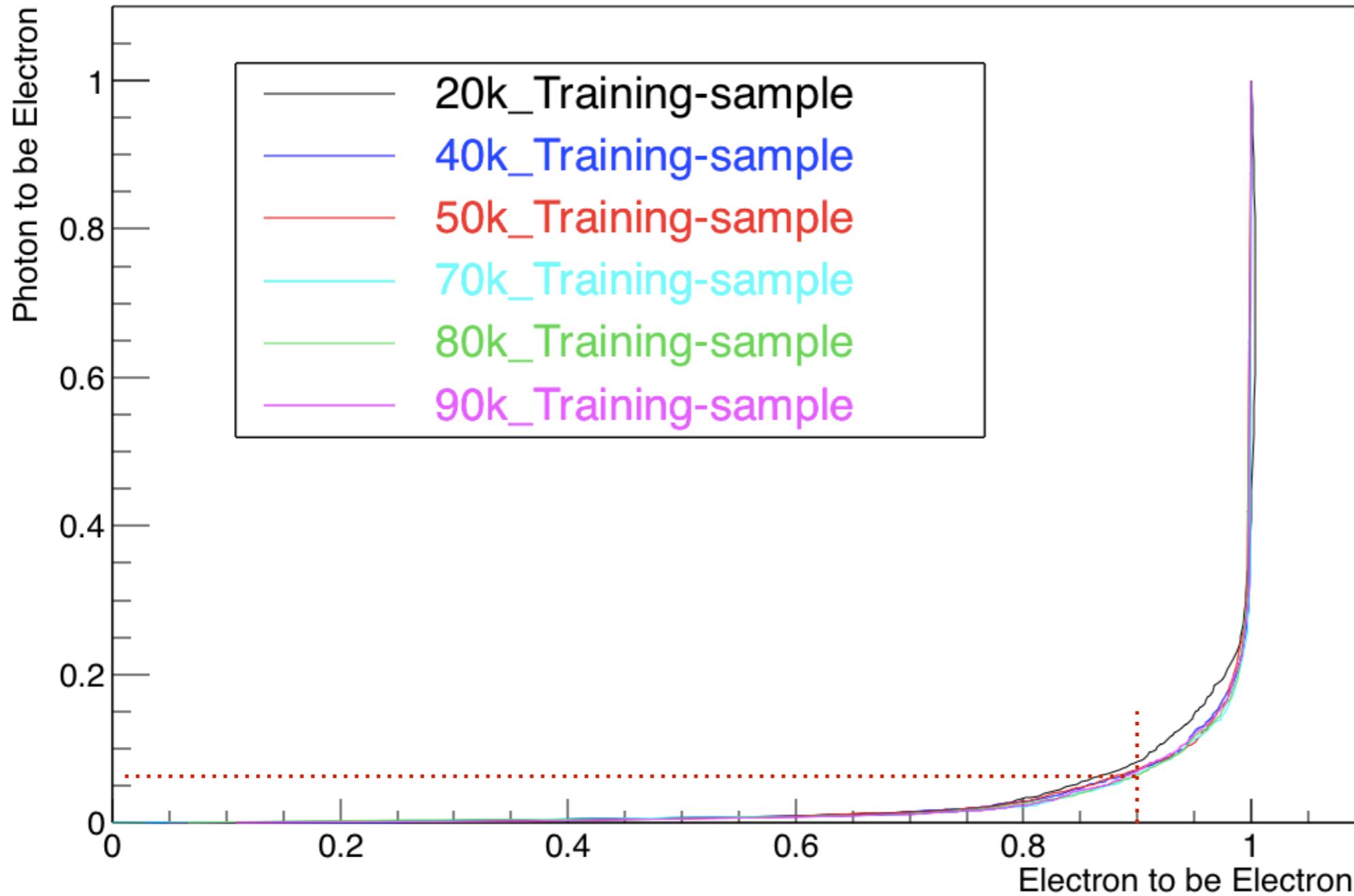


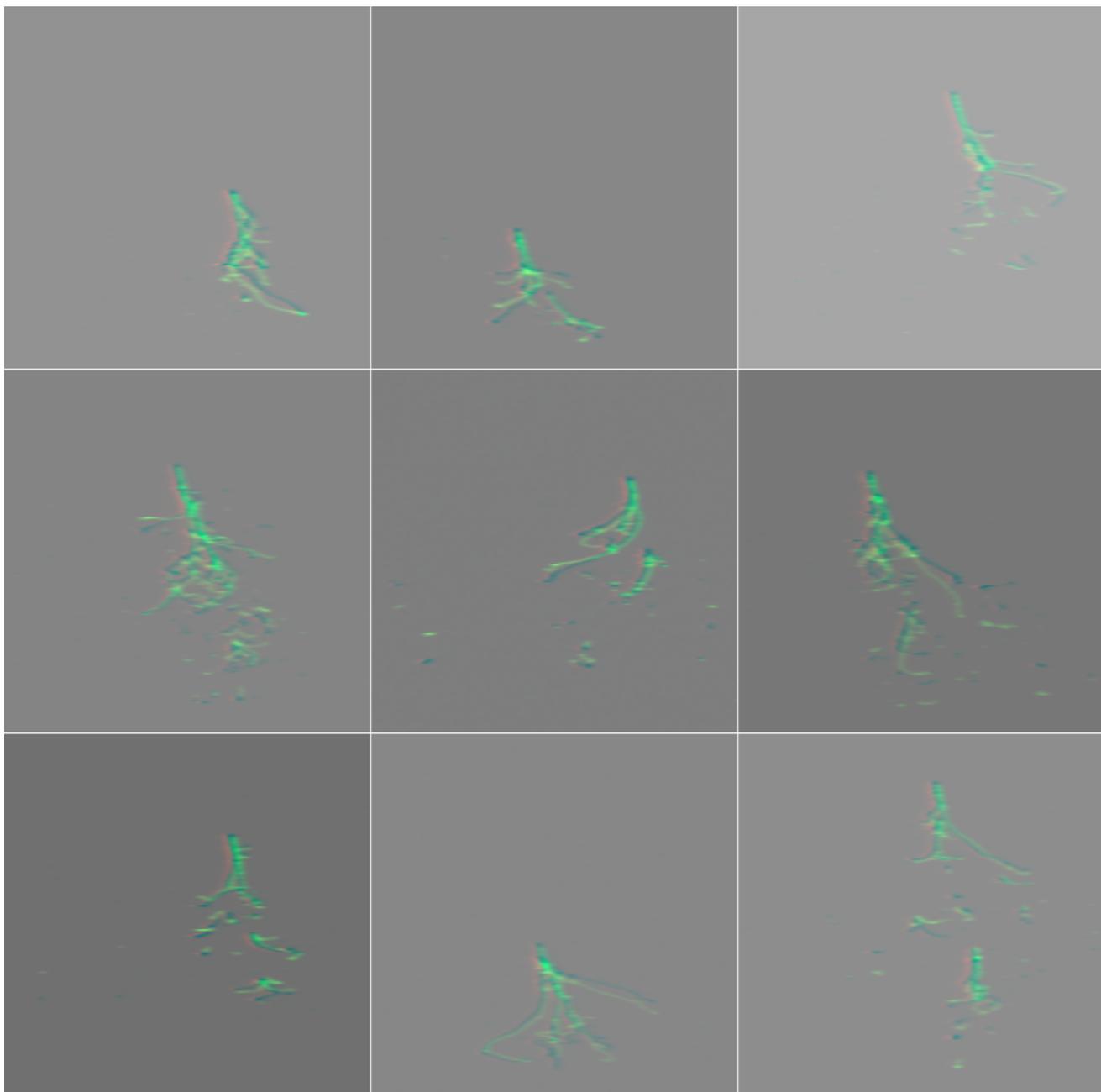
Table 1: GoogLeNet incarnation of the Inception architecture

# Electron vs Photon

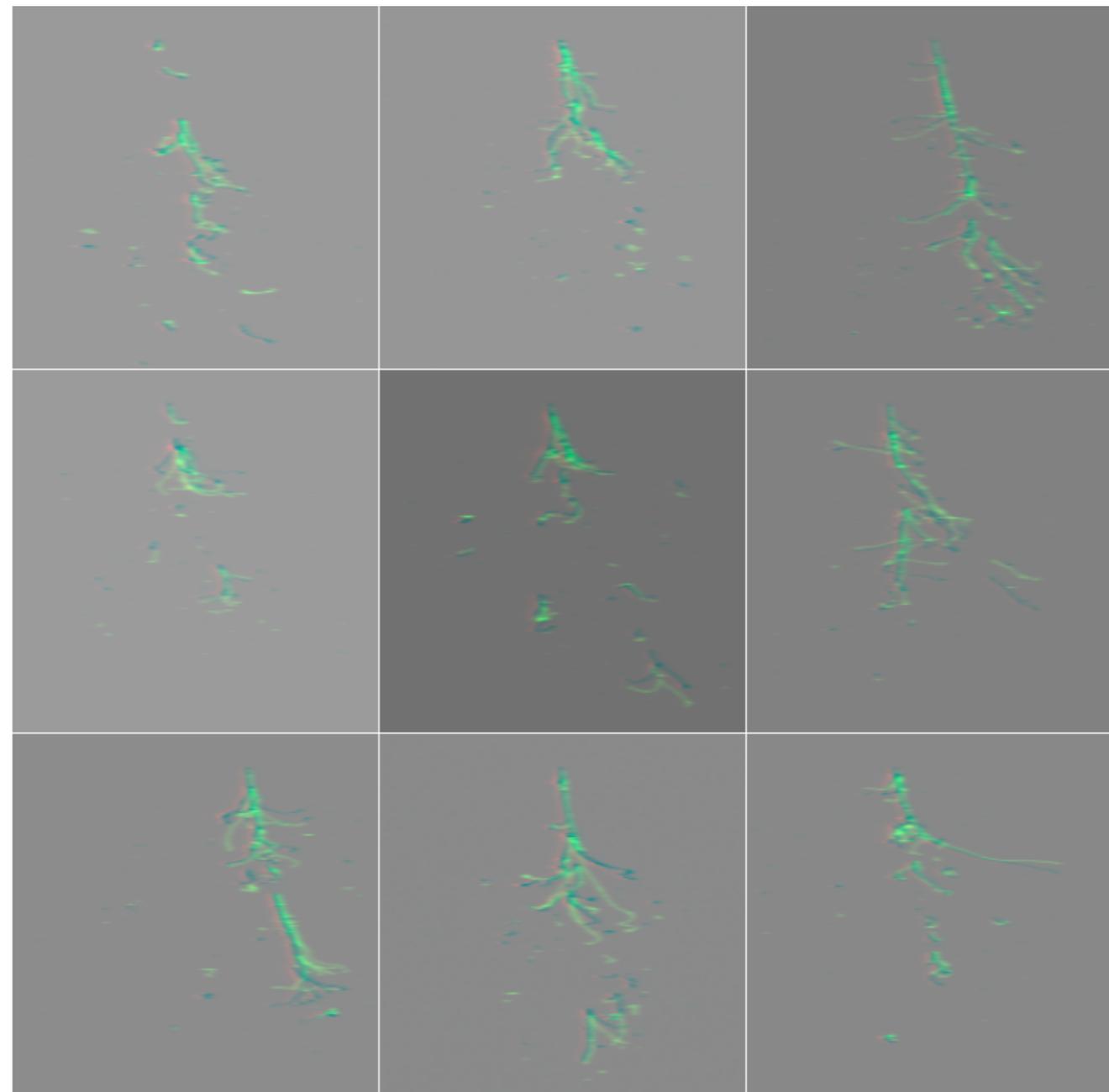


How you train is important

# Electron vs Photon

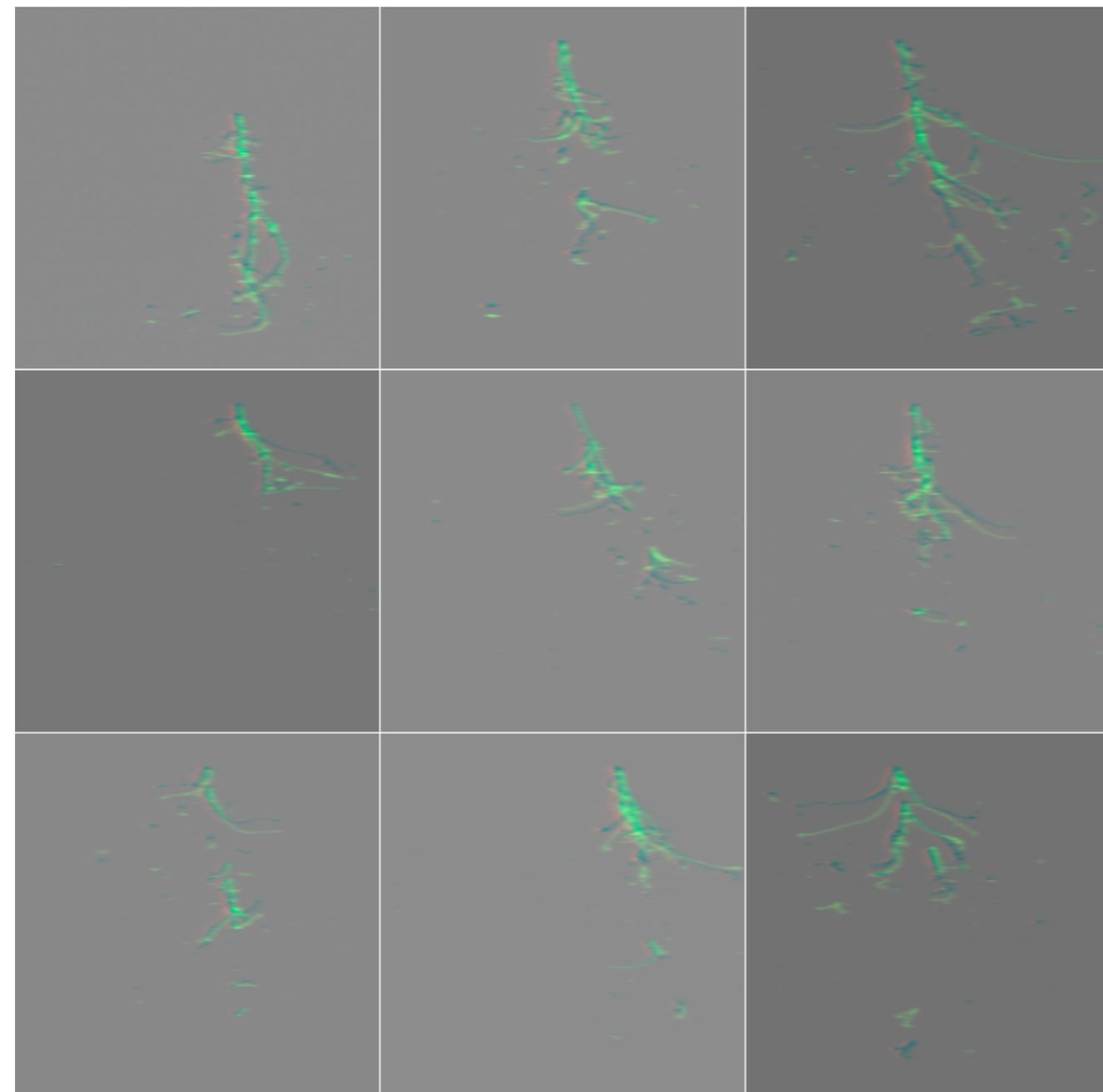


Real Photons ID as Photons

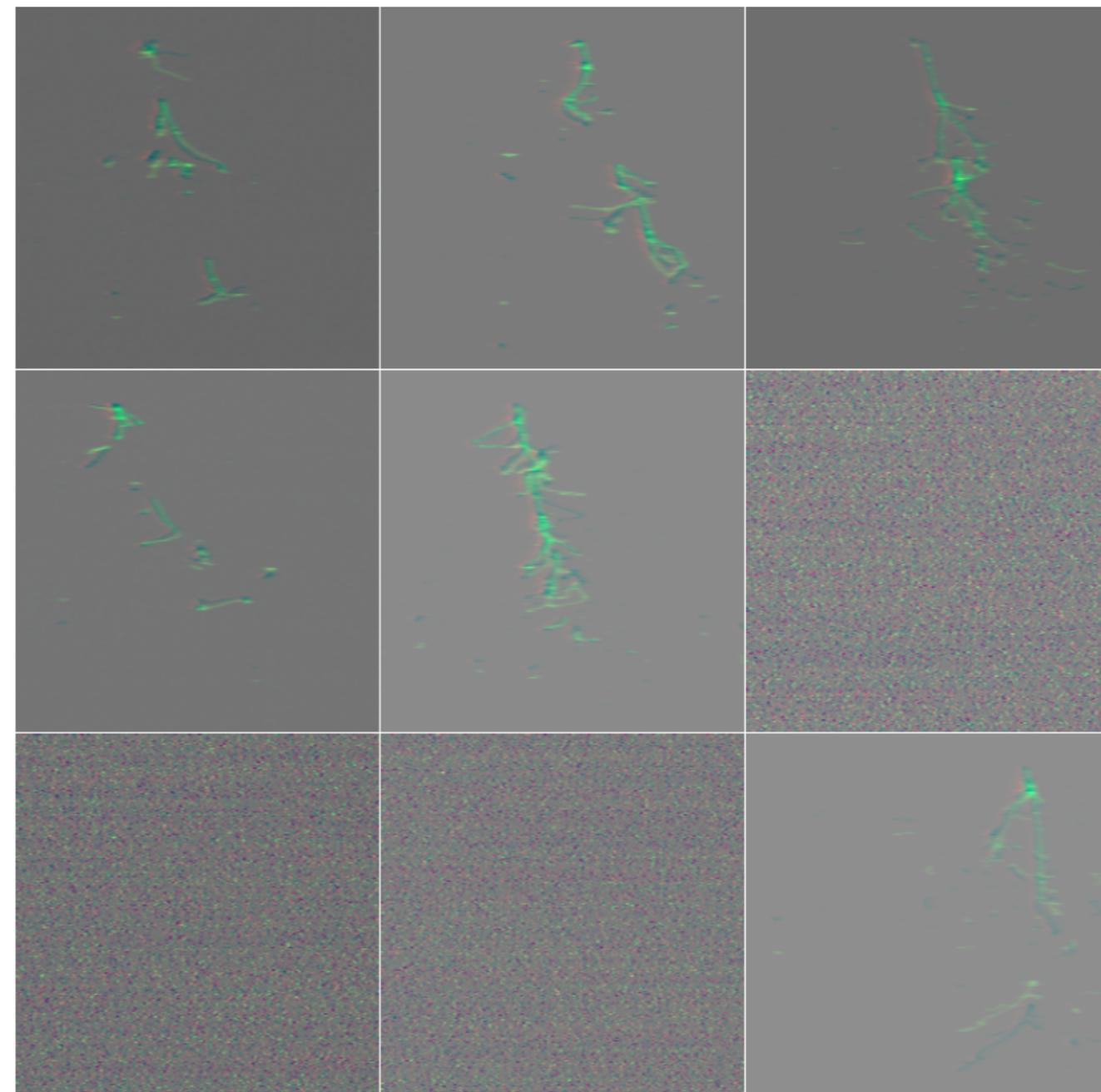


Real Electrons ID as Electrons

# Electron vs Photon

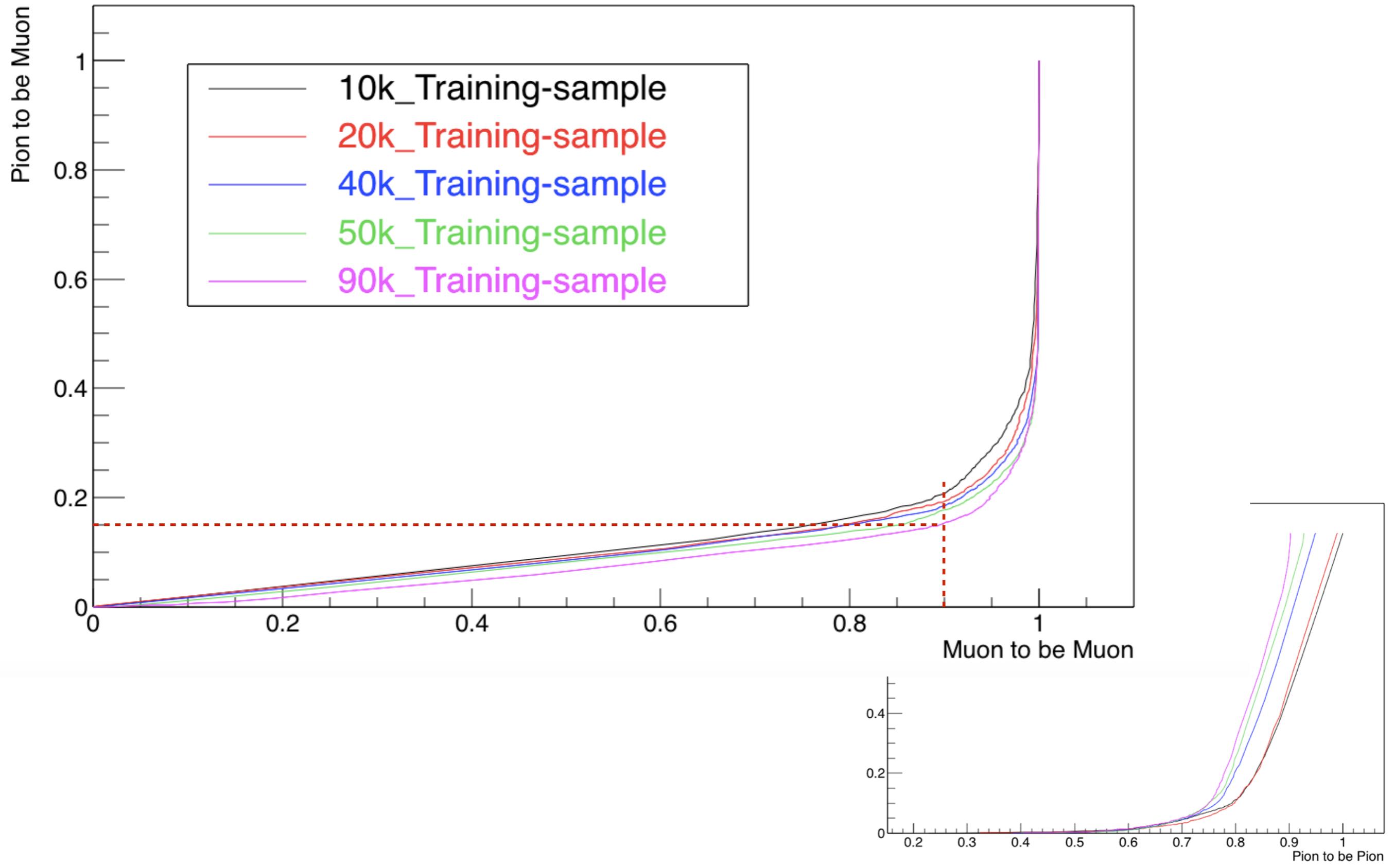


Real Electrons ID as Photons

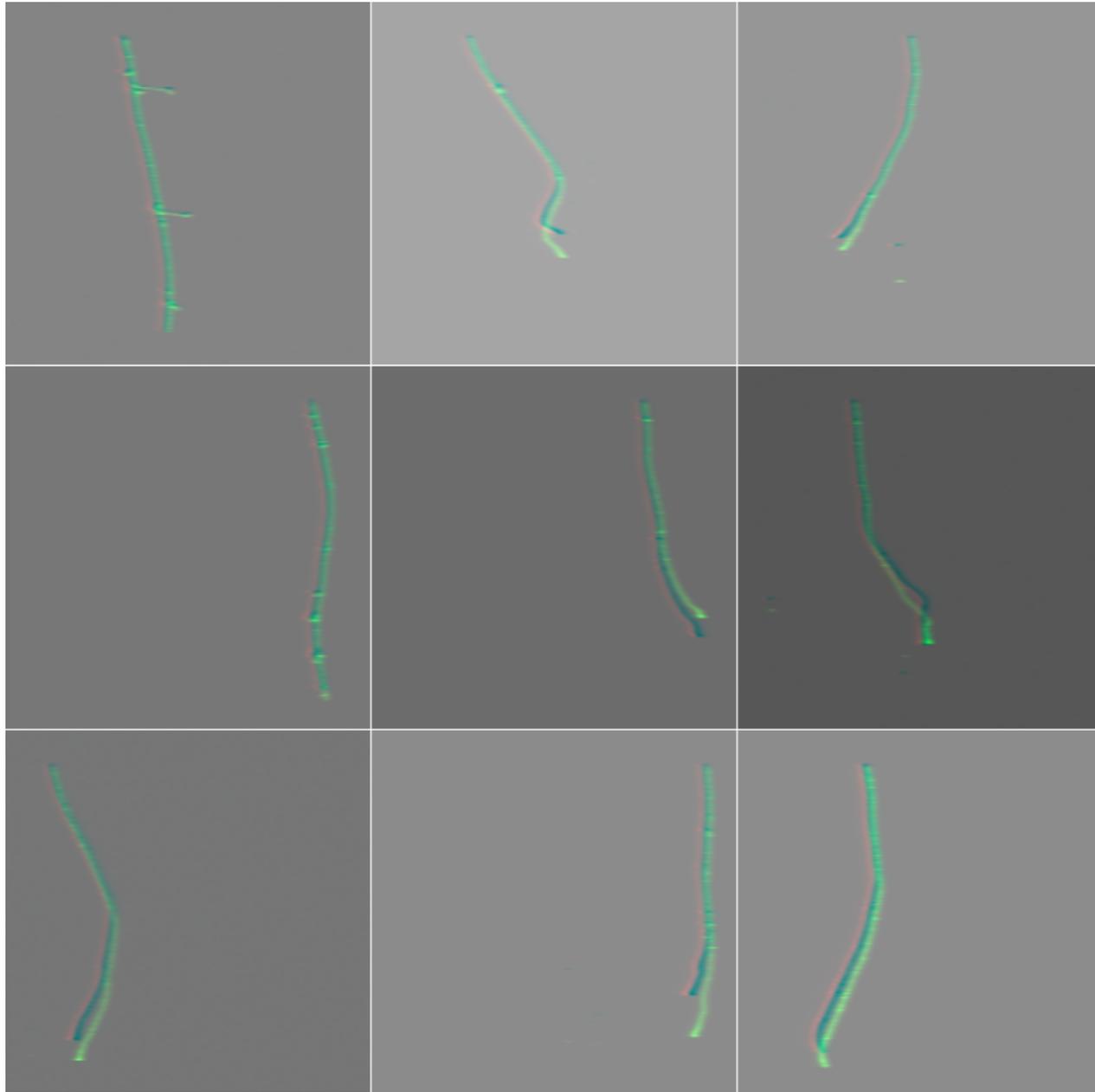


Real Photons ID as Electrons

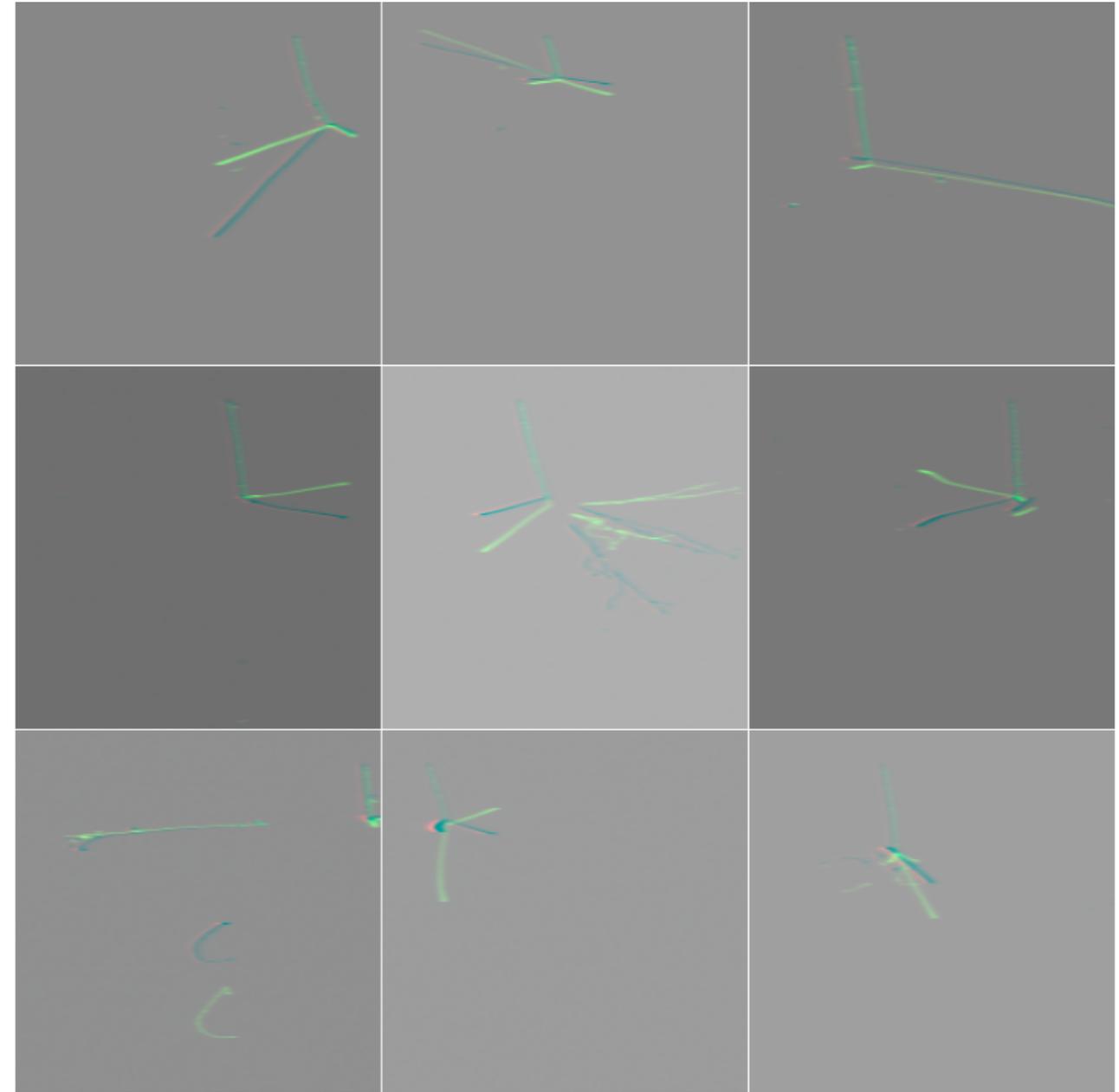
# Muon vs Pion



# Muon vs Pion

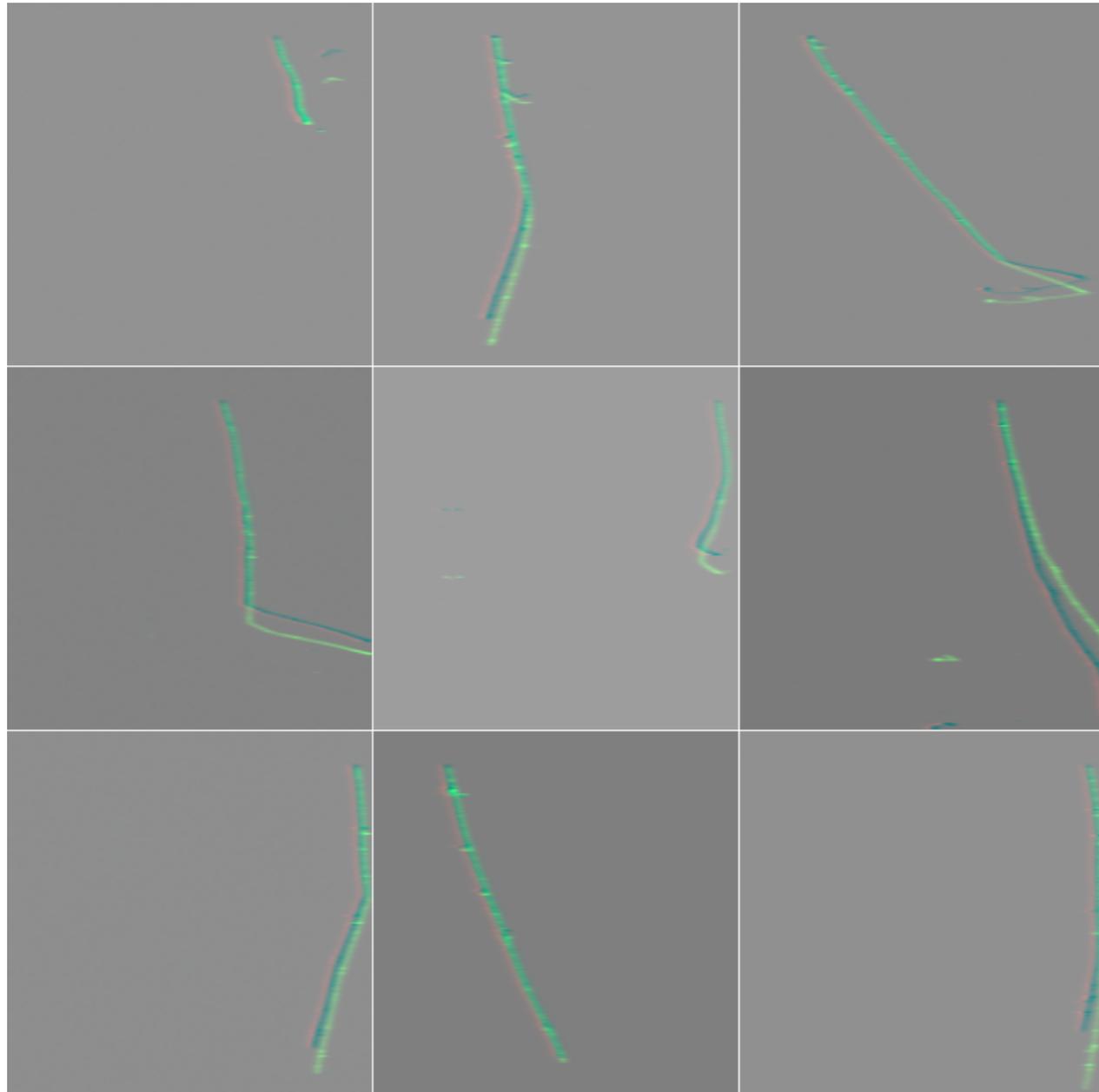


Real Muons ID as Muons

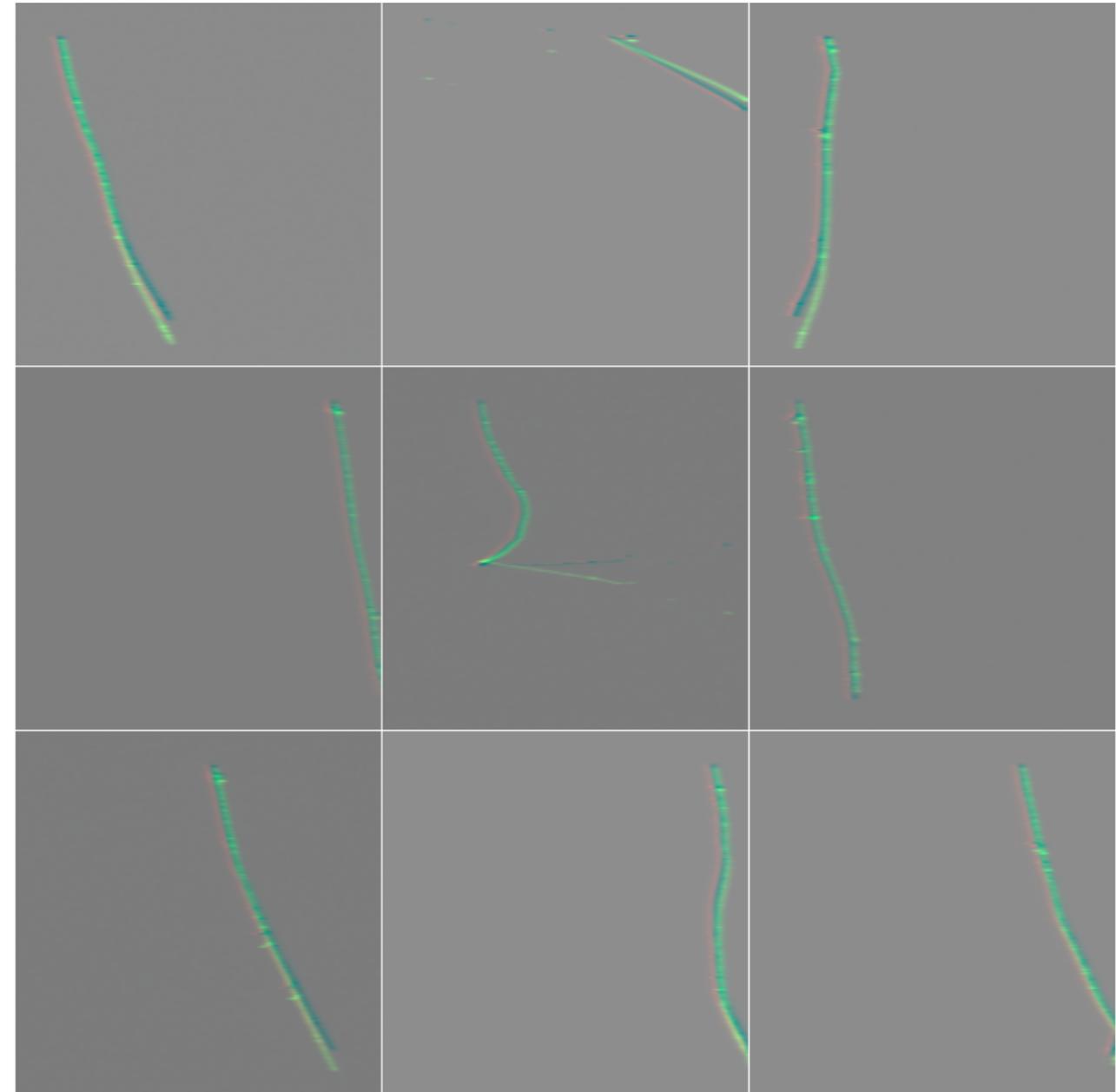


Real Pions ID as Pions

# Muon vs Pion

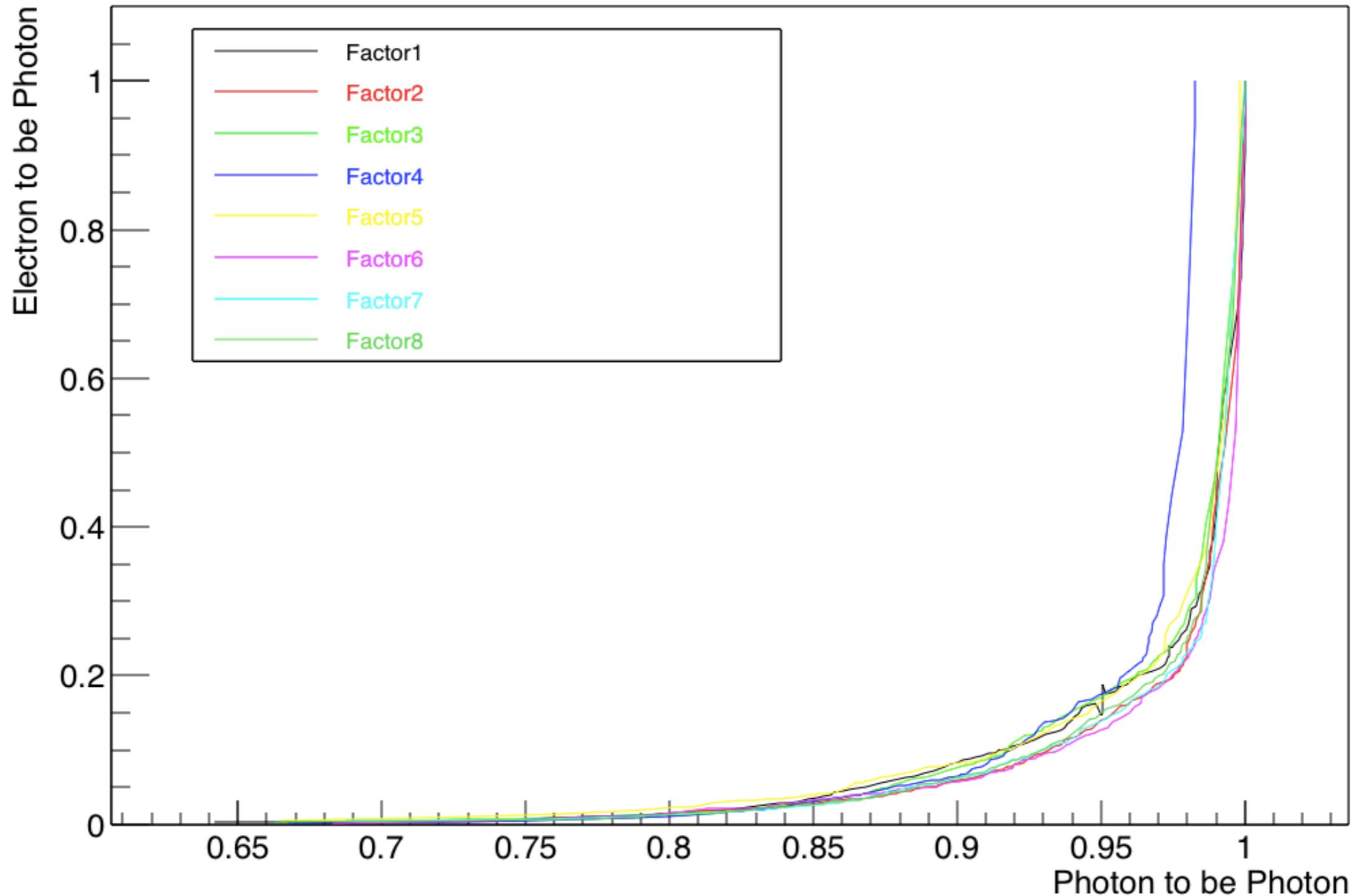


Real Muons ID as Pion



Real Pions ID as Muon

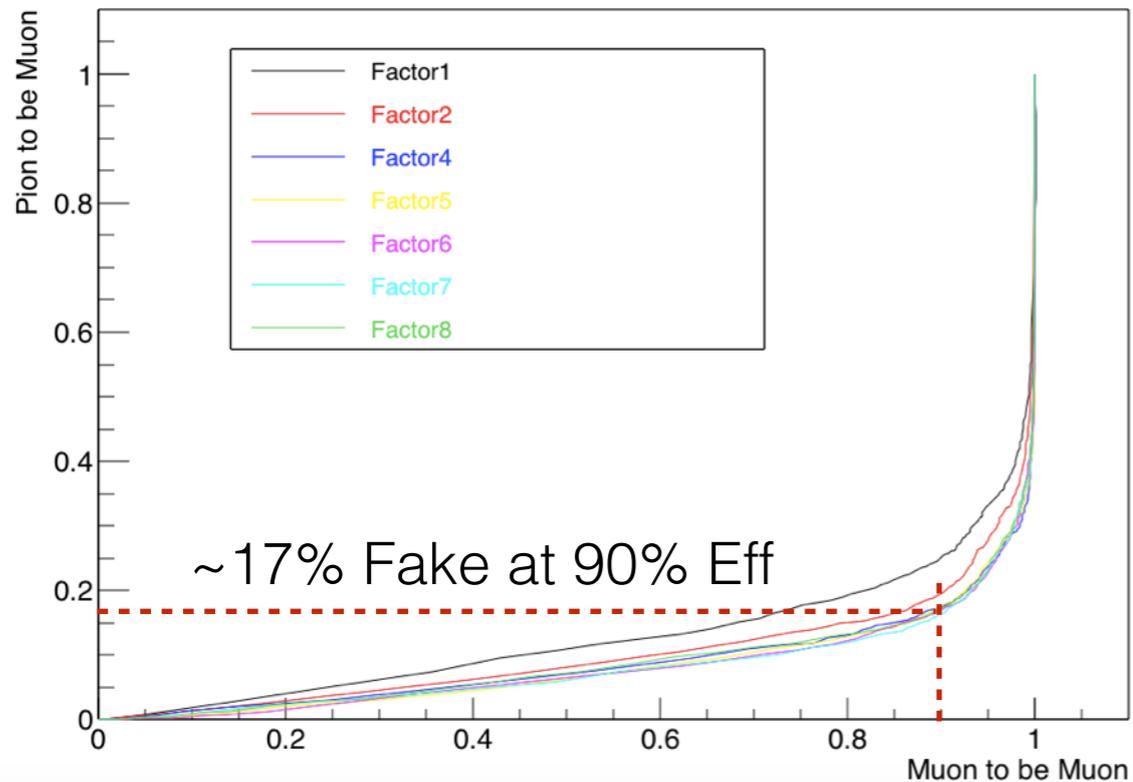
# Down Sampling



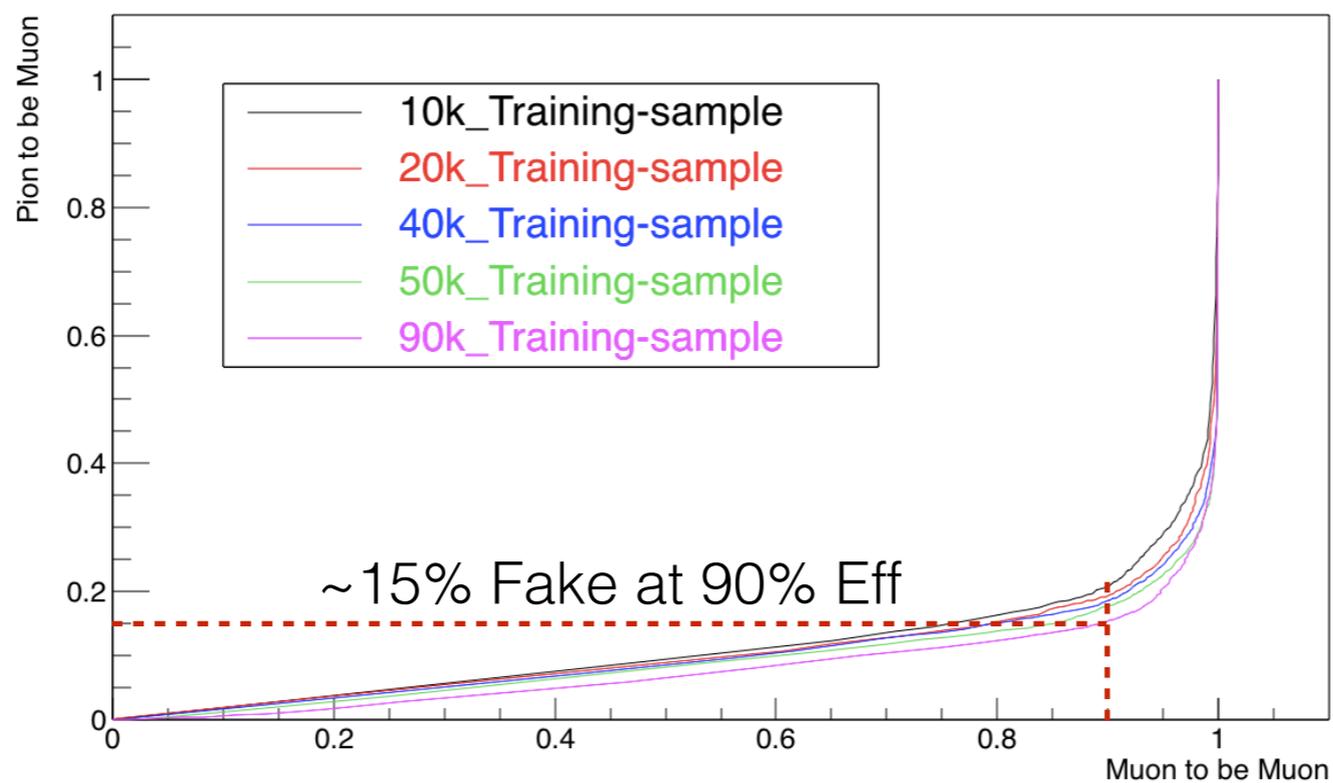
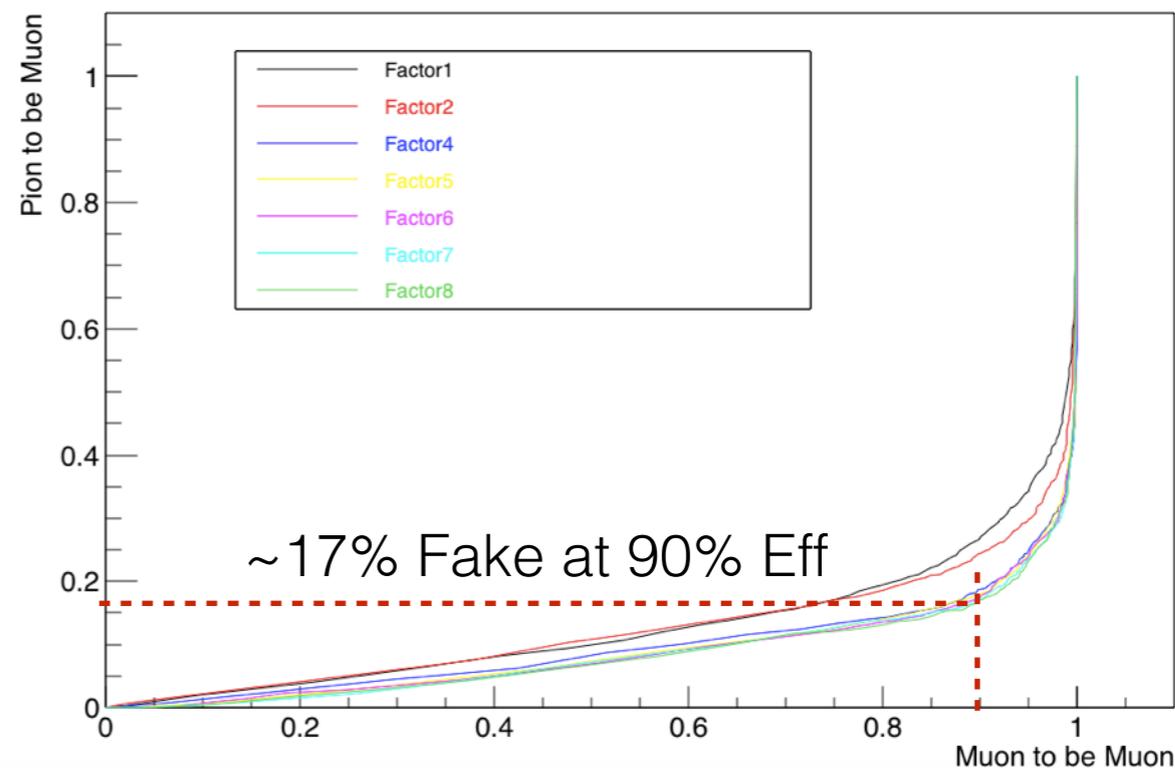
Competition between resolution vs capturing whole interaction

# Single Plane

Collection Plane \_ Efficiency Plot

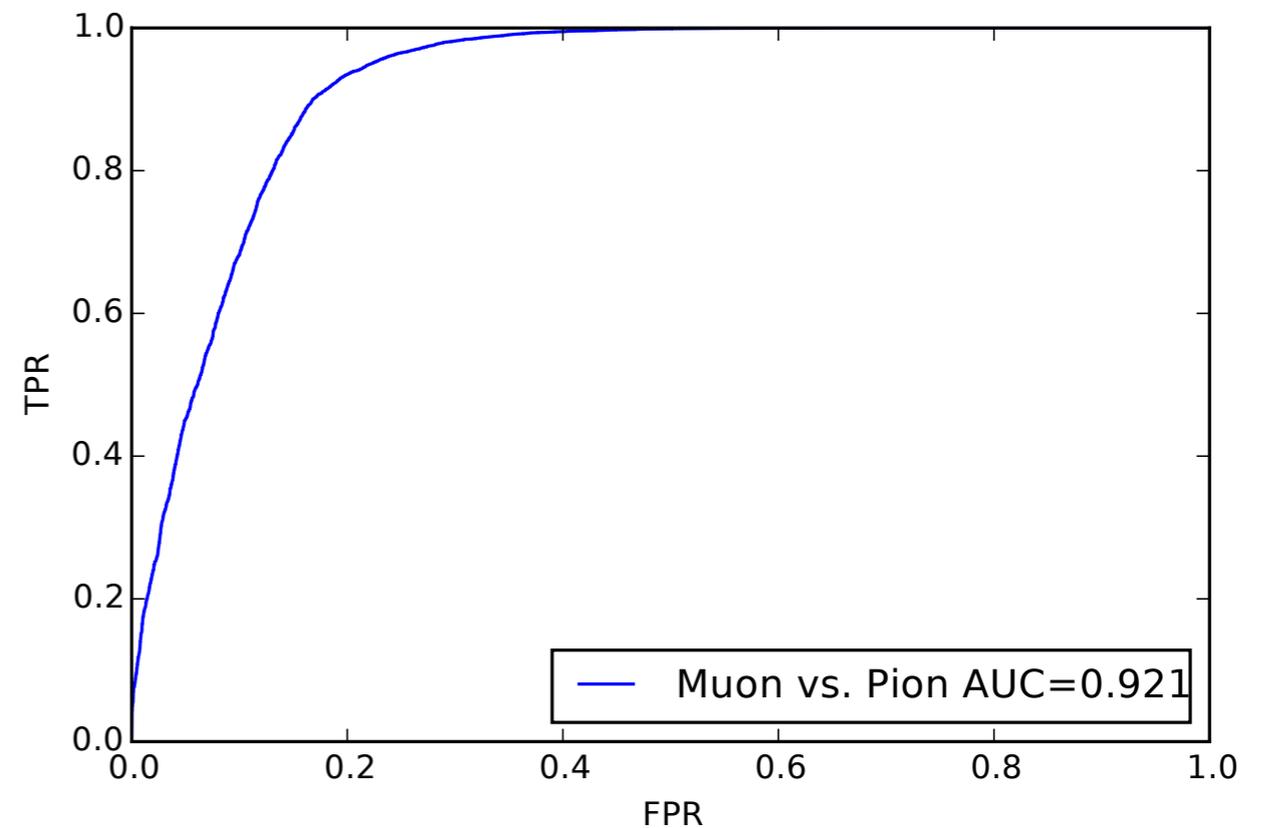
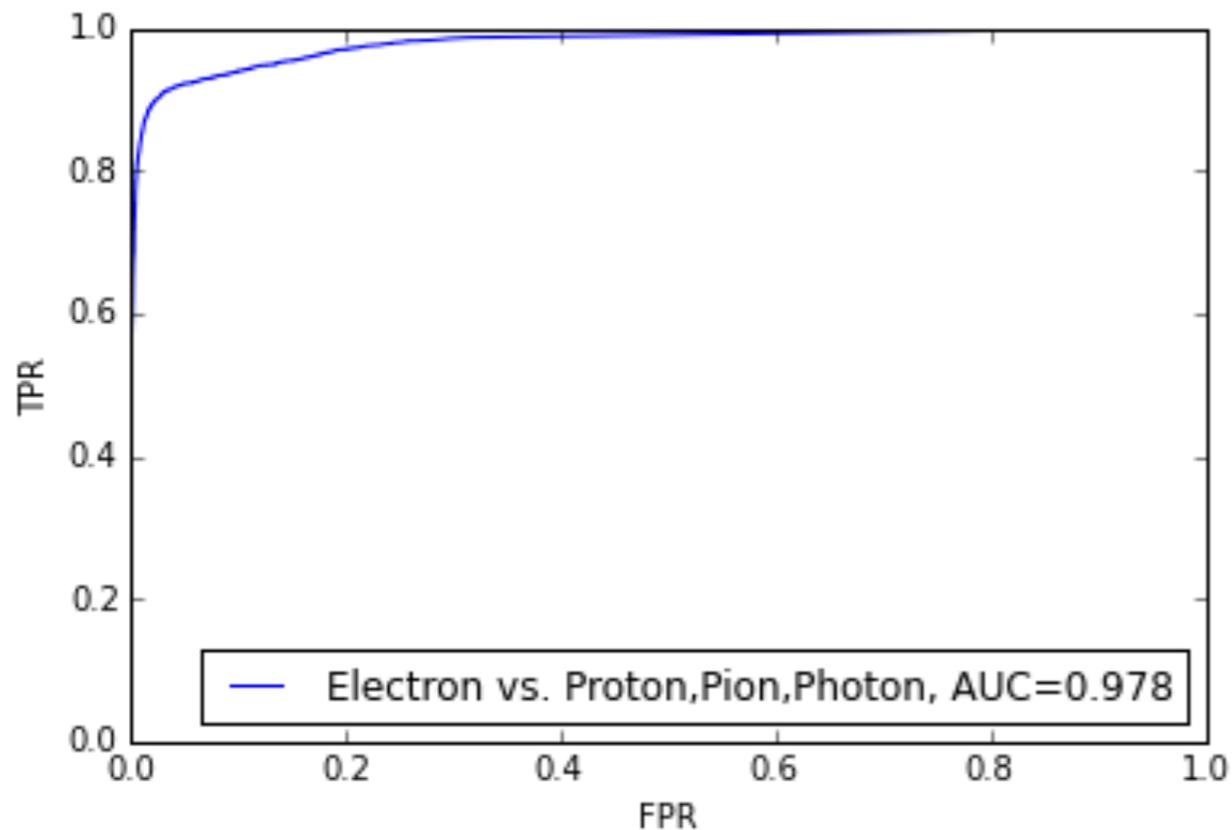


Induction Plane \_ Efficiency Plot



# Collaborators

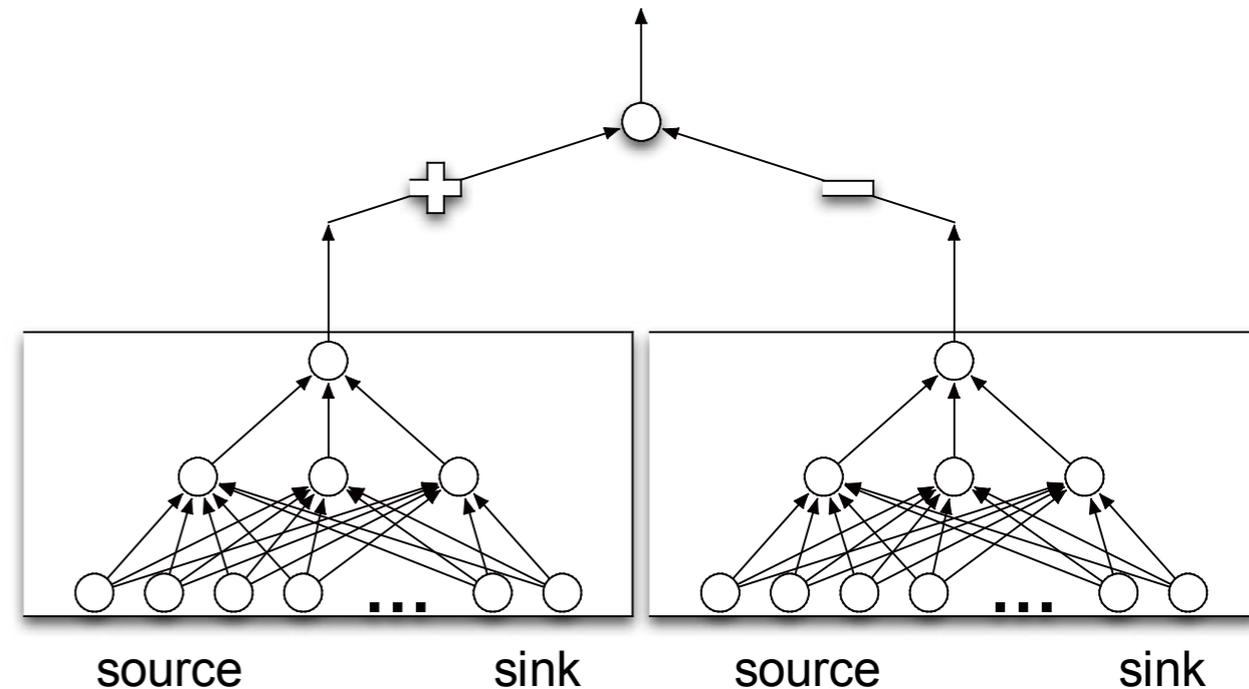
- Shared data with P. Baldi and P. Sadowski (UCI). Very First Attempt...
- Using similar inception-based CNN with 8x down-sampling and packing planes into different color intensities.
  - Better tuned training. Clear indications of over-training.
  - Signal: Electrons, Bkg: Photons, Pions, Proton → 2% fake rate for 90% signal efficiency
  - Signal: Electron, Bkg: Photons → 6% fake rate for 90% signal efficiency
  - Signal: Muon, Bkg: Pion → 17% fake rate at 90% signal efficiency



# Next Steps

- Training neutrino samples right now.
- First task: how do we properly feed in the different views?
  - Siamese NN (similar to Nova)
  - Perhaps we could train a network to resolve the ambiguities ( $1D \times 1D \rightarrow 2D$ ), similar to WireCell?
  - Perhaps  $2D \times 2D \rightarrow 3D$ , to take advantage of “connectivity” and topology.
- We are over-training
  - What is appropriate size of convolutions/ network? Hyper-parameter optimization.
- Adding regression to estimate the energy should be straight forward.
- Sub-classify events depending on final state and topology.

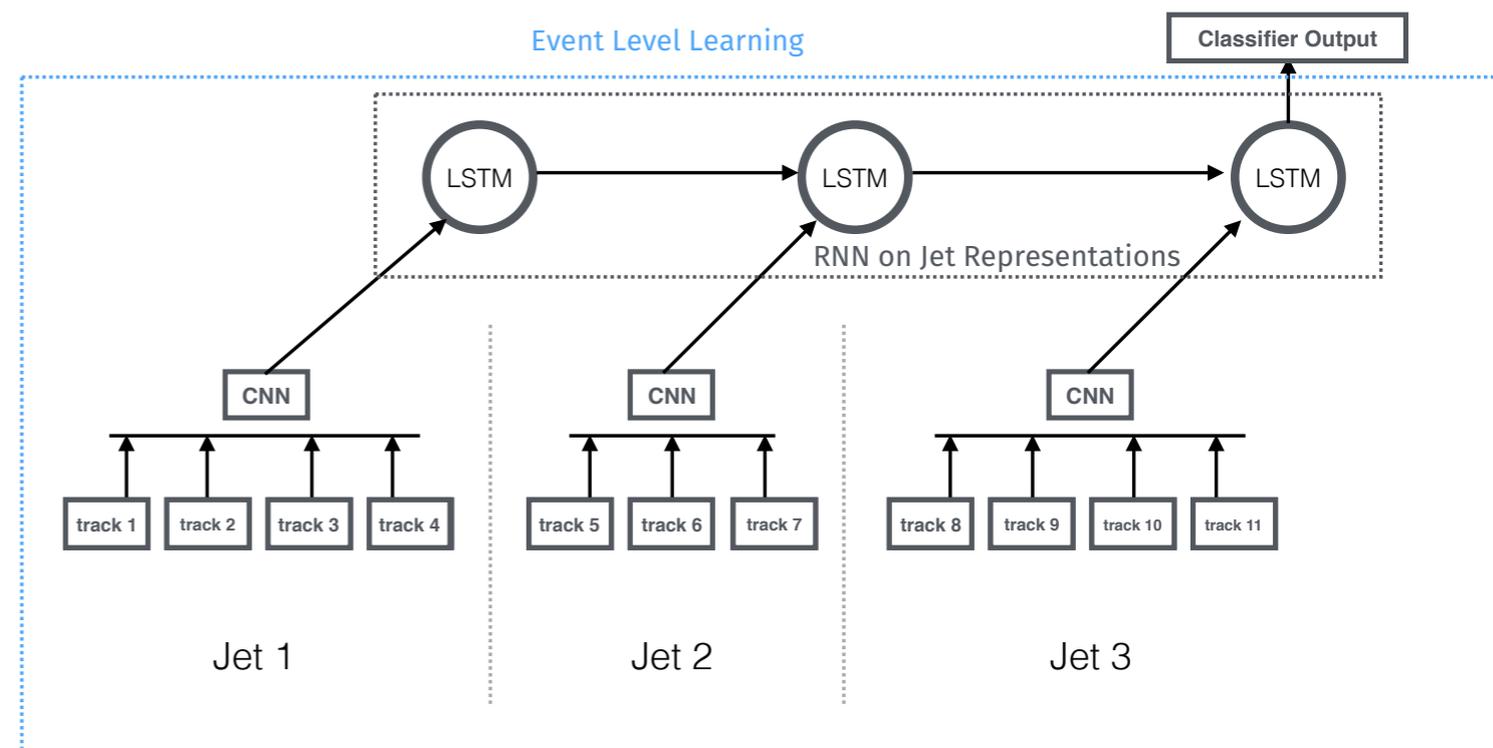
## Siamese Architecture



- Identify misclassified events
  - additional training on sample of such events.
  - add new classes to better separate them.
- Create classes based on final state content and topology.
  - Need to come up with some classification based on truth.

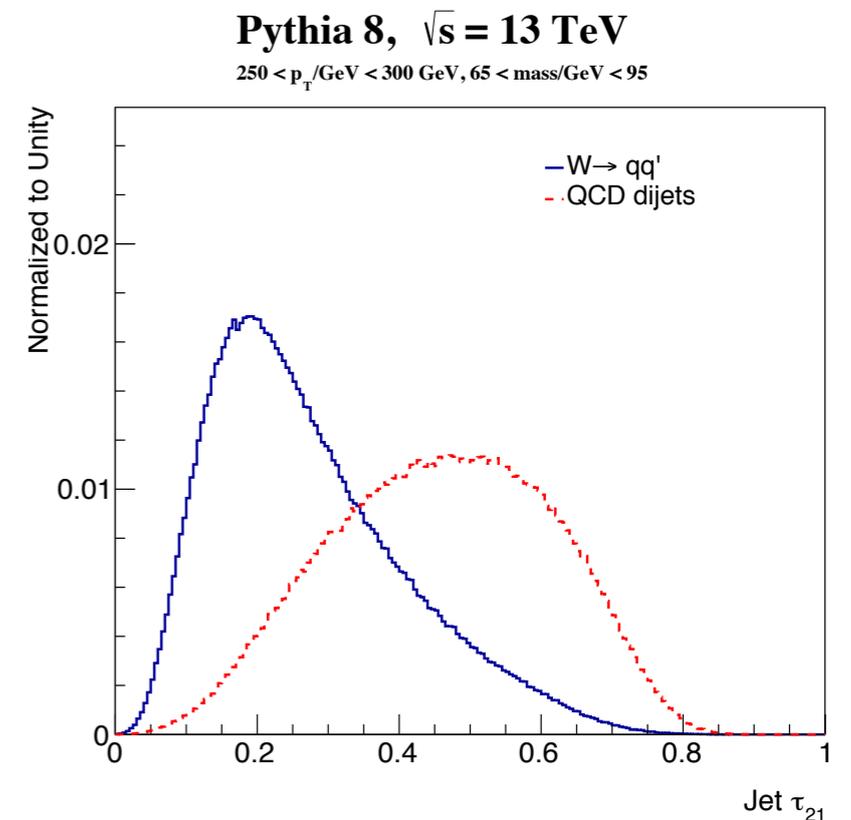
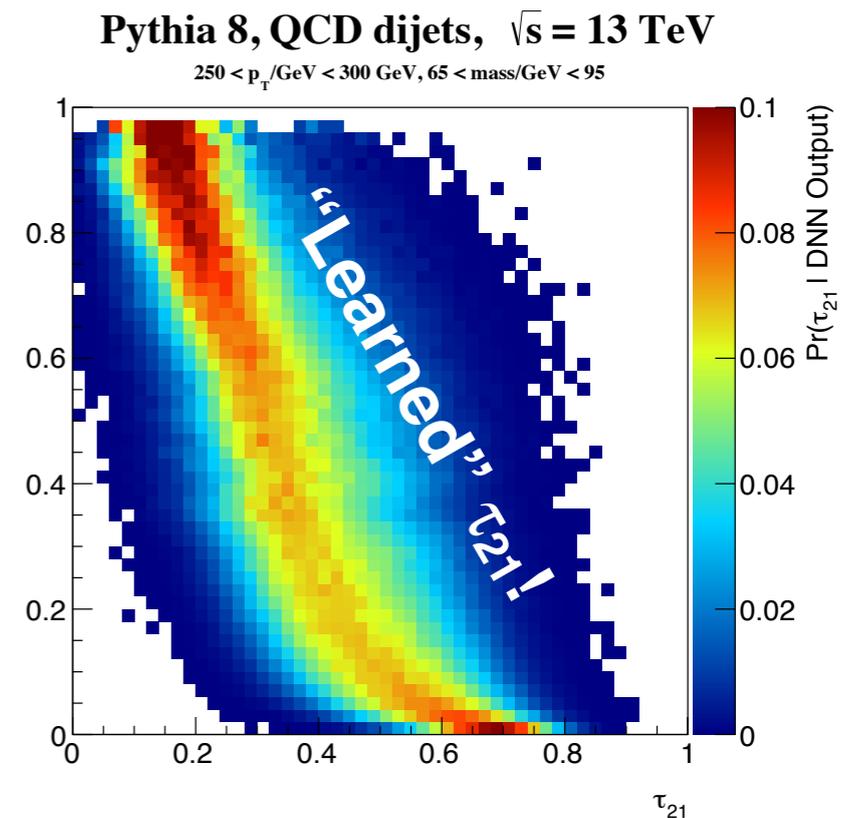
# Other Ideas

- Spatial transformer network module
- Unsupervised pre-training?
  - Better initialization of earlier layers
  - Better generalization...
- Reinforcement training using Hand Scans.
- Currently I'm classifying whole events.
  - Instead use Regional-CNNs to identify regions with individual particles in multi-particle events.
- Remove Cosmics.
- Feed to RNNs to classify full drift.



# Inside the Black Box

- Identify features
  - Find the training example/region that maximally activates each feature.
- Look for correlations between DNN output and known features
  - Did the DNN figure out this feature?
  - How important is this feature?
  - e.g. Studying e/gamma classification
    - vs electron stub length → spacial resolution/topology
    - vs direction → “seeing” in 3D? Disambiguation?
- Studying DNNs can reveal useful features which we don't know about.
  - SLAC group got insight into color-flow!



# Systematics

- Think of DNN Reco as Hand Scan... use same methods.

- Fancier approach:

<http://arxiv.org/abs/1506.02169>

- Parameterized Classifiers (Kyle Cranmer): Make systematic variations new classes and train to separate.

- Let  $\theta_0, \theta_1$  be 2 systematic variation  $s(x; \theta_0, \theta_1) = \frac{p(x|\theta_1)}{p(x|\theta_0) + p(x|\theta_1)}$

- Test Statistic:  $T(D; \theta_0, \theta_1) = \prod_e \frac{p(x_e|\theta_0)}{p(x_e|\theta_1)} = \prod_e \frac{p(s(x_e; \theta_0, \theta_1)|\theta_0)}{p(s(x_e; \theta_0, \theta_1)|\theta_1)}$

- More generally,  $c=\{\text{sig, bkg1, bkg2}\}$  :  $p(x|\theta) = \sum_c w_c(\theta) p_c(x)$

- need a classifier for each pairwise (c vs. c') comparison ( $n * (n-1)/2$  of them)

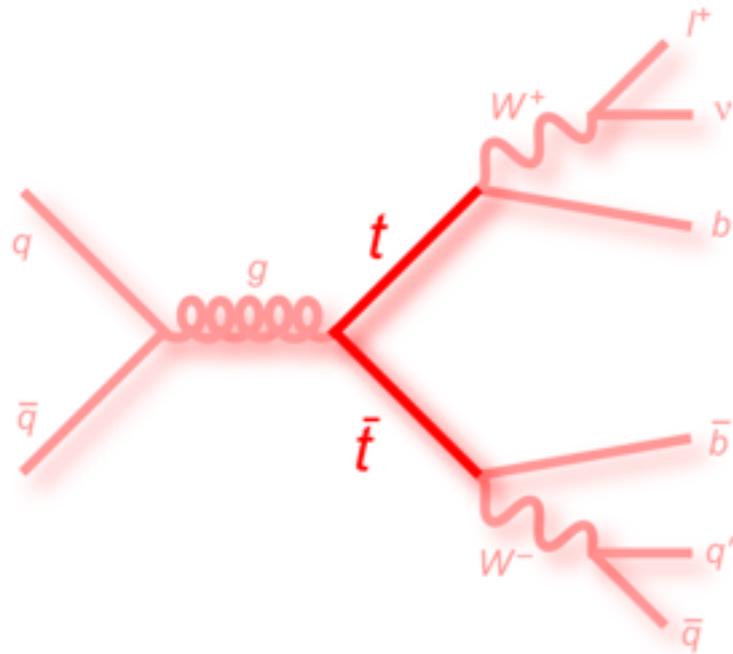
$$\frac{p(x|\theta_0)}{p(x|\theta_1)} = \frac{\sum_c w_c(\theta_0) p_c(x)}{\sum_{c'} w_{c'}(\theta_1) p_{c'}(x)} = \sum_c \left[ \sum_{c'} \frac{w_{c'}(\theta_1) p_{c'}(x)}{w_c(\theta_0) p_c(x)} \right]^{-1}$$

$$= \sum_c \left[ \sum_{c'} \frac{w_{c'}(\theta_1) p_{c'}(s_{c,c'})}{w_c(\theta_0) p_c(s_{c,c'})} \right]$$

# PARAMETRIZED CLASSIFIERS WITH DNN

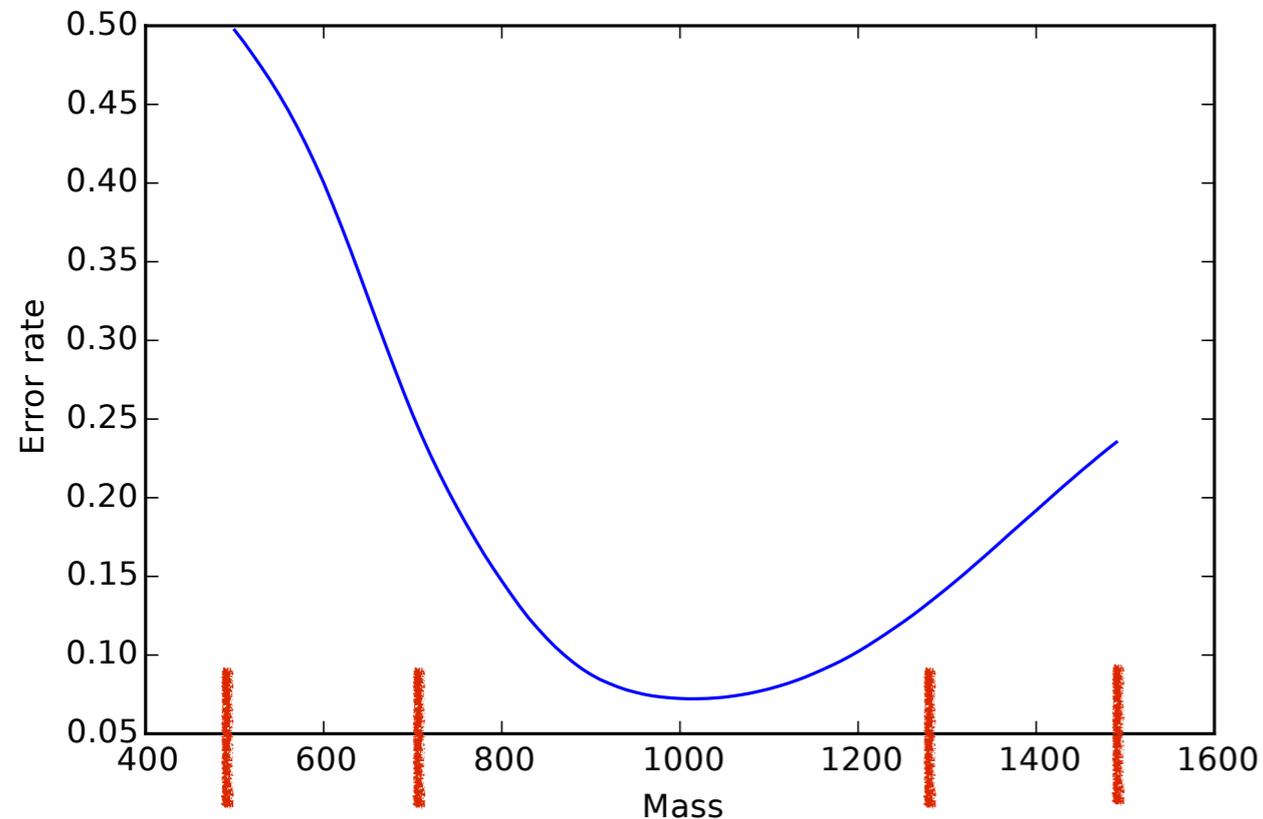
Example:  $Z' \rightarrow t\bar{t}$

together with:



Peter Sadowski , Daniel Whiteson, Pierre Baldi, Taylor Faucett

The networks were trained on 28 features: 22 low-level, 5 high-level, and the mass



Train at  $m_{Z'} = 500, 750, 1250, 1500$  GeV

Almost identical performance to dedicated training at  $m_{Z'} = 1000$  GeV

From K. Cranmer's [slides](#)

# DNN Reco Discussion

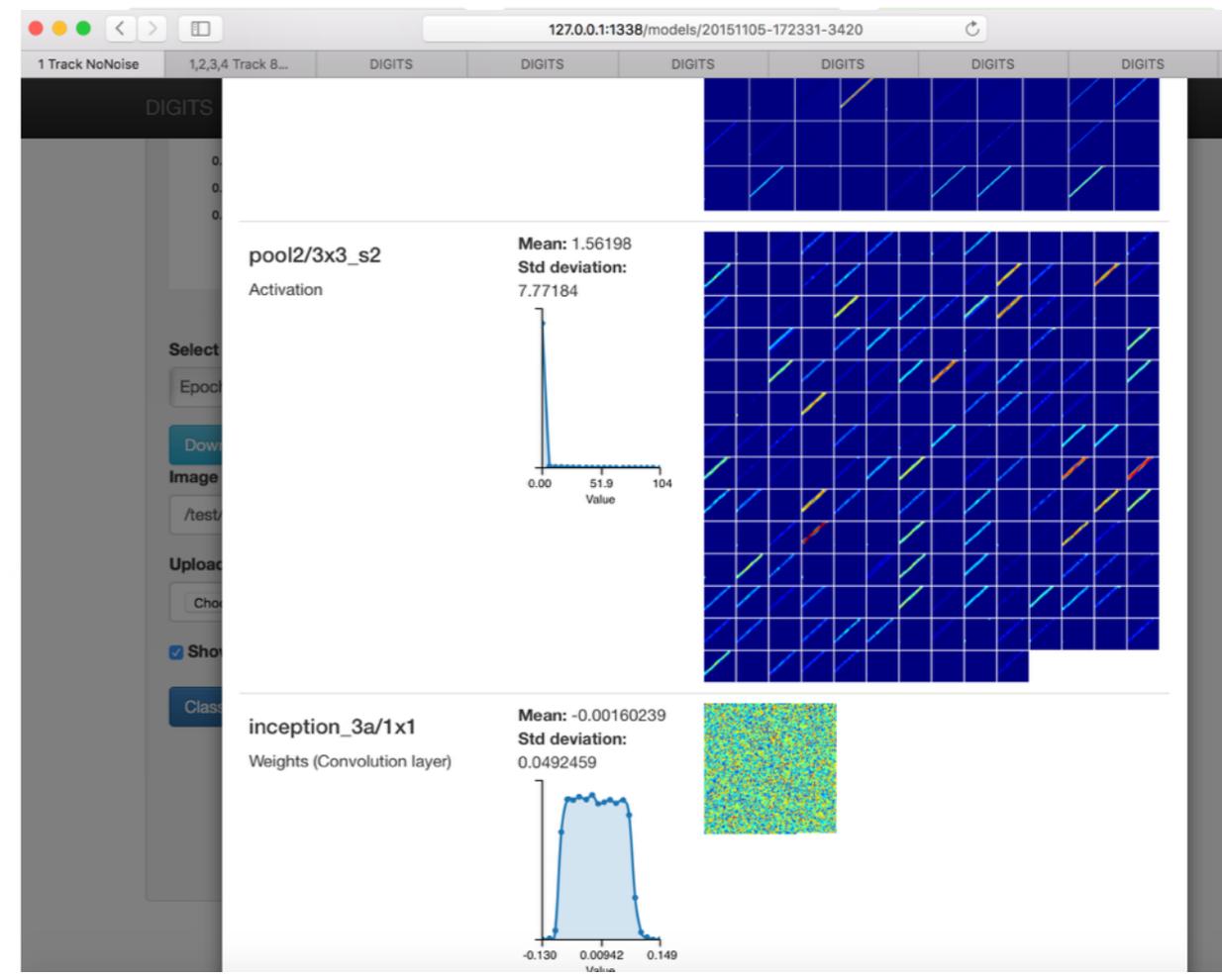
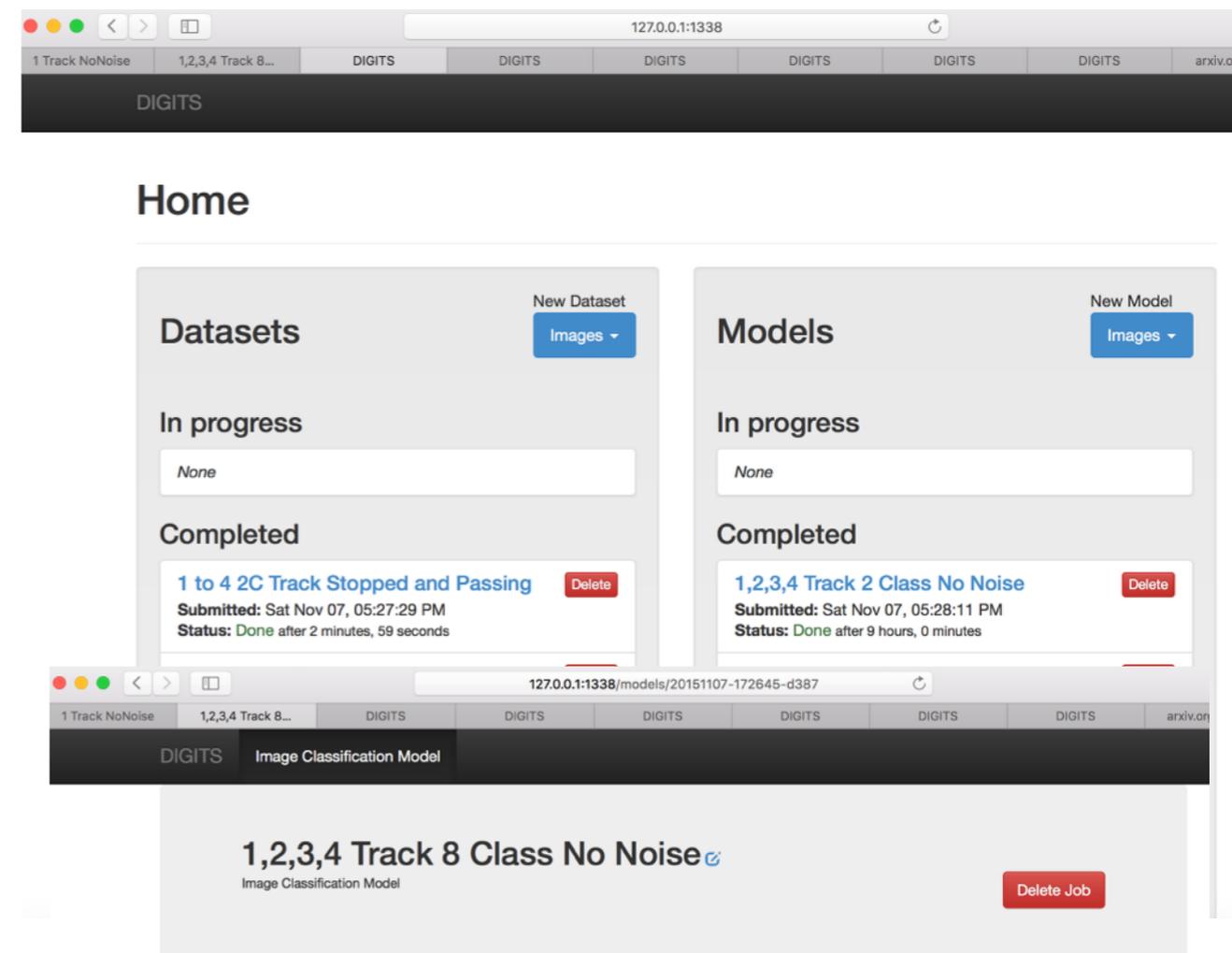
- DNNs replace algorithmic reconstruction?
  - Perhaps for Neutrino physics since classifying events and measure neutrino energy is sufficient.
- Really 4 different reconstruction options: Automatic Reco, Hand Scan, Assisted Reco, or DNN.
- Is it easier?
  - Optimal network and training requires careful craftsmanship and lots of “experiments”.
    - This work is analogous to writing algorithms and performing studies...
      - Arguably *less technically challenging*. No code necessary.
  - Systematics are still the same. Analysts will mostly still have to do all of the same work.
    - Assemble necessary signal and background samples. Make estimates.
    - Optimize cuts: Choose a point on the roc curve. (1-D)
    - Estimate systematics. New Systematics due the DNN?

# DNN Reco Application

- LArIAT is ideally sized for first studies of DL on raw LArTPC Data.
  - Just starting... large idea phase-space to explore.
  - Great exercise would be to apply this this technique in parallel to a planned LArIAT measurement.
    - Work out the issues: e.g. systematics.
- Filtering DUNE 35 ton events could be very useful.
- My next goal is to apply technique to assist in DUNE Detector Optimization.
  - Studies using Full Simulation require high performing Automatic Reconstruction, tuned to each geometry.
  - Backup: DNN-based Reco may provide a high performing metric easily tuned for geometry.

# DIGITS

- Great way to play.
- Essentially a web server interface to a batch system. Multi-GPU support.
- Only image classification (for now?).
- Great potential of evolving to more a general tool that will also make DNN accessible to everyone.
  - A graphical model editor would be awesome.



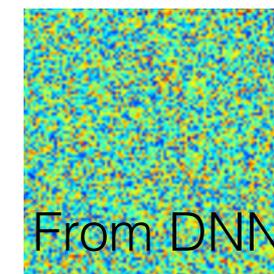
# DAG-Based Frameworks

# HEP SW Wish List

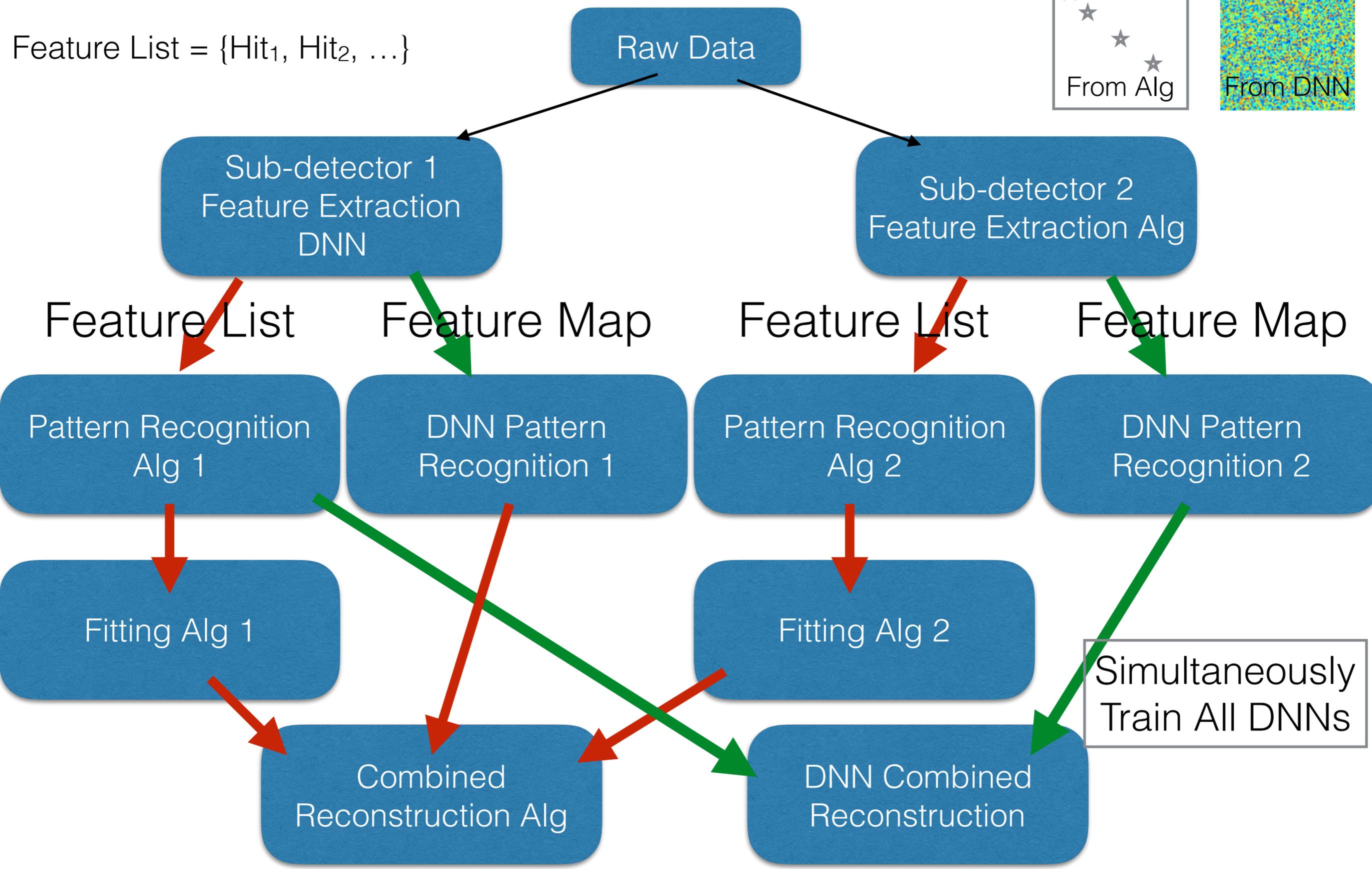
- Reconstruction closely integrating:
  - Traditional Algorithms like ones in HEP SW today.
  - Deep Neural Networks (and other ML Techniques)
    - Automatic training/monitoring (e.g. for reproducing training in every release)
    - NN visualization (structure and weights), Hyper-parameter scans.
  - Image processing algorithms
  - Event Display / Hand Scan (e.g. for re-enforcement training)
- Data structures optimized for architecture and computation, with automatic data transformations.
- Algorithms can process many events at once and one event in parallel threads.
- Automatically optimized for all/any CPU or GPU architectures. Future proof.
- Allow physicists to focus on the method and performance not implementation.
  - Easier to hand off problem to professional programmers.

# Weaving-in DNN Reco

Feature Map =



Feature List = {Hit<sub>1</sub>, Hit<sub>2</sub>, ...}

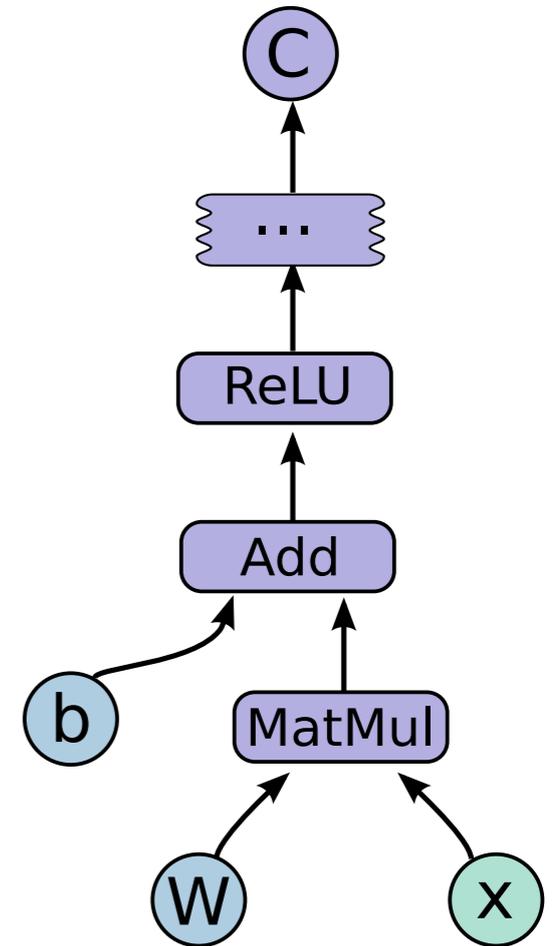


# Looking Ahead...

- Concurrency (simultaneously processing many events) is a hot topic. 2 types
  - *Task Parallel*: Many threads, each processing one event.
  - *Data Parallel*: Algorithms processing many events at once.
- The LHC experiments are confronting this issue. Current focus on Task Parallelism:
  - CMS already has multi-threaded ART.
  - ATLAS using plans to build on Gaudi-Hive for Run 3.
  - There are schemes to push some algs to co-processors... but not ideal.
- Experiments will have lifetime of decades (e.g. 30 years for DUNE). We need to insulate ourselves from architecture transitions.
- *My opinion*: We need new frameworks on the time-scale of HL-LHC, DUNE, ILC.
  - R&D Now. Framework in early 2020s. Reimplement software chain by 2025.

# DNN Software

- Basic steps
  - Prepare data
  - Build Model
  - Define Cost/Loss Function
  - Run training (most commonly Gradient Decent)
  - Assess performance.
  - Run lots of experiments...
- 2 Classes of DNN Software: (Both build everything at runtime)
  - Hep-Framework-Like: e.g. Torch, Caffe, ...
    - C++ Layers (i.e. Algorithms) steered/configured via interpreted script:
  - General Computation Frameworks: Theano and TensorFlow
    - Everything build by building mathematical expression for Model, Loss, Training from primitive ops on Tensors
    - Symbolic derivatives for the Gradient Decent
    - Builds Directed Acyclic Graph of the computation, performs optimizations
    - Theano-based High-level tools make this look like HEP Frameworks (e.g. pylearn2, Lasagna, Keras, ...)



# DNN Software

- Data is typically represented as numpy-like Tensors (N dim arrays).
  - Easily change between CPU and GPU implementations of tensor.
  - Usually automatically transferred between CPU/GPU memory.
  - Usually persisted in hdf5... sometimes stored in databases.
    - Supposedly very optimized.
- GPUs?
  - Most eventually call the same libraries (e.g. cuCNN) for optimized implementation.
  - Most have some library of algorithms with both CPU and GPU implementations.
  - Theano generates code for kernels and uses kernel libraries when appropriate.
  - Almost all on NVidia/CUDA... seeing OpenCL now.
    - I've seen significantly better performance from AMD over NVidia for some kernels.

# DAG Framework?

- Today with Theano
  - A Physicists can write down math expression for their computation or algorithm. Theano auto optimizes...
    - no computing expertise necessary.
  - Can pass expression to professionals who tune optimization/code generation in Theano
    - No physics understanding necessary.
  - Code generation can be optimized for each architecture.
- Naively, we should consider completely different approach to writing software:
  - High level description of algorithms/data by physicists (new language?)
    - The representation of the data and the implementation of the computation is changeable.
  - Automatic analysis of the computation graph and targeted code generation, developed/optimized by experts

# TensorFlow

- Google Deep Learning tool, many similarities to Theano, recently open sourced. C++ and python API.
  - Computation is done by building a DAG in a *Session*. Performs DAG optimizations.
  - Library of *operations* which wrap CPU and GPU *kernels*
    - Extendable.
    - Ops can provide gradient implementation.
    - Doesn't generate code like Theano.
    - Provides control flow operations which allows implementing loops.
  - *Variables* provide mutable data.
  - *Containers* allow sharing mutable data between disjoint computation graphs.
  - *Queues* automatically provide asynchronous computation when possible.
  - Designed for heterogeneous and distributed computing.
    - Point is that the same code generate same computation on any system.
    - From mobile phones, to multi-gpu systems, to heterogeneous distributed clusters.
    - Optimizes kernel placement using cost models, simulations, measurements of performance of kernels on devices and data transfer times.
    - Has Fault Tolerance.
  - *TensorBoard* provides graph and data visualization.

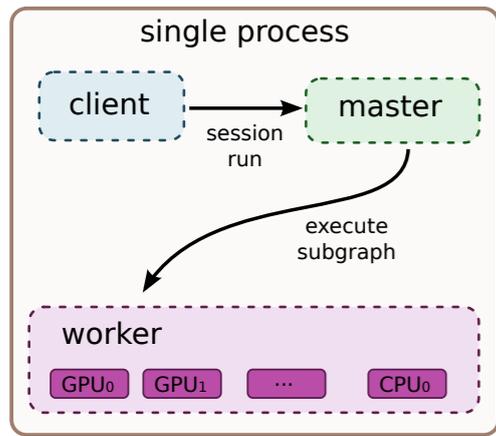


Figure 3: Single machine and distributed system structure

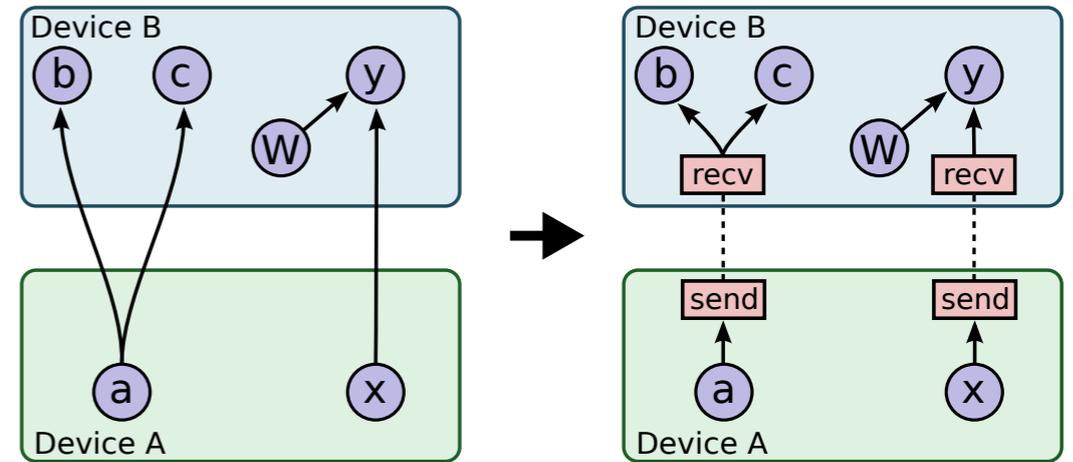
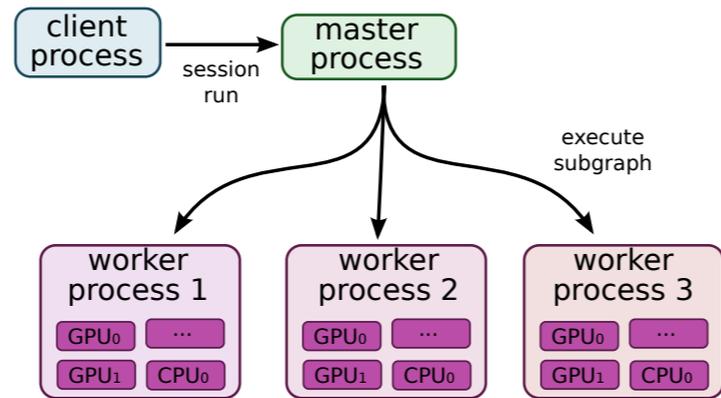


Figure 4: Before & after insertion of Send/Receive nodes

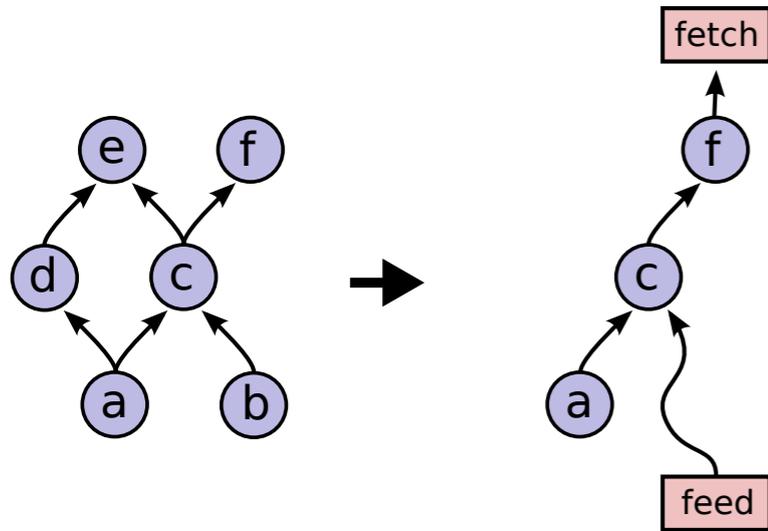


Figure 6: Before and after graph transformation for partial execution

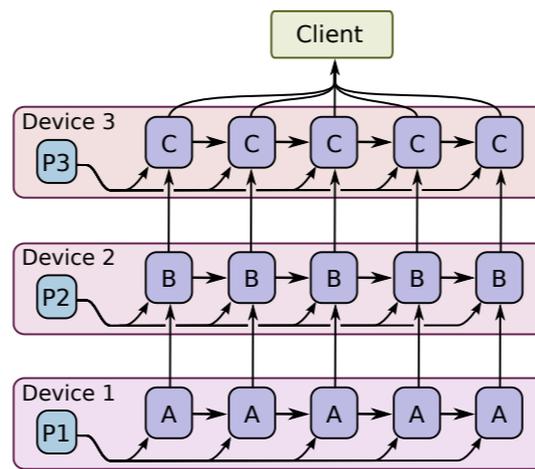


Figure 8: Model parallel training

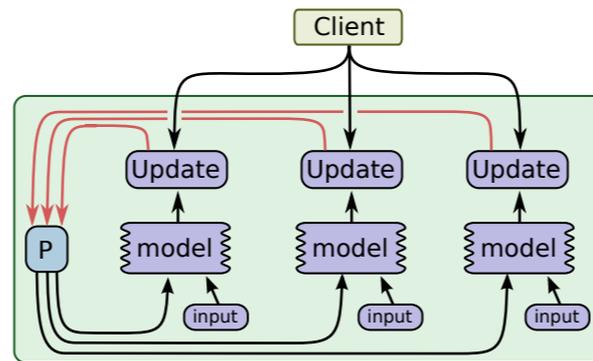
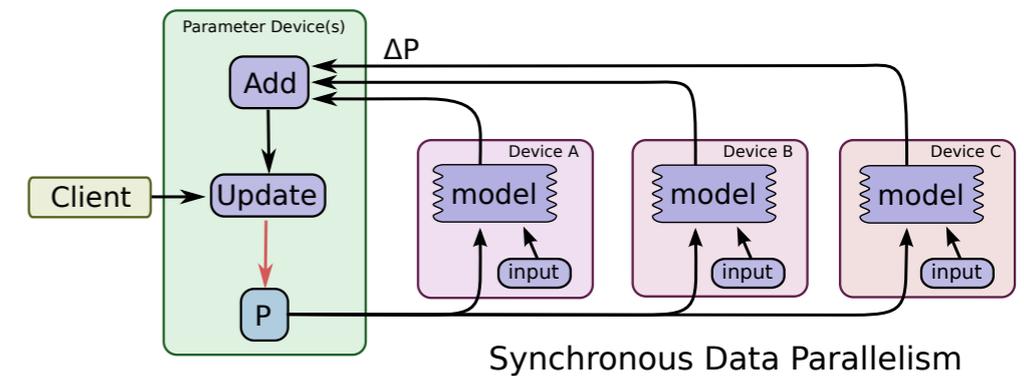
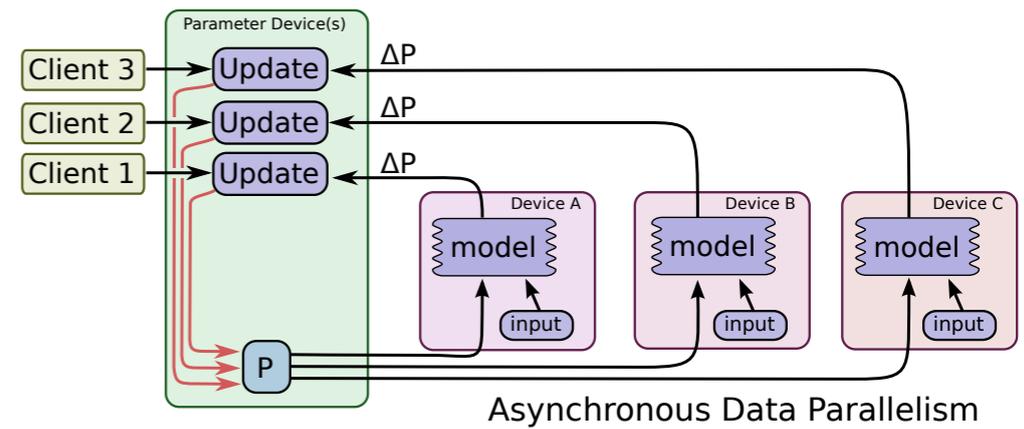


Figure 9: Concurrent steps



Synchronous Data Parallelism



Asynchronous Data Parallelism

Figure 7: Synchronous and asynchronous data parallel training

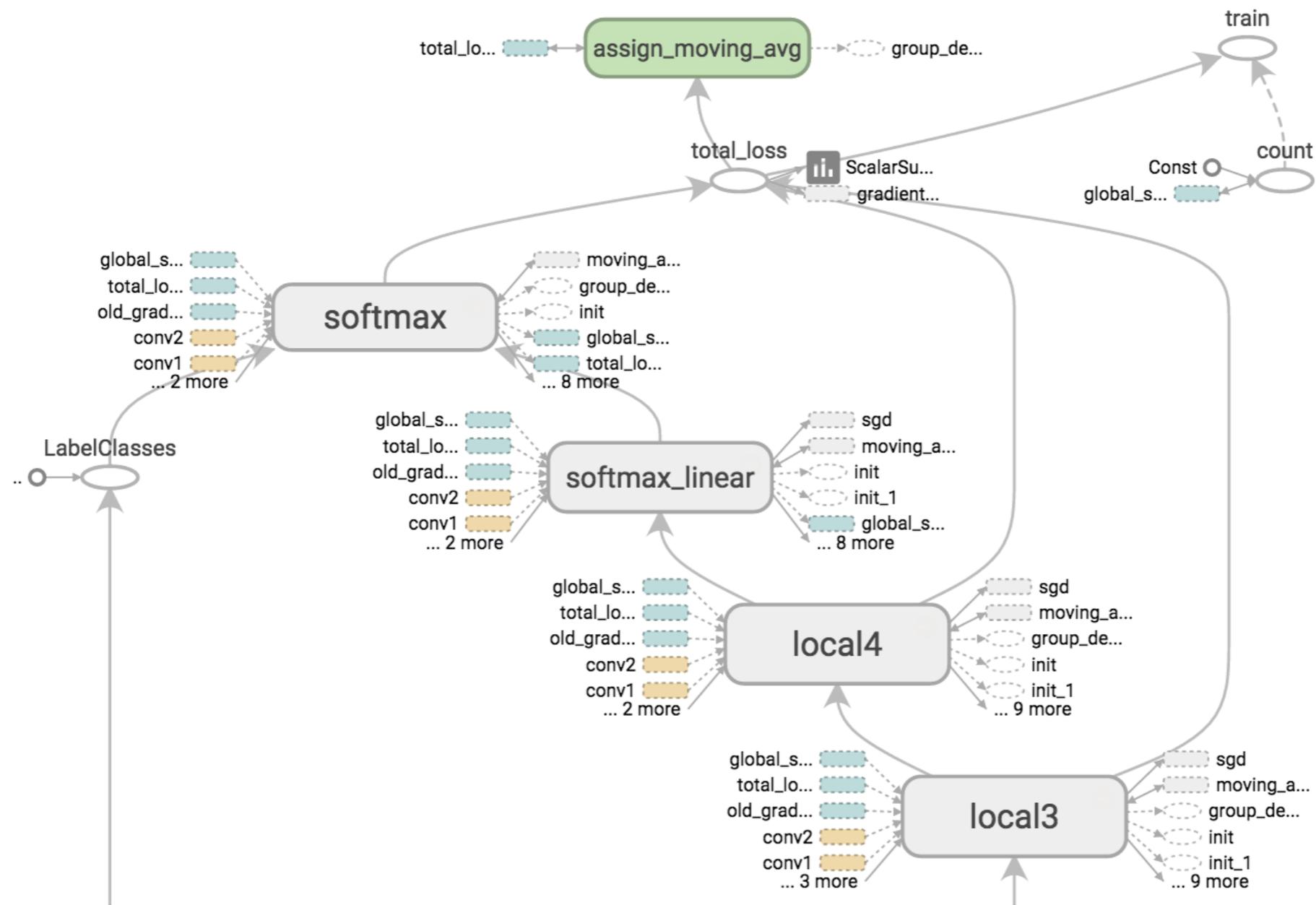


Figure 10: TensorBoard graph visualization of a convolutional neural network model

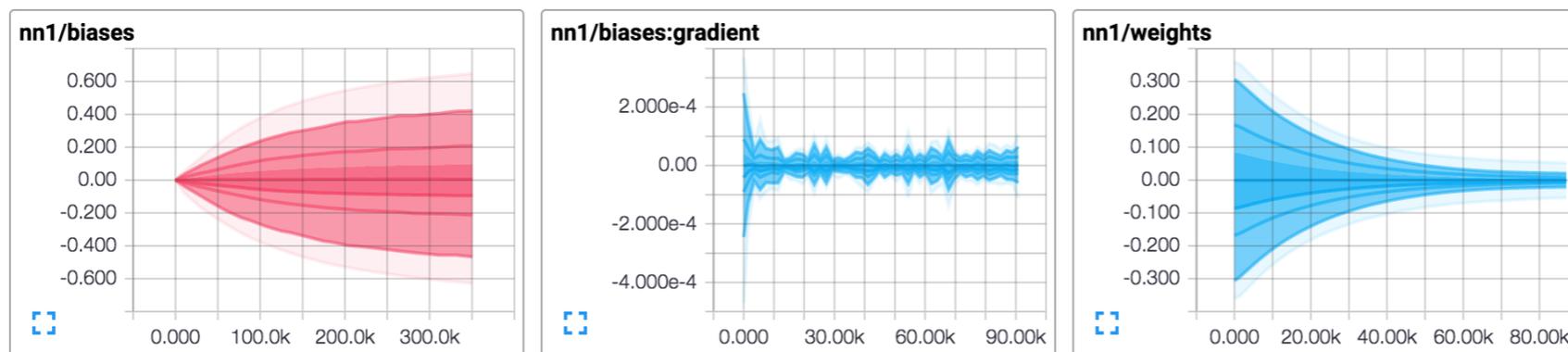


Figure 11: TensorBoard graphical display of model summary statistics time series data

# DAG/Tensor HEP Framework?

- I can imagine building Reconstruction, DAQ, or Trigger based on this system.
- Reconstruction is extremely complicated...
  - Some steps (e.g. FFT deconvolution) could be implemented in a few lines using existing ops.
  - Complex algorithms can be written into Ops/Kernels.
  - These can be wrapped by an abstraction analogous to ART Modules/Gaudi Algorithms.
    - Similar to high level DNN SW build on top of Theano.
- Our Data is complicated
  - Representation as tensors is efficient for vectorization, GPUs, and easily segmenting data.
  - High Level Objects can wrap these objects.
  - Not sure Event Store is necessary in Data Flow approach... could be faked (e.g. simple python dictionary)
    - Data isn't fetched at run-time.
- Not sure how to handle Conditions Database, Geometry, etc...
- I think implementing WireCell in TensorFlow or Theano is perfect R&D for such an idea.

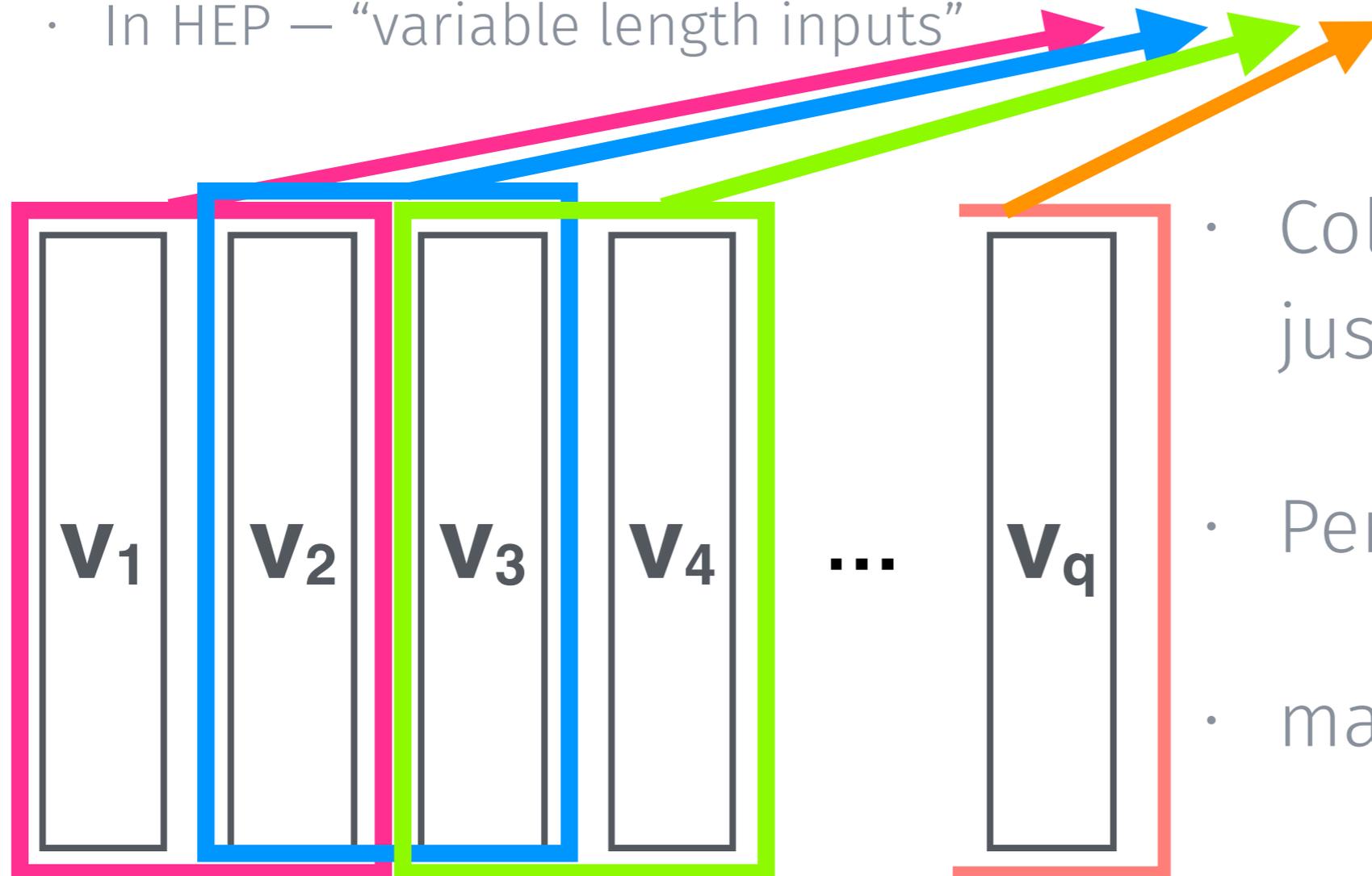
# Final Thoughts

- LArTPC is ideally suited for Deep Learning-based Reconstruction
  - LArTPC can serve as a stepping stone toward much more difficult problems, e.g. HL-LHC.
- Deep Learning may allow us to replace feature engineering (i.e. writing reconstruction algorithms) with network design/training.
  - No code... just configuration.
  - Closer to the physics, further from the implementation.
- I believe future frameworks will have many features already in DNN SW like TensorFlow
  - CMS-SW, Gaudi-Hive, WireCell 2.0, ... all use data-flow programming with schedulers
  - Minimally, we should learn from the DL systems.
- Building our frameworks on top of a broadly used system is highly desirable.
  - Potential to outsource the low level details and optimization to Google, community, and non HEP-professionals.
  - Allow physicists to focus on physics. Easy to try ideas. Let “professionals” worry about implementation.
  - Provide access to broader range of architectures and resources... more future-proof.

Extra Slides

# CNNs — a more general usage

- Not only for images! Can use them with sequential data that has an ordering
  - In HEP — “variable length inputs”

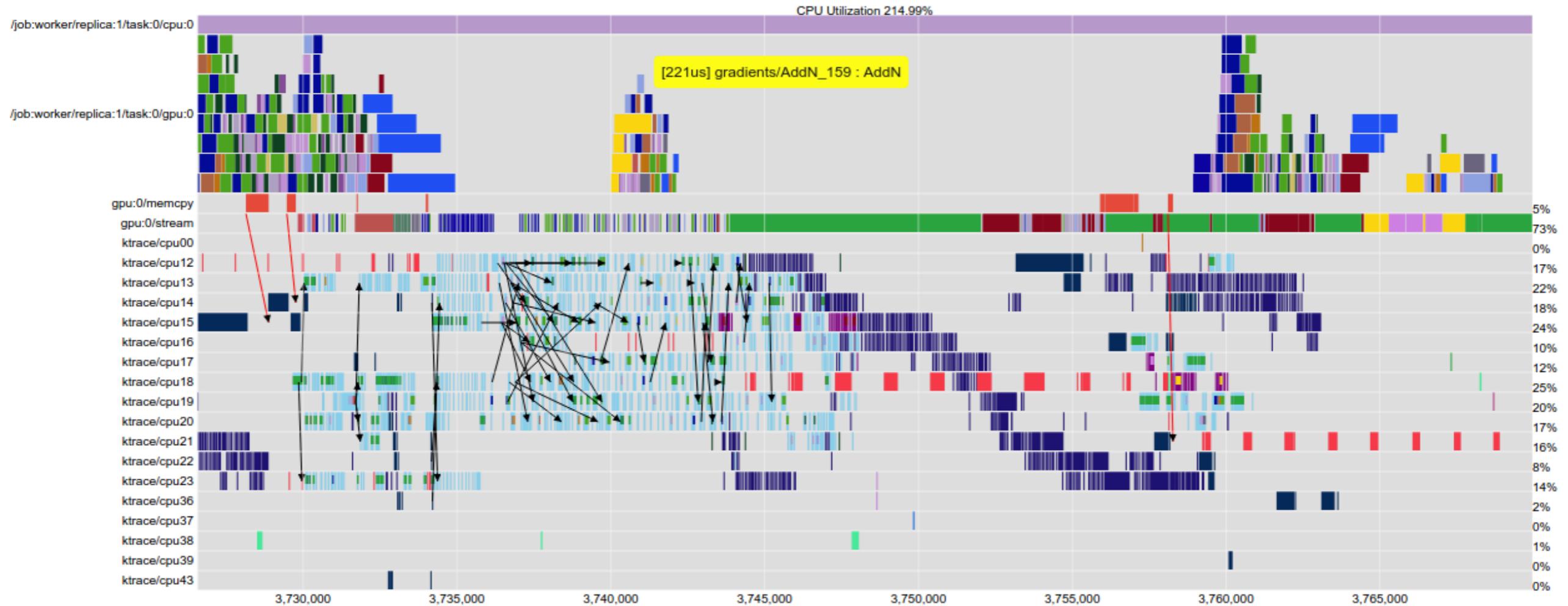


- Color boxes are just sums!

- Perform:

- $\max(\text{pink}, \text{blue}, \text{green}, \dots, \text{red})$

# Latency vs Throughput



It may be that TensorFlow scheduler is optimized for low-latency not high throughput.

# DAG/Tensor Based Systems

- TensorFlow paper points out that there are many other similar systems
  - None provide the level of support for heterogeneous and distributed systems.
  - Few have features that TensorFlow intends to implement.
    - For example automatically generating kernels that more efficient by combining operations.
- Clearly there is a some of commonality with Gaudi-Hive, WireCell 2.0, etc...
- The appeal of TensorFlow is that it will be supported by Google and open source community.
  - Potential to outsource the low level details and optimization to Google, community, and non HEP-professionals.
- If not the basis, these systems may provide inspiration for future framework.

# Math in Python

- **numpy**: Matrix manipulation like matlab
  - $C=A*B$  performs a computation on numbers in A and B matrixes
- **sympy**: Symbolic manipulation like mathematica
  - $C=A*B ; D=A^{-1} C \implies D=B$
- **Theano**:
  - Symbolic representation and operations (e.g. derivatives)
  - Based on Tensors with numpy-like functionality
  - Computation tree optimization
  - Transparently compiles into CPU, OpenMP, CUDA, and OpenCL.
    - Many missing/non-optimal features in GPU implementation
  - Provides a framework for implementing new operations, optimizations, and backends.
    - Provides an environment built for optimizing calculations on CPUs and GPUs.
- Why? instead of writing code to perform you calculation, use these systems to write down the mathematical expressions... and they will generate *optimized* code.

# Working with Theano

- Easy to switch from C to Python to Theano to Sympy, etc...
  - Just build your expression with python functions and feed the different objects for different versions.
    - consider: `def f(x): x*x`
      - python: `y=f(2) -> y=4` (regular python float)
      - sympy: `y=f(x) -> y = symbolic rep`
      - theano: `y=f(x) -> y = symbolic rep`
        - `compute_y=function([x],y)` optimizes/compiles
        - `compute_y(2) -> 4`
- Various ways to convert sympy -> theano:
  - `theano_function`: takes a sympy expressions and translates it into Theano expression.
  - `SymCFunc`: creates efficient c-code for scalar expression which Theano can wrap...
    - The c-function can be faster.
    - But then Theano can't optimize it.
- Parallelization: tensor representation... numpy broadcasting for scaling.
- Loops = contraction of indexes... makes reordering loops easy
- Iteration = shared variables (keep state) and update mechanism.

# MEM with Theano

- Matrix Element Method is in principle the most sensitive technique for searches, but has been prohibitively CPU intensive.
- Simple case (6 diagrams):  $u u \rightarrow 3$  gamma matrix element. One of the first examples of ME on GPU from 2010. (based on <http://arxiv.org/pdf/0908.4403.pdf>)
  - Note interesting LHC processes have  $O(100)$  diagrams.
- Focusing on ME evaluations only... no integration, change of vars, etc...
- I count  $\sim 500$  real and  $500$  complex numerical operations in the ME calculation ( $\sim 1500$  total).
  - c-code  $\rightarrow$  python  $\rightarrow$  Theano reduces operations to  $\sim 1000$ .
  - c-code  $\rightarrow$  python  $\rightarrow$  sympy  $\rightarrow$  Theano reduces the operations to 321 (116 on GPU, but I can't compile!)
  - 5.1x (4.6x) faster per event single thread computation time float (double).

# Theano

- Might be trivial to implement some algorithms with Theano.
  - Anything you can write as a formula can be easily expressed in Theano and automatically optimized.
  - Many things are already implemented.
  - For example, Kalman Filter (from: <http://matthewrocklin.com/blog/work/2013/04/05/SymPy-Theano-part-3/>)

```
from sympy import MatrixSymbol, latex
n      = 1000          # Number of variables in our system/current state
e
k      = 500          # Number of variables in the observation
mu     = MatrixSymbol('mu', n, 1)      # Mean of current state
Sigma  = MatrixSymbol('Sigma', n, n)   # Covariance of current state
H      = MatrixSymbol('H', k, n)      # A measurement operator on current state
R      = MatrixSymbol('R', k, k)      # Covariance of measurement noise
data   = MatrixSymbol('data', k, 1)    # Observed measurement data

newmu   = mu + Sigma*H.T * (R + H*Sigma*H.T).I * (H*mu - data)      # Updated mean
newSigma= Sigma - Sigma*H.T * (R + H*Sigma*H.T).I * H * Sigma      # Updated covariance

inputs = [mu, Sigma, H, R, data]
outputs = [newmu, newSigma]
dtypes = {inp: 'float64' for inp in inputs}

from sympy.printing.theanocode import theano_function
f = theano_function(inputs, outputs, dtypes=dtypes)
import numpy
ninputs = [numpy.random.rand(*i.shape).astype('float64') for i in inputs]
nmu, nSigma = f(*ninputs)
```