```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

def trainCNN( ):

    tf.keras.backend.clear_session()

    base_dir = 'G:\GIEyA\TFG\meteor_classification\labeledData\evenData'
    results_dir_weights = 'G:\GIEyA\TFG\meteor_classification\\results\weights'

    train_dir = join(base_dir, 'train')
    validation_dir = join(base_dir, 'valid')

    train_meteors_dir = join(train_dir, 'meteors')
    train_non_meteors_dir = join(train_dir, 'non_meteors')
    validation_meteors_dir = join(validation_dir, 'meteors')
    validation_non_meteors_dir = join(validation_dir, 'non_meteors')

    print('total training meteors images: ', len(os.listdir(train_meteors_dir)))
    print('total training non-meteors images: ', len(os.listdir(train_non_meteors_dir)))
    print('total validation meteors images: ', len(os.listdir(validation_meteors_dir)))
    print('total validation non-meteors images: ',
len(os.listdir(validation_non_meteors_dir)))


    #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'
                                       )

    test_datagen = ImageDataGenerator(rescale=1.0/255.)

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

```
       class_mode='binary',
       color_mode='grayscale',
       target_size=(480, 480))


    model = tf.keras.models.Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=(480, 480, 1)), MaxPooling2D(2,2), # Dropout(0.05),
        Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_uniform'), MaxPooling2D(2, 2), #Dropout(0.05),
        Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_uniform'), MaxPooling2D(2, 2), #Dropout(0.05),
        Conv2D(12, (2, 2), activation='relu', kernel_initializer='he_uniform'), MaxPooling2D(2, 2), #Dropout(0.05),
        Conv2D(8,  (2, 2), activation='relu', kernel_initializer='he_uniform'), MaxPooling2D(2, 2), #Dropout(0.05),
        Conv2D(4,  (2, 2), activation='relu', kernel_initializer='he_uniform'), MaxPooling2D(2, 2), #Dropout(0.05),
        #Conv2D(4, (2, 2), activation='relu'),
        Flatten(),
        Dense(144, activation='relu', kernel_initializer='he_uniform'),
        #Dense(32, activation='relu', kernel_initializer='he_uniform'),
        Dense(8, activation='relu', kernel_initializer='he_uniform'),
        Dense(1, activation='sigmoid', kernel_initializer='he_uniform')
    ])

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

    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_acc_' +  str(logs.get('accuracy'))[0:5] + '.h5'), save_format='h5')

    callback90 = SaveModelCallback(0.90)

    #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
                        epochs=15, #Later train with more epochs if neccessary
                        validation_steps=617, #617
                        verbose=1,
                        callbacks=[callback90])

    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

    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()


```