```
 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 = (200, 200)  # (640, 360)
19     ImageResolutionGrayScale = (200, 200, 1) # (640,
   360, 1)
20     modelNumber = 'Model20'
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')
26     validation_dir = join(base_dir, 'validation')
27     test_dir = join(base_dir, 'test')
28
29     #Rescale all images by 1./255
30
31     train_datagen = ImageDataGenerator(rescale=1.0/
   255#,
32                                        #
   rotation_range=10, # Range from 0 to 180 degrees to
   randomly rotate images
33                                        #
```

```
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
42     validation_datagen = ImageDataGenerator(rescale=1
   .0/255.)
43     test_datagen = ImageDataGenerator(rescale=1.0/255
   .0)
44
45     train_generator = train_datagen.
   flow_from_directory(train_dir,
46
      batch_size=16, #16
47
      class_mode='binary',
48
      color_mode='grayscale',
49
      target_size=ImageResolution) # 640x360 = 480x480
   . (640, 360)
50
51     validation_generator = validation_datagen.
   flow_from_directory(validation_dir,
52
                batch_size=16, #16
53
                class_mode='binary',
54
                color_mode='grayscale',
55
                target_size=ImageResolution)
56
57     test_generator = test_datagen.flow_from_directory
```

```
57 (test_dir,
58 
     batch_size=1,
59 
     class_mode='binary',
60 
     color_mode='grayscale',
61 
     target_size=ImageResolution,
62 
     shuffle=False)
63 
64     model = tf.keras.models.Sequential([
65         Conv2D(16, (5, 5), activation='relu',
   input_shape=ImageResolutionGrayScale, strides=1),
66         MaxPooling2D(pool_size=(3, 3)),
67         Dropout(0.25),
68 
69         Conv2D(12, (3, 3), activation='relu',
   kernel_initializer='he_uniform'),
70         #Conv2D(12, (3, 3), activation='relu',
   kernel_initializer='he_uniform'),
71         MaxPooling2D(pool_size=(2, 2)),
72         Dropout(0.25),
73 
74         Conv2D(12, (2, 2), activation='relu',
   kernel_initializer='he_uniform'),
75         #Conv2D(16, (3, 3), activation='relu',
   kernel_initializer='he_uniform'),
76         MaxPooling2D(pool_size=(2, 2)),
77         Dropout(0.25),
78 
79         Conv2D(12, (2, 2), activation='relu',
   kernel_initializer='he_uniform'),
80         #Conv2D(8, (3, 3), activation='relu',
   kernel_initializer='he_uniform'),
81         MaxPooling2D(pool_size=(2, 2)),
82 
83         Flatten(),
84         Dense(588, activation='relu',
   kernel_initializer='he_uniform'),
85         Dropout(0.30),
86         Dense(16, activation='relu',
   kernel_initializer='he_uniform'),
```

```
 87         Dropout(0.20),
 88         Dense(1, activation='sigmoid',
    kernel_initializer='he_uniform')
 89     ])
 90
 91     print(model.summary())
 92     optimizer = Adam(learning_rate=5e-4) #3e-3 # Try
     with more and less learning rate # 5e-3
 93     model.compile(optimizer=optimizer,
 94                   loss='binary_crossentropy',
 95                   metrics=['accuracy'])
 96     #model.load_weights(join(results_dir_weights, '
    model_acc_0.926.h5'))
 97
 98     class SaveModelCallback(Callback):
 99         def __init__(self, thresholdTrain,
    thresholdValid):
100             super(SaveModelCallback, self).__init__
    ()
101             self.thresholdTrain = thresholdTrain
102             self.thresholdValid = thresholdValid
103
104         def on_epoch_end(self, epoch, logs=None):
105             if((logs.get('accuracy') >= self.
    thresholdTrain) and (logs.get('val_accuracy') >=
    self.thresholdValid)):
106                 model.save_weights(join(
    results_dir_weights, modelNumber + '_acc_' +  str(
    logs.get('accuracy'))[0:6] + '_val_acc' + str(logs.
    get('val_accuracy'))[0:6] + '.h5'), save_format='h5'
    )
107
108     callback92 = SaveModelCallback(0.900, 0.900)
109
110     #39.480 -> Training 39480 = 2 x 2 x 2 × 3 × 5 ×
    7 × 47
111     #9.872 -> Validation = 2 x 2 x 2 x 2 × 617
112     history = model.fit(train_generator,
113                         validation_data=
    validation_generator,
114                         steps_per_epoch=2467, #2467
     #4934
115                         epochs=25, #Later train with
     more epochs if neccessary
```

```
116                         validation_steps=617, #617 #
    1234
117                         verbose=1,
118                         callbacks=[callback92])
119 
120 
    ##################################################
    ########################################
121 
    ##################################################
    ########################################
122 
123     prob_predicted = model.predict(test_generator,
    steps=len(test_generator.filenames))
124     test_labels = []
125 
126     for i in range(0, len(test_generator.filenames
    )):
127         test_labels.extend(np.array(test_generator[i
    ][1]))
128 
129     # Get the confusion matrix:
130     truePositives = 0
131     trueNegatives = 0
132     falsePositives = 0
133     falseNegatives = 0
134 
135     for i in range(len(prob_predicted)):
136         if(prob_predicted[i] >= 0.5 and test_labels[
    i] == 1.0):
137             truePositives += 1
138         elif(prob_predicted[i] >= 0.5 and
    test_labels[i] == 0.0):
139             falsePositives += 1
140         elif(test_labels[i] == 0.0):
141             trueNegatives += 1
142         elif(test_labels[i] == 1.0):
143             falseNegatives += 1
144 
145     performanceFile = open(join(results_dir_weights
    , '\performance' + modelNumber + '.txt'), 'w')
146     performanceFile.write(
    '*****************************************\n')
147     performanceFile.write('confusion matrix: \n')
```

```
148     performanceFile.write('true positives: {}\n'.
    format(truePositives))
149     performanceFile.write('false positives: {}\n'.
    format(falsePositives))
150     performanceFile.write('true negatives: {}\n'.
    format(trueNegatives))
151     performanceFile.write('false negatives: {}\n'.
    format(falseNegatives))
152     performanceFile.write(
    '*********************************************\n')
153 
154     modelPrecision = (truePositives) / (
    truePositives + falsePositives)
155     modelRecall = (truePositives) / (truePositives
     + falseNegatives)
156     modelF1score = (2 * (modelPrecision *
    modelRecall)) / (modelPrecision + modelRecall)
157 
158     performanceFile.write(
    '*********************************************\n')
159     performanceFile.write('Performance metrics: \n')
160     performanceFile.write('Model Precision: {}\n'.
    format(modelPrecision))
161     performanceFile.write('Model Recall: {}\n'.
    format(modelRecall))
162     performanceFile.write('Model F1 Score: {}\n'.
    format(modelF1score))
163     performanceFile.write(
    '*********************************************\n')
164 
165     performanceFile.close()
166 
167 
    ##################################################
    ########################################
168 
    ##################################################
    ########################################
169 
170     acc      = history.history['accuracy']
171     val_acc  = history.history['val_accuracy']
172     loss     = history.history['loss']
173     val_loss = history.history['val_loss']
174     epochs = range(len(acc)) #Get number of epochs
```

```
175
176     plt.plot(epochs, acc)
177     plt.plot(epochs, val_acc)
178     plt.title('Meteor detection training and
    validation accuracy')
179
180     plt.figure()
181     plt.plot(epochs, loss)
182     plt.plot(epochs, val_loss)
183     plt.title('Meteor detection training and
    validation loss')
184
185     plt.show()
186
187
188 if __name__ == '__main__':
189     p = multiprocessing.Process(target=trainCNN)
190     p.start()
191     p.join()
192
193
```