Master Thesis
Denoising by Deep Neural Network in COMET experiment

Saitama University
Graduate School of Science and Engineering

Physical function system Major
Physics course Department

21MP114 Ikuya Sato

February 27, 2023

Abstract

The Standard Model can explain a lot of phenomena with four fundamental forces. And it is consistent with many results from experiments. But there are still some mysteries even with Standard Model. So, we want to find the "Beyond the Standard Model" theory, called the BSM theory.

To collect BSM physical information, there are some particle physics experiments around the world. COMET experiment which is conducted in Japan is one such experiment. In this experiment, we expect to see muon-to-electron conversion, called μ - e conversion. This conversion is not imagined from Standard Model in particle physics. It is because Standard Model does not expect such an interaction which has lepton flavor violations, called LFV. But we already know that there should be such interaction, for example, neutrino oscillation, in nature. Because of this, we can expect μ - e conversion though it's still not discovered. To know whether this phenomenon can occur or not, we need to see this phenomenon with high sensitivity. To achieve this high sensitivity, we need to remove noise hits from background phenomena.

Then, we have been wondering whether true signal hits and background hits can be identified or not. We hope there are any features to identify in trajectory information. Maybe it is a Geometrical pattern. If so, it might be that we can achieve this de-noising using a deep neural network model since in general, Deep Neural Network is known to outperform with such a pattern recognition.

In this thesis, I will show how to remove noise hits using deep neural networks, especially segmentation models.

Contents

1 Introduction 4

2 COMET experiment 4
- 2.1 Introduction . . . 4
- 2.2 Standard Model . . . 4
- 2.3 Muon to Electron Conversion . . . 5
- 2.4 Experimental Flow . . . 6
  - 2.4.1 Detector System . . . 10

3 Deep Learning 12
- 3.1 Introduction . . . 12
- 3.2 Neural Network . . . 12
- 3.3 Activation Function . . . 13
- 3.4 Learning Algorithms . . . 14
  - 3.4.1 Loss and Metric . . . 14
  - 3.4.2 Back Propagation . . . 16
  - 3.4.3 Adam . . . 17
  - 3.4.4 OneCycleLR . . . 18
- 3.5 Convolutional Neural Network . . . 18
  - 3.5.1 Convolution layer . . . 18
  - 3.5.2 Pooling layer . . . 19
- 3.6 Models . . . 19
  - 3.6.1 Denoising Convolutional Neural Network . . . 19
  - 3.6.2 Unet . . . 20
  - 3.6.3 Feature Pyramid Network . . . 21
  - 3.6.4 DeepLabV3+ . . . 22

4 Evaluation way 22
- 4.0.1 ROC curves . . . 22

5 Application of DL in COMET experiment 23
- 5.1 Simulation Data . . . 24
- 5.2 Making Image Datasets . . . 28
  - 5.2.1 Translation from Hit-level to Cell-level . . . 28
  - 5.2.2 Scaling scheme . . . 28
  - 5.2.3 Reduction of image-size . . . 29

6 Result 29
- 6.1 Scaling Scheme Comparison . . . 30
  - 6.1.1 with score of 2D Histogram . . . 30
  - 6.1.2 with Monte-Carlo Truth information . . . 30
- 6.2 Model Comparison . . . 31
  - 6.2.1 Summary of Models . . . 31
- 6.3 Sample size Comparison . . . 32
- 6.4 Translation Comparison . . . 33

7 Summary 34
- 7.1 Future work . . . 35

A 2D Histogram 36

B Graph Neural Network 37
- B.1 Basics of Graph . . . 37
- B.2 Graph with Neural Network . . . 37
  - B.2.1 Graph Convolution . . . 37

Acknowledgement **39**

References **42**

List of Figures

1 Standard Model . . . 5
2 DIO spectrum and conversion-electron spectrum calculated[1] . . . 7
3 Total schematic layout of COMET experiment. Yellow part corresponds Phase-I [1] . . . 8
4 Schematic layout of COMET Phase-I [1] . . . 9
5 A cross-sectional view of CyDet showing the layout of the CTH. [1] . . . 10
6 Simulated drift motion in the cell. [1] . . . 11
7 How Cylindrical Trigger Hodoscope works with four-fold coincidence [1] . . . 11
8 Mathematical Neuron Networks. x_i is input data. w_i is The trainable parameter. y is the output of the neuron. θ is a threshold value. . . . 12
9 Neural Networks. Green nodes are the input layer. Blue nodes are the middle layer. Red nodes are the output layer. . . . 13
10 Classic Activation functions [2] . . . 14
11 ReLU functions [2] . . . 14
12 Example of Training curve . . . 16
13 Learning rate example when I adopt OneCycleLR. The horizontal Axis means epoch number. . . . 18
14 How Convolution filter works. Blue grid means input images. Yellow grid means convolution filter. Convolution operation means summing up following convolution filter . . . 19
15 DnCNN architecture . . . 19
16 Unet architecture [3] . . . 20
17 Especially (a), How Evolution from U-Net to UNet++. [4] . . . 21
18 FPN architecture . . . 21
19 The architecture of DeepLabV3Plus [5] . . . 22
20 Purity Efficiency curve as an example. Relatively large square plots mean when threshold values are 0.3, 0.6, 0.9. . . . 23
21 Rejection Efficiency curve as an example. Relatively large square plots mean when threshold values are 0.3, 0.6, 0.9. . . . 23
22 Left: Cells colored by hit information of representative hits. Right: Cells that contain hits of conversion-electron colored with red. . . . 24
23 Distributions of time information on 30000 events. The 1st figure means the distribution of all hits. The 2nd figure means the distribution of first-turn hits. The 3rd figure means the distribution of all conversion-electron hits. The 4th figure means the distribution of noise hits. . . . 25
24 Distributions of ADCSUM [6] on 30000 events. The 1st figure means the distribution of all hits. The 2nd figure means the distribution of first-turn hits. The 3rd figure means the distribution of all conversion-electron hits. The 4th figure means the distribution of noise hits. . . . 26
25 Distributions of delta phi on 30000 events. The 1st figure means the distribution of all hits. The 2nd figure means the distribution of the first hits on each cell. The 3rd figure means the distribution of all conversion-electron hits. The 4th figure means the distribution of noise hits. . . . 27
26 How reduction works. . . . 29
27 Purity Efficiency curves with scaling schemes. . . . 30
28 Rejection Efficiency curves with scaling schemes. . . . 30
29 Purity Efficiency curves with scaling schemes. . . . 31
30 Rejection Efficiency curves with scaling schemes. . . . 31
31 Purity Efficiency curves with various models. . . . 32
32 Rejection Efficiency curves with various models. . . . 32
33 Purity Efficiency curves with various training sample sizes. . . . 33
34 Rejection Efficiency curves with various training sample sizes. . . . 33
35 Purity Efficiency curves with various translations from Hit-level to Cell-level. . . . 34
36 Rejection Efficiency curves with various translations from Hit-level to Cell-level. . . . 34
37 2D Histogram suggested by Chen Wu . . . 36
38 Left: image in Euclidean space. Right: graph in non-Euclidean space. [7] . . . 37

1 Introduction

It is known that the Standard Model can explain a lot of phenomena around particle physics. But there are still some mysteries, for example, neutrino oscillation, dark matter, etc...

Recent research in particle physics aims to discover a new theory called "Beyond the Standard Model"(in short "BSM"). COMET experiment is one such experimental research.

COMET experiment aims to observe μ - e conversion as mentioned later. The μ - e conversion was identified by energy-momentum on each event. At this moment, of course, the detector system would suffer from noisy background hits. To get high sensitivity and do computation efficiently, we thought that we need to remove noisy background hits. In detail, we thought that there are 3 stages to get accurate momentum on each event:

- De-noising: Remove background hits in the detector on each event.
- First Turn Extraction: Extract 1st turn hits from hits which conversion electron left
- Energy Momentum Estimation: Estimate energy-momentum from 1st turn hits

In this thesis, I'll mention the 1st stage, de-noising. Technically we need to classify all hits to hits of conversion-electron or just noise hits for energy-momentum estimation. But for now, I tried to remove noise hits with my Cell-level analysis though we need actually Hit-level analysis, not Cell-level analysis. The "Cell-level" means that we can only recognize whether each cell has hits of conversion-electron or not, in other words, we don't have the ability to label each hit to something.

2 COMET experiment

2.1 Introduction

This experiment will be done in J-PARC. J-PARC has a world-class large facility that emits proton-beam. Pion which is created by proton-beam is desirable as the source of muon this experiment requires. In this section, we overview the COMET experiment but I mention about Standard Model before.

2.2 Standard Model

The Standard Model (theory) can explain a lot of phenomena in particle physics. This model assumes that there are fundamental 3 interactions which are strong interaction, weak interaction, and electromagnetic interaction. In this theory, many phenomena such as muon decay can be explained well but the existence of dark matter and Lepton Flavor Violation, called LFV, mentioned later cannot be explained. Recent particle physics experiments are done to collect information about physical phenomena that still cannot be explained. In especially COMET experiment will be done to collect information about the LFV process.

In Standard Model, there are lepton particles that interact with weak interaction and electromagnetic interaction (See Figure 1). Leptons have 3 generations same as quarks. Electron and muon and tau particles are called charged leptons. In contrast to it, the neutrino has 3 kinds same as charged leptons. In addition to them, there are anti-particles to each particle.

By the way, in Standard Model, the generation also known as the flavor is important because such a process violating flavor is definitely prohibited. But in fact, we know that a neutrino can become another neutrino that belongs to different species because of neutrino oscillation. Of course, it's flavor violating process. For this reason, the COMET experiment expects that such a flavor-violating process also happens between charged leptons. This is μ - e conversion.

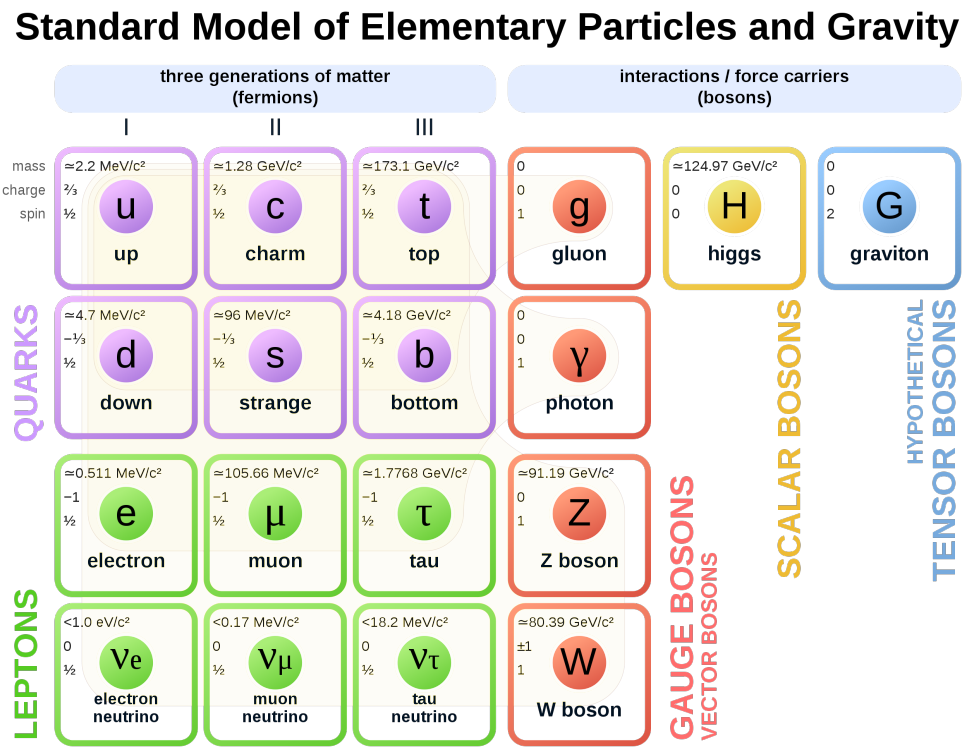

Figure 1: Standard Model

2.3 Muon to Electron Conversion

In general, Muon can decay into 3 particles including electrons.

$$\mu^- \to e^- + \bar{\nu}_e + \nu_\mu \tag{1}$$

But this is called muon-decay not μ - e conversion. Muon-decay conserves lepton flavor. See Table 1.

Table 1: lepton flavor of μ - e conversion

	muon number	electron number	tau number
start	+1	0	0
end	+1	0 (= 1 − 1)	0

where anti-particle has −1 as lepton flavor. We can see muon-decay conserves lepton flavor easily.

On the other hand, μ - e conversion would be interaction between atomic nucleus and μ^- [8]. So, the muonic atom would be required. A muonic atom is an atom that binds a muon instead of an electron.

$$\mu^- + N(A, Z) \to e^- + N(A, Z) \tag{2}$$

Neglecting atomic nucleus, lepton flavors are like Table 2.

Table 2: lepton flavor of μ - e conversion

	muon number	electron number	tau number
start	+1	0	0
end	0	+1	0

Obviously, the lepton flavor is violated.

As a kinematic result, since the atomic nucleus is much heavier than the electron mass, electrons coming from μ - e conversion, which is called conversion-electron, can have an energy of about muon mass. In detail, the expected e which is conversion-electron should have

$$E_{\mu e} = m_\mu - B_\mu - E_{recoil} \tag{3}$$

where $E_{\mu e}$ is energy of conversion-electron, m_μ is muon mass (105 MeV), B_μ is binding energy between μ and atomic nucleus, E_{recoil} is recoil energy.

To observe μ - e conversion, we have a detector that can identify whether a charged particle has 105 MeV or not.

2.4 Experimental Flow

Basically, the flow of this experiment is:

1. π are created by proton decaying
2. π are decayed into μ
3. Make muonic atom by a collision between μ and Aluminum
4. (Expect to see) μ - e conversion

As I mentioned, we want to identify conversion-electron by detecting minus charged particle which has 105 MeV. But we can do this only if there is not any such energetic electron coming from the other phenomena. In fact, there is an obstacle that emits electron which has over 100 MeV. It's decay-in-orbit, called DIO.

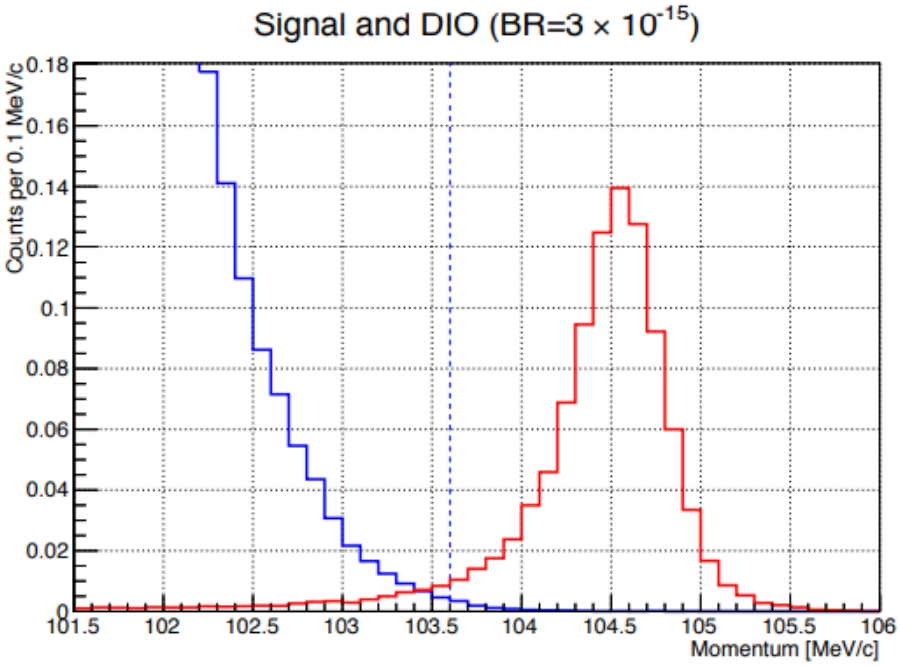

Figure 2: DIO spectrum and conversion-electron spectrum calculated[1]

To understand this, see Figure 2. Blue peak means DIO spectrum. Red peak means μ - e conversion spectrum. The two peaks has bit overlap. So, to identify conversion-electron by the energy-momentum, we need high resolution enough to divide this two peaks.

In COMET experiment Phase-I, we demonstrate a part of the COMET experiment (See Figure 3 and Figure 4).

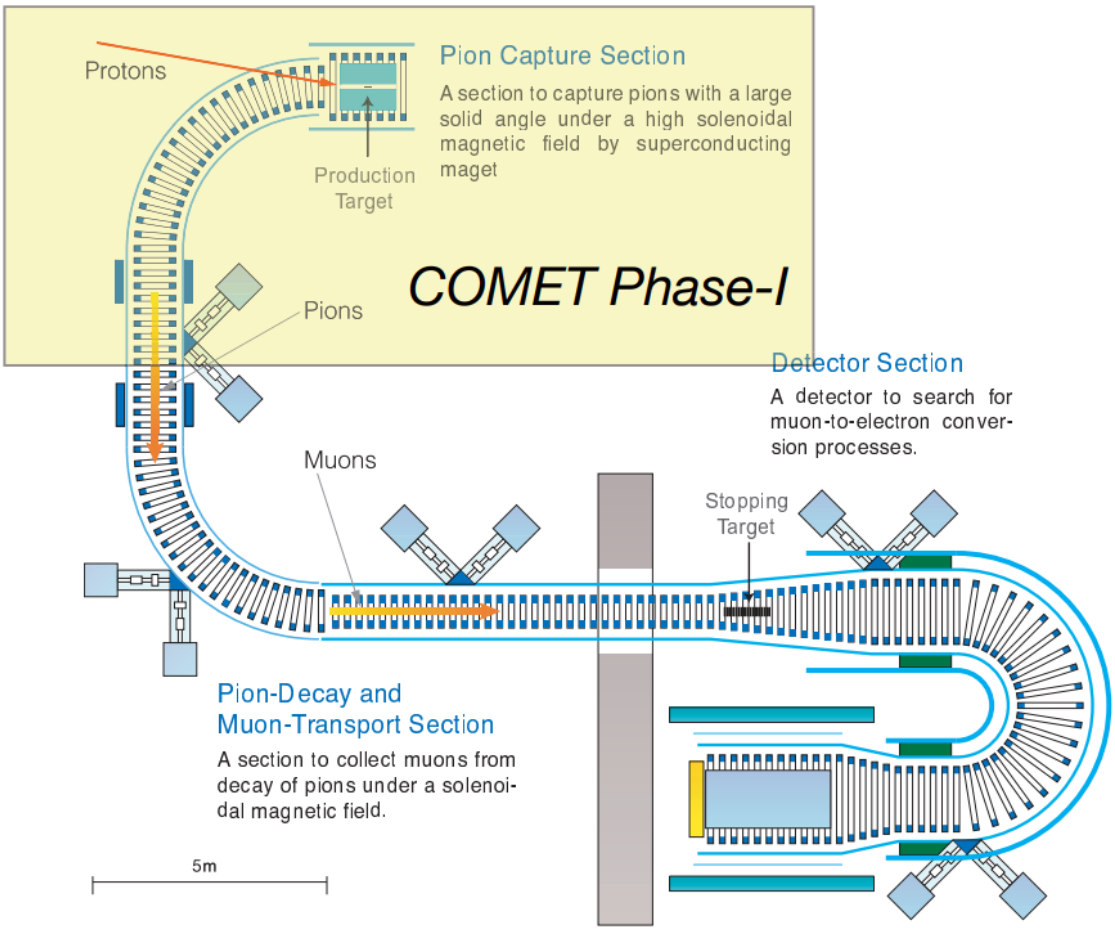

Figure 3: Total schematic layout of COMET experiment. Yellow part corresponds Phase-I [1]

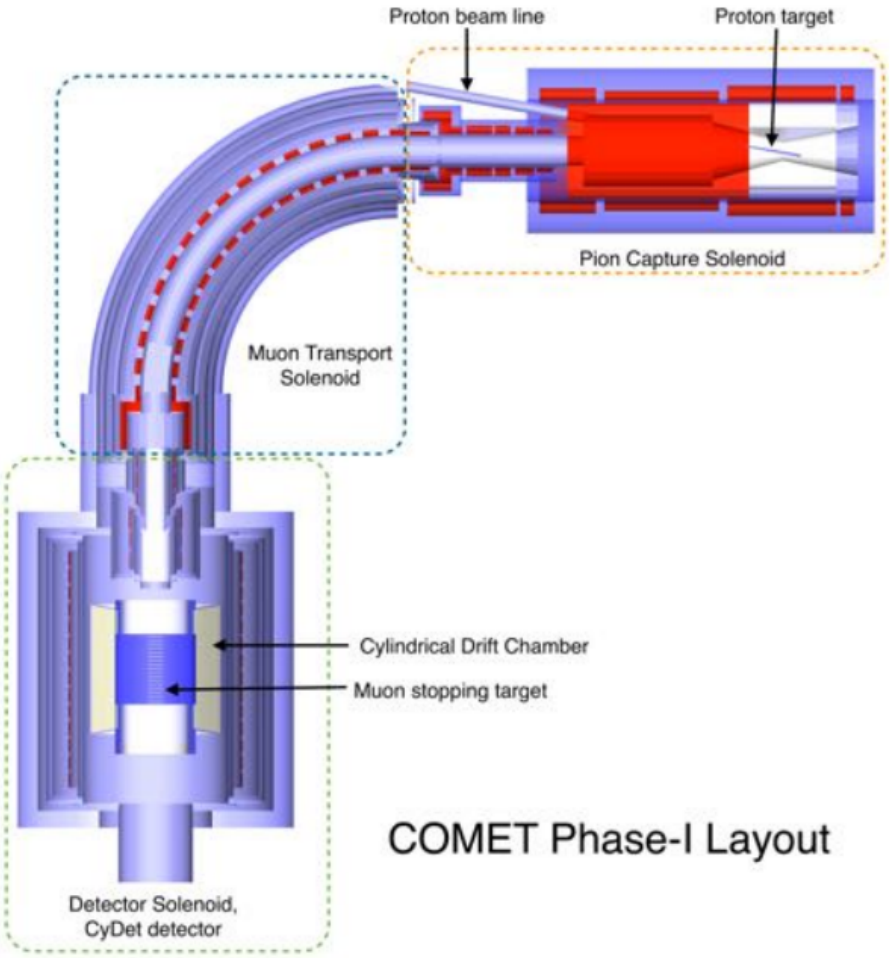

Figure 4: Schematic layout of COMET Phase-I [1]

In the following section, I concentrate to explain only the detector part in COMET Phase-I.

2.4.1 Detector System

In this experiment, Cylindrical Drift Chamber called CDC which is a kind of cylindrical detector (called CyDet) is adopted as the detector. In addition to it, Cylindrical Trigger Hodoscope called CTH and Muon Stopping target are. See Figure 5

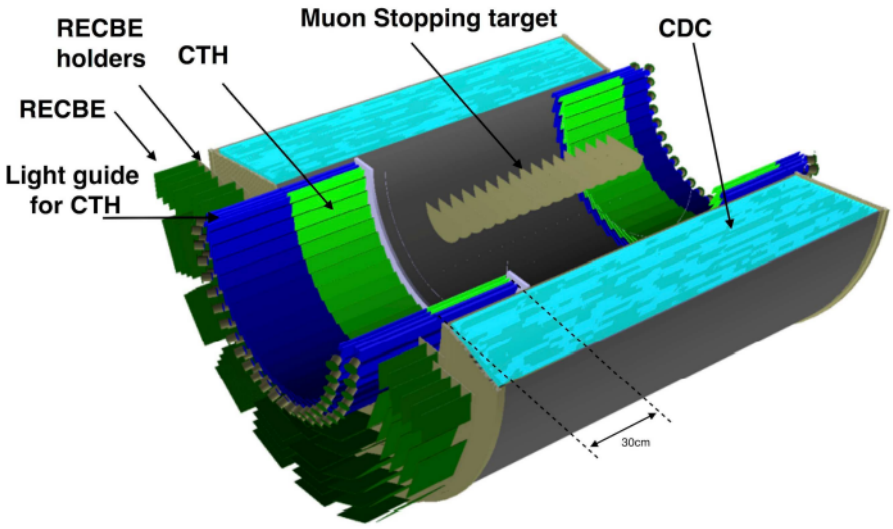

Figure 5: A cross-sectional view of CyDet showing the layout of the CTH. [1]

Muon Stopping targets are several aluminum (in detail it's not pure aluminum) disks to make muonic aluminum. There were several other candidates, for example, Ti and Pb, aluminum are decided for considering the lifetime of a muonic atom and etc [1].

Cylindrical Drift Chamber consists of about 4 thousand cells and wires. In this chamber, the magnetic field is applied along the z-axis. Ideally, a conversion electron should be emitted from a muonic atom. Then, the electron should follow the spiral path in the magnetic field depending on its momentum. The cells and wires tell us the electron's trajectory in the x-y plane. Actually, to observe rough momentum along the z-axis of charged particles, wires, and cells are fixed to have a slight slope with the z-axis.

CDC has a sensor layer that consists of many cells lined up. Cells have an electric field to collect ionized particles in each cell into a central sense wire which is charged positively. The size of the cells is 16.8mm and 16.0mm in width and height. These values are mostly constant in all parts of the CDC. The ionized particle which will be provided by a collision between charged particles such as conversion electron and mixed gas particle diffused in CDC makes Avalanche amplification by collision with another gas particle. Of course, the ionized particle is drifted while it goes to the central sense wire. To see this drift, see Figure 6. The collected ionized particle can be read out as an energy deposit (It will be transformed into ADC SUM mentioned later [6]). The inner diameter and outer diameter of CDC were decided so that CDC avoids hits from DIO electrons having a smaller energy-momentum than 60 MeV while CDC can detect conversion electron has about 105MeV.

Thus we can track charged particle roughly because we can know where charged particle passes through in CDC.

The detector part also includes CTH. There is the CTH in upstream and downstream along the inner wall of the CDC. CTH consists of 48 modules which are a pair of plastic scintillators and a Cherenkov counter. This part tells us which hits are electrons from the protons from nuclear muon capture and cosmic-ray muons by a four-fold coincidence. See Figure 7. CTH also provides a criterion of time information called trigger timing. For our method, we need time information (See Equation (4)) of each hit though we'll mention it later.

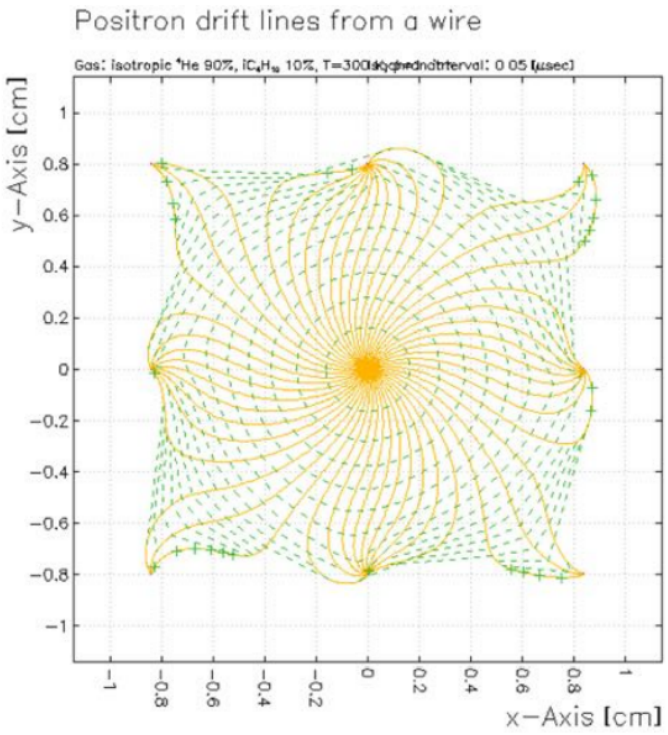

Figure 6: Simulated drift motion in the cell. [1]

$$\begin{aligned}(TimeInformation) &= (CdcHitTiming) - (CthTriggerTiming) \\ &= (DriftTime) + (FlightTime) - (CthTriggerTiming)\end{aligned} \tag{4}$$

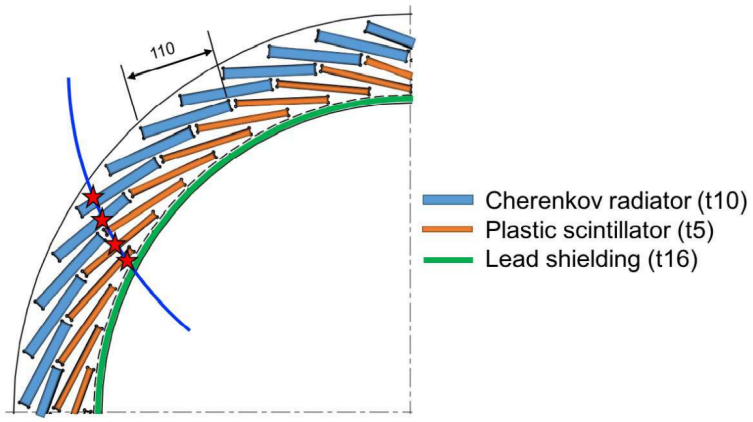

Figure 7: How Cylindrical Trigger Hodoscope works with four-fold coincidence [1]

This detector was designed to avoid hits that are from DIO electron and proton beams it can. In this detector, major particles which leave background hits are DIO electrons and less energetic protons. Regarding the protons, we can identify them easily because protons leave big energy deposits about 100 times what an electron leaves [1].

In this way, we can identify which cells are on the path of charged particles. In my method, In

addition to location information of cells, I also use time information and ADC SUM and relative position referring CTH information.

3 Deep Learning

3.1 Introduction

As I mentioned above, Deep Learning has a grateful capability. Especially for image recognition, there is so many application. It's a kind of machine-learning algorithm. Machine learning algorithms have Boltzmann machines, and support vector machines other than deep learning. In the first place, machine learning is a kind of function that doesn't need to be explicitly implemented by humans to execute any tasks. The machine learning model has a trainable parameter to get a smaller loss. If we can train the model successfully, then it looks like getting spontaneously smart to do a given task.

In the following sections, I mention about the models and algorithms of deep learning.

3.2 Neural Network

Neural Network also called NN is literally a kind of network imitating a neuron's network. So, we can say that the concept of NN is based on biological networks. Please see Figure 8 to understand the behavior.

This neuron has an input and output path which individually has different weight values indicating the strength of the connection between two neurons. If the summations of all input values are bigger than the threshold, then it is activated and output to the next neurons. These combinations of simple forwarding enable to make complex decisions.

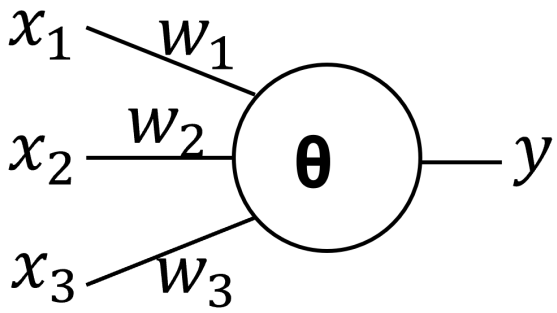

Figure 8: Mathematical Neuron Networks. x_i is input data. w_i is The trainable parameter. y is the output of the neuron. θ is a threshold value.

In Figure 8, x_i is input to the neuron, w_i is the strength of weights with the next neuron, θ is what neuron has as the threshold. In the following, if there is not any notice, x_i and w_i and θ will be defined like this. Then the output of a neuron is:

$$y = \theta'(\sum_i w_i x_i - \theta) \tag{5}$$

where y is output and θ is *Heaviside*'s step function. Since w_i is the trainable parameter, NN can make complex decisions if we can adjust successfully. In general, if we stack many neuron layers then NN can perform to express complicity. To understand the general architecture of NN, see Figure 9

The leftmost layer is the input layer which is fed some input data. The rightmost layer is the output layer which literally outputs data. The middle layers are called hidden layers. Now let us pay attention to the j-th unit in the l-th layer. Defining summation of all input values as u, threshold function which is called activation function as f, trainable bias each unit has as b, output value should be:

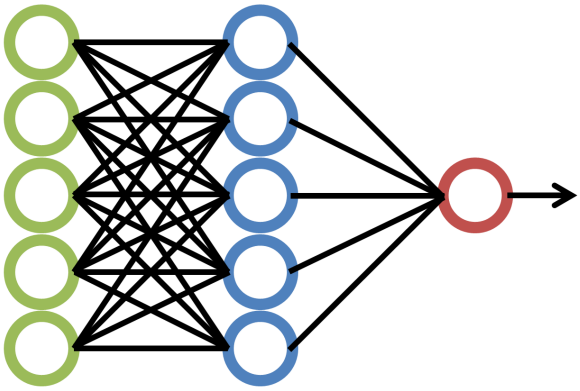

Figure 9: Neural Networks. Green nodes are the input layer. Blue nodes are the middle layer. Red nodes are the output layer.

$$z_j^{(l)} = f^{(l)}(u_j^{(l)} + b_j^{(l)}) = f^{(l)}(\sum_i w_{ji}^{(l)} z_i^{(l-1)} + b_j^{(l)}) \tag{6}$$

where w_{ji} means weight value between unit i and j, in addition to it I sum up about index i because this is index of unit in front layer. And I assume that all activation functions in l-th layer $f^{(l)}$ are the same at all.

Since as for the other output values $z_k^{(l)}$ we can write down similarly, we can confirm in vector form:

$$\boldsymbol{u}^{(l)} = \boldsymbol{W}^{(l)} \boldsymbol{z}^{(l-1)} \tag{7}$$

$$\boldsymbol{z}^{(l)} = f^{(l)}(\boldsymbol{u}^{(l)} + \boldsymbol{b}^{(l)}) \tag{8}$$

where activation function $f^{(l)}$ operates individually elements of vectors. Here if I assumes that I add unit $z_0^{(l)} = 1$ which always outputs 1 as 0-th unit in l-th layer and $w_{j0}^{(l)} = b_j^{(l)}$, then we can write more clearly:

$$\boldsymbol{z}^{(l)} = f^{(l)}(\boldsymbol{u}^{(l)}) \tag{9}$$

In the following I adopt this form having bias. Now we can write down recurrently

$$\boldsymbol{y} \equiv \boldsymbol{z}^{(L)} = f^{(L)}(\boldsymbol{W}^{(L)} f^{(L-1)}(\boldsymbol{W}^{(L-1)} f^{(L-2)} \dots \boldsymbol{W}^{(2)} f^{(1)}(\boldsymbol{x}))) \tag{10}$$

from Equation (7) and Equation (9).

In general, It is known that performance will be improved if we grow the number of hidden layers. This is why we call "Deep" Neural Network.

I mentioned about feed-forward Neural Network which passes signals forwardly above. There are also other kinds of NN, for example, recurrent ones, but I omit them in this thesis.

3.3 Activation Function

As I mentioned above, Neural Networks imitate biological neuron networks. Neuron has a threshold to make whether it is activated and output to the next neuron or not. In Neural Network, the threshold is called the activation function. We have some kinds of functions as activation functions in the neural network. To know some activation functions, see Figure 11. The activation function mainly used in our models is the ReLU function. There are many other activation functions [2].

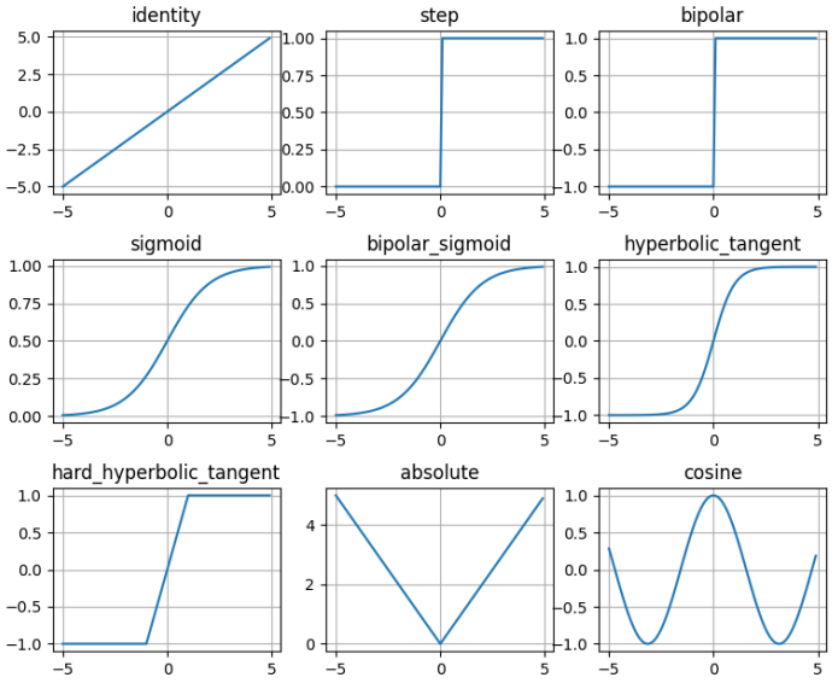

Figure 10: Classic Activation functions [2]

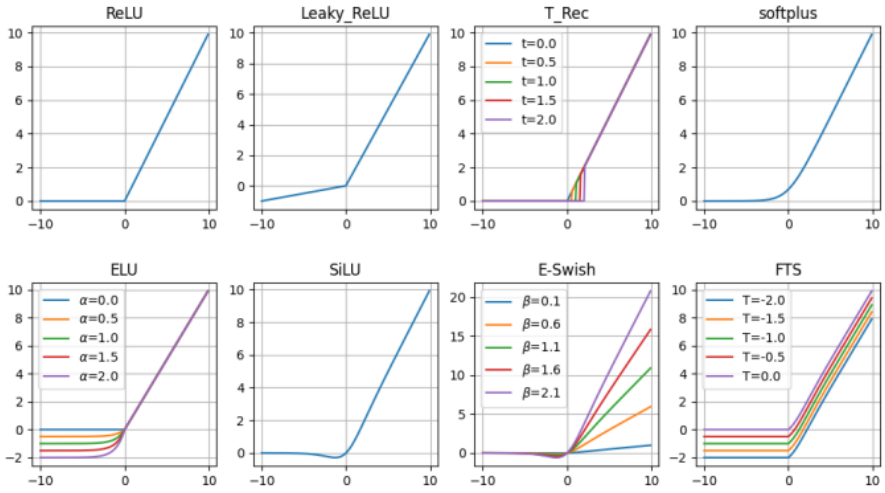

Figure 11: ReLU functions [2]

3.4 Learning Algorithms

Deep Learning models can be trainable when we implement them with trainable parameters and loss functions. We generally need to label data and input data which generates output from NN to calculate the loss. There are many Deep Learning models, learning ways, and loss functions in the computer science field. But in this thesis, I concentrate on what I often use for our study in the following.

3.4.1 Loss and Metric

Deep Learning models are trained to try to minimize loss value with any learning algorithms. Here I mention about loss function, and metric function to see the training status.

We want to make binary classification models that tell us "which hits are what really conversion-electron left". So, we chose binary cross entropy loss (called BCEloss) as the loss function:

$$BCEloss = \frac{-1}{N}\sum_{n=1}^{\infty}\{y_n \log x_n + (1-y_n)\log(1-x_n)\} \tag{11}$$

where N is the number of pixels, x_n is output from model to n-th pixel, y_n is n-th pixel of label image. We calculate this loss function with input-output-image pair for our study. We call this learning way which needs both input and output data $SupervisedLearning$ in computer science.

And we chose Intersection over Union(called IoU; also known as Jaccard Index) as the metric function:

$$IoU = \frac{TP}{TP+FP+FN} \tag{12}$$

where TP, FP, FN are the number of true positive, false positive, and false negative hits. Hence, NN was trained following the binary cross-entropy loss and we can monitor the network model by IoU. It's called a learning curve which has the values of loss and metric. (See Figure 12) We can choose the best-tuned parameters of the network by monitoring IoU. In training, we use Adam as an optimizer. We run 50 epochs on each dataset, and then we chose the best-tuned model.

By the way, learning way of Deep Learning model is ideally minimization of loss function:

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\mathrm{argmin}} L(\boldsymbol{w}) \tag{13}$$

where $\boldsymbol{w}$ is trainable weights and L means loss function. To achieve this, more pragmatically we rely on Gradient Descent. Gradient Descent literally means:

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + \Delta\boldsymbol{w}^{(\boldsymbol{t})} \tag{14}$$

$$\Delta\boldsymbol{w}^{(\boldsymbol{t})} \equiv -\eta\nabla L(\boldsymbol{w}^{(t)}) \tag{15}$$

where η means the learning rate which should be from 0 to 1. The η adjusts how the gradient should be scaled. This is a kind of hyper-parameter that we cannot decide through training. The (t) or $(t+1)$ means the index of the update iteration. In this way, the Deep Learning model can be trained through the loss function on every update iteration. Deep Learning is based on this way. In the following section, I mention more pragmatic details.

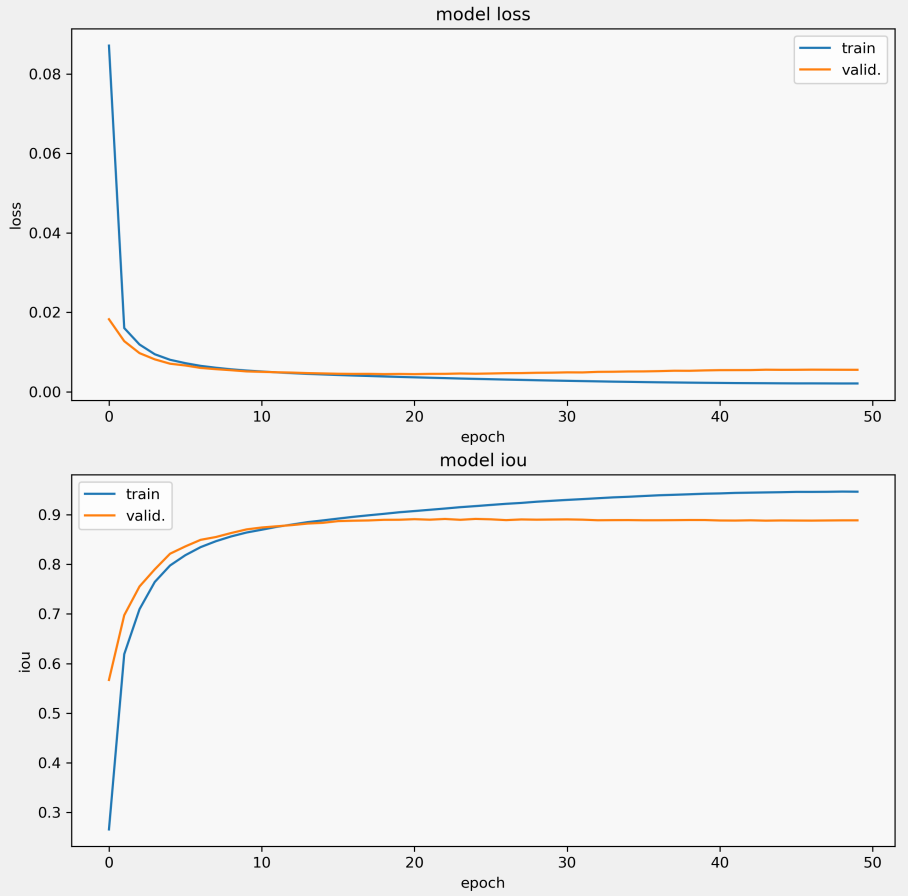

Figure 12: Example of Training curve

3.4.2 Back Propagation

As I mention, we need to calculate $\Delta \boldsymbol{w}^{(t)}$ to train NN models. As we know our models can be smarter when it becomes deeper. So we need a more efficient way to calculate the gradient (Equation (15)), not like individual differentiation to all weight parameters.

Back Propagation did it. Now consider $L(\boldsymbol{w})$ as a loss function. To one weight parameters, using differentiation of composite functions, we get:

$$\frac{\partial L}{\partial w_{ji}^{(l)}} = \frac{\partial L}{\partial u_j^{(l)}} \frac{\partial u_j^{(l)}}{\partial w_{ji}^{(l)}} \tag{16}$$

where $u_j^{(l)}$ is summation of all input values to j-th unit on l-th layer. Here define:

$$\delta_j^{(l)} \equiv \frac{\partial L}{\partial u_j^{(l)}} \tag{17}$$

On the other hand, seeing Equation (9), another factor on the right-hand side should be like the output of the unit. Therefore,

$$\frac{\partial L}{\partial w_{ji}^{(l)}} = \delta_j^{(l)} z_i^{(l-1)} \tag{18}$$

By the way, as for δ we can apply differentiation of composite functions:

$$\delta_j^{(l)} = \frac{\partial L}{\partial u_j^{(l)}} = \sum_{k=0}^{d_{l+1}-1} \frac{\partial L}{\partial u_k^{(l+1)}} \frac{\partial u_k^{(l+1)}}{\partial u_j^{(l)}} \tag{19}$$

whether d_{l+1} is the number of units in $l+1$-th layer. Now we can write

$$u_k^{(l+1)} = \sum_{j'} w_{kj'}^{(l+1)} f(u_{j'}^{(l)}) \tag{20}$$

So, we can get

$$\delta_j^{(l)} = \sum_{k=0}^{d_{l+1}-1} \delta_k^{(l+1)} w_{kj}^{(l+1)} f'(u_j^{(l)}) \tag{21}$$

This equation explicitly provides us δ which passes backwardly. We don't need to calculate differentiation to all weight parameters individually anymore. This saves computation costs.

3.4.3 Adam

In recent research, Adam is often chosen as an optimizer. If without an optimization method like this, we wouldn't get stable and fast training. There are many other optimizers to solve instability in training, e.g. RMSprop, AdaGrad [9] and AdaDelta [10].

Adam is a kind of optimizer. This optimization method aims to use estimates on momentum when it descends on loss function gradient well [11]. To see what "momentum" means, see the first term in Equation (22).

$$\Delta \boldsymbol{w}^{(t)} = \mu \Delta \boldsymbol{w}^{(t-1)} - (1-\mu)\eta \nabla L(\boldsymbol{w}) \tag{22}$$

It just includes the last incremental weight values scaling with μ which is from 0 to 1. It's a momentum factor. By the way, Adam rewrites Equation (22) with the first-order moment and second-order moment. See Equation (23) and Equation (24)

$$m_{i,t} = \rho_1 m_{i,t-1} + (1-\rho_1)\nabla L(\boldsymbol{w}^{(t)})_i \tag{23}$$

$$v_{i,t} = \rho_2 v_{i,t-1} + (1-\rho_2)(\nabla L(\boldsymbol{w}^{(t)})_i)^2 \tag{24}$$

where ρ_1, ρ_2 are hyper-parameters. Both moments are initialized with 0. Solving Equation (24), we can get:

$$v_{i,t} = (1-\rho_2)\sum_{s=1}^{t} (\rho_2)^{(t-s)} (\nabla L(\boldsymbol{w}^{(s)})_i)^2 \tag{25}$$

Since we choose we should consider expectation values:

$$\begin{aligned} E[v_{i,t}] &= (1-\rho_2)\sum_{s=1}^{t} (\rho_2)^{(t-s)} E[(\nabla L(\boldsymbol{w}^{(s)})_i)^2] \\ &\approx E[(\nabla L(\boldsymbol{w}^{(t)})_i)^2](1-\rho_2)\sum_{s=1}^{t} (\rho_2)^{(t-s)} \\ &= E[(\nabla L(\boldsymbol{w}^{(t)})_i)^2](1-\rho_2^t) \end{aligned} \tag{26}$$

The approximation in the second line can be done because $E[v_{i,t}]$ is really stationary or ρ_2 is small enough to neglect contribution from too far in the past. Hence, we can adapt

$$\hat{v}_{i,t} = \frac{v_{i,t}}{(1-\rho_2^t)} \tag{27}$$

As for the first-order moment, something similar can be done. So

$$\hat{m}_{i,t} = \frac{m_{i,t}}{(1-\rho_1^t)} \tag{28}$$

Therefore we can adopt Equation (29) as incremental weight values:

$$\Delta \boldsymbol{w}_i^{(t)} = -\eta \frac{\hat{m}_{i,t}}{\sqrt{\hat{v}_{i,t} + \epsilon}} \tag{29}$$

This is Adam.

3.4.4 OneCycleLR

As I mentioned above, Neural Network models need a learning rate while it's trained but we cannot decide it through training. In this section, I mention how it can be dealt with. How the learning rate should be decided was talked about in the early epoch of Deep Learning. Many ways were proposed. But I use OneCycleLR.

OneCycleLR was invented by observing that a big learning rate helped to train in a much smaller time (This phenomenon is named "super-convergence".) [12]. But if we have a big learning rate constantly, we would've seen an oscillation of loss value at the end of the learning iteration. So OneCycleLR adapts to decreasing learning rate as it proceeds. To look at the learning rate curve for our study, see Figure 13

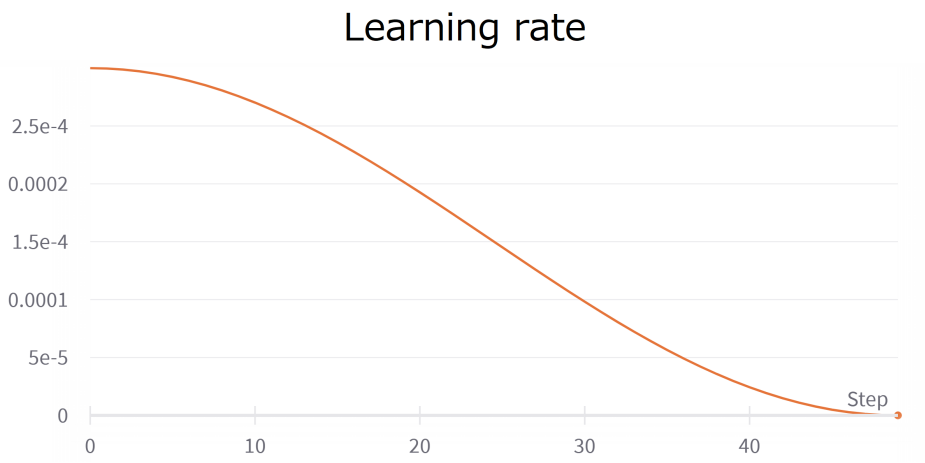

Figure 13: Learning rate example when I adopt OneCycleLR. The horizontal Axis means epoch number.

3.5 Convolutional Neural Network

I already mentioned the basis of the Neural Network above. In this section, let me explain especially how the Neural Network works with images as input and output data successfully. Most of the tips are based on the visual cortex. In the following subsections, I'll explain how it is implemented mathematically in Deep Learning algorithms.

3.5.1 Convolution layer

As I mentioned, in general, the Deep Learning model has so many layers. To imitate the visual cortex, the convolution layer was invented. There are many variants [13]. The convolution layer aggregates pixels using trainable filters also known as convolution filters and kernel. To understand how it works, see Figure 14

In this figure, the blue grid means the input image. The yellow grid means the convolution layer which actually has floating values, not like integers. Changing the position of the kernel, the convolution layer sums pixels in the kernel multiplying trainable parameters. Depending on trainable parameters, we can detect "line", "curve" or other features on the image.

We call models which have a Convolution layer Convolutional Neural Networks. In general, when we use the convolution layer, we increase the number of channels though the number is basically, initially 3 which means red, green, and blue. It corresponds to increasing the number of feature maps simultaneously. This means $WeightSharing$. In addition to it, usually, our kernel size of convolution is limited to smaller than the original image size like Figure 14. This helps us to limit weighted connections and not to be dense. This means a reduction of computation cost.

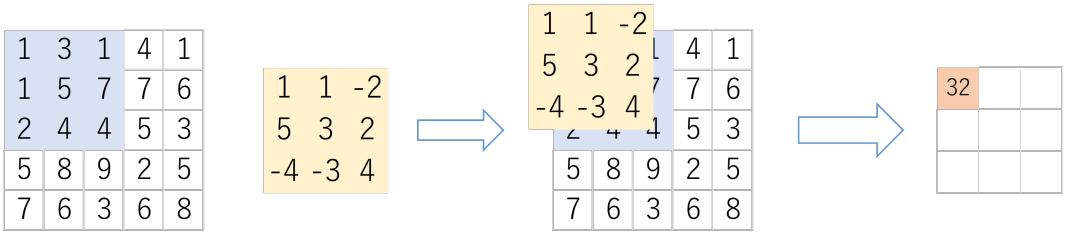

Figure 14: How Convolution filter works. Blue grid means input images. Yellow grid means convolution filter. Convolution operation means summing up following convolution filter

3.5.2 Pooling layer

The pooling layer doesn't have any trainable parameters. This pooling layer aggregates pixel intensities in the pooling kernel. There are some aggregation ways but I introduce 2 basic pooling operations here.

- Max Pooling: summing over neighborhood pixels in the kernel.
- Average Pooling: averaging over neighborhood pixels in the kernel.

There are many variants [14]. Unlike the convolution layer, it doesn't need any trainable parameters.

3.6 Models

In this subsection, I will show you some models which work with images for image processing.

3.6.1 Denoising Convolutional Neural Network

Denoising Convolutional Neural Network called DnCNN is oriented to remove some stains from images [15]. DnCNN has simpler architecture than following some image processing models. You can see the architecture in Figure 15. The "Conv2d" means convolutional layer, which has trainable parameters to work with 2d-like inputs. ReLU means a layer that has ReLU functions as activation functions. Sigmoid means a layer that has sigmoid functions as activation functions.

DnCNN Blocks has three kinds of layers which are the convolution layer and Batch-Normalization layer [16] and ReLU functions. We chose 17 as the number of stacked DnCNN Blocks. Especially we call this DnCNN in this thesis.

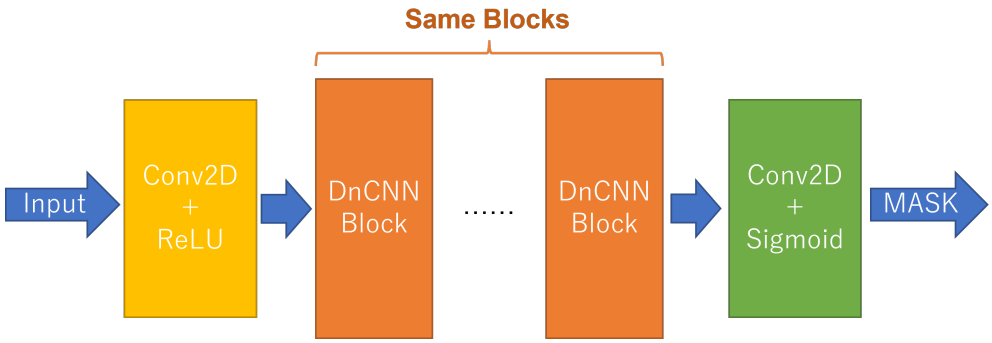

Figure 15: DnCNN architecture

The output of the sigmoid function has range from 0 to 1. So we can consider it as the probability to be true electron signals.

3.6.2 Unet

Unet [3] was developed for image segmentation tasks. Image segmentation tasks are mainly divided into the following 3 sections which are instance segmentation, semantic segmentation, and panoptic segmentation [17]. The concept of this network is "We should have clearly divided, several feature maps which have scale dependency". As we know, most convolutions reduce image size unless you take some specific operation such as padding, stride, etc. The output feature map should be smaller than the input one. Anyway, to retain different size feature maps, unet has $skip - connection$. (See Figure 16)

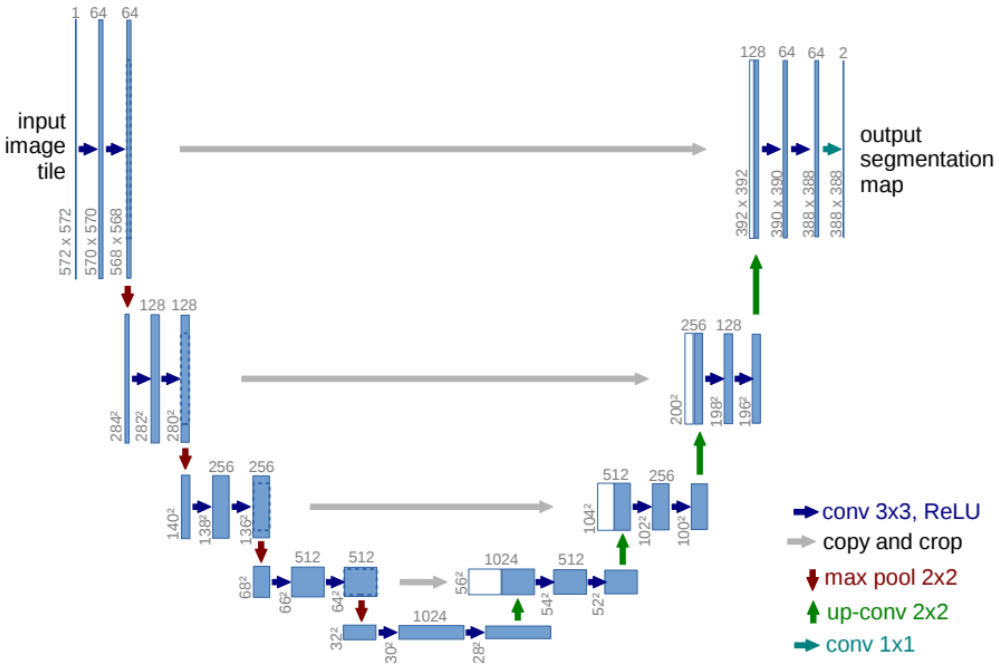

Figure 16: Unet architecture [3]

In addition to these plain Unet models, there is Unet++ which has more adaptive feature maps [4]. To understand how Unet evolved to Unet++, see Figure 17 displaying the difference between plain Unet models and Unet++. Red, Green, and blue mean new architecture.

The main idea to invent Unet++ is training becomes easier when feature maps are semantically modified to be concatenated. To achieve it, just skip connection in Unet is replaced with a dense connection in Unet++.

In this thesis, I combine it with spatial & channel SE modules proposed in [18].

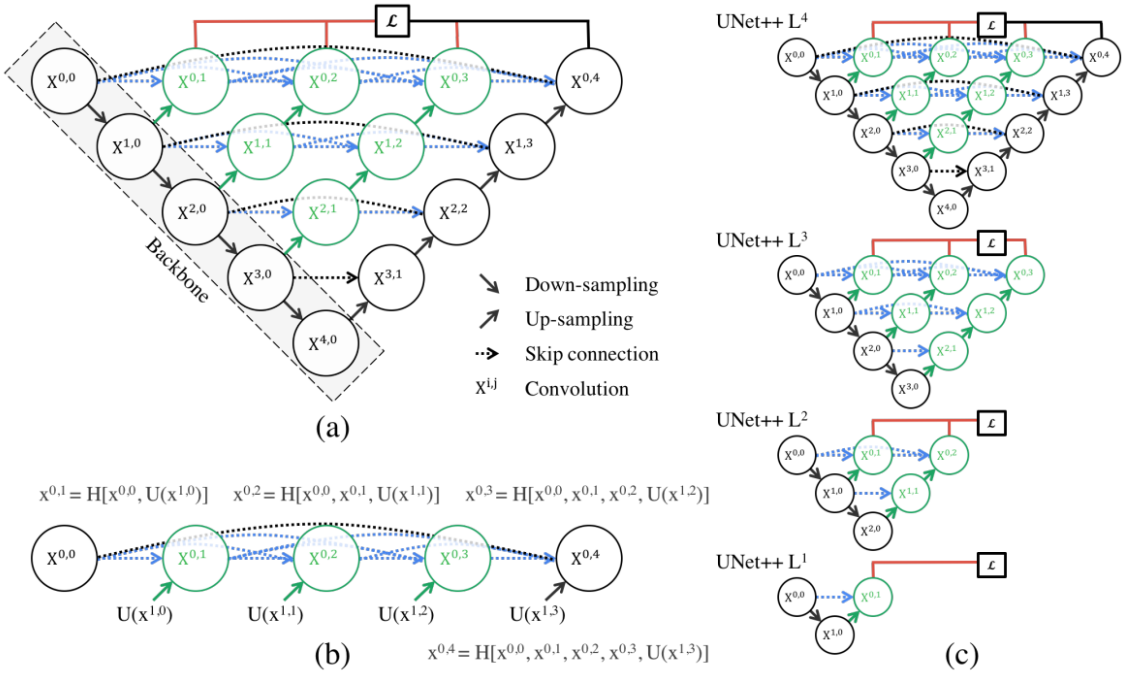

Figure 17: Especially (a), How Evolution from U-Net to UNet++. [4]

3.6.3 Feature Pyramid Network

Feature Pyramid Network called FPN is also a segmentation model[19]. As for the architecture, see Figure 18. FPN has a lot of convolution layers and skip-connection like Unet between encoder and decoder architecture. Now encoder-decoder architecture is also famous for transformer model [20] not only image segmentation models.

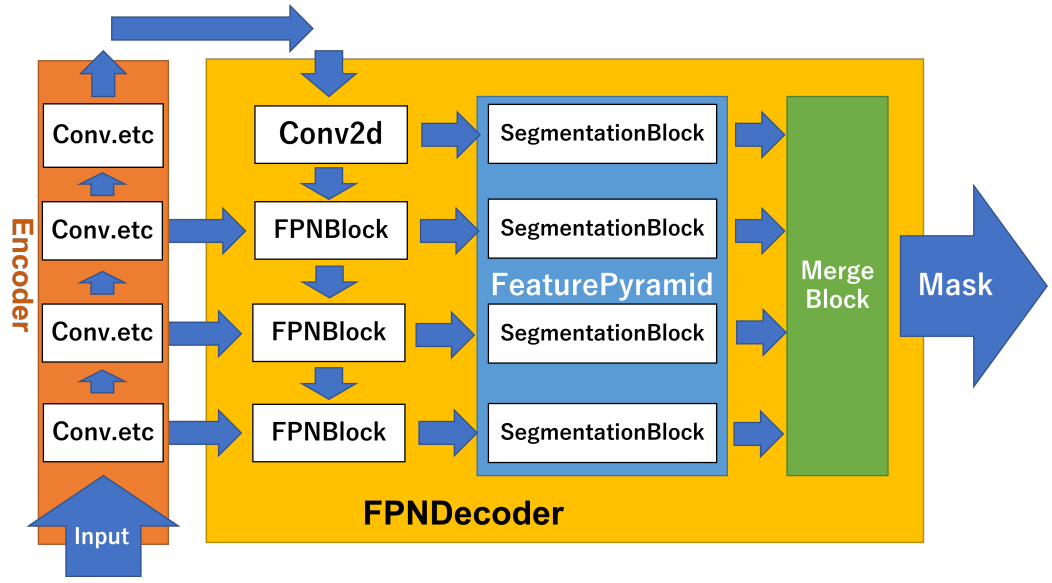

Figure 18: FPN architecture

FPN has an encoder(called backbone) part and a decoder part. The encoder part should be something like convolving input image several times, contracting image size, and outputting to the decoder part. (We have so many options for the encoder part. e.g. Resnet, Efficientnet, etc.) So, the

size of the encoder output is varied. Data flow coming from the encoder part is passing through the convolution layer and FPNBlock which contains the convolution layer and extending layer. And then, SegmentationBlock makes the sizes of data flow the same. MergeBlock literally merges four data-flow to make a segmentation mask.

FPN was invented to predict multi-scale objects in images, unlike Unet.

3.6.4 DeepLabV3+

DeepLabV3 is based on DeepLab which has atrous convolution also known as dilated convolution [21]. Atrous convolution is a variant of convolution. Usually, the convolution layer has a dense convolution filter that convolves some adjacent pixels. But the atrous convolution layer has a sparse convolution filter that convolves pixels separated from each other. Iteration of atrous convolutions can keep image size relatively bigger [21]. DeepLabV3+ operates multiple atrous convolutions parallelly in Atrous Spatial Pyramid Pooling called ASPP module [22]. Because of these modules, models can segment some objects at multiple scales and can have richer feature maps. In Figure 19, the ASPP module is at the top of the encoder. There are atrous convolutions that have different dilation rates in them. After that, the output from the ASPP module will be concatenated with another output from the DCNN encoder.

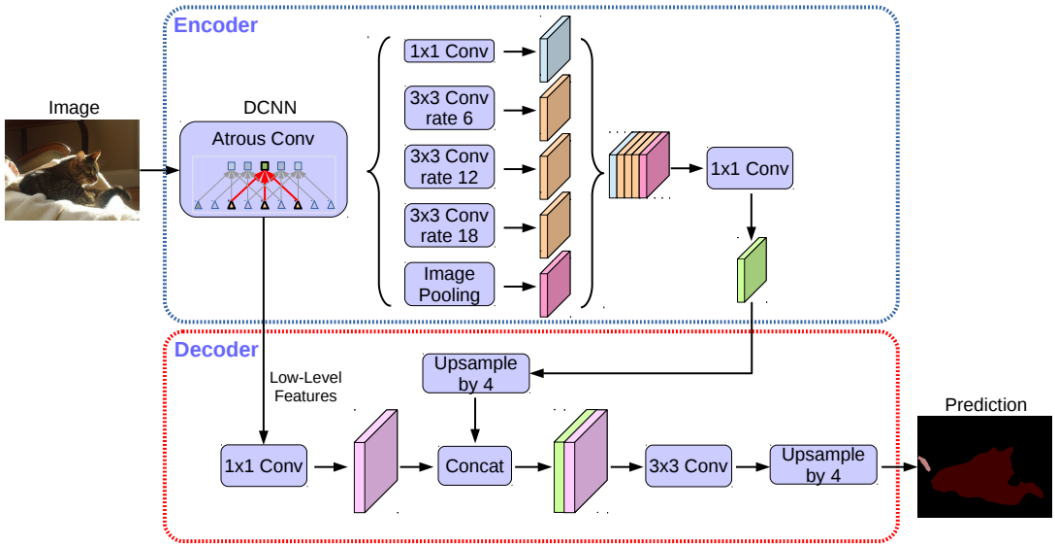

Figure 19: The architecture of DeepLabV3Plus [5]

4 Evaluation way

In this section, I mention about evaluation way of models.

4.0.1 ROC curves

First, when we once decide threshold value, we can calculate the numbers of true positive(TP), false positive(FP), true negative(TN), and false negative(FN) on each prediction event. Second, we combine them to get Retention (also known as Recall and Efficiency), Rejection, and Purity (also known as Precision):

$$Efficiency = \frac{TP}{TP + FN} \tag{30}$$

$$Rejection = \frac{TN}{TN + FP} \tag{31}$$

$$Purity = \frac{TP}{TP + FP} \tag{32}$$

These 3 values can be varied with changing threshold followed by an output layer. Since we don't know the best threshold value, we don't have any other choice than we consider all possible threshold values to evaluate the model trained. So we plot 3 values in many cases, gradually changing the threshold values. This is *ReceiverOperatingCharacteristic* curve called the Rejection-Efficiency curve and Purity-Efficiency curve. To know the shape of the regular curves as an example, see Figure 21

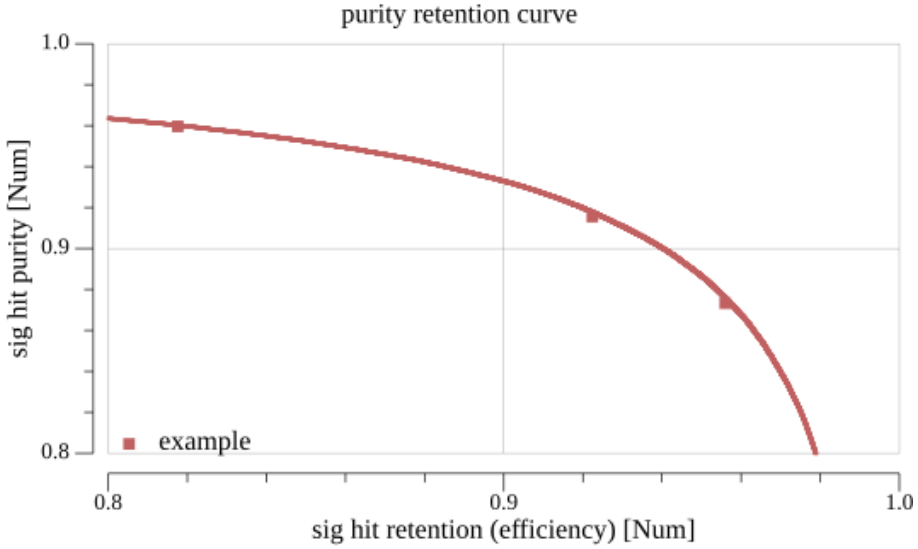

Figure 20: Purity Efficiency curve as an example. Relatively large square plots mean when threshold values are 0.3, 0.6, 0.9.

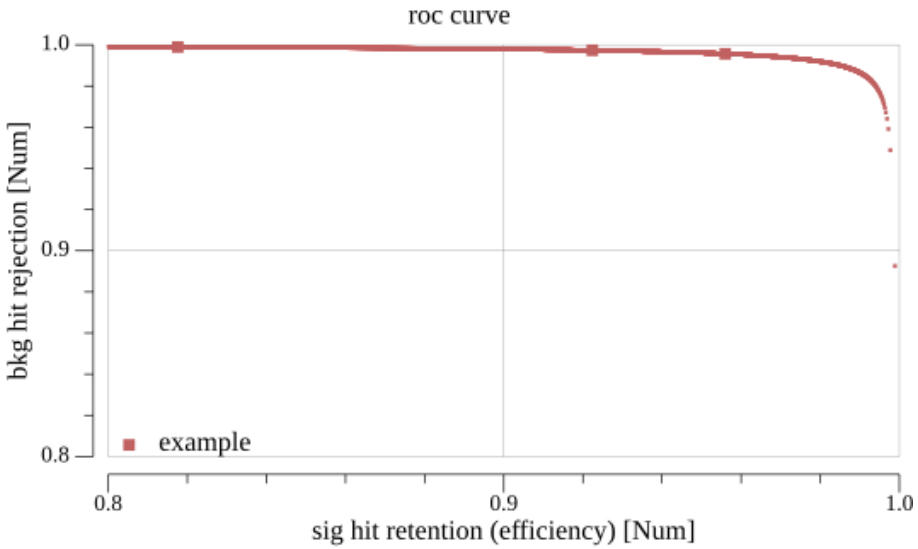

Figure 21: Rejection Efficiency curve as an example. Relatively large square plots mean when threshold values are 0.3, 0.6, 0.9.

Obviously, if this curve is likely to be square, then it means that the model was trained successfully.

5 Application of DL in COMET experiment

In this chapter, I will show you how to apply Deep Neural Networks to our COMET experiment.

I use PyTorch [23] as a deep learning library.

5.1 Simulation Data

COMET experiment is already started but it's incomplete. So I need to make datasets from simulation data, not real ones. The full simulation requires us a huge amount of time to simulate all processes from proton beam simulation to charged particle in CDC simulation. So I use events which has conversion-electron hits(We also call these just "signal hits") simulated in CDC and unreasonable noise hits randomly generated on each CDC cell. When I generate noise hits, signal hits could be shadowed because of their time information. To see an example, See Figure 22

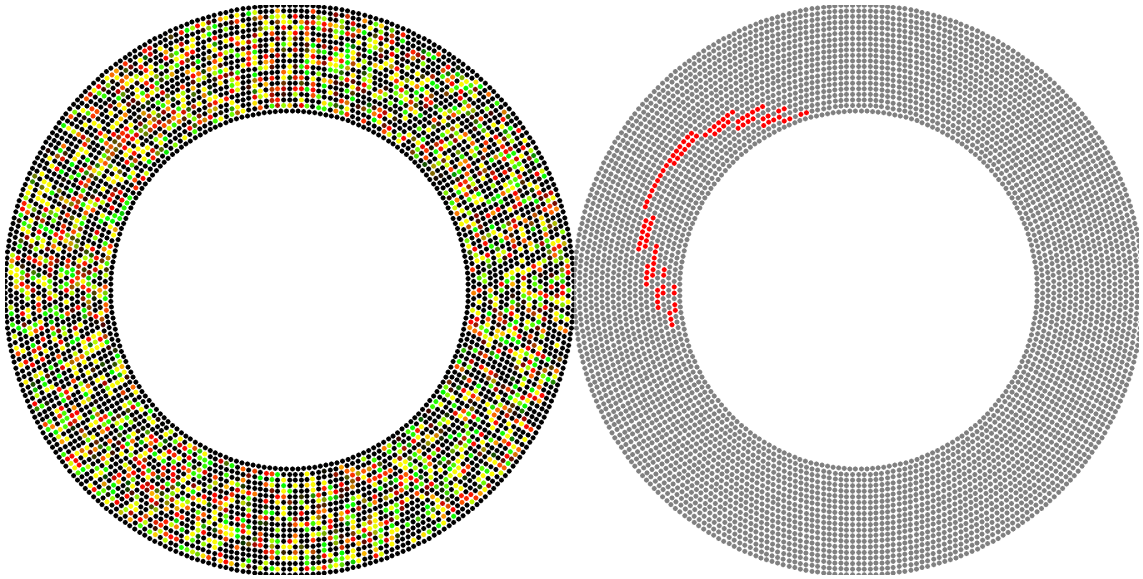

Figure 22: Left: Cells colored by hit information of representative hits.
Right: Cells that contain hits of conversion-electron colored with red.

We can get ADC SUM[6] which we can get from a transformation of energy deposit, and time information which corresponds to Equation (4), and delta phi which is the azimuthal angle between the CTH trigger point and each cell from the simulation.

To look at the distributions of Time information, ADC SUM[6] and delta phi, see Figure 23, Figure 24 and Figure 25.

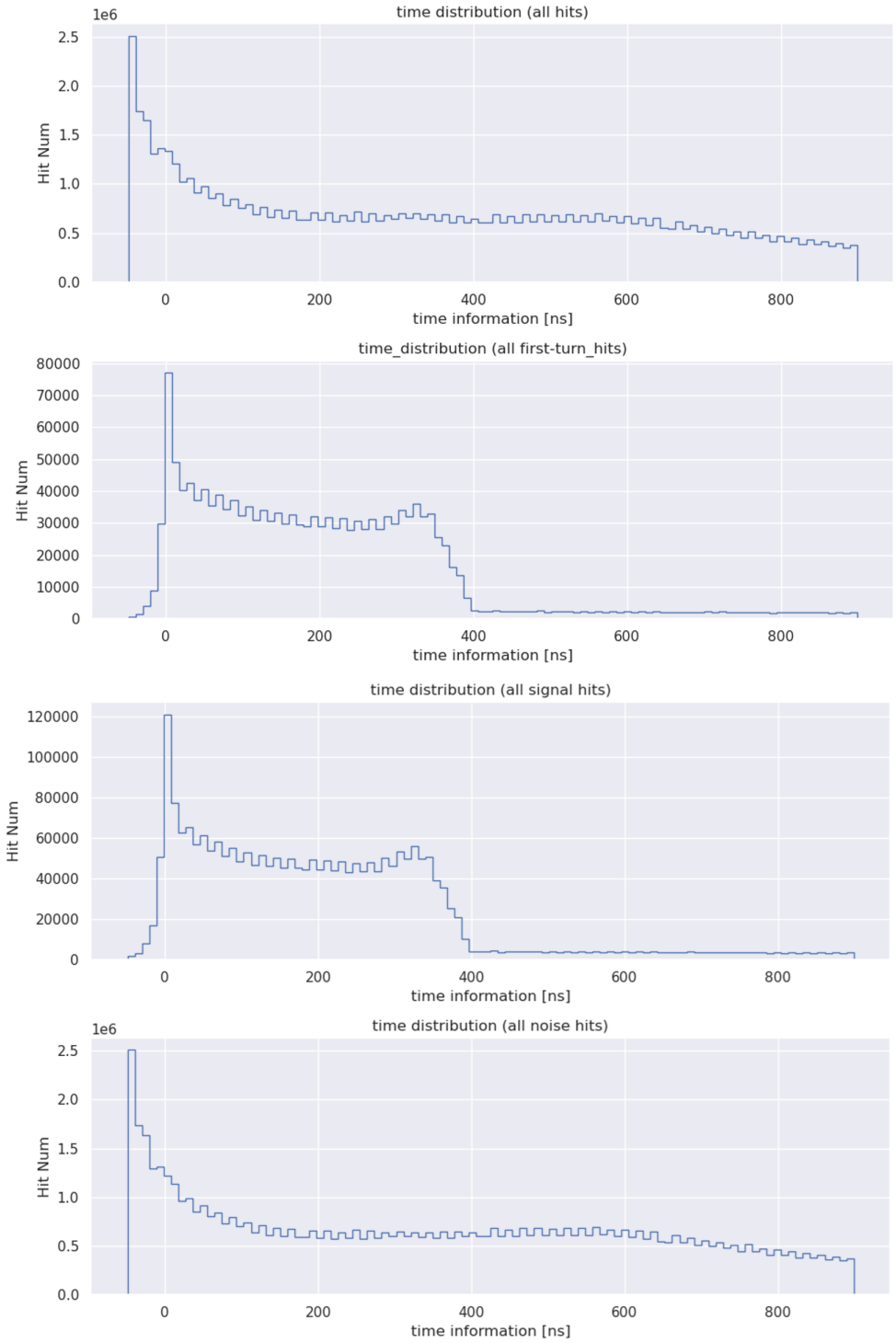

Figure 23: Distributions of time information on 30000 events. The 1st figure means the distribution of all hits. The 2nd figure means the distribution of first-turn hits. The 3rd figure means the distribution of all conversion-electron hits. The 4th figure means the distribution of noise hits.

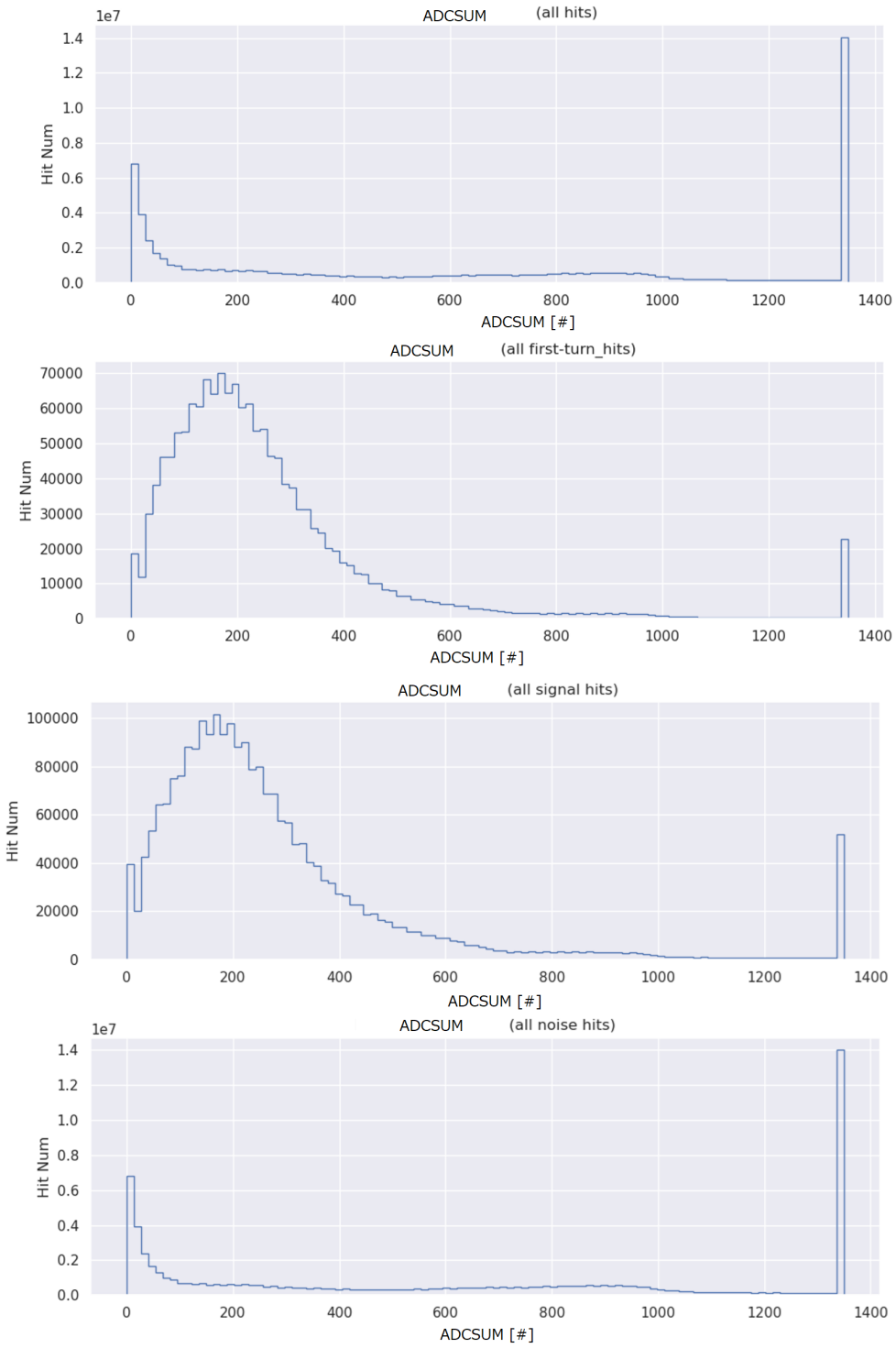

Figure 24: Distributions of ADCSUM [6] on 30000 events. The 1st figure means the distribution of all hits. The 2nd figure means the distribution of first-turn hits. The 3rd figure means the distribution of all conversion-electron hits. The 4th figure means the distribution of noise hits.

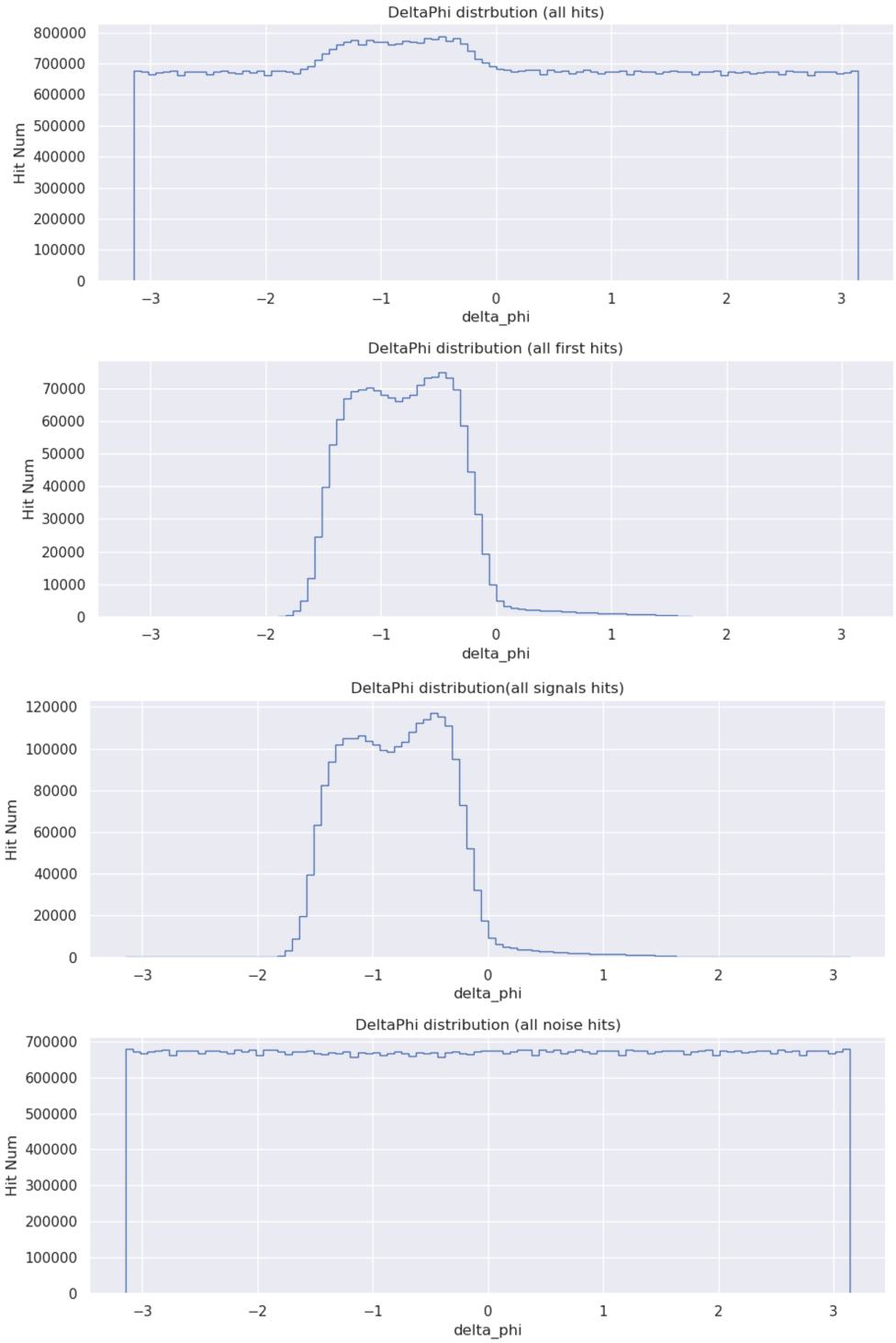

Figure 25: Distributions of delta phi on 30000 events. The 1st figure means the distribution of all hits. The 2nd figure means the distribution of the first hits on each cell. The 3rd figure means the distribution of all conversion-electron hits. The 4th figure means the distribution of noise hits.

5.2 Making Image Datasets

As you know, I applied Deep Convolutional Neural Network (called DCNN) to this analysis in the COMET experiment. DCNN must be fed image, not raw binary data. Hence I needed to transform raw binary simulation data to input and output image pairs to train DCNN models event by event.

In addition to this transformation, I need to pre-process the input and output data to make input-label image pairs. All the processes are:

1. Read binary data which is simulation data.
2. Make label image.
3. Choose 1 hit on each CDC cell to make an input image.
4. Scale energy deposit, time information, and delta phi for pixel color values.
5. Making square wire image, not an x-y image.

As for making a label image, we need a label image that tells us whether the signal or contained or not. Here I assume that we need to care only if each cell contains conversion-electron hits or not, to get enough energy-momentum estimation.

5.2.1 Translation from Hit-level to Cell-level

As for choosing hits, we need to make an input image that has information from CDC cells. In my analysis scheme, I assigned each cell to each pixel, that is one on one, of images for DCNN. I chose 1 hit on each cell to assign to a pixel on an image because one pixel can have only 3 values as RGB in an image, not an infinite number of values, though there can be multiple hits on each cell in simulation data. Here I had 3 ways to select the hit on each cell:

- Take first hit based on time information of hits.
- Take hit which has highest score predicted by 2d-histogram is mentioned in Appendix A.
- Take hit which is one hit of 1st turn track based on Monte-Carlo information

By the way, RGB values on each pixel correspond to:

1. First channel (Red) : Time information of the hit
2. Second channel (Green) : ADCSUM [6] of the cell
3. Third channel (Blue) : Delta phi which is the azimuthal angle between the cell and CTH trigger

5.2.2 Scaling scheme

As for the scaling, I need to transform the 3 kinds of values to be in the range $[5, 250]$ to be assigned in images (By the way ranges [0.5), (250.255] are not used.). To achieve this, I have a combination of linear scaling and overflow bin about Time Information and ADC SUM [6]. See Figure 23 and Figure 24.

In the simulation, most of the conversion-electron hits are in the range (-50.0 nsec, 400 nsec) of time information and range (0, 700) of ADC SUM. So I applied just linear scaling the hits in such range on time information and ADC SUM and I made overflow bin (See Section 5.2.2) about the others.

Table 3: Scaling Scheme

Scaling Scheme	Time Info to color	ADC SUM to color
just Linear Scale	$[-50$ ns, 900 ns$] \mapsto [5, 250]$	$[0, 1350] \mapsto [5, 250]$
Limit Scale	$[-50$ ns, 400 ns$] \mapsto [5, 250]$ (400 ns, 900 ns$] \mapsto 250$	$[0, 700] \mapsto [5, 250]$ $(700, 1350] \mapsto 250$

On the other hand, delta phi was scaled with just linear scaling. See the 4th plot in Figure 25 to see distribution.

5.2.3 Reduction of image-size

As for the making square wire image, to reduce image size, I transform though I break geometrical x-y correlation information. See Figure 36.

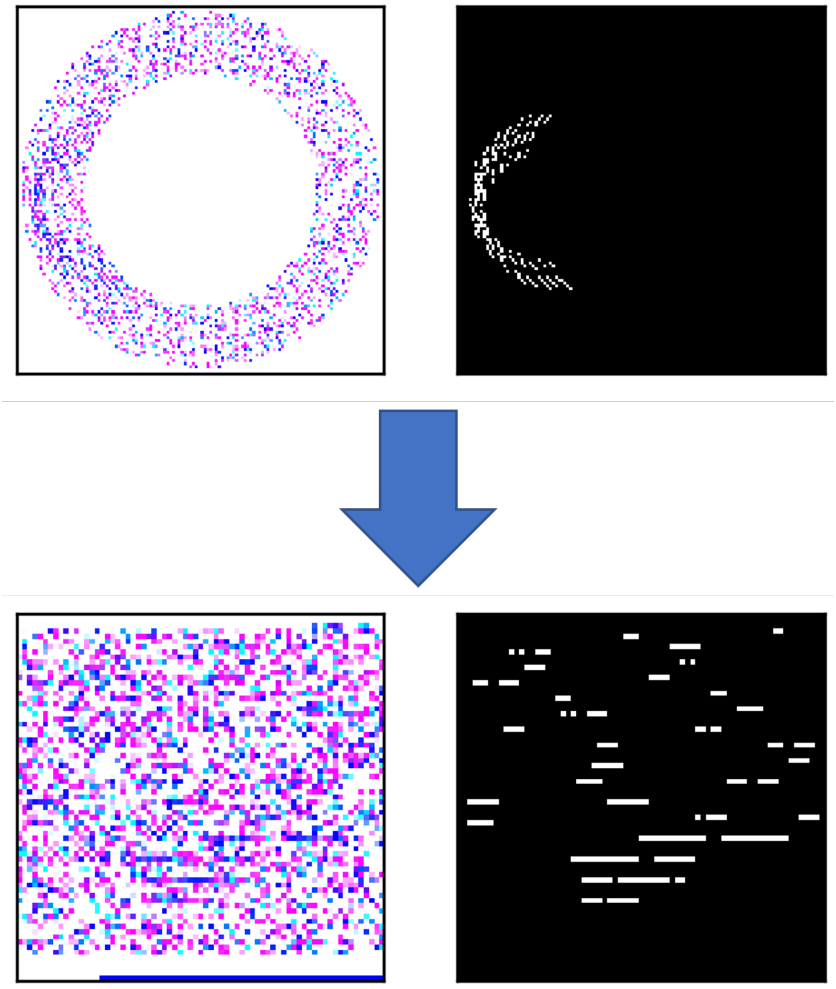

Figure 26: How reduction works.

The left upper image means the shape of the input image before this reduction. Right upper image means label image before this reduction. The left lower image means the shape of the input image after this reduction. The right lower image means label image after this reduction. This operation reduces the data size average from 3.24 kilobytes to 0.8 kilobytes on the input image and 270 bytes to 220 bytes.

6 Result

In the following 4 subsections, there are results for the comparison of:

- Scaling Scheme: just linear scaling, limited scaling mentioned in Section 5.2.2
- Model: DnCNN, Unet++, FPN, DeepLabV3Plus mentioned in Section 3.6
- Training sample size: 40000, 56000, 72000 events
- Ways of translation from Hit-level to Cell-level mentioned in Section 5.2.1

The whole computation was done by a computer with 4 GPUs which are 4 "NVIDIA GeForce GTX 1080 Ti".

6.1 Scaling Scheme Comparison

6.1.1 with score of 2D Histogram

I tried to train FPN and DnCNN with 2 datasets that have 72000 events as training samples. One is made with just linear scale, the other is made with limit scale. Both datasets are based on the score of the 2D Histogram.

For prediction, I used datasets that have other 9000 events. They were made similarly to training samples.

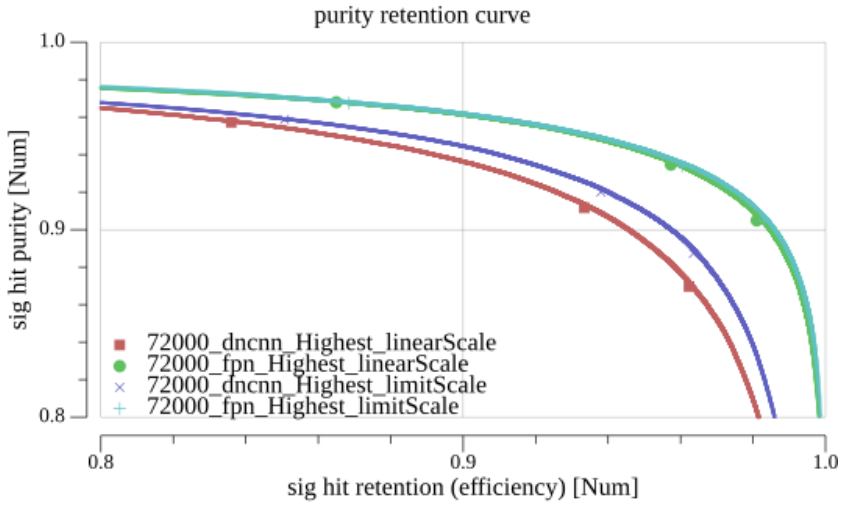

Figure 27: Purity Efficiency curves with scaling schemes.

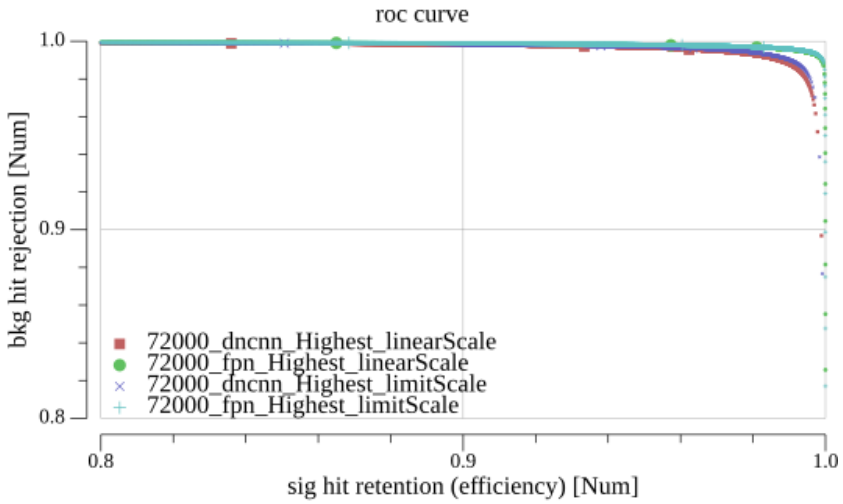

Figure 28: Rejection Efficiency curves with scaling schemes.

6.1.2 with Monte-Carlo Truth information

I tried to train FPN and DnCNN with 2 datasets that have 72000 events as training samples. One is made with just linear scale, the other is made with limit scale. Both datasets are based on Monte-Carlo Truth information.

For prediction, I used datasets that have other 9000 events. They were made similarly to training samples.

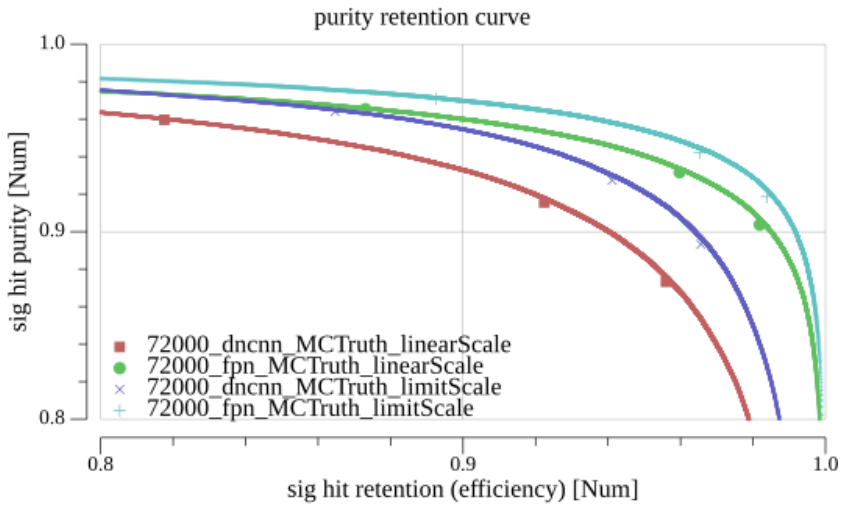

Figure 29: Purity Efficiency curves with scaling schemes.

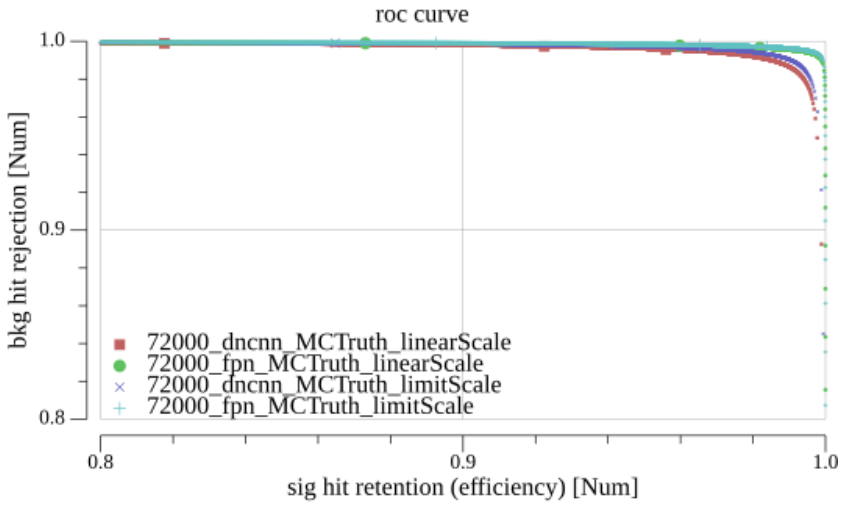

Figure 30: Rejection Efficiency curves with scaling schemes.

6.2 Model Comparison

6.2.1 Summary of Models

Here is a table that summarizes the following models (See Table 4). Of course, we can grow the number of trainable parameters (maybe by the growing depth of models) in models. But for simplicity, we fixed the number of trainable parameters. We calculated on 4 GPUs (4 "NVIDIA GeForce GTX 1080 Ti") for all models. We used a mini-batch including 32 events (images). So it means that 32 events were processed in one iteration.

Table 4: Summary of models

Model	#Trainable parameters	Training Speed [iter/s]	Evaluating Speed [iter/s]
DnCNN	556608	21.98	45.95
Unet++ (with SCSE)	10478611	1.94	5.36
FPN	9464931	3.93	9.01
DeepLabV3+	8642739	0.7692	0.8474

I tried to train 4 models mentioned in Section 3.6. For training, we used one dataset which has 40000 events. For prediction, we used 5000 events. All events are based on just taking the first hit by referring to Time Information.

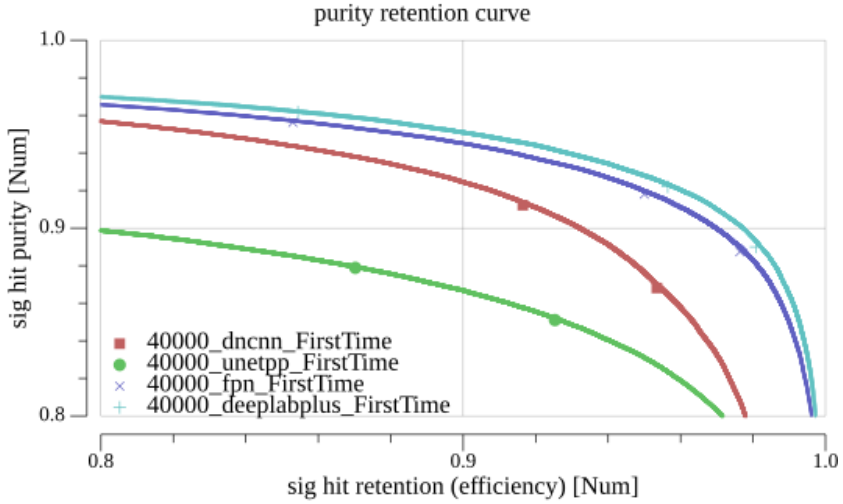

Figure 31: Purity Efficiency curves with various models.

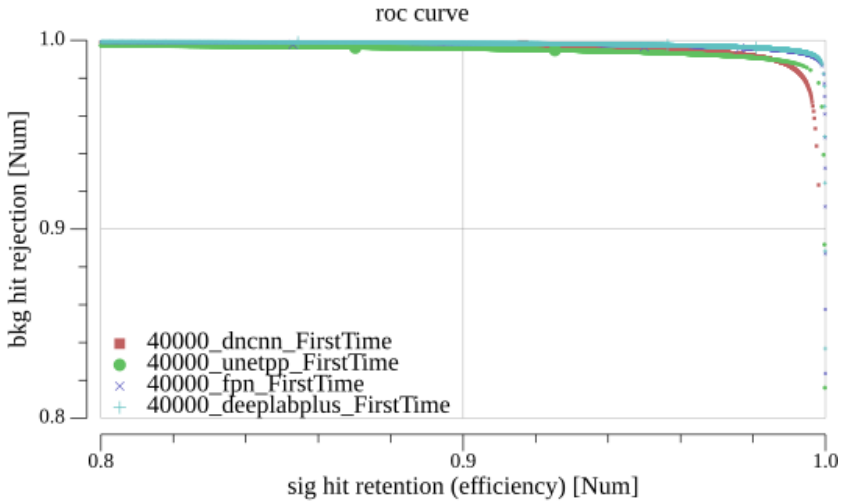

Figure 32: Rejection Efficiency curves with various models.

6.3 Sample size Comparison

I tried to train FPN and DnCNN with 3 datasets. Each dataset contains 40000, 56000, and 72000 events and is based on Monte-Carlo Truth information to translate from Hit-level to Cell-level.

For prediction, I used datasets that have other 5000 events. They were made similarly to training samples.

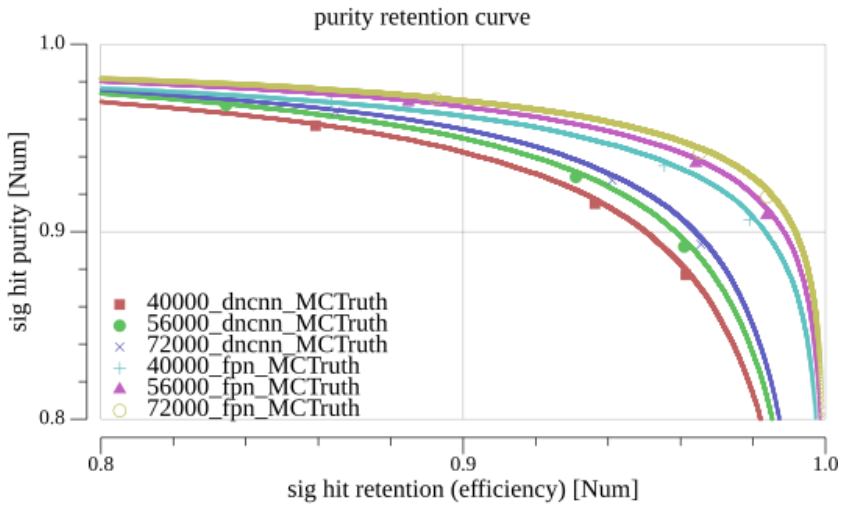

Figure 33: Purity Efficiency curves with various training sample sizes.

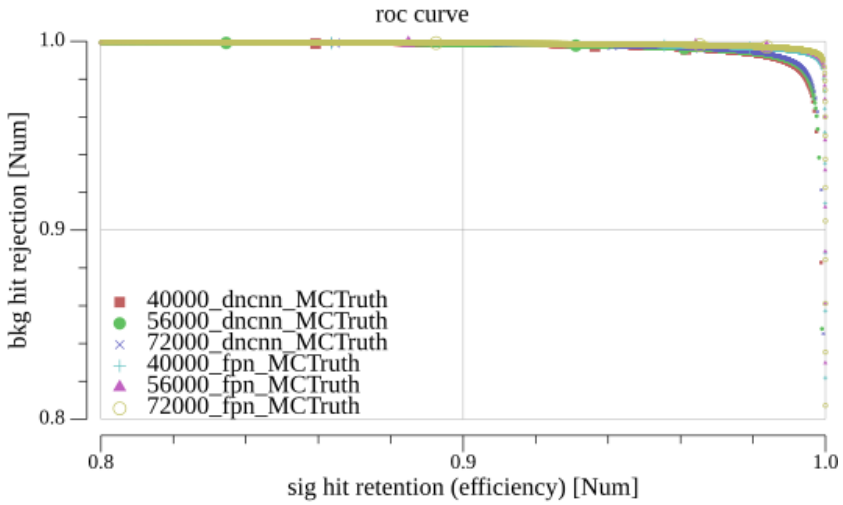

Figure 34: Rejection Efficiency curves with various training sample sizes.

6.4 Translation Comparison

I tried to train FPN and DnCNN with 3 datasets that contain 72000 events as training samples. The datasets were made by 3 ways to translate from Hit-level to Cell-level as mentioned in Section 5.2.1. After training, I made them predict every 3 datasets which have 9000 events.

Gray and Black curves are more special than other curves because the two curves were drawn when I use different translation methods between training samples and prediction samples. I use a dataset based on Monte-Carlo information as training samples but a dataset based on 2D-Histogram scores as prediction samples. It's because, actually, in real experiment, we can't use Monte-Carlo information as prediction samples.

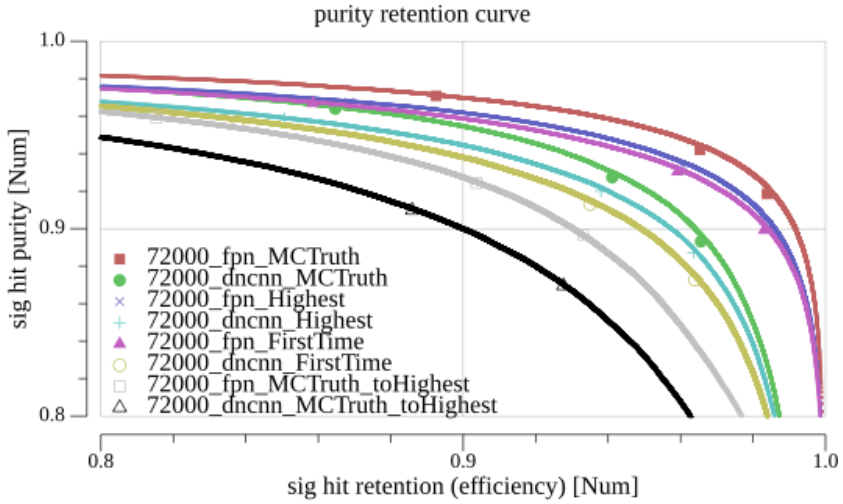

Figure 35: Purity Efficiency curves with various translations from Hit-level to Cell-level.

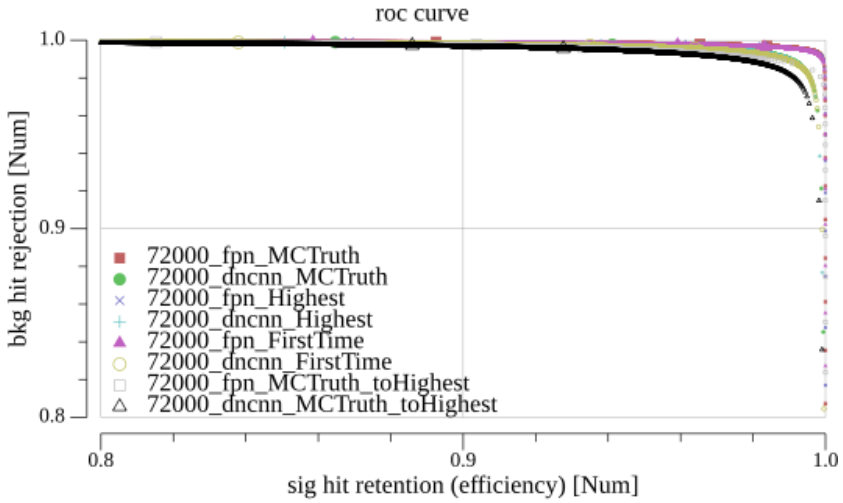

Figure 36: Rejection Efficiency curves with various translations from Hit-level to Cell-level.

7 Summary

In the results, I got the ROC curves about 4 comparisons.

- As for the scaling scheme, we should use overflow-bin by referring distribution of ADC SUM and Time Information. It has a (bit) contribution to ROC curves. Especially when we use Monte Carlo information, the overflow bin works well. Presumably, it's because most of the hits in the overflow bin are exactly noise hits. It would help NN models to learn the pattern. But when we use 2D-Histgram, some signal hits would be in overflow-bin unfortunately. It would confuse NN models.
- As for models, we can't strongly conclude something. We can't say that many trainable parameters have a better contribution. Obviously, DeepLabV3+ has the best ROC curves in both cases but its processing speed is definitely slow. On the other hand, Unet++ has the worst Purity Efficiency curve though it has the most trainable parameters. We need to keep investigating.
- As for sample size, we can't see what training sample size is the best. But we see the usual tendency which is "If we have more training samples, then performance will improve.".

- As for translation, basically we can think that it is easier and more efficient for NN to understand the pattern since using Monte-Carlo information or the score of a 2d-histogram can keep what conversion-electron left than just taking the first hit by referring to Time Information.

Finally, let me emphasize this again. In this thesis, Cell-level analysis was done with events that have simulated signal hits and randomly generated noisy hits.

7.1 Future work

Obviously, we have next or future works. There is something like this:

- Hit-level analysis.
- Threshold optimization.
- Training with reasonable noisy events.
- Sim-to-Real Domain Adaptation.
- Energy Momentum Estimation by NN.

In this thesis, Cell-level analysis was done. But actually, we need to label each hit to something, not each cell for energy-momentum estimation. In my analysis scheme, we make each CDC cells correspond to pixels in images. If we want to achieve hit-level analysis, it is straightforward to grow the number of channels in each image. But it would drastically grow computation cost though we just want to consider additional few hits. It's because we need another channel to assign time information of hits in addition to one hit. So we need a more efficient way.

To tackle this problem, I imagine that Graph Neural Network [7] would solve this problem and geometry breaking because of its flexibility (See Appendix B).

For now, we evaluate NN models with varying the threshold of the output layer in NN models. Pragmatically, for classification, we need to decide the threshold value for each event. This problem should be solved in the future.

Now the COMET experiment is going to be just begun. In the early phase, there should be a test observation of the background from the proton beam. At that moment, we can get real background hits. To train with reasonable noisy events, it might be that we can make a bit of reasonably noisy training samples by overlaying simulated signal hits and real noise hits though we are going to be definitely careful how to overlay them.

For now, we believe that NN models trained with simulation data will work well with real experimental data. But there must be some gap between simulation data and real experimental data. To deal with the gap, maybe we need to do Sim-to-Real Domain Adaptation mentioned in [24]. This problem should be solved in the future.

We also need an efficient way (or just another way for cross-checking) to estimate energy-momentum from CDC hits the information. This problem would be future work.

A 2D Histogram

This method was suggested by Chen Wu, who is one of the authors of [6], as a selection method of hits that represent each cell in CDC to be assigned to pixels in an image. This method strongly depends on the distribution of ADC SUM [6] and time information in simulation data. 2D Histogram requires some simulation data in advance. See Figure 37

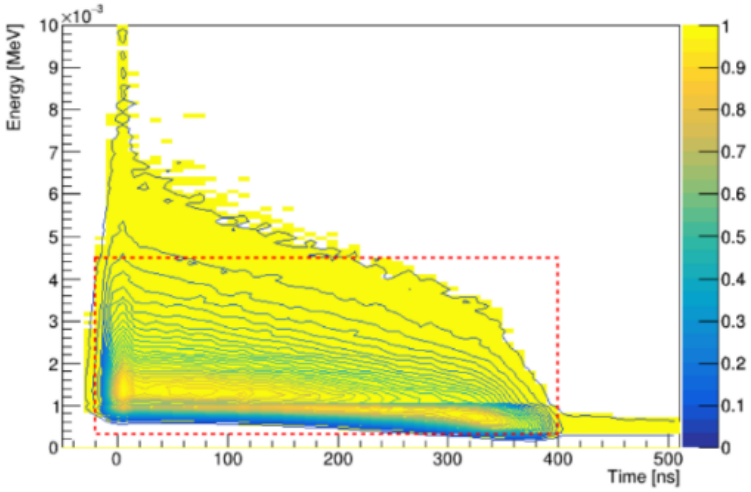

Figure 37: 2D Histogram suggested by Chen Wu

The vertical axis means energy deposit. The horizontal axis means time information.

We can make a 2D Histogram about ADC SUM [6] and time information from the huge amount of hits information in simulation. If a hit is in simulation, we know the turn index of trajectory which has the hit. So we can count the number of conversion-electron hits(called "signals" here) and noise hits about each bin. So we can calculate a score like Equation (33).

$$2dHistogramScore = \frac{TheNumberOfSignals}{TheNumberOfSignalHits + TheNumberOfNoiseHits} \tag{33}$$

B Graph Neural Network

In recent several years, Graph Neural Networks, called GNN, attracts attention to analyze graph-like objects e.g. social media networks, and molecules. Many GNN models are also proposed to be used in particle physics experiments [25][26]. In this thesis, I assigned each cell of CDC to pixels in the image event by event. But I broke a totally geometrical correlation because the position of cells is not lined like the grid of pixels mentioned in [25]. I hope that there should be such analysis keeping geometrical correlation. There is hope in Graph Neural Networks. So I mention it here.

B.1 Basics of Graph

Basically, a graph consists of nodes and edges. Mathematically nodes can have individual feature vectors. In particle physics experiments, nodes are likely to represent detectors. Feature vectors correspond to some information that each detector has. In most cases, edges represent the strength of connections between two nodes. There are both cases edges are undirected or directed. They are likely to be weighted by distances between two nodes for a particle physics experiment. See Right graph on Figure 38

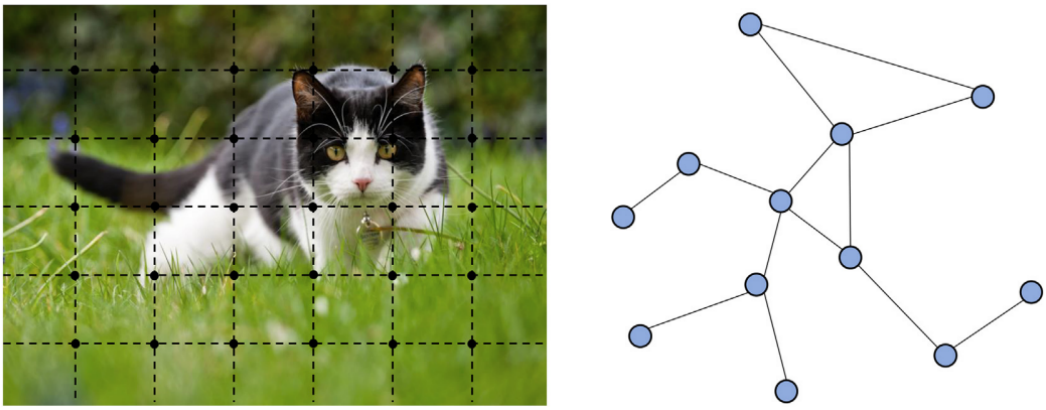

Figure 38: Left: image in Euclidean space. Right: graph in non-Euclidean space. [7]

B.2 Graph with Neural Network

GNN is one of NN but it expects to be fed with a graph. GNN means consecutive operations on a graph. The shape of the output of GNN can be varied. If the output is any scaler value, then maybe GNN works like graph classification. Or if the output is sub(or entire)-nodes of the input graph, then maybe GNN works like node classification.

Obviously, graph structure doesn't have the constraint to be a 2D grid. We can recognize it is a generalization of images(See Figure 38). But We have difficulty defining convolution-kernel since the local graph structure is not constant in general. There are struggles to make kernel in graph convolution [27].

B.2.1 Graph Convolution

There are so many kinds of graph convolutions in GNN [27, 28, 29, 30]. Many researchers have been trying to discover graph convolution as a straightforward generalization of convolution on images.

The simplest graph convolution was suggested in [29]. It's just propagating the following (weighted) adjacency matrix after the linear transformation of input graph data with trainable parameters. Unfortunately, this convolution is too simple to miss the geometrical correlation of nodes (The "nodes" are likely to be detector's cells in particle physics experiments) though the usual convolution takes into account the relative position between pixels by convolution-kernels.

Actually, I tried to use GCNConv based on [29] for the denoising task on the COMET experiment. But it doesn't work well. I think that It's because this GCNConv is much simple. As seen in [25], maybe the regression task can be relatively solved. But maybe the supervised nodes classification task needs more semantic features and complicated operations.

Anyway, so we need to employ any other graph convolution which has something like convolution-kernels [27].

Acknowledgement

First of all, I would like to express my deepest appreciation to my advisor, Prof. Joe Sato. Thanks to him, I could do what I want to do even though I'm in a theoretical laboratory. He set up a seminar for me to understand machine learning theory. And he gave me many chances I went to many councils, to give my talks.

Besides my advisor, I also would like to express my gratitude to Prof. Yoshitaka Kuno, and Chen Wu, Dorian PIETERS at Osaka University. They discussed experimental problems with me and provided simulation data to me for my research. And they care especially when I gave my talk on some councils.

I thank my fellow labmates. They helped me a lot with a theoretical seminar. They sometimes gave me some ideas, and sometimes gave me fun. I thank our friendships.

Of course, I want to thank my mother and father for supporting me financially, mentally, and physically. Especially I devote this thesis to my father in heaven.

Anyway, I want to thank all people which contribute, and support me.

References

[1] R. Abramishvili, et al., COMET phase-i technical design report, Progress of Theoretical and Experimental Physics 2020 (3) (mar 2020). doi:10.1093/ptep/ptz125.
URL https://doi.org/10.1093%2Fptep%2Fptz125

[2] A. Apicella, F. Donnarumma, F. Isgrò, R. Prevete, A survey on modern trainable activation functions, Neural Networks 138 (2021) 14–32. doi:10.1016/j.neunet.2021.01.026.
URL https://doi.org/10.1016%2Fj.neunet.2021.01.026

[3] O. Ronneberger, P. Fischer, T. Brox, U-net: Convolutional networks for biomedical image segmentation (2015). doi:10.48550/ARXIV.1505.04597.
URL https://arxiv.org/abs/1505.04597

[4] Z. Zhou, M. M. R. Siddiquee, N. Tajbakhsh, J. Liang, Unet++: A nested u-net architecture for medical image segmentation (2018). doi:10.48550/ARXIV.1807.10165.
URL https://arxiv.org/abs/1807.10165

[5] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, H. Adam, Encoder-decoder with atrous separable convolution for semantic image segmentation (2018). doi:10.48550/ARXIV.1802.02611.
URL https://arxiv.org/abs/1802.02611

[6] C. Wu, T. Wong, Y. Kuno, M. Moritsu, Y. Nakazawa, A. Sato, H. Sakamoto, N. Tran, M. Wong, H. Yoshida, T. Yamane, J. Zhang, Test of a small prototype of the COMET cylindrical drift chamber, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 1015 (2021) 165756. doi:10.1016/j.nima.2021.165756.
URL https://doi.org/10.1016%2Fj.nima.2021.165756

[7] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, M. Sun, Graph neural networks: A review of methods and applications, AI Open 1 (2020) 57–81. doi:https://doi.org/10.1016/j.aiopen.2021.01.001.
URL https://www.sciencedirect.com/science/article/pii/S2666651021000012

[8] Y. Kuno, Y. Okada, Muon decay and physics beyond the standard model, Reviews of Modern Physics 73 (1) (2001) 151–202. doi:10.1103/revmodphys.73.151.
URL https://doi.org/10.1103%2Frevmodphys.73.151

[9] J. Duchi, E. Hazan, Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization, Journal of Machine Learning Research 12 (61) (2011) 2121–2159.
URL http://jmlr.org/papers/v12/duchi11a.html

[10] M. D. Zeiler, Adadelta: An adaptive learning rate method (2012). doi:10.48550/ARXIV.1212.5701.
URL https://arxiv.org/abs/1212.5701

[11] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization (2014). doi:10.48550/ARXIV.1412.6980.
URL https://arxiv.org/abs/1412.6980

[12] L. N. Smith, N. Topin, Super-convergence: Very fast training of neural networks using large learning rates (2017). doi:10.48550/ARXIV.1708.07120.
URL https://arxiv.org/abs/1708.07120

[13] F. Yu, V. Koltun, Multi-scale context aggregation by dilated convolutions (2015). doi:10.48550/ARXIV.1511.07122.
URL https://arxiv.org/abs/1511.07122

[14] B. McFee, J. Salamon, J. P. Bello, Adaptive pooling operators for weakly labeled sound event detection (2018). doi:10.48550/ARXIV.1804.10070.
URL https://arxiv.org/abs/1804.10070

[15] K. Zhang, W. Zuo, Y. Chen, D. Meng, L. Zhang, Beyond a gaussian denoiser: Residual learning of deep CNN for image denoising, IEEE Transactions on Image Processing 26 (7) (2017) 3142–3155. doi:10.1109/tip.2017.2662206.
URL https://doi.org/10.1109%2Ftip.2017.2662206

[16] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift (2015). doi:10.48550/ARXIV.1502.03167.
URL https://arxiv.org/abs/1502.03167

[17] A. Kirillov, K. He, R. Girshick, C. Rother, P. Dollár, Panoptic segmentation (2018). doi:10.48550/ARXIV.1801.00868.
URL https://arxiv.org/abs/1801.00868

[18] A. G. Roy, N. Navab, C. Wachinger, Recalibrating fully convolutional networks with spatial and channel 'squeeze amp; excitation' blocks (2018). doi:10.48550/ARXIV.1808.08127.
URL https://arxiv.org/abs/1808.08127

[19] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, S. Belongie, Feature pyramid networks for object detection (2016). doi:10.48550/ARXIV.1612.03144.
URL https://arxiv.org/abs/1612.03144

[20] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need (2017). doi:10.48550/ARXIV.1706.03762.
URL https://arxiv.org/abs/1706.03762

[21] L.-C. Chen, G. Papandreou, F. Schroff, H. Adam, Rethinking atrous convolution for semantic image segmentation (2017). doi:10.48550/ARXIV.1706.05587.
URL https://arxiv.org/abs/1706.05587

[22] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, A. L. Yuille, Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs (2016). doi:10.48550/ARXIV.1606.00915.
URL https://arxiv.org/abs/1606.00915

[23] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, Pytorch: An imperative style, high-performance deep learning library, in: Advances in Neural Information Processing Systems 32, Curran Associates, Inc., 2019, pp. 8024–8035.
URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learni
pdf

[24] M. Baalouch, M. Defurne, J.-P. Poli, N. Cherrier, Sim-to-real domain adaptation for high energy physics (2019). doi:10.48550/ARXIV.1912.08001.
URL https://arxiv.org/abs/1912.08001

[25] M. Bachlechner, T. Birkenfeld, P. Soldin, A. Stahl, C. Wiebusch, Partition pooling for convolutional graph network applications in particle physics, Journal of Instrumentation 17 (10) (2022) P10004. doi:10.1088/1748-0221/17/10/p10004.
URL https://doi.org/10.1088%2F1748-0221%2F17%2F10%2Fp10004

[26] J. Kahn, I. Tsaklidis, O. Taubert, L. Reuter, G. Dujany, T. Boeckh, A. Thaller, P. Goldenzweig, F. Bernlochner, A. Streit, M. Götz, Learning tree structures from leaves for particle decay reconstruction, Machine Learning: Science and Technology 3 (3) (2022) 035012. doi:10.1088/2632-2153/ac8de0.
URL https://doi.org/10.1088%2F2632-2153%2Fac8de0

[27] H. Lei, N. Akhtar, A. Mian, Spherical kernel for efficient graph convolution on 3d point clouds (2019). doi:10.48550/ARXIV.1909.09287.
URL https://arxiv.org/abs/1909.09287

[28] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, M. Grohe, Weisfeiler and leman go neural: Higher-order graph neural networks (2018). doi:10.48550/ARXIV.1810.02244.
URL https://arxiv.org/abs/1810.02244

[29] T. N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks (2016). doi:10.48550/ARXIV.1609.02907.
URL https://arxiv.org/abs/1609.02907

[30] Z. Ma, J. Xuan, Y. G. Wang, M. Li, P. Liò , Path integral based convolution and pooling for graph neural networks∗, Journal of Statistical Mechanics: Theory and Experiment 2021 (12) (2021) 124011. doi:10.1088/1742-5468/ac3ae4.
URL https://doi.org/10.1088%2F1742-5468%2Fac3ae4