```
import pandas as pd
import numpy as np


import pyarrow.parquet as pq

def read_file(path):
  chunk_size = 25

# Create a Parquet file reader object
  parquet_file = pq.ParquetFile(path)

# Determine the total number of rows in the file
  total_rows = parquet_file.metadata.num_rows

# Loop over the file in chunks
  data = []
  for i in range(0, total_rows, chunk_size):
    # Read a chunk of rows from the file
      chunk = (parquet_file.read_row_group(i))
      dm = (chunk.to_pandas())
#     print(i)
      data.append(dm)

# Concatenate all the DataFrames into a single DataFrame
  df = pd.concat(data, ignore_index=True)
  print(parquet_file.read_row_group(0).to_pandas())
  return df


df1 = read_file('/content/drive/MyDrive/download/QCDToGGQQ_IMGjet_RH1all_jet0_run0_n36272.test.snappy.parquet')
df2 = read_file('/content/drive/MyDrive/download/QCDToGGQQ_IMGjet_RH1all_jet0_run1_n47540.test.snappy.parquet')
df3 = read_file('/content/drive/MyDrive/download/QCDToGGQQ_IMGjet_RH1all_jet0_run2_n55494.test.snappy.parquet')
```

```
                                              X_jets          pt         m0  \
0  [[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...  112.411095  21.098248

     y
0  0.0
                                              X_jets          pt         m0  \
0  [[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...  147.686737  32.114449

     y
0  0.0
                                              X_jets          pt         m0  \
0  [[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...  107.854118  18.723455

     y
0  0.0
```

```
df = pd.concat([df1,df2,df3],ignore_index=True)
del [[df1,df2,df3]]


def to_3d(arr):
    x_jets=[]
    for i in range (0,3):
        jets=np.stack(np.stack(arr)[i],axis=-1)
        x_jets.append(jets)
    x_jets=np.array(x_jets)
    return x_jets


data_img = []
for i in range (0,5573):
   data_img.append(np.transpose(to_3d(df['X_jets'][i])))

data_img = np.asarray(data_img)


df = df.drop(['X_jets'],axis=1)
y = df['y'].values


from sklearn.model_selection import train_test_split
x_train, X_test, y_train, Y_test = train_test_split(data_img,y,test_size=0.2,random_state=42)
x_test, x_val, y_test, y_val = train_test_split(X_test,Y_test,test_size=0.5,random_state=42)


from tensorflow.keras.layers import Input, Conv2D, BatchNormalization, Activation, Add, MaxPooling2D, Flatten, Dense
from tensorflow.keras.models import Model

def res_block(input_data, filters, stride):
    x = Conv2D(filters, kernel_size=3, strides=stride, padding='same')(input_data)
```

```
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv2D(filters, kernel_size=3, strides=1, padding='same')(x)
    x = BatchNormalization()(x)

    shortcut = input_data
    if stride != 1 or input_data.shape[-1] != filters:
        shortcut = Conv2D(filters, kernel_size=1, strides=stride)(input_data)
        shortcut = BatchNormalization()(shortcut)

    x = Add()([x, shortcut])
    x = Activation('relu')(x)
    return x

def build_resnet():
    input_layer = Input(shape=(125, 125, 3))
    x = Conv2D(32, kernel_size=3, strides=1, padding='same')(input_layer)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

    # Add 3 residual blocks
    x = res_block(x, filters=32, stride=1)
    x = res_block(x, filters=32, stride=1)
    x = res_block(x, filters=32, stride=1)

    x = MaxPooling2D(pool_size=(2, 2))(x)

    # Add 3 more residual blocks
    x = res_block(x, filters=64, stride=1)
    x = res_block(x, filters=64, stride=1)
    x = res_block(x, filters=64, stride=1)

    x = MaxPooling2D(pool_size=(2, 2))(x)

    # Add 3 more residual blocks
    x = res_block(x, filters=128, stride=1)
    x = res_block(x, filters=128, stride=1)
    x = res_block(x, filters=128, stride=1)

    x = MaxPooling2D(pool_size=(2, 2))(x)

    x = Flatten()(x)
    x = Dense(256, activation='relu')(x)
    x = Dense(128, activation='relu')(x)
    output_layer = Dense(1, activation='sigmoid')(x)

    model = Model(inputs=input_layer, outputs=output_layer)
    return model

model = build_resnet()
model.summary()
```

```
 conv2d_20 (Conv2D)             (None, 31, 31, 128)  147584      ['activation_17[0][0]']

 batch_normalization_20 (BatchN  (None, 31, 31, 128)  512        ['conv2d_20[0][0]']
 ormalization)

 add_8 (Add)                    (None, 31, 31, 128)  0           ['batch_normalization_20[0][0]',
                                                                  'activation_16[0][0]']

 activation_18 (Activation)     (None, 31, 31, 128)  0           ['add_8[0][0]']

 max_pooling2d_2 (MaxPooling2D)  (None, 15, 15, 128)  0          ['activation_18[0][0]']

 flatten (Flatten)              (None, 28800)        0           ['max_pooling2d_2[0][0]']

 dense (Dense)                  (None, 256)          7373056     ['flatten[0][0]']

 dense_1 (Dense)                (None, 128)          32896       ['dense[0][0]']

 dense_2 (Dense)                (None, 1)            129         ['dense_1[0][0]']

==================================================================================================
Total params: 8,494,081
Trainable params: 8,490,945
Non-trainable params: 3,136
```

```
from keras.optimizers import Adam

model.compile(loss='binary_crossentropy',
              optimizer=Adam(learning_rate=0.0001),
              metrics=['accuracy'])
model.fit(x_train,y_train,validation_data=(x_val,y_val),epochs=20)
```

```
Epoch 1/20
140/140 [==============================] - 33s 219ms/step - loss: 0.4759 - accuracy: 0.7705 - val_loss: 0.6307 - val_accu
Epoch 2/20
140/140 [==============================] - 31s 220ms/step - loss: 0.4281 - accuracy: 0.7945 - val_loss: 0.6764 - val_accu
Epoch 3/20
140/140 [==============================] - 30s 213ms/step - loss: 0.3870 - accuracy: 0.8208 - val_loss: 0.7103 - val_accu
Epoch 4/20
140/140 [==============================] - 30s 216ms/step - loss: 0.3351 - accuracy: 0.8544 - val_loss: 0.7865 - val_accu
Epoch 5/20
140/140 [==============================] - 30s 216ms/step - loss: 0.2839 - accuracy: 0.8809 - val_loss: 0.7891 - val_accu
Epoch 6/20
140/140 [==============================] - 30s 217ms/step - loss: 0.2172 - accuracy: 0.9136 - val_loss: 0.9699 - val_accu
Epoch 7/20
140/140 [==============================] - 30s 216ms/step - loss: 0.1707 - accuracy: 0.9379 - val_loss: 0.9999 - val_accu
Epoch 8/20
140/140 [==============================] - 30s 218ms/step - loss: 0.1100 - accuracy: 0.9625 - val_loss: 1.4067 - val_accu
Epoch 9/20
140/140 [==============================] - 30s 216ms/step - loss: 0.0796 - accuracy: 0.9758 - val_loss: 1.2394 - val_accu
Epoch 10/20
140/140 [==============================] - 31s 218ms/step - loss: 0.0536 - accuracy: 0.9830 - val_loss: 1.6278 - val_accu
Epoch 11/20
140/140 [==============================] - 30s 216ms/step - loss: 0.0369 - accuracy: 0.9917 - val_loss: 1.5671 - val_accu
Epoch 12/20
140/140 [==============================] - 30s 216ms/step - loss: 0.0168 - accuracy: 0.9973 - val_loss: 1.8043 - val_accu
Epoch 13/20
140/140 [==============================] - 30s 218ms/step - loss: 0.0083 - accuracy: 0.9989 - val_loss: 2.0898 - val_accu
Epoch 14/20
140/140 [==============================] - 30s 216ms/step - loss: 0.0170 - accuracy: 0.9964 - val_loss: 2.0307 - val_accu
Epoch 15/20
140/140 [==============================] - 30s 216ms/step - loss: 0.0290 - accuracy: 0.9926 - val_loss: 1.7104 - val_accu
Epoch 16/20
140/140 [==============================] - 31s 218ms/step - loss: 0.0271 - accuracy: 0.9917 - val_loss: 1.8290 - val_accu
Epoch 17/20
140/140 [==============================] - 31s 220ms/step - loss: 0.0093 - accuracy: 0.9984 - val_loss: 1.9989 - val_accu
Epoch 18/20
140/140 [==============================] - 30s 217ms/step - loss: 0.0093 - accuracy: 0.9984 - val_loss: 2.9420 - val_accu
Epoch 19/20
140/140 [==============================] - 31s 218ms/step - loss: 0.0323 - accuracy: 0.9917 - val_loss: 1.7441 - val_accu
Epoch 20/20
140/140 [==============================] - 30s 217ms/step - loss: 0.0132 - accuracy: 0.9971 - val_loss: 2.0963 - val_accu
<keras.callbacks.History at 0x7fa42b2e14c0>
```

```
from sklearn.metrics import roc_auc_score
pred_prob = model.predict(x_test)
auc_score = roc_auc_score(y_test, pred_prob[:])
auc_score
```

```
18/18 [==============================] - 2s 78ms/step
0.7153596545661391
```

We can get even more higher accuracy if we concatenate the output of our resnet model for the image with the other two energy criteria given in the dataset and then running it from a simple neural network with 2 or 3 hidden layers with relu but I can't achieve that due to my limitation of computational resources