```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
!pip install opencv-python # install the OpenCV library
```

```
Requirement already satisfied: opencv-python in /usr/local/lib/python3.11/dist-packages (4.11.0.86)
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.11/dist-packages (from opencv-python) (2.0.2)
```

```
import h5py
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import Input, Dense, Flatten, Reshape
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam

def read_images_from_hdf5(hdf5_file_path):
    try:
        with h5py.File(hdf5_file_path, 'r') as hf:
            X = hf['X'][:]
            y = hf['y'][:]
            return X, y
    except Exception as e:
        print(f"Error reading HDF5 file: {e}")
        return None, None

df1= '/content/drive/MyDrive/GSOC/GSOC/task 1(A)/SingleElectronPt50_IMGCROPS_n249k_RHv1.hdf5'
X1, y1 = read_images_from_hdf5(df1)
X1 = X1.astype('float32') / 255.
y1 = np.zeros(y1.shape[0])


df2 = '/content/drive/MyDrive/GSOC/GSOC/task 1(A)/SinglePhotonPt50_IMGCROPS_n249k_RHv1 (1).hdf5'
X2, y2 = read_images_from_hdf5(df2)
X2 = X2.astype('float32') / 255.
y2 = np.ones(y2.shape[0])
```

```
import numpy as np
X = np.concatenate((X1, X2), axis=0)
y = np.concatenate((y1, y2), axis=0)

dataset = (X, y)
```

## Resnet 101

```
import torch
import torchvision.models as models
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader, random_split
import numpy as np
import torch.optim as optim
from torch.cuda.amp import GradScaler, autocast  # For mixed precision training


class CustomDataset(Dataset):
    def __init__(self, data, labels):
        self.data = data
        self.labels = labels

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        sample = self.data[idx]
        label = self.labels[idx]
        sample = torch.tensor(sample, dtype=torch.float32).permute(2, 0, 1)
        label = torch.tensor(label, dtype=torch.long)
```

```
        label = torch.tensor(label, dtype=torch.long)
        return sample, label

dataset = CustomDataset(X, y)

train_size = int(0.8 * len(dataset))
test_size = len(dataset) - train_size
train_dataset, test_dataset = random_split(dataset, [train_size, test_size])

batch_size = 128
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True, pin_memory=True, num_workers=4)  # Add pin_memory and num_w
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False, pin_memory=True, num_workers=4) # Add pin_memory and num_wor

model = models.resnet101(pretrained=True)
model.conv1 = nn.Conv2d(2, 64, kernel_size=7, stride=2, padding=3, bias=False)
num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, 1)
model.sigmoid = nn.Sigmoid()

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model.to(device)

criterion = nn.BCEWithLogitsLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
scaler = GradScaler()  # For mixed precision
```

```
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated sinc
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `None`
  warnings.warn(msg)
<ipython-input-11-305220b07f2a>:46: FutureWarning: `torch.cuda.amp.GradScaler(args...)` is deprecated. Please use `torch.amp.GradSca
  scaler = GradScaler()  # For mixed precision
```

```
import torch



num_epochs = 12
patience = 3  # Number of epochs to wait for improvement
best_val_loss = float('inf')
counter = 0

for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device, non_blocking=True), labels.to(device, non_blocking=True).float().view(-1, 1)
        optimizer.zero_grad()
        with torch.cuda.amp.autocast():  # Enable mixed precision
            outputs = model(inputs)
            loss = criterion(outputs, labels)
        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()
        running_loss += loss.item()

    print(f"Epoch {epoch + 1}, Loss: {running_loss / len(train_loader)}")

    # Validation
    model.eval()
    val_loss = 0.0
    with torch.no_grad():
        for inputs, labels in test_loader:
            inputs, labels = inputs.to(device, non_blocking=True), labels.to(device, non_blocking=True).float().view(-1, 1)
            with torch.cuda.amp.autocast():
                outputs = model(inputs)
                loss = criterion(outputs, labels)
            val_loss += loss.item()
    avg_val_loss = val_loss / len(test_loader)
    print(f"Epoch {epoch + 1}, Validation Loss: {avg_val_loss}")

    # Early stopping
    if avg_val_loss < best_val_loss:
        best_val_loss = avg_val_loss
        counter = 0
        # Optionally save the best model here:
        # torch.save(model.state_dict(), 'best_model.pth')
    else:
        counter += 1
        if counter >= patience:
```

```
            print("Early stopping triggered!")
            break

# Optionally load the best model:
# model.load_state_dict(torch.load('best_model.pth'))
```

```
<ipython-input-6-ab85f41bcaf1>:16: FutureWarning: `torch.cuda.amp.autocast(args...)` is deprecated. Please use `torch.amp.autocast(
  with torch.cuda.amp.autocast():  # Enable mixed precision
Epoch 1, Loss: 0.6692228934177432
<ipython-input-6-ab85f41bcaf1>:32: FutureWarning: `torch.cuda.amp.autocast(args...)` is deprecated. Please use `torch.amp.autocast(
  with torch.cuda.amp.autocast():
Epoch 1, Validation Loss: 0.6495453148353391
Epoch 2, Loss: 0.6349871239783179
Epoch 2, Validation Loss: 0.6307118173129139
Epoch 3, Loss: 0.5961407799353907
Epoch 3, Validation Loss: 0.6536196679144677
Epoch 4, Loss: 0.5818177659699972
Epoch 4, Validation Loss: 0.6753151659328916
Epoch 5, Loss: 0.5717245742913193
Epoch 5, Validation Loss: 0.5773331666023342
Epoch 6, Loss: 0.5701823152608899
Epoch 6, Validation Loss: 0.5879374582975605
Epoch 7, Loss: 0.5686407246334623
Epoch 7, Validation Loss: 0.5859232479096683
Epoch 8, Loss: nan
Epoch 8, Validation Loss: nan
Early stopping triggered!
```

```
model_save_path = '/content/drive/MyDrive/GSOC/saved_model1.pth'  # Specify the desired path
torch.save(model.state_dict(), model_save_path)
print(f"Model saved to: {model_save_path}")
```

```
Model saved to: /content/drive/MyDrive/GSOC/saved_model1.pth
```

```
num_epochs = 6
patience = 3  # Number of epochs to wait for improvement
best_val_loss = float('inf')
counter = 0

for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device, non_blocking=True), labels.to(device, non_blocking=True).float().view(-1, 1)
        optimizer.zero_grad()
        with torch.cuda.amp.autocast():  # Enable mixed precision
            outputs = model(inputs)
            loss = criterion(outputs, labels)
        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()
        running_loss += loss.item()

    print(f"Epoch {epoch + 1}, Loss: {running_loss / len(train_loader)}")
```

```
model.eval()
correct = 0
total = 0
with torch.no_grad():
    for inputs, labels in test_loader:
        inputs, labels = inputs.to(device, non_blocking=True), labels.to(device, non_blocking=True).float().view(-1, 1) # Add non_block:
        outputs = model(inputs)
        predicted = (torch.sigmoid(outputs) > 0.5).float()
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f"Accuracy on test set: {100 * correct / total}%")
```

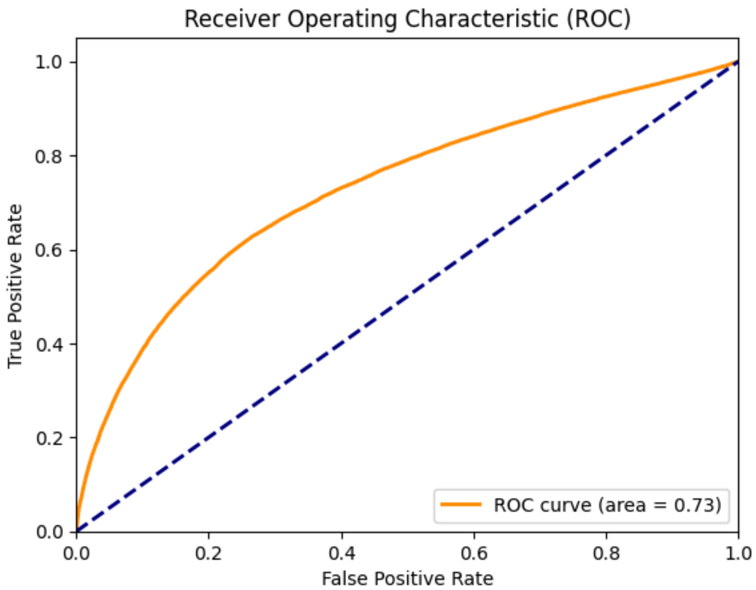
```
Accuracy on test set: 64.30923694779116%
```

```
model.eval()
all_preds = []
all_labels = []
with torch.no_grad():
    for inputs, labels in test_loader:
        inputs, labels = inputs.to(device, non_blocking=True), labels.to(device, non_blocking=True).float().view(-1, 1)
        outputs = model(inputs)
        predicted = torch.sigmoid(outputs)  # Get probabilities
```

```
        all_preds.extend(predicted.cpu().numpy().flatten())
        all_labels.extend(labels.cpu().numpy().flatten())
```

```
from sklearn.metrics import f1_score, roc_curve, auc, classification_report
import matplotlib.pyplot as plt

threshold = 0.5  # Adjust threshold as needed
binary_preds = [1 if p > threshold else 0 for p in all_preds]
f1 = f1_score(all_labels, binary_preds)
print(f"F1 Score: {f1}")

# Calculate ROC curve and AUC
fpr, tpr, thresholds = roc_curve(all_labels, all_preds)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, 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()

# Classification report
print("Classification Report:")
print(classification_report(all_labels, binary_preds))
```

```
F1 Score: 0.5193228222949401
```




```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.59      0.90      0.72     49807
         1.0       0.79      0.39      0.52     49793

    accuracy                           0.64     99600
   macro avg       0.69      0.64      0.62     99600
weighted avg       0.69      0.64      0.62     99600
```