```
1  #Προσθήκη όλων των απαραίτητων πακέτων δηλαδή libraries, modules, frameworks (
   βιβλιοθήκες, ενότητες, πλαίσια) ή μερος αυτών
2  #για να μπορέσει να εκτελεστεί ο κώδικας. Κάποια πακέτα εισάγονται ολόκληρα απλά
   με την εντολή "import όνομα_επιθυμητού_πακέτου".
3  #Σε άλλες περιπτώσεις εισάγονται μέρη αυτών που θα χρειαστούν ή κάποιες functions,
   methods τους. Για την εισαγωγή ενός
4  #μόνο μέρους του πακέτου χρησιμοποιείται η σύνταξη "from
   όνομα_επιθυμητού_πακέτου import όνομα_επιθυμητού_μέρους_του". Επίσης
5  #με την προσθήκη της εντολής "as επιθυμητό_νέο_όνομα" δίπλα απο την εντολή
   εισαγωγής κάποιου  πακέτου ή μερους αυτού, δίνεται η
6  #δυνατότητα στον κώδικα να απευθύνεται σε αυτό με ένα νέο όνομα, συνήθως πιο
   σαφές,απλό και σύντομο.
7
8  #Το module os ανήκει στην βιβλιοθήκη της Python και περιέχει εντολές που
   αλληλεπιδρούν με το λειτουργικό σύστημα όπως
9  #για παράδειγμα η πλοήγηση σε φακέλους ή η πρόσβαση σε αρχεία.
10 import os
11
12 #Το module "sys" ανήκει στην βιβλιοθήκη της Python και περιέχει εντολές που
   επιτρέπουν την πρόσβαση σε ορίσματα της γραμμής
13 #εντολών (command line), ειδικές παραμέτρους του συστήματος και συναρτήσεις για
   τον χειρισμό του περιβάλλοντος της Python.
14 import sys
15
16 #Το "tensorflow" είναι ένα framework ανοικτού κώδικα (open-source code) όπου η
   βιβλιοθήκη του περιέχει εντολές με σκοπό τη
17 #δημιουργία και εκπαίδευση μοντέλων μηχανικής μάθησης (machine learning models).
18 import tensorflow
19
20 #Εισαγωγή της συνάρτησης (function) "load_img" απο το "tensorflow.keras.
   preprocessing.image" module. Η συνάρτηση αυτή
21 #φορτώνει ένα αρχείο εικόνας απο τον τοπικό δίσκο του υπολογιστή μέσα στον κώδικα.
22 from tensorflow.keras.preprocessing.image import load_img
23
24 #Εισαγωγή της συνάρτησης "img_to_array" απο το "tensorflow.keras.preprocessing.
   image" module. Η συνάρτηση αυτή, μετατρέπει
25 #μια εικόνα σε διάνυσμα της βιβλιοθήκης NumPy έτσι ώστε αυτή να μπορεί να
   υποβληθεί σε επεξεργασία από μοντέλαμηχανικής
26 #εκμάθησης.
27 from tensorflow.keras.preprocessing.image import img_to_array
28
29 #Εισαγωγή της συνάρτησης "preprocess_input" απο το "tensorflow.keras.applications.
   mobilenet_v2" module. Η συνάρτηση αυτή
30 #προεπεξεργάζεται τις εισαγόμενες εικόνες και τις προετοιμάζει με βάση την
   αρχιτεκτονική του μοντέλου MobileNetV2 που
31 #αποτελεί το συνελικτικό νευρωνικό δίκτυο του μοντέλου που θα εκπαιδευτεί.
32 from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
33
34 #Εισαγωγή της κλάσης(class) "LabelBinarizer" απο το "sklearn.preprocessing"
   module. Η κλάση αυτή χρησιμοποιείται για τη
35 #δυαδοποίηση (binarizing) των ετικετών (labels) ή τη μετατροπή κατηγορικών ετικετών
    (categorical labels) σε δυαδικά διανύσματα.
36 from sklearn.preprocessing import LabelBinarizer
37
```

```
38 #Εισαγωγή της συνάρτησης "to_categorical" απο το "tensorflow.keras.utils" module. Η
    συνάρτηση αυτή μετατρέπει ακέραιες
39 #ετικέτες σε κωδικοποιημένα διανύσματα της μεθόδου one-hot encoding.
40 from tensorflow.keras.utils import to_categorical
41
42 #Εισαγωγή της βιβλιοθήκης "numpy" με το ψευδώνυμο "np" που χρησιμεύει για
    αριθμητικούς υπολογισμούς στην Python. Επίσης
43 #παρέχει υποστήριξη για μεγάλους πολυδιάστατους πίνακες ή πίνακες διανυσμάτων,
    αφού περιέχει μια συλλογή μαθηματικών
44 #συναρτήσεων για την αποτελεσματική επεξεργασία τους.
45 import numpy as np
46
47 #Εισαγωγή της συνάρτησης "train_test_split" απο το "sklearn.model_selection"
    module. Η συνάρτηση αυτή χωρίζει το σύνολο
48 #δεδομένων (image dataset) σε υποσύνολα εκπαίδευσης (training set/subset) και
    δοκιμής (testing set/subset).
49 from sklearn.model_selection import train_test_split
50
51 #Εισαγωγή της κλάσης "ImageDataGenerator" απο το "tensorflow.keras.
    preprocessing.image" module. Η κλάση αυτή αυξάνει
52 #τα δεδομένα, δημιουργώντας ομάδες αυξημένων εικόνων για την εκπαίδευση μοντέλων
    βαθιάς μάθησης (deep learning models).
53 from tensorflow.keras.preprocessing.image import ImageDataGenerator
54
55 #Εισαγωγή της αρχιτεκτονικής του μοντέλου "MobileNetV2" από το "tensorflow.keras.
    applications" module. Το MobileNetV2
56 #είναι μια προεκπαιδευμένη αρχιτεκτονική συνελικτικού νευρωνικού δικτύου που
    μπορεί να χρησιμοποιηθεί σε περιπτώσεις
57 #ταξινόμησης εικόνων.
58 from tensorflow.keras.applications import MobileNetV2
59
60 #Εισαγωγή του επιπέδου (layer) "Input" απο το "tensorflow.keras.layers" module. Το
    επίπεδο αυτό αντιπροσωπεύει την είσοδο
61 #στο πλήρως συνδεδεμένο νευρωνικό δίκτυο (Fully Connected Neural Network) του
    μοντέλου, ορίζοντας το σχήμα και τον τύπο
62 #των δεδομένων εισόδου.
63 from tensorflow.keras.layers import Input
64
65 #Εισαγωγή του επιπέδου "AveragePooling2D" απο το "tensorflow.keras.layers"
    module. Το επίπεδο αυτό εκτελεί μέση συγκέντρωση
66 #(Average Pooling), η οποία μειώνει τις χωρικές διαστάσεις των δεδομένων εισόδου
    λαμβάνοντας το μέσο όρο κάθε παραθύρου
67 #συγκέντρωσης.
68 from tensorflow.keras.layers import AveragePooling2D
69
70 #Εισαγωγή του επιπέδου "Flatten" απο το "tensorflow.keras.layers" module. Το
    επίπεδο αυτό μετατρέπει την πολυδιάστατη είσοδο
71 #σε ένα μονοδιάστατο διάνυσμα, σε κατάλληλη δηλαδή μορφή για την εισαγωγή του στο
    επόμενο επίπεδο που είναι πλήρως συνδεδεμένο
72 #με το γειτονικό του.
73 from tensorflow.keras.layers import Flatten
74
75 #Εισαγωγή του επιπέδου "Dense" απο το "tensorflow.keras.layers" module. Το επίπεδο
    αυτό αντιπροσωπεύει ένα πλήρως συνδεδεμένο
```

```
76  #επίπεδο, όπου κάθε νευρώνας συνδέεται με κάθε νευρώνα του προηγούμενου
    επιπέδου.
77  from tensorflow.keras.layers import Dense
78
79  #Εισαγωγή του επιπέδου "Dropout" απο το "tensorflow.keras.layers" module. Το
    επίπεδο αυτό ρυθμίζει τυχαία ένα μέρος των
80  #εισόδων του επόμενου επιπέδου στο 0 κατά τη διάρκεια της εκπαίδευσης, κάτι που
    βοηθά στην αποφυγή της υπερβολικής
81  #προσαρμογής (overfitting).
82  from tensorflow.keras.layers import Dropout
83
84  #Εισαγωγή της κλάσης "Model" απο το "tensorflow.keras.models" module. Η κλάση
    αυτή δημιουργεί ένα αντικείμενο μοντέλου που
85  #καθορίζει την αρχιτεκτονική ενός νευρωνικού δικτύου.
86  from tensorflow.keras.models import Model
87
88  #Εισαγωγή του βελτιστοποιητή (optimizer) "Adam" από το "tensorflow.keras.
    optimizers" module. Ο Adam είναι ένας αλγόριθμος
89  #βελτιστοποίησης που χρησιμοποιείται συνήθως για την εκπαίδευση μοντέλων βαθιάς
    μάθησης (deep learning).
90  from tensorflow.keras.optimizers import Adam
91
92  #Εισαγωγή των συναρτήσεων "roc_curve" και "auc" απο το "sklearn.metrics" module
    . Οι συναρτήσεις αυτές  υπολογίζουν την
93  #καμπύλη ROC και την περιοχή AUC (AUC area) κάτω από την καμπύλη αυτή για την
     αξιολόγηση μοντέλων ταξινόμησης.
94  from sklearn.metrics import roc_curve, auc
95
96  #Εισαγωγή της συνάρτησης "classification_report" απο το "sklearn.metrics" module.
    Η συνάρτηση αυτή Δημιουργεί μια αναφορά
97  #με διάφορες μετρήσεις ταξινόμησης, όπως ακρίβεια, ανάκληση και βαθμολογία F1,
    για την αξιολόγηση της απόδοσης ενός μοντέλου
98  #ταξινόμησης.
99  from sklearn.metrics import classification_report
100
101 #Εισαγωγή του module "pyplot" απο τη βιβλιοθήκη "matplotlib" με το ψευδώνυμο "
    plt". Το Matplotlib είναι μια βιβλιοθήκη
102 #σχεδίασης για την γλώσσα προγραμματισμού Python και το pyplot παρέχει μια απλή
    διεπαφή για τη δημιουργία διαφόρων τύπων
103 #γραφημάτων και απεικονίσεων.
104 import matplotlib.pyplot as plt
105
106 #Αρχικοποίηση των μεταβλητών initial learning rate, epochs, batch size και image
    size. Κάνοντας αλλαγές στις παραμέτρους αυτές,
107 #μπορούμε να εκπαιδεύσουμε διαφορετικά μοντέλα και να τα συγκρίνουμε ώστε στο
    τέλος να κρατήσουμε εκείνο με τα καλύ-
108 #τερα δυνατά αποτελέσματα και το μεγαλύτερο ποσοστό επιτυχίας.
109 #1)Initial learning Rate(INIT_LR): Είναι μια υπερπαράμετρος που καθορίζει πόσο
    γρήγορα ή αργά η συνάρτηση βελτιστοποίησης
110 #του σφάλματος που έχουμε επιλέξει (Adam), κατεβαίνει την καμπύλη σφάλματος.
    Συνήθως η τιμή της βρίσκεται ανάμεσα στο
111 #0.0001 and 0.01.
112 #2)Epochs(EPOCHS): Είναι μια υπερπαράμετρος η οποία ορίζει τον αριθμό των
    επαναλήψεων που θα χρειαστεί να εκτελεστούν
```

```
#για την εκπαίδευση του μοντέλου.
#3)Batch size(BS): Είναι μια υπερπαράμετρος η οποία θέτει τον αριθμό των δεδομένων εκπαίδευσης (train_images, train_labels) που
#χρησιμοποιούμε σε μια εποχή (epoch) για να εκπαιδεύσουμε το νευρωνικό δίκτυο. Συνήθως για τα CNN επιλέγεται το 32.
#4)Image size(IMAGE_SIZE): Είναι μια παράμετρος η οποία ορίζει τις νέες διαστάσεις που θα έχουν οι εικόνες για την
#εκπαίδευση του μοντέλου.

INIT_LR = 1e-4
EPOCHS = 20
BS = 32
IMAGE_SIZE = 224


#                           """Μέρος 1ο - Data Preprocessing"""


#Παρακάτω ακολουθεί η διαδικασία δημιουργίας μοναδικού φακέλου για το μοντέλο που θα εκπαιδευτεί, ο οποίος θα περιέχει
#όλες τις απαραίτητες πληροφορίες για αυτό. Αρχικά, δίνεται το όνομα του φακέλου στη μεταβλητή "folder_of_model" που είναι
#ο συνδυασμός των τριών βασικών υπερπαραμέτρων που θα χρησιμοποιηθούν για την εκπαίδευση του μοντέλου και δηλώθηκαν προηγουμένως.
#Επίσης το string θα περιέχει το όνομα του φακέλου "models/", που θα περιέχει τον νέο υποφάκελο, σε συνδυασμό με το
#όνομα του υποφακέλου αυτού, έτσι ώστε να ελεγχθεί παρακάτω η ακριβής τοποθεσία του υποφακέλου.
folder_of_model = f"models/{INIT_LR}_{EPOCHS}_{BS}"

#Έλεγχος εάν υπάρχει ο φάκελος με το όνομα του "folder_of_model" μέσα στον φάκελο "models/".
if os.path.exists(folder_of_model):
    #Εάν ο φάκελος υπάρχει τότε εκτυπώνεται ενημερωτικό μήνυμα στο τερματικό, που φαίνεται παρακάτω, όπου το "{folder_of_model[7:]}"
    #θα αντικατασταθεί με την ονομασία αυτού του φακέλου. Το "[7:]" λέγεται string slicing και καταργεί τους πρώτους επτά
    #χαρακτήρες του string "folder_of_model" ώστε να μην εμφανιστεί στην οθόνη το "models/", αλλιώς θα εμφανιζόταν η τοποθεσία
    #του υποφακέλου και όχι η ονομασία του.
    print(f"Αυτή η έκδοση του μοντέλου που προσπαθήσατε να εκπαιδεύσετε ({folder_of_model[7:]}) έχει ήδη δημιουργηθεί."
    f" Το πρόγραμμα θα τερματιστεί τώρα...")
    sys.exit()

#Εκτύπωση ενημερωτικού μηνύματος στην οθόνη.
print(f"[ΕΝΗΜΕΡΩΣΗ] Η εκπαίδευση του μοντέλου με ονομασία έκδοσης '{folder_of_model[7:]}' ξεκίνησε...")

#Δημιουργία του νέου φακέλου μέσα στον φάκελο "models/" με την εντολή "makedirs()" της βιβλιοθήκης os.
os.makedirs(folder_of_model)

#Δημιουργία μιας μεταβλητής (string type) με περιεχόμενο την τοποθεσία των εικόνων
```

```
151 ,
152 #που θα χρησιμοποιηθούν για την εκπαίδευση του μοντέλου.
153 dataset_location = r"D:\projects\face_mask_detection\dataset"
154
155 #Δημιουργία μιας λίστας με δύο περιεχόμενα, το "with_mask" και το "without_mask".
156 dataset_classes = ["with_mask", "without_mask"]
157
158 #Δημιουργία μιας λίστας που αργότερα θα περιέχει όλες τις εικόνες ως αριθμούς και
159 #πιο συγκεκριμένα ως arrays.
160 data = []
161
162 #Δημιουργία μιας λίστας, που αργότερα θα περιέχει για κάθε μία απο τις εικόνες της
    λίστας data
163 #αντίστοιχα έναν αριθμό, ο οποίος θα αντιπροσωπεύει το label "with_mask" ή το "
    without_mask".
164 labels = []
165
166 #Εκτύπωση ενημερωτικού μηνύματος στην οθόνη.
167 print("[ΕΝΗΜΕΡΩΣΗ] Η φόρτωση των εικόνων ξεκίνησε...")
168
169 #Δημιουργία βρόγχου επανάληψης που την πρώτη φορά το "dataset_class" = "
    with_mask"
170 #και τη δεύτερη/τελευταία φορά θα είναι "dataset_class" = "without_mask".
171 for dataset_class in dataset_classes:
172
173     #Δημιουργία μεταβλητής τύπου string η οποία περιέχει το αποτέλεσμα
174     #της ένωσης δύο επιμέρους αλφαριθμητικών, του "dataset_location" και του "
    dataset_class"
175     #παραδείγματος χάριν:
176     #dataset_location = r"D:\projects\face_mask_detection\dataset\"
177     #dataset_class = "with_mask"
178     #Άρα path = "D:\projects\face_mask_detection\dataset\with_mask"
179     path = os.path.join(dataset_location, dataset_class)
180
181     #Δημιουργία βρόγχου επανάληψης, που σε κάθε επανάληψή του το περιεχόμενο της
    μεταβλητής "img_name"
182     #θα είναι η ονομασία μίας από τις εικόνες του φακέλου που δείχνει το path.
183     #Πιο συγκεκριμένα, το "os.listdir(path)" δημιουργεί μία λίστα με όλες τις ονομασίες
184     #των εικόνων που περιέχει ο φάκελος που δείχνει το path.
185     for img_name in os.listdir(path):
186         #Δημιουργία μεταβλητής τύπου string η οποία περιέχει το αποτέλεσμα
187         #της ένωσης δύο επιμέρους αλφαριθμητικών, του "path" και του "img_name"
188         #παραδείγματος χάριν:
189         #path = r"D:\projects\face_mask_detection\dataset\with_mask\"
190         #img_name = "0_0_21.jpg"
191         #Άρα img_path = "D:\projects\face_mask_detection\dataset\with_mask\0_0_21.
    jpg"
192         img_path = os.path.join(path, img_name)
193
194         #Εφόσον το "img_path" δείχνει στην τοποθεσία του αρχείου μιας συγκεκριμένης
    εικόνας,
195         #το "load_img" φορτώνει στην μεταβλητή "image" την εικόνα αυτή με τις
    συγκεκριμένες
196         #διαστάσεις που ορίζει το "target_size". Άρα με αυτήν την εντολή προσαρμόζουμε
```

```
196  όλες
197      #τις εικόνες έτσι ώστε να έχουν την ίδια διάσταση με το ίδιο aspect ratio που
     είχαν.
198      image = load_img(img_path, target_size=(IMAGE_SIZE, IMAGE_SIZE))
199
200      #Μετατροπή της εικόνας που βρίσκεται στο image σε μορφή πίνακα (array) για
     να μπορούμε
201      #να την επεξεργαστούμε αργότερα πιο εύκολα. Ένα κομμάτι του array αυτού θα
     έχει για
202      #παράδειγμα την παρακάτω μορφή:
203      #[84. 58. 45.]
204      #[84. 58. 45.]
205      #[84. 58. 45.]
206      image = img_to_array(image)
207
208      #Επειδή για το CNN model μας θα χρησιμοποιήσουμε την αρχιτεκτονική του
     μοντέλου
209      #mobilenet_v2, χρειάζεται να χρησιμοποιήσουμε την εντολή "preprocess_input"
     πάνω
210      #στο array της εικόνας μας. Η εντολή αυτή κάνει κάποιες μετατροπές και
     προσαρμογές
211      #στο array έτσι ώστε αυτό να είναι συμβατό κατά την εκπαίδευση του μοντέλου
     μας.
212      #Λαμβάνοντας υπόψιν το παράδειγμα απεικόνισης ενός μέρους του array απο την
      προηγούμενη
213      #εντολή μπορούμε να δούμε την διαφορά του παρακάτω, αφού δηλαδή υποστεί
     την εντολή του "preprocess_input()":
214      #[-0.34117645 -0.54509807 -0.64705884]
215      #[-0.34117645 -0.54509807 -0.64705884]
216      #[-0.34117645 -0.54509807 -0.64705884]
217      image = preprocess_input(image)
218
219      #Προσθήκη με τη σειρά, όλων των arrays των εικόνων στη λίστα data ώστε να
     είναι
220      #αποθηκευμένες με αυτή τη μορφή σε ένα μέρος που θα μας διευκολύνει στην
     μετέπειτα
221      #επεξεργασία τους.
222      data.append(image)
223
224      #Για κάθε μία απο τις εικόνες αποθηκεύεται αντίστοιχα και ένα label με την
     κατάσταση της
225      #εικόνας. Δηλαδή, είτε "with mask" είτε "without_mask". Όλα τα labels
     αποθηκεύονται σε μία λίστα,
226      #όπως και όλες οι εικόνες στην προηγούμενη εντολή, για να μπορούν να
     επεξεργαστούν αργότερα τα δεδομένα
227      #με μεγαλύτερη ευκολία.
228      labels.append(dataset_class)
229
230  #Επειδή τα deep learning μοντέλα λειτουργούν σωστά με δεδομένα της μορφής arrays
     , τα labels παρακάτω,
231  #από λίστα μετατρέπονται και αυτά σε array και ειδικότερα τα δεδομένα του που ήταν
232  #προηγουμένως αλφαριθμητικά, πλέον θα είναι αριθμοί.
233  #Αρχικά καλείται η κλάση "LabelBinarizer()" η οποία δημιουργεί το αντικείμενο (
     object) lb.
```

```
234 #Αυτό γίνεται για να μπορούμε να χρησιμοποιήσουμε τις μεθόδους (class methods) της
    προαναφερόμενης
235 #κλάσης πιο εύκολα.
236 lb = LabelBinarizer()
237
238 #Το object lb καλεί την μέθοδο fit_transform πάνω στη λίστα labels και μετατρέπει
    όλα τα δεδομένα
239 #της σε 0 και 1. Δηλαδή το αλφαριθμητικό "with_mask" αντικαταστάθηκε με τον
    αριθμό 0 και το "without_mask"
240 #με τον αριθμό 1. Επίσης το labels απο λίστα μετατρέπεται σε array (class numpy.
    ndarray int32) με δεδομένα της μορφής
241 #[0] ή [1].
242 labels = lb.fit_transform(labels)
243
244 #Η μέθοδος "to_categorial" χρησιμοποιεί την κωδικοποίηση One-hot encoding, η
    οποία μετατρέπει μια λίστα/array που περιέχει
245 #κατηγορίες όπως το labels σε μορφή τέτοια που μπορεί να χρησιμοποιηθεί εύκολα
    από αλγόριθμους μηχανικής
246 #εκμάθησης (machine learning algorithms).Η βασική ιδέα της κωδικοποίησης αυτής
    είναι η δημιουργία νέων μεταβλητών που
247 #λαμβάνουν τις τιμές 0 και 1 για να αντιπροσωπεύουν τις αρχικές κατηγορικές τιμές.
    Το labels μετά την κωδικοποίηση
248 #θα περιέχει δεδομένα της μορφής [1. 0.] για "with_mask" ή [0. 1.] για "
    without_mask" και θα έχει τα εξής χαρακτηριστικά
249 #(class numpy.ndarray float32).
250 labels = to_categorical(labels)
251
252 #Μετατροπή με τη βοήθεια της βιβλιοθήκης numpy(np) της λίστας data που περιέχει
    όλα τα array των εικόνων,
253 #ως ένα array και συγκεκριμένα τύπου float32.
254 data = np.array(data, dtype="float32")
255
256 #Το labels επειδή μετατράπηκε προηγουμένως σε array τύπου float32 δεν χρειάζεται
    θεωρητικά να εκτελεστεί η παρακάτω
257 #εντολή, αλλά για λόγους τυπικότητας και ασφάλειας πρέπει να εκτελεστεί.
258 labels = np.array(labels)
259
260 #Η μέθοδος "train_test_split" της βιβλιοθήκης "scikit-learn(sklearn.model_selection
    )" χωρίζει τα δεδομένα των data και
261 #labels σε τέσσερα arrays δύο κατηγοριών. Ουσιαστικά τα δεδομένα τους θα
    χωριστούν σύμφωνα με το ποσοστό που ορίζει η ιδιότητα
262 #"test_size" όπου στην προκειμένη περίπτωση είναι 0.2 ή αλλιώς 20%. Άρα το 20%
    των δεδομένων θα αποθηκευτεί στα arrays
263 #"test_images" και "test_labels" που αφορούν τη κατηγορία testing και το 80% θα
    αποθηκευτεί στα "train_images" και "train_labels" της κατηγορίας
264 #training. Τα "train_images" και "train_labels" θα χρησιμοποιηθούν αργότερα για
    την εκπαίδευση του μοντέλου, το οποίο αφού εκπαιδευτεί
265 #θα δοκιμαστεί με τα "test_images" και "test_labels", για την απόδοση, την
    αποτελεσματικότητα και το ποσοστό επιτυχίας του. Η ιδιότητα
266 #"stratify" δείχνει στο array των labels έτσι ώστε ο διαχωρισμός των δεδομένων στα "
    train_images", "test_images", "train_labels", "test_labels" να
267 #γίνει με ομοιόμορφη κατανομή και να μην έχουμε σφάλματα κατά την εκπαίδευση.
    Εάν δεν ορίζαμε το "stratify" ως προς το labels,
268 #τότε το πρόγραμμα μπορεί να αποθήκευε τυχαία στα training arrays μόνο τα
```

```
268 δεδομένα που αντιστοιχούν σε labels=[0. 1] και έτσι
269 #αργότερα το μοντέλο να έχει εκπαιδευτεί μόνο για ανθρώπους που δε φοράνε μάσκα. Η ιδιότητα "random_state" παίρνει ως όρισμα έναν
270 #αριθμό, ο οποίος συνήθως είναι το 42. Αυτός ο αριθμός ορίζει την τυχαιότητα που θα χωριστούν τα δεδομένα στα arrays "train_images",
271 #"test_images", "train_labels", "test_labels". Αυτή η ιδιότητα βοηθάει έτσι ώστε αν μελλοντικά θέλουμε να συγκρίνουμε διαφορετικά μοντέλα
272 #μεταξύ τους να είμαστε σίγουροι ότι τα δεδομένα μας χωρίστηκαν με τον ίδιο τρόπο ( λογική τυχαιότητας ν. 42) για την εκπαίδευση
273 #όλων των υπόλοιπων μοντέλων μας.
274 (train_images, test_images, train_labels, test_labels) = train_test_split(data, labels, test_size=0.20, stratify=labels, random_state=42)
275
276
277 #                    """Μέρος 2ο - Data augmentation-Αύξηση δεδομένων"""
278
279
280 #Δημιουργία του αντικειμένου aug_gen από τη κλάση ImageDataGenerator της βιβλιοθήκης tensorflow.keras.preprocessing.image.
281 #Το "ImageDataGenerator()" είναι μια κλάση που βοηθάει στην αύξηση των δεδομένων (data augmentation) και συγκεκριμένα
282 #των εικόνων που θα εκπαιδευτεί το μοντέλο. Το βασικό πλεονέκτημα της αύξησης αυτής είναι πως δεν χρειάζεται να
283 #αναζητήσει κάποιος χειροκίνητα νέες εικόνες στο διαδίκτυο για να εμπλουτίσει το dataset. Η κλάση αυτή λαμβάνει
284 #κάθε εικόνα του "train_images" με τη σειρά κατά την εκπαίδευση του μοντέλου, την αντιγράφει μερικές φορές και επεξεργάζεται
285 #αυτά τα αντίγραφά της με τέτοιο τρόπο ώστε να φαίνονται σαν να είναι νέες, διαφορετικές εικόνες από το πρωτότυπο.
286 #Έτσι το dataset μας θα έχει μεγαλύτερη ποικιλία εικόνων. Οι επεξεργασίες που θα δεχτούν τα αντίγραφα είναι συγκεκριμένες
287 #και εξαρτώνται από τα ορίσματα/ιδιότητες της κλάσης αυτής που θα επιλεχθούν.
288 #Συγκεκριμένα επιλέχθηκαν:
289 #1) rotation_range=20: Τυχαία περιστροφή των αντιγράφων ανάμεσα στα όρια των -20 με 20 μοιρών.
290 #2)zoom_range=0.15: Τυχαία εφαρμογή μεγέθυνσης ή σμίκρυνσης στα αντίγραφα. Όταν η τιμή είναι μικρότερη από 1,0, η εικόνα
291 #σμικρύνεται, με αποτέλεσμα να φαίνεται μικρότερη. Αντίθετα, μία τιμή μεγαλύτερη από 1,0 κάνει ζουμ στην εικόνα,
292 #κάνοντάς τη να φαίνεται μεγαλύτερη. Για παράδειγμα, εάν το εύρος ζουμ έχει οριστεί σε [0,8, 1,2], οι εικόνες μπορούν
293 #να υποστούν τυχαίο ζουμ μεταξύ 80% και 120% του αρχικού τους μεγέθους, είτε δηλαδή να μικραίνουν είτε να μεγαλώνουν.
294 #Στην περίπτωσή μας οι εικόνες σμικρύνονται μόνο.
295 #3)width_shift_range=0.2: Τυχαία μετατόπιση των αντιγράφων μέχρι και ένα ποσοστό του πλάτους τους. Απο 0% έως 20% στη
296 #συγκεκριμένη περίπτωση.
297 #4)height_shift_range=0.2: Τυχαία μετατόπιση των αντιγράφων μέχρι και ένα ποσοστό του ύψους τους. Απο 0% έως 20% στη
298 #συγκεκριμένη περίπτωση.
299 #5)shear_range=0.15: Τυχαία διαστρέβλωση των αντιγράφων ως προς τον οριζόντιο ή τον κάθετο άξονα. Η γωνία της διαστρέβλωσης
300 #στη συγκεκριμένη περίπτωση έχει ορισθεί απο -0.15 έως 0.15 ακτίνια (radians) και όχι μοίρες όπως στην παράμετρο "rotation_range"
```

```
#όπου προαναφέρθηκε.
#6)horizontal_flip=True: Ενεργοποίηση της οριζόντιας τυχαίας αναστροφής των αντιγράφων. Η αναστροφή γίνεται ως προς τον
#κατακόρυφο άξονα όπου η αριστερή μεριά της εικόνας μεταφέρεται δεξιά και η δεξιά στα αριστερά. Αυτό είναι χρήσιμο εάν σε
#κάποιες εικόνες υπάρχει συμμετρία μεταξύ της πάνω και κάτω μεριάς της εικόνας, οπότε έτσι πετυχαίνουμε να κάνουμε ένα
#αντίγραφο να φαίνεται σαν μία εντελώς διαφορετική εικόνα.
#7)fill_mode="nearest": Αυτή η παράμετρος είναι αρκετά σημαντική διότι « επιδιορθώνει» τα νέα αντίγραφα που υπέστησαν όλες
#τις αλλαγές που αναφέραμε στις προηγούμενες παραμέτρους. Ειδικότερα, λόγω των μεταμορφώσεών (transformation) τους, τα αντίγραφα
#ενδέχεται να περιέχουν κάποια νέα pixel ή να έχουν δημιουργήσει κενές περιοχές. Η παράμετρος "fill_mode" τα διορθώνει,
#με τη μεθοδολογία "nearest" στη συγκεκριμένη περίπτωση, όπου γεμίζει αυτά τα νέα ή κενά pixel με τιμές χρώματος που έχει
#το pixel στην αντίστοιχη θέση της αυθεντικής εικόνας.
aug_gen = ImageDataGenerator(
rotation_range=20,
zoom_range=0.15,
width_shift_range=0.2,
height_shift_range=0.2,
shear_range=0.15,
horizontal_flip=True,
fill_mode="nearest")


#                """Μέρος 3ο - Κατασκευή του μοντέλου με τη μέθοδο TRANSFER LEARNING """


#Δημιουργία ενός απο τα δύο μέρη του μοντέλου που θα εκπαιδευτεί. Το πρώτο αυτό μέρος ονομάζεται βασικό μοντέλο (baseModel)
#και χρησιμοποιεί την αρχιτεκτονική του συνελικτικού νευρωνικού δικτύου MobileNetV2 που έχει ήδη εκπαιδευτεί (pre-trained
# model) στο παρελθόν. Η μεταβλητή baseModel μετατρέπεται σε νευρωνικό δίκτυο άρα και σε αντικείμενο (object) της βιβλιοθήκης
#keras.engine.functional.Functional, με χαρακτηριστικά που δηλώνονται στις παρακάτω ιδιότητες.
#1)weights: Εδώ αρχικοποιούνται τα βάρη του νευρωνικού δικτύου όπου επιλέχθηκαν να είναι ίσα με τα προεκπαιδευμένα (pre-trained)
#βάρη που προέκυψαν από την εκπαίδευση του MobileNetV2 πάνω στο dataset του Imagenet στο παρελθόν. Το Imagenet είναι ένα
#τεράστιο dataset που περιέχει εκκατομύρια labels εικόνων διαφορετικών κατηγοριών . Εκπαιδεύοντας το μοντέλο μας με αυτά τα
#βάρη, επιτυγχάνεται μείωση του χρόνου εκπαίδευσης και καλύτερα αποτελέσματα αφού αυτά τα βάρη μπορούν να αναγνωρίζουν κάποια
#γενικά οπτικά μοτίβα (general visual patterns) μέσα σε μία εικόνα.
#2)include_top: Με τη boolean τιμή False αφαιρούνται από την κορυφή του νευρωνικού δικτύου τα πλήρως συνδεδεμένα επίπεδα
#(FC layers) που ήταν υπεύθυνα για την ταξινόμηση (classification) των τελικών αποτελεσμάτων. Αυτά τα FC layers(Fully Connected Layers)
#καταργούνται διότι αργότερα θα προσθέσουμε τα δικά μας FC layers που θα εκπαιδεύσουμε στο headModel και θα ταξινομούν τα δεδομένα
```

```
#στις κατηγορίες που θέλουμε εμείς για την συγκεκριμένη εργασία.
#3)input_tensor: Εδώ δηλώνονται οι διαστάσεις των εικόνων του dataset που θα εκπαιδευτεί το μοντέλο καθώς και το είδος των
#εικόνων όπου στην συγκεκριμένη περίπτωση θα είναι έγχρωμες τριών επιπέδων ( RGB).
baseModel = MobileNetV2(weights="imagenet", include_top=False, input_tensor= Input(shape=(IMAGE_SIZE, IMAGE_SIZE, 3)))

#Δημιουργία του δεύτερου μέρους του μοντέλου που θα εκπαιδετεί. Αυτό ονομάζεται " headModel", το οποίο αποτελείται κυρίως από τα
#FC layers που προαναφέρθηκαν και είναι υπεύθυνο για την ταξινόμηση των δεδομένων και την σωστή επιλογή των τελικών αποτελεσμάτων
#(with mask/without mask). Αρχικά με την εντολή baseModel.output δηλώνεται ότι η μεταβλητή "headModel" θα είναι η έξοδος
#του "baseModel", άρα θα δέχεται τα δεδομένα (εξαγόμενα χαρακτηριστικά μιας εικόνας) που επεξεργάστηκε το "baseModel" και με τη
#σειρά του θα τα περνάει και αυτό από επεξεργασία για την τελική τους ταξινόμηση. Πιο συγκεκριμένα αυτό το σύνολο δεδομένων
#που θα δεχτεί το "headModel" ονομάζεται feature map και έχει τριδιάστατη μορφή 7x7x1280. Το feature map περιέχει πληροφορίες
#για τα χαρακτηριστικά μιας εικόνας. Επίσης τα layers που θα χρησιμοποιηθούν βρίσκονται στο tensorflow.keras.layers.
headModel = baseModel.output

#Ενσωμάτωση του AveragePooling2D layer στο headModel. Αυτό το layer μετατρέπει την μορφή του feature map απο 7x7x1280
#σε 1x1x1280 εάν το pool_size επλιλεχθεί (7,7). Αυτός ο μετασχηματισμός δέχεται κάθε κανάλι (channel) του 7x7x1280 με τη σειρά
#και απο 49(7x7) pixel το αλλάζει σε 1(1x1). Συγκεκριμένα υπολογίζεται ο μέσος όρος των τιμών που περιέχουν τα 49 pixel
#και αυτός αποθηκεύεται στη νέα μορφή του feature map. Κάποια απο τα πλεονεκτήματα αυτής της διαδικασίας είναι η μείωση των
#χωρικών διαστάσεων (spatial dimensions), η μείωση του θορύβου και η βελτίωση της υπολογιστικής απόδοσης.
headModel = AveragePooling2D(pool_size=(7, 7))(headModel)

#Ενσωμάτωση του Flatten στο headModel. Το layer αυτό δέχεται το αποτέλεσμα του AveragePooling2D καλώντας δίπλα από την εντολή
#Flatten(name="flatten") το "(headModel)". Συγκεκριμένα δέχεται το feature map της μορφής 1x1x1280 το οποίο έχει 1280 κανάλια
#και το μετατρέπει σε ένα διάνυσμα (vector) μίας μόνο διάστασης, δηλαδή 1x1280. Αυτή η διαδικασία βοηθάει κυρίως στη συμβατότητα,
#αφού από το επόμενο βήμα που θα ακολουθήσουν τα FC layers αυτά θα δεχτούν ως inputs το feature map που θα πρέπει να έχει
#τη μορφή μονοδιάστατου διανύσματος. Ειδικότερα το Flatten layer παίζει κρίσιμο ρόλο στη μετάβαση από τα CNN επίπεδα στα
#FC επίπεδα των νευρωνικών δικτύων. Ακόμα, κάθε στοιχείο του διανύσματος πλέον θεωρείται ως ένας νευρώνας, άρα το FC layers
#θα δεχτεί 1280 τιμές ως εισόδους σε κάθε νευρώνα που περιέχει το επόμενο layer.
headModel = Flatten(name="flatten")(headModel)

#Δημιουργία ενός FC layer. Το Dense δέχεται ως είσοδο τις 1280 τιμές του διανύσματος που δημιούργησε το Flatten αναγνωρίζοντας
#αυτές ως αρχικούς νευρώνες και δημιουργεί ένα FC layer που αποτελείται απο 128
```

```
367 νέους νευρώνες. Κάθε νέος νευρώνας από τους
368 #128 δέχεται την τιμή που περιέχει κάθε νευρώνας από τους 1280 την οποία
    πολλαπλασιάζει με τυχαία βάρη (weights) αφού αυτά
369 #θα πάρουν τις τελικές τους σωστές τιμές μετά την εκπαίδευση του τελικού μοντέλου.
    Στη συνέχεια, όλα αυτά τα σύνολα
370 #γινομένων προστίθενται μεταξύ τους και δίνουν μία τιμή ως αποτέλεσμα, η οποία
    είναι πιθανόν να έχει αρνητική τιμή πέρα από
371 #μηδενική ή θετική. Επιπλέον, στο προηγούμενο άθροισμα μπορεί να προστεθεί και
    μία τιμή που ονομάζεται πόλωση (bias) η οποία
372 #επίσης θα αλλάξει με την εκπαίδευση. Επειδή οι τιμές θέλουμε να είναι θετικές
    χρησιμοποιείται η συνάρτηση ενεργοποίησης
373 #Rectified Linear Units(Relu) η οποία ανιχνεύει την τιμή που έχει ο νέος νευρώνας
    και αν αυτή είναι αρνητική την μετατρέπει
374 #σε μηδέν. Σε κάθε άλλη περίπτωση η τιμή παραμένει ίδια.
375 headModel = Dense(128, activation="relu")(headModel)
376
377 #Εφαρμογή του "Dropout" στις τιμές που παρήγαγαν οι νευρώνες του προηγούμενου
    layer. Το dropout είναι μία τεχνική τακτοποίησης
378 #(regularization technique) που απορρίπτει τυχαία το 50% των εξόδων των νευρώνων
     από το προηγούμενο layer κατά τη διάρκεια
379 #της εκπαίδευσης του τελικού μοντέλου. Αυτό βοηθά στην αποφυγή της
    υπερεκπαίδευσης (overfitting) που είναι η κατάσταση στην οποία
380 #το μοντέλο έχει καταφέρει να εκπαιδευτεί πολύ καλά πάνω στο dataset που ορίστηκε
    για την εκπαίδευση του, με αποτέλεσμα να μην
381 #ανταποκρίνεται σωστά σε άλλα δεδομένα που του δίνονται. Πιο συγκεκριμένα, με το "
    Dropout" γίνεται αποφυγή της υπερεκπαίδευσης
382 #κάνοντας το δίκτυο να μη βασίζεται μόνο σε συγκεκριμένους νευρώνες αλλά να
    εκπαιδεύεται με πιο γενικευμένα χαρακτηριστικά
383 #(generalized features).
384 headModel = Dropout(0.5)(headModel)
385
386 #Δημιουργία του δεύτερου και τελευταίου FC layer. Το "Dense" αυτή τη φορά δέχεται
    στην είσοδό του τις εξόδους του προηγούμενου
387 #FC layer με τους 128 νευρώνες, των οποίων οι μισές τιμές θα είναι πλέον μηδενικές
    λόγω του "Dropout". Το "Dense" εδώ δημιουργεί
388 #τους δύο τελικούς νευρώνες του δικτύου και αναλόγως την τιμή τους το μοντέλο θα
    έχει πάρει την τελική απόφαση, δηλαδή εάν η
389 #εικόνα που επεξεργάστηκε ανήκει στην κατηγορία "with mask" ή "without mask".
    Όπως και στο προηγούμενο FC layer έτσι και εδώ
390 #κάθε ένας απο τους δύο νευρώνες θα λαμβάνει στην είσοδό του όλες τις εξόδους των
    προηγούμενων 128 νευρώνων. Αυτές οι τιμές
391 #θα πολλαπλασιάζονται και εδώ με κάποια βάρη και έπειτα υπολογίζεται το άθροισμα
    όλων αυτών των γινομένων και ενδεχομένως
392 #η πρόσθεση σε αυτό το άθροισμα μίας τιμής του bias. Έτσι οι δύο νευρώνες θα έχουν
    από μία τιμή ο καθένας, η οποία όμως
393 #δεν θα έχει τη σωστή μορφή οπότε θα πρέπει να την μετατρέψουμε σε μορφή
    πιθανότητας, δηλαδή ανάμεσα στο 0 και το 1. Αυτό
394 #το επιτυγχάνουμε με την συνάρτηση ενεργοποίησης "softmax" η οποία προσαρμόζει
    τις τιμές στο πεδίο που θέλουμε[0-1]. Η
395 #πιθανότητα αυτή θα είναι το τελικό αποτέλεσμα που θα κρίνει το μοντέλο εάν
    κατάφερε να αναγνωρίσει την κατάσταση της εικόνας
396 #που δέχτηκε σαν είσοδο κατά την εκπαίδευση και ποιά είναι η πιθανότητα/το ποσοστό
     που το κατάφερε. Πρέπει να αναφερθεί
397 #πως η "softmax" επηρεάζει και τις δύο τιμές των τελικών νευρώνων, οι οποίες η μία
```

```
397 συμπληρώνει την άλλη και έχουν άθροισμα το 1.
398 #Αναλόγως την πιθανότητα και το ποσοστό επιτυχίας του, το νευρωνικό δίκτυο ενημερώνει τα βάρη και τα bias του σε κάθε
399 #επανάληψη εκπαίδευσης του, με σκοπό να μειώσει όσο περισσότερο μπορεί το ποσοστό λάθους.
400 headModel = Dense(2, activation="softmax")(headModel)
401
402 #Παρακάτω φαίνονται αναλυτικά όλα τα layers του headModel και τα χαρακτηριστικά τους:
403 # KerasTensor(type_spec=TensorSpec(shape=(None, 7, 7, 1280), dtype=tf.float32, name=None), name='out_relu/Relu6:0', description="created by layer 'out_relu'")
404 # KerasTensor(type_spec=TensorSpec(shape=(None, 1, 1, 1280), dtype=tf.float32, name=None), name='average_pooling2d/AvgPool:0', description="created by layer 'average_pooling2d'")
405 # KerasTensor(type_spec=TensorSpec(shape=(None, 1280), dtype=tf.float32, name =None), name='flatten/Reshape:0', description="created by layer 'flatten'")
406 # KerasTensor(type_spec=TensorSpec(shape=(None, 128), dtype=tf.float32, name= None), name='dense/Relu:0', description="created by layer 'dense'")
407 # KerasTensor(type_spec=TensorSpec(shape=(None, 128), dtype=tf.float32, name= None), name='dropout/Identity:0', description="created by layer 'dropout'")
408 # KerasTensor(type_spec=TensorSpec(shape=(None, 2), dtype=tf.float32, name= None), name='dense_1/Softmax:0', description="created by layer 'dense_1'")
409
410 #Ενοποίηση των δύο επιμέρους νευρωνικών δικτύων που δημιουργήθηκαν σε 1(model ), δηλαδή του "baseModel" που είναι το CNN για την δημιουργία
411 #feature map από τις εικόνες του training dataset και του "headModel" που περιέχει τα FC layers, τα οποία είναι υπεύθυνα
412 #για την ταξινόμηση του feature map στις κατηγορίες "mask" ή "without mask". Μετά την σύνδεση του "headModel" στην κορυφή του
413 #"baseModel" το ολοκληρωμένο νευρωνικό δίκτυο που θα εκπαιδεύσουμε θα έχει την ονομασία "model". Η ένωση επιτυγχάνεται με την
414 #εντολή "Model()" που βρίσκεται στο tensorflow.keras.models.
415 model = Model(inputs=baseModel.input, outputs=headModel)
416
417 #Απενεργοποίηση της δυνατότητας εκπαίδευσης όλων των layers του "baseModel" αφού είναι ήδη εκπαιδευμένο και δεν θέλουμε να
418 #υποστεί κάποια αλλαγή στα βάρη ή γενικότερα στις παραμέτρους του. Αντίθετα, το "headModel" είναι αυτό που θα εκπαιδευτεί
419 #εξ' ολοκλήρου και από την αρχή γιατί εκείνο αφορά την ταξινόμηση στις δύο κλάσεις που επιθυμούμε για αυτό το project.
420 for layer in baseModel.layers:
421     layer.trainable = False
422
423 #Δημιουργία του αντικειμένου "adam_optim" το οποίο θα περιέχει μία παραλλαγή του αλγορίθμου βελτιστοποίησης gradient descent
424 #(κάθοδος βασισμένη στην κλίση) που ονομάζεται Adam (Adaptive Moment Estimation). Κατά την εκπαίδευση του μοντέλου, στο
425 #τέλος κάθε επανάληψης υπολογίζεται η συνάρτηση σφάλματος (loss function) συγκρίνοντας το αποτέλεσμα που υπολογίστηκε σε
426 #σχέση με το πραγματικό αποτέλεσμα. Έπειτα υπολογίζονται οι κλίσεις (gradients) για κάθε παράμετρο στο loss function με μία
427 #μέθοδο που ονομάζεται backprobagation. Αυτή η μέθοδος περιλαμβάνει τον υπολογισμό των μερικών παραγώγων ως όλων των παραμέτρων
428 #(weights και bias) πάνω στο loss function χρησιμοποιώντας τον κανόνα της αλυσίδας
```

```
428  (chain rule). Αφού υπολογιστούν το loss
429 #function και τα gradients, τα δέχεται ο Adam σαν δεδομένα και με τη σερά του
    υπολογίζει τις νέες τιμές που θα πάρουν οι
430 #παράμετροι κατά την επόμενη επανάληψη της εκπαίδευσης. Ο Adam συνδυάζει ιδέες
    τόσο από μεθόδους που βασίζονται στην ορμή
431 #(momentum) όσο και από μεθόδους προσαρμοστικού ρυθμού μάθησης (learning rate
    ) για να ενημερώσει τις παραμέτρους ενός μοντέλου.
432 #Επίσης σαν είσοδο ο Adam δέχεται την υπερπαράμετρο "INIT_LR" που έχουμε ορίσει
     στην αρχή του κώδικα και του δείχνει το
433 #μέγεθος του βήματος που πρέπει να κάνει για την διόρθωση του σφάλματος σε κάθε
    επανάληψη κατά την εκπαίδευση.
434 adam_optim = Adam(learning_rate=INIT_LR)
435
436 #Εκτύπωση ενημερωτικού μηνύματος στην οθόνη.
437 print("[ΕΝΗΜΕΡΩΣΗ] Γίνεται μεταγλώττιση του μοντέλου...")
438
439 #Μεταγλώτισση (compile) και προετοιμασία του μοντέλου πριν από την εκπαίδευση.
    Εδώ το μοντέλο δέχεται την τελευταία
440 #ενημέρωσή του, όπου δίνονται πληροφορίες για το είδος της loss function(
    binary_crossentropy) που επιθυμούμε να χρησιμοποιήσει,
441 #τον αλγόριθμο βελτιστοποίησης (Adam) καθώς και το είδος των μετρήσεων (metrics)
    που επιθυμούμε να υπολογίζονται και να
442 #εμφανίζονται στην οθόνη κατά την διαδικασία εκπαίδευσης. Η συνάρτηση απώλειας "
    binary_crossentropy" μετρά την απόκλιση
443 #μεταξύ του προβλεπόμενου αποτελέσματος του μοντέλου και του πραγματικού στόχου
    . Τα "metrics" έχουν οριστεί ως "accuracy"
444 #για να γίνεται ενημέρωση ως προς την ακρίβεια του μοντέλου σε κάθε επανάληψη.
    Επίσης, λόγω του "metrics=["accuracy"]"
445 #αργότερα θα αποθηκευτούν στην μεταβλητή "history" πληροφορίες για κάθε
    επανάληψη εκπαίδευσης σχετικά:
446 #1)Με τις απώλειες κατά την εκπαίδευση με το "training set(loss)"
447 #2)Με την ακρίβεια κατά την εκπαίδευση με το "training set(accuracy)"
448 #3)Με τις απώλειες κατά την εκπαίδευση με το "testing set(val_loss)"
449 #4)Με την ακρίβεια κατά την εκπαίδευση με το "testing set(val_accuracy)".
450
451 model.compile(loss="binary_crossentropy", optimizer=adam_optim, metrics=["
    accuracy"])
452
453
454 #                """Μέρος 4ο - Εκπαίδευση του μοντέλου """
455
456
457 #Εκτύπωση ενημερωτικού μηνύματος στην οθόνη.
458 print("[ΕΝΗΜΕΡΩΣΗ] Η εκπαίδευση του μοντέλου(head μέρος) ξεκίνησε...")
459
460 #Εκπαίδευση του head μέρους του μοντέλου, δηλαδή του "headModel" που περιέχει τα
     FC layers, με την μέθοδο ".fit()". Κάποιες
461 #πληροφορίες σχετικά με το training θα αποθηκευτούν στην μεταβλητή HISTORY
    όπου θα περιέχει κατηγοριες δεδομένων που ορίστηκαν
462 #στο compile του μοντέλου με το "metrics=["accuracy"]". Κατά την κλίση της
    μεθόδου ".fit()" δίνονται τιμές σε κάποιες ιδιότητες
463 #προκειμένου η εκπαίδευση να γίνει με συγκεκριμένο τρόπο.
464 #1) Αρχικά καλείται η μέθοδος ".flow()" πάνω στο αντικείμενο "aug_gen" στην οποία
    εισάγονται τα δεδομένα που θα εκπαιδευτεί το
```

```
#μοντέλο (training set) μαζί με το "batch size" που έχουμε ορίσει στην αρχή του κώδικα και είναι ο αριθμός που θα χωρίσει
#αυτό το σύνολο δεδομένων σε ομάδες για την καλύτερη δυνατή εκπαίδευση. Το "aug_gen" δίνει πληροφορίες στην μέθοδο ".flow" σχετικά
#με τις δυνατότητες τροποποίησης των εικόνων, όπως του είχαν ορισθεί, και η ".flow()" εκτελεί την αύξηση των δεδομένων
#(data augmentation) κάνοντάς τους αυτές τις τροποποιήσης. Έτσι θα δεχτεί το ".fit()" τα νέα "train_images" και "train_labels" set τα οποία
#θα περιέχουν μεγαλύτερη ποικιλία και πλήθος εικόνων.
#2) Η δεύτερη ιδιότητα είναι η "steps_per_epoch" στην οποία δίνουμε το ακέραιο αποτέλεσμα της διαίρεσης του αριθμού όλων των
#εικόνων του training set με το batch size. Αυτό διασφαλίζει ότι ολόκληρο το training set θα χρησιμοποιείται σε κάθε επανάληψη.
#Πιο συγκεκριμένα, αυτή η ιδιότητα καθορίζει τον αριθμό των βημάτων (steps/batches) που πρέπει να υποστούν επεξεργασία σε κάθε επανάληψη.
#3) Στην τρίτη ιδιότητα εισάγονται τα δεδομένα του testing set. Το testing set περιέχει εικόνες που το νευρωνικό δίκτυο δεν
#έχει ξαναδεί κατά την εκπαίδευση, οπότε γίνεται έλεγχος της απόδοσής του καθώς και παρακολούθηση της ικανότητας γενίκευσής
#(generalization ability) του.
#4) Η λογική της τέταρτης ιδιότητας είναι ίδια με αυτή της δεύτερης με την διαφορά ότι εδώ λαμβάνονται υπόψιν τα δεδομένα του
#testing set και όχι του training set. Δηλαδή για την ιδιότητα υπολογίζεται το ακέραιο αποτέλεσμα της διαίρεσης του αριθμού
#όλων των εικόνων του testing set με το batch size. Και εδώ γίνεται διασφάλιση του ότι θα χρησιμοποιηθούν όλα τα δεδομένα
#του testing set σε κάθε επανάληψη.
#5) Στην πέμπτη και τελευταία ιδιότητα δίνεται η τιμή που περιέχει η υπερπαράμετρος "EPOCHS" που ορίστηκε στην αρχή του κώδικα.
#Αυτό σημαίνει ότι το μοντέλο θα επαναλάβει τη διαδικασία εκπαίδευσης, πάνω σε ολόκληρο το training set, τόσες φορές όσο η
#τιμή των epochs (εποχές). Κάθε εποχή αποτελείται από βήματα που ορίζονται από τα "step_per_epoch" για εκπαίδευση (training) και
#"validation_steps" για έλεγχο (testing).
HISTORY = model.fit( aug_gen.flow(train_images, train_labels, batch_size=BS), steps_per_epoch=len(train_images) // BS, validation_data=(test_images, test_labels), validation_steps=len(test_images) // BS, epochs=EPOCHS)

#Εκτύπωση ενημερωτικού μηνύματος στην οθόνη.
print("[ΕΝΗΜΕΡΩΣΗ] Αποθήκευση του μοντέλου ανίχνευσης μάσκας στον φάκελο...")

#Αποθήκευση του μοντέλου στον υπολογιστή σε μορφή .h5 αρχείου. Αυτό γίνεται για να μην χρειαστεί να εκπαιδευτεί ξανά
#απο την αρχή το μοντέλο, κάθε φορά που το χρειαζόμαστε, κάτι το οποίο είναι συνήθως χρονοβόρο. Επίσης το αποθηκευμένο μοντέλο
#μπορεί να κληθεί μέσα από κάποιο άλλο πρόγραμμα και να χρησιμοποιηθεί όποτε είναι αναγκαίο με την εντολή "load_model" από
#τη βιβλιοθήκη tensorflow.keras.models.
model.save(f"{folder_of_model}/mask_detection_model.h5")


#                """Μέρος 5ο - Μετατροπή του μοντέλου απο .h5 σε .tflite """

```

```
498
499 #Εκτύπωση ενημερωτικού μηνύματος στην οθόνη.
500 print("[ΕΝΗΜΕΡΩΣΗ] Μετατροπή του μοντέλου απο .h5 σε μορφή .tflite...")
501
502 #Σε αυτό το σημείο ξεκινά η διαδικασία μετατροπής του μοντέλου που δημιουργήθηκε
    σε πιο ελαφρυά έκδοση, δηλαδή σε μορφή
503 #.tflite. Τα μοντέλα .tflite φτιάχνονται για να λειρουργούν πιο γρήγορα και πιο
    αποτελεσματικά στα λειτουργικά των edge
504 #devices. Γι' αυτό ένα μοντέλο .tflite εάν χρησιμοποιηθεί π.χ. σε υπολογιστή με
    Windows 10, θα αργεί περισσότερο τον κώδικα
505 #σε αντίθεση με ένα .h5 μοντέλο, παρόλο που το πρώτο σαν αρχείο είναι πολύ πιο
    ελαφρύ. Απο την άλλη, ένα .tflite μοντέλο
506 #εκτελεί τον κώδικα πολύ πιο γρήγορα σε ένα raspberry pi απ' ότι ένα .h5 μοντέλο.
    Επίσης πρέπει να αναφερθεί πως ένα μοντέλο
507 #δεν μπορεί να δημιουργηθεί κατευθείαν σε μορφή .tflte αλλά μπορεί μόνο να
    μετατραπεί σε αυτό αφού πρώτα δημιουργηθεί το
508 #μοντέλο σε μορφή .h5 ή μορφή φακέλου. Η βασική διαφορά της μορφής .h5  από τη
    μορφή φακέλου είναι πως στην πρώτη όλα τα
509 #αρχεία που δημιουργούνται και αποτελούν το μοντέλο τοποθετούνται μέσα σε ένα
    μόνο αρχείο με κατάληξη .h5, ενώ στην δέυτερη
510 #μορφή όλα αυτά τα αρχεία τοποθετούνται σε έναν φάκελο. Έτσι η μορφή .h5 είναι πιο
     βολική στην περίπτωση του συγκεκριμένου
511 #project.
512 #Αρχικά γίνεται φόρτωση του μοντέλου που αποθηκέυτηκε στη μεταβλητή "
    loaded_model" για χρήση του στα επόμενα βήματα και τη μετατροπή του
513 #σε μορφή .tflite. Η φόρτωση γίνεται καλώντας την μέθοδο "load_model" της
    βιβλιοθήκης tensorflow.keras.models και δίνοντάς
514 #της ως όρισμα την διεύθυνση της τοποθεσίας που έχει αποθηκευτεί το αρχείο του
    μοντέλου στον υπολογιστή.
515 loaded_model = tensorflow.keras.models.load_model(f"{folder_of_model}/
    mask_detection_model.h5")
516
517 #Δημιουργία της μεταβλητής/του αντικειμένου "converter" που θα περιέχει όλες τις
    πληροφορίες σχετικά με την μετατροπή.
518 #Συγκεκριμένα το "converter" ενημερώνεται σχετικά με το μοντέλο που θα γίνει η
    επεξεργασία, τα βάρη του κ.λπ. Η βιβλιοθήκη
519 #που χρησιμοποιείται είναι η "tensorflow.lite" που περιέχει διάφορες συναρτήσεις
    χρήσιμες, είτε για την δημιουργία ενός .tflite
520 #αρχείου, είτε για την επεξεργασία του.
521 converter = tensorflow.lite.TFLiteConverter.from_keras_model(loaded_model)
522
523 #Το "converter.optimizations" ενεργοποιεί το optimization του νέου μοντέλου, που
    είναι μία διαδικασία πολύ σημαντική και
524 #χρήσιμη. Ενώ είναι προεραιτική εντολή, είναι πρακτικά απαραίτητη αφού μειώνει το
    μέγεθος του μοντέλου αρκετά σε σχέση
525 #με το αρχικό μοντέλο .h5 και το κάνει πιο γρήγορο. Χωρίς αυτή την εντολή, το νέο
    μοντέλο πάλι θα ήταν πιο ελαφρύ, αλλά
526 #όχι τόσο πολύ.
527 converter.optimizations = [tensorflow.lite.Optimize.DEFAULT]
528
529 #Μετατροπή του μοντέλου από την μορφή .h5 σε μορφή .tflite συμβατή με το raspberry
     pi χρησιμοποιώντας την μέθοδο .convert()
530 #πάνω στο αντικείμενο converter. Όλες οι πληροφορίες για το νέο μοντέλο
    αποθηκεύονται προσωρινά στη μεταβλητή "tflite_model".
```

```
tflite_model = converter.convert()

#Αποθήκευση του νέου μοντέλου με όνομα "mask_detection_model_optim.tflite" σε μορφή αρχείου, στον φάκελο που έχει δημιουργηθεί
#ήδη για την συγκεκριμένη εκτέλεση του κώδικα.
open(f"{folder_of_model}/mask_detection_model_optim.tflite", "wb").write(tflite_model)


#                """Μέρος 6ο - Αξιολόγηση του μοντέλου """


#Εκτύπωση ενημερωτικού μηνύματος στην οθόνη.
print("[ΕΝΗΜΕΡΩΣΗ] Αξιολόγηση του νευρωνικού δικτύου...")

#Πρόβλεψη αποτελεσμάτων του μοντέλου που εκπαιδεύτηκε χρησιμοποιώντας το testing set. Η μέθοδος predict() εισάγει τις εικόνες
#του testing set χωρίζοντας αυτες σε ίσα Batches και αποθηκεύει στη μεταβλητή "predictions" τις προβλέψεις ως πιθανότητες.
predictions = model.predict(test_images, batch_size=BS)

#Δημιουργία array, που θα περιέχει τα labels του testing set, σε μορφή τέτοια έτσι ώστε να την επεξεργαστεί παρακάτω
#η μέθοδος roc_curve χωρίς προβλήματα συμβατότητας.
test_labels_binary = np.argmax(test_labels, axis=1)

#Δημιουργία array, που θα περιέχει τις προβλέψεις που έγιναν για το μοντέλο πάνω στις εικόνες του testing set, σε μορφή
#τέτοια έτσι ώστε να το επεξεργαστεί παρακάτω η μέθοδος roc_curve χωρίς προβλήματα συμβατότητας.
predictions_binary = np.argmax(predictions, axis=1)

#Υπολογισμός των FPR (False Positive Rate), TPR (True Positive Rate), και thresholds (τα αντίστοιχα κατώφλια τους) χρησιμοποιώντας
#την μέθοδο "roc_curve()" της βιβλιοθήκης NumPy. Τα ορίσματα που δέχεται η μέθοδος είναι οι μεταβλητές "test_labels_binary" και
#"predictions_binary" που δημιουργήθηκαν παραπάνω και περιέχουν τις εικόνες του testing set καθώς και τα labels τους σε μορφή
#binary. Τα FPR, TPR και τα thresholds τους θα δημιουργήσουν παρακάτω στον κώδικα το διάγραμμα ROC (Receiver Operating
#Characteristic).
fpr, tpr, thresholds = roc_curve(test_labels_binary, predictions_binary)

#Υπολογισμός του AUC (Area Under the ROC Curve), που είναι η περιοχή κάτω απο την ROC curve, από τα "fpr" και "tpr" χρησιμοποιώντας την μέθοδο "auc()" της
#βιβλιοθήκης scikit-learn. Ο αριθμός AUC δείχνει την απόδοση του μοντέλου.
auc_score = auc(fpr, tpr)

#Η μέθοδος .argmax της βιβλιοθήκης numpy, μετατρέπει τις προβλεπόμενες πιθανότητες, του "predictions" σε ετικέτες κλάσεων
#(class labels) επιλέγοντας τον δείκτη (index) με την υψηλότερη πιθανότητα για κάθε πρόβλεψη. Ο άξονας που αναζητά η μέθοδος
#τον δείκτη της υψηλότερης πιθανότητας είναι ο νούμερο ένα, δηλαδή οι γραμμές (row) όπου κάθε γραμμή είναι και μία πρόβλεψη.
```

```
570 #Για παράδειγμα, έστω ότι το "predictions" περιέχει τα δεδομένα [0.3 0.7] στην
    πρώτη του γραμμή. Αυτό σημαίνει πως για την πρώτη
571 #εικόνα του πρώτου batch του "test_images" έγινε η πρόβλεψη με αποτέλεσμα 0.3(30
    %) πιθανότητα για το "with_mask" και 0,7(70%) πιθανότητα
572 #για το "without_mask". Έτσι το argmax θα επιστρέψει τη θέση του δείκτη με τη
    μεγαλύτερη πιθανότητα που στην προκειμένη
573 #περίπτωση θα ήταν ο 1 και όχι ο 0, αφού στη γλώσσα προγραμματισμού Python η
    πρώτη στήλη αριθμείται ως μηδέν. Επίσης
574 #το "predictions" μετά την εκτέλεση της εντολής "np.argmax(predictions, axis=1)"
    μετατρέπεται από ένα δισδιάστατο διάνυσμα (n γραμμών
575 #και 2 στηλών), σε μονοδιάστατο (1 γραμμής και n στηλών).
576 predictions = np.argmax(predictions, axis=1)
577
578 #Εμφάνιση στην οθόνη του τερματικού μίας αναφοράς ταξινόμησης (classification
    report) που παρέχει μετρήσεις όπως η
579 #ανάκληση (recall), η βαθμολογία F1 (F1-score), η ακρίβεια (precision), η ακρίβεια (
    accuracy) καθώς και έναν μέσο όρο σε όλες
580 #τις κλάσεις. Για τη διαδικασία παραγωγής όλων των παραπάνω πληροφοριών είναι
    υπεύθυνη η εντολή "classification_report()"
581 #της βιβλιοθήκης sklearn.metrics. Αυτή η εντολή δέχεται αρχικά ως δεδομένα τις
    πραγματικές τιμές των labels που αντιστοιχούν
582 #στο "test_images" σετ. Τα δεδομένα αυτά βρίσκονται θεωρητικά στην μεταβλητή "
    test_labels" αλλά επειδή αυτά τα δεδομένα είναι της μορφής
583 #one-hot encoding, θα πρέπει πρώτα να μετατραπούν στην μορφή μονοδιάστατου
    διανύσματος. Για αυτήν τη διαδικασία χρησιμοποιείται
584 #η μέθοδος .argmax η οποία συμπεριφέρεται με τον ίδιο τρόπο που εκτελέστηκε και
    για το "predictions" προηγουμένως. Πιο συγκεκριμένα,
585 #η .argmax θα ελέγξει κάθε γραμμή του "test_labels" array αφού το "axis" έχει
    ορισθεί ως 1 και θα επιστρέψει τον δείκτη με τη μεγαλύτερη
586 #τιμή. Παίρνοντας ως παράδειγμα μία σειρά του "test_labels" που έχει τη μορφή [0. 1
    .] αυτή θα ελεγχθεί και θα επιστρέψει την τιμή 1
587 #στο νέο μονοδιάστατο διάνυσμα του "test_labels", αφού μεταξύ του 0 και 1,
    μεγαλύτερο είναι το 1 που βρίσκεται στη στήλη 1. Ακόμα το
588 #"classification_report" δέχεται ως δεδομένα τις τιμές "predictions" που
    προβλέφθηκαν και προορίζονται για σύγκριση με τις πραγματικές
589 #τιμές του "test_labels". Τέλος ενεργοποιείται η ιδιότητα "target_names" στην οποία
    δίνεται η τιμή lb.classes_ που επιστρέφει το array
590 #με τις αλφαριθμητικές ονομασίες των δύο labels (with_mask και without_mask).
    Αυτές δίνονται στο "classification_report" έτσι
591 #ώστε αυτό να προβάλλει τις πληροφορίες με περισσότερη αναγνωσιμότητα.
    Παρακάτω αναλύονται τα metrics που θα εκτυπωθούν:
592 #1) ακρίβεια (precision): Υπολογίζεται ως η αναλογία μεταξύ του αριθμού των
    Θετικών δειγμάτων (π.χ. with_mask) που ταξινομήθηκαν
593 #σωστά προς τον συνολικό αριθμό των δειγμάτων που ταξινομήθηκαν ως Θετικά (είτε
    σωστά είτε λανθασμένα) και μετρά την ακρίβεια
594 #του μοντέλου στην ταξινόμηση ενός δείγματος ως θετικού. Έτσι, προβάλλει πόσο
    αξιόπιστο είναι το μοντέλο στην ταξινόμηση
595 #των δειγμάτων ως Θετικών.
596 #2) ανάκληση (recall):  Υπολογίζεται ως η αναλογία μεταξύ του αριθμού των Θετικών
    δειγμάτων (π.χ. with_mask) που ταξινομήθηκαν
597 #σωστά ως Θετικά προς τον συνολικό αριθμό των Θετικών δειγμάτων. Η ανάκληση
    μετράει την ικανότητα του μοντέλου να ανιχνεύει
598 #Θετικά δείγματα. Όταν η ανάκληση είναι υψηλή τότε το μοντέλο μπορεί να
    ταξινομήσει σωστά όλα τα θετικά δείγματα ως Θετικά
```

```
#και θεωρείται αξιόπιστο ως προς την ικανότητά του να ανιχνεύει θετικά δείγματα.
#3) βαθμολογία F1 (F1-score): Μετρά την ακρίβεια ενός μοντέλου συνδυάζοντας τις τιμές του recall και του precision του που
#έχουν ήδη υπολογιστεί. Ειδικότερα, παρέχει μία ισορροπημένη μέτρηση της απόδοσης του μοντέλου, ειδικά όταν υπάρχει
#ανισορροπία μεταξύ του αριθμού των δειγμάτων σε διαφορετικά classes.
#4) ακρίβεια (accuracy): Υπολογίζεται ως η αναλογία μεταξύ του αριθμού των σωστών προβλέψεων (θετικών και αρνητικών σωστών
#δειγμάτων) προς τον συνολικό αριθμό των προβλέψεων και περιγράφει την απόδοση του μοντέλου σε όλες τις κλάσεις. Γενικότερα
#μετρά τη συνολική ορθότητα των προβλέψεων του μοντέλου.
report = classification_report(test_labels.argmax(axis=1), predictions, target_names=lb.classes_)
print(report)

#Δημιουργία ενός νέου text αρχείου με όνομα "classification_report.txt" και άνοιγμά του για εγγραφή πληροφοριών μέσα
#σε αυτό.
with open(f'{folder_of_model}/classification_report.txt', 'w') as file:
    #Αποθήκευση των πληροφοριών που παρήχθησαν απο το classification report σε αρχείο .txt .
    file.write(report)
    #Κλείσιμο του αρχείου αφού τελείωσε η επεξεργασία του.
    file.close()

#                """Μέρος 7ο - Δημιουργία διαγραμμάτων loss, accuracy και ROC"""

#Εκτύπωση ενημερωτικού μηνύματος στην οθόνη.
print("[ΕΝΗΜΕΡΩΣΗ] Η γραφική απεικόνιση των μετρήσεων ξεκίνησε...")

#Υπολογισμός των τελικών τιμών των "loss" και "accuracy" για την γραφική απεικόνισή τους πάνω στα διαγράμματα που θα φτιαχτούν.
#Συνολικά θα δημιουργηθούν τέσσερις μεταβλητές αφου υπάρχουν δύο τελικές τιμές για το "loss", μια για το training και μία για το
#testing/validating, καθώς και δύο για το accuracy, μια για το training και μια για το testing/validating. Οι τιμές αυτές
#υπολογίζονται χρησιμοποιώντας το αντικείμενο "HISTORY" που δημιουργήθηκε κατά την εκπαίδευση του μοντέλου και εκτελώντας
#πάνω του την μέθοδο ".history" με τα αντίστοιχα ορίσματα (loss, accuracy, val_loss, val_accuracy). Επίσης δίπλα απο κάθε
#όρισμα υπάρχει το [-1] που δηλώνει ποια από όλες τις τιμές του κάθε metric θέλουμε να υπολογίσουμε. Επειδή το κάθε metric
#έχει τη μορφή array, για ένα array το [-1] σημαίνει η τελευταία τιμή του στη γλώσσα προγραμματισμού Python.
final_train_loss = HISTORY.history["loss"][-1]
final_train_acc = HISTORY.history["accuracy"][-1]
final_val_loss = HISTORY.history["val_loss"][-1]
final_val_acc = HISTORY.history["val_accuracy"][-1]

#Διαδικασία δημιουργίας του 1ου διαγράμματος που θα περιέχει τα training loss και testing/validation loss.

#Επιλογή του στυλ (style) που θα έχει το διάγραμμα, δηλαδή η φωτεινότητά του, το χρώμα στο παρασκήνιο, τα χρώματα των
```

```
637 #γραμμών κ.λπ. Εδώ επιλέχθηκε το στυλ bmh (Bayesian Methods for Hackers), διότι
    παρουσιάζει τα δεδομένα στα γραφήματα με
638 #μεγαλύτερη ευκρίνεια και πιο ωραίο τρόπο. Όλες οι μέθοδοι που θα χρειαστούν για τη
     δημιουργία του διαγράμματος βρίσκονται
639 #στη βιβλιοθήκη matplotlib ή αλλιώς plt όπως ονομάστηκε για συντομία. Οπότε για το
     στυλ χρησιμοποιήθηκε η μέθοδος .style()
640 #με όρισμα το bmh στυλ.
641 plt.style.use("bmh")
642
643 #Ορισμός του μεγέθους του πλαισίου (figure) που θα περιέχει το διάγραμμα και
    συγκεκριμένα με τη μέθοδο .figure() και ορίσματα 8,6. Αυτά τα ορίσματα
644 #μετριούνται σε ίντσες και επιλέχθηκαν μετά απο διάφορες δοκιμές διότι εμφάνιζαν
    καλύτερα τα δεδομένα και τους άξονες.
645 plt.figure(figsize=(8, 6))
646
647 #Εισαγωγή των δεδομένων της μέτρησης "loss" (που αφορά το training) στον άξονα y
    του διαγράμματος και στις τιμές του άξονα x
648 #τοποθετούνται οι αριθμοί 1 έως 10, δηλαδή κάθε εποχή/επανάληψη εκπαίδευσης.
    Έτσι για κάθε εποχή απεικονίζεται η αντίστοιχη
649 #τιμή του "loss". Επιπλέον δίνεται μία ονομασία για τα δεδομένα αυτά με την ιδιότητα
    label και όνομα το training loss.
650 plt.plot(np.arange(1, EPOCHS+1), HISTORY.history["loss"], label="training loss"
    )
651
652 #Εδώ γίνεται ακριβώς η ίδια διαδικασία με την προηγούμενη εντολή, με τη διαφορά
    ότι προσθέτουμε στον άξονα y του διαγράμματος
653 #τα δεδομένα του metric "val_loss" (αφορούν τα testing δεδομένα) και τους δίνουμε
    την ονομασία validation loss.
654 plt.plot(np.arange(1, EPOCHS+1), HISTORY.history["val_loss"], label="
    validation loss")
655
656 #Με τη μέθοδο .title() δίνεται ο τίτλος του διαγράμματος ο οποίος θα εμφανίζεται
    πάνω από το διάγραμμα.
657 plt.title("Training and Validation Loss")
658
659 #Η μέθοδος .xlabel() θέτει τον τίτλο του x άξονα που θα εμφανίζεται κάτω από αυτόν.
660 plt.xlabel("Epoch #")
661
662 #Η μέθοδος .ylabel() αντίστοιχα θέτει τον τίτλο του y άξονα που θα εμφανίζεται
    αριστερά του.
663 plt.ylabel("Loss")
664
665 #Δημιουργία με τη μέθοδο .legend() του υπομνήματος που θα περιέχει τις ονομασίες
    των δύο διαφορετικών γραμμών για το
666 #"loss" (training και validation) καθώς και το χρώμα που αντιπροσωπεύει κάθε
    γραμμή. Το υπόμνημα με την ιδιότητα "loc"
667 #τοποθετείται πάνω δεξιά (upper right) στο διάγραμμα.
668 plt.legend(loc="upper right")
669
670 #Με τη μέθοδο .annotate επισημαίνεται πάνω στο διάγραμμα το μήνυμα "Final Train
    Loss:" μαζί με την αριθμητική τιμή της
671 #μεταβλητής "final_train_loss" με τέσσερα δεκαδικά ψηφία (.4f). Το κείμενο
    επισήμανσης αυτό τοποθετείται στις συντεταγμένες όπου
672 #το x είναι ίσο με την τελευταία εποχή της εκπαίδευσης και το y ίσο με την τιμή της
```

```
μεταβλητής "final_train_loss". Επίσης
#με το "textcoords" και την τιμή του "offset points" μετατοπίζεται η επισήμανση από το σημείο των συντεταγμένων που δώσαμε
#κατά τόσο όσο ορίζεται στην ιδιότητα "xytext". Η ιδιότητα "ha" παίρνει την τιμή "right" η οποία ορίζει τη στοίχιση του κειμένου
#της επισήμανσης ως δεξιά στοιχισμένο και η "color" δέχεται την τιμή "blue" που κάνει το κείμενο μπλε για να ταιριάζει με το
#χρώμα της γραμμής των training loss τιμών.
plt.annotate(f"Final Train Loss: {final_train_loss:.4f}", (EPOCHS, final_train_loss), textcoords="offset points", xytext=(-10, 40), ha='right', color='blue')

#Εδώ γίνεται επισήμανση πάνω στο διάγραμμα σχετικά με την τελευταία τιμή του validation loss (final_val_loss) ακριβώς με τον
#ίδιο τρόπο όπως και για την τελική τιμή του training loss.
plt.annotate(f"Final Val Loss: {final_val_loss:.4f}", (EPOCHS, final_val_loss), textcoords="offset points", xytext=(-10, 20), ha='right', color='red')

#Η μέθοδος .margind() θέτει τα περιθώρια του διαγράμματος ως 0 και για τους δύο άξονες (x και y). Αυτό διασφαλίζει ότι οι
#γραμμές με τα δεδομένα φτάνουν μέχρι τις άκρες του διαγράμματος, εκμεταλλεύοντας έτσι τον διαθέσιμο χώρο πάνω στο διάγραμμα.
plt.margins(x=0, y=0)

#Μετατροπή των τιμών του x άξονα σε ακέραιους αριθμούς, διότι οι τιμές είναι καλό να μη φαίνονται δεκαδικές αφού οι επαναλήψεις
#αντιπροσωπεύουν ακέραιους αριθμούς. Επίσης δεν είναι ευανάγνωστο να έχουμε όλες τις τιμές των επαναλήψεων στον άξονα x
#αν αυτές είναι πολλές π.χ. 40 οπότε στη μεταβλητή "tick_locations" αποθηκεύονται οι τιμές του x άξονα ανά 5, δηλαδή 0,5,10,15
#κ.λπ.
tick_locations = np.arange(0, EPOCHS+1, 5)

#Επειδή στην προηγούμενη εντολή ορίστηκαν οι τιμές των επαναλήψεων να ξεκινάνε απο το 0 γίνεται τροποποίηση της πρώτης τιμής
#του άξονα και ορίζεται το 1 ως τιμή έναρξης.
tick_locations[0] = 1

#Τοποθέτηση των παραπάνω δεδομένων στον πραγματικό άξονα του διαγράμματος και ενημέρωσή του.
plt.xticks(tick_locations, tick_locations)

#Αποθήκευση του διαγράμματος με την μέθοδο .savefig() στον επιθυμητό φάκελο σε μορφή εικόνας τύπου .png, με ανάλυση 300dpi
#(dots per inch) εξασφαλίζοντας έτσι υψηλή ανάλυση. Η ανάλυση ορίστηκε με την ιδιότητα dpi. Επίσης η ιδιότητα bbox_inches
#με όρισμα το tight δηλώνει ότι η αποθηκευμένη εικόνα θα περιλαμβάνει μόνο την πραγματική περιοχή του διαγράμματος χωρίς
#περιττά κενά.
plt.savefig(f"{folder_of_model}/loss_plot.png", dpi=300, bbox_inches="tight")

#Διαδικασία δημιουργίας του 2ου διαγράμματος που θα περιέχει τα training accuracy και testing/validation accuracy.
#Για τη δημιουργία του 2ου διαγράμματος ακολουθείται ακριβώς η ίδια διαδικασία με το πρώτο διάγραμμα αλλάζοντας μόνο τα
```

```
#δεδομένα του διαγράμματος και το όνομα του αρχείου που θα αποθηκευτεί το διάγραμμα.
plt.style.use("bmh")
plt.figure(figsize=(8, 6))
plt.plot(np.arange(1, EPOCHS+1), HISTORY.history["accuracy"], label="training accuracy")
plt.plot(np.arange(1, EPOCHS+1), HISTORY.history["val_accuracy"], label="validation accuracy")
plt.title("Training and Validation Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Accuracy")
plt.legend(loc="lower right")
plt.annotate(f"Final Train Acc: {final_train_acc:.4f}", (EPOCHS, final_train_acc), textcoords="offset points", xytext=(-10, -40), ha='right', color='blue')
plt.annotate(f"Final Val Acc: {final_val_acc:.4f}", (EPOCHS, final_val_acc), textcoords="offset points", xytext=(-10, -20), ha='right', color='red')
plt.margins(x=0, y=0)
tick_locations = np.arange(0, EPOCHS+1, 5)
tick_locations[0] = 1
plt.xticks(tick_locations, tick_locations)
plt.savefig(f"{folder_of_model}/accuracy_plot.png", dpi=300, bbox_inches="tight")

#Διαγράμματα με dark background. Τα παρακάτω δύο διαγράμματα (3ο και 4ο) είναι ακριβώς ίδια με το 1ο και 2ο, με τη μόνη
#διαφορά στο στυλ. Εδώ, συγκεκριμένα, το στυλ είναι σκοτεινό και αυτό μπορεί να είναι πιο φιλικό στο μάτι κάποιων χρηστών.
#Η εναλλακτική αυτή επιλογή έγινε καθαρά για θέματα που αφορούν την καλύτερη και πιο άνετη αναγνωσιμότητα των αποτελεσμάτων
#από τον χρήστη.

#Διαδικασία δημιουργίας του 3ου διαγράμματος που θα περιέχει τα training loss και testing/validation loss.
#Για τη δημιουργία του 3ου διαγράμματος ακολουθείται ακριβώς η ίδια διαδικασία με το πρώτο διάγραμμα αλλάζοντας μόνο το
#στυλ του διαγράμματος, τα δεδομένα του και το όνομα του αρχείου που θα αποθηκευτεί.
plt.style.use("dark_background")
plt.figure(figsize=(8, 6))
plt.plot(np.arange(1, EPOCHS+1), HISTORY.history["loss"], label="training loss", color='blue')
plt.plot(np.arange(1, EPOCHS+1), HISTORY.history["val_loss"], label="validation loss", color='yellow')
plt.title("Training and Validation Loss")
plt.xlabel("Epoch #")
plt.ylabel("Loss")
plt.legend(loc="upper right")
plt.annotate(f"Final Train Loss: {final_train_loss:.4f}", (EPOCHS, final_train_loss), textcoords="offset points", xytext=(-10, 40), ha='right', color='blue')
plt.annotate(f"Final Val Loss: {final_val_loss:.4f}", (EPOCHS, final_val_loss), textcoords="offset points", xytext=(-10, 20), ha='right', color='yellow')
plt.margins(x=0, y=0)
tick_locations = np.arange(0, EPOCHS+1, 5)
tick_locations[0] = 1
```

```
746 plt.xticks(tick_locations, tick_locations)
747 plt.savefig(f"{folder_of_model}/loss_plot_dark_background.png", dpi=300,
    bbox_inches="tight")
748
749 #Διαδικασία δημιουργίας του 4ου διαγράμματος που θα περιέχει τα training accuracy
    και testing/validation accuracy.
750 #Για τη δημιουργία του 4ου διαγράμματος ακολουθείται ακριβώς η ίδια διαδικασία με
    το πρώτο διάγραμμα αλλάζοντας μόνο το
751 #στυλ του διαγράμματος, τα δεδομένα του και το όνομα του αρχείου που θα
    αποθηκευτεί.
752 plt.style.use("dark_background")
753 plt.figure(figsize=(8, 6))
754 plt.plot(np.arange(1, EPOCHS+1), HISTORY.history["accuracy"], label="training
     accuracy", color='blue')
755 plt.plot(np.arange(1, EPOCHS+1), HISTORY.history["val_accuracy"], label="
    validation accuracy", color='yellow')
756 plt.title("Training and Validation Accuracy")
757 plt.xlabel("Epoch #")
758 plt.ylabel("Accuracy")
759 plt.legend(loc="lower right")
760 plt.annotate(f"Final Train Acc: {final_train_acc:.4f}", (EPOCHS, final_train_acc),
    textcoords="offset points", xytext=(-10, -40), ha='right', color='blue')
761 plt.annotate(f"Final Val Acc: {final_val_acc:.4f}", (EPOCHS, final_val_acc),
    textcoords="offset points", xytext=(-10, -20), ha='right', color='yellow')
762 plt.margins(x=0, y=0)
763 tick_locations = np.arange(0, EPOCHS+1, 5)
764 tick_locations[0] = 1
765 plt.xticks(tick_locations, tick_locations)
766 plt.savefig(f"{folder_of_model}/accuracy_plot_dark_background.png", dpi=300,
    bbox_inches="tight")
767
768 #Δημιουργία διαγράμματος σχετικά με την ROC curve.
769
770 #Για τη δημιουργία αυτού του διαγράμματος ακολουθείται η ίδια διαδικασία με το 1ο
    διάγραμμα αλλάζοντας μόνο τα δεδομένα
771 #που θα εμφανιστούν. Επιπλέον γίνεται ορισμός των τιμών που θα εμφανίζει ο άξονας
    x και y με τις μεθόδους xlim και ylim
772 #αντίστοιχα.
773 plt.style.use("bmh")
774 plt.figure(figsize=(8, 6))
775 plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % auc_score)
776 plt.plot([0, 1], [0, 1], 'k--')
777 plt.xlim([0.0, 1.0])
778 plt.ylim([0.0, 1.05])
779 plt.xlabel('False Positive Rate')
780 plt.ylabel('True Positive Rate')
781 plt.title('Receiver Operating Characteristic')
782 plt.legend(loc="lower right")
783 plt.margins(x=0, y=0)
784 plt.savefig(f"{folder_of_model}/ROC_plot.png", dpi=300, bbox_inches="tight")
785
786 #Εκτύπωση ενημερωτικού μηνύματος στην οθόνη.
787 print("[ΕΝΗΜΕΡΩΣΗ] Τέλος εκπαίδευσης του μοντέλου. Τερματισμός
    προγράμματος...")
```