```
import numpy as np
import h5py


hf = h5py.File('/content/drive/MyDrive/data/SingleElectronPt50_IMGCROPS_n249k_RHv1.hdf5','r')
x = hf.get('X')[:50000]
y = hf.get('y')[:50000]

x_electron = np.asarray(x)
y_electron = np.asarray(y)


hf1 = h5py.File('/content/drive/MyDrive/data/SinglePhotonPt50_IMGCROPS_n249k_RHv1.hdf5','r')
x1 = hf1.get('X')[:50000]
y1 = hf1.get('y')[:50000]

x_photon = np.asarray(x1)
y_photon = np.asarray(y1)


x_data = np.concatenate((x_electron,x_photon),axis=0)
y_data = np.concatenate((y_electron,y_photon),axis=0)
avg_channel = np.mean(x_data[:,:, :, :2], axis=-1, keepdims=True)

# Concatenate the average channel with the original image
x_data = np.concatenate((x_data, avg_channel), axis=-1)


import tensorflow as tf
from tensorflow import keras
from keras import layers
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
%matplotlib inline


x_train, x_validate, y_train, y_validate = train_test_split(x_data, y_data, test_size=0.2, random_state=42)
x_test, x_val, y_test, y_val = train_test_split(x_validate, y_validate, test_size=0.5, random_state=42)


import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

# Define the number of classes in the classification task
num_classes = 1

# Define the patch size and number of patches
patch_size = 4
num_patches = (32 // patch_size) ** 2

# Define the embedding dimension and transformer block parameters
embedding_dim = 64
transformer_units = [
    embedding_dim * 2,
    embedding_dim,
]

# Define the input layer for the image
inputs = layers.Input(shape=(32, 32, 3))

# Split the image into patches
patches = layers.Reshape((num_patches, patch_size * patch_size * 3))(inputs)

# Add a learnable embedding to each patch vector
embedding_layer = layers.Dense(embedding_dim)
embedded_patches = embedding_layer(patches)

# Add positional encoding to the embedded patches
positional_encoding_layer = layers.Embedding(input_dim=num_patches, output_dim=embedding_dim)
positions = tf.range(start=0, limit=num_patches, delta=1)
position_encodings = positional_encoding_layer(positions)
encoded_patches = embedded_patches + position_encodings

# Apply a stack of transformer blocks
for units in transformer_units:
    transformer_layer = layers.MultiHeadAttention(num_heads=8, key_dim=embedding_dim // 8)
    x1 = transformer_layer(encoded_patches, encoded_patches)
    x1 = layers.Dropout(0.1)(x1)
    x2 = layers.Add()([x1, encoded_patches])
    x3 = layers.LayerNormalization()(x2)
    x4 = layers.Dense(units, activation="relu")(x3)
```

```
    x5 = layers.Dropout(0.1)(x4)
    x6 = layers.Dense(embedding_dim)(x5)
    x7 = layers.Dropout(0.1)(x6)
    encoded_patches = layers.Add()([x7, x2])
    encoded_patches = layers.LayerNormalization()(encoded_patches)

# Apply global average pooling to obtain a single feature vector
features = layers.GlobalAveragePooling1D()(encoded_patches)

# Add a classification output layer
outputs = layers.Dense(num_classes, activation="sigmoid")(features)

# Create the model
model = keras.Model(inputs=inputs, outputs=outputs)

# Print the model summary
model.summary()
```

```
 dropout_6 (Dropout)            (None, 64, 64)       0           ['multi_head_attention_2[0][0]']

 add_4 (Add)                    (None, 64, 64)       0           ['dropout_6[0][0]',
                                                                  'tf.__operators__.add_1[0][0]']

 layer_normalization_4 (LayerNo  (None, 64, 64)      128         ['add_4[0][0]']
 rmalization)

 dense_7 (Dense)                (None, 64, 128)      8320        ['layer_normalization_4[0][0]']

 dropout_7 (Dropout)            (None, 64, 128)      0           ['dense_7[0][0]']

 dense_8 (Dense)                (None, 64, 64)       8256        ['dropout_7[0][0]']

 dropout_8 (Dropout)            (None, 64, 64)       0           ['dense_8[0][0]']

 add_5 (Add)                    (None, 64, 64)       0           ['dropout_8[0][0]',
                                                                  'add_4[0][0]']

 layer_normalization_5 (LayerNo  (None, 64, 64)      128         ['add_5[0][0]']
 rmalization)

 multi_head_attention_3 (MultiH  (None, 64, 64)      16640       ['layer_normalization_5[0][0]',
 eadAttention)                                                    'layer_normalization_5[0][0]']

 dropout_9 (Dropout)            (None, 64, 64)       0           ['multi_head_attention_3[0][0]']

 add_6 (Add)                    (None, 64, 64)       0           ['dropout_9[0][0]',
                                                                  'layer_normalization_5[0][0]']

 layer_normalization_6 (LayerNo  (None, 64, 64)      128         ['add_6[0][0]']
 rmalization)

 dense_9 (Dense)                (None, 64, 64)       4160        ['layer_normalization_6[0][0]']

 dropout_10 (Dropout)           (None, 64, 64)       0           ['dense_9[0][0]']

 dense_10 (Dense)               (None, 64, 64)       4160        ['dropout_10[0][0]']

 dropout_11 (Dropout)           (None, 64, 64)       0           ['dense_10[0][0]']

 add_7 (Add)                    (None, 64, 64)       0           ['dropout_11[0][0]',
                                                                  'add_6[0][0]']

 layer_normalization_7 (LayerNo  (None, 64, 64)      128         ['add_7[0][0]']
 rmalization)

 global_average_pooling1d_1 (Gl  (None, 64)          0           ['layer_normalization_7[0][0]']
 obalAveragePooling1D)

 dense_11 (Dense)               (None, 1)            65          ['global_average_pooling1d_1[0][0
                                                                 ]']

==================================================================================================
Total params: 61,889
Trainable params: 61,889
Non-trainable params: 0
__________________________________________________________________________________________________
```

```
from keras.optimizers import Adam

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

```
Epoch 1/20
2500/2500 [==============================] - 32s 10ms/step - loss: 0.6667 - accuracy: 0.5893 - val_loss: 0.6465 - val_acc
Epoch 2/20
```

```
2500/2500 [==============================] - 24s 10ms/step - loss: 0.6378 - accuracy: 0.6429 - val_loss: 0.6370 - val_acc
Epoch 3/20
2500/2500 [==============================] - 24s 9ms/step - loss: 0.6237 - accuracy: 0.6584 - val_loss: 0.6350 - val_accu
Epoch 4/20
2500/2500 [==============================] - 24s 10ms/step - loss: 0.6137 - accuracy: 0.6705 - val_loss: 0.6126 - val_acc
Epoch 5/20
2500/2500 [==============================] - 26s 10ms/step - loss: 0.6058 - accuracy: 0.6795 - val_loss: 0.6070 - val_acc
Epoch 6/20
2500/2500 [==============================] - 26s 10ms/step - loss: 0.6021 - accuracy: 0.6819 - val_loss: 0.6126 - val_acc
Epoch 7/20
2500/2500 [==============================] - 25s 10ms/step - loss: 0.5987 - accuracy: 0.6878 - val_loss: 0.6146 - val_acc
Epoch 8/20
2500/2500 [==============================] - 23s 9ms/step - loss: 0.5948 - accuracy: 0.6898 - val_loss: 0.6149 - val_accu
Epoch 9/20
2500/2500 [==============================] - 23s 9ms/step - loss: 0.5933 - accuracy: 0.6919 - val_loss: 0.5969 - val_accu
Epoch 10/20
2500/2500 [==============================] - 23s 9ms/step - loss: 0.5913 - accuracy: 0.6940 - val_loss: 0.6010 - val_accu
Epoch 11/20
2500/2500 [==============================] - 25s 10ms/step - loss: 0.5891 - accuracy: 0.6967 - val_loss: 0.5953 - val_acc
Epoch 12/20
2500/2500 [==============================] - 23s 9ms/step - loss: 0.5865 - accuracy: 0.6984 - val_loss: 0.5944 - val_accu
Epoch 13/20
2500/2500 [==============================] - 23s 9ms/step - loss: 0.5858 - accuracy: 0.6979 - val_loss: 0.5894 - val_accu
Epoch 14/20
2500/2500 [==============================] - 24s 10ms/step - loss: 0.5847 - accuracy: 0.6999 - val_loss: 0.5899 - val_acc
Epoch 15/20
2500/2500 [==============================] - 24s 9ms/step - loss: 0.5828 - accuracy: 0.7000 - val_loss: 0.6084 - val_accu
Epoch 16/20
2500/2500 [==============================] - 22s 9ms/step - loss: 0.5816 - accuracy: 0.7015 - val_loss: 0.6027 - val_accu
Epoch 17/20
2500/2500 [==============================] - 23s 9ms/step - loss: 0.5819 - accuracy: 0.7020 - val_loss: 0.5967 - val_accu
Epoch 18/20
2500/2500 [==============================] - 24s 10ms/step - loss: 0.5814 - accuracy: 0.7022 - val_loss: 0.5921 - val_acc
Epoch 19/20
2500/2500 [==============================] - 24s 9ms/step - loss: 0.5795 - accuracy: 0.7048 - val_loss: 0.5915 - val_accu
Epoch 20/20
2500/2500 [==============================] - 22s 9ms/step - loss: 0.5790 - accuracy: 0.7055 - val_loss: 0.5975 - val_accu
<keras.callbacks.History at 0x7fd9e5914670>
```

```
from sklearn.metrics import roc_auc_score
pred_prob1 = model.predict(x_test)
auc_score1 = roc_auc_score(y_test, pred_prob1[:])
auc_score1
```

```
313/313 [==============================] - 2s 5ms/step
0.7651813666322997
```


✓ 0s completed at 20:57