Deep Learning based Quark-Gluon Classification

In [1]:
```
# import necessary libraries
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import pyarrow.parquet as pq
import pandas as pd
import os
from sklearn.preprocessing import StandardScaler
```

In [2]:
```
DATA_DIR = '/kaggle/input/ml4sci'
```

In [3]:
```
chunk_size = 50

def get_dataset(data_dir):
    dfs = []
    for file_name in os.listdir(data_dir):
        parquet_file = pq.ParquetFile(f'{DATA_DIR}/{file_name}')
        total_rows = parquet_file.metadata.num_rows

        for i in range(0, total_rows, chunk_size):
            chunk = parquet_file.read_row_group(i)
            df = chunk.to_pandas()
            dfs.append(df)

    dataset = pd.concat(dfs, ignore_index=True)
    dataset['y'] = dataset['y'].astype(int)
    return dataset


dataset = get_dataset(DATA_DIR)
dataset['pt'] = StandardScaler().fit_transform(dataset[['pt']])
dataset['m0'] = StandardScaler().fit_transform(dataset[['m0']])
print('Dataset Length:', len(dataset))
dataset.head()
```

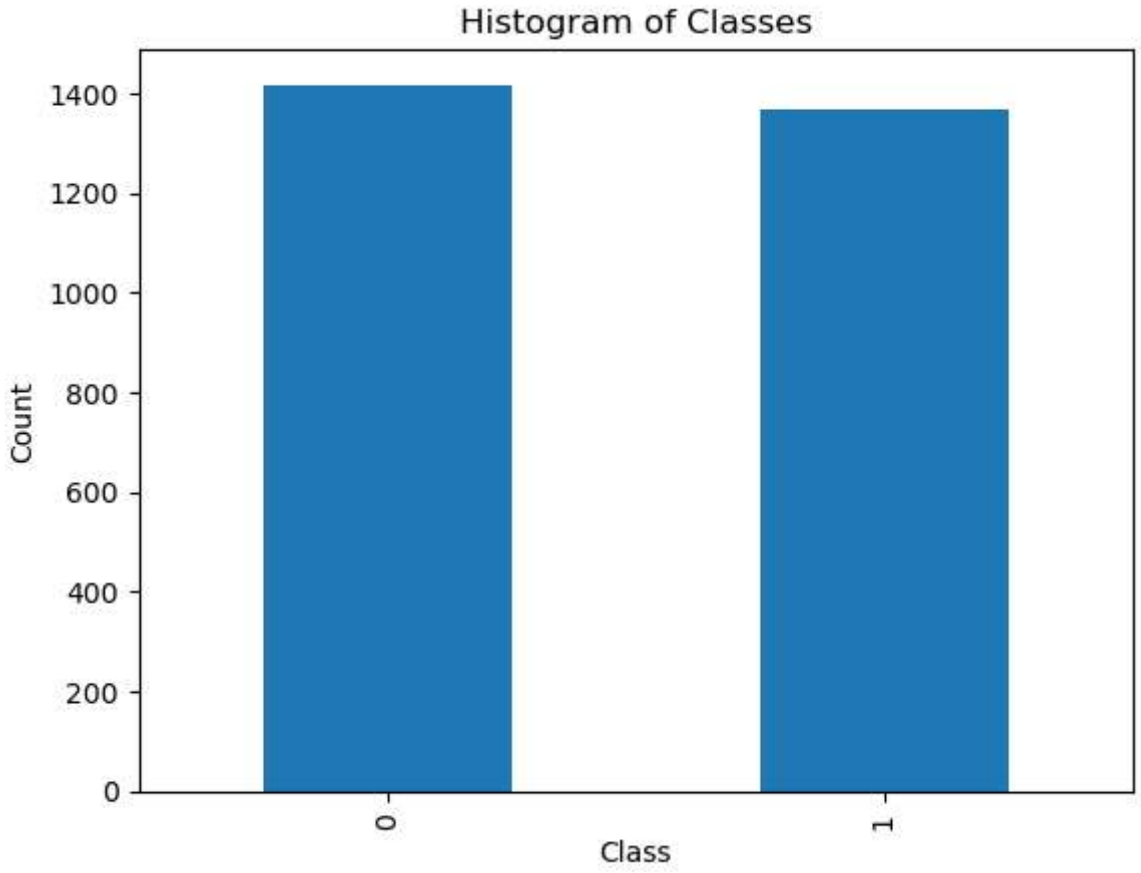
```
Dataset Length: 2787
```

Out[3]:

	X_jets	pt	m0	y
0	[[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...	-0.173715	-0.045484	0
1	[[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...	-1.226041	-0.832781	0
2	[[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...	-1.010251	-0.045075	0
3	[[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...	1.094995	2.271322	0
4	[[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0...	-1.550861	-0.575182	0

In [4]:

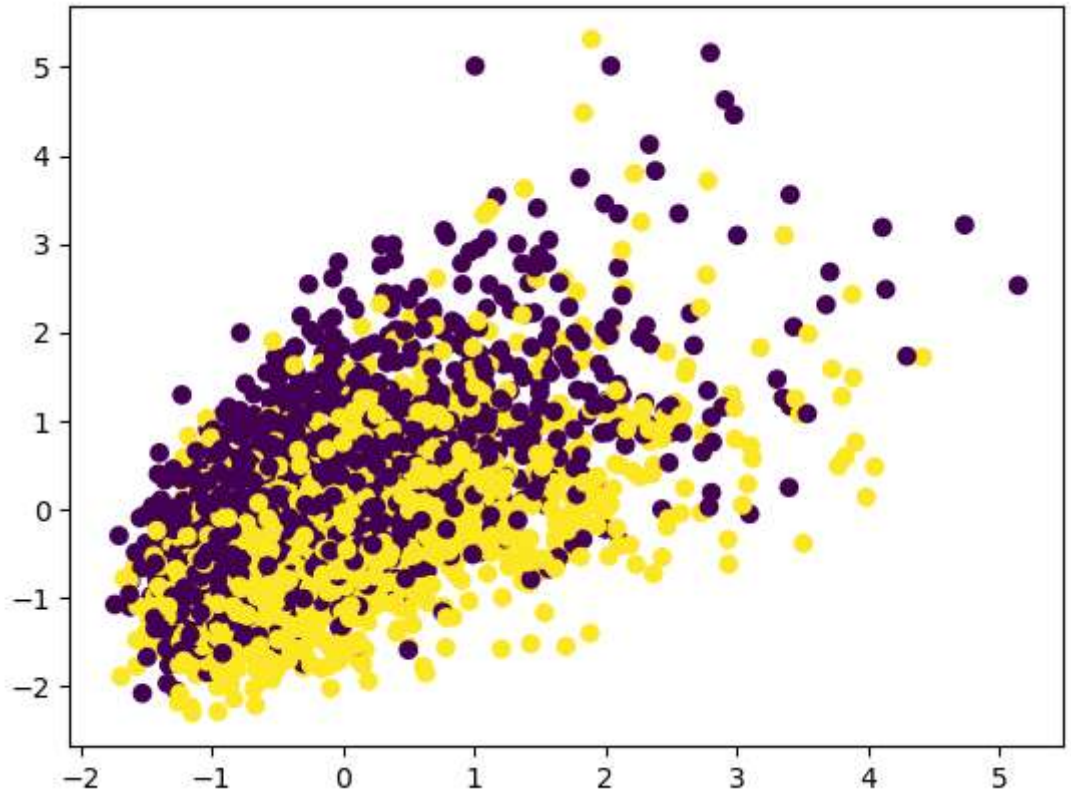
```
# check for class imbalance
counts = dataset['y'].value_counts()
counts.plot(kind='bar')
plt.xlabel('Class')
plt.ylabel('Count')
plt.title('Histogram of Classes')
plt.show()
```



No Class Imbalance!

In [5]:

```
# check for co-relation between pt and m0 given the `y` column
plt.scatter(dataset['pt'], dataset['m0'], c=dataset['y'])
plt.show()
```



In [6]:

```
def to_numpy_array(inp):
    arr = []
    for i in range(0, 3):
        vis = np.stack(np.stack(inp)[i], axis=-1)
        arr.append(vis)

    arr = np.array(arr)
    arr = arr.reshape((125, 125, 3))
    return arr

dataset['X_jets'] = dataset['X_jets'].apply(to_numpy_array)
```

In [7]:

```
from sklearn.model_selection import train_test_split

train_df, test_df = train_test_split(dataset, test_size=0.2, random_state=42)
```

In [8]:

```
BATCH_SIZE = 64

train_dataset_1 = tf.data.Dataset.from_tensor_slices(
    (list(train_df.X_jets), list(train_df.y))
).batch(BATCH_SIZE)
train_dataset_1 = train_dataset_1.shuffle(len(train_dataset_1))

test_dataset_1 = tf.data.Dataset.from_tensor_slices(
    (list(test_df.X_jets), list(test_df.y))
).batch(BATCH_SIZE)
```

In [9]:

```
# creating model
model_1 = tf.keras.Sequential([
    tf.keras.layers.Conv2D(filters=1, kernel_size=(3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2,2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(units=1, activation='sigmoid')
])

model_1.compile(
    optimizer=tf.keras.optimizers.Adam(),
    loss=tf.keras.losses.BinaryCrossentropy(),
    metrics=['accuracy', tf.keras.metrics.AUC(curve='ROC')]
)
```

In [10]:

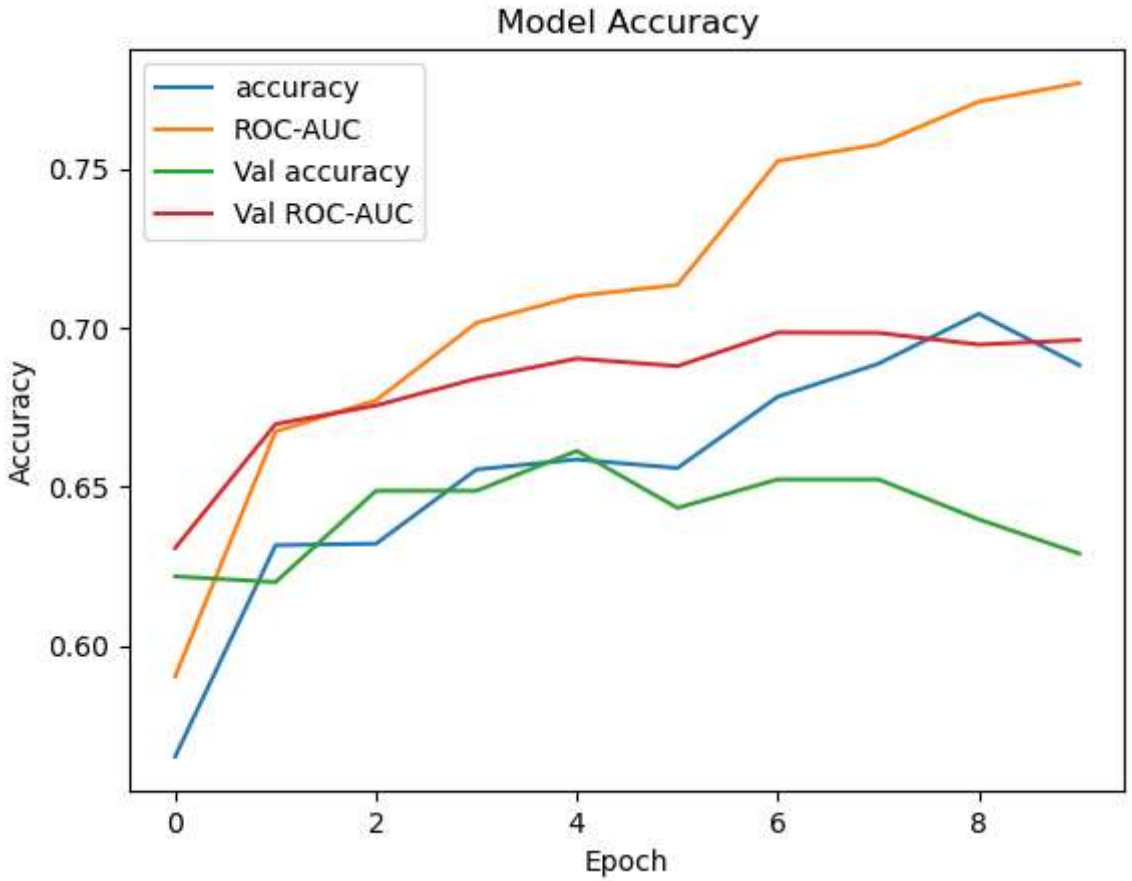
```
history_1 = model_1.fit(
    train_dataset_1,
    validation_data=test_dataset_1,
    epochs=10
)
```

```
Epoch 1/10
35/35 [==============================] - 9s 37ms/step - loss: 0.6873 - accuracy: 0.5653 - auc: 0.5905 - val_loss: 0.6736
- val_accuracy: 0.6219 - val_auc: 0.6308
Epoch 2/10
35/35 [==============================] - 2s 23ms/step - loss: 0.6542 - accuracy: 0.6317 - auc: 0.6674 - val_loss: 0.6570
- val_accuracy: 0.6201 - val_auc: 0.6697
Epoch 3/10
35/35 [==============================] - 1s 16ms/step - loss: 0.6481 - accuracy: 0.6321 - auc: 0.6771 - val_loss: 0.6543
- val_accuracy: 0.6487 - val_auc: 0.6755
```

```
Epoch 4/10
35/35 [==============================] - 1s 16ms/step - loss: 0.6309 - accuracy: 0.6555 - auc: 0.7015 - val_loss: 0.6473
- val_accuracy: 0.6487 - val_auc: 0.6840
Epoch 5/10
35/35 [==============================] - 1s 21ms/step - loss: 0.6252 - accuracy: 0.6586 - auc: 0.7100 - val_loss: 0.6458
- val_accuracy: 0.6613 - val_auc: 0.6903
Epoch 6/10
35/35 [==============================] - 1s 21ms/step - loss: 0.6229 - accuracy: 0.6559 - auc: 0.7134 - val_loss: 0.6484
- val_accuracy: 0.6434 - val_auc: 0.6879
Epoch 7/10
35/35 [==============================] - 1s 21ms/step - loss: 0.5990 - accuracy: 0.6783 - auc: 0.7524 - val_loss: 0.6407
- val_accuracy: 0.6523 - val_auc: 0.6985
Epoch 8/10
35/35 [==============================] - 1s 22ms/step - loss: 0.6061 - accuracy: 0.6886 - auc: 0.7576 - val_loss: 0.6401
- val_accuracy: 0.6523 - val_auc: 0.6984
Epoch 9/10
35/35 [==============================] - 1s 16ms/step - loss: 0.5816 - accuracy: 0.7044 - auc: 0.7710 - val_loss: 0.6418
- val_accuracy: 0.6398 - val_auc: 0.6947
Epoch 10/10
35/35 [==============================] - 1s 16ms/step - loss: 0.5730 - accuracy: 0.6882 - auc: 0.7768 - val_loss: 0.6438
- val_accuracy: 0.6290 - val_auc: 0.6961
```

In [12]:

```
plt.title('Model Accuracy')
plt.plot(history_1.history['accuracy'], label='accuracy')
plt.plot(history_1.history['auc'], label='ROC-AUC')
plt.plot(history_1.history['val_accuracy'], label='Val accuracy')
plt.plot(history_1.history['val_auc'], label='Val ROC-AUC')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend()
plt.show()
```



ROC AUC Score (Train): 0.77

ROC AUC Score (Test) : 0.70

Even after training the model with CNN filters = 1 and dropout = 0.5 the model is slightly overfitting.

*The values above may slightly change with each run

Using pt and m0

Now I will use `pt` and `m0` to predict the `y`

In [13]:
```
train_dataset_2 = tf.data.Dataset.from_tensor_slices(
    (train_df[['pt', 'm0']].values, list(train_df.y))
).batch(BATCH_SIZE)
train_dataset_2 = train_dataset_2.shuffle(len(train_dataset_2))

test_dataset_2 = tf.data.Dataset.from_tensor_slices(
    (test_df[['pt', 'm0']].values, list(test_df.y))
).batch(BATCH_SIZE)
```

In [14]:
```
# creating model

model_2 = tf.keras.Sequential([
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

model_2.compile(
    optimizer=tf.keras.optimizers.Adam(),
    loss=tf.keras.losses.BinaryCrossentropy(),
    metrics=['accuracy', tf.keras.metrics.AUC(curve='ROC')]
)
```

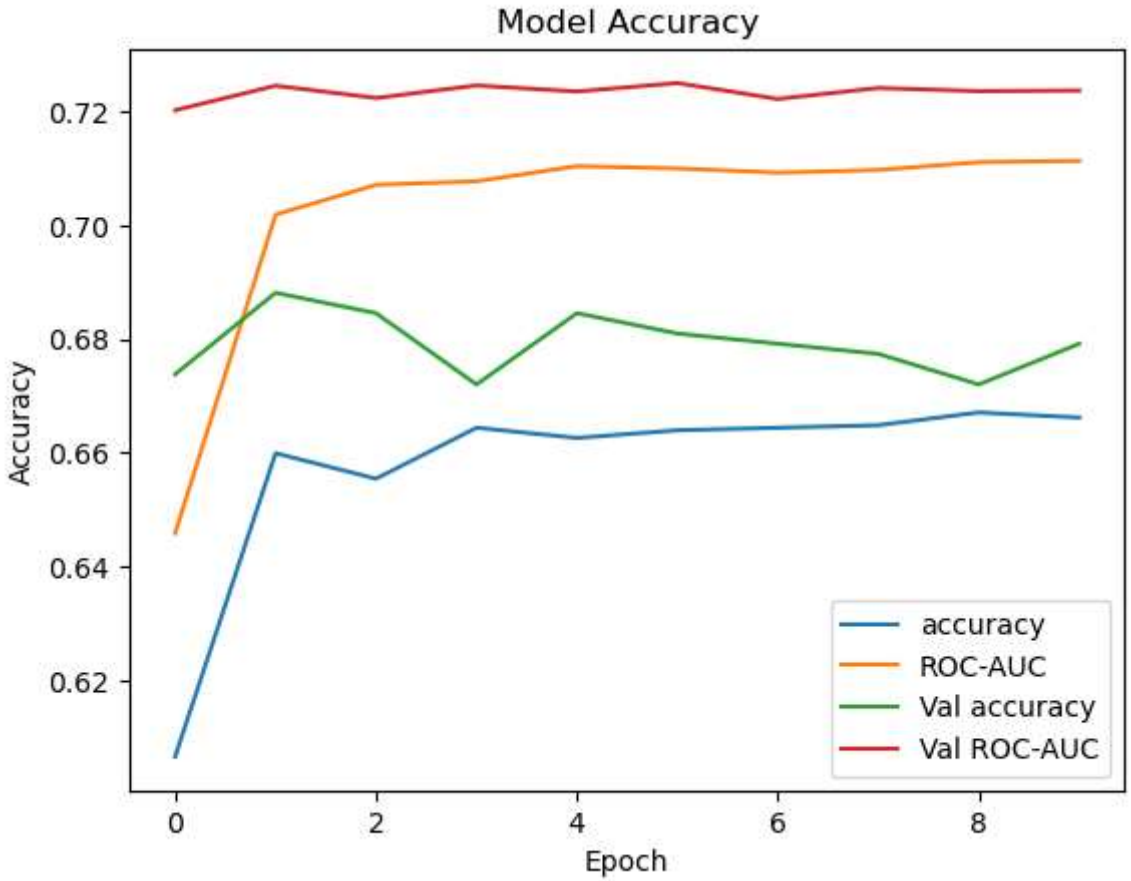
In [15]:
```
history_2 = model_2.fit(
    train_dataset_2,
    validation_data=test_dataset_2,
    epochs=10
)
```

```
Epoch 1/10
35/35 [==============================] - 2s 10ms/step - loss: 0.6735 - accuracy: 0.6066 - auc_1: 0.6459 - val_loss: 0.639
5 - val_accuracy: 0.6738 - val_auc_1: 0.7203
Epoch 2/10
35/35 [==============================] - 0s 5ms/step - loss: 0.6347 - accuracy: 0.6599 - auc_1: 0.7019 - val_loss: 0.6139
- val_accuracy: 0.6882 - val_auc_1: 0.7246
Epoch 3/10
35/35 [==============================] - 0s 4ms/step - loss: 0.6254 - accuracy: 0.6555 - auc_1: 0.7072 - val_loss: 0.6123
- val_accuracy: 0.6846 - val_auc_1: 0.7225
Epoch 4/10
```

```
35/35 [==============================] - 0s 4ms/step - loss: 0.6243 - accuracy: 0.6644 - auc_1: 0.7078 - val_loss: 0.6132
- val_accuracy: 0.6720 - val_auc_1: 0.7247
Epoch 5/10
35/35 [==============================] - 0s 4ms/step - loss: 0.6225 - accuracy: 0.6626 - auc_1: 0.7105 - val_loss: 0.6120
- val_accuracy: 0.6846 - val_auc_1: 0.7236
Epoch 6/10
35/35 [==============================] - 0s 9ms/step - loss: 0.6230 - accuracy: 0.6640 - auc_1: 0.7101 - val_loss: 0.6110
- val_accuracy: 0.6810 - val_auc_1: 0.7251
Epoch 7/10
35/35 [==============================] - 0s 5ms/step - loss: 0.6225 - accuracy: 0.6644 - auc_1: 0.7093 - val_loss: 0.6135
- val_accuracy: 0.6792 - val_auc_1: 0.7223
Epoch 8/10
35/35 [==============================] - 0s 4ms/step - loss: 0.6227 - accuracy: 0.6649 - auc_1: 0.7098 - val_loss: 0.6116
- val_accuracy: 0.6774 - val_auc_1: 0.7242
Epoch 9/10
35/35 [==============================] - 0s 6ms/step - loss: 0.6219 - accuracy: 0.6671 - auc_1: 0.7112 - val_loss: 0.6138
- val_accuracy: 0.6720 - val_auc_1: 0.7237
Epoch 10/10
35/35 [==============================] - 0s 4ms/step - loss: 0.6215 - accuracy: 0.6662 - auc_1: 0.7114 - val_loss: 0.6126
- val_accuracy: 0.6792 - val_auc_1: 0.7238
```

In [17]:

```
plt.title('Model Accuracy')
plt.plot(history_2.history['accuracy'], label='accuracy')
plt.plot(history_2.history['auc_1'], label='ROC-AUC')
plt.plot(history_2.history['val_accuracy'], label='Val accuracy')
plt.plot(history_2.history['val_auc_1'], label='Val ROC-AUC')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend()
plt.show()
```



ROC AUC Score (Train): 0.71

ROC AUC Score (Test) : 0.72

No Overfitting!