```python
import tensorflow as tf
from tensorflow.keras.callbacks import LearningRateScheduler
from tensorflow.keras.layers import Dense,  Conv2D, MaxPooling2D, Dropout, Flatten, GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam, SGD, RMSprop
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import Callback
import numpy as np
import matplotlib.pyplot as plt
import os
from os.path import join
import multiprocessing
from sklearn.metrics import confusion_matrix

ImageResolution = (640, 360)
ImageResolutionGrayScale = (640, 360, 1)

def trainCNN( ):

    tf.keras.backend.clear_session()

    base_dir = 'C:\work_dir\meteorData\extraData'
    results_dir_weights = 'G:\GIEyA\TFG\meteor_classification\\results\weights\model_19'

    train_dir = join(base_dir, 'train')
    validation_dir = join(base_dir, 'validation')
    test_dir = join(base_dir, 'test')

    #Rescale all images by 1./255

    train_datagen = ImageDataGenerator(rescale=1.0/255#,
                                       # rotation_range=10, # Range from 0 to 180 degrees to randomly rotate images
                                       # width_shift_range=0.05,
                                       # height_shift_range=0.05,
                                       #shear_range=5, # Shear the image by 5 degrees
                                       #zoom_range=0.1,
                                       #horizontal_flip=True,
                                       #vertical_flip=True,
                                       #fill_mode='nearest'
                                       )

    validation_datagen = ImageDataGenerator(rescale=1.0/255.)
    test_datagen = ImageDataGenerator(rescale=1.0/255.0)

    train_generator = train_datagen.flow_from_directory(train_dir,
                                                        batch_size=16, #16
                                                        class_mode='binary',
                                                        color_mode='grayscale',
                                                        target_size=ImageResolution) # 640x360 = 480x480. (640, 360)

    validation_generator = validation_datagen.flow_from_directory(validation_dir,
                                                                  batch_size=16, #16
                                                                  class_mode='binary',
                                                                  color_mode='grayscale',
                                                                  target_size=ImageResolution)

    test_generator = test_datagen.flow_from_directory(test_dir,
                                                      batch_size=1,
        class_mode='binary',
        color_mode='grayscale',
        target_size=ImageResolution,
        shuffle=False)

    model = tf.keras.models.Sequential([
        Conv2D(16, (11, 11), activation='relu', input_shape=ImageResolutionGrayScale, strides=1),
        MaxPooling2D(pool_size=(3, 3)),
        Dropout(0.25),

        Conv2D(12, (7, 7), activation='relu', kernel_initializer='he_uniform'),
        #Conv2D(12, (3, 3), activation='relu', kernel_initializer='he_uniform'),
        MaxPooling2D(pool_size=(2, 2)),
        Dropout(0.25),

        Conv2D(12, (5, 5), activation='relu', kernel_initializer='he_uniform'),
        #Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_uniform'),
        MaxPooling2D(pool_size=(2, 2)),
        Dropout(0.25),

        Conv2D(12, (3, 3), activation='relu', kernel_initializer='he_uniform'),
        #Conv2D(8, (3, 3), activation='relu', kernel_initializer='he_uniform'),
        MaxPooling2D(pool_size=(2, 2)),

        Flatten(),
        Dense(480, activation='relu', kernel_initializer='he_uniform'),
        Dropout(0.30),
        Dense(16, activation='relu', kernel_initializer='he_uniform'),
        Dropout(0.20),
        Dense(1, activation='sigmoid', kernel_initializer='he_uniform')
```

```python
    ])

    print(model.summary())
    optimizer = Adam(learning_rate=5e-4) #3e-3 # Try with more and less learning rate # 5e-3
    model.compile(optimizer=optimizer,
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
    #model.load_weights(join(results_dir_weights, 'model_acc_0.926.h5'))

    class SaveModelCallback(Callback):
        def __init__(self, threshold):
            super(SaveModelCallback, self).__init__()
            self.threshold = threshold

        def on_epoch_end(self, epoch, logs=None):
            if(logs.get('accuracy') >= self.threshold):
                model.save_weights(join(results_dir_weights, 'model_19_acc_' +  str(logs.get('accuracy'))[0:6] + '_val_acc' + str(logs.get('val_accuracy'))[0:6] + '.h5'), save_format='h5')

    callback92 = SaveModelCallback(0.920)

    #39.480 -> Training 39480 = 2 x 2 x 2 × 3 × 5 × 7 × 47
    #9.872 -> Validation = 2 x 2 x 2 x 2 × 617
    history = model.fit(train_generator,
                        validation_data=validation_generator,
                        steps_per_epoch=2467, #2467 #4934
                        epochs=150, #Later train with more epochs if neccessary
                        validation_steps=617, #617 #1234
                        verbose=1,
                        callbacks=[callback92])

    acc      = history.history['accuracy']
    val_acc  = history.history['val_accuracy']
    loss     = history.history['loss']
    val_loss = history.history['val_loss']
    epochs = range(len(acc)) #Get number of epochs

    prob_predicted = model.predict_generator(test_generator, steps=len(test_generator.filenames))
    test_labels = []

    for i in range(0, len(test_generator.filenames)):
        test_labels.extend(np.array(test_generator[i][1]))

    # Get the confusion matrix:
    truePositives = 0
    trueNegatives = 0
    falsePositives = 0
    falseNegatives = 0

    for i in range(len(prob_predicted)):
        if(prob_predicted[i] >= 0.5 and test_labels[i] == 1.0):
            truePositives += 1
        elif(prob_predicted[i] >= 0.5 and test_labels[i] == 0.0):
            falsePositives += 1
        elif(test_labels[i] == 0.0):
            trueNegatives += 1
        elif(test_labels[i] == 1.0):
            falseNegatives += 1

    print('*********************************************')
    print('confusion matrix: ')
    print('true positives: {}'.format(truePositives))
    print('false positives: {}'.format(falsePositives))
    print('true negatives: {}'.format(trueNegatives))
    print('false negatives: {}'.format(falseNegatives))
    print('*********************************************')
    plt.plot(epochs, acc)
    plt.plot(epochs, val_acc)
    plt.title('Meteor detection training and validation accuracy')

    plt.figure()
    plt.plot(epochs, loss)
    plt.plot(epochs, val_loss)
    plt.title('Meteor detection training and validation loss')

    plt.show()


if __name__ == '__main__':
    p = multiprocessing.Process(target=trainCNN)
    p.start()
    p.join()
```