Introduction

Machine Learning algorithms have become an increasingly important tool for analyzing the data from the Large Hadron Collider (LHC). Identification of particles in LHC collisions is an important task of LHC detector reconstruction algorithms.

Here we present a challenge where one of the detectors (the Electromagnetic Calorimeter or ECAL) is used as a camera to analyze detector images from two types of particles: electrons and photons that deposit their energy in this detector.

Dataset

Each pixel in the image corresponds to a detector cell, while the intensity of the pixel corresponds to how much energy is measured in that cell. Timing of the energy deposits are also available, though this may or may not be relevant. The dataset contains 32x32 Images of the energy hits and their timing (channel 1: hit energy and channel 2: its timing) in each calorimeter cell (one cell = one pixel) for the two classes of particles: Electrons and Photons. The dataset contains around four hundred thousand images for electrons and photons. Please note that your final model will be evaluated on an unseen test dataset.

Algorithm

Please use a Machine Learning model of your choice to achieve the highest possible classification performance on the provided dataset. Please provide a Jupyter Notebook that shows your solution.

Evaluation Metrics ROC curve (Receiver Operating Characteristic curve) and AUC score (Area Under the ROC Curve) Training and Validation Accuracy The model performance will be tested on a hidden test dataset based on the above metrics.

Deliverables

Google Colab Jupyter Notebook showing your solution along with the final model accuracy (Training and Validation), ROC curve and AUC score. More details regarding the format of the notebook can be found in the sample Google Colab notebook provided for this challenge. The final trained model including the model architecture and the trained weights (For example: HDF5 file, .pb file, .pt file, etc.). You are free to choose Machine Learning Framework of your choice.

Run in Google Colab

Create the appropriate project folder

```
mkdir Particle_Images
```

```
    mkdir: cannot create directory ‘Particle_Images’: File exists
```

```
cd Particle_Images
```

```
    /content/Particle_Images
```

```
mkdir data/
```

```
    mkdir: cannot create directory ‘data/’: File exists
```

Download the Dataset

This will download 83MB for SingleElectron and 76MB for SinglePhoton.

```
#!/bin/bash
!wget https://cernbox.cern.ch/index.php/s/sHjzCNFTFxutYCj/download -O data/SingleElectronPt50_IMGCROPS_n249k_RHv1.hdf5
!wget https://cernbox.cern.ch/index.php/s/69nGEZjOy3xGxBq/download -O data/SinglePhotonPt50_IMGCROPS_n249k_RHv1.hdf5
```

```
    --2022-04-03 16:38:14--  https://cernbox.cern.ch/index.php/s/sHjzCNFTFxutYCj/download
    Resolving cernbox.cern.ch (cernbox.cern.ch)... 137.138.120.151, 188.184.97.72, 128.142.53.28, ...
    Connecting to cernbox.cern.ch (cernbox.cern.ch)|137.138.120.151|:443... connected.
    HTTP request sent, awaiting response... 200 OK
    Length: 87010508 (83M) [application/octet-stream]
    Saving to: ‘data/SingleElectronPt50_IMGCROPS_n249k_RHv1.hdf5’

    data/SingleElectron 100%[===================>]  82.98M  19.3MB/s    in 4.3s

    Last-modified header invalid -- time-stamp ignored.
    2022-04-03 16:38:20 (19.3 MB/s) - ‘data/SingleElectronPt50_IMGCROPS_n249k_RHv1.hdf5’ saved [87010508/87010508]

    --2022-04-03 16:38:21--  https://cernbox.cern.ch/index.php/s/69nGEZjOy3xGxBq/download
    Resolving cernbox.cern.ch (cernbox.cern.ch)... 137.138.120.151, 188.184.97.72, 128.142.53.28, ...
    Connecting to cernbox.cern.ch (cernbox.cern.ch)|137.138.120.151|:443... connected.
    HTTP request sent, awaiting response... 200 OK
    Length: 79876391 (76M) [application/octet-stream]
    Saving to: ‘data/SinglePhotonPt50_IMGCROPS_n249k_RHv1.hdf5’

    data/SinglePhotonPt 100%[===================>]  76.18M  18.4MB/s    in 4.1s

    Last-modified header invalid -- time-stamp ignored.
    2022-04-03 16:38:27 (18.4 MB/s) - ‘data/SinglePhotonPt50_IMGCROPS_n249k_RHv1.hdf5’ saved [79876391/79876391]
```

Import modules

```
import numpy as np
np.random.seed(1337)  # for reproducibility
import h5py
from keras.models import Sequential
from keras.initializers import TruncatedNormal
from keras.layers import Input, Dense, Dropout, Flatten, Conv2D, MaxPooling2D
from keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import roc_curve, auc
from tensorflow import keras
import matplotlib.pyplot as plt
```

```
import tensorflow as tf
tf.config.list_physical_devices('GPU')
```

```
    [PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

Keras Model Parameters

```
lr_init     = 1.e-3    # Initial learning rate
batch_size  = 512       # Training batch size
train_size  = 160000     # Training size
valid_size  = 25600     # Validation size
test_size   = 25600     # Test size
epochs      = 50       # Number of epochs
```

It is recommended to use GPU for training and inference if possible.

Load Image Data

- Two classes of particles: electrons and photons
- 32x32 matrices (two channels - hit energy and time) for the two classes of particles electrons and photons impinging on a calorimeter (one calorimetric cell = one pixel).
- Note that although timing channel is provided, it may not necessarily help the performance of the model.

```
img_rows, img_cols, nb_channels = 32, 32, 2
input_dir = 'data'
decays = ['SinglePhotonPt50_IMGCROPS_n249k_RHv1', 'SingleElectronPt50_IMGCROPS_n249k_RHv1']

def load_data(decays, start, stop):
    global input_dir
    dsets = [h5py.File('%s/%s.hdf5'%(input_dir,decay)) for decay in decays]
    X = np.concatenate([dset['/X'][start:stop] for dset in dsets])
    y = np.concatenate([dset['/y'][start:stop] for dset in dsets])
    assert len(X) == len(y)
    return X, y
```

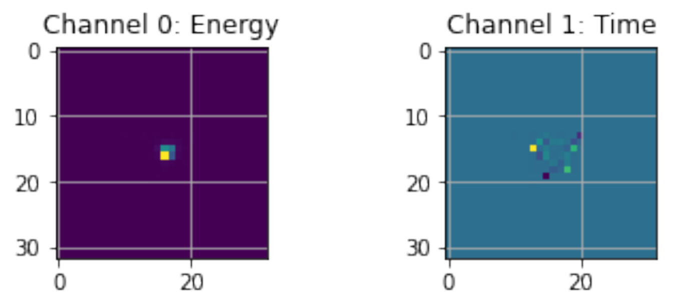
Configure Training / Validation / Test Sets

```
# Set range of training set
train_start, train_stop = 0, train_size
assert train_stop > train_start
assert (len(decays)*train_size) % batch_size == 0
X_train, y_train = load_data(decays,train_start,train_stop)

# Set range of validation set
valid_start, valid_stop = 160000, 160000+valid_size
assert valid_stop  >  valid_start
assert valid_start >= train_stop
X_valid, y_valid = load_data(decays,valid_start,valid_stop)

# Set range of test set
test_start, test_stop = 160000+valid_size, 160000+valid_size+test_size
assert test_stop  >  test_start
assert test_start >= valid_stop
X_test, y_test = load_data(decays,test_start,test_stop)

samples_requested = len(decays) * (train_size + valid_size + test_size)
samples_available = len(y_train) + len(y_valid) + len(y_test)
assert samples_requested == samples_available
```

Plot sample of training images

Note that although the timing channel is provided, it may not necessarily help the performance of the model.

```
plt.figure(1)

plt.subplot(221)
plt.imshow(X_train[1,:,:,0])
plt.title("Channel 0: Energy")  # Energy
plt.grid(True)

plt.subplot(222)
plt.imshow(X_train[1,:,:,1])
plt.title("Channel 1: Time")  # Time
plt.grid(True)


plt.show()
```



Define CNN Model

This is a sample model. You can experiment with the model and try various architectures and other models to achieve the highest possible performance.

```
keras.backend.clear_session()
model = Sequential()
model.add(Conv2D(32, activation='relu', kernel_size=3, padding='same', kernel_initializer='TruncatedNormal', input_shape=(img_rows, img_cols, nb_channels)))
model.add(Conv2D(32, activation='relu', kernel_size=3, padding='same', kernel_initializer='TruncatedNormal'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, activation='relu', kernel_size=3, padding='same', kernel_initializer='TruncatedNormal'))
model.add(Conv2D(64, activation='relu', kernel_size=3, padding='same', kernel_initializer='TruncatedNormal'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, activation='relu', kernel_size=3, padding='same', kernel_initializer='TruncatedNormal'))
model.add(Conv2D(128, activation='relu', kernel_size=3, padding='same', kernel_initializer='TruncatedNormal'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(256, activation='relu', kernel_initializer='TruncatedNormal'))
model.add(Dropout(0.4))
model.add(Dense(128, activation='relu', kernel_initializer='TruncatedNormal'))
model.add(Dropout(0.4))
model.add(Dense(128, activation='relu', kernel_initializer='TruncatedNormal'))
model.add(Dropout(0.25))
model.add(Dense(64, activation='relu', kernel_initializer='TruncatedNormal'))
model.add(Dropout(0.3))
model.add(Dense(1, activation='sigmoid', kernel_initializer='TruncatedNormal'))
model.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=lr_init), metrics=['accuracy'])
model.summary()
```

```
    Model: "sequential"
    _________________________________________________________________
     Layer (type)                Output Shape              Param #   
    =================================================================
     conv2d (Conv2D)             (None, 32, 32, 32)        608       
                                                                     
     conv2d_1 (Conv2D)           (None, 32, 32, 32)        9248      
                                                                     
     max_pooling2d (MaxPooling2D  (None, 16, 16, 32)       0         
     )                                                               
                                                                     
     conv2d_2 (Conv2D)           (None, 16, 16, 64)        18496     
                                                                     
     conv2d_3 (Conv2D)           (None, 16, 16, 64)        36928     
                                                                     
     max_pooling2d_1 (MaxPooling  (None, 8, 8, 64)         0         
     2D)                                                             
                                                                     
     conv2d_4 (Conv2D)           (None, 8, 8, 128)         73856     
                                                                     
     conv2d_5 (Conv2D)           (None, 8, 8, 128)         147584    
                                                                     
     max_pooling2d_2 (MaxPooling  (None, 4, 4, 128)        0         
     2D)                                                             
                                                                     
     flatten (Flatten)           (None, 2048)              0         
                                                                     
     dense (Dense)               (None, 256)               524544    
                                                                     
     dropout (Dropout)           (None, 256)               0         
                                                                     
     dense_1 (Dense)             (None, 128)               32896     
                                                                     
     dropout_1 (Dropout)         (None, 128)               0         
                                                                     
     dense_2 (Dense)             (None, 128)               16512     
                                                                     
     dropout_2 (Dropout)         (None, 128)               0         
                                                                     
     dense_3 (Dense)             (None, 64)                8256      
                                                                     
     dropout_3 (Dropout)         (None, 64)                0         
                                                                     
     dense_4 (Dense)             (None, 1)                 65        
                                                                     
    =================================================================
    Total params: 868,993
    Trainable params: 868,993
    Non-trainable params: 0
    _________________________________________________________________
```

Train the Model

You may further optimize the model, tune hyper-parameters, etc. accordingly to achieve the best performance possible.

```
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=2, min_delta=0.0001, cooldown=1, min_lr=1e-10, mode='auto')
checkpoint_cb = tf.keras.callbacks.ModelCheckpoint("model.h5", save_best_only=True)
early_stopping_cb = tf.keras.callbacks.EarlyStopping(patience=4, restore_best_weights = True)
history=model.fit(X_train, y_train,\
        batch_size=batch_size,\
        epochs=epochs,\
        validation_data=(X_valid, y_valid),\
        callbacks=[reduce_lr, checkpoint_cb, early_stopping_cb],\
        verbose=1, shuffle=True, initial_epoch=0
        )
model = tf.keras.models.load_model("model.h5")
```

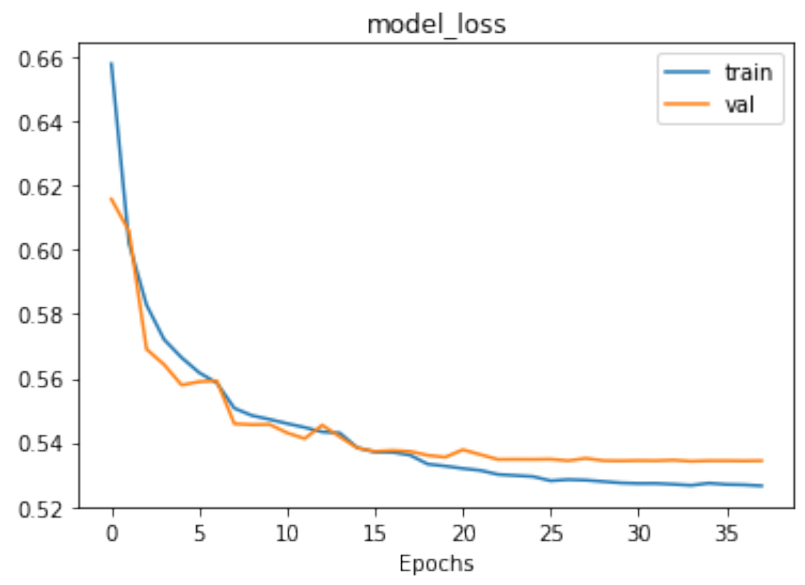
```
625/625 [==============================] - 60s 96ms/step - loss: 0.5484 - accuracy: 0.7293 - val_loss: 0.5456 - val_accuracy: 0.7296 - lr: 5.0000e-04
Epoch 10/50
625/625 [==============================] - 60s 95ms/step - loss: 0.5472 - accuracy: 0.7297 - val_loss: 0.5457 - val_accuracy: 0.7287 - lr: 5.0000e-04
Epoch 11/50
625/625 [==============================] - 62s 99ms/step - loss: 0.5459 - accuracy: 0.7311 - val_loss: 0.5430 - val_accuracy: 0.7339 - lr: 5.0000e-04
Epoch 12/50
625/625 [==============================] - 60s 95ms/step - loss: 0.5447 - accuracy: 0.7322 - val_loss: 0.5412 - val_accuracy: 0.7335 - lr: 5.0000e-04
Epoch 13/50
625/625 [==============================] - 62s 99ms/step - loss: 0.5433 - accuracy: 0.7329 - val_loss: 0.5454 - val_accuracy: 0.7317 - lr: 5.0000e-04
Epoch 14/50
625/625 [==============================] - 59s 95ms/step - loss: 0.5430 - accuracy: 0.7337 - val_loss: 0.5418 - val_accuracy: 0.7330 - lr: 5.0000e-04
Epoch 15/50
625/625 [==============================] - 60s 95ms/step - loss: 0.5384 - accuracy: 0.7370 - val_loss: 0.5383 - val_accuracy: 0.7353 - lr: 2.5000e-04
Epoch 16/50
625/625 [==============================] - 60s 96ms/step - loss: 0.5370 - accuracy: 0.7373 - val_loss: 0.5373 - val_accuracy: 0.7365 - lr: 2.5000e-04
Epoch 17/50
625/625 [==============================] - 59s 95ms/step - loss: 0.5370 - accuracy: 0.7376 - val_loss: 0.5376 - val_accuracy: 0.7358 - lr: 2.5000e-04
Epoch 18/50
625/625 [==============================] - 60s 95ms/step - loss: 0.5361 - accuracy: 0.7382 - val_loss: 0.5373 - val_accuracy: 0.7358 - lr: 2.5000e-04
Epoch 19/50
625/625 [==============================] - 62s 99ms/step - loss: 0.5333 - accuracy: 0.7399 - val_loss: 0.5361 - val_accuracy: 0.7372 - lr: 1.2500e-04
Epoch 20/50
625/625 [==============================] - 60s 96ms/step - loss: 0.5327 - accuracy: 0.7402 - val_loss: 0.5355 - val_accuracy: 0.7379 - lr: 1.2500e-04
Epoch 21/50
625/625 [==============================] - 62s 98ms/step - loss: 0.5319 - accuracy: 0.7406 - val_loss: 0.5378 - val_accuracy: 0.7379 - lr: 1.2500e-04
Epoch 22/50
625/625 [==============================] - 62s 99ms/step - loss: 0.5314 - accuracy: 0.7412 - val_loss: 0.5363 - val_accuracy: 0.7364 - lr: 1.2500e-04
Epoch 23/50
625/625 [==============================] - 62s 99ms/step - loss: 0.5301 - accuracy: 0.7424 - val_loss: 0.5348 - val_accuracy: 0.7377 - lr: 6.2500e-05
Epoch 24/50
625/625 [==============================] - 62s 99ms/step - loss: 0.5298 - accuracy: 0.7424 - val_loss: 0.5348 - val_accuracy: 0.7371 - lr: 6.2500e-05
Epoch 25/50
625/625 [==============================] - 59s 95ms/step - loss: 0.5294 - accuracy: 0.7426 - val_loss: 0.5348 - val_accuracy: 0.7384 - lr: 6.2500e-05
Epoch 26/50
625/625 [==============================] - 62s 99ms/step - loss: 0.5281 - accuracy: 0.7433 - val_loss: 0.5348 - val_accuracy: 0.7380 - lr: 3.1250e-05
Epoch 27/50
625/625 [==============================] - 62s 100ms/step - loss: 0.5285 - accuracy: 0.7435 - val_loss: 0.5344 - val_accuracy: 0.7383 - lr: 3.1250e-05
```

```
Epoch 28/50
625/625 [==============================] - 62s 100ms/step - loss: 0.5283 - accuracy: 0.7432 - val_loss: 0.5351 - val_accuracy: 0.7384 - lr: 3.1250e-05
Epoch 29/50
625/625 [==============================] - 62s 99ms/step - loss: 0.5279 - accuracy: 0.7436 - val_loss: 0.5344 - val_accuracy: 0.7380 - lr: 3.1250e-05
Epoch 30/50
625/625 [==============================] - 60s 96ms/step - loss: 0.5275 - accuracy: 0.7439 - val_loss: 0.5343 - val_accuracy: 0.7380 - lr: 1.5625e-05
Epoch 31/50
625/625 [==============================] - 60s 96ms/step - loss: 0.5273 - accuracy: 0.7437 - val_loss: 0.5344 - val_accuracy: 0.7378 - lr: 1.5625e-05
Epoch 32/50
625/625 [==============================] - 62s 99ms/step - loss: 0.5273 - accuracy: 0.7439 - val_loss: 0.5344 - val_accuracy: 0.7381 - lr: 7.8125e-06
Epoch 33/50
625/625 [==============================] - 62s 99ms/step - loss: 0.5270 - accuracy: 0.7446 - val_loss: 0.5346 - val_accuracy: 0.7378 - lr: 7.8125e-06
Epoch 34/50
625/625 [==============================] - 63s 100ms/step - loss: 0.5267 - accuracy: 0.7439 - val_loss: 0.5342 - val_accuracy: 0.7381 - lr: 3.9063e-06
Epoch 35/50
625/625 [==============================] - 60s 96ms/step - loss: 0.5274 - accuracy: 0.7442 - val_loss: 0.5344 - val_accuracy: 0.7380 - lr: 3.9063e-06
Epoch 36/50
625/625 [==============================] - 60s 96ms/step - loss: 0.5270 - accuracy: 0.7440 - val_loss: 0.5344 - val_accuracy: 0.7377 - lr: 3.9063e-06
Epoch 37/50
625/625 [==============================] - 62s 99ms/step - loss: 0.5269 - accuracy: 0.7444 - val_loss: 0.5343 - val_accuracy: 0.7379 - lr: 1.9531e-06
Epoch 38/50
625/625 [==============================] - 60s 96ms/step - loss: 0.5266 - accuracy: 0.7444 - val_loss: 0.5344 - val_accuracy: 0.7380 - lr: 1.9531e-06
```

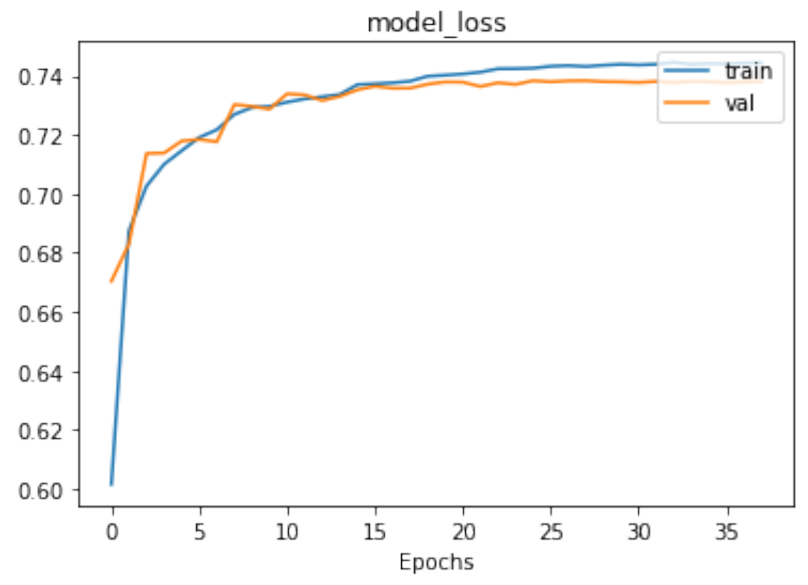
Evaluate the Model

Along with the model accuracy, the prefered metric for evaluation is ROC (Receiver Operating Characteristic) curve and the AUC score (Area under the ROC Curve).

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model_loss')
plt.xlabel('Epochs')
plt.legend(['train','val'], loc='upper right')
plt.show()
```



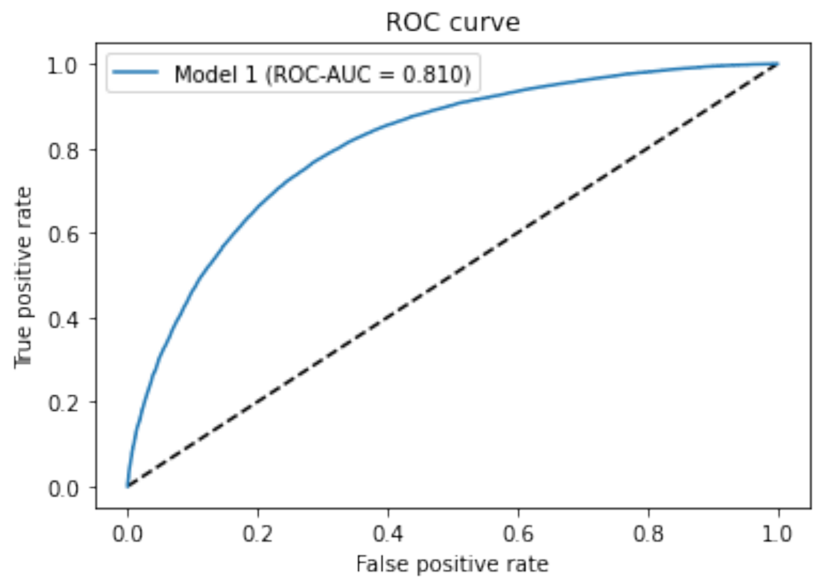
```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model_loss')
plt.xlabel('Epochs')
plt.legend(['train','val'], loc='upper right')
plt.show()
```



```
# Evaluate on validation set
score = model.evaluate(X_valid, y_valid, verbose=1)
print('\nValidation loss / accuracy: %0.4f / %0.4f'%(score[0], score[1]))
y_pred = model.predict(X_valid)
fpr, tpr, _ = roc_curve(y_valid, y_pred)
roc_auc = auc(fpr, tpr)
print('Validation ROC AUC:', roc_auc)

# Evaluate on test set
score = model.evaluate(X_test, y_test, verbose=1)
print('\nTest loss / accuracy: %0.4f / %0.4f'%(score[0], score[1]))
y_pred = model.predict(X_test)
fpr, tpr, _ = roc_curve(y_test, y_pred)
roc_auc = auc(fpr, tpr)
print('Test ROC AUC:', roc_auc)
```

```
    1600/1600 [==============================] - 13s 8ms/step - loss: 0.5342 - accuracy: 0.7381

    Validation loss / accuracy: 0.5342 / 0.7381
    Validation ROC AUC: 0.8081198852539062
    1600/1600 [==============================] - 10s 6ms/step - loss: 0.5317 - accuracy: 0.7404

    Test loss / accuracy: 0.5317 / 0.7404
    Test ROC AUC: 0.8104314468383789
```

```
plt.plot([0, 1], [0, 1], 'k--')
#plt.legend(loc=2, prop={'size': 15})
plt.plot(fpr, tpr, label='Model 1 (ROC-AUC = {:.3f})'.format(roc_auc))
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')
plt.show()
```



Submission format:

Submit the Google Colab Jupyter Notebook demonstrating your solution in the similar format as illustrated in this notebook. It should contain:

- The final model architecture, parameters and hyper-parameters yielding the best possible performance,
- Its Training and Validation accuracy,
- ROC curve and the AUC score as shown above.
- Also, please submit the final trained model containing the model architecture and its trained weights along with this notebook (For example: HDF5 file, .pb file, .pt file, etc.). Either in this notebook or in a separate notebook show how to load and use your model.

To load saved model to any madel say test_model
test_model = tf.keras.models.load_model("model.h5")

Double-click (or enter) to edit

✓ 49s completed at 12:34 PM