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

def trainCNN( ):

    tf.keras.backend.clear_session()

    base_dir = 'G:\GIEyA\TFG\meteor_classification\labeledData\evenData'
    train_dir = os.path.join(base_dir, 'train')
    validation_dir = os.path.join(base_dir, 'valid')

    train_meteors_dir = os.path.join(train_dir, 'meteors')
    train_non_meteors_dir = os.path.join(train_dir, 'non_meteors')
    validation_meteors_dir = os.path.join(validation_dir, 'meteors')
    validation_non_meteors_dir = os.path.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=48,
                                                        class_mode='binary',
                                                        color_mode='grayscale',
                                                        target_size=(640, 360))
    validation_generator = test_datagen.flow_from_directory(validation_dir,
                                                            batch_size=48,
                                                            class_mode='binary',
                                                            color_mode='grayscale',
                                                            target_size=(640, 360))
```

```python


    model = tf.keras.models.Sequential([
        Conv2D(16, (2,2), activation='relu', input_shape=(640, 360, 1)),
        MaxPooling2D(2,2),
        Dropout(0.2),
        Conv2D(16, (2, 2), activation='relu'),
        MaxPooling2D(2, 2),
        #Dropout(0.2), # No dropout in transitions
        Conv2D(12, (2, 2), activation='relu'),
        MaxPooling2D(2, 2),
        Dropout(0.2),
        Conv2D(12, (2, 2), activation='relu'),
        #MaxPooling2D(2, 2),
        #Dropout(0.2), # No dropout in transitions
        Conv2D(12, (2, 2), activation='relu'),
        MaxPooling2D(2, 2),
        Dropout(0.2),
        Conv2D(8, (2, 2), activation='relu'),
        MaxPooling2D(2, 2),
        Dropout(0.2),
        Conv2D(8, (2, 2), activation='relu'),
        Dropout(0.2),
        Conv2D(8, (2, 2), activation='relu'),
        Dropout(0.2),
        Conv2D(4, (2, 2), activation='relu'),
        Dropout(0.2),
        Conv2D(4, (2, 2), activation='relu'),
        Dropout(0.2),
        Conv2D(4, (2, 2), activation='relu'),
        MaxPooling2D(2, 2),
        #Dropout(0.2), # No dropout in transitions
        Flatten(),
        Dense(48, activation='relu'),
        Dense(24, activation='relu'),
        Dense(12, activation='relu'),
        Dense(1, activation='sigmoid')])

    print(model.summary())
    optimizer = Adam(learning_rate=3e-3)
    model.compile(optimizer=optimizer,
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

    #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=822,
                        epochs=10, #Later train with more epochs if neccessary
                        validation_steps=205,
                        verbose=1)

    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()
```