# High Energy Muon Detection

Ipsita Praharaj

ipsita.praharaj1997@gmail.com

# TO DO

TASK 1:  MUON MOMENTUM INFERENCE USING DEEP NEURAL NETWORKS

- Develop a Fully-Connected Network using a framework of your choice and evaluate its ability in classifying muon momentum ranges using the raw data muon data that we provided.

TASK 2:  IMAGE-BASED CLASSIFICATION OF MUON MOMENTA

- Similar to Task 1, this task requires you to implement a FCN and a CNN. Only this time, you have to project the hits provided by the raw data into images

Reference paper

- Boosted decision trees in the CMS Level-1 endcap muon trigger -PoS(TWEPP-17)143

# Steps followed

1. Pre processing data
2. Model Implementation for (1/PT)
3. Prediction on (1/pt) and (pt)*
4. Metrics

**Pre processing data**

- Considered features of only ME Chambers belonging to the CSC

**Model Implementation**

- Fed all the muon data to train
- Trained and stored the weights of best performing models

**Metrics:**

- Regression:
  - MAE (For Test & Train)
- Classification
  - Accuracy(For Test & Train)

# Workflow

PRE PROCESS

MODELLED WHOLE DATASET

METRICS PRODUCED BY USING SAVED MODELS FOR MULTIPLE RANGES(1/PT) & (PT)

VISUALIZATION OF METRICS PRODUCED

- ONLY CSC parameters used
- Removed Nan values
- Normalised the train data
- Initialized bins for later use

1. **design matrix, X(1/pt)**
2. phi:0-11 (phi coordinate of a hit)
3. theta: 12-23 (theta coordinate of a hit)
4. bend: 24-35 (bend angle inside the detector; e.g. CSC is made of 6 layers)
5. time: 36-47 (some time info; I don't use it)
6. ring: 48-59 (ring number; )
7. fr: 60-71 (front or rear part of detector)
8. x_mask: 72-83 (mask for NaN value; detectors are not 100% efficient so sometimes miss a hit)
9. x_road: 84,85,86

**Total 23 variables**

**Model implemented on x>pt, where pt is a threshold set initially**

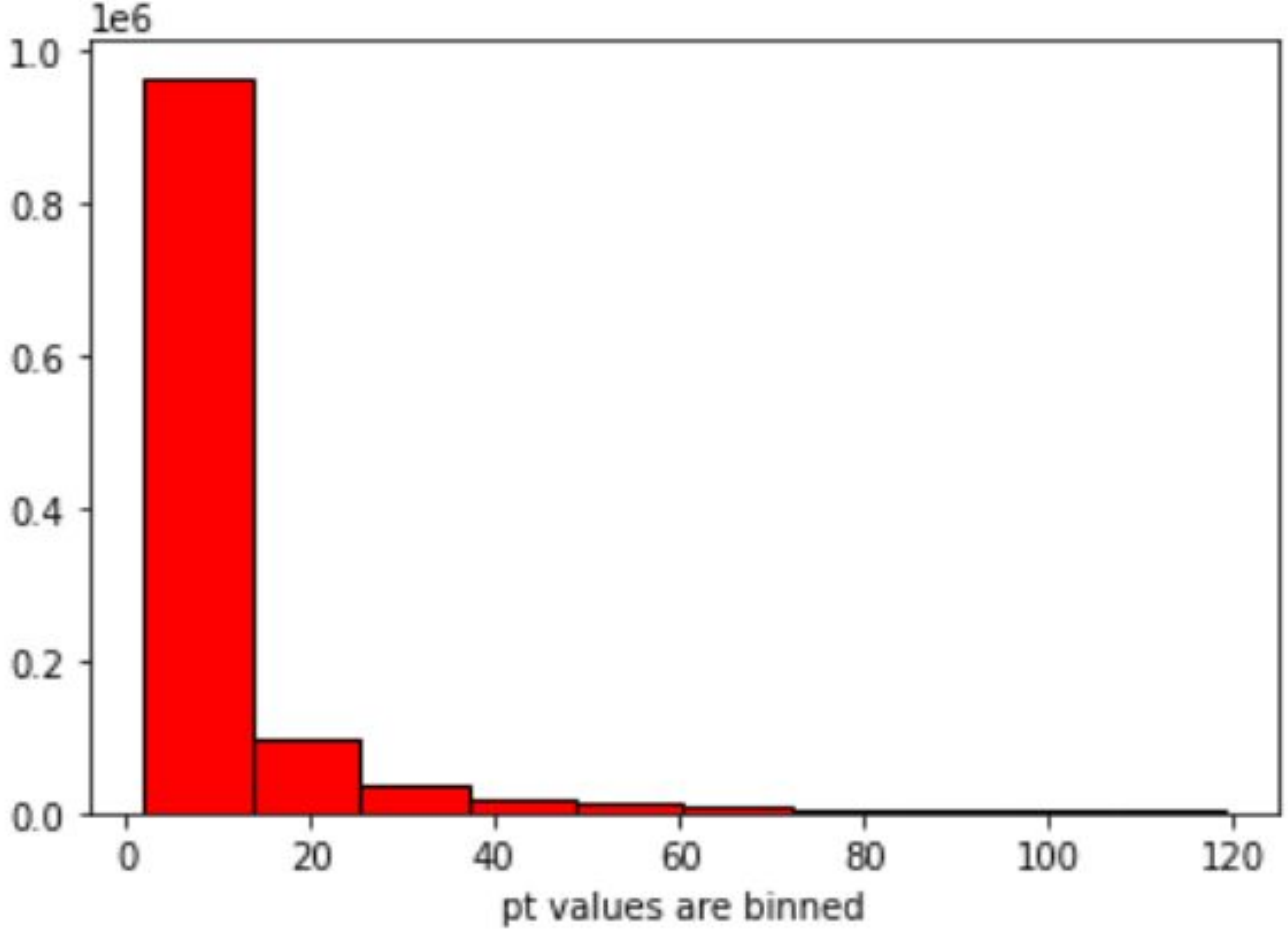
- Ran model for recursively for the ranges like (x>0) ,(x>1), (x>119)
- Recorded the y_test_pred & y_train_pred for every class
- **F1(weighted), accuracy & MAE was recorded**

# Experiment data

Absolute (1/pt) binned into 11 classes and predicted against features:

**Design matrix,  X [1179356 rows, 23 columns , y , label]**

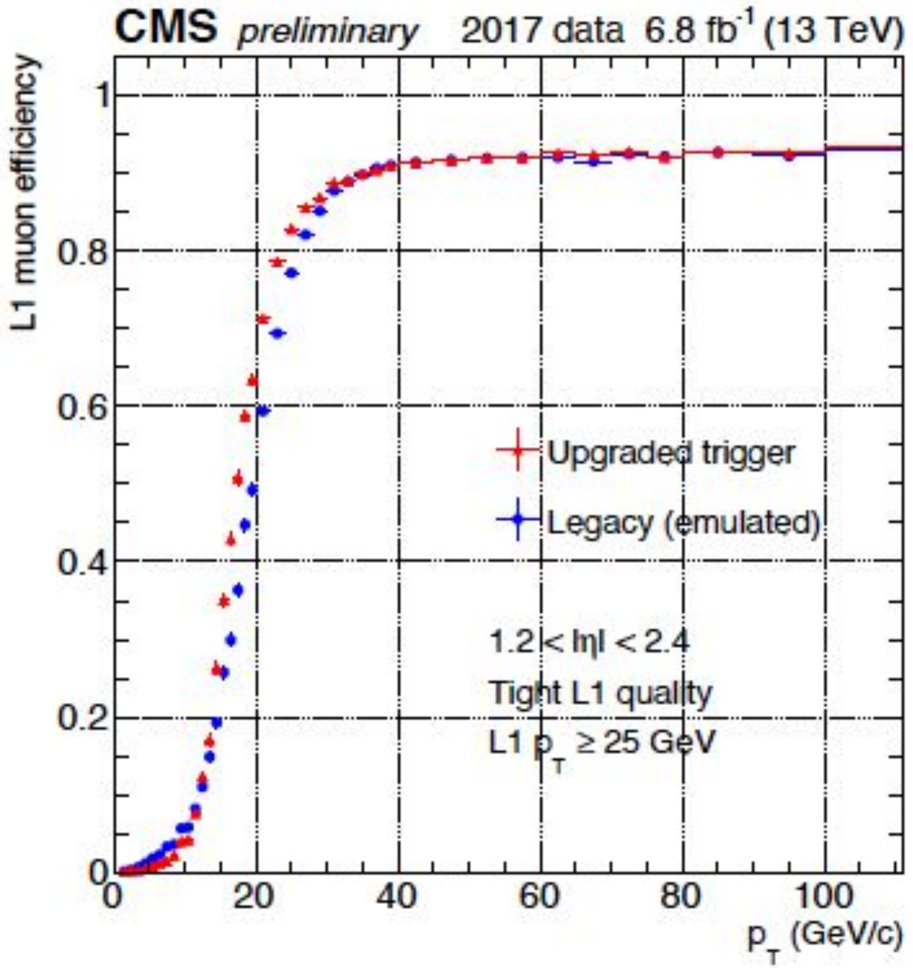
1. phi:0-11 (phi coordinate of a hit)
2. theta: 12-23 (theta coordinate of a hit)
3. bend: 24-35 (bend angle inside the detector; e.g. CSC is made of 6 layers)
4. time: 36-47 (some time info; I don't use it)
5. ring: 48-59 (ring number; )
6. fr: 60-71 (front or rear part of detector)
7. x_mask: 72-83 (mask for NaN value; detectors are not 100% efficient so sometimes miss a hit)
8. x_road: 84,85,86



# Models Used

1. Regression & classification
    a. BDT + Bayesian Hyperparameter tuning
    b. CNN 1D
    c. CNN 2D

TARGET



# XGB +
# Bayesian parameter tuning

# Steps involved

In [79]:
```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=123)
dtrain = xgb.DMatrix(data=X_train,label=y_train)
dtest = xgb.DMatrix(data=X_test,label=y_test)
```

In [80]:
```python
def xgb_evaluate(max_depth, gamma, colsample_bytree):
    params = {'eval_metric': 'rmse',
              'max_depth': int(max_depth),
              'subsample': 0.8,
              'eta': 0.1,
              'gamma': gamma,
              'colsample_bytree': colsample_bytree}
    # Used around 1000 boosting rounds in the full model
    cv_result = xgb.cv(params, dtrain, num_boost_round=100, nfold=3)

    # Bayesian optimization only knows how to maximize, not minimize, so return the negative RMSE
    return -1.0 * cv_result['test-rmse-mean'].iloc[-1]
```

In [81]:
```python
xgb_bo = BayesianOptimization(xgb_evaluate, {'max_depth': (3, 7),
                                             'gamma': (0, 1),
                                             'colsample_bytree': (0.3, 0.9)})
# Use the expected improvement acquisition function to handle negative numbers
# Optimally needs quite a few more initiation points and number of iterations
xgb_bo.maximize(init_points=3, n_iter=20, acq='ei')
```

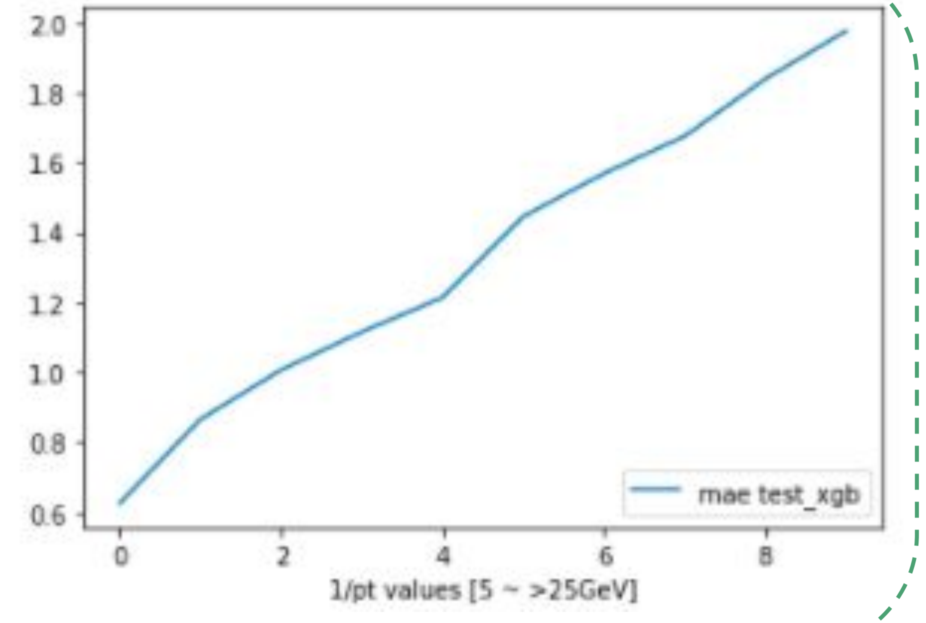
| iter | target | colsam... | gamma | max_depth |
|---|---|---|---|---|
| 1 | -0.1006 | 0.5898 | 0.5726 | 6.252 |
| 2 | -0.1047 | 0.3321 | 0.06579 | 3.036 |
| 3 | -0.1005 | 0.8088 | 0.6342 | 6.069 |
| 4 | -0.1006 | 0.9 | 1.0 | 7.0 |
| 5 | -0.1004 | 0.9 | 0.0 | 6.905 |
| 6 | -0.1004 | 0.8881 | 0.01765 | 6.233 |
| 7 | -0.1004 | 0.9 | 0.3127 | 6.512 |
| 8 | -0.1029 | 0.3472 | 0.01955 | 6.993 |
| 9 | -0.1049 | 0.3 | 1.0 | 5.569 |
| 10 | -0.1006 | 0.9 | 0.8306 | 6.468 |
| 11 | -0.1005 | 0.8724 | 0.3703 | 6.106 |
| 12 | -0.1004 | 0.9 | 0.0 | 6.588 |
| 13 | -0.1005 | 0.8653 | 0.5657 | 6.28 |
| 14 | -0.1005 | 0.9 | 0.4804 | 7.0 |
| 15 | -0.1004 | 0.9 | 0.0 | 5.622 |
| 16 | -0.1004 | 0.6976 | 0.01982 | 5.931 |
| 17 | -0.1004 | 0.9 | 0.0 | 5.888 |
| 18 | -0.1005 | 0.9 | 0.0 | 4.873 |
| 19 | -0.1004 | 0.9 | 0.0 | 5.2 |
| 20 | -0.1014 | 0.4772 | 0.0 | 4.998 |
| 21 | -0.1004 | 0.8986 | 0.3343 | 6.822 |
| 22 | -0.1005 | 0.8997 | 0.6965 | 6.807 |
| 23 | -0.1005 | 0.8858 | 0.007266 | 4.426 |

Trained on 1/pt & predict (pt)

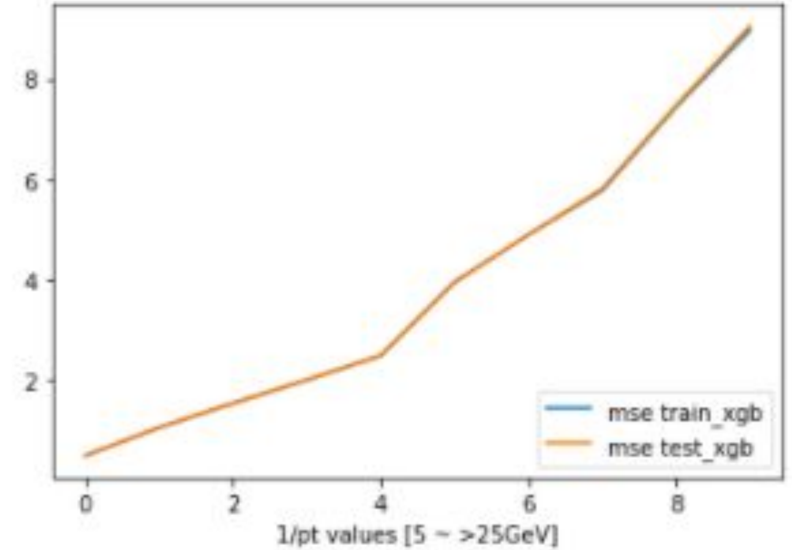
In [40]:
```python
print("Mean absolute error =", round(sm.mean_absolute_error(y_test, y_pred), 4))
print("Mean squared error =", round(sm.mean_squared_error(y_test, y_pred), 4))
print("Median absolute error =", round(sm.median_absolute_error(y_test, y_pred), 4))
print("Explain variance score =", round(sm.explained_variance_score(y_test, y_pred), 4))
print("R2 score =", round(sm.r2_score(y_test, y_pred), 4))
```

```
Mean absolute error = 19.1209
Mean squared error = 16265.128
Median absolute error = 2.0138
Explain variance score = 0.0271
R2 score = 0.0271
```

```
bin 0 ---> 2.0000038 GeV to 5 GeV
bin 1 ---> 5 GeV to 10 GeV
bin 2 ---> 10 GeV to 15 GeV
bin 3 ---> 15 GeV to 20 GeV
bin 4 ---> 20 GeV to 25 GeV
bin 5 ---> 25 GeV to 40 GeV
bin 6 ---> 40 GeV to 50 GeV
bin 7 ---> 50 GeV to 60 GeV
bin 8 ---> 60 GeV to 80 GeV
bin 9 ---> 80 GeV to 100 GeV
bin 10 ---> 100 GeV to 6955.571 GeV
```




1. Trained 1/pt wrt to 23 variables
2. Selected the best optimized hyperparameter
3. Used recursively for the modeling of different pt thresholds




```
bin 0 ---> 2.0000038 GeV to 6955.571 GeV
bin 1 ---> 5 GeV to 6955.571 GeV
bin 2 ---> 10 GeV to 6955.571 GeV
bin 3 ---> 15 GeV to 6955.571 GeV
bin 4 ---> 20 GeV to 6955.571 GeV
bin 5 ---> 25 GeV to 6955.571 GeV
bin 6 ---> 40 GeV to 6955.571 GeV
bin 7 ---> 50 GeV to 6955.571 GeV
bin 8 ---> 60 GeV to 6955.571 GeV
bin 9 ---> 80 GeV to 6955.571 GeV
bin 10 ---> 100 GeV to 6955.571 GeV
```

| | bin 0 -> 2.0000038 to 6956.0 | bin 1 -> 5 to 6956.0 | bin 2 -> 10 to 6956.0 | bin 3 -> 15 to 6956.0 | bin 4 -> 20 to 6956.0 | bin 5 -> 25 to 6956.0 | bin 6 -> 40 to 6956.0 | bin 7 -> 50 to 6956.0 | bin 8 -> 60 to 6956.0 | bin 9 -> 80 to 6956.0 | bin 10 -> 100 to 6956.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **r2** | 0.606172 | 0.711088 | 0.731276 | 0.733137 | 0.718085 | 0.666869 | 0.630745 | 0.600164 | 0.551927 | 0.509348 | 0.037406 |
| **mse_train** | 0.505885 | 1.063595 | 1.545954 | 2.011343 | 2.494370 | 3.964123 | 4.902457 | 5.784265 | 7.444005 | 8.970324 | 125.201378 |
| **mse_test** | 0.507481 | 1.070481 | 1.555378 | 2.014256 | 2.504588 | 3.948766 | 4.906351 | 5.821937 | 7.480084 | 9.055583 | 132.175278 |
| **mae_test** | 0.625634 | 0.865305 | 1.006574 | 1.115249 | 1.213775 | 1.444961 | 1.568316 | 1.674903 | 1.838826 | 1.975836 | 4.480676 |

## XGB

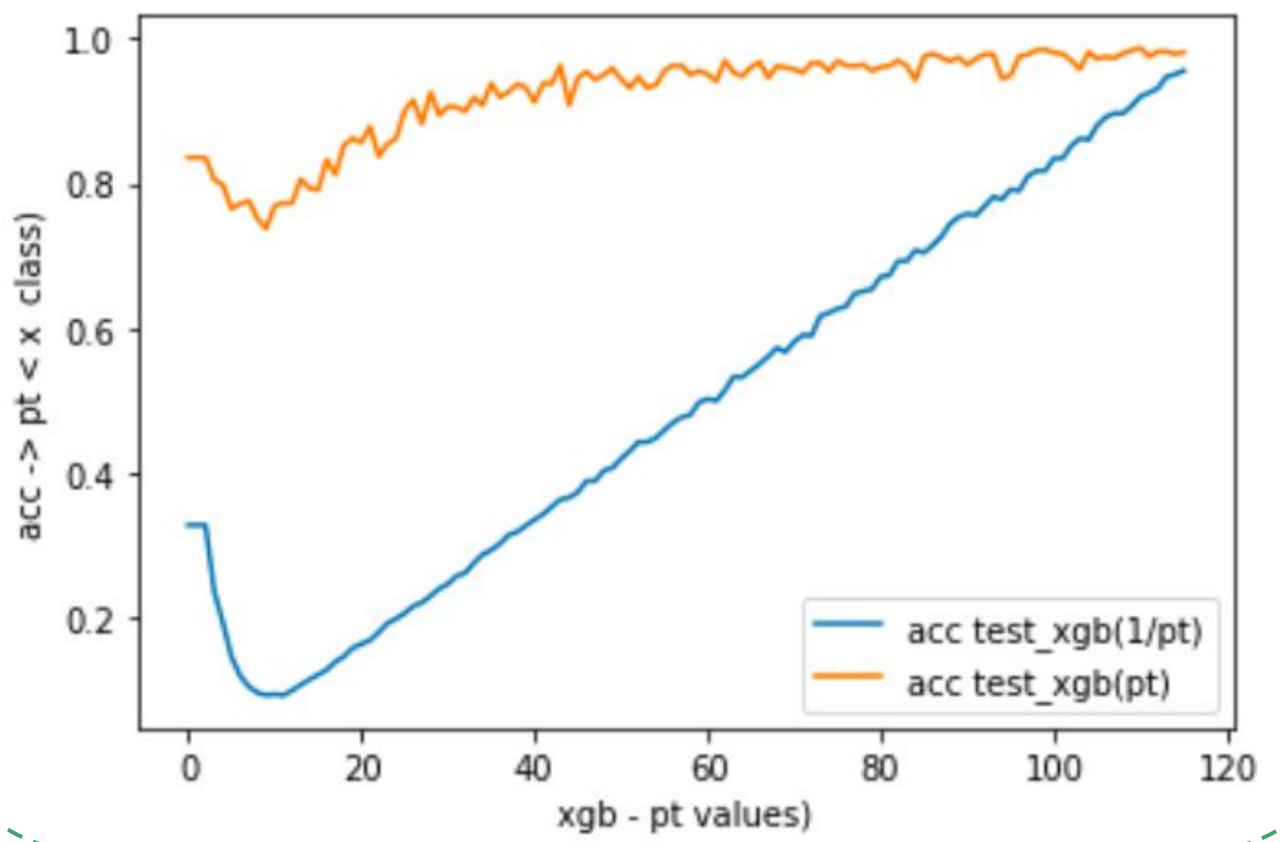
```
In [44]: params = pd.read_csv('/Users/ipsitapraharaj/Desktop/CERN/params_20.csv', index_col = 0)
         params = params.loc[0,:].to_dict()
         params['max_depth'] = int(params['max_depth'])
```

for binwise prediction

```
In [45]: def xgb_model(i):

             data_new = data[1/abs(data.y)>= bins[i]]
             X_new, y_new = data_new.iloc[:,:-2],data_new.iloc[:,-2]
             X_new = (X_new-X_new.min())/(X_new.max()-X_new.min())

             e = np.digitize(y_new,bins)
             X_train, X_test, y_train, y_test = train_test_split(X_new, y_new, test_size=0.2, random_state=123, stratify = e )
             dtrain = xgb.DMatrix(data=X_train,label=y_train)
             dtest = xgb.DMatrix(data=X_test,label=y_test)

             # Train a new model with the best parameters from the search
             model2 = xgb.train(params, dtrain, num_boost_round=25)

             # Predict on testing and training set
             y_pred = model2.predict(dtest)
             y_train_pred = model2.predict(dtrain)

             acc = round(accuracy_score(np.digitize(1/(abs(y_test)), bins), np.digitize(1/(abs(y_pred)), bins)),4)
             mae = round(sm.mean_absolute_error(y_test, y_pred), 4)
             f1 = f1_score(np.digitize(1/(abs(y_test)), bins), np.digitize(1/(abs(y_pred[:,0])), bins), average='weighted')

             return acc, mae, f1,  y_test, y_pred, y_train, y_train_pred
```

```
In [46]: xgb_results = np.array([xgb_model(i) for i in tqdm(range(len(bins[5:6])))])
```

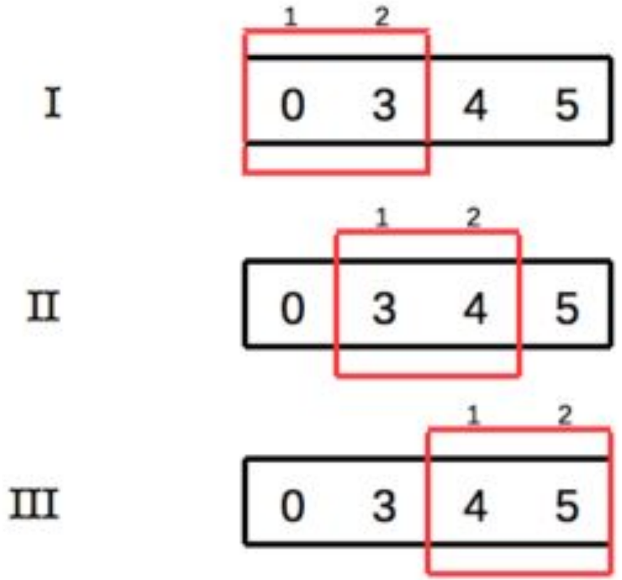
Trained on every threshold(0-120)



# CNN 1D

# CNN 1D

This layer creates a convolution kernel that is convolved with the layer input over a single spatial (or temporal) dimension to produce a tensor of outputs. [n X 1]




## CNN 1D

Please download the model_cnn1.json & model_cnn1.h5 files to the local directory to run this model

The results are displayed corresponding to each cell

In [6]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=123)
```

In [7]:
```
x = X.to_numpy().reshape(X.to_numpy().shape[0], X.to_numpy().shape[1], 1)
print(x.shape)
x = np.nan_to_num(x)
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=123)
```

```
(1179356, 19, 1)
```

In [70]:
```
model = Sequential()
model.add(Conv1D(64, 2, activation="relu", input_shape=(x.shape[1:])))
model.add(Conv1D(64, 2, activation="relu"))
model.add(Conv1D(64, 2, activation="relu"))
model.add(Flatten())
model.add(Dense(1, activation="relu"))
model.compile(loss="mae", optimizer="adam")

model.summary()
```

```
Model: "sequential_14"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_21 (Conv1D)           (None, 18, 64)            192
_________________________________________________________________
conv1d_22 (Conv1D)           (None, 17, 64)            8256
_________________________________________________________________
conv1d_23 (Conv1D)           (None, 16, 64)            8256
_________________________________________________________________
flatten_7 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_56 (Dense)             (None, 1)                 1025
=================================================================
Total params: 17,729
Trainable params: 17,729
Non-trainable params: 0
_________________________________________________________________
```

In [74]:
```
model.fit(X_train, y_train, batch_size=1000,epochs=20, verbose=2)
```

```
In [85]: ypred = model3.predict(X_test)
         print(model.evaluate(X_train, y_train))

         print("MAE: %.4f" % sm.mean_absolute_error(y_test, ypred))

29484/29484 [==============================] - 96s 3ms/step - loss: 12.2749 - accuracy: 0.0000e+00
[12.274883270263672, 0.0]
MAE: 12.1647
```

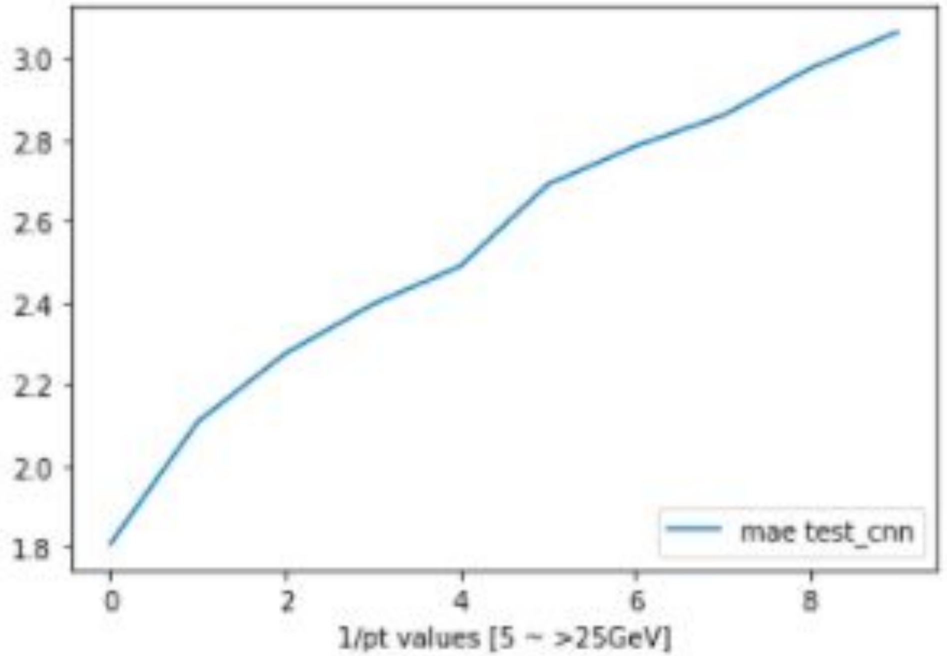
```
In [86]: print("R2 score =", round(sm.r2_score(y_test, ypred), 4))

R2 score = 0.0584
```

Trained on 1/pt predicted on (pt)

```
In [28]: print("Mean absolute error =", round(sm.mean_absolute_error(y_test, ypred), 4))
         print("Mean squared error =", round(sm.mean_squared_error(y_test, ypred), 4))
         print("Median absolute error =", round(sm.median_absolute_error(y_test, ypred), 4))
         print("Explain variance score =", round(sm.explained_variance_score(y_test, ypred), 4))
         print("R2 score =", round(sm.r2_score(y_test, ypred), 4))

Mean absolute error = 0.0273
Mean squared error = 0.0016
Median absolute error = 0.0185
Explain variance score = 0.9749
R2 score = 0.9745
```

Trained on 1/pt Predicted (1/pt)

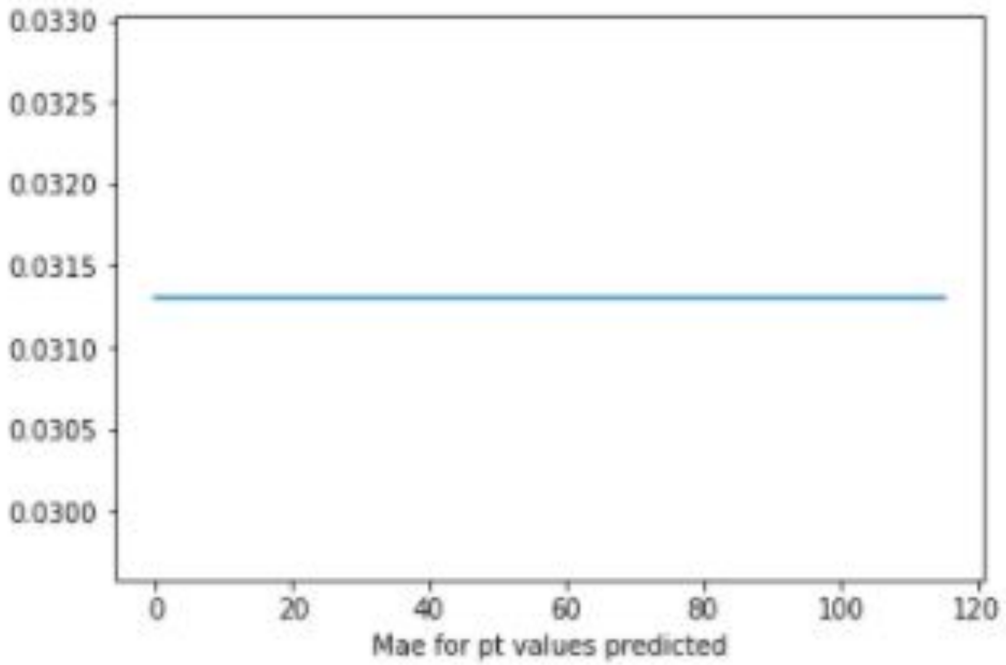
```
bin 0 ---> 2.0000038 GeV to 5 GeV
bin 1 ---> 5 GeV to 10 GeV
bin 2 ---> 10 GeV to 15 GeV
bin 3 ---> 15 GeV to 20 GeV
bin 4 ---> 20 GeV to 25 GeV
bin 5 ---> 25 GeV to 40 GeV
bin 6 ---> 40 GeV to 50 GeV
bin 7 ---> 50 GeV to 60 GeV
bin 8 ---> 60 GeV to 80 GeV
bin 9 ---> 80 GeV to 100 GeV
bin 10 ---> 100 GeV to 6955.571 GeV
```



| | bin 0 -> 2.0000038 to 6956.0 | bin 1 -> 5 to 6956.0 | bin 2 -> 10 to 6956.0 | bin 3 -> 15 to 6956.0 | bin 4 -> 20 to 6956.0 | bin 5 -> 25 to 6956.0 | bin 6 -> 40 to 6956.0 | bin 7 -> 50 to 6956.0 | bin 8 -> 60 to 6956.0 | bin 9 -> 80 to 6956.0 | bin 10 -> 100 to 6956.0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **r2** | -16.536550 | -5.002075 | -2.984570 | -2.171672 | -1.729100 | -1.119027 | -0.921520 | -0.788301 | -0.626749 | -0.527603 | -0.022625 |
| **mse_train** | 3.390981 | 4.882520 | 5.989531 | 6.949671 | 7.810274 | 10.003409 | 11.198399 | 12.303960 | 14.255304 | 15.938505 | 136.675171 |
| **mse_test** | 3.386399 | 4.879184 | 5.989268 | 6.944072 | 7.792682 | 9.959150 | 11.192256 | 12.312518 | 14.252540 | 15.978457 | 136.234482 |
| **mae_test** | 1.809168 | 2.108613 | 2.275817 | 2.396538 | 2.490270 | 2.690480 | 2.784478 | 2.859688 | 2.975137 | 3.065159 | 4.502475 |

Recursive segment

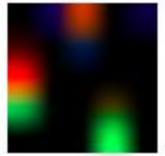
```
In [37]: def cnn1_model(i):


             data_new = data[1/abs(data.y)>= bins[1]]
             data_new = data[coll]
             X_new, y_new = data_new.iloc[:,:-2],(data_new.iloc[:,-2])
             X_new = (X_new-X_new.min())/(X_new.max()-X_new.min())

             X_new = X_new.to_numpy().reshape(X_new.to_numpy().shape[0], X_new.to_numpy().shape[1], 1)
             X_new = np.nan_to_num(X_new)

             e = np.digitize(y_new,bins)
             X_train, X_test, y_train, y_test = train_test_split(X_new, y_new, test_size=0.2, random_state=123, stratify = e )

             # Predict on testing and training set
             y_pred = loaded_model1.predict(X_test)
             y_train_pred = loaded_model1.predict(X_train)

             acc = round(accuracy_score(np.digitize(1/(abs(y_test)), bins), np.digitize(1/(abs(y_pred[:,0])), bins)),4)
             mae = round(sm.mean_absolute_error(y_test, y_pred[:,0]), 4)
             f1 = f1_score(np.digitize(1/(abs(y_test)), bins), np.digitize(1/(abs(y_pred[:,0])), bins), average='weighted')


             return acc, mae, f1,  y_test, y_pred, y_train, y_train_pred

In [39]: cnn1_results = np.array([cnn1_model(i) for i in tqdm(range(len(bins)))])
```




Trained on 1/pt predicted 1/pt

# CNN 2d

In [85]:
```python
Image.fromarray(X[1], 'RGB').resize((100,100))# 4px X 4px image
```
Out[85]:



In [86]:
```python
Image.fromarray(X[1], 'RGB').resize((100,100)).convert('L') # 4px X 4px image
```
Out[86]:

In [88]:
```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=123)
# building the input vector from the 28x28 pixels
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

# normalizing the data to help with the training
X_train /= 255
X_test /= 255
```

In [91]:
```python
# building a linear stack of layers with the sequential model
model = models.Sequential()
model.add(layers.Conv2D(64, (3, 3), activation='relu', input_shape=(X_test.shape[1:])))
model.add(layers.Conv2D(64, (2, 2), activation='relu'))
model.add(layers.Conv2D(32, (2, 2), activation='relu'))
model.add(layers.Conv2D(16, (1, 1), activation='relu'))
model.add(layers.Flatten())
model.add(Dense(1, kernel_initializer='normal'))
model.summary()
```

```
Model: "sequential"
```

Model: "sequential"

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d (Conv2D) | (None, 3, 3, 64) | 640 |
| conv2d_1 (Conv2D) | (None, 2, 2, 64) | 16448 |
| conv2d_2 (Conv2D) | (None, 1, 1, 32) | 8224 |
| conv2d_3 (Conv2D) | (None, 1, 1, 16) | 528 |
| flatten (Flatten) | (None, 16) | 0 |
| dense_1 (Dense) | (None, 1) | 17 |

```
Total params: 25,857
Trainable params: 25,857
Non-trainable params: 0
```

```
Epoch 1/20
29484/29484 [==============================] - 93s 3ms/step - loss: 0.0024 - accuracy:
0.0000e+00 - val_loss: 0.0024 - val_accuracy: 0.0000e+00
Epoch 2/20
29484/29484 [==============================] - 94s 3ms/step - loss: 0.0015 - accuracy:
0.0000e+00 - val_loss: 0.0014 - val_accuracy: 0.0000e+00
Epoch 3/20
29484/29484 [==============================] - 93s 3ms/step - loss: 0.0014 - accuracy:
0.0000e+00 - val_loss: 0.0034 - val_accuracy: 0.0000e+00
Epoch 4/20
29484/29484 [==============================] - 93s 3ms/step - loss: 0.0014 - accuracy:
0.0000e+00 - val_loss: 0.0010 - val_accuracy: 0.0000e+00
Epoch 5/20
29484/29484 [==============================] - 97s 3ms/step - loss: 0.0014 - accuracy:
0.0000e+00 - val_loss: 0.0013 - val_accuracy: 0.0000e+00
Epoch 6/20
29484/29484 [==============================] - 83s 3ms/step - loss: 0.0014 - accuracy:
0.0000e+00 - val_loss: 8.8866e-04 - val_accuracy: 0.0000e+00
Epoch 7/20
29484/29484 [==============================] - 88s 3ms/step - loss: 0.0013 - accuracy:
0.0000e+00 - val_loss: 3.4024e-04 - val_accuracy: 0.0000e+00
Epoch 8/20
29484/29484 [==============================] - 108s 4ms/step - loss: 0.0013 - accuracy:
0.0000e+00 - val_loss: 5.2093e-04 - val_accuracy: 0.0000e+00
Epoch 9/20
29484/29484 [==============================] - 138s 5ms/step - loss: 0.0016 - accuracy:
0.0000e+00 - val_loss: 0.0014 - val_accuracy: 0.0000e+00
Epoch 10/20
29484/29484 [==============================] - 155s 5ms/step - loss: 0.0014 - accuracy:
0.0000e+00 - val_loss: 0.0023 - val_accuracy: 0.0000e+00
Epoch 11/20
29484/29484 [==============================] - 151s 5ms/step - loss: 0.0013 - accuracy:
0.0000e+00 - val_loss: 0.0022 - val_accuracy: 0.0000e+00
Epoch 12/20
29484/29484 [==============================] - 120s 4ms/step - loss: 0.0013 - accuracy:
0.0000e+00 - val_loss: 4.0372e-04 - val_accuracy: 0.0000e+00
Epoch 13/20
29484/29484 [==============================] - 123s 4ms/step - loss: 0.0013 - accuracy:
0.0000e+00 - val_loss: 0.0019 - val_accuracy: 0.0000e+00
Epoch 14/20
29484/29484 [==============================] - 121s 4ms/step - loss: 0.0013 - accuracy:
0.0000e+00 - val_loss: 4.1798e-04 - val_accuracy: 0.0000e+00
Epoch 15/20
29484/29484 [==============================] - 123s 4ms/step - loss: 0.0013 - accuracy:
0.0000e+00 - val_loss: 0.0018 - val_accuracy: 0.0000e+00
Epoch 16/20
29484/29484 [==============================] - 145s 5ms/step - loss: 0.0012 - accuracy:
0.0000e+00 - val_loss: 0.0016 - val_accuracy: 0.0000e+00
Epoch 17/20
29484/29484 [==============================] - 127s 4ms/step - loss: 0.0012 - accuracy:
0.0000e+00 - val_loss: 9.8926e-04 - val_accuracy: 0.0000e+00
Epoch 18/20
29484/29484 [==============================] - 131s 4ms/step - loss: 0.0012 - accuracy:
0.0000e+00 - val_loss: 0.0019 - val_accuracy: 0.0000e+00
Epoch 19/20
29484/29484 [==============================] - 131s 4ms/step - loss: 0.0012 - accuracy:
0.0000e+00 - val_loss: 0.0016 - val_accuracy: 0.0000e+00
Epoch 20/20
29484/29484 [==============================] - 147s 5ms/step - loss: 0.0012 - accuracy:
0.0000e+00 - val_loss: 0.0012 - val_accuracy: 0.0000e+00
```

```
94]: test_loss, test_acc = loaded_model.evaluate(X_test,  y_test, verbose=2)
     print((test_acc*100))
     print(test_loss)
     print("R2 score =", round(sm.r2_score(y_test, loaded_model.predict(X_test)), 4))
```

```
7371/7371 - 19s - loss: 0.0015 - accuracy: 0.0000e+00
0.0
0.0014776100870221853
R2 score = 0.9999
```

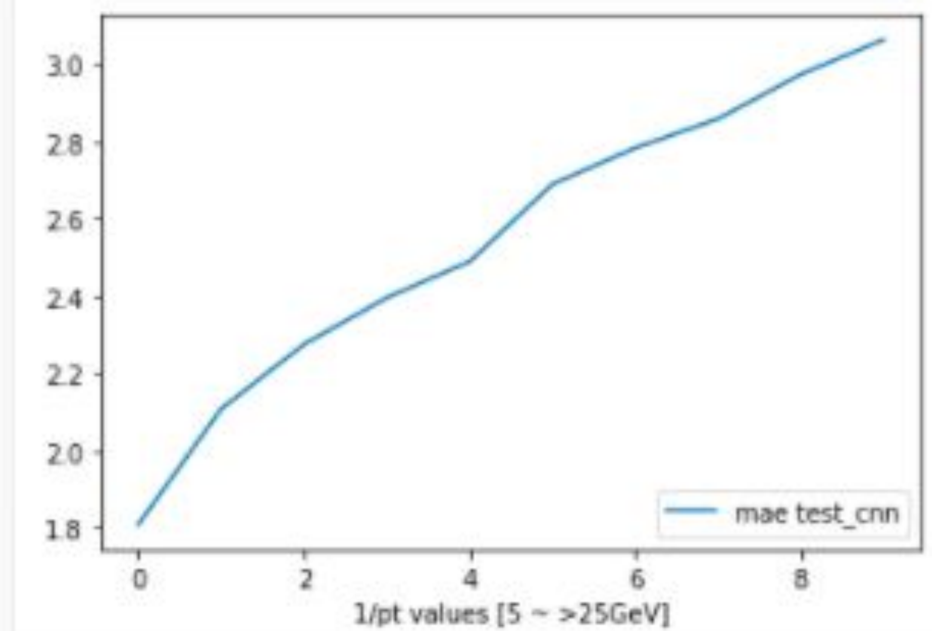
Trained & predicted on (1/pt)

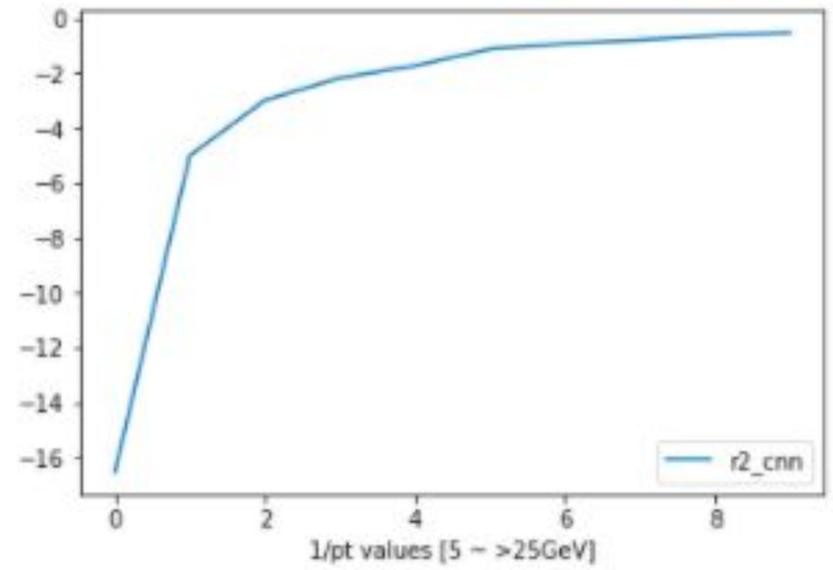
```
bin 0 ---> 2.0000038 GeV to 5 GeV
bin 1 ---> 5 GeV to 10 GeV
bin 2 ---> 10 GeV to 15 GeV
bin 3 ---> 15 GeV to 20 GeV
bin 4 ---> 20 GeV to 25 GeV
bin 5 ---> 25 GeV to 40 GeV
bin 6 ---> 40 GeV to 50 GeV
bin 7 ---> 50 GeV to 60 GeV
bin 8 ---> 60 GeV to 80 GeV
bin 9 ---> 80 GeV to 100 GeV
bin 10 ---> 100 GeV to 6955.571 GeV
```

```
153]: test_loss, test_acc = model.evaluate(X_test,  y_test, verbose=2)
      print((test_acc*100))
      print(test_loss)
      print("R2 score =", round(sm.r2_score(y_test, model3.predict(X_test)), 4))
```

```
7371/7371 - 10s - loss: 3.3616 - accuracy: 0.0000e+00
0.0
3.3616254329681396
R2 score = 0.6585
```

predicted (pt)



# KEY TAKEWAYS

- MAE ~ 3 GEV- 12 GEV
- PREDICTING 1/PT GIVES BETTER R2 VALUES (>0.9)
- CNN 2D GAVE HIGHEST R2 ~ 0.99 (PREDICTED ON 1/PT)