```
1 import tensorflow as tf
 2 from tensorflow.keras.callbacks import
   LearningRateScheduler
 3 from tensorflow.keras.layers import Dense, Conv2D,
   MaxPooling2D, Dropout, Flatten,
   GlobalAveragePooling2D
 4 from tensorflow.keras.optimizers import Adam, SGD,
   RMSprop
 5 from tensorflow.keras.preprocessing.image import
   ImageDataGenerator
 6 from tensorflow.keras.callbacks import Callback
 7 import numpy as np
8 import matplotlib.pyplot as plt
9 import os
10 from os.path import join
11 import multiprocessing
12 from sklearn.metrics import confusion_matrix
13
14 def trainCNN():
15
16
       tf.keras.backend.clear_session()
17
18
       ImageResolution = (480, 480)
19
       ImageResolutionGrayScale = (480, 480, 1)
20
       modelNumber = 'Model23'
21
22
       base_dir = 'C:\work_dir\meteorData\extraData'
23
       results_dir_weights = 'G:\GIEyA\TFG\
   meteor_classification\\results\weights\\' +
   modelNumber
24
25
       train_dir = join(base_dir, 'train')
       validation_dir = join(base_dir, 'validation')
26
27
       test_dir = join(base_dir, 'test')
28
       #Rescale all images by 1./255
29
30
31
       train_datagen = ImageDataGenerator(rescale=1.0/
   255#,
32
   rotation_range=10, # Range from 0 to 180 degrees to
   randomly rotate images
33
                                           #
   width_shift_range=0.05,
```

```
34
                                            #
   height_shift_range=0.05,
35
                                            #shear_range=5
   , # Shear the image by 5 degrees
36
                                            #zoom_range=0.
   1,
37
                                            #
   horizontal_flip=True,
38
                                            #vertical_flip
   =True,
39
                                            #fill_mode='
   nearest'
40
                                            )
41
       validation_datagen = ImageDataGenerator(rescale=1
42
   .0/255.)
43
44
       test_datagen = ImageDataGenerator(rescale=1.0/255
   .0)
45
46
       train_generator = train_datagen.
   flow_from_directory(train_dir,
47
      batch_size=16, #16
48
      class_mode='binary',
49
      color_mode='grayscale',
50
      target_size=ImageResolution) # 640x360 = 480x480
     (640, 360)
51
52
       validation_generator = validation_datagen.
   flow_from_directory(validation_dir,
53
                 batch_size=16, #16
54
                 class_mode='binary',
55
                 color_mode='grayscale',
56
                 target_size=ImageResolution)
57
58
       test_generator = test_datagen.flow_from_directory
```

```
58 (test_dir,
59
     batch_size=1,
60
     class_mode='binary',
61
     color_mode='grayscale',
62
     target_size=ImageResolution,
63
     shuffle=False)
64
       model = tf.keras.models.Sequential([
65
           Conv2D(16, (9, 9), activation='relu',
66
   input_shape=ImageResolutionGrayScale, strides=1),
           MaxPooling2D(pool_size=(3, 3)),
67
68
           Dropout(0.25),
69
           Conv2D(16, (1, 1), activation='relu',
70
   kernel_initializer='he_uniform'),
           Conv2D(16, (7, 7), activation='relu',
71
   kernel_initializer='he_uniform'),
           MaxPooling2D(pool_size=(3, 3)),
72
73
           Dropout(0.25),
74
           Conv2D(12, (1, 1), activation='relu',
75
   kernel_initializer='he_uniform'),
           Conv2D(12, (5, 5), activation='relu',
76
   kernel_initializer='he_uniform'),
77
           MaxPooling2D(pool_size=(2, 2)),
           Dropout(0.25),
78
79
           Conv2D(12, (1, 1), activation='relu',
80
   kernel_initializer='he_uniform'),
           Conv2D(12, (3, 3), activation='relu',
81
   kernel_initializer='he_uniform'),
           MaxPooling2D(pool_size=(2, 2)),
82
           Dropout(0.25),
83
84
           Conv2D(24, (1, 1), activation='relu',
85
   kernel_initializer='he_uniform'),
           Conv2D(24, (3, 3), activation='relu',
86
   kernel_initializer='he_uniform'),
           MaxPooling2D(pool_size=(2, 2)),
87
```

```
88
 89
            Flatten(),
            Dense(384, activation='relu',
 90
    kernel_initializer='he_uniform'),
 91
            Dropout(0.30),
            Dense(16, activation='relu',
 92
    kernel_initializer='he_uniform'),
 93
            Dropout(0.20),
            Dense(1, activation='sigmoid',
 94
    kernel_initializer='he_uniform')
 95
        1)
 96
 97
        print(model.summary())
 98
        optimizer = Adam(learning_rate=5e-4) #3e-3 # Try
     with more and less learning rate # 5e-3
 99
        model.compile(optimizer=optimizer,
                      loss='binary_crossentropy',
100
                      metrics=['accuracy'])
101
        model.load_weights(join(results_dir_weights, '
102
    Model23_acc_0.9264_val_acc_0.8619.h5'))
103
104
        class SaveModelCallback(Callback):
105
            def __init__(self, thresholdTrain,
    thresholdValid):
106
                super(SaveModelCallback, self).__init__
    ()
107
                self.thresholdTrain = thresholdTrain
108
                self.thresholdValid = thresholdValid
109
110
            def on_epoch_end(self, epoch, logs=None):
                if((logs.get('accuracy') >= self.
111
    thresholdTrain) and (logs.get('val_accuracy') >=
    self.thresholdValid)):
112
                    model.save_weights(join(
    results_dir_weights, modelNumber + '_acc_' +
    logs.get('accuracy'))[0:6]
113
                                             + '_val_acc_
    ' + str(logs.get('val_accuracy'))[0:6] + '.h5'),
    save format='h5')
114
115
        callback_90_85 = SaveModelCallback(0.900, 0.850)
116
117
        # Training -> 66947
        # Validation -> 13388
118
```

```
# Test -> 8928
119
120
121
      history = model.fit(train_generator,
122
                        validation_data=
   validation_generator,
123
                        steps_per_epoch=4184, #4184
124
                        epochs=25, #Later train with
    more epochs if neccessary
                        validation_steps=836, #836
125
126
                        shuffle=True,
127
                        verbose=1,
128
                        callbacks=[callback_90_85])
129
130
   131
   132
133
      prob_predicted = model.predict(test_generator,
   steps=len(test_generator.filenames))
      test_labels = []
134
135
      for i in range(0, len(test_generator.filenames
136
   )):
137
          test_labels.extend(np.array(test_generator[i
   ][1]))
138
139
      # Get the confusion matrix:
140
      truePositives = 0
141
      trueNegatives = 0
142
      falsePositives = 0
143
      falseNegatives = 0
144
      for i in range(len(prob_predicted)):
145
          if(prob_predicted[i] >= 0.5 and test_labels[
146
   i] == 1.0):
147
             truePositives += 1
          elif(prob_predicted[i] >= 0.5 and
148
   test_labels[i] == 0.0):
149
             falsePositives += 1
          elif(test_labels[i] == 0.0):
150
151
             trueNegatives += 1
```

```
elif(test_labels[i] == 1.0):
152
153
             falseNegatives += 1
154
155
      performanceFile = open(join(results_dir_weights)
     'performance' + modelNumber + '.txt'), 'w')
156
      performanceFile.write(
   performanceFile.write('confusion matrix: \n')
157
158
      performanceFile.write('true positives: {}\n'.
   format(truePositives))
      performanceFile.write('false positives: {}\n'.
159
   format(falsePositives))
      performanceFile.write('true negatives: {}\n'.
160
   format(trueNegatives))
161
      performanceFile.write('false negatives: {}\n'.
   format(falseNegatives))
162
      performanceFile.write(
   '************************************/n')
163
      modelPrecision = (truePositives) / (
164
   truePositives + falsePositives)
165
      modelRecall = (truePositives) / (truePositives
    + falseNegatives)
      modelF1score = (2 * (modelPrecision *
166
   modelRecall)) / (modelPrecision + modelRecall)
167
168
      performanceFile.write(
   performanceFile.write('Performance metrics: \n')
169
170
      performanceFile.write('Model Precision: {}\n'.
   format(modelPrecision))
      performanceFile.write('Model Recall: {}\n'.
171
   format(modelRecall))
172
      performanceFile.write('Model F1 Score: {}\n'.
   format(modelF1score))
      performanceFile.write(
173
   174
175
      performanceFile.close()
176
177
   178
```

```
178
   179
               = history.history['accuracy']
180
       acc
181
       val_acc = history.history['val_accuracy']
182
               = history.history['loss']
       loss
       val_loss = history.history['val_loss']
183
       epochs = range(len(acc)) #Get number of epochs
184
185
186
       plt.plot(epochs, acc)
187
       plt.plot(epochs, val_acc)
       plt.title('Meteor detection training and
188
   validation accuracy')
189
       plt.figure()
190
       plt.plot(epochs, loss)
191
       plt.plot(epochs, val_loss)
192
       plt.title('Meteor detection training and
193
   validation loss')
194
195
       plt.show()
196
197
198 if __name__ == '__main__':
       p = multiprocessing.Process(target=trainCNN)
199
200
       p.start()
201
       p.join()
202
203
```