Graph Neural Networks for End-to-End Particle Identification with the CMS Experiment

written by Khalid Bagus Pratama Darmadi, S.Si (Khalid); Papua, Indonesia

for ML4SCI on Google Summer of Code 2023

Common Task 1. Electron/photon classification Datasets:

> https://cernbox.cern.ch/index.php/s/AtBT8y4MiQYFcgc (photons) https://cernbox.cern.ch/index.php/s/FbXw3V4XNyYB3oA (electrons)

Description: 32x32 matrices (two channels - hit energy and time) for two classes of particles electrons and photons impinging on a calorimeter

Please use a deep learning method of your choice to achieve the highest possible classification on this dataset (we ask that you do it both in Keras/Tensorflow and in PyTorch). Please provide a Jupyter notebook that shows your solution. The model you submit should have a ROC AUC score of at least 0.80.

Solution:

For this task, here is the model description for classifying whether the output would be an electron or a photon. The data will be preprocessed with normalization and quantization to 8-bits.

This model combine custom layers and EfficientNetB2 for purpose of effectively learn and classify the features of electrons and photons impinging on a calorimeter. The residual block, channel reduction, and EfficientNetB2 suppose to help the model learn complex features, while the dense layers, dropout, and global average pooling enable it to perform the final classification.

1. Input and concatenation layer

> The input layer accepts a tensor of shape (32, 32, 2), which corresponds to the hit energy and time data. The model then concatenates the input data along the channel axis to create a 3-channel tensor, which is required as input for the EfficientNetB2 model.

1. Residual block layer

> The model incorporates a residual block that consists of two convolutional layers (Conv2D and DepthwiseConv2D) with a GELU (Gaussian Error Linear Unit) activation function, followed by batch normalization layers. The output of the residual block is added back to the original input. This design allows the model to learn complex patterns and feature representations while reducing the risk of vanishing gradients. The GELU activation function improves the model's learning capabilities compared to more traditional activation functions like ReLU.

1. Channel reduction layer

> A 1x1 convolution layer is used to reduce the number of channels in the output tensor of the residual block from 6 to 3. This operation not only reduces the complexity of the model but also allows it to match the input requirements of the EfficientNetB2 model.

1. EfficientNetB2 layer

> This is a state-of-the-art convolutional neural network architecture that has been optimized for better performance and smaller model size. By incorporating EfficientNetB2 into the model, it can effectively learn more complex and abstract features from the input data, leading to better classification performance. The EfficientNetB2 model is pretrained on the ImageNet dataset, which means it already has a good understanding of general features, and this can help in faster convergence and better performance in the specific task of electron/photon classification.

1. Global Average Pooling

> This layer is used to reduce the spatial dimensions of the EfficientNetB2 output. By taking the average value of each feature map, it generates a fixed-length feature vector that can be fed into the subsequent dense layers.

1. Dense layers and Dropout

> The model uses a dense layer with 256 units and a GELU activation function to learn a higher-level representation of the input features. The dropout layer with a rate of 0.5 helps prevent overfitting by randomly setting a fraction of the input units to 0 during training.

2. Output layer

> The final dense layer with one unit and a sigmoid activation function produces a probability value between 0 and 1, representing the likelihood of the input belonging to the positive class (either electron or photon, depending on how the labels are assigned).

In [ ]:

```
import h5py
import pandas as pd
import numpy as np

electron_file = h5py.File('/content/drive/MyDrive/GSoC/task_1/SingleElectronPt50_IMGCROPS_n249k_RHv1.hdf5')
photon_file = h5py.File('/content/drive/MyDrive/GSoC/task_1/SinglePhotonPt50_IMGCROPS_n249k_RHv1.hdf5')

print("Keys in electron_file:")
print(list(electron_file.keys()))
```

```
print("\nKeys in photon_file:")
print(list(photon_file.keys()))
```

```
Keys in electron_file:
['X', 'y']

Keys in photon_file:
['X', 'y']
```

In [ ]:

```
# If the files have a nested structure, we can explore it further like this:

print("\nStructure of electron_file:")
def print_structure(name, obj):
    print(f"{name}: {type(obj)}")

electron_file.visititems(print_structure)

print("\nStructure of photon_file:")
photon_file.visititems(print_structure)
```

```
Structure of electron_file:
X: <class 'h5py._hl.dataset.Dataset'>
y: <class 'h5py._hl.dataset.Dataset'>

Structure of photon_file:
X: <class 'h5py._hl.dataset.Dataset'>
y: <class 'h5py._hl.dataset.Dataset'>
```

In [ ]:

```
# Load the data and labels

electron_X = electron_file['X']
electron_y = electron_file['y']
photon_X = photon_file['X']
photon_y = photon_file['y']
```

In [ ]:

```
# Inspect the shapes

print("Shapes:")
print(f"Electron data (X): {electron_X.shape}")
print(f"Electron labels (y): {electron_y.shape}")
print(f"Photon data (X): {photon_X.shape}")
print(f"Photon labels (y): {photon_y.shape}")
```

```
Shapes:
Electron data (X): (249000, 32, 32, 2)
Electron labels (y): (249000,)
Photon data (X): (249000, 32, 32, 2)
Photon labels (y): (249000,)
```

In [ ]:

```python
# Print the first few samples and labels

print("\nFirst 5 samples and labels for electrons:")
for i in range(5):
    print(f"Sample {i}:")
    print(electron_X[i])
    print(f"Label {i}: {electron_y[i]}")

print("\nFirst 5 samples and labels for photons:")
for i in range(5):
    print(f"Sample {i}:")
    print(photon_X[i])
    print(f"Label {i}: {photon_y[i]}")
```

```
First 5 samples and labels for electrons:
Sample 0:
[[[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]

 [[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]

 [[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
```

```
  [0. 0.]]

 ...

 [[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]

 [[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]

 [[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]]
Label 0: 1.0
Sample 1:
[[[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]

 [[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]
```

```
  [[0. 0.]
   [0. 0.]
   [0. 0.]
   ...
   [0. 0.]
   [0. 0.]
   [0. 0.]]

  ...

  [[0. 0.]
   [0. 0.]
   [0. 0.]
   ...
   [0. 0.]
   [0. 0.]
   [0. 0.]]

  [[0. 0.]
   [0. 0.]
   [0. 0.]
   ...
   [0. 0.]
   [0. 0.]
   [0. 0.]]

  [[0. 0.]
   [0. 0.]
   [0. 0.]
   ...
   [0. 0.]
   [0. 0.]
   [0. 0.]]]
Label 1: 1.0
Sample 2:
[[[0. 0.]
   [0. 0.]
   [0. 0.]
   ...
   [0. 0.]
   [0. 0.]
   [0. 0.]]

  [[0. 0.]
   [0. 0.]
```

```
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]

 [[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]

 ...

 [[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]

 [[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]

 [[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]]
Label 2: 1.0
Sample 3:
[[[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
```

```
 [0. 0.]
 [0. 0.]
 [0. 0.]]

[[0. 0.]
 [0. 0.]
 [0. 0.]
 ...
 [0. 0.]
 [0. 0.]
 [0. 0.]]

[[0. 0.]
 [0. 0.]
 [0. 0.]
 ...
 [0. 0.]
 [0. 0.]
 [0. 0.]]

...

[[0. 0.]
 [0. 0.]
 [0. 0.]
 ...
 [0. 0.]
 [0. 0.]
 [0. 0.]]

[[0. 0.]
 [0. 0.]
 [0. 0.]
 ...
 [0. 0.]
 [0. 0.]
 [0. 0.]]

[[0. 0.]
 [0. 0.]
 [0. 0.]
 ...
 [0. 0.]
 [0. 0.]
 [0. 0.]]]
```

```
Label 3: 1.0
Sample 4:
[[[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]

 [[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]

 [[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]

 ...

 [[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]

 [[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]

 [[0. 0.]
```

```
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]]
Label 4: 1.0

First 5 samples and labels for photons:
Sample 0:
[[[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]

 [[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]

 [[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]

 ...

 [[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]

 [[0. 0.]
```

```
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]

 [[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]]
Label 0: 0.0
Sample 1:
[[[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]

 [[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]

 [[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]

 ...

 [[0. 0.]
  [0. 0.]
  [0. 0.]
```

```
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]

 [[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]

 [[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]]
Label 1: 0.0
Sample 2:
[[[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]

 [[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]

 [[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]
```

```
 ...

 [[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]

 [[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]

 [[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]]
Label 2: 0.0
Sample 3:
[[[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]

 [[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]

 [[0. 0.]
```

```
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]

 ...

 [[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]

 [[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]

 [[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]]
Label 3: 0.0
Sample 4:
[[[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]

 [[0. 0.]
  [0. 0.]
  [0. 0.]
```

```
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]

 [[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]

 ...

 [[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]

 [[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]

 [[0. 0.]
  [0. 0.]
  [0. 0.]
  ...
  [0. 0.]
  [0. 0.]
  [0. 0.]]]
Label 4: 0.0
```

In [ ]:

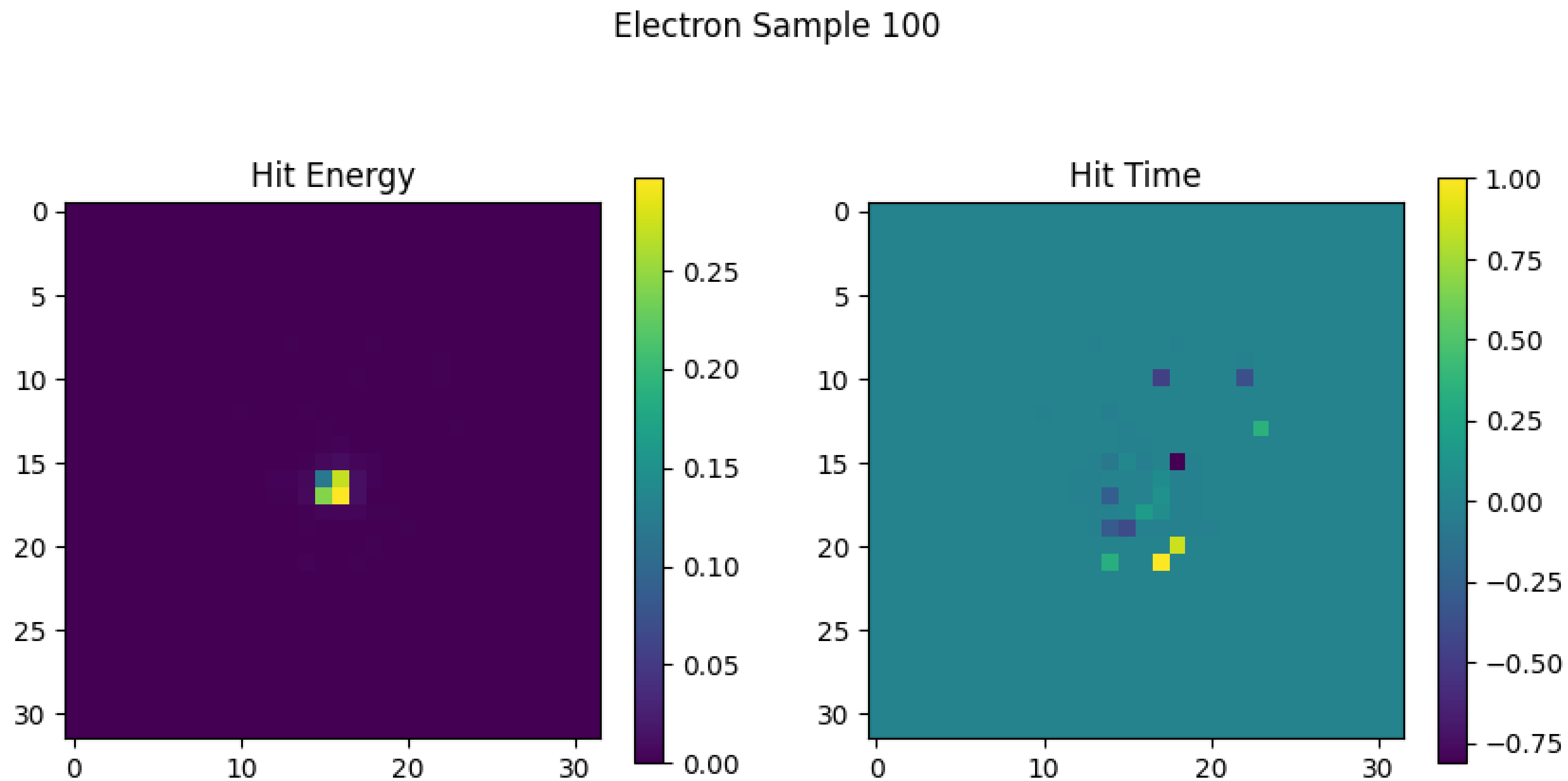
```
import matplotlib.pyplot as plt

electron_X_np = np.array(electron_X)
photon_X_np = np.array(photon_X)
```

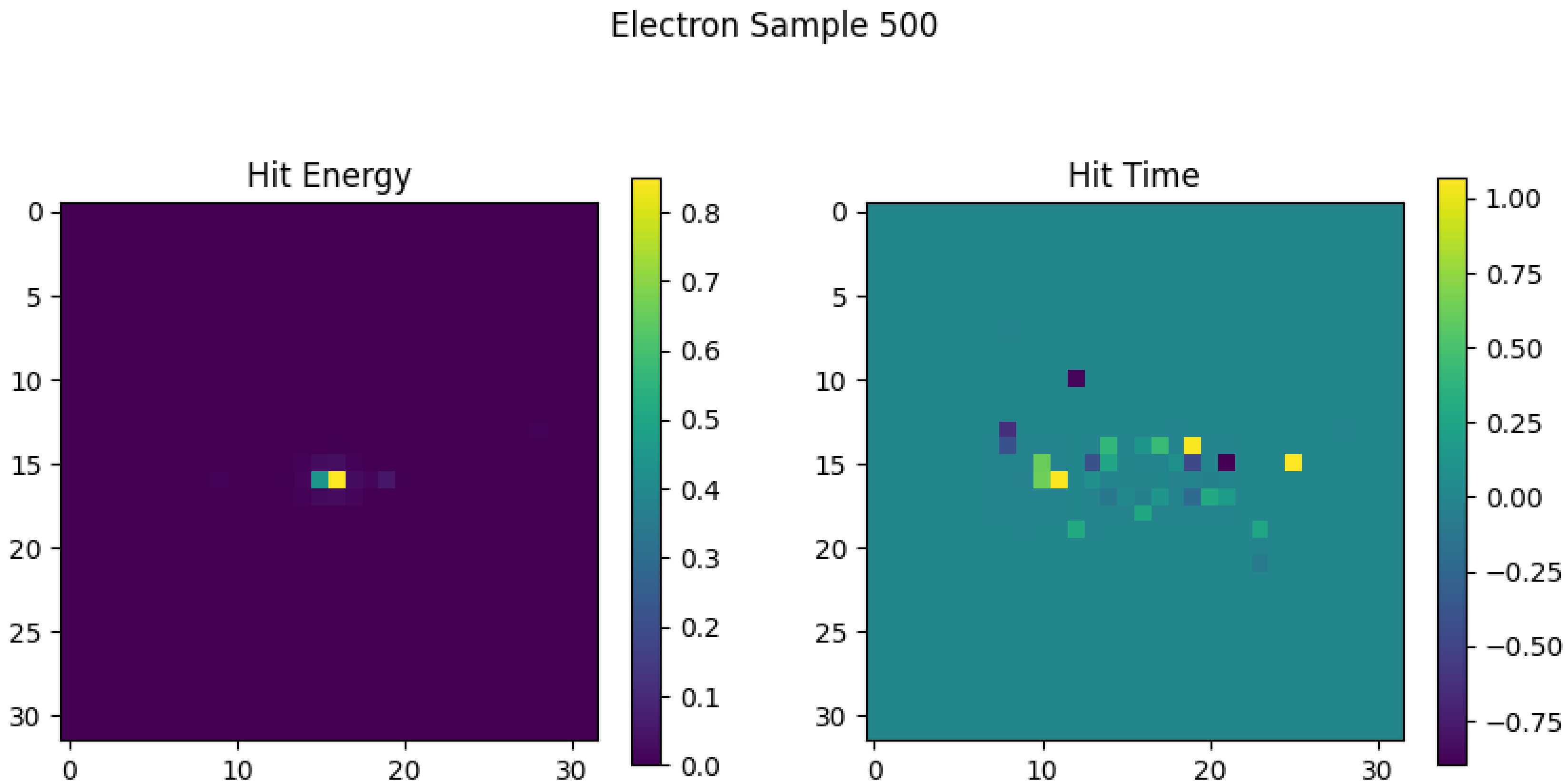
```
def display_statistics(data):
    energy_data = data[:, :, :, 0]
    time_data = data[:, :, :, 1]

    print("Hit Energy Channel:")
    print(f"  Min: {energy_data.min()}")
    print(f"  Max: {energy_data.max()}")
    print(f"  Mean: {energy_data.mean()}")
    print(f"  Std: {energy_data.std()}")

    print("\nHit Time Channel:")
    print(f"  Min: {time_data.min()}")
    print(f"  Max: {time_data.max()}")
    print(f"  Mean: {time_data.mean()}")
    print(f"  Std: {time_data.std()}")

print("Electron data statistics:")
display_statistics(electron_X_np)
print("\nPhoton data statistics:")
display_statistics(photon_X_np)

print("Electron data statistics:")
display_statistics(electron_X_np)
print("\nPhoton data statistics:")
display_statistics(photon_X_np)

def plot_sample(sample, title):
    energy = sample[:, :, 0]
    time = sample[:, :, 1]

    fig, axs = plt.subplots(1, 2, figsize=(10, 5))
    im1 = axs[0].imshow(sample[:, :, 0], cmap='viridis')
    axs[0].imshow(energy, cmap='viridis')
    cbar1 = plt.colorbar(im1, ax=axs[0], shrink=0.8)
    axs[0].set_title('Hit Energy')
    im2 = axs[1].imshow(sample[:, :, 1], cmap='viridis')
    axs[1].imshow(time, cmap='viridis')
    cbar2 = plt.colorbar(im2, ax=axs[1], shrink=0.8)
    axs[1].set_title('Hit Time')
    plt.suptitle(title)
    plt.show()

# Visualize a few samples from electron and photon datasets
plot_sample(electron_X_np[100], 'Electron Sample 100')
plot_sample(electron_X_np[500], 'Electron Sample 500')
```

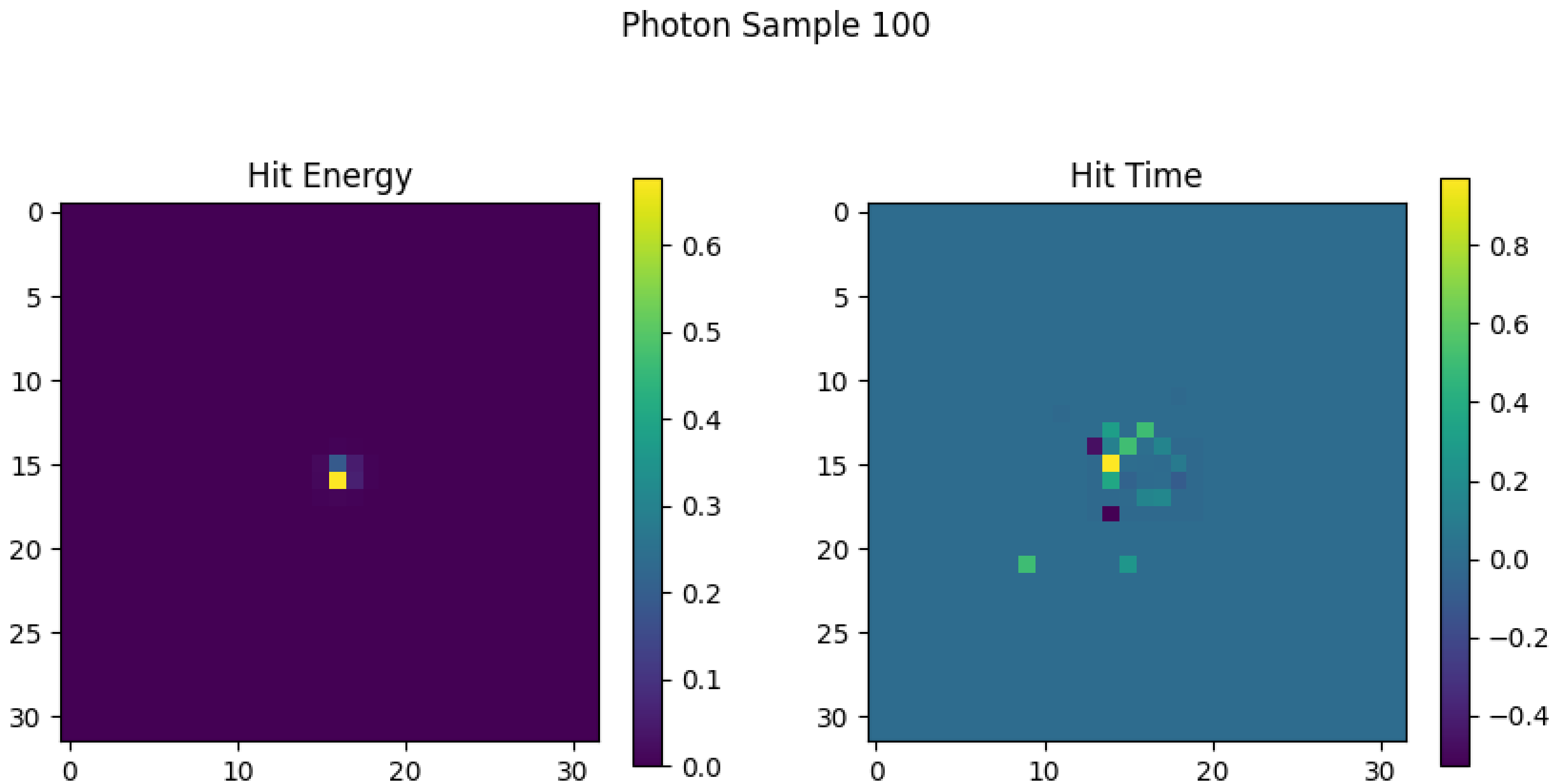
```
plot_sample(photon_X_np[100], 'Photon Sample 100')
plot_sample(photon_X_np[500], 'Photon Sample 500')
```

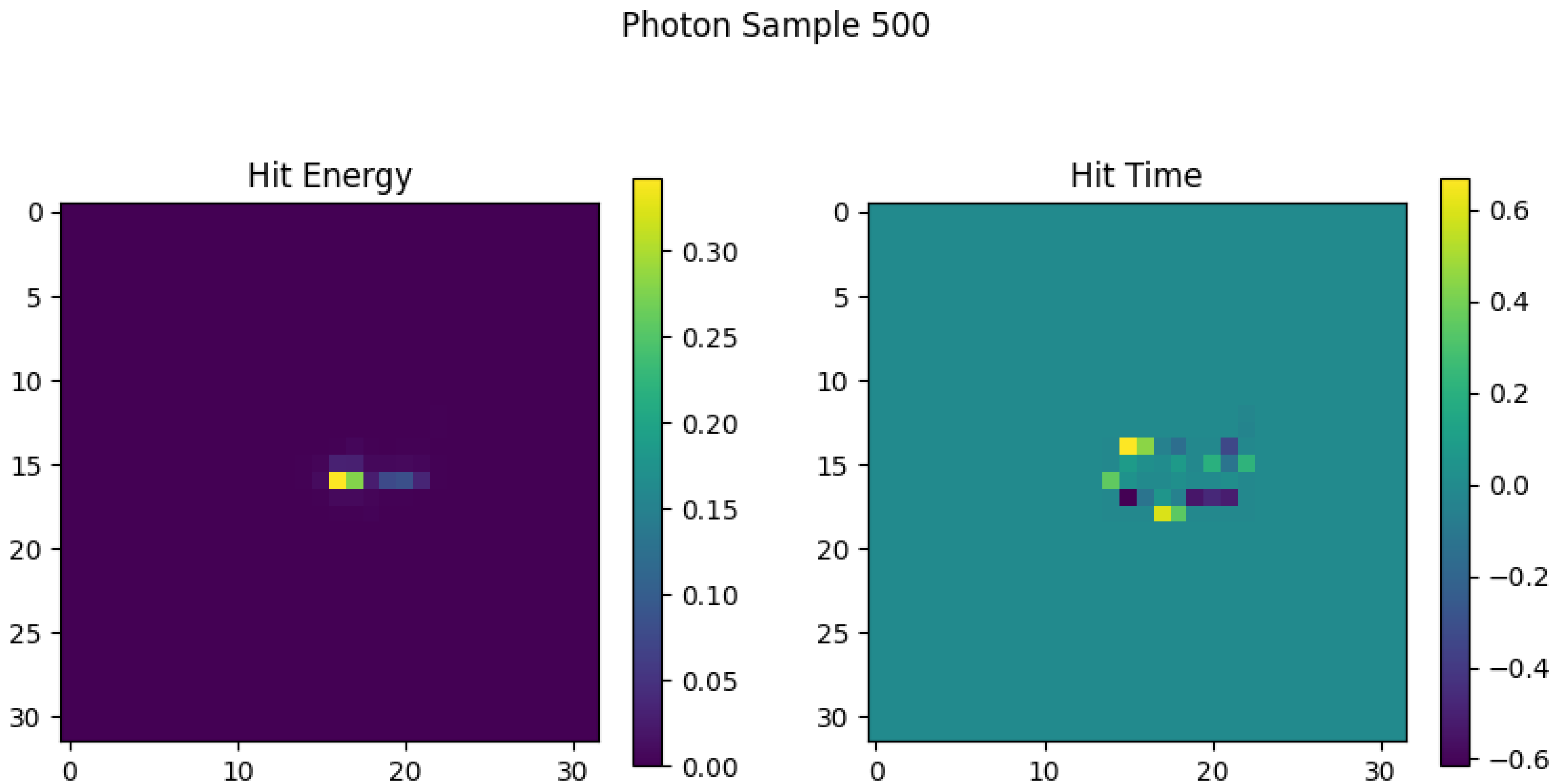
```
Electron data statistics:
Hit Energy Channel:
  Min: 0.0
  Max: 1.4318130016326904
  Mean: 0.001215839758515358
  Std: 0.022602548822760582

Hit Time Channel:
  Min: -2.512557029724121
  Max: 2.275660276412964
  Mean: -0.0002865783462766558
  Std: 0.06925680488348007

Photon data statistics:
Hit Energy Channel:
  Min: 0.0
  Max: 1.4849443435668945
  Mean: 0.0012234959285706282
  Std: 0.02478918246924877

Hit Time Channel:
  Min: -2.512557029724121
  Max: 2.2779698371887207
  Mean: -0.00023703569604549557
  Std: 0.06545672565698624
Electron data statistics:
Hit Energy Channel:
  Min: 0.0
  Max: 1.4318130016326904
  Mean: 0.001215839758515358
  Std: 0.022602548822760582

Hit Time Channel:
  Min: -2.512557029724121
  Max: 2.275660276412964
  Mean: -0.0002865783462766558
  Std: 0.06925680488348007

Photon data statistics:
Hit Energy Channel:
  Min: 0.0
  Max: 1.4849443435668945
```

```
  Mean: 0.0012234959285706282
  Std: 0.02478918246924877

Hit Time Channel:
  Min: -2.512557029724121
  Max: 2.2779698371887207
  Mean: -0.00023703569604549557
  Std: 0.06545672565698624
```




Electron Sample 500
Hit Energy
Hit Time
0
5
10
15
20
25
30
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75


Photon Sample 100
Hit Energy
Hit Time
0
5
10
15
20
25
30
0.0
0.1
0.2
0.3
0.4
0.5
0.6
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8



In [ ]:

```
# Combine electron and photon data and labels

X = np.concatenate((electron_X, photon_X), axis=0)
y = np.concatenate((electron_y, photon_y), axis=0)
```

In [ ]:

```
# Normalize the data
# Assuming X is a 4D array (samples, width, height, channels)

X_normalized = X / X.max(axis=(0, 1, 2), keepdims=True)
```

In [ ]:

```
# Apply quantization

n_bits = 8
X_quantized = np.round(X_normalized * (2**n_bits - 1)).astype(np.uint8)
```

In [ ]:

```
# Split the data into training and validation sets

from sklearn.model_selection import train_test_split

X_train, X_val, y_train, y_val = train_test_split(X_quantized, y, test_size=0.2, random_state=42, stratify=y)
```

In [ ]:

```
# Create stratified batches for training and validation data

from sklearn.model_selection import StratifiedShuffleSplit

batch_size = 8300  # Adjust this based on the desired batch size
sss = StratifiedShuffleSplit(n_splits=1, test_size=batch_size, random_state=42)

def create_stratified_batches(X_data, y_data, batch_size):
    n_splits = int(np.ceil(len(X_data) / batch_size))
    sss = StratifiedShuffleSplit(n_splits=n_splits, test_size=batch_size, random_state=42)
    batches = []

    for train_index, test_index in sss.split(X_data, y_data):
        X_batch, y_batch = X_data[test_index], y_data[test_index]
        batches.append((X_batch, y_batch))

    return batches

train_batches = create_stratified_batches(X_train, y_train, batch_size)
val_batches = create_stratified_batches(X_val, y_val, batch_size)
```

In [ ]:

```
# Save stratified batches into Parquet files

import pickle

def save_batches(batches, prefix):
    for i, (batch_data, batch_labels) in enumerate(batches):
        # Save data and labels as binary data
        with open(f'{prefix}_batch_{i}_data.pkl', 'wb') as data_file:
```

```
            pickle.dump(batch_data, data_file)
        with open(f'{prefix}_batch_{i}_labels.pkl', 'wb') as labels_file:
            pickle.dump(batch_labels, labels_file)

save_batches(train_batches, '/content/drive/MyDrive/GSoC/task_1/prcsd_training/training')
save_batches(val_batches, '/content/drive/MyDrive/GSoC/task_1/prcsd_val/validation')
```

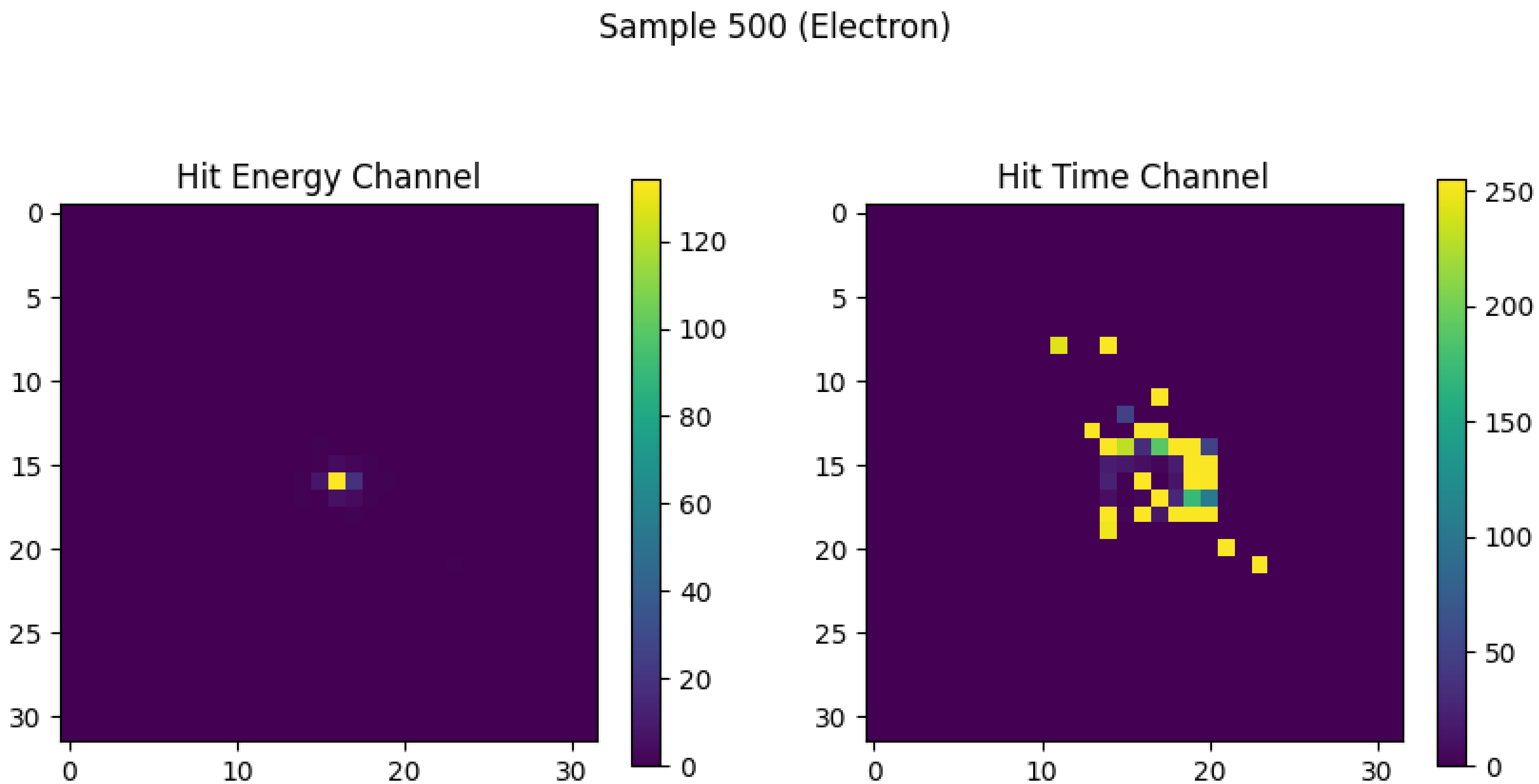
In [ ]:

```
import pickle
import matplotlib.pyplot as plt

def display_statistics(data):
    energy_data = data[:, :, :, 0]
    time_data = data[:, :, :, 1]

    print("Hit Energy Channel:")
    print(f"  Min: {energy_data.min()}")
    print(f"  Max: {energy_data.max()}")
    print(f"  Mean: {energy_data.mean()}")
    print(f"  Std: {energy_data.std()}")

    print("\nHit Time Channel:")
    print(f"  Min: {time_data.min()}")
    print(f"  Max: {time_data.max()}")
    print(f"  Mean: {time_data.mean()}")
    print(f"  Std: {time_data.std()}")

# Load data and labels from a .pkl file
def load_batch_data(prefix, batch_index):
    with open(f'{prefix}_batch_{batch_index}_data.pkl', 'rb') as data_file:
        batch_data = pickle.load(data_file)
    with open(f'{prefix}_batch_{batch_index}_labels.pkl', 'rb') as labels_file:
        batch_labels = pickle.load(labels_file)

    n_electron = sum(batch_labels == 1)
    n_photon = sum(batch_labels == 0)

    return batch_data, batch_labels, n_electron, n_photon

# Display label counts for each batch
prefix = '/content/drive/MyDrive/GSoC/task_1/prcsd_training/training'
n_batches = 10

print(f"{'Batch':<10} {'Electron':<10} {'Photon':<10}")
for i in range(n_batches):
```

```
    _, _, n_electron, n_photon = load_batch_data(prefix, i)
    print(f"{i:<10} {n_electron:<10} {n_photon:<10}")

# Load the first batch of training data
batch_data, batch_labels, _, _ = load_batch_data(prefix, 0)

# Display statistics
display_statistics(batch_data)

# Plot the data (first sample)
sample_index = 500
sample = batch_data[sample_index]
label = batch_labels[sample_index]

fig, ax = plt.subplots(1, 2, figsize=(10, 5))
im1 = ax[0].imshow(sample[:, :, 0], cmap='viridis')
ax[0].set_title("Hit Energy Channel")
cbar1 = plt.colorbar(im1, ax=ax[0], shrink=0.8)
im2 = ax[1].imshow(sample[:, :, 1], cmap='viridis')
ax[1].set_title("Hit Time Channel")
cbar2 = plt.colorbar(im2, ax=ax[1], shrink=0.8)
plt.suptitle(f"Sample {sample_index} ({'Electron' if label == 1 else 'Photon'})")
plt.show()
```

```
Batch      Electron   Photon
0          4150       4150
1          4150       4150
2          4150       4150
3          4150       4150
4          4150       4150
5          4150       4150
6          4150       4150
7          4150       4150
8          4150       4150
9          4150       4150
Hit Energy Channel:
  Min: 0
  Max: 246
  Mean: 0.20384012612951807
  Std: 4.064403768854381

Hit Time Channel:
  Min: 0
  Max: 255
```

```
Mean: 9.92738234186747
Std: 47.70885034491328
```



In [ ]:
```
!pip install -q tensorflow
!pip install -q tensorflow-addons
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.1/1.1 MB 17.6 MB/s eta 0:00:00
```

In [ ]:
```
import tensorflow as tf
import tensorflow_addons as tfa
from tensorflow.keras.applications.efficientnet import EfficientNetB2

# Create the hybrid model
def create_hybrid_model(input_shape):
    inputs = tf.keras.layers.Input(shape=input_shape)

    # Reshape the input data to have 3 channels
```

```
    x = tf.keras.layers.Concatenate()([inputs, inputs, inputs])

    # Residual block
    res_block = tf.keras.layers.Conv2D(6, (3, 3), activation='gelu', padding='same')(x)
    res_block = tf.keras.layers.BatchNormalization()(res_block)
    res_block = tf.keras.layers.DepthwiseConv2D((3, 3), activation='gelu', padding='same')(res_block)
    res_block = tf.keras.layers.BatchNormalization()(res_block)
    res_block = tf.keras.layers.add([x, res_block])

    # Reduce channels to 3 using a 1x1 convolution
    reduced_channels = tf.keras.layers.Conv2D(3, (1, 1), activation='gelu')(res_block)

    # EfficientNet
    efficientnet = EfficientNetB2(include_top=False, input_shape=(32, 32, 3), weights='imagenet')
    efficientnet.trainable = True

    x = efficientnet(reduced_channels)
    x = tf.keras.layers.GlobalAveragePooling2D()(x)
    x = tf.keras.layers.Dense(256, activation='gelu', kernel_initializer='lecun_normal')(x)
    x = tf.keras.layers.Dropout(0.5)(x)
    outputs = tf.keras.layers.Dense(1, activation='sigmoid', kernel_initializer='lecun_normal')(x)

    model = tf.keras.Model(inputs=inputs, outputs=outputs)
    return model

input_shape = (32, 32, 2)
model = create_hybrid_model(input_shape)
model.summary()
```

```
/usr/local/lib/python3.9/dist-packages/tensorflow_addons/utils/ensure_tf_install.py:53: UserWarning: Tensorflow Addons supports using Python ops for all Tensorfl
ow versions above or equal to 2.9.0 and strictly below 2.12.0 (nightly versions are not supported).
 The versions of TensorFlow you are currently using is 2.12.0 and is not supported.
Some things might work, some things might not.
If you were to encounter a bug, do not file an issue.
If you want to make sure you're using a tested and supported configuration, either change the TensorFlow version or the TensorFlow Addons's version.
You can find the compatibility matrix in TensorFlow Addon's readme:
https://github.com/tensorflow/addons
  warnings.warn(
Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb2_notop.h5
31790344/31790344 [==============================] - 0s 0us/step
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 32, 32, 2)]  0           []
```

```
 concatenate (Concatenate)      (None, 32, 32, 6)    0           ['input_1[0][0]',
                                                                  'input_1[0][0]',
                                                                  'input_1[0][0]']

 conv2d (Conv2D)                (None, 32, 32, 6)    330         ['concatenate[0][0]']

 batch_normalization (BatchNorm  (None, 32, 32, 6)   24          ['conv2d[0][0]']
 alization)

 depthwise_conv2d (DepthwiseCon  (None, 32, 32, 6)   60          ['batch_normalization[0][0]']
 v2D)

 batch_normalization_1 (BatchNo  (None, 32, 32, 6)   24          ['depthwise_conv2d[0][0]']
 rmalization)

 add (Add)                      (None, 32, 32, 6)    0           ['concatenate[0][0]',
                                                                  'batch_normalization_1[0][0]']

 conv2d_1 (Conv2D)              (None, 32, 32, 3)    21          ['add[0][0]']

 efficientnetb2 (Functional)    (None, 1, 1, 1408)   7768569     ['conv2d_1[0][0]']

 global_average_pooling2d (Glob  (None, 1408)        0           ['efficientnetb2[0][0]']
 alAveragePooling2D)

 dense (Dense)                  (None, 256)          360704      ['global_average_pooling2d[0][0]'
                                                                 ]

 dropout (Dropout)              (None, 256)          0           ['dense[0][0]']

 dense_1 (Dense)                (None, 1)            257         ['dropout[0][0]']

==================================================================================================
Total params: 8,129,989
Trainable params: 8,062,390
Non-trainable params: 67,599
__________________________________________________________________________________________________
```

In [ ]:

```
from sklearn.metrics import roc_auc_score

class StopAtROCAUC(tf.keras.callbacks.Callback):
    def __init__(self, threshold):
        super(StopAtROCAUC, self).__init__()
        self.threshold = threshold
```

```
    def on_epoch_end(self, epoch, logs=None):
        # Get the validation data from the generator
        val_data, val_labels = [], []
        for i in range(num_val_batches):
            X_val_batch, y_val_batch = load_batch_data(val_prefix, i)
            val_data.append(X_val_batch)
            val_labels.append(y_val_batch)

        X_val = np.concatenate(val_data, axis=0)
        y_val = np.concatenate(val_labels, axis=0)

        # Compute the ROC-AUC score
        y_pred = self.model.predict(X_val)
        roc_auc = roc_auc_score(y_val, y_pred)

        print(f"\nROC-AUC score for epoch {epoch + 1}: {roc_auc:.4f}")

        # Check if the ROC-AUC score has reached the threshold
        if roc_auc >= self.threshold:
            print(f"\nReached {self.threshold * 100:.2f}% ROC-AUC score. Stopping training.")
            self.model.stop_training = True
```

In [ ]:

```
import os
import numpy as np
import pickle
from tensorflow.keras.optimizers import RMSprop
from sklearn.preprocessing import LabelBinarizer
from tensorflow.keras.metrics import AUC

# Initialize LabelBinarizer
label_binarizer = LabelBinarizer()
label_binarizer.fit([0, 1])

# Load data and labels from pkl files
def load_batch_data(prefix, batch_index):
    with open(f'{prefix}_batch_{batch_index}_data.pkl', 'rb') as data_file:
        batch_data = pickle.load(data_file)
    with open(f'{prefix}_batch_{batch_index}_labels.pkl', 'rb') as labels_file:
        batch_labels = pickle.load(labels_file)

    # Transform labels into one-hot encoded format
    batch_labels = label_binarizer.transform(batch_labels)
```

```
    return batch_data, batch_labels

# Data generator
def data_generator(prefix, num_batches):
    while True:
        for i in range(num_batches):
            X, y = load_batch_data(prefix, i)
            yield X, y

# Set parameters
train_prefix = '/content/drive/MyDrive/GSoC/task_1/prcsd_training/training'
val_prefix = '/content/drive/MyDrive/GSoC/task_1/prcsd_val/validation'
num_train_batches = 48  # Adjust this based on the number of training batches you have
num_val_batches = 12  # Adjust this based on the number of validation batches you have
batch_size = 8300

# Create data generators
train_generator = data_generator(train_prefix, num_train_batches)
val_generator = data_generator(val_prefix, num_val_batches)

# Compile the model
rmsprop_optimizer = RMSprop(learning_rate=1e-4)
loss = 'binary_crossentropy'
metrics = [AUC(name='auc'), 'accuracy']

model.compile(optimizer=rmsprop_optimizer, loss=loss, metrics=metrics)

# Set the ROC-AUC threshold to 0.8
stop_at_roc_auc = StopAtROCAUC(threshold=0.8)

# Train the model
epochs = 50
steps_per_epoch = num_train_batches
validation_steps = num_val_batches

history = model.fit(train_generator,
                    epochs=epochs,
                    steps_per_epoch=steps_per_epoch,
                    validation_data=val_generator,
                    validation_steps=validation_steps,
                    verbose=1,
                    callbacks=[stop_at_roc_auc])
```

```
Epoch 1/50
3113/3113 [==============================] - 31s 9ms/step
```

```
ROC-AUC score for epoch 1: 0.4657
48/48 [==============================] - 132s 1s/step - loss: 0.7089 - auc: 0.5283 - accuracy: 0.5202 - val_loss: 0.6948 - val_auc: 0.4663 - val_accuracy: 0.4752
Epoch 2/50
3113/3113 [==============================] - 29s 9ms/step

ROC-AUC score for epoch 2: 0.5016
48/48 [==============================] - 60s 1s/step - loss: 0.6945 - auc: 0.5589 - accuracy: 0.5422 - val_loss: 0.6934 - val_auc: 0.5025 - val_accuracy: 0.5011
Epoch 3/50
3113/3113 [==============================] - 29s 9ms/step

ROC-AUC score for epoch 3: 0.5653
48/48 [==============================] - 60s 1s/step - loss: 0.6880 - auc: 0.5738 - accuracy: 0.5534 - val_loss: 0.6890 - val_auc: 0.5651 - val_accuracy: 0.5269
Epoch 4/50
3113/3113 [==============================] - 29s 9ms/step

ROC-AUC score for epoch 4: 0.5593
48/48 [==============================] - 60s 1s/step - loss: 0.6839 - auc: 0.5836 - accuracy: 0.5601 - val_loss: 0.6900 - val_auc: 0.5593 - val_accuracy: 0.5227
Epoch 5/50
3113/3113 [==============================] - 29s 9ms/step

ROC-AUC score for epoch 5: 0.5589
48/48 [==============================] - 60s 1s/step - loss: 0.6805 - auc: 0.5929 - accuracy: 0.5673 - val_loss: 0.6905 - val_auc: 0.5588 - val_accuracy: 0.5190
Epoch 6/50
3113/3113 [==============================] - 30s 9ms/step

ROC-AUC score for epoch 6: 0.5623
48/48 [==============================] - 61s 1s/step - loss: 0.6780 - auc: 0.5996 - accuracy: 0.5721 - val_loss: 0.6905 - val_auc: 0.5622 - val_accuracy: 0.5218
Epoch 7/50
3113/3113 [==============================] - 29s 9ms/step

ROC-AUC score for epoch 7: 0.5630
48/48 [==============================] - 60s 1s/step - loss: 0.6756 - auc: 0.6059 - accuracy: 0.5770 - val_loss: 0.6903 - val_auc: 0.5631 - val_accuracy: 0.5239
Epoch 8/50
3113/3113 [==============================] - 29s 9ms/step

ROC-AUC score for epoch 8: 0.5616
48/48 [==============================] - 60s 1s/step - loss: 0.6727 - auc: 0.6131 - accuracy: 0.5814 - val_loss: 0.6914 - val_auc: 0.5616 - val_accuracy: 0.5234
Epoch 9/50
3113/3113 [==============================] - 29s 9ms/step

ROC-AUC score for epoch 9: 0.5634
48/48 [==============================] - 61s 1s/step - loss: 0.6707 - auc: 0.6183 - accuracy: 0.5852 - val_loss: 0.6928 - val_auc: 0.5633 - val_accuracy: 0.5258
Epoch 10/50
3113/3113 [==============================] - 29s 9ms/step
```

```
ROC-AUC score for epoch 10: 0.5693
48/48 [==============================] - 60s 1s/step - loss: 0.6691 - auc: 0.6223 - accuracy: 0.5879 - val_loss: 0.6915 - val_auc: 0.5694 - val_accuracy: 0.5310
Epoch 11/50
3113/3113 [==============================] - 28s 9ms/step

ROC-AUC score for epoch 11: 0.5758
48/48 [==============================] - 60s 1s/step - loss: 0.6666 - auc: 0.6279 - accuracy: 0.5916 - val_loss: 0.6901 - val_auc: 0.5757 - val_accuracy: 0.5400
Epoch 12/50
3113/3113 [==============================] - 29s 9ms/step

ROC-AUC score for epoch 12: 0.5853
48/48 [==============================] - 60s 1s/step - loss: 0.6649 - auc: 0.6318 - accuracy: 0.5932 - val_loss: 0.6881 - val_auc: 0.5851 - val_accuracy: 0.5503
Epoch 13/50
3113/3113 [==============================] - 29s 9ms/step

ROC-AUC score for epoch 13: 0.5907
48/48 [==============================] - 60s 1s/step - loss: 0.6623 - auc: 0.6376 - accuracy: 0.5975 - val_loss: 0.6857 - val_auc: 0.5908 - val_accuracy: 0.5610
Epoch 14/50
3113/3113 [==============================] - 29s 9ms/step

ROC-AUC score for epoch 14: 0.5955
48/48 [==============================] - 60s 1s/step - loss: 0.6601 - auc: 0.6423 - accuracy: 0.6012 - val_loss: 0.6836 - val_auc: 0.5954 - val_accuracy: 0.5691
Epoch 15/50
3113/3113 [==============================] - 29s 9ms/step

ROC-AUC score for epoch 15: 0.5958
48/48 [==============================] - 61s 1s/step - loss: 0.6576 - auc: 0.6473 - accuracy: 0.6041 - val_loss: 0.6835 - val_auc: 0.5958 - val_accuracy: 0.5727
Epoch 16/50
3113/3113 [==============================] - 29s 9ms/step

ROC-AUC score for epoch 16: 0.5944
48/48 [==============================] - 60s 1s/step - loss: 0.6543 - auc: 0.6541 - accuracy: 0.6092 - val_loss: 0.6847 - val_auc: 0.5945 - val_accuracy: 0.5696
Epoch 17/50
3113/3113 [==============================] - 29s 9ms/step

ROC-AUC score for epoch 17: 0.5933
48/48 [==============================] - 60s 1s/step - loss: 0.6518 - auc: 0.6589 - accuracy: 0.6130 - val_loss: 0.6863 - val_auc: 0.5934 - val_accuracy: 0.5693
Epoch 18/50
3113/3113 [==============================] - 29s 9ms/step

ROC-AUC score for epoch 18: 0.5910
48/48 [==============================] - 60s 1s/step - loss: 0.6488 - auc: 0.6643 - accuracy: 0.6162 - val_loss: 0.6880 - val_auc: 0.5910 - val_accuracy: 0.5695
Epoch 19/50
3113/3113 [==============================] - 29s 9ms/step
```

```
ROC-AUC score for epoch 19: 0.5892
48/48 [==============================] - 60s 1s/step - loss: 0.6452 - auc: 0.6707 - accuracy: 0.6207 - val_loss: 0.6904 - val_auc: 0.5891 - val_accuracy: 0.5662
Epoch 20/50
3113/3113 [==============================] - 29s 9ms/step

ROC-AUC score for epoch 20: 0.5877
48/48 [==============================] - 60s 1s/step - loss: 0.6404 - auc: 0.6789 - accuracy: 0.6267 - val_loss: 0.6931 - val_auc: 0.5877 - val_accuracy: 0.5649
Epoch 21/50
3113/3113 [==============================] - 29s 9ms/step

ROC-AUC score for epoch 21: 0.5861
48/48 [==============================] - 60s 1s/step - loss: 0.6369 - auc: 0.6844 - accuracy: 0.6304 - val_loss: 0.6957 - val_auc: 0.5860 - val_accuracy: 0.5631
Epoch 22/50
3113/3113 [==============================] - 29s 9ms/step

ROC-AUC score for epoch 22: 0.5837
48/48 [==============================] - 60s 1s/step - loss: 0.6322 - auc: 0.6925 - accuracy: 0.6373 - val_loss: 0.7001 - val_auc: 0.5838 - val_accuracy: 0.5616
Epoch 23/50
3113/3113 [==============================] - 29s 9ms/step

ROC-AUC score for epoch 23: 0.5839
48/48 [==============================] - 60s 1s/step - loss: 0.6277 - auc: 0.6994 - accuracy: 0.6418 - val_loss: 0.7039 - val_auc: 0.5839 - val_accuracy: 0.5612
Epoch 24/50
3113/3113 [==============================] - 29s 9ms/step

ROC-AUC score for epoch 24: 0.5782
48/48 [==============================] - 60s 1s/step - loss: 0.6231 - auc: 0.7062 - accuracy: 0.6477 - val_loss: 0.7083 - val_auc: 0.5782 - val_accuracy: 0.5570
Epoch 25/50
3113/3113 [==============================] - 30s 9ms/step

ROC-AUC score for epoch 25: 0.5772
48/48 [==============================] - 61s 1s/step - loss: 0.6178 - auc: 0.7138 - accuracy: 0.6525 - val_loss: 0.7141 - val_auc: 0.5773 - val_accuracy: 0.5551
Epoch 26/50
3113/3113 [==============================] - 29s 9ms/step

ROC-AUC score for epoch 26: 0.5748
48/48 [==============================] - 60s 1s/step - loss: 0.6125 - auc: 0.7210 - accuracy: 0.6580 - val_loss: 0.7195 - val_auc: 0.5748 - val_accuracy: 0.5533
Epoch 27/50
3113/3113 [==============================] - 29s 9ms/step

ROC-AUC score for epoch 27: 0.5745
48/48 [==============================] - 60s 1s/step - loss: 0.6066 - auc: 0.7292 - accuracy: 0.6649 - val_loss: 0.7237 - val_auc: 0.5746 - val_accuracy: 0.5538
Epoch 28/50
3113/3113 [==============================] - 29s 9ms/step
```

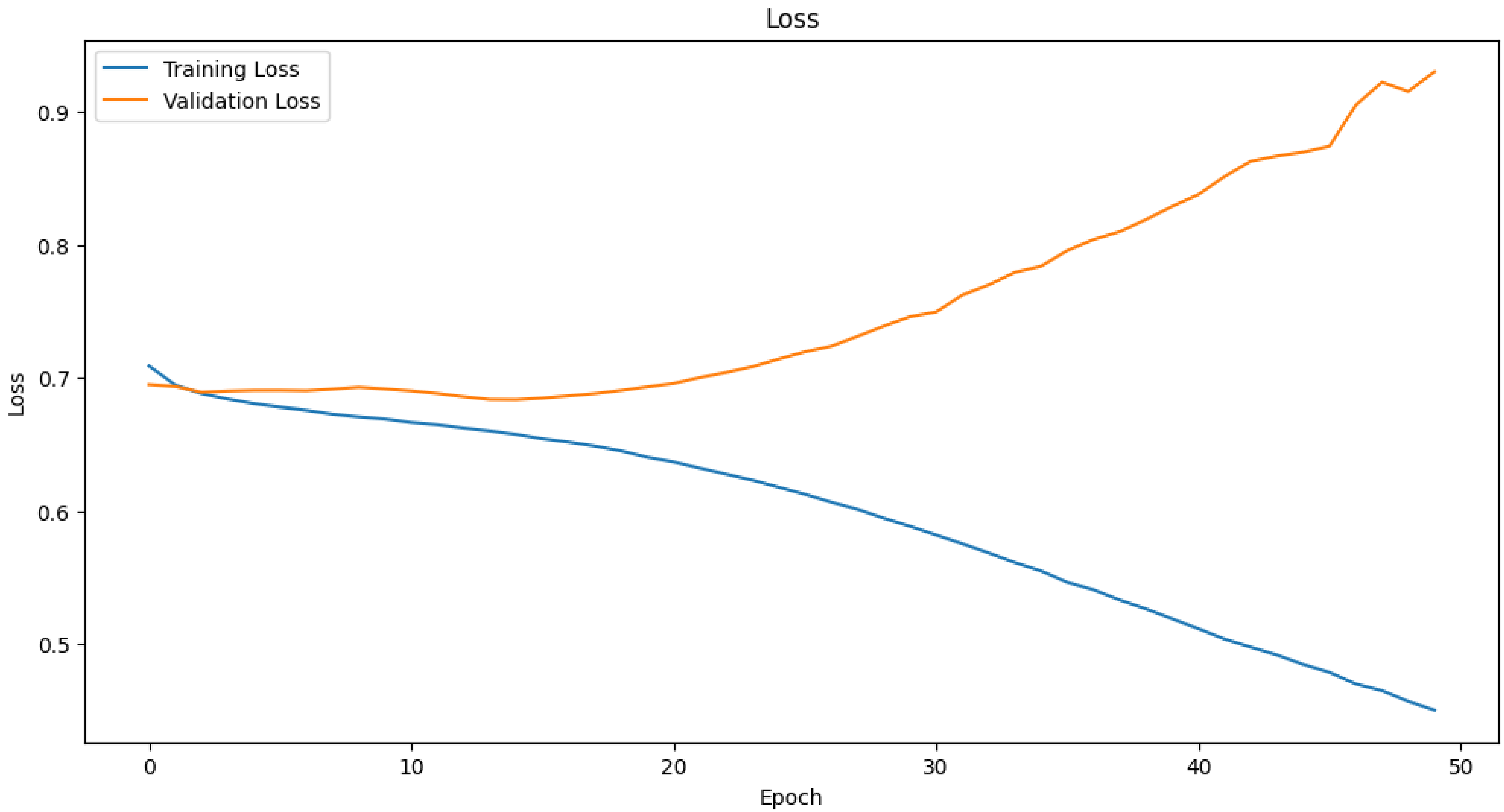
```
ROC-AUC score for epoch 28: 0.5717
48/48 [==============================] - 60s 1s/step - loss: 0.6012 - auc: 0.7356 - accuracy: 0.6692 - val_loss: 0.7311 - val_auc: 0.5718 - val_accuracy: 0.5509
Epoch 29/50
3113/3113 [==============================] - 29s 9ms/step

ROC-AUC score for epoch 29: 0.5699
48/48 [==============================] - 60s 1s/step - loss: 0.5945 - auc: 0.7437 - accuracy: 0.6757 - val_loss: 0.7388 - val_auc: 0.5700 - val_accuracy: 0.5517
Epoch 30/50
3113/3113 [==============================] - 29s 9ms/step

ROC-AUC score for epoch 30: 0.5676
48/48 [==============================] - 60s 1s/step - loss: 0.5885 - auc: 0.7508 - accuracy: 0.6807 - val_loss: 0.7459 - val_auc: 0.5676 - val_accuracy: 0.5486
Epoch 31/50
3113/3113 [==============================] - 29s 9ms/step

ROC-AUC score for epoch 31: 0.5662
48/48 [==============================] - 60s 1s/step - loss: 0.5818 - auc: 0.7585 - accuracy: 0.6874 - val_loss: 0.7495 - val_auc: 0.5661 - val_accuracy: 0.5476
Epoch 32/50
3113/3113 [==============================] - 29s 9ms/step

ROC-AUC score for epoch 32: 0.5636
48/48 [==============================] - 60s 1s/step - loss: 0.5753 - auc: 0.7659 - accuracy: 0.6935 - val_loss: 0.7623 - val_auc: 0.5637 - val_accuracy: 0.5448
Epoch 33/50
3113/3113 [==============================] - 29s 9ms/step

ROC-AUC score for epoch 33: 0.5629
48/48 [==============================] - 60s 1s/step - loss: 0.5684 - auc: 0.7727 - accuracy: 0.6981 - val_loss: 0.7698 - val_auc: 0.5628 - val_accuracy: 0.5428
Epoch 34/50
3113/3113 [==============================] - 29s 9ms/step

ROC-AUC score for epoch 34: 0.5641
48/48 [==============================] - 60s 1s/step - loss: 0.5611 - auc: 0.7804 - accuracy: 0.7051 - val_loss: 0.7793 - val_auc: 0.5640 - val_accuracy: 0.5446
Epoch 35/50
3113/3113 [==============================] - 29s 9ms/step

ROC-AUC score for epoch 35: 0.5603
48/48 [==============================] - 61s 1s/step - loss: 0.5548 - auc: 0.7865 - accuracy: 0.7095 - val_loss: 0.7839 - val_auc: 0.5602 - val_accuracy: 0.5424
Epoch 36/50
3113/3113 [==============================] - 29s 9ms/step

ROC-AUC score for epoch 36: 0.5590
48/48 [==============================] - 60s 1s/step - loss: 0.5462 - auc: 0.7949 - accuracy: 0.7166 - val_loss: 0.7957 - val_auc: 0.5588 - val_accuracy: 0.5422
Epoch 37/50
3113/3113 [==============================] - 29s 9ms/step
```

```
ROC-AUC score for epoch 37: 0.5586
48/48 [==============================] - 60s 1s/step - loss: 0.5407 - auc: 0.8002 - accuracy: 0.7209 - val_loss: 0.8040 - val_auc: 0.5586 - val_accuracy: 0.5414
Epoch 38/50
3113/3113 [==============================] - 28s 9ms/step

ROC-AUC score for epoch 38: 0.5579
48/48 [==============================] - 59s 1s/step - loss: 0.5330 - auc: 0.8067 - accuracy: 0.7263 - val_loss: 0.8099 - val_auc: 0.5579 - val_accuracy: 0.5403
Epoch 39/50
3113/3113 [==============================] - 28s 9ms/step

ROC-AUC score for epoch 39: 0.5563
48/48 [==============================] - 59s 1s/step - loss: 0.5262 - auc: 0.8127 - accuracy: 0.7311 - val_loss: 0.8190 - val_auc: 0.5562 - val_accuracy: 0.5407
Epoch 40/50
3113/3113 [==============================] - 28s 9ms/step

ROC-AUC score for epoch 40: 0.5549
48/48 [==============================] - 59s 1s/step - loss: 0.5188 - auc: 0.8191 - accuracy: 0.7369 - val_loss: 0.8289 - val_auc: 0.5548 - val_accuracy: 0.5396
Epoch 41/50
3113/3113 [==============================] - 28s 9ms/step

ROC-AUC score for epoch 41: 0.5542
48/48 [==============================] - 59s 1s/step - loss: 0.5113 - auc: 0.8252 - accuracy: 0.7421 - val_loss: 0.8379 - val_auc: 0.5542 - val_accuracy: 0.5386
Epoch 42/50
3113/3113 [==============================] - 27s 9ms/step

ROC-AUC score for epoch 42: 0.5534
48/48 [==============================] - 59s 1s/step - loss: 0.5035 - auc: 0.8313 - accuracy: 0.7466 - val_loss: 0.8515 - val_auc: 0.5535 - val_accuracy: 0.5393
Epoch 43/50
3113/3113 [==============================] - 28s 9ms/step

ROC-AUC score for epoch 43: 0.5538
48/48 [==============================] - 59s 1s/step - loss: 0.4974 - auc: 0.8358 - accuracy: 0.7501 - val_loss: 0.8629 - val_auc: 0.5538 - val_accuracy: 0.5397
Epoch 44/50
3113/3113 [==============================] - 28s 9ms/step

ROC-AUC score for epoch 44: 0.5543
48/48 [==============================] - 59s 1s/step - loss: 0.4915 - auc: 0.8404 - accuracy: 0.7548 - val_loss: 0.8669 - val_auc: 0.5542 - val_accuracy: 0.5391
Epoch 45/50
3113/3113 [==============================] - 28s 9ms/step

ROC-AUC score for epoch 45: 0.5531
48/48 [==============================] - 59s 1s/step - loss: 0.4844 - auc: 0.8457 - accuracy: 0.7597 - val_loss: 0.8698 - val_auc: 0.5531 - val_accuracy: 0.5380
Epoch 46/50
3113/3113 [==============================] - 28s 9ms/step
```

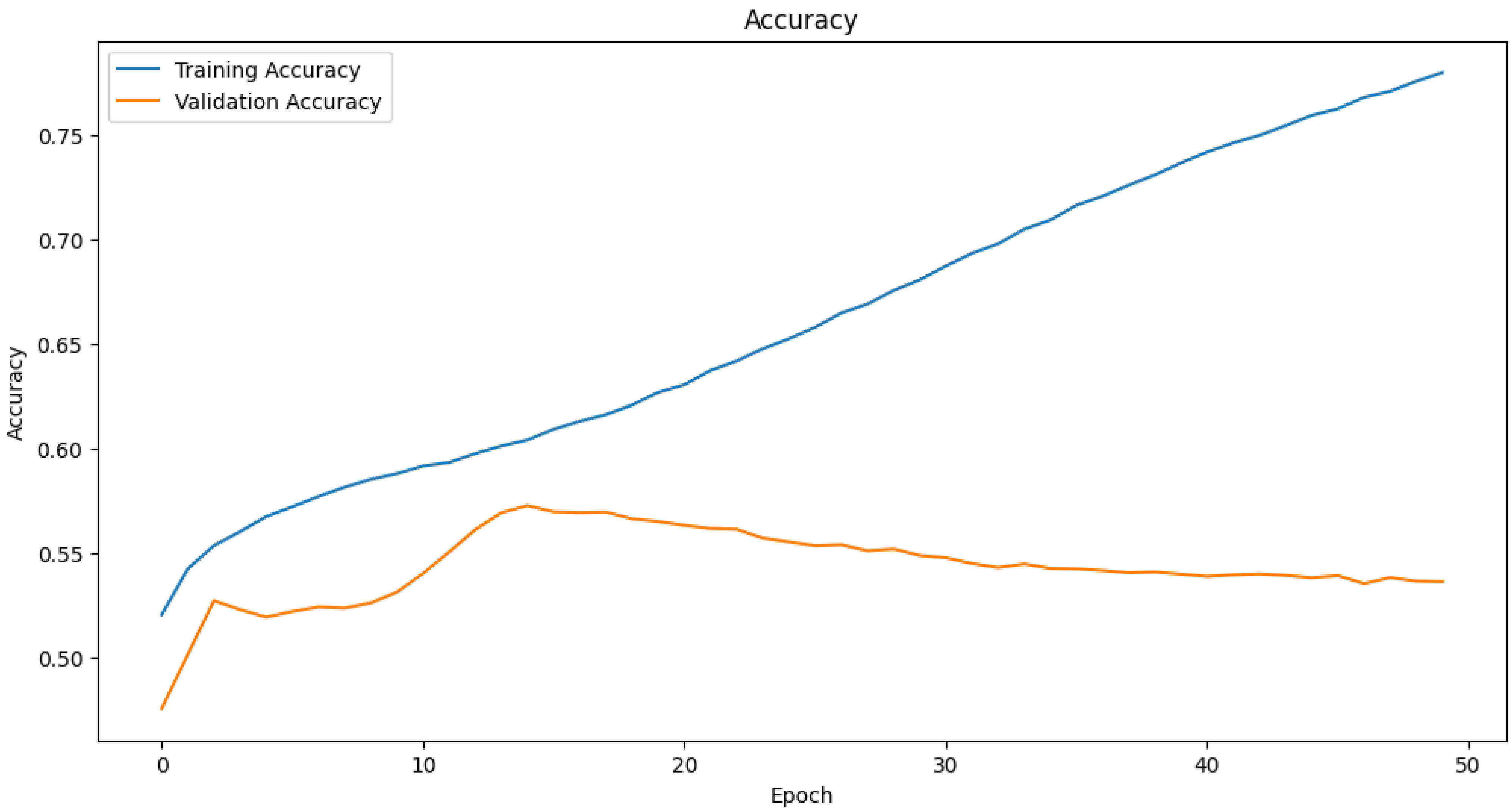
```
ROC-AUC score for epoch 46: 0.5520
48/48 [==============================] - 59s 1s/step - loss: 0.4785 - auc: 0.8500 - accuracy: 0.7628 - val_loss: 0.8742 - val_auc: 0.5519 - val_accuracy: 0.5389
Epoch 47/50
3113/3113 [==============================] - 27s 9ms/step

ROC-AUC score for epoch 47: 0.5500
48/48 [==============================] - 58s 1s/step - loss: 0.4698 - auc: 0.8560 - accuracy: 0.7684 - val_loss: 0.9052 - val_auc: 0.5500 - val_accuracy: 0.5351
Epoch 48/50
3113/3113 [==============================] - 28s 9ms/step

ROC-AUC score for epoch 48: 0.5520
48/48 [==============================] - 59s 1s/step - loss: 0.4647 - auc: 0.8595 - accuracy: 0.7713 - val_loss: 0.9224 - val_auc: 0.5520 - val_accuracy: 0.5380
Epoch 49/50
3113/3113 [==============================] - 27s 9ms/step

ROC-AUC score for epoch 49: 0.5498
48/48 [==============================] - 58s 1s/step - loss: 0.4567 - auc: 0.8646 - accuracy: 0.7761 - val_loss: 0.9155 - val_auc: 0.5497 - val_accuracy: 0.5363
Epoch 50/50
3113/3113 [==============================] - 27s 9ms/step

ROC-AUC score for epoch 50: 0.5501
48/48 [==============================] - 59s 1s/step - loss: 0.4500 - auc: 0.8692 - accuracy: 0.7802 - val_loss: 0.9303 - val_auc: 0.5502 - val_accuracy: 0.5360
```

In [ ]:

```
import matplotlib.pyplot as plt

# Plot loss
plt.figure(figsize=(12, 6))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Plot accuracy
plt.figure(figsize=(12, 6))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



Loss
Training Loss
Validation Loss
0.9
0.8
0.7
0.6
0.5
Loss
0
10
20
30
40
50
Epoch

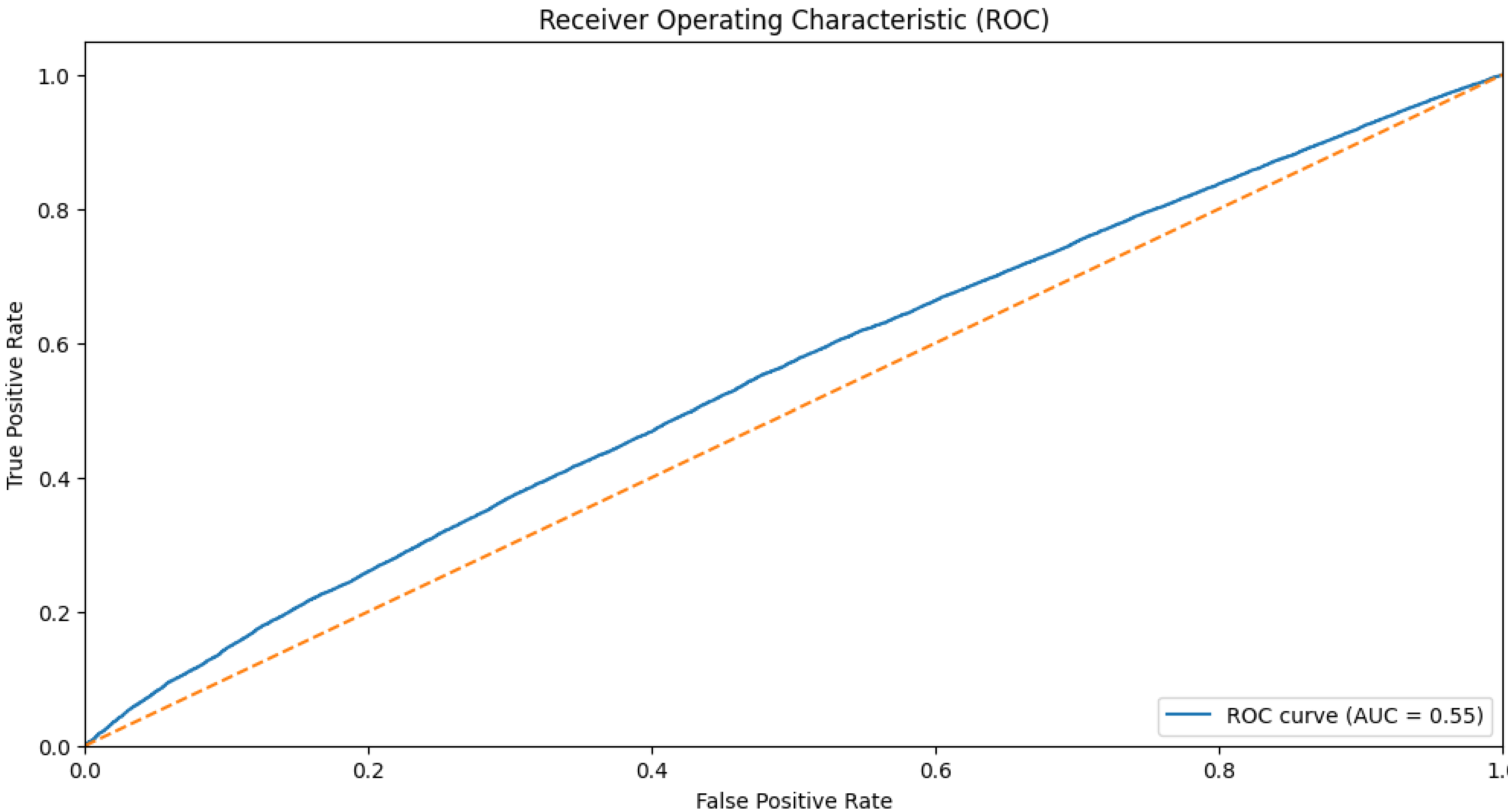


In [ ]:

```
import numpy as np
from sklearn.metrics import roc_curve, auc

# Concatenate validation data
X_val = []
y_val = []

for i in range(num_val_batches):
    X_batch, y_batch = load_batch_data(val_prefix, i)
    X_val.append(X_batch)
```

```
    y_val.append(y_batch)

X_val = np.concatenate(X_val)
y_val = np.concatenate(y_val)

# Get the predicted probabilities for the positive class (class 1)
y_pred_probs = model.predict(X_val)[:, 0]

# Compute the ROC curve
fpr, tpr, thresholds = roc_curve(y_val, y_pred_probs)

# Compute the AUC score
roc_auc = auc(fpr, tpr)

# Plot the ROC curve
plt.figure(figsize=(12, 6))
plt.plot(fpr, tpr, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 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 (ROC)')
plt.legend(loc="lower right")
plt.show()
```

```
3113/3113 [==============================] - 28s 9ms/step
```



In [ ]:
```
model.save_weights('/content/drive/MyDrive/GSoC/task_1/model/model_frozen_efficientnet_weights.h5')
```

In [ ]:
```
from tensorflow.keras import backend as K

# Clear GPU memory
K.clear_session()
```

In [ ]:

```
!pip install -q tensorflow
!pip install -q tensorflow-addons
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.1/1.1 MB 50.5 MB/s eta 0:00:00
```

In [ ]:

```
import tensorflow as tf
import tensorflow_addons as tfa
from tensorflow.keras.applications.efficientnet import EfficientNetB2

# Create the hybrid model
def create_hybrid_model(input_shape):
    inputs = tf.keras.layers.Input(shape=input_shape)

    # Reshape the input data to have 3 channels
    x = tf.keras.layers.Concatenate()([inputs, inputs, inputs])

    # Residual block
    res_block = tf.keras.layers.Conv2D(6, (3, 3), activation='gelu', padding='same')(x)
    res_block = tf.keras.layers.BatchNormalization()(res_block)
    res_block = tf.keras.layers.DepthwiseConv2D((3, 3), activation='gelu', padding='same')(res_block)
    res_block = tf.keras.layers.BatchNormalization()(res_block)
    res_block = tf.keras.layers.add([x, res_block])

    # Reduce channels to 3 using a 1x1 convolution
    reduced_channels = tf.keras.layers.Conv2D(3, (1, 1), activation='gelu')(res_block)

    # EfficientNet
    efficientnet = EfficientNetB2(include_top=False, input_shape=(32, 32, 3), weights='imagenet')
    efficientnet.trainable = True

    x = efficientnet(reduced_channels)
    x = tf.keras.layers.GlobalAveragePooling2D()(x)
    x = tf.keras.layers.Dense(256, activation='gelu', kernel_initializer='lecun_normal')(x)
    x = tf.keras.layers.Dropout(0.5)(x)
    outputs = tf.keras.layers.Dense(1, activation='sigmoid', kernel_initializer='lecun_normal')(x)

    model = tf.keras.Model(inputs=inputs, outputs=outputs)
    return model
```

```
/usr/local/lib/python3.9/dist-packages/tensorflow_addons/utils/ensure_tf_install.py:53: UserWarning: Tensorflow Addons supports using Python ops for all Tensorfl
ow versions above or equal to 2.9.0 and strictly below 2.12.0 (nightly versions are not supported).
 The versions of TensorFlow you are currently using is 2.12.0 and is not supported.
Some things might work, some things might not.
```

```
If you were to encounter a bug, do not file an issue.
If you want to make sure you're using a tested and supported configuration, either change the TensorFlow version or the TensorFlow Addons's version.
You can find the compatibility matrix in TensorFlow Addon's readme:
https://github.com/tensorflow/addons
  warnings.warn(
```

In [ ]:

```
input_shape = (32, 32, 2)
loaded_model = create_hybrid_model(input_shape)
loaded_model.load_weights('/content/drive/MyDrive/GSoC/task_1/model/model_frozen_efficientnet_weights.h5')
loaded_model.summary()
```

```
Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb2_notop.h5
31790344/31790344 [==============================] - 2s 0us/step
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to                     
==================================================================================================
 input_1 (InputLayer)           [(None, 32, 32, 2)]  0           []                               
                                                                                                  
 concatenate (Concatenate)      (None, 32, 32, 6)    0           ['input_1[0][0]',                
                                                                  'input_1[0][0]',                
                                                                  'input_1[0][0]']                
                                                                                                  
 conv2d (Conv2D)                (None, 32, 32, 6)    330         ['concatenate[0][0]']            
                                                                                                  
 batch_normalization (BatchNorm  (None, 32, 32, 6)   24          ['conv2d[0][0]']                 
 alization)                                                                                       
                                                                                                  
 depthwise_conv2d (DepthwiseCon  (None, 32, 32, 6)   60          ['batch_normalization[0][0]']    
 v2D)                                                                                             
                                                                                                  
 batch_normalization_1 (BatchNo  (None, 32, 32, 6)   24          ['depthwise_conv2d[0][0]']       
 rmalization)                                                                                     
                                                                                                  
 add (Add)                      (None, 32, 32, 6)    0           ['concatenate[0][0]',            
                                                                  'batch_normalization_1[0][0]']  
                                                                                                  
 conv2d_1 (Conv2D)              (None, 32, 32, 3)    21          ['add[0][0]']                    
                                                                                                  
 efficientnetb2 (Functional)    (None, 1, 1, 1408)   7768569     ['conv2d_1[0][0]']               
                                                                                                  
 global_average_pooling2d (Glob  (None, 1408)        0           ['efficientnetb2[0][0]']         
 alAveragePooling2D)                                                                              
```

```
 dense (Dense)                  (None, 256)          360704      ['global_average_pooling2d[0][0]'
                                                                  ]

 dropout (Dropout)              (None, 256)          0           ['dense[0][0]']

 dense_1 (Dense)                (None, 1)            257         ['dropout[0][0]']

==================================================================================================
Total params: 8,129,989
Trainable params: 8,062,390
Non-trainable params: 67,599
__________________________________________________________________________________________________
```

In [ ]:

```python
from sklearn.metrics import roc_auc_score

class StopAtROCAUC(tf.keras.callbacks.Callback):
    def __init__(self, threshold):
        super(StopAtROCAUC, self).__init__()
        self.threshold = threshold

    def on_epoch_end(self, epoch, logs=None):
        # Get the validation data from the generator
        val_data, val_labels = [], []
        for i in range(num_val_batches):
            X_val_batch, y_val_batch = load_batch_data(val_prefix, i)
            val_data.append(X_val_batch)
            val_labels.append(y_val_batch)

        X_val = np.concatenate(val_data, axis=0)
        y_val = np.concatenate(val_labels, axis=0)

        # Compute the ROC-AUC score
        y_pred = self.model.predict(X_val)
        roc_auc = roc_auc_score(y_val, y_pred)

        print(f"\nROC-AUC score for epoch {epoch + 1}: {roc_auc:.4f}")

        # Check if the ROC-AUC score has reached the threshold
        if roc_auc >= self.threshold:
            print(f"\nReached {self.threshold * 100:.2f}% ROC-AUC score. Stopping training.")
            self.model.stop_training = True
```

In [ ]:
```
import os
import numpy as np
import pickle
from tensorflow.keras.optimizers import RMSprop
from sklearn.preprocessing import LabelBinarizer
from tensorflow.keras.metrics import AUC
from tensorflow.keras.optimizers.schedules import CosineDecay

# Initialize LabelBinarizer
label_binarizer = LabelBinarizer()
label_binarizer.fit([0, 1])

# Load data and labels from pkl files
def load_batch_data(prefix, batch_index):
    with open(f'{prefix}_batch_{batch_index}_data.pkl', 'rb') as data_file:
        batch_data = pickle.load(data_file)
    with open(f'{prefix}_batch_{batch_index}_labels.pkl', 'rb') as labels_file:
        batch_labels = pickle.load(labels_file)

    # Transform labels into one-hot encoded format
    batch_labels = label_binarizer.transform(batch_labels)

    return batch_data, batch_labels

# Data generator
def data_generator(prefix, num_batches):
    while True:
        for i in range(num_batches):
            X, y = load_batch_data(prefix, i)
            yield X, y

# Set parameters
train_prefix = '/content/drive/MyDrive/GSoC/task_1/prcsd_training/training'
val_prefix = '/content/drive/MyDrive/GSoC/task_1/prcsd_val/validation'
num_train_batches = 48  # Adjust this based on the number of training batches
num_val_batches = 12  # Adjust this based on the number of validation batches
batch_size = 8300

# Create data generators
train_generator = data_generator(train_prefix, num_train_batches)
val_generator = data_generator(val_prefix, num_val_batches)


# Set up the cosine annealing schedule for fine-tuning
```

```
fine_tuning_epochs = 50
steps_per_epoch = num_train_batches
validation_steps = num_val_batches
initial_learning_rate = 1e-4  # Set learning rate for fine-tuning
total_steps = fine_tuning_epochs * steps_per_epoch
cosine_decay = CosineDecay(initial_learning_rate, total_steps)

# Set the ROC-AUC threshold to 0.8
stop_at_roc_auc = StopAtROCAUC(threshold=0.8)

# Compile the loaded model with CosineDecay
rmsprop_optimizer = RMSprop(learning_rate=cosine_decay)
loss = 'binary_crossentropy'
metrics = [AUC(name='auc'), 'accuracy']

loaded_model.compile(optimizer=rmsprop_optimizer, loss=loss, metrics=metrics)

# Train the loaded model with CosineDecay
fine_tuning_history = loaded_model.fit(train_generator,
                                       epochs=fine_tuning_epochs,
                                       steps_per_epoch=steps_per_epoch,
                                       validation_data=val_generator,
                                       validation_steps=validation_steps,
                                       verbose=1,
                                       callbacks=[stop_at_roc_auc])
```

```
Epoch 1/50
3113/3113 [==============================] - 37s 11ms/step

ROC-AUC score for epoch 1: 0.5499
48/48 [==============================] - 182s 2s/step - loss: 0.4464 - auc: 0.8711 - accuracy: 0.7816 - val_loss: 0.9434 - val_auc: 0.5497 - val_accuracy: 0.5334
Epoch 2/50
3113/3113 [==============================] - 35s 11ms/step

ROC-AUC score for epoch 2: 0.5490
48/48 [==============================] - 66s 1s/step - loss: 0.4374 - auc: 0.8769 - accuracy: 0.7876 - val_loss: 0.9579 - val_auc: 0.5490 - val_accuracy: 0.5352
Epoch 3/50
3113/3113 [==============================] - 35s 11ms/step

ROC-AUC score for epoch 3: 0.5525
48/48 [==============================] - 66s 1s/step - loss: 0.4310 - auc: 0.8809 - accuracy: 0.7912 - val_loss: 0.9604 - val_auc: 0.5525 - val_accuracy: 0.5373
Epoch 4/50
3113/3113 [==============================] - 34s 11ms/step

ROC-AUC score for epoch 4: 0.5482
```

```
48/48 [==============================] - 66s 1s/step - loss: 0.4262 - auc: 0.8837 - accuracy: 0.7935 - val_loss: 0.9686 - val_auc: 0.5483 - val_accuracy: 0.5344
Epoch 5/50
3113/3113 [==============================] - 34s 11ms/step

ROC-AUC score for epoch 5: 0.5491
48/48 [==============================] - 65s 1s/step - loss: 0.4190 - auc: 0.8880 - accuracy: 0.7977 - val_loss: 0.9911 - val_auc: 0.5492 - val_accuracy: 0.5352
Epoch 6/50
3113/3113 [==============================] - 34s 11ms/step

ROC-AUC score for epoch 6: 0.5471
48/48 [==============================] - 66s 1s/step - loss: 0.4130 - auc: 0.8913 - accuracy: 0.8011 - val_loss: 0.9939 - val_auc: 0.5472 - val_accuracy: 0.5326
Epoch 7/50
3113/3113 [==============================] - 34s 11ms/step

ROC-AUC score for epoch 7: 0.5489
48/48 [==============================] - 66s 1s/step - loss: 0.4075 - auc: 0.8941 - accuracy: 0.8033 - val_loss: 1.0089 - val_auc: 0.5490 - val_accuracy: 0.5360
Epoch 8/50
3113/3113 [==============================] - 34s 11ms/step

ROC-AUC score for epoch 8: 0.5479
48/48 [==============================] - 66s 1s/step - loss: 0.4026 - auc: 0.8970 - accuracy: 0.8064 - val_loss: 1.0271 - val_auc: 0.5478 - val_accuracy: 0.5347
Epoch 9/50
3113/3113 [==============================] - 34s 11ms/step

ROC-AUC score for epoch 9: 0.5484
48/48 [==============================] - 66s 1s/step - loss: 0.3952 - auc: 0.9010 - accuracy: 0.8106 - val_loss: 1.0360 - val_auc: 0.5484 - val_accuracy: 0.5350
Epoch 10/50
3113/3113 [==============================] - 35s 11ms/step

ROC-AUC score for epoch 10: 0.5486
48/48 [==============================] - 66s 1s/step - loss: 0.3896 - auc: 0.9040 - accuracy: 0.8142 - val_loss: 1.0507 - val_auc: 0.5486 - val_accuracy: 0.5372
Epoch 11/50
3113/3113 [==============================] - 34s 11ms/step

ROC-AUC score for epoch 11: 0.5477
48/48 [==============================] - 65s 1s/step - loss: 0.3854 - auc: 0.9061 - accuracy: 0.8160 - val_loss: 1.0642 - val_auc: 0.5477 - val_accuracy: 0.5358
Epoch 12/50
3113/3113 [==============================] - 34s 11ms/step

ROC-AUC score for epoch 12: 0.5460
48/48 [==============================] - 65s 1s/step - loss: 0.3800 - auc: 0.9090 - accuracy: 0.8193 - val_loss: 1.0599 - val_auc: 0.5461 - val_accuracy: 0.5326
Epoch 13/50
3113/3113 [==============================] - 34s 11ms/step

ROC-AUC score for epoch 13: 0.5460
```

```
48/48 [==============================] - 65s 1s/step - loss: 0.3748 - auc: 0.9116 - accuracy: 0.8219 - val_loss: 1.0673 - val_auc: 0.5459 - val_accuracy: 0.5334
Epoch 14/50
3113/3113 [==============================] - 34s 11ms/step

ROC-AUC score for epoch 14: 0.5477
48/48 [==============================] - 66s 1s/step - loss: 0.3699 - auc: 0.9138 - accuracy: 0.8234 - val_loss: 1.0814 - val_auc: 0.5476 - val_accuracy: 0.5345
Epoch 15/50
3113/3113 [==============================] - 34s 11ms/step

ROC-AUC score for epoch 15: 0.5484
48/48 [==============================] - 65s 1s/step - loss: 0.3664 - auc: 0.9155 - accuracy: 0.8261 - val_loss: 1.0953 - val_auc: 0.5484 - val_accuracy: 0.5369
Epoch 16/50
3113/3113 [==============================] - 34s 11ms/step

ROC-AUC score for epoch 16: 0.5476
48/48 [==============================] - 66s 1s/step - loss: 0.3613 - auc: 0.9181 - accuracy: 0.8284 - val_loss: 1.1130 - val_auc: 0.5476 - val_accuracy: 0.5352
Epoch 17/50
3113/3113 [==============================] - 34s 11ms/step

ROC-AUC score for epoch 17: 0.5458
48/48 [==============================] - 66s 1s/step - loss: 0.3570 - auc: 0.9199 - accuracy: 0.8307 - val_loss: 1.1018 - val_auc: 0.5461 - val_accuracy: 0.5322
Epoch 18/50
3113/3113 [==============================] - 34s 11ms/step

ROC-AUC score for epoch 18: 0.5463
48/48 [==============================] - 66s 1s/step - loss: 0.3535 - auc: 0.9216 - accuracy: 0.8317 - val_loss: 1.1110 - val_auc: 0.5460 - val_accuracy: 0.5335
Epoch 19/50
3113/3113 [==============================] - 35s 11ms/step

ROC-AUC score for epoch 19: 0.5467
48/48 [==============================] - 66s 1s/step - loss: 0.3491 - auc: 0.9235 - accuracy: 0.8339 - val_loss: 1.1309 - val_auc: 0.5467 - val_accuracy: 0.5342
Epoch 20/50
3113/3113 [==============================] - 35s 11ms/step

ROC-AUC score for epoch 20: 0.5466
48/48 [==============================] - 66s 1s/step - loss: 0.3460 - auc: 0.9248 - accuracy: 0.8352 - val_loss: 1.1371 - val_auc: 0.5467 - val_accuracy: 0.5339
Epoch 21/50
3113/3113 [==============================] - 35s 11ms/step

ROC-AUC score for epoch 21: 0.5461
48/48 [==============================] - 66s 1s/step - loss: 0.3415 - auc: 0.9271 - accuracy: 0.8384 - val_loss: 1.1520 - val_auc: 0.5461 - val_accuracy: 0.5354
Epoch 22/50
3113/3113 [==============================] - 34s 11ms/step

ROC-AUC score for epoch 22: 0.5473
```

```
48/48 [==============================] - 65s 1s/step - loss: 0.3378 - auc: 0.9284 - accuracy: 0.8390 - val_loss: 1.1567 - val_auc: 0.5473 - val_accuracy: 0.5349
Epoch 23/50
3113/3113 [==============================] - 32s 10ms/step

ROC-AUC score for epoch 23: 0.5464
48/48 [==============================] - 63s 1s/step - loss: 0.3341 - auc: 0.9302 - accuracy: 0.8419 - val_loss: 1.1648 - val_auc: 0.5463 - val_accuracy: 0.5353
Epoch 24/50
3113/3113 [==============================] - 27s 9ms/step

ROC-AUC score for epoch 24: 0.5455
48/48 [==============================] - 58s 1s/step - loss: 0.3318 - auc: 0.9311 - accuracy: 0.8424 - val_loss: 1.1858 - val_auc: 0.5454 - val_accuracy: 0.5325
Epoch 25/50
3113/3113 [==============================] - 27s 9ms/step

ROC-AUC score for epoch 25: 0.5472
48/48 [==============================] - 59s 1s/step - loss: 0.3283 - auc: 0.9328 - accuracy: 0.8454 - val_loss: 1.1764 - val_auc: 0.5472 - val_accuracy: 0.5344
Epoch 26/50
3113/3113 [==============================] - 27s 9ms/step

ROC-AUC score for epoch 26: 0.5465
48/48 [==============================] - 58s 1s/step - loss: 0.3255 - auc: 0.9337 - accuracy: 0.8459 - val_loss: 1.1807 - val_auc: 0.5466 - val_accuracy: 0.5341
Epoch 27/50
3113/3113 [==============================] - 27s 9ms/step

ROC-AUC score for epoch 27: 0.5461
48/48 [==============================] - 58s 1s/step - loss: 0.3234 - auc: 0.9348 - accuracy: 0.8473 - val_loss: 1.1790 - val_auc: 0.5461 - val_accuracy: 0.5327
Epoch 28/50
3113/3113 [==============================] - 27s 9ms/step

ROC-AUC score for epoch 28: 0.5449
48/48 [==============================] - 58s 1s/step - loss: 0.3206 - auc: 0.9356 - accuracy: 0.8475 - val_loss: 1.1825 - val_auc: 0.5449 - val_accuracy: 0.5316
Epoch 29/50
3113/3113 [==============================] - 27s 9ms/step

ROC-AUC score for epoch 29: 0.5451
48/48 [==============================] - 58s 1s/step - loss: 0.3173 - auc: 0.9372 - accuracy: 0.8503 - val_loss: 1.2042 - val_auc: 0.5451 - val_accuracy: 0.5330
Epoch 30/50
3113/3113 [==============================] - 27s 9ms/step

ROC-AUC score for epoch 30: 0.5440
48/48 [==============================] - 58s 1s/step - loss: 0.3168 - auc: 0.9374 - accuracy: 0.8505 - val_loss: 1.1963 - val_auc: 0.5439 - val_accuracy: 0.5317
Epoch 31/50
3113/3113 [==============================] - 27s 9ms/step

ROC-AUC score for epoch 31: 0.5456
```

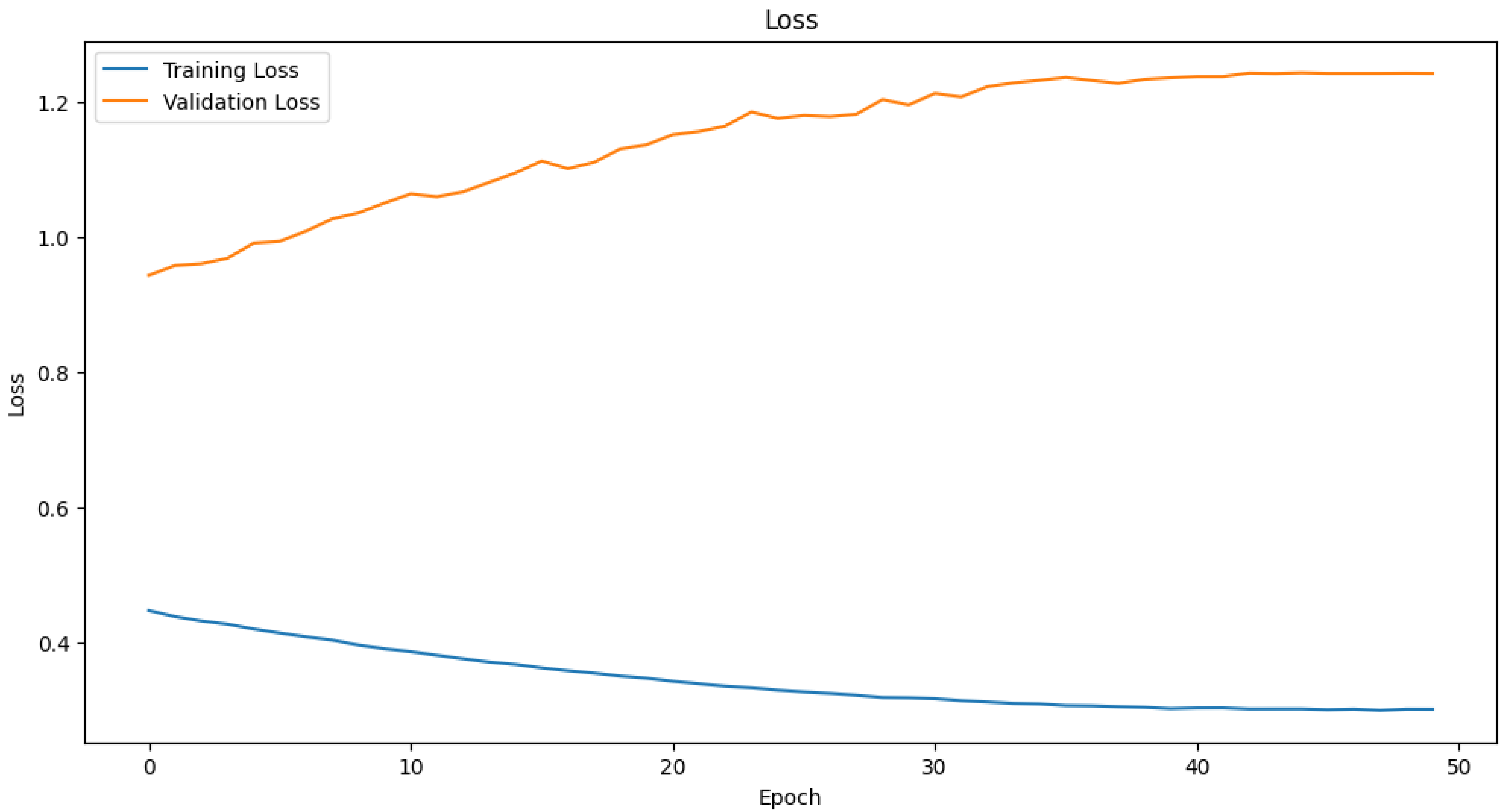
```
48/48 [==============================] - 58s 1s/step - loss: 0.3157 - auc: 0.9379 - accuracy: 0.8514 - val_loss: 1.2134 - val_auc: 0.5455 - val_accuracy: 0.5346
Epoch 32/50
3113/3113 [==============================] - 28s 9ms/step

ROC-AUC score for epoch 32: 0.5454
48/48 [==============================] - 59s 1s/step - loss: 0.3126 - auc: 0.9390 - accuracy: 0.8524 - val_loss: 1.2081 - val_auc: 0.5455 - val_accuracy: 0.5347
Epoch 33/50
3113/3113 [==============================] - 27s 9ms/step

ROC-AUC score for epoch 33: 0.5450
48/48 [==============================] - 58s 1s/step - loss: 0.3108 - auc: 0.9398 - accuracy: 0.8536 - val_loss: 1.2233 - val_auc: 0.5449 - val_accuracy: 0.5325
Epoch 34/50
3113/3113 [==============================] - 27s 9ms/step

ROC-AUC score for epoch 34: 0.5450
48/48 [==============================] - 58s 1s/step - loss: 0.3086 - auc: 0.9405 - accuracy: 0.8540 - val_loss: 1.2289 - val_auc: 0.5449 - val_accuracy: 0.5330
Epoch 35/50
3113/3113 [==============================] - 27s 9ms/step

ROC-AUC score for epoch 35: 0.5447
48/48 [==============================] - 58s 1s/step - loss: 0.3078 - auc: 0.9409 - accuracy: 0.8550 - val_loss: 1.2328 - val_auc: 0.5447 - val_accuracy: 0.5333
Epoch 36/50
3113/3113 [==============================] - 27s 9ms/step

ROC-AUC score for epoch 36: 0.5447
48/48 [==============================] - 58s 1s/step - loss: 0.3054 - auc: 0.9418 - accuracy: 0.8558 - val_loss: 1.2371 - val_auc: 0.5447 - val_accuracy: 0.5337
Epoch 37/50
3113/3113 [==============================] - 28s 9ms/step

ROC-AUC score for epoch 37: 0.5449
48/48 [==============================] - 59s 1s/step - loss: 0.3050 - auc: 0.9420 - accuracy: 0.8563 - val_loss: 1.2325 - val_auc: 0.5448 - val_accuracy: 0.5336
Epoch 38/50
3113/3113 [==============================] - 33s 11ms/step

ROC-AUC score for epoch 38: 0.5445
48/48 [==============================] - 64s 1s/step - loss: 0.3038 - auc: 0.9424 - accuracy: 0.8564 - val_loss: 1.2282 - val_auc: 0.5444 - val_accuracy: 0.5341
Epoch 39/50
3113/3113 [==============================] - 33s 11ms/step

ROC-AUC score for epoch 39: 0.5449
48/48 [==============================] - 64s 1s/step - loss: 0.3029 - auc: 0.9427 - accuracy: 0.8567 - val_loss: 1.2342 - val_auc: 0.5448 - val_accuracy: 0.5335
Epoch 40/50
3113/3113 [==============================] - 33s 10ms/step

ROC-AUC score for epoch 40: 0.5435
```

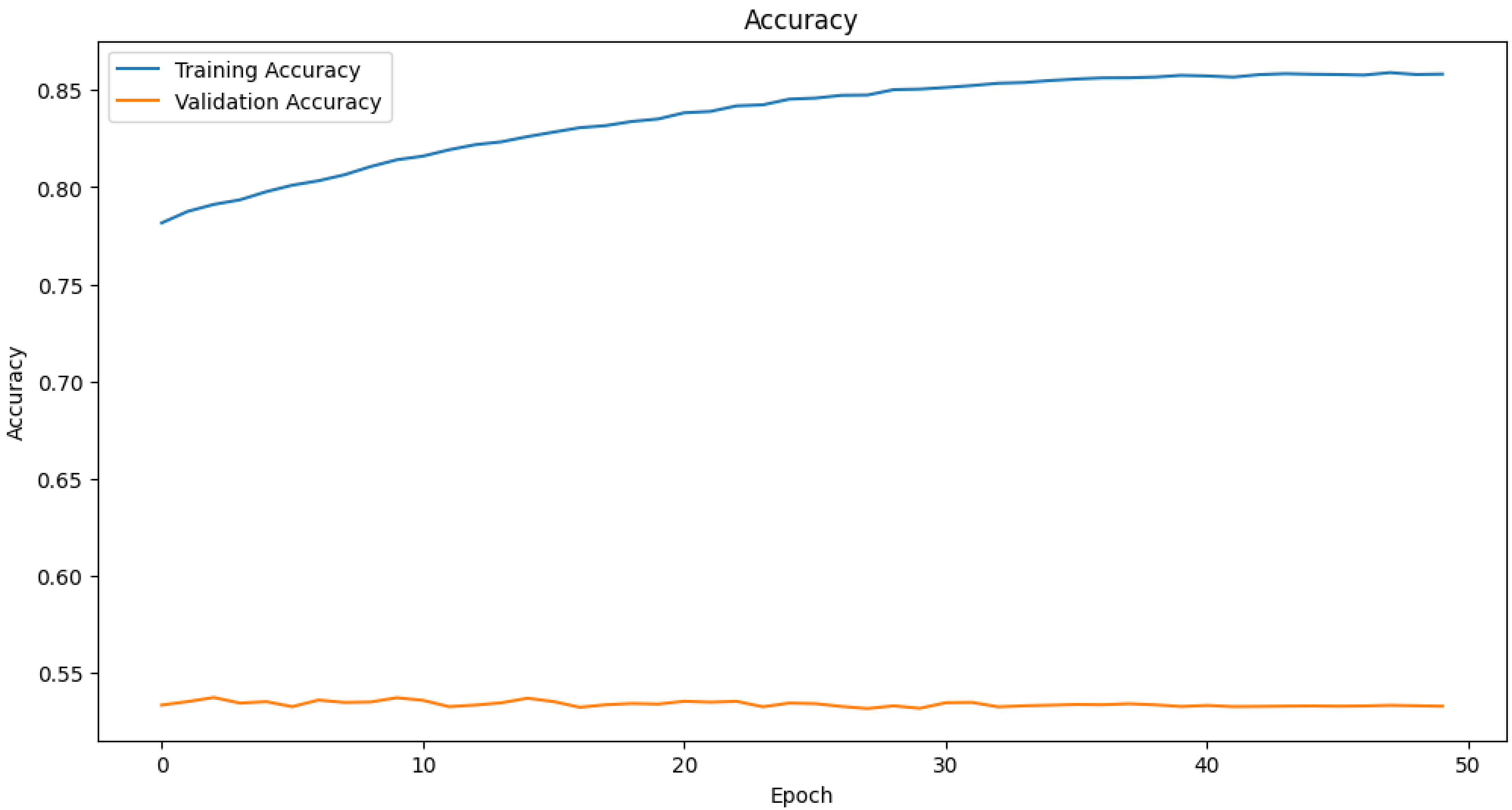
```
48/48 [==============================] - 64s 1s/step - loss: 0.3009 - auc: 0.9435 - accuracy: 0.8576 - val_loss: 1.2366 - val_auc: 0.5435 - val_accuracy: 0.5327
Epoch 41/50
3113/3113 [==============================] - 33s 11ms/step

ROC-AUC score for epoch 41: 0.5442
48/48 [==============================] - 64s 1s/step - loss: 0.3019 - auc: 0.9431 - accuracy: 0.8573 - val_loss: 1.2384 - val_auc: 0.5442 - val_accuracy: 0.5332
Epoch 42/50
3113/3113 [==============================] - 33s 11ms/step

ROC-AUC score for epoch 42: 0.5444
48/48 [==============================] - 64s 1s/step - loss: 0.3020 - auc: 0.9430 - accuracy: 0.8567 - val_loss: 1.2385 - val_auc: 0.5444 - val_accuracy: 0.5326
Epoch 43/50
3113/3113 [==============================] - 33s 11ms/step

ROC-AUC score for epoch 43: 0.5446
48/48 [==============================] - 64s 1s/step - loss: 0.3003 - auc: 0.9437 - accuracy: 0.8580 - val_loss: 1.2435 - val_auc: 0.5445 - val_accuracy: 0.5327
Epoch 44/50
3113/3113 [==============================] - 33s 10ms/step

ROC-AUC score for epoch 44: 0.5444
48/48 [==============================] - 64s 1s/step - loss: 0.3004 - auc: 0.9438 - accuracy: 0.8585 - val_loss: 1.2429 - val_auc: 0.5444 - val_accuracy: 0.5328
Epoch 45/50
3113/3113 [==============================] - 33s 11ms/step

ROC-AUC score for epoch 45: 0.5444
48/48 [==============================] - 64s 1s/step - loss: 0.3004 - auc: 0.9437 - accuracy: 0.8582 - val_loss: 1.2440 - val_auc: 0.5443 - val_accuracy: 0.5329
Epoch 46/50
3113/3113 [==============================] - 33s 11ms/step

ROC-AUC score for epoch 46: 0.5444
48/48 [==============================] - 64s 1s/step - loss: 0.2993 - auc: 0.9441 - accuracy: 0.8581 - val_loss: 1.2431 - val_auc: 0.5442 - val_accuracy: 0.5328
Epoch 47/50
3113/3113 [==============================] - 33s 11ms/step

ROC-AUC score for epoch 47: 0.5446
48/48 [==============================] - 64s 1s/step - loss: 0.3001 - auc: 0.9438 - accuracy: 0.8578 - val_loss: 1.2431 - val_auc: 0.5445 - val_accuracy: 0.5329
Epoch 48/50
3113/3113 [==============================] - 33s 11ms/step

ROC-AUC score for epoch 48: 0.5445
48/48 [==============================] - 64s 1s/step - loss: 0.2983 - auc: 0.9445 - accuracy: 0.8591 - val_loss: 1.2432 - val_auc: 0.5444 - val_accuracy: 0.5332
Epoch 49/50
3113/3113 [==============================] - 33s 11ms/step

ROC-AUC score for epoch 49: 0.5445
```

```
48/48 [==============================] - 64s 1s/step - loss: 0.3000 - auc: 0.9439 - accuracy: 0.8581 - val_loss: 1.2434 - val_auc: 0.5444 - val_accuracy: 0.5330
Epoch 50/50
3113/3113 [==============================] - 33s 11ms/step

ROC-AUC score for epoch 50: 0.5446
48/48 [==============================] - 64s 1s/step - loss: 0.2999 - auc: 0.9438 - accuracy: 0.8583 - val_loss: 1.2431 - val_auc: 0.5444 - val_accuracy: 0.5328
```

In [ ]:

```
import matplotlib.pyplot as plt

# Plot loss
plt.figure(figsize=(12, 6))
plt.plot(fine_tuning_history.history['loss'], label='Training Loss')
plt.plot(fine_tuning_history.history['val_loss'], label='Validation Loss')
plt.title('Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Plot accuracy
plt.figure(figsize=(12, 6))
plt.plot(fine_tuning_history.history['accuracy'], label='Training Accuracy')
plt.plot(fine_tuning_history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```


Loss
Training Loss
Validation Loss
1.2
1.0
0.8
0.6
0.4
Loss
0
10
20
30
40
50
Epoch

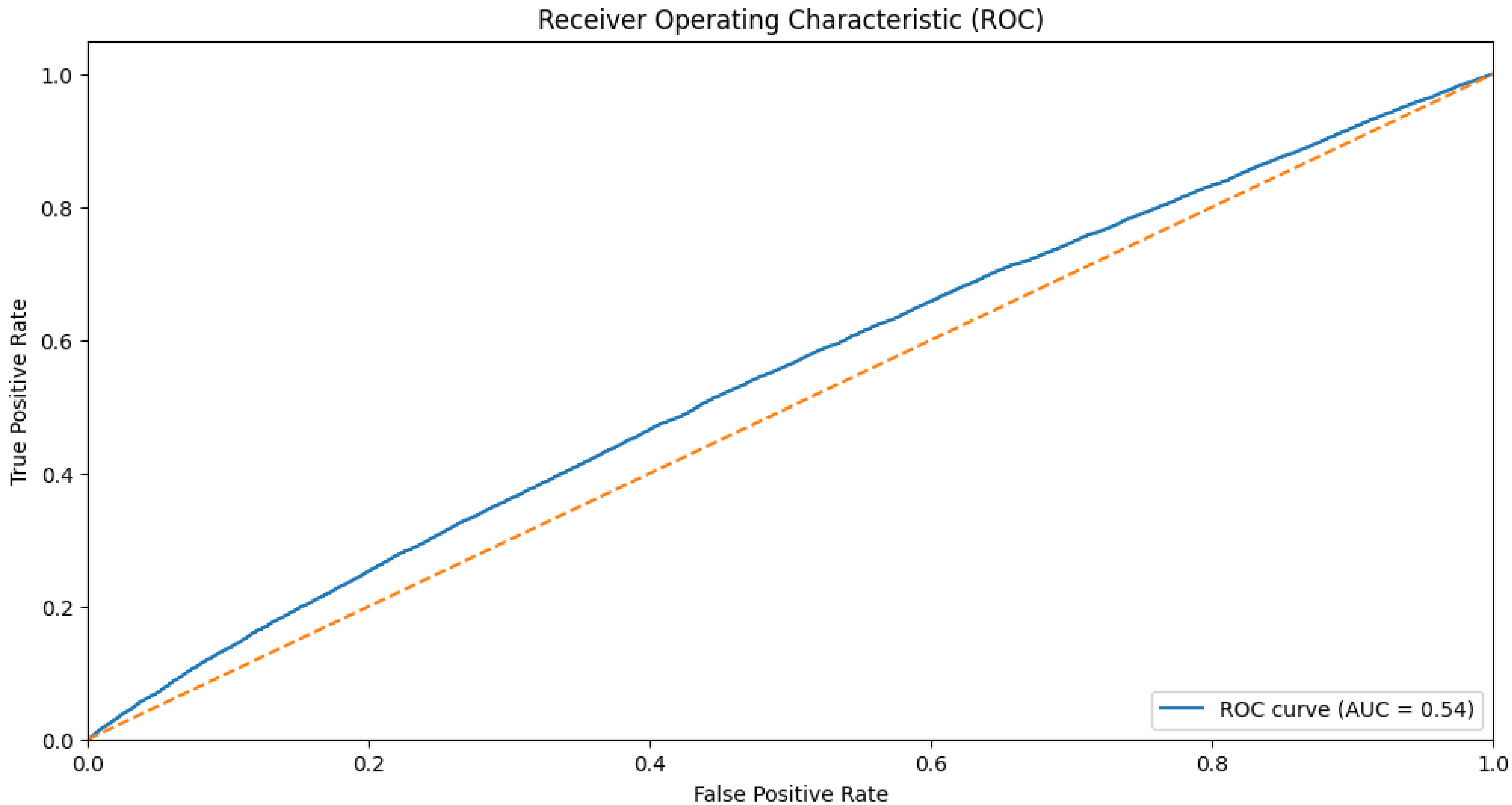


In [ ]:

```
import numpy as np
from sklearn.metrics import roc_curve, auc

# Concatenate validation data
X_val = []
y_val = []

for i in range(num_val_batches):
    X_batch, y_batch = load_batch_data(val_prefix, i)
    X_val.append(X_batch)
```

```
    y_val.append(y_batch)

X_val = np.concatenate(X_val)
y_val = np.concatenate(y_val)

# Get the predicted probabilities for the positive class (class 1)
y_pred_probs = loaded_model.predict(X_val)[:, 0]

# Compute the ROC curve
fpr, tpr, thresholds = roc_curve(y_val, y_pred_probs)

# Compute the AUC score
roc_auc = auc(fpr, tpr)

# Plot the ROC curve
plt.figure(figsize=(12, 6))
plt.plot(fpr, tpr, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 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 (ROC)')
plt.legend(loc="lower right")
plt.show()
```

```
3113/3113 [==============================] - 33s 11ms/step
```



In [ ]:

```
# Save the fine-tuned model
loaded_model.save_weights('/content/drive/MyDrive/GSoC/task_1/model/model_fine_tuned_weights.h5')
```