In [1]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np

import h5py
```

In [2]:
```
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
```

```
Num GPUs Available:  1
```

In [3]:
```
print(tf.__version__)
```

```
2.10.0
```

In [4]:
```
sys_details = tf.sysconfig.get_build_info()
sys_details["cuda_version"]
```

Out[4]:
```
'64_112'
```

In [5]:
```
photon_file_path = "SinglePhotonPt50_IMGCROPS_n249k_RHv1.hdf5"
electron_file_path = "SingleElectronPt50_IMGCROPS_n249k_RHv1.hdf5"

# Load data from the provided HDF5 files
with h5py.File(photon_file_path, 'r') as f:
    X_photon = f['X'][:]
    y_photon = f['y'][:]

with h5py.File(electron_file_path, 'r') as f:
    X_electron = f['X'][:]
    y_electron = f['y'][:]

# Concatenate the datasets
X = np.concatenate([X_photon, X_electron], axis=0)
y = np.concatenate([y_photon, y_electron], axis=0)

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Convert labels to one-hot encoding
y_train = tf.keras.utils.to_categorical(y_train, num_classes=2)
y_test = tf.keras.utils.to_categorical(y_test, num_classes=2)
```

In [6]:
```
from tensorflow.keras import layers

class ResBlock(tf.keras.Model):
    def __init__(self, in_channels, out_channels, stride=1):
        super(ResBlock, self).__init__()
        self.conv1 = layers.Conv2D(out_channels, kernel_size=3, strides=stride, pac
        self.bn1 = layers.BatchNormalization()
        self.conv2 = layers.Conv2D(out_channels, kernel_size=3, strides=1, padding=
        self.bn2 = layers.BatchNormalization()
        self.shortcut = tf.keras.Sequential()
        if stride != 1 or in_channels != out_channels:
            self.shortcut = tf.keras.Sequential([
                layers.Conv2D(out_channels, kernel_size=1, strides=stride, use_bias
                layers.BatchNormalization()
            ])

    def call(self, x):
        out = self.conv1(x)
        out = self.bn1(out)
        out = tf.nn.relu(out)

        out = self.conv2(out)
        out = self.bn2(out)

        out += self.shortcut(x)
        out = tf.nn.relu(out)
        return out


class ResNet(tf.keras.Model):
    def __init__(self, num_classes=2):
        super(ResNet, self).__init__()
        self.in_channels = 16

        self.conv1 = layers.Conv2D(16, kernel_size=3, strides=1, padding="same", us
        self.bn1 = layers.BatchNormalization()
        self.relu = tf.nn.relu
        self.layer1 = self._make_layer(ResBlock, 16, 2, stride=1)
        self.layer2 = self._make_layer(ResBlock, 32, 2, stride=2)
        self.layer3 = self._make_layer(ResBlock, 64, 2, stride=2)
        self.avgpool = layers.GlobalAveragePooling2D()
        self.fc = layers.Dense(num_classes, activation='softmax')

    def _make_layer(self, block, out_channels, blocks, stride=1):
        layers = []
        layers.append(block(self.in_channels, out_channels, stride))
        self.in_channels = out_channels
        for i in range(1, blocks):
            layers.append(block(out_channels, out_channels))
        return tf.keras.Sequential(layers)

    def call(self, x):
        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)

        out = self.layer1(out)
        out = self.layer2(out)
        out = self.layer3(out)

        out = self.avgpool(out)
        out = self.fc(out)

        return out
```

In [ ]:
```
# Create the model
model = ResNet()
# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', metr

# Train the model
model.fit(X_train, y_train, batch_size=32, epochs=30, validation_split=0.1)
```

```
Epoch 1/30
11205/11205 [==============================] - 113s 10ms/step - loss: 0.6090 - ac
curacy: 0.6729 - val_loss: 0.5687 - val_accuracy: 0.7179
Epoch 2/30
11205/11205 [==============================] - 109s 10ms/step - loss: 0.5690 - ac
curacy: 0.7133 - val_loss: 0.5516 - val_accuracy: 0.7243
Epoch 3/30
11205/11205 [==============================] - 109s 10ms/step - loss: 0.5582 - ac
curacy: 0.7212 - val_loss: 0.5681 - val_accuracy: 0.7169
Epoch 4/30
11205/11205 [==============================] - 112s 10ms/step - loss: 0.5525 - ac
curacy: 0.7255 - val_loss: 0.5728 - val_accuracy: 0.7085
Epoch 5/30
11205/11205 [==============================] - 110s 10ms/step - loss: 0.5487 - ac
curacy: 0.7276 - val_loss: 0.5521 - val_accuracy: 0.7241
Epoch 6/30
11205/11205 [==============================] - 109s 10ms/step - loss: 0.5454 - ac
curacy: 0.7298 - val_loss: 0.5437 - val_accuracy: 0.7311
Epoch 7/30
11205/11205 [==============================] - 110s 10ms/step - loss: 0.5429 - ac
curacy: 0.7318 - val_loss: 0.5483 - val_accuracy: 0.7286
Epoch 8/30
11205/11205 [==============================] - 110s 10ms/step - loss: 0.5409 - ac
curacy: 0.7336 - val_loss: 0.5505 - val_accuracy: 0.7252
Epoch 9/30
11205/11205 [==============================] - 113s 10ms/step - loss: 0.5389 - ac
curacy: 0.7352 - val_loss: 0.5607 - val_accuracy: 0.7155
Epoch 10/30
11205/11205 [==============================] - 112s 10ms/step - loss: 0.5374 - ac
curacy: 0.7356 - val_loss: 0.5352 - val_accuracy: 0.7384
Epoch 11/30
11205/11205 [==============================] - 112s 10ms/step - loss: 0.5354 - ac
curacy: 0.7373 - val_loss: 0.5423 - val_accuracy: 0.7303
Epoch 12/30
11205/11205 [==============================] - 110s 10ms/step - loss: 0.5339 - ac
curacy: 0.7378 - val_loss: 0.5348 - val_accuracy: 0.7377
Epoch 13/30
11205/11205 [==============================] - 119s 11ms/step - loss: 0.5328 - ac
curacy: 0.7384 - val_loss: 0.5403 - val_accuracy: 0.7351
Epoch 14/30
11205/11205 [==============================] - 112s 10ms/step - loss: 0.5315 - ac
curacy: 0.7395 - val_loss: 0.5363 - val_accuracy: 0.7373
Epoch 15/30
11205/11205 [==============================] - 112s 10ms/step - loss: 0.5302 - ac
curacy: 0.7400 - val_loss: 0.5307 - val_accuracy: 0.7421
Epoch 16/30
11205/11205 [==============================] - 112s 10ms/step - loss: 0.5288 - ac
curacy: 0.7414 - val_loss: 0.5333 - val_accuracy: 0.7386
Epoch 17/30
11205/11205 [==============================] - 112s 10ms/step - loss: 0.5277 - ac
curacy: 0.7420 - val_loss: 0.5334 - val_accuracy: 0.7400
Epoch 18/30
11205/11205 [==============================] - 111s 10ms/step - loss: 0.5263 - ac
curacy: 0.7431 - val_loss: 0.5327 - val_accuracy: 0.7392
```

```
Epoch 19/30
11205/11205 [==============================] - 110s 10ms/step - loss: 0.5249 - ac
curacy: 0.7445 - val_loss: 0.5382 - val_accuracy: 0.7351
Epoch 20/30
11205/11205 [==============================] - 110s 10ms/step - loss: 0.5236 - ac
curacy: 0.7449 - val_loss: 0.5373 - val_accuracy: 0.7356
Epoch 21/30
11205/11205 [==============================] - 109s 10ms/step - loss: 0.5224 - ac
curacy: 0.7458 - val_loss: 0.5324 - val_accuracy: 0.7394
Epoch 22/30
11205/11205 [==============================] - 109s 10ms/step - loss: 0.5209 - ac
curacy: 0.7470 - val_loss: 0.5318 - val_accuracy: 0.7397
Epoch 23/30
11205/11205 [==============================] - 110s 10ms/step - loss: 0.5193 - ac
curacy: 0.7472 - val_loss: 0.5335 - val_accuracy: 0.7412
Epoch 24/30
11205/11205 [==============================] - 109s 10ms/step - loss: 0.5179 - ac
curacy: 0.7486 - val_loss: 0.5331 - val_accuracy: 0.7388
Epoch 25/30
11205/11205 [==============================] - 111s 10ms/step - loss: 0.5161 - ac
curacy: 0.7506 - val_loss: 0.5373 - val_accuracy: 0.7370
Epoch 26/30
11205/11205 [==============================] - 110s 10ms/step - loss: 0.5148 - ac
curacy: 0.7508 - val_loss: 0.5379 - val_accuracy: 0.7389
Epoch 27/30
11205/11205 [==============================] - 110s 10ms/step - loss: 0.5126 - ac
curacy: 0.7517 - val_loss: 0.5357 - val_accuracy: 0.7412
Epoch 28/30
11205/11205 [==============================] - 110s 10ms/step - loss: 0.5109 - ac
curacy: 0.7531 - val_loss: 0.5354 - val_accuracy: 0.7379
Epoch 29/30
11205/11205 [==============================] - 109s 10ms/step - loss: 0.5086 - ac
curacy: 0.7548 - val_loss: 0.5532 - val_accuracy: 0.7282
Epoch 30/30
 5263/11205 [=============>................] - ETA: 57s - loss: 0.5058 - accurac
y: 0.7558
```

In [ ]:

```
# Evaluate the model
test_loss, test_acc = model.evaluate(X_test, y_test)
```

In [ ]:

```
# Get probabilities and true labels
probs = model.predict(X_test,batch_size=32)[:, 1]
```

In [ ]:
```
# Compute ROC curve and AUC score
y_test_original = np.argmax(y_test, axis=1)
fpr, tpr, _ = roc_curve(y_test_original, probs)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure()
lw = 2
plt.plot(fpr, tpr, color='darkorange', lw=lw, label='ROC curve (area = %0.2f)' % rc
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```

In [ ]: