USRI RESEARCH REVIEW 2021

A Transformer-Based Classification System for Volcanic Seismic Signals

RESEARCH GOALS

Volcanic seismic signals are a key element in volcano monitoring to assess the state of unrest and a possible eruption style and timing. Different sources such as brittle fracture (volcano-tectonic - VT) or fluid movement (long period - LP) generate signals with distinct characteristics in frequency content and shape, but site effects such as attenuation or background noise make their determination difficult to the untrained eye. In cases of unrest or an eminent eruption, the amount of data would require a fast and reliable source of pre-classification to classify and catalogue to aid in the job usually done by a human.

To model the problem, we will develop a custom-made Transformer model. Transformers are state-of-the-art deep learning methodologies that work with sequence-based data such as audio, text or, in this case, volcanic signals. The power of transformers lies in their ability to identify complex, disconnected patterns and then use them to identify phenomena in a very effective manner. We will be building the model architecture in TensorFlow and will be running them through SHARCNET.

Unfiltered continuous data from seismic stations in Villarrica volcano will be used as train dataset and catalogued from at least these two types of events (VT and LP). The model will be then tested with a different set of stations to assess changes in the signal due to attenuation at the site. This will allow to discriminate the same event in different stations.

RESEARCH

My research project this summer can very much be summarized as a three stage project. These stages include any and all preliminary research performed prior to understand aspects of the material I would encounter later in my research, the recreation of a former research team's model and the creation of our own which will eliminate costly and time consuming pre-processing of the signals into spectrograms (our model utilizes the signal itself).

PRELIMINARY RESEARCH

This stage of research was concerned with all aspects of processing the data, calculating time series specific transforms (Continuous Wavelet Transforms, Short-Term Fourier Transforms, Smoothing through Moving Averages and Spectrograms). Additionally during this period which is not included in my research output is all the papers I was required to read and summarize as well as an overview of deep learning as to make sure we were all on the same page.

In [1]:

```
### IMPORT STATEMENTS ###

import seaborn as sns
import numpy as np
import pandas as pd
import os
```

```
import shutil
import random
import h5py
import pywt
import scipy
%matplotlib inline
import sys
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import optimizers
from tensorflow.keras.models import *
from tensorflow.keras.layers import *
from tensorflow import Tensor
import cv2
import datetime as dt
import matplotlib.pyplot as plt
import matplotlib.pylab as plb
from glob import glob
import librosa
import librosa.display
from sklearn.model_selection import *
from sklearn.metrics import roc_curve, auc, confusion_matrix
from sklearn import datasets
from sklearn.preprocessing import label_binarize
```

In [2]:

```
### DATA PROCESSING AND CLEANING ###

LP_data = h5py.File('lp.hdf5', 'r')
TC_data = h5py.File('tc.hdf5', 'r')
TR_data = h5py.File('tr.hdf5', 'r')
VT_data = h5py.File('vt.hdf5', 'r')
LP_Data = LP_data['LP']
TC_Data = TC_data['TC']
TR_Data = TR_data['TR']
VT_Data = VT_data['VT']
```

In [3]:

```
### FUNCTIONS FOR GENERATING DATAFRAME CONTAINING CONTINUOUS WAVE TRANSFORMS (CWT) AND
# REALIZED THERE WAS A BETTER WAY OF CALCULATING SPECTROGRAMS.  NEVER USED BUT USEFUL F

def convert_to_list(df):
  data_list = []
  for i in range(0, df.shape[0]):
    data_list.append(df[i,:,:].flatten())
  return data_list

def calculate_waveform(array, scales):
  coef_list = []
  freq_list = []
  for j in range(0, len(array)):
    coef, freqs = pywt.cwt(array[j], scales,'morl')
    coef_list.append(coef)
    freq_list.append(freqs)
  return coef_list, freq_list
```

In [ ]:

```
LP_Data = convert_to_list(LP_Data)
TC_Data = convert_to_list(TC_Data)
TR_Data = convert_to_list(TR_Data)
```

```
VT_Data = convert_to_list(VT_Data)

# FREQUENCY RANGES FOR VARIOUS SIGNAL TYPES
#LP LB: 15Hz UB: 163Hz
#TR LB: 28Hz UB: 163Hz
#TC LB: 0.5Hz UB: 20Hz
#VT LB: 0.5Hz UB: 20Hz

# FOR THE OTHER WAVEFORMS YOU JUST NEED TO CHANGE THE FREQUENCY RANGES TO CALCULATE THE
# AND THEN ADD TO VERTICALLY STACK THEM INTO THE DATA FRAME
LB = 162.5
UB = 16.25
dt = 0.01
frequencies = pywt.scale2frequency('morl', [28]) / dt
coef, freqs = pywt.cwt(df['signal_data'][0],np.arange(16.25,162.5),'morl')

LP_waveform_data = calculate_waveform(LP_Data, np.arange(16.25,162.5))

df = pd.DataFrame(columns = ['class', 'signal_data'])
df['signal_data'] = df['signal_data'].astype(object)
df['cwt_coefficents'] = df['signal_image'].astype(object)
df['cwt_frequencies'] = df['signal_image'].astype(object)
df['signal_image'] = df['signal_image'].astype(object)
df['spectrogram'] = df['spectrogram'].astype(object)
df['class'] = 'LP'
df['signal_data'] = LP_Data
df['cwt_coefficents'] = LP_waveform_data[0]
df['cwt_frequencies'] = LP_waveform_data[1]
df['signal_image'] = 0
df['spectrogram'] = 0

df1 = pd.DataFrame(columns = ['class', 'signal_data'])
df1['signal_data'] = df1['signal_data'].astype(object)
df1['signal_data'] = TC_Data
df1['class'] = 'TC'
df1['cwt_coefficents'] = 0
df1['cwt_frequencies'] = 0
df1['signal_image'] = 0
df1['signal_image'] = df1['signal_image'].astype(object)
df1['spectrogram'] = 0
df1['spectrogram'] = df1['spectrogram'].astype(object)

df2 = pd.DataFrame(columns = ['class', 'signal_data'])
df2['signal_data'] = df2['signal_data'].astype(object)
df2['signal_data'] = TR_Data
df2['class'] = 'TR'
df2['cwt_coefficents'] = 0
df2['cwt_frequencies'] = 0
df2['signal_image'] = 0
df2['signal_image'] = df2['signal_image'].astype(object)
df2['spectrogram'] = 0
df2['spectrogram'] = df2['spectrogram'].astype(object)

df3 = pd.DataFrame(columns = ['class', 'signal_data'])
df3['signal_data'] = df3['signal_data'].astype(object)
df3['signal_data'] = VT_Data
df3['class'] = 'VT'
df3['cwt_coefficents'] = 0
df3['cwt_frequencies'] = 0
df3['signal_image'] = 0
df3['signal_image'] = df3['signal_image'].astype(object)
```

```
df3['spectrogram'] = 0
df3['spectrogram'] = df3['spectrogram'].astype(object)


df = pd.concat([df, df1, df2, df3], ignore_index = True)
```

In [ ]:

```
### MISADVENTURE USING PACKAGE FOR SPECTROGRAMS ###
# Looked into siganalysis due to built in smoothing function

x = np.linspace(0,2*np.pi,100)
y = np.sin(x) + np.random.random(100) * 0.2
smoothed_signal = siganalysis.smooth(y.flatten(), window_len=2, window='flat')
```

In [ ]:

```
### CALCULATING SPECTROGRAMS ###

# DUE TO RUNNING OUT OF RAM (MEMORY) I HAD TO RUN EACH GROUP SEPERATELY SO JUST MATCH T
# AND RERUN

# LIST ARRAY OPTIONS: LP_Data, TC_Data, TR_Data, VT_Data
# NAMES ARRAY OPTIONS: 'LP', 'TC', 'TR', 'VT'
# FOLDER NAMES ARRAY: 'larger_lp_images/', 'larger_tc_images/', 'larger_tr_images/', 'l
#list_array = [VT_Data]
#names_array = ['VT']
#folder_names = ['larger_vt_images/']

for i in range(0, len(list_array)):
  for j in range(0, len(list_array[i])):
    rolling = pd.Series(list_array[i][j,:,:].flatten()).rolling(min_periods=1, window=1
    rolling_mean = rolling.mean()
    X = librosa.core.stft(rolling_mean.to_numpy(), n_fft=100, hop_length=5)
    Xdb = librosa.amplitude_to_db(abs(X))
    plt.figure(figsize=(1, 1), dpi=1024)
    librosa.display.specshow(Xdb, sr=100)
    plt.ylim([1, 20])
    filename = str(names_array[i]) + '_' + str(j) +'.jpg'
    plt.savefig(str(folder_names[i]) + str(filename), bbox_inches='tight')
```

In [5]:

```
### INITIAL WAY OF SPLITING SPECTROGRAMS INTO APPROPRIATE TRAINING, TESTING AND VALIDAT

# CREATING TRAINING, TESTING AND VALIDATION SETS (CAN ONLY RUN ONCE WITHOUT ERROR)
root_dir = '/content/drive/MyDrive/spectrograms'
classes_dir = ['/LP', '/TC', '/TR', '/VT']

val_ratio = 0.15
test_ratio = 0.05

for cls in classes_dir:
    os.makedirs(root_dir +'/train' + cls)
    os.makedirs(root_dir +'/val' + cls)
    os.makedirs(root_dir +'/test' + cls)


    # CREATING PARTITIONS AFTER SHUFFLING
    src = root_dir + cls # FOLDER TO COPY SPECTROGRAMS FROM

    allFileNames = os.listdir(src)
    np.random.shuffle(allFileNames)
```

```python
    train_FileNames, val_FileNames, test_FileNames = np.split(np.array(allFileNames),
                                                              [int(len(allFileNames)* (
                                                               int(len(allFileNames)* (
```

In [ ]:

```python
### WAY SPECTROGRAMS WERE ACTUALLY SPLIT INTO FOLDERS ###

splitfolders.ratio('/content/drive/MyDrive/spectrograms', output="train_test_val_datase
```

In [7]:

```python
### FUNCTION FOR CREATING DATASET ###

def create_dataset(img_folder):

    img_data_array=[]
    class_name=[]

    for dir1 in os.listdir(img_folder):
        for file in os.listdir(os.path.join(img_folder, dir1)):

            image_path= os.path.join(img_folder, dir1,  file)
            image= cv2.imread( image_path, cv2.COLOR_BGR2RGB)
            image=cv2.resize(image, (IMG_HEIGHT, IMG_WIDTH),interpolation = cv2.INTER_A
            image=np.array(image)
            image = image.astype('float32')
            image /= 255
            img_data_array.append(image)
            class_name.append(dir1)
    return img_data_array, class_name
```

In [ ]:

```python
### CREATING SPECTROGRAM DATASET ###

IMG_WIDTH=20
IMG_HEIGHT=20
img_folder=r'/content/drive/MyDrive/train_test_val_dataset/train'

# EXTRACT SPECTROGRAM ARRAY AND CLASS NAME ASSOCIATED WITH IT
img_data, class_name = create_dataset(r'/content/drive/MyDrive/train_test_val_dataset/t
img_data2, class_name2 = create_dataset(r'/content/drive/MyDrive/train_test_val_dataset

# FOR REFERENCE PURPOSES THE SPECTROGRAM TYPES ARE ENCODED INTO INTEGERS BY THE FOLLOWI
# {'LP': 0, 'TC': 1, 'TR': 2, 'VT': 3}

target_dict={k: v for v, k in enumerate(np.unique(class_name))}
target_vall=  [target_dict[class_name[i]] for i in range(len(class_name))]
target_val = tf.keras.utils.to_categorical(target_vall, num_classes=4, dtype='float32')

target_dict2={k: v for v, k in enumerate(np.unique(class_name2))}
target_vall2=  [target_dict2[class_name2[i]] for i in range(len(class_name2))]
target_val2 = tf.keras.utils.to_categorical(target_vall2, num_classes=4, dtype='float32
```

RECREATING CURRILEM'S MODEL

Using the spectrograms and continuous wavelet transforms which we didn't actually end up using we recreated Currilem's model following their paper. The model utilizes common computer vision techniques (a computer vision based classification model that classifies images based upon their features, the images being our spectrograms in this case).

In [4]:

```
### CURRILEM'S MODEL ARCHITECTURE ###

model = Sequential(name='SeismicNet')
model.add(InputLayer(input_shape = (20, 20, 3)))
model.add(Conv2D(filters=32, kernel_size=(3,3), strides=1, padding="valid", activation=
model.add(MaxPooling2D(pool_size= (2,2), strides=2))
model.add(Dropout(0.15))
model.add(Conv2D(filters=64, kernel_size=(3,3), strides=1, padding="valid", activation=
model.add(MaxPooling2D(pool_size= (2,2), strides=2))
model.add(Flatten())
model.add(Dense(700, activation="relu"))
model.add(Dropout(0.75))
model.add(Dense(4, activation = "softmax"))
model.summary()
```

```
Model: "SeismicNet"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 18, 18, 32)        896       
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 9, 9, 32)          0         
_________________________________________________________________
dropout (Dropout)            (None, 9, 9, 32)          0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 7, 7, 64)          18496     
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 3, 3, 64)          0         
_________________________________________________________________
flatten (Flatten)            (None, 576)               0         
_________________________________________________________________
dense (Dense)                (None, 700)               403900    
_________________________________________________________________
dropout_1 (Dropout)          (None, 700)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 2804      
=================================================================
Total params: 426,096
Trainable params: 426,096
Non-trainable params: 0
_________________________________________________________________
```

In [5]:

```
### DATA AUGMENTATION AND CREATING GENERATORS FOR PASSING SPECTROGRAMS INTO MODEL FOR T

train_datagen = ImageDataGenerator(
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True,
                                   vertical_flip=False,
                                   validation_split = 0.2
                                   )

test_datagen = ImageDataGenerator(
                                   shear_range=0,
                                   zoom_range=0,
                                   horizontal_flip=False,
                                   vertical_flip=False,
                                   validation_split = 0
                                   )
batch_size = 128
```

```
train_data_dir = '/content/drive/MyDrive/train_test_val_dataset/train'
(img_height, img_width) = (20, 20)

train_generator = train_datagen.flow_from_directory(
                                                    train_data_dir,
                                                    target_size=(img_height, img_width)
                                                    batch_size=batch_size,
                                                    class_mode='categorical',
                                                    subset = 'training',
                                                    shuffle = True
                                                    )

validation_generator = train_datagen.flow_from_directory(
                                                    train_data_dir,
                                                    target_size=(img_height, img_width)
                                                    batch_size=batch_size,
                                                    class_mode='categorical',
                                                    subset = 'validation',
                                                    shuffle = True
                                                    )

test_data_dir = '/content/drive/MyDrive/train_test_val_dataset/test'
test_generator = test_datagen.flow_from_directory(
                                                  test_data_dir,
                                                  target_size=(img_height, img_width),
                                                  batch_size=1,
                                                  class_mode='categorical',
                                                  shuffle = False
                                                  )
```

```
Found 2300 images belonging to 4 classes.
Found 573 images belonging to 4 classes.
Found 361 images belonging to 4 classes.
```

In [6]:

```
### FUNCTION FOR CALCULATING F1 SCORE AS THEY CALCULATE IN CURRILEM'S MODEL ###

import keras.backend as K
def F1_Score(y_true, y_pred): #taken from old keras source code
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    recall = true_positives / (possible_positives + K.epsilon())
    f1_val = 2*(precision*recall)/(precision+recall+K.epsilon())
    return f1_val
```

In [7]:

```
### TRAINING MODEL ###

optimizer = optimizers.Adam(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'
epochs = 100

model.fit(
            train_generator,
            epochs=epochs,
            validation_data=validation_generator
          )
```

```
Epoch 1/100
```

```
18/18 [==============================] - 6s 300ms/step - loss: 23.6342 - accuracy: 0.334
3 - F1_Score: 0.3305 - val_loss: 1.3543 - val_accuracy: 0.3962 - val_F1_Score: 0.0000e+0
0
Epoch 2/100
18/18 [==============================] - 5s 284ms/step - loss: 1.4782 - accuracy: 0.3696
- F1_Score: 0.1169 - val_loss: 1.3588 - val_accuracy: 0.4154 - val_F1_Score: 0.0000e+00
Epoch 3/100
18/18 [==============================] - 5s 285ms/step - loss: 1.3180 - accuracy: 0.3978
- F1_Score: 0.0779 - val_loss: 1.3039 - val_accuracy: 0.4154 - val_F1_Score: 0.0000e+00
Epoch 4/100
18/18 [==============================] - 5s 283ms/step - loss: 1.2708 - accuracy: 0.4135
- F1_Score: 0.1650 - val_loss: 1.2652 - val_accuracy: 0.4154 - val_F1_Score: 0.0000e+00
Epoch 5/100
18/18 [==============================] - 5s 281ms/step - loss: 1.2193 - accuracy: 0.4348
- F1_Score: 0.2534 - val_loss: 1.2249 - val_accuracy: 0.4712 - val_F1_Score: 0.0396
Epoch 6/100
18/18 [==============================] - 5s 284ms/step - loss: 1.1580 - accuracy: 0.5009
- F1_Score: 0.3138 - val_loss: 1.0762 - val_accuracy: 0.6283 - val_F1_Score: 0.3272
Epoch 7/100
18/18 [==============================] - 5s 278ms/step - loss: 1.0330 - accuracy: 0.5835
- F1_Score: 0.4661 - val_loss: 1.0659 - val_accuracy: 0.5707 - val_F1_Score: 0.2956
Epoch 8/100
18/18 [==============================] - 5s 285ms/step - loss: 0.9472 - accuracy: 0.6126
- F1_Score: 0.5111 - val_loss: 0.8090 - val_accuracy: 0.7103 - val_F1_Score: 0.5576
Epoch 9/100
18/18 [==============================] - 5s 280ms/step - loss: 0.8461 - accuracy: 0.6691
- F1_Score: 0.6343 - val_loss: 0.5775 - val_accuracy: 0.7888 - val_F1_Score: 0.7951
Epoch 10/100
18/18 [==============================] - 5s 283ms/step - loss: 0.6781 - accuracy: 0.7426
- F1_Score: 0.7466 - val_loss: 0.4611 - val_accuracy: 0.8325 - val_F1_Score: 0.8329
Epoch 11/100
18/18 [==============================] - 5s 284ms/step - loss: 0.5638 - accuracy: 0.8030
- F1_Score: 0.8059 - val_loss: 0.3481 - val_accuracy: 0.8517 - val_F1_Score: 0.8725
Epoch 12/100
18/18 [==============================] - 5s 284ms/step - loss: 0.4784 - accuracy: 0.8322
- F1_Score: 0.8344 - val_loss: 0.2977 - val_accuracy: 0.8813 - val_F1_Score: 0.8905
Epoch 13/100
18/18 [==============================] - 5s 282ms/step - loss: 0.4287 - accuracy: 0.8422
- F1_Score: 0.8480 - val_loss: 0.2809 - val_accuracy: 0.8970 - val_F1_Score: 0.9028
Epoch 14/100
18/18 [==============================] - 5s 284ms/step - loss: 0.3605 - accuracy: 0.8665
- F1_Score: 0.8710 - val_loss: 0.2808 - val_accuracy: 0.9005 - val_F1_Score: 0.9022
Epoch 15/100
18/18 [==============================] - 5s 284ms/step - loss: 0.3814 - accuracy: 0.8748
- F1_Score: 0.8766 - val_loss: 0.2747 - val_accuracy: 0.9005 - val_F1_Score: 0.9088
Epoch 16/100
18/18 [==============================] - 5s 284ms/step - loss: 0.3576 - accuracy: 0.8804
- F1_Score: 0.8826 - val_loss: 0.2786 - val_accuracy: 0.8918 - val_F1_Score: 0.9061
Epoch 17/100
18/18 [==============================] - 5s 279ms/step - loss: 0.3440 - accuracy: 0.8848
- F1_Score: 0.8840 - val_loss: 0.2452 - val_accuracy: 0.9075 - val_F1_Score: 0.9099
Epoch 18/100
18/18 [==============================] - 5s 287ms/step - loss: 0.3231 - accuracy: 0.9017
- F1_Score: 0.9000 - val_loss: 0.2721 - val_accuracy: 0.9005 - val_F1_Score: 0.9027
Epoch 19/100
18/18 [==============================] - 5s 285ms/step - loss: 0.3288 - accuracy: 0.9017
- F1_Score: 0.9011 - val_loss: 0.2433 - val_accuracy: 0.9162 - val_F1_Score: 0.9191
Epoch 20/100
18/18 [==============================] - 5s 284ms/step - loss: 0.3220 - accuracy: 0.8935
- F1_Score: 0.8926 - val_loss: 0.2244 - val_accuracy: 0.9145 - val_F1_Score: 0.9211
Epoch 21/100
18/18 [==============================] - 5s 283ms/step - loss: 0.3146 - accuracy: 0.9070
- F1_Score: 0.9048 - val_loss: 0.2211 - val_accuracy: 0.9267 - val_F1_Score: 0.9244
Epoch 22/100
18/18 [==============================] - 5s 284ms/step - loss: 0.3256 - accuracy: 0.8970
```

```
- F1_Score: 0.8967 - val_loss: 0.2127 - val_accuracy: 0.9284 - val_F1_Score: 0.9228
Epoch 23/100
18/18 [==============================] - 5s 282ms/step - loss: 0.3025 - accuracy: 0.9030
- F1_Score: 0.9048 - val_loss: 0.2194 - val_accuracy: 0.9354 - val_F1_Score: 0.9339
Epoch 24/100
18/18 [==============================] - 5s 286ms/step - loss: 0.3181 - accuracy: 0.8935
- F1_Score: 0.8915 - val_loss: 0.2281 - val_accuracy: 0.9197 - val_F1_Score: 0.9159
Epoch 25/100
18/18 [==============================] - 5s 285ms/step - loss: 0.3050 - accuracy: 0.9104
- F1_Score: 0.9091 - val_loss: 0.2140 - val_accuracy: 0.9267 - val_F1_Score: 0.9262
Epoch 26/100
18/18 [==============================] - 5s 282ms/step - loss: 0.2995 - accuracy: 0.9035
- F1_Score: 0.9042 - val_loss: 0.2306 - val_accuracy: 0.9319 - val_F1_Score: 0.9312
Epoch 27/100
18/18 [==============================] - 5s 282ms/step - loss: 0.3060 - accuracy: 0.9043
- F1_Score: 0.9055 - val_loss: 0.2096 - val_accuracy: 0.9319 - val_F1_Score: 0.9308
Epoch 28/100
18/18 [==============================] - 5s 280ms/step - loss: 0.2859 - accuracy: 0.9109
- F1_Score: 0.9100 - val_loss: 0.1943 - val_accuracy: 0.9337 - val_F1_Score: 0.9311
Epoch 29/100
18/18 [==============================] - 5s 282ms/step - loss: 0.2731 - accuracy: 0.9122
- F1_Score: 0.9114 - val_loss: 0.2086 - val_accuracy: 0.9424 - val_F1_Score: 0.9395
Epoch 30/100
18/18 [==============================] - 5s 285ms/step - loss: 0.2778 - accuracy: 0.9183
- F1_Score: 0.9170 - val_loss: 0.2272 - val_accuracy: 0.9232 - val_F1_Score: 0.9231
Epoch 31/100
18/18 [==============================] - 5s 283ms/step - loss: 0.2986 - accuracy: 0.9104
- F1_Score: 0.9064 - val_loss: 0.2003 - val_accuracy: 0.9389 - val_F1_Score: 0.9306
Epoch 32/100
18/18 [==============================] - 5s 285ms/step - loss: 0.2673 - accuracy: 0.9191
- F1_Score: 0.9190 - val_loss: 0.1958 - val_accuracy: 0.9284 - val_F1_Score: 0.9311
Epoch 33/100
18/18 [==============================] - 5s 283ms/step - loss: 0.2809 - accuracy: 0.9178
- F1_Score: 0.9177 - val_loss: 0.1984 - val_accuracy: 0.9407 - val_F1_Score: 0.9382
Epoch 34/100
18/18 [==============================] - 5s 291ms/step - loss: 0.2702 - accuracy: 0.9222
- F1_Score: 0.9207 - val_loss: 0.1854 - val_accuracy: 0.9407 - val_F1_Score: 0.9449
Epoch 35/100
18/18 [==============================] - 5s 280ms/step - loss: 0.2630 - accuracy: 0.9239
- F1_Score: 0.9209 - val_loss: 0.1790 - val_accuracy: 0.9407 - val_F1_Score: 0.9414
Epoch 36/100
18/18 [==============================] - 5s 305ms/step - loss: 0.2653 - accuracy: 0.9261
- F1_Score: 0.9240 - val_loss: 0.1925 - val_accuracy: 0.9424 - val_F1_Score: 0.9368
Epoch 37/100
18/18 [==============================] - 5s 283ms/step - loss: 0.2693 - accuracy: 0.9283
- F1_Score: 0.9264 - val_loss: 0.2058 - val_accuracy: 0.9319 - val_F1_Score: 0.9329
Epoch 38/100
18/18 [==============================] - 5s 284ms/step - loss: 0.2473 - accuracy: 0.9278
- F1_Score: 0.9280 - val_loss: 0.1800 - val_accuracy: 0.9546 - val_F1_Score: 0.9559
Epoch 39/100
18/18 [==============================] - 5s 281ms/step - loss: 0.2493 - accuracy: 0.9287
- F1_Score: 0.9268 - val_loss: 0.1941 - val_accuracy: 0.9459 - val_F1_Score: 0.9408
Epoch 40/100
18/18 [==============================] - 5s 278ms/step - loss: 0.2745 - accuracy: 0.9239
- F1_Score: 0.9218 - val_loss: 0.1798 - val_accuracy: 0.9354 - val_F1_Score: 0.9379
Epoch 41/100
18/18 [==============================] - 5s 280ms/step - loss: 0.2534 - accuracy: 0.9291
- F1_Score: 0.9260 - val_loss: 0.1828 - val_accuracy: 0.9494 - val_F1_Score: 0.9467
Epoch 42/100
18/18 [==============================] - 5s 284ms/step - loss: 0.2344 - accuracy: 0.9270
- F1_Score: 0.9267 - val_loss: 0.1749 - val_accuracy: 0.9511 - val_F1_Score: 0.9482
Epoch 43/100
18/18 [==============================] - 5s 286ms/step - loss: 0.2548 - accuracy: 0.9283
- F1_Score: 0.9265 - val_loss: 0.1736 - val_accuracy: 0.9459 - val_F1_Score: 0.9442
Epoch 44/100
```

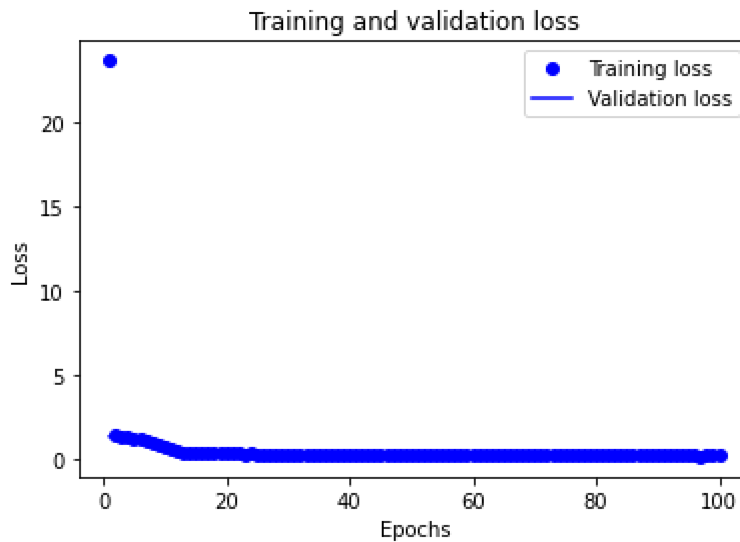
```
18/18 [==============================] - 5s 294ms/step - loss: 0.2444 - accuracy: 0.9291
- F1_Score: 0.9261 - val_loss: 0.1549 - val_accuracy: 0.9546 - val_F1_Score: 0.9546
Epoch 45/100
18/18 [==============================] - 5s 283ms/step - loss: 0.2350 - accuracy: 0.9296
- F1_Score: 0.9294 - val_loss: 0.1894 - val_accuracy: 0.9354 - val_F1_Score: 0.9330
Epoch 46/100
18/18 [==============================] - 5s 289ms/step - loss: 0.2541 - accuracy: 0.9196
- F1_Score: 0.9187 - val_loss: 0.1740 - val_accuracy: 0.9459 - val_F1_Score: 0.9429
Epoch 47/100
18/18 [==============================] - 5s 283ms/step - loss: 0.2333 - accuracy: 0.9339
- F1_Score: 0.9329 - val_loss: 0.1952 - val_accuracy: 0.9442 - val_F1_Score: 0.9455
Epoch 48/100
18/18 [==============================] - 5s 281ms/step - loss: 0.2478 - accuracy: 0.9248
- F1_Score: 0.9242 - val_loss: 0.2485 - val_accuracy: 0.9145 - val_F1_Score: 0.9151
Epoch 49/100
18/18 [==============================] - 5s 283ms/step - loss: 0.2653 - accuracy: 0.9291
- F1_Score: 0.9285 - val_loss: 0.1865 - val_accuracy: 0.9442 - val_F1_Score: 0.9421
Epoch 50/100
18/18 [==============================] - 5s 284ms/step - loss: 0.2363 - accuracy: 0.9339
- F1_Score: 0.9318 - val_loss: 0.1609 - val_accuracy: 0.9564 - val_F1_Score: 0.9541
Epoch 51/100
18/18 [==============================] - 5s 288ms/step - loss: 0.2400 - accuracy: 0.9343
- F1_Score: 0.9316 - val_loss: 0.1886 - val_accuracy: 0.9407 - val_F1_Score: 0.9438
Epoch 52/100
18/18 [==============================] - 5s 284ms/step - loss: 0.2515 - accuracy: 0.9265
- F1_Score: 0.9265 - val_loss: 0.1648 - val_accuracy: 0.9494 - val_F1_Score: 0.9551
Epoch 53/100
18/18 [==============================] - 5s 285ms/step - loss: 0.2356 - accuracy: 0.9330
- F1_Score: 0.9315 - val_loss: 0.1711 - val_accuracy: 0.9424 - val_F1_Score: 0.9380
Epoch 54/100
18/18 [==============================] - 5s 280ms/step - loss: 0.2211 - accuracy: 0.9300
- F1_Score: 0.9281 - val_loss: 0.1580 - val_accuracy: 0.9494 - val_F1_Score: 0.9509
Epoch 55/100
18/18 [==============================] - 5s 281ms/step - loss: 0.2321 - accuracy: 0.9374
- F1_Score: 0.9357 - val_loss: 0.1579 - val_accuracy: 0.9476 - val_F1_Score: 0.9402
Epoch 56/100
18/18 [==============================] - 5s 284ms/step - loss: 0.2237 - accuracy: 0.9365
- F1_Score: 0.9380 - val_loss: 0.1887 - val_accuracy: 0.9407 - val_F1_Score: 0.9284
Epoch 57/100
18/18 [==============================] - 5s 285ms/step - loss: 0.2362 - accuracy: 0.9343
- F1_Score: 0.9338 - val_loss: 0.1717 - val_accuracy: 0.9476 - val_F1_Score: 0.9487
Epoch 58/100
18/18 [==============================] - 5s 286ms/step - loss: 0.2248 - accuracy: 0.9383
- F1_Score: 0.9375 - val_loss: 0.1653 - val_accuracy: 0.9424 - val_F1_Score: 0.9457
Epoch 59/100
18/18 [==============================] - 5s 288ms/step - loss: 0.2419 - accuracy: 0.9309
- F1_Score: 0.9301 - val_loss: 0.1962 - val_accuracy: 0.9442 - val_F1_Score: 0.9380
Epoch 60/100
18/18 [==============================] - 5s 288ms/step - loss: 0.2271 - accuracy: 0.9352
- F1_Score: 0.9336 - val_loss: 0.1791 - val_accuracy: 0.9389 - val_F1_Score: 0.9447
Epoch 61/100
18/18 [==============================] - 5s 287ms/step - loss: 0.2453 - accuracy: 0.9313
- F1_Score: 0.9311 - val_loss: 0.1533 - val_accuracy: 0.9616 - val_F1_Score: 0.9678
Epoch 62/100
18/18 [==============================] - 5s 280ms/step - loss: 0.2257 - accuracy: 0.9339
- F1_Score: 0.9358 - val_loss: 0.1596 - val_accuracy: 0.9529 - val_F1_Score: 0.9508
Epoch 63/100
18/18 [==============================] - 5s 283ms/step - loss: 0.2243 - accuracy: 0.9313
- F1_Score: 0.9320 - val_loss: 0.1668 - val_accuracy: 0.9476 - val_F1_Score: 0.9444
Epoch 64/100
18/18 [==============================] - 5s 282ms/step - loss: 0.2289 - accuracy: 0.9291
- F1_Score: 0.9316 - val_loss: 0.1725 - val_accuracy: 0.9476 - val_F1_Score: 0.9459
Epoch 65/100
18/18 [==============================] - 5s 286ms/step - loss: 0.2238 - accuracy: 0.9396
- F1_Score: 0.9384 - val_loss: 0.1567 - val_accuracy: 0.9511 - val_F1_Score: 0.9419
```

```
Epoch 66/100
18/18 [==============================] - 5s 282ms/step - loss: 0.2251 - accuracy: 0.9322
- F1_Score: 0.9332 - val_loss: 0.1628 - val_accuracy: 0.9511 - val_F1_Score: 0.9538
Epoch 67/100
18/18 [==============================] - 5s 286ms/step - loss: 0.2257 - accuracy: 0.9352
- F1_Score: 0.9338 - val_loss: 0.1600 - val_accuracy: 0.9564 - val_F1_Score: 0.9546
Epoch 68/100
18/18 [==============================] - 5s 286ms/step - loss: 0.2340 - accuracy: 0.9322
- F1_Score: 0.9302 - val_loss: 0.1748 - val_accuracy: 0.9511 - val_F1_Score: 0.9536
Epoch 69/100
18/18 [==============================] - 5s 281ms/step - loss: 0.2251 - accuracy: 0.9300
- F1_Score: 0.9293 - val_loss: 0.1521 - val_accuracy: 0.9564 - val_F1_Score: 0.9595
Epoch 70/100
18/18 [==============================] - 5s 283ms/step - loss: 0.2127 - accuracy: 0.9378
- F1_Score: 0.9357 - val_loss: 0.1642 - val_accuracy: 0.9476 - val_F1_Score: 0.9461
Epoch 71/100
18/18 [==============================] - 5s 282ms/step - loss: 0.2178 - accuracy: 0.9387
- F1_Score: 0.9383 - val_loss: 0.1682 - val_accuracy: 0.9424 - val_F1_Score: 0.9314
Epoch 72/100
18/18 [==============================] - 5s 280ms/step - loss: 0.2309 - accuracy: 0.9352
- F1_Score: 0.9334 - val_loss: 0.1726 - val_accuracy: 0.9494 - val_F1_Score: 0.9558
Epoch 73/100
18/18 [==============================] - 5s 282ms/step - loss: 0.2160 - accuracy: 0.9374
- F1_Score: 0.9368 - val_loss: 0.1619 - val_accuracy: 0.9494 - val_F1_Score: 0.9472
Epoch 74/100
18/18 [==============================] - 5s 283ms/step - loss: 0.2071 - accuracy: 0.9409
- F1_Score: 0.9412 - val_loss: 0.1712 - val_accuracy: 0.9511 - val_F1_Score: 0.9490
Epoch 75/100
18/18 [==============================] - 5s 289ms/step - loss: 0.2409 - accuracy: 0.9335
- F1_Score: 0.9335 - val_loss: 0.1608 - val_accuracy: 0.9494 - val_F1_Score: 0.9499
Epoch 76/100
18/18 [==============================] - 5s 283ms/step - loss: 0.2351 - accuracy: 0.9352
- F1_Score: 0.9329 - val_loss: 0.1617 - val_accuracy: 0.9529 - val_F1_Score: 0.9523
Epoch 77/100
18/18 [==============================] - 5s 286ms/step - loss: 0.2167 - accuracy: 0.9383
- F1_Score: 0.9375 - val_loss: 0.1496 - val_accuracy: 0.9511 - val_F1_Score: 0.9454
Epoch 78/100
18/18 [==============================] - 5s 282ms/step - loss: 0.2155 - accuracy: 0.9330
- F1_Score: 0.9349 - val_loss: 0.1691 - val_accuracy: 0.9546 - val_F1_Score: 0.9517
Epoch 79/100
18/18 [==============================] - 5s 280ms/step - loss: 0.2021 - accuracy: 0.9404
- F1_Score: 0.9388 - val_loss: 0.1652 - val_accuracy: 0.9529 - val_F1_Score: 0.9527
Epoch 80/100
18/18 [==============================] - 5s 285ms/step - loss: 0.2205 - accuracy: 0.9378
- F1_Score: 0.9384 - val_loss: 0.1530 - val_accuracy: 0.9529 - val_F1_Score: 0.9509
Epoch 81/100
18/18 [==============================] - 5s 283ms/step - loss: 0.2041 - accuracy: 0.9413
- F1_Score: 0.9417 - val_loss: 0.1738 - val_accuracy: 0.9546 - val_F1_Score: 0.9556
Epoch 82/100
18/18 [==============================] - 5s 284ms/step - loss: 0.2285 - accuracy: 0.9322
- F1_Score: 0.9329 - val_loss: 0.1745 - val_accuracy: 0.9546 - val_F1_Score: 0.9566
Epoch 83/100
18/18 [==============================] - 5s 280ms/step - loss: 0.2182 - accuracy: 0.9361
- F1_Score: 0.9357 - val_loss: 0.1755 - val_accuracy: 0.9546 - val_F1_Score: 0.9525
Epoch 84/100
18/18 [==============================] - 5s 281ms/step - loss: 0.2423 - accuracy: 0.9296
- F1_Score: 0.9314 - val_loss: 0.2059 - val_accuracy: 0.9319 - val_F1_Score: 0.9361
Epoch 85/100
18/18 [==============================] - 5s 283ms/step - loss: 0.2126 - accuracy: 0.9422
- F1_Score: 0.9406 - val_loss: 0.2032 - val_accuracy: 0.9354 - val_F1_Score: 0.9349
Epoch 86/100
18/18 [==============================] - 5s 285ms/step - loss: 0.2172 - accuracy: 0.9383
- F1_Score: 0.9376 - val_loss: 0.1798 - val_accuracy: 0.9459 - val_F1_Score: 0.9452
Epoch 87/100
18/18 [==============================] - 5s 287ms/step - loss: 0.2145 - accuracy: 0.9417
```

```
- F1_Score: 0.9428 - val_loss: 0.1542 - val_accuracy: 0.9529 - val_F1_Score: 0.9465
Epoch 88/100
18/18 [==============================] - 5s 277ms/step - loss: 0.2343 - accuracy: 0.9387
- F1_Score: 0.9396 - val_loss: 0.1856 - val_accuracy: 0.9372 - val_F1_Score: 0.9348
Epoch 89/100
18/18 [==============================] - 5s 289ms/step - loss: 0.2166 - accuracy: 0.9330
- F1_Score: 0.9332 - val_loss: 0.1431 - val_accuracy: 0.9546 - val_F1_Score: 0.9522
Epoch 90/100
18/18 [==============================] - 5s 289ms/step - loss: 0.2245 - accuracy: 0.9409
- F1_Score: 0.9406 - val_loss: 0.1517 - val_accuracy: 0.9476 - val_F1_Score: 0.9525
Epoch 91/100
18/18 [==============================] - 5s 283ms/step - loss: 0.2166 - accuracy: 0.9383
- F1_Score: 0.9395 - val_loss: 0.1620 - val_accuracy: 0.9459 - val_F1_Score: 0.9483
Epoch 92/100
18/18 [==============================] - 5s 286ms/step - loss: 0.2360 - accuracy: 0.9352
- F1_Score: 0.9338 - val_loss: 0.1680 - val_accuracy: 0.9459 - val_F1_Score: 0.9513
Epoch 93/100
18/18 [==============================] - 5s 288ms/step - loss: 0.1945 - accuracy: 0.9439
- F1_Score: 0.9427 - val_loss: 0.1677 - val_accuracy: 0.9389 - val_F1_Score: 0.9437
Epoch 94/100
18/18 [==============================] - 5s 283ms/step - loss: 0.2166 - accuracy: 0.9357
- F1_Score: 0.9357 - val_loss: 0.1798 - val_accuracy: 0.9424 - val_F1_Score: 0.9388
Epoch 95/100
18/18 [==============================] - 5s 287ms/step - loss: 0.2030 - accuracy: 0.9365
- F1_Score: 0.9353 - val_loss: 0.1532 - val_accuracy: 0.9581 - val_F1_Score: 0.9580
Epoch 96/100
18/18 [==============================] - 5s 284ms/step - loss: 0.2037 - accuracy: 0.9430
- F1_Score: 0.9423 - val_loss: 0.1858 - val_accuracy: 0.9459 - val_F1_Score: 0.9417
Epoch 97/100
18/18 [==============================] - 5s 284ms/step - loss: 0.1832 - accuracy: 0.9457
- F1_Score: 0.9450 - val_loss: 0.1657 - val_accuracy: 0.9546 - val_F1_Score: 0.9532
Epoch 98/100
18/18 [==============================] - 5s 285ms/step - loss: 0.2039 - accuracy: 0.9335
- F1_Score: 0.9335 - val_loss: 0.1527 - val_accuracy: 0.9459 - val_F1_Score: 0.9503
Epoch 99/100
18/18 [==============================] - 5s 284ms/step - loss: 0.2057 - accuracy: 0.9439
- F1_Score: 0.9419 - val_loss: 0.1736 - val_accuracy: 0.9494 - val_F1_Score: 0.9536
Epoch 100/100
18/18 [==============================] - 5s 282ms/step - loss: 0.2192 - accuracy: 0.9348
- F1_Score: 0.9366 - val_loss: 0.1461 - val_accuracy: 0.9511 - val_F1_Score: 0.9507
```

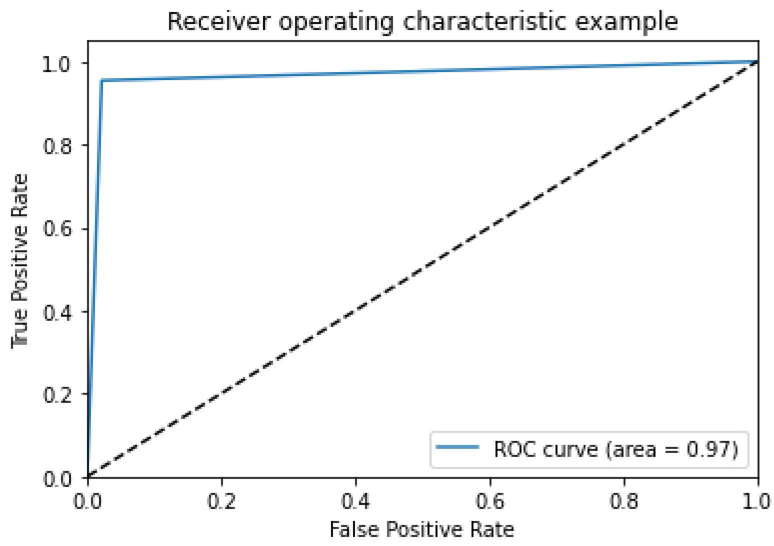
Out[7]:
```
<keras.callbacks.History at 0x7fa00c72fc90>
```

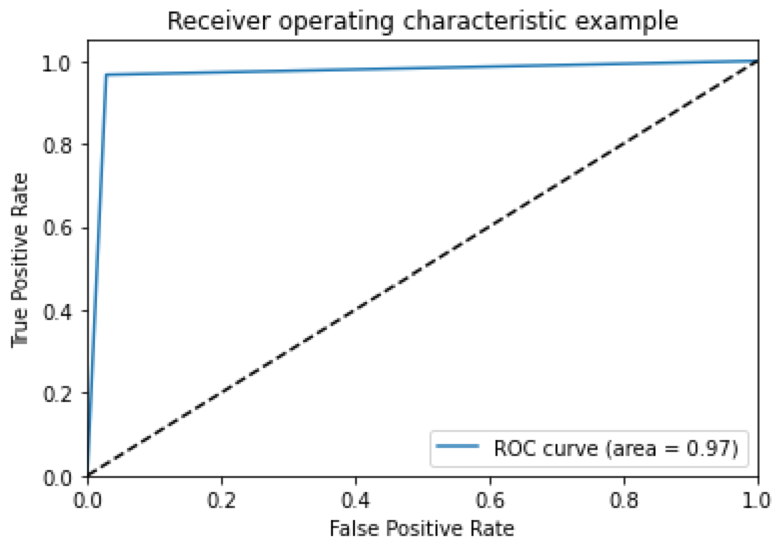
In [8]:
```
### PLOTTING TRAINING PROGRESS ###

loss = model.history.history['loss']
val_loss = model.history.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

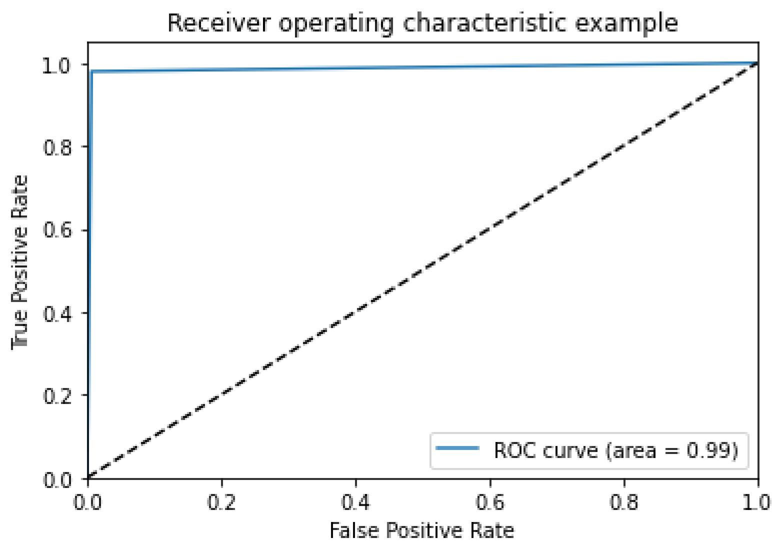


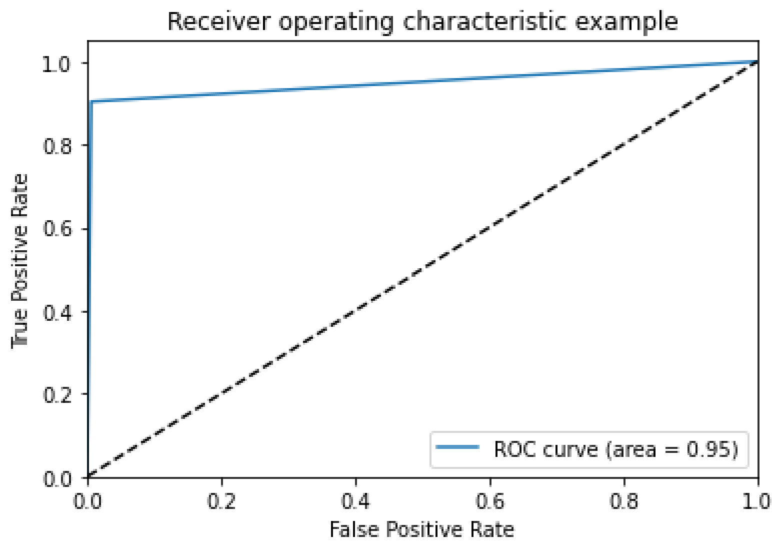
In [9]:

```
### PREDICTING USING MODEL AND GENERATING SUBSEQUENT ROC CURVES AND METRICS ###

test_generator.reset()

# CALCULATING CLASS PROBABILITIES
y_pred = model.predict(test_generator)
pred_classes = np.argmax(y_pred, axis=1)
labels = test_generator.classes

y_test = label_binarize(labels, classes=[0,1,2,3])
pred_classes = label_binarize(np.array(pred_classes), classes=[0,1,2,3])
n_classes = 4

# COMPUTE ROC CURVE AND AUROC
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], pred_classes[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# PLOT ROC CURVE FOR EACH CLASS IF WAVEFORMS
for i in range(n_classes):
    plt.figure()
    plt.plot(fpr[i], tpr[i], label='ROC curve (area = %0.2f)' % roc_auc[i])
    plt.plot([0, 1], [0, 1], 'k--')
    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 example')
    plt.legend(loc="lower right")
    plt.show()
```


Receiver operating characteristic example
True Positive Rate
False Positive Rate
ROC curve (area = 0.97)
0.0
0.2
0.4
0.6
0.8
1.0


Receiver operating characteristic example
True Positive Rate
False Positive Rate
ROC curve (area = 0.97)
0.0
0.2
0.4
0.6
0.8
1.0


Receiver operating characteristic example
True Positive Rate
False Positive Rate
ROC curve (area = 0.99)
0.0
0.2
0.4
0.6
0.8
1.0



In [15]:

```
### SAVING MODEL AND WEIGHTS ###

model.save('/content/drive/MyDrive/VolcanicSignalClassifier.h5')
```

```
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be bu
ilt. `model.compile_metrics` will be empty until you train or evaluate the model.
/usr/local/lib/python3.7/dist-packages/keras/utils/generic_utils.py:497: CustomMaskWarni
ng: Custom mask layers require a config and must override get_config. When loading, the
custom mask layer must be passed to the custom_objects argument.
  category=CustomMaskWarning)
```

GENERATING OUR OWN MODEL

Time for our own research! Again one of the main purposes behind our research was to cut out the lengthy and tedious pre-processing associated with the calculation of both the spectrograms as well as the Continuous Wavelet Transforms so you will notice that our code is far more concise in nature. *** Note that our code is not quite complete and will still require testing and parameter tuning as well as exploration of how it performs on other datasets (a step above Currilem's model which served more as a proof of concept as to whether these signals could be classified to any meaningful degree)

In [10]:

```
### CREATING AN ARRAY OF ARRAYS CONTAINING THE WAVEFORMS THEMSELVES ###

n_LPs = LP_Data.shape[0]
n_TCs = TC_Data.shape[0]
n_TRs = TR_Data.shape[0]
n_VTs = VT_Data.shape[0]
n = n_LPs + n_TCs + n_TRs + n_VTs

labels = np.concatenate((['LP']*n_LPs, ['TC']*n_TCs))
labels = np.concatenate((labels, ['TC']*n_TRs))
labels = np.concatenate((labels, ['TC']*n_VTs))
array = np.vstack((LP_Data, TC_Data))
array = np.vstack((array, TR_Data))
array = np.vstack((array, VT_Data))
```

In [11]:

```
### GENERATING AND VISUALIZING DATAFRAME OF WAVEFORMS AND THEIR LABELS ###
```

```python
df = pd.DataFrame(columns = ['class', 'signal_data'])
df['class'] = labels
#df['signal_data'] = array

for i in range(0, n):
  df['signal_data'][i] = array[i,:,:]
df.head()
```

Out[11]:

	class	signal_data
0	LP	[[1.5479434094570514], [1.5497955558905743], [...
1	LP	[[1.4560196685301103], [1.4525966232876937], [...
2	LP	[[1.5116154410699076], [1.5140446912500658], [...
3	LP	[[1.5039532305021819], [1.5040567013200363], [...
4	LP	[[1.5493120999506453], [1.55696367853308], [1....

In [12]:

```python
### CODE FOR GENERATING RNN WHICH WILL BE INCLUDED IN OUR MODEL ###

def relu_bn(inputs: Tensor) -> Tensor:
    relu = ReLU()(inputs)
    bn = BatchNormalization()(relu)
    return bn

def residual_block(x: Tensor, downsample: bool, filters: int=32, kernel_size: int=8) ->
    y = Conv1D(kernel_size=kernel_size,
               strides= (1 if not downsample else 2),
               filters=filters,
               padding="same")(x)
    y = relu_bn(y)
    y = Conv1D(kernel_size=kernel_size,
               strides=1,
               filters=filters,
               padding="same")(y)

    if downsample:
        x = Conv1D(kernel_size=1,
                   strides=2,
                   filters=filters,
                   padding="same")(x)
    out = Add()([x, y])
    out = relu_bn(out)
    out = SpatialDropout1D(rate=0.25)(out)
    return out

def block_layer(t: Tensor):
  num_filters = 32
  num_blocks_list = [1, 1, 1, 1]
  for i in range(len(num_blocks_list)):
    num_blocks = num_blocks_list[i]
    for j in range(num_blocks):
      t = residual_block(t, downsample=(j==0 and i!=0), filters=num_filters)
    num_filters *= 2
  t = Conv1D(filters=num_filters, kernel_size=8, activation='relu')(t)
  t = MaxPool1D(strides=2)(t)
  return t
```

In [13]:

```
### DETERMINING INPUT DIMENSION SO THAT CODE WILL WORK FOR INPUT WITH DIFFERENT LENGTH

input_dim = 0
for i in range(0, n):
  if (df['signal_data'][i].shape[0] > input_dim):
    input_dim = df['signal_data'][i].shape[0]
```

In [14]:

```
### OUR MODEL ARCHITECTURE ###

input = Input(shape=(None, input_dim))
model = Conv1D(filters=32, kernel_size=8, activation='relu')(input)
model = block_layer(model)
model = LSTM(8, input_shape=(8, 256), return_sequences=False)(model)
#model = MultiHeadAttention(num_heads=2, key_dim=2)(model)
model = Dense(4, activation = "softmax")(model)
model = Model(inputs=input, outputs=model)
model.summary()
```

```
Model: "model"
____________________________________________________________________________________________________
__________
Layer (type)                    Output Shape         Param #     Connected to
====================================================================================================
==========
input_2 (InputLayer)            [(None, None, 6000)] 0

____________________________________________________________________________________________________
__________
conv1d (Conv1D)                 (None, None, 32)     1536032     input_2[0][0]
____________________________________________________________________________________________________
__________
conv1d_1 (Conv1D)               (None, None, 32)     8224        conv1d[0][0]
____________________________________________________________________________________________________
__________
re_lu (ReLU)                    (None, None, 32)     0           conv1d_1[0][0]
____________________________________________________________________________________________________
__________
batch_normalization (BatchNorma (None, None, 32)     128         re_lu[0][0]
____________________________________________________________________________________________________
__________
conv1d_2 (Conv1D)               (None, None, 32)     8224        batch_normalization[0]
[0]
____________________________________________________________________________________________________
__________
add (Add)                       (None, None, 32)     0           conv1d[0][0]
                                                                 conv1d_2[0][0]
____________________________________________________________________________________________________
__________
re_lu_1 (ReLU)                  (None, None, 32)     0           add[0][0]
____________________________________________________________________________________________________
__________
batch_normalization_1 (BatchNor (None, None, 32)     128         re_lu_1[0][0]
____________________________________________________________________________________________________
__________
spatial_dropout1d (SpatialDropo (None, None, 32)     0           batch_normalization_1
[0][0]
____________________________________________________________________________________________________
__________
conv1d_3 (Conv1D)               (None, None, 64)     16448       spatial_dropout1d[0][0]
____________________________________________________________________________________________________
__________
```

```
re_lu_2 (ReLU)                  (None, None, 64)     0           conv1d_3[0][0]
__________________________________________________________________________________________________
batch_normalization_2 (BatchNor (None, None, 64)     256         re_lu_2[0][0]
__________________________________________________________________________________________________
conv1d_5 (Conv1D)               (None, None, 64)     2112        spatial_dropout1d[0][0]
__________________________________________________________________________________________________
conv1d_4 (Conv1D)               (None, None, 64)     32832       batch_normalization_2
[0][0]
__________________________________________________________________________________________________
add_1 (Add)                     (None, None, 64)     0           conv1d_5[0][0]
                                                                 conv1d_4[0][0]
__________________________________________________________________________________________________
re_lu_3 (ReLU)                  (None, None, 64)     0           add_1[0][0]
__________________________________________________________________________________________________
batch_normalization_3 (BatchNor (None, None, 64)     256         re_lu_3[0][0]
__________________________________________________________________________________________________
spatial_dropout1d_1 (SpatialDro (None, None, 64)     0           batch_normalization_3
[0][0]
__________________________________________________________________________________________________
conv1d_6 (Conv1D)               (None, None, 128)    65664       spatial_dropout1d_1[0]
[0]
__________________________________________________________________________________________________
re_lu_4 (ReLU)                  (None, None, 128)    0           conv1d_6[0][0]
__________________________________________________________________________________________________
batch_normalization_4 (BatchNor (None, None, 128)    512         re_lu_4[0][0]
__________________________________________________________________________________________________
conv1d_8 (Conv1D)               (None, None, 128)    8320        spatial_dropout1d_1[0]
[0]
__________________________________________________________________________________________________
conv1d_7 (Conv1D)               (None, None, 128)    131200      batch_normalization_4
[0][0]
__________________________________________________________________________________________________
add_2 (Add)                     (None, None, 128)    0           conv1d_8[0][0]
                                                                 conv1d_7[0][0]
__________________________________________________________________________________________________
re_lu_5 (ReLU)                  (None, None, 128)    0           add_2[0][0]
__________________________________________________________________________________________________
batch_normalization_5 (BatchNor (None, None, 128)    512         re_lu_5[0][0]
__________________________________________________________________________________________________
spatial_dropout1d_2 (SpatialDro (None, None, 128)    0           batch_normalization_5
[0][0]
__________________________________________________________________________________________________
conv1d_9 (Conv1D)               (None, None, 256)    262400      spatial_dropout1d_2[0]
[0]
__________________________________________________________________________________________________
re_lu_6 (ReLU)                  (None, None, 256)    0           conv1d_9[0][0]
__________________________________________________________________________________________________
```

```
____________
batch_normalization_6 (BatchNor (None, None, 256)    1024        re_lu_6[0][0]
__________________________________________________________________________________________
____________
conv1d_11 (Conv1D)              (None, None, 256)    33024       spatial_dropout1d_2[0]
[0]
__________________________________________________________________________________________
____________
conv1d_10 (Conv1D)              (None, None, 256)    524544      batch_normalization_6
[0][0]
__________________________________________________________________________________________
____________
add_3 (Add)                     (None, None, 256)    0           conv1d_11[0][0]
                                                                 conv1d_10[0][0]
__________________________________________________________________________________________
____________
re_lu_7 (ReLU)                  (None, None, 256)    0           add_3[0][0]
__________________________________________________________________________________________
____________
batch_normalization_7 (BatchNor (None, None, 256)    1024        re_lu_7[0][0]
__________________________________________________________________________________________
____________
spatial_dropout1d_3 (SpatialDro (None, None, 256)    0           batch_normalization_7
[0][0]
__________________________________________________________________________________________
____________
conv1d_12 (Conv1D)              (None, None, 512)    1049088     spatial_dropout1d_3[0]
[0]
__________________________________________________________________________________________
____________
max_pooling1d (MaxPooling1D)    (None, None, 512)    0           conv1d_12[0][0]
__________________________________________________________________________________________
____________
lstm (LSTM)                     (None, 8)            16672       max_pooling1d[0][0]
__________________________________________________________________________________________
____________
dense_2 (Dense)                 (None, 4)            36          lstm[0][0]
==========================================================================================
==========
Total params: 3,698,660
Trainable params: 3,696,740
Non-trainable params: 1,920
__________________________________________________________________________________________
____________
```

In [ ]:

```
### TRAINING MODEL ###

optimizer = optimizers.Adam(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'
epochs = 100

model.fit(df['signal_data'], df['class'], epochs = 100, batch_size = 32)
```

FINAL SUMMARY AND FURTHER RESEARCH

In summary it's been a very challenging summer and certainly I wish I could have done more however I feel I have learned a ton and I think it will ultimately help me significantly looking forward onto other projects in the future. In regards to our research I am going to complete the testing phase on my own time and hopefully following our test on a number of other datasets we can summarize our work and publish. In particular I think we are all looking forward to seeing the

portions of the signals that the computer actually uses the identify the various signals (that is a characteristic of using a transformer in our model).