

```

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 = (640, 360)
19     ImageResolutionGrayScale = (640, 360, 1)
20     modelNumber = 'Model22'
21
22     base_dir = 'C:\work_dir\meteorData\extraData'
23     results_dir_weights = 'G:\GIEy\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                                     #
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
    (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, (11, 11), activation='relu',
66         input_shape=ImageResolutionGrayScale, strides=1),
67         MaxPooling2D(pool_size=(3, 3)),
68         Dropout(0.25),
69         Conv2D(16, (7, 7), activation='relu',
70         kernel_initializer='he_uniform'),
71         #Conv2D(12, (3, 3), activation='relu',
72         kernel_initializer='he_uniform'),
73         MaxPooling2D(pool_size=(3, 3)),
74         Dropout(0.25),
75         Conv2D(12, (5, 5), activation='relu',
76         kernel_initializer='he_uniform'),
77         #Conv2D(16, (3, 3), activation='relu',
78         kernel_initializer='he_uniform'),
79         MaxPooling2D(pool_size=(2, 2)),
80         Dropout(0.25),
81         Conv2D(8, (3, 3), activation='relu',
82         kernel_initializer='he_uniform'),
83         MaxPooling2D(pool_size=(2, 2)),
84         Flatten(),
85         Dense(840, activation='relu',
86         kernel_initializer='he_uniform'),
87         Dropout(0.30),
88         Dense(32, activation='relu',
89         kernel_initializer='he_uniform'),
90         Dropout(0.20),
91         Dense(1, activation='sigmoid',
92         kernel_initializer='he_uniform')
```

```

88     ])
89
90     print(model.summary())
91     optimizer = Adam(learning_rate=5e-4) #3e-3 # Try
with more and less learning rate # 5e-3
92     model.compile(optimizer=optimizer,
93                   loss='binary_crossentropy',
94                   metrics=['accuracy'])
95     #model.load_weights(join(results_dir_weights, 'model_acc_0.926.h5'))
96
97     class SaveModelCallback(Callback):
98         def __init__(self, thresholdTrain,
99                     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(
107                     results_dir_weights, modelName + '_acc_' + str(
108                         logs.get('accuracy'))[0:6] + '_val_acc' + str(logs.get('val_accuracy'))[0:6] + '.h5'), save_format='h5')
109
110
111     callback90 = SaveModelCallback(0.900, 0.900)
112
113     # Training -> 66947
114     # Validation -> 13388
115     # Test -> 8928
116
117     history = model.fit(train_generator,
118                        validation_data=
119                        validation_generator,
120                        steps_per_epoch=4184, #4184
121                        epochs=120, #Later train
122                        with more epochs if neccessary
123                        validation_steps=836, #836
124                        verbose=1,
125                        callbacks=[callback90])

```

```

120
121     #####
122     #####
123
124     prob_predicted = model.predict(test_generator,
125     steps=len(test_generator_filenames))
126     test_labels = []
127     for i in range(0, len(test_generator_filenames
128     )):
129         test_labels.extend(np.array(test_generator[i
130         ][1]))
131
132     # Get the confusion matrix:
133     truePositives = 0
134     trueNegatives = 0
135     falsePositives = 0
136     falseNegatives = 0
137
138     for i in range(len(prob_predicted)):
139         if(prob_predicted[i] >= 0.5 and test_labels[
140         i] == 1.0):
141             truePositives += 1
142         elif(prob_predicted[i] >= 0.5 and
143         test_labels[i] == 0.0):
144             falsePositives += 1
145         elif(test_labels[i] == 0.0):
146             trueNegatives += 1
147         elif(test_labels[i] == 1.0):
148             falseNegatives += 1
149
150     performanceFile = open(join(results_dir_weights
151     , 'performance' + modelNumber + '.txt'), 'w')
152     performanceFile.write(
153     '*****\n')
154     performanceFile.write('confusion matrix: \n')
155     performanceFile.write('true positives: {}\n'.
156     format(truePositives))
157     performanceFile.write('false positives: {}\n'.
158     format(falsePositives))

```

```

151     performanceFile.write('true negatives: {}\n'.
format(trueNegatives))
152     performanceFile.write('false negatives: {}\n'.
format(falseNegatives))
153     performanceFile.write(
'*****\n')
154
155     modelPrecision = (truePositives) / (
truePositives + falsePositives)
156     modelRecall = (truePositives) / (truePositives
+ falseNegatives)
157     modelF1score = (2 * (modelPrecision *
modelRecall)) / (modelPrecision + modelRecall)
158
159     performanceFile.write(
'*****\n')
160     performanceFile.write('Performance metrics: \n')
161     performanceFile.write('Model Precision: {}\n'.
format(modelPrecision))
162     performanceFile.write('Model Recall: {}\n'.
format(modelRecall))
163     performanceFile.write('Model F1 Score: {}\n'.
format(modelF1score))
164     performanceFile.write(
'*****\n')
165
166     performanceFile.close()
167
168
#####
#####
169
#####
#####
170
171     acc      = history.history['accuracy']
172     val_acc  = history.history['val_accuracy']
173     loss     = history.history['loss']
174     val_loss = history.history['val_loss']
175     epochs = range(len(acc)) #Get number of epochs
176
177     plt.plot(epochs, acc)
178     plt.plot(epochs, val_acc)
179     plt.title('Meteor detection training and

```

```
179 validation accuracy')
180
181     plt.figure()
182     plt.plot(epochs, loss)
183     plt.plot(epochs, val_loss)
184     plt.title('Meteor detection training and
validation loss')
185
186     plt.show()
187
188
189 if __name__ == '__main__':
190     p = multiprocessing.Process(target=trainCNN)
191     p.start()
192     p.join()
193
194
```