# RICH AI

Mukund Iyer, Nico Van den Hooff,
Rakesh Pandey, Shiva Jena

*MDS Mentor: Arman Seyed-Ahmadi*

# Agenda

1) Background and scope (Mukund)
2) Data (Mukund)
3) Machine learning models (Shiv)
4) Point clouds (Mukund)
5) Deep learning (Nico)
6) Model summary (Nico)
7) Data product (Shiv)

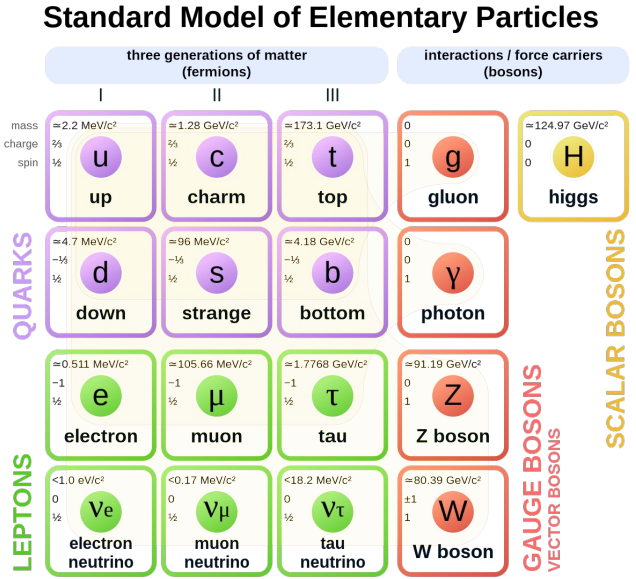
TRIUMF

# Background and scope

# The Standard Model describes the composition of matter

*Physicists' Objective: verify the "Standard Model"*

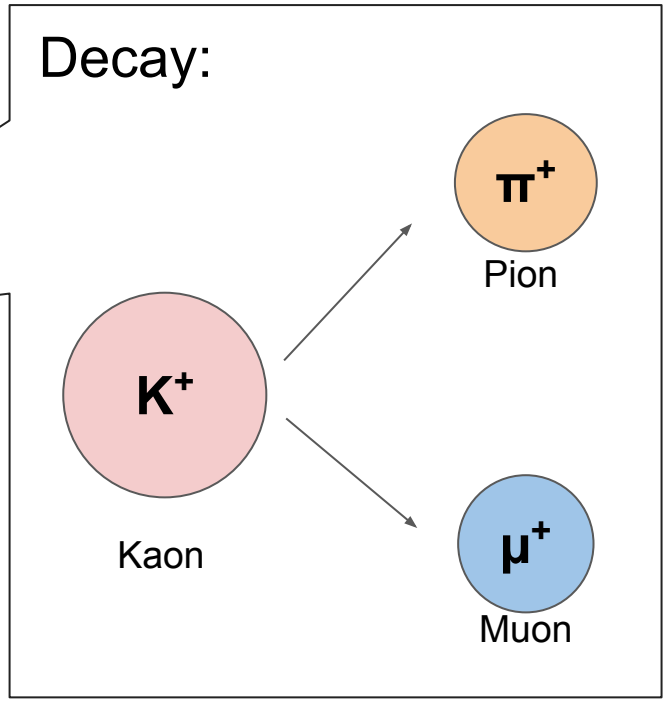
Source: https://exoplanets.nasa.gov/what-is-an-exoplanet/what-is-the-universe/



# The NA62 experiment at CERN is used to study pions

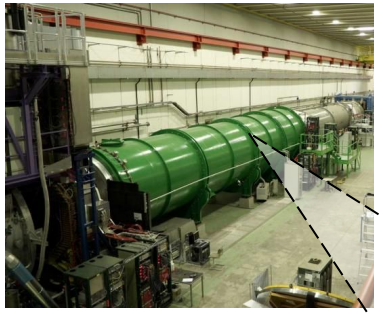
Source: https://home.cern/science/experiments/na62

Geneva, Switzerland



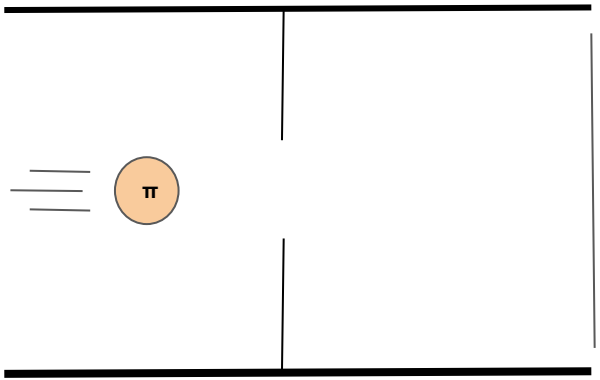

**Need to distinguish between these particles!**

# Need to analyze 2 sets of features to determine the subatomic particle

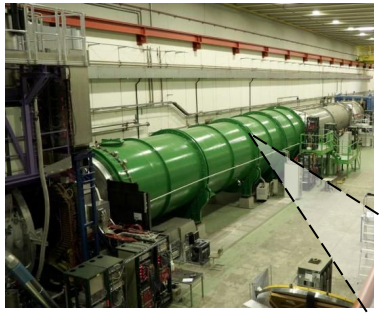


*cross sectional view

**1. Particle Motion**



# Need to analyze 2 sets of features to determine the subatomic particle

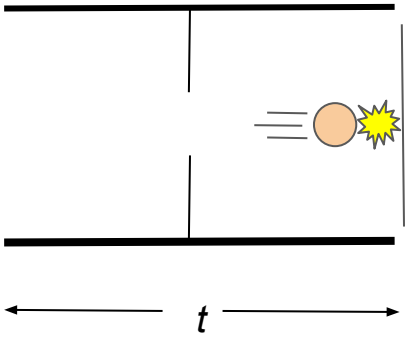


*cross sectional view

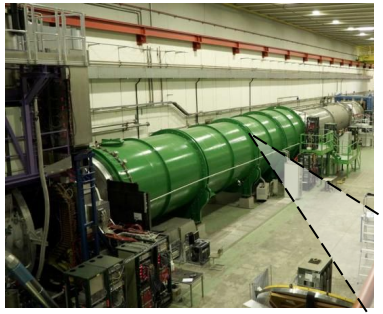
**Particle Momentum**

$\pi^+$

**Particle movement time (CHOD time)**

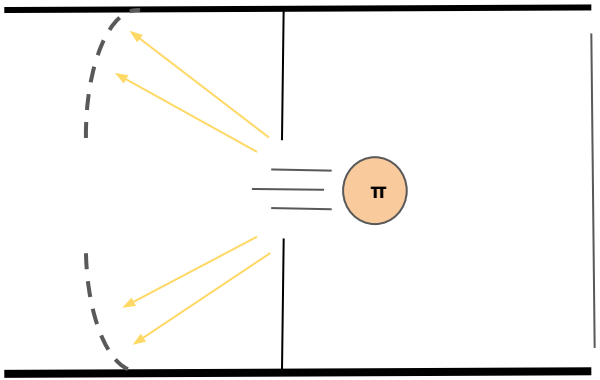


# Need to analyze 2 sets of features to determine the subatomic particle

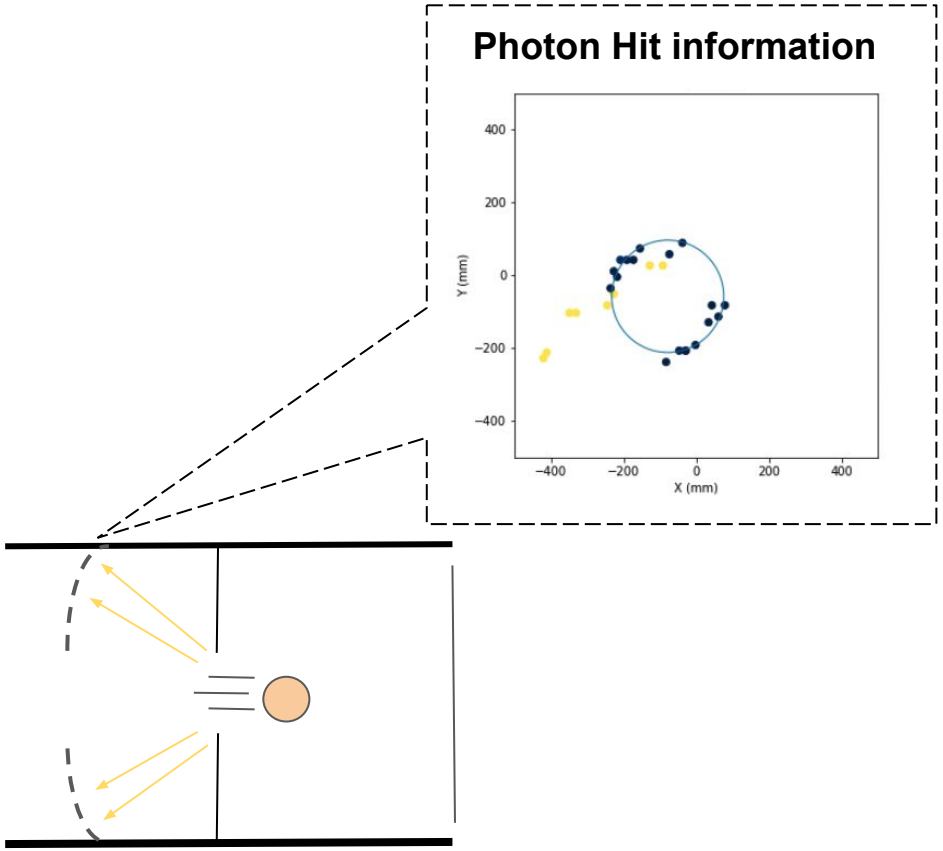


*cross sectional view

**2. Photon (light) hits**



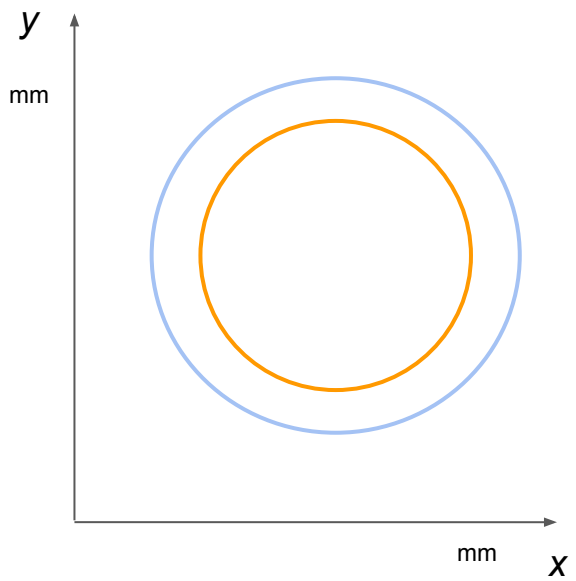
# Multiple features are extracted from the photon hits for each event




X, Y coordinates

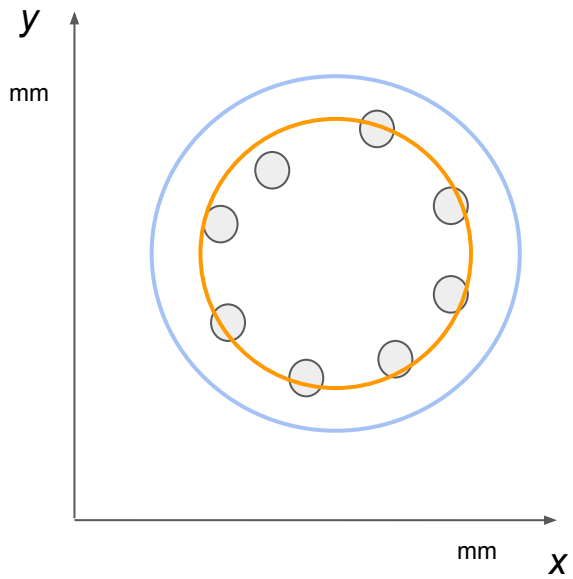
Hit times

# Can distinguish particle by ring size… binary classification!




For a given momentum,
the ring size for a
**pion** is **smaller**
than for a **muon!**

# TRIUMF use a MLE estimate to determine the particle




*P(pion) = 0.8*

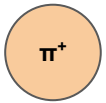
*P(muon) = 0.2*

This is a **pion**!

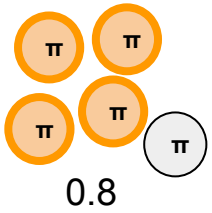
This method is purely analytical!

# Objective: surpass the current performance of the MLE algorithm

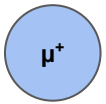
The two main metrics of interest are:



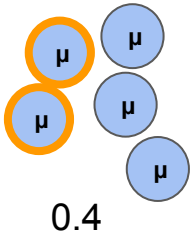

Pion Efficiency = $\frac{\text{\# Pions Identified}}{\text{\# Total pions}}$




**Pion Recall!**



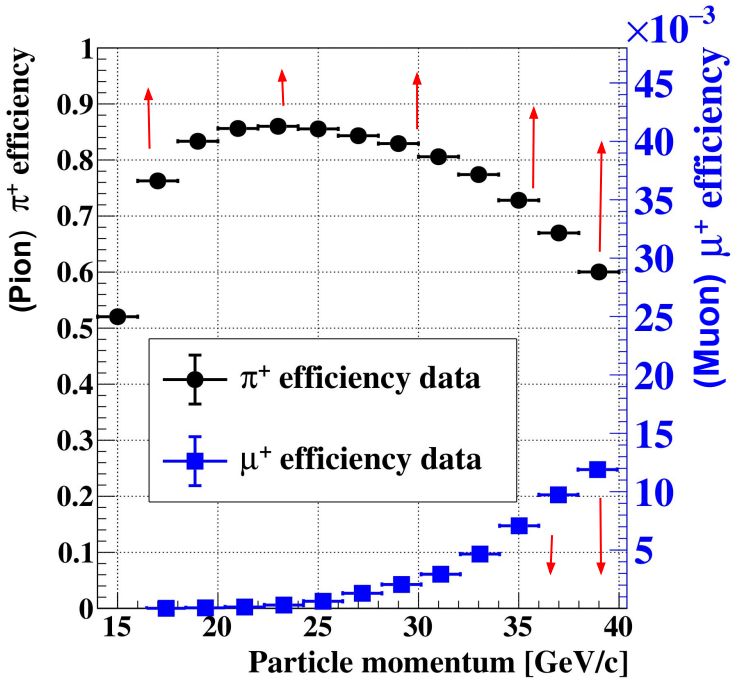

Muon Efficiency = $\frac{\text{\# Muons as pions}}{\text{\# Total Muons}}$




**1 - Muon Recall!**

# Need to surpass the current performance generated from the MLE

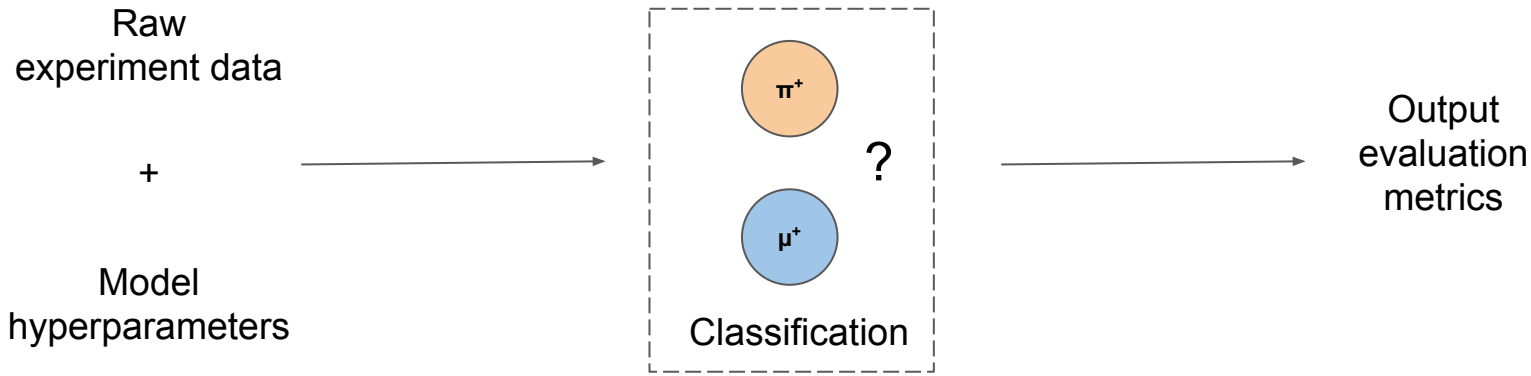
Pion Recall




1 - **Muon Recall**

Cortina Gil et. al, An investigation of the very rare $K^+ \to \pi^+ \nu\nu^-$ decay

# Objective: build a modularized ML pipeline for a binary classifier



TRIUMF

# Data

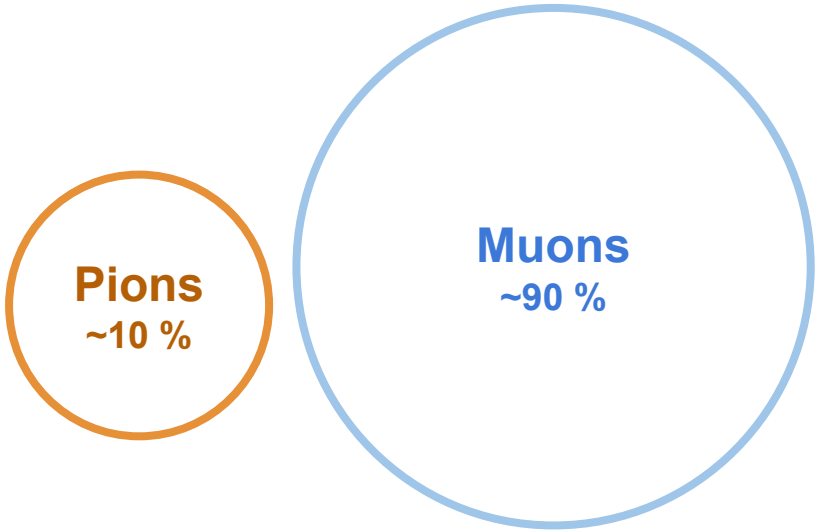
# Data on 11.5 million events with a class imbalance

**1 decay event = 1 example**

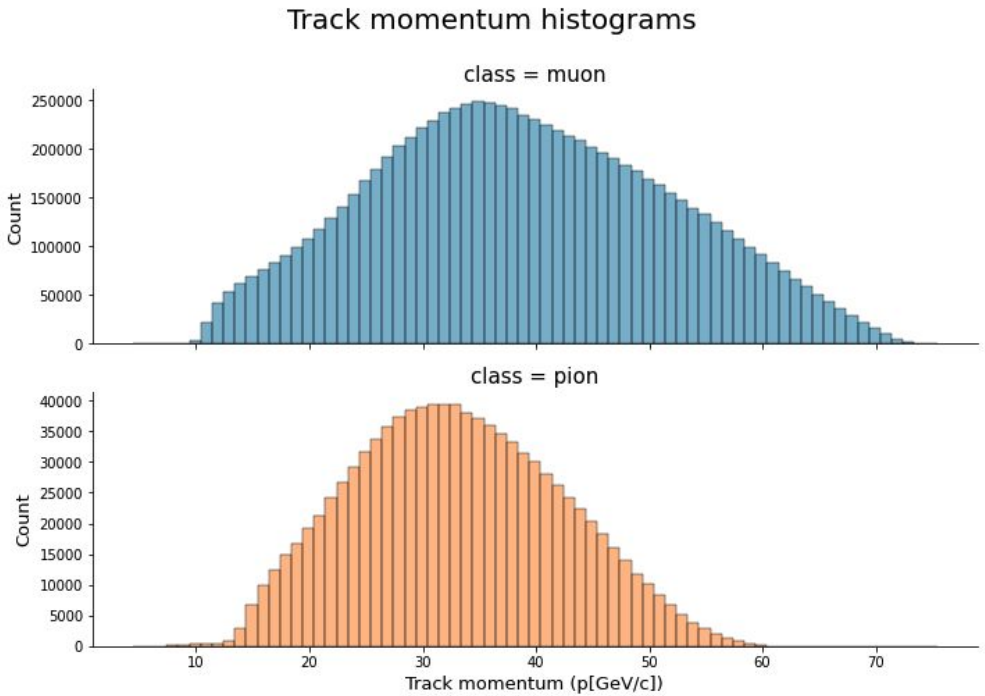
**Labelled data**

**~11.5 million examples**

**Class Imbalance**



# Systematic difference in the momentum distribution among particles




- Artifact of the experimental set-up

- Do not want ML models to pick up on this difference

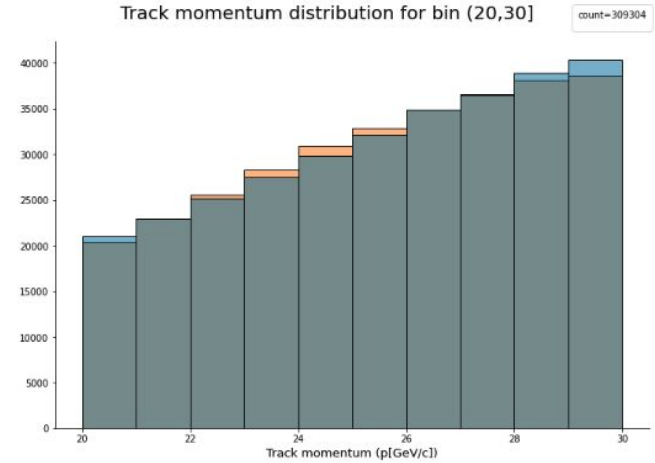
# Undersample using momentum bins!

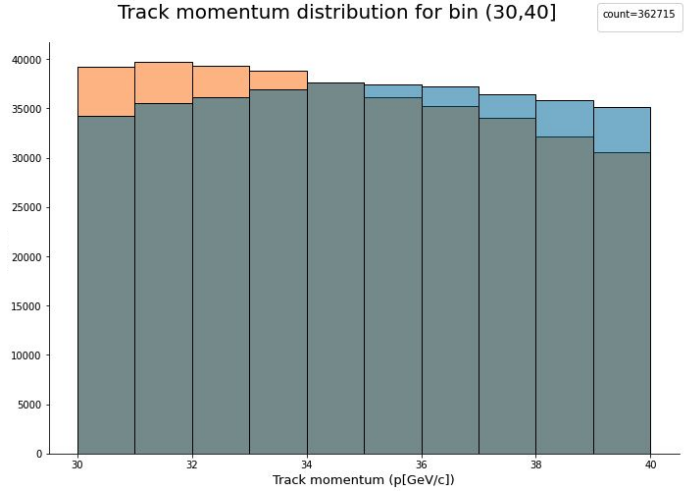
**Procedure:**

Slice data by bins of size $10\ \mathrm{GeV}/c^2$ → Count number of pions in bin → Sample equal number of muons

**~ 2 million samples**

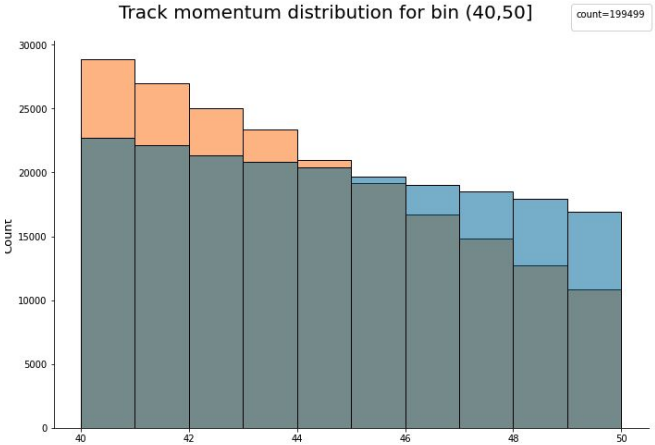
# Undersample using momentum bins!




Pion

Muon




- Distribution looks **even** across all bins
- Evidence that momentum is **controlled**

TRIUMF

*Baseline machine learning model*

XGBoost

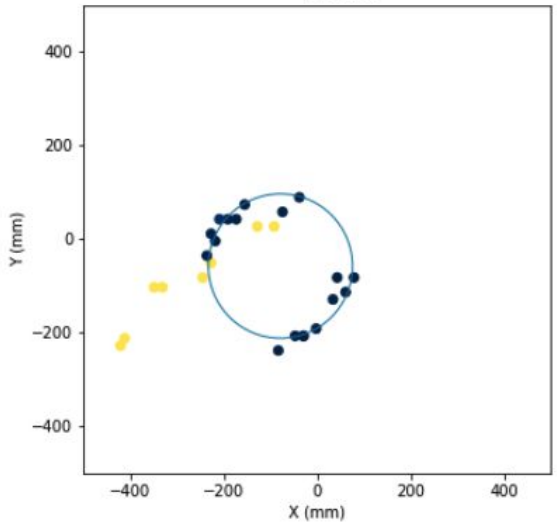
# Benchmarking with XGBoost as baseline: Features

***Features***

1. MLE radius
2. # Hits
3. Momentum

***Refresher:*** MLE radius and # Hits engineered from photon hit information

**Photon Hit information**



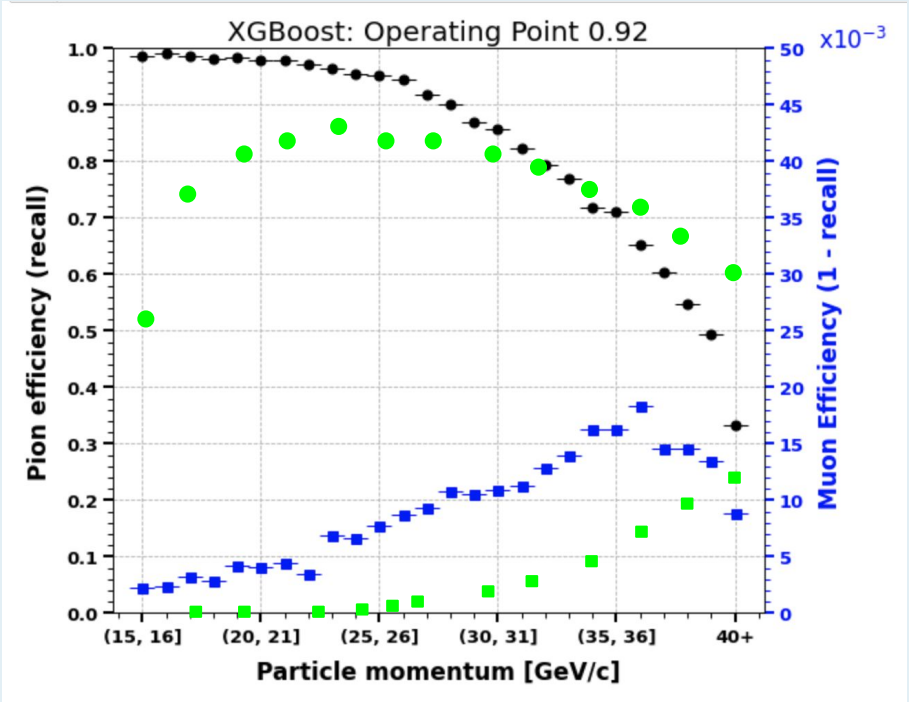

Ring Radius - MLE fit

- done by TRIUMF
- does not use X,Y coordinates

# Hits - noise reduction

# Benchmarking with XGBoost as baseline: Performance

***Model performance against TRIUMF's method***




***Hyperparameter optimisation ?***

- Results did not improve significantly.

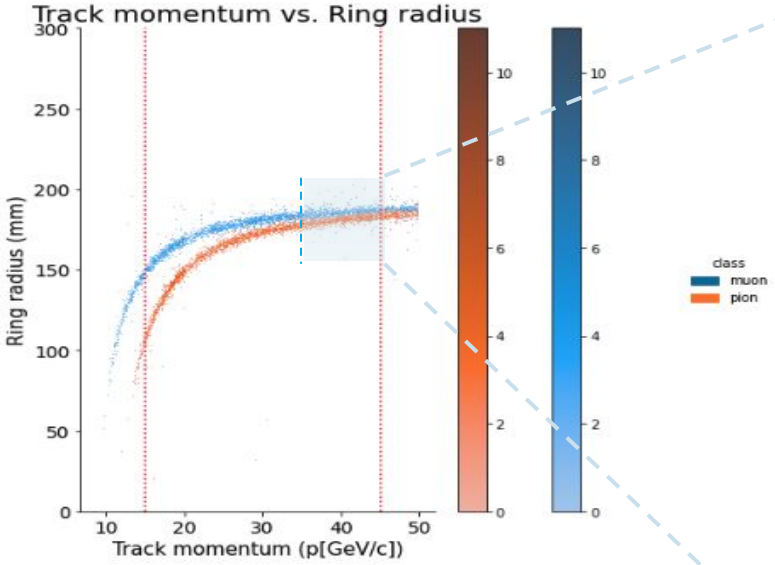
***Why more complex models?***

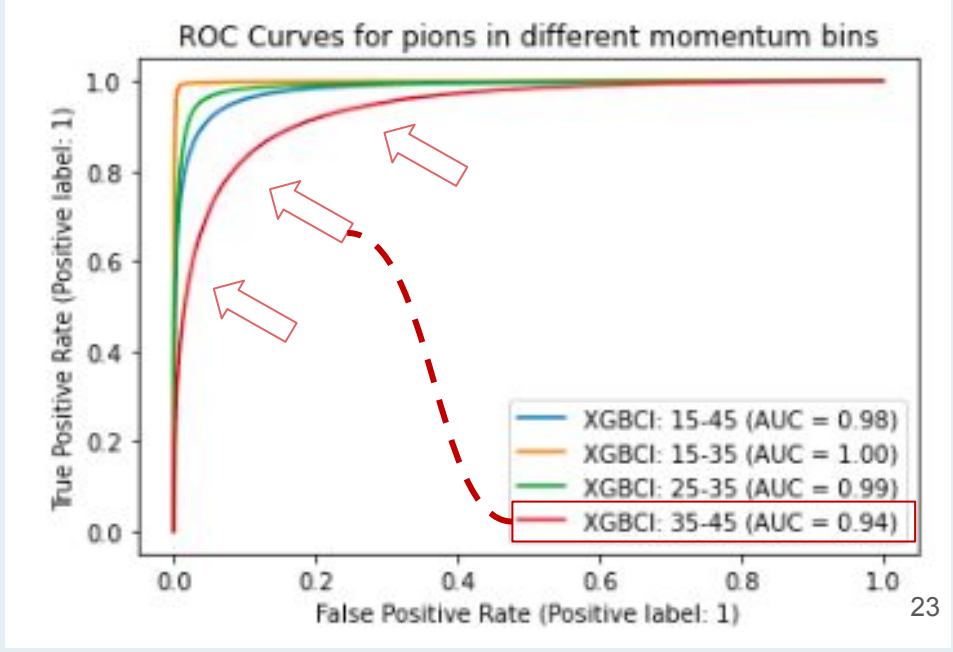
- Neural networks: more accurate in feature extraction (from photon hits data)

# Benchmarking with XGBoost as baseline: Precise target area

***Refresher:*** relationship between radius & momentum gets tricky in higher momentum

***Discovery of a precise avenue***
*Better performance possible in higher momentum bins*



TRIUMF

*Deep Learning*
Point clouds

# Point clouds are a set of points in space

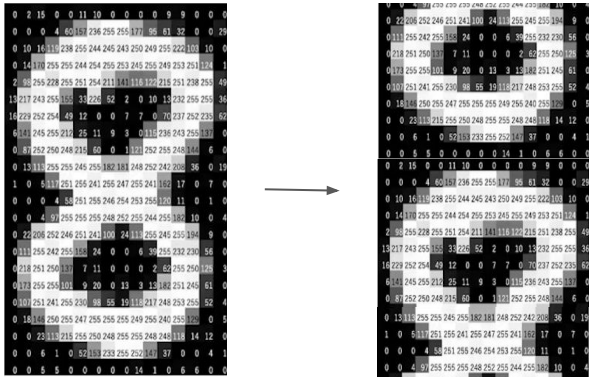
Source: https://github.com/nikitakaraevv/pointnet

Individual point contains **spatial** information

**Together** the points form an image or object

# Point clouds do not have order

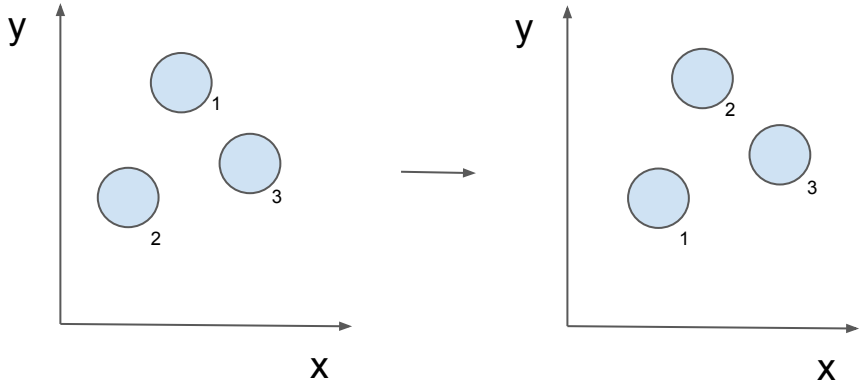
Image



Source: https://www.analyticsvidhya.com/blog/2021/03/grayscale-and-rgb-format-for-storing-images/

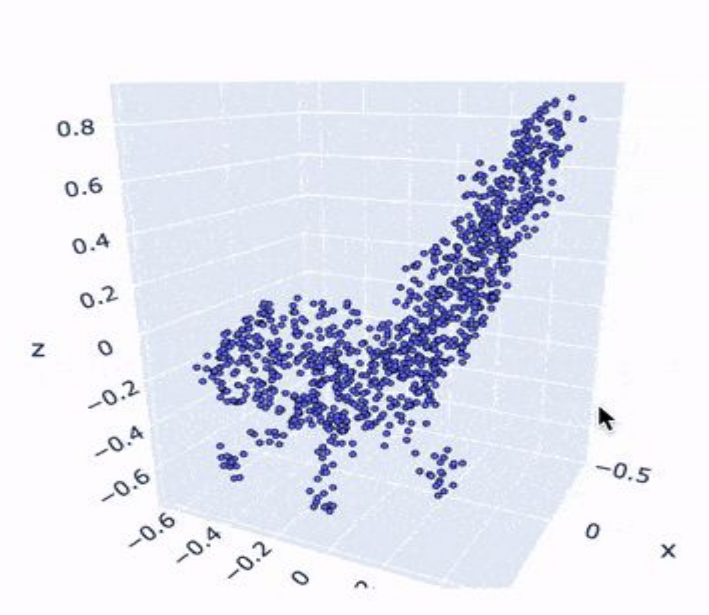
Changing the points changes the overall information

Point cloud




No change in information when changing points around

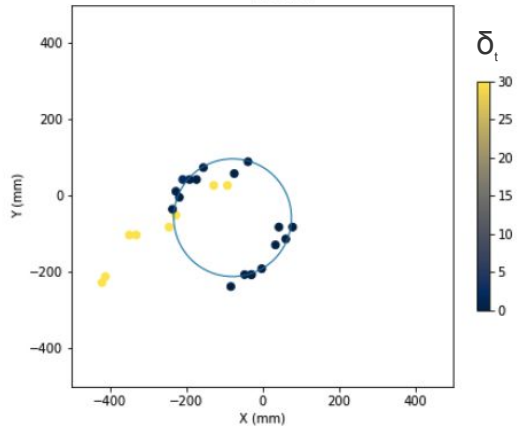
# Point clouds are invariant to geometric transformations

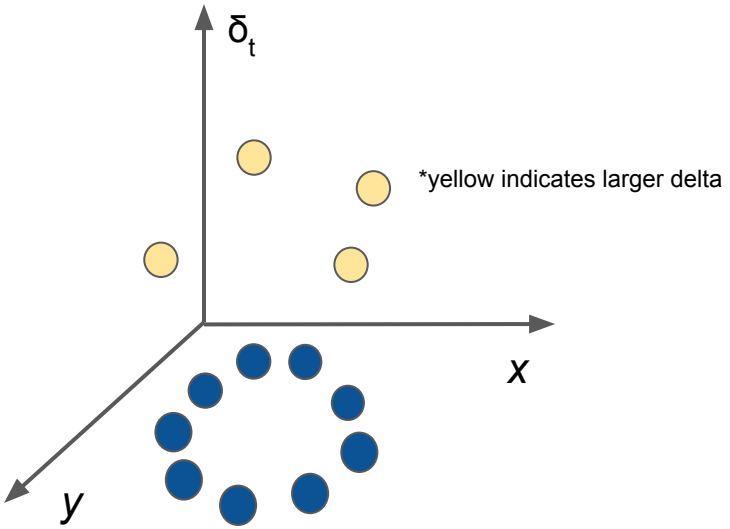



Source: https://github.com/nikitakaraevv/pointnet

The object is still the same regardless of how it is rotated(or shift axis)!

# Third dimension of time added to the hits data to generate Point clouds




$\delta_t$ = difference between particle travel time and time of the photon hits $\approx$ 0

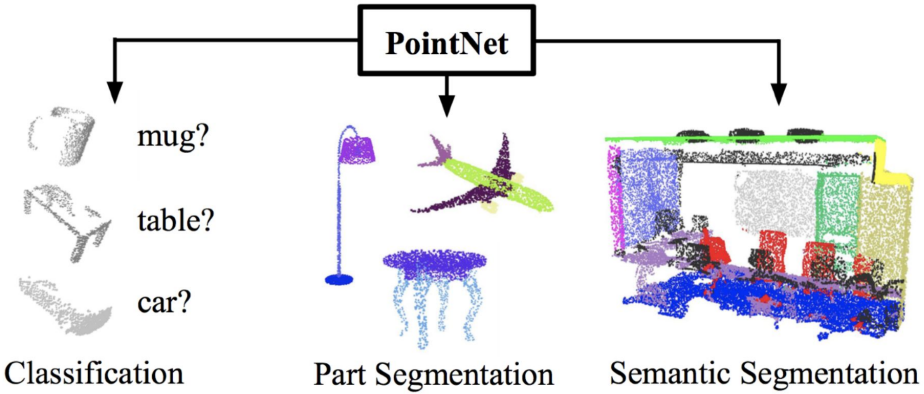
TRIUMF

# *Deep Learning*

# The models that we used

## Model 1: PointNet

- Developed by researchers at Stanford
- One of the first neural networks specifically meant for point clouds




## Model 2: Dynamic Graph CNN

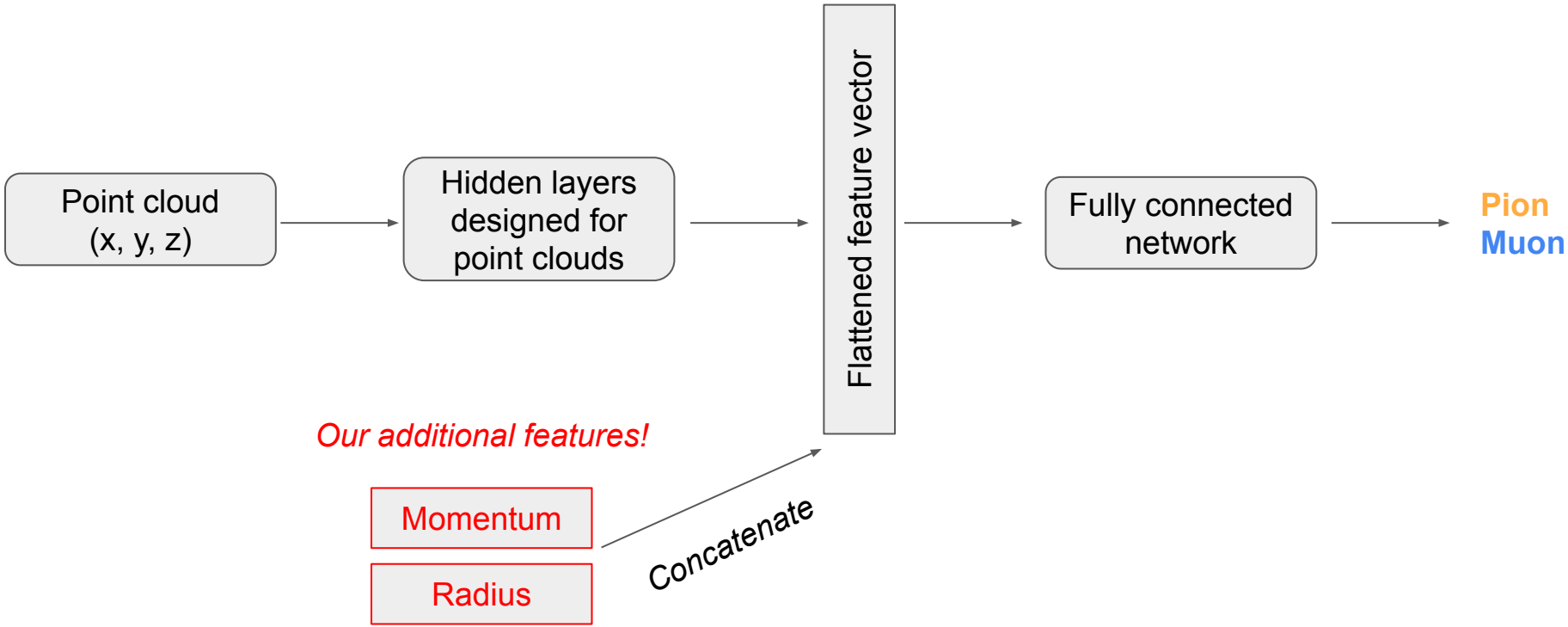
- Developed by researchers at MIT
- Also intended for point cloud data
- Able to extract local information within pointcloud (PointNet cannot)

### *Dynamic Graph CNN Key Idea*

*“Edge convolutions”*

Source: https://arxiv.org/pdf/1612.00593.pdf

# High level model architecture (both models)



TRIUMF

# *Deep Learning*
# Model #1: PointNet

# How PointNet addresses problems with point clouds

**Problem 1**

Point clouds are unordered sets of coordinates, changing the order within the set should not change the prediction



**Solution 1**

Apply max pooling (symmetric function) to transformed inputs

**Problem 2**

Rotating the point cloud should not change the corresponding prediction (robust to spatial variability)

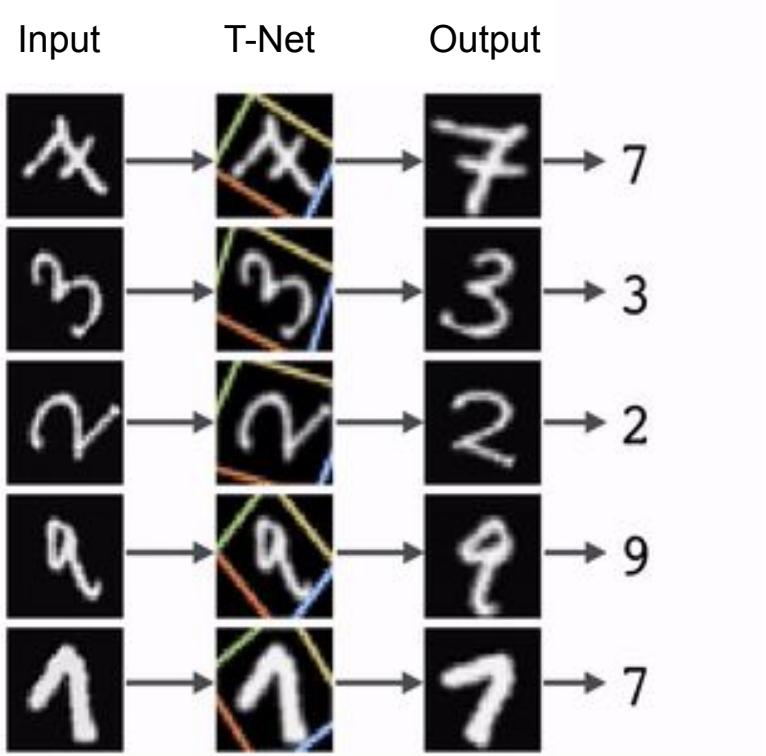


**Solution 2**

Spatial transformer network (“T-net”)

# Spatial Transformer Network

- T-nets come from separate research
- T-nets are robust to input rotations
- PointNet uses a T-net to take advantage of this!




Source: https://towardsdatascience.com/review-stn-spatial-transformer-network-image-classification-d3cbd98a70aa

TRIUMF

# *Deep Learning*
# PointNet Results

# Tuning PointNet

## Feature combinations

- **Hits (x, y, z), momentum and radius**
- **Hits (x, y, z), momentum**

## Time filter values (delta)

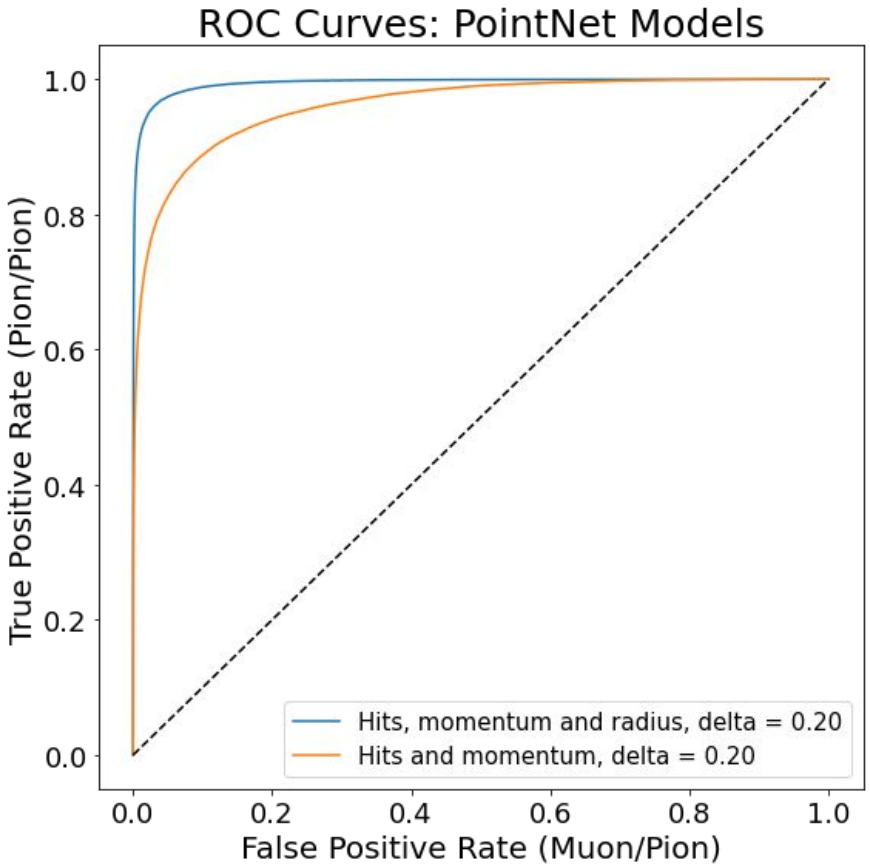
- 0.20ns to 0.50ns

## Learning rate

- Constant learning rate
- Learning rate scheduler

## Epochs

- Up to 24 epochs



# Best PointNet model

## Best feature combination

- Hits point cloud (x, y, z)
- Momentum
- Radius

## Best hyperparameters

- Time delta = 0.20 ns
- 16 epochs
- Learning rate scheduler
  - 0.003 to 0.03 to 0.003

## Chosen operating point (ROC)

- 0.93

## Results

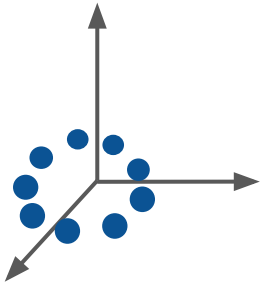
- Exceeds TRIUMFS prior **pion efficiency** results!
- Good muon efficiency, better than TRIUMF at *some* momentums

TRIUMF

# *Deep Learning*
# Model #2: Dynamic Graph CNN

# Comparing differences in model architectures

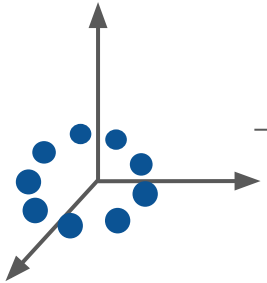
**PointNet**



T-net → Fully Connected Network → Classification

Spatial variability

**Dynamic Graph CNN**



Extracts local information!

Create graph with KNN → **Edge Convolutions** → Fully Connected Network → Classification

TRIUMF

# *Deep Learning*
# Dynamic Graph CNN Results

# Tuning Dynamic Graph CNN

## Feature combinations

- Hits (x, y, z), momentum and radius
- Hits (x, y, z), momentum

## k for KNN graph

- 8 to 20 nearest neighbors

## Time filter values (delta)

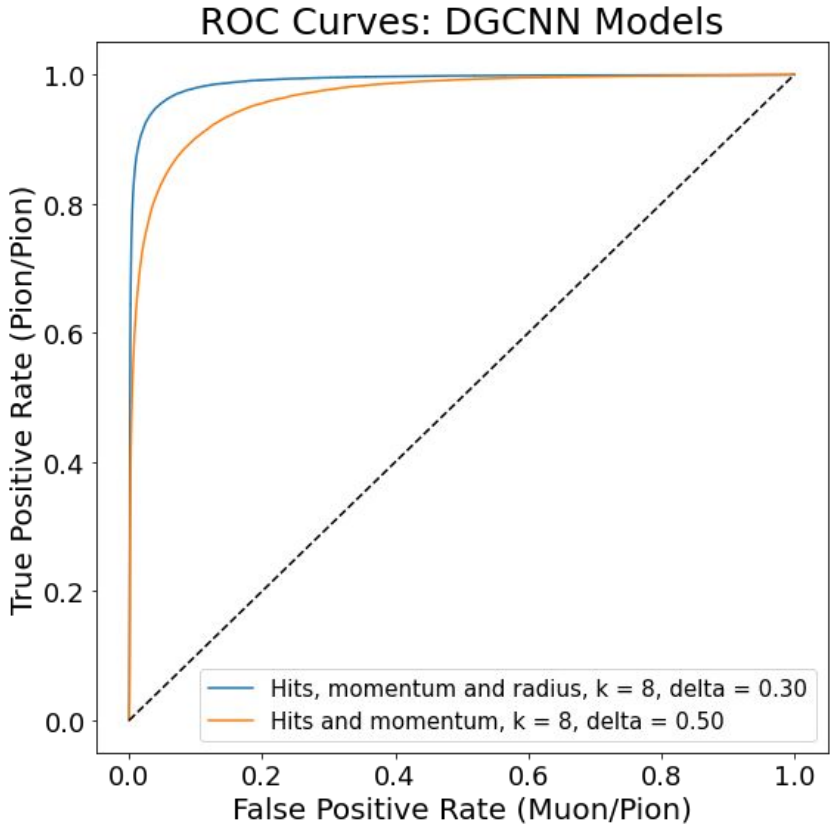
- 0.20ns to 0.50ns

## Learning rate

- Constant learning rate
- Learning rate scheduler

## Epochs

- Up to 12 epochs



# Dynamic Graph CNN best model summary

## Best feature combination

- Hits point cloud (x, y, z)
- Momentum
- Radius

## Best hyperparameters

- k = 8 nearest neighbors
- Time delta = 0.30 ns
- 8 epochs
- Learning rate scheduler
  - 0.003 to 0.03 to 0.003

## Chosen operating point (ROC)

- 0.96

## Results

- *Similar* pion efficiency to PointNet
- *Worse* muon efficiency to PointNet

TRIUMF

*Deep learning*

# Overall summary of model results

# Deep learning overall model results

**Model performance**

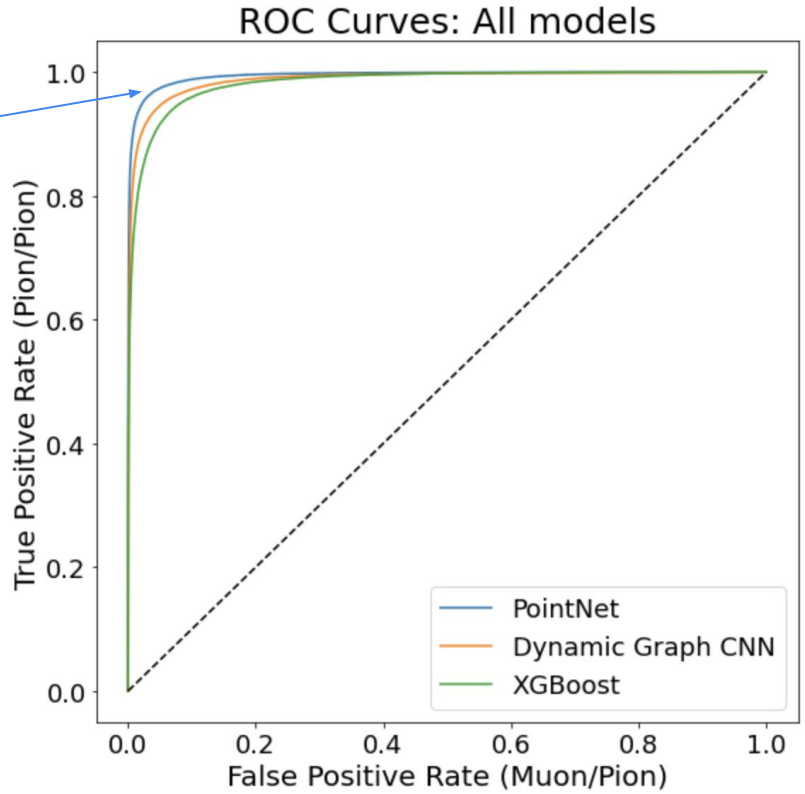
- **Overall best model: PointNet**
- Runner up: DGCNN
- Both models beat baseline!

**PointNet Pros**

- Exceeds all prior TRIUMF **pion efficiency**
- Pions more accurately identified
- Maintains strong muon efficiency

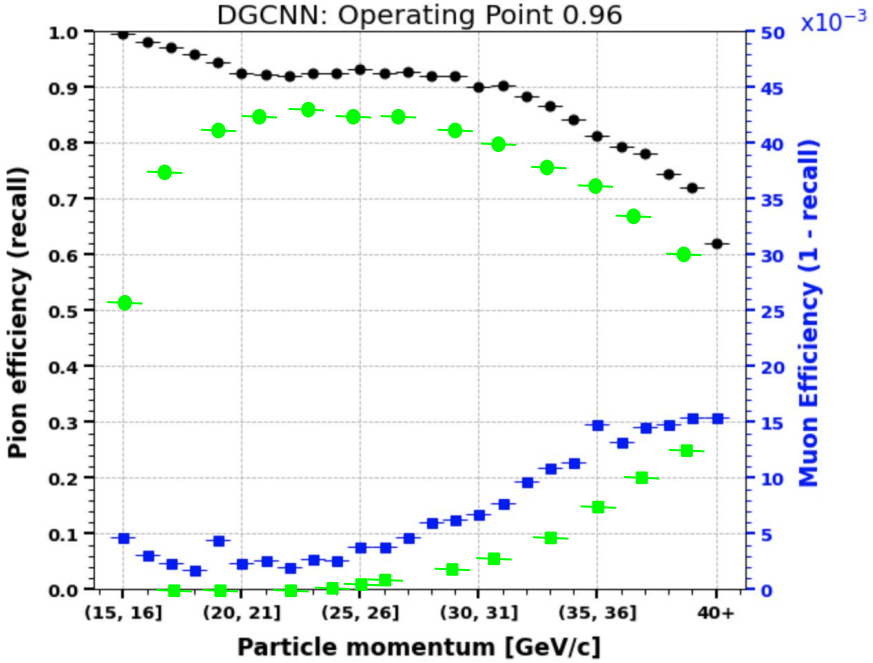
**PointNet Cons**

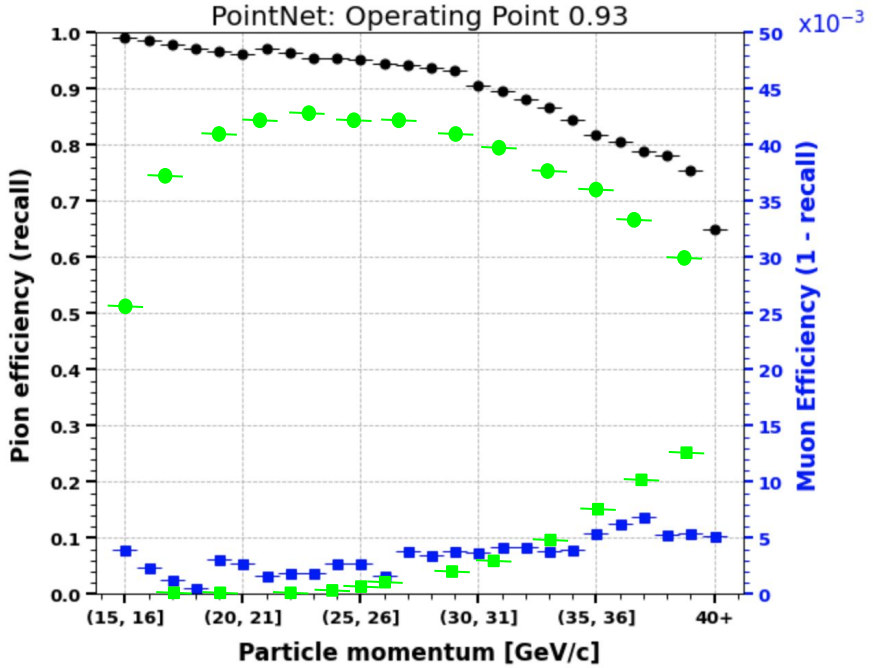
- Longer training time (double DGCNN)
- More parameters, potential to overfit



# DGCNN vs. PointNet vs. TRIUMF

Prior TRIUMF performance




**Better pion efficiency** while *maintaining* good muon efficiency!

# Potential suggestions for the future

1) Further tuning of **hyperparameters** (e.g. Bayesian optimization)

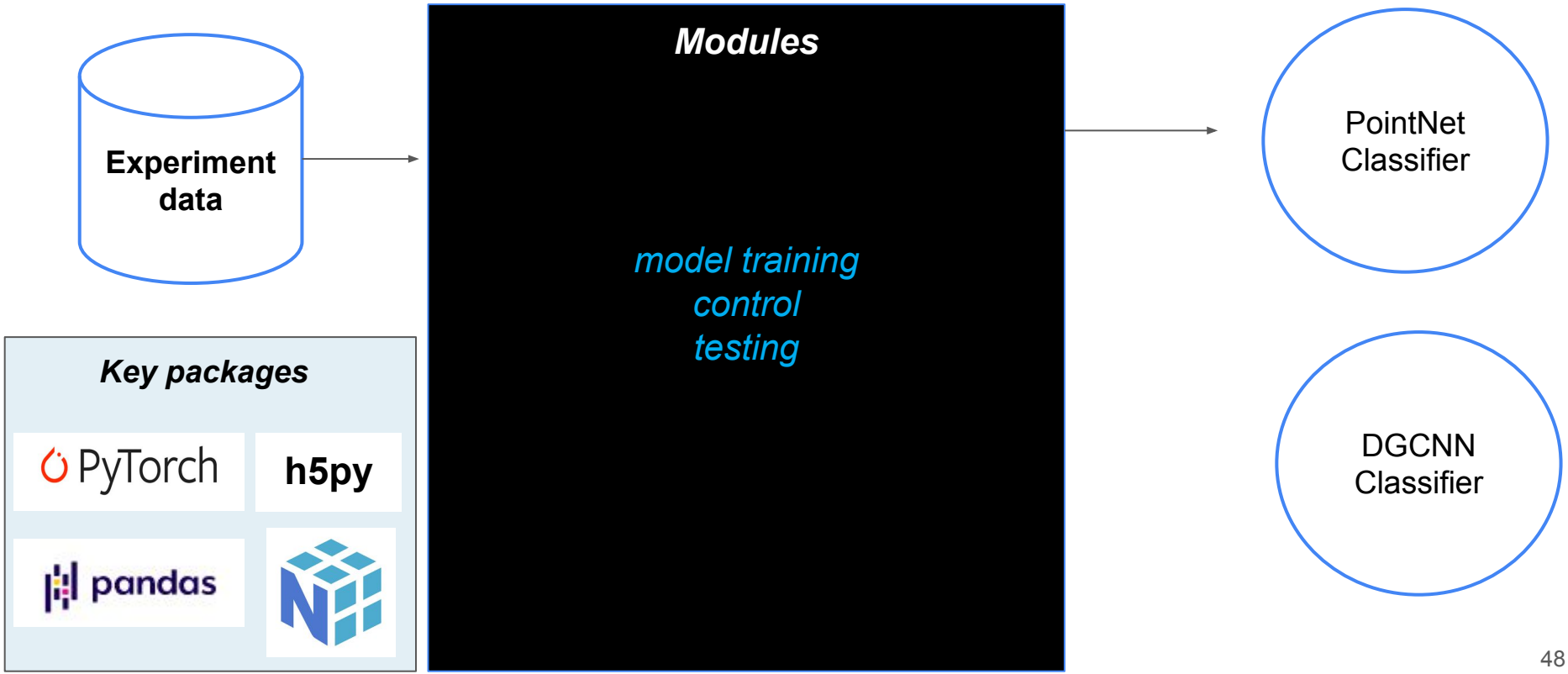
2) Try **other models** (e.g. PointNet++)

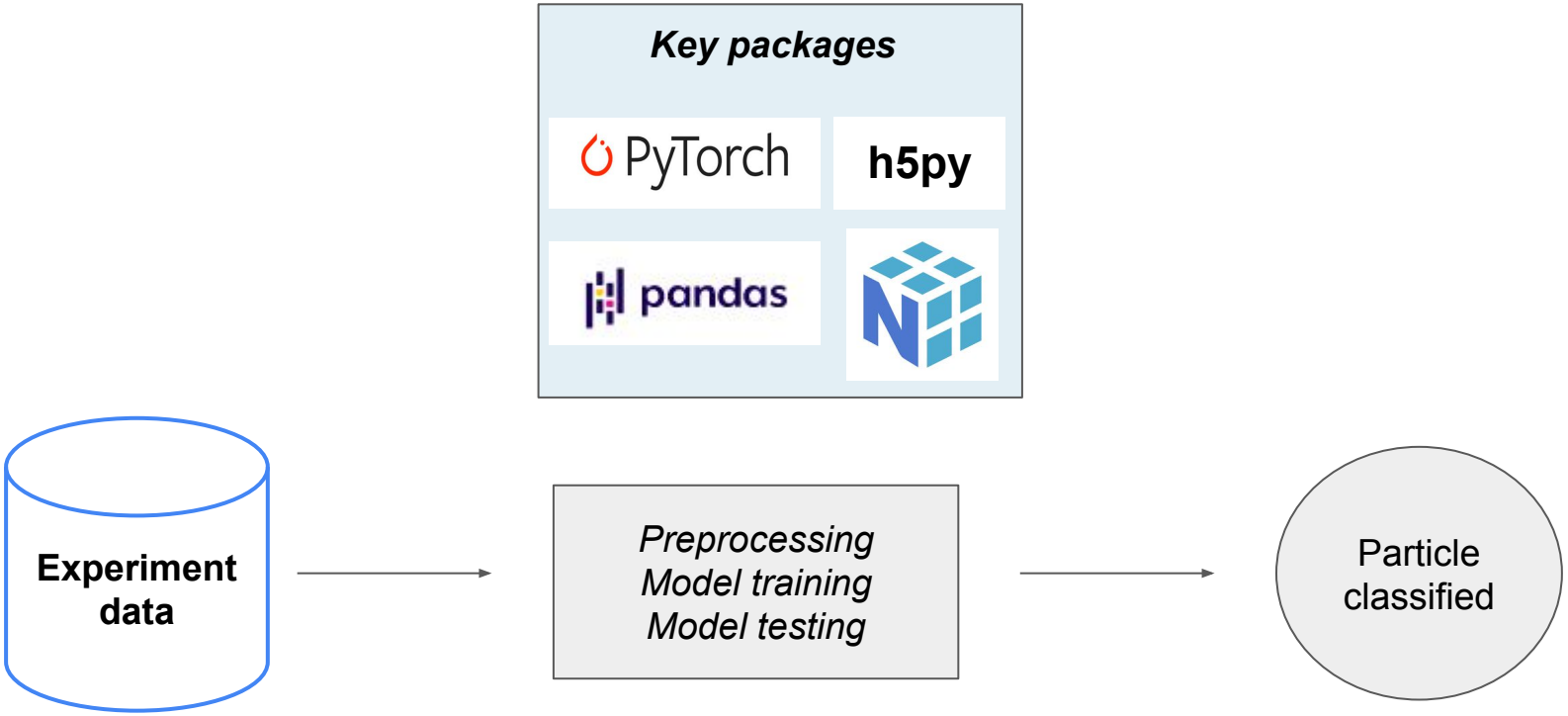
3) Add **more muons** to the training dataset to help the model identify muons more accurately and further reduce muon efficiency

TRIUMF

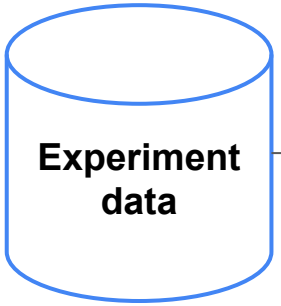
# Final data product

# Final data product: Modularized Neural Network classifiers




Key packages
PyTorch
h5py
pandas
Experiment data
Preprocessing
Model training
Model testing
Particle classified

# Final data product: Modularized Neural Network classifiers




**Key packages**

PyTorch

**h5py**

pandas

**Key Modules**

```
.
└── richai/
    ├── configs/
    │   └── configs.yaml
    ├── dataset/
    │   ├── rich_dataset.py
    │   └── data_loader.py
    ├── models/
    │   ├── pointnet.py
    │   └── dgcnn.py
    ├── train_evaluate/
    │   ├── train.py
    │   └── evaluate.py
    ├── utils
    └── docs
```

*Brief snapshot*

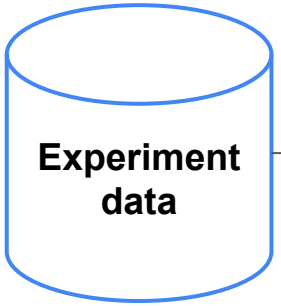
```yaml
data_loader:
  batch_size: 64
  num_workers: 8
  drop_last: True

model:
  pointnet:
    num_classes: 1
    epochs: 16
    saved_model: "saved_models/pointnet_1.
    predictions: "saved_models/pointnet_1_
    momentum: True

gpu:
  - 1
  - 2
  - 3

seed: 2022
```

# Final data product: Modularized Neural Network classifiers




**Key packages**

PyTorch

**h5py**

pandas

**Key Modules**

```
.
└── richai/
    ├── configs/
    │   └── configs.yaml
    ├── dataset/
    │   ├── rich_dataset.py
    │   └── data_loader.py
    ├── models/
    │   ├── pointnet.py
    │   └── dgcnn.py
    ├── train_evaluate/
    │   ├── train.py
    │   └── evaluate.py
    ├── utils
    └── docs
```

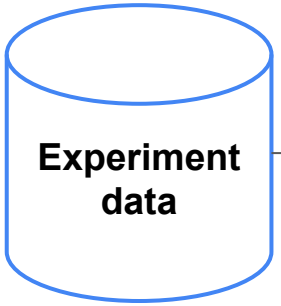
*Brief snapshot*

```python
from torch.utils.data import DataLoader
from torch.utils.data.sampler import SubsetRandomSampler
from utils.helpers import get_config

def data_loader(dset):
    """Create pytorch data loader"""
    train_loader = DataLoader(
        dset,
        batch_size=get_config("data_loader.batch_size"),
        shuffle=False,
        sampler=SubsetRandomSampler(dset.train_indices),
        num_workers=get_config("data_loader.num_workers")
        drop_last=get_config("data_loader.drop_last"),
    )

    val_loader = DataLoader(
        dset,
        batch_size=get_config("data_loader.batch_size"),
        shuffle=False,
        sampler=SubsetRandomSampler(dset.val_indices),
        num_workers=get_config("data_loader.num_workers")
        drop_last=get_config("data_loader.drop_last"),
    )
```

# Final data product: Modularized Neural Network classifiers




**Key packages**

PyTorch

**h5py**

pandas

***Key Modules***

```
.
└── richai/
    ├── configs/
    │   └── configs.yaml
    ├── dataset/
    │   ├── rich_dataset.py
    │   └── data_loader.py
    ├── models/
    │   ├── pointnet.py
    │   └── dgcnn.py
    ├── train_evaluate/
    │   ├── train.py
    │   └── evaluate.py
    ├── utils
    └── docs
```

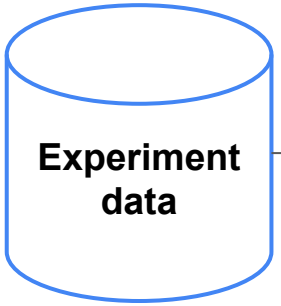
*Brief snapshot*

```
class STNkd(nn.Module):
    def __init__(self, k=64):
        super().__init__()
        self.k = k
        self.conv1 = nn.Conv1d(k, 64, 1)
        self.conv2 = nn.Conv1d(64, 128, 1)
        self.conv3 = nn.Conv1d(128, 1024, 1)
        self.fc1 = nn.Linear(1024, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, k * k)
        self.relu = nn.ReLU()

        self.bn1 = nn.BatchNorm1d(64)
        self.bn2 = nn.BatchNorm1d(128)
        self.bn3 = nn.BatchNorm1d(1024)
        self.bn4 = nn.BatchNorm1d(512)
        self.bn5 = nn.BatchNorm1d(256)

        self.iden = Variable(
            torch.from_numpy(np.eye(self.k).fla
        ).view(1, -1)
```

# Final data product: Modularized Neural Network classifiers




**Key packages**

PyTorch

**h5py**

pandas

**Key Modules**

```
.
└── richai/
    ├── configs/
    │   └── configs.yaml
    ├── dataset/
    │   ├── rich_dataset.py
    │   └── data_loader.py
    ├── models/
    │   ├── pointnet.py
    │   └── dgcnn.py
    ├── train_evaluate/
    │   ├── train.py
    │   └── evaluate.py
    ├── utils
    └── docs
```

*Brief snapshot*

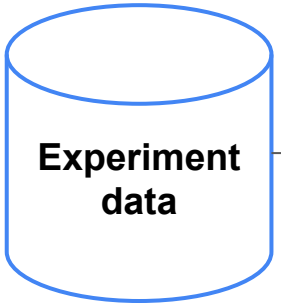
```python
def trainer(
    model,
    optimizer,
    train_loader,
    val_loader=None,
    criterion=None,
    epochs=5,
    scheduler=None,
    device='cuda',
    results=False,
):
    """Trainer for PyTorch"""

    logger.info(f'Starting training...')
    training_start = time.time()

    train_losses, train_accs = [], []
    valid_losses, valid_accs = [], []

    model.to(device)
```

# Final data product: Modularized Neural Network classifiers




**Key Modules**

```
.
└── richai/
    ├── configs/
    │   └── configs.yaml
    ├── dataset/
    │   ├── rich_dataset.py
    │   └── data_loader.py
    ├── models/
    │   ├── pointnet.py
    │   └── dgcnn.py
    ├── train_evaluate/
    │   ├── train.py
    │   └── evaluate.py
    ├── utils
    └── docs
```

**Key packages**

PyTorch

**h5py**

pandas

**PointNet** Classifier

**DGCNN** Classifier

TRIUMF

CERN NA62

# Thank You!

Questions?

TRIUMF

# Appendix

TRIUMF

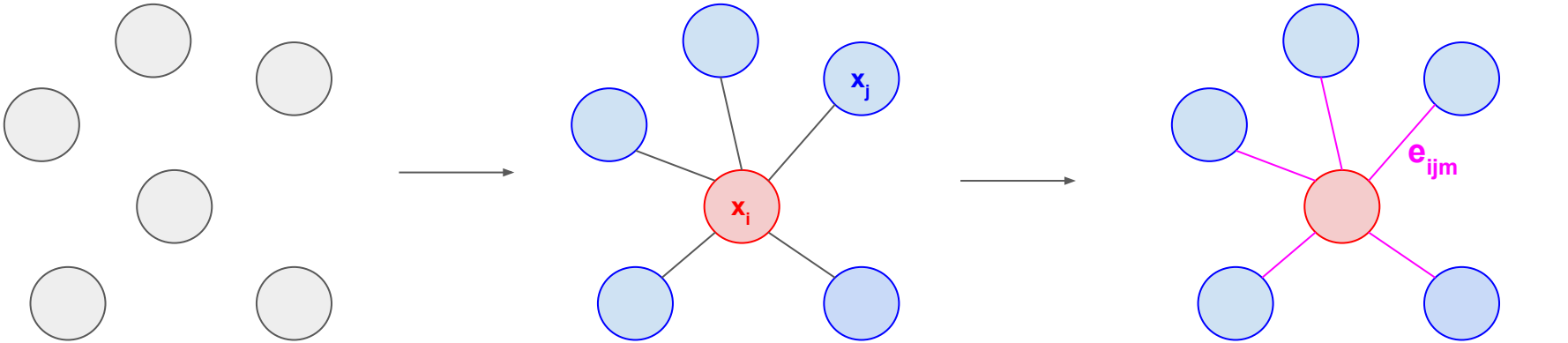
# Appendix A: Edge Convolutions

# Edge convolutions

**Step 1**
Raw point cloud input

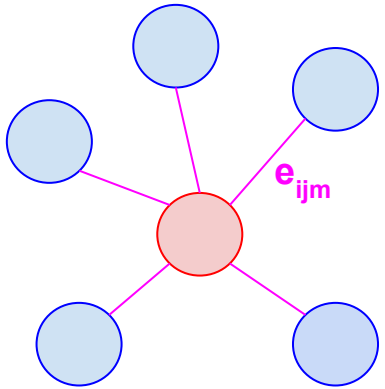
**Step 2**
Connect each node $x_i$ to *k* nearest neighbors $x_j$ … $x_k$ to create graph

**Step 3**
For each node and for each edge calculate edge features $e_{ijm}$



# Edge convolutions continued…

**Step 3**
Calculate edge features for each edge $e_{ijm}$




***Edge feature formula***

- *Weights $\theta$ and $\phi$ learned during training*
- *Encodes:*
  - *Global shape structure*
  - *Local information*

$$e_{ijm} = \theta_m \cdot (x_j - x_i) + \phi_m \cdot x_i$$

**Step 4**
Apply activation and aggregation functions

Leaky ReLU Activation
+
Max Pooling

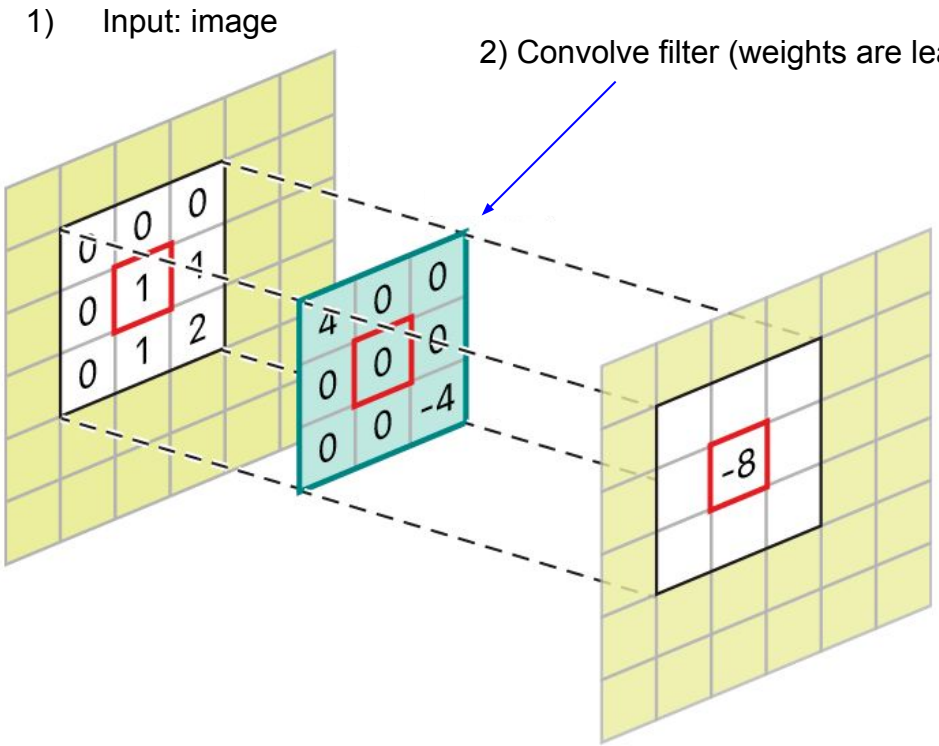
**Step 5**
Output

Next EdgeConv layer
OR
Fully connected layer

TRIUMF

# Appendix B:
# Image Convolutions vs. Edge Convolutions

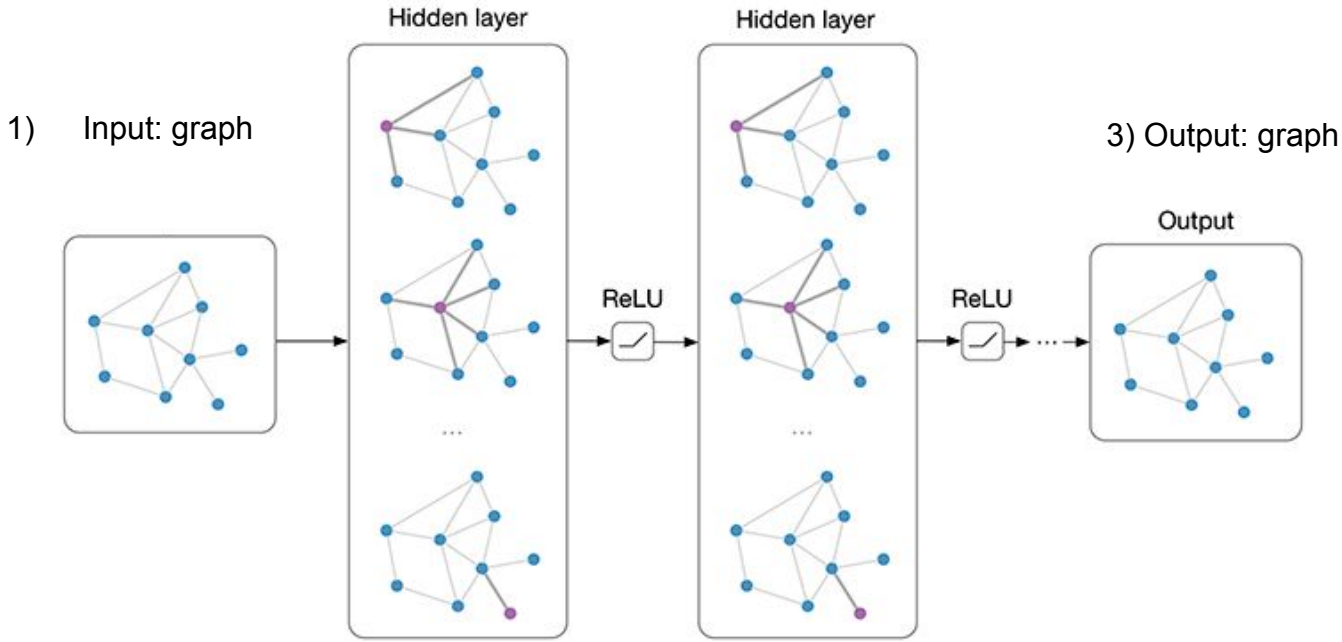
# Image convolution: refresher




Source: https://developer.apple.com/documentation/accelerate/blurring_an_image

# Traditional graph convolutions

2) Convolve filter and learn edge weights



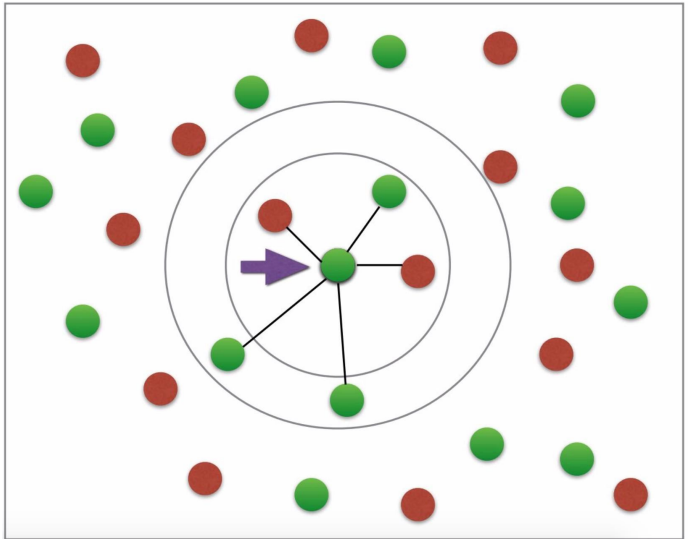

*Note: Graph structure does not change from input to output!*

Source: https://tkipf.github.io/graph-convolutional-networks/

# Edge convolutions: improved graph convolutions

- Graph structure is not fixed!
- Graph is dynamically updated at each layer
- KNN algorithm is used
- Convolutions then performed on edges

KNN graph

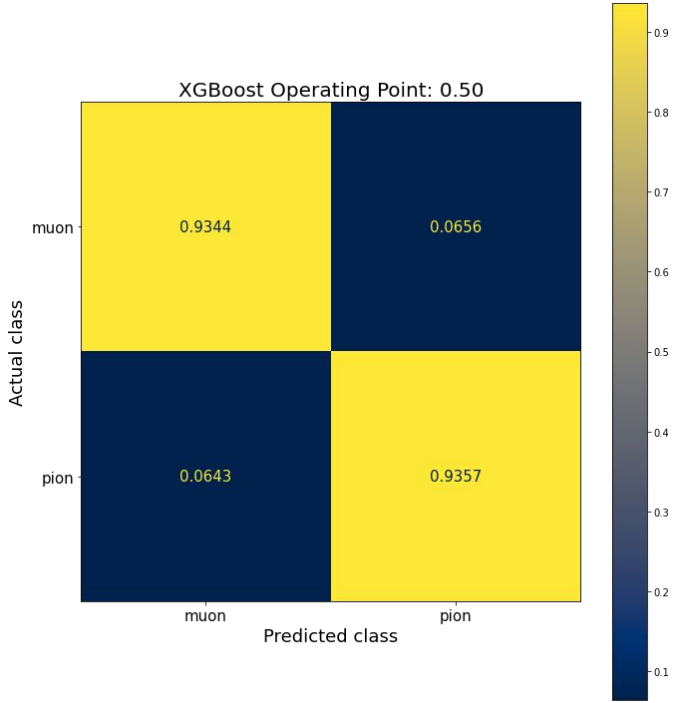


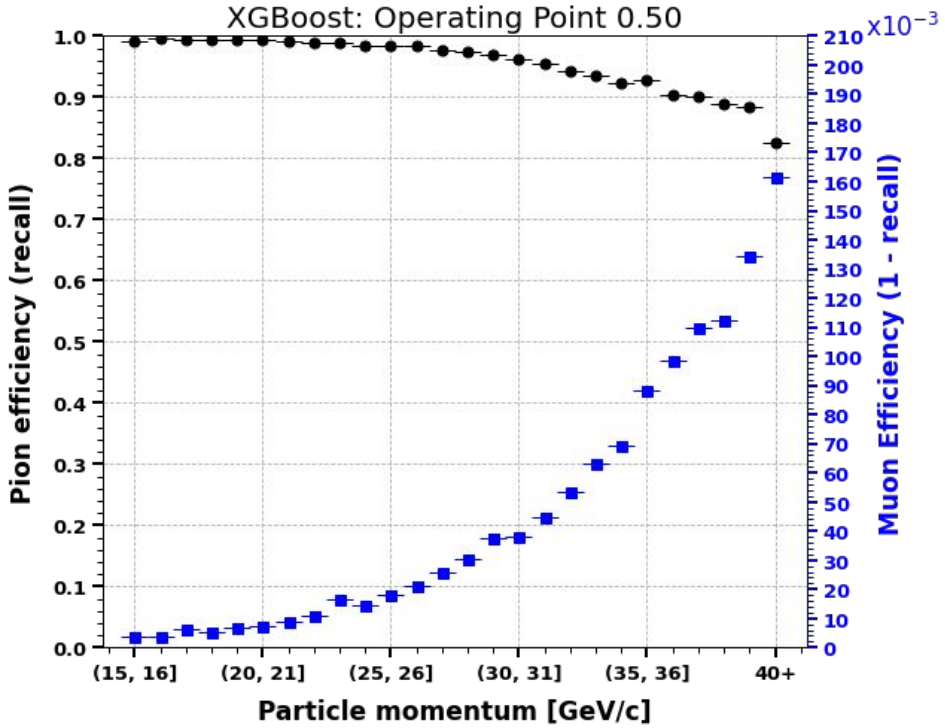
Source: https://i.stack.imgur.com/rIpDf.jpg

TRIUMF

# Appendix C:
# All Models with operating points 0.5

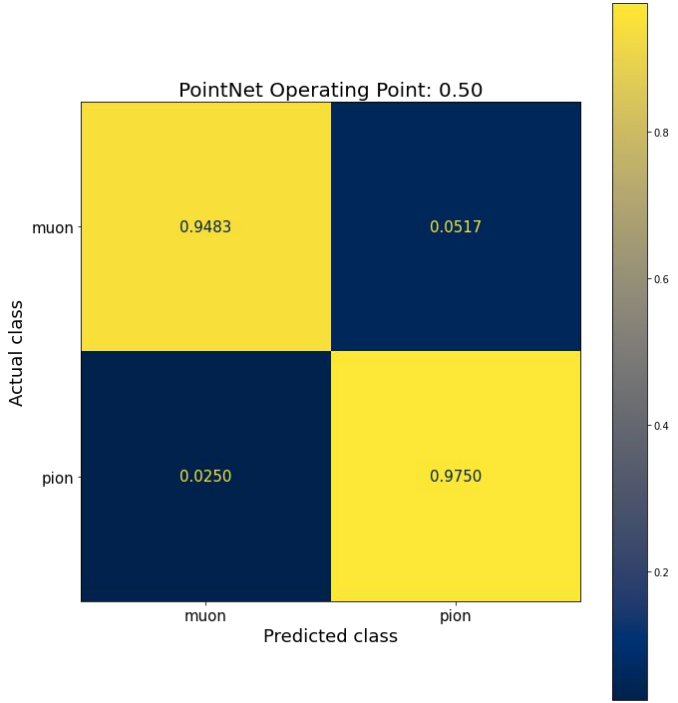
# XGBoost with base 0.5 operating point

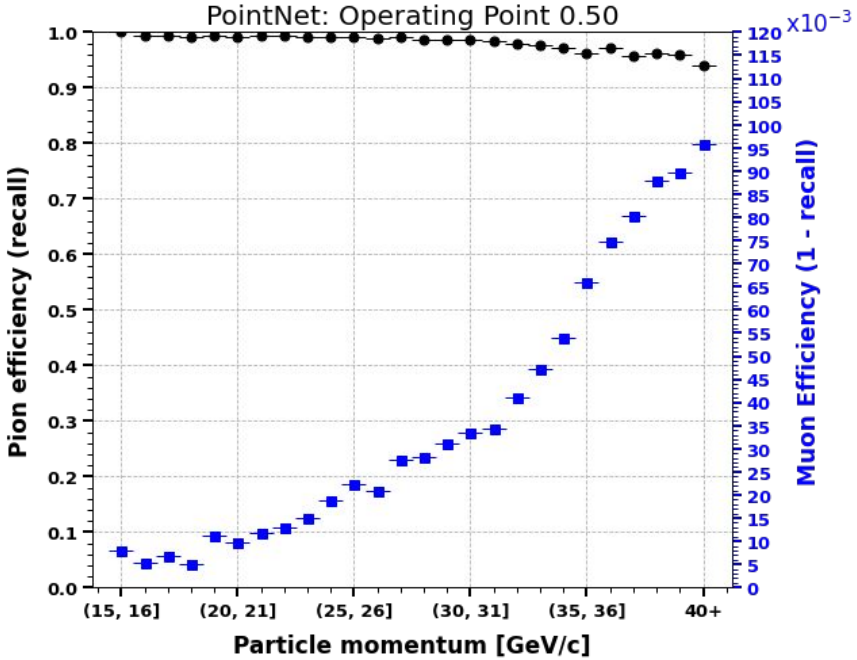



1. As a reminder, pion efficiency is recall for pion.

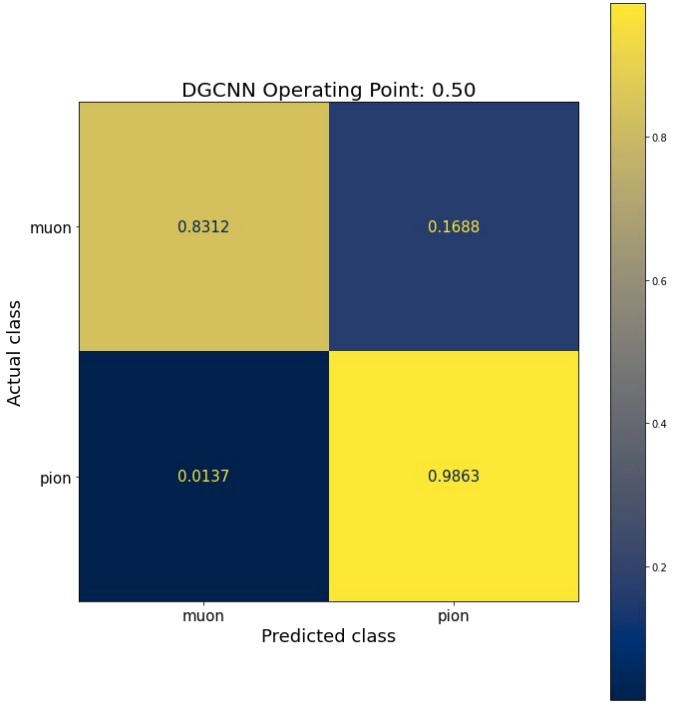
# PointNet with base 0.5 operating point

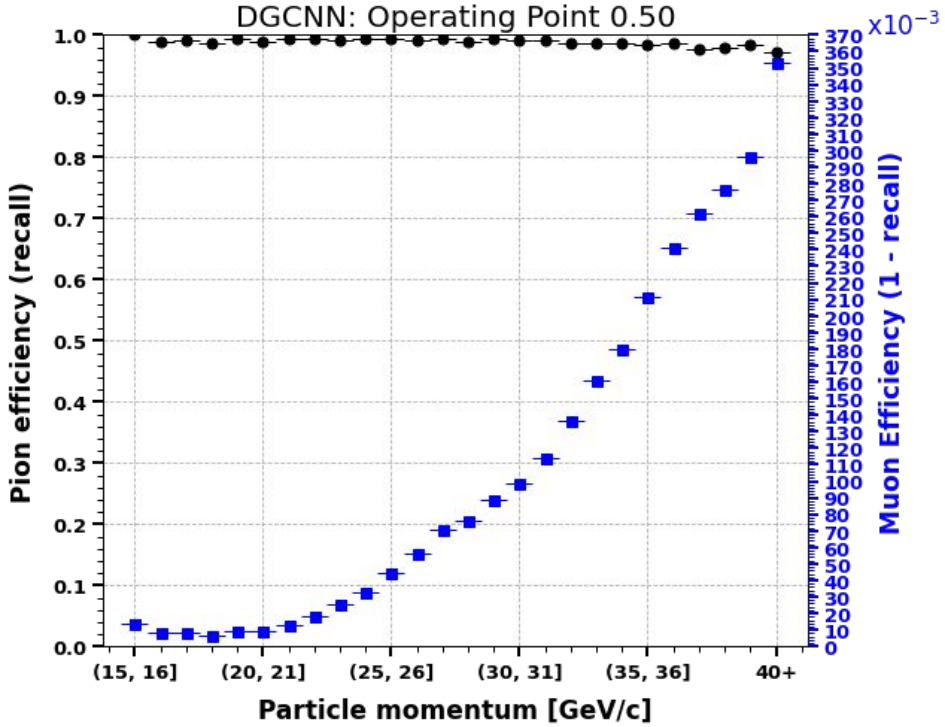



1. As a reminder, pion efficiency is recall for pion.

# Dynamic Graph CNN with base 0.5 operating point



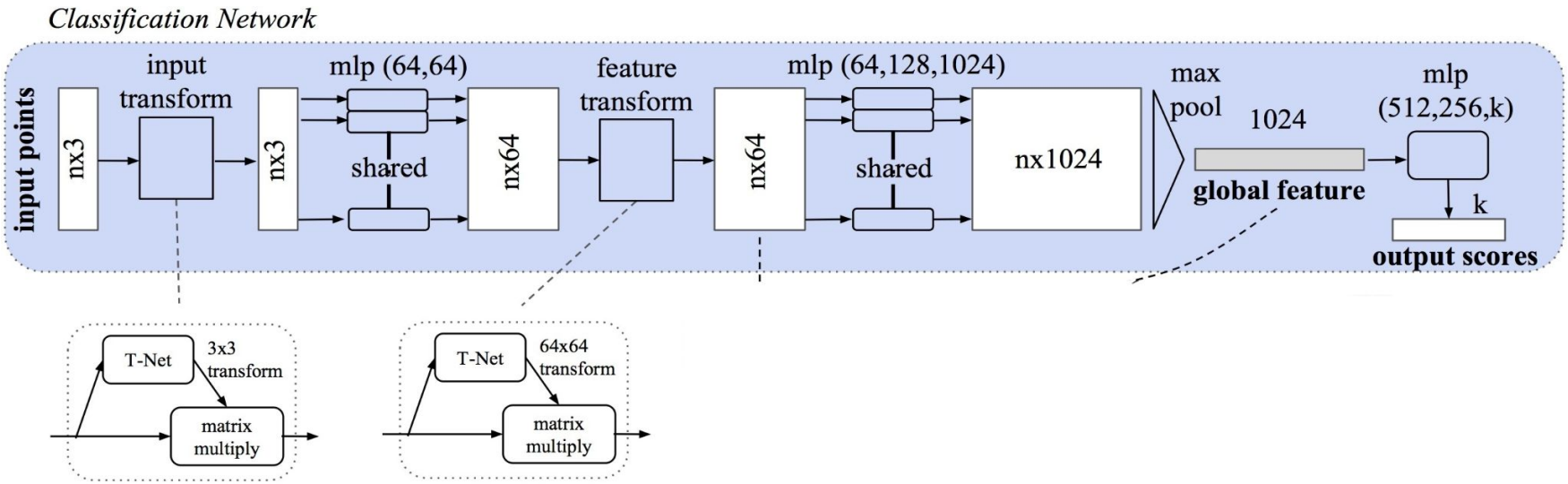




1. As a reminder, pion efficiency is recall for pion.

TRIUMF

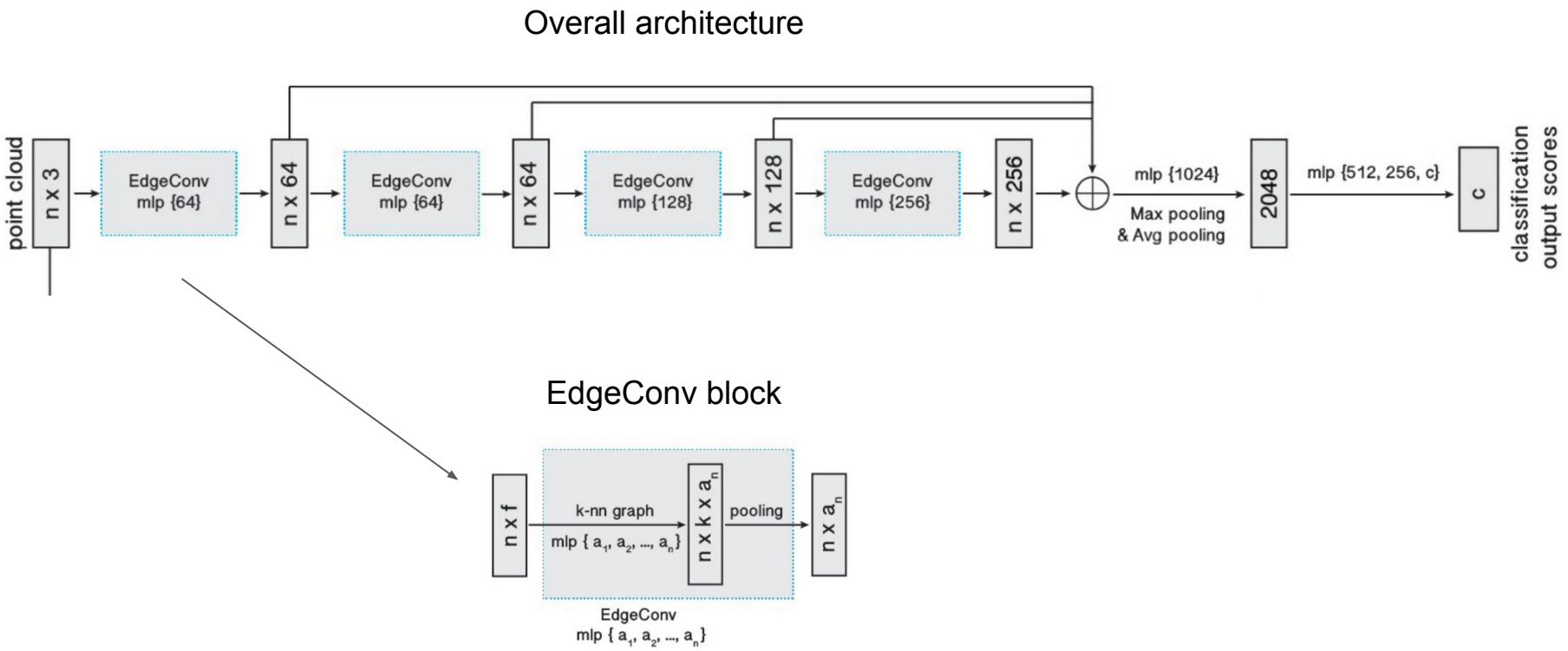
# Appendix D:
# Deep learning architectures

# PointNet Architecture




Source: http://stanford.edu/~rqi/pointnet/

# Dynamic Graph CNN Architecture

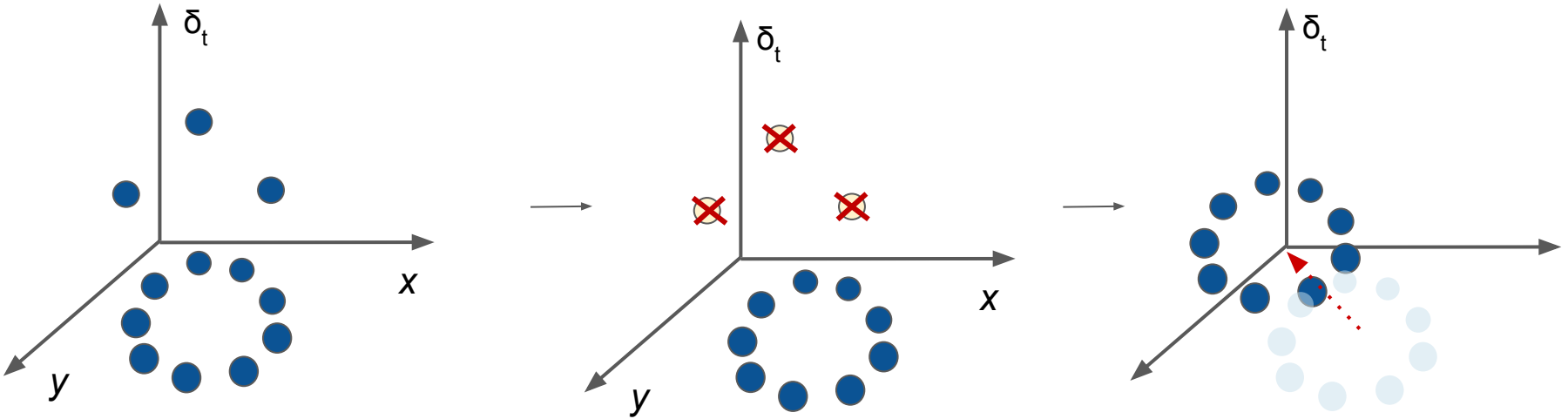



Source: https://github.com/antao97/dgcnn.pytorch

TRIUMF

# Appendix E:
# Physics Related

# The feed into the deep learning models requires 2 filtering steps

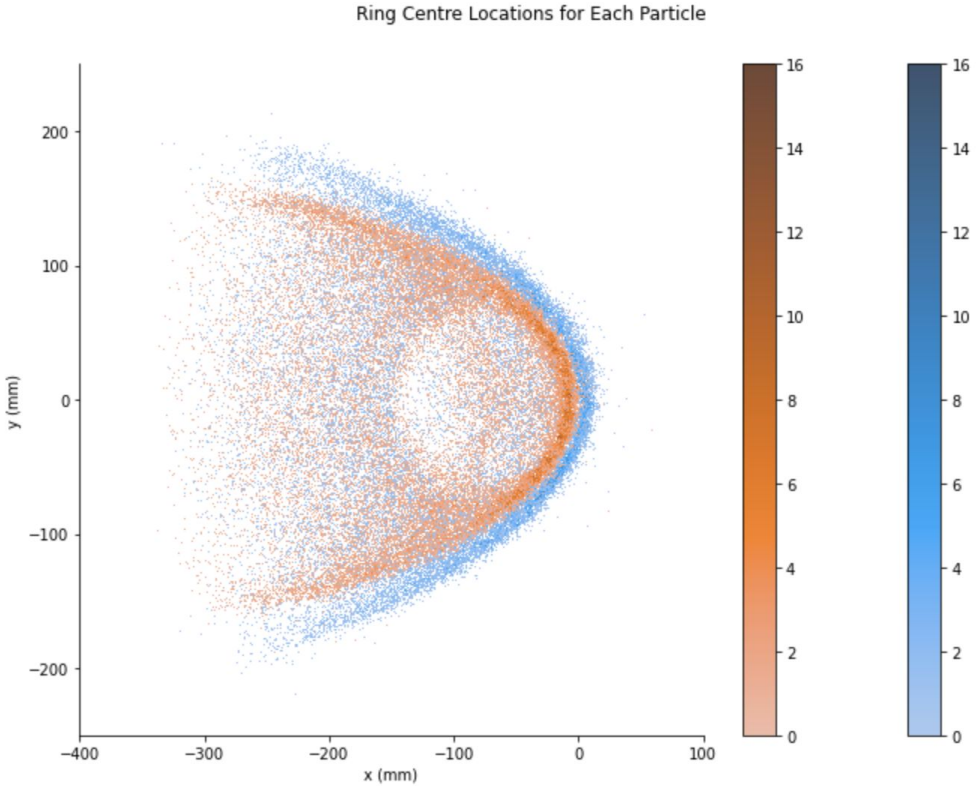



Add time dimension to X,Y coordinates

Filter for time

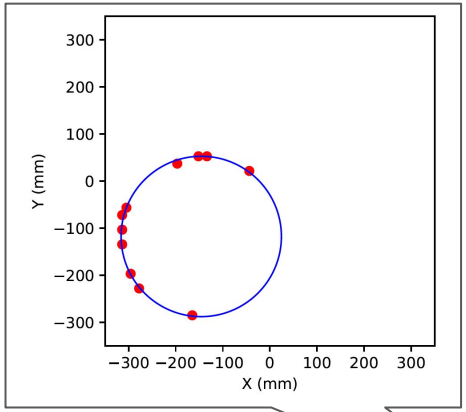
Recenter by subtracting global means for X and Y

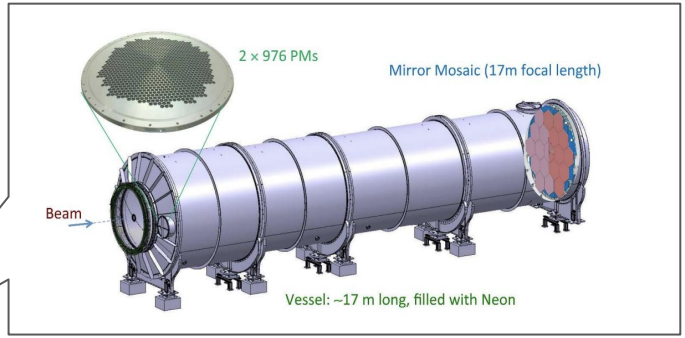
# There is a bias in the ring center




- Ring center distribution is **different** across the particles
- **Recentered** using the global mean X and Y positions

# Each decay process produces light in the RICH detector




RICH detector

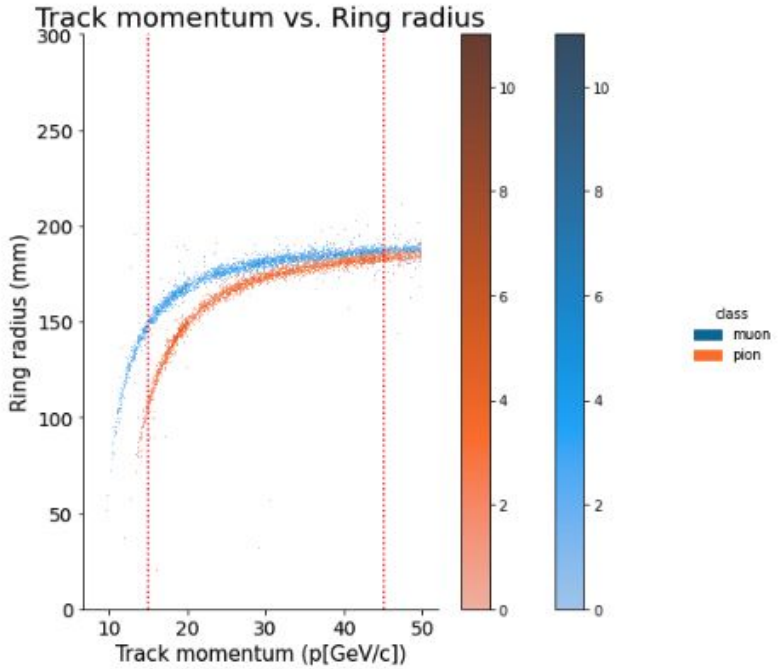
Hodoscope

Pion/Muon

photons

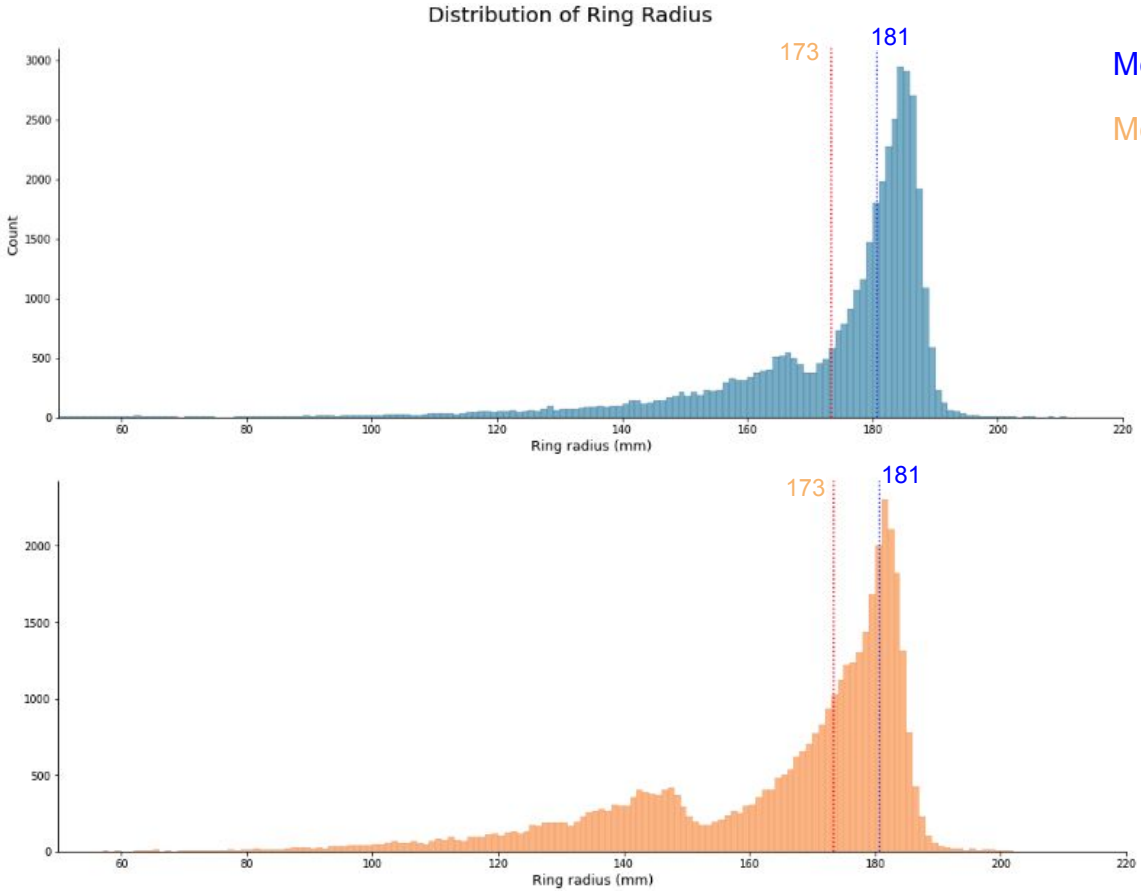
screen

# Undersampling is the answer but…




…need to account for the relationship between the momentum and ring radius

# There is still a distinction in the ring radius between the particles




Median muon radius

Median pion radius