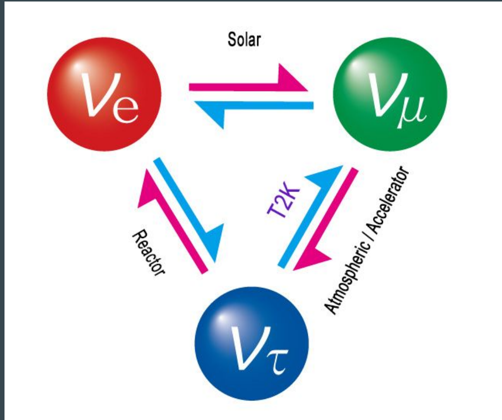
Application of Convolutional Neural Networks (CNNs) to Neutrino Interaction Identification

Group 8 Members:
Carlos Pareja, Billy Haoyang Li, Jay Sun, Sahil Bhalla

Why Is Neutrino Important?

Neutrino can oscillate



Neutrino oscillation between three generations

Pontecorvo–Maki–Nakagawa–Sakata (PMNS) matrix

$$\begin{bmatrix} \nu_e \\ \nu_\mu \\ \nu_\tau \end{bmatrix} = \begin{bmatrix} U_{e1} & U_{e2} & U_{e3} \\ U_{\mu 1} & U_{\mu 2} & U_{\mu 3} \\ U_{\tau 1} & U_{\tau 2} & U_{\tau 3} \end{bmatrix} \begin{bmatrix} \nu_1 \\ \nu_2 \\ \nu_3 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & c_{23} & s_{23} \\ 0 & -s_{23} & c_{23} \end{bmatrix} \begin{bmatrix} c_{13} & 0 & s_{13} e^{-i\delta_{CP}} \\ 0 & 1 & 0 \\ -s_{13} e^{i\delta_{CP}} & 0 & c_{13} \end{bmatrix} \begin{bmatrix} c_{12} & s_{12} & 0 \\ -s_{12} & c_{12} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} c_{12} c_{13} & s_{12} c_{13} & s_{13} e^{-i\delta_{CP}} \\ -s_{12} c_{23} - c_{12} s_{23} s_{13} e^{i\delta_{CP}} & c_{12} c_{23} - s_{12} s_{23} s_{13} e^{i\delta_{CP}} & s_{23} c_{13} \\ s_{12} s_{23} - c_{12} c_{23} s_{13} e^{i\delta_{CP}} & -c_{12} s_{23} - s_{12} c_{23} s_{13} e^{i\delta_{CP}} & c_{23} c_{13} \end{bmatrix}.$$

δ_CP is related to charge-parity violation

c_ij and s_ij can be parameterized by advanced theories

Dataset: Simulated LArTPC Signals

Neutrinos are hard to detect directly ...

$$\nu_\mu + n \rightarrow \mu^- + p$$

Detect the product particles instead!

Simulated signals of charged particles in LArTPC

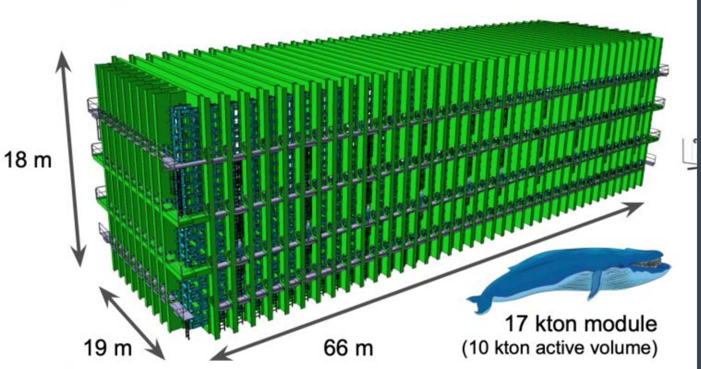
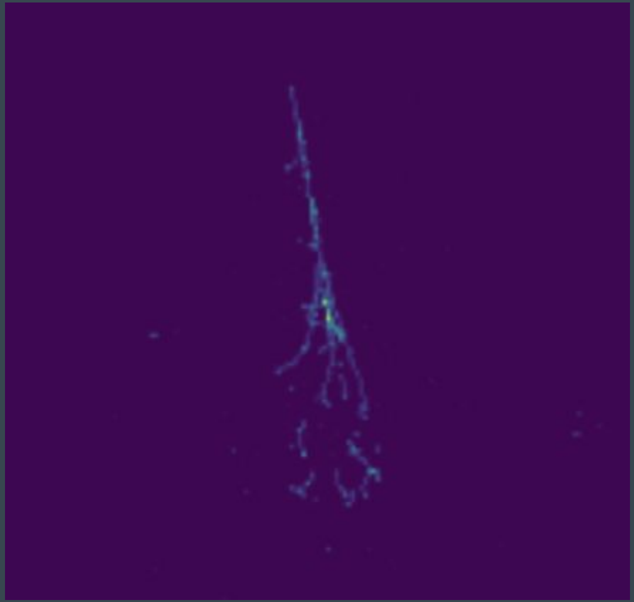


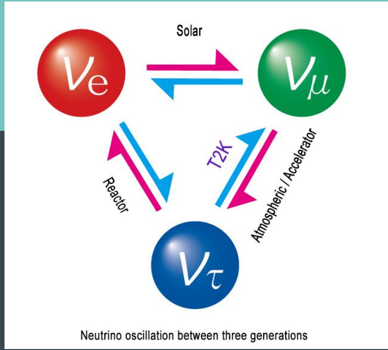
Figure: DUNE LArTPC

Input: 256x256x1 tensor per view: XY, YZ, XZ



Target: one-hot vector of 5 categories: electron, muon, photon, pion, proton

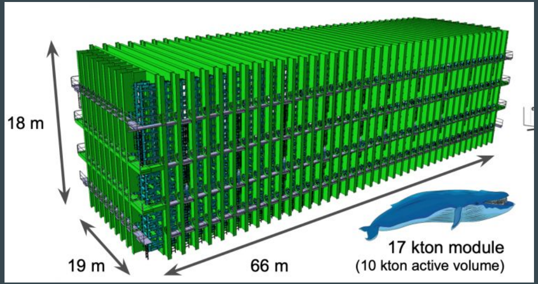
Data Analysis

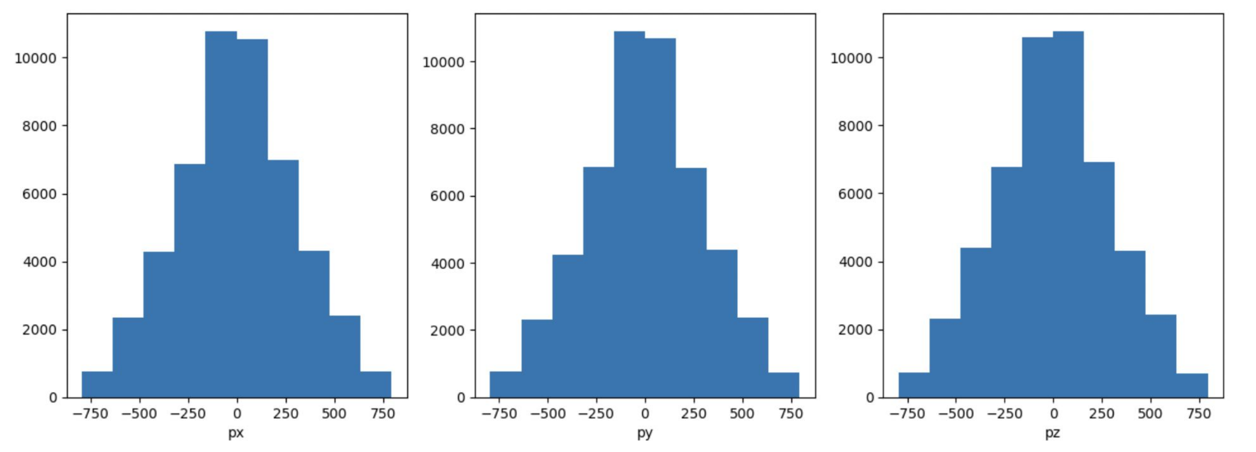


Large dataset: 50k images. Training time is about 2 hours per epoch.

Dataset is balanced: 10k for each class

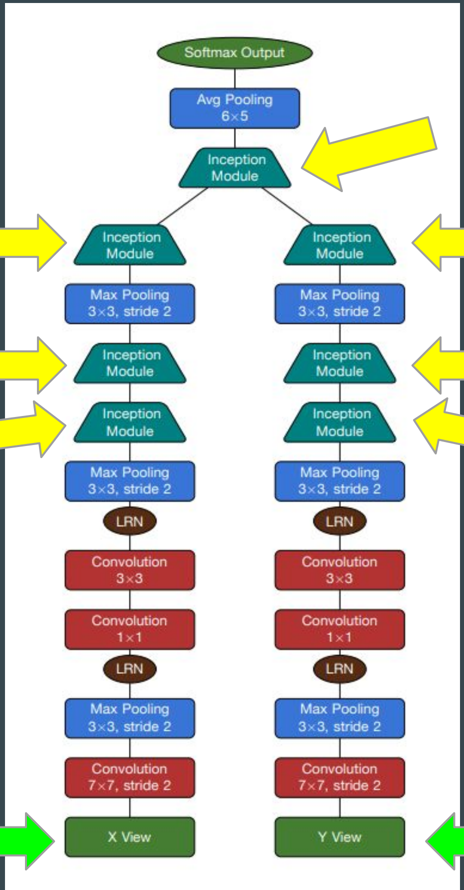
No primary "beam" direction in simulation

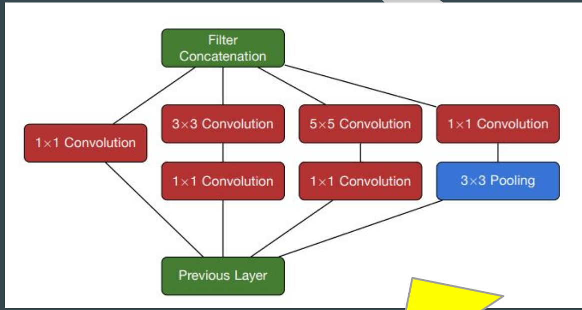




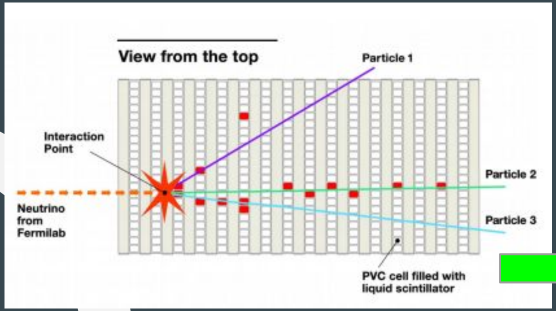
Implies XY, YX, YZ views are equally important in our dataset!

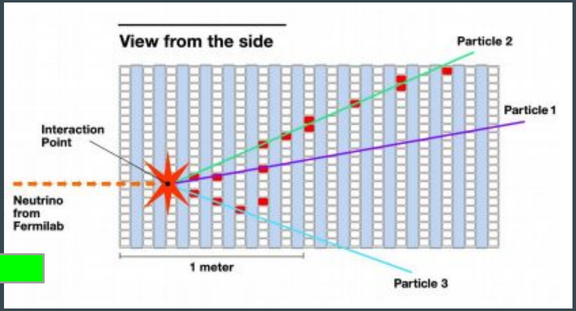
Convolutional Visual Network (CVN) Architecture



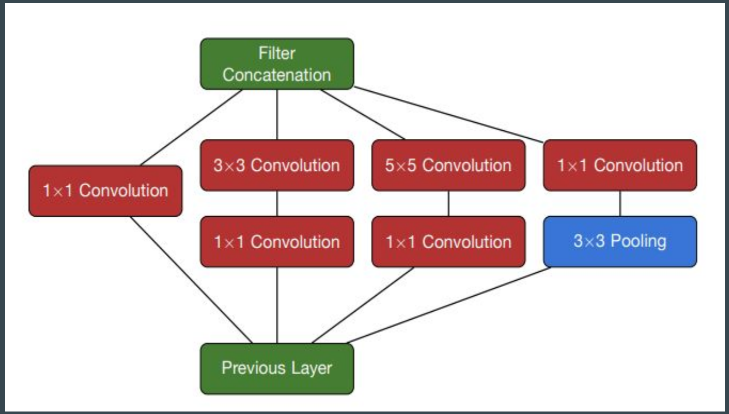


https://arxiv.org/abs/1604.01444



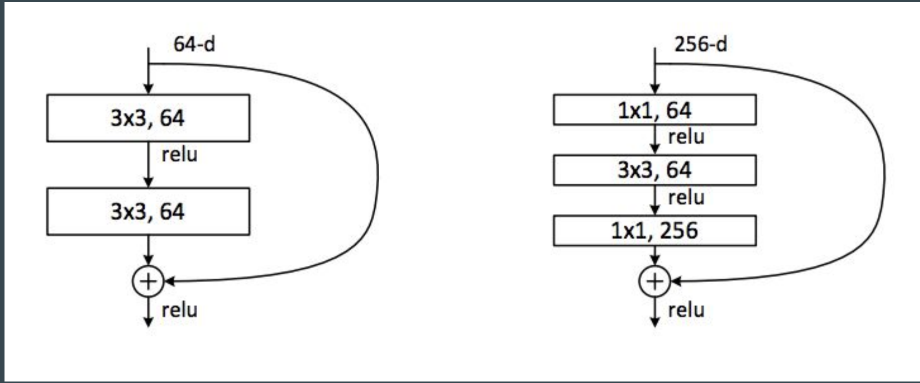


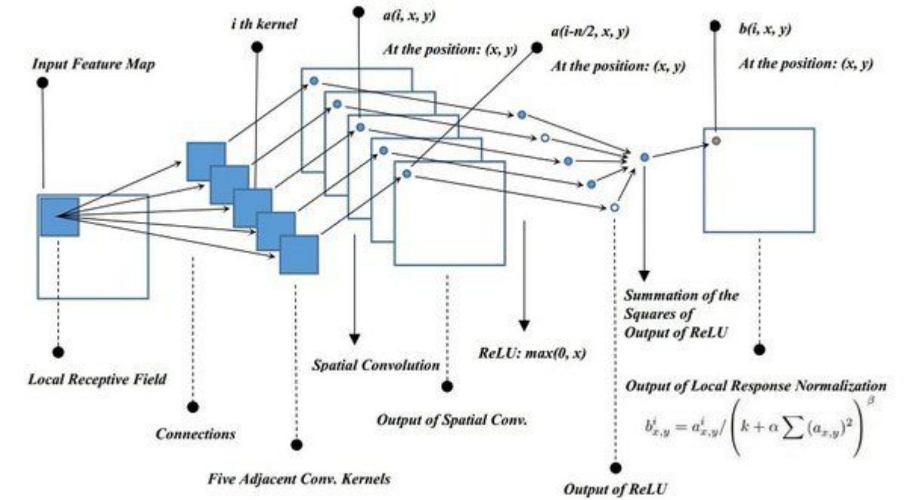
Inception Module

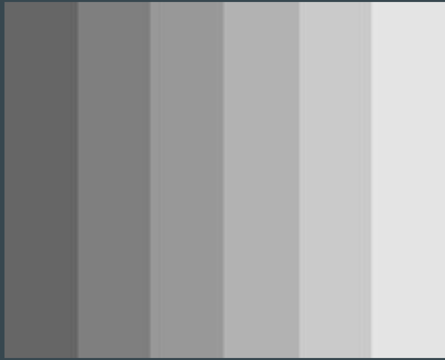



```python
class Inception_Block(nn.Module):
    def __init__(
    ):
        super(Inception_Block, self).__init__()
        self.branch1 = ConvBlock(in_channels, output_1x1, kernel_size = 1)
        self.branch2 = nn.Sequential(
        ConvBlock(in_channels, out_channels = output_1x1_block2, kernel_size = 1),
        ConvBlock(output_1x1_block2, output_3x3, kernel_size = 3, padding = 1)
        )
        self.branch3 = nn.Sequential(
            ConvBlock(in_channels, out_channels = output_5x5_reduce, kernel_size = 1),
            ConvBlock(in_channels= output_5x5_reduce, out_channels=output_5x5, kernel_size = 5, paddin
        )
        self.branch4 = nn.Sequential(
            nn.MaxPool2d(kernel_size=3, stride=1, padding=1),
            ConvBlock(in_channels, output_pool, kernel_size = 1)
        )
    def forward(self, x):
        first_block = self.branch1(x)
        second_block = self.branch2(x)
        third_block = self.branch3(x)
        fourth_block = self.branch4(x)
        output_concat = torch.cat([first_block, second_block, third_block, fourth_block], dim  = 1)

        return output_concat
```

1x1 Kernel and Local Response Normalization







Additional Implementation Details

```
class x_model(nn.Module):
    def __init__(self):
        super(x_model,self).__init__()
        self.conv_7x7 = nn.Conv2d(in_channels=1, out_channels=64,  kernel_size=7, stride=2)
        self.max_pool = nn.MaxPool2d(kernel_size=3, stride = 2)
        self.lrn_norm = nn.LocalResponseNorm(size=5,  alpha=0.0001, beta=0.75)
        self.conv_1x1 = nn.Conv2d(in_channels=64, out_channels= 64, kernel_size=1)
        self.conv3_3x3 = nn.Conv2d(in_channels=64, out_channels=256, kernel_size=3)
        self.inception3a = Inception_Block(in_channels=256,output_1x1=64, output_1x1_block2=96
        self.inception3b = Inception_Block(in_channels=256,output_1x1=64, output_1x1_block2=96
        self.inception4a = Inception_Block(in_channels=256,output_1x1=64, output_1x1_block2=96


    def forward(self, x):
        x = self.max_pool(F.relu(self.conv_7x7(x)))
        x = self.lrn_norm(x)
        x = F.relu(self.conv_1x1(x))
        x = F.relu(self.conv3_3x3(x))
        x = self.max_pool(self.lrn_norm(x))
        x = self.inception3a(x)
        x = self.inception3b(x)
        x = self.max_pool(x)
        x = self.inception4a(x)
        return x
```

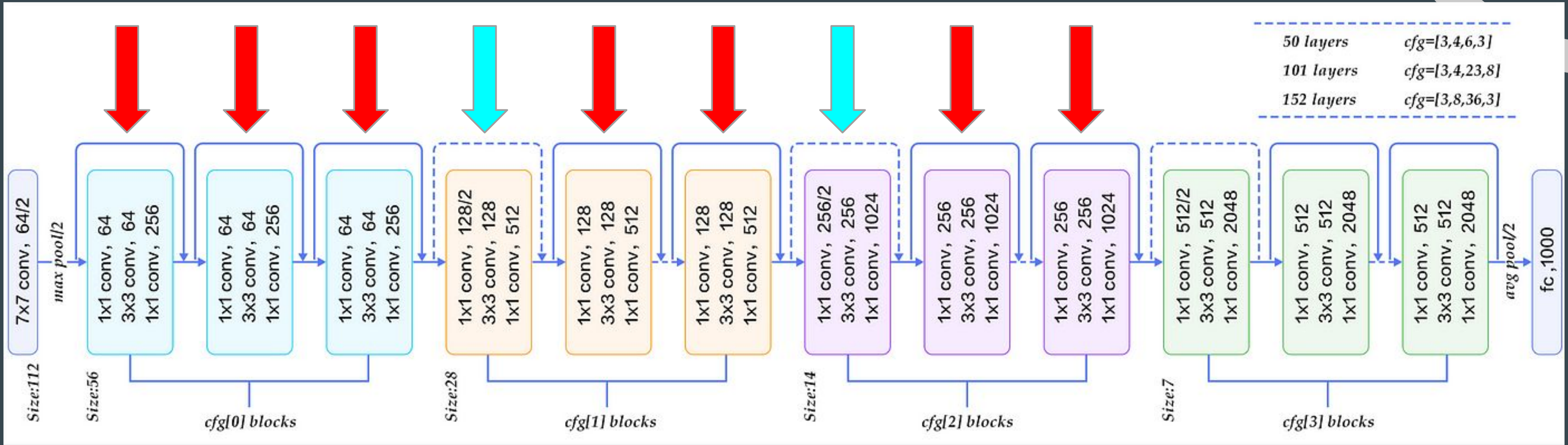
- Relu Activation
- Learning Rate 0.0001
- Batch size 64
- 500 epochs
- Adam Optimizer
- Cross Entropy Loss

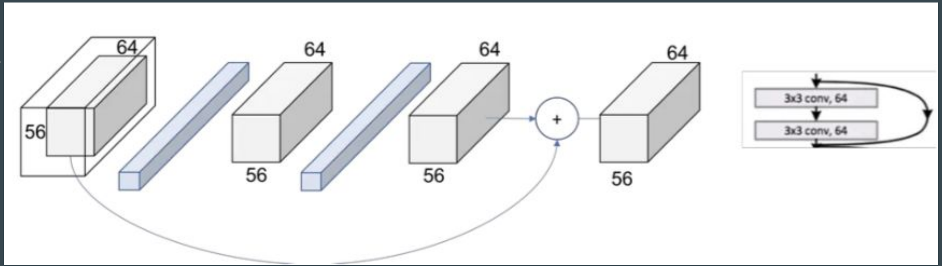
CVN Results

- Currently Flawed, with constant loss and accuracy
- Believe it is some pytorch issue caused by some mistake in our code in the inception model since our later skip-connection model trained better
- Approximately 2,000,000 parameters

```
epoch,loss,accuracy,val_loss,val_acc
Epoch 0: Loss = 99.97349772582183,train acc = 21.240641711229948,Val Loss = 101.7370273347885, Val acc = 25.106703489831784
Epoch 1: Loss = 101.67752165407748,train acc = 21.390374331550802,Val Loss = 101.75219820416163, Val acc = 25.106703489831784
Epoch 2: Loss = 101.6772843180476,train acc = 21.390374331550802,Val Loss = 101.75219820416163, Val acc = 25.106703489831784
Epoch 3: Loss = 101.6772843180476,train acc = 21.390374331550802,Val Loss = 101.75219820416163, Val acc = 25.106703489831784
Epoch 4: Loss = 101.6772843180476,train acc = 21.390374331550802,Val Loss = 101.75219820416163, Val acc = 25.106703489831784
Epoch 5: Loss = 101.6772843180476,train acc = 21.390374331550802,Val Loss = 101.75219820416163, Val acc = 25.106703489831784
Epoch 6: Loss = 101.6772843180476,train acc = 21.390374331550802,Val Loss = 101.75219820416163, Val acc = 25.106703489831784
Epoch 7: Loss = 101.6772843180476,train acc = 21.390374331550802,Val Loss = 101.75219820416163, Val acc = 25.106703489831784
Epoch 8: Loss = 101.6772843180476,train acc = 21.390374331550802,Val Loss = 101.75219820416163, Val acc = 25.106703489831784
Epoch 0: Loss = 101.67927579622011,train acc = 21.28342245989305,Val Loss = 101.75219820416163, Val acc = 25.106703489831784
Epoch 9: Loss = 101.6772843180476,train acc = 21.390374331550802,Val Loss = 101.75219820416163, Val acc = 25.106703489831784
Epoch 1: Loss = 101.6772843180476,train acc = 21.390374331550802,Val Loss = 101.75219820416163, Val acc = 25.106703489831784
Epoch 10: Loss = 101.6772843180476,train acc = 21.390374331550802,Val Loss = 101.75219820416163, Val acc = 25.106703489831784
Epoch 2: Loss = 101.6772843180476,train acc = 21.390374331550802,Val Loss = 101.75219820416163, Val acc = 25.106703489831784
Epoch 11: Loss = 101.6772843180476,train acc = 21.390374331550802,Val Loss = 101.75219820416163, Val acc = 25.106703489831784
Epoch 3: Loss = 101.6772843180476,train acc = 21.390374331550802,Val Loss = 101.75219820416163, Val acc = 25.106703489831784
Epoch 12: Loss = 101.6772843180476,train acc = 21.390374331550802,Val Loss = 101.75219820416163, Val acc = 25.106703489831784
Epoch 4: Loss = 101.6772843180476,train acc = 21.390374331550802,Val Loss = 101.75219820416163, Val acc = 25.106703489831784
Epoch 13: Loss = 101.6772843180476,train acc = 21.390374331550802,Val Loss = 101.75219820416163, Val acc = 25.106703489831784
Epoch 5: Loss = 101.6772843180476,train acc = 21.390374331550802,Val Loss = 101.75219820416163, Val acc = 25.106703489831784
Epoch 14: Loss = 101.6772843180476,train acc = 21.390374331550802,Val Loss = 101.75219820416163, Val acc = 25.106703489831784
Epoch 6: Loss = 101.6772843180476,train acc = 21.390374331550802,Val Loss = 101.75219820416163, Val acc = 25.106703489831784
Epoch 15: Loss = 101.6772843180476,train acc = 21.390374331550802,Val Loss = 101.75219820416163, Val acc = 25.106703489831784
Epoch 7: Loss = 101.6772843180476,train acc = 21.390374331550802,Val Loss = 101.75219820416163, Val acc = 25.106703489831784
Epoch 16: Loss = 101.6772843180476,train acc = 21.390374331550802,Val Loss = 101.75219820416163, Val acc = 25.106703489831784
```

ResNet50 Architecture (We pick only cfg[0] to cfg[2])





Basically the same as the red arrow, but the skip connection goes through 1 convolution.

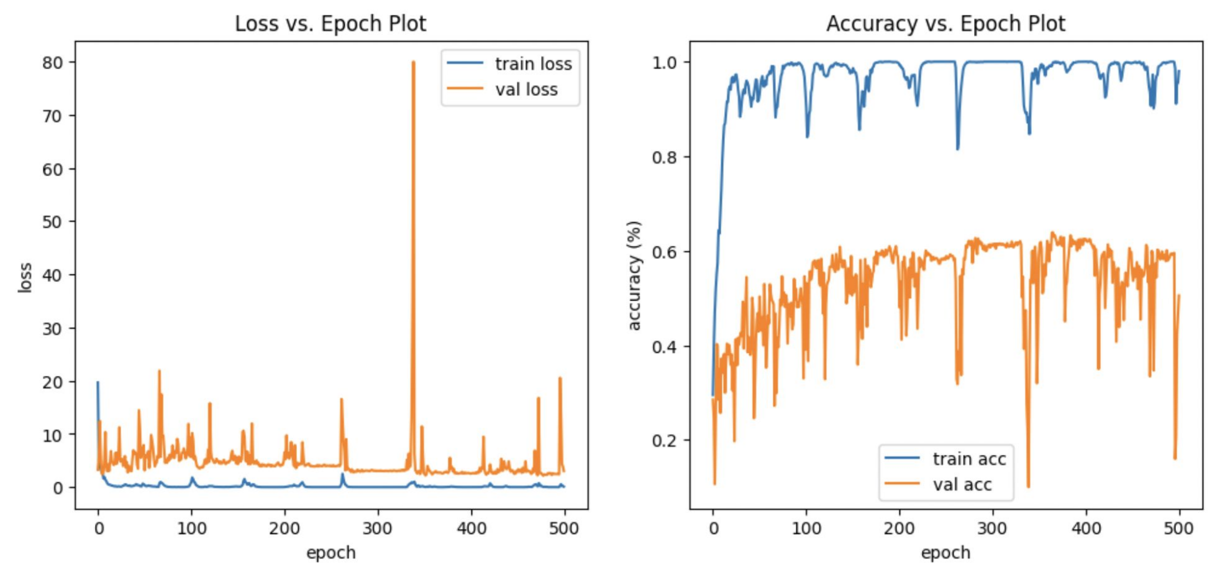
TensorFlow Implementation of ResNet-50

- ReLU activation function
- Batch size = 128
- Optimizer: Keras.Adam
- 500 epochs
- Learning rate = 0.0005

```
# Identity Block of conv4
resnet50_conv4_identity_block_input = Input(shape=(16,16,1024), name='conv4_identity_block_input' )
x = Conv2D( 256, kernel_size=1, strides=1, activation='relu', padding='valid', kernel_constraint=keras.constrai
x = BatchNormalization()(x)
x = Conv2D( 256, kernel_size=3, strides=1, activation='relu', padding='same', kernel_constraint=keras.constrain
x = BatchNormalization()(x)
x = Conv2D( 1024, kernel_size=1, strides=1, activation='relu', padding='valid', kernel_constraint=keras.constra
x = BatchNormalization()(x)

x = Add()([x, resnet50_conv4_identity_block_input ])
resnet50_conv4_identity_block_output = ReLU()(x)

resnet50_conv4_identity_block = Model(
    inputs = resnet50_conv4_identity_block_input,
    outputs= resnet50_conv4_identity_block_output,
    name = 'resnet50_conv4_identity_block'
)

```

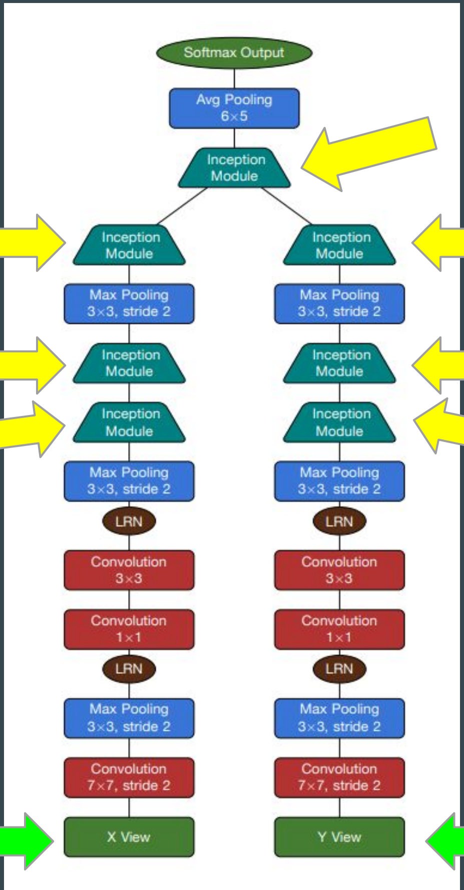
ResNet Results

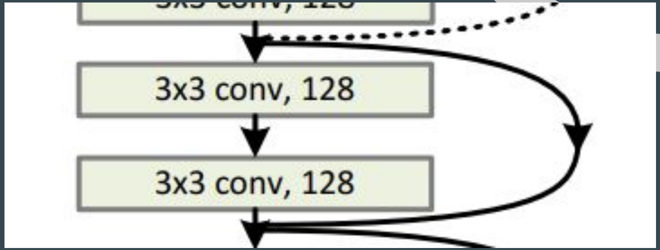
Looks like overfitting.

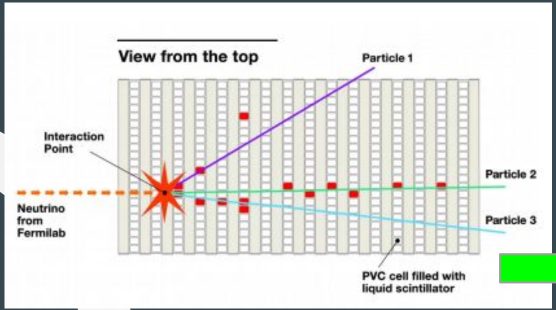


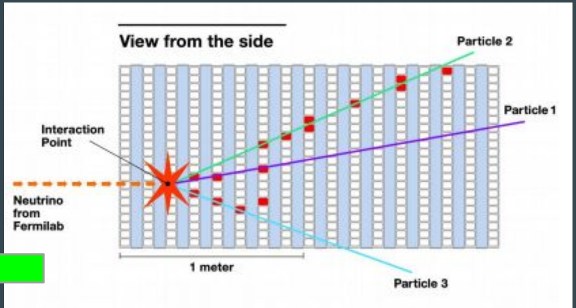
A mix of CVN and ResNet: 2-view Resnet

We replaced the inception module with skip connections.









PyTorch Implementation Details

```
class Skip_Connection_Block(nn.Module):
    def __init__( self, in_channels ):

        super(Skip_Connection_Block, self).__init__()


        #  in_channels = 256
        self.through_connection = nn.Sequential(
            nn.Conv2d(
                in_channels = in_channels, out_channels = in_channels,
                kernel_size=3, padding='same' ),

            nn.ReLU(),
            nn.BatchNorm2d(in_channels),
            nn.Conv2d( in_channels = in_channels, out_channels = in_channels,
                      kernel_size=3, padding='same' ),
            nn.ReLU(),
            nn.BatchNorm2d(in_channels)
        )


    def forward(self, x):
        y = self.through_connection(x)
        output = torch.add(x, y)

        return output
```

Skip connection block, as used in ResNet

Learning Rate 0.0001

Batch size 16

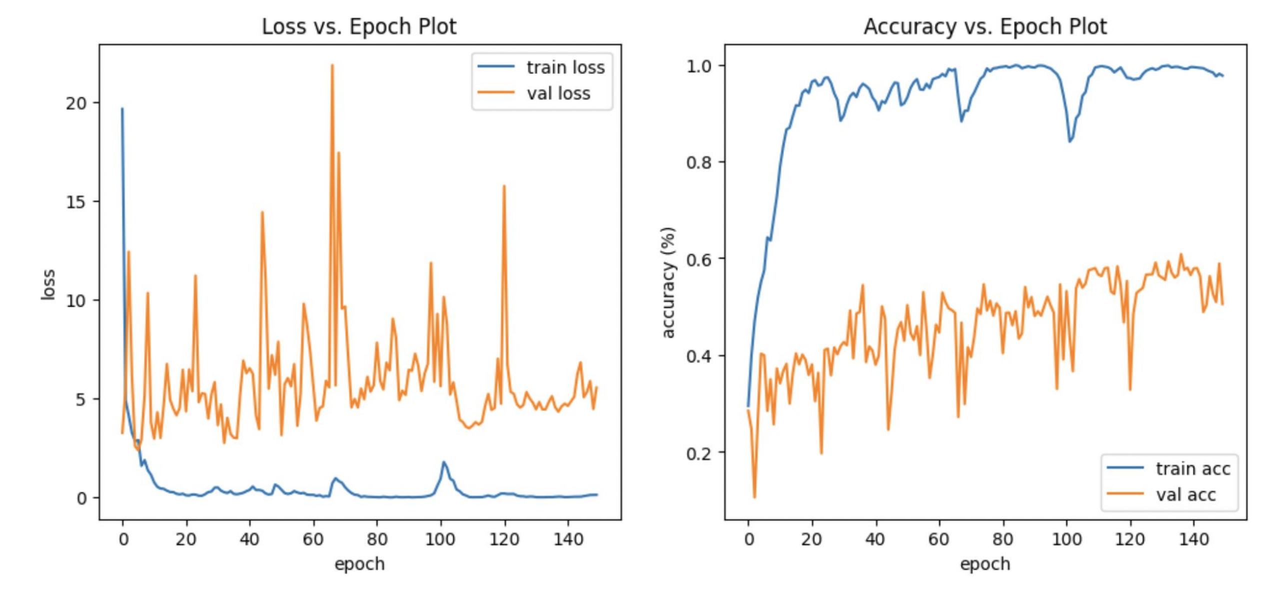
~ 150 epochs

Adam Optimizer

2 view ResNet Results

First 150 epochs.

Still a work in progress.



Comparing 3 different networks

Once we iron on the bugs in our models we plan to compare the CVN, Resnet, and the 2-view Resnet

Does the 2-view really help given the computational costs?

What's the difference in performance of the inception blocks and residual connections in the Resnet

Summary/Outlook

Our project aims to understand what design a CNN should have for neutrino detection:

Encoded detector geometry: beam direction, detector views, etc.

Feature extracting modules: Inception module, residual connection, etc.

We would focus on reducing the degree of overfitting by:

Training the model on a larger dataset (Currently, 10% of total dataset is used for training and other 10% used for testing)

Adding dropout layers

...

Thank You!

Billy Haoyang Li: ResNet4Block, dataloader

Carlos Parej: CVN, dataloader

Sahil Bhalla: CVN, dataloader

Jay Sun: ResNet50, 2-View ResNet

Github: https://github.com/cpareja3025/phys139-239_final_project