Πρόγραμμα Μεταπτυχιακών Σπουδών στις Σύγχρονες Ηλεκτρονικές Τεχνολογίες

Τμήμα Φυσικής

Σχολή Θετικών Επιστημών

Πανεπιστήμιο Ιωαννίνων

ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ (Μ.Δ.Ε)

Εφαρμογές τεχνητής νοημοσύνης με υπολογιστική όραση για ιατρικές διαγνώσεις

Αρβανιτόπουλος Νικηφόρος
Αριθμός μητρώου: 835

Επιβλέπων καθηγητής: Κόκκας Παναγιώτης

Ιωάννινα
2024

Περίληψη

Στην παρούσα εργασία μελετάται η λειτουργία και λογική σχεδιασμού βαθιών νευρωνικών δικτύων, με σκοπό την ταξινόμηση εικόνων, οι οποίες αντιστοιχούν σε ιατρικά δεδομένα όπως ακτινογραφίες, υπερήχους, φωτογραφίες μορφωμάτων κ.λ.π. Προς επίτευξη αυτού, ελήφθησαν δημόσιες ιατρικές βάσεις δεδομένων από το διαδίκτυο, οι οποίες χρησιμοποιήθηκαν για την εκπαίδευση των δικτύων. Τα δίκτυα προγραμματίστηκαν σε γλώσσα Python με τη χρήση της βιβλιοθήκης Tensorflow, η οποία έχει σχεδιαστεί για μηχανική μάθηση, ενώ ταυτόχρονα προσφέρει ενσωματωμένα το Keras API, εργαλείο που απλοποιεί και διευκολύνει τη διαδικασία για τον χρήστη. Συνολικά δημιουργήθηκαν πέντε νευρωνικά δίκτυα, τα οποία επικεντρώνονται σε διαφορετικές νόσους και παθήσεις και έχουν μελετηθεί ως προς διάφορες παραμέτρους, με τελικό σκοπό τη μεγιστοποίηση της ακρίβειας που αποδίδουν.

Πίνακας Περιεχομένων

Κεφάλαιο 1

Εισαγωγή στη Μηχανική και Βαθιά Μάθηση

1.1 Στοιχεία Μηχανικής Μάθησης

1.1.1 Ορισμός

Από την στιγμή της πρώτης χρήσης του όρου "τεχνητή νοημοσύνη", υπήρξε πάντα μια διαρκής προσπάθεια για την δημιουργία μιας πραγματικά έξυπνης μηχανής, όπως αυτή περιγράφεται από τον Alan Turing στη δημοσίευση του, "Computing Machinery and Intelligence".[1] Στο πλαίσιο της προσπάθειας αυτής, έγιναν πολλές απόπειρες προσέγγισης μιας μηχανής η οποία μπορεί να πλησιάσει την ανθρώπινη ευφυΐα. Μια από τις προσεγγίσεις αυτές, και ίσως μια από τις πιο πρακτικές, είναι η μηχανική μάθηση.

Η πρώτη χρήση του όρου ήταν για να περιγράψει τη διαδικασία εκμάθησης ενός υπολογιστή να παίζει μια παρτίδα ντάμας καλύτερα από τον ίδιο τον προγραμματιστή με τη χρήση αλγορίθμων.[2] Με βάση αυτό, μπορεί κάποιος να γενικεύσει, περιγράφοντας τη μηχανική μάθηση ως "την ανάπτυξη και χρήση αλγορίθμων, εκπαιδευμένων σε διαθέσιμα δεδομένα, οι οποίοι μπορούν να εκτελέσουν εργασίες απουσίας ρητού προγραμματισμού". Ο σκοπός λοιπόν, είναι το τελικό πρόγραμμα να μπορεί να εκτελέσει μια διαδικασία χωρίς ανθρώπινη επιτήρηση. Για να επιτευχθεί αυτό, δίνεται ιδιαίτερη προσοχή και ενδιαφέρον στην δυνατότητα του αλγορίθμου να μπορεί να μεγιστοποιήσει την απόδοση του σε δεδομένα που δεν έχει επεξεργαστεί κατά τη διάρκεια της εκπαίδευσής του. Η ιδιότητα αυτή, στα πλαίσια του τομέα της μηχανικής μάθησης, ονομάζεται **γενίκευση**[3] και αποτελεί βασικό πυλώνα αυτού.

1.1.2 Βασική Θεωρία

Σημαντική για την κατανόηση της μηχανικής μάθησης είναι η εξοικείωση με τη θεωρία που την πλαισιώνει, καθώς και με τις διάφορες κατηγορίες από τις οποίες αποτελείται. Ένα μοντέλο μηχανικής μάθησης έχει ως βασικό στόχο την εξαγωγή διακριτών μοτίβων από δεδομένα. Τα δεδομένα αυτά ονομάζονται **training data** και είναι η βάση κάθε τέτοιου μοντέλου, καθώς η φύση τους καθορίζει τη δομή και τη λειτουργία του. Ανάλογα με τα training data, επιλέγεται και ο αντίστοιχος αλγόριθμος, ο οποίος καλείται να εντοπίσει σχέσεις μεταξύ των δειγμάτων

και να τις χρησιμοποιήσει για την πρόβλεψη νέων δεδομένων. Αφού τα δεδομένα εισαχθούν στο μοντέλο, ο αλγόριθμος τα επεξεργάζεται και από αυτά, εξάγει συμπεράσματα σε μορφή βαρών, τα οποία χρησιμοποιεί για τις προβλέψεις. Στη διαδικασία αυτή σημαντικό ρόλο παίζει η ποιότητα των δεδομένων, καθώς μικρό μέγεθος ή σημαντικός θόρυβος μπορούν να επηρεάσουν την αποτελεσματικότητα του μοντέλου. Για να ελέγξει κανείς την ιδιότητα αυτή, κοινή μεθοδολογία κατά τη δημιουργία μοντέλων είναι η χρήση μιας δεύτερης βάσης δεδομένων, η οποία ονομάζεται **test data**. Τα δεδομένα αυτά είναι άγνωστα προς το μοντέλο, καθιστώντας τα έτσι ιδανικά για διάγνωση του. Ο στόχος είναι η ελαχιστοποίηση της συνάρτησης σφάλματος, καθώς μικρότερες τιμές αυτής μεταφράζονται σε μεγαλύτερη ακρίβεια. Υπάρχουν διάφορες συναρτήσεις σφάλματος, κάποιες εκ των οποίων θα αναλυθούν στα πλαίσια της έργασίας αυτής, ενώ η χρήση τους εξαρτάται, όπως και με τα προηγούμενα, από το είδος μοντέλου που θέλει κανείς να εκπαιδεύσει.

1.1.3 Κατηγορίες Μεθόδων Εκμάθησης

Μέχρι στιγμής, έχει δοθεί μεγάλη προσοχή στο γεγονός ότι τα training data είναι αυτά τα οποία καθορίζουν και το τελικό μοντέλο. Αυτό μεταφράζεται άμεσα και στις μεθόδους εκμάθησης που χρησιμοποιούνται, καθώς αυτές επηρεάζουν και την χρήση του. Παρατίθενται στη συνέχεια οι κατηγορίες αυτών, όπως και πληροφορίες απαραίτητες προς την κατανόηση τους.

Επιτηρούμενη Μάθηση

Αναφερόμενη στη βιβλιογραφία ως **supervised learning**, η μέθοδος αυτή έχει ως στόχο τον εντοπισμό κοινών χαρακτηριστικών στα training data, τα οποία έχουν χωριστεί σε κατηγορίες πριν την εκπαίδευση.[4] Έπειτα, ο αλγόριθμος εκπαίδευσης αναλύει τα δεδομένα αυτά, κατατάσσοντας τα σε κατηγορίες και επαληθεύοντας την πρόβλεψη του με τις αντίστοιχες στην είσοδο. Ένας επιπλέον τρόπος επαλήθευσης της σωστής εκπαίδευσης του μοντέλου είναι η χρήση ενός δεύτερου, μικρότερου σετ δεδομένων, στα οποία κάνει προβλέψεις καθ όλη τη διάρκεια της. Αυτά ονομάζονται "δεδομένα επαλήθευσης", αναφερόμενα με τον όρο **validation data**. Το μοντέλο συγκρίνει τις προβλέψεις του κατα τη διάρκεια της εκπαίδευσης με αυτά τα ανεξάρτητα δεδομένα, έτσι ώστε να επιλέξει τις παραμέτρους που του προσφέρουν καλύτερη επίδοση.[5] Τέλος, αφού τελειώσει η διαδικασία εκμάθησης, γίνεται τελικός έλεγχος της ακρίβειας του προκύπτοντος μοντέλου με επίσης άγνωστα σε αυτό test data, τα οποία και ταξινομεί. Ως μέθοδος, η επιτηρούμενη μάθηση βρίσκει χρήση σε περιπτώσεις ταξινόμησης δεδομένων ή πρόβλεψης καινούργιων με βάση τα δεδομένα στη είσοδο του μοντέλου.

Μη Επιτηρούμενη Μάθηση

Σε αντίθεση με την επιτηρούμενη μάθηση, αυτή η μέθοδος, κοινώς γνωστή ως **unsupervised learning**, βασίζεται στην μη-ταξινόμηση των training data, καθώς το επιθυμητό αποτέλεσμα είναι ο εντοπισμός σχέσεων μεταξύ των δεδομένων οι οποίες είναι άγνωστες ως προς τον χρήστη.[6] Συνήθεις διαδικασίες που εφαρμόζονται στο unsupervised learning είναι αυτές του clustering, όπου τα δεδομένα ομαδοποιούνται σύμφωνα με κάποια κοινά τους χαρακτηριστικά[7] και του dimensionality reduction, όπου στόχος είναι η μείωση της διάστασης των πληροφοριών

που εμπεριέχουν τα δεδομένα, διατηρώντας όμως τις βασικές τους ιδιότητες που τα διακρίνουν.[8] Πέρα από ομαδοποίηση δεδομένων, τέτοια μοντέλα είναι ιδανικά και για εντοπισμό στοιχείων τα οποία έχουν διαφορετικές ιδιότητες από άλλα σε ένα σετ.

Ενισχυμένη Μάθηση

Η ενισχυμένη μάθηση (**reinforcment learning**) αποτελεί μια πολύ διαφορετική προσέγγιση στην διαδικασία της εκπαίδευσης, καθώς υπάρχει απουσία training data. Αντί αυτών, διαμορφώνεται ένα περιβάλλον, το οποίο βρίσκεται σε μια κατάσταση. Ο αλγόριθμος έχει τη δυνατότητα να αλληλεπιδράσει με το περιβάλλον αυτό, αλλάζοντας έτσι την κατάσταση στην οποία βρίσκεται. Η αλλαγή αυτή έπειτα βαθμολογείται ως θετική ή αρνητική, δίνοντας στο σύστημα μια αξιολόγηση σε μορφή ανταμοιβής, με επιθυμητό αποτέλεσμα τη μεγιστοποίηση των θετικών ανταμοιβών.[9] Σημειώνεται επίσης πως ανάλογα με το είδος του τελικού στόχου, το μοντέλο καλείται να εκτελέσει διαφορετικές διαδικασίες για να επιτύχει το στόχο του, όπως τον καθορισμό της βέλτιστης διαδρομής για την επίτευξη του σκοπού, την επιλογή της καλύτερης δράσης σε διαφορετικές καταστάσεις ή τη βαθμολόγηση μιας πράξης, έτσι ώστε αυτή να προσφέρει τη μέγιστη ανταμοιβή σε βάθος χρόνου.[10]

1.1.4 Προβλήματα Εφαρμογής Μεθόδων Μηχανικής Μάθησης

Από τα παραπάνω, γίνεται ξεκάθαρο πως η χρήση μηχανικής μάθησης είναι ένα πολύτιμο εργαλείο στην επεξεργασία δεδομένων. Όπως είναι όμως αναμενόμενο, υπάρχουν κάποια δομικά προβλήματα τα οποία επηρεάζουν (σε μικρότερο ή μεγαλύτερο βαθμό, ανάλογα τη μεθοδολογία), τη διαδικασία εκμάθησης. Για πληρότητα, αναφέρονται κάποιες από τις πιο μείζονος σημασίας στη συνέχεια.

Ποιότητα Δεδομένων

Η μηχανική μάθηση έχει ως βασικό θεμέλιο τα διαθέσιμα προς τον προγραμματιστή δεδομένα, προκαλώντας έτσι αναγκαιότητα για όσο τη δυνατόν καλύτερη ποιότητα αυτών. Συχνά προβλήματα που εμφανίζονται σε βάσεις δεδομένων είναι χαμηλή ποιότητα ή ποικιλία, μικρός αριθμός στοιχείων, περιορισμένη πρόσβαση σε δεδομένα κ.α.[11] Ειδικότερα, σε περιπτώσεις επιτηρούμενης (και λιγότερο συχνά μη) μάθησης, αν το δίκτυο δεν έχει πρόσβαση σε αρκετά δεδομένα, τότε παρουσιάζει μεγαλύτερη ακρίβεια στην σωστή αναγνώριση training data, σε σχέση με τα test data. Το φαινόμενο αυτό ονομάζεται **overfitting**. Στην αντίθετη περίπτωση, όπου το μοντέλο δεν είναι αρκετά περίπλοκο σε αρχιτεκτονική, υφίσταται το αντίστροφο, κάτι που μεταφράζεται ως αδυναμία του αλγορίθμου να εκμεταλλευτεί στο μέγιστο τα διαθέσιμα δεδομένα και να βελτιστοποιήσει την ακρίβειά του στο τέλος της εκπαίδευσης. Η περίπτωση αυτή ονομάζεται **underfitting** στη βιβλιογραφία.

Κατανόηση Αποφάσεων

Από τα μοντέλα που δημιουργούνται, υπάρχουν πολλά τα οποία είναι αρκετά περίπλοκα, με αποτέλεσμα να είναι αδύνατον ένας άνθρωπος να εξηγήσει τη διαδικασία που ακολούθησε εκάστοτε μοντέλο για φτάσει σε μια απόφαση. Έτσι, υπάρχει μια τάση απώλειας εμπιστοσύνης σε

αυτά από τον τελικό χρήστη, ιδιαίτερα σε περιπτώσεις που αυτός δεν κατέχει τις κατάλληλες γνώσεις πάνω σε ζητήματα μηχανικής μάθησης.[12]

Ηθική

Το προαναφερθέν πρόβλημα περί ποιότητας δεδομένων έχει ως αποτέλεσμα την παρουσίαση ζητημάτων ηθικής, τόσο ως προς τη χρήση αυτών όσο και σε σχέση με τις τελικές προβλέψεις του μοντέλου. Το φαινόμενο της προκατάληψης (**bias**) είναι μείζονος σημασίας, καθώς επηρεάζει άμεσα τη χρησιμότητα ενός μοντέλου ως προς την απόδοσή του. Αυτό το πρόβλημα προκύπτει με 3 διαφορετικούς τρόπους:[13]

- **Δεδομένα Εκπαίδευσης προς Αλγόριθμο**
  Στη προκειμένη περίπτωση, τα δεδομένα δεν παρουσιάζουν ομοιογένεια, με αποτέλεσμα ο αλγόριθμος να έχει την τάση να προτιμάει συγκεκριμένες προβλέψεις σε σχέση με άλλες. Αποτελεί ένα από τα πιο σημαντικά προβλήματα σε περιπτώσεις supervised learning.

- **Αλγόριθμος προς Χρήστη**
  Σημαντικό πρόβλημα επίσης είναι η τάση ενός χρήστη να είναι προκατειλημμένος ως προς συγκεκριμένα αποτελέσματα του μοντέλου, μειώνοντας έτσι την αξία τους.

- **Συλλογή Δεδομένων**
  Φυσικό επακόλουθο των παραπάνω είναι ότι το ζήτημα της περισυλλογής των απαραίτητων δεδομένων και η προσοχή που δίνεται στη διαδικασία αυτή είναι επίσης πολύ σημαντικό. Αν ένας χρήστης δεν λάβει μέτρα για να διασφαλίσει όσο το δυνατόν καλύτερο υλικό εκμάθησης, τότε διατρέχει τον κίνδυνο ενός λιγότερο αποτελεσματικού μοντέλου.

Η ύπαρξη και χρήση μη-αξιοκρατικών αλγορίθμων έχει ως άμεσο αποτέλεσμα διακρίσεις στον τρόπο λειτουργίας τους, όπως φυλετικές. Συνεπώς, είναι απαραίτητο, για την όσο πιο δίκαιη συμπεριφορά ενός μοντέλου, να δίνεται ιδιαίτερη προσοχή και φροντίδα στην αποφυγή των παραπάνω.

1.1.5 Αξιολόγηση Μοντέλων Μηχανικής Μάθησης

Εφόσον ένα μοντέλο σχεδιαστεί και εκπαιδευτεί, είναι έτοιμο να αξιολογηθεί, έτσι ώστε να διασφαλιστεί η όσο δυνατόν καλύτερη απόδοση του. Για τον σκοπό αυτό, γίνεται χρήση διαφόρων μετρικών μεθόδων (metrics), εξαρτώμενων συνήθως από τα δεδομένα που χρησιμοποιήθηκαν. Στη συνέχεια παρουσιάζονται οι βασικές τους κατηγορίες.

Γενικές Μέθοδοι Metrics

- **Metrics Μέσου Bias**
  Πρόκεινται για μετρικές μεθόδους που σε αντίθεση με τις υπόλοιπες τεχνικές μπορούν να εμφανίσουν και θετικές και αρνητικές τιμές, με αυτές πιο κοντά στο μηδέν να μεταφράζονται σε καλύτερη ποιότητα μοντέλου. Απευθύνονται κυρίως σε περιπτώσεις όπου η τάση ανόδου ή καθόδου μιας τιμής οδηγεί σε πιο εύκολη κατανόηση της αξιολόγησης (π.χ χρηματοοικονομικοί τομείς).

- **Metrics Απόλυτης Διαφοράς**
  Στη κατηγορία αυτή, στόχος είναι ο υπολογισμός του μέσου ή διάμεσου απόλυτου σφάλματος ενός μοντέλου. Προτιμούνται σε περιπτώσεις όπου υπάρχουν προβλήματα στα δεδομένα εισόδου, ενώ αξιολογούν καλύτερα μοντέλα σε σχέση με τη μέση απόδοση τους κατά πολλές προβλέψεις.

- **Metrics Τετραγώνων Σφάλματος** Ακολουθούν την ίδια λογική χρήσης με τη προηγούμενη κατηγορία, με τη διαφορά πως παρουσιάζουν μεγαλύτερη ευαισθησία σε αποκλίνοντα δεδομένα.

- **Metrics Αναλογίας** Χρησιμοποιούνται σε περιπτώσεις όπου δίνεται έμφαση σε αναλογίες μεταξύ προβλεπόμενων δεδομένων και αληθινών. Όπως προηγουμένως, εφαρμόζονται ως απόλυτες τιμές σφάλματος ή τετραγώνων αυτών.

- **Ποσοστιαία Metrics** Αν υπάρχει ανάγκη η αξιολόγηση να γίνει βάση κάποιο όριο απόδοσης, τότε είναι δυνατή η χρήση ποσοστών (π.χ η ακρίβεια ενός οργάνου δεν πρέπει να αποκλίνει πάνω από 5% μιας γνωστής ή πραγματικής τιμής).[14]

Classification Metrics

Σε προβλήματα ταξινόμησης (**classification**) απαιτείται διαφορετική προσέγγιση στη μέθοδο αξιολόγησης των προβλέψεων ενός μοντέλου, με σκοπό την καλύτερη κατανόηση των τελικών τιμών από το χρήστη. Ένα μοντέλο ταξινόμησης κατατάσσει δεδομένα εισόδου σε κλάσεις με βάση τα χαρακτηριστικά τους, έχοντας ως σημείο αναφοράς τα δεδομένα που έχει επεξεργαστεί κατά τη διάρκεια της εκπαίδευσης. Όταν το μοντέλο αυτό επιχειρεί να ταξινομήσει ένα αντικείμενο μπορεί να έχει τέσσερις πιθανές τιμές εξόδου. Αυτές είναι:

- Αληθινά θετική τιμή (true positive, TP), όπου το μοντέλο επιτυχώς αναγνώρισε ότι το στοιχείο εισόδου ανήκει στη σωστή κλάση.

- Ψευδώς θετική (false positive, FP), όπου η πρόβλεψη εσφαλμένα αναγνωρίζει ότι τα δεδομένα εισόδου ανήκουν σε μια κλάση.

- Αληθινά αρνητική τιμή (true negative, TN), με το μοντέλο να προβλέπει σωστά ότι το στοιχείο εισόδου δεν ανήκει στην σωστή κλάση.

- Ψευδώς αρνητική τιμή (false negative, FN), όπου τα δεδομένα εισόδου στην πραγματικότητα ανήκουν σε μια συγκεκριμένη κλάση, ενώ το μοντέλο βρίσκεται σε διαφωνία μέσω της πρόβλεψής του.

Από τα παραπάνω, είναι δυνατός ο σχεδιασμός ενός πίνακα που ονομάζεται confusion matrix, στον οποίον και απεικονίζονται οι προβλέψεις αυτές. Στη συνέχεια, οι μετρήσεις μπορούν να χρησιμοποιηθούν για τον υπολογισμό διαφόρων τιμών, παρέχοντας στον χρήστη metrics, τα οποία αντιπροσωπεύουν την απόδοση του μοντέλου.[15] Το πιο απλό παράδειγμα είναι αυτό της ακρίβειας (**accuracy**). Πρόκειται για το άθροισμα των σωστών προβλέψεων ως προς το σύνολο αυτών.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1.1}$$

Αποτελεί ένα πολύ απλό και ευανάγνωστο διαγνωστικό εργαλείο, καθώς είναι εύκολο στην κατανόησή του. Καθώς όμως το accuracy δεν είναι αντιπροσωπευτικό σε σετ δεδομένων, όπου υπάρχει ανομοιογένεια στοιχείων (περισσότερα στοιχεία σε μια κλάση, σε σύγκριση με άλλες), δεν δύναται η χρήση της συγκεκριμένης μεθόδου σε όλες τις περιπτώσεις. Για αυτό το λόγο έχουν αναπτυχθεί και άλλες κατηγορίες metrics, οι οποίες ανταποκρίνονται σε πιο ιδιαίτερα προβλήματα.

- **Precision - Recall**
  Αυτό το σετ metrics αποσκοπεί στην αξιολόγηση της δυνατότητας ενός μοντέλου να προβλέψει θετικές τιμές. Το precision είναι το σύνολο των πραγματικά θετικών τιμών που αναγνώρισε το μοντέλο προς το σύνολο των στοιχείων που ταξινόμησε ως θετικά (ανεξαρτήτως του αν η πρόβλεψη ήταν σωστή), ενώ το recall υπολογίζεται ως το σύνολο των πραγματικά θετικών προς το άθροισμα αυτών με τις ψευδώς αρνητικές προβλέψεις).

$$Precision = \frac{TP}{TP + FP} \tag{1.2}$$

$$Recall = \frac{TP}{TP + FN} \tag{1.3}$$

  Γίνεται ευκόλως αντιληπτό πως το precision αξιολογεί τη τάση του μοντέλου να ταξινομεί αντικείμενα σε μια κλάση, ενώ το recall υπολογίζει πόσα από τα στοιχεία που άνηκαν στην κλάση αυτή μπόρεσε να αναγνωρίσει το μοντέλο. Αν και μπορούν να χρησιμοποιηθούν και ανεξάρτητα, είναι προτιμητέο να γίνεται παράλληλη χρήση των μετρικών αυτών, με τελικό στόχο την βελτίωση του precision χωρίς να επηρεάζεται το recall.

- **F-1 Score** Εφαρμογή της παραπάνω λογικής αποτελεί το F-1 score, το οποίο πρόκειται για το γινόμενο precision και recall προς το άθροισμα τους, δίνοντας έτσι στο χρήστη μια γρήγορη ματιά στη σχέση μεταξύ των δύο τιμών.

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

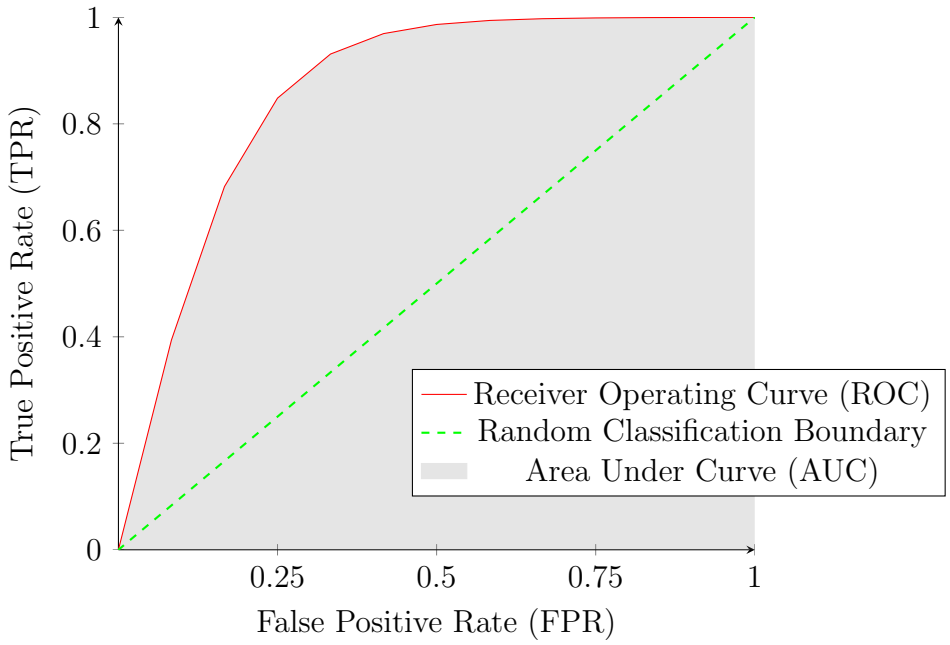
- **Καμπύλη Receiver Operating Characteristic** Ένα χρήσιμο εργαλείο στην αξιολόγηση ενός μοντέλου είναι η χαρακτηριστική καμπύλη μεταξύ αληθινά θετικών και ψευδώς αρνητικών προβλέψεων. Αυτή σχηματίζεται με το ρυθμό ψευδών στον x άξονα (false positive rate, FPR) και τον ρυθμό εντοπισμού αληθινών (true positive rate, TPR) στον y.

$$FPR = \frac{FP}{FP + TN} \tag{1.4}$$

$$TPR = Recall = \frac{TP}{TP + FP} \tag{1.5}$$

  Και οι δύο άξονες βαθμονομούνται από μηδέν έως ένα και ανάλογα με τις προβλέψεις του μοντέλου, σχηματίζεται η καμπύλη, καθώς και μια διαγώνιος που περνάει από την αρχή των αξόνων, η οποία και λειτουργεί ως όριο για την αξιοπιστία του δικτύου. Αν οι τιμές της καμπύλης τείνουν να βρίσκονται υπό της διαγωνίου, τότε το μοντέλο θεωρείται πως είναι λιγότερο αποδοτικό στις μετρήσεις του απ'οτι θα ήταν αν ταξινομούσε τα δεδομένα

εισόδου σε κλάσεις τυχαία. Η καμπύλη ενός καλού μοντέλου θα πρέπει να βρίσκεται πάνω από τη διαγώνιο, καθώς έτσι μεγιστοποιείται και ο ρυθμός ορθών ταξινομήσεων. Η τελική τιμή που χρησιμοποιείται στην αξιολόγηση είναι το εμβαδόν της επιφάνειας που βρίσκεται κάτω από την καμπύλη (area under curve, AUC), με ιδανική τιμή τη μονάδα και αντιπροσωπεύει τη πιθανότητα το μοντέλο να προτιμήσει μια σωστή πρόβλεψη σε σχέση με μια ανακριβής.[16]



Σχήμα 1.1: Γραφική αναπαράσταση ROC-AUC.

1.1.6 Εφαρμογές Μηχανικής Μάθησης

Η μηχανική μάθηση αποτελεί έναν διεπιστημονικό κλάδο, καθώς παρουσιάζει ευελιξία ως προς το είδος των δεδομένων που χρησιμοποιεί. Συνεπώς, βρίσκει εφαρμογές σε πολλούς τομείς, τόσο ακαδημαϊκούς και ερευνητικούς, όσο και στην αγορά εργασίας. Μερικοί από τους τομείς αυτούς είναι η οικονομία (ανάλυση τάσεων σε τιμές μετοχών, αξιολόγηση ρίσκου), η γεωργία (πρόβλεψη τελικής παραγωγής, εντοπισμός ασθενειών και ζιζανίων σε φυτείες, αξιολόγηση ποιότητας σοδειάς, ταξινόμηση ειδών φυτών)[17], η βιομηχανία (αυτοματισμός και επίβλεψη γραμμής παραγωγής)[18], η άμυνα (παρακολούθηση στόχων/συνόρων, κυβερνοασφάλεια)[19] και η υγεία (διάγνωση ασθενειών, αναγνώριση συμπτωμάτων, γενετική και φαρμακοβιομηχανία).[20]

1.2 Τεχνητά Νευρωνικά Δίκτυα

1.2.1 Βασική Αρχή Νευρωνικών Δικτύων

Κρίσιμη στην λειτουργία ενός μοντέλου είναι η επιλογή της δομής αυτού, καθώς και της μεθόδου εκπαίδευσης που θα χρησιμοποιηθεί για την υλοποίησή του. Μια από τις πιο δημοφιλείς προσεγγίσεις αποτελεί αυτή που αποσκοπεί στον σχεδιασμό ενός μοντέλου το οποίο προσομοιώνει νευρώνες, όπως αυτοί συναντώνται σε εγκεφάλους θηλαστικών (και κατ'επέκταση στον άνθρωπο), αποτελώντας τον κλάδο των **τεχνητών νευρωνικών δικτύων** (artificial neural networks, ANN). Τα ANN πρόκεινται για τη βάση της βαθιάς μάθησης και των εφαρμογών της. Συνεπώς είναι απαραίτητη η εξοικείωση με τη δομή και τον βασικό τρόπο λειτουργίας τους για την κατανόηση αυτής.

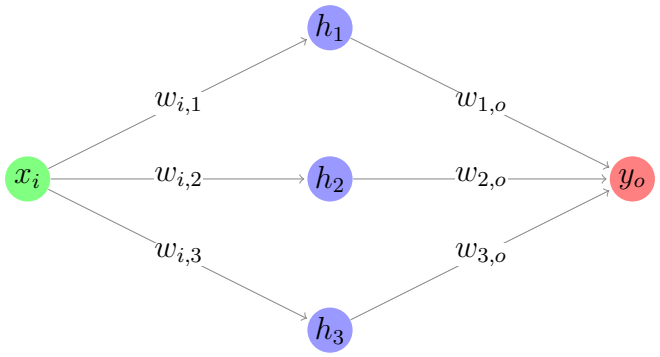
Νευρώνας

Με τον όρο αυτό γίνεται αναφορά στο πιο στοιχειώδες δομικό στοιχείο που απαρτίζει όλα τα νευρωνικά δίκτυα. Πρόκειται για ένα σημείο το οποίο δέχεται και μεταβιβάζει πληροφορία. Ανάλογα με τη θέση του στο δίκτυο, έχει τη δυνατότητα να μετασχηματίζει τα δεδομένα που δέχεται, πριν τα μεταφέρει σε επόμενο νευρώνα.

1.2.2 Δομή και Ροή Εκπαίδευσης Νευρωνικών Δικτύων

Όπως αναφέρθηκε παραπάνω, η δομή ενός νευρωνικού δικτύου εμπνέεται από τους νευρώνες σε εγκεφάλους διαφόρων θηλαστικών, με σκοπό την μίμηση του τρόπου με τον οποίο αποσπούν, μαθαίνουν και μεταφέρουν πληροφορίες. Με τον ίδιο τρόπο, οι τεχνητοί νευρώνες συνδέονται μεταξύ τους, για την εκπαίδευση του μοντέλου το οποίο απαρτίζουν. Τα νευρωνικά δίκτυα αποτελούνται τυπικά από τρία διακριτά μέρη, τα οποία ονομάζονται layers (στρώματα). Παρακάτω αναλύονται η βασική ροή εκπαίδευσης ενός νευρωνικού δικτύου, η γενική δομή του και σημαντικές έννοιες.

Παράδειγμα Fully Connected Neural Network



Σχήμα 1.2: Παράδειγμα νευρωνικού δικτύου ενός input, τριών νευρώνων στο hidden layer και ενός τελικού output. Οι νευρώνες βρίσκονται σε fully-connected διάταξη. Στις συνδέσεις απεικονίζονται τα βάρη μεταξύ προηγούμενου και επόμενου νευρώνα.

Στρώμα Εισόδου (Input Layer)

Το input layer είναι το σημείο εισαγωγής της πληροφορίας από την οποία είναι επιθυμητό να μάθει το μοντέλο, με το κάθε στοιχείο των δεδομένων να αντιπροσωπεύει έναν νευρώνα εισόδου. Στη συνέχεια, ο κάθε νευρώνας μεταδίδει τα δεδομένα σε νευρώνες του επόμενου layer, όπου και επεξεργάζονται. Σημειώνεται πως ένας νευρώνας μπορεί να συνδέεται με πολλούς νευρώνες του επόμενου layer, όπως και συμβαίνει σε πολλές περιπτώσεις, ακόμα και σε σχετικά απλές αρχιτεκτονικές.

Κρυφό Στρώμα (Hidden Layer)

Στο hidden layer γίνονται στη πραγματικότητα όλη η ουσιαστική επεξεργασία των δεδομένων ως προς την εκμάθηση ενός μοντέλου. Εδώ, ο κάθε νευρώνας πολλαπλασιάζει την πληροφορία που έλαβε στην είσοδο με στατιστικά βάρη και οι προκύπτουσες τιμές αυτές αθροίζονται. Έπειτα, στο άθροισμα αυτό προστίθεται ένας δείκτης μεροληψίας (bias), για λόγο που θα αναλυθεί στη συνέχεια, με το αποτέλεσμα να ονομάζεται weighted sum.
Μέχρι στιγμής, το δίκτυο δεν διαφέρει από ένα κλασικό γραμμικό μοντέλο μηχανικής μάθησης (π.χ. linear regression). Συγκεκριμένα, οι νευρώνες πραγματοποιούν μόνο γραμμικές μετατροπές, με αποτέλεσμα μια καινούργια γραμμική συνάρτηση στο τέλος, κάνοντας άσκοπη τη χρήση της αρχιτεκτονικής νευρωνικού δικτύου. Η διαφοροποίηση υφίσταται με τη χρήση μιας **συνάρτησης ενεργοποίησης (activation function)**, εξαρτώμενης του weighted sum, η οποία αποσκοπεί στην εισαγωγή μη γραμμικότητας στο μοντέλο, και καθορίζει αν η πληροφορία θα μεταβιβαστεί στο επόμενο layer, ή αλλιώς αν ο νευρώνας θεωρείται ενεργοποιημένος. Αυτό συνδέεται άμεσα με την συνάρτηση που θα επιλεχθεί για activation.[21, p. 5] Η χρήση bias, στη πραγματικότητα, επηρεάζει την ενεργοποίηση της activation function, καθώς μεταβάλλει την προκύπτουσα τιμή, αλλάζοντας έτσι την ευαισθησία του νευρώνα στα εξερχόμενα δεδομένα.

Ο αριθμός νευρώνων στο hidden layer καθορίζεται από τον προγραμματιστή και είναι σημαντικός για τη σωστή λειτουργία του νευρωνικού δικτύου. Αν ο αριθμός τους είναι υπερβολικά μεγάλος, τότε το δίκτυο έχει τάση overfitting, με την αντίστροφη περίπτωση να οδηγεί σε underfitting. Αν τα εξερχόμενα δεδομένα όλων των νευρώνων ενός layer συνδέονται με τον κάθε νευρώνα του επομένου, τότε το μετέπειτα layer χαρακτηρίζεται ως πλήρως συνδεδεμένο **(Fully Connected Layer, FC)**.

Στρώμα Εξόδου (Output Layer)

Εφόσον τα δεδομένα έχουν αναλυθεί από το δίκτυο, τα αποτελέσματα εμφανίζονται στο output layer, το οποίο αποτελείται από έναν ή παραπάνω νευρώνες, ανάλογα με το τελικό στόχο λειτουργίας του μοντέλου. Σε εφαρμογές ταξινόμησης, τα δεδομένα του output layer φιλτράρονται μέσω μιας activation function, όπως και στο hidden layer.

Παράμετροι και Υπερπαράμετροι Νευρωνικών Δικτύων

Ένα δίκτυο αποτελείται από δύο είδη τιμών, τα οποία επηρεάζουν άμεσα τη λειτουργία του. Παράμετροι ονομάζονται οι διάφορες τιμές σε ένα δίκτυο, στις οποίες δεν υπάρχει άμεση πρόσβαση από το χρήστη και μεταβάλλονται αποκλειστικά μέσω της εκμάθησης, ενώ υπερπαράμετροι

(hyperparameters) είναι μεταβλητές που ορίζονται πριν ξεκινήσει η εκπαίδευση και έχουν ως στόχο την αλλαγή της συμπεριφοράς του μοντέλου κατά τη διάρκειά της.

Backpropagation

Κατά τη διαδικασία της εκπαίδευσης, είναι σημαντικό τα βάρη να αλλάζουν έτσι ώστε να γίνεται ελαχιστοποίηση του συνολικού σφάλματος. Για το λόγο αυτό, τα δεδομένα που εξέρχονται του output layer αναλύονται μέσω μια διαδικασίας που ονομάζεται backpropagation, η οποία αποσκοπεί στη βελτιστοποίηση της τιμής των βαρών στο μοντέλο. Αυτό συνήθως γίνεται μέσω ενός αλγορίθμου τύπου **gradient descent**, όπου υπολογίζεται η κλίση της συνάρτησης σφάλματος ως προς τα βάρη, με στόχο τον εντοπισμό ενός τοπικού ελαχίστου της. Εφόσον οι τελικές τιμές συγκριθούν με τις αρχικές, υπολογίζεται η συνάρτηση σφάλματος (κοινώς αναφερόμενη ως **loss function**) για κάθε τιμή και τα σφάλματα αυτά αθροίζονται. Στη συνέχεια, υπολογίζεται η μερική παράγωγος του συνολικού σφάλματος ως προς κάθε βάρος που χρησιμοποιήθηκε στη διαδικασία της εκπαίδευσης, με τη χρήση του μαθηματικού κανόνα αλυσίδας:

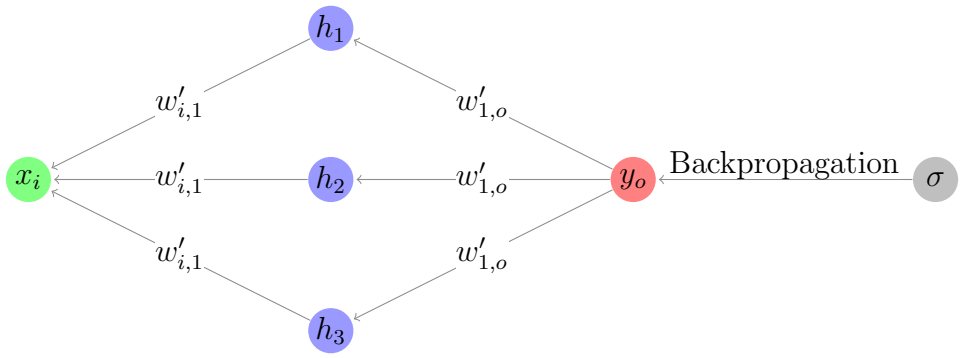
$$\frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial W_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial W_{ij}} \tag{1.6}$$

όπου E το συνολικό σφάλμα, W_{ij} το βάρος μεταξύ νευρώνων i και j, o_j η έξοδος του νευρώνα μετά τη χρήση της activation function και net_j το weighted sum. Έπειτα από πράξεις, προκύπτει η ακόλουθη σχέση:

$$\Delta W_{ij} = -\eta\frac{\partial E}{\partial W_{ij}} \tag{1.7}$$

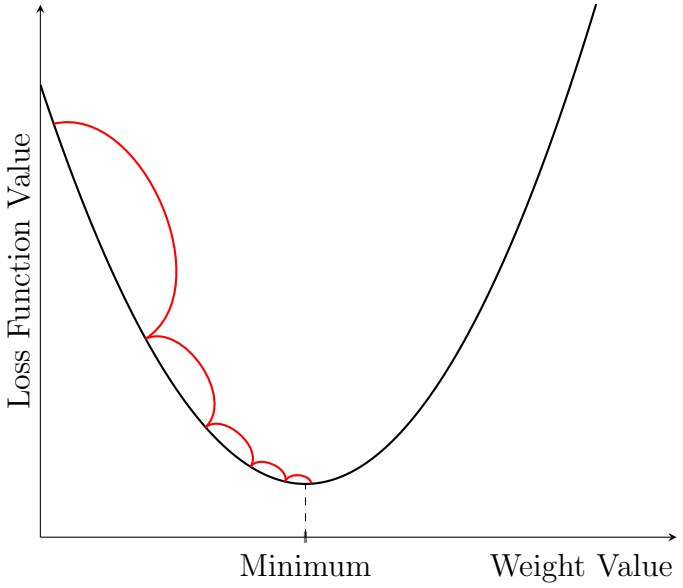
όπου ΔW_{ij} η μεταβολή που θα προστεθεί στο αντίστοιχο βάρος και η το step size ή κοινώς **learning rate** του αλγορίθμου.

Backpropagation για Σχήμα 1.2



Σχήμα 1.3: Αναπαράσταση backpropagation. Έπειτα του υπολογισμού συνολικού σφάλματος, τα weights ανανεώνονται με βάση τα αποτελέσματα του gradient descent. Τα καινούργια weights σημειώνονται ως w'_{ij}.

Το learning rate είναι από τις πιο σημαντικές υπερπαραμέτρους σε ένα νευρωνικό δίκτυο, καθώς επηρεάζει άμεσα το πόσο γρήγορα το μοντέλο **συγκλίνει** (φτάνει σε σημείο όπου το σφάλμα σταματά να μειώνεται, θεωρείται αποδεκτό ή σταματάει η εκπαίδευση, γνωστό στη βιβλιογραφία ως convergence). Μικρό learning rate σημαίνει πιο ομαλή αλλά αργή σύγκλιση, ενώ μεγάλες τιμές αυτού μεταφράζονται σε μεγαλύτερη ταχύτητα στην ανάλυση κάποιων υποσυνόλων των training data.[22] Παρατηρεί κανείς πως το backpropagation είναι δυνατό μόνο λόγω της μη-γραμμικότητας που εισάγεται στο σύστημα από την activation function, εφόσον η μερική παράγωγος μιας γραμμικής συνάρτησης είναι σταθερά και ανεξάρτητη μεταβλητών.



Σχήμα 1.4: Παράδειγμα εφαρμογής gradient descent. Η τιμή της loss function μειώνεται έπειτα από κάθε epoch λόγω της μεταβολής που υφίστανται τα weights κατά τη διάρκεια του backpropagation. Ο τελικός στόχος είναι η χρήση gradient descent, έτσι ώστε η loss function να ισούται με τη τιμή τοπικού ή ολικού ελαχίστου της γραφικής παράστασης (Minimum). Εκεί εντοπίζεται και η καλύτερη δυνατή τιμή ως προς τα weights.

Επανάληψη

Άλλη μια σημαντική έννοια και υπερπαράμετρος στην εκπαίδευση είναι αυτή της **εποχής (epoch)**. Πρόκειται για τον αριθμό επαναλήψεων ενός κύκλου εκπαίδευσης, έτσι ώστε να πραγματοποιηθεί backpropagation αρκετές φόρες για την ελαχιστοποίηση των σφαλμάτων στις προβλέψεις του δικτύου. Η τιμή αυτή συνεπώς επηρεάζει άμεσα την απόδοση ενός νευρωνικού δικτύου.

1.2.3 Είδη Activation Functions

Η επιλογή της σωστής activation function είναι ζωτικής σημασίας, καθώς καθορίζει το πως ένα δίκτυο μαθαίνει από τις πληροφορίες που δίνονται σε αυτό. Ένα από τα βασικά προβλήματα που πρέπει να ληφθούν υπόψιν είναι αυτό του **vanishing gradient**, όπου μετά από πολλές

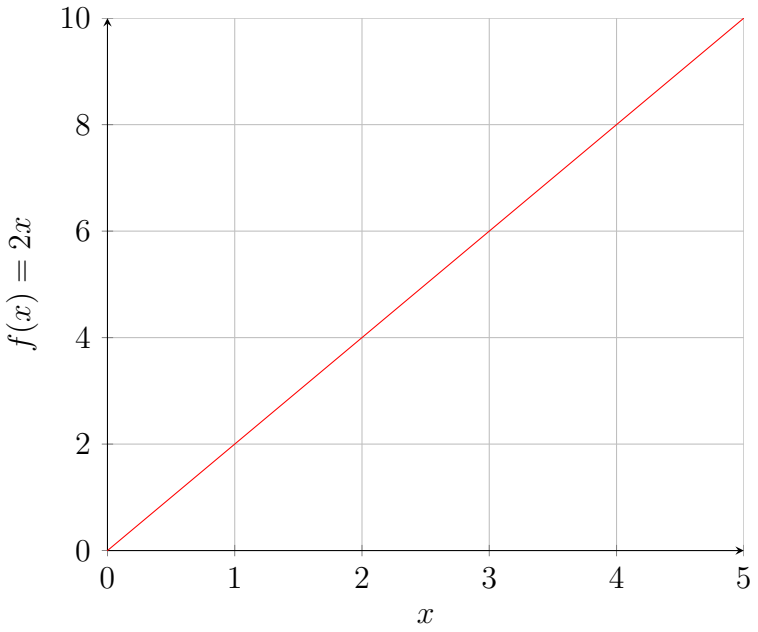
επαναλήψεις backpropagation, η κλίση της activation function γίνεται αρκετά μικρή, αυξάνοντας δραματικά τον χρόνο εκμάθησης ή οδηγώντας σε αδυναμία περαιτέρω εκπαίδευσης του δικτύου. Ταυτόχρονα, το ζητούμενο αποτέλεσμα στο output layer δημιουργεί δίλημμα ως προς την επιλογή της σωστής συνάρτησης. Στη συνέχεια παρατίθενται κάποιες βασικές activation functions, συμπεριλαμβανομένων των χαρακτηριστικών, πλεονεκτημάτων και μειονεκτημάτων τους.[23]

Γραμμική Συνάρτηση

Μια τέτοια συνάρτηση είναι της μορφής:

$$f(x) = cx \tag{1.8}$$

και όπως αναφέρθηκε προηγουμένως, δεν χρησιμοποιείται σε νευρωνικά δίκτυα, καθώς η έλλειψη μη-γραμμικότητας δεν τα διαχωρίζει από μοντέλα linear regression και καθιστά αδύνατη τη χρήση backpropagation.



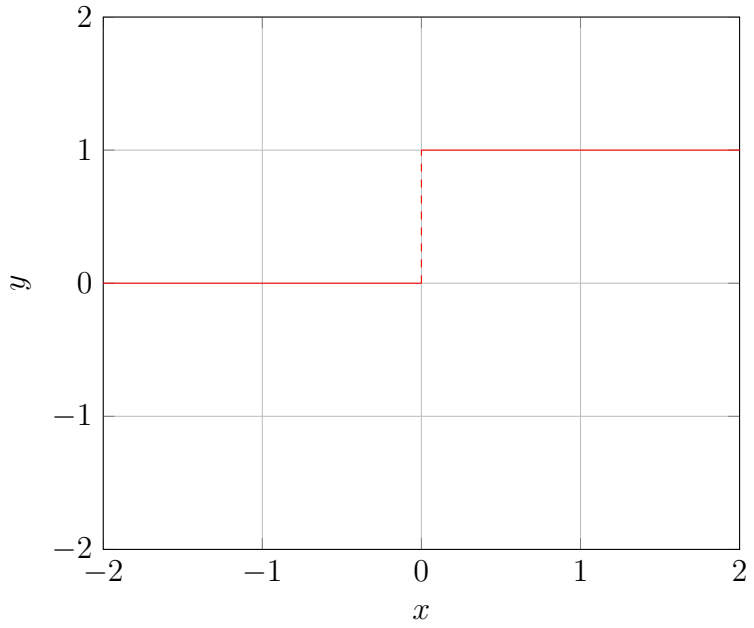
Σχήμα 1.5: Γραφική παράσταση γραμμικής συνάρτησης για $c = 2$.

Μια λογική προσέγγιση στην επιλογή μιας activation function είναι η ευκολία στην ενεργοποίηση (ή μη) του εκάστοτε νευρώνα. Μια φαινομενικά καλή λύση είναι αυτή της συνάρτησης βήματος (step function):

$$f(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases} \tag{1.9}$$

Στη προκειμένη περίπτωση, είναι δυνατή η ρύθμιση της συνάρτησης με κάποιο συντελεστή, οριοθετώντας έτσι τις τιμές του output που μπορούν να ενεργοποιήσουν τον νευρώνα, κάνοντας την ιδανική για δυαδικές ταξινομήσεις (καθώς το αποτέλεσμα θα είναι πάντα μηδέν ή ένα).

Παρόλα αυτά, η κλίση της συνάρτησης βήματος είναι πάντα μηδέν, δημιουργώντας πρόβλημα στη διαδικασία backpropagation, εφόσον δεν προκύπτουν καινούργιες τιμές για τα βάρη.



Σχήμα 1.6: Αναπαράσταση step function.

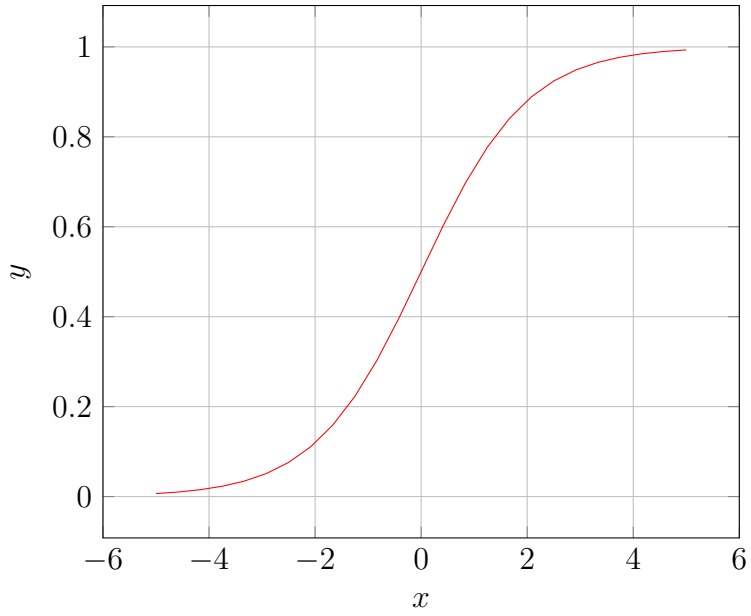
Τα παραπάνω παραδείγματα, αν και τετριμμένα, δείχνουν τη βασική λογική με την οποία μπορεί κανείς να αξιολογήσει την χρησιμότητα μιας συνάρτησης στα πλαίσια της ενεργοποίησης νευρώνων, ως προς τη μη-γραμμικότητα και την διευκόλυνση του backpropagation. Είναι λοιπόν αναμενόμενο, πως οι πιο συχνές activation functions, όπως αυτές συναντιόνται σε πολλές δομές νευρωνικών δικτύων, υπακούν τις δύο αυτές προϋποθέσεις.

Σιγμοειδής Συνάρτηση

Η σιγμοειδής συνάρτηση (**sigmoid** ή logistic function) αποτελεί το πρώτο ουσιαστικό βήμα στην επίλυση των προβλημάτων που παρουσίαζαν οι προαναφερθείσες συναρτήσεις και ακολουθεί την εξής σχέση:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{1.10}$$

Παίρνει το όνομα της από τη γραφική της παράσταση, με τις τιμές της να κυμαίνονται στο διάστημα $[0, 1]$. Λόγω του εύρους αυτού, χρησιμοποιείται συνήθως για προβλέψεις πιθανοτήτων, μέσω εφαρμογής της στο output layer.



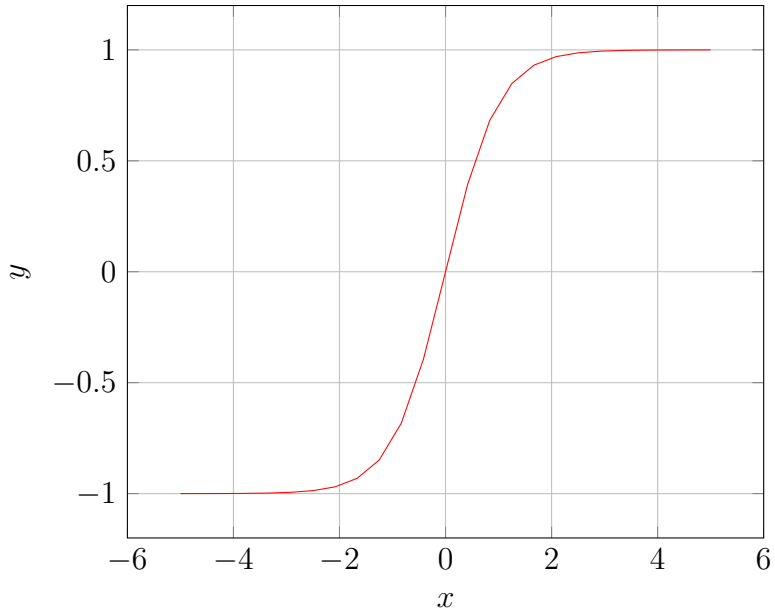
Σχήμα 1.7: Γραφική παράσταση sigmoid function.

Υπερβολική Εφαπτομενική Συνάρτηση

Η συνάρτηση αυτή (κοινώς αναφερόμενη ως **tanh**) ακολουθεί παρόμοια λογική με αυτή της σιγμοειδούς, περιορίζοντας την τελική τιμή σε ένα διάστημα. Συγκεκριμένα, εκφράζεται από τη σχέση:

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{1.11}$$

με την έξοδό της να βρίσκεται στο όριο $[-1, 1]$. Η συμμετρία της συνάρτησης, ως προς τις τιμές της, μεταφράζεται σε μεγαλύτερες κλίσεις, που οδηγεί σε πιο σημαντικές μεταβολές στα βάρη του δικτύου, κάτι που συνήθως προκαλεί πιο γρήγορη σύγκλιση κατά την εκπαίδευση. Η ιδιότητα αυτή την καθιστά προτιμητέα σε hidden layers, σε σχέση με την σιγμοειδή, η οποία δεν παρουσιάζει συμμετρία γύρω από το μηδέν. Το κοινό πρόβλημα μεταξύ των δύο αυτών συναρτήσεων είναι αυτό του vanishing gradient. Στην περίπτωση της sigmoid συνάρτησης, υπερβολικά μεγάλες ή μικρές τιμές παρουσιάζουν κορεσμό, καθώς τείνουν προς το ένα και μηδέν αντίστοιχα, όπου η γραφική παράσταση προσεγγίζει ευθεία. Αυτό σημαίνει πως η κλίση για τις τιμές αυτές είναι πολύ μικρή και συνεπώς τα βάρη αλλάζουν λίγο έως καθόλου στο backpropagation. Το ίδιο πρόβλημα ισχύει και για την tanh συνάρτηση. Άλλη δυσκολία παρουσιάζεται στις υπολογιστικές απαιτήσεις των δύο εξισώσεων, λόγω της εκθετικότητάς τους. Συνεπώς, έχουν αναπτυχθεί και άλλες συναρτήσεις, με στόχο την εξάλειψη ή ελαχιστοποίηση των εμποδίων αυτών.



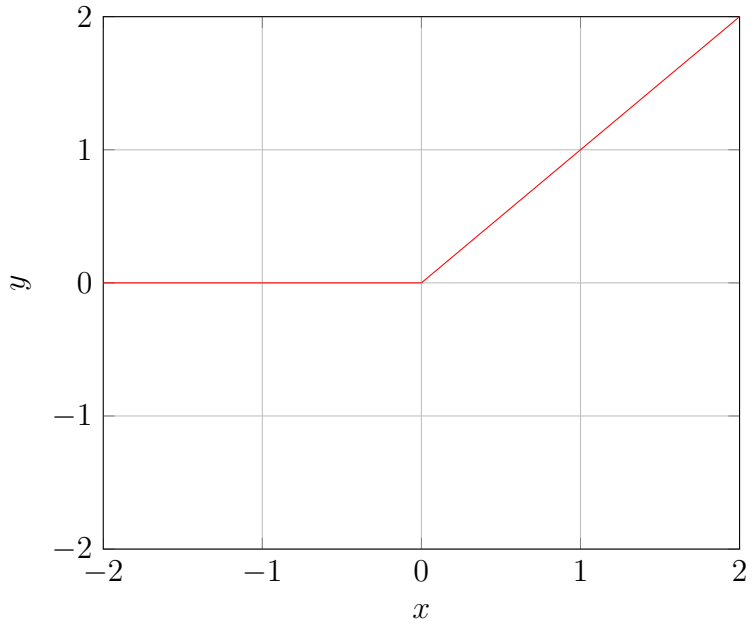
Σχήμα 1.8: Γραφική παράσταση tanh function. Φαίνεται εύκολα η αλλαγή του σημείου συμμετρίας σε σύγκριση με τη sigmoid και η ομοιότητα ως προς την γραμμικότητα στα άκρα της.

Συνάρτηση ReLU

Η **ReLU (Rectified Linear Unit)** πρόκειται για συνάρτηση της μορφής:

$$f(x) = max(0, x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases} \tag{1.12}$$

Η ReLU αρχικά φαίνεται να παρουσιάζει γραμμική συμπεριφορά, αλλά στην πραγματικότητα είναι μη-γραμμική λόγω της εναλλαγής στην μορφή της για αρνητικές τιμές. Καθώς η μερική παράγωγος δεν ορίζεται στο σημείο μηδέν, η κλίση της ReLU ταυτίζεται σε πιθανές τιμές με την συνάρτηση βήματος, απλοποιώντας αισθητά το backpropagation και μειώνοντας την απαιτούμενη υπολογιστική δύναμη. Ταυτόχρονα, εφόσον όλες οι τιμές όπου $x < 0$ ισούνται με μηδέν, ο αριθμός των ενεργοποιημένων νευρώνων στο δίκτυο είναι μικρότερος. Αυτή η συμπεριφορά όμως μπορεί, όπως και πριν, να προκαλέσει φαινόμενα vanishing gradient, σε μικρότερο όμως ποσοστό από τις περιπτώσεις sigmoid ή tanh.



Σχήμα 1.9: Γραφική παράσταση συνάρτησης ReLU.

Leaky ReLU και Paremetric ReLU

Μια αποτελεσματική μετατροπή της συνάρτησης ReLU ως προς αρνητικές τιμές x είναι η εξής:

$$f(x) = \begin{cases} x, & x \geq 0 \\ 0.01x, & x < 0 \end{cases} \tag{1.13}$$

όπου πλέον δεν υπάρχει όριο στις προκύπτουσες τιμές. Η πολλαπλασιαστική σταθερά που εφαρμόζεται μπορεί να θεωρηθεί υπερπαράμετρος, σε περιπτώσεις όπου υπάρχει άμεσο ενδιαφέρον στον εντοπισμό βέλτιστης κλίσης. Γενικεύοντας λοιπόν:

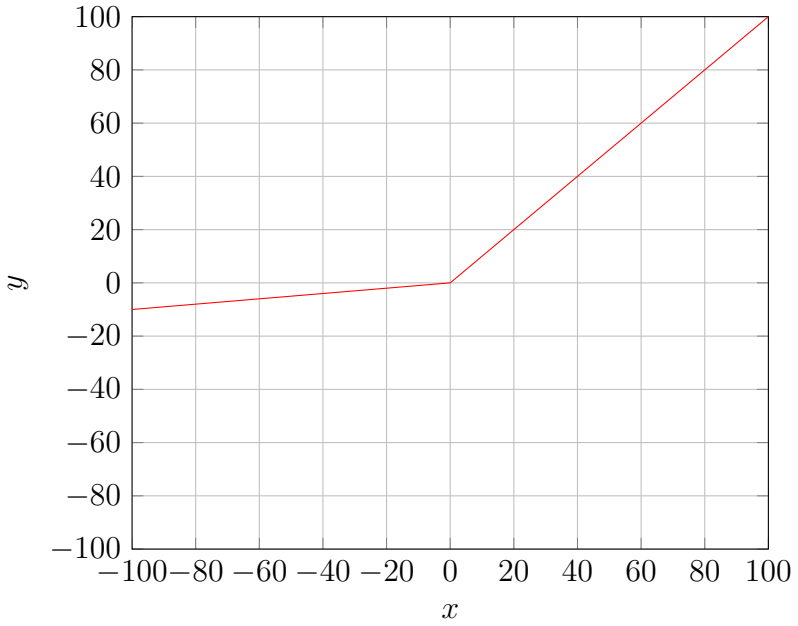
$$f(x) = \begin{cases} x, & x \geq 0 \\ px, & x < 0 \end{cases} \tag{1.14}$$

Η συνάρτηση αυτή ονομάζεται **paremetric ReLU**. Δίνεται ιδιαίτερη προσοχή στην επιλογή της υπερπαραμέτρου p, διότι οι παραπάνω υπόκεινται εύκολα σε φαινόμενα overfitting.

Λοιπές Παραλλαγές ReLU

Λόγω της ευελιξίας που την χαρακτηρίζει, η ReLU είναι βάση για πολλές άλλες activation functions. Συνεπώς, υπάρχει ένα μεγάλο πλήθος παραλλαγών αυτής, με σκοπό την αντιμετώπιση των προβλημάτων που αντιμετωπίζει. Πιθανές παραλλαγές περιλαμβάνουν:

- Αλλαγή της συμπεριφοράς ReLU ως προς αρνητικές τιμές.
- Αύξηση της μη-γραμμικότητας που προσφέρει η βασική συνάρτηση ReLU.
- Οριοθέτηση προκυπτουσών τιμών.
- Εισαγωγή εκθετικότητας στη συνάρτηση.



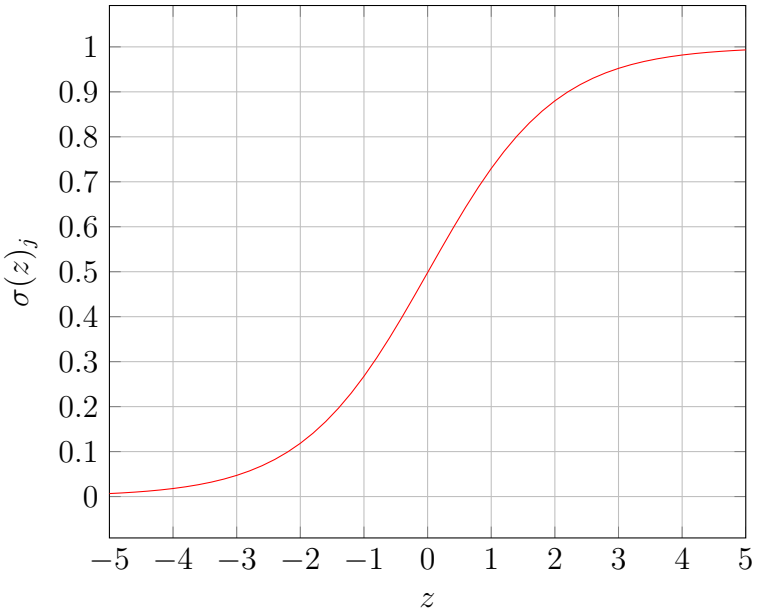
Σχήμα 1.10: Γραφική παράσταση συνάρτησης Parametric ReLU για $p = 0.1$.

Συνάρτηση Softmax

Σε περιπτώσεις όπου ζητείται η πρόβλεψη πιθανοτήτων για κλάσεις των οποίων οι πιθανότητες είναι ανεξάρτητες μεταξύ αυτών, απαιτείται η χρήση συνάρτησης ικανής να αντιπροσωπεύσει πολυωνυμική κατανομή πιθανοτήτων. Μια καλή συνάρτηση που πληρεί τις προδιαγραφές αυτές είναι η λεγόμενη **softmax**:

$$f(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}} \tag{1.15}$$

όπου $\vec{z}$ το διάνυσμα εισόδου, αποτελούμενο από N στοιχεία που αντιστοιχούν σε N αριθμό κλάσεων, z_i το i στοιχείο του διανύσματος εισόδου και z_j το j στοιχείο του διανύσματος στην έξοδο. Η μεγαλύτερη τιμή επιστρέφει μονάδα, ενώ όλες οι υπόλοιπες μηδέν, ταξινομώντας την πιθανότερη κλάση για τα εκάστοτε δεδομένα, αποκλείοντας τις υπόλοιπες. Έτσι, η συνάρτηση αυτή είναι προτιμητέα σε εφαρμογές πολλαπλής ταξινόμησης.[21, p. 8]



Σχήμα 1.11: Γραφική παράσταση Softmax. Η ομοιότητα με τη sigmoid δεν είναι τυχαία, καθώς η ταξινόμηση ακολουθεί παρόμοια λογική, με τη διαφορά πως η softmax λαμβάνει υπόψιν όλες τις διαθέσιμες κλάσεις.

1.2.4 Συναρτήσεις Σφάλματος

Στην υποενότητα περί backpropagation έγινε αναφορά στη συνάρτηση σφάλματος (loss function) ενός νευρωνικού δικτύου και τον ρόλο της στη διαδικασία τροποποίησης των βαρών στις συνδέσεις μεταξύ των νευρώνων αυτού. Όπως και με τις activation functions, η επιλογή της loss function πρέπει να γίνεται με προσοχή και παίρνοντας υπόψιν την τελική λειτουργία του μοντέλου. Στη συνέχεια, θα αναλυθούν κάποιες από αυτές, καθώς και οι περιπτώσεις στις οποίες χρησιμοποιούνται.[24]

Συναρτήσεις Σφάλματος σε Εφαρμογές Παλινδρόμησης

Λόγω της πολυπλοκότητας τους, τα νευρωνικά δίκτυα δεν εκπαιδεύονται συχνά για τέτοιες χρήσεις. Παρόλα αυτά, υπάρχει η δυνατότητα να επιλύσουν προβλήματα παλινδρόμησης, ενώ οι loss functions αυτών είναι αρκετά απλές, έτσι ώστε να αποτελούν μια εισαγωγή σε αντίστοιχες για ταξινόμηση. Συνεπώς, αναφέρονται ενδεικτικά δύο από τις πιο συχνές, καθώς και κάποια βασικά χαρακτηριστικά τους, οι οποίες παρατέθηκαν και στην υποενότητα 1.1.5 (καθώς έχουν διττή λειτουργία και ως metrics απόδοσης).

- **Μέσο Τετράγωνο Σφάλματος (Mean Squared Error)**: Το MSE είναι συνάρτηση της μορφής:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y_i})^2, \tag{1.16}$$

όπου n ο αριθμός των δειγμάτων, y_i η τιμή που προέβλεψε το δίκτυο και $\bar{y_i}$ η πραγματική τιμή.

- **Μέσο Απόλυτο Σφάλμα (Mean Absolute Error)**: Αναφερόμενο συντόμως ως MAE, εκφράζεται ως:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y_i}), \tag{1.17}$$

με τις μεταβλητές να αντιστοιχούν σε αυτές του MSE.

Η έλλειψη τετραγώνου στη διαφορά πρόβλεψης - πραγματικής τιμής καθιστά την MAE γραμμική συνάρτηση. Συνεπώς, αν και πιο ιδανική για περιπτώσεις όπου υπάρχουν αποκλίνουσες τιμές, δεν είναι διαφορίσιμη, κάνοντας αδύνατο το backpropagation.[24, p.8-9]

Συναρτήσεις Σφάλματος σε Εφαρμογές Ταξινόμησης

Σε προβλήματα classification, ο στόχος είναι η σωστή αντιστοίχηση δεδομένων σε κάποια κλάση. Η απαίτηση αυτή έχει ως αποτέλεσμα την ανάγκη για πιο εξειδικευμένες loss functions, οι οποίες έχουν πιο άμεση εξάρτηση από το τι είδους ταξινόμηση καλείται να διεκπεραιώσει το δίκτυο στο τέλος της εκπαίδευσής του.

- **Binary Cross Entropy Loss (BCE)**: Πρόκειται για λογαριθμική συνάρτηση η οποία αποσκοπεί να παρουσιάσει τη διαφορά μεταξύ προβλεπόμενης και αληθινής πιθανότητας εκάστοτε δείγμα να ανήκει στη ζητούμενη τάξη (true class). Σε binary classification, η true class παίρνει συνήθως ως τιμή τη μονάδα ενώ η άλλη κλάση την τιμή μηδέν, με την πρόβλεψη να είναι διάνυσμα του οποίου τα στοιχεία αντιστοιχούν στις πιθανότητες τα δεδομένων να ανήκουν σε κάποια από τις κλάσεις αυτές. Αυτή η επεξεργασία κλάσεων ονομάζεται **one-hot encoding**. Για κάθε δείγμα, ισχύει:

$$BCE(y, p) = -(ylog(p) + (1 - y)log(1 - p)), \tag{1.18}$$

όπου y η τιμή της true class και p η προβλεπόμενη πιθανότητα τα δεδομένα να ανήκουν στην κλάση αυτή. Λόγω της εξάρτησης της συνάρτησης από το y, μια παρόμοια διατύπωση της BCE είναι η ακόλουθη:

$$BCE(y, p) = \begin{cases} -log(p), & y = 1 \\ -log(1 - p), & y = 0 \end{cases} \tag{1.19}$$

Εφόσον ο στόχος στην εκπαίδευση είναι πάντα η ελαχιστοποίηση του σφάλματος, θα πρέπει η τιμή του p να πλησιάζει όσο το δυνατόν περισσότερο αυτή του y. Η BCE πληρεί όλες τις πλέον βασικές προϋποθέσεις (διαφορίσιμη, υπολογιστικά συμφέρουσα, ευκολία στην κατανόηση), όμως λόγω της binary ιδιότητάς της, έχει προβλήματα σε περιπτώσεις όπου η μια εκ των κλάσεων διαθέτει περισσότερα δείγματα (π.χ. 100 θετικά

προς 50 αρνητικά).[24, p.14] Γιαυτό τον λόγο, σε πολλές εφαρμογές όπου οι συνθήκες δεν επιτρέπουν ισορροπία στον αριθμό των δειγμάτων ανά κλάση, χρησιμοποιείται παραλλαγή της BCE με εισαγωγή κάποιου βάρους **(Weighted BCE)**:

$$BCE(y, p) = -(w_i \cdot ylog(p) + w_i(1 - y)log(1 - p)), \tag{1.20}$$

όπου w_i το βάρος δείγματος i κατά την εκπαίδευση. Η χρήση βάρους επιτρέπει στο δίκτυο να δώσει περισσότερη σημασία στην κλάση που διαθέτει τα λιγότερα στοιχεία.[24, p.15]

- **Categorical Cross-entropy Loss (CCE)**: Αν ο στόχος είναι η ταξινόμηση δεδομένων σε πολλές κλάσεις, η εφαρμογή της BCE δεν είναι αρκετή, καθώς δεν μπορεί να προβλέψει παραπάνω από δύο σε κάθε περίπτωση. Αντί αυτής, χρησιμοποιείται η CCE:

$$CCE = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{i,j}log(p_{i,j}), \tag{1.21}$$

με το N να συμβολίζει τον αριθμό των δειγμάτων, C το σύνολο των κλάσεων προς classification, y η σωστή κλάση και p η πιθανότητα το κάθε δείγμα να ανήκει σε αυτή. Όπως και προηγουμένως, οι κλάσεις είναι one-hot encoded (π.χ. για στοιχείο που ανήκει στην κλάση 1 εκ τριών, $y = [1, 0, 0]$).[24, p.15] Σε πολλές περιπτώσεις όμως, για διευκόλυνση στην κατανόηση των κλάσεων, είναι επιθυμητή η χρήση ακεραίων αντί για μηδέν ή ένα, με τη χρήση μιας παραλλαγής της CCE, την επονομαζόμενη **Sparse CCE**:

$$SCCE = \frac{1}{N}\sum_{i=1}^{N} log(y^{-}_{i,y_i}) \tag{1.22}$$

με το y_i να αντιστοιχεί στον αριθμό της σωστής κλάσης του στοιχείου i και το y_{i,y_i} στην προβλεπόμενη.[24, p.15-16]

1.2.5 Μέθοδοι Βελτιστοποίησης Backpropagation

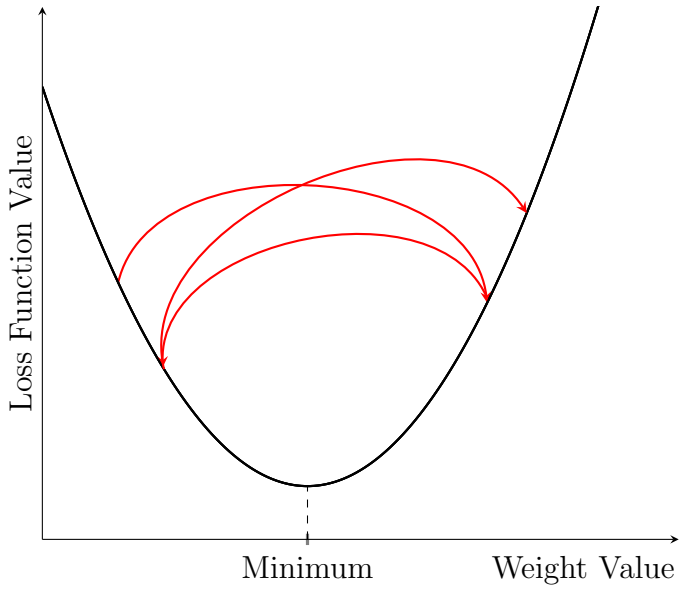
Κατά την ανάλυση της διαδικασίας του backpropagation εξηγήθηκε εκτενώς η έννοια του gradient descent και πως αυτό κατέχει έναν από τους σημαντικότερους ρόλους στην εκπαίδευση ενός δικτύου. Παρόλα αυτά, το gradient descent δημιουργεί επιπλοκές όταν το μέγεθος των training data είναι αρκετά μεγάλο, καθώς πρέπει να ληφθεί υπόψιν το loss όλως των δειγμάτων για να γίνει μεταβολή των βαρών (σχέση 1.7). Συνεπώς, αναπτύχθηκαν και άλλες μέθοδοι, με σκοπό την αντιμετώπιση του προβλήματος αυτού, οι οποίες ονομάζονται **optimizers**. Ακολούθως, δίνονται κάποιοι από τους πιο συχνούς σε εφαρμογή optimizers, ενώ ταυτόχρονα αναλύονται ως προς τη βασική τους λειτουργία.[25]

Stochastic Gradient Descent (SGD)

Μια λύση είναι η επιλογή ενός δείγματος εκ των training data, από το οποίο θα γίνει ο υπολογισμός των βαρών όλου του δικτύου. Με αυτό τον τρόπο, το loss ενημερώνεται πολύ πιο γρήγορα, με αποτέλεσμα πιο γρήγορη σύγκλιση. Σημαντικό μειονέκτημα της προσέγγισης αυτής είναι πως η εξαίρεση όλων των υπολοίπων δειγμάτων μπορεί να οδηγήσει σε μεγαλύτερο

loss, απ'οτι με τη χρήση απλού gradient descent. Ένας συμβιβασμός των δύο μεθόδων μπορεί να επιτευχθεί λοιπόν, αν αντί για ένα μόνο δείγμα επιλεχθεί ένα υποσύνολο των training data. Η παραλλαγή αυτή του SGD ονομάζεται **mini-batch gradient descent.**

Ένα ακόμα πρόβλημα, το οποίο παρατηρείται σε GD και SGD, είναι αυτό της σταθερότητας, λόγω της εξάρτησης τους από τον υπολογισμό της κλίσης. Όπως έχει ήδη αναφερθεί, ο τελικός στόχος είναι η ελαχιστοποίηση του loss, δηλαδή η εύρεση κάποιου ελαχίστου σημείου στην επιφάνεια της loss function. Αν όμως το learning rate είναι πολύ μικρό, δεν επιτυγχάνεται γρήγορο convergence, ενώ αντίθετα πολύ μεγάλη τιμή του μπορεί να "παγιδεύσει" τον αλγόριθμο σε κάποιο τοπικό ελάχιστο, όπου λόγω των μεγάλων steps (αλλαγών στα βάρη) η τιμή του loss αυξομειώνεται χωρίς να συγκλίνει προς την ελάχιστη τιμή. Είναι δυνατή η τροποποίηση του learning rate κατα τη διάρκεια της εκπαίδευσης είτε σε προκαθορισμένα διαστήματα ή με τον ορισμό κάποιου στόχου που πρέπει να επιτευχθεί κατα την εκπαίδευση. Λόγω της έλλειψης προσαρμοστικότητας των μεθόδων αυτών όμως, τα παραπάνω προβλήματα δεν είναι εύκολα να επιλυθούν. Συνεπώς, οι GD και SGD δεν αρκούν πάντα, ειδικά σε προβλήματα όπου τα δεδομένα παρουσιάζουν μεγαλύτερη πολυπλοκότητα.[25, p.2-3]



Σχήμα 1.12: Παράδειγμα αδυναμίας σύγκλισης προς ελάχιστη τιμή μέσω gradient descent. Στην προκειμένη περίπτωση, το learning rate είναι υπερβολικά μεγάλο και έτσι δεν παρουσιάζεται μείωση της loss function.

SGD με Momentum

Ένας τρόπος να αποφευχθεί το πρόβλημα της ταλάντωσης του loss είναι μια τροποποίηση του SGD, έτσι ώστε να ακολουθεί την καμπύλη της επιφάνειας προς το ελάχιστο, το οποίο επιτυγχάνεται με την εισαγωγή ενός παράγοντα "ορμής" (momentum), επιταχύνοντας το convergence και μειώνοντας τη διακύμανση του loss.[25, p.4]

Προσαρμοστικές Μέθοδοι Βελτιστοποίησης

Για να γίνει πιο δυναμική η διαδικασία ανανέωσης των βαρών σε ένα νευρωνικό δίκτυο, είναι απαραίτητη η προσαρμογή του learning rate ως προς τις παραμέτρους του καθ'όλη τη διάρκεια της εκπαίδευσης. Η ανάγκη αυτή οδήγησε στη δημιουργία του **Adagrad**, αλγόριθμος που αλλάζει την τιμή του learning rate με βάση τις προηγουμένως υπολογισμένες κλίσεις, προσφέροντας πολύ καλές αποδόσεις. Λόγω σχεδιασμού όμως, ο Adagrad τείνει έπειτα από πολλά epochs να μειώνει το learning rate σε πολύ μικρές τιμές, με αποτέλεσμα τα βάρη να μην αλλάζουν. Με βάση τον Adagrad συνεπώς αναπτύχθηκαν δύο ακόμα πιο αποτελεσματικοί αλγόριθμοι, οι **Adadelta** και **RMSProp**, οι οποίοι, αντί να αποθηκεύουν όλες τις κλίσεις που έχουν υπολογιστεί, χρησιμοποιούν τη μέση τετραγωνική ρίζα αυτών, ενώ διαφέρουν ως προς το learning rate (ο Adadelta δεν χρησιμοποιεί learning rate, ενώ ο RMSProp ναι). Και οι δύο μέθοδοι έχουν εξαιρετικές αποδόσεις και επιλύουν τα βασικά προβλήματα που αντιμετώπιζαν οι GD και SGD.[25, p.5-7]

Adaptive Moment Estimation Optimizer (Adam)

Ο αλγόριθμος Adam αποτελεί ένα συνδυασμό του RMSProp μαζί με την ιδέα του momentum, με σκοπό ακόμα μεγαλύτερη προσαρμοστική ευελιξία. Η διακύμανση που προκαλεί η εισαγωγή του momentum διορθώνεται και ελέγχεται μέσω της διαρκούς αλλαγής του learning rate, δημιουργώντας έτσι έναν εξαιρετικά σταθερό αλγόριθμο, ο οποίος ταυτόχρονα οδηγεί σε ταχύτερο convergence και δεν απαιτεί συνεχή παρακολούθηση από τον τελικό χρήστη για την βελτιστοποίηση του. Για αυτούς τους λόγους, βρίσκει χρήση σε πολλά νευρωνικά δίκτυα.[25, p.7-8]

1.2.6 Perceptron

Ένα απλό παράδειγμα νευρωνικού δικτύου είναι η περίπτωση του perceptron. Είναι απλό σε δομή, σχεδιασμένο για binary classification. Αποτελείται μόνο από input και output layers, που σημαίνει πως δεν υπάρχουν ενδιάμεσοι νευρώνες ανάμεσα στην είσοδο και την έξοδο του δικτύου. Τα δεδομένα εισόδου πολλαπλασιάζονται κατευθείαν με βάρη, αθροίζονται και εισάγονται στην activation function, για να καθοριστεί αν ενεργοποιείται ο νευρώνας. Αν ναι, η τιμή ένα εμφανίζεται στην έξοδο για θετική πρόβλεψη και μηδέν για αρνητική. Τέλος, η έξοδος συγκρίνεται με την είσοδο και πραγματοποιείται backpropagation για διόρθωση βαρών.

1.2.7 Feedforward και Recurrent Neural Networks

Κατά συντριπτική πλειοψηφία, η πληροφορία σε ένα δίκτυο κινείται προς μια κατεύθυνση (εξαιρώντας τη διαδικασία backpropagation), από το input στο output layer. Αυτού του είδους μοντέλα ονομάζονται feedforward neural networks. Σε περιπτώσεις όμως όπου υφίσταται χρονικά συνεχής αλληλουχία δεδομένων, τότε ενδεικνύεται η χρήση των recurrent neural networks. Στα δίκτυα αυτά, η έξοδος του κάθε νευρώνα στο hidden layer αναμεταδίδεται πίσω στην είσοδο τους, εισάγοντας έτσι μνήμη στο μοντέλο, καθώς πλέον δύναται να μάθει από δικές του προβλέψεις.[26] Η επιπλέον πολυπλοκότητα καθιστά τα δίκτυα αυτά πιο ακριβή σε σχέση με τα αντίστοιχα feedforward, με κόστος μεγαλύτερες υπολογιστικές απαιτήσεις, οδηγώντας

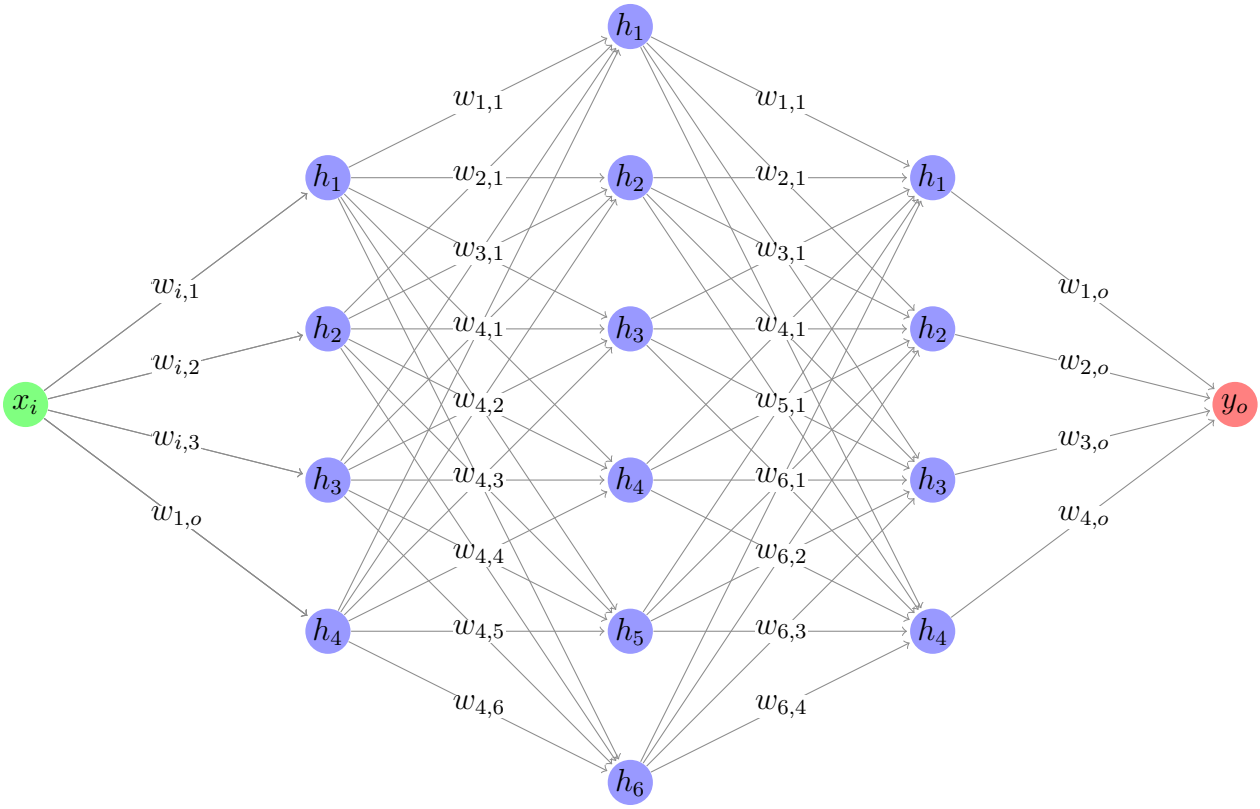
έτσι στην εφαρμογή τους κυρίως σε περιπτώσεις όπου τα δεδομένα του μοντέλου έχουν χρονική εξάρτηση (π.χ. πρόβλεψη τιμής μετοχών σε χρηματιστήριο).

1.3 Βαθιά Νευρωνικά Δίκτυα

1.3.1 Ορισμός

Τα **βαθιά νευρωνικά δίκτυα (deep neural networks, DNN)** είναι αυτά τα οποία αποτελούνται από αριθμό hidden layers μεγαλύτερου του ενός, με σκοπό την ανάλυση πιο πολύπλοκων δεδομένων. Ορίζονται ως δίκτυα απαρτιζόμενα από πολλαπλούς, μη γραμμικούς νευρώνες σε πολλά διαφορετικά στρώματα, ικανά να εξάγουν υψηλού επιπέδου χαρακτηριστικά από δοθέντα δεδομένα.[27]

Αρχιτεκτονική DNN



Σχήμα 1.13: Παράδειγμα αρχιτεκτονικής ενός deep neural networks τριών hidden layers. Η εισαγωγή περισσοτέρων νευρώνων σε fully-connected διάταξη μεταξύ αυτών έχει ως αποτέλεσμα την δυνατότητα επεξεργασίας πιο περίπλοκων δεδομένων, λόγω της ύπαρξης περισσοτέρων weights και συνεπώς μεγαλύτερο πλήθος παραμέτρων προς μελέτη από το δίκτυο.

1.3.2 Είδη DNN

Η εισαγωγή περισσοτέρων hidden layers σε ένα νευρωνικό δίκτυο προσφέρει μεγαλύτερη σχεδιαστική ελευθερία και πολυπλοκότητα. Η γνώση κάποιον βασικών διατάξεων είναι σημαντική, καθώς επιτρέπει στον οποιοδήποτε προγραμματιστή να συνδυάσει ή αλλάξει αρχιτεκτονικές, ανάλογα με τις υπάρχουσες απαιτήσεις. Σημειώνεται πως πολλά παραδείγματα πρόκεινται για απλά νευρωνικά δίκτυα, στα οποία προστέθηκαν επιπλέον στρώματα.

Multi-layer Perceptrons

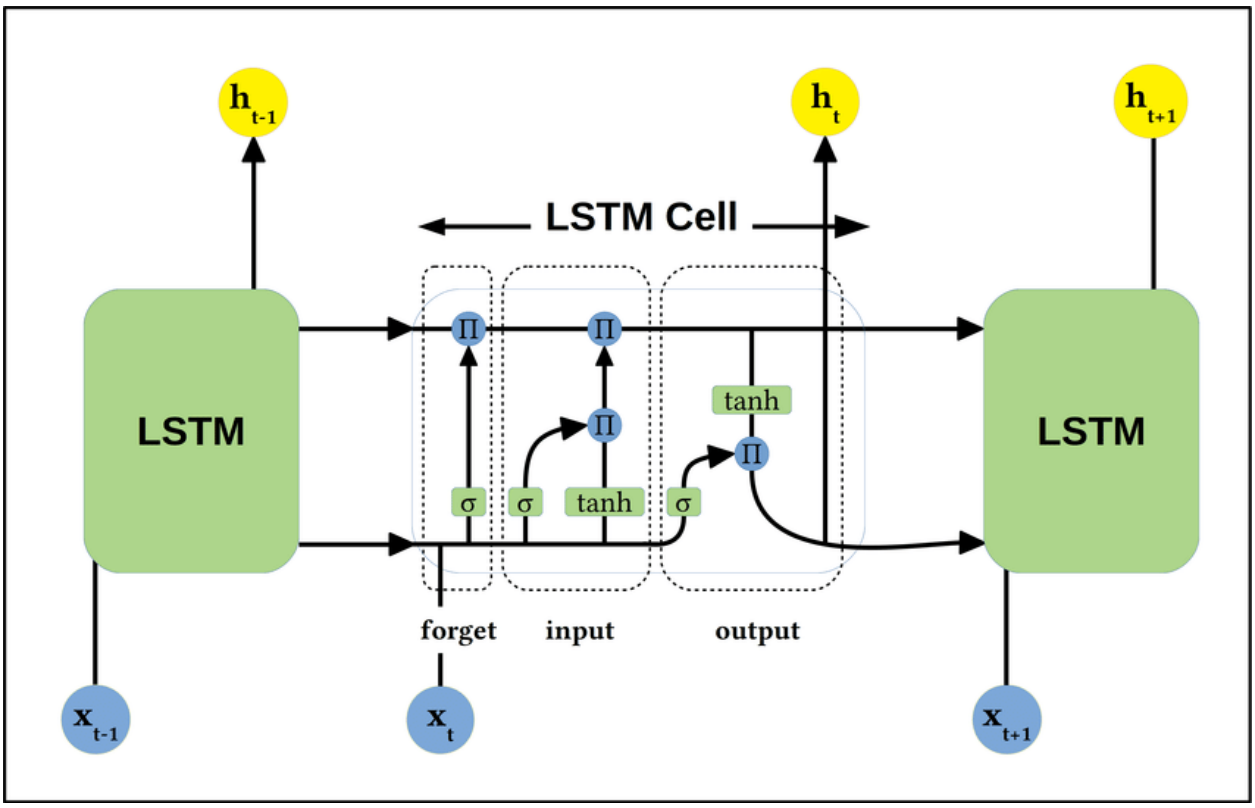
Τα multi-layer perceptrons (MLP) είναι εφαρμογή βαθιάς αρχιτεκτονικής στον απλό perceptron, όπου πλέον γίνεται χρήση ενός ή περισσοτέρων hidden layers, αυξάνοντας έτσι τις δυνατότητες του δικτύου ως προς την ανίχνευση μοτίβων κ.α. Ο αριθμός των hidden layers, καθώς και το πλήθος των νευρώνων σε αυτούς, καθορίζεται άμεσα από την πολυπλοκότητα των δεδομένων, με τα προβλήματα overfitting και underfitting να υφίστανται όπως και στα αντίστοιχα νευρωνικά δίκτυα ενός στρώματος. Στις περισσότερες περιπτώσεις, επιλέγεται αριθμός ενός έως τριών ενδιάμεσων στρωμάτων[28], σε fully connected συνδεσμολογία μεταξύ τους.[29]

Deep RNNs

Πρόκεινται για επέκταση των RNN, στα οποία εισάγονται παραπάνω hidden layers. Χρησιμοποιούνται κυρίως σε εφαρμογές ανάλυσης ανθρωπίνου λόγου (natural language processing, NLP), φωνητικής αναγνώρισης και ταξινόμισης οπτικοακουστικού υλικού.

Long Short Term Memory (LSTM)

Ένα μειονέκτημα των RNN βρίσκεται στη σχετικά μικρή μνήμη που τους προσφέρει η αρχιτεκτονική τους. Αυτό συμβαίνει λόγω της φύσης του gradient descent και της επίδρασής του στα weights του δικτύου. Καθώς τα RNN είναι σχεδιασμένα ώστε να εντοπίζουν χρονικές σχέσεις μεταξύ δεδομένων, παρουσιάζουν ιδιαίτερη ευαισθησία σε τυχόν μεταβολές στα weights τους. Συνεπώς, έπειτα από μακροχρόνιες μεταβολές, το δίκτυο "ξεχνάει" τυχόν συμπεράσματα τα οποία εξήγαγε κατά την λειτουργία του.[30] Για αυτό το λόγο, έχουν προταθεί διάφορες σχεδιαστικές λύσεις για την αντιμετώπιση του συγκεκριμένου προβλήματος. Μια από τις πιο δημοφιλείς είναι αυτή του Long Short Term Memory. Στα δίκτυα αυτά, εισάγεται ένα στοιχείο μνήμης (memory cell), το οποίο επιβλέπει την πληροφορία όπως αυτή μεταφέρεται μέσα στο δίκτυο και επιλέγει ποιο ποσοστό αυτής θα αποθηκευτεί και μεταβιβαστεί. Αυτή η διάταξη επιτρέπει στα LSTM να διατηρούν την εσωτερική μνήμη τους για μεγαλύτερα διαστήματα και για περισσότερες πληροφορίες, εμφανίζοντας ταυτόχρονα αδυναμία να επεξεργαστούν δεδομένα που δεν έχουν λογική συνέχεια στη δομή τους. Εναλλακτική των LSTM είναι τα Bi-LSTM (εκπαίδευση προς δύο κατευθύνσεις στο εσωτερικό του μοντέλου) και τα Gate Recurrent Units (GRU), όπου σκοπός είναι η απλοποίηση των LSTM δικτύων.[31, p.4-6]



Σχήμα 1.14: Αρχιτεκτονική RNN τύπου LSTM. Αποτελείται από τρία διακριτά μέρη. Η forget gate αποφασίζει πόση πληροφορία θα διατηρηθεί από τον προηγούμενο κύκλο λειτουργίας του μοντέλου. Η input gate είναι υπεύθυνη για το ποσοστό πληροφορίας που θα προστεθεί στη μνήμη, ενώ η output gate διαβιβάζει πληροφορία στην έξοδο.[32]

Εφαρμογές σε Unsupervised και Reinforcement Learning

Τα DNN, λόγω των δυνατοτήτων τους, είναι αντικείμενο έρευνας στην δημιουργία unsupervised μοντέλων, με κύριο παράδειγμα τα generative models. Τα μοντέλα αυτά έχουν ως στόχο την δημιουργία output, τα οποία παρομοιάζουν τα input data που χρησιμοποίησαν στην εκπαίδευση. Δύο σημαντικά παραδείγματα είναι αυτά των autoencoders και Generative Adversarial Networks (GAN), με το πρώτο να εφαρμόζεται σε κλασικά προβλήματα unsupervised learning και το δεύτερο να χρησιμοποιείται για παραγωγή εικόνων, σύνθεση λόγου κ.α.
Με την ίδια λογική, η ιδιότητα των DNN να δημιουργούν ουσιώδεις σχέσεις μεταξύ δεδομένων μέσω των νευρώνων και hidden layers, επιτρέπει τον σχεδιασμό πιο εξειδικευμένων και ικανών reinforcement learning μοντέλων, όπου η σύνδεση απόφασης και ανταμοιβής καθορίζει την αποτελεσματικότητά τους.[31, p.6-8]

1.4 Convolutional Neural Networks

Στη μηχανική μάθηση, μια κατηγορία δεδομένων που παρουσιάζει ιδιαίτερο ενδιαφέρον ως προς τη επεξεργασία τους είναι αυτή των οπτικών μέσων (φωτογραφίες και βίντεο). Χρησιμοποιώντας εικόνες, γίνεται δυνατή η ανάλυση τους, με στόχο την ταξινόμηση τους σε κατηγορίες, εντοπισμό αντικειμένων κ.α. Το πρόβλημα βρίσκεται στην πολυπλοκότητα που εμπεριέχεται στις περισσότερες εικόνες, καθώς στην πράξη, το κάθε pixel από τα οποία αποτελούνται, αντιστοιχεί σε πληροφορία την οποία ένα μοντέλο καλείται να επεξεργαστεί, έτσι ώστε να εξάγει κάποιο ουσιώδες συμπέρασμα ως προς τα δεδομένα που δέχεται. Αυτό καθιστά τις εικόνες υπολογιστικά ακριβές, απαιτώντας πολλούς πόρους με τη χρήση παραδοσιακών μοντέλων μηχανικής μάθησης. Ένας τρόπος αντιμετώπισης της δυσκολίας αυτής είναι η χρήση μιας κατηγορίας DNN, τα οποία ονομάζονται **συνελικτικά νευρωνικά δίκτυα** ή όπως αναφέρονται σε πρακτικές εφαρμογές, **convolutional neural networks (CNN)**. Τα δίκτυα αυτά είναι ικανά να επεξεργαστούν εικόνες εξαιρετικής πολυπλοκότητας, διατηρώντας ταυτόχρονα μεγάλο accuracy σε εφαρμογές classification, object detection κ.α. Στη συνέχεια θα γίνει παρουσίαση του τρόπου λειτουργίας τους και η δομή ενός απλού CNN, με στόχο την εξοικείωση του αναγνώστη με τις βασικές αρχές τους.

1.4.1 Οπτικά Μέσα ως Δεδομένα Εκπαίδευσης

Όπως ήδη αναφέρθηκε, οι εικόνες παρουσιάζουν δυσκολίες ως προς τη χρήση τους στη μηχανική μάθηση. Ως προς τη δομή τους, εκφράζονται σαν πίνακες δύο διαστάσεων (ύψος και πλάτος), με το κάθε στοιχείο να αντιστοιχεί σε ένα pixel, με τιμή στο διάστημα $[0, 255]$, που εκπροσωπεί την ένταση του ως προς το χρώμα. Σε ασπρόμαυρες εικόνες, η τιμή 0 μεταφράζεται σε μαύρο, η 255 σε λευκό, και οι ενδιάμεσες σε συνδυασμό των δύο. Η πολυπλοκότητα αυξάνεται όταν η εικόνα είναι έγχρωμη. Στην περίπτωση αυτή, το χρώμα του κάθε pixel προκύπτει από τρεις διαφορετικές τιμές στο ίδιο εύρος, οι οποίες ισοδυναμούν σε κόκκινο, πράσινο και μπλε (RGB). Συνεπώς, ο αριθμός παραμέτρων που διαθέτει μια εικόνα μπορεί να εκφραστεί ως:

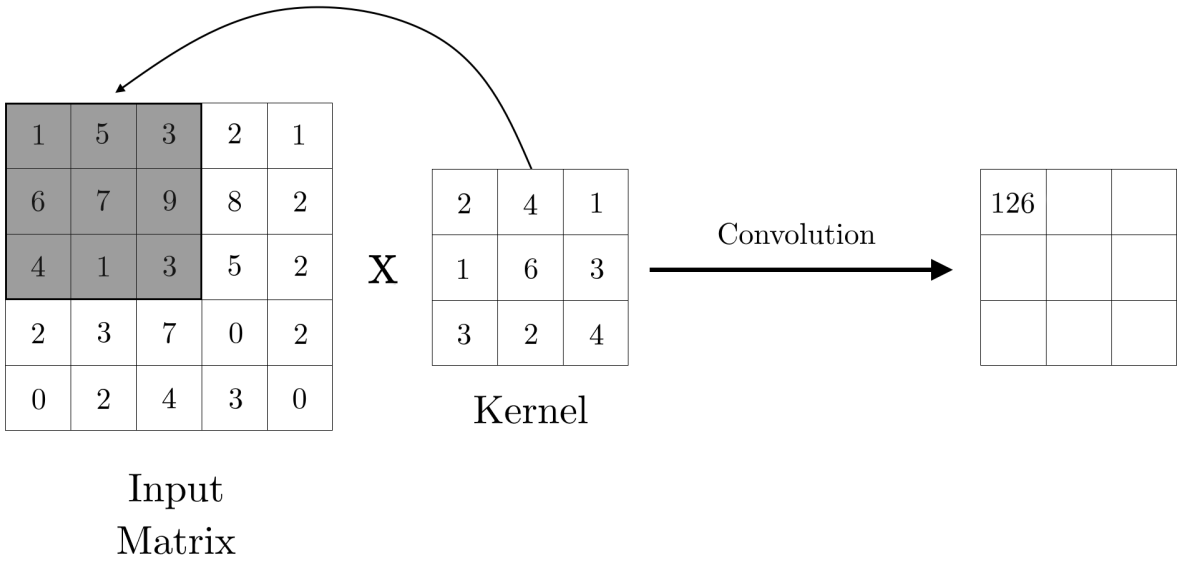
$$Parameters = H \times W \times D, \tag{1.23}$$

όπου H το ύψος (height), W το πλάτος (width) και D το βάθος (depth) της εικόνας. Για παράδειγμα, μια εικόνα 32×32 τύπου RGB ισούται σε $32 \times 32 \times 3 = 3072$ παραμέτρους, όταν αυτή εισέρχεται σε ένα νευρωνικό δίκτυο.

1.4.2 Convolution Layer

Τη λύση ως προς τις υπολογιστικές απαιτήσεις προσφέρει το **convolution layer**, το δομικό στοιχείο των CNN. Στο layer αυτό γίνεται χρήση **kernel** (αλλιώς ονομαζόμενα ως **features**), πινάκων μικρών διαστάσεων και τιμών που αντιστοιχούν σε βάρη, οι οποίοι εφαρμόζονται πάνω στην εικόνα. Τα kernel λειτουργούν ως αντίστοιχο των νευρώνων στο convolution layer, με καθένα από αυτά να επεξεργάζεται ένα συγκεκριμένο τμήμα της κάθε εικόνας που εισέρχεται στο layer. Καθώς το κάθε kernel αναλύει μόνο τμήμα του συνόλου των δεδομένων, τα βάρη που εμπεριέχονται στο layer μειώνονται, που σημαίνει ότι το δίκτυο γίνεται υπολογιστικά πιο αποδοτικό. Το kernel "σκανάρει" κατά μήκος όλης της εικόνας, υπολογίζοντας σε κάθε βήμα

το γινόμενο μεταξύ των τιμών στους πίνακες ανά στοιχείο και προσθέτοντάς τα. Η προκύπτουσα τιμή στη συνέχεια εισάγεται σε έναν τελικό πίνακα. Η διαδικασία αυτή επαναλαμβάνεται αριθμό φορών ίσο με τη τιμή depth του τανυστή που εκφράζει τα δεδομένα εισόδου (π.χ. κάθε εφαρμογή του kernel σε μια περιοχή RGB εικόνας θα επαναληφθεί τρεις φορές για να προκύψει ένα τελικό στοιχείο). Το output κάθε νευρώνα ενός convolution layer ονομάζεται **feature map**. Τα feature maps απεικονίζουν περιοχές οι οποίες προέκυψαν από την εφαρμογή του kernel στην εικόνα, ενώ η εξαγωγή των χαρακτηριστικών από μια εικόνα ονομάζεται **feature extraction**. Ανάλογα με το kernel, τα feature maps μπορούν να αντιστοιχούν στον εντοπισμό διαφόρων χαρακτηριστικών όπως σχήματα ή πιο περίπλοκες περιοχές ενδιαφέροντος.[33, p.5-6] Τέλος, στο output του convolution layer, εφαρμόζεται κάποια activation function (συνήθως συναρτήσεις τύπου ReLU).



Σχήμα 1.15: Διαδικασία convolution για input διαστάσεων 5×5 και kernel 3×3. Το kernel εφαρμόζεται σε περιοχή του input ίση με το μέγεθος του. Αφού γίνει πολλαπλασιασμός ανά στοιχείο, παράγεται η τελική τιμή του feature map. Στη συνέχεια, το kernel προχωράει στην επόμενη στήλη.

Convolution Layer Hyperparameters

Η απόδοση ενός convolution layer σχετίζεται άμεσα με μερικές hyperparameters, οι οποίες επιλέγονται πριν ξεκινήσει η διαδικασία της εκπαίδευσης. Γίνεται ευκόλως κατανοητό πως ο αριθμός των kernel που θα χρησιμοποιηθούν είναι ζωτικής σημασίας, καθώς μεγαλύτερη ποσότητα αυτών μπορεί να οδηγήσει στην αναγνώριση πιο σύνθετων χαρακτηριστικών στα εισερχόμενα δεδομένα, διατρέχοντας όμως τον κίνδυνο το δίκτυο να γίνει υπερβολικά περίπλοκα, προκαλώντας φαινόμενα overfitting. Παρόλα αυτά, υπάρχουν και άλλοι τρόποι να επηρεάσει

κανείς την λειτουργία του layer.[33, p.7-8]

- **Stride και Διάσταση Kernel**: Ο όρος αυτός αναφέρεται στο πόσο μεγάλα "βήματα" ένα kernel πραγματοποιεί όταν προσπελάζει μια εικόνα. Αν για παράδειγμα το stride έχει ως τιμή τη μονάδα, τότε το kernel θα μετακινείται μια θέση τη φορά κατά τη διαδικασία του convolution. Αυτό προσφέρει μεγαλύτερη ακρίβεια (καθώς η εικόνα αναλύεται παραπάνω), αλλά οδηγεί σε μεγαλύτερα feature maps στο output. Την ίδια σημασία σε αυτό το στάδιο έχει και το μέγεθος του kernel, καθώς όσο μεγαλύτερο είναι σε διαστάσεις, τόσο μικρότερο θα είναι το τελικό feature map. Παράδειγμα αποτελεί το σχήμα 1.15, όπου για stride ίσο με μονάδα, το προκύπτον feature map έχει διαστάσεις 3×3.

- **Zero-Padding**: Σε περίπτωση όπου γίνεται χρήση πολλών convolution layers σε ένα νευρωνικό δίκτυο, υπάρχει κίνδυνος να χαθούν πληροφορίες από το ένα layer στο επόμενο λόγω της μείωσης των διαστάσεων που προκύπτει ως αποτέλεσμα του convolution. Προς αποφυγή του φαινομένου αυτού, είναι δυνατή η πλασματική αύξηση των διαστάσεων της εικόνας εισόδου, απλά εισάγοντας νέες σειρές και στήλες στον πίνακα αυτής με όλες τις τιμές του να ισούνται με μηδέν.

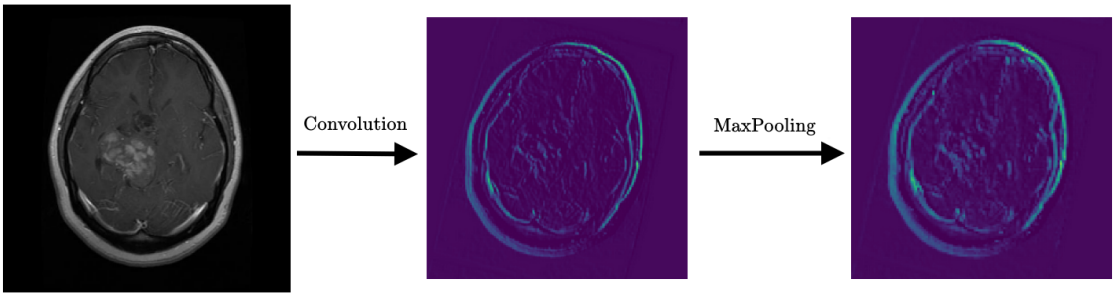
Από συνδυασμό των παραπάνω, εύκολα προκύπτει σχέση που εκφράζει τις διαστάσεις του κάθε feature map στην έξοδο του convolution layer:

$$O = 1 + \frac{I + 2P - F}{S} \tag{1.24}$$

όπου I οι διαστάσεις (H,W) της εικόνας στην είσοδο, P ο αριθμός padding που εισήχθη σε αυτήν, F το μέγεθος του kernel και S το stride. Για παράδειγμα, αν αναλύεται τμήμα 5×5 από kernel διαστάσεων 3×3, stride 1 και padding 1, το τελικό feature map θα έχει διαστάσεις ίσες με αυτές στην είσοδο, διατηρώντας έτσι όλη την πληροφορία του αρχικού δείγματος.

1.4.3 Pooling Layer

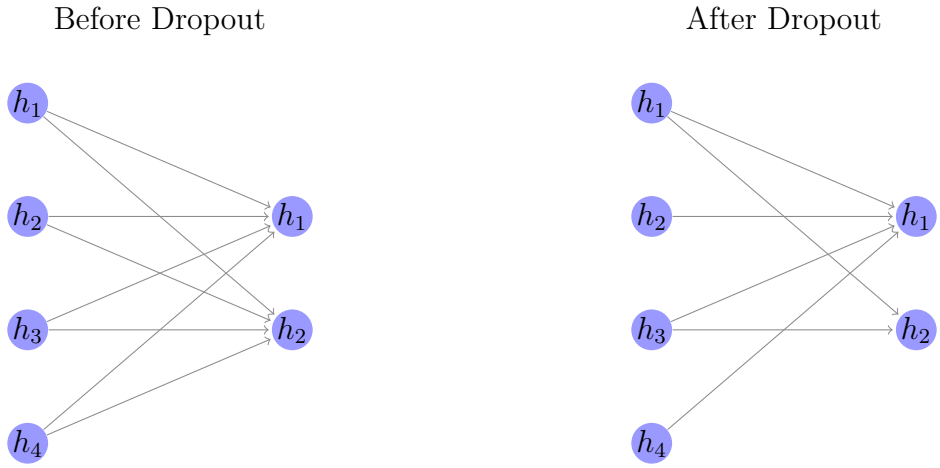
Ο στόχος του pooling layer είναι να μειώσει τις διαστάσεις των feature maps που παράγονται από το convolution layer, με σκοπό την μείωση των παραμέτρων και της πολυπλοκότητας του δικτύου, διατηρώντας ταυτόχρονα τις σημαντικότερες περιοχές ενδιαφέροντος. Το pooling πραγματοποιείται με χρήση kernels τα οποία κινούνται με συγκεκριμένο stride πάνω σε περιοχές του feature map, χωρισμένες σε τετράγωνα. Έπειτα, επιλέγονται τιμές της εκάστοτε περιοχής, ανάλογα με το επιθυμητό είδος pooling και υπόκεινται σε επεξεργασία. Η πιο συχνή μορφή pooling είναι η **MaxPooling**, όπου μόνο η μεγαλύτερη τιμή ανά περιοχή διατηρείται. Άλλα παραδείγματα είναι αυτά του average pooling, όπου η προκύπτουσα τιμή είναι η μέση όλων μέσα στο kernel και το mixed pooling, συνδυασμός των παραπάνω.[34] Κατά τη διαδικασία του pooling, ένα μεγάλο μέρος της πληροφορίας κινδυνεύει να χαθεί, καθώς ο τελικός στόχος είναι η μείωση των διαστάσεων. Συνίσταται λοιπόν χρήση μικρών kernel και stride (συνήθως 2x2, stride 2), για να προσπελαστούν τα δεδομένα επί το πλείστον χωρίς απώλειες.[35]



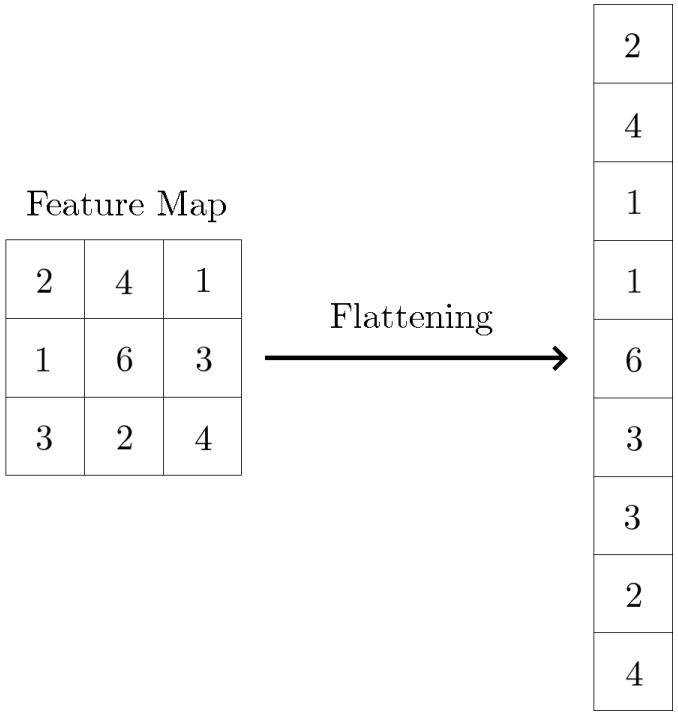
Σχήμα 1.16: Μαγνητική τομογραφία η οποία υπόκειται σε convolution και στη συνέχεια Max-Pooling. Διακρίνονται περιοχές στην περιφέρεια του κρανίου και το εσωτερικό του εγκεφάλου, οι οποίες έχουν οξυνθεί έπειτα του pooling. Αυτό συνεισφέρει στην εξαγωγή χαρακτηριστικών σε μετέπειτα layers.

1.4.4 Μετάβαση σε FC Layers και Output Δικτύου

Σε περιπτώσεις classification και έπειτα από αλληλουχία convolution layers, γίνεται μετάβαση σε fully connected layers, έτσι ώστε τα δεδομένα να είναι στην κατάλληλη μορφή. Για αυτό το λόγο, γίνεται πρώτα επεξεργασία αυτών μέσω μιας διαδικασίας, η οποία ονομάζεται **flattening**, όπου τα δεδομένα μετατρέπονται σε διάνυσμα. Στα περισσότερα CNN γίνεται χρήση τουλάχιστον ενός FC layer πριν το τελικό output, ενώ ταυτόχρονα δίνεται η δυνατότητα χρήσης εργαλείων optimization, προς αποφυγή overfitting. Μια τέτοια τεχνική είναι αυτή του **dropout**, όπου τυχαίοι νευρώνες αφαιρούνται από το δίκτυο (μόνο κατά την εκπαίδευση), με σκοπό την καλύτερη γενίκευση σε άγνωστα δεδομένα.[36] Τέλος, γίνεται εισαγωγή ενός output layer, του οποίου οι νευρώνες και activation function εξαρτώνται από το είδος classification που απαιτείται.

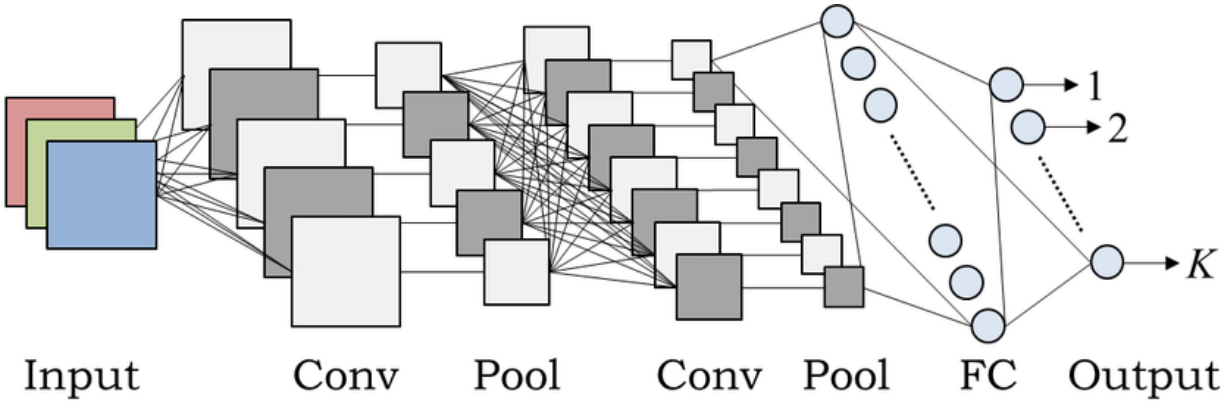


Σχήμα 1.17: Επίδραση μεθόδου dropout παράγοντα 0.25 σε συνδέσεις μεταξύ νευρώνων δύο hidden layers. Παρατηρείται πως το 25% των συνδέσμων αφαιρέθηκε από τη διάταξη.



Σχήμα 1.18: Αποτέλεσμα εφαρμογής flattening σε feature map διαστάσεων 3×3. Τα δεδομένα μετατρέπονται σε μονοδιάστατη μορφή, η οποία μπορεί να χρησιμοποιηθεί από fully-connected layers.

Στο σχήμα 1.19 απεικονίζεται ένα convolutional neural network. Η έγχρωμη εικόνα αναλύεται στα τρία μέρη που την αποτελούν και εισάγεται στο convolution layer. Έπειτα, το output αυτού φιλτράρεται μέσω MaxPooling και τροφοδοτείται στο επόμενο layer. Αφού έρθει εις πέρας η διαδικασία του ολικού convolution και εξαχθεί ο επιθυμητός αριθμός feature maps, αυτά υπόκεινται flattening και εισέρχονται σε fully-connected layer, το οποίο καταλήγει σε κατάλληλο output. Οι περισσότερες εφαρμογές **image classification** ακολουθούν παρόμοια αρχιτεκτονική, με μόνη σημαντική διαφορά των αριθμό convolution και fully-connected layers.



Σχήμα 1.19: Πλήρης αναπαράσταση ενός convolutional neural network.[37]

1.4.5 Εφαρμογές CNN

Ήδη έχει γίνει αναφορά στο πως τα CNN χρησιμοποιούνται για image classification. Όμως, ταυτόχρονα, αποτελούν βάση και για άλλες εφαρμογές, μερικές εκ των οποίων παρατίθενται στη συνέχεια. Πριν την διάδοση των CNN, όλες οι μέθοδοι εντοπισμού αντικειμένων υπήρξαν πολύ αναποτελεσματικές ως προς το υπολογιστικό κόστος. Καθώς όμως, η μέθοδος του convolution μπορεί να εξάγει μοτίβα από δεδομένα με μεγάλη ακρίβεια, αποτελεί μια εξαιρετική λύση στο πρόβλημα αυτό. Κάνοντας χρήση διαφόρων μεθόδων, όπως την οριοθέτηση περιοχών ενδιαφέροντος σε εικόνες με χρήση πλαισίων (object detection), και σε συνδυασμό με άλλες αρχιτεκτονικές βαθιάς μάθησης, ο τομέας έχει πλέον προχωρήσει σε μεγαλύτερη βιωσιμότητα και εύρος πρακτικών εφαρμογών. Με βάση τις ίδιες αρχές είναι δυνατός ο εντοπισμός και η αναγνώριση κειμένου, καθώς και η περιγραφή εικόνων, σε σχέση με το περιεχόμενό τους. Μετατρέποντας επίσης φωνητικά δεδομένα σε εικόνες (π.χ. φασματογραφήματα), τα CNN είναι ικανά να αναγνωρίσουν και να καταγράψουν σε κείμενο ανθρώπινο λόγο, καθώς και να συνθέσουν φωνητικά σήματα. Τέλος, τα CNN εφαρμόζονται και σε παραγωγή κειμένου.[38]

Κεφάλαιο 2

Εργαλεία και Βιβλιοθήκες Ανάπτυξης CNN

Εφόσον πλέον έχει γίνει μια ανάλυση του κλάδου της μηχανικής μάθησης και των convolutional neural networks, μπορεί κανείς πλέον να προχωρήσει στην εκπαίδευση και δημιουργία ενός μοντέλου. Υπάρχουν πολλοί τρόποι ανάπτυξης ενός νευρωνικού δικτύου, όμως λόγω ευκολίας ως προς την προσβασιμότητα και την κατανόησή της, προτείνεται η προγραμματιστική γλώσσα **Python**, καθώς διαθέτει όλα τα απαραίτητα εργαλεία για τον στόχο αυτό. Στη συνέχεια, θα αναλυθούν βασικά εργαλεία, τα οποία χρησιμοποιούνται κατά κόρον στην έρευνα και σε καθημερινές εφαρμογές, όσο αναφορά την δημιουργία αποτελεσματικών image classification μοντέλων.

2.1 Tensorflow

Η βιβλιοθήκη Tensorflow είναι εργαλείο ανάπτυξης και εφαρμογής αλγορίθμων μηχανικής μάθησης. Το Tensorflow προσφέρει τη δυνατότητα εκπαίδευσης τέτοιων αλγορίθμων σε μια πληθώρα συσκευών, από κινητές συσκευές περιορισμένων πόρων, μέχρι υπολογιστές και κάρτες γραφικών χωρίς να απαιτούνται αλλαγές στον κώδικα, προσφέροντας έτσι μεγάλη ευελιξία. Η βιβλιοθήκη επιτρέπει επίσης την ανάπτυξη μοντέλων βαθιάς μάθησης. Κατά τη διάρκεια της εκπαίδευσης, το Tensorflow χρησιμοποιεί πόρους του συστήματος στο οποίο εφαρμόζεται (CPU, GPU και TPU), κατανέμοντας την κατάλληλη μνήμη με σκοπό την διεκπεραίωση της εργασίας. Τα δεδομένα εσωτερικά της βιβλιοθήκης αποθηκεύονται σε μορφή τελεστών (tensors), αυθαίρετων διαστάσεων, οι οποίοι χρησιμοποιούνται για όλες τις εσωτερικές διεργασίες. Είναι επίσης δυνατή η ταυτόχρονη χρήση πολλών συσκευών κατά την εκπαίδευση.[39]

2.2 Keras API

Το Keras είναι API το οποίο ενσωματώθηκε στο Tensorflow 2.0 και αποτελεί εργαλείο που επιτρέπει τον εύκολο σχεδιασμό λύσεων μηχανικής μάθησης με έμφαση στα DNN. Προσφέρει γρήγορη μεταβολή layers, hyperparameters και data processing, επιταχύνοντας σημαντικά τον ρυθμό πειραματισμού με πιθανές διατάξεις δικτύων, απλοποιώντας ταυτόχρονα την όλη διαδικασία, στοχεύοντας σε καθαρό, ευανάγνωστο κώδικα. Η αναφορά σφαλμάτων έχει επίσης

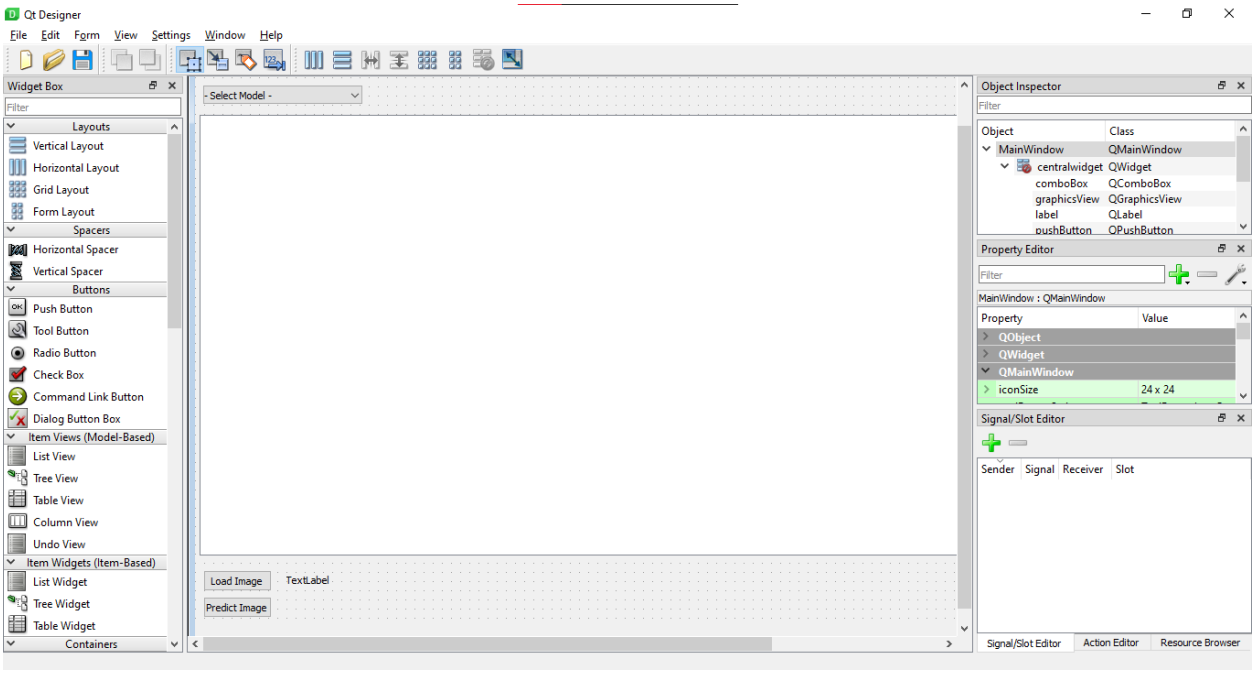
απλοποιηθεί, διευκολύνοντας έτσι το debugging. Η χρήση του επιτρέπει πρόσβαση σε όλες τα απαραίτητα στοιχεία για την δημιουργία ενός δικτύου, από διάφορα είδη layers μέχρι και πληθώρα activation functions, optimizers και μεθόδων metrics.[40]

2.3 Pandas

Εφόσον η εκπαίδευση ενός δικτύου βασίζεται άμεσα στην ποιότητα των δεδομένων τα οποία επεξεργάζεται κατά τη διαδικασία αυτή, είναι απαραίτητη η ύπαρξη εργαλείου που είναι σε θέση να επιτρέψει στον προγραμματιστή να οργανώσει τα δεδομένα αυτά πριν προχωρήσει στη δημιουργία μοντέλου. Μια δημοφιλής βιβλιοθήκη για τον σκοπό αυτό ειναι η pandas, η οποία επιτρέπει την εισαγωγή δεδομένων από διάφορα αρχεία (όπως .xml, .json, .csv κ.α.) σε δομές οι οποίες ονομάζονται dataframes. Τα αρχεία αυτά συνήθως περιέχουν πληροφορίες όπως τις ονομασίες των δεδομένων που θα χρησιμοποιηθούν ως training, validation και test data, καθώς και τις κλάσεις στις οποίες ανήκουν. Επειδή όμως πολλές φορές τα αρχεία αυτά δεν δίνονται σε μορφή ιδανική για εκπαίδευση, απαιτείται η επεξεργασία αυτών. Το Pandas πέρα από την δημιουργία, επιτρέπει και την αλλαγή της διάταξης των dataframes, έτσι ώστε είναι δυνατή η διαγραφή άσκοπων δεδομένων, η μετονομασία λέξεων-κλειδιών κ.α. Το Tensorflow είναι συμβατό με dataframes, αυξάνοντας έτσι την χρησιμότητα του Pandas σε συνδυασμό με αυτό.[41]

2.4 QT Designer

Η Python προσφέρει την δυνατότητα ανάπτυξης γραφικού περιβάλλοντος χρήστη (Graphical User Interface, GUI) μέσω διαφόρων βιβλιοθηκών. Για τις ανάγκες της παρούσας εργασίας, επιλέχθηκε η βιβλιοθήκη PyQT5, η οποία είναι ενσωματωμένη στο πρόγραμμα Qt Designer. Το πρόγραμμα αυτό επιτρέπει την εύκολη ανάπτυξη ενός GUI, καθώς διαθέτει drag and drop αντικείμενα τα οποία αντιστοιχούν σε συγκεκριμένες λειτουργίες όπως παράθυρα pop-up, μενού, κουμπιά, λίστες, προβολή εικόνων κ.α. Αφού επιλεχθούν και τοποθετηθούν όλα τα στοιχεία της εφαρμογής, παράγεται ένα αρχείο επέκτασης .ui, στο οποίο περιέχονται πληροφορίες ως προς τη διάταξη των στοιχείων αυτών και των λειτουργιών για τις οποίες είναι υπεύθυνα. Το αρχείο αυτό στη συνέχεια μπορεί να μετατραπεί σε τύπου Python με χρήση της ενσωματωμένης στο QT εντολής **pyuic5**. Μετά την εξαγωγή, είναι δυνατή η επεξεργασία του προκύπτοντος κώδικα, έτσι ώστε να εισαχθούν αναπτυγμένες λειτουργίες οι οποίες δεν μπορούν να υλοποιηθούν μέσω του περιβάλλοντος του QT Designer.



Σχήμα 2.1: Περιβάλλον QT Designer. Στο κέντρο βρίσκεται η επιφάνεια εργασίας, όπου είναι δυνατή η μετακίνηση των διαφόρων στοιχείων από τα οποία θα αποτελείται η τελική εφαρμογή.

Τα παραπάνω αποτελούν τη βάση των μοντέλων τα οποία δημιουργήθηκαν στα πλαίσια της εργασίας αυτής. Στο επόμενο κεφάλαιο, θα γίνει μια πλήρης ανάλυση δημιουργίας δύο διαφορετικών νευρωνικών δικτύων image classification, binary και categorical. Μέσα από αυτή, θα παρουσιαστούν διάφορες τεχνικές, οι οποίες χρησιμοποιούνται για την βελτίωση της ποιότητας των δεδομένων και του τελικό accuracy του μοντέλου. Στη συνέχεια, θα αξιολογηθούν τα αποτελέσματα μέσω διαφόρων metrics, καθώς και οι πιθανές αλλαγές που μπορούν να γίνουν, για καλύτερες αποδόσεις στο τέλος της εκπαίδευσης. Κλείνοντας, θα αναλυθεί ένα γραφικό περιβάλλον χρήστη, το οποίο επιτρέπει την χρήση των μοντέλων που αναπτύχθηκαν απουσία εξειδίκευσης ή γνώσεων προγραμματισμού.

Κεφάλαιο 3

Παραδείγματα Ανάπτυξης και Εφαρμογής CNN σε Image Classification

Εφόσον πλέον υφίστανται σωστές βάσεις ως προς το θεωρητικό υπόβαθρο των DNN, είναι δυνατή η δημιουργία αυτών. Καθώς αποτελούν τον στόχο της παρούσας εργασίας, το τρέχον κεφάλαιο θα επικεντρωθεί στη διαδικασία ανάπτυξης δικτύων image classification δύο τύπων: binary και categorical. Οι παρακάτω διατάξεις έχουν βελτιστοποιηθεί με στόχο την μέγιστη δυνατή απόδοση, με ταυτόχρονη μείωση φαινομένων overfitting και underfitting. Η ανάπτυξη έγινε με τη χρήση του προγράμματος JupyterLab, στο οποίο είναι δυνατή η δημιουργία notebooks, όπου ο προγραμματιστής μπορεί να τρέχει τμήματα κώδικα ξεχωριστά, με σκοπό την επιτόπου δοκιμή σε πιθανές αλλαγές. Αν και τα παρακάτω είναι ενδεικτικά (καθώς εξαρτώνται άμεσα από τα δεδομένα που χρησιμοποιούνται), προτείνεται χρήση αυτών από αρχάριους, καθώς είναι δομημένα έτσι ώστε όχι μόνο να δημιουργούν και να βαθμονομούν ένα μοντέλο με λίγες αλλαγές, αλλά επίσης είναι αρκετά απλά έτσι ώστε κάποιος να μπορεί να πειραματιστεί εύκολα με άλλες διατάξεις layers, τεχνικές processing κλπ. Αφού γίνει ανάλυση του κώδικα, θα παρουσιαστούν τα σχετικά αποτελέσματα, τα οποία και θα αξιολογηθούν. Οι βάσεις δεδομένων που χρησιμοποιούνται είναι διαθέσιμες για το κοινό στην ιστοσελίδα **Kaggle**.

3.1 Binary Image Classification για Εντοπισμό Εγκεφαλικών Όγκων

Το πρώτο παράδειγμα που θα εξεταστεί είναι ένα binary classification DNN πάνω σε μια βάση δεδομένων εγκεφαλικών όγκων.[42] Η βάση αυτή είναι ισορροπημένη, με τις δύο classes να έχουν σχεδόν τον ίδιο αριθμό δειγμάτων, κάτι που την καθιστά ιδανική για δημιουργία δικτύων με εν γένει μεγάλη ακρίβεια. Η ιδιότητα αυτή είναι επίσης χρήσιμη στην εκμάθηση δημιουργίας DNN, καθώς η απόδοση δεν παρουσιάζει μεγάλη ευαισθησία ως προς τη δομή του δικτύου.

3.1.1 Εισαγωγή Βιβλιοθηκών

Το πρώτο βήμα στην εκπαίδευση είναι η επιλογή κατάλληλων βιβλιοθηκών, οι οποίες προσφέρουν τα κατάλληλα εργαλεία για την επεξεργασία δεδομένων και δημιουργία του τελικού μοντέλου. Παρακάτω, θα παρατεθούν κάποιες από αυτές, καθώς και η χρησιμότητα τους.

```
#Module Initialization
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import random
import shutil
import kaggle
import time
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import (Conv2D, Dense, Flatten, Dropout, Activation,
                                     MaxPooling2D, RandomFlip, RandomRotation)
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
```

Σχήμα 3.1: Οι βιβλιοθήκες που χρησιμοποιήθηκαν στη δημιουργία του DNN.

Βιβλιοθήκες Γενικής Χρήσης

- **OS**: Η βιβλιοθήκη **os** χρησιμοποιείται ως εργαλείο για την αλληλεπίδραση του κώδικα με το λειτουργικό σύστημα του υπολογιστή. Επιτρέπει τη δημιουργία και επεξεργασία directories, κάτι εξαιρετικά χρήσιμο στην οργάνωση δεδομένων. Μετά την εισαγωγή της, η εντολή που την ακολουθεί απαγορεύει την τύπωση κάποιων μηνυμάτων σχετικών με το Tensorflow, τα οποία είναι διαγνωστικές πληροφορίες.

- **random**: Η βιβλιοθήκη **random** παράγει ψευδό-τυχαίους αριθμούς και χρησιμοποιείται για κατανομή των διαθέσιμων δειγμάτων σε training, validation και test data.

- **shutil**: Η **shutil** είναι ιδιαίτερα χρήσιμη σε εφαρμογές αντιγραφής ή μεταφοράς αρχείων, κάτι που απαιτείται πολύ συχνά κατά την επεξεργασία των δεδομένων πριν την εκπαίδευση.

- **kaggle**: Η ιστοσελίδα Kaggle προσφέρει την δυνατότητα χρήσης βιβλιοθήκης Python, με σκοπό την καταφόρτωση των επιθυμητών βάσεων δεδομένων απευθείας μέσω του κώδικα ως διεπαφή. Ο ενδιαφερόμενος χρήστης γράφεται στη σελίδα και με τη χρήση ενός μοναδικού token είναι σε θέση να κατεβάσει και να κάνει επιτόπου εξαγωγή των δεδομένων.[43]

- **time**: Πρόκειται για απλή βιβλιοθήκη που χρησιμοποιήθηκε για τη χρονομέτρηση της διαδικασίας εκπαίδευσης.

- **pandas**: Όπως αναφέρθηκε και παραπάνω, η pandas είναι μια εξαιρετικά σημαντική βιβλιοθήκη, καθώς η χρήση dataframes απλοποιεί τη διαδικασία εκμάθησης.
- **numpy**: Η **numpy** είναι υπολογιστική βιβλιοθήκη που επιτρέπει την χρήση πολυδιάστατων πινάκων, δομικών στοιχείων της Tensorflow.[44]
- **matplotlib.pyplot**: Η **matplotlib** (συνήθως αναφερόμενη ως **plt**) χρησιμοποιείται για το σχεδιασμό γραφικών παραστάσεων και άλλων στοιχείων.[45]

Preprocessing

Το Keras επιτρέπει την προεπεξεργασία εικόνων όπως μεγέθυνση, αλλαγή διαστάσεων και φωτεινότητας, περικοπή, επαναπροσανατολισμός κ.α, πριν αυτές εισαχθούν στο μοντέλο για εκπαίδευση. Αυτό στη προκειμένη περίπτωση επιτυγχάνεται μέσω της κλάσης **ImageDataGenerator**, ενώ σημειώνεται πως υπάρχουν και άλλοι μέθοδοι, αναλόγως της δομής των δεδομένων.

Keras Layers

Το Keras προσφέρει μια πληθώρα layers, έτσι ώστε να επιτρέπει την δημιουργία πολλών διακριτών δικτύων. Παρακάτω εξηγούνται αναφορικά τα layers που εισήχθησαν στο πρόγραμμα, λειτουργίες των οποίων θα αναλυθούν περαιτέρω κατά την ανάλυση της αρχιτεκτονικής του μοντέλου.

- **Conv2D**: Layer που πραγματοποιεί convolution σε δισδιάστατα δεδομένα (π.χ εικόνες).
- **Dense**: Απλό fully connected layer.
- **Flatten**: Στρώμα που έχει ως ρόλο το flattening των δεδομένων όταν εξέρχονται από convolution layers.
- **Dropout**: Ρυθμιζόμενο στρώμα που εισάγεται έπειτα άλλων με σκοπό την εκτέλεση dropout στους νευρώνες.
- **Activation**: Στοιχείο που πραγματοποιεί activation σε προηγούμενο layer.
- **MaxPooling2D**: Χρησιμοποιείται για διαδικασία pooling σε δισδιάστατα στοιχεία.
- **RandomFlip**: Layer προεπεξεργασίας που αλλάζει τυχαία τον προσανατολισμό των εισακτέων εικόνων.
- **RandomRoatation**: Όπως και το **RandomFlip**, επεξεργάζεται την εικόνα πριν την έναρξη της εκπαίδευσης, περιστρέφοντας την.

Keras Models

Το Keras διαθέτει τρία είδη διατάξεων για τη δημιουργία ενός μοντέλου. Σε περιπτώσεις όπου είναι επιθυμητή η χρήση γραμμικής αλληλουχίας layers, προτείνεται η εισαγωγή του **Sequential** μοντέλου, κατάλληλο για μοντέλα image classification λόγω αυτής του της ιδιότητας.

Callbacks

Όπως αναφέρθηκε στην υποενότητα περί backpropagation (1.2.5), υπάρχει η πιθανότητα ένα μοντέλο να σταματήσει την βελτίωσή του όσο συνεχίζεται η διαδικασία της εκπαίδευσης. Για την εξοικονόμηση χρόνου και την πιο αποτελεσματική μεταβολή των hyperparameters ενός δικτύου, υπάρχουν κάποια εργαλεία τα οποία παρακολουθούν την πρόοδο του ανά epoch, έτσι ώστε να διακόψουν την εκπαίδευση σε περιπτώσεις στασιμότητας. Αυτές οι μέθοδοι εφαρμόζονται μετά από κάθε epoch και ονομάζονται **callbacks**. Στο συγκεκριμένο μοντέλο γίνεται χρήση τριών εκ'αυτών:

- **EarlyStopping**: Η διαδικασία του early stopping έχει στόχο την διακοπή της εκπαίδευσης, πριν αυτή εκπληρώσει τον αριθμό epochs που έχει οριστεί ως hyperparamater, επιβλέποντας την βελτίωση κάποιας διαγνωστικής τιμής όπως loss, accuracy κ.α.

- **ModelCheckpoint**: Η μέθοδος αυτή αποθηκεύει αυτόματα ένα μοντέλο έπειτα από κάθε epoch στο οποίο η επιβλεπόμενη τιμή παρουσιάζει καλύτερες αποδόσεις σε σχέση με προηγούμενος κύκλους. Εφαρμόζεται συνήθως μαζί με early stopping έτσι ώστε το τελικό μοντέλο να είναι πάντα το βέλτιστο δυνατό.

- **ReduceLROnPlateau**: Όπως αναλύθηκε και στο θεωρητικό μέρος της παρούσας εργασίας, μια εξαιρετικά σημαντική hyperparameter είναι αυτή του learning rate, καθώς η τιμή της επηρεάζει το backpropagation και συνεπώς την δυνατότητα βελτίωσης του μοντέλου κατά την εκπαίδευση. Σε περιπτώσεις όπου παρατηρείται στασιμότητα ή αδυναμία εξαγωγής συμπερασμάτων, είναι δυνατή η δυναμική μείωση του learning rate, σε μια προσπάθεια πιο ομαλού convergence. Αυτή τη λειτουργία προσφέρει το ReduceLROnPlateau, το οποίο, όπως και τα παραπάνω, εφαρμόζεται ανάλογα με τη μεταβολή ή μη κάποιας διαγνωστικής τιμής.

3.1.2 Επεξεργασία Βάσης Δεδομένων για Εκπαίδευση

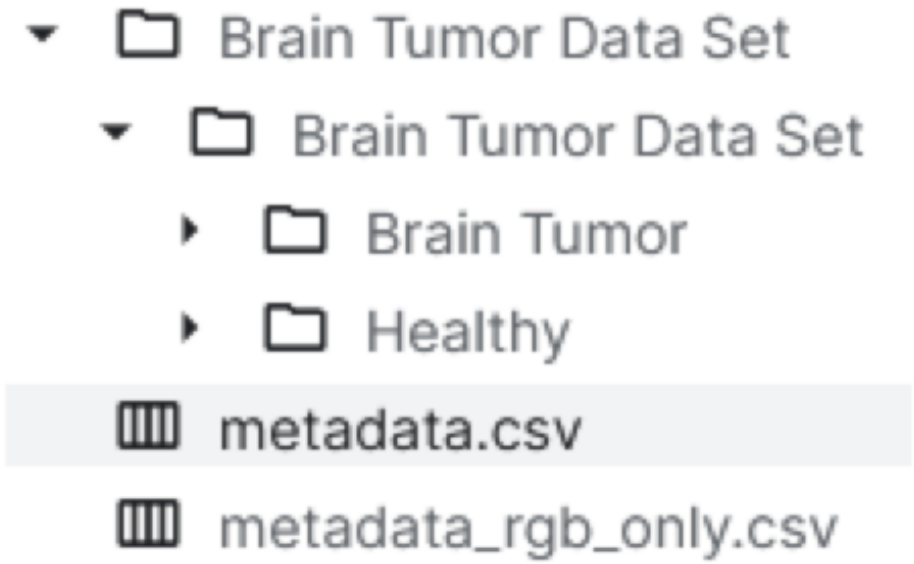
Εφόσον οι σχετικές βιβλιοθήκες έχουν επιλεγεί και εισαχθεί στο μοντέλο, το επόμενο βήμα είναι η επεξεργασία των διαθέσιμων δεδομένων, έτσι ώστε να βρίσκονται σε μορφή κατάλληλη για να διαπεραστεί από το δίκτυο. Ξεκινώντας, πρέπει να οριστούν οι διαδρομές στις οποίες βρίσκονται. Στην συγκεκριμένη περίπτωση, αυτό σημαίνει απόκτηση της βάσης δεδομένων από το Kaggle, χρησιμοποιώντας το API που προσφέρεται από την πλατφόρμα.

```
dl_path = r'/path/to/folder'

if not os.path.exists(dl_path):
    kaggle.api.authenticate()
    kaggle.api.dataset_download_files('preetviradiya/brian-tumor-dataset', path = dl_path, unzip=True)
    print('Files successfully downloaded!')
else:
    print('Data has already been downloaded.')
```

Σχήμα 3.2: Κώδικας υπεύθυνος για την καταφόρτωση και εξαγωγή των δεδομένων μέσω Kaggle.

Αφού οριστεί η διαδρομή (εδώ ως variable ονόματος dl_path), επιλέγεται η βάση δεδομένων που θα χρησιμοποιηθεί κατά τη διαδικασία εκμάθησης. Πρώτα, αναγνωρίζεται το token του χρήστη (authenticate), καταγράφεται στον κώδικα το URL στο οποίο βρίσκονται τα δεδομένα, τα οποία κατεβαίνουν αυτόματα, με τη δυνατότητα απευθείας εξαγωγής τους (unzip = True). Εφόσον για ένα άγνωστο σετ δειγμάτων υπάρχει μεγάλη πιθανότητα να γίνουν αρκετές μεταβολές στην διαμόρφωση του δικτύου και στις hyperparameters του, προτείνεται η χρήση εντολών if-else, έτσι ώστε να μην χρειάζεται να κατεβαίνουν εκ νέου τα αρχεία.

Η βάση δεδομένων που χρησιμοποιείται στο παράδειγμα αυτό δεν βρίσκεται σε μορφή κατάλληλη για εισαγωγή σε νευρωνικό δίκτυο. Σημαντικό είναι λοιπόν ένας χρήστης να βλέπει τα διαθέσιμα αρχεία, σε περίπτωση που απαιτούνται περισσότερα προπαρασκευαστικά βήματα. Εδώ, αν και τα αρχεία μπορούν να περαστούν ως έχει στο μοντέλο, δεν υπάρχει η δυνατότητα να χρησιμοποιηθεί ένα υποσύνολό τους ως test data (καθώς το ImageDataGenerator επιτρέπει μόνο διαχωρισμό σε training και validation). Συνεπώς, για τη σωστή αξιολόγηση του τελικού νευρωνικού δικτύου, θα πρέπει να γίνει επέμβαση στη δομή της βάσης.



Σχήμα 3.3: Η κατανομή των δεδομένων, όπως αυτά είναι διαθέσιμα. Αν και δύναται η χρήση τους στην συγκεκριμένη μορφή, είναι προτιμητέα η κατανομή σε υποσύνολα για λόγους αξιολόγησης και επαλήθευσης της σωστής λειτουργίας του μοντέλου στο τέλος.

```
data_path = dl_path + 'Brain Tumor Data Set/Brain Tumor Data Set/'
tumor_path = data_path + 'Brain Tumor/'
healthy_path = data_path + 'Healthy/'
main_folder = dl_path + 'Segmented Images/'

if not os.path.exists(main_folder):
    #Pathing the folders necessary
    train_folder = os.path.join(main_folder, 'train')
    val_folder = os.path.join(main_folder, 'val')
    test_folder = os.path.join(main_folder, 'test')

    #Defining the file format of the images
    file_formats = ['.jpg', '.JPG', '.tif', '.png']

    #List of image filenames
    imgs_list = [filename for filename in os.listdir(tumor_path) if os.path.splitext(filename)[-1] in file_formats]
    imgs_list2 = [filename for filename in os.listdir(healthy_path) if os.path.splitext(filename)[-1] in file_formats]

    #Randomize the three sets using a seed and the image list generated above
    random.seed(42)
    random.shuffle(imgs_list)
    random.shuffle(imgs_list2)
```

Σχήμα 3.4: Πρώτο βήμα κατανομής αρχείων σε training, validation και test data. Παρομοίως με προηγουμένως, γίνεται χρήση if-else προς αποφυγή εκτέλεσης του συγκεκριμένου κώδικα σε εκ νέου εφαρμογή του τελικού προγράμματος.

Αρχικά, ορίζονται οι διαδρομές, όπως αυτές υφίστανται στα αποκτηθέντα δεδομένα. Έπειτα, δίνεται η διαδρομή στην οποία θα κατανεμηθούν τα αρχεία σε φακέλους. Οι τύποι των εικόνων που είναι διαθέσιμες καταγράφονται σε Python list και έπειτα όσα αρχεία είναι των μορφών αυτών επεξεργάζονται παρομοίως για τις 2 κλάσεις. Στη συνέχεια, οι 2 λίστες με τα ονόματα των αρχείων υφίστανται τυχαιοποίηση, έτσι ώστε να επιτευχθεί στο τέλος τυχαία κατανομή στα επιθυμητά υποσύνολα.

```
#Split data in ratios: 70% train, 15% validation, 15% test
train_size = int(len(imgs_list) * 0.70)
val_size = int(len(imgs_list) * 0.15)
test_size = int(len(imgs_list) * 0.15)

train_size2 = int(len(imgs_list2) * 0.70)
val_size2 = int(len(imgs_list2) * 0.15)
test_size2 = int(len(imgs_list2) * 0.15)
```

Σχήμα 3.5: Ορισμός ποσοστών κατανομής υποσυνόλων.

Μια σημαντική επιλογή είναι αυτή της αναλογίας μεταξύ των υποσυνόλων στην εκπαίδευση. Σύνηθες είναι τα training data να αποτελούν περίπου το 70-80% των δεδομένων, με τα υπόλοιπα να χωρίζονται σε validation και test. Εδώ, ο αριθμός των διακριτών αρχείων στις λίστες που ορίστηκαν παραπάνω πολλαπλασιάζεται με το τελικώς επιλεγμένο ποσοστό για τις 2 κατηγορίες εικόνων.

```
    # Create destination folders if they don't exist
    for folder_path in [train_folder, val_folder, test_folder]:
        if not os.path.exists(folder_path):
            os.makedirs(folder_path)

    # Copy image files to destination folders
    for i, f in enumerate(imgs_list):
        if i < train_size:
            dest_folder = train_folder
        elif i < train_size + val_size:
            dest_folder = val_folder
        else:
            dest_folder = test_folder
        shutil.move(os.path.join(tumor_path, f), os.path.join(dest_folder, f))

    for i, f in enumerate(imgs_list2):
        if i < train_size2:
            dest_folder = train_folder
        elif i < train_size2 + val_size2:
            dest_folder = val_folder
        else:
            dest_folder = test_folder
        shutil.move(os.path.join(healthy_path, f), os.path.join(dest_folder, f))
    shutil.rmtree(tumor_path)
    shutil.rmtree(healthy_path)
    shutil.rmtree(dl_path + 'Brain Tumor Data Set/')
else:
    print('Folders already exist!')
```

Σχήμα 3.6: Κώδικας υπεύθυνος για την κατανομή των αρχείων με χρήση των προηγουμένως ορισμένων μεταβλητών.

Αφού πρώτα ελεγχθεί η ύπαρξη ή μη του τελικού φακέλου, είναι δυνατόν πλέον τα αρχεία να διαχωριστούν. Ο κώδικας συγκρίνει τη θέση του κάθε αρχείου στις λίστες που δημιουργήθηκαν προηγουμένως με τα ποσοστά που ορίστηκαν και μεταφέρει το καθένα στον κατάλληλο φάκελο. Έπειτα, οι αρχικές διαδρομές διαγράφονται για λόγους οργάνωσης.

3.1.3 Εισαγωγή Δεδομένων σε Tensorflow

Καθώς τα δεδομένα προς χρήση δεν εμφανίζουν άλλες ιδιαιτερότητες, μπορούν πλέον να προσπελαστούν μέσω του ImageDataGenerator σε μορφή κατάλληλη για εκπαίδευση.

```
df = pd.read_csv(dl_path + 'metadata.csv')
df = df.rename(columns={"class": "status"})
#Assign classes to the imagesets by cross-referencing file names with the "class" column from the CSV file
#and prepare images for overall processing from the neural network.

datagen = ImageDataGenerator(rescale=1./255)
train_data = datagen.flow_from_dataframe(dataframe = df, directory = main_folder + 'train/', x_col = "image", y_col = "status",
                                         class_mode = "binary", target_size = (256, 256), batch_size = 32)
valid_data = datagen.flow_from_dataframe(dataframe = df, directory = main_folder + 'val/', x_col = "image", y_col = "status",
                                         class_mode = "binary", target_size = (256, 256), batch_size = 32)
test_data = datagen.flow_from_dataframe(dataframe = df, directory = main_folder + 'test/', x_col = "image", y_col = "status",
                                        class_mode = "binary", target_size = (256, 256), batch_size = 32)
```

```
Found 3215 validated image filenames belonging to 2 classes.
Found 688 validated image filenames belonging to 2 classes.
Found 691 validated image filenames belonging to 2 classes.
/anaconda3/envs/tf-gpu/lib/python3.11/site-packages/keras/src/preprocessing/image.py:1137: UserWarning: Found 1385
invalid image filename(s) in x_col="image". These filename(s) will be ignored.
  warnings.warn(
/anaconda3/envs/tf-gpu/lib/python3.11/site-packages/keras/src/preprocessing/image.py:1137: UserWarning: Found 3912
invalid image filename(s) in x_col="image". These filename(s) will be ignored.
  warnings.warn(
/anaconda3/envs/tf-gpu/lib/python3.11/site-packages/keras/src/preprocessing/image.py:1137: UserWarning: Found 3909
invalid image filename(s) in x_col="image". These filename(s) will be ignored.
  warnings.warn(
```

Σχήμα 3.7: Χρήση ImageDataGenerator για κανονικοποίηση και δημιουργία τανυστών από διαθέσιμα δεδομένα.

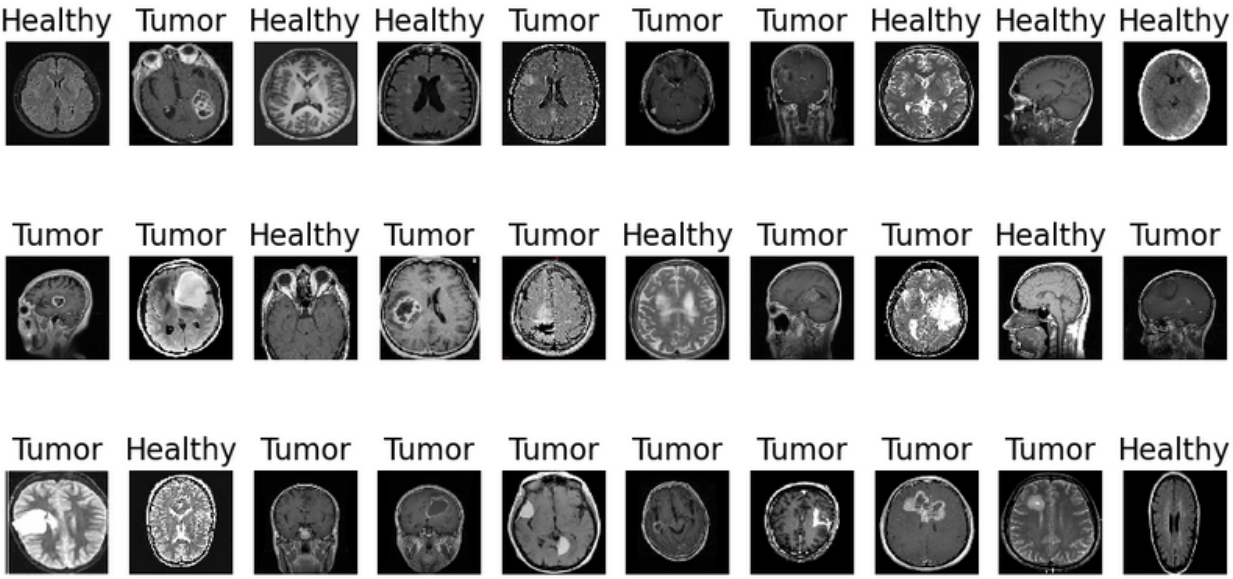
Πρώτο βήμα είναι η αξιοποίηση του αρχείου .csv, στο οποίο βρίσκονται τα ονόματα όλων των εικόνων μαζί με την κατηγορία στην οποία ανήκουν. Μέσω Pandas, το αρχείο αυτό εισάγεται σε μορφή dataframe, η οποία είναι συμβατή για χρήση με το ImageDataGenerator. Σε πολλές περιπτώσεις, αυτό δεν είναι αρκετό, καθώς μπορεί οι πληροφορίες που χρειάζονται να είναι σε κακιά μορφή (π.χ απουσία μορφής αρχείου στο όνομα, ελλείπείς πληροφορίες κ.α). Για ενδεικτικούς λοιπόν λόγους, παρουσιάζεται μια απλή μετονομασία στήλης του προκύπτοντος dataframe. Έπειτα, ορίζονται οι επεξεργαστικές μέθοδοι που θα εκτελέσει το ImageDataGenerator. Η πιο συχνή επεξεργασία στην οποία υπόκεινται εικόνες προς χρήση σε CNN είναι αυτή της **κανονικοποίησης** (**normalization ή rescaling**). Όπως αναφέρθηκε και στην υποενότητα περί οπτικών μέσων (1.4.1), κάθε pixel αντιστοιχεί σε τιμή διαστήματος $[0, 255]$, ως προς την ένταση του χρώματός του. Εφόσον όμως κάθε εικόνα, όταν αυτή εισάγεται σε ένα convolution layer, παράγει feature maps ίσα με τον αριθμό των filters που ορίστηκαν ως hyperparameters, διατρέχεται κίνδυνος μεγάλης διαφοράς στην κατανομή των τιμών αυτών, με αποτέλεσμα την δυσκολία κατά το backpropagation. Συνεπώς, είναι προτιμητέα η αλλαγή της κλίμακας των δυνατών τιμών σε διάστημα $[0, 1]$, έτσι ώστε να αποφευχθούν πιθανές υπερδιορθώσεις βαρών. Με τη μέθοδο αυτή αποφεύγονται επίσης περιπτώσεις όπου μετά απο convolution, το προκύπτον feature map εμπεριέχει τιμές που ξεπερνάνε το 255, καθώς αυτές δεν αντιστοιχούν σε πραγματικές τιμές pixel. Στη συνέχεια, καλείται η μέθοδος **flow_from_dataframe**, η οποία παράγει τους τανυστές που θα εισαχθούν στο δίκτυο με βάση το dataframe που δημιουργήθηκε προηγουμένως, όπου επιλέγονται στήλες αυτού τις οποίες εμπεριέχονται τα ονόματα των αρχείων και η κλάση τους. Σε αυτό το σημείο είναι δυνατή η επιλογή των δι-

αστάσεων των εικόνων, το χρωματικό τους προφίλ (ασπρόμαυρες ή έγχρωμες), ο τρόπος της τελικής ταξινόμησης που θα εκτελεί το μοντέλο κ.α. Τα αρχεία και η class στην οποία ανήκουν ορίζονται μέσω των **x_col** και **y_col**, με την μεταβλητή να αντιστοιχεί στην κατάλληλη στήλη του dataframe που δημιουργήθηκε. Μια από τις πιο σημαντικές hyperparamaters σε αυτό το στάδιο είναι οι διαστάσεις των εικόνων. Μεγαλύτερη ευκρίνεια απαιτεί και αύξηση στον αριθμό feature maps που θα πρέπει να δημιουργηθούν, αυξάνοντας έτσι τις υπολογιστικές απαιτήσεις. Επιπροσθέτως, οι μεγαλύτερες διαστάσεις δεν μεταφράζονται άμεσα σε καλύτερες αποδόσεις από το τελικό μοντέλο. Συνεπώς, συνίσταται ιδιαίτερη προσοχή ως προς τον ορισμό της hyperparamater αυτής. Εξίσου σημαντικό είναι το **batch size**, δηλαδή ο αριθμός δειγμάτων που εισέρχεται στο δίκτυο ανά φορά. Τα δείγματα χωρίζονται σε πακέτα (batches) ορισμένου μεγέθους και κάθε epoch θεωρείται λήξαν αφού όλα τα batches έχουν αναλυθεί από το δίκτυο. Μεγάλυτερο batch size οδηγεί σε καλύτερη μεταβολή του gradient descent μέσω του optimizer, αλλά έχει ως μειονέκτημα το μεγαλύτερο χρόνο εκπαίδευσης ή και αδυναμία αυτής αν το σύστημα δεν διαθέτει αρκετούς πόρους. Εν συνεχεία, θα πρέπει κανείς να επιβεβαιωθεί πως οι εικόνες έχουν επεξεργαστεί επιτυχώς. Αυτό επιτυγχάνεται καλώντας έναν από τους τανυστές που δημιουργήθηκαν προηγουμένως και προβάλλοντας δείγμα των δεδομένων που περιέχει.

```python
# Code that ensures we correctly loaded the images to be used for training.
# This is used for visualization purposes.
def plots(ims, figsize=(12,6), rows=3, interp=False, titles=None):
    if type(ims[0]) is np.ndarray:
        ims = np.array(ims)
        if (ims.shape[-1] != 3):
            ims = ims.transpose((0,2,3,1))
    f = plt.figure(figsize=figsize)
    cols = len(ims)//rows if len(ims) % 2 == 0 else len(ims)//rows + 1
    for i in range(len(ims)):
        try:
            sp = f.add_subplot(rows, cols, i+1)
            sp.axis('Off')
        except ValueError:
            break
        if titles is not None:
            sp.set_title(titles[i], fontsize=16)
        plt.imshow(ims[i], interpolation=None if interp else 'none')


imgs, labels = next(train_data)
labels_new = []
for i in range(0,len(labels)):
    if np.allclose(labels[i],np.array([1])) == True:
        labels_new.append('Tumor')
    elif np.allclose(labels[i],np.array([0])) == True:
        labels_new.append('Healthy')
plots(imgs, titles = labels_new)
```

Σχήμα 3.8: Πιθανός τρόπος προβολής εικόνων τανυστή από ImageDataGenerator. Ο ορισμός της μεθόδου ως function προσφέρει τη δυνατότητα απεικόνισης δεδομένων και από τα άλλα υποσύνολα.

Στη μέθοδο αυτή γίνεται χρήση της βιβλιοθήκης **Numpy**, καθώς είναι δυνατή η εφαρμογή των λειτουργιών της σε τανυστές. Αρχικά, δημιουργείται μια function η οποία δέχεται ως όρισμα τα δεδομένα προς απεικόνιση, τον αριθμό αυτών, καθώς και οι τίτλοι τους. Αφού πρώτα η είσοδος ελεγχθεί, έτσι ώστε να είναι στην κατάλληλη μορφή, σχεδιάζονται οι γραμμές και στήλες του σχήματος μέσω της **MatPlotLib** και οι εικόνες ετοιμάζονται για να προβληθούν. Αφού ολοκληρωθεί ο προγραμματισμός της function, οι εικόνες και κλάσεις των δεδομένων καλούνται από τον τανυστή των training data και χρησιμοποιούνται ως είσοδος της. Για διευκόλυνση ως προς την κατανόηση των τίτλων της κάθε εικόνας, γίνεται αλλαγή των ετικετών κάθε κλάσης από αριθμούς στις ονομασίες αυτών. Στην προκειμένη περίπτωση, καθώς 0 αντιστοιχεί σε υγιές δείγμα και 1 σε ασθενή, γίνονται και οι κατάλληλες μετατροπές, όπως φαίνεται και στο σχήμα 3.9.



Σχήμα 3.9: Τα δεδομένα που εισήχθησαν στον τανυστή training.

3.1.4 Σχεδιασμός Δομής Μοντέλου

Εφόσον είναι βέβαιο ότι τα δεδομένα εισήχθησαν σωστά σε τανυστές, το επόμενο βήμα είναι αυτό του σχεδιασμού νευρωνικού δικτύου προς εκπαίδευση.

```
# Defining model architecture
model = Sequential()

# Pre-processing Layer
model.add(RandomFlip("horizontal_and_vertical", input_shape = (256, 256, 3)))
model.add(RandomRotation(0.1))

# Convolution Layer 1
model.add(Conv2D(32, (3,3), strides=(1,1), padding = "same"))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size = (2,2)))

# Convolution Layer 2
model.add(Conv2D(64, (3,3), strides=(1,1), padding = "same"))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size = (2,2)))

# Convolution Layer 3
model.add(Conv2D(128, (3,3), strides=(1,1), padding = "same"))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size = (2,2)))
```

Σχήμα 3.10: Είσοδος, προεπεξεργασία και convolution layers δικτύου. Σε περιπτώσεις όπου προκύπτει πιο περίπλοκη δομή, είναι δυνατή η χρήση functions για πιο γρήγορες αλλαγές σε hyperparameters.

Καθώς οι απαιτήσεις για τη δημιουργία ενός image classification μοντέλου από πλευράς layers είναι σχετικά χαμηλές, αρκεί μια προσέγγιση τύπου script στη διάταξη του κώδικα. Σε πιο ανεπτυγμένες εφαρμογές, όπου είναι προτιμητέα η κατασκευή ειδικών layers, μπορεί να γίνει χρήση Python classes και functions. Αρχικά, ορίζεται η αρχιτεκτονική του μοντέλου, η οποία, όπως αναφέρθηκε και στην υποενότητα περί βιβλιοθηκών (3.1.1), είναι η Sequential. Στη συνέχεια, εισάγονται τα διάφορα layers που θα αποτελέσουν την τελική διάταξη.

Data Augmentation ως Ενίσχυση Πλήθους Δεδομένων

Κατά την ανάλυση της λειτουργίας του ImageDataGenerator, έγινε αναφορά στην δυνατότητα του να εφαρμόσει μεθόδους προ-επεξεργασίας στα δεδομένα εισόδου. Με τη χρήση rescaling, οι εικόνες μετατράπηκαν σε μορφή η οποία διευκολύνει το backpropagation. Υπάρχουν όμως

περιπτώσεις, στις οποίες η απουσία μεγάλου πλήθους δεδομένων επίσης δυσχεραίνει την εκπαίδευση. Για αυτό τον λόγο μπορεί να γίνει χρήση και άλλων μεθόδων, οι οποίες αυξάνουν τον αριθμό των διαθέσιμων δειγμάτων, με βάση τα ήδη υπάρχοντα. Έτσι, εισάγονται κάποια προπαρασκευαστικά layers πριν αυτά των convolution, τα οποία επαναπροσανατολίζουν και περιστρέφουν τις εικόνες εισόδου, δημιουργώντας έτσι περισσότερα δεδομένα. Αυτή η τεχνική ονομάζεται **data augmentation**. Το ποσοστό της επεξεργασίας αυτής μπορεί επίσης να θεωρηθεί hyperparameter, καθώς υπερβολές στις μεταβολές των εικόνων μπορεί να έχουν ως αποτέλεσμα δείγματα μη-αντιπροσωπευτικά των ζητούμενων. Σημειώνεται πως τα layers αυτά δεν αποθηκεύονται στο τελικό μοντέλο, λαμβάνοντας μέρος μόνο στη διαδικασία εκμάθησης. Επίσης προσοχή συνίσταται στον ορισμό του πρώτου εκ αυτών, καθώς απαιτείται ο καθορισμός των διαστάσεων των δεδομένων εισόδου (διαστάσεις 256×256, depth 3 στην προκειμένη περίπτωση).

Convolution Layers

Η δημιουργία ενός πλήρους convolution layer ακολουθεί τη μεθοδολογία που παρατέθηκε στη σχετική υποενότητα του πρώτου κεφαλαίου (1.4.2), με το Tensorflow να απλοποιεί περαιτέρω τη διαδικασία. Ξεκινώντας, ορίζονται τα filters, δηλαδή ο αριθμός των συνολικών feature maps στην έξοδο. Έπειτα, επιλέγονται οι διαστάσεις των kernel και το stride που θα πραγματοποιήσουν. Σε περιπτώσεις όπου το δίκτυο περιλαμβάνει πολλαπλά convolution layers στη σειρά, προτείνεται η χρήση padding για την αποτροπή πιθανής απώλειας στην πληροφορία των δειγμάτων. Το activation μπορεί να γίνει είτε στο ίδιο το layer (μέσω παραμέτρου), είτε με την προσθήκη ενός activation layer. Τέλος, εισάγεται ένα pooling layer, το οποίο δέχεται ως όρισμα τις διαστάσεις των pooling kernels αλλά και αν είναι επιθυμητό, padding. Η επιλογή των hyperparamaters εξαρτάται από την φύση των δεδομένων εκπαίδευσης, αλλά μια συνήθης τακτική είναι αυτή της κλιμάκωσης των kernels από ένα convolution layer στο επόμενο. Απαιτείται προσεκτική προσέγγιση ως προς την αύξηση των filters, καθώς μεγαλύτερο πλήθος αυτών μεταφράζεται άμεσα σε υψηλότερες υπολογιστικές απαιτήσεις. Ένα εξίσου σημαντικό πρόβλημα που μπορεί επίσης να δημιουργηθεί είναι αυτο του overfitting, εφόσον περισσότερα filters σημαίνουν μεγαλύτερη περιπλοκότητα δικτύου, που μπορεί να οδηγήσει σε αδυναμία εξαγωγής σημαντικών πληροφοριών από τα διαθέσιμα δεδομένα.

FC Layers και Output

Αφού έχει πραγματοποιηθεί το convolution, πρέπει να γίνει σύνδεση του output των προηγούμενων στρωμάτων με ένα (ή περισσότερα) fully connected layer, έτσι ώστε να είναι δυνατή η εξαγωγή αποτελεσμάτων. Για να γίνει αυτό, πρέπει να γίνει flattening των δεδομένων, δυνατό καλώντας την λειτουργία **Flatten**. Στη συνέχεια εισάγεται το FC layer (το οποίο στο Tensorflow ονομάζεται Dense layer), για το οποίο και καθορίζεται ο αριθμός των εξόδων του. Όπως και προηγουμένως, το layer υφίσταται activation και στη συνέχεια ή συνδέεται με το επόμενο ή εφαρμόζεται τεχνική dropout (καλείται με το ίδιο όνομα κατά Tensorflow), στην οποία μπορεί να οριστεί το ποσοστό των νευρώνων που θα αποσυνδεθούν. Καθώς οι νευρώνες αυτοί αποσυνδέονται με τυχαίο τρόπο, υπάρχει πιθανότητα μεγαλύτερα ποσοστά να έχουν αρνητικές επιπτώσεις στην τελική απόδοση του μοντέλου και έτσι η τιμή αυτή πρέπει να επιλέγεται με σύνεση.

```
# Fully Connected Layer
model.add(Flatten())
model.add(Dense(128))
model.add(Activation("relu"))
model.add(Dropout(0.15))

# Output
model.add(Dense(1))
model.add(Activation("sigmoid"))

# Compiling model
model.compile(loss = "binary_crossentropy", optimizer = "adam", metrics = ['accuracy', 'AUC'])
```

Σχήμα 3.11: Fully connected layer και έξοδος μοντέλου.

Όπως και με το convolution, ο αριθμός των dense layers εξαρτάται πάντα από τα δεδομένα τα οποία χρησιμοποιεί το δίκτυο για εκπαίδευση, ενώ ισχύουν τα δύο προαναφερθέντα προβλήματα περί υπολογιστικής δύναμης και περιπλοκότητας. Εφόσον επιλεχθεί και εισαχθεί ο επιθυμητός αριθμός αυτών, πρέπει να οριστεί το output layer. Είναι επίσης dense layer, ο αριθμός εξόδων του οποίου καθορίζεται από το σύνολο των classes προς πρόβλεψη. Εφόσον το παρόν παράδειγμα πρόκειται για εφαρμογή binary classification, η έξοδος του layer ορίζεται ως μονάδα, ενώ η activation function που χρησιμοποιείται είναι η sigmoid. Τέλος, το νεοσύστατο δίκτυο ολοκληρώνεται με την χρήση της εντολής **compile**, στην οποία και ορίζονται οι συναρτήσεις σφάλματος και ο optimizer που θα χρησιμοποιηθεί, καθώς και τα metrics που θα παρακολουθούνται κατά την εκπαίδευση. Εδώ, επειδή δεν αρκεί μόνο το αντικειμενικό accuracy αλλά επίσης και η τάση του μοντέλου προς θετικές προβλέψεις, το μοντέλο αξιολογείται και με Area Under Curve (AUC). Επειδή το τελικό μοντέλο θα εκτελεί binary classification, η loss function που επιλέγεται είναι η **binary crossentropy**, με το optimization να εκτελείται με χρήση **Adam**. Εφόσον αυτή η διαδικασία έρθει εις πέρας, το τελικό βήμα είναι αυτό της έναρξης εκμάθησης.

3.1.5 Εκπαίδευση Αλγορίθμου

Πριν ξεκινήσει η εκπαίδευση, ο χρήστης πρέπει να ορίσει τυχόν callbacks, τα οποία επιθυμεί να εκτελούνται κατά τη διάρκειά της. Στην αρχή καλείται η βιβλιοθήκη **time**, έτσι ώστε στο τέλος να έχει χρονομετρηθεί το σύνολο της διαδικασίας. Έπειτα, ρυθμίζονται τα **callback routines**, ξεκινώντας με το **EarlyStopping**. Ως επιτηρούμενη τιμή επιλέγεται αυτή του validation loss, όπου κριτήριο είναι η μείωση της μετά από κάθε epoch.

```
# We use a simple time module to record how much time the network took to train.
# This can be safely removed, unless the user is curious for learning/diagnostic purposes.
# If training on a CPU, the process will take significantly longer.
start_time = time.time()

# We use various callbacks to monitor the model's training and ensure minimum time losses in case
# the model starts to overfit/the process is lengthy enough to make manual monitoring impossible.
# It is generally recommended to use all 3 callbacks below to ensure best model performance.
#
# We use Early Stopping to monitor a specific metric and terminate training if there's no improvement
# after a set number of epochs. Checkpoint saves the model when a monitored value is improved.
# Finally, we can automatically tweak the learning rate of the model if our preferred monitored values
# shows no improvement after a certain number of epochs.
early_stopping = EarlyStopping(monitor='val_loss', patience = 6, restore_best_weights = True)
checkpoint = ModelCheckpoint("Brain Tumor Checkpoint.h5", monitor='val_accuracy', verbose = 1, save_best_only = True, mode='max')
lr_monitor = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=3, min_lr=0.00001)
```

Σχήμα 3.12: Διαγνωστικά εργαλεία προόδου ανάπτυξης του μοντέλου. Η αναλυτική λειτουργία των callbacks στο σχήμα αναλύονται στην υποενότητα 3.1.1.

Στη συνέχεια, ορίζεται ο αριθμός epochs που θα πρέπει να παρέλθουν πριν την διακοπή της εκπαίδευσης (**patience**), επαναφέροντας τα weights για τα οποία σημειώθηκε η καλύτερη δυνατή τιμή (**restore_best_weights**) του επί παρακολούθηση metric. Στη συνέχεια, ως δικλείδα ασφαλείας σε περίπτωση που η εκπαίδευση διακοπεί λόγω απρόβλεπτων αιτιών, χρησιμοποιείται το ModelCheckpoint, που θα αποθηκεύει αυτόματα την πιο πρόσφατη έκδοση του μοντέλου, με σημείο αναφοράς την βελτίωση του validation accuracy. Αυτό γίνεται δυνατό μέσω των ορισμάτων save_best_only και **mode = max** (Η τιμή max δηλώνει πως η επιβλεπόμενη τιμή πρέπει να αυξάνεται), καθώς μη χρήση αυτών έχει ως αποτέλεσμα την αποθήκευση ανεξαρτήτως βελτίωσης. Τέλος, προς αποτροπή στασιμότητας στο μοντέλο καλείται η ReduceLROnPlateau. Η τιμή προς επίβλεψη είναι η validation loss, η οποία αν παραμένει στάσιμη για τρία epochs, θα προκαλεί μείωση του learning rate στο 20% της προηγούμενης τιμής του(**factor**), ενώ τίθεται κατώτατη τιμή αυτού το 10^{-5}.

```
# Initiating training.
history = model.fit(train_data, validation_data = valid_data, epochs = 30, verbose = 1,
                    callbacks = [early_stopping, checkpoint, lr_monitor])

# Tensorflow recommends saving models in .keras format. To comply with that, we save the best model generated
# by the callback functions used above.
model.save('Brain Tumor Classification.keras')
print(time.time() - start_time, 'sec')

# Plot model accuracy for training and validation datasets
plt.plot(history.history['accuracy'], label = 'accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.title('Correlation of Training vs Validation Data')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(['Training Data','Validation Data'], loc='lower right')
```

Σχήμα 3.13: Έναρξη εκπαίδευσης, σχεδίαση καμπύλης metrics και αποθήκευση μοντέλου με το πέρας της διαδικασίας.

Για να ξεκινήσει η εκπαίδευση (Σχ. 3.13), καλείται η λειτουργία **fit** στο προηγουμένως ορισμένο μοντέλο. Εδώ, δίνονται οι μεταβλητές στις οποίες αποθηκεύτηκαν τα training και validation data, ο αριθμός epochs και η λίστα των callbacks που θα χρησιμοποιηθούν κατά τη διάρκεια εκπαίδευσης του νευρωνικού δικτύου. Όταν τελειώσει η διαδικασία αυτή, το μοντέλο αποθηκεύεται μέσω της **save** και στη συνέχεια εξάγονται τα τελικά metrics. Στην προκειμένη περίπτωση αντικείμενο ενδιαφέροντος είναι η σύγκριση μεταξύ accuracy και validation accuracy, έτσι ώστε να μπορούν να εντοπιστούν τυχόν προβλήματα overfitting και underfitting. Καθώς η πορεία των τιμών αυτών ανά epoch έχει διατηρηθεί ως μεταβλητή στη μνήμη, μπορούν να κλιθούν και να σχεδιαστούν σε γραφική παράσταση μέσω MatPlotLib. Υπάρχει η δυνατότητα αναπαράστασης και άλλων τιμών metrics, ανάλογα με τον τελικό στόχο ως προς την αξιολόγηση της επίδοσης του τελικού μοντέλου.

```
Epoch 1/30
101/101 [==============================] - ETA: 0s - loss: 0.7203 - accuracy: 0.6535 - auc: 0.6971
Epoch 1: val_accuracy improved from -inf to 0.76017, saving model to Brain Tumor Checkpoint.h5
                                                                                UserWarning: You are saving
your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
101/101 [==============================] - 12s 86ms/step - loss: 0.7203 - accuracy: 0.6535 - auc: 0.6971 - val_loss: 0.5684 - val_
accuracy: 0.7602 - val_auc: 0.8282 - lr: 0.0010
Epoch 2/30
100/101 [============================>.] - ETA: 0s - loss: 0.5070 - accuracy: 0.7653 - auc: 0.8283
Epoch 2: val_accuracy improved from 0.76017 to 0.82558, saving model to Brain Tumor Checkpoint.h5
101/101 [==============================] - 8s 77ms/step - loss: 0.5064 - accuracy: 0.7655 - auc: 0.8289 - val_loss: 0.3784 - val_a
ccuracy: 0.8256 - val_auc: 0.9063 - lr: 0.0010
Epoch 3/30
100/101 [============================>.] - ETA: 0s - loss: 0.4339 - accuracy: 0.8068 - auc: 0.8779
Epoch 3: val_accuracy did not improve from 0.82558
101/101 [==============================] - 7s 72ms/step - loss: 0.4332 - accuracy: 0.8068 - auc: 0.8781 - val_loss: 0.4024 - val_a
ccuracy: 0.8052 - val_auc: 0.9111 - lr: 0.0010
Epoch 4/30
101/101 [==============================] - ETA: 0s - loss: 0.3869 - accuracy: 0.8330 - auc: 0.9077
```

Σχήμα 3.14: Τυπικό log output σε εκπαίδευση με verbose = 1. Καταγράφονται τα metrics, ο χρόνος ανα epoch καθώς και ο αριθμός της.

3.1.6 Αξιολόγηση Εξαχθέντος Μοντέλου

Η γραφική παράσταση των metrics βρίσκεται σε θέση να παρουσιάσει την σχετική απόδοση που αναμένεται από το νευρωνικό δίκτυο, όταν αυτό εφαρμοστεί σε πραγματικές συνθήκες. Αν δεν εντοπίζονται προβλήματα μείζονος σημασίας σε αυτή, το δίκτυο πρέπει να αξιολογηθεί περαιτέρω για λόγους επαλήθευσης της σωστής λειτουργίας του. Η διαδικασία αυτή ονομάζεται **evaluation**. Αρχικά, αναλύεται η δομή του μοντέλου και τυπώνεται στην κονσόλα μέσω της εντολής **summary**, όπου φαίνονται τα layers και οι παράμετροι που χρησιμοποιήθηκαν σε καθένα από αυτά. Αυτό πραγματοποιείται έτσι ώστε να είναι σίγουρο πως δεν υπήρξε κάποιο λάθος κατά την εκπαίδευση. Εν συνεχεία, με χρήση της λειτουργίας **evaluate**, το μοντέλο ταξινομεί τα δείγματα και των τριών συνόλων που διαχωρίστηκαν στο προπαρασκευαστικό στάδιο. Τα προκύπτοντα metrics για κάθε υποσύνολο αποθηκεύονται σε list και μπορούν εύκολα να διαβαστούν μέσω της εντολής **print**.

```
[9]: # Generate model summary
     model.summary()

     Model: "sequential"
     _________________________________________________________________
      Layer (type)                 Output Shape              Param #
     =================================================================
      random_flip (RandomFlip)     (None, 256, 256, 3)       0

      random_rotation (RandomRot   (None, 256, 256, 3)       0
      ation)

      conv2d (Conv2D)              (None, 256, 256, 32)      896

      activation (Activation)      (None, 256, 256, 32)      0

      max_pooling2d (MaxPooling2   (None, 128, 128, 32)      0
      D)

      conv2d_1 (Conv2D)            (None, 128, 128, 64)      18496

      activation_1 (Activation)    (None, 128, 128, 64)      0

      max_pooling2d_1 (MaxPoolin   (None, 64, 64, 64)        0
      g2D)

      conv2d_2 (Conv2D)            (None, 64, 64, 128)       73856

      activation_2 (Activation)    (None, 64, 64, 128)       0

      max_pooling2d_2 (MaxPoolin   (None, 32, 32, 128)       0
      g2D)
```

Σχήμα 3.15: Τμήμα της δομής του τελικού μοντέλου. Σε περιπτώσεις όπου απαιτείται αναδιάρθρωση της αρχιτεκτονικές, προτείνεται προσεκτική μελέτη των παραμέτρων που δημιουργούνται κατά την εκπαίδευση.

```
# Generate evaluation metrics for all 3 datasets
train_eval = model.evaluate(train_data, verbose = 0)
val_eval = model.evaluate(valid_data, verbose = 0)
test_eval = model.evaluate(test_data, verbose = 0)
print('Accuracy, loss and AUC scored compared to training data is:', train_eval[1], ',', train_eval[0], ',', train_eval[2])
print('Accuracy, loss and AUC scored compared to validation data is:', val_eval[1],',', val_eval[0],',', val_eval[2])
print('Accuracy, loss and AUC scored compared to test data is:', test_eval[1],',', test_eval[0],',', test_eval[2])

Accuracy, loss and AUC scored compared to training data is: 0.9891135096549988 , 0.02847857028245926 , 0.999255895614624
Accuracy, loss and AUC scored compared to validation data is: 0.9723837375640869 , 0.0804215669631958 , 0.9938881397247314
Accuracy, loss and AUC scored compared to test data is: 0.986975371837616 , 0.05567790940403938 , 0.9968262314796448
```

Σχήμα 3.16: Μπλόκ κώδικα υπεύθυνο για το evaluation του δικτύου και τελικές τιμές metrics.

Σχήμα 3.17: Γραφική παράσταση τιμών metrics ανά epoch εκπαίδευσης.

Η σημασία του evaluation φαίνεται όταν συγκριθούν τα σχήματα 3.16 και 3.17. Εξετάζοντας την γραφική παράσταση, παρατηρείται μια τάση προς overfitting στις 5 τελικές epochs. Όμως, εφαρμογή evaluation στο σύνολο των δεδομένων αποδεικνύει πως το accuracy και AUC μεταξύ training και test βρίσκεται σε συμφωνία, που μεταφράζεται άμεσα σε ποιοτικό δίκτυο, καθώς δεν παρατηρούνται φαινόμενα overfitting και underfitting.

3.1.7 Πραγματοποίηση Προβλέψεων

Εφόσον έχει διασφαλιστεί η αξιοπιστία και απόδοση του μοντέλου, είναι πλέον δυνατή η χρήση αυτού για ταξινόμηση σε πραγματικά δεδομένα. Το μοντέλο μπορεί να φορτωθεί με τη χρήση της εντολής **tf.keras.models.load_model**. Αν και ο έλεγχος τυχαίων εικόνων από τα test data συνήθως αρκεί, παρουσιάζεται ιδιαίτερο ενδιαφέρον στην μελέτη της ποσοστιαίας ικανότητας του μοντέλου να αναγνωρίζει σωστά αρνητικά και θετικά δείγματα. Η διαδικασία αυτή, αν και προαιρετική, αποτελεί ακόμα μια μέθοδο για την αξιολόγηση του μοντέλου και μπορεί να χρησιμοποιηθεί για τον εντοπισμό πιθανού bias προς συγκεκριμένες προβλέψεις.

```
# Code block to be used later for prediction percentages.
pos_num = 0
neg_num = 0

# We then create a test database, which we'll use to cross-reference the two classes of the model
# with the test data we segmented earlier.
df_test = pd.DataFrame(data=os.listdir(dl_path + 'Segmented Images/test/'), columns = ['image'])
df_test = df_test.merge(df, how='inner')

for i in range (0,len(os.listdir(dl_path + 'Segmented Images/test/'))):
    if str(df_test['status'][i]) == 'tumor':
        pos_num = pos_num + 1
    else:
        neg_num = neg_num + 1
```

Σχήμα 3.18: Αρίθμηση θετικών και αρνητικών δειγμάτων σε test data.

Στην αρχή, καταγράφεται το πλήθος των δειγμάτων που απαρτίζουν την κάθε class. Οι τιμές αυτές θα αποτελούν το απόλυτο μέτρο, με το οποίο θα μπορεί να υπολογιστεί η απόδοση του δικτύου. Ο τρόπος καταγραφής που επιλέγεται είναι η δημιουργία ενός dataframe στο οποίο εισάγονται τα ονόματα των εικόνων στο υποσύνολο των test data, καθώς και σε ποιο class ανήκουν. Η διαδικασία αυτή είναι σχετικά εύκολη, λόγω της δυνατότητας της Pandas να συμπτύσσει δύο dataframes, διατηρώντας μόνο στοιχεία τα οποία έχουν από κοινού. Έπειτα, η κάθε γραμμή ελέγχεται ως προς τις classes και ανάλογα με αυτές αυξάνεται η τιμή του αντίστοιχου συνόλου. Εφόσον στόχος είναι ο έλεγχος της δυνατότητας του μοντέλου να ταξινομήσει σωστά εκάστοτε δείγμα, τότε αρκεί να δοθούν σε αυτό εικόνες που ανήκουν αποκλειστικά σε κάποια συγκεκριμένη class.

```
def predictions(class_name, class_num, class_id):
    class_count = 0
    for i in range (0,len(os.listdir(dl_path + 'Segmented Images/test/'))):
        if str(df_test.status[i]) == class_name:
            img_path = dl_path + 'Segmented Images/test/' + str(df_test.image[i])
            img = keras.preprocessing.image.load_img(img_path, target_size=(256, 256))
            img_array = keras.preprocessing.image.img_to_array(img)
            img_array = np.expand_dims(img_array, axis=0)
            img_array /= 255.

            prediction = model.predict(img_array, verbose = 0)
            predicted_class = np.round(prediction)
            if predicted_class == class_id:
                class_count = class_count + 1
            else:
                continue
    print('Ratio of correct predictions for class ' + class_name + ' to total is: ', class_count/class_num)
```

Σχήμα 3.19: Μπλοκ κώδικα υπεύθυνο για την καταγραφή σωστής πρόβλεψης από το δίκτυο.

Για να γίνει αυτό, ορίζεται ένα for loop, το οποίο θα πραγματοποιήσει την διαδικασία αριθμό φορών ίσο με αυτών των επιλεγμένων δειγμάτων. Τα δείγματα αναγνωρίζονται αυτόματα με τη χρήση του dataframe που δημιουργήθηκε προηγουμένως, θα προσπελαστούν ως αρχεία με χρήση keras και έπειτα θα μετατραπούν σε μορφή numpy array. Στη συνέχεια, οι τιμές του προκύπτοντος πίνακα θα κανονικοποιηθούν, καθιστώντας το δείγμα έτοιμο προς ταξινόμηση. Κατά τη πρόβλεψη σε binary classification, προκύπτει μια τιμή που κυμαίνεται στο διάστημα $[0, 1]$, λόγω της sigmoid function που χρησιμοποιείται στο output layer. Τιμές πάνω του 0.5 θεωρούνται ότι ανήκουν στην class 1, ενώ οι υπόλοιπες στην class 0. Πρέπει λοιπόν, να πραγματοποιηθεί στρογγυλοποίηση του prediction, ώστε να γίνεται ταχύτερα αντιληπτή η ταξινόμηση κατά το μοντέλο. Αφού η πρόβλεψη στρογγυλοποιηθεί, η τιμή συγκρίνεται με την επιθυμητή class. Αν η προβλεπόμενη class είναι σωστή, ο counter αυξάνεται. Η ίδια διαδικασία εκτελείται για όλες τις classes και τα αποτελέσματα τυπώνονται στο τέλος. Το ποσοστό υπολογίζεται ως εξής:

$$Correct\ Prediction\ Ratio = \frac{Correct\ Predictions}{Sum\ of\ Class\ Samples} \tag{3.1}$$

Στο σχήμα 3.20 παρουσιάζονται τα αποτελέσματα για το τελικό μοντέλο που δημιουργήθηκε, με βάση τα όσα αναλύθηκαν στην ενότητα αυτή.

```
predictions('normal', neg_num, 0)
predictions('tumor', pos_num, 1)
```

```
Number of correct predictions for class normal to total is:  0.9840255591054313
Number of correct predictions for class tumor to total is:  0.9894179894179894
```

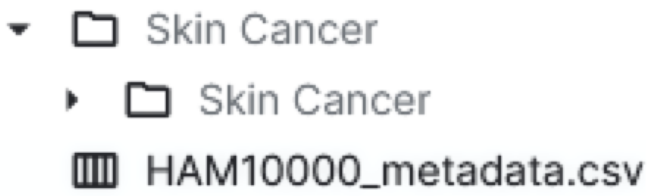
Σχήμα 3.20: Ακρίβεια μοντέλου σε προβλέψεις δείγματος ανα class. Οι τιμές δεν απέχουν πολύ από το evaluation, ενώ η μικρή ποσοστιαία διαφορά υπέρ των θετικών προβλέψεων μπορεί να εξηγηθεί, παίρνοντας υπόψιν ότι η tumor class αποτελείται από περισσότερα στοιχεία.

3.2 Categorical Image Classification σε Ταξινόμηση Δερματικών Καρκινικών Παθήσεων

Το δεύτερο είδος classification που είναι δυνατόν να πραγματοποιήσει ένα δίκτυο είναι αυτό του categorical clasification. Όπως έχει ήδη αναφερθεί, η διαδικασία αυτή διευρύνει τον αριθμό των classes προς ταξινόμηση, καθώς πλέον τα δεδομένα μπορεί να αντιστοιχούν σε μια πληθώρα χαρακτηριστικών. Η εφαρμογή της μεθοδολογίας αυτής είναι πιο ευέλικτη σε σύγκριση με την binary, εφόσον πλέον οι εικόνες προς αναγνώριση μπορούν να εμφανίζουν εξαιρετικά μεγαλύτερη ποικιλία, ανάλογα με τις εκάστοτε απαιτήσεις. Σημαντικό μειονέκτημα όμως αποτελεί η ανάγκη για επίσης μεγάλη αύξηση στον αριθμό των διαθέσιμων δειγμάτων, έτσι ώστε το τελικό μοντέλο να μπορεί με ευκολία να διαχωρίζει μεταξύ αυτών. Ως παράδειγμα, θα αναλυθεί στη συνέχεια μια παραλλαγή του προηγούμενου μοντέλου, το οποίο πραγματεύεται την ταξινόμηση διαφόρων ειδών καρκίνου του δέρματος.[46] Θα εξηγηθεί η βασική μέθοδος ως προς τη δημιουργία ενός categorical classification δικτύου, ενώ θα παρουσιαστεί η επίδραση ενός μη-ισορροπημένου, ως προς τις classes του, dataset στην τελική απόδοση και bias που καταγράφεται.

3.2.1 Προετοιμασία για Εκπαίδευση

Η εισαγωγή βιβλιοθηκών και διαδικασία εξαγωγής της βάσης δεδομένων είναι σχεδόν ίδιες με αυτές του προηγούμενου μοντέλου. Η βάση αυτή παρουσιάζει παρόμοια δομή με αυτή των εγκεφαλικών μαγνητικών τομογραφιών, με τη βασική διαφορά πως τα αρχεία δεν είναι διαχωρισμένα σε φακέλους ανα class.



Σχήμα 3.21: Δομή dataset. Οι εικόνες βρίσκονται στον φάκελο Skin Cancer χωρίς να έχουν διαχωριστεί. Απαιτείται δημιουργία φακέλων όπως και το binary classification μοντέλο.

Το .csv αρχείο διαθέτει την ονομασία της κάθε εικόνας, όπως και την class στην οποία ανήκει. Αυτό καθιστά δυνατή τη χρήση dataframe, έτσι ώστε να μπορούν τα δεδομένα να εισαχθούν σε τανυστές, με κάποιες μικρές τροποποιήσεις. Συγκεκριμένα, όπως φαίνεται και στο σχήμα 3.22, απουσιάζουν οι επεκτάσεις τύπου αρχείου στην στήλη όπου ονομάζονται τα στοιχεία της βάσης δεδομένων (**image id**). Συνεπώς, είναι απαραίτητη η προσθήκη αυτών, έτσι ώστε να μπορούν να αναγνωριστούν από το ImageDataGenerator. Καθώς όλες οι εικόνες είναι τύπου .jpg, αρκεί μια απλή εντολή μέσω pandas για να μετατραπούν τα στοιχεία της στήλης στην κατάλληλη μορφή. Τα αρχεία στη συνέχεια εισάγονται σε τελεστές. Εδώ, το **class_mode** ορίζεται ως **categorical**, λόγω των πολλών classes που διαθέτει το dataset.

lesion˙id	image˙id	dx	dx˙type	age	sex	localization
HAM˙0000118	ISIC˙0027419	bkl	histo	80.0	male	scalp
HAM˙0000118	ISIC˙0025030	bkl	histo	80.0	male	scalp
HAM˙0002730	ISIC˙0026769	bkl	histo	80.0	male	scalp
HAM˙0002730	ISIC˙0025661	bkl	histo	80.0	male	scalp
HAM˙0001466	ISIC˙0031633	bkl	histo	75.0	male	ear
HAM˙0001466	ISIC˙0027850	bkl	histo	75.0	male	ear
HAM˙0002761	ISIC˙0029176	bkl	histo	60.0	male	face
HAM˙0002761	ISIC˙0029068	bkl	histo	60.0	male	face
HAM˙0005132	ISIC˙0025837	bkl	histo	70.0	female	back
HAM˙0005132	ISIC˙0025209	bkl	histo	70.0	female	back
HAM˙0001396	ISIC˙0025276	bkl	histo	55.0	female	trunk

Σχήμα 3.22: Πίνακας .csv από τον οποίο προήλθαν πληροφορίες για τα δεδομένα. Στην στήλη **image id** εμπεριέχονται τα ονόματα των αρχείων και στην **dx** η αντίστοιχη class τους. Η απουσία επέκτασης στα ονόματα των αρχείων δημιουργεί την απαίτηση παρέμβασης στον πίνακα.

```
#Load Skin Cancer Database CSV file
predf = pd.read_csv(dl_path + 'HAM10000_metadata.csv')

#Add filename suffix to entries in the "image_id" column
df = predf.copy()
df['image_id'] = df['image_id'] + '.jpg'

#Assign classes to the imagesets by cross-referencing file names with the "dx" column from the CSV file
#and prepare images for overall processing from the neural network.
datagen = ImageDataGenerator(rescale=1./255)
train_data = datagen.flow_from_dataframe(dataframe = df, directory = image_path + 'train', x_col = "image_id",
                                         y_col = "dx", class_mode = "categorical", target_size = (256, 256),
                                         batch_size = 32)
valid_data = datagen.flow_from_dataframe(dataframe = df, directory = image_path + 'valid', x_col = "image_id",
                                         y_col = "dx", class_mode = "categorical", target_size = (256, 256),
                                         batch_size = 32)
test_data = datagen.flow_from_dataframe(dataframe = df, directory = image_path + 'test', x_col = "image_id",
                                        y_col = "dx", class_mode = "categorical", target_size = (256, 256),
                                        batch_size = 32)
```

Σχήμα 3.23: Τροποποίηση της στήλης στην οποία καταγράφονται τα ονόματα των αρχείων και εισαγωγή αυτών σε Tensorflow. Οι δυνατότητες επεξεργασίας που προσφέρονται απο το pandas καθιστά τη διαδικασία εξαιρετικά απλή.

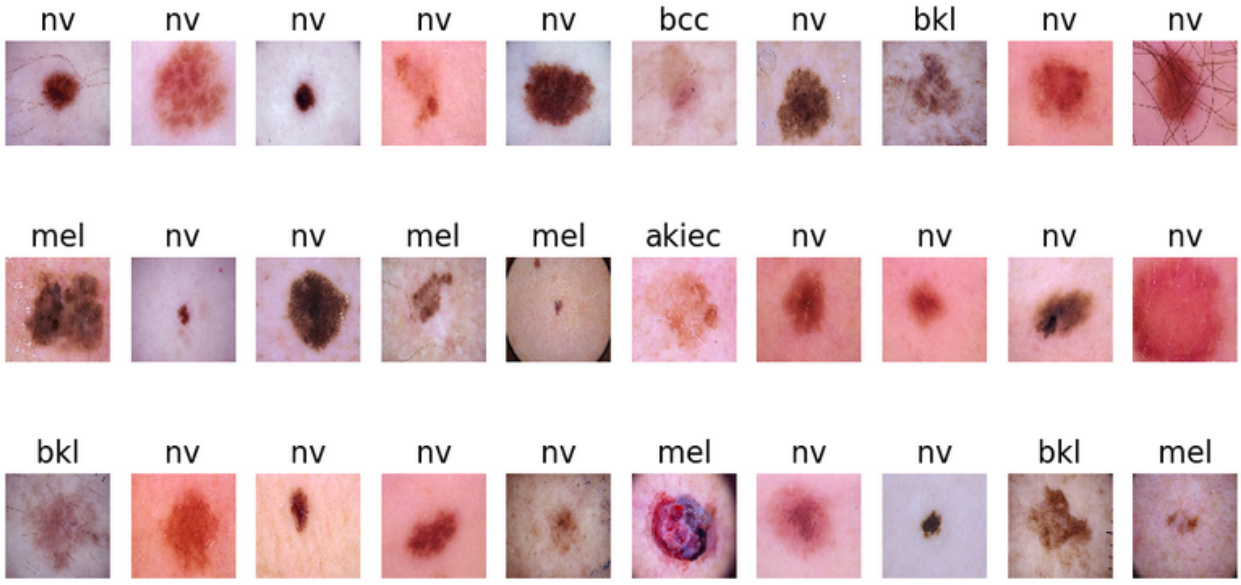
Προς εξοικείωση και ανάλυση ενός dataset, συνίσταται η εξέταση των δειγμάτων ανά class αυτού. Ένας απλός τρόπος είναι να πραγματοποιηθεί καταμέτρηση των αρχείων στον .csv πίνακα ανάλογα με σημείο αναφοράς την class στην οποία ανήκουν. Η διαδικασία αυτή γίνεται εύκολη μέσω **pandas**, εκτελώντας την με μία μόνο εντολή (**value_counts**). Τα αποτελέσματα στη συνέχεια οργανώνονται αλφαβητικά ως προς τις classes έτσι ώστε να βρίσκονται σε συμφωνία με τον τρόπο κατανομής αυτών από το Tensorflow.

```
counts = df.dx.value_counts().sort_index(ascending=True)
print(counts)
```

```
dx
akiec     327
bcc       514
bkl      1099
df        115
mel      1113
nv       6705
vasc      142
```

Σχήμα 3.24: Εντολές υπεύθυνες για την καταμέτρηση δειγμάτων ανα class και αποτελέσματα αυτού. Παρατηρείται μη-ισορροπημένη κατανομή των δεδομένων.

Με χρήση print, τα στοιχεία τα οποία προσμετρήθηκαν εμφανίζονται στο log για ανάλυση. Απευθείας παρατηρείται το βασικό πρόβλημα της βάσης δεδομένων, με την πρώτη class να αποτελεί το 67% του συνόλου των δεδομένων, ενώ η απευθείας επόμενη σε πλήθος φτάνει μόλις το 11%. Αυτό σημαίνει πως κατά την εκπαίδευση, το δίκτυο θα επεξεργάζεται επί το πλείστον στοιχεία της πρώτης class, με αποτέλεσμα να παρουσιάζει αδυναμία στην επιτυχή ταξινόμηση δειγμάτων που ανήκουν στις υπόλοιπες. Ένας από τους μόνους διαθέσιμους τρόπους, έτσι ώστε να μεγιστοποιηθεί η πιθανότητα καλύτερης τελικής απόδοσης είναι η εφαρμογή data augmentation. Ακόμα όμως και η μέθοδος αυτή έχει όρια, όπως αναφέρθηκε και στην προηγούμενη ενότητα.



Σχήμα 3.25: Απεικόνιση δειγμάτων βάσης δεδομένων.

3.2.2 Δομή Δικτύου Τύπου Categorical Image Classification

Εφόσον τα δεδομένα δεν έχουν τον κατάλληλο αριθμό εικόνων ανά class, έτσι ώστε να διασφαλιστεί η αμεροληψία του μοντέλου κατά την πρόβλεψη, θα γίνει απόπειρα αύξησης των συνολικών δειγμάτων. Εκτός της αλλαγής προσανατολισμού και της εισαγωγής περιστροφής, θα πραγματοποιηθεί και μέθοδος μεγέθυνσης μέσω της **RandomZoom**.

```
# Defining model architecture
model = Sequential()

#Pre-processing Layer
model.add(RandomFlip("horizontal_and_vertical", input_shape = (256, 256, 3)))
model.add(RandomRotation(0.2))
model.add(RandomZoom(0.15))

#Layer 1
model.add(Conv2D(16, (3,3), strides=(1,1), padding = "same"))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size = (2,2)))

model.add(Conv2D(32, (3,3), strides=(1,1), padding = "same"))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size = (2,2)))

model.add(Conv2D(64, (3,3), strides=(1,1), padding = "same"))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size = (2,2)))

model.add(Conv2D(128, (3,3), strides=(1,1), padding = "same"))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size = (2,2)))
```

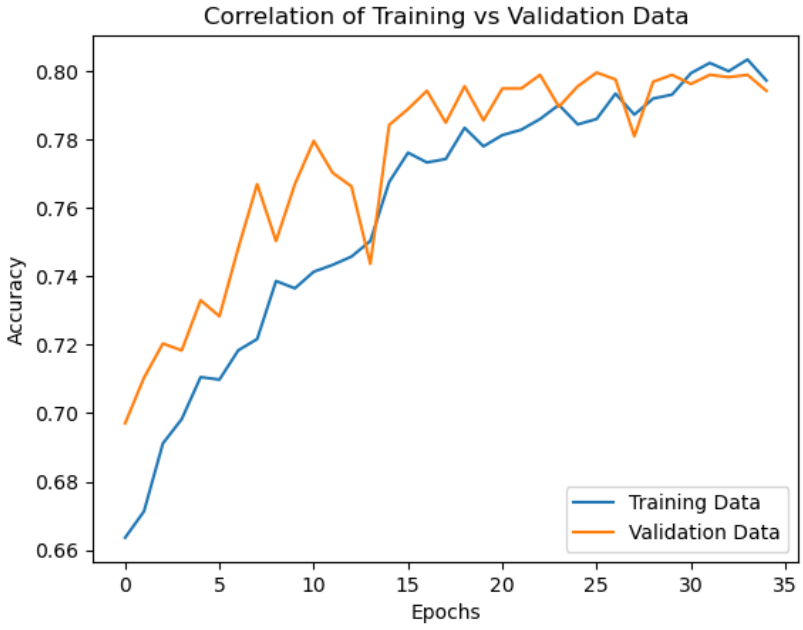
Σχήμα 3.26: Προσπέλαση δεδομένων από augmentation, εισαγωγή στο μοντέλο και διάταξη convolution layers.

Επίσης, λόγω της ποικιλίας των διακριτών χαρακτηριστικών μεταξύ των διαφόρων καρκινωμάτων, θα εισαχθεί ένα ακόμα convolution layer, σε σχέση με τη διάταξη που εμφανίζεται στο binary classification. Το επιπλέον layer αυτό θα αποπειραθεί να εξάγει περισσότερη πληροφορία από τις εικόνες σε μορφή feature maps, με τελικό στόχο την βελτίωση της ποιότητας του τελικού δικτύου. Οι περισσότερες hyperparameters παραμένουν ίδιες με προηγουμένως, καθώς κρίθηκαν αξιόπιστες μετέπειτα σύγκρισης με άλλες.

```
model.add(Flatten())
model.add(Dense(128))
model.add(Activation("relu"))
model.add(Dropout(0.15))

#Layer 5
model.add(Dense(7))
model.add(Activation("softmax"))

#Compiling model
model.compile(loss = "categorical_crossentropy", optimizer = "adam", metrics = ['accuracy', 'AUC'])
```

Σχήμα 3.27: Flattening, ορισμός fully connected layer και εξόδου δικτύου.

Το υπόλοιπο της αρχιτεκτονικής παραμένει ίδιο ως προς τον αριθμό layers. Παρόλα αυτά, αυξάνεται ελαφρώς η hyperparameter του dropout, με στόχο την αποφυγή σοβαρού overfit με το πέρας της εκπαίδευσης, ενώ η τελική activation function που χρησιμοποιείται ειναι η **Softmax**, καθώς πλέον το μοντέλο πραγματεύεται περισσότερες classes, κάνοντας αδύνατη τη χρήση της sigmoid function. Για τον ίδιο λόγο, η loss function που επιλέγεται είναι η **categorical cross-entropy**. Δεν σημειώνονται αλλαγές στον ορισμό metrics, callbacks και στην έναρξη της εκμάθησης, παρά μόνο στον αριθμό των συνολικών epochs σε σύγκριση με το μοντέλο εγκεφαλικών όγκων.




```
Accuracy, loss and AUC scored compared to training data is: 0.8085591793060303 , 0.491282194852829 , 0.960544764995575
Accuracy, loss and AUC scored compared to validation data is: 0.7942743301391602 , 0.5605661869049072 , 0.9302592873573303
Accuracy, loss and AUC scored compared to test data is: 0.7671324014663696 , 0.6045231819152832 , 0.9346031546592712
```

Σχήμα 3.28: Γραφική παράσταση accuracy μεταξύ training και validation data κατά την εκπαίδευση και evaluation metrics.

Εξετάζοντας την προκύπτουσα γραφική παράσταση στο σχήμα 3.28, γίνεται αντιληπτό πως το

συγκεκριμένο μοντέλο εμφανίζει φαινόμενα overfitting. Προς επιβεβαίωση, πραγματοποιείται evaluation, το οποίο επικυρώνει την υπόθεση αυτή. Ενδιαφέρον παρουσιάζει η διαφορά μεταξύ accuracy και AUC. Η τιμή του accuracy δεν ξεπερνά το 76% σε test data, κάτι που δεν ισχύει για τα AUC metrics, τα οποία τείνουν να επηρεάζονται σε μεγάλο βαθμό από το data imbalancing που διαγνώστηκε κατά την ανάλυση των δεδομένων, ενώ το μοντέλο υπόκειται σε overfitting της τάξης του 3%. Το ποσοστό αυτό είναι σχετικά μικρό και συνεπώς μπορεί να χαρακτηριστεί ως αποδεκτό, καθώς η βάση δεδομένων που χρησιμοποιήθηκε δεν ήταν αρκετά ποικιλόμορφη. Ωστόσο, πρέπει να πραγματοποιηθεί prediction στα δεδομένα του κάθε class, έτσι ώστε να εντοπιστούν τυχόν τάσεις bias.

```
Ratio of correct predictions for class akiec to total is:  0.35
Ratio of correct predictions for class bcc to total is:  0.6219512195121951
Ratio of correct predictions for class bkl to total is:  0.514792899408284
Ratio of correct predictions for class df to total is:  0.13636363636363635
Ratio of correct predictions for class mel to total is:  0.3333333333333333
Ratio of correct predictions for class nv to total is:  0.9302093718843469
Ratio of correct predictions for class vasc to total is:  0.8
```

Σχήμα 3.29: Τελικά predictions του μοντέλου σε δείγματα test data ανά class.

Τα τελικά αποτελέσματα παρουσιάζουν ιδιαίτερο ενδιαφέρον. Όπως ήταν αναμενόμενο, οι classes με τα μεγαλύτερο πλήθος δεδομένων επιδεικνύουν μεγαλύτερη ακρίβεια. Εξαιρέσεις όμως αποτελούν οι περιπτώσεις των "mel" και "vasc". Η vasc, μια από τις μικρότερες classes, εντοπίζεται πιο εύκολα από το δίκτυο, σε σχέση με τη mel, η οποία αποτελεί τη δεύτερη μεγαλύτερη. Εξήγηση του φαινομένου αυτού είναι η ιδιαιτερότητα των δειγμάτων τύπου vasc σε σχέση με τα υπόλοιπα δεδομένα. Ο μικρός επίσης αριθμός αυτών στα test data ενδέχεται να επηρεάζει την τιμή αυτής.

3.3 Εφαρμογές σε Άλλα Βιοϊατρικά Δεδομένα

Μελετώντας τα παραπάνω, γίνεται αντιληπτό το εύρος των διαφόρων παθήσεων, για τις οποίες μπορεί να γίνει εφαρμογή classification, με σκοπό την διευκόλυνση του εκάστοτε χρήστη ως προς τη διάγνωση αυτών. Ως προς διερεύνηση των πιθανών αυτών εφαρμογών, σχεδιάστηκαν τρία ακόμα μοντέλα, τα οποία επικεντρώνονται στην επεξεργασία ακτινογραφιών και μαγνητικών τομογραφιών. Τα μοντέλα αυτά θα αναλυθούν στη συνέχεια ως προς τις βάσεις δεδομένων, μέσω των οποίων πραγματοποιήθηκε η εκπαίδευση, και ως προς την απόδοση που προσφέρουν.

3.3.1 Διάγνωση COVID-19 με Ανάλυση Ακτινογραφιών Θώρακος

Ο COVID-19 πρόκειται για μια σχετικά καινούργια ασθένεια, η οποία αλλάζει συνεχώς μορφή, λόγω της ιδιότητας της ως ιό. Υπάρχει λοιπόν, μια συνεχής ανάγκη ώς προς τη μελέτη του ιού αυτού, αλλά και ως προς την έγκυρη και γρήγορη διάγνωσή του. Για τον σκοπό αυτό, αναπτύχθηκε νευρωνικό δίκτυο με τη χρήση μια εκτενούς βάσης δεδομένων[47], η οποία αποτελείται από αριθμό δειγμάτων που ξεπερνάει τα $80,000$. Αν και μεγάλη σε μέγεθος, η βάση αυτή πάσχει από μη-ισορροπημένο αριθμό στοιχείων στις δύο classes που την αποτελούν, με τα

θετικά κρούσματα να είναι πέντε φορές μεγαλύτερα σε πλήθος από τα αρνητικά στο training set.

Type	COVID-19 Negative	COVID-19 Positive	Total
train	10664	57199	67863
val	4232	4241	8473
test	4241	4241	8482

Σχήμα 3.30: Κατανομή στοιχείων ανά class σε κάθε υποσύνολο δεδομένων.

Με σκοπό την απόπειρα ελαχιστοποίησης του πιθανού bias που είναι δυνατόν να προκύψει στις προβλέψεις του τελικού μοντέλου, τα στοιχεία πρώτα ανακατανέμονται τυχαία και στη συνέχεια οργανώνονται σε φακέλους με την ίδια μέθοδο που χρησιμοποιήθηκε και στα προηγούμενα παραδείγματα.

```
train_folder = dl_path + 'train/'
validation_folder = dl_path + 'val/'
test_folder = dl_path + 'test/'
image_folder = dl_path + 'Images/'
final_folder = (dl_path + 'Segmented Images/')
source_paths = [train_folder, validation_folder, test_folder]

# The following code ensures everything is moved to the right places.
if not os.path.exists(final_folder):
    os.makedirs(image_folder)
    for i in range (0,3):
        files = os.listdir(source_paths[i])
        for file in files:
            file_name = os.path.join(source_paths[i], file)
            shutil.move(file_name, final_folder)
    print("Files Moved")
    for i in range (0,3):
        shutil.rmtree(source_paths[i])
else:
    print("Files already collected.")
```

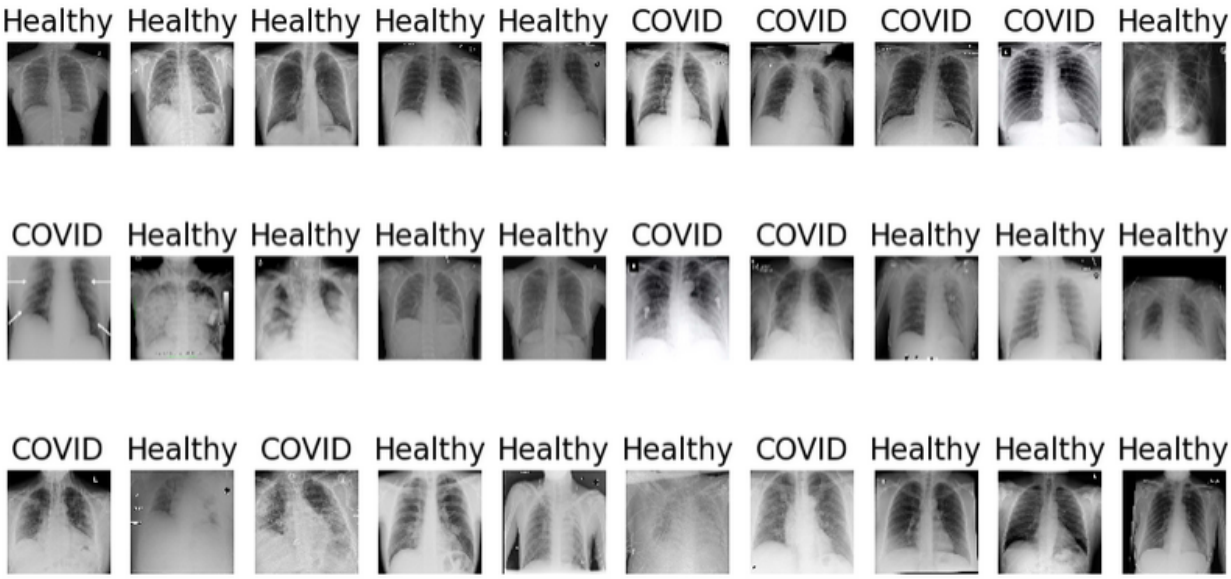
Σχήμα 3.31: Συλλογή αρχείων από φακέλους και μεταφορά σε κοινή διαδρομή.

Αφού πρώτα οριστούν οι διαδρομές των αρχικών υποσυνόλων, οι εικόνες συλλέγονται και εναποθέτονται σε ένα κοινό φάκελο. Έπειτα, γίνεται εκ νέου διαχωρισμός όπως και προηγουμένως. Στη συνέχεια, τα .txt αρχεία στα οποία καταγράφονται τα ονόματα των δειγμάτων και οι classes στα οποία αντιστοιχουν ενώνονται σε ένα καθολικό .csv αρχείο, το οποίο και θα χρησιμοποιηθεί για τη δημιουργία dataframe. Η διαδικασία αυτή είναι απαραίτητη, καθώς πλέον τα αρχεία έχουν εκ νέου κατανεμηθεί τυχαία σε training, evaluation και test.

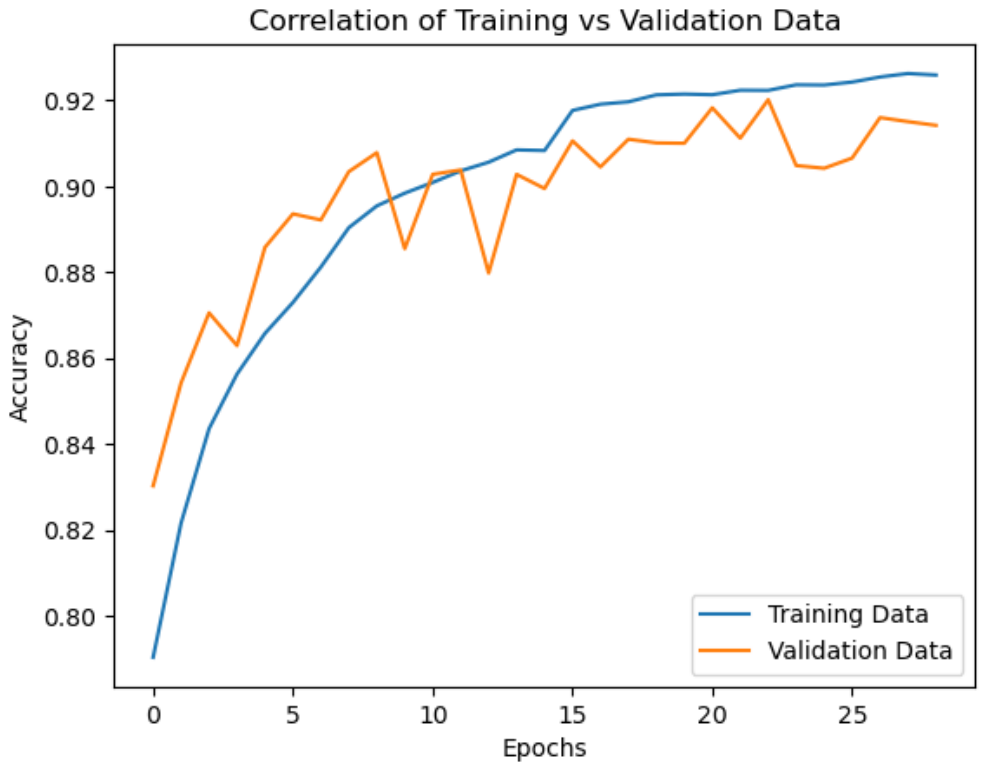
```
if not os.path.isfile(dl_path + 'data.csv'):
    train_file = pd.read_csv(dl_path + 'train.txt', sep = " ", header=None)
    train_file.columns = ['Patient_ID', 'Image_ID', 'Status', 'Source']
    train_file.to_csv(dl_path + 'train.csv', index=None)
    os.remove(dl_path + 'train.txt')

    val_file = pd.read_csv(dl_path + 'val.txt', sep = " ", header=None)
    val_file.columns = ['Patient_ID', 'Image_ID', 'Status', 'Source']
    val_file.to_csv(dl_path + 'val.csv', index=None)
    os.remove(dl_path + 'val.txt')

    test_file = pd.read_csv(dl_path + 'test.txt', sep = " ", header=None)
    test_file.columns = ['Patient_ID', 'Image_ID', 'Status', 'Source']
    test_file.to_csv(dl_path + 'test.csv', index=None)
    os.remove(dl_path + 'test.txt')

    df = pd.concat([train_file, val_file, test_file])
    df.to_csv(dl_path + 'data.csv')
    df = pd.read_csv(dl_path + 'data.csv')
    os.remove(dl_path + 'train.csv')
    os.remove(dl_path + 'val.csv')
    os.remove(dl_path + 'test.csv')
    print('CSV successfully created!')
```

Σχήμα 3.32: Δημιουργία .csv αρχείου.

Τα .txt αρχεία υπόκεινται επεξεργασία μέσω Pandas, μετατρέπονται σε .csv και μετά ενώνονται, έτσι ώστε να παραχθεί το τελικό μητρώο που θα αποτελέσει βάση για το ImageDataGenerator. Αφού δημιουργηθούν οι τελεστές, γίνεται απεικόνιση των εικόνων από τους οποίους αποτελούνται και στη συνέχεια ορίζεται η αρχιτεκτονική του νευρωνικού δικτύου, η οποία παρατίθεται στο **Παράρτημα Γ** και είναι παρόμοια με αυτή που χρησιμοποιήθηκε στην διάγνωση μέσω τομογραφιών εγκεφαλικών όγκων. Με το πέρας της εκπαίδευσης, η προηγούμενη αναφορά στην έλλειψη ισορροπίας μεταξύ των classes προϊδεάζει για πιθανά προβλήματα overfitting ή/και bias. Το πρώτο βήμα ως προς την αξιολόγηση είναι η ανάλυση της γραφικής παράστασης και η σύγκριση αυτής με το evaluation.



Σχήμα 3.33: Δείγματα από τα οποία αποτελείται η βάση δεδομένων.




```
Accuracy, loss and AUC scored compared to training data is: 0.9113540053367615 , 0.26297226548194885 , 0.9219939112663269
Accuracy, loss and AUC scored compared to validation data is: 0.9072473049163818 , 0.2739132344722748 , 0.912965714931488
Accuracy, loss and AUC scored compared to test data is: 0.907897830009460 4 , 0.28045347332954407 , 0.912735104560852
```

Σχήμα 3.34: Γραφική παράσταση training/validation accuracy μοντέλου διάγνωσης COVID και evaluation metrics.

Αν και η γραφική παράσταση στο σχήμα 3.34 φαίνεται να παρουσιάζει overfitting της τάξης του 1%, τα evaluation metrics δείχνουν πως η διαφορά είναι ακόμα μικρότερη. Εφόσον όμως τίθεται ακόμα το ζήτημα του bias, πρέπει να γίνει και ανάλυση των προβλέψεων του δικτύου ως προς την κάθε class. Αυτό επιτυγχάνεται με την ίδια μέθοδο που χρησιμοποιήθηκε στα πλαίσια της αξιολόγησης του binary classification δικτύου εγκεφαλικών όγκων.

```
Ratio of correct predictions for class positive to total is: 0.9842551473556722
Ratio of correct predictions for class negative to total is: 0.6393326233581824
```

Σχήμα 3.35: Ακρίβεια προβλέψεων μοντέλου ανά class.

Από το σχήμα γίνεται ξεκάθαρο πως αν και το μοντέλο έχει υψηλούς δείκτες accuracy και AUC, παρουσιάζει τάσεις να διαγνώσει εκάστοτε ασθενή ως θετικό κρούσμα COVID-19, ακόμα και όταν ο ασθενής δεν πάσχει από τη νόσο. Η συμπεριφορά αυτή ήταν αναμενόμενη, καθώς είχε ήδη διαγνωστεί πρόβλημα ισορροπίας στην ομοιομορφία των δεδομένων, κάτι το οποίο δεν βελτιώθηκε επαρκώς με αλλαγή στην κατανομή τους ή μέσω data augmentation.

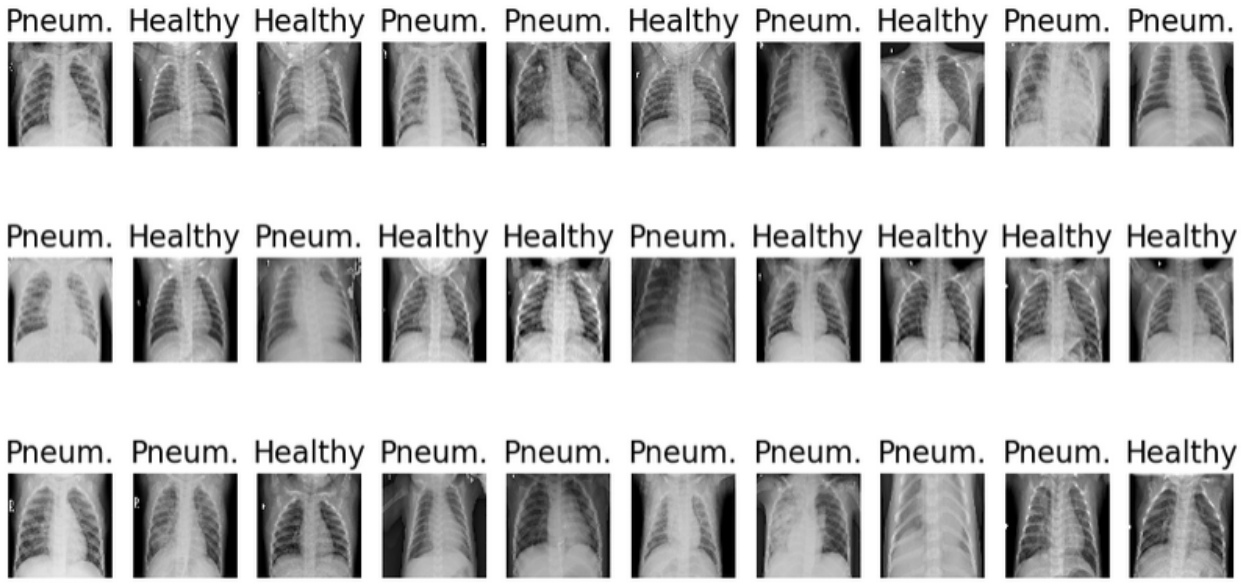
3.3.2 Αναγνώριση Πνευμονίας με Χρήση Ακτινογραφιών

Γενικεύοντας στο ζήτημα περί αναπνευστικών λοιμώξεων, διερευνάται η δημιουργία μοντέλου, το οποίο επικεντρώνεται στον εντοπισμό ενδείξεων πνευμονίας σε ακτινογραφίες θώρακος. Το dataset που χρησιμοποιήθηκε περιέχει περίπου 5,900 δείγματα, τα οποία ανήκουν σε δύο διακριτές classes, υγιών ασθενών και μη.[48] Η βάση είναι δομημένη έτσι ώστε οι εικόνες να είναι διαχωρισμένες σε training και test data. Καθώς δεν παρέχεται αρχείο από το οποίο μπορεί να παραχθεί dataframe, απαιτείται διαφορετική προσέγγιση ως προς τη χρήση του ImageDataGenerator.

```
#Defining Data Generators for train and validation data.

datagen_train = ImageDataGenerator(rescale=1./255, validation_split = 0.3)
datagen_test = ImageDataGenerator(rescale=1./255)

#Defining training and test image directories.
train_dir = image_folder + 'train'
test_dir = image_folder + 'test'

#Creating database files
train_data = datagen_train.flow_from_directory(train_dir, batch_size = 32, shuffle = True, class_mode = 'binary',
                                               target_size = (256,256), seed = 4, subset = "training")
valid_data = datagen_train.flow_from_directory(train_dir, batch_size = 32, shuffle = True, class_mode = 'binary',
                                               target_size = (256,256), seed = 42, subset = "validation")
test_data = datagen_test.flow_from_directory(test_dir, class_mode = 'binary',
                                             target_size = (256,256), batch_size = 32)
print(train_data.class_indices)
```

Σχήμα 3.36: Εναλλακτική μέθοδος δημιουργίας τελεστών δεδομένων απουσίας αρχείου μητρώου.

Η δημιουργία των επιθυμητών τελεστών και του validation set γίνεται μέσω της εντολής **flow_from_directory**, όπου σε κάθε φάκελο υποσυνόλου, εντοπίζονται υποφάκελοι, των οποίων η ονομασία αντιστοιχεί στις classes προς ταξινόμηση και περιέχουν τις εικόνες που ανήκουν σε αυτές. Ταυτόχρονα, ορίζονται δύο διαφορετικά αντικείμενα ImageDataGenerator, με το πρώτο να καθορίζει το ποσοστό στοιχείων του φακέλου που θα χρησιμοποιηθεί για validation. Στη συνέχεια, δημιουργείται αναπαράσταση των δεδομένων και προσμετρούνται τα δείγματα ανά class, όπου και παρατηρείται ακόμη μια περίπτωση μη-ισορροπημένης βάσης δεδομένων. Συγκεκριμένα, τα στοιχεία που αντιστοιχούν σε ασθενείς πνευμονίας είναι τρεις φορές μεγαλύτερα σε σχέση με τους μη νοσούντες. Ως προς αντιμετώπιση και διαχείριση της ιδιαιτερότητας αυτής, πραγματοποιείται data augmentation καθώς τα δεδομένα εισάγονται στο δίκτυο.

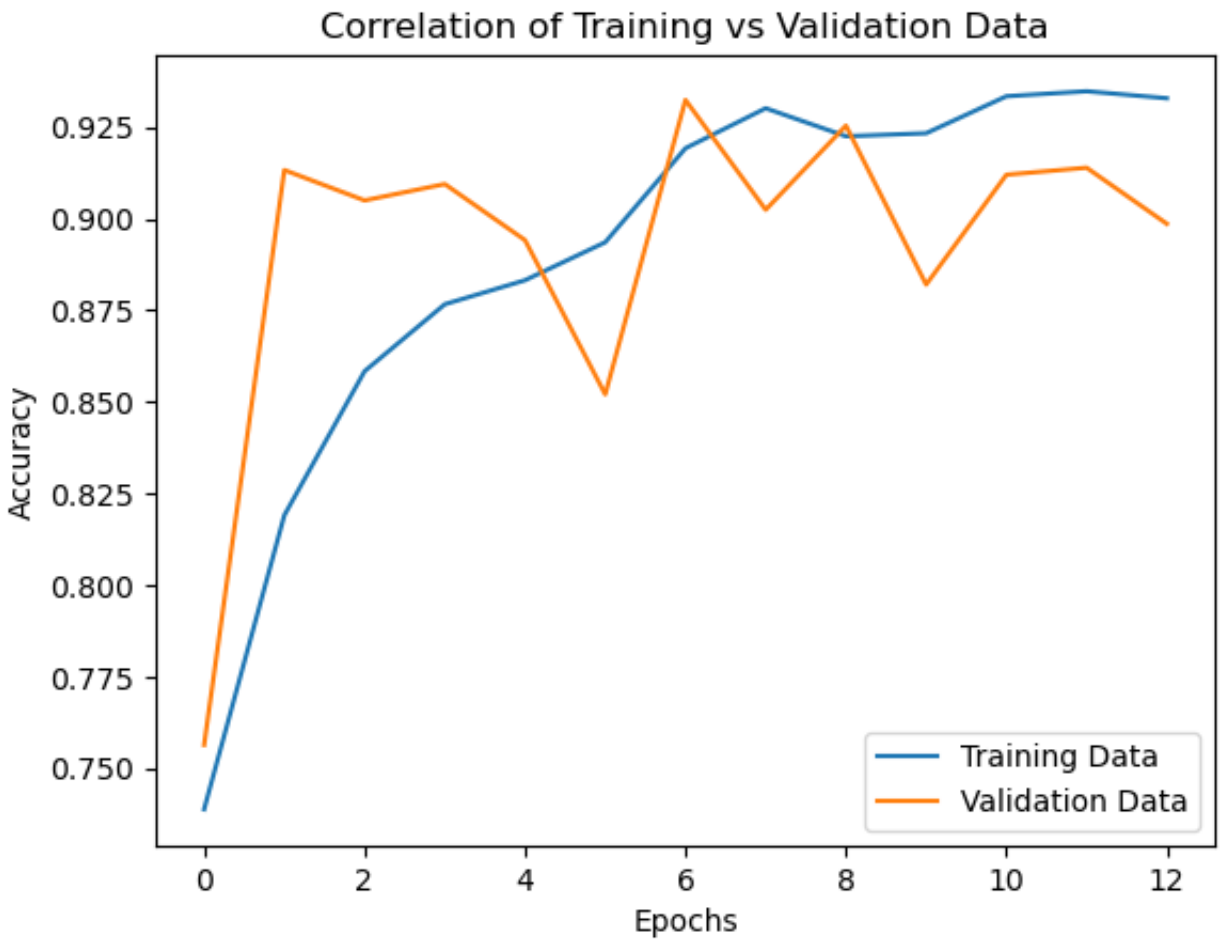


Σχήμα 3.37: Οπτική αναπαράσταση δεδομένων.

```
model.add(RandomFlip("horizontal_and_vertical", input_shape = (256, 256, 3)))
model.add(RandomRotation(0.5))
model.add(RandomZoom(0.2))
```

Σχήμα 3.38: Data augmentation στην είσοδο του δικτύου. Λόγω της μεγάλης διαφοράς μεταξύ των classes και του σχετικά μικρού αριθμού δειγμάτων, η ισχύς του augmentation είναι υψηλή, με τελικό στόχο τη μείωση του overfitting.

Αφού το μοντέλο εκπαιδευτεί και αποθηκευτεί, θα πρέπει να εξεταστούν τα metrics και να εκτελεστεί το evaluation.




```
Accuracy, loss and AUC scored compared to training data is: 0.9208515286445618 , 0.1948971450328827 , 0.9749553799629211
Accuracy, loss and AUC scored compared to validation data is: 0.9323979616165161 , 0.17646153271198273 , 0.9776791334152222
Accuracy, loss and AUC scored compared to test data is: 0.9150640964508057 , 0.27144166827201843 , 0.9667598009109497
```

Σχήμα 3.39: Γραφική παράσταση training/validation accuracy μοντέλου διάγνωσης COVID και evaluation metrics.

Όπως και στο προηγούμενο παράδειγμα, τα metrics που καταγράφτηκαν κατά την εκπαίδευση εμφανίζουν μερικό overfit, κάτι το οποίο όμως δεν υφίσταται σύμφωνα με τα αποτελέσματα του evaluation. Συγκεκριμένα, η διαφορά στις τιμές training και test είναι λιγότερη απο 1%, το οποίο αποτελεί σημάδι ενός αποδοτικού μοντέλου. Για να επιβεβαιωθεί η υπόθεση αυτή, θα πρέπει να γίνει προσεκτική μελέτη των προβλέψεων του μοντέλου, έτσι ώστε να μπορεί να διεξαχθεί έγκυρος εντοπισμός πιθανού bias, το οποίο είναι δυνατόν να προκύψει λόγω των αρχικών δεδομένων.

```
{'NORMAL': 0, 'PNEUMONIA': 1}
Number of correct positive predictions to total is:  0.9435897435897436
Ratio of correct negative predictions is:  0.8675213675213675
```

Σχήμα 3.40: Ποσοστό επιτυχών προβλέψεων δειγμάτων ανα class στο test dataset.

Αν και φαινομενικά η διαφορά στο πλήθος στοιχείων μεταξύ των δύο classes είναι αρκετά μεγάλη, η διαφορά στην ακρίβεια των προβλέψεων είναι μικρότερη σε σχέση με το αντίστοιχο dataset για COVID-19. Πιθανοί λόγοι για το αποτέλεσμα αυτό είναι το μικρότερο μέγεθος της βάσης δεδομένων πνευμονίας, ο χαμηλότερος συντελεστής διαφοράς μεταξύ των δειγμάτων ανα class καθώς και η μεγαλύτερη επεξεργασία μέσω μεθόδων data augmentation.

3.3.3 Αναγνώριση Νευροεκφυλισμού Λόγω Νόσου Alzheimer

Η νόσος Alzheimer πρόκειται για νευροεκφυλιστική νόσο, η οποία καταστρέφει τα εγκεφαλικά κύτταρα και είναι ένας από τους κύριους παράγοντες για την εμφάνιση συμπτωμάτων άνοιας, όπως δυσκολία στην σκέψη και την δυνατότητα λήψης αποφάσεων. Ένα βασικό χαρακτηριστικό της ασθένειας αυτής είναι η εμφανής ζημία που προκαλεί στον εγκέφαλο, η οποία οδηγεί σε σχετική διάγνωση έπειτα από εξέταση μαγνητικών τομογραφιών του ασθενούς.[49] Εφόσον λοιπόν η νόσος μπορεί να εντοπιστεί με οπτικά μέσα, είναι δυνατή η δημιουργία μοντέλου classification, το οποίο έχει ως στόχο τον εντοπισμό αυτής, καθώς και το στάδιο εκφυλισμού του εγκεφάλου. Για την υλοποίηση ενός τέτοιου δικτύου, καταφορτώθηκε βάση δεδομένων τεσσάρων classes, οι οποίες αντιστοιχούν σε διάφορα στάδια διάβρωσης των νευρώνων, με την πλειοψηφία των δειγμάτων να ανήκει σε δύο εξ αυτών.[50] Το dataset αποτελείται από περίπου 5000 εικόνες, ενώ οργανώνεται σε φακέλους train και test, απουσίας αρχείου μητρώου. Καθώς η βάση πάσχει από σοβαρές ελλείψεις, αναμένεται χαμηλή απόδοση από το τελικό μοντέλο. Για αυτό τον λόγο, θα πραγματοποιηθούν δύο διαφορετικές προσεγγίσεις ως προς το πρόβλημα αυτό. Πρώτα θα γίνει σχεδιασμός του μοντέλου με τις μεθόδους που έχουν χρησιμοποιηθεί και στα προηγούμενα παραδείγματα. Αφού το δίκτυο αυτό αξιολογηθεί, θα γίνει εκπαίδευση ενός δευτέρου μέσω μιας διαδικασίας που ονομάζεται **transfer learning**. Η μέθοδος αυτή κάνει χρήση της αρχιτεκτονικής ήδη εδραιωμένων μοντέλων, τα οποία έχουν εκπαιδευτεί σε μια πληθώρα δεδομένων. Χρησιμοποιώντας τα weights και convolution layers αυτών ως feature extractors, είναι δυνατή η βελτίωση του τελικού αποτελέσματος. Στην προκειμένη περίπτωση θα γίνει χρήση του μοντέλου VGG19, το οποίο είναι σχεδιασμένο έτσι ώστε να επιτυγχάνει γενίκευση σε μια πληθώρα προβλημάτων classification.[51]

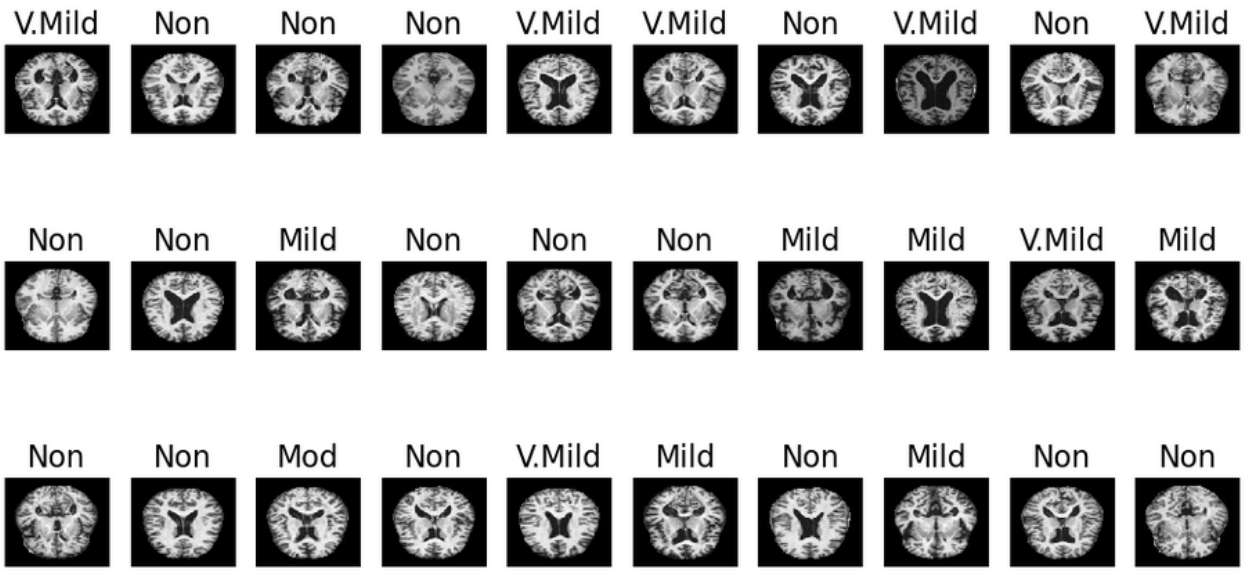
Εφαρμογή Custom Αρχιτεκτονικής

Ξεκινώντας, τα δεδομένα εισάγονται σε τελεστές με την ίδια διαδικασία που ακολουθήθηκε στο μοντέλο περί πνευμονίας. Έπειτα, καταγράφεται το πλήθος των στοιχείων ανα class και οπτικοποιούνται τα δεδομένα.

```
num_mild = len(os.listdir(os.path.join(train_dir, 'MildDemented')))
num_moderate = len(os.listdir(os.path.join(train_dir, 'ModerateDemented')))
num_non = len(os.listdir(os.path.join(train_dir, 'NonDemented')))
num_vmild = len(os.listdir(os.path.join(train_dir, 'VeryMildDemented')))
print(f"Patients with Very Mild Dementia = {num_vmild}")
print(f"Patients with Mild Dementia = {num_mild}")
print(f"Patients with Moderate Dementia = {num_moderate}")
print(f"Patients without Dementia = {num_non}")
```

```
Patients with Very Mild Dementia = 1792
Patients with Mild Dementia = 717
Patients with Moderate Dementia = 52
Patients without Dementia = 2560
```

Σχήμα 3.41: Αριθμός εικόνων για κάθε κατηγορία. Η μεγάλη διαφορά μεταξύ των δύο κύριων classes και των υπολοίπων θα οδηγήσει σε μειωμένες αποδόσεις.



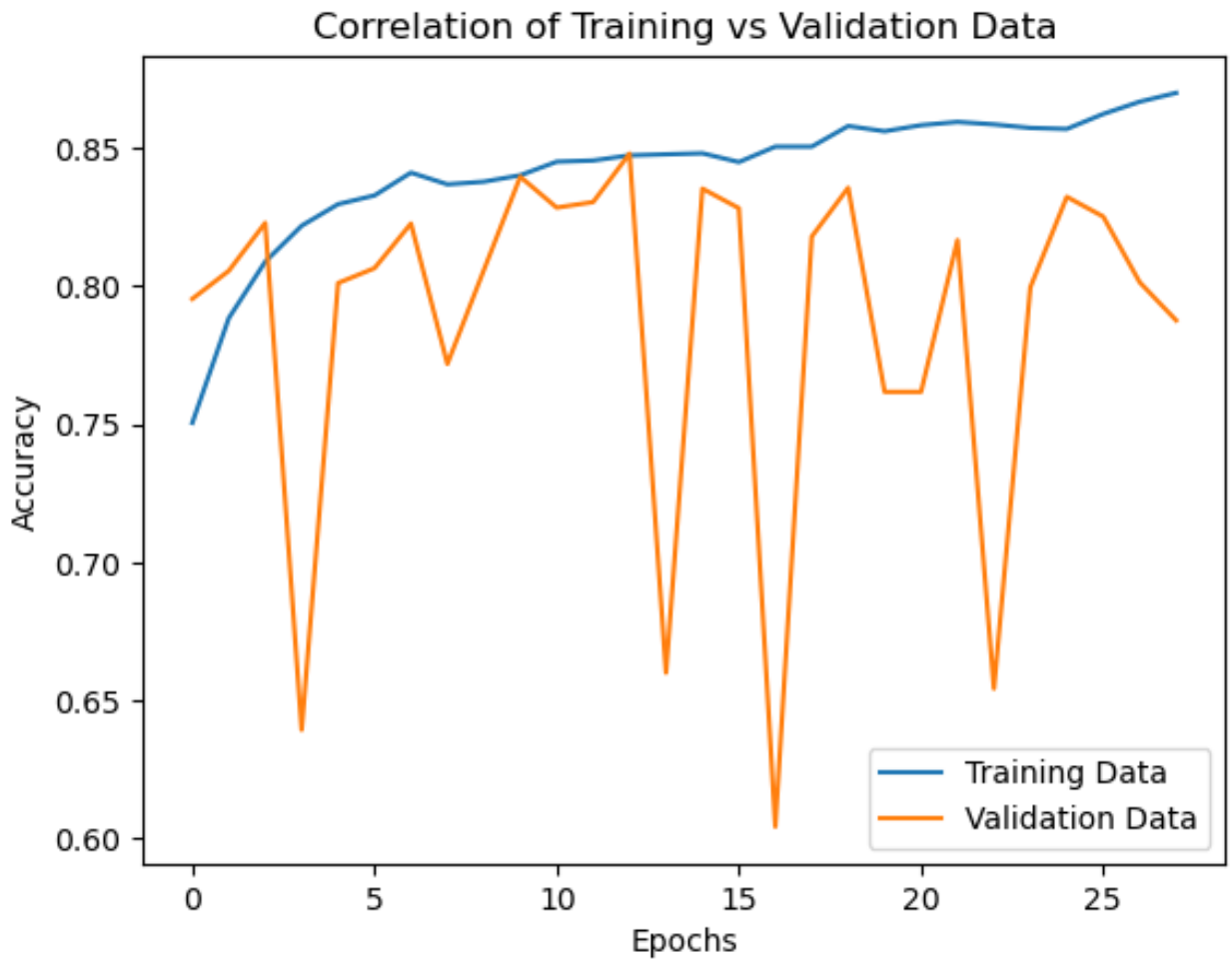
Σχήμα 3.42: Απεικόνιση δειγμάτων. Αν και τα δείγματα επιλέγονται τυχαία, ενδιαφέρον παρουσιάζει η έλλειψη ομοιομορφίας ως προς τις κατηγορίες αυτών.

Εν συνεχεία, ορίζεται η αρχιτεκτονική του μοντέλου. Καθώς η βάση δεδομένων είναι imbalanced, θα πραγματοποιηθεί data augmentation, με προσοχή ως προς την επεξεργασία, έτσι ώστε να αποφευχθεί το σενάριο αδυναμίας εξαγωγής λεπτομερειών από τις διαθέσιμες εικόνες. Η αρχιτεκτονική ακολουθεί λογική παρόμοια με τις προηγούμενες, με μια βασική εξαίρεση. Στο output του τελευταίου convolution layer θα εφαρμοστεί μια μέθοδος που ονομάζεται **batch normalization**. Μέσω αυτής, τα δεδομένα στην έξοδο θα κανονικοποιηθούν έτσι ώστε να έχουν μέση τιμή μηδέν και τυπική απόκλιση μονάδα, με στόχο την μείωση του overfitting.

```
model.add(Conv2D(32, (3,3), strides=(1,1), padding = "same"))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size = (2,2)))

model.add(Conv2D(64, (3,3), strides=(1,1), padding = "same"))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size = (2,2)))

model.add(Conv2D(128, (3,3), strides=(1,1), padding = "same"))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(BatchNormalization())

model.add(Flatten())
model.add(Dense(128))
model.add(Activation("relu"))
model.add(Dropout(0.2))

model.add(Dense(64))
model.add(Activation("relu"))
model.add(Dropout(0.2))

#Layer 5
model.add(Dense(4))
model.add(Activation("softmax"))
```

Σχήμα 3.43: Αρχιτεκτονική δικτύου απουσίας transfer learning. Η εισαγωγή περισσοτέρων dense layers γίνεται έτσι ώστε να εξαχθούν περισσότερες πληροφορίες πριν την χρήση τεχνικών dropout.

Τέλος, διεξάγεται η εκπαίδευση και πραγματοποιείται αξιολόγηση ως προς τα metrics και την ακρίβεια πρόβλεψης ανα class. Η χρήση του accuracy ως αξιόπιστο metric στην συγκεκριμένη εφαρμογή είναι άσκοπη, καθώς δεν προβλέπεται η τιμή της να αυξηθεί, λόγω της χαμηλής ποιότητας της βάσης δεδομένων. Αν και το accuracy θα καταγραφεί για τυπικούς λόγους, το κύριο metric αξιολόγησης θα είναι το ROC-AUC, όπου βαθμολογείται η πιθανότητα σωστής πρόβλεψης, σε συνδυασμό με τα ποσοστά επιτυχών προβλέψεων ανά class.




```
Accuracy, loss and AUC scored compared to training data is: 0.5627135038375854 , 0.8652839064598083 , 0.8561987280845642
Accuracy, loss and AUC scored compared to validation data is: 0.567937433719635 , 0.8642225861549377 , 0.8478372097015381
Accuracy, loss and AUC scored compared to test data is: 0.5527756214141846 , 0.8999865055084229 , 0.8423264026641846
```

```
Number of correct predictions for class MildDemented to total is:  0.0
Number of correct predictions for class ModerateDemented to total is:  0.0
Number of correct predictions for class NonDemented to total is:  0.7453125
Number of correct predictions for class VeryMildDemented to total is:  0.5133928571428571
```

Σχήμα 3.44: Γραφική παράσταση training/validation AUC custom αρχιτεκτονικής, evaluation metrics και προβλέψεις ανά κατηγορία δειγμάτων.

Η πρώτη παρατήρηση γίνεται στην αστάθεια της τιμής του validation AUC κατά την εκπαίδευση, όπως καταγράφεται στην γραφική παράσταση του σχήματος 3.44. Πρόκειται για αποτέλεσμα της χρήσης batch normalization, χωρίς απαραίτητα να μεταφράζεται ως αρνητικό φαινόμενο, καθώς η κανονικοποίηση μπορεί να επηρεάσει προσωρινά τα weights, όταν αυτά ανανεώνονται. Η χρήση callbacks είναι αρκετή, έτσι ώστε να διασφαλιστεί το βέλτιστο δυνατό δίκτυο στο τέλος της εκπαίδευσης. Πέραν αυτού, παρουσιάζεται ένα μικρό overfit στο διάστημα $2-3\%$,

κάτι που επιβεβαιώνεται και στο evaluation, όπου αποτυπώνεται και η πολύ χαμηλή επίδοση ως προς το accuracy, η οποία ήταν και αναμενόμενη. Στο prediction παρατηρείται πως το δίκτυο αδυνατεί να αναγνωρίσει περιπτώσεις ήπιας και σοβαρής άνοιας στους ασθενείς, οι οποίες ήταν και αυτές με τα μικρότερα πλήθη δειγμάτων. Το μοντέλο επιδεικνύει bias ως προς τις δύο κυρίαρχες classes και συνεπώς δεν μπορεί ως έχει να χρησιμοποιηθεί σε πρακτικές εφαρμογές. Επόμενο βήμα είναι να εξεταστεί αν η χρήση transfer learning δύναται να λύσει το πρόβλημα αυτό.

Transfer Learning με Χρήση VGG19

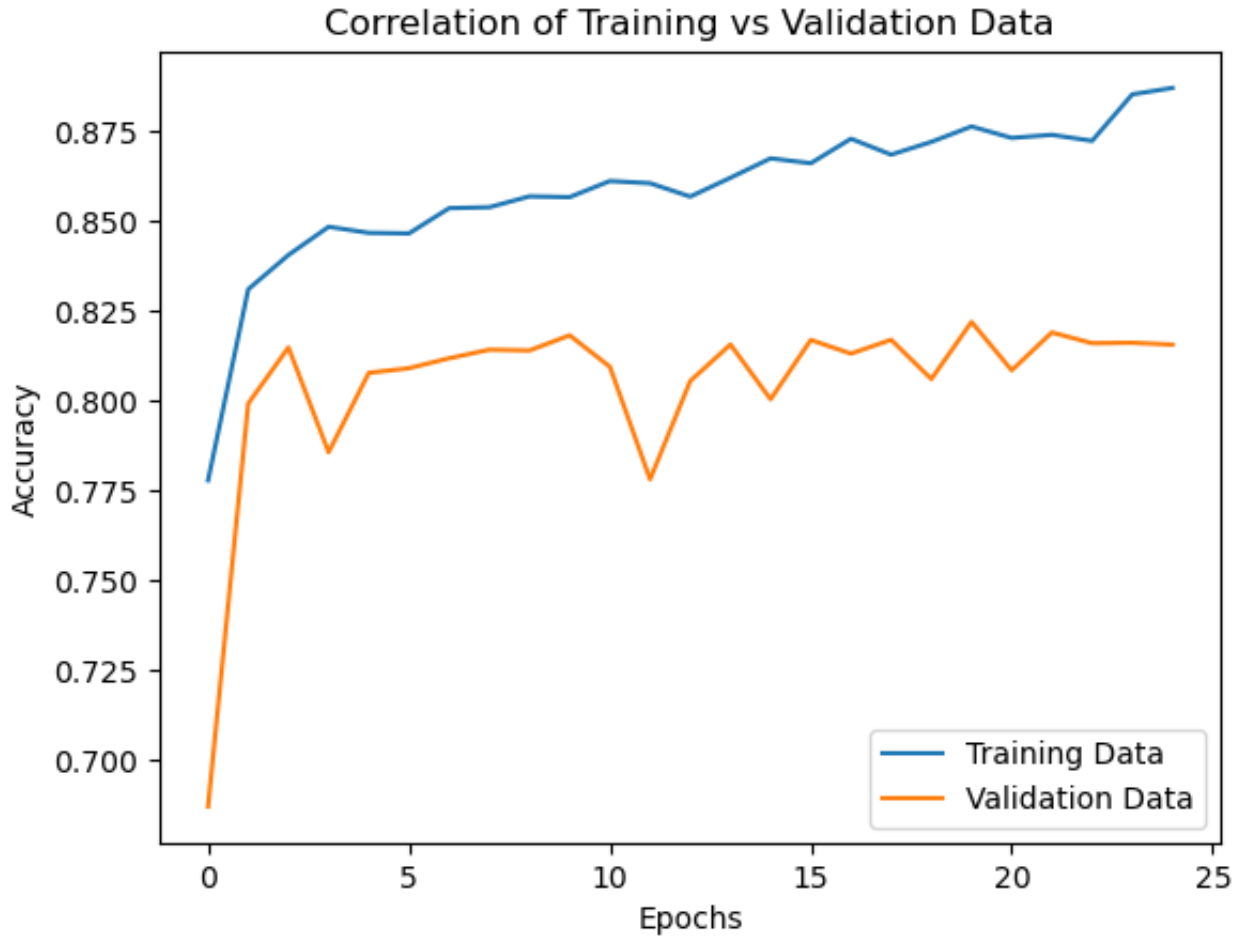
Η επεξεργασία των δεδομένων μέχρι το σημείο ορισμού της αρχιτεκτονικής του δικτύου παραμένει ίδια με την πρώτη περίπτωση. Καθώς το transfer learning χρησιμοποιεί μια υπάρχουσα δομή, πρέπει να γίνουν κάποια επιπλέον βήματα πριν προχωρήσει κανείς στην εκπαίδευση.

```
transfer_model = tf.keras.applications.VGG19(input_shape=(176,208,3), include_top=False, weights='imagenet')
for layer in transfer_model.layers:
    layer.trainable = False

input = tf.keras.Input(shape = (176, 208, 3))
x = tf.keras.layers.RandomFlip('horizontal_and_vertical', input_shape = (176, 208, 3))(input)
x = tf.keras.layers.RandomRotation(0.2, input_shape = (176, 208, 3))(x)
x = transfer_model(x)
x = Flatten()(x)
x = tf.keras.layers.Dense(128, activation = 'relu')(x)
x = tf.keras.layers.Dropout(0.1)(x)
prediction = tf.keras.layers.Dense(4, activation = 'softmax')(x)
model = tf.keras.Model(input, prediction)
model.compile(loss = "categorical_crossentropy", optimizer = "adam", metrics = ['accuracy', 'AUC'])
```

Σχήμα 3.45: Μπλόκ κώδικα υπεύθυνο για τον ορισμό δικτύου με χρήση transfer learning.

Γίνεται απευθείας αντιληπτό πως η διάταξη του δικτύου αυτού, όπως φαίνεται στο σχήμα 3.45, είναι διαφορετική σε σχέση με τα προηγούμενα. Αυτό συμβαίνει διότι, για να χρησιμοποιηθεί ένα έτοιμο δίκτυο ως βάση άλλων, δεν μπορεί να κληθεί το Sequential API. Αντι αυτού, απαιτείται η χρήση του λεγόμενου Functional API. Το Functional API επιτρέπει μη-γραμμικές συνδέσεις σε αρχιτεκτονικές, όπου γίνεται, παραδείγματος χάριν, μια είσοδος να συνδέεται με δύο παράλληλα layers, τα οποία εκτελούν διαφορετικές λειτουργίες και καταλήγουν σε διαφορετικά outputs ή να τοποθετηθούν δύο ξεχωριστά μοντέλα σε σειρά, όπου το output του ενός αποτελεί την είσοδο του επομένου, όπως σε εφαρμογές transfer learning. Πριν γίνει ορισμός της αρχιτεκτονικής, θα πρέπει να καταφορτωθεί το VGG19, μαζί με τα weights του. Ταυτοχρόνως, τα classification layers του μοντέλου εξαιρούνται μέσω της εντολής **include_top = False**, έτσι ώστε να πραγματοποιεί αποκλειστικά feature extraction. Στη συνέχεια, διευκρινίζεται πως τα layers δεν θα υπόκεινται σε ανανεωμένα weights κατά την εκπαίδευση, καθώς θα γίνει χρήση των ήδη διαθέσιμων. Αφού γίνουν τα παραπάνω, ξεκινά ο ορισμός της τελικής διάταξης. Αφού εισαχθεί η αρχική είσοδος **Input**, όπου ορίζεται το σχήμα των δεδομένων, δημιουργείται μια αλυσίδα layers, η οποία καταλήγει σε μια μεγαλύτερη δομή. Συγκεκριμένα, το Input συνδέεται με δύο data augmentation layers (ίδια με το προηγούμενο μοντέλο για λόγους συνοχής ως προς τη σύγκριση των δύο μεθόδων), αποτελώντας την είσοδο του VGG19, το οποίο, αφού

εκτελέσει feature extraction καταλήγει σε ένα fully connected layer και στο τελικό output. Η σύνδεση όλων αυτών των δομικών στοιχείων του δικτύου γίνεται καλώντας το προηγούμενο layer στο τέλος της εισαγωγής του επομένου. Η υπόλοιπη διαδικασία παραμένει ίδια, όπως και προηγουμένως. Παρατίθενται τα αποτελέσματα της εκπαίδευσης, τα evaluation metrics και τα ποσοστά ακρίβειας των προβλέψεων του μοντέλου αυτού.




```
Accuracy, loss and AUC scored compared to training data is: 0.598340630531311 , 0.8309465646743774 , 0.8702921271324158
Accuracy, loss and AUC scored compared to validation data is: 0.528836727142334 , 0.9635621309280396 , 0.8181109428405762
Accuracy, loss and AUC scored compared to test data is: 0.5480844378471375 , 0.9410459399223328 , 0.839423656463623
```

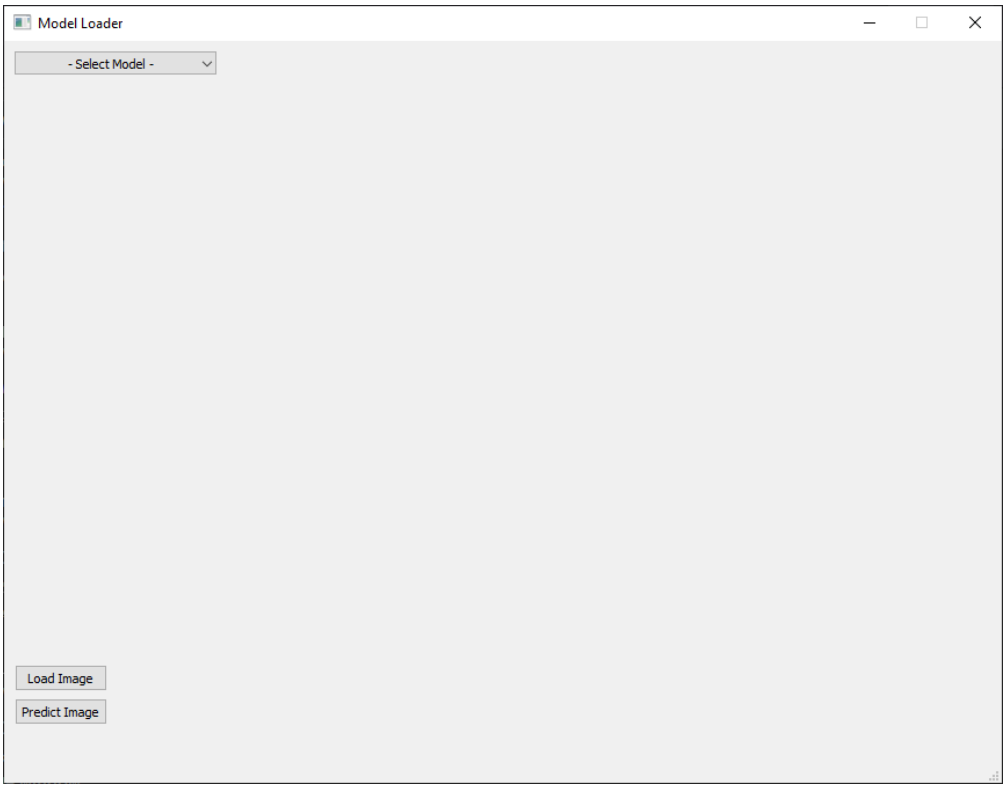
```
Number of correct predictions for class MildDemented to total is:  0.1005586592178771
Number of correct predictions for class ModerateDemented to total is:  0.0
Number of correct predictions for class NonDemented to total is:  0.915625
Number of correct predictions for class VeryMildDemented to total is:  0.21651785714285715
```

Σχήμα 3.46: Γραφική παράσταση training/validation AUC με transfer learning, evaluation metrics και προβλέψεις ανά κατηγορία δειγμάτων.

Δεν παρουσιάζεται σημαντική βελτιώση όσο αναφορά την τιμή του AUC στη γραφική παράσταση του σχήματος 3.46. Αν και η τιμή αυξήθηκε, το ίδιο συνέβη και με το overfit. Αυτό επιβεβαιώνεται και στο evaluation, με το AUC να παρουσιάζει τιμή σχεδόν ίδια με του προηγούμενου δικτύου, ενώ το accuracy παραμένει χαμηλό. Το bias που εντοπίστηκε στις προβλέψεις του μοντέλου απουσίας transfer learning υφίσταται και στην παρούσα περίπτωση, αν και αυξήθηκε μερικώς η ικανότητα πρόβλεψης ως προς την class ήπιας άνοιας. Η μέθοδος transfer learning απέτυχε να βελτιώσει τις αποδόσεις του μοντέλου, το οποίο κρίνεται αναξιόπιστο και συνεπώς δεν είναι δυνατή η χρήση του για την πραγματοποίηση προβλέψεων σε πραγματικές συνθήκες.

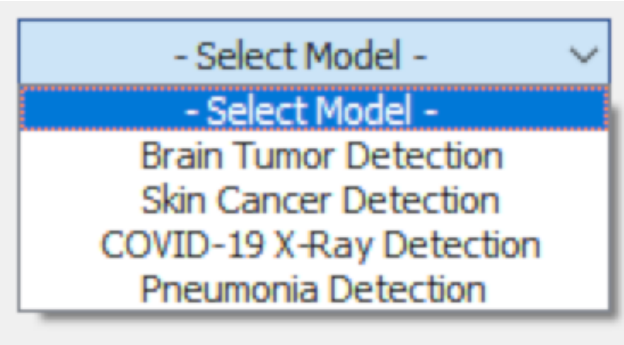
3.4 Χρήση Μοντέλων Μέσω Διαδραστικού Προγράμματος

Μέχρι στιγμής, η διαδικασία πρόβλεψης διεξαγόταν μέσω κώδικα, ο οποίος γράφτηκε με βασικό στόχο την εκπαίδευση. Αν επιθυμείται η χρήση των μοντέλων από κάποιον τελικό χρήστη, ο οποίος δεν είναι εξοικειωμένος με προγραμματισμό και τις βασικές αρχές των βαθιών νευρωνικών δικτύων, θα πρέπει να αναπτυχθεί πρόγραμμα το οποίο είναι ικανό να τα φορτώσει και χρησιμοποιήσει όσο πιο απλά γίνεται. Αυτό μπορεί να επιτευχθεί μέσω ενός GUI, στο οποίο είναι δυνατόν να επιλέγεται το επιθυμητό μοντέλο, να εισάγεται η σχετική εικόνα και να εμφανίζεται η πρόβλεψη. Στο παρακάτω σχήμα απεικονίζεται η διάταξη ενός τέτοιο προγράμματος.



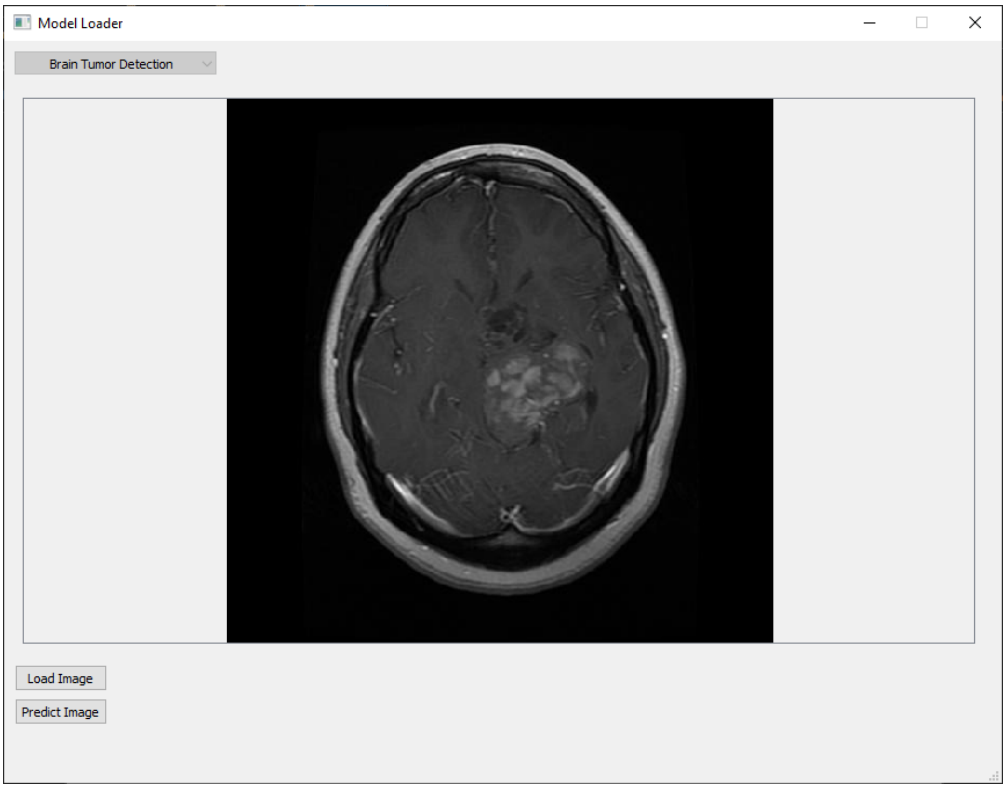
Σχήμα 3.47: Αρχική επιφάνεια GUI.

Ξεκινώντας, ο χρήστης πρέπει να επιλέξει την λίστα στο επάνω-αριστερά μέρος του παραθύρου. Εκεί εμφανίζονται όλα τα μοντέλα που είναι διαθέσιμα προς χρήση. Μόλις επιλεχθεί το επιθυμητό μοντέλο, φορτώνεται μέσω Tensorflow και είναι έτοιμο να πραγματοποιήσει πρόβλεψη.



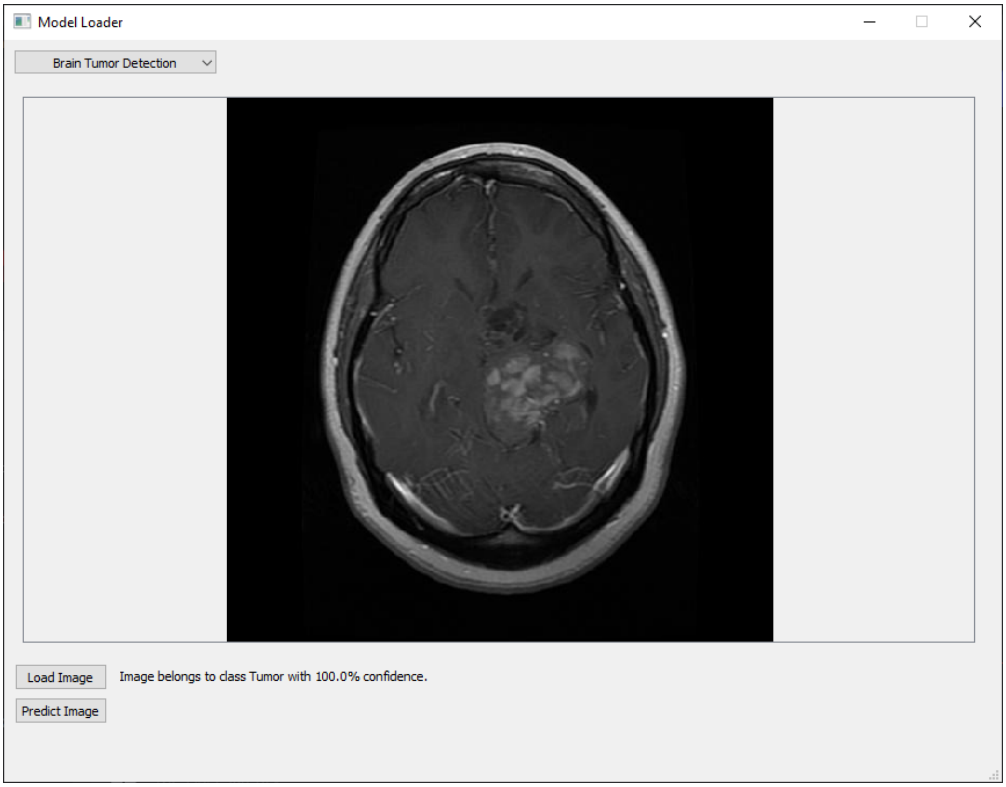
Σχήμα 3.48: Λίστα διαθέσιμων μοντέλων.

Στη συνέχεια, θα πρέπει να εισαχθεί η εικόνα προς ταξινόμηση. Πατώντας το κουμπί "Load Image", ανοίγει αυτόματα παράθυρο διαλόγου, από το οποίο μπορεί να γίνει επιλογή του αρχείου. Η εφαρμογή υποστηρίζει εικόνες τύπου .jpg, .png, .bmp και .gif. Αφού η εικόνα επιλεχθεί, προβάλλεται στο παράθυρο.



Σχήμα 3.49: Παράδειγμα εικόνας που έχει φορτωθεί στην εφαρμογή.

Το τελευταίο βήμα είναι αυτό της πρόβλεψης. Το μόνο που χρειάζεται είναι να πατηθεί το κουμπί με την ονομασία "Predict Image". Αυτομάτως, στο κάτω μέρος του προγράμματος θα εμφανιστεί η class στην οποία ταξινομείται η εικόνα σύμφωνα με το επιλεγμένο μοντέλο.



Σχήμα 3.50: Πρόβλεψη εικόνας. Στο παράδειγμα αυτό γίνεται χρήση του μοντέλου εγκεφαλικών όγκων, με την εικόνα να προέρχεται από το αντίστοιχο υποσύνολο test data.

Με το γραφικό περιβάλλον έτοιμο, το μόνο που χρειάζεται είναι η εξαγωγή του κώδικα σε μορφή προγράμματος .exe, έτσι ώστε να μπορεί να χρησιμοποιηθεί απουσίας Python interpreter. Υπάρχουν πολλές διαθέσιμες βιβλιοθήκες, οι οποίες είναι ικανές να πραγματοποιήσουν την διαδικασία αυτή. Στην συγκεκριμένη περίπτωση επιλέχθηκε το PyInstaller, εργαλείο που εντοπίζει αυτόματα τις απαραίτητες βιβλιοθήκες για την λειτουργία του προγράμματος και δημιουργεί ένα πλήρες πακέτο, έτοιμο προς χρήση από οποιονδήποτε ενδιαφερόμενο, χωρίς επιπλέον βήματα. Η εξάρτηση από τις βιβλιοθήκες που χρησιμοποιήθηκαν κατά την ανάπτυξη οδηγεί στο ότι, ανεξαρτήτως υπολογιστή, το πρόγραμμα θα χρησιμοποιεί Tensorflow για την εκτέλεση εργασιών πρόβλεψης. Συνεπώς, υφίσταται πρόβλημα απόδοσης ως προς την διαθέσιμη υπολογιστική δύναμη, σε περιπτώσεις όπου το μηχάνημα του τελικού χρήστη δεν πληρεί κατάλληλες προδιαγραφές.

Κεφάλαιο 4

Συμπεράσματα

Η εργασία αυτή είχε ως στόχο την ανάπτυξη νευρωνικών δικτύων για την πραγματοποίηση ταξινόμησης βιοϊατρικών δεδομένων διαφόρων παθήσεων. Ξεκινώντας, έγινε εισαγωγή στην μηχανική μάθηση, στις κατηγορίες, εφαρμογές και τρόπους αξιολόγησης των διαφόρων μεθόδων αυτής. Στη συνέχεια, αναλύθηκαν τα **τεχνητά νευρωνικά δίκτυα (ANN)**, η δομή αυτών, οι θεμελιώδεις έννοιες που τα χαρακτηρίζουν και επηρεάζουν την απόδοσή τους, καθώς και κάποια παραδείγματα υλοποίησής τους. Έπειτα, μελετήθηκε μια βασική υποκατηγορία ANN, τα **βαθιά νευρωνικά δίκτυα (DNN)**, όπου αναλύθηκε ο τρόπος κατασκευής, τα διαφορά στοιχεία που τα απαρτίζουν, η λειτουργία τους, καθώς και παραδείγματα αυτών. Κλείνοντας το θεωρητικό μέρος, έγινε αναφορά στα **Convolutional Neural Networks (CNN)**, τα οποία αποτελούν τον βασικό πυλώνα εφαρμογών τύπου **image classification**. Εξηγήθηκαν όλα τα δομικά στοιχεία ενός **convolution layer**, ο τρόπος λειτουργίας του και το πως μπορεί να χρησιμοποιηθεί για ταξινόμηση δεδομένων, καθώς και η διάταξη ενός πλήρους CNN. Έπειτα, παρατέθηκαν κάποια εργαλεία μείζονος σημασίας ως προς την ανάπτυξη νευρωνικών δικτύων και τελικού προγράμματος, στα οποία δύναται να ενσωματωθούν. Ακολούθως, πραγματοποιήθηκε πλήρης ανάλυση ως προς την ανάπτυξη δύο μορφών μοντέλων image classification, **binary** και **categorical**. Αφού αναφέρθηκαν συνήθεις βιβλιοθήκες και μέθοδοι επεξεργασίας δεδομένων, παρουσιάστηκε ο τρόπος υλοποίησης μιας αρχιτεκτονικής νευρωνικών δικτύων, η εκπαίδευση αυτών και δυνατές μέθοδοι αξιολόγησης. Εν συνεχεία, μελετήθηκαν άλλα τρία παραδείγματα μοντέλων, πιθανές επιπλοκές ως προς την δημιουργία αυτών και διάφορες τεχνικές βελτίωσης στην απόδοσή τους. Τέλος, εξηγήθηκε πως τα μοντέλα αυτά μπορούν να χρησιμοποιηθούν από μη-εξειδικευμένους χρήστες μέσω ενός γραφικού περιβάλλοντος χρήστη (GUI).

Από τα αποτελέσματα των ταξινομήσεων που πραγματοποιήθηκαν, μέσω των κατά τη διάρκεια της εργασίας αναπτυγμένων μοντέλων, παρατηρείται πως η εφαρμογή image classification ως συμβουλευτικό εργαλείο στη διάγνωση παθήσεων είναι βιώσιμη. Η ιδιότητα των μοντέλων να προσφέρουν μια δεύτερη γνώμη επί των δεδομένων που επεξεργάζονται έχει τη δυνατότητα να διευκολύνει το έργο ιατρικών προσωπικών, όπως φαίνεται ευκόλως στις περιπτώσεις εγκεφαλικών όγκων και πνευμονίας, τα οποία χαρακτηρίζονται από εξαιρετική ακρίβεια ως προς τις προβλέψεις τους. Παρ'όλη τη χρησιμότητα τους όμως, υπάρχουν εμπόδια τα οποία δύναται να δυσκολέψουν την ανάπτυξη τέτοιων μεθόδων. Σε παραδείγματα όπου οι classes των databases παρουσίαζαν ανομοιομορφία, ο τελικός αλγόριθμος υπέκειτο σε φαινόμενα bias, όπου υπήρχε προτίμηση ως προς συγκεκριμένες ταξινομήσεις. Άλλο πρόβλημα είναι αυτό της σχέσης μεταξύ

αρχιτεκτονικής και εξαγόμενων αποτελεσμάτων. Καθώς η εξάρτηση των προβλέψεων από τις διάφορες hyperparameters, τον αριθμό και τη φύση των layers, και του τρόπου αξιολόγησης, είναι άμεση, και πηγάζει εν μέρει από τη φύση των διαθέσιμων δεδομένων και της επεξεργασίας που δέχονται, η διαδικασία καθίσταται επί το πλείστον εμπειρική. Συνέπεια αυτού είναι πως, αν και η πρόσβαση στα απαραίτητα εργαλεία είναι εύκολη, υπάρχει ευμεγέθης βαθμός δυσκολίας ως προς τη δημιουργία ενός αποδοτικού μοντέλου. Τέλος, είναι σημαντικό να γίνει αναφορά ως προς την υπολογιστική δύναμη. Η εκμάθηση ενός αλγορίθμου είναι απαιτητική ως προς τους διαθέσιμους πόρους ενός συστήματος, που σημαίνει ότι δεν έχουν όλοι την ικανότητα να αναπτύξουν περίπλοκα νευρωνικά δίκτυα με την ίδια ευχέρεια. Σημειώνεται πως οι εφαρμογές βαθιάς μάθησης δεν περιορίζονται μόνο στο image classification. Ιδιαίτερο ενδιαφέρον παρουσιάζουν οι τεχνικές object detection, οι οποίες, με τα κατάλληλα δεδομένα, μπορούν όχι μόνα να αναγνωρίζουν τις εμπεριεχόμενες παθήσεις, αλλά επίσης και να υποδείξουν την περιοχή στην οποία εντοπίζονται. Τέλος, συνεργασία με υγειονομικούς φορείς θα μπορούσε να αποτελέσει πρόσβαση σε ένα πλήθος δεδομένων, τα οποία θα επιτρέψουν την ανάπτυξη πιο αξιόπιστων και ποικιλόμορφων μοντέλων.

Βιβλιογραφία

[1] A. M. Turing. "Computing Machinery and Intelligence". English. In: *Mind*. New Series 59.236 (1950), pp. 433–460. ISSN: 00264423. URL: http://www.jstor.org/stable/2251299.

[2] A. L. Samuel. "Some Studies in Machine Learning Using the Game of Checkers". In: *IBM Journal of Research and Development* 3.3 (1959), pp. 210–229. DOI: 10.1147/rd.33.0210.

[3] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. 1st ed. Springer, 2007. Chap. 1, p. 3. ISBN: 0387310738.

[4] R.M. Golden. "Statistical Pattern Recognition". In: *International Encyclopedia of the Social & Behavioral Sciences*. Ed. by Neil J. Smelser and Paul B. Baltes. Oxford: Pergamon, 2001, p. 15043. ISBN: 978-0-08-043076-8. DOI: https://doi.org/10.1016/B0-08-043076-7/00619-7.

[5] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. 1st ed. Springer, 2007. Chap. 1, p. 32. ISBN: 0387310738.

[6] Khadija El Bouchefry and Rafael S. de Souza. "Chapter 12 - Learning in Big Data: Introduction to Machine Learning". In: *Knowledge Discovery in Big Data from Astronomy and Earth Observation*. Ed. by Petr Škoda and Fathalrahman Adam. Elsevier, 2020, p. 228. ISBN: 978-0-12-819154-5. DOI: https://doi.org/10.1016/B978-0-12-819154-5.00023-0.

[7] Derek Greene, Pádraig Cunningham, and Rudolf Mayer. "Unsupervised Learning and Clustering". In: *Machine Learning Techniques for Multimedia: Case Studies on Organization and Retrieval*. Ed. by Matthieu Cord and Pádraig Cunningham. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, p. 52. ISBN: 978-3-540-75171-7. DOI: 10.1007/978-3-540-75171-7_3. URL: https://doi.org/10.1007/978-3-540-75171-7_3.

[8] Michail Vlachos. "Dimensionality Reduction". In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, p. 274. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_216. URL: https://doi.org/10.1007/978-0-387-30164-8_216.

[9] Sandhya Armoogum and XiaoMing Li. "Chapter 2 - Big Data Analytics and Deep Learning in Bioinformatics With Hadoop". In: *Deep Learning and Parallel Computing Environment for Bioengineering Systems*. Ed. by Arun Kumar Sangaiah. Academic Press, 2019, p. 29. ISBN: 978-0-12-816718-2. DOI: https://doi.org/10.1016/B978-0-12-816718-2.00009-9.

[10] Andrew G. Barto. "Chapter 2 - Reinforcement Learning". In: *Neural Systems for Control*. Ed. by Omid Omidvar and David L. Elliott. San Diego: Academic Press, 1997, p. 8. ISBN: 978-0-12-526430-3. DOI: https://doi.org/10.1016/B978-012526430-3/50003-9.

[11] Lukas Budach et al. "The Effects of Data Quality on Machine Learning Performance". In: 2022, pp. 1–2. arXiv: 2207.14529 [cs.DB].

[12] Jeremy Petch, Shuang Di, and Walter Nelson. "Opening the Black Box: The Promise and Limitations of Explainable Machine Learning in Cardiology". In: *Canadian Journal of Cardiology* 38.2 (2022). Focus Issue: New Digital Technologies in Cardiology, pp. 204–205. ISSN: 0828-282X. DOI: https://doi.org/10.1016/j.cjca.2021.09.004.

[13] Agnieszka Mikołajczyk-Bareła and Michał Grochowski. "A survey on bias in machine learning research". In: 2023, p. 5. arXiv: 2308.11254 [cs.LG].

[14] Robert J. Hill Miriam Steurer and Norbert Pfeifer. "Metrics for evaluating the performance of machine learning based automated valuation models". In: *Journal of Property Research* 38.2 (2021), pp. 103–111. DOI: 10.1080/09599916.2020.1858937.

[15] Ajay Kulkarni, Deri Chong, and Feras A. Batarseh. "5 - Foundations of data imbalance and solutions for a data democracy". In: *Data Democracy*. Ed. by Feras A. Batarseh and Ruixin Yang. Academic Press, 2020, pp. 86–89. ISBN: 978-0-12-818366-3. DOI: https://doi.org/10.1016/B978-0-12-818366-3.00005-8.

[16] Anke Meyer-Baese and Volker Schmid. "Chapter 6 - Statistical and Syntactic Pattern Recognition". In: *Pattern Recognition and Signal Analysis in Medical Imaging (Second Edition)*. Ed. by Anke Meyer-Baese and Volker Schmid. Second Edition. Oxford: Academic Press, 2014, p. 181. ISBN: 978-0-12-409545-8. DOI: https://doi.org/10.1016/B978-0-12-409545-8.00006-6.

[17] Konstantinos G. Liakos et al. "Machine Learning in Agriculture: A Review". In: *Sensors* 18.8 (2018). ISSN: 1424-8220. DOI: 10.3390/s18082674. URL: https://www.mdpi.com/1424-8220/18/8/2674.

[18] Ziqiu Kang, Cagatay Catal, and Bedir Tekinerdogan. "Machine learning applications in production lines: A systematic literature review". In: *Computers & Industrial Engineering* 149 (2020), p. 106773. ISSN: 0360-8352. DOI: https://doi.org/10.1016/j.cie.2020.106773.

[19] Peter Svenmarck et al. "Possibilities and Challenges for Artificial Intelligence in Military Applications". In: May 2018, pp. 1–2.

[20] Mohammad Shehab et al. "Machine learning in medical applications: A review of state-of-the-art methods". In: *Computers in Biology and Medicine* 145 (2022), p. 105458. ISSN: 0010-4825. DOI: https://doi.org/10.1016/j.compbiomed.2022.105458.

[21] Chigozie Nwankpa et al. "Activation Functions: Comparison of trends in Practice and Research for Deep Learning". In: 2018, pp. 5, 8. arXiv: 1811.03378 [cs.LG].

[22] Paul Munro. "Backpropagation". In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 69–70. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_51. URL: https://doi.org/10.1007/978-0-387-30164-8_51.

[23] Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. "Activation Functions in Deep Learning: A Comprehensive Survey and Benchmark". In: 2022, pp. 2–9. arXiv: 2109.14545 [cs.LG].

[24] Juan Terven et al. "Loss Functions and Metrics in Deep Learning". In: 2023, pp. 8–16. arXiv: 2307.02694 [cs.LG].

[25] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: 2017, pp. 2–10. arXiv: 1609.04747 [cs.LG].

[26] Robin M. Schmidt. "Recurrent Neural Networks (RNNs): A gentle Introduction and Overview". In: *CoRR* abs/1912.05911 (2019), p. 1. URL: http://arxiv.org/abs/1912.05911.

[27] S.J. Pawan and Jeny Rajan. "Capsule networks for image classification: A review". In: *Neurocomputing* 509 (2022), pp. 102–120. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2022.08.073.

[28] Fionn Murtagh. "Multilayer perceptrons for classification and regression". In: *Neurocomputing* 2.5 (1991), pp. 188–189. ISSN: 0925-2312. DOI: https://doi.org/10.1016/0925-2312(91)90023-5.

[29] Marius-Constantin Popescu et al. "Multilayer perceptron and neural networks". In: *WSEAS Transactions on Circuits and Systems* 8 (July 2009), p. 581.

[30] Sepp Hochreiter. "The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions". In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6 (Apr. 1998), pp. 107–108. DOI: 10.1142/S0218488598000094.

[31] Farhad Mortezapour Shiri et al. "A Comprehensive Overview and Comparative Analysis on Deep Learning Models: CNN, RNN, LSTM, GRU". In: 2023, pp. 4–8. arXiv: 2305.17473 [cs.LG].

[32] Alex Shenfield and Martin Howarth. "A Novel Deep Learning Model for the Detection and Identification of Rolling Element-Bearing Faults". In: *Sensors (Basel, Switzerland)* 20 (Sept. 2020), pp. 4–5. DOI: 10.3390/s20185112.

[33] Mohammad Mustafa Taye. "Theoretical Understanding of Convolutional Neural Network: Concepts, Architectures, Applications, Future Directions". In: *Computation* 11.3 (2023), pp. 5–8. ISSN: 2079-3197. DOI: 10.3390/computation11030052. URL: https://www.mdpi.com/2079-3197/11/3/52.

[34] Hossein Gholamalinezhad and Hossein Khosravi. "Pooling Methods in Deep Neural Networks, a Review". In: 2020, p. 3. arXiv: 2009.07485 [cs.CV].

[35] Keiron O'Shea and Ryan Nash. "An Introduction to Convolutional Neural Networks". In: 2015, p. 8. arXiv: 1511.08458 [cs.NE].

[36] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1951–1952. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[37] Akinori Hidaka and Takio Kurita. "Consecutive Dimensionality Reduction by Canonical Correlation Analysis for Visualization of Convolutional Neural Networks". In: vol. 2017. Dec. 2017, p. 161. DOI: 10.5687/sss.2017.160.

[38] Jiuxiang Gu et al. "Recent Advances in Convolutional Neural Networks". In: *CoRR* abs/1512.07108 (2015), pp. 21–27. arXiv: 1512.07108. URL: http://arxiv.org/abs/1512.07108.

[39] Martín Abadi et al. "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems". In: Software available from tensorflow.org. 2015, pp. 1–4. URL: https://www.tensorflow.org/.

[40] URL: https://www.tensorflow.org/guide/keras.

[41] Wes McKinney. "Data Structures for Statistical Computing in Python". In: *Proceedings of the 9th Python in Science Conference*. Ed. by Stéfan van der Walt and Jarrod Millman. 2010, pp. 56–61. DOI: 10.25080/Majora-92bf1922-00a.

[42] Preet Viradiya. *Brain Tumor Dataset*. May 16, 2021. URL: https://www.kaggle.com/datasets/preetviradiya/brian-tumor-dataset (visited on 09/22/2023).

[43] Kaggle. *GitHub - Kaggle/kaggle-api: Official Kaggle API*. URL: https://github.com/Kaggle/kaggle-api.

[44] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585 (2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2.

[45] John D Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in science & engineering* 9.3 (2007), pp. 90–95.

[46] *Skin cancer Dataset*. Version V1. Nov. 1, 2022. URL: https://www.kaggle.com/datasets/farjanakabirsamanta/skin-cancer-dataset (visited on 09/18/2023).

[47] Linda Wang, Zhong Qiu Lin, and Alexander Wong. "COVID-Net: a tailored deep convolutional neural network design for detection of COVID-19 cases from chest X-ray images". In: *Scientific Reports* 10.1 (Nov. 2020), p. 19549. ISSN: 2045-2322. DOI: 10.1038/s41598-020-76550-z. URL: https://doi.org/10.1038/s41598-020-76550-z.

[48] Daniel Kermany, Kang Zhang, and Michael H. Goldbaum. *Large Dataset of Labeled Optical Coherence Tomography (OCT) and Chest X-Ray Images*. Version V3. June 1, 2018. DOI: 10.17632/rscbjbr9sj.3. URL: http://data.mendeley.com/datasets/rscbjbr9sj/3 (visited on 09/28/2023).

[49] Zeinab Breijyeh and Rafik Karaman. "Comprehensive review on Alzheimer's disease: Causes and treatment". en. In: *Molecules* 25.24 (Dec. 2020), p. 5789.

[50] Dubey. *Alzheimer's Dataset (4 class of Images)*. Version V1. Dec. 26, 2019. URL: https://www.kaggle.com/datasets/tourist55/alzheimers-dataset-4-class-of-images/ (visited on 10/11/2023).

[51] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556 [cs.CV].

Παράρτημα

Παράρτημα Α

Brain Tumor Binary Classification Model

```
#Module Initialization
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import random
import shutil
import kaggle
import time
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Conv2D, Dense, Flatten, Dropout,
 Activation, MaxPooling2D, RandomFlip, RandomRotation
from tensorflow.keras.models import Sequential
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint,
 ReduceLROnPlateau

dl_path = 'path/to/folder'

if not os.path.exists(dl_path):
  kaggle.api.authenticate()
  kaggle.api.dataset_download_files('preetviradiya/brian-tumor-dataset',
  path = dl_path, unzip=True)
  print('Files successfully downloaded!')
else:
  print('Data has already been downloaded.')

data_path = dl_path + 'Brain Tumor Data Set/Brain Tumor Data Set/'
tumor_path = data_path + 'Brain Tumor/'
healthy_path = data_path + 'Healthy/'
main_folder = dl_path + 'Segmented Images/'

if not os.path.exists(main_folder):
  #Pathing the folders necessary
  train_folder = os.path.join(main_folder, 'train')
  val_folder = os.path.join(main_folder, 'val')
```

```
36    test_folder = os.path.join(main_folder, 'test')
37
38    #Defining the file format of the images
39    file_formats = ['.jpg', '.JPG', '.tif', '.png']
40
41    #List of image filenames
42    imgs_list = [filename for filename in os.listdir(tumor_path) if os.
      path.splitext(filename)[-1] in file_formats]
43    imgs_list2 = [filename for filename in os.listdir(healthy_path) if os.
      path.splitext(filename)[-1] in file_formats]
44
45    #Randomize the three sets using a seed and the image list generated
      above
46    random.seed(42)
47    random.shuffle(imgs_list)
48    random.shuffle(imgs_list2)
49
50    #Split data in ratios: 70% train, 15% validation, 15% test
51    train_size = int(len(imgs_list) * 0.70)
52    val_size = int(len(imgs_list) * 0.15)
53    test_size = int(len(imgs_list) * 0.15)
54
55    train_size2 = int(len(imgs_list2) * 0.70)
56    val_size2 = int(len(imgs_list2) * 0.15)
57    test_size2 = int(len(imgs_list2) * 0.15)
58
59    # Create destination folders if they don't exist
60    for folder_path in [train_folder, val_folder, test_folder]:
61      if not os.path.exists(folder_path):
62        os.makedirs(folder_path)
63
64    # Copy image files to destination folders
65    for i, f in enumerate(imgs_list):
66      if i < train_size:
67        dest_folder = train_folder
68      elif i < train_size + val_size:
69        dest_folder = val_folder
70      else:
71        dest_folder = test_folder
72      shutil.move(os.path.join(tumor_path, f), os.path.join(dest_folder, f
      ))
73
74    for i, f in enumerate(imgs_list2):
75      if i < train_size2:
76        dest_folder = train_folder
77      elif i < train_size2 + val_size2:
78        dest_folder = val_folder
79      else:
80        dest_folder = test_folder
81      shutil.move(os.path.join(healthy_path, f), os.path.join(dest_folder,
       f))
82    shutil.rmtree(tumor_path)
83    shutil.rmtree(healthy_path)
84    shutil.rmtree(dl_path + 'Brain Tumor Data Set/')
```

```
  else:
    print('Folders already exist!')

df = pd.read_csv(dl_path + 'metadata.csv')
df = df.rename(columns={"class": "status"})
#Assign classes to the imagesets by cross-referencing file names with the "class" column from the CSV file
#and prepare images for overall processing from the neural network.

datagen = ImageDataGenerator(rescale=1./255)
train_data = datagen.flow_from_dataframe(dataframe = df, directory = main_folder + 'train/', x_col = "image", y_col = "status", class_mode = "binary", target_size = (256, 256), batch_size = 32)
valid_data = datagen.flow_from_dataframe(dataframe = df, directory = main_folder + 'val/', x_col = "image", y_col = "status", class_mode = "binary", target_size = (256, 256), batch_size = 32)
test_data = datagen.flow_from_dataframe(dataframe = df, directory = main_folder + 'test/', x_col = "image", y_col = "status", class_mode = "binary", target_size = (256, 256), batch_size = 32)

# Code that ensures we correctly loaded the images to be used for training.
# This is used for visualization purposes.
def plots(ims, figsize=(12,6), rows=3, interp=False, titles=None):
  if type(ims[0]) is np.ndarray:
    ims = np.array(ims)
    if (ims.shape[-1] != 3):
      ims = ims.transpose((0,2,3,1))
  f = plt.figure(figsize=figsize)
  cols = len(ims)//rows if len(ims) % 2 == 0 else len(ims)//rows + 1
  for i in range(len(ims)):
    try:
      sp = f.add_subplot(rows, cols, i+1)
      sp.axis('Off')
    except ValueError:
      break
    if titles is not None:
      sp.set_title(titles[i], fontsize=16)
    plt.imshow(ims[i], interpolation=None if interp else 'none')


imgs, labels = next(train_data)
labels_new = []
for i in range(0,len(labels)):
  if np.allclose(labels[i],np.array([1])) == True:
    labels_new.append('Tumor')
  elif np.allclose(labels[i],np.array([0])) == True:
    labels_new.append('Healthy')
plots(imgs, titles = labels_new)

# Defining model architecture
model = Sequential()

# Pre-processing Layer
```

```
model.add(RandomFlip("horizontal_and_vertical", input_shape = (256, 256, 3)))
model.add(RandomRotation(0.1))

# Convolution Layer 1
model.add(Conv2D(32, (3,3), strides=(1,1), padding = "same"))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size = (2,2)))

# Convolution Layer 2
model.add(Conv2D(64, (3,3), strides=(1,1), padding = "same"))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size = (2,2)))

# Convolution Layer 3
model.add(Conv2D(128, (3,3), strides=(1,1), padding = "same"))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size = (2,2)))

# Fully Connected Layer
model.add(Flatten())
model.add(Dense(128))
model.add(Activation("relu"))
model.add(Dropout(0.15))

# Output
model.add(Dense(1))
model.add(Activation("sigmoid"))

# Compiling model
model.compile(loss = "binary_crossentropy", optimizer = "adam", metrics = ['accuracy', 'AUC'])

# We use a simple time module to record how much time the network took to train.
# This can be safely removed, unless the user is curious for learning/diagnostic purposes.
# If training on a CPU, the process will take significantly longer.
start_time = time.time()

# We use various callbacks to monitor the model's training and ensure minimum time losses in case
# the model starts to overfit/the process is lengthy enough to make manual monitoring impossible.
# It is generally recommended to use all 3 callbacks below to ensure best model performance.
#
# We use Early Stopping to monitor a specific metric and terminate training if there's no improvement
# after a set number of epochs. Checkpoint saves the model when a monitored value is improved.
# Finally, we can automatically tweak the learning rate of the model if our preferred monitored values
# shows no improvement after a certain number of epochs.
```

```
175  early_stopping = EarlyStopping(monitor='val_loss', patience = 6,
      restore_best_weights = True)
176  checkpoint = ModelCheckpoint("Brain Tumor Checkpoint.h5", monitor='
      val_accuracy', verbose = 1, save_best_only = True, mode='max')
177  lr_monitor = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience
      =3, min_lr=0.00001)
178
179  # Initiating training.
180  history = model.fit(train_data, validation_data = valid_data, epochs =
      30, verbose = 1, callbacks = [early_stopping, checkpoint, lr_monitor])
181
182  # Tensorflow recommends saving models in .keras format. To comply with
      that, we save the best model generated
183  # by the callback functions used above.
184  model.save('Brain Tumor Classification.keras')
185  print(time.time() - start_time, 'sec')
186
187  # Plot model accuracy for training and validation datasets
188  plt.plot(history.history['accuracy'], label = 'accuracy')
189  plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
190  plt.title('Correlation of Training vs Validation Data')
191  plt.xlabel('Epochs')
192  plt.ylabel('Accuracy')
193  plt.legend(['Training Data','Validation Data'], loc='lower right')
194
195  # Generate model summary
196  model.summary()
197
198  # Generate evaluation metrics for all 3 datasets
199  train_eval = model.evaluate(train_data, verbose = 0)
200  val_eval = model.evaluate(valid_data, verbose = 0)
201  test_eval = model.evaluate(test_data, verbose = 0)
202  print('Accuracy, loss and AUC scored compared to training data is:',
      train_eval[1], ',', train_eval[0], ',', train_eval[2])
203  print('Accuracy, loss and AUC scored compared to validation data is:',
      val_eval[1],',', val_eval[0],',', val_eval[2])
204  print('Accuracy, loss and AUC scored compared to test data is:',
      test_eval[1],',', test_eval[0],',', test_eval[2])
205
206  # Code block to be used later for prediction percentages.
207  pos_num = 0
208  neg_num = 0
209
210  # We then create a test database, which we'll use to cross-reference the
       two classes of the model
211  # with the test data we segmented earlier.
212  df_test = pd.DataFrame(data=os.listdir(dl_path + 'Segmented Images/test/
      '), columns = ['image'])
213  df_test = df_test.merge(df, how='inner')
214
215  for i in range (0,len(os.listdir(dl_path + 'Segmented Images/test/'))):
216    if str(df_test['status'][i]) == 'tumor':
217      pos_num = pos_num + 1
218    else:
```

```
219       neg_num = neg_num + 1
220
221   # Defininging function that will evaluate single-batch predictions per
       class.
222   # class_name is the string value of the class, as it is described in .
       csv file.
223   # class_num is the number of images matching with each class.
224   def predictions(class_name, class_num, class_id):
225     class_count = 0
226     for i in range (0,len(os.listdir(dl_path + 'Segmented Images/test/')))
      :
227       if str(df_test.status[i]) == class_name:
228       img_path = dl_path + 'Segmented Images/test/' + str(df_test.image[i
      ])
229       img = keras.preprocessing.image.load_img(img_path, target_size=(256,
       256))
230       img_array = keras.preprocessing.image.img_to_array(img)
231       img_array = np.expand_dims(img_array, axis=0)
232       img_array /= 255.
233
234       prediction = model.predict(img_array, verbose = 0)
235       predicted_class = np.round(prediction)
236       if predicted_class == class_id:
237         class_count = class_count + 1
238   print('Number of correct predictions for class ' + class_name + ' to
       total is: ', class_count/class_num)
239
240   predictions('normal', neg_num, 0)
241   predictions('tumor', pos_num, 1)
```

Παράρτημα Β

Skin Cancer Categorical Classification Model

```
#Module Initialization
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import random
import shutil
import kaggle
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import (Conv2D, Dense, Flatten, Dropout,
 Activation, MaxPooling2D,
RandomFlip, RandomZoom, RandomRotation)
from tensorflow.keras.models import Sequential
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint,
 ReduceLROnPlateau

dl_path = 'path/to/folder'

if not os.path.exists(dl_path):
  kaggle.api.authenticate()
  kaggle.api.dataset_download_files('farjanakabirsamanta/skin-cancer-
 dataset', path = dl_path, unzip=True)
  print('Files successfully downloaded!')
else:
  print('Data has already been downloaded.')

#Split Images in random train/test/validation sets
image_path = dl_path +'Skin Cancer/Skin Cancer/'

if not os.path.exists(image_path + 'train'):
  #Pathing the folders necessary
  train_folder = os.path.join(image_path, 'train')
  val_folder = os.path.join(image_path, 'valid')
  test_folder = os.path.join(image_path, 'test')

```

```
    #Defining the file format of the images
    file_formats = ['.jpg', ]

    #List of image filenames
    imgs_list = [filename for filename in os.listdir(image_path) if os.
   path.splitext(filename)[-1] in file_formats]

    #Randomize the three sets using a seed and the image list generated
   above
    random.seed(42)
    random.shuffle(imgs_list)

    #Split data in ratios: 70% train, 15% validation, 15% test
    train_size = int(len(imgs_list) * 0.70)
    val_size = int(len(imgs_list) * 0.15)
    test_size = int(len(imgs_list) * 0.15)

    # Create destination folders if they don't exist
    for folder_path in [train_folder, val_folder, test_folder]:
      if not os.path.exists(folder_path):
      os.makedirs(folder_path)

    # Copy image files to destination folders
    for i, f in enumerate(imgs_list):
      if i < train_size:
        dest_folder = train_folder
      elif i < train_size + val_size:
        dest_folder = val_folder
      else:
        dest_folder = test_folder
      shutil.move(os.path.join(image_path, f), os.path.join(dest_folder, f
    ))
  else:
    print('Files already segmented!')

  #Load Skin Cancer Database CSV file
  predf = pd.read_csv(dl_path + 'HAM10000_metadata.csv')

  #Add filename suffix to entries in the "image_id" column
  df = predf.copy()
  df['image_id'] = df['image_id'] + '.jpg'

  #Assign classes to the imagesets by cross-referencing file names with
    the "dx" column from the CSV file
  #and prepare images for overall processing from the neural network.
  datagen = ImageDataGenerator(rescale=1./255)
  train_data = datagen.flow_from_dataframe(dataframe = df, directory =
    image_path + 'train', x_col = "image_id", y_col = "dx", class_mode = "
    categorical", target_size = (256, 256), batch_size = 32)
  valid_data = datagen.flow_from_dataframe(dataframe = df, directory =
    image_path + 'valid', x_col = "image_id", y_col = "dx", class_mode = "
    categorical", target_size = (256, 256), batch_size = 32)
  test_data = datagen.flow_from_dataframe(dataframe = df, directory =
    image_path + 'test', x_col = "image_id", y_col = "dx", class_mode = "
```

```
    categorical", target_size = (256, 256), batch_size = 32)

#We will proceed to evaluate data ratios in our imageset.
counts = df.dx.value_counts().sort_index(ascending=True)
print(counts)

# Code that ensures we correctly loaded the images to be used for
  training.
# This is used for visualization purposes.
def plots(ims, figsize=(12,6), rows=3, interp=False, titles=None):
  if type(ims[0]) is np.ndarray:
    ims = np.array(ims)
    if (ims.shape[-1] != 3):
      ims = ims.transpose((0,2,3,1))
  f = plt.figure(figsize=figsize)
  cols = len(ims)//rows if len(ims) % 2 == 0 else len(ims)//rows + 1
  for i in range(len(ims)):
    try:
      sp = f.add_subplot(rows, cols, i+1)
      sp.axis('Off')
    except ValueError:
      break
    if titles is not None:
      sp.set_title(titles[i], fontsize=16)
    plt.imshow(ims[i], interpolation=None if interp else 'none')


imgs, labels = next(train_data)
labels_new = []
for i in range(0,len(labels)):
  if np.allclose(labels[i],np.array([1, 0, 0, 0, 0, 0, 0])) == True:
    labels_new.append('akiec')
  elif np.allclose(labels[i],np.array([0, 1, 0, 0, 0, 0, 0])) == True:
    labels_new.append('bcc')
  elif np.allclose(labels[i],np.array([0, 0, 1, 0, 0, 0, 0])) == True:
    labels_new.append('bkl')
  elif np.allclose(labels[i],np.array([0, 0, 0, 1, 0, 0, 0])) == True:
    labels_new.append('df')
  elif np.allclose(labels[i],np.array([0, 0, 0, 0, 1, 0, 0])) == True:
    labels_new.append('mel')
  elif np.allclose(labels[i],np.array([0, 0, 0, 0, 0, 1, 0])) == True:
    labels_new.append('nv')
  elif np.allclose(labels[i],np.array([0, 0, 0, 0, 0, 0, 1])) == True:
    labels_new.append('vasc')
plots(imgs, titles = labels_new)

# Defining model architecture
model = Sequential()

#Pre-processing Layer
model.add(RandomFlip("horizontal_and_vertical", input_shape = (256, 256,
  3)))
model.add(RandomRotation(0.2))
model.add(RandomZoom(0.15))
```

```
132
133 #Layer 1
134 model.add(Conv2D(16, (3,3), strides=(1,1), padding = "same"))
135 model.add(Activation("relu"))
136 model.add(MaxPooling2D(pool_size = (2,2)))
137
138 model.add(Conv2D(32, (3,3), strides=(1,1), padding = "same"))
139 model.add(Activation("relu"))
140 model.add(MaxPooling2D(pool_size = (2,2)))
141
142 model.add(Conv2D(64, (3,3), strides=(1,1), padding = "same"))
143 model.add(Activation("relu"))
144 model.add(MaxPooling2D(pool_size = (2,2)))
145
146 model.add(Conv2D(128, (3,3), strides=(1,1), padding = "same"))
147 model.add(Activation("relu"))
148 model.add(MaxPooling2D(pool_size = (2,2)))
149
150 model.add(Flatten())
151 model.add(Dense(128))
152 model.add(Activation("relu"))
153 model.add(Dropout(0.15))
154
155 #Layer 5
156 model.add(Dense(7))
157 model.add(Activation("softmax"))
158
159 #Compiling model
160 model.compile(loss = "categorical_crossentropy", optimizer = "adam",
     metrics = ['accuracy', tf.keras.metrics.AUC(multi_label=True)])
161
162 #Monitoring validation loss and establishing checkpoints for the CNN to
     be saved at
163 early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
     patience = 5, restore_best_weights = True)
164 checkpoint = ModelCheckpoint("Skin Cancer Optimization 6.h5", monitor='
     val_accuracy', verbose = 1, save_best_only = True, mode='max')
165 lr_monitor = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience
     =3, min_lr=0.00001)
166
167 #Fitting model and saving for later use
168 history = model.fit(train_data, validation_data = valid_data, epochs =
     35, verbose = 1, callbacks = [early_stopping, checkpoint, lr_monitor])
169 #history = model.fit(train_data, validation_data = valid_data,
     class_weight=weights, epochs = 30, verbose = 1, callbacks = [
     early_stopping, checkpoint])
170 model.save('SC Final2.keras')
171 #Plot model accuracy for training and validation datasets
172 plt.plot(history.history['accuracy'], label = 'accuracy')
173 plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
174 plt.title('Correlation of Training vs Validation Data')
175 plt.xlabel('Epochs')
176 plt.ylabel('Accuracy')
177 plt.legend(['Training Data','Validation Data'], loc='lower right')
```

```
178
179 # Generate model summary
180 model.summary()
181
182 #Generate evaluation metrics for all 3 datasets
183 train_eval = model.evaluate(train_data, verbose = 0)
184 val_eval = model.evaluate(valid_data, verbose = 0)
185 test_eval = model.evaluate(test_data, verbose = 0)
186 print('Accuracy, loss and AUC scored compared to training data is:',
     train_eval[1], ',', train_eval[0], ',', train_eval[2])
187 print('Accuracy, loss and AUC scored compared to validation data is:',
     val_eval[1],',', val_eval[0],',', val_eval[2])
188 print('Accuracy, loss and AUC scored compared to test data is:',
     test_eval[1],',', test_eval[0],',', test_eval[2])
189
190 # Code block to be used later for prediction percentages.
191 df_test = pd.DataFrame(data=os.listdir(image_path + 'test/' ), columns =
      ['image_id'])
192 df_test = df_test.merge(df, how='inner')
193
194 class_names = list(train_data.class_indices.keys())
195 class_ids = list(train_data.class_indices.values())
196 class_counts= df_test.dx.value_counts().sort_index(ascending=True)
197
198 # Defininingfunction that will evaluate single-batch predictions per
     class.
199 # class_name is the string value of the class, as it is described in .
     csv file.
200 # class_num is the number of images matching with each class.
201 # class_id is the integer assigned to the class by Tensorflow.
202 def predictions(class_name, class_num, class_id):
203   class_count = 0
204   for i in range (0,len(os.listdir(image_path + 'test/'))):
205     if str(df_test.dx[i]) == class_name:
206       img_path = image_path + 'test/' + str(df_test.image_id[i])
207       img = keras.preprocessing.image.load_img(img_path, target_size
     =(256, 256))
208       img_array = keras.preprocessing.image.img_to_array(img)
209       img_array = np.expand_dims(img_array, axis=0)
210       img_array /= 255.
211
212       prediction = model.predict(img_array, verbose = 0)
213       predicted_class = np.argmax(prediction)
214       if predicted_class == class_id:
215         class_count = class_count + 1
216 print('Ratio of correct predictions for class ' + class_name + ' to
     total is: ', class_count/class_num)
217
218 for i in range(0,len(class_names)):
219   predictions(class_names[i], class_counts[i], class_ids[i])
```

Παράρτημα Γ

COVID-19 Binary Classification Model

```
#Module Initialization
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import random
import shutil
import kaggle
import time
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Conv2D, Dense, Flatten, Dropout,
 Activation, MaxPooling2D, RandomFlip, RandomRotation
from tensorflow.keras.models import Sequential
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint,
 ReduceLROnPlateau

dl_path = 'path/to/folder'

if not os.path.exists(dl_path):
  kaggle.api.authenticate()
  kaggle.api.dataset_download_files('andyczhao/covidx-cxr2', path =
 dl_path, unzip=True)
  print('Files successfully downloaded!')
else:
  print('Data has already been downloaded.')

# The dataset is segmented in a way that makes training overfit. We will
  attempt to instead shuffle the
# files downloaded from Kaggle manually, in order to try and increase
 model performance.
train_folder = dl_path + 'train/'
validation_folder = dl_path + 'val/'
test_folder = dl_path + 'test/'
image_folder = dl_path + 'Images/'
final_folder = (dl_path + 'Segmented Images/')
```

```
source_paths = [train_folder, validation_folder, test_folder]

# The following code ensures everything is moved to the right places.
if not os.path.exists(final_folder):
  os.makedirs(image_folder)
  for i in range (0,3):
    files = os.listdir(source_paths[i])
    for file in files:
      file_name = os.path.join(source_paths[i], file)
      shutil.move(file_name, final_folder)
  print("Files Moved")
  for i in range (0,3):
    shutil.rmtree(source_paths[i])
else:
  print("Files already collected.")

#Convert .txt files in CSV format
if not os.path.isfile(dl_path + 'data.csv'):
  train_file = pd.read_csv(dl_path + 'train.txt', sep = " ", header=None
 )
  train_file.columns = ['Patient_ID', 'Image_ID', 'Status', 'Source']
  train_file.to_csv(dl_path + 'train.csv', index=None)
  os.remove(dl_path + 'train.txt')

  val_file = pd.read_csv(dl_path + 'val.txt', sep = " ", header=None)
  val_file.columns = ['Patient_ID', 'Image_ID', 'Status', 'Source']
  val_file.to_csv(dl_path + 'val.csv', index=None)
  os.remove(dl_path + 'val.txt')

  test_file = pd.read_csv(dl_path + 'test.txt', sep = " ", header=None)
  test_file.columns = ['Patient_ID', 'Image_ID', 'Status', 'Source']
  test_file.to_csv(dl_path + 'test.csv', index=None)
  os.remove(dl_path + 'test.txt')

  df = pd.concat([train_file, val_file, test_file])
  df.to_csv(dl_path + 'data.csv')
  df = pd.read_csv(dl_path + 'data.csv')
  os.remove(dl_path + 'train.csv')
  os.remove(dl_path + 'val.csv')
  os.remove(dl_path + 'test.csv')
  print('CSV successfully created!')

else:
  df = pd.read_csv(dl_path + 'data.csv')
  print('CSV already exists.')

# The following code splits the images to training, validation and test
 folders.
# This is important, in order to ensure a good ratio of each data
 catagory.
final_folder = (dl_path + 'Segmented Images/')

if not os.path.exists(dl_path + 'Segmented Images/test/'):
  #Pathing the folders necessary
```

```
    train_folder = os.path.join(final_folder, 'train')
    val_folder = os.path.join(final_folder, 'val')
    test_folder = os.path.join(final_folder, 'test')

    #Defining the file format of the images
    file_formats = ['.jpg', '.jpeg', '.png', '.PNG', '.JPG']

    #List of image filenames
    imgs_list = [filename for filename in os.listdir(image_folder) if os.
   path.splitext(filename)[-1] in file_formats]

    #Randomize the three sets using a seed and the image list generated
   above
    random.seed(42)
    random.shuffle(imgs_list)

    #Split data in ratios: 70% train, 15% validation, 15% test
    train_size = int(len(imgs_list) * 0.70)
    val_size = int(len(imgs_list) * 0.15)
    test_size = int(len(imgs_list) * 0.15)

# Create destination folders if they don't exist
for folder_path in [train_folder, val_folder, test_folder]:
  if not os.path.exists(folder_path):
    os.makedirs(folder_path)

  # Copy image files to destination folders
  for i, f in enumerate(imgs_list):
    if i < train_size:
      dest_folder = train_folder
    elif i < train_size + val_size:
      dest_folder = val_folder
    else:
      dest_folder = test_folder
    shutil.move(os.path.join(image_folder, f), os.path.join(dest_folder,
   f))
  shutil.rmtree(image_folder)
  print('Files successfully segmented!')
else:
  print('Files already segmented!')

# Assign classes to the imagesets by cross-referencing file names with
 the "Status" column from the CSV file
# and prepare images for overall processing from the neural network.

datagen = ImageDataGenerator(rescale=1./255)
train_data = datagen.flow_from_dataframe(dataframe = df, directory =
 final_folder + 'train/', x_col = "Image_ID", y_col = "Status",
 class_mode = "binary", target_size = (256, 256), batch_size = 32)
valid_data = datagen.flow_from_dataframe(dataframe = df, directory =
 final_folder + 'val/', x_col = "Image_ID", y_col = "Status", class_mode
  = "binary",  target_size = (256, 256), batch_size = 32)
test_data = datagen.flow_from_dataframe(dataframe = df, directory =
 final_folder + 'test/', x_col = "Image_ID", y_col = "Status",
```

```
    class_mode = "binary", target_size = (256, 256), batch_size = 32)

# Code that ensures we correctly loaded the images to be used for training.
# This is used for visualization purposes.
def plots(ims, figsize=(12,6), rows=3, interp=False, titles=None):
    if type(ims[0]) is np.ndarray:
        ims = np.array(ims)
        if (ims.shape[-1] != 3):
        ims = ims.transpose((0,2,3,1))
    f = plt.figure(figsize=figsize)
    cols = len(ims)//rows if len(ims) % 2 == 0 else len(ims)//rows + 1
    for i in range(len(ims)):
        try:
            sp = f.add_subplot(rows, cols, i+1)
            sp.axis('Off')
        except ValueError:
            break
    if titles is not None:
        sp.set_title(titles[i], fontsize=16)
    plt.imshow(ims[i], interpolation=None if interp else 'none')


imgs, labels = next(train_data)
labels_new = []
for i in range(0,len(labels)):
    if np.allclose(labels[i],np.array([1])) == True:
        labels_new.append('COVID')
    elif np.allclose(labels[i],np.array([0])) == True:
        labels_new.append('Healthy')
plots(imgs, titles = labels_new)

# Defining model architecture
model = Sequential()

# Pre-processing Layer
model.add(RandomFlip("horizontal_and_vertical", input_shape = (256, 256, 3)))
model.add(RandomRotation(0.1))


# Convolution Layer 1
model.add(Conv2D(32, (3,3), strides=(1,1), padding = "same"))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size = (2,2)))

# Convolution Layer 2
model.add(Conv2D(64, (3,3), strides=(1,1), padding = "same"))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size = (2,2)))

# Convolution Layer 3
model.add(Conv2D(128, (3,3), strides=(1,1), padding = "same"))
model.add(Activation("relu"))
```

```
181 model.add(MaxPooling2D(pool_size = (2,2)))
182
183 # Fully Connected Layer
184 model.add(Flatten())
185 model.add(Dense(128))
186 model.add(Activation("relu"))
187 model.add(Dropout(0.1))
188
189 # Output
190 model.add(Dense(1))
191 model.add(Activation("sigmoid"))
192
193 # Compiling model
194 model.compile(loss = "binary_crossentropy", optimizer = "adam", metrics
     = ['accuracy', 'AUC'])
195
196 # We use a simple time module to record how much time the network took
     to train.
197 # This can be safely removed, unless the user is curious for learning/
     diagnostic purposes.
198 # If training on a CPU, the process will take significantly longer.
199 start_time = time.time()
200
201 # We use various callbacks to monitor the model's training and ensure
     minimum time losses in case
202 # the model starts to overfit/the process is lengthy enough to make
     manual monitoring impossible.
203 # It is generally recommended to use all 3 callbacks below to ensure
     best model performance.
204 #
205 # We use Early Stopping to monitor a specific metric and terminate
     training if there's no improvement
206 # after a set number of epochs. Checkpoint saves the model when a
     monitored value is improved.
207 # Finally, we can automatically tweak the learning rate of the model if
     our preferred monitored values
208 # shows no improvement after a certain number of epochs.
209 early_stopping = EarlyStopping(monitor='val_loss', patience = 6,
     restore_best_weights = True)
210 checkpoint = ModelCheckpoint("COVID CRX-4 Checkpoint.h5", monitor='
     val_accuracy', verbose = 1, save_best_only = True, mode='max')
211 lr_monitor = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience
     =3, min_lr=0.00001)
212
213 # Initiating training.
214 history = model.fit(train_data, validation_data = valid_data, epochs =
     30, verbose = 1, callbacks = [early_stopping, checkpoint, lr_monitor])
215
216 # Tensorflow recommends saving models in .keras format. To comply with
     that, we save the best model generated
217 # by the callback functions used above.
218 model.save('COVID CRX-4.keras')
219 print(time.time() - start_time, 'sec')
220
```

```
# Plot model accuracy for training and validation datasets
plt.plot(history.history['accuracy'], label = 'accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.title('Correlation of Training vs Validation Data')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(['Training Data','Validation Data'], loc='lower right')

# Generate model summary
model.summary()

# Generate evaluation metrics for all 3 datasets
train_eval = model.evaluate(train_data, verbose = 0)
val_eval = model.evaluate(valid_data, verbose = 0)
test_eval = model.evaluate(test_data, verbose = 0)
print('Accuracy, loss and AUC scored compared to training data is:',
 train_eval[1], ',', train_eval[0], ',', train_eval[2])
print('Accuracy, loss and AUC scored compared to validation data is:',
 val_eval[1],',', val_eval[0],',', val_eval[2])
print('Accuracy, loss and AUC scored compared to test data is:',
 test_eval[1],',', test_eval[0],',', test_eval[2])

# Code block to be used later for prediction percentages.
pos_num = 0
neg_num = 0
df_test = pd.DataFrame(data=os.listdir(dl_path + 'Segmented Images/test/
 '), columns = ['Image_ID'])
df_test = df_test.merge(df, how='inner')

for i in range (0,len(os.listdir(dl_path + 'Segmented Images/test/'))):
  if str(df_test.Status[i]) == 'positive':
    pos_num = pos_num + 1
  else:
    neg_num = neg_num + 1

# Defininging function that will evaluate single-batch predictions per
 class.
# class_name is the string value of the class, as it is described in .
 csv file.
# class_num is the number of images matching with each class.
# class_id is the integer assigned to the class by Tensorflow.
def predictions(class_name, class_num, class_id):
  class_count = 0
  for i in range (0,len(os.listdir(dl_path + 'Segmented Images/test/')))
 :
    if str(df_test.Status[i]) == class_name:
      img_path = dl_path + 'Segmented Images/test/' + str(df_test.
 Image_ID[i])
      img = keras.preprocessing.image.load_img(img_path, target_size
 =(256, 256))
      img_array = keras.preprocessing.image.img_to_array(img)
      img_array = np.expand_dims(img_array, axis=0)
      img_array /= 255.

```

```
266         prediction = model.predict(img_array, verbose = 0)
267         predicted_class = np.round(prediction)
268         if predicted_class == class_id:
269           class_count = class_count + 1
270     print('Number of correct predictions for class ' + class_name + ' to
      total is: ', class_count/class_num)
271
272   predictions('negative',neg_num, 0)
273   predictions('positive',pos_num, 1)
```

Παράρτημα Δ

Pneumonia Binary Classification Model

```
#Module Initialization
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import kaggle
import time
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Conv2D, Dense, Flatten, Dropout,
 Activation, MaxPooling2D, RandomFlip, RandomZoom, RandomRotation
from tensorflow.keras.models import Sequential
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint,
 ReduceLROnPlateau

dl_path = 'path/to/folder'

if not os.path.exists(dl_path):
  kaggle.api.authenticate()
  kaggle.api.dataset_download_files('tolgadincer/labeled-chest-xray-
 images', path = dl_path, unzip=True)
  print('Files successfully downloaded!')
else:
  print('Data has already been downloaded.')

# Defining image directory.
image_folder = dl_path + 'chest_xray/'

# Defining training and test image directories.
train_dir = image_folder + 'train'
test_dir = image_folder + 'test'

# We will proceed to evaluate data ratios in our imageset.
num_pneumonia = len(os.listdir(os.path.join(train_dir, 'PNEUMONIA')))
num_normal = len(os.listdir(os.path.join(train_dir, 'NORMAL')))
print(f"PNEUMONIA={num_pneumonia}")
print(f"NORMAL={num_normal}")
```

```
36
37  #Defining Data Generators for train and validation data.
38
39  datagen_train = ImageDataGenerator(rescale=1./255, validation_split =
     0.3)
40  datagen_test = ImageDataGenerator(rescale=1./255)
41
42  #Creating database files
43  train_data = datagen_train.flow_from_directory(train_dir, batch_size =
     32, shuffle = True, class_mode = 'binary',
44  target_size = (256,256), seed = 42, subset = "training")
45  valid_data = datagen_train.flow_from_directory(train_dir, batch_size =
     32, shuffle = True, class_mode = 'binary',
46  target_size = (256,256), seed = 42, subset = "validation")
47  test_data = datagen_test.flow_from_directory(test_dir, class_mode = '
     binary',
48  target_size = (256,256), batch_size = 32)
49  print(train_data.class_indices)
50
51  # Code that ensures we correctly loaded the images to be used for
     training.
52  # This is used for visualization purposes.
53  def plots(ims, figsize=(12,6), rows=3, interp=False, titles=None):
54    if type(ims[0]) is np.ndarray:
55      ims = np.array(ims)
56      if (ims.shape[-1] != 3):
57      ims = ims.transpose((0,2,3,1))
58    f = plt.figure(figsize=figsize)
59    cols = len(ims)//rows if len(ims) % 2 == 0 else len(ims)//rows + 1
60    for i in range(len(ims)):
61      try:
62        sp = f.add_subplot(rows, cols, i+1)
63        sp.axis('Off')
64      except ValueError:
65        break
66      if titles is not None:
67        sp.set_title(titles[i], fontsize=16)
68      plt.imshow(ims[i], interpolation=None if interp else 'none')
69
70
71  imgs, labels = next(train_data)
72  labels_new = []
73  for i in range(0,len(labels)):
74    if np.allclose(labels[i],np.array([1])) == True:
75      labels_new.append('Pneum.')
76    elif np.allclose(labels[i],np.array([0])) == True:
77      labels_new.append('Healthy')
78  plots(imgs, titles = labels_new)
79
80  # Defining model architecture
81  model = Sequential()
82
83  #Pre-processing Layer
84  model.add(RandomFlip("horizontal_and_vertical", input_shape = (256, 256,
```

```
    3)))
model.add(RandomRotation(0.5))
model.add(RandomZoom(0.2))

#Layer 1
model.add(Conv2D(16, (3,3), strides=(1,1), padding = "same"))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size = (2,2)))

model.add(Conv2D(32, (3,3), strides=(1,1), padding = "same"))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size = (2,2)))

model.add(Conv2D(64, (3,3), strides=(1,1), padding = "same"))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size = (2,2)))

model.add(Conv2D(128, (3,3), strides=(1,1), padding = "same"))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size = (2,2)))

model.add(Flatten())
model.add(Dense(128))
model.add(Activation("relu"))
model.add(Dropout(0.2))

#Layer 5
model.add(Dense(1))
model.add(Activation("sigmoid"))

#Compiling model
model.compile(loss = "binary_crossentropy", optimizer = "adam", metrics
 = ['accuracy', 'AUC'])

# We use a simple time module to record how much time the network took
 to train.
# This can be safely removed, unless the user is curious for learning/
 diagnostic purposes.
# If training on a CPU, the process will take significantly longer.
start_time = time.time()

# We use various callbacks to monitor the model's training and ensure
 minimum time losses in case
# the model starts to overfit/the process is lengthy enough to make
 manual monitoring impossible.
# It is generally recommended to use all 3 callbacks below to ensure
 best model performance.
#
# We use Early Stopping to monitor a specific metric and terminate
 training if there's no improvement
# after a set number of epochs. Checkpoint saves the model when a
 monitored value is improved.
# Finally, we can automatically tweak the learning rate of the model if
 our preferred monitored values
```

```
# shows no improvement after a certain number of epochs.
early_stopping = EarlyStopping(monitor='val_loss', patience = 6, restore_best_weights = True)
checkpoint = ModelCheckpoint("Pneumonia X-Ray Checkpoint.h5", monitor='val_accuracy', verbose = 1, save_best_only = True, mode='max')
lr_monitor = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=3, min_lr=0.00001)

# Initiating training.
history = model.fit(train_data, validation_data = valid_data, epochs = 30, verbose = 1, callbacks = [early_stopping, checkpoint, lr_monitor])

# Tensorflow recommends saving models in .keras format. To comply with that, we save the best model generated
# by the callback functions used above.
model.save('Pneumonia X-Ray Classification.keras')
print(time.time() - start_time, 'sec')

# Plot model accuracy for training and validation datasets
plt.plot(history.history['accuracy'], label = 'accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.title('Correlation of Training vs Validation Data')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(['Training Data','Validation Data'], loc='lower right')

# Generate model summary
model.summary()

# Generate evaluation metrics for all 3 datasets
train_eval = model.evaluate(train_data, verbose = 0)
val_eval = model.evaluate(valid_data, verbose = 0)
test_eval = model.evaluate(test_data, verbose = 0)
print('Accuracy, loss and AUC scored compared to training data is:', train_eval[1], ',', train_eval[0], ',', train_eval[2])
print('Accuracy, loss and AUC scored compared to validation data is:', val_eval[1],',', val_eval[0],',', val_eval[2])
print('Accuracy, loss and AUC scored compared to test data is:', test_eval[1],',', test_eval[0],',', test_eval[2])

# Defining number of samples per class in test data.

pneum_num = len(os.listdir(image_folder + 'test/PNEUMONIA'))
norm_num = len(os.listdir(image_folder + 'test/NORMAL'))

# The following function is called in order to calculate the ratio of correct predictions per class sample.
# test_num = The variables instantiated above.
# class_name = The string value of the class we want to predict.
# class_num = The numerical value of the predicted class, as defined in class_names.

def predictions(class_name, class_num, class_id):
class_count = 0
```

```
    for i in range (0, class_num):
# Load image and normalize it.
      img_path = image_folder + 'test/' + class_name +'/' + str(os.listdir
  (image_folder + 'test/' + class_name)[i])
      img = keras.preprocessing.image.load_img(img_path, target_size=(256,
   256))
      img_array = keras.preprocessing.image.img_to_array(img)
      img_array = np.expand_dims(img_array, axis=0)
      img_array /= 255.

# Generate prediction. If the model correctly identifies the sample,
  increase
# the counter.
      prediction = model.predict(img_array, verbose = 0)
      predicted_class = np.round(prediction)
      if predicted_class == class_id:
        class_count = class_count + 1
    print('Ratio of correct predictions for class ' + class_name + ' to
  total is: ', class_count/class_num)

predictions('NORMAL',norm_num,0)
predictions('PNEUMONIA',pneum_num,1)
```

Παράρτημα Ε

Alzheimer's Categorical Classification Model

```
#Module Initialization
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import kaggle
import time
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Conv2D, Dense, Flatten, Dropout,
 Activation, MaxPooling2D, RandomFlip, RandomRotation,
 BatchNormalization
from tensorflow.keras.models import Sequential
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint,
 ReduceLROnPlateau

dl_path = '/path/to/folder/'

if not os.path.exists(dl_path):
  kaggle.api.authenticate()
  kaggle.api.dataset_download_files('tourist55/alzheimers-dataset-4-
 class-of-images', path = dl_path, unzip=True)
  print('Files successfully downloaded!')
else:
  print('Data has already been downloaded.')

# Defining image directory.
image_folder = dl_path + 'Alzheimer_s Dataset/'

# Defining training and test image directories.
train_dir = image_folder + 'train'
test_dir = image_folder + 'test'

#Defining Data Generators for train and validation data.

datagen_train = ImageDataGenerator(rescale=1./255, validation_split =
 0.2)
```

```
datagen_test = ImageDataGenerator(rescale=1./255)

#Creating database files
train_data = datagen_train.flow_from_directory(train_dir, batch_size =
 32, shuffle = True, class_mode = 'categorical', target_size = (176,208)
 , seed = 69, subset = "training")
valid_data = datagen_train.flow_from_directory(train_dir, batch_size =
 32, shuffle = True, class_mode = 'categorical', target_size = (176,208)
 , seed = 69, subset = "validation")
test_data = datagen_test.flow_from_directory(test_dir, class_mode = '
 categorical', target_size = (176,208), batch_size = 32)

# We will proceed to evaluate data ratios in our imageset.
num_mild = len(os.listdir(os.path.join(train_dir, 'MildDemented')))
num_moderate = len(os.listdir(os.path.join(train_dir, 'ModerateDemented'
 )))
num_non = len(os.listdir(os.path.join(train_dir, 'NonDemented')))
num_vmild = len(os.listdir(os.path.join(train_dir, 'VeryMildDemented')))
print(f"Patients with Very Mild Dementia = {num_vmild}")
print(f"Patients with Mild Dementia = {num_mild}")
print(f"Patients with Moderate Dementia = {num_moderate}")
print(f"Patients without Dementia = {num_non}")

# Code that ensures we correctly loaded the images to be used for
 training.
# This is used for visualization purposes.
def plots(ims, figsize=(12,6), rows=3, interp=False, titles=None):
  if type(ims[0]) is np.ndarray:
    ims = np.array(ims)
    if (ims.shape[-1] != 3):
      ims = ims.transpose((0,2,3,1))
  f = plt.figure(figsize=figsize)
  cols = len(ims)//rows if len(ims) % 2 == 0 else len(ims)//rows + 1
  for i in range(len(ims)):
    try:
      sp = f.add_subplot(rows, cols, i+1)
      sp.axis('Off')
    except ValueError:
      break
    if titles is not None:
      sp.set_title(titles[i], fontsize=16)
    plt.imshow(ims[i], interpolation=None if interp else 'none')


imgs, labels = next(train_data)
labels_new = []
for i in range(0,len(labels)):
  if np.allclose(labels[i],np.array([1, 0, 0, 0])) == True:
    labels_new.append('Mild')
  elif np.allclose(labels[i],np.array([0, 1, 0, 0])) == True:
    labels_new.append('Mod')
  elif np.allclose(labels[i],np.array([0, 0, 1, 0])) == True:
    labels_new.append('Non')
  elif np.allclose(labels[i],np.array([0, 0, 0, 1])) == True:
```

```
81      labels_new.append('V.Mild')
82  plots(imgs, titles = labels_new)
83
84  # Defining model architecture
85  model = Sequential()
86
87  #Pre-processing Layer
88  model.add(RandomFlip("horizontal_and_vertical", input_shape = (176, 208,
      3)))
89  model.add(RandomRotation(0.2))
90
91  #Layer 1
92
93  model.add(Conv2D(32, (3,3), strides=(1,1), padding = "same"))
94  model.add(Activation("relu"))
95  model.add(MaxPooling2D(pool_size = (2,2)))
96
97  model.add(Conv2D(64, (3,3), strides=(1,1), padding = "same"))
98  model.add(Activation("relu"))
99  model.add(MaxPooling2D(pool_size = (2,2)))
100
101 model.add(Conv2D(128, (3,3), strides=(1,1), padding = "same"))
102 model.add(Activation("relu"))
103 model.add(MaxPooling2D(pool_size = (2,2)))
104 model.add(BatchNormalization())
105
106 model.add(Flatten())
107 model.add(Dense(128))
108 model.add(Activation("relu"))
109 model.add(Dropout(0.2))
110
111 model.add(Dense(64))
112 model.add(Activation("relu"))
113 model.add(Dropout(0.2))
114
115 #Layer 5
116 model.add(Dense(4))
117 model.add(Activation("softmax"))
118
119 #Compiling model
120 model.compile(loss = "categorical_crossentropy", optimizer = "adam",
     metrics = ['accuracy', 'AUC'])
121
122 # We use a simple time module to record how much time the network took
     to train.
123 # This can be safely removed, unless the user is curious for learning/
     diagnostic purposes.
124 # If training on a CPU, the process will take significantly longer.
125 start_time = time.time()
126
127 # We use various callbacks to monitor the model's training and ensure
     minimum time losses in case
128 # the model starts to overfit/the process is lengthy enough to make
     manual monitoring impossible.
```

```
# It is generally recommended to use all 3 callbacks below to ensure best model performance.
#
# We use Early Stopping to monitor a specific metric and terminate training if there's no improvement
# after a set number of epochs. Checkpoint saves the model when a monitored value is improved.
# Finally, we can automatically tweak the learning rate of the model if our preferred monitored values
# shows no improvement after a certain number of epochs.
early_stopping = EarlyStopping(monitor='val_loss', patience = 15, restore_best_weights = True)
checkpoint = ModelCheckpoint("Alzheimer's Checkpoint.h5", monitor='val_auc', verbose = 1, save_best_only = True, mode='max')
lr_monitor = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=13, min_lr=0.00001)

# Initiating training.
history = model.fit(train_data, validation_data = valid_data, epochs = 100, verbose = 1, callbacks = [early_stopping, checkpoint, lr_monitor])

# Tensorflow recommends saving models in .keras format. To comply with that, we save the best model generated
# by the callback functions used above.
model.save('Alzheimer Disease Categorical Classification Model.keras')
print(time.time() - start_time, 'sec')

# Plot model accuracy for training and validation datasets
plt.plot(history.history['auc'], label = 'accuracy')
plt.plot(history.history['val_auc'], label = 'val_accuracy')
plt.title('Correlation of Training vs Validation Data')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(['Training Data','Validation Data'], loc='lower right')

# Generate model summary
model.summary()

#Generate evaluation metrics for all 3 datasets
train_eval = model.evaluate(train_data, verbose = 0)
val_eval = model.evaluate(valid_data, verbose = 0)
test_eval = model.evaluate(test_data, verbose = 0)
print('Accuracy, loss and AUC scored compared to training data is:', train_eval[1], ',', train_eval[0], ',', train_eval[2])
print('Accuracy, loss and AUC scored compared to validation data is:', val_eval[1],',', val_eval[0],',', val_eval[2])
print('Accuracy, loss and AUC scored compared to test data is:', test_eval[1],',', test_eval[0],',', test_eval[2])

# First, we print the class names so that we know how to evaluate each prediction.
# Afterwards, we create a list that will contain the string values of each class.
class_names = list(train_data.class_indices.keys())
```

```
class_ids = list(train_data.class_indices.values())

# We next define the list of image files in each directory. We will use that to create a custom generator that will
# validate each picture separately using the model generated.

mild_test = len(os.listdir(image_folder + 'test/MildDemented'))
mod_test = len(os.listdir(image_folder + 'test/ModerateDemented'))
non_test = len(os.listdir(image_folder + 'test/NonDemented'))
vmild_test = len(os.listdir(image_folder + 'test/VeryMildDemented'))

class_counts = [mild_test, mod_test, non_test, vmild_test]

# The following function is called in order to calculate the ratio of correct predictions per class sample.
# test_num = The variables instantiated above.
# class_name = The string value of the class we want to predict.
# class_num = The numerical value of the predicted class, as defined in class_names.

def prediction_ratio(class_name, class_num, class_id):
class_count = 0
  for i in range (0, class_num):
  # Load image and normalize it.
    img_path = image_folder + 'test/' + class_name +'/' + str(os.listdir(image_folder + 'test/' + class_name)[i])
    img = keras.preprocessing.image.load_img(img_path, target_size=(176, 208))
    img_array = keras.preprocessing.image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array /= 255.

  # Generate prediction. If the model correctly identifies the sample, increase
  # the counter.
    prediction = model.predict(img_array, verbose = 0)
    predicted_class = np.argmax(prediction)
    if predicted_class == class_id:
      class_count = class_count + 1
  print('Number of correct predictions for class ' + class_name + ' to total is: ', class_count/class_num)

for i in range (0,len(class_counts)):
  prediction_ratio(class_names[i],class_counts[i],class_ids[i])
```

Παράρτημα ΣΤ

Alzheimer's Transfer Learning Based Classification Model

```
#Module Initialization
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import kaggle
import time
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint,
 ReduceLROnPlateau

dl_path = '/path/to/folder/'

if not os.path.exists(dl_path):
  kaggle.api.authenticate()
  kaggle.api.dataset_download_files('tourist55/alzheimers-dataset-4-
 class-of-images', path = dl_path, unzip=True)
  print('Files successfully downloaded!')
else:
  print('Data has already been downloaded.')

# Defining image directory.
image_folder = dl_path + 'Alzheimer_s Dataset/'

# Defining training and test image directories.
train_dir = image_folder + 'train'
test_dir = image_folder + 'test'

#Defining Data Generators for train and validation data.

datagen_train = ImageDataGenerator(rescale=1./255, validation_split =
 0.2)
datagen_test = ImageDataGenerator(rescale=1./255)

#Creating database files
train_data = datagen_train.flow_from_directory(train_dir, batch_size =
```

```
    32, shuffle = True, class_mode = 'categorical', target_size = (176,208)
    , seed = 69, subset = "training")
valid_data = datagen_train.flow_from_directory(train_dir, batch_size =
    32, shuffle = True, class_mode = 'categorical', target_size = (176,208)
    , seed = 69, subset = "validation")
test_data = datagen_test.flow_from_directory(test_dir, class_mode = '
    categorical', target_size = (176,208), batch_size = 32)

# We will proceed to evaluate data ratios in our imageset.
num_mild = len(os.listdir(os.path.join(train_dir, 'MildDemented')))
num_moderate = len(os.listdir(os.path.join(train_dir, 'ModerateDemented'
    )))
num_non = len(os.listdir(os.path.join(train_dir, 'NonDemented')))
num_vmild = len(os.listdir(os.path.join(train_dir, 'VeryMildDemented')))
print(f"Patients with Very Mild Dementia = {num_vmild}")
print(f"Patients with Mild Dementia = {num_mild}")
print(f"Patients with Moderate Dementia = {num_moderate}")
print(f"Patients without Dementia = {num_non}")

# Code that ensures we correctly loaded the images to be used for
    training.
# This is used for visualization purposes.
def plots(ims, figsize=(12,6), rows=3, interp=False, titles=None):
  if type(ims[0]) is np.ndarray:
    ims = np.array(ims)
    if (ims.shape[-1] != 3):
      ims = ims.transpose((0,2,3,1))
  f = plt.figure(figsize=figsize)
  cols = len(ims)//rows if len(ims) % 2 == 0 else len(ims)//rows + 1
  for i in range(len(ims)):
    try:
    sp = f.add_subplot(rows, cols, i+1)
    sp.axis('Off')
    except ValueError:
      break
    if titles is not None:
      sp.set_title(titles[i], fontsize=16)
    plt.imshow(ims[i], interpolation=None if interp else 'none')


imgs, labels = next(train_data)
labels_new = []
for i in range(0,len(labels)):
  if np.allclose(labels[i],np.array([1, 0, 0, 0])) == True:
    labels_new.append('Mild')
  elif np.allclose(labels[i],np.array([0, 1, 0, 0])) == True:
    labels_new.append('Mod')
  elif np.allclose(labels[i],np.array([0, 0, 1, 0])) == True:
    labels_new.append('Non')
  elif np.allclose(labels[i],np.array([0, 0, 0, 1])) == True:
    labels_new.append('V.Mild')
plots(imgs, titles = labels_new)

transfer_model = tf.keras.applications.VGG19(input_shape=(176,208,3),
```

```
 include_top=False, weights='imagenet')
for layer in transfer_model.layers:
  layer.trainable = False

input = tf.keras.Input(shape = (176, 208, 3))
x = tf.keras.layers.RandomFlip('horizontal_and_vertical', input_shape =
 (176, 208, 3))(input)
x = tf.keras.layers.RandomRotation(0.2, input_shape = (176, 208, 3))(x)
x = transfer_model(x)
x = tf.keras.layers.Flatten()(x)
x = tf.keras.layers.Dense(128, activation = 'relu')(x)
x = tf.keras.layers.Dropout(0.1)(x)
prediction = tf.keras.layers.Dense(4, activation = 'softmax')(x)
model = tf.keras.Model(input, prediction)
model.compile(loss = "categorical_crossentropy", optimizer = "adam",
 metrics = ['accuracy', 'AUC'])

# We use a simple time module to record how much time the network took
 to train.
# This can be safely removed, unless the user is curious for learning/
 diagnostic purposes.
# If training on a CPU, the process will take significantly longer.
start_time = time.time()

# We use various callbacks to monitor the model's training and ensure
 minimum time losses in case
# the model starts to overfit/the process is lengthy enough to make
 manual monitoring impossible.
# It is generally recommended to use all 3 callbacks below to ensure
 best model performance.
#
# We use Early Stopping to monitor a specific metric and terminate
 training if there's no improvement
# after a set number of epochs. Checkpoint saves the model when a
 monitored value is improved.
# Finally, we can automatically tweak the learning rate of the model if
 our preferred monitored values
# shows no improvement after a certain number of epochs.
early_stopping = EarlyStopping(monitor='val_loss', patience = 15,
 restore_best_weights = True)
checkpoint = ModelCheckpoint("Alzheimer's Checkpoint.h5", monitor='
 val_auc', verbose = 1, save_best_only = True, mode='max')
lr_monitor = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience
 =13, min_lr=0.00001)

# Initiating training.
history = model.fit(train_data, validation_data = valid_data, epochs =
 100, verbose = 1, callbacks = [early_stopping, checkpoint, lr_monitor])

# Tensorflow recommends saving models in .keras format. To comply with
 that, we save the best model generated
# by the callback functions used above.
model.save('Alzheimer VGG 19 Transfer Learning Model.keras')
print(time.time() - start_time, 'sec')
```

```
121
122 # Plot model accuracy for training and validation datasets
123 plt.plot(history.history['auc'], label = 'accuracy')
124 plt.plot(history.history['val_auc'], label = 'val_accuracy')
125 plt.title('Correlation of Training vs Validation Data')
126 plt.xlabel('Epochs')
127 plt.ylabel('Accuracy')
128 plt.legend(['Training Data','Validation Data'], loc='lower right')
129
130 # Generate model summary
131 model.summary()
132
133 #Generate evaluation metrics for all 3 datasets
134 train_eval = model.evaluate(train_data, verbose = 0)
135 val_eval = model.evaluate(valid_data, verbose = 0)
136 test_eval = model.evaluate(test_data, verbose = 0)
137 print('Accuracy, loss and AUC scored compared to training data is:',
     train_eval[1], ',', train_eval[0], ',', train_eval[2])
138 print('Accuracy, loss and AUC scored compared to validation data is:',
     val_eval[1],',', val_eval[0],',', val_eval[2])
139 print('Accuracy, loss and AUC scored compared to test data is:',
     test_eval[1],',', test_eval[0],',', test_eval[2])
140
141 # First, we print the class names so that we know how to evaluate each
     prediction.
142 # Afterwards, we create a list that will contain the string values of
     each class.
143 class_names = list(train_data.class_indices.keys())
144 class_ids = list(train_data.class_indices.values())
145
146 # We next define the list of image files in each directory. We will use
     that to create a custom generator that will
147 # validate each picture separately using the model generated.
148
149 mild_test = len(os.listdir(image_folder + 'test/MildDemented'))
150 mod_test = len(os.listdir(image_folder + 'test/ModerateDemented'))
151 non_test = len(os.listdir(image_folder + 'test/NonDemented'))
152 vmild_test = len(os.listdir(image_folder + 'test/VeryMildDemented'))
153
154 class_counts = [mild_test, mod_test, non_test, vmild_test]
155
156 # The following function is called in order to calculate the ratio of
     correct predictions per class sample.
157 # test_num = The variables instantiated above.
158 # class_name = The string value of the class we want to predict.
159 # class_num = The numerical value of the predicted class, as defined in
     class_names.
160
161 def prediction_ratio(class_name, class_num, class_id):
162   class_count = 0
163   for i in range (0, class_num):
164   # Load image and normalize it.
165     img_path = image_folder + 'test/' + class_name +'/' + str(os.listdir
    (image_folder + 'test/' + class_name)[i])
```

```
166       img = keras.preprocessing.image.load_img(img_path, target_size=(176,
      208))
167       img_array = keras.preprocessing.image.img_to_array(img)
168       img_array = np.expand_dims(img_array, axis=0)
169       img_array /= 255.
170
171       # Generate prediction. If the model correctly identifies the sample,
      increase
172       # the counter.
173       prediction = model.predict(img_array, verbose = 0)
174       predicted_class = np.argmax(prediction)
175       if predicted_class == class_id:
176         class_count = class_count + 1
177     print('Number of correct predictions for class ' + class_name + ' to
    total is: ', class_count/class_num)
178
179   for i in range (0,len(class_counts)):
180     prediction_ratio(class_names[i],class_counts[i],class_ids[i])
```

Παράρτημα Ζ

Graphical User Interface Source Code

```
import sys
import tensorflow as tf
import numpy as np
from tensorflow import keras
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QApplication, QFileDialog, QMessageBox,
 QSizePolicy
from PyQt5.QtGui import QImage, QPixmap, QPalette
tf.config.threading.set_intra_op_parallelism_threads(1)
tf.config.threading.set_inter_op_parallelism_threads(1)
model_exists = False
fileName_exists = False
fileName = ()
model = ()

# Initialize main GUI elements
class Ui_MainWindow(object):

  def setupUi(self, MainWindow):
    MainWindow.setObjectName("MainWindow")
    MainWindow.resize(860, 640)
    MainWindow.setMinimumSize(QtCore.QSize(860, 640))
    MainWindow.setMaximumSize(QtCore.QSize(860, 640))
    MainWindow.setBaseSize(QtCore.QSize(860, 640))
    self.centralwidget = QtWidgets.QWidget(MainWindow)
    self.centralwidget.setObjectName("centralwidget")
    self.comboBox = QtWidgets.QComboBox(self.centralwidget)
    self.comboBox.setGeometry(QtCore.QRect(9, 9, 174, 20))
    self.comboBox.setSizeAdjustPolicy(QtWidgets.QComboBox.
 AdjustToContentsOnFirstShow)
    self.comboBox.setFrame(False)
    self.comboBox.setObjectName("comboBox")
    self.comboBox.addItem("")
    self.comboBox.addItem("")
    self.comboBox.addItem("")
    self.comboBox.addItem("")
    self.comboBox.addItem("")
    self.imageLabel = QtWidgets.QLabel(self.centralwidget)
```

```
37      self.imageLabel.setGeometry(QtCore.QRect(16, 49, 821, 471))
38      self.imageLabel.setObjectName("Image Label")
39      self.imageLabel.setBackgroundRole(QPalette.Window)
40      self.imageLabel.setSizePolicy(QSizePolicy.Preferred, QSizePolicy.
    Preferred)
41      self.scrollArea = QtWidgets.QScrollArea(self.centralwidget)
42      self.scrollArea.setGeometry(QtCore.QRect(16, 49, 821, 471))
43      self.scrollArea.setBackgroundRole(QPalette.Dark)
44      self.scrollArea.setWidget(self.imageLabel)
45      self.scrollArea.setVerticalScrollBarPolicy(QtCore.Qt.
    ScrollBarAlwaysOff)
46      self.scrollArea.setHorizontalScrollBarPolicy(QtCore.Qt.
    ScrollBarAlwaysOff)
47      self.scrollArea.setVisible(False)
48      self.pushButton = QtWidgets.QPushButton(self.centralwidget)
49      self.pushButton.setGeometry(QtCore.QRect(9, 538, 80, 23))
50      self.pushButton.setObjectName("pushButton")
51      self.pushButton.clicked.connect(self.open)
52      self.pushButton_2 = QtWidgets.QPushButton(self.centralwidget)
53      self.pushButton_2.setGeometry(QtCore.QRect(9, 567, 80, 23))
54      self.pushButton_2.setObjectName("pushButton_2")
55      self.pushButton_2.clicked.connect(self.Predict)
56      self.label = QtWidgets.QLabel(self.centralwidget)
57      self.label.setGeometry(QtCore.QRect(100, 540, 751, 16))
58      self.label.setObjectName("label")
59      MainWindow.setCentralWidget(self.centralwidget)
60      self.menubar = QtWidgets.QMenuBar(MainWindow)
61      self.menubar.setGeometry(QtCore.QRect(0, 0, 860, 21))
62      self.menubar.setObjectName("menubar")
63      MainWindow.setMenuBar(self.menubar)
64      self.statusbar = QtWidgets.QStatusBar(MainWindow)
65      self.statusbar.setObjectName("statusbar")
66      MainWindow.setStatusBar(self.statusbar)
67      self.retranslateUi(MainWindow)
68      self.comboBox.setCurrentIndex(0)
69      QtCore.QMetaObject.connectSlotsByName(MainWindow)
70
71    def retranslateUi(self, MainWindow):
72      _translate = QtCore.QCoreApplication.translate
73      MainWindow.setWindowTitle(_translate("MainWindow", "Model Loader"))
74      self.comboBox.setCurrentText(_translate("MainWindow", "
      - Select Model -"))
75      self.comboBox.setPlaceholderText(_translate("MainWindow", "Please
    Select Model"))
76      self.comboBox.setItemText(0, _translate("MainWindow", "
      - Select Model -"))
77      self.comboBox.setItemText(1, _translate("MainWindow", "
    Brain Tumor  Detection"))
78      self.comboBox.setItemText(2, _translate("MainWindow", "
    Skin Cancer Detection"))
79      self.comboBox.setItemText(3, _translate("MainWindow", "       COVID
    -19 X-Ray Detection"))
80      self.comboBox.setItemText(4, _translate("MainWindow", "
    Pneumonia Detection"))
```

```
81      self.comboBox.currentIndexChanged.connect(self.LoadModel)
82      self.pushButton.setText(_translate("MainWindow", "Load Image"))
83      self.pushButton_2.setText(_translate("MainWindow", "Predict Image"))
84
85  # Define Main Window and functions
86  class MainWindow(QtWidgets.QMainWindow, Ui_MainWindow):
87    def __init__(self, *args, obj=None, **kwargs):
88      super(MainWindow, self).__init__(*args, **kwargs)
89      self.setupUi(self)
90      self.msg = QMessageBox()
91
92  # Function responsible for opening dialog window to choose image.
93    def open(self):
94      options = QFileDialog.Options()
95      global fileName
96      fileName, _ = QFileDialog.getOpenFileName(self, 'QFileDialog.
    getOpenFileName()', '',
97      'Images (*.png *.jpeg *.jpg *.bmp *.gif)', options=options)
98      self.imageLabel.setPixmap(QtGui.QPixmap())
99      if fileName:
100       image = QImage(fileName)
101       if image.isNull():
102         QMessageBox.information(self, "Image Viewer", "Cannot load %s."
    % fileName)
103         return
104
105
106       self.imageLabel.setPixmap(QPixmap.fromImage(image).scaled(self.
    imageLabel.size(), QtCore.Qt.KeepAspectRatio))
107       self.imageLabel.setAlignment(QtCore.Qt.AlignCenter)
108       self.scaleFactor = 1.0
109       self.scrollArea.setVisible(True)
110       print(fileName)
111       global fileName_exists
112       fileName_exists = True
113
114 # Function handling model loading based on dropdown menu pick.
115   def LoadModel(self):
116     if not self.comboBox.currentIndex() == 0:
117       self.pushButton.setEnabled(False)
118       self.pushButton_2.setEnabled(False)
119       self.comboBox.setEnabled(False)
120       QApplication.processEvents()
121       global model
122       if self.comboBox.currentIndex() == 1:
123         model = tf.keras.models.load_model("models/brain.h5", compile=
    False)
124         model.compile(loss = "binary_crossentropy", optimizer = "adam",
    metrics = ['accuracy'])
125         print('brain loaded')
126       elif self.comboBox.currentIndex() == 2:
127         model = tf.keras.models.load_model("models/skin.h5", compile=
    False)
128         model.compile(loss = "categorical_crossentropy", optimizer = "
```

```
   adam", metrics = ['accuracy'])
129          print('skin loaded')
130        elif self.comboBox.currentIndex() == 3:
131          model = tf.keras.models.load_model("models/covid.h5", compile=
   False)
132          model.compile(loss = "binary_crossentropy", optimizer = "adam",
   metrics = ['accuracy'])
133          print('covid loaded')
134        elif self.comboBox.currentIndex() == 4:
135          model = tf.keras.models.load_model("models/xray.h5", compile=
   False)
136          model.compile(loss = "binary_crossentropy", optimizer = "adam",
   metrics = ['accuracy'])
137          print('xray loaded')
138        global model_exists
139        model_exists = True
140        self.pushButton.setEnabled(True)
141        self.pushButton_2.setEnabled(True)
142        self.comboBox.setEnabled(True)
143      else:
144        model = ()
145        model_exists = False
146
147  # Main prediction function. Also handles GUI exceptions if a user hasn't
     chosen a
148  # model or image prior to attempting a prediction.
149    def Predict(self):
150      self.pushButton.setEnabled(False)
151      self.pushButton_2.setEnabled(False)
152      self.comboBox.setEnabled(False)
153      if model_exists is False:
154        self.msg.setWindowTitle("Error")
155        self.msg.setText("You haven't loaded a model yet, please try again
   !")
156        self.msg.setIcon(QMessageBox.Critical)
157        self.msg.setStandardButtons(QMessageBox.Ok)
158        self.msg.setDefaultButton(QMessageBox.Ok)
159        self.msg.show()
160        self.pushButton.setEnabled(True)
161        self.pushButton_2.setEnabled(True)
162        self.comboBox.setEnabled(True)
163      else:
164        if fileName_exists is False or fileName == '':
165          self.msg = QMessageBox()
166          self.msg.setWindowTitle("Error")
167          self.msg.setText("You haven't loaded an image yet, please try
   again!")
168          self.msg.setIcon(QMessageBox.Critical)
169          self.msg.setStandardButtons(QMessageBox.Ok)
170          self.msg.setDefaultButton(QMessageBox.Ok)
171          self.msg.show()
172          self.pushButton.setEnabled(True)
173          self.pushButton_2.setEnabled(True)
174          self.comboBox.setEnabled(True)
```

```
        else:
          img = keras.preprocessing.image.load_img(fileName, target_size
    =(256, 256))
          img_array = keras.preprocessing.image.img_to_array(img)
          img_array = np.expand_dims(img_array, axis=0)
          img_array /= 255.

          #Print classes to be used for cross-referencing
          if self.comboBox.currentIndex() == 1:
            class_names = {0: 'Healthy', 1:"Tumor"}
            print(class_names)
          elif self.comboBox.currentIndex() == 2:
            class_names = {0: 'akiec', 1:"bcc", 2:'bkl', 3:'df', 4:'mel',
    5:'nv', 6:'vasc'}
            print(class_names)
          elif self.comboBox.currentIndex() == 3:
            class_names = {0: 'Healthy', 1:"COVID"}
            print(class_names)
          elif self.comboBox.currentIndex() == 4:
            class_names = {0: 'Healthy', 1:"Pneumonia"}
            print(class_names)

        #Prediction of model
     if self.comboBox.currentIndex() == 2:
       prediction = model.predict(img_array, verbose = 0)
       predicted_class = np.argmax(prediction)
       print(predicted_class)
       pred_list = prediction.ravel().tolist()
       #print(100*(pred_list[predicted_class]))
       print ("Image belongs to class " + class_names[predicted_class] + "
    with " +  str(round(100*(pred_list[predicted_class]),3)) + "%
    confidence." )
       self.label.setText("Image belongs to class " + class_names[
    predicted_class] + " with " + str(round(100*(pred_list[predicted_class
    ]),3)) + "% confidence.")
     else:
       prediction = model.predict(img_array, verbose = 0)
       predicted_class = np.round(prediction)
       prediction_new = prediction.ravel().tolist()
       print(predicted_class)
       if predicted_class == 1:
         print ("Image belong to class " + class_names[1] + " with " + str(
    round(prediction_new[0]*100,3)) + "% confidence." )
         self.label.setText("Image belongs to class " + class_names[1] + "
    with " + str(round(prediction_new[0]*100, 3)) + "% confidence.")
       else:
         print ("Image belongs to class " + class_names[0] + " with " + str
    (round(100*(1-(prediction_new[0])),3)) + "% confidence." )
         self.label.setText("Image belongs to class " + class_names[0] + "
    with " + str(round(100*(1-(prediction_new[0])),3)) + "% confidence." )
     print(prediction)
     self.pushButton.setEnabled(True)
     self.pushButton_2.setEnabled(True)
     self.comboBox.setEnabled(True)
```

```
219
220 app = QtWidgets.QApplication(sys.argv)
221
222 window = MainWindow()
223 window.show()
224 app.exec()
```