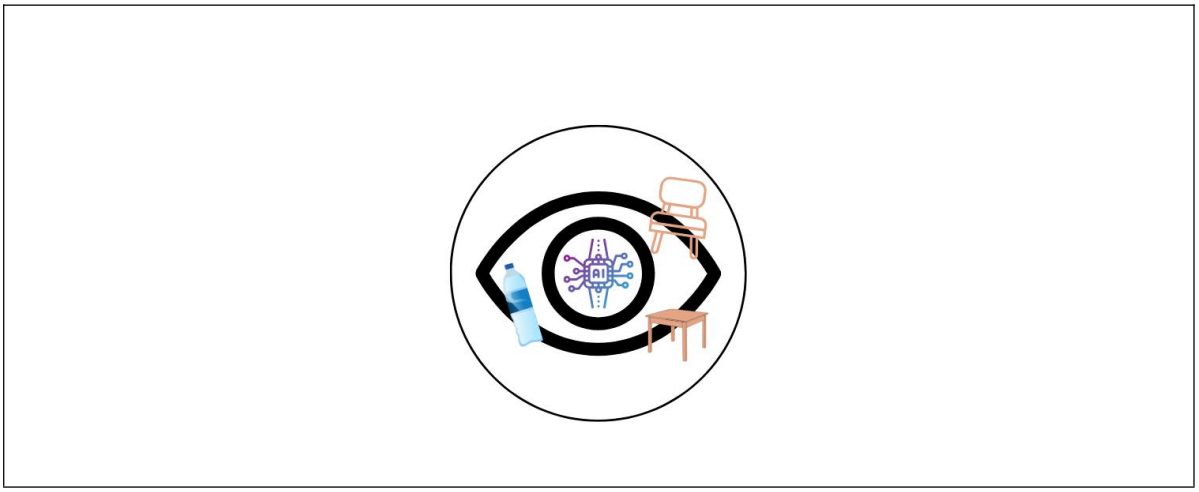
ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

«Δημιουργία εφαρμογής android για ανίχνευση αντικειμένων μέσω κάμερας με την βοήθεια της τεχνητής νοημοσύνης και υπολογισμό της απόστασης τους από τον χρήστη και ενημέρωση του χρήστη μέσω φωνητικής αλληλεπίδρασης.»



Του φοιτητή
Μήτσης Θεόδωρος
Αρ. Μητρώου: 185230

Επιβλέπων
Ονοματεπώνυμο: Κοκκώνης Γεώργιος
Βαθμίδα: Επίκουρος Καθηγητής

Ημερομηνία ……

Τίτλος Π.Ε. Δημιουργία εφαρμογής android για ανίχνευση αντικειμένων μέσω κάμερας με την βοήθεια της τεχνητής νοημοσύνης και υπολογισμό της απόστασης τους από τον χρήστη και ενημέρωση του χρήστη μέσω φωνητικής αλληλεπίδρασης.

Κωδικός Π.Ε. 25247

Ονοματεπώνυμο φοιτητή/τών Θεόδωρος Μήτσης

Ονοματεπώνυμο εισηγητή Γεώργιος Κοκκώνης

Ημερομηνία ανάληψης Π.Ε. 03/04/25

Ημερομηνία περάτωσης Π.Ε. …

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Η έγκριση της διπλωματικής εργασίας από το Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνούς Πανεπιστημίου της Ελλάδος, δεν υποδηλώνει απαραιτήτως και αποδοχή των απόψεων του συγγραφέα, εκ μέρους του Τμήματος.

«Αφιέρωση»

Πρόλογος

Η τεχνητή νοημοσύνη αποτελεί έναν από τους πιο συναρπαστικούς και εξελισσόμενους τομείς της σύγχρονης τεχνολογίας. Με την δυνατότητα της να «μαθαίνει» από δεδομένα και να παίρνει αποφάσεις, μεταμορφώνει σταδιακά τον τρόπο με τον οποίο καταλαβαίνουμε τον κόσμο και αλληλεπιδρούμε με αυτόν. Στο σήμερα, η τεχνητή νοημοσύνη έχει πλέον μπεί στις ζωές μας, από τις έξυπνες εφαρμογές που μας βοηθούν μέχρι τα συστήματα πλοήγησης και τις υπηρεσίες υγείας. Ανάμεσα στις πιο ουσιαστικές της εφαρμογές ξεχωρίζει η ικανότητα να ενισχύει την αυτονομία και την ποιότητα ζωής ατόμων με προβλήματα όρασης, δίνοντάς τους πρόσβαση σε πληροφορίες που παλαιότερα δεν ήταν άμεσα διαθέσιμες. Επέλεξα να ασχοληθώ με αυτό το θέμα γιατί μου αρέσει πάρα πολύ ο κόσμος της τεχνητής νοημοσύνης και οι προκλήσεις που προκύπτουν από τη δημιουργία καινοτόμων εφαρμογών. Μέσα από αυτή τη πτυχιακή εργασία είχα την ευκαιρία να συνδυάσω τη δημιουργικότητα με την τεχνολογία, με στόχο τη βελτίωση της καθημερινότητας ανθρώπων με προβλήματα όρασης.

Περίληψη

Η παρούσα πτυχιακή εργασία εξετάζει την χρησιμότητα της Τεχνητής Νοημοσύνης (ΤΝ) για την βοήθεια ατόμων με προβλήματα όρασης, μέσα από την ανάπτυξη μιας εφαρμογής για Android συσκευές. Σκοπός της εφαρμογής είναι η ανίχνευση αντικειμένων σε πραγματικό χρόνο μέσω κάμερας και μοντέλου μηχανικής μάθησης, ενώ ταυτόχρονο υπολογίζει την απόσταση κάθε αντικειμένου απο τον χρήστη και με φωνητικές εντολές του ανακοινώνει που βρίσκονται τα αντικείμενα, με στόχο τη βελτίωση της κινητικότητας και την αποφυγή εμποδίων.

Η εργασία καλύπτει τον σχεδιασμό, την υλοποίηση και την αξιολόγηση της εφαρμογής, ενώ παράλληλα αναδεικνύει τις δυνατότητες της ΤΝ ως υποβοηθητικής τεχνολογίας για άτομα με προβλήματα όρασης. Μέσα από τη μελέτη αυτή προκύπτουν χρήσιμα συμπεράσματα για τη χρηστικότητα και τις μελλοντικές προοπτικές τέτοιων λύσεων.

« Development of an Android application for object detection through the camera using artificial intelligence, with distance estimation and user notification via voice interaction. »

Mitsis Theodoros

Abstract

This thesis investigates the use of Artificial Intelligence (AI) as a supportive tool for visually impaired individuals through the development of an Android application. The application enables real-time object detection using a camera and a machine learning model, calculates the distance of each object from the user, and provides voice guidance for their identification, aiming to enhance mobility and obstacle avoidance. The study covers the design, implementation, and evaluation of the application, highlighting the potential of AI as an assistive technology. The results emphasize the usability of the application and offer valuable insights into the future prospects of such solutions.

Ευχαριστίες

Θα ήθελα να εκφράσω την ευγνωμοσύνη μου στον καθηγητή, Γεώργιο Κοκκώνη, για την πολύτιμη καθοδήγηση, τις συμβουλές και την υποστήριξή του καθ' όλη τη διάρκεια της εκπόνησης αυτής της πτυχιακής εργασίας. Επιπλέον, θα ήθελα να ευχαριστώ την οικογένειά μου για την συμπαράσταση σε όλη τη διάρκεια των σπουδών μου.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Κατάλογος Εικόνων

Συντομογραφίες

Δ.Ε.	Διπλωματική Εργασία
ΔΙΠΑΕ	Διεθνές Πανεπιστήμιο Ελλάδος
Π.Ε.	Πτυχιακή Εργασία

Κεφάλαιο 1ο: Εισαγωγή και Πλαίσιο της Εργασίας

1.1 Εισαγωγή

Η ραγδαία εξέλιξη της τεχνολογίας και ειδικότερα της Τεχνητής Νοημοσύνης έχει ανοίξει νέους δρόμους σε πολλούς τομείς, συμπεριλαμβανομένων και των υποβοηθητικών τεχνολογιών για άτομα με αναπηρία. Ένα από τα σημαντικότερα πεδία εφαρμογής αφορά την υποστήριξη ατόμων με προβλήματα όρασης, προσφέροντας λύσεις που ενισχύουν την αυτονομία τους και βελτιώνουν την καθημερινότητά τους.

Η ενσωμάτωση της ΤΝ σε έξυπνες συσκευές, όπως τα σύγχρονα κινητά τηλέφωνα, επιτρέπει την ανάπτυξη εφαρμογών που αναγνωρίζουν αντικείμενα στο περιβάλλον, υπολογίζουν αποστάσεις και παρέχουν πληροφορίες στον χρήστη μέσω φωνητικής καθοδήγησης. Τέτοιες εφαρμογές συνδυάζουν τεχνολογίες μηχανικής μάθησης, υπολογιστικής όρασης και διεπαφών χρήστη, αποτελώντας καινοτόμες προσεγγίσεις στην υποστήριξη της προσβασιμότητας.

Στην παρούσα πτυχιακή εργασία εξετάζεται η ανάπτυξη μιας τέτοιας εφαρμογής για Android συσκευές, με στόχο την παροχή πληροφοριών στον χρήστη σε πραγματικό χρόνο, μέσω της κάμερας, ενισχύοντας την πλοήγηση και την κατανόηση του χώρου γύρω του.

1.2 Πλαίσιο και Αναγκαιότητα

Σύμφωνα με τον Παγκόσμιο Οργανισμό Υγείας, εκατομμύρια άνθρωποι σε όλο τον κόσμο έχουν προβλήματα όρασης[1], κάτι το οποιό επηρεάζει σημαντικά την καθημερινότητά τους και δεν μπορούν να είναι ανεξάρτητοι[2]. Η δυνατότητα αυτονομίας, η αντίληψη του χώρου και η δυνατότητα αναγνώρισης αντικειμένων αποτελούν μείζονες προκλήσεις για άτομα με μερική ή ολική απώλεια όρασης[2].

Η ραγδαία πρόοδος στον τομέα της Τεχνητής Νοημοσύνης, ειδικά στην Υπολογιστική Όραση και τα Νευρωνικά Δίκτυα, έχει δημιουργήσει τις προϋποθέσεις για την ανάπτυξη εργαλείων που μπορούν να καλύψουν αυτές τις ανάγκες[3],[5]. Αυτό σημαίνει πως δίνει την δυνατότητα ακόμα και σε κινητές συσκευές να δημιουργούν «έξυπνες» εφαρμογές που αναγνωρίζουν αντικείμενα, υπολογίζουν αποστάσεις και παρέχουν καθοδήγηση σε πραγματικό χρόνο[4],[5] για να βοηθήσουν τους ανθρώπους με προβλήματα όρασης να ζούν ποιο ανεξάρτητοι.

Η συγκεκριμένη εργασία εστιάζει στη δημιουργία μιας εφαρμογής Android, η οποία αξιοποιεί τις δυνατότητες της ΤΝ προκειμένου να βοηθά ανθρώπους με προβλήματα όρασης να «βλέπουν» καλύτερα τον κόσμο γύρω τους. Μέσα από αυτή την προσέγγιση, τονίζεται η σημασία της τεχνολογίας ως μέσο κοινωνικής ένταξης και ισότιμης συμμετοχής.

1.3 Σκοπός και Στόχοι της εργασίας

Ο βασικός στόχος αυτής της εργασίας είναι να δημιουργήσουμε μια εφαρμογή για κινητά τηλέφωνα, που θα βοηθάει ανθρώπους με πρόβλημα όρασης να αντιλαμβάνονται καλύτερα τα πράγματα γύρω τους. Η εφαρμογή θα χρησιμοποιεί τεχνολογία Τεχνητής Νοημοσύνης για να αναγνωρίζει αντικείμενα, απόσταση αντικειμένων και να τους δίνει πληροφορίες με φωνητικές εντολές.

Πιο συγκεκριμένα θέλουμε:

- Να κατασκευάσουμε μια εύχρηστη εφαρμογή, απλή στη χρήση ακόμα και για ανθρώπους που δεν έχουν μεγάλη εμπειρία με την τεχνολογία.
- Να βελτιώσουμε την ακρίβεια στην αναγνώριση αντικειμένων μέσα από την κάμερα του κινητού.
- Να κάνουμε την εφαρμογή όσο πιο γρήγορη και αξιόπιστη γίνεται, ώστε να μπορεί να βοηθάει σε πραγματικό χρόνο.
- Να ενσωματώσουμε φωνητική έξοδο, ώστε να μη χρειάζεται να κοιτάει κανείς την οθόνη, κάτι που κάνει την εφαρμογή πιο φιλική για άτομα με σοβαρότερα προβλήματα όρασης.

Με αυτόν τον τρόπο, πιστεύουμε ότι η εφαρμογή θα γίνει ένα χρήσιμο εργαλείο για καθημερινή υποστήριξη και ανεξαρτησία.

1.4 Δομή της Πτυχιακής Εργασίας

Η παρούσα εργασία αποτελείται από έξι βασικά κεφάλαια. Στο πρώτο κεφάλαιο γίνεται μια εισαγωγή στο θέμα, παρουσιάζεται το γενικό πλαίσιο της εργασίας, οι λόγοι για τους οποίους κρίθηκε σημαντική η υλοποίησή της, καθώς και οι στόχοι που θέτει. Επιπλέον, γίνεται μια σύντομη αναφορά στη συμβολή της Τεχνητής Νοημοσύνης σε σύγχρονες εφαρμογές. Στην συνέχεια και στο επόμενο κεφάλαιο παρουσιάζονται βασικές έννοιες της Τεχνητής Νοημοσύνης και της Υπολογιστικής Όρασης, όπως η μηχανική μάθηση και τα νευρωνικά δίκτυα. Γίνεται επίσης αναφορά στον τρόπο με τον οποίο τέτοιες τεχνολογίες εφαρμόζονται σε κινητές συσκευές και πώς μπορούν να αναγνωρίζουν αντικείμενα σε πραγματικό χρόνο. Το τρίτο κεφάλαιο επικεντρώνεται στα μέσα αλληλεπίδρασης με την εφαρμογή, δίνοντας έμφαση στη φωνητική και απτική επικοινωνία. Εξετάζεται επίσης η σημασία του σωστού σχεδιασμού της εμπειρίας χρήστη για άτομα με προβλήματα όρασης. Στο τέταρτο κεφάλαιο περιγράφεται αναλυτικά η ανάπτυξη της εφαρμογής σε Android Studio. Εξετάζεται ο τρόπος ενσωμάτωσης του μοντέλου για αναγνώριση αντικειμένων, η διαδικασία υπολογισμού αποστάσεων, καθώς και η φωνητική ανατροφοδότηση προς τον χρήστη. Στο πέμπτο κεφάλαιο παρουσιάζονται τα σενάρια χρήσης της εφαρμογής, η μεθοδολογία των δοκιμών και η ανατροφοδότηση που συλλέχθηκε από χρήστες, ώστε να αξιολογηθεί η πρακτική χρησιμότητά της. Τέλος, στο έκτο κεφάλαιο συνοψίζονται τα συμπεράσματα που προέκυψαν από την εργασία, καταγράφονται οι δυσκολίες που αντιμετωπίστηκαν κατά την υλοποίηση και διατυπώνονται προτάσεις για μελλοντικές βελτιώσεις και επεκτάσεις.

1.5 Η Συμβολή της Τεχνητής Νοημοσύνης

Η Τεχνητή Νοημοσύνη τα τελευταία χρόνια έχει αρχίσει να παίζει σημαντικό ρόλο σε πολλές πτυχές της καθημερινής ζωής, προσφέροντας λύσεις που μέχρι πρόσφατα θεωρούνταν αδύνατες[1]. Στον τομέα της προσβασιμότητας, η συνεισφορά της είναι ιδιαίτερα σημαντική, καθώς ανοίγει νέους δρόμους για την αυτονομία και την ανεξάρτητη διαβίωση ατόμων με αναπηρίες και ειδικά για ανθρώπους με προβλήματα όρασης[2]. Στην περίπτωση αυτής της εργασίας, η ΤΝ αξιοποιείται μέσα από ένα μοντέλο αναγνώρισης αντικειμένων, το οποίο λειτουργεί σε πραγματικό χρόνο μέσω της κάμερας μιας κινητής συσκευής[3],[5]. Ο χρήστης ενημερώνεται φωνητικά για το τι υπάρχει μπροστά του[4], κάτι που ενισχύει την αντίληψή του για τον χώρο γύρω του. Με αυτόν τον τρόπο, η τεχνολογία δεν παραμένει απλώς ένα εργαλείο, αλλά γίνεται ουσιαστικά ένα “βοηθητικό μάτι”[4], που στηρίζει ενεργά τον χρήστη στην καθημερινότητά του.
Η ΤΝ, σε συνδυασμό με τεχνικές όπως η φωνητική αλληλεπίδραση και η απτική ανατροφοδότηση, αποδεικνύεται ικανή να αλλάξει ριζικά τον τρόπο με τον οποίο τα άτομα με προβλήματα όρασης αντιλαμβάνονται και αλληλεπιδρούν με το περιβάλλον τους. Η εργασία αυτή αποτελεί ένα παράδειγμα του πώς η καινοτομία μπορεί να έχει πραγματικό κοινωνικό αντίκτυπο[1],[2].

1.6 Επίλογος

Το πρώτο κεφάλαιο έθεσε τα θεμέλια της παρούσας πτυχιακής εργασίας, παρουσιάζοντας το γενικό πλαίσιο, την ανάγκη ύπαρξης της εφαρμογής, καθώς και τον σκοπό και τους στόχους που επιχειρεί να καλύψει. Αναδείχθηκε η σημασία της Τεχνητής Νοημοσύνης στην υποστήριξη ατόμων με προβλήματα όρασης και έγινε σαφές πως η συγκεκριμένη τεχνολογία, όταν αξιοποιείται κατάλληλα, μπορεί να προσφέρει ουσιαστικές λύσεις σε καθημερινές προκλήσεις.

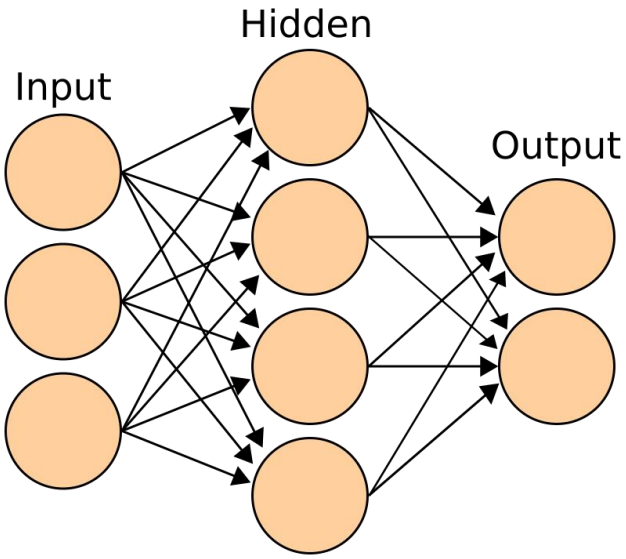
Κεφάλαιο 2ο: Τεχνητή Νοημοσύνη και Όραση Υπολογιστών

2.1 Εισαγωγή

Η υπολογιστική όραση (computer vision) αποτελεί πεδίο της ΤΝ που ασχολείται με την εξαγωγή χρήσιμων πληροφοριών από ψηφιακές εικόνες και βίντεο[6]. Όπως επισημαίνει η IBM, η όραση υπολογιστών «μαθαίνει» τα συστήματα να «βλέπουν», να αναγνωρίζουν αντικείμενα και να λαμβάνουν αποφάσεις όταν εντοπίζουν ελαττώματα ή σημαντικές αλλαγές στις εικόνες[6]. Σύγχρονα δίκτυα CNN (convolutional neural networks) διευκολύνουν τη διαδικασία αυτή, αναλύοντας εικόνες σε pixel και εφαρμόζοντας συνελίξεις για να εντοπίσουν χαρακτηριστικά και σχήματα. Οι εξελίξεις στη βαθιά μάθηση έχουν καταστήσει την όραση υπολογιστών «πανταχού παρούσα» με εφαρμογές σε τομείς όπως αναζήτηση εικόνων, ιατρική απεικόνιση, ρομποτική, drones, χαρτογράφηση και αυτόνομα οχήματα[7]. Κοινές εργασίες στον τομέα περιλαμβάνουν την ταξινόμηση εικόνων (image classification), την τοπική σήμανση (localization) και την ανίχνευση αντικειμένων (object detection)[7]. Επισημαίνεται ότι η ραγδαία πρόοδος των νευρωνικών δικτύων επιτρέπει πλέον την επίτευξη υψηλής ακρίβειας σε πολύπλοκους οπτικούς προβληματισμούς. Στόχος αυτού του κεφαλαίου είναι η παρουσίαση των βασικών εννοιών της μηχανικής μάθησης και της όρασης υπολογιστών, καθώς και των σύγχρονων τεχνικών και συστημάτων αναγνώρισης, ιδίως σε φορητές συσκευές.

2.2 Βασικές Έννοιες: Μηχανική Μάθηση, Νευρωνικά Δίκτυα

Η μηχανική μάθηση (machine learning, ML) αποτελεί κύρια κατηγορία της ΤΝ: είναι ένα σύνολο αλγορίθμων που επιτρέπουν στο λογισμικό να «μαθαίνει» από δεδομένα και να προβλέπει αποτελέσματα χωρίς ρητό προγραμματισμό[9]. Όπως αναφέρουν πρόσφατες μελέτες, η μηχανική μάθηση βασίζεται σε ιστορικά δεδομένα ώστε να κατηγοριοποιεί ή να προβλέπει νέες τιμές[9]. Οι νευρωνικές δομές επιτρέπουν την απόδοση της μάθησης σε πολλά στρώματα συνδεδεμένων «νευρώνων». Ένα τεχνητό νευρωνικό δίκτυο (ANN) μιμείται εν μέρει τον ανθρώπινο εγκέφαλο, με στρώσεις εισόδου, κρυφές και εξόδου όπου κάθε κόμβος («νευρώνας») έχει βάρος και κατώφλι ενεργοποίησης[8]. Κατά τη διάρκεια της εκπαίδευσης, οι σύνδεσμοι (βάρη) προσαρμόζονται ανάλογα με τα παρεχόμενα δεδομένα, επιτρέποντας στο δίκτυο να μαθαίνει σχέσεις εισόδου–εξόδου[8]. Τεχνικές βαθιάς μάθησης (deep learning) χρησιμοποιούν πολυεπίπεδα ANN για την επίλυση σύνθετων προβλημάτων, αξιοποιώντας την ικανότητα των δικτύων να προσεγγίζουν μη-γραμμικές συναρτήσεις και να αναγνωρίζουν πολύπλοκα μοτίβα[9],[8].



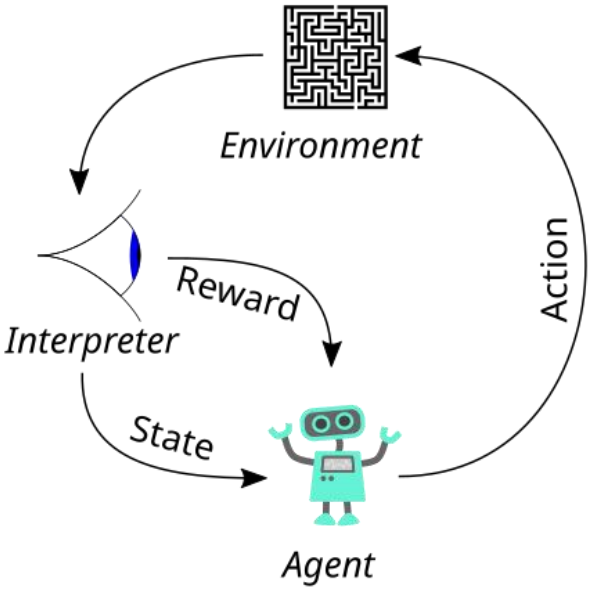
Σχήμα 2.1: Αρχιτεκτονική ενός τεχνητού νευρωνικού δικτύου με είσοδο, κρυφά επίπεδα και έξοδο.[26]

2.3 Εποπτευόμενη, Μη Εποπτευόμενη και Ενισχυτική Μάθηση

Οι κύριες κατηγορίες μάθησης που χρησιμοποιούνται σήμερα είναι:

- **Εποπτευόμενη μάθηση (Supervised Learning):** Το μοντέλο εκπαιδεύεται σε σετ δεδομένων για το οποίο υπάρχουν έτοιμες ετικέτες ή τιμές-στόχοι. Στο διάστημα εκπαίδευσης, το σύστημα εντοπίζει σχέσεις εισόδου–εξόδου μέσα από τα παρεχόμενα δεδομένα, ώστε να προβλέπει την σωστή ετικέτα για νέες παρατηρήσεις. Ο στόχος είναι η συστηματική ταξινόμηση ή πρόβλεψη ανάλογα με όσα «έμαθε» από τα δεδομένα εκπαίδευσης[9].
- **Μη εποπτευόμενη μάθηση (Unsupervised Learning):** Αντιθέτως, δεν απαιτούνται επισημασμένα δεδομένα. Το σύστημα ανακαλύπτει αυτομάτως δομές ή ομάδες μέσα στο σύνολο δεδομένων χωρίς οδηγητικές ετικέτες[9]. Μέσω αλγορίθμων clustering ή μείωσης διαστάσεων, επιτυγχάνεται εντοπισμός κρυφών συσχετίσεων ή υποκατηγοριών που ίσως δεν γίνονταν αντιληπτές σε ανθρώπινο επίπεδο[9].
- **Ενισχυτική μάθηση (Reinforcement Learning):** Σε αυτό το πλαίσιο, ένας «πράκτορας» αλληλεπιδρά με ένα περιβάλλον και μαθαίνει μέσω δοκιμών-λάθους, λαμβάνοντας *ανταμοιβές* (ή *ποινές*) για τις ενέργειές του. Μέσω επαναλήψεων, ο πράκτορας ρυθμίζει τη στρατηγική του έτσι ώστε να μεγιστοποιεί τη συνολική ανταμοιβή. Αν και η ενισχυτική μάθηση δεν στηρίζεται σε επισημασμένα δεδομένα, έχει αποδειχθεί αποτελεσματική σε προβλήματα ελέγχου, παιχνιδιών στρατηγικής και ρομποτικής.

Τα παραπάνω μοντέλα εκμάθησης απαρτίζουν τα βασικά εργαλεία στα σύγχρονα συστήματα ΤΝ[9], και συχνά συνδυάζονται μεταξύ τους ανάλογα με την εφαρμογή.



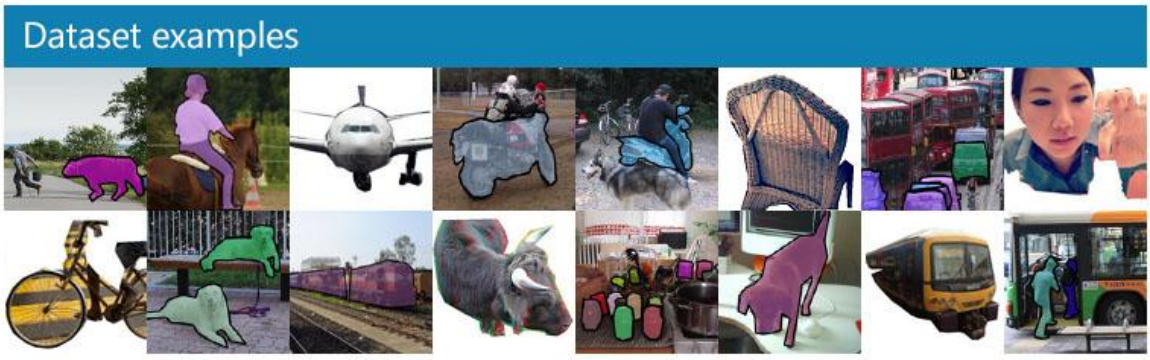
Σχήμα 2.2: Γραφική απεικόνιση ενός συστήματος Ανατροφοδοτούμενης Εκμάθησης[27]

2.4 Ανίχνευση Αντικειμένων και Υπολογιστική Όραση

Η ανίχνευση αντικειμένων είναι θεμελιώδης εργασία της υπολογιστικής όρασης: στόχος είναι η αναγνώριση όλων των σχετικών αντικειμένων σε μια εικόνα μαζί με την ακριβή θέση τους, συνήθως με ορθογώνια όρια (bounding boxes) και κλάσεις σημείων[10]. Ένα συνεπές πρόβλημα της όρασης υπολογιστών είναι ότι πρέπει να ληφθούν υπόψη τόσο η χωρική τοποθέτηση όσο και η κατηγορία των αντικειμένων, συχνά σε πραγματικό χρόνο και υπό περιορισμούς υπολογιστικής ισχύος[10]. Χάρη στις εξελίξεις της βαθιάς μάθησης, οι τεχνικές ανίχνευσης αντικειμένων έχουν γίνει βασικό κομμάτι εφαρμογών σε τομείς όπως επιτήρηση, αυτόνομα οχήματα και ιατρική απεικόνιση[10]. Για παράδειγμα, οι αλγόριθμοι YOLO («You Only Look Once») και R-CNN προσφέρουν ισχυρές λύσεις: το YOLO εισάγει μια ενιαία αρχιτεκτονική που προβλέπει ταυτόχρονα τα κουτιά και τις κλάσεις σε μία διέλευση της εικόνας[10], μεγιστοποιώντας την ταχύτητα ενώ διατηρεί ανταγωνιστική ακρίβεια.
Οι βασικές εργασίες της όρασης υπολογιστών περιλαμβάνουν, μεταξύ άλλων, τις εξής:

- Κατηγοριοποίηση εικόνων (image classification): Ανάθεση ετικέτας σε ολόκληρη την εικόνα[7].
- Εντοπισμός θέσης και ανίχνευση αντικειμένων (localization & object detection): Βρίσκει και επισημαίνει αντικείμενα με πλαίσια και ετικέτες[7],[10].
- Τμηματοποίηση (segmentation): Ομαδοποίηση των pixel που ανήκουν στο ίδιο αντικείμενο ή κατηγορία[10].

Σήμερα, οι αλγόριθμοι YOLO θεωρούνται ιδιαίτερα σημαντικοί για το ρεαλιστικό-χρόνου object detection λόγω της απλής και γρήγορης προσέγγισής τους[10]. Η επιτυχία των YOLO απέδειξε την αξία των end-to-end νευρωνικών μοντέλων που συνδυάζουν εντοπισμό και ταξινόμηση σε ενιαίο βήμα, δίνοντας έμφαση στην αποτελεσματικότητα και την κλιμάκωση για πραγματικές εφαρμογές.



Εικόνα 2.1: Παράδειγμα εικόνας από το dataset COCO[28]

2.5 Συστήματα Αναγνώρισης σε Κινητές Συσκευές

Η όραση υπολογιστών επεκτείνεται πλέον και σε φορητές και ενσωματωμένες συσκευές (smartphones, drones, IoT συσκευές κ.ά.), αλλά και σε συστήματα edge computing. Ωστόσο, οι περιορισμοί πόρων (επεξεργαστική ισχύς, μνήμη, ενέργεια) δημιουργούν σημαντικές προκλήσεις[11]. Για τον λόγο αυτό, οι ερευνητές έχουν αναπτύξει εξειδικευμένα, *ελαφριά* νευρωνικά δίκτυα που στοχεύουν στην αποδοτική εκτέλεση σε κινητές συσκευές. Παραδείγματα τέτοιων μοντέλων είναι τα **MobileNet** και **Tiny YOLO**, τα οποία χρησιμοποιούν επαναληπτικές/διπλές συνελίξεις και μικρότερο πλήθος παραμέτρων ώστε να μειώνουν την υπολογιστική πολυπλοκότητα ενώ διατηρούν ικανοποιητική ακρίβεια[11]. Επιπλέον, εφαρμόζονται τεχνικές συμπίεσης μοντέλων, όπως η ποσοτικοποίηση (quantization) και η προτρύπηση (pruning), για μείωση του μεγέθους των δικτύων και ταχύτερη εκτέλεση. Χρησιμοποιούνται επίσης επιταχυντές ειδικού σκοπού (ενσωματωμένες GPU, AI cores, FPGA) και ελαφριά frameworks (π.χ. TensorFlow Lite, TensorRT) που βελτιστοποιούν τη ροή υπολογισμού στην κινητή συσκευή[11]. Το αποτέλεσμα είναι συστήματα αναγνώρισης που μπορούν να λειτουργήσουν σε πραγματικό χρόνο (π.χ. επιτόπιες εφαρμογές επαυξημένης πραγματικότητας, βιομετρική ταυτοποίηση προσώπου, αυτόνομα ρομπότ) παρόλο που το διαθέσιμο υλικό είναι περιορισμένο[11].

2.6 Επίλογος

Συνοψίζοντας, η συνδυαστική χρήση μηχανικής μάθησης και νευρωνικών δικτύων έχει φέρει επανάσταση στον χώρο της όρασης υπολογιστών. Οι βασικές έννοιες που παρουσιάστηκαν (κατηγορίες μάθησης, αρθρώσεις ANN) αποτελούν τη θεμελιώδη βάση για σύγχρονες εφαρμογές CV[9]. Οι πρόσφατες προόδους στα συστήματα αντικειμενο-εντοπισμού, όπως τα μοντέλα YOLO, επιτρέπουν αξιόπιστη αναγνώριση και εντοπισμό αντικειμένων με υψηλή ακρίβεια και ταχύτητα[9]. Παράλληλα, η ενσωμάτωση αυτών των αλγορίθμων σε φορητές πλατφόρμες και edge συσκευές είναι εφικτή χάρη σε ελαφριά αρχιτεκτονική και τεχνικές βελτιστοποίησης[11]. Το κεφάλαιο αυτό έδειξε ότι οι σύγχρονες προσεγγίσεις στην ΤΝ και την Όραση Υπολογιστών είναι ελκυστικά εργαλεία για λύση πολλών πρακτικών προβλημάτων, ανοίγοντας το δρόμο για περαιτέρω έρευνα στον συνεχώς αναπτυσσόμενο τομέα.

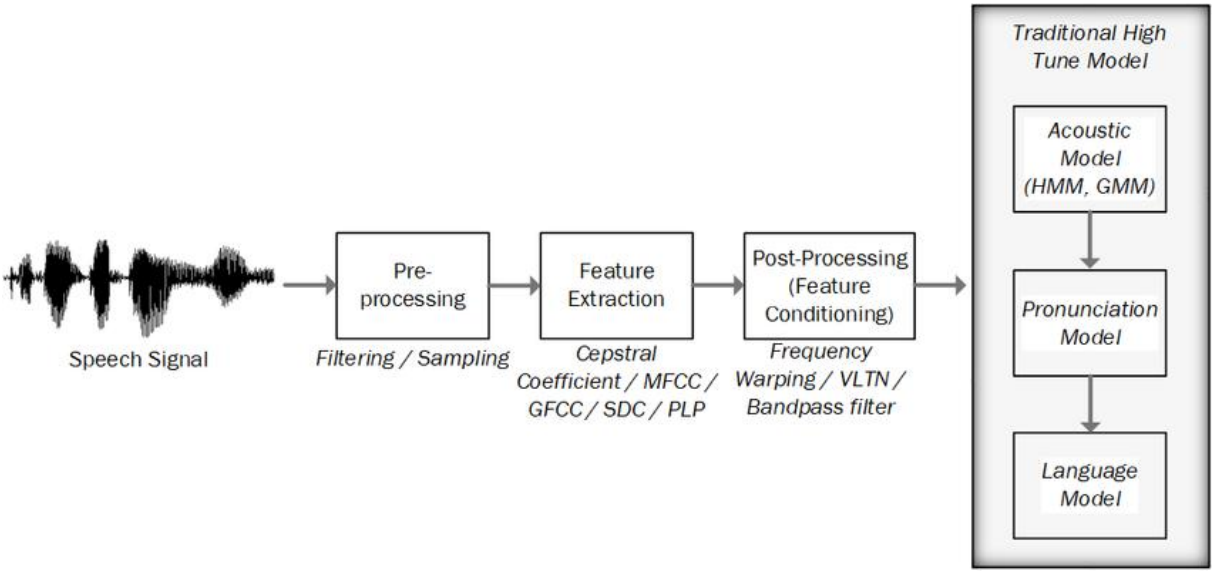
Κεφάλαιο 3ο: Φωνητική και Απτική Αλληλεπίδραση

3.1 Εισαγωγή

Οι σύγχρονες διεπαφές ανθρώπου-υπολογιστή εκμεταλλεύονται ολοένα και περισσότερο τη φωνητική αναγνώριση και τις απτικές μορφές ανάδρασης για να υποστηρίξουν τη μη-οπτική αλληλεπίδραση. Για χρήστες με προβλήματα όρασης, οι τεχνολογίες αυτές (π.χ. βοηθήματα φωνητικής πλοήγησης ή συσκευές Braille) ενσωματώνονται σε συστήματα όπως αναγνώστες οθόνης, κινητά και έξυπνες συσκευές. Οι τελευταίες δεκαετίες έφεραν ραγδαία πρόοδο στην αυτόματη αναγνώριση ομιλίας (ASR) χάρη σε συστήματα βαθιάς μάθησης όπως τα μοντέλα Transformer. Ταυτόχρονα, η απτική σχεδίαση – δηλαδή η χρήση δονήσεων και άλλων αφής-βασισμένων σημάτων – καθίσταται όλο και πιο καθοριστική: ο συνδυασμός οπτικής (όταν υπάρχει), φωνητικής και απτικής ανάδρασης έχει αποδειχθεί κρίσιμος στην προσβασιμότητα και ανεξαρτησία των τυφλών χρηστών. Στο κεφάλαιο αυτό εξετάζονται σύγχρονες μελέτες και τεχνολογίες φωνητικής αναγνώρισης και συστήματος απόκρισης (συμπεριλαμβανομένων των Text-to-Speech), καθώς και οι αρχές σχεδίασης χειρονομιών (π.χ. διπλό πάτημα) και απτικής ανάδρασης σε πλαίσια προσβασιμότητας.

3.2 Φωνητική Αναγνώριση και Αντίδραση Συστήματος

Η φωνητική αναγνώριση (automatic speech recognition, ASR) μετατρέπει την ανθρώπινη ομιλία σε κείμενο μέσω προηγμένων αλγορίθμων. Πρόσφατες έρευνες επισημαίνουν ότι η ενσωμάτωση τεχνικών βαθιάς μάθησης (π.χ. Conformer, transformer-based μοντέλα) έχει βελτιώσει σημαντικά την ακρίβεια και ταχύτητα των ASR συστημάτων. Αυτές οι προόδοι επιτρέπουν σε ψηφιακούς βοηθούς και εφαρμογές να κατανοούν πολύπλοκες εντολές ομιλίας και να λειτουργούν σε πραγματικό χρόνο. Ως απόκριση, το σύστημα συχνά παρέχει ηχητικά μηνύματα μέσω σύνθεσης ομιλίας (Text-to-Speech, TTS). Τα σύγχρονα TTS συστήματα, ειδικά αυτά βασισμένα σε νευρωνικά δίκτυα, παράγουν σχεδόν ανθρωποειδές αποτέλεσμα ομιλίας υψηλής ποιότητας. Μετατρέπουν κείμενο σε προφορικό λόγο, διευκολύνοντας την ακουστική πρόσβαση στο ψηφιακό περιεχόμενο για άτομα με προβλήματα όρασης. Για παράδειγμα, αναγνώστες οθόνης ενσωματώνουν TTS για να «διαβάζουν» το περιεχόμενο της οθόνης, επιτρέποντας στον χρήστη να πλοηγηθεί σε εφαρμογές, να διαβάσει κείμενα και να αλληλεπιδράσει με ψηφιακούς πόρους χωρίς οπτική συμβολή. Πολλές σύγχρονες φωνητικές διεπαφές παρέχουν επίσης άμεση φωνητική ανάδραση και εννοιολογικές εντολές. Μελέτες δείχνουν ότι οι πιο δημοφιλείς λειτουργίες σε διεπαφές φωνητικής αναγνώρισης είναι η ηχητική επιβεβαίωση (vocal feedback) και η χρήση ενστικτωδών φωνητικών εντολών. Με άλλα λόγια, μετά την αναγνώριση της φωνής, το σύστημα συνήθως απαντά λεκτικά ή ενεργοποιεί μια προκαθορισμένη ενέργεια, σύμφωνα με καθορισμένα πρότυπα αλληλεπίδρασης.



Σχήμα 3.1: Τυπικό pipeline ενός συστήματος αυτόματης αναγνώρισης ομιλίας (ASR)[29]

3.3 Διπλό Πάτημα (Double Tap) και Απτική Σχεδίαση

Στην απτική διεπαφή κινητών συσκευών, το «διπλό πάτημα» αποτελεί βασική χειρονομία για την επιβεβαίωση/ενεργοποίηση επιλογών. Π.χ. στα συστήματα ανάγνωσης οθόνης, ένας τυπικός χειρισμός είναι να μετακινηθεί ο δείκτης σε ένα αντικείμενο με κύλιση και έπειτα να πραγματοποιηθεί διπλό πάτημα για να εκτελεστεί ή να ανοίξει η εν λόγω επιλογή. Συγκεκριμένα, το διπλό πάτημα επιτρέπει στον χρήστη να «βυθιστεί» σε βαθύτερο επίπεδο πληροφοριών (drill-down) ή να επιβεβαιώσει ενέργειες. Για πιο σύνθετες ενέργειες, συχνά συνδυάζεται με παρατεταμένο πάτημα (double tap & hold). Ως παράδειγμα, σε εφαρμογές προσβασιμότητας τύπου ChartA11y, όλες οι γρήγορες πλοηγήσεις (swipe) ξεκινούν με διπλό πάτημα-παραμονή ακολουθούμενο από κίνηση προς κάποια κατεύθυνση, ώστε να πηγαίνει ο χρήστης σε διαφορετικές περιοχές ή επίπεδα πληροφορίας. Έτσι, οι χρήστες δεν χρειάζεται να θυμούνται πολλές διαφορετικές χειρονομίες, καθώς όλες ξεκινούν με την ίδια αρχική κίνηση (διπλό πάτημα με κράτημα)[13].

Εικόνα 3.1: Εικονίδιο που απεικονίζει τη χειρονομία “double tap”[30]

Παράλληλα, η **απτική σχεδίαση** υποστηρίζει την επικοινωνία με τον χρήστη μέσω δονήσεων και άλλων αισθητικών ερεθισμάτων. Οι κινητές συσκευές διαθέτουν μηχανισμούς δόνησης που μπορούν να παρέχουν διαφορετικά μοτίβα ανάδρασης π.χ. μικρές δονήσεις κατά το πάτημα κουμπιών ή έντονες

δονήσεις σε σημαντικά γεγονότα. Μελέτες δείχνουν ότι η προσθήκη απτικής ανάδρασης σε εφαρμογές για τυφλούς χρήστες βελτιώνει σημαντικά την εμπειρία χρήσης: σε ένα παράδειγμα εφαρμογής με αναγνώριση αντικειμένων, η ενεργοποίηση απτικών σημάτων αύξησε το επίπεδο ικανοποίησης των χρηστών και διευκόλυνε την πλοήγηση στο περιβάλλον. Γενικά, η απτική ανάδραση προσφέρει άμεση επιβεβαίωση των ενεργειών χωρίς να απαιτείται οπτική επιβεβαίωση, γεγονός ιδιαίτερα πολύτιμο για χρήστες με προβλήματα όρασης. Οι πρόσφατες ανασκοπήσεις επισημαίνουν ότι, συγκριτικά με την καθαρά ηχητική πληροφόρηση, η απτική παρέχει έναν πιο διαισθητικό τρόπο κατανόησης γεωμετρικών και περιεχομένων χωρικών στοιχείων. Έτσι, οι σχεδιαστές προσθέτουν δονητικά σήματα (π.χ. χρωματικά μοτίβα, παρατεταμένες δονήσεις) για να υπογραμμίζουν καταστάσεις (π.χ. σφάλμα, νέο στοιχείο) ή να κατευθύνουν τη συλλογή πληροφοριών μέσα από την αφή.

Σύμφωνα με πρόσφατες έρευνες, η συνδυαστική χρήση **gestures** και απτικής ανάδρασης βελτιώνει σημαντικά την εμπειρία χρήστη σε ομάδες ατόμων με οπτική αναπηρία, καθώς μειώνει τον χρόνο εκμάθησης και αυξάνει την ακρίβεια εκτέλεσης εντολών [15], [16]. Επιπλέον, η απόκριση της απτικής ανάδρασης πρέπει να είναι χρονικά συγχρονισμένη με τη χειρονομία, καθώς ακόμη και μικρές καθυστερήσεις μπορούν να δημιουργήσουν αβεβαιότητα και μειωμένη αποτελεσματικότητα [17].

Πίνακας 3.1: Ορολογίες απτικής αλληλεπίδρασης [31]

ιδιοδεκτική αίσθηση	Μας επιτρέπει να γνωρίζουμε τη θέση και την κίνηση του σώματός μας στο χώρο, ακόμα και με κλειστά μάτια. Χάρη στην ιδιοδεκτική αίσθηση, μπορούμε να συντονίσουμε τις κινήσεις μας, να διατηρήσουμε την ισορροπία μας και να πιάσουμε αντικείμενα με ακρίβεια.
Αιθουσαία αίσθηση	Είναι υπεύθυνη για την αίσθηση της ισορροπίας και του προσανατολισμού. Χάρη στην αιθουσαία αίσθηση, μπορούμε να αντιληφθούμε την κίνηση του κεφαλιού μας, να διατηρήσουμε την ισορροπία μας και να αποφύγουμε την ζαλάδα.
Κιναισθητική αίσθηση	Μας επιτρέπει να νιώθουμε την τάση στους μυς και τις αρθρώσεις μας, καθώς και την κίνηση των μελών του σώματός μας. Χάρη στην κιναισθητική αίσθηση, μπορούμε να ελέγξουμε την δύναμη που ασκούμε, να σηκώσουμε αντικείμενα και να αλληλεπιδράσουμε με το περιβάλλον με ακρίβεια.
Δερματική αίσθηση	Είναι η αίσθηση που προέρχεται από το δέρμα μας και μας επιτρέπει να νιώθουμε την πίεση, τη θερμοκρασία και τον πόνο. Χάρη στην δερματική αίσθηση, μπορούμε να αποφύγουμε τραυματισμούς, να αντιληφθούμε την υφή των αντικειμένων και να βιώσουμε συναισθήματα αφής.
Απτική αίσθηση	Είναι η λεπτή αίσθηση αφής που μας επιτρέπει να διακρίνουμε τα σχήματα, τις υφές και τις λεπτομέρειες των αντικειμένων. Χάρη στην απτική αίσθηση, μπορούμε να διαβάσουμε

	Braille, να γράψουμε με στυλό και να νιώσουμε την απαλότητα ενός υφάσματος.
Ανάδραση δύναμης	Είναι η αίσθηση της αντίθετης δύναμης που ασκείται σε ένα αντικείμενο. Χάρη στην ανάδραση δύναμης, μπορούμε να ελέγξουμε πόση δύναμη ασκείται, να χειριστούμε εύθραυστα αντικείμενα και να πιάσουμε αντικείμενα με ακρίβεια.

3.4 Προσβασιμότητα και UX για Τυφλούς Χρήστες

Η ενσωμάτωση των φωνητικών και απτικών τεχνολογιών καθιστά ευκολότερη την πλοήγηση και αλληλεπίδραση των τυφλών χρηστών με ψηφιακά συστήματα. Σε πρακτικό επίπεδο, ο χρήστης αναγνωστικού λογισμικού (screen reader) κινεί το δάκτυλό του κατά μήκος της οθόνης (finger reading), ενώ η εφαρμογή ανακοινώνει με φωνητικά μηνύματα το περιεχόμενο κάτω από το δάκτυλο. Όταν επιλεγεί ένα στοιχείο, η ενέργεια επιβεβαιώνεται με διπλό πάτημα και συχνά συνοδεύεται από σύντομη δόνηση ή άλλο απτικό σήμα. Με τον τρόπο αυτό, ο χρήστης λαμβάνει διαρκή ηχητική και απτική ανατροφοδότηση καθ' όλη τη διάρκεια της πλοήγησης. Σύμφωνα με βιβλιογραφικές ανασκοπήσεις, οι εφαρμογές φωνητικής αναγνώρισης που απευθύνονται σε χρήστες με προβλήματα όρασης αναπτύσσονται συνήθως για φορητές συσκευές (κινητά), επιτραπέζιες (desktop) ή διαδικτυακές πλατφόρμες[12]. Αυτό δείχνει την ευρεία υιοθέτηση των φωνητικών διεπαφών σε πολλαπλά περιβάλλοντα και την ανάγκη διαμόρφωσης βέλτιστων πρακτικών σχεδίασης σε κάθε περίπτωση. Στην πράξη, η σχεδίαση UX για τυφλούς ακολουθεί βασικές αρχές προσβασιμότητας: καθαρή διάταξη με λίγα στοιχεία ανά οθόνη, ξεκάθαρες φωνητικές οδηγίες, δυνατότητα ρύθμισης ταχύτητας ομιλίας και έντασης, καθώς και ομαλές μεταβάσεις μεταξύ οθονών. Παράλληλα, όπως επισημαίνεται σε πρόσφατες μελέτες (π.χ. ChartA11y), η ενοποίηση των εισόδων (π.χ. όλα τα «χρηστικά» γρήγορα gestures να ξεκινούν με διπλό πάτημα) μειώνει το γνωστικό φορτίο των χρηστών και κάνει την πλοήγηση πιο διαισθητική. Τέλος, η ποιοτική αξιολόγηση τέτοιων συστημάτων δείχνει ότι οι πολυτροπικοί (φωνητικοί + απτικοί) οδηγοί επιτρέπουν στους τυφλούς χρήστες να εξερευνούν το περιεχόμενο πιο ανεξάρτητα και με μεγαλύτερη αυτοπεποίθηση.

3.5 Επίλογος

Συνοψίζοντας, οι σύγχρονες τεχνολογίες φωνητικής αναγνώρισης και TTS καθώς και οι απτικές διεπαφές έχουν καταστεί αναπόσπαστο μέρος του σχεδιασμού προσβάσιμων συστημάτων. Η ραγδαία βελτίωση στα ASR μοντέλα (μέσω DL και Transformers) και στα νευρωνικά TTS προσφέρει πλέον φυσική, ανθρωποειδή ομιλία με χαμηλή υστέρηση. Παράλληλα, η στρατηγική χρήση χειρονομιών, όπως το διπλό πάτημα, μαζί με δονητικά σήματα, καθιστούν την αλληλεπίδραση των τυφλών χρηστών αποτελεσματικότερη και διαισθητική. Η βιβλιογραφία μετά το 2020 υπογραμμίζει σταθερά ότι ο σωστός συνδυασμός φωνής και αφής αυξάνει την ανεξαρτησία και την ευχρηστία για άτομα με προβλήματα όρασης. Η συνεχιζόμενη έρευνα στο πεδίο αυτό εστιάζει στην προσαρμογή των αλγορίθμων ASR σε περιβάλλοντα με θόρυβο, στη βελτίωση του TTS για πιο ζωντανή απόδοση και

στην ανάπτυξη ποιοτικότερων απτικών συσκευών – με στόχο τη βέλτιστη υποστήριξη της προσβασιμότητας και της εμπειρίας χρήστη.

Κεφάλαιο 4ο: Σχετικές εργασίες

4.1 Εισαγωγή

Το κεφάλαιο αυτο εξετάζει σχετικές εργασίες στους τομείς της τεχνολογίας ανίχνευσης αντικειμένων και των βοηθητικών εφαρμογών για χρήστες με προβλήματα όρασης. Η έρευνα σε αυτούς τους τομείς έχει αναπτυχθεί τα τελευταία χρόνια, κυρίως λόγω της προόδου στην τεχνητή νοημοσύνη και στην επεξεργασία εικόνας σε πραγματικό χρόνο. Μέσα από τη μελέτη της βιβλιογραφίας παρουσιάζονται τεχνολογικές προσεγγίσεις που αξιοποιούν αλγορίθμους βαθιάς μάθησης για την αναγνώριση αντικειμένων και την παροχή άμεσης πληροφόρησης στον χρήστη.

4.2 Σχετικές εργασίες

Η ανάπτυξη υποστηρικτικών τεχνολογιών, και ιδιαίτερα εκείνων που στοχεύουν στη βελτίωση της ανεξαρτησίας και της ποιότητας ζωής των ατόμων με προβλήματα όρασης, έχει αποκτήσει σημαντική σημασία τα τελευταία χρόνια. Καθώς η τεχνολογία συνεχίζει να εξελίσσεται, η ενσωμάτωση της τεχνητής νοημοσύνης (AI) και της μηχανικής μάθησης σε αυτές τις τεχνολογίες έχει ανοίξει νέες πόρτες για καινοτόμες εφαρμογές. Ένας από τους πιο σημαντικούς τομείς εστίασης ήταν η χρήση τεχνολογιών ανίχνευσης αντικειμένων, προκειμένου να βοηθηθούν τα άτομα με προβλήματα όρασης να αναγνωρίζουν το περιβάλλον τους και να κινούνται μέσα σε αυτό.

Η ανίχνευση αντικειμένων, η οποία περιλαμβάνει την αναγνώριση και την ταξινόμηση αντικειμένων μέσα σε μια εικόνα ή ροή βίντεο, έχει εξελιχθεί μέσω της εφαρμογής μοντέλων βαθιάς μάθησης όπως το EfficientDet-lite2, το οποίο είναι γνωστό για την αποτελεσματικότητά του σε κινητές συσκευές [32]. Αυτές οι τεχνολογίες έχουν εκπαιδευτεί χρησιμοποιώντας μεγάλα σύνολα δεδομένων όπως το σύνολο δεδομένων COCO (Common Objects in Context)[33], το οποίο περιέχει μια μεγάλη ποικιλία από αντικείμενα με ετικέτες που συναντώνται συνήθως στην καθημερινή ζωή. Ενσωματώνοντας αυτά τα συστήματα ανίχνευσης αντικειμένων σε εφαρμογές για κινητά, οι προγραμματιστές μπόρεσαν να δημιουργήσουν προσβάσιμα εργαλεία που προσφέρουν αναγνώριση αντικειμένων σε πραγματικό χρόνο, επιτρέποντας στους χρήστες με προβλήματα όρασης να αλληλεπιδρούν καλύτερα και με μεγαλύτερη αυτοπεποίθηση με το περιβάλλον τους. Ταυτόχρονα, έχει δοθεί αυξανόμενη έμφαση στον σχεδιασμό προσβάσιμων διεπαφών και στην ενσωμάτωση πολυτροπικών συστημάτων ανατροφοδότησης —όπως η απτική και ακουστική ανατροφοδότηση— σε υποστηρικτικές τεχνολογίες. Η απτική ανατροφοδότηση, ειδικότερα, έχει αποδειχθεί σημαντικό χαρακτηριστικό για τους χρήστες με προβλήματα όρασης, παρέχοντας άμεση απτική επιβεβαίωση όταν αλληλεπιδρούν με ψηφιακά ή φυσικά στοιχεία[34]. Αυτή έχει γίνει μια ολοένα και πιο δημοφιλής προσέγγιση σε εφαρμογές που έχουν σχεδιαστεί για να βοηθούν άτομα με προβλήματα όρασης, καθώς επιτρέπει στους χρήστες να επιβεβαιώνουν τις αλληλεπιδράσεις τους χωρίς να βασίζονται σε οπτική ανατροφοδότηση. Ομοίως, οι ηχητικές ειδοποιήσεις προσφέρουν ένα επιπλέον επίπεδο προσβασιμότητας, επιτρέποντας στους χρήστες να λαμβάνουν λεκτικά σήματα σχετικά με το περιβάλλον ή τις ενέργειές τους. Συνδυάζοντας τόσο την απτική όσο και την ακουστική ανατροφοδότηση, οι προγραμματιστές μπορούν να δημιουργήσουν μια πιο ολοκληρωμένη και συμπεριληπτική εμπειρία χρήστη, διασφαλίζοντας ότι οι χρήστες με προβλήματα όρασης λαμβάνουν συμπληρωματικές ενδείξεις σχετικά με το περιβάλλον τους.

Αρκετές μελέτες έχουν καταδείξει την αξία του συνδυασμού της ανίχνευσης αντικειμένων που βασίζεται στην Τεχνητή Νοημοσύνη με σχέδια προσβάσιμων διεπαφών. Για παράδειγμα, οι φορητές συσκευές και οι εφαρμογές για κινητά που ενσωματώνουν μηχανισμούς ανίχνευσης αντικειμένων και ανατροφοδότησης σε πραγματικό χρόνο έχουν δείξει μεγάλες δυνατότητες στο να διευκολύνουν τους χρήστες να πλοηγούνται τόσο σε οικεία όσο και σε άγνωστα περιβάλλοντα. Η έρευνα έχει επίσης επισημάνει τη σημασία της προσαρμογής αυτών των τεχνολογιών για την κάλυψη των συγκεκριμένων αναγκών των ατόμων με προβλήματα όρασης[34]. Αυτό περιλαμβάνει τον σχεδιασμό εφαρμογών που δεν είναι μόνο ακριβείς και αποτελεσματικές στις δυνατότητες ανίχνευσης, αλλά και διαισθητικές και εύχρηστες, με σαφή και συνεπή ανατροφοδότηση. Επίσης, η χρήση πλατφορμών για κινητές συσκευές για την παροχή αυτών των λύσεων έχει αποδειχθεί ιδιαίτερα ωφέλιμη, καθώς τα smartphones είναι ευρέως προσβάσιμα και διαθέτουν την απαραίτητη υπολογιστική ισχύ για την εκτέλεση εξελιγμένων μοντέλων μηχανικής μάθησης.

Ο Kuriakose et al.[35] παρουσιάζουν μια ανασκόπηση συστημάτων πολυτροπικής πλοήγησης που έχουν σχεδιαστεί για να βοηθούν χρήστες με προβλήματα όρασης. Η εργασία εξετάζει διάφορες προσεγγίσεις που συνδυάζουν αισθητηριακές μεθόδους, όπως ακουστικές, απτικές και οπτικές ενδείξεις, για τη βελτίωση της πλοήγησης σε εσωτερικά και εξωτερικά περιβάλλοντα. Οι συγγραφείς αναλύουν τα δυνατά και τα περιοριστικά σημεία αυτών των συστημάτων και προτείνουν συγκεκριμένα χαρακτηριστικά που μπορούν να βελτιώσουν τα συστήματα πολυτροπικής πλοήγησης σε σχέση με την προσβασιμότητά τους, ενώ ταυτόχρονα τονίζουν τη σημασία της Τεχνητής Νοημοσύνης για την αντιμετώπιση των διαφόρων προκλήσεων στα πολυτροπικά συστήματα.

Επίσης, ο Kuriakose et al.[36] ανέπτυξαν το DeepNAVI, ένα σύστημα υποβοήθησης πλοήγησης για τυφλούς χρησιμοποιώντας ένα smartphone και ένα σετ ακουστικών οστικής αγωγιμότητας, ενσωματώνοντας ανίχνευση εμποδίων, απόστασης, σκηνής, θέσης και κίνησης για την ενίσχυση της χωρικής επίγνωσης χωρίς απώλεια της επίγνωσης της κατάστασης. Αυτό το σύστημα εξασφαλίζει φορητότητα και ευκολία, παρέχοντας παράλληλα βοηθήματα πλοήγησης σε χρήστες με προβλήματα όρασης.

Ο Rakkolainen et al.[37] διεξήγαγαν μια ανασκόπηση σχετικά με τις τεχνολογίες που επιτρέπουν την πολυτροπική αλληλεπίδραση στην εκτεταμένη πραγματικότητα (XR), συμπεριλαμβανομένης της εικονικής και της επαυξημένης πραγματικότητας, με έμφαση στην προσβασιμότητα. Η ανασκόπηση υπογραμμίζει τον τρόπο με τον οποίο οι πολυτροπικές διεπαφές, συμπεριλαμβανομένης της απτικής και ακουστικής ανατροφοδότησης, μπορούν να χρησιμοποιηθούν για να κάνουν τα περιβάλλοντα XR πιο προσβάσιμα σε άτομα με προβλήματα όρασης. Η εργασία δίνει έμφαση στις δυνατότητες χρήσης τεχνολογιών XR ως εργαλεία για βελτιωμένη αντίληψη για χρήστες με προβλήματα όρασης.

Ο Jiménez et al.[38] ανέπτυξαν μια βοηθητική συσκευή μετακίνησης που παρέχει απτική ανατροφοδότηση για την καθοδήγηση ατόμων με προβλήματα όρασης. Η συσκευή χρησιμοποιεί αισθητήρες για την ανίχνευση εμποδίων και προσφέρει απτική ανατροφοδότηση σε πραγματικό χρόνο για να βοηθήσει τους χρήστες να πλοηγούνται με ασφάλεια. Η μελέτη υπογραμμίζει πώς η απτική ανατροφοδότηση μπορεί να βελτιώσει την επίγνωση του φυσικού περιβάλλοντος και την αυτονομία για άτομα με προβλήματα όρασης, παρέχοντας ακριβή απτική καθοδήγηση.

See et al.[39] ανέπτυξαν ένα φορετό σύστημα απτικής ανατροφοδότησης σχεδιασμένο για να βοηθά άτομα με προβλήματα όρασης και τυφλά άτομα να εντοπίζουν εμπόδια σε πολλαπλές περιοχές γύρω τους. Το σύστημα χρησιμοποιεί αισθητήρες για την παρακολούθηση του περιβάλλοντος και παρέχει στοχευμένη απτική ανατροφοδότηση για να ειδοποιεί τους χρήστες για εμπόδια σε διαφορετικές κατευθύνσεις, βελτιώνοντας την χωρική τους επίγνωση και κινητικότητά τους.

Ο Palani et al.[40] διεξήγαγαν μια αξιολόγηση συμπεριφοράς που συνέκρινε την εκμάθηση χαρτών μεταξύ οπτικών και απτικών οθονών που βασίζονται σε οθόνες αφής, στοχεύοντας τόσο σε τυφλούς όσο και σε χρήστες με προβλήματα όρασης. Η μελέτη αξιολόγησε πόσο αποτελεσματικά τα άτομα με προβλήματα όρασης μπορούν να μάθουν και να πλοηγηθούν σε χάρτες χρησιμοποιώντας απτική ανατροφοδότηση σε σύγκριση με τις παραδοσιακές οπτικές μεθόδους. Τα ευρήματα υποδηλώνουν ότι οι απτικές οθόνες είναι ιδιαίτερα αποτελεσματικές για τη μεταφορά χωρικών πληροφοριών, προσφέροντας μια προσβάσιμη εναλλακτική λύση για την εκμάθηση χαρτών για χρήστες με προβλήματα όρασης.

Ο Raynal et al.[41] παρουσίασε το FlexiBoard, μια φορητή πολυτροπική συσκευή που παρέχει απτική και απτή ανατροφοδότηση για να βοηθήσει τους χρήστες με προβλήματα όρασης στην ερμηνεία γραφικών και χωρικών πληροφοριών. Το σύστημα υποστηρίζει αλληλεπίδραση μέσω αφής, επιτρέποντας στους χρήστες να εξερευνήσουν φυσικές αναπαραστάσεις γραφικών ενώ λαμβάνουν ακουστική ανατροφοδότηση για επιπλέον περιεχόμενο. Η μελέτη καταδεικνύει την αποτελεσματικότητα των απτικών γραφικών στην ενίσχυση της μάθησης και της προσβασιμότητας για άτομα με προβλήματα όρασης.

Ο Νικόλαος Τζίμος κ.ά.[42] εξέτασαν την ενσωμάτωση της απτικής ανατροφοδότησης σε εικονικά περιβάλλοντα για την ενίσχυση της αλληλεπίδρασης των χρηστών, ωφελώντας ιδιαίτερα τα άτομα με προβλήματα όρασης κατά την εξερεύνηση του απτικού διαδικτύου.

Οι λειτουργίες προσβασιμότητας στις εφαρμογές για κινητά είναι κρίσιμες για να διασφαλιστεί ότι οι χρήστες με προβλήματα όρασης μπορούν να αλληλεπιδρούν αποτελεσματικά με το ψηφιακό περιεχόμενο. Αυτές οι λειτουργίες συχνά περιλαμβάνουν αναγνώστες οθόνης, απτική ανατροφοδότηση και τεχνολογίες αναγνώρισης φωνής. Οι πρόσφατες εξελίξεις έχουν οδηγήσει στην ενσωμάτωση της τεχνητής νοημοσύνης και της μηχανικής μάθησης για την περαιτέρω βελτίωση της προσβασιμότητας.

Ο Billi et al.[43] πρότεινε μια μεθοδολογία για την αξιολόγηση της προσβασιμότητας και της χρηστικότητας των εφαρμογών για κινητά, συνδυάζοντας τις αρχές σχεδιασμού με επίκεντρο τον χρήστη με τις οδηγίες προσβασιμότητας. Η μελέτη τους δείχνει πώς μια συστηματική προσέγγιση στην αξιολόγηση μπορεί να βελτιώσει τόσο τη χρηστικότητα όσο και την προσβασιμότητα των εφαρμογών για κινητά, διασφαλίζοντας ότι καλύπτουν τις ανάγκες των χρηστών με αναπηρίες. Οι Ballantyne et al.[44] εξέτασαν τις υπάρχουσες οδηγίες προσβασιμότητας για εφαρμογές για κινητά, αναλύοντας πόσο καλά εφαρμόζονται αυτές οι οδηγίες στον σχεδιασμό εφαρμογών για κινητά. Η μελέτη αξιολογεί τη συμμόρφωση με τα πρότυπα προσβασιμότητας σε διάφορες εφαρμογές, επισημαίνοντας κοινά εμπόδια που αντιμετωπίζουν οι χρήστες με αναπηρίες, συμπεριλαμβανομένων των προβλημάτων όρασης. Τα ευρήματά τους υπογραμμίζουν την ανάγκη για καλύτερη εφαρμογή και σαφέστερες οδηγίες για τη βελτίωση της συμπερίληψης των εφαρμογών για κινητά.

Ο Park et al.[45] ανέπτυξαν ένα αυτοματοποιημένο εργαλείο αξιολόγησης που έχει σχεδιαστεί για την αξιολόγηση της προσβασιμότητας των εφαρμογών Android. Το εργαλείο παρέχει ανατροφοδότηση σε πραγματικό χρόνο σχετικά με τη συμμόρφωση με την προσβασιμότητα των εφαρμογών, εντοπίζει κοινά προβλήματα και προσφέρει προτάσεις για βελτιώσεις. Αυτή η έρευνα

επισημαίνει τη σημασία της αυτοματοποίησης της διαδικασίας αξιολόγησης προσβασιμότητας, ώστε να διευκολυνθούν οι προγραμματιστές να δημιουργήσουν πιο συμπεριληπτικές εφαρμογές.

Ο Acosta-Vargas et al.[46] επικεντρώθηκαν στην αξιολόγηση της προσβασιμότητας των εφαρμογών για κινητά Android, χρησιμοποιώντας ανθρώπινους παράγοντες και μεθοδολογίες αλληλεπίδρασης συστήματος. Η μελέτη εντόπισε βασικά προβλήματα προσβασιμότητας που αντιμετωπίζουν οι χρήστες με αναπηρίες και παρουσίασε προτάσεις για τη βελτίωση της προσβασιμότητας των εφαρμογών για κινητά μέσω της καλύτερης τήρησης των αρχών και των οδηγιών σχεδιασμού.

Ο Mateus et al.[47] διεξήγαγαν μια αξιολόγηση εφαρμογών για κινητά με επίκεντρο τον χρήστη, εστιάζοντας στην προσβασιμότητα για άτομα με προβλήματα όρασης. Η μελέτη συνέκρινε τις εμπειρίες χρηστών με προβλήματα όρασης με τα αποτελέσματα των αυτοματοποιημένων εργαλείων αξιολόγησης προσβασιμότητας. Η έρευνα ανέδειξε τις αποκλίσεις μεταξύ των αυτοματοποιημένων αξιολογήσεων και των πραγματικών εμπειριών των χρηστών, υπογραμμίζοντας τη σημασία της συμμετοχής των χρηστών στη διαδικασία αξιολόγησης.

Ο Acosta-Vargas et al.[48] διερεύνησαν την προσβασιμότητα των εγγενών εφαρμογών για κινητά για άτομα με αναπηρίες στην ανασκόπηση πεδίου εφαρμογής τους. Η ανασκόπηση ανέδειξε σημαντικά κενά στη συμμόρφωση με την προσβασιμότητα και την ανάγκη για καλύτερα εργαλεία και μεθοδολογίες για να διασφαλιστεί ότι οι εφαρμογές για κινητά είναι προσβάσιμες σε όλους τους χρήστες, ιδίως σε εκείνους με προβλήματα όρασης και σωματικές βλάβες.

Οι τεχνολογίες ανίχνευσης αντικειμένων έχουν βελτιωθεί σημαντικά, ιδιαίτερα σε θορυβώδη ή δυναμικά περιβάλλοντα. Για παράδειγμα, οι Zheng et al. [49] πρότειναν ένα ασύμμετρο τριγραμμικό δίκτυο προσοχής με συναρτήσεις ενεργοποίησης θορύβου για την ενίσχυση της αυτόματης ταξινόμησης διαμόρφωσης υπό παραμορφώσεις σήματος, καταδεικνύοντας πώς οι μηχανισμοί προσοχής μπορούν να σταθεροποιήσουν την απόδοση σε δύσκολες συνθήκες. Ομοίως, οι Liu et al. [50] άλλαξαν την ιεραρχία χαρακτηριστικών πολλαπλών κλιμάκων στους μετασχηματιστές ανίχνευσης (DETR), αντιμετωπίζοντας την ανίχνευση αντικειμένων με μεταβλητή κλίμακας.

Κεφάλαιο 5ο: Ανάπτυξη Εφαρμογής σε Android Studio

5.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα δούμε πώς δημιουργήθηκε η εφαρμογή που συνδυάζει διάφορες τεχνολογίες για να βοηθήσει ανθρώπους με προβλήματα όρασης. Η εφαρμογή χρησιμοποιεί τεχνητή νοημοσύνη για να αναγνωρίζει αντικείμενα σε πραγματικό χρόνο και ενημερώνει τον χρήστη φωνητικά και μέσω απτικής ανατροφοδότησης.

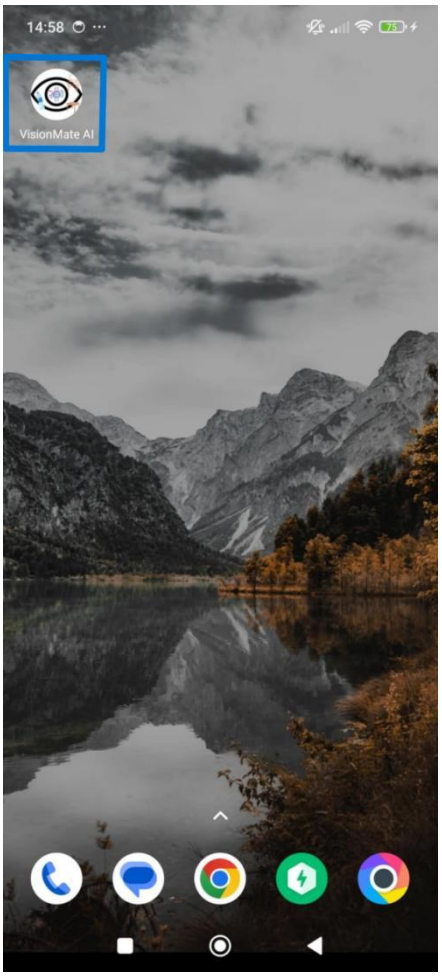
Η εφαρμογή ονομάζεται **VisionMate AI**, ένα όνομα που φέρνει μαζί τις έννοιες της όρασης, της βοήθειας και της τεχνολογίας. Ο στόχος της είναι να προσφέρει στους χρήστες μια πιο εύκολη και ανεξάρτητη καθημερινότητα, βοηθώντας τους να «δουν» τον κόσμο γύρω τους μέσα από την κάμερα του κινητού τους και με την βοήθεια φωνητικών εντολών.

Στη συνέχεια, παρουσιάζεται η διαδικασία ανάπτυξης της εφαρμογής, από την αρχική σχεδίαση και υλοποίηση της διεπαφής χρήστη, μέχρι την ενσωμάτωση προηγμένων αλγορίθμων τεχνητής νοημοσύνης, όπως το YOLOv11n Segmentation στο περιβάλλον του TensorFlow Lite (TFLite). Επιπλέον, περιγράφονται οι προσθήκες που βελτιώνουν τη λειτουργικότητα, όπως η φωνητική αναγνώριση εντολών, η μέτρηση απόστασης αντικειμένων και ο σχεδιασμός της φωνητικής εμπειρίας χρήστη.

Η εφαρμογή αυτή δεν είναι απλώς μια τεχνολογική λύση, αλλά ένα εργαλείο που προσπαθεί να φέρει πραγματική βοήθεια και αυτονομία σε όσους την έχουν ανάγκη.

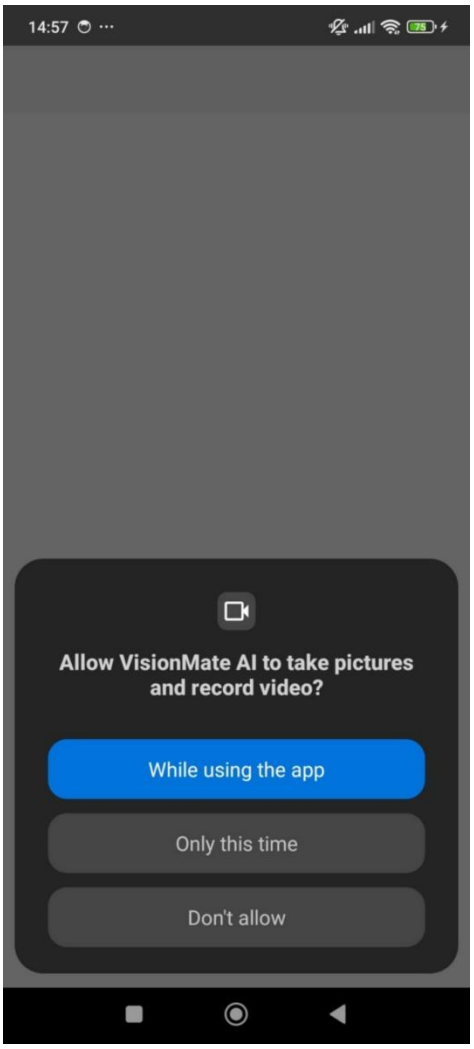
5.2 Έναρξη της εφαρμογής

- Η εφαρμογή VisionMats AI εκκινεί πατώντας το αντίστοιχο εικονίδιο στην αρχική οθόνη του κινητού, όπως φαίνεται παρακάτω. Στόχος είναι να προσφέρει άμεση και εύκολη πρόσβαση στους χρήστες με προβλήματα όρασης μέσω μιας απλής διεπαφής.



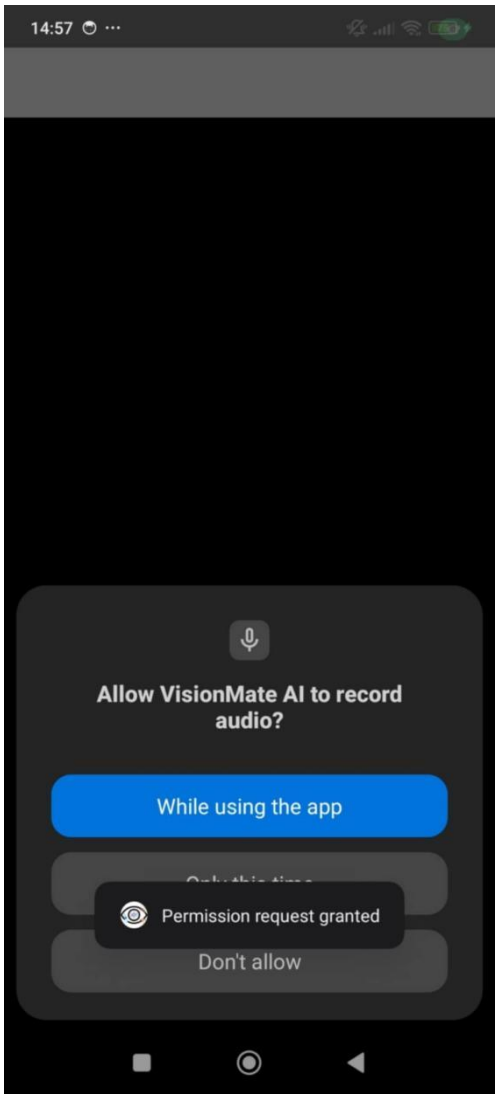
Εικόνα 5.1: Εκκίνηση εφαρμογής

- Με την εκκίνηση της εφαρμογής, εμφανίζεται ένα αναδυόμενο παράθυρο (popup) που ζητά από τον χρήστη την άδεια πρόσβασης στην κάμερα της συσκευής. Η άδεια αυτή είναι απαραίτητη για τη βασική λειτουργία της VisionMate AI, καθώς επιτρέπει την ανίχνευση αντικειμένων σε πραγματικό χρόνο.



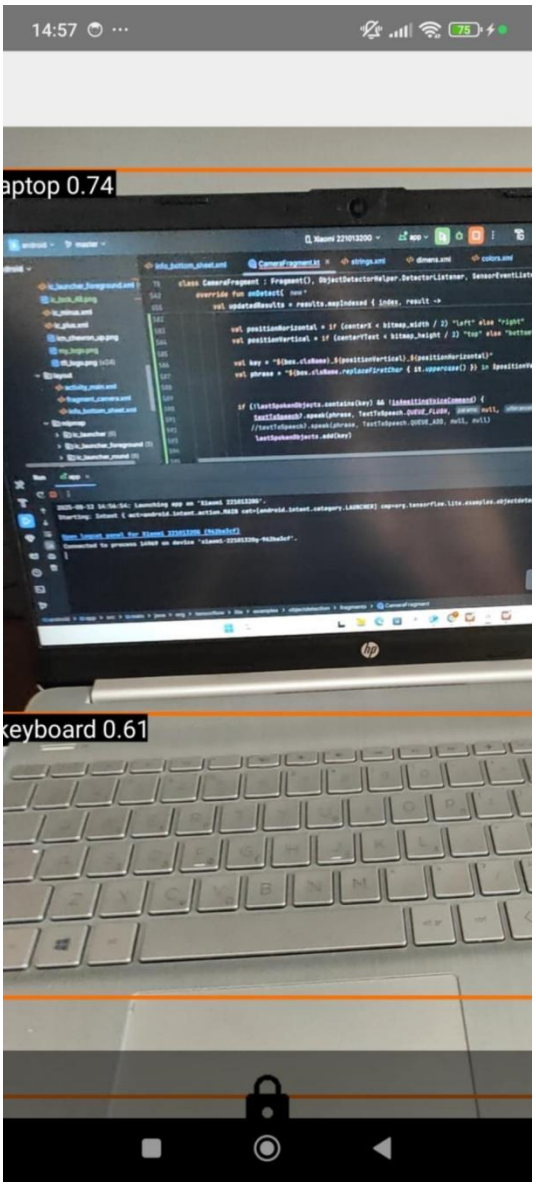
Εικόνα 5.2: Διεπαφή αδειών κάμερας

- Μετά την έγκριση για την κάμερα, η εφαρμογή ζητά άδεια πρόσβασης και στο μικρόφωνο της συσκευής. Η άδεια αυτή είναι αναγκαία για την υποστήριξη των φωνητικών εντολών και της εκφώνησης των αποτελεσμάτων ανίχνευσης.



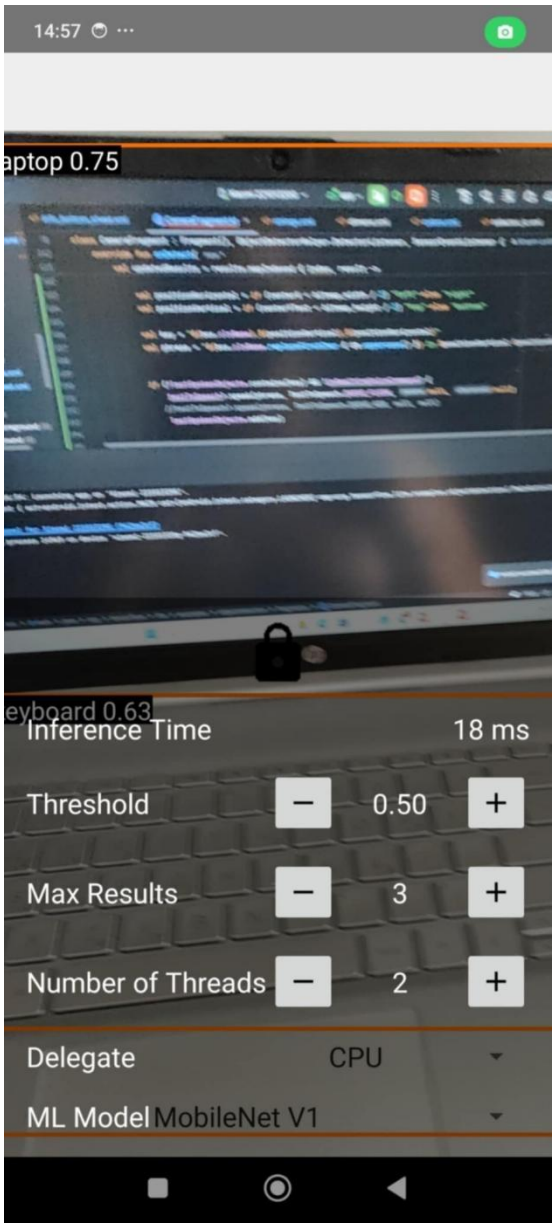
Εικόνα 5.3: Διεπαφή αδειών μικροφώνου

- Με την παροχή άδειας στην κάμερα, η εφαρμογή ξεκινά άμεσα τη λήψη καρέ σε συνεχή ροή (live streaming). Αυτό επιτρέπει την πραγματοποίηση της ανίχνευσης αντικειμένων σε πραγματικό χρόνο, στοιχείο κρίσιμο για την άμεση ενημέρωση του χρήστη.



Εικόνα 5.4: Προεπισκόπηση κάμερας

- Αν ο χρήστης θέλει να ανοίξει το μενού ρυθμίσεων, αρκεί να πραγματοποιήσει διπλό πάτημα (double tap) οπουδήποτε στην οθόνη της εφαρμογής. Με την ενέργεια αυτή ξεκινά ένας φωνητικός διάλογος, όπου το σύστημα ρωτά: «Do you want to open settings? Say yes or no.» Αν ο χρήστης απαντήσει καταφατικά, με φράσεις όπως "yes", "yeah" ή "ok", το μενού ρυθμίσεων ανοίγει και η εφαρμογή ανακοινώνει φωνητικά: «Opening settings.»Η φωνητική αυτή αλληλεπίδραση κάνει την πλοήγηση απλή και προσβάσιμη, ειδικά για χρήστες με προβλήματα όρασης, αφού μπορεί να γίνει χωρίς την ανάγκη οπτικής επαφής με τα κουμπιά της εφαρμογής.



Εικόνα 5.5: Διεπαφή ρυθμίσεων

5.3 Περιγραφή της Διεπαφής Χρήστη

Το μενού ρυθμίσεων της εφαρμογής **VisionMate** εμφανίζεται ως **αναδυόμενο πάνελ (bottom sheet)**, το οποίο ανοίγει ομαλά προς τα πάνω από το κάτω μέρος της οθόνης με φωνητική εντολή. Αυτή η σχεδίαση επιτρέπει στον χρήστη να έχει γρήγορη πρόσβαση στις ρυθμίσεις χωρίς να απομακρύνεται από την κύρια οθόνη της εφαρμογής.

Η ενεργοποίηση του μενού γίνεται με διπλό πάτημα (double tap) οπουδήποτε στην οθόνη, είτε μέσω φωνητικής εντολής, όπου ο χρήστης καλείται με φωνητικό διάλογο:

Ο χρήστης για να ενεργοποιήση το μενου κανει διπλό πάτημα οπουδήποτε στην οθόνη και ξεκιναει ενα διάλογος. "Do you want to open settings? Say yes or no." Με την επιβεβαίωση ("yes", "yeah", "ok"), το μενού αναδύεται και είναι έτοιμο για χρήση.

Οι βασικές επιλογές του μενού ρυθμίσεων είναι:

1. Inference Time

a) Εμφανίζει τον χρόνο εκτέλεσης (σε milliseconds) που απαιτείται για την επεξεργασία κάθε καρέ.

b) Χρησιμοποιείται κυρίως για την αξιολόγηση της απόδοσης του μοντέλου σε πραγματικό χρόνο.

2. Threshold

a) Καθορίζει το ελάχιστο ποσοστό εμπιστοσύνης (confidence score) που πρέπει να έχει μια ανίχνευση ώστε να θεωρηθεί έγκυρη.

b) Για παράδειγμα, με threshold 0.5, η εφαρμογή θα εμφανίζει μόνο αντικείμενα που το μοντέλο εντοπίζει με ακρίβεια ≥ 50%.

3. Max Results

a) Ορίζει τον μέγιστο αριθμό αντικειμένων που μπορεί να επιστρέψει το μοντέλο για κάθε καρέ.

b) Χρήσιμο για τη διαχείριση του φόρτου εργασίας και την αποφυγή υπερφόρτωσης της οθόνης με πληροφορίες.

4. Number of Threads

a) Επιτρέπει στον χρήστη να καθορίσει πόσα threads (νήματα επεξεργασίας) θα χρησιμοποιηθούν για την εκτέλεση του μοντέλου.

b) Η αύξηση των threads μπορεί να βελτιώσει την ταχύτητα, αλλά μπορεί να επηρεάσει την κατανάλωση μπαταρίας.

5. Delegate

a) Δίνει τη δυνατότητα επιλογής του υποσυστήματος εκτέλεσης του μοντέλου, π.χ. CPU, GPU ή NNAPI.

b) Κάθε επιλογή έχει πλεονεκτήματα και μειονεκτήματα ανάλογα με τη συσκευή.

6. ML Model

a) Επιτρέπει την επιλογή του μοντέλου μηχανικής μάθησης που θα χρησιμοποιηθεί, π.χ. YOLOv11n-seg ή άλλα διαθέσιμα μοντέλα.

b) Η δυνατότητα αυτή είναι χρήσιμη για δοκιμές διαφορετικών μοντέλων με ποικίλα επίπεδα ακρίβειας και ταχύτητας.

5.4 YOLOv11 TFLite: Ενσωμάτωση και Λειτουργία

Η επιλογή του μοντέλου YOLOv11n-segmentation στην εφαρμογή VisionMate πραγματοποιήθηκε με τον σκοπό να βελτιωθεί η επίδοση και την συνεχή εξέλιξη της λειτουργικότητας του συστήματος. Σε σύγκριση με προηγούμενες εκδόσεις και άλλες μεθόδους ανίχνευσης, το YOLOv11 προσφέρει:

- Ταχύτερη επεξεργασία καρέ σε συσκευές με περιορισμένους πόρους.
- Καλύτερη ακρίβεια αναγνώρισης ακόμη και σε πολύπλοκα περιβάλλοντα.
- Υποστήριξη instance segmentation, επιτρέποντας στο σύστημα να παρέχει στον χρήστη πέρα απο το όνομα και τη θέση ενός αντικειμένου, αλλά και το περίγραμμά του.

Η ενσωμάτωση του YOLOv11 μέσω TensorFlow Lite (TFLite) καθιστά δυνατή την εκτέλεση του μοντέλου σε πραγματικό χρόνο πάνω σε Android συσκευές, χωρίς σημαντικές καθυστερήσεις και χωρίς υπερβολική κατανάλωση πόρων. Η προσθήκη του μοντέλου ενισχύει σημαντικά την εμπειρία του χρήστη, ειδικά για άτομα με προβλήματα όρασης, καθώς οι πληροφορίες που παρέχονται είναι ακριβέστερες και πιο αξιόπιστες.

Η επιλογή του YOLOv11n για την εφαρμογή VisionMate έγινε με κύριο στόχο την άυξηση των επιδόσεων και την εξέλιξη της εφαρμογής σε σχέση με προηγούμενες εκδόσεις μοντέλων ανίχνευσης αντικειμένων. Το YOLOv11 αποτελεί την τελευταία έκδοση της οικογένειας YOLO της Ultralytics, προσφέροντας υψηλή ακρίβεια ανίχνευσης, ταχύτητα επεξεργασίας και υποστήριξη πολλαπλών τύπων ανίχνευσης, όπως τμηματοποίηση αντικειμένων και εκτίμηση στάσης σώματος [18], [19].
Η ενσωμάτωση του μοντέλου σε μορφή TFLite επιτρέπει την εκτέλεση σε κινητές συσκευές Android με περιορισμένους πόρους, διατηρώντας παράλληλα την ακρίβεια ανίχνευσης σε πραγματικό χρόνο. Τα κύρια χαρακτηριστικά που αξιοποιούνται στην εφαρμογή περιλαμβάνουν:

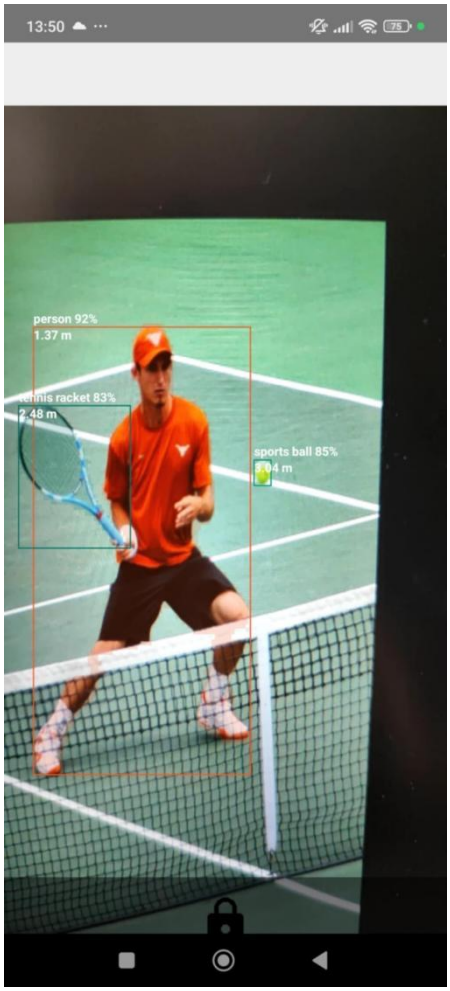
- Υποστήριξη πέντε μεγεθών μοντέλου: Nano, Small, Medium, Large, X [18].
- Δυνατότητα εκτέλεσης τόσο σε CPU όσο και σε GPU ή με χρήση delegate (π.χ., NNAPI) για επιτάχυνση.
- Ενσωματωμένη τμηματοποίηση αντικειμένων για ακριβέστερη αναγνώριση και οριοθέτηση.
- Βελτιστοποιημένη αρχιτεκτονική για ταχύτητα και μικρή καθυστέρηση (low-latency inference).

Ιστορικά, το YOLO (You Only Look Once) παρουσιάστηκε για πρώτη φορά το 2016 και έφερε μια επανάσταση στην ανίχνευση αντικειμένων με την προσέγγιση ενός βήματος (single-shot detection) [20]. Η εξέλιξή του σε YOLOv11 περιλαμβάνει σημαντικές βελτιώσεις, όπως:

- **Ακριβέστερη εξαγωγή χαρακτηριστικών**: Χρησιμοποιεί βελτιωμένα convolutional layers και modules attention για καλύτερη αναγνώριση μικρών και κοντά αντικειμένων [19].
- **Υψηλότερη ταχύτητα εκτέλεσης**: Η εκτέλεση σε κινητές συσκευές είναι άμεση, διατηρώντας frame rates συμβατά με real-time εφαρμογές [18].
- **Ευελιξία στις εφαρμογές**: Υποστηρίζει εκτός από ανίχνευση και τμηματοποίηση και άλλες λειτουργίες όπως tracking και pose estimation [19].

Η χρήση του YOLOv11 σε συνδυασμό με το TFLite επιτρέπει στην εφαρμογή **VisionMate** να δίνει στον χρήστη με προβλήματα όρασης την δυνατότητα ακριβείς πληροφορίας για τα αντικείμενα που βρίσκονται στον γύρω χώρο, σε πραγματικό χρόνο. Αυτή η επιλογή βελτιώνει τη συνολική εμπειρία

χρήστη και αποτελεί τη βάση για την υλοποίηση των επόμενων λειτουργιών, όπως ο υπολογισμός αποστάσεων και η φωνητική αναγνώριση αντικειμένων.



Εικόνα 5.6: Άνθρωπος, ρακέτα του τέννις και μπαλάκι του τεννις

Εικόνα 5.7: Λεωφορεία

5.4.1 Διαδικασία ενσωμάτωσης

Η ενσωμάτωση του μοντέλου **YOLOv11** στην εφαρμογή **VisionMate** πραγματοποιήθηκε σε διαδοχικά στάδια, με στόχο την ομαλή λειτουργία σε Android συσκευές με περιορισμένους πόρους.

1. Μετατροπή και προετοιμασία του μοντέλου

Το αρχικό μοντέλο YOLOv11n-segmentation μετατράπηκε σε μορφή TensorFlow Lite (.tflite) χρησιμοποιώντας τον TFLite Converter. Η μορφή του μοντέλου σε TFLite μας βοηθάει για να εκτελεστεί το μοντέλο σε Android συσκευες σε πραγματικό χρόνο.

2. Ενσωμάτωση στο Android Studio

Αρχικά ενσωματώθηκε το μοντέλο σε μορφή .tflite στο έργο στο Android Studio και συνδέθηκε με τον TensorFlow Lite Interpreter γραμμένο σε Kotlin, ώστε να μπορεί να εκτελείται απευθείας στην εφαρμογή. Προσαρμόστηκαν οι βασικές ρυθμίσεις εισόδου και εξόδου για σωστή ανίχνευση και τμηματοποίηση αντικειμένων.

3. Επιτάχυνση εκτέλεσης

Για την αύξηση της ταχύτητας, το YOLOv11 μπορεί να εκτελείται σε **CPU**, **GPU delegate**, ή με χρήση του **NNAPI**. Το frame rate ρυθμίστηκε ώστε να επιτυγχάνεται **real-time ανίχνευση**, διατηρώντας παράλληλα χαμηλή κατανάλωση μπαταρίας.

4. Σύνδεση με UI και φωνητική αναγνώριση

Η ενσωμάτωση περιλαμβάνει επίσης την σύνδεση με το UI και τις φωνητικές εντολές. Κάθε ανίχνευση αντικειμένου παρουσιάζεται φωνητικα απο την εφαρμογήκαι εμφανίζονται στην οθόνη. Η λειτουργία double tap επιτρέπει στο χρήστη να ανοίγει το μενού επιλογών και να αλληλεπιδρά με την εφαρμογή χωρίς οπτική επαφή.

5.4.2 Λειτουργία στην πράξη

Στην εφαρμογή **VisionMate**, το μοντέλο **YOLOv11n-Segmentation** εκτελείται σε πραγματικό χρόνο, ανιχνεύοντας αντικείμενα που εμφανίζονται μπροστά από την κάμερα της συσκευής. Κάθε ανίχνευση περιλαμβάνει:

- **Όνομα αντικειμένου:** Αναγνωρίζει και ανακοινώνει το είδος του αντικειμένου.
- **Θέση και περίγραμμα:** Παρέχεται οπτική αναπαράσταση για όσους μπορούν να δουν την οθόνη, αλλά κυρίως χρησιμοποιείται για τον υπολογισμό απόστασης.
- **Ενημέρωση χρήστη:** Οι πληροφορίες μετατρέπονται σε φωνητική καθοδήγηση μέσω ενσωματωμένου TTS (Text-to-Speech).

Η εφαρμογή υποστηρίζει **double tap gesture** για να ανοίγει το μενού με φωνητικές εντολές, ενώ οι εντολές φωνής επιτρέπουν την ενεργοποίηση πρόσθετων λειτουργιών όπως η επιλογή διαφορετικών τύπων αναγνώρισης ή η αναφορά συγκεκριμένων αντικειμένων.

Η χρήση του YOLOv11 σε συνδυασμό με TFLite και την προσαρμοσμένη διεπαφή εξασφαλίζει αξιόπιστη και γρήγορη αναγνώριση αντικειμένων, προσφέροντας στον χρήστη με προβλήματα όρασης ακριβείς πληροφορίες για τον περιβάλλοντα χώρο σε πραγματικό χρόνο.

5.5 Υπολογισμός Απόστασης Αντικειμένων με Κάμερα

Ο υπολογισμός της απόστασης των αντικειμένων στην εφαρμογή **VisionMate** υλοποιήθηκε με βάση τις αρχές της τριγωνομετρίας [21]. Θεωρούμε οτι η συσκευή ειναι σε σταθερό ύψος περίπου **1,20 μέτρων** από το έδαφος και επίσης θεωρούμε ως δεδομένο οτι το αντικείμενο βρίσκεται στο έδαφος.

Η μέθοδος που εφαρμόστηκε χρησιμοποιεί την πληροφορία της γωνίας κλίσης της κάμερας, η οποία αντλείται από τους ενσωματωμένους αισθητήρες κίνησης και θέσης της συσκευής (inclinometer/gyroscope) [22]. Όταν εντοπιστεί ένα αντικείμενο, η εφαρμογή καταγράφει την κάθετη διαφορά ύψους μεταξύ της κάμερας και του αντικειμένου (γνωστή σταθερά στην παρούσα υλοποίηση) και, σε συνδυασμό με τη μετρημένη γωνία, υπολογίζει την απόσταση του αντικειμένου από τον χρήστη [21].

Ο τρόπος αυτός είναι παρόμοιος με τεχνικές που χρησιμοποιούνται σε ρομποτικά συστήματα όρασης και σε εφαρμογές υπολογιστικής όρασης για αθλητικές ή βιομηχανικές μετρήσεις [21]. Επιλέχθηκε διότι δεν απαιτεί ειδικό εξοπλισμό (όπως LiDAR ή stereo cameras) και μπορεί σε οποιαδήποτε συσκευή Android να τρέξει αν έχει βασικούς αισθητήρες και κάμερα[23].

Η ακρίβεια του υπολογισμού εξαρτάται από την ορθή βαθμονόμηση του ύψους της κάμερας και από την αξιοπιστία των αισθητήρων θέσης της συσκευής [22]. Σε δοκιμές που πραγματοποιήθηκαν, η

μέθοδος παρουσίασε ικανοποιητικά αποτελέσματα για αποστάσεις έως και περίπου 5 μέτρα, κάτι που θεωρείται επαρκές για την καθοδήγηση ατόμων με προβλήματα όρασης σε εσωτερικούς και εξωτερικούς χώρους.

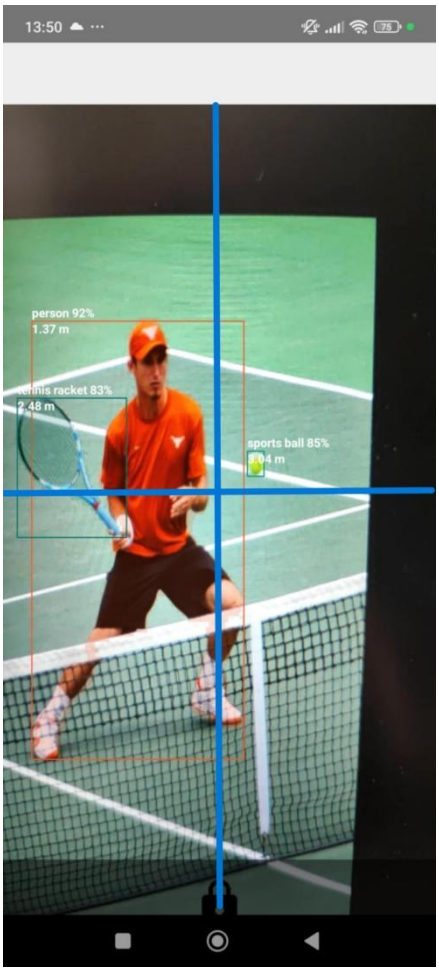
5.6 Σχεδιασμός Φωνητικής Εμπειρίας Χρήστη

Η φωνητική παρουσίαση αντικειμένων της εφαρμογής VisionMate σχεδιάστηκε ώστε να παρέχει στον χρήστη με προβλήματα όρασης άμεση καθοδήγηση χωρίς να απαιτείται οπτική επαφή με την οθόνη. Η βασική φιλοσοφία στηρίζεται σε αρχές φυσικής διαλόγου και προσβασιμότητας, όπου η πληροφορία παρέχεται με σύντομες, ευανάγνωστες ηχητικές οδηγίες [24].

Η εφαρμογή αξιοποιεί τα APIs φωνητικής εξόδου (Text-to-Speech) της πλατφόρμας Android, επιτρέποντας την προσαρμογή του ρυθμού ομιλίας και της έντασης ανάλογα με τις προτιμήσεις του χρήστη [25]. Οι οδηγίες διατυπώνονται σε απλή γλώσσα, αποφεύγοντας τεχνικούς όρους, ώστε να είναι πιο εύκολο και κατανοητό για τον χρήστη.

Για να γίνεται πιο κατανοητή η θέση των αντικειμένων στην οθόνη, η οθόνη χωρίστηκε σε τέσσερα τεταρτημόρια: πάνω αριστερά, πάνω δεξιά, κάτω αριστερά και κάτω δεξιά. Η φωνητική καθοδήγηση αναφέρει σε ποιο τεταρτημόριο βρίσκεται κάθε αντικείμενο, δίνοντας στον χρήστη σαφή αίσθηση του χώρου και της κατεύθυνσης [25].

Στο πλαίσιο δοκιμών χρηστών, παρατηρήθηκε ότι οι φωνητικές οδηγίες μειώνουν σημαντικά την αβεβαιότητα κατά την κίνηση σε εσωτερικούς χώρους, ενώ η δυνατότητα ρύθμισης του ρυθμού ομιλίας επιτρέπει στον χρήστη να προσαρμόζει την εμπειρία στις προσωπικές του ανάγκες [24][25].



Εικόνα 5.8: Διάταξη και χωρισμός της οθόνης

5.7 Δομή Κώδικα: Περιγραφή βασικών Κλάσεων

Η εφαρμογή ανίχνευσης αντικειμένων βασίζεται σε μια έτοιμη πλατφόρμα **TensorFlow Lite**, την οποία πήραμε από το GitHub. Στην αρχική αυτή βάση πρόσθεσα τα εξής στοιχεία:

- **YOLOv11n** για την ανίχνευση αντικειμένων σε πραγματικό χρόνο.
- **Μέτρηση απόστασης αντικειμένων** με χρήση τριγωνομετρίας
- **Φωνητική εκφώνηση αντικειμένων** στην ελληνική γλώσσα.
- **Double tap gesture**, που ξεκινάει τον διάλογο για το άνοιγμα του μενού.

Η ανάπτυξη έγινε στο περιβάλλον **Android Studio**, επομένως η εφαρμογή είναι κατάλληλη για κινητές συσκευές με Android. Παρακάτω περιγράφονται οι βασικές κλάσεις και η λειτουργία τους.

5.7.1 Περιγραφή Κλάσης “CameraFragment”

Στην παρακάτω εικόνα φαίνεται το τμήμα του κώδικα όπου δηλώνονται οι απαραίτητες βιβλιοθήκες για την κλάση CameraFragment. Αυτά τα imports επιτρέπουν στην εφαρμογή να χρησιμοποιεί τις λειτουργίες της κάμερας, την ανίχνευση αντικειμένων με YOLOv11n, την επεξεργασία εικόνας και την υλοποίηση των gestures και της φωνητικής περιγραφής. Αναλυτικότερα, θα δούμε τον κώδικα τι κάνει γραμμή γραμμή.

Δήλωση Πακέτου(γραμμή 1): Δηλώνεται το πακέτο στο οποίο ανήκει η κλάση CameraFragment.

Imports(γραμμές 16-74): Σε αυτό το τμήμα, ο κώδικας εισάγει απαραίτητες βιβλιοθήκες που χρειάζεται η εφαρμογή για να λειτουργήσει.

```
1  / Copyright 2022 The TensorFlow Authors. All Rights Reserved. .../
16 package org.tensorflow.lite.examples.objectdetection.fragments
17
18 import android.annotation.SuppressLint
19 import android.content.Context
20 import android.content.Intent
21 import android.content.res.Configuration
22 import android.graphics.Bitmap
23 import android.graphics.Matrix
24 import android.hardware.Sensor
25 import android.hardware.SensorEvent
26 import android.hardware.SensorEventListener
27 import android.hardware.SensorManager
28 import android.hardware.camera2.CameraCharacteristics
29 import android.hardware.camera2.CameraManager
30 import android.os.Bundle
31 import android.os.Handler
32 import android.os.Looper
33 import android.speech.RecognitionListener
34 import android.speech.RecognizerIntent
35 import android.speech.SpeechRecognizer
36 import android.speech.tts.TextToSpeech
37 import android.util.Log
38 import android.view.GestureDetector
39 import android.view.LayoutInflater
40 import android.view.MotionEvent
41 import android.view.View
42 import android.view.ViewGroup
43 import android.widget.AdapterView
44 import android.widget.Toast
45 import androidx.activity.result.contract.ActivityResultContracts
46 import androidx.camera.core.AspectRatio
47 import androidx.camera.core.Camera
```

Εικόνα 5.9: Imports

```
48  import androidx.camera.core.CameraSelector
49  import androidx.camera.core.ImageAnalysis
50  import androidx.camera.core.ImageAnalysis.OUTPUT_IMAGE_FORMAT_RGBA_8888
51  import androidx.camera.core.ImageProxy
52  import androidx.camera.core.Preview
53  import androidx.camera.lifecycle.ProcessCameraProvider
54  import androidx.core.content.ContextCompat
55  import androidx.fragment.app.Fragment
56  import androidx.navigation.Navigation
57  import com.google.android.material.bottomsheet.BottomSheetBehavior
58  import java.util.LinkedList
59  import java.util.concurrent.ExecutorService
60  import java.util.concurrent.Executors
61  import org.tensorflow.lite.examples.objectdetection.ObjectDetectorHelper
62  import org.tensorflow.lite.examples.objectdetection.R
63  import org.tensorflow.lite.examples.objectdetection.SegmentationResult
64  import org.tensorflow.lite.examples.objectdetection.databinding.FragmentCameraBinding
65  import org.tensorflow.lite.task.vision.detector.Detection
66  import org.tensorflow.lite.examples.objectdetection.DrawImages
67  import org.tensorflow.lite.examples.objectdetection.ImageUtils.getLowestActivePixel
68  import java.util.Locale
69  import kotlin.math.atan
70  import kotlin.math.PI
71  import kotlin.math.abs
72  import kotlin.math.absoluteValue
73  import kotlin.math.tan
74  import org.tensorflow.lite.examples.objectdetection.InstanceSegmentation
75
```

Εικόνα 5.10: Imports

Δήλωση Κλάσης και Μεταβλητών (γραμμές 78-114):
Η κλάση CameraFragment κληρονομεί την κλάση Fragment και υλοποιεί τα interfaces ObjectDetectorHelper.DetectorListener και SensorEventListener. Ορίζεται μια σταθερά TAG για τον εντοπισμό μηνυμάτων log που σχετίζονται με την ανίχνευση αντικειμένων.

Δηλώνονται οι παρακάτω μεταβλητές:

- textToSpeech: για την εκφώνηση των αντικειμένων σε πραγματικό χρόνο.
- lastSpokenObjects: ένα Set που αποθηκεύει τα τελευταία αντικείμενα που εκφωνήθηκαν, για να αποφεύγεται η επαναλαμβανόμενη ομιλία.
- isYoloSelected: σημαία για την επιλογή YOLO μοντέλου ανίχνευσης.
- instanceSegmentation: αντικείμενο InstanceSegmentation για την υποστήριξη ανίχνευσης και τμηματοποίησης αντικειμένων.
- gestureDetector: διαχειριστής gestures για τον έλεγχο της εφαρμογής μέσω κινήσεων.
- speechRecognizer: διαχειριστής φωνητικών εντολών.
- isAwaitingVoiceCommand: σημαία που δείχνει αν η εφαρμογή περιμένει φωνητική εντολή.
- isSettingsOpen: σημαία που δείχνει αν το μενού ρυθμίσεων είναι ανοιχτό.
- behavior: αντικείμενο BottomSheetBehavior για τον χειρισμό του interface.
- audioPermissionRequest: διαχείριση αδειών για πρόσβαση στο μικρόφωνο.
- rotatedBitmap: αποθηκεύει το περιστρεφόμενο bitmap της εικόνας από την κάμερα.
- latestPitch: τελευταίο γωνιακό pitch της συσκευής από τους αισθητήρες.
- sensorManager και rotationSensor: για την παρακολούθηση κίνησης της συσκευής.
- verticalFOV: κάθετο πεδίο όρασης της κάμερας.

- _fragmentCameraBinding και fragmentCameraBinding: πρόσβαση στα στοιχεία του XML interface.
- drawImages: για την απόδοση εικόνων και bounding boxes πάνω στην οθόνη.
- objectDetectorHelper: χειριστής του μοντέλου ανίχνευσης αντικειμένων.
- bitmapBuffer: προσωρινή αποθήκευση εικόνων σε μορφή bitmap.
- preview, imageAnalyzer και camera: διαχείριση λειτουργιών κάμερας και ανάλυσης εικόνας.
- cameraProvider: πρόσβαση στις λειτουργίες της κάμερας της συσκευής.
- cameraExecutor: εκτέλεση blocking λειτουργιών κάμερας σε ξεχωριστό thread.

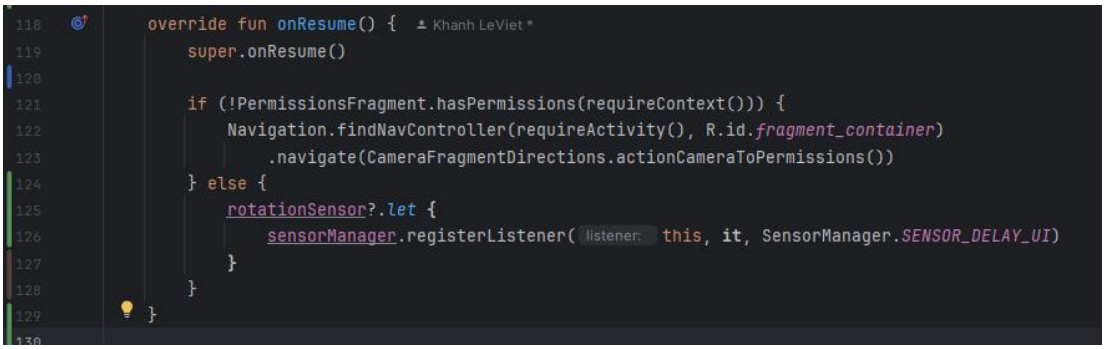
```
class CameraFragment : Fragment(), ObjectDetectorHelper.DetectorListener, SensorEventListener {  Khanh LeViet *

    private val TAG = "ObjectDetection"
    private var textToSpeech: TextToSpeech? = null
    private val lastSpokenObjects = mutableSetOf<String>()
    private var isYoloSelected = false
    private var instanceSegmentation: InstanceSegmentation? = null

    private lateinit var gestureDetector: GestureDetector
    private lateinit var speechRecognizer: SpeechRecognizer
    private var isAwaitingVoiceCommand = false
    private var isSettingsOpen = false
    private lateinit var behavior: BottomSheetBehavior<*>
    private val audioPermissionRequest =
        registerForActivityResult(ActivityResultContracts.RequestPermission()) { isGranted ->
            if (!isGranted) {
                Toast.makeText(requireContext(), text: "Audio permission is required for voice control.", Toast.LENGTH_LONG).show()
            }
        }
    private lateinit var rotatedBitmap: Bitmap
    private var latestPitch: Double = 0.0
    private lateinit var sensorManager: SensorManager
    private var rotationSensor: Sensor? = null
    private var verticalFOV: Double = 60.0
    private var _fragmentCameraBinding: FragmentCameraBinding? = null
    private var drawImages: DrawImages? = null
    private val fragmentCameraBinding  Khanh LeViet
        get() = _fragmentCameraBinding!!
    private lateinit var objectDetectorHelper: ObjectDetectorHelper
    private lateinit var bitmapBuffer: Bitmap
    private var preview: Preview? = null
    private var imageAnalyzer: ImageAnalysis? = null
    private var camera: Camera? = null
    private var cameraProvider: ProcessCameraProvider? = null

    /** Blocking camera operations are performed using this executor */
    private lateinit var cameraExecutor: ExecutorService
```

Εικόνα 5.11: Μεταβλητές

Μέθοδος onResume() (γραμμές 118-129): Η μέθοδος onResume καλείται όταν η οθόνη της εφαρμογής ενεργοποιείται. Ο σκοπός της είναι να διασφαλίσει ότι η εφαρμογή έχει όλες τις απαραίτητες άδειες.

Αναλυτικά:

- Ελέγχεται αν η εφαρμογή έχει τις απαιτούμενες άδειες μέσω της PermissionsFragment.hasPermissions(requireContext()).
- Αν οι άδειες δεν υπάρχουν, πηγαίνει στη σελίδα αδειών (PermissionsFragment) ώστε ο χρήστης να εγκρίνει την ενεργοποίηση των αδειών.
- Αν οι άδειες έχουν ήδη χορηγηθεί, καταχωρείται ο rotationSensor στον sensorManager για την παρακολούθηση της κίνησης της συσκευής.



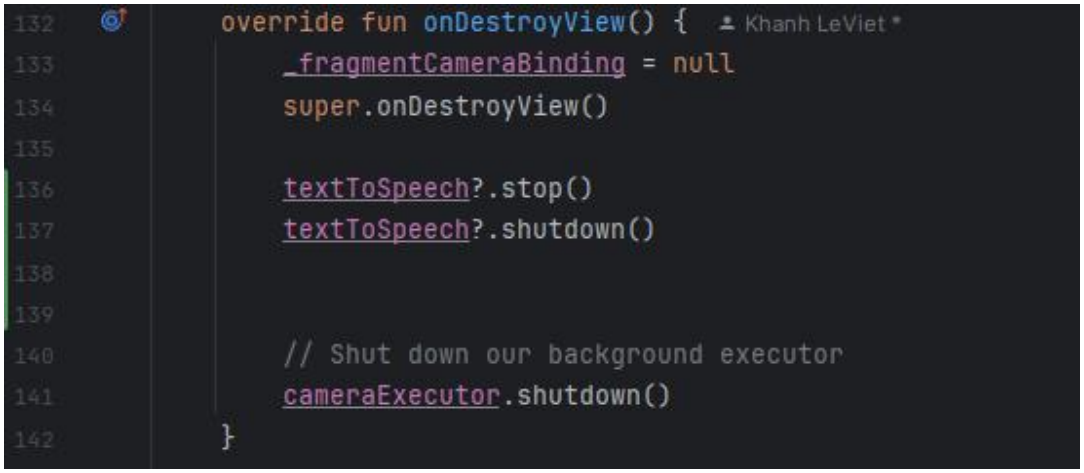
Εικόνα 5.12: Μέθοδος onResume

Μέθοδος onDestroyView() **(γραμμές 132-142)**: Η μέθοδος onDestroyView() καλείται όταν κλείνει η εφαρμογή απο την οθόνη. Η κύρια λειτουργία της είναι ο καθαρισμός των πόρων για να αποφευχθούν διαρροές μνήμης.

Αναλυτικά:

- Η μεταβλητή _fragmentCameraBinding τίθεται σε null για να απελευθερωθεί η αναφορά στο interface της οθόνης.
- Ο TextToSpeech σταματάει και τερματίζεται με τις μεθόδους stop() και shutdown() για να απελευθερωθούν οι πόροι του συστήματος.
- Ο cameraExecutor, ο οποίος εκτελεί τις λειτουργίες της κάμερας σε ξεχωριστό thread, τερματίζεται με τη μέθοδο shutdown() για να διασφαλιστεί ότι δεν υπάρχουν ενεργές διεργασίες στο background.

Αυτό εξασφαλίζει σωστή απελευθέρωση πόρων και σταθερή λειτουργία της εφαρμογής.



Εικόνα 5.13: Μέθοδος onDestroyView

Μέθοδος onCreateView() (γραμμές 144-157): Η μέθοδος onCreateView() καλείται κατά τη δημιουργία του CameraFragment και αυτή έχει την ευθύνη για την εκχώρηση της διάταξης της οθόνης και την αρχικοποίηση βασικών αντικειμένων.

Αναλυτικά:

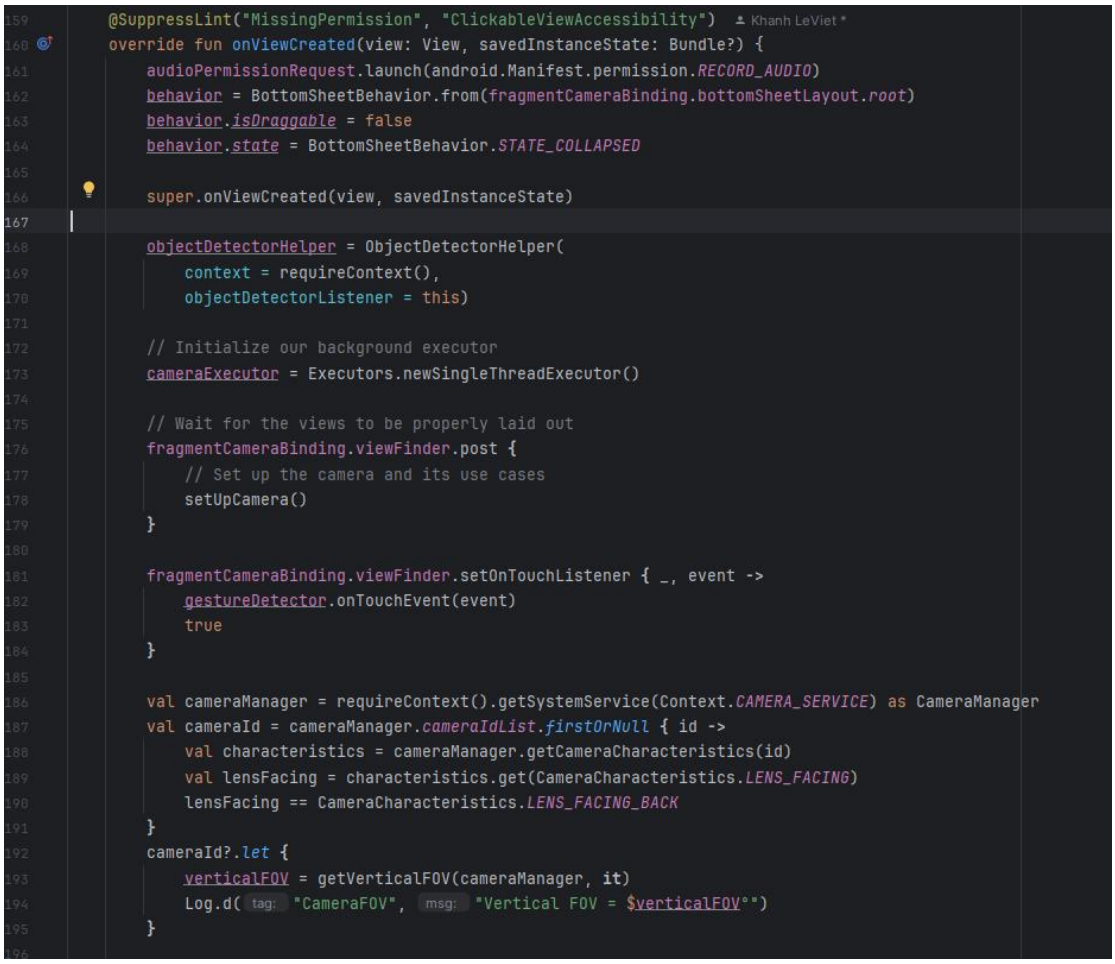
- Η _fragmentCameraBinding αρχικοποιείται με το FragmentCameraBinding.inflate(...) για να συνδεθεί η κλάση με το XML layout της οθόνης.
- Δημιουργείται ένα αντικείμενο DrawImages το οποίο χρησιμοποιείται για την δημιουργεία των κουτιών γύρω απο τα αντικειμένα που εντοπίζει η κάμερα.
- Ο sensorManager αρχικοποιείται για να αντλείσουμε δεδομένα απο τους αισθητήρες της συσκευής.
- Ο rotationSensor λαμβάνει τον αισθητήρα, ο οποίος χρησιμοποιείται για υπολογισμούς προσανατολισμού και κλίσης της συσκευής.
- Τέλος, επιστρέφεται η root view του binding για να εμφανιστεί το layout στην οθόνη.

```
144    override fun onCreateView(   Khanh LeViet *
145        inflater: LayoutInflater,
146        container: ViewGroup?,
147        savedInstanceState: Bundle?
148    ): View {
149
150        _fragmentCameraBinding = FragmentCameraBinding.inflate(inflater, container,  attachToParent:  false)
151        drawImages = DrawImages(requireContext())
152
153        sensorManager = requireContext().getSystemService(Context.SENSOR_SERVICE) as SensorManager
154        rotationSensor = sensorManager.getDefaultSensor(Sensor.TYPE_ROTATION_VECTOR)
155
156        return fragmentCameraBinding.root
157    }
```

Εικόνα 5.14: Μέθοδος onCreateView

Μέθοδος onViewCreated(γραμμές 160-257): Η μέθοδος onViewCreated καλείται αφού η διάταξη της οθόνης έχει δημιουργηθεί. Η συγκεκριμένη κλάση δημιουργεί ένα αντικείμενο ObjectDetectorHelper. Επίσης γίνεται το αίτημα άδειας μικροφώνου για τις φωνητικές εντολές, ρύθμιση της διεπαφής χρήστη, εκκίνηση της κάμερας, υπολογισμός του κάθετου πεδίου θέασης, και ρύθμιση του τρόπου εκφώνησης και το double tap για το άνοιγμα ή κλείσιμο του μενου.

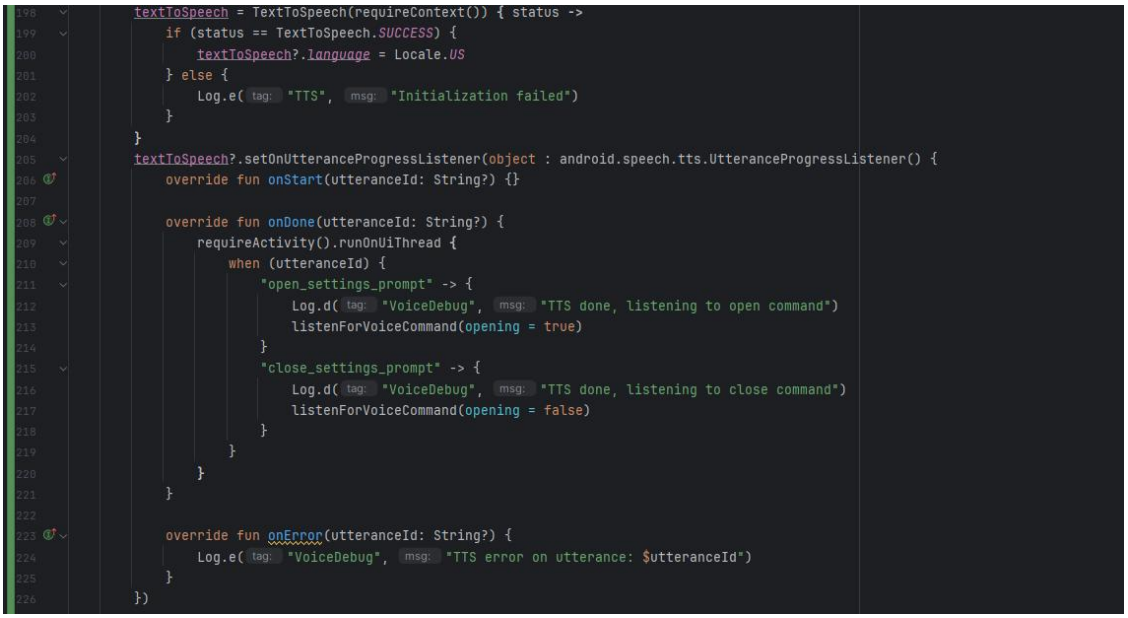
Η μέθοδος ξεκινά με την εκτέλεση του audioPermissionRequest.launch(...), με αυτόν τον τρόπο ζητάμε την άδεια από τον χρήστη, ώστε να μπορεί να λειτουργήσουν οι φωνητικές εντολές. Στην συνέχεια χρησιμοποιείται το BottomSheetBehavior για τη διαχείριση του αναδυόμενου μενου. Το isDraggable τίθεται σε false για να αποτρέπεται το χειροκίνητο σύρσιμο προς τα πάνω, ενώ η αρχική του κατάσταση ορίζεται σε STATE_COLLAPSED. Μετά δημιουργείται ένα αντικείμενο ObjectDetectorHelper με παράμετρο το context και το objectDetectorListener που είναι το ίδιο το CameraFragment. Το αντικείμενο αυτό χειρίζεται την εκτέλεση του μοντέλου ανίχνευσης αντικειμένων. Επιπλέον αρχικοποιείται ένας ExecutorService για εκτέλεση λειτουργιών κάμερας σε ξεχωριστό νήμα. Με την εντολή viewFinder.post { setUpCamera() } η κάμερα ρυθμίζεται μόλις ολοκληρωθεί η φόρτωση των views. Με τη χρήση του CameraManager εντοπίζεται η πίσω κάμερα της συσκευής, παίρνουμε τα χαρακτηριστικά της και υπολογίζουμε το verticalFOV.



```
@SuppressLint("MissingPermission", "ClickableViewAccessibility")  Khanh LeViet *
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    audioPermissionRequest.launch(android.Manifest.permission.RECORD_AUDIO)
    behavior = BottomSheetBehavior.from(fragmentCameraBinding.bottomSheetLayout.root)
    behavior.isDraggable = false
    behavior.state = BottomSheetBehavior.STATE_COLLAPSED

    super.onViewCreated(view, savedInstanceState)

    objectDetectorHelper = ObjectDetectorHelper(
        context = requireContext(),
        objectDetectorListener = this)

    // Initialize our background executor
    cameraExecutor = Executors.newSingleThreadExecutor()

    // Wait for the views to be properly laid out
    fragmentCameraBinding.viewFinder.post {
        // Set up the camera and its use cases
        setUpCamera()
    }

    fragmentCameraBinding.viewFinder.setOnTouchListener { _, event ->
        gestureDetector.onTouchEvent(event)
        true
    }

    val cameraManager = requireContext().getSystemService(Context.CAMERA_SERVICE) as CameraManager
    val cameraId = cameraManager.cameraIdList.firstOrNull { id ->
        val characteristics = cameraManager.getCameraCharacteristics(id)
        val lensFacing = characteristics.get(CameraCharacteristics.LENS_FACING)
        lensFacing == CameraCharacteristics.LENS_FACING_BACK
    }
    cameraId?.let {
        verticalFOV = getVerticalFOV(cameraManager, it)
        Log.d( tag: "CameraFOV",  msg: "Vertical FOV = $verticalFOV°")
    }
```

Εικόνα 5.15: Μέθοδος onViewCreated

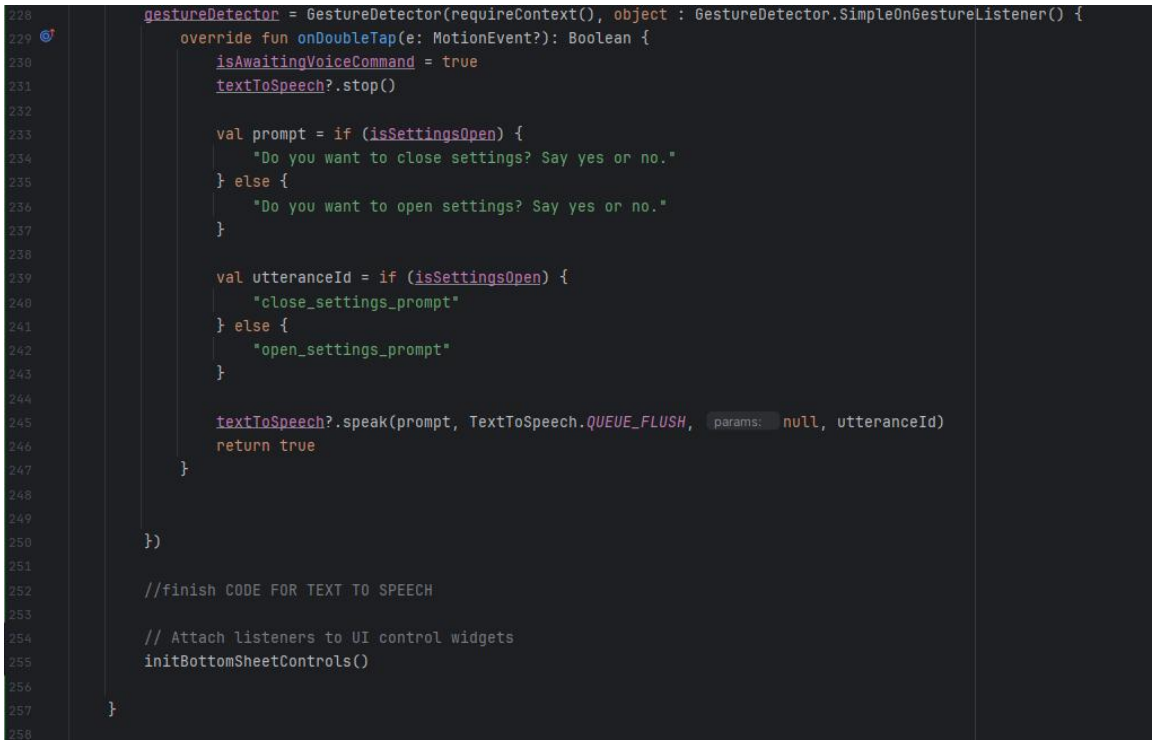
Το αντικείμενο TextToSpeech δημιουργείται με γλώσσα Locale.US. Επίσης ορίζεται OnUtteranceProgressListener που επιτρέπει στο σύστημα να ξεκινήσει αναμονή φωνητικής εντολής μετά την εκφώνηση την πρότασης (π.χ. “Do you want to open settings?”).



```
        textToSpeech = TextToSpeech(requireContext()) { status ->
            if (status == TextToSpeech.SUCCESS) {
                textToSpeech?.language = Locale.US
            } else {
                Log.e( tag: "TTS",  msg: "Initialization failed")
            }
        }
        textToSpeech?.setOnUtteranceProgressListener(object : android.speech.tts.UtteranceProgressListener() {
            override fun onStart(utteranceId: String?) {}

            override fun onDone(utteranceId: String?) {
                requireActivity().runOnUiThread {
                    when (utteranceId) {
                        "open_settings_prompt" -> {
                            Log.d( tag: "VoiceDebug",  msg: "TTS done, listening to open command")
                            listenForVoiceCommand(opening = true)
                        }
                        "close_settings_prompt" -> {
                            Log.d( tag: "VoiceDebug",  msg: "TTS done, listening to close command")
                            listenForVoiceCommand(opening = false)
                        }
                    }
                }
            }

            override fun onError(utteranceId: String?) {
                Log.e( tag: "VoiceDebug",  msg: "TTS error on utterance: $utteranceId")
            }
        })
```

Εικόνα 5.16: Μέθοδος onStart,OnDone,onError

Στην συνέχεια αρχικοποιείται το GestureDetector με υπερκάλυψη της μεθόδου onDoubleTap(). Κάθε διπλό πάτημα ενεργοποιεί εκφώνηση ερωτήματος και αναμονή φωνητικής απάντησης, για άνοιγμα ή κλείσιμο του μενού ρυθμίσεων. Τέλος, καλείται η μέθοδος initBottomSheetControls() που συνδέει τα κουμπιά και τα στοιχεία ελέγχου της διεπαφής με τον αντίστοιχο κώδικα λειτουργίας.



```
        gestureDetector = GestureDetector(requireContext(), object : GestureDetector.SimpleOnGestureListener() {
            override fun onDoubleTap(e: MotionEvent?): Boolean {
                isAwaitingVoiceCommand = true
                textToSpeech?.stop()

                val prompt = if (isSettingsOpen) {
                    "Do you want to close settings? Say yes or no."
                } else {
                    "Do you want to open settings? Say yes or no."
                }

                val utteranceId = if (isSettingsOpen) {
                    "close_settings_prompt"
                } else {
                    "open_settings_prompt"
                }

                textToSpeech?.speak(prompt, TextToSpeech.QUEUE_FLUSH,  params: null, utteranceId)
                return true
            }


        })

        //finish CODE FOR TEXT TO SPEECH

        // Attach listeners to UI control widgets
        initBottomSheetControls()

    }
```

Εικόνα 5.17: Μέθοδος onDoubleTap

Η μέθοδος **listenForVoiceCommand** έχει ως σκοπό να αναλάβει να ακούσει και να επεξεργαστεί την απάντηση του χρήστη στην ερώτηση που θα του γίνει μετά το double tap. Η παράμετρος opening καθορίζει αν η ερώτηση αφορά το άνοιγμα (true) ή το κλείσιμο (false) του μενού ρυθμίσεων.
Αρχικά δημιουργείται νέο αντικείμενο SpeechRecognizer. Στην συνέχεια δημειουργείται Intent τύπου RecognizerIntent.ACTION_RECOGNIZE_SPEECH με παραμέτρους:

- EXTRA_LANGUAGE_MODEL → LANGUAGE_MODEL_FREE_FORM για ελεύθερη αναγνώριση φωνής.
- EXTRA_LANGUAGE → Locale.ENGLISH για χρήση της Αγγλικής γλώσσας.

Στην συνέχεια ρυθμίζεται ανώνυμη κλάση RecognitionListener για να "ακούσουμε" τι απάντησε ο χρήστης:

- Ελέγχεται λοιπόν η απάντηση που έδωσε ο χρήστης.
  - Αν είναι **"yes"**, **"yeah"**, ή **"ok"** τότε ανοίγει ή κλείνει το μενού ανάλογα σε τι κατάσταση βρίσκεται, εκφωνώντας σχετικό μήνυμα.
  - Αν περιέχει **"no"** τότε εκφωνεί "Okay." χωρίς αλλαγή κατάστασης.
  - Οτιδήποτε άλλο τότε εκφωνεί μήνυμα μη κατανόησης.

onError():
Εκφωνεί την φράση ("I didn't catch that...") και σταματά την αναμονή.

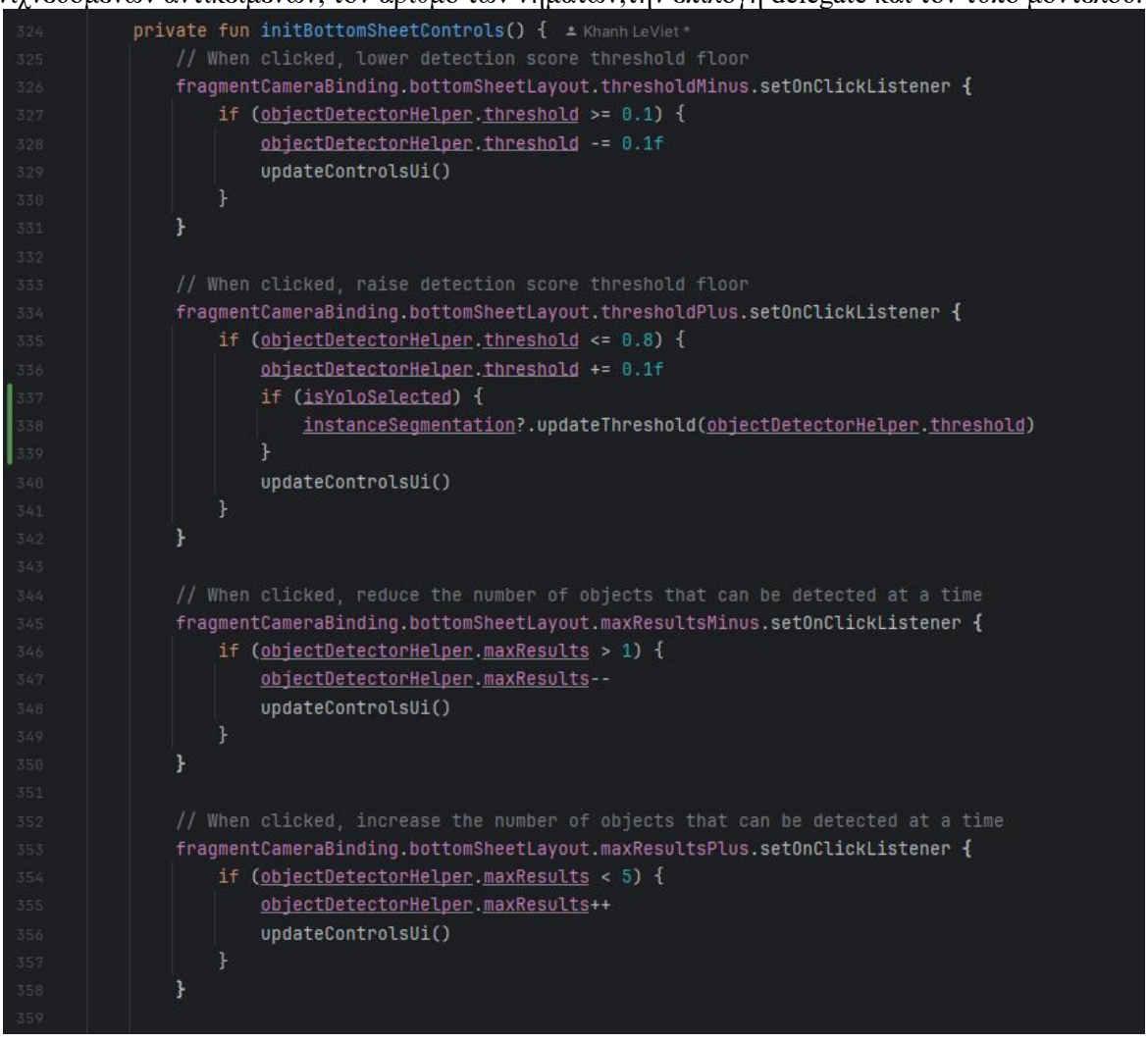
```
private fun listenForVoiceCommand(opening: Boolean) {
    speechRecognizer = SpeechRecognizer.createSpeechRecognizer(requireContext())

    val intent = Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH).apply {
        putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL, RecognizerIntent.LANGUAGE_MODEL_FREE_FORM)
        putExtra(RecognizerIntent.EXTRA_LANGUAGE, Locale.ENGLISH)
    }

    speechRecognizer.setRecognitionListener(object : RecognitionListener {
        override fun onResults(results: Bundle?) {
            val spokenText = results
                ?.getStringArrayList(SpeechRecognizer.RESULTS_RECOGNITION)
                ?.firstOrNull()
                ?.lowercase(Locale.getDefault())

            Log.d( tag: "SpeechInput",  msg: "Heard: $spokenText")



            if (spokenText != null) {
                when {
                    spokenText.contains( other: "yes") || spokenText.contains( other: "yeah") || spokenText.contains( other: "ok") -> {
                        if (opening) {
                            behavior.state = BottomSheetBehavior.STATE_EXPANDED
                            isSettingsOpen = true
                            textToSpeech?.speak( text: "Opening settings.", TextToSpeech.QUEUE_FLUSH,  params: null,  utteranceId: null)
                        } else {
                            behavior.state = BottomSheetBehavior.STATE_COLLAPSED
                            isSettingsOpen = false
                            textToSpeech?.speak( text: "Closing settings.", TextToSpeech.QUEUE_FLUSH,  params: null,  utteranceId: null)
                        }
                    }

                    spokenText.contains( other: "no") -> {
                        textToSpeech?.speak( text: "Okay.", TextToSpeech.QUEUE_FLUSH,  params: null,  utteranceId: null)
                    }

                    else -> {
                        textToSpeech?.speak( text: "I didn't understand. Please try again.", TextToSpeech.QUEUE_FLUSH,  params: null,  utteranceId: null)
                    }
```

Εικόνα 5.18: Μέθοδος listenForVoiceCommand

Η μέθοδος ολοκληρώνεται με την κλήση speechRecognizer.startListening(intent), ξεκινώντας την ακρόαση φωνής του χρήστη.

Εικόνα 5.19: Μέθοδος onReadyForSpeech,onBeginningofSpeech,onRasChanged,onBufferReceived,onEndofSpeech,onError,onPartialResults,onEvent

Μέθοδος initBottomSheetControls(γραμμές 324-395): Η μέθοδος initBottomSheetControls προσθέτει listeners στα στοιχεία ελέγχου του bottom sheet που επιτρέπουν στον χρήστη να ρυθμίσει τις παραμέτρους ανίχνευσης αντικειμένων, όπως το ποσοστό ευαισθησίας, τον αριθμό των ανιχνευόμενων αντικειμένων, τον αριθμό των νημάτων,την επιλογή delegate και τον τύπο μοντέλου.



Εικόνα 5.20: Μέθοδος initBottomSheetControls

```
359
360         // When clicked, decrease the number of threads used for detection
361         fragmentCameraBinding.bottomSheetLayout.threadsMinus.setOnClickListener {
362             if (objectDetectorHelper.numThreads > 1) {
363                 objectDetectorHelper.numThreads--
364                 updateControlsUi()
365             }
366         }
367
368         // When clicked, increase the number of threads used for detection
369         fragmentCameraBinding.bottomSheetLayout.threadsPlus.setOnClickListener {
370             if (objectDetectorHelper.numThreads < 4) {
371                 objectDetectorHelper.numThreads++
372                 updateControlsUi()
373             }
374         }
375
376         fragmentCameraBinding.bottomSheetLayout.spinnerDelegate.setSelection( position: 0,  animate: false)
377         fragmentCameraBinding.bottomSheetLayout.spinnerDelegate.onItemSelectedListener =
378             object : AdapterView.OnItemSelectedListener {
379                 override fun onItemSelected(p0: AdapterView<*>?, p1: View?, p2: Int, p3: Long) {
380                     objectDetectorHelper.currentDelegate = p2
381                     updateControlsUi()
382                 }
383
384                 override fun onNothingSelected(p0: AdapterView<*>?) {}
385             }
386
387         fragmentCameraBinding.bottomSheetLayout.spinnerModel.setSelection( position: 0,  animate: false)
388         fragmentCameraBinding.bottomSheetLayout.spinnerModel.onItemSelectedListener =
389             object : AdapterView.OnItemSelectedListener {
390                 override fun onItemSelected(p0: AdapterView<*>?, p1: View?, p2: Int, p3: Long) {
391                     val modelName = resources.getStringArray(R.array.models_spinner_titles)[p2]
392                     objectDetectorHelper.currentModel = p2
393                     isYoloSelected = modelName == "YOLOv11n Segmentation"
394                     updateControlsUi()
395                 }
396
```

Εικόνα 5.21: Μέθοδος initBottomSheetControls,onItemSelected,onNothingSelected,onItemSelected

Μέθοδος updateControlsUi(γραμμές 402-422): Αυτή η μέθοδος ενημερώνει τα στοιχεία του UI στο μενού ρυθμίσεων με τις τρέχουσες τιμές των παραμέτρων ανίχνευσης αντικειμένων. Αρχικά ενημερώνει τα πεδία κειμένου ώστε να εμφανίζουν τις τρέχουσες τιμές του maxResults, threshold και numThreads. Μετά ελέγχει αν το μοντέλο που έχει επιλεχθεί είναι το YOLO. Αν το YOLO έχει επιλεχθεί τότε καλεί τις αντίστοιχες μεθόδους ενημέρωσεις για να προσαρμοστούν οι παράμετροι. Αν δέν έχει επιλεχθεί το YOLO καθαρίζει τον τρέχοντα ανιχνευτή και τον επααρυθμίζει απο την αρχή. Τέλος καθαρίζει το overlay στην οθόνη για να διαγραφούν τα τυχόν προηγούμενα γραφικά ανίχνευσης και να εμφανιστούν μόνο τα αποτελέσματα της επόμενης εκτέλεσης.

```
402     private fun updateControlsUi() {  Khanh LeViet *
403         fragmentCameraBinding.bottomSheetLayout.maxResultsValue.text =
404             objectDetectorHelper.maxResults.toString()
405         fragmentCameraBinding.bottomSheetLayout.thresholdValue.text =
406             String.format("%.2f", objectDetectorHelper.threshold)
407         fragmentCameraBinding.bottomSheetLayout.threadsValue.text =
408             objectDetectorHelper.numThreads.toString()
409
410
411         Log.d(TAG,  msg: "selected model: $isYoloSelected")
412
413         if (isYoloSelected) {
414             objectDetectorHelper.updateYoloThreshold(objectDetectorHelper.threshold)
415             objectDetectorHelper.updateYoloThreads(objectDetectorHelper.numThreads)
416             objectDetectorHelper.updateYoloMaxResults(objectDetectorHelper.maxResults)
417         } else {
418             objectDetectorHelper.clearObjectDetector()
419             objectDetectorHelper.setupObjectDetector()
420         }
421
422         fragmentCameraBinding.overlay.clear()
423     }
424
425
```

Εικόνα 5.22: Μέθοδος updateControlsUi

Μέθοδος setUpCamera(γραμμές 426-432): Η δουλεία της μεθόδου αυτής ειναι να προετοιμάσει την κάμερα για χρήση. Αρχικά, καλεί την ProcessCameraProvider.getInstance() για να πάρει έναν CameraProvider, δηλαδή το αντικείμενο που διαχειρίζεται την κάμερα. Μόλις η λήψη του provider ολοκληρωθει, αποθηκεύεται το αποτέλεσμα στην μεταβλητή cameraProvider και στην συνέχεια καλείται η μέθοδος bindCameraUseCases(). Εκεί γίνεται η πραγματική "δέσμευση" των λειτουργιών της κάμερας για να ξεκινήσει η ζωντανή ροή εικόνας στην εφαρμογή.

```
426     private fun setUpCamera() {  Khanh LeViet *
427         val cameraProviderFuture = ProcessCameraProvider.getInstance(requireContext())
428         cameraProviderFuture.addListener({
429             cameraProvider = cameraProviderFuture.get()
430             bindCameraUseCases()
431         }, ContextCompat.getMainExecutor(requireContext()))
432     }
433
```

Εικόνα 5.23: Μέθοδος setUpCamera

Μέθοδος bindCameraUseCases(434-471): Η μέθοδος bindCameraUseCases δημιουργεί και συνδέει τις περιπτώσεις χρήσης της κάμερας:

Αρχικά, ελέγχει αν υπάρχει διαθέσιμο cameraProvider, αλλιώς πετάει εξαίρεση.
Έπειτα, δημιουργεί ένα CameraSelector που ζητά ρητά την πίσω κάμερα της συσκευής.

Στη συνέχεια ορίζονται δύο βασικές λειτουργίες:

1. **Preview:** Φτιάχνεται ένα αντικείμενο Preview με αναλογία εικόνας 4:3 και περιστροφή σύμφωνα με την οθόνη, ώστε η εικόνα να εμφανίζεται σωστά.
2. **ImageAnalysis:** Δημιουργείται ένας αναλυτής εικόνας με τις ίδιες ρυθμίσεις αναλογίας και περιστροφής, στρατηγική STRATEGY_KEEP_ONLY_LATEST (για να επεξεργάζεται πάντα το πιο πρόσφατο frame), και έξοδο σε μορφή RGBA. Ο αναλυτής εκτελείται στο cameraExecutor

και καλεί τη detectObjects(image). Στην πρώτη εκτέλεση αρχικοποιείται το bitmapBuffer με βάση το frame.

Πριν γίνει η νέα δέσμευση, καλείται η μέθοδος unbindAll() για να αποδεσμευτούν τυχόν προηγούμενες περιπτώσεις χρήσης. Τέλος, επιχειρεί να δεσμεύσει (bindToLifecycle) την προεπισκόπηση και τον αναλυτή εικόνας στον κύκλο ζωής του fragment. Αν πετύχει, η επιφάνεια προβολής (surfaceProvider) συνδέεται με το viewFinder του layout. Αν αποτύχει, καταγράφεται σφάλμα στα logs.

```
433
434     @SuppressLint("UnsafeOptInUsageError")  Khanh LeViet *
435     private fun bindCameraUseCases() {
436         val cameraProvider = cameraProvider ?: throw IllegalStateException("Camera initialization failed.")
437         val cameraSelector = CameraSelector.Builder().requireLensFacing(CameraSelector.LENS_FACING_BACK).build()
438
439         preview = Preview.Builder()
440             .setTargetAspectRatio(AspectRatio.RATIO_4_3)
441             .setTargetRotation(fragmentCameraBinding.viewFinder.display.rotation)
442             .build()
443
444         imageAnalyzer = ImageAnalysis.Builder()
445             .setTargetAspectRatio(AspectRatio.RATIO_4_3)
446             .setTargetRotation(fragmentCameraBinding.viewFinder.display.rotation)
447             .setBackpressureStrategy(ImageAnalysis.STRATEGY_KEEP_ONLY_LATEST)
448             .setOutputImageFormat(OUTPUT_IMAGE_FORMAT_RGBA_8888)
449             .build()
450             .also {
451                 it.setAnalyzer(cameraExecutor) { image ->
452                     if (!::bitmapBuffer.isInitialized) {
453                         bitmapBuffer = Bitmap.createBitmap(
454                             image.width,
455                             image.height,
456                             Bitmap.Config.ARGB_8888
457                         )
458                     }
459                     detectObjects(image)
460                 }
461             }
462
463         cameraProvider.unbindAll()
464
465         try {
466             camera = cameraProvider.bindToLifecycle( lifecycleOwner: this, cameraSelector, preview, imageAnalyzer)
467             preview?.setSurfaceProvider(fragmentCameraBinding.viewFinder.surfaceProvider)
468         } catch (exc: Exception) {
469             Log.e(TAG,  msg: "Use case binding failed", exc)
470         }
471     }
```

Εικόνα 5.24: Μέθοδος bindCameraUseCases

Μέθοδος detectObjects(image: ImageProxy)(γραμμές 473-487): Αυτή η μέθοδος καλείται για να πάρει το τρέχον frame από την κάμερα, να το περιστρέψει σωστά και να το περάσει στο σύστημα ανίχνευσης αντικειμένων. Αρχικά διαβάζονται τα δεδομένα της εικόνας απο το ImageProxy και αντιγράφονται μέσα στο bitmapBuffer. Στην συνέχεια παίρνει την πληροφορία rotationDegrees απο τα Metadata του frame. Έπειτα, δημιουργείται ένα Matrix και εφαρμόζεται η περιστοφή που χρειάζεται για να είναι σωστά στην οθόνη η εικόνα. Μετά, δημιουργείται το rotatedBitmap και τέλος η μέθοδος καλεί την objectDetectorHelper.detect(rotatedBitmap, 0) ώστε να εκτελέσει την ανίχνευσει αντικειμένων πάνω στην επεξεργασμένη εικόνα.

```
473     private fun detectObjects(image: ImageProxy) {  Khanh LeViet *
474         image.use { bitmapBuffer.copyPixelsFromBuffer(image.planes[0].buffer) }
475
476         val imageRotation = image.imageInfo.rotationDegrees
477
478         val matrix = Matrix().apply {
479             postRotate(imageRotation.toFloat())
480         }
481
482         rotatedBitmap = Bitmap.createBitmap(
483             bitmapBuffer,  x: 0,  y: 0, bitmapBuffer.width, bitmapBuffer.height, matrix,  filter: true
484         )
485
486         objectDetectorHelper.detect(rotatedBitmap,  imageRotation: 0)
487     }
488
```

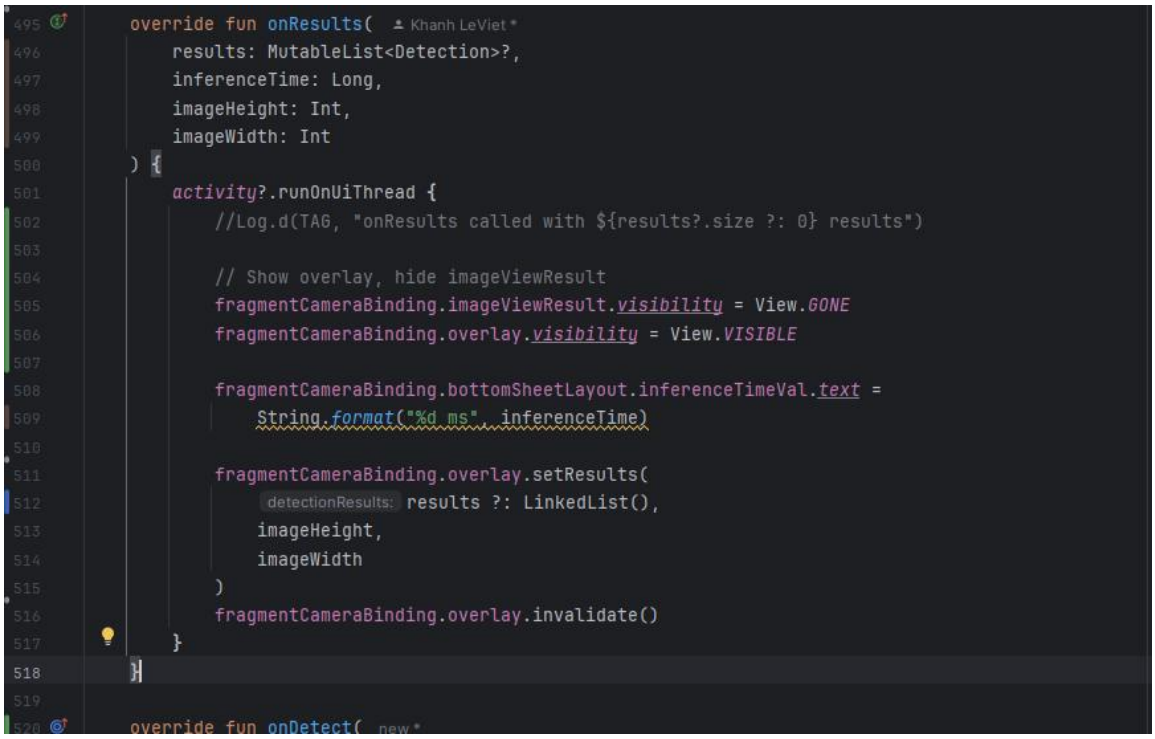
Εικόνα 5.25: Μέθοδος detectObjects

Μέθοδος onConfigurationChanged(γραμμές 490-493): Η μέθοδος αυτή καλείται μόλις αλλάζει κάτι στην συσκευή. Για παράδειγμα μόλις περιστραφεί η οθόνης της. Γίνεται η κλήση της υπερκλάσης για να διατηρηθεί η βασικλη λειτουργικότητα. Μετά ενημερώνεται το imageAnalyzer ώστε η ιδιότητα targetRotation να ευθυγραμμιστεί με τη νέα περιστροφή και να φαίνονται τα αντικείμενα σωστά στην οθόνη.

```
490     override fun onConfigurationChanged(newConfig: Configuration) {  Khanh LeViet
491         super.onConfigurationChanged(newConfig)
492         imageAnalyzer?.targetRotation = fragmentCameraBinding.viewFinder.display.rotation
493     }
```

Εικόνα 5.26: Μέθοδος onConfigurationChanged

Μέθοδος onResults (γραμμές 495-518): Η μέθοδος αυτή τρέχει κάθε φορά που τελειώνει η διαδικασία ανίχνευσης αντικειμένων και επιστρέφονται αποτελέσματα από το μοντέλο. Η εκτέλεση γίνεται στο UI thread μέσω του activity?.runOnUiThread, ώστε να μπορούν να ενημερωθούν σωστά τα γραφικά στοιχεία. Αρχικά, η εικόνα αποτελεσμάτων κρύβεται, ενώ το overlay εμφανίζεται για να εμφανίσει τα νέα δεδομένα. Στη συνέχεια, στο κάτω μέρος του μενού ενημερώνεται η ένδειξη του χρόνου επεξεργασίας σε χιλιοστά του δευτερολέπτου.
Με την κλήση της setResults στο overlay περνιούνται τα αποτελέσματα της ανίχνευσης, όπως και οι διαστάσεις της αρχικής εικόνας. Τέλος, με την invalidate() το overlay επανασχεδιάζεται, ώστε να εμφανιστούν γρήγορα τα αντικείμενα που βρέθηκαν στην οθόνη.



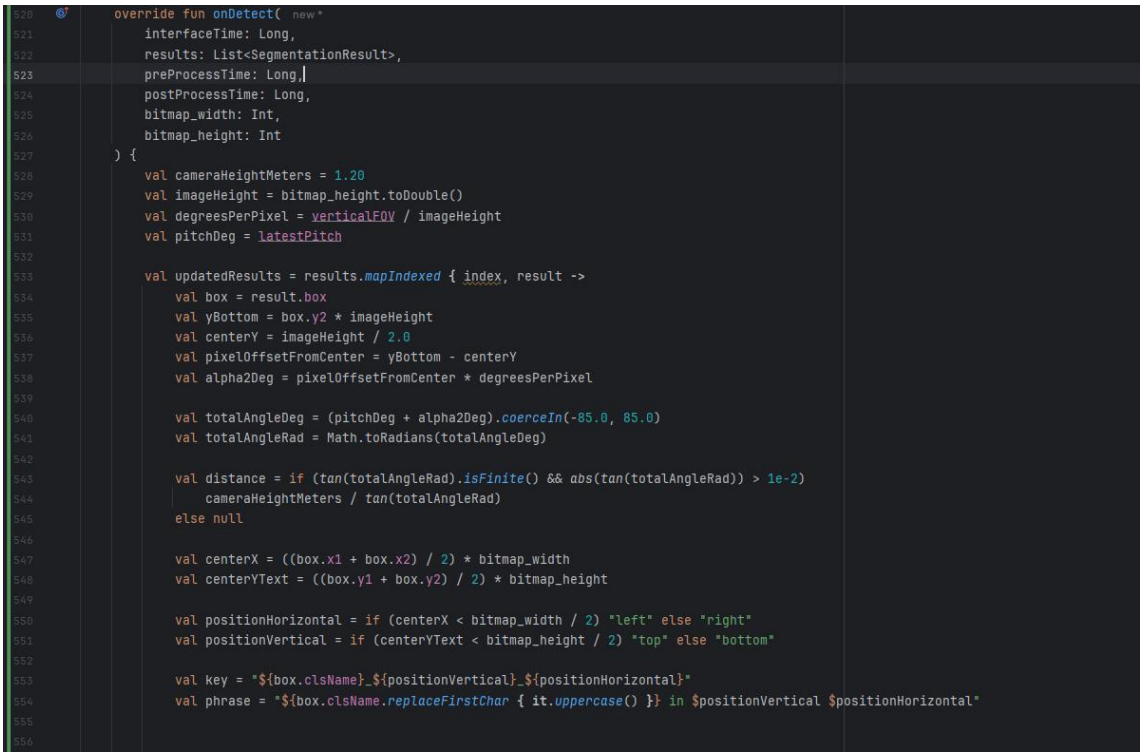
Εικόνα 5.27: Μέθοδος onResults

Μέθοδος onDetect (γραμμές 520-592):Η μέθοδος onDetect τρέχει μόλις ολοκληρωθεί η διαδικασία εντοπισμου αντικειμένων με segmentation. Ουσιαστικά είναι η μέθοδος που κάνει την “βαριά” δουλειά. Αρχικά, γίνονται υπολογισμοί για να βρούμε τη απόσταση κάθε αντικειμένου απο την κάμερα. Αυτό γίνεται με βάση το ύψος της κάμερας (1,20 μ.), το κατακόρυφο οπτικό πεδίο (verticalFOV), και τη γωνία κλίσης της συσκευής (latestPitch).
Για κάθε αποτέλεσμα, υπολογίζεται:

- Η θέση του αντικειμένου στην εικόνα .
- Η γωνία προς το αντικείμενο και από εκεί η εκτιμώμενη απόστασή του.
- Μια περιγραφική φράση, π.χ. *"Person in top left"*, η οποία εκφωνείται μέσω TTS αν δεν έχει ακουστεί πρόσφατα, ώστε να μην επαναλαμβάνεται συνεχώς.

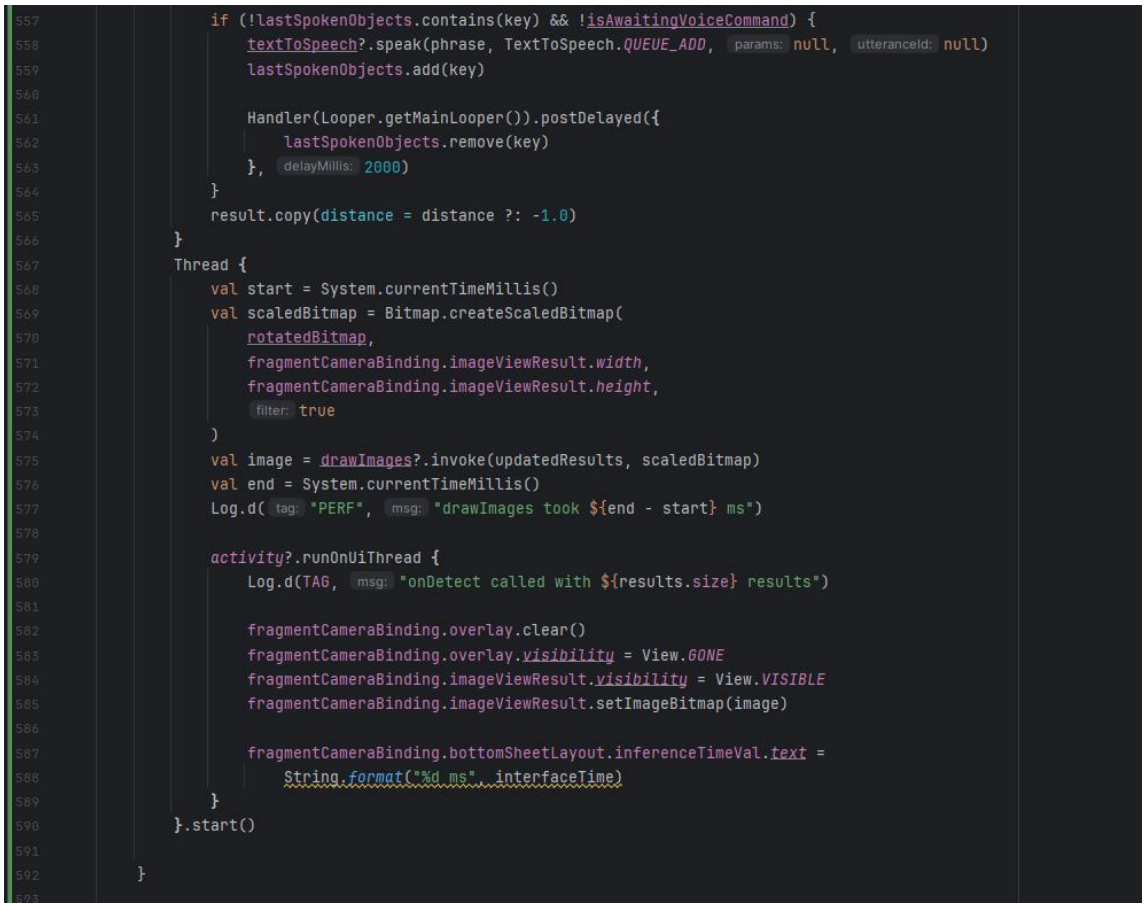
Αφού ενημερωθεί η λίστα με τα αποτελέσματα και τις αποστάσεις, σε ξεχωριστό νήμα δημιουργείται ένα νέο bitmap, στο οποίο σχεδιάζονται οι περιοχές που έχει ανιχνευθεί κάποιο αντικείμενο. Όταν η διαδικασία σχεδίασης ολοκληρωθεί, γίνεται ενημέρωση του UI στο κύριο νήμα:

- Καθαρίζεται το overlay.
- Εμφανίζεται η τελική εικόνα με τα σχεδιασμένα αποτελέσματα.
- Ενημερώνεται ο χρόνος επεξεργασίας (inference time) στο κάτω μέρος του μενού.



```
override fun onDetect( new*
    interfaceTime: Long,
    results: List<SegmentationResult>,
    preProcessTime: Long,
    postProcessTime: Long,
    bitmap_width: Int,
    bitmap_height: Int
) {
    val cameraHeightMeters = 1.20
    val imageHeight = bitmap_height.toDouble()
    val degreesPerPixel = verticalFOV / imageHeight
    val pitchDeg = latestPitch

    val updatedResults = results.mapIndexed { index, result ->
        val box = result.box
        val yBottom = box.y2 * imageHeight
        val centerY = imageHeight / 2.0
        val pixelOffsetFromCenter = yBottom - centerY
        val alpha2Deg = pixelOffsetFromCenter * degreesPerPixel

        val totalAngleDeg = (pitchDeg + alpha2Deg).coerceIn(-85.0, 85.0)
        val totalAngleRad = Math.toRadians(totalAngleDeg)

        val distance = if (tan(totalAngleRad).isFinite() && abs(tan(totalAngleRad)) > 1e-2)
            cameraHeightMeters / tan(totalAngleRad)
        else null

        val centerX = ((box.x1 + box.x2) / 2) * bitmap_width
        val centerYText = ((box.y1 + box.y2) / 2) * bitmap_height

        val positionHorizontal = if (centerX < bitmap_width / 2) "left" else "right"
        val positionVertical = if (centerYText < bitmap_height / 2) "top" else "bottom"

        val key = "${box.clsName}_${positionVertical}_${positionHorizontal}"
        val phrase = "${box.clsName.replaceFirstChar { it.uppercase() }} in $positionVertical $positionHorizontal"
```

Εικόνα 5.28: Μέθοδος onDetect



```
            if (!lastSpokenObjects.contains(key) && !isAwaitingVoiceCommand) {
                textToSpeech?.speak(phrase, TextToSpeech.QUEUE_ADD, params: null, utteranceId: null)
                lastSpokenObjects.add(key)

                Handler(Looper.getMainLooper()).postDelayed({
                    lastSpokenObjects.remove(key)
                }, delayMillis: 2000)
            }
            result.copy(distance = distance ?: -1.0)
        }
        Thread {
            val start = System.currentTimeMillis()
            val scaledBitmap = Bitmap.createScaledBitmap(
                rotatedBitmap,
                fragmentCameraBinding.imageViewResult.width,
                fragmentCameraBinding.imageViewResult.height,
                filter: true
            )
            val image = drawImages?.invoke(updatedResults, scaledBitmap)
            val end = System.currentTimeMillis()
            Log.d( tag: "PERF", msg: "drawImages took ${end - start} ms")

            activity?.runOnUiThread {
                Log.d(TAG, msg: "onDetect called with ${results.size} results")

                fragmentCameraBinding.overlay.clear()
                fragmentCameraBinding.overlay.visibility = View.GONE
                fragmentCameraBinding.imageViewResult.visibility = View.VISIBLE
                fragmentCameraBinding.imageViewResult.setImageBitmap(image)

                fragmentCameraBinding.bottomSheetLayout.inferenceTimeVal.text =
                    String.format("%d ms", interfaceTime)
            }
        }.start()

    }
```

Εικόνα 5.29: Μέθοδος onDetect

Μέθοδος onError (γραμμές 594-598): Η μέθοδος καλείται όταν παρουσιαστεί κάποιο σφάλμα κατά την εκτέλεση. Το μήνυμα λάθους εμφανίζεται στον χρήστη μέσω ενός Toast στο κύριο νήμα της εφαρμογής, ώστε να είναι άμεσα ορατό χωρίς να επηρεάζεται η ροή του υπόλοιπου κώδικα.

```
593
594     override fun onError(error: String) {  Khanh LeViet
595         activity?.runOnUiThread {
596             Toast.makeText(requireContext(), error, Toast.LENGTH_SHORT).show()
597         }
598     }
```

Εικόνα 5.30: Μέθοδος onError

Μέθοδος getVerticalFOV (γραμμές 602-612): Η μέθοδος υπολογίζει το κάθετο πεδίο θέασης της κάμερας. Αρχικά λαμβάνει τα χαρακτηριστικά της κάμερας, στην συνέχεια αποκτά το εστιακό μήκος και το φυσικό μέγεθος του αισθητήρα. Χρησιμοποιεί το πρώτο διαθέσιμο εστιακό μήκος και το ύψος του αισθητήρα για τον υπολογισμό. Και στο τέλος το αποτέλεσμα μετατρέπεται σε μοίρες και επιστρέφεται.

```
601
602     fun getVerticalFOV(cameraManager: CameraManager, cameraId: String): Double {  new *
603         val characteristics = cameraManager.getCameraCharacteristics(cameraId)
604         val focalLengths = characteristics.get(CameraCharacteristics.LENS_INFO_AVAILABLE_FOCAL_LENGTHS)
605         val sensorSize = characteristics.get(CameraCharacteristics.SENSOR_INFO_PHYSICAL_SIZE)
606
607         val focalLength = focalLengths?.getOrNull( index: 0) ?: return 60.0
608         val height = sensorSize?.height ?: return 60.0
609
610         val fovRadians = 2 * atan( x: height / (2 * focalLength))
611         return Math.toDegrees(fovRadians.toDouble())
612     }
```

Εικόνα 5.31: Μέθοδος getVerticalFOV

Μέθοδος onSensorChanged (γραμμές 618-639): Η μέθοδος παρακολουθεί τις αλλαγές από τους αισθητήρες της συσκευής. Ειδικά εδώ ασχολούμαστε με τον αισθητήρα τύπου ROTATION_VECTOR, ο οποίος δίνει πληροφορίες για τον προσανατολισμό της συσκευής.

```
618     override fun onSensorChanged(event: SensorEvent) {  new *
619         if (event.sensor.type == Sensor.TYPE_ROTATION_VECTOR) {
620             val rotationMatrix = FloatArray( size: 9)
621             val adjustedRotationMatrix = FloatArray( size: 9)
622             val orientationAngles = FloatArray( size: 3)
623
624             SensorManager.getRotationMatrixFromVector(rotationMatrix, event.values)
625
626             SensorManager.remapCoordinateSystem(
627                 rotationMatrix,
628                 SensorManager.AXIS_X,
629                 SensorManager.AXIS_Z,
630                 adjustedRotationMatrix
631             )
632
633             SensorManager.getOrientation(adjustedRotationMatrix, orientationAngles)
634
635             val pitch = Math.toDegrees(orientationAngles[1].toDouble())  // pitch σε °
636
637             latestPitch = pitch
638         }
639     }
```

Εικόνα 5.32: Μέθοδος onSensorChanged

5.7.2 Περιγραφή Κλάσης “PermissionFragment”

Σκοπός της κλάσης αυτής ειναι η διαχείριση των αδειών που χρειάζεται η εφαρμογή, όπως η πρόσβαση στην κάμερα. Δηλαδή αυτό το fragment εμφανίζεται οταν η εφαρμογή θέλει να ζητήσει άδεια για να λειτουργήσει.

Δήλωση Πακέτου(γραμμή 17): Δηλώνεται το πακέτο στο οποίο ανήκει η κλάση PermissionsFragment.

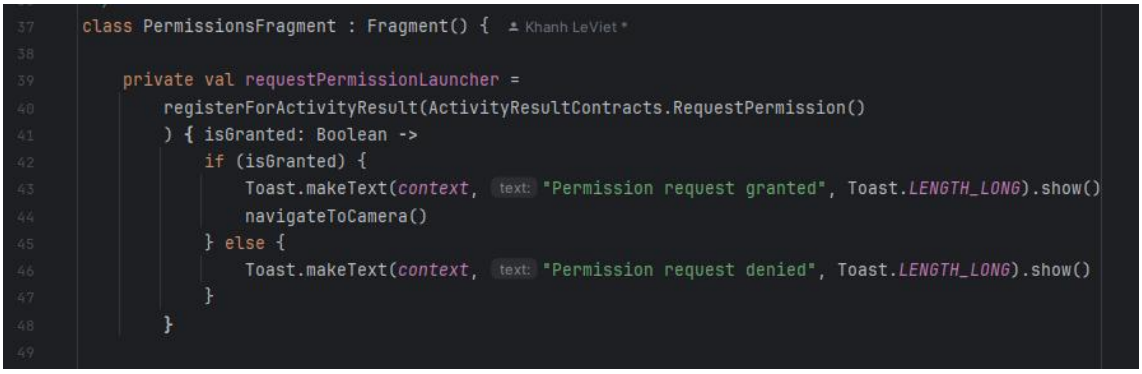
Imports(γραμμές 19-29): Σε αυτό το τμήμα, ο κώδικας εισάγει απαραίτητες βιβλιοθήκες που χρειάζεται η εφαρμογή για να λειτουργήσει.

Δήλωση αδειών(γραμμη 31): Με τη σταθερά PERMISSIONS_REQUIRED καθορίζουμε ποιες άδειες χρειάζεται η εφαρμογή για να λειτουργήσει.

```
17  package org.tensorflow.lite.examples.objectdetection.fragments
18
19  import android.Manifest
20  import android.content.Context
21  import android.content.pm.PackageManager
22  import android.os.Bundle
23  import android.widget.Toast
24  import androidx.activity.result.contract.ActivityResultContracts
25  import androidx.core.content.ContextCompat
26  import androidx.fragment.app.Fragment
27  import androidx.lifecycle.lifecycleScope
28  import androidx.navigation.Navigation
29  import org.tensorflow.lite.examples.objectdetection.R
30
31  private val PERMISSIONS_REQUIRED = arrayOf(Manifest.permission.CAMERA)
32
```

Εικόνα 5.33: Imports

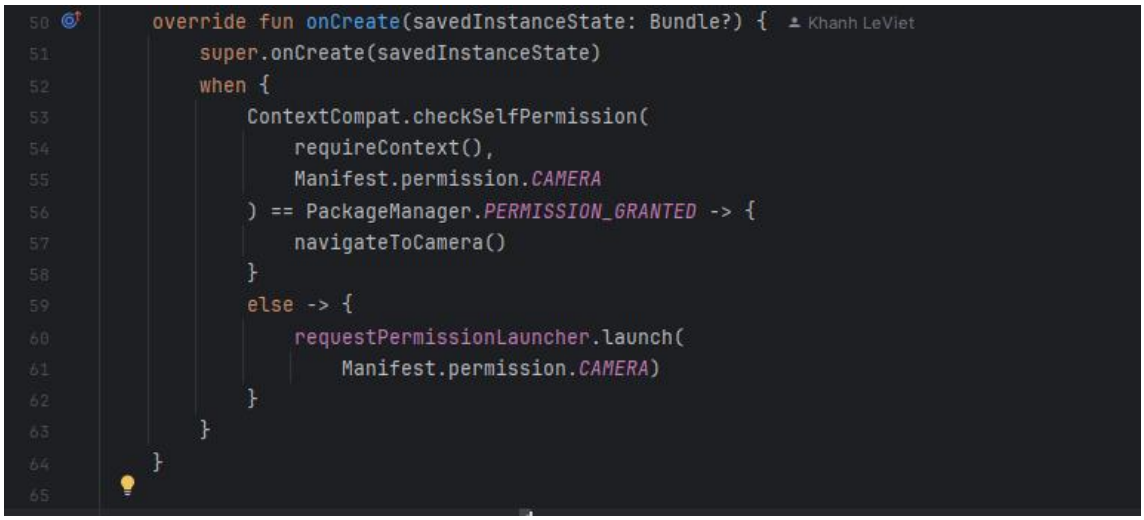
Μέθοδος requestPermissionLauncher (γραμμές 39-48): Αυτή η μέθοδος είναι ένας launcher που χρησιμοποιείται για την έναρξη της διαδικασίας αίτησης άδειας. Αν ο χρήστης δώσει την άδεια, εμφανίζει ένα μήνυμα επιβεβαίωσης και καλεί τη navigateToCamera. Αν αρνηθεί, εμφανίζει ένα μήνυμα άρνησης.



Εικόνα 5.34: Δηλώσεις Μεταβλητών

Μέθοδος onCreate (γραμμές 39-48): Αυτή η μέθοδος καλείται όταν δημιουργείται το fragment. Ελέγχει εάν η άδεια χρήσης της κάμερας έχει ήδη δοθεί:

- Αν ναι, καλεί τη μέθοδο navigateToCamera για να μεταβεί στο fragment της κάμερας.
- Αν όχι, καλεί τη requestPermissionLauncher για να ζητήσει την άδεια από τον χρήστη.

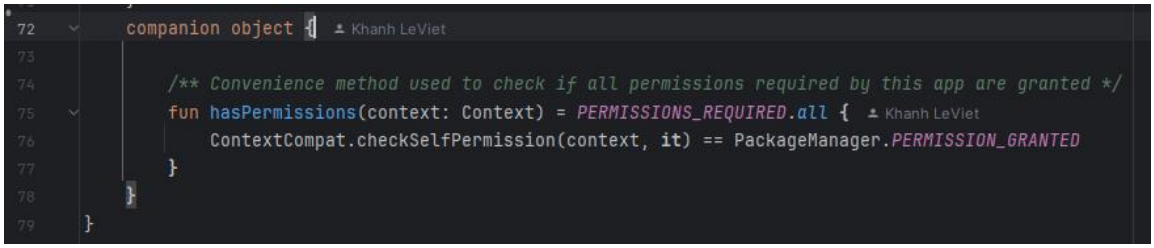


Εικόνα 5.35: Μέθοδος onCreate

Μέθοδος navigateToCamera (γραμμές 66-71): Αυτή η μέθοδος χρησιμοποιεί το Navigation για να μεταβεί από το fragment των αδειών στο fragment της κάμερας.

Εικόνα 5.36: Μέθοδος navigateToCamera

Μέθοδος hasPermissions (γραμμές 75-77): Αυτή η static μέθοδος είναι μια βοηθητική συνάρτηση που ελέγχει εάν όλες οι απαιτούμενες άδειες έχουν δοθεί.



Εικόνα 5.37: Μέθοδος hasPermission

5.7.3 Περιγραφή Κλάσης "MainActivity"

Σκοπός της MainActivity είναι η αρχική ρύθμιση της εφαρμογής. Συνοπτικά φορτώνει το κύριο layout της εφαρμογής και λειτουργεί ως container για τα άλλα fragments. Δηλαδή δεν υλοποιεί άμεσα λειτουργίες ανίχνευσης αντικειμένων αλλά παρέχει το "κάδρο" όπου τα fragments εκτελούν την κύρια δουλειά τους.

Δήλωση Πακέτου(γραμμή 17): Δηλώνεται το πακέτο στο οποίο ανήκει η κλάση MainActivity.

Imports(γραμμές 19-22): Σε αυτό το τμήμα, ο κώδικας εισάγει απαραίτητες βιβλιοθήκες που χρειάζεται η εφαρμογή για να λειτουργήσει.

```
16
17  package org.tensorflow.lite.examples.objectdetection
18
19  import android.os.Build
20  import android.os.Bundle
21  import androidx.appcompat.app.AppCompatActivity
22  import org.tensorflow.lite.examples.objectdetection.databinding.ActivityMainBinding
23
24  /**
25   * Main entry point into our app. This app follows the single-activity pattern, and all
26   * functionality is implemented in the form of fragments.
27   */
28  class MainActivity : AppCompatActivity() {  Khanh LeViet
29
30      private lateinit var activityMainBinding: ActivityMainBinding
31
32      override fun onCreate(savedInstanceState: Bundle?) {  Khanh LeViet
33          super.onCreate(savedInstanceState)
34          activityMainBinding = ActivityMainBinding.inflate(layoutInflater)
35          setContentView(activityMainBinding.root)
36      }
37
38      override fun onBackPressed() {  Khanh LeViet
39          if (Build.VERSION.SDK_INT == Build.VERSION_CODES.Q) {
40              // Workaround for Android Q memory leak issue in IRequestFinishCallback$Stub.
41              // (https://issuetracker.google.com/issues/139738913)
42              finishAfterTransition()
43          } else {
44              super.onBackPressed()
45          }
46      }
47  }
48
```

Εικόνα 5.38: Imports, Μεταβλητές, Μέθοδοι onCreate & onbackPressed

5.7.4 Περιγραφή Κλάσης “ObjectDetectorHelper”

Η κλάση ObjectDetectorHelper παρέχει μεθόδους για την αρχικοποίηση, ρυθμίσεις και χρήση ενός ανιχνευτή αντικειμένων.Χρησιμοποιεί ObjectDetector για να ανιχνεύσει αντικείμενα σε εικόνες και ενημερώνει ένα αντικείμενο DetectorListener για σφάλματα και αποτελέσματα. Επίσης παρέχει συμβατότητα με διαφορετικά pipelines όπως το YOLO και ειδοποιεί έναν listener με τα αποτελέσματα ανιχνευσης, τον χρόνο επεξεργασίας και τις διαστάσεις της εικόνας.

Δήλωση Πακέτου(γραμμή 16): Δηλώνεται το πακέτο στο οποίο ανήκει η κλάση ObjectDetectorHelper.

Imports(γραμμές 18-28): Σε αυτό το τμήμα, ο κώδικας εισάγει απαραίτητες βιβλιοθήκες που χρειάζεται η εφαρμογή για να λειτουργήσει.

```
16  package org.tensorflow.lite.examples.objectdetection
17
18  import android.content.Context
19  import android.graphics.Bitmap
20  import android.os.SystemClock
21  import android.util.Log
22  import org.tensorflow.lite.gpu.CompatibilityList
23  import org.tensorflow.lite.support.image.ImageProcessor
24  import org.tensorflow.lite.support.image.TensorImage
25  import org.tensorflow.lite.support.image.ops.Rot90Op
26  import org.tensorflow.lite.task.core.BaseOptions
27  import org.tensorflow.lite.task.vision.detector.Detection
28  import org.tensorflow.lite.task.vision.detector.ObjectDetector
29
```

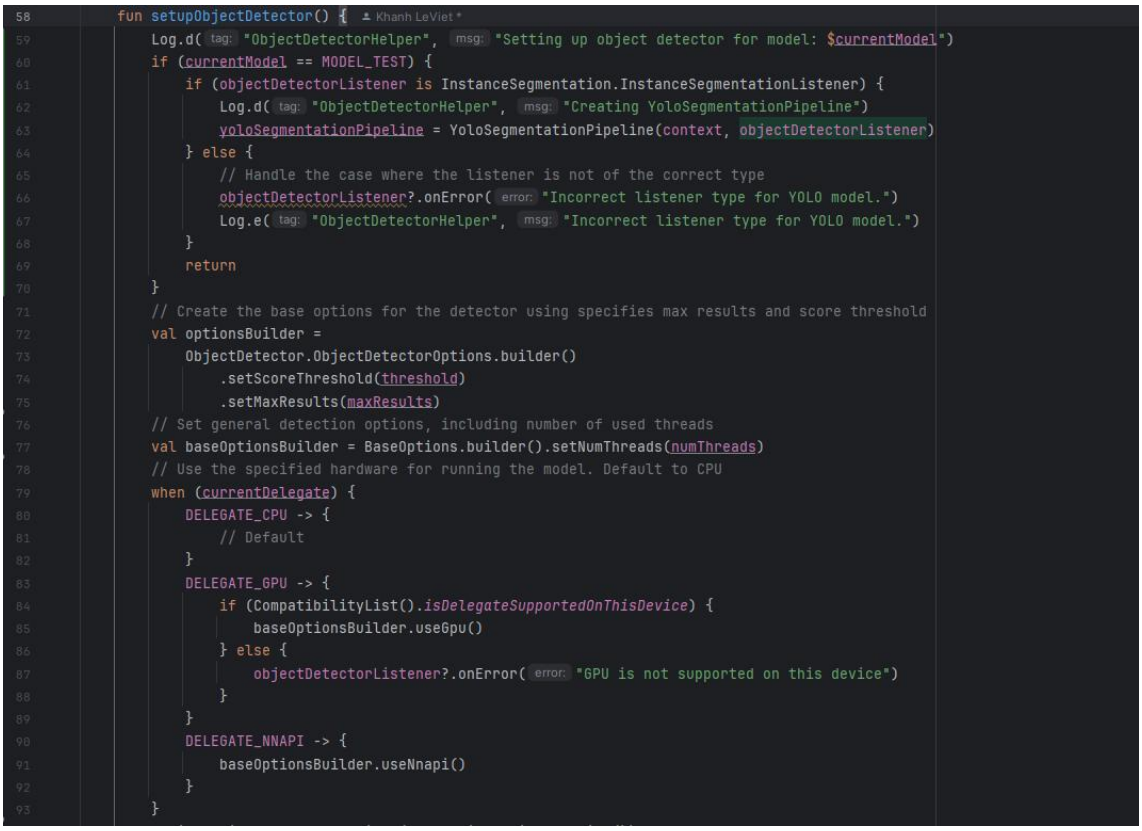
Εικόνα 5.39: Imports

Μετά τα imports εχουμε την δήλωση των βασικλων παραμέτρων και μεταβλητών. Επίσης στίς γραμμές 42,43 βλέπουμε δύο ιδιωτικά αντικείμενα (το objectDetector που είναι ο βασικός TensorFlow Lite object detector και το yoloSegmentationPipeline που ειναι το pipeline για το YOLO). Παρακάτω μέσα στο init καλείται η μέθοδος setupObjectDetector(), η οποία αρχικοποιεί τον ανιχνευτεί και τέλος η clearObjectDetector η οποία καθαρίζει τους πόρους οταν χρεαιάζεται.

```
30  class ObjectDetectorHelper(   Khanh LeViet *
31    var threshold: Float = 0.5f,
32    var numThreads: Int = 2,
33    var maxResults: Int = 3,
34    var currentDelegate: Int = 0,
35    var currentModel: Int = 0,
36    val context: Context,
37    val objectDetectorListener: DetectorListener?
38  ) {
39
40      // For this example this needs to be a var so it can be reset on changes. If the ObjectDetector
41      // will not change, a lazy val would be preferable.
42      private var objectDetector: ObjectDetector? = null
43      private var yoloSegmentationPipeline: YoloSegmentationPipeline? = null
44
45      init {   Khanh LeViet
46          setupObjectDetector()
47      }
48
49      fun clearObjectDetector() {   Khanh LeViet *
50          objectDetector = null
51          yoloSegmentationPipeline = null
52      }
```

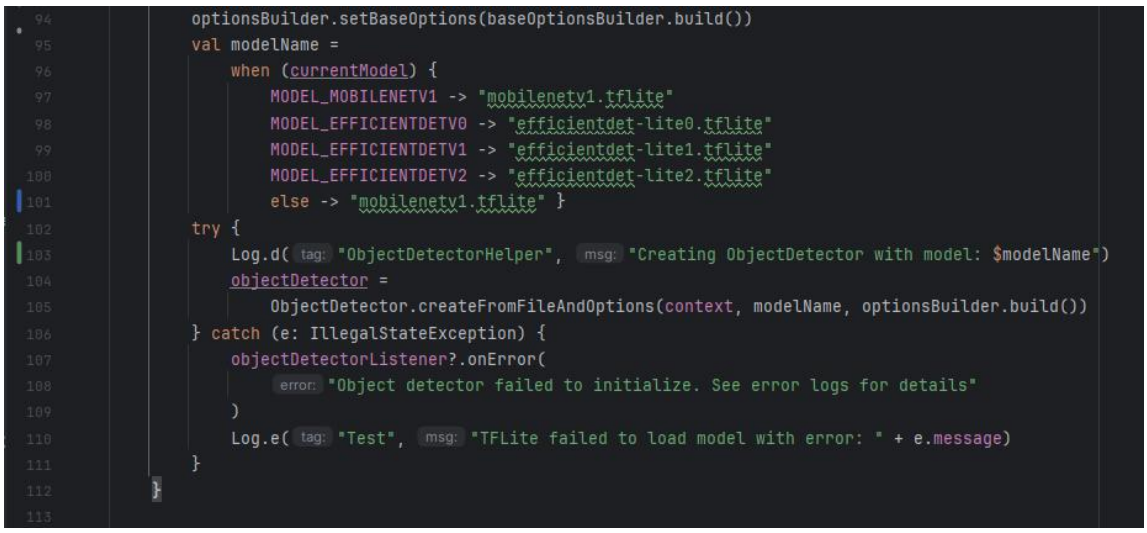
Εικόνα 5.40: Δηλώσεις μεταβλητών και μέθοδος clearObjectDetector

Στην συνέχεια έχουμε την **μέθοδο setupOnjectDetector() (γραμμές 58-112)** στην οποία γίνεται ο έλεεγχος μοντέλου δηλαδή αν το currentModel είναι το MODEL_TEST (YOLO) τότε δημιουργείται το YoloSegmentationPipeline και ο listener πρεπει να είναι τύπου InstanceSegmentationListener. Αν οχι επιστρέφεται μήνυμα λάθους. Στην συνέχεια δημιουργείται ένα ObjectDetectorOptions.builder() και ορίζονται οι παράμετροι threshold και maxResults. ‘Επειτα το BaseOptions.builder() ορίζει τον αριθμό threads που θα χρησιμοποιηθούν (numThreads) και τον hardware delegate (CPU, GPU, NNAPI). Να αναφερθεί πως υπάρχει έλεγχος για GPU υποστήριξη στη συσκευή με fallback μήνυμα λάθους αν δεν υποστηρίζεται. Μετά επιλέγεται το κατάλληλο αρχείο .tflite ανάλογα με το currentModel. Τέλος ο TensorFlow Lite ObjectDetector δημιουργείται από το αρχείο μοντέλου και τις ρυθμίσεις. Σε περίπτωση αποτυχίας, καλείται ο listener με μήνυμα λάθους και καταγράφεται στο log.



```
fun setupObjectDetector() {  Khanh LeViet *
    Log.d( tag: "ObjectDetectorHelper",  msg: "Setting up object detector for model: $currentModel")
    if (currentModel == MODEL_TEST) {
        if (objectDetectorListener is InstanceSegmentation.InstanceSegmentationListener) {
            Log.d( tag: "ObjectDetectorHelper",  msg: "Creating YoloSegmentationPipeline")
            yoloSegmentationPipeline = YoloSegmentationPipeline(context, objectDetectorListener)
        } else {
            // Handle the case where the listener is not of the correct type
            objectDetectorListener?.onError( error: "Incorrect listener type for YOLO model.")
            Log.e( tag: "ObjectDetectorHelper",  msg: "Incorrect listener type for YOLO model.")
        }
        return
    }
    // Create the base options for the detector using specifies max results and score threshold
    val optionsBuilder =
        ObjectDetector.ObjectDetectorOptions.builder()
            .setScoreThreshold(threshold)
            .setMaxResults(maxResults)
    // Set general detection options, including number of used threads
    val baseOptionsBuilder = BaseOptions.builder().setNumThreads(numThreads)
    // Use the specified hardware for running the model. Default to CPU
    when (currentDelegate) {
        DELEGATE_CPU -> {
            // Default
        }
        DELEGATE_GPU -> {
            if (CompatibilityList().isDelegateSupportedOnThisDevice) {
                baseOptionsBuilder.useGpu()
            } else {
                objectDetectorListener?.onError( error: "GPU is not supported on this device")
            }
        }
        DELEGATE_NNAPI -> {
            baseOptionsBuilder.useNnapi()
        }
    }
```

Εικόνα 5.41: Μέθοδος setupObjectDetector



```
    optionsBuilder.setBaseOptions(baseOptionsBuilder.build())
    val modelName =
        when (currentModel) {
            MODEL_MOBILENETV1 -> "mobilenetv1.tflite"
            MODEL_EFFICIENTDETV0 -> "efficientdet-lite0.tflite"
            MODEL_EFFICIENTDETV1 -> "efficientdet-lite1.tflite"
            MODEL_EFFICIENTDETV2 -> "efficientdet-lite2.tflite"
            else -> "mobilenetv1.tflite" }
    try {
        Log.d( tag: "ObjectDetectorHelper",  msg: "Creating ObjectDetector with model: $modelName")
        objectDetector =
            ObjectDetector.createFromFileAndOptions(context, modelName, optionsBuilder.build())
    } catch (e: IllegalStateException) {
        objectDetectorListener?.onError(
            error: "Object detector failed to initialize. See error logs for details"
        )
        Log.e( tag: "Test",  msg: "TFLite failed to load model with error: " + e.message)
    }
}
```

Εικόνα 5.42: Μέθοδος setupObjectDetector

Επόμενη μέθοδος η detect() (γραμμές 114-152) η οποία είναι υπεύθυνη για την εκτέλεση της ανίχνευσης αντικειμένων πάνω σε μια εικόνα. Αναλυτικότερα αν το currentModel ειναι YOLO, ελέγχει αν η yoloSegmentationPipeline έχει αρχικοποιηθεί, αλλιώς καλεί το setupObjectDetector() και στην συνέχεια εκτελεί την ανίχνευση μέσω του yoloSegmentationPipeline και επιστρέφει. Αν ο objectDetector δεν έχει αρχικοποιηθεί, το αρχικοποιεί με setupObjectDetector(). Έπειτα καταγράφει το χρόνο έναρξης με SystemClock.uptimeMillis() για να υπολογίσει τη διάρκεια εκτέλεσης του

μοντέλου. Στην συνέχεια κάνει προεπεξεργασία της εικόνας και μετλα καλεί τον objectDetector.detect() με την επεξεργασμένη εικόνα και παίρνει τα αποτελέσματα. Τέλος στέλνει τα αποτελέσματα στον listener περνώντας τη λίστα ανιχνεύσεων, τον χρόνο εκτέλεσης και το ύψος και το πλάτος της εικόννας.

```
114     fun detect(image: Bitmap, imageRotation: Int) {  Khanh LeViet *
115         //Log.d("ObjectDetectorHelper", "Detecting objects for model: $currentModel")
116         if (currentModel == MODEL_TEST) {
117             if (yoloSegmentationPipeline == null) {
118
119                 setupObjectDetector()
120             }
121             //Log.d("ObjectDetectorHelper", "Using YoloSegmentationPipeline for detection")
122             yoloSegmentationPipeline?.detect(image, imageRotation)
123             return
124         }
125
126         if (objectDetector == null) {
127             setupObjectDetector()
128         }
129
130         // Inference time is the difference between the system time at the start and finish of the
131         // process
132         var inferenceTime = SystemClock.uptimeMillis()
133
134         // Create preprocessor for the image.
135         // See https://www.tensorflow.org/lite/inference_with_metadata/
136         //            lite_support#imageprocessor_architecture
137         val imageProcessor =
138             ImageProcessor.Builder()
139                 .add(Rot90Op( k: -imageRotation / 90))
140                 .build()
141
142         // Preprocess the image and convert it into a TensorImage for detection.
143         val tensorImage = imageProcessor.process(TensorImage.fromBitmap(image))
144
145         val results = objectDetector?.detect(tensorImage)
146         inferenceTime = SystemClock.uptimeMillis() - inferenceTime
147         objectDetectorListener?.onResults(
148             results,
149             inferenceTime,
150             tensorImage.height,
151             tensorImage.width)
152     }
153
```

Εικόνα 5.43: Μέθοδος detect

Οι επόμενοι 3 μέθοδοι (updateYoloThreshold,updateYoloThreads,updateYoloMaxResults)(γραμμές 154-164) παρέχουν δυναμική ρύθμιση παραμέτρων για το YOLO χωρίς να χρειάζεται επανεκκίνηση της εφαρμογής.

```
    fun updateYoloThreshold(threshold: Float) {  new *
        yoloSegmentationPipeline?.updateThreshold(threshold)
    }

    fun updateYoloThreads(threads: Int) {  new *
        yoloSegmentationPipeline?.updateThreads(threads)
    }

    fun updateYoloMaxResults(maxResults: Int) {  new *
        yoloSegmentationPipeline?.updateMaxResults(maxResults)
    }
```

Εικόνα 5.44: Μέθοδος updateYoloThreshold,updateYoloThreads,updateYoloMaxResults

Σε αυτό το τελευταίο κομμάτι έχουμε ένα interface επικοινωνίας ανάμεσα στον ανιχνευτή αντικειμένωνκαι την υπόλοιπη εφαρμογή. Η onResults καλείται ότα υπάρχουν αποτελέσματα απο το object detection. Η onEmpty() για την περίπτωση που δεν εντοπιστούν αντικείμενα και η onDetect είναι κενή ώστε να μπορεί να χρησιμοποιηθεί αν χρειαστεί. Τέλος στο companion object ορίζονται σταθερές τιμές για την επιλογή υλικού και μοντέλων.

```
interface DetectorListener : InstanceSegmentation.InstanceSegmentationListener {  Khanh LeViet *
    fun onResults(  Khanh LeViet *
        results: MutableList<Detection>?,
        inferenceTime: Long,
        imageHeight: Int,
        imageWidth: Int
    )

    override fun onEmpty() {}  new *

    override fun onDetect(  new *
        interfaceTime: Long,
        results: List<SegmentationResult>,
        preProcessTime: Long,
        postProcessTime: Long,
        bitmap_width: Int,
        bitmap_height: Int
    ) {

    }

}

companion object {  Khanh LeViet *
    const val DELEGATE_CPU = 0
    const val DELEGATE_GPU = 1
    const val DELEGATE_NNAPI = 2
    const val MODEL_MOBILENETV1 = 0
    const val MODEL_EFFICIENTDETV0 = 1
    const val MODEL_EFFICIENTDETV1 = 2
    const val MODEL_EFFICIENTDETV2 = 3
    const val MODEL_TEST = 4
}
}
```

Εικόνα 5.45: Μέθοδος onResults, onEmpty, onDetect, Companion Object

5.7.5 Περιγραφή Κλάσης “OverlayView”

Ο ρόλος της OverlayView είναι να σψεδιάζει τα ορθογώνια περιγράμματα γύρω από τα αντικείμενα που έχουν ανιχνευτοί, να εμφανίζει ετικέτες με την κατηγορία και την ακρίβεια. Με αυτόν τον τρόπο, η OverlayView παρουσιάζει στον χρήστη τα αποτελέσματα του συστήματος ανίχνευσης.

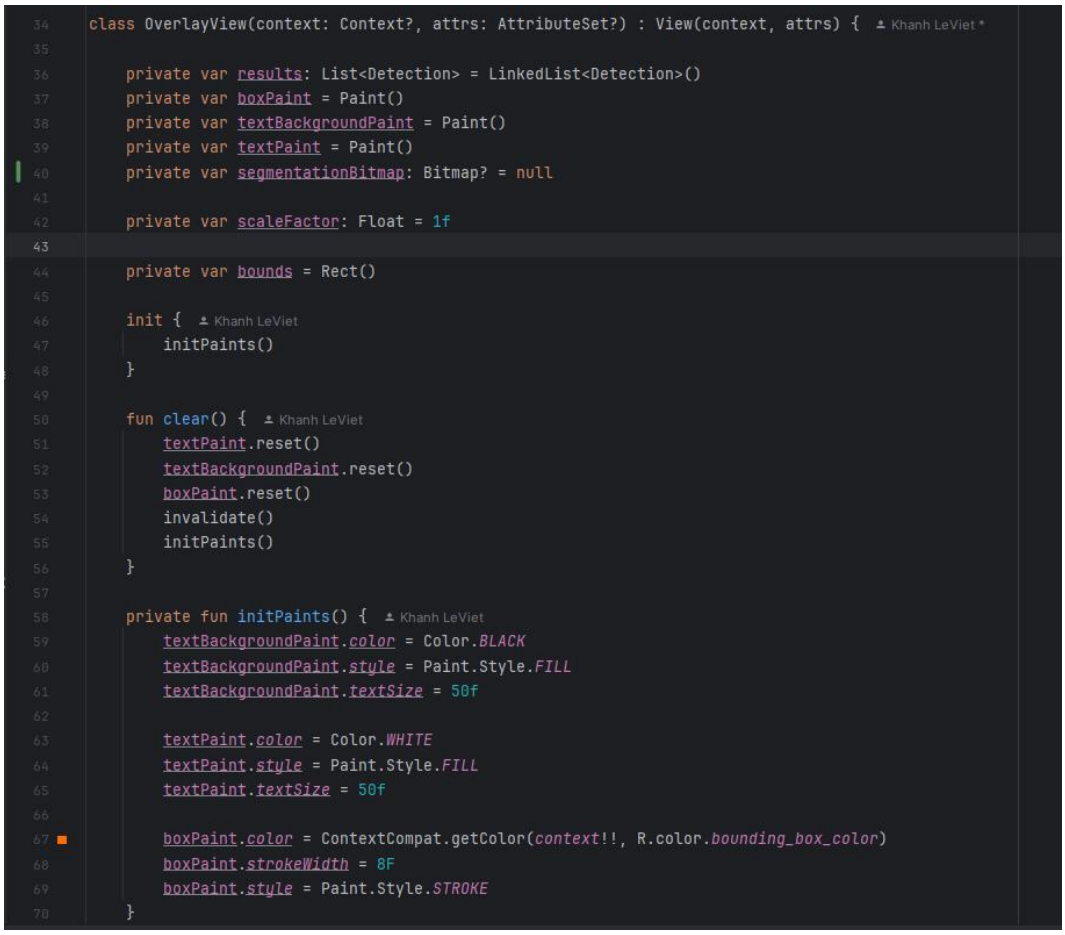
Δήλωση Πακέτου(γραμμή 17): Δηλώνεται το πακέτο στο οποίο ανήκει η κλάση OverlayView.

Imports(γραμμές 19-32): Σε αυτό το τμήμα, ο κώδικας εισάγει απαραίτητες βιβλιοθήκες που χρειάζεται η εφαρμογή για να λειτουργήσει.

```
17  package org.tensorflow.lite.examples.objectdetection
18
19  import android.content.Context
20  import android.graphics.Bitmap
21  import android.graphics.Canvas
22  import android.graphics.Color
23  import android.graphics.Paint
24  import android.graphics.Rect
25  import android.graphics.RectF
26  import android.util.AttributeSet
27  import android.util.Log
28  import android.view.View
29  import androidx.core.content.ContextCompat
30  import java.util.LinkedList
31  import kotlin.math.max
32  import org.tensorflow.lite.task.vision.detector.Detection
33
34  class OverlayView(context: Context?, attrs: AttributeSet?) : View(context, attrs) {  Khanh LeViet *
```

Εικόνα 5.46: Imports

Στο παρακάτω τμήμα του κώδικα η κλάση αρχικοποιεί τα απαραίτητα στοιχεία για να μπορεί να σχεδιάζει τα κουτιά πάνω στην εικόνα. Στην συνέχεια καλεί την initPaints() για να ρυθμίσει τα χρώματα, το στύλ και το μέγεθος. Μετά έχουμε τις μεθόδους clear() και initPaints(). Η clear() καθαρίζει τα paint αντικείμενα, κάνει reset τις ρυθμίσεις και καλεί invalidate() για να ανανεωθεί η προβολή. Μετά η initPaints() βάζει τις αρχικές ιδιότητες στα Paint.



```
class OverlayView(context: Context?, attrs: AttributeSet?) : View(context, attrs) {  Khanh LeViet *

    private var results: List<Detection> = LinkedList<Detection>()
    private var boxPaint = Paint()
    private var textBackgroundPaint = Paint()
    private var textPaint = Paint()
    private var segmentationBitmap: Bitmap? = null

    private var scaleFactor: Float = 1f

    private var bounds = Rect()

    init {  Khanh LeViet
        initPaints()
    }

    fun clear() {  Khanh LeViet
        textPaint.reset()
        textBackgroundPaint.reset()
        boxPaint.reset()
        invalidate()
        initPaints()
    }

    private fun initPaints() {  Khanh LeViet
        textBackgroundPaint.color = Color.BLACK
        textBackgroundPaint.style = Paint.Style.FILL
        textBackgroundPaint.textSize = 50f

        textPaint.color = Color.WHITE
        textPaint.style = Paint.Style.FILL
        textPaint.textSize = 50f

        boxPaint.color = ContextCompat.getColor(context!!, R.color.bounding_box_color)
        boxPaint.strokeWidth = 8F
        boxPaint.style = Paint.Style.STROKE
    }
```

Εικόνα 5.47: Δηλώσεις μεταβλητών, Μέθοδος clear & initPaints

Το επόμενο κομμάτι κώδικα είναι το πιο σημαντικό της κλάσης γιατι εκεί γίνεται το πραγματικό σχεδίασμα των αποτελεσμάτων στην οθόνη του χρήστη. Η super.draw(canvas) καλεί την βασική μέθοδο της κλάσης View για να διατηρηθούν τυχόν ήδη υπάρχοντα σχέδια. Μετά γίνεται έλεγχος για segmentaion εικόνα αλλιώς σχεδιάζει bounding boxes και labels.

```
override fun draw(canvas: Canvas) {  Khanh LeViet *
    super.draw(canvas)

    segmentationBitmap?.let {
        canvas.drawBitmap(it,  left: 0f,  top: 0f,  paint: null)
    } ?: run {
        for (result in results) {
            val boundingBox = result.boundingBox

            val top = boundingBox.top * scaleFactor
            val bottom = boundingBox.bottom * scaleFactor
            val left = boundingBox.left * scaleFactor
            val right = boundingBox.right * scaleFactor

            // Draw bounding box around detected objects
            val drawableRect = RectF(left, top, right, bottom)
            canvas.drawRect(drawableRect, boxPaint)

            // Create text to display alongside detected objects
            val drawableText =
                result.categories[0].label + " " +
                        String.format("%.2f", result.categories[0].score)

            // Draw rect behind display text
            textBackgroundPaint.getTextBounds(drawableText,  start: 0, drawableText.length, bounds)
            val textWidth = bounds.width()
            val textHeight = bounds.height()
            canvas.drawRect(
                left,
                top,
                 right: left + textWidth + Companion.BOUNDING_RECT_TEXT_PADDING,
                 bottom: top + textHeight + Companion.BOUNDING_RECT_TEXT_PADDING,
                textBackgroundPaint
            )

            // Draw text for detected object
            canvas.drawText(drawableText, left,  y: top + bounds.height(), textPaint)
        }
    }
}
```

Εικόνα 5.48: Μέθοδος draw

Στην συνέχεια έχουμε το κομμάτι που “ταίζει” την OverlayView με τα νέα αποτελέσματα ανίχνευσης ώστε να τα σχεδιάσει στην οθόνη. Ο ρόλος της setResults() είναι να συνδέσει τον μηχανισμό ανίχνευσεις αντικειμένων με το UI. Χωρίς αυτήν, η draw() δεν θα ήξερε τι να σχεδιάσει ή σε τί μέγεθος.

```
112
113     fun setResults(  Khanh LeViet *
114       detectionResults: MutableList<Detection>,
115       imageHeight: Int,
116       imageWidth: Int,
117     ) {
118         results = detectionResults
119         segmentationBitmap = null
120
121         // PreviewView is in FILL_START mode. So we need to scale up the bounding box to match with
122         // the size that the captured images will be displayed.
123         scaleFactor = max( a: width * 1f / imageWidth,  b: height * 1f / imageHeight)
124     }
125
126
127     companion object {  Khanh LeViet
128         private const val BOUNDING_RECT_TEXT_PADDING = 8
129     }
130 }
131
```

Εικόνα 5.49: Μέθοδος setResults

5.7.6 Περιγραφή Κλάσης “DetectionPipeline”

Η παρακάτω κλάση είναι **interface** και παίζει τον ρόλο ενός «συμβολαίου» για οποιαδήποτε υλοποίηση μηχανισμού ανίχνευσης. Λειτουργεί σαν κοινό σημείο αναφοράς, το χρησιμοποιεί η YoloSegmentationPipeline ή ο ObjectDetectorHelper για να εξασφαλίσει οτι οποιοσδήποτε αγωγός ανίχνευσης θα έχει τις βασικές λειτουργίες detect και clear.

```
1   package org.tensorflow.lite.examples.objectdetection
2
3   import android.graphics.Bitmap
4
5   interface DetectionPipeline {  new *
6       fun detect(bitmap: Bitmap, rotation: Int)  new *
7       fun clear()  new *
8   }
9
```

Εικόνα 5.50: Imports, μέθοδος detect & clear

5.7.7 Περιγραφή Κλάσης “DrawImages”

Η κλάση DrawImages είναι υπεύθυνη για να παίρνει μια εικόνα και τα αποτελέσματα segmentation και να «ζωγραφίζει» πάνω τους, ώστε να φαίνονται οι περιοχές ανίχνευσης με χρωματιστές μάσκες, πλαίσια και ετικέτες.

Δήλωση Πακέτου(γραμμή 1): Δηλώνεται το πακέτο στο οποίο ανήκει η κλάση DrawImages.

Imports(γραμμές 3-10): Σε αυτό το τμήμα, ο κώδικας εισάγει απαραίτητες βιβλιοθήκες που χρειάζεται η εφαρμογή για να λειτουργήσει.

```
1  package org.tensorflow.lite.examples.objectdetection
2
3  import android.content.Context
4  import android.graphics.Bitmap
5  import android.graphics.Canvas
6  import android.graphics.Color
7  import android.graphics.Paint
8  import android.graphics.Typeface
9  import android.util.Log
10 import androidx.core.content.ContextCompat
11
12 class DrawImages(private val context: Context) {  new *
13
```

Εικόνα 5.51: Imports

Αρχικά η κλάση δέχεται ένα Context από το Android, ώστε να μπορεί να έχει πρόσβαση σε πόρους όπως χρώματα. Στην συνέχεια η λίστα boxColors περιέχει τα χρώματα που θα χρησιμοποιηθούν για τον σχεδιασμό των κουτιων.

```
12 class DrawImages(private val context: Context) {  new *
13
14     private val boxColors = listOf(
15         R.color.overlay_orange,
16         R.color.overlay_blue,
17         R.color.overlay_green,
18         R.color.overlay_red,
19         R.color.overlay_pink,
20         R.color.overlay_cyan,
21         R.color.overlay_purple,
22         R.color.overlay_gray,
23         R.color.overlay_teal,
24         R.color.overlay_yellow,
25     )
```

Εικόνα 5.52: BoxColors

Η μέθοδος invoke( ) εκτελεί την κύρια δουλειά. Δηλαδή λαμβάνει την λίστα αποτελεσμάτων ανίχνευσης και την βάση της εικόνας. Έπειτα δημιουργεί ένα αντίγραφο της βάσης εικόνας και ένα Canvas για να σχεδιάσει πάνω της. Μετά παίρνει το πλάτος και το ύψος. Για κάθε αποτέλεσμα επιλέγει ένα χρώμα απο τη λίστα boxColors βάσει της κλάσης cls και καλεί τη μέθοδο applyTransparentOverlay.

```
fun invoke(results: List<SegmentationResult>, baseBitmap: Bitmap): Bitmap {
    val output = baseBitmap.copy(Bitmap.Config.ARGB_8888, isMutable: true)
    val canvas = Canvas(output)

    // Δεν εφαρμόζουμε zoom ή νέα bitmap διάσταση
    val width = output.width
    val height = output.height

    results.forEachIndexed { index, result ->
        val colorResId = boxColors[result.box.cls % boxColors.size]
        applyTransparentOverlay(
            context,
            canvas,
            result,
            colorResId,
            width,
            height
        )
    }

    return output
}
```

Εικόνα 5.53: Μέθοδος invoke

Η επόμενη μέθοδος applyTransparentOverlay χειρλιζεται το οπτικό overlay για κάθε ανιχνευμένο αντικείμενο. Πιο αναλυτικά η μέθοδος παίρνει την περιοχή του αντικειμένου (box) από το SegmentationResult και επιλέγει το χρώμα και το κάνει διάφανο με τη βοήθεια της μεθόδου applyTransparentOverlayColor. Στην συνέχεια δημιουργεί μάσκα bitmap και μετά κλιμακώνει το mask bitmap στο μέγεθος του canvas και το σχεδιάζει πάνω στην εικόνα. Έπειτα υπολογίζει τις συντεταγμένες του τετραγλωνου πλανω στην εικόνα και το σχεδιάζει. Τέλος, δημιουργεί ένα paint για το κείμενο, γράφει την κλάση και το confidence σε ποσοστό πάνω απο το αντικείμενο.

```kotlin
    private fun applyTransparentOverlay( new *
        context: Context,
        canvas: Canvas,
        segmentationResult: SegmentationResult,
        overlayColorResId: Int,
        targetWidth: Int,
        targetHeight: Int
    ) {
        val box = segmentationResult.box

        // Χρώματα
        val overlayColor = applyTransparentOverlayColor(
            ContextCompat.getColor(context, overlayColorResId)
        )

        val mask = segmentationResult.mask
        val maskHeight = mask.size
        val maskWidth = mask[0].size

        // Δημιουργία bitmap μάσκας
        val maskBitmap = Bitmap.createBitmap(maskWidth, maskHeight, Bitmap.Config.ARGB_8888)
        for (y in 0 ≤ until < maskHeight) {
            for (x in 0 ≤ until < maskWidth) {
                if (mask[y][x] > 0) {
                    maskBitmap.setPixel(x, y, overlayColor)
                }
            }
        }

        // Scale το mask bitmap στο μέγεθος του output
        val scaledMask = Bitmap.createScaledBitmap(maskBitmap, targetWidth, targetHeight, filter: false)
        canvas.drawBitmap(scaledMask, left: 0f, top: 0f, paint: null)

        // Σχεδίαση bounding box
        val left = box.x1 * targetWidth
        val top = box.y1 * targetHeight
        val right = box.x2 * targetWidth
        val bottom = box.y2 * targetHeight
```

Εικόνα 5.54: Μέθοδος applyTransparentOverlay

```kotlin

        val boxPaint = Paint().apply {
            color = ContextCompat.getColor(context, overlayColorResId)
            strokeWidth = 3f
            style = Paint.Style.STROKE
        }
        canvas.drawRect(left, top, right, bottom, boxPaint)

        // Σχεδίαση ετικέτας
        val labelPaint = Paint().apply {
            color = Color.WHITE
            textSize = 30f
            style = Paint.Style.FILL
            isAntiAlias = true
            typeface = Typeface.DEFAULT_BOLD
        }

        val confidenceText = "${box.clsName} ${(box.cnf * 100).toInt()}%"
        val distanceText = String.format("%.2f m", segmentationResult.distance)

// Απόσταση πιο κάτω από το label
        canvas.drawText(confidenceText, left, y: top - 10, labelPaint)
        canvas.drawText(distanceText, left, y: top + 30, labelPaint)

    }
```

Εικόνα 5.55: Μέθοδος applyTransparentOverlay

Στο τελευταίο κομμάτι της DrawImages είναι η μέθοδος applyTransparentOverlayColor η οποία παίρνειένα χρώμα και δημειουργεί ένα νέο χρώμα με διαφάνεια. Κρατά τα αρχικά κόκκινα, πράσινα και μπλε συστατικά του χρώματος και τα συνδυάζει με τη νέα τιμή alpha μέσω της Color.argb() με σκοπό να δημιουργήσει ενα ημιδιαφανές χρλωμα που χρησιμοποιείται για να γεμίσει τη μάσκα πάνω στα αντικείμενα.

```
113
114     private fun applyTransparentOverlayColor(color: Int): Int {  new*
115         val alpha = 48
116         val red = Color.red(color)
117         val green = Color.green(color)
118         val blue = Color.blue(color)
119
120         return Color.argb(alpha, red, green, blue)
121     }
122 }
```

Εικόνα 5.56: Μέθοδος applyTransparentOverlayColor

5.7.8 Περιγραφή Κλάσης “ImageUtils”

Η κλάση ImageUtils είναι ένα utility object που έχει κάποιες μεθόδους για την επεξεργασία εικόνων και μασκών, μας βοηθάει για να σχεδιάσουμε τα boxes του YOLO. Δηλαδή λειτουργεί σαν “εργαλειοθήκη” για την επεξεργασία pixel-based δεδομένων.

Δήλωση Πακέτου(γραμμή 1): Δηλώνεται το πακέτο στο οποίο ανήκει η κλάση ImageUtils.

Imports(γραμμές 3-7): Σε αυτό το τμήμα, ο κώδικας εισάγει απαραίτητες βιβλιοθήκες που χρειάζεται η εφαρμογή για να λειτουργήσει.

```
1  package org.tensorflow.lite.examples.objectdetection
2
3  import android.graphics.Bitmap
4  import android.graphics.Canvas
5  import android.graphics.Color
6  import android.graphics.Paint
7  import android.graphics.Path
8
```

Εικόνα 5.57: Imports

Στο παρακάτω κομμάτι κώδικα η ImageUtils παρέχει δύο βασικές λειτουργίες. Αρχικά με το clone δημιουργεί ένα πλήρες αντίγραφο μιας λίστας που παρέχει δισδιάστατους πίνακες. Έτσι αποφεύγεται η αλλαγή των αρχικλων δεδομένων όταν γίνονται αλλαγές. Στην συνέχεια με την scaleMask, για να

ταιριάζει σε συγκεκριμένο πλάτος και ύψος αλλάζει το μέγεθος της μάσκας. Αυτό γίνεται με υπολογισμό αναλογιών (xRatio και yRatio) και αντιστοίχισει τιμών απο το αρχικό στο νέο μέγεθος.

```
object ImageUtils {
    fun List<Array<FloatArray>>.clone(): List<Array<FloatArray>> {
        return this.map { array -> array.map { it.clone() }.toTypedArray() }
    }

    fun Array<FloatArray>.scaleMask(targetWidth: Int, targetHeight: Int): Array<FloatArray> {
        val originalHeight = this.size
        val originalWidth = this[0].size

        val xRatio = (originalWidth shl 16) / targetWidth
        val yRatio = (originalHeight shl 16) / targetHeight

        val output = Array(targetHeight) { FloatArray(targetWidth) }

        for (y in 0 until targetHeight) {
            val origY = (y * yRatio) ushr 16
            for (x in 0 until targetWidth) {
                val origX = (x * xRatio) ushr 16
                output[y][x] = this[origY][origX]
            }
        }

        return output
    }

```

Εικόνα 5.58: Μέθοδος clone & scaleMask

Στο επόμενο τμήμα και συγκεκριμένα στις γραμμές (37-51) έχουμε την toMask και την smooth. Η toMask μετατρέπει έναν πίνακα Array<FloatArray> σε δυαδική μάσκα Array<IntArray> όπου κάθε τιμή γίνεται 1 αν είναι μεγαλύτερη από το μηδέν και 0 αν δεν είναι. Η smooth παίρνει μια δυαδική μάσκα και εφαρμόζει εξομάλυνση Gaussian, δηλαδη αρχικά μετατρέπει τη μάσκα σε float για πιο ακριβείς υπολογισμούς. Στη συνέχεια δημιουργεί έναν Gaussian kernel με βάση το μέγεθος (kernel). Μετά εφαρμόζει Gaussian blur στην εικόνα και τέλος επαναφέρει το αποτέλεσμα σε δυαδική μάσκα μέσω κατωφλίου.

```
    fun Array<FloatArray>.toMask(): Array<IntArray> =
        map { row -> row.map { if (it > 0) 1 else 0 }.toIntArray() }.toTypedArray()


    fun Array<IntArray>.smooth(kernel: Int) : Array<IntArray> {
        // Using Array because it is faster then List
        val maskFloat = Array(this.size) { i ->
            FloatArray(this[i].size) { j ->
                if (this[i][j] > 0) 1F else 0F
            }
        }
        val gaussianKernel = createGaussianKernel(kernel)
        val blurredImage = applyGaussianBlur(maskFloat, gaussianKernel)
        return thresholdImage(blurredImage)
    }
```

Εικόνα 5.59: Μέθοδος toMask & smooth

Η μέθοδος createGaussianKernel δημιουργεί έναν δισδιάστατο Gaussian πυρήνα (Gaussian blur kernel) συγκεκριμένου μεγέθους.

- Ορίζει μια σταθερή τιμή sigma (= 2F), που καθορίζει πόσο έντονη θα είναι η εξομάλυνση.
- Δημιουργεί έναν πίνακα kernel διαστάσεων size × size.
- Υπολογίζει το "κέντρο" του πίνακα (mean) για να ξέρει σε ποια απόσταση βρίσκεται κάθε κελί.
- Γεμίζει το kernel με τιμές που ακολουθούν την **Gaussian κατανομή**:
  - Κοντά στο κέντρο → υψηλότερες τιμές.
  - Μακριά από το κέντρο → χαμηλότερες τιμές.
- Υπολογίζει το άθροισμα όλων των τιμών και στη συνέχεια τις **κανονικοποιεί** ώστε το άθροισμα του kernel να είναι 1 (απαραίτητο για να μην αλλάξει η συνολική φωτεινότητα της εικόνας).

```
53      private fun createGaussianKernel(size: Int): Array<FloatArray> {  new *
54          val sigma = 2F
55          val kernel = Array(size) { FloatArray(size) }
56          val mean = size / 2
57          var sum = 0F
58
59          for (x in 0 ≤ until < size) {
60              for (y in 0 ≤ until < size) {
61                  kernel[x][y] = (1F / (2F * Math.PI.toFloat() * sigma * sigma)) * exp(
62                      x: -((x - mean) * (x - mean) + (y - mean) * (y - mean)) / (2F * sigma * sigma)
63                  )
64                  sum += kernel[x][y]
65              }
66          }
67
68          for (x in 0 ≤ until < size) {
69              for (y in 0 ≤ until < size) {
70                  kernel[x][y] /= sum
71              }
72          }
73
74          return kernel
75      }
```

Εικόνα 5.60: Μέθοδος createGaussianKernel

Η μέθοδος **applyGaussianBlur** εφαρμόζει το Gaussian blur φίλτρο σε μια εικόνα (ή μάσκα) χρησιμοποιώντας τον Gaussian πυρήνα που φτιάχτηκε πριν. Αρχικά παίρνει το ύψος το πλάτος της εικόνας και το μέγεθος του Gaussian kernel. Μετά Υπολογίζει το offset. Στην συνέχεια δημιουργεί τον πίνακα blurredImage για να αποθηκεύσει τα αποτελέσματα. Για κάθε pixel αν είναι πολύ κοντά στην άκρη (δεν χωράει ολόκληρο το kernel), το αφήνει ως έχει, διαφορετικά κάνει συνέλιξη δηλαδή Πολλαπλασιάζει κάθε τιμή γύρω από το pixel με την αντίστοιχη τιμή του kernel και Προσθέτει όλα αυτά τα γινόμενα για να βγάλει τη νέα τιμή του pixel. Τέλος επιστρέφει την εικόνα.

```kotlin
private fun applyGaussianBlur(image: Array<FloatArray>, kernel: Array<FloatArray>): Array<FloatArray> {  new *
    val height = image.size
    val width = image[0].size
    val kernelSize = kernel.size
    val offset = kernelSize / 2
    val blurredImage = Array(height) { FloatArray(width) }

    for (i in image.indices) {
        for (j in image[i].indices) {
            if (i < offset || j < offset || i >= height - offset || j >= width - offset) {
                blurredImage[i][j] = image[i][j]
                continue
            }

            var sum = 0F
            for (ki in kernel.indices) {
                for (kj in kernel[ki].indices) {
                    sum += image[i - offset + ki][j - offset + kj] * kernel[ki][kj]
                }
            }
            blurredImage[i][j] = sum
        }
    }

    return blurredImage
}
```

Εικόνα 5.61: Μέθοδος applyGaussianBlur

Η επόμενη μέθοδος ειναι η thresholdImage η οποία κάνει δυαδικοποίση μιας εικόνας βάσει ενός σταθερού ορίου. Πώς λειτουργει:

1. Παίρνει μια εικόνα σε μορφή `Array<FloatArray>` (τιμές συνήθως 0.0–1.0).
2. Υπολογίζει το ύψος (`height`) και το πλάτος (`width`).
3. Δημιουργεί έναν νέο πίνακα `Array<IntArray>` ίδιων διαστάσεων.
4. Για κάθε pixel:
    a) Αν η τιμή του είναι **μεγαλύτερη από 0.9**, γίνεται `1` (ενεργό pixel).
    b) Διαφορετικά, γίνεται `0` (ανενεργό pixel).

Αυτό είναι το τελευταίο βήμα μετά το Gaussian blur, ώστε η μάσκα να γίνει καθαρή και ευκρινής για επεξεργασία ή σχεδίαση.

```kotlin
private fun thresholdImage(image: Array<FloatArray>): Array<IntArray> {  new *
    val height = image.size
    val width = image[0].size
    return Array(height) { i ->
        IntArray(width) { j ->
            if (image[i][j] > 0.9F) 1 else 0
        }
    }
}
```

Εικόνα 5.62: Μέθοδος thresholdImage

Η αμέσως επόμενη εικόνα παρουσιαζει την **scaleToFit (γραμμες 114-132)** η οποία προσαρμόζει ενα Bitmap ώστε να χωράει μέσα σε έναν συγκεκριμένο "στόχο" (targetWidth, targetHeight) χωρίς να παραμορφώνει την εικόνα. Η μέθοδος κάνει resize της εικόνας ώστε να χωράει μέσα στο target box, αλλά κρατάει σωστές αναλογίες. Στις **(γραμμες 134-149)** είναι η μέθοδος **getLowesrActivePixel** ψάχνει μέσα σε μια μάσκα και βρίσκει το χαμηλότερο pixel (δηλαδή πιο κοντά στο κάτω μέρος της εικόνας) που ξεπερνάει ένα συγκεκριμένο όριο. Για παράδειγμα αν η μάσκα αντιστοιχεί σε έναν άνθρωπο όρθιο μέσα στην εικόνα, η μέθοδος θα επιστρέψει το **pixel πιο κάτω (τα πόδια)** που υπάρχει ενεργό. Έτσι μπορούμε να ξέρουμε πού τελειώνει το αντικείμενο μέσα στο frame.

```
113
114     fun Bitmap.scaleToFit(targetWidth: Int, targetHeight: Int): Bitmap {  new *
115         val aspectRatio = width.toFloat() / height.toFloat()
116         val targetRatio = targetWidth.toFloat() / targetHeight.toFloat()
117
118         val scaledWidth: Int
119         val scaledHeight: Int
120
121         if (aspectRatio > targetRatio) {
122
123             scaledWidth = targetWidth
124             scaledHeight = (targetWidth / aspectRatio).toInt()
125         } else {
126
127             scaledHeight = targetHeight
128             scaledWidth = (targetHeight * aspectRatio).toInt()
129         }
130
131         return Bitmap.createScaledBitmap( src: this, scaledWidth, scaledHeight,  filter: true)
132     }
133
134     fun getLowestActivePixel(mask: Array<FloatArray>, threshold: Float = 0.5f): Int {  new *
135         val height = mask.size
136         val width = mask[0].size
137
138         for (y in height - 1 ≥  downTo  ≥ 0) {
139             for (x in 0 ≤  until  < width) {
140                 if (mask[y][x] >= threshold) {
141                     return y
142                 }
143             }
144         }
145
146         return -1 // Αν δεν βρεθεί τίποτα
147     }
148
149 }
```

Εικόνα 5.63: Μέθοδος scaleToFit & getLowestActivePixel

5.7.9 Περιγραφή Κλάσης “InstanceSegmentation”

Η κλάση **InstanceSegmentation** αναλαμβάνει να φορτώσει το YOLO μοντέλο και τις ετικέτες των αντικειμένων, να προεπεξεργαστεί τα καρέ εικόνας, να τα περάσει στον interpreter και να παραλάβει τα αποτελέσματα. Μετά τρέχει **post-processing** για να βρεί τα καλύτερα bounding boxes, να εφαρμόσει NMS (Non-Max Suppression), να δημιουργήσει τις αντίστοιχες μάσκες για κάθε αντικείμενο και να τις κλιμακώσει στις πραγματικές διαστάσεις της εικόνας. Μέσα από το InstanceSegmentationListener επιστρέφει στον χρήστη τα τελικά αποτελέσματα (bounding boxes, μάσκες και χρόνους εκτέλεσης) ή μηνύματα σφάλματος.

Δήλωση Πακέτου(γραμμή 1): Δηλώνεται το πακέτο στο οποίο ανήκει η κλάση InstanceSegmentation.

Imports(γραμμές 4-25): Σε αυτό το τμήμα, ο κώδικας εισάγει απαραίτητες βιβλιοθήκες που χρειάζεται η εφαρμογή για να λειτουργήσει.

```
1  package org.tensorflow.lite.examples.objectdetection                    ⚠8 ⚠2
2
3
4  import android.content.Context
5  import android.graphics.Bitmap
6  import android.hardware.Sensor
7  import android.hardware.SensorEvent
8  import android.hardware.SensorManager
9  import android.os.SystemClock
10 import android.util.Log
11 import org.tensorflow.lite.examples.objectdetection.ImageUtils.clone
12 import org.tensorflow.lite.examples.objectdetection.ImageUtils.scaleMask
13 import org.tensorflow.lite.examples.objectdetection.ImageUtils.toMask
14
15 import org.tensorflow.lite.DataType
16 import org.tensorflow.lite.Interpreter
17 import org.tensorflow.lite.examples.objectdetection.MetaData.extractNamesFromLabelFile
18 import org.tensorflow.lite.examples.objectdetection.MetaData.extractNamesFromMetadata
19 import org.tensorflow.lite.support.common.FileUtil
20 import org.tensorflow.lite.support.common.ops.CastOp
21 import org.tensorflow.lite.support.common.ops.NormalizeOp
22 import org.tensorflow.lite.support.image.ImageProcessor
23 import org.tensorflow.lite.support.image.TensorImage
24 import org.tensorflow.lite.support.tensorbuffer.TensorBuffer
25 import java.nio.ByteBuffer
```

Εικόνα 5.64: Imports

Η κλάση InstanceSegmentation δημιουργείται με **constructor parameters**:

- **context**: το Android context για πρόσβαση σε πόρους (π.χ. model file).
- **modelPath**: η διαδρομή του TensorFlow Lite μοντέλου.
- **labelPath**: προαιρετική διαδρομή για labels (ονόματα κλάσεων).
- **instanceSegmentationListener**: callback interface που επιστρέφει τα αποτελέσματα ή σφάλματα.
- **message**: συνάρτηση για logging/προβολή μηνυμάτων.

Μέσα στην κλάση:

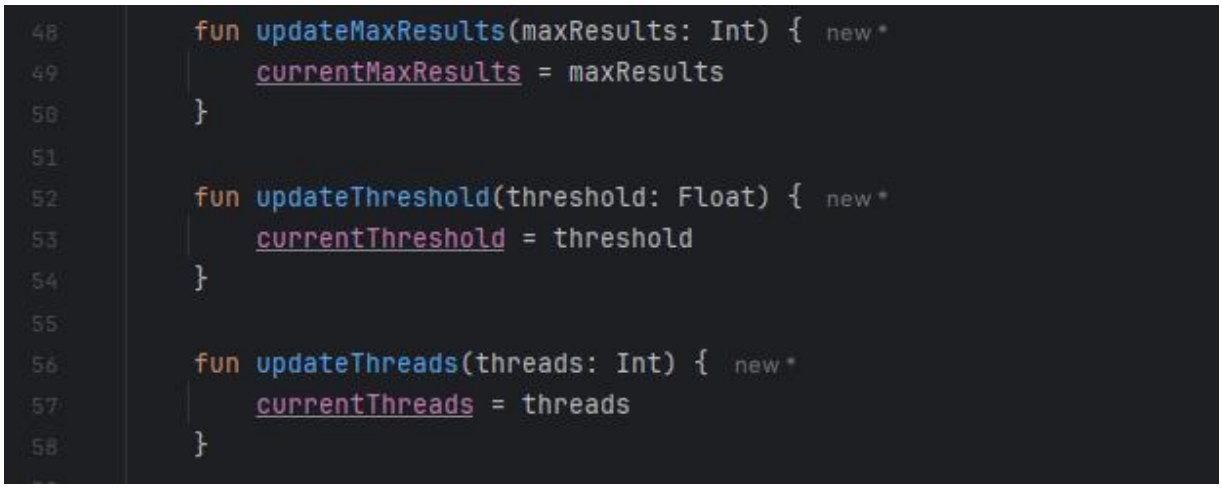
- **interpreter**: το **TensorFlow Lite Interpreter**, δηλαδή ο κινητήρας που τρέχει το μοντέλο.
- **labels**: λίστα με τις ετικέτες/ονόματα κλάσεων του μοντέλου (π.χ. "person", "car").
- Οι μεταβλητές **tensorWidth**, **tensorHeight, numChannel, numElements, xPoints, yPoints, masksNum:** κρατούν πληροφορίες για το σχήμα των εισόδων και εξόδων του μοντέλου (διαστάσεις εικόνας, αριθμό καναλιών, μάσκες).
- **currentMaxResults**: μέγιστος αριθμός αποτελεσμάτων (bounding boxes/μάσκες) που θα επιστραφούν· ξεκινά με default τιμή 3.
- **currentThreshold**: Ορίζει πόσο "σίγουρο" πρέπει να είναι το YOLO μοντέλο για να δεχτεί μια ανίχνευση ως έγκυρη.
- **currentThreads**: αριθμός threads που θα χρησιμοποιεί το TFLite Interpreter

```
27  class InstanceSegmentation(  new*
28      context: Context,
29      modelPath: String,
30      labelPath: String?,
31      private val instanceSegmentationListener: InstanceSegmentationListener,
32      private val message: (String) -> Unit
33  ) {
34      private var interpreter: Interpreter
35      private var labels = mutableListOf<String>()
36
37      private var tensorWidth = 0
38      private var tensorHeight = 0
39      private var numChannel = 0
40      private var numElements = 0
41      private var xPoints = 0
42      private var yPoints = 0
43      private var masksNum = 0
44      private var currentMaxResults: Int = 3  // αρχική τιμή
45      private var currentThreshold: Float = CONFIDENCE_THRESHOLD
46      private var currentThreads: Int = 2
```

Εικόνα 5.65: Δηλώσεις μεταβλητών

Οι επόμενες συνερτήσεις ειναι πο setters που μας βοηθανε να αλλαξουμε δυναμικά ορισμένες παραμέτρους.

1. **updateMaxResults(maxResults: Int):** Αλλάζει το μέγιστο αριθμό αποτελεσμάτων (ανιχνεύσεων) που θα εμφανίζει στην εφαρμογή.
2. **updateThreshold(threshold: Float):** Αλλάζει το κατώφλι εμπιστοσύνης. Δηλαδή ποια αντικείμενα θα εμφανίζονται ανάλογα το threshold
3. **updateThreads(threads: Int):** Αλλάζει πόσα threads χρησιμοποιεί το TensorFlow Lite interpreter



Εικόνα 5.66: Μέθοδος updateMaxResults, updateThreshold & updateThreads

Στο επόμενο κομμάτι στις **γραμμές 60-63** είναι η imageProcessor η οποία ορίζει μια αλυσίδα απο βήματα προεπεξεργασίας που θα εφαρμόζονται σε κάθε εικόνα πριν μπει στο μοντέλο.

- **NormalizeOp(INPUT_MEAN, INPUT_STANDARD_DEVIATION):** Κανονικοποιεί τα pixel (0–255) σε μια πιο κατάλληλη μορφή για το μοντέλο
- **CastOp(INPUT_IMAGE_TYPE):** Μετατρέπει την εικόνα σε συγκεκριμένο τύπο δεδομένων, που είναι ο τύπος που περιμένει το TFLite μοντέλο

Στις **γραμμες 65-124** έχουμε το **init block.** Εδώ γίνεται όλη η αρχικοποίηση του TensorFlow Lite interpreter και η ανάγνωση πληροφοριών για το input/output του μοντέλου. Αρχικά κάνει δημιουργία ρυθμίσεων **(γραμμές 66,67).** Στην συνέχεια φορτώνει το μοντέλο **(γραμμές 70,71).** Μετά γίνεται φόρτωση labels **(γραμμες 73-81)**, αν δεν έχει δοθεί αρχείο labelPath, εμφανίζει μήνυμα και φορτώνει προσωρινά classes απο MetaData.TEMP_CLASSES. Αν υπάρχει labelPath φορτώνει labels από το αρχείο. Στο υπόλοιπο κομμάτι ελέγχει το σχήμα του input tensor και διαβάζει Output shapes.

```
60      private val imageProcessor = ImageProcessor.Builder()
61          .add(NormalizeOp(INPUT_MEAN, INPUT_STANDARD_DEVIATION))
62          .add(CastOp(INPUT_IMAGE_TYPE))
63          .build()
64
65      init {  new *
66          val options = Interpreter.Options()
67          options.setNumThreads(currentThreads)
68
69
70          val model = FileUtil.loadMappedFile(context, modelPath)
71          interpreter = Interpreter(model, options)
72
73          labels.addAll(extractNamesFromMetadata(model))
74          if (labels.isEmpty()) {
75              if (labelPath == null) {
76                  message("Model not contains metadata, provide LABELS_PATH in Constants.kt")
77                  labels.addAll(MetaData.TEMP_CLASSES)
78              } else {
79                  labels.addAll(extractNamesFromLabelFile(context, labelPath))
80              }
81          }
82
83          val inputShape = interpreter.getInputTensor( inputIndex: 0)?.shape()
84          val outputShape0 = interpreter.getOutputTensor( outputIndex: 0)?.shape()
85          val outputShape1 = interpreter.getOutputTensor( outputIndex: 1)?.shape()
86
```

Εικόνα 5.67: initBlock

```
86
87          if (inputShape != null) {
88              tensorWidth = inputShape[1]
89              tensorHeight = inputShape[2]
90
91              // If in case input shape is in format of [1, 3, ..., ...]
92              if (inputShape[1] == 3) {
93                  tensorWidth = inputShape[2]
94                  tensorHeight = inputShape[3]
95              }
96          }
97
98          if (outputShape0 != null) {
99              numChannel = outputShape0[1]
100             numElements = outputShape0[2]
101         }
102
103         if (outputShape1 != null) {
104             if (outputShape1[1] == 32) {
105                 masksNum = outputShape1[1]
106                 xPoints = outputShape1[2]
107                 yPoints = outputShape1[3]
108             } else {
109                 xPoints = outputShape1[1]
110                 yPoints = outputShape1[2]
111                 masksNum = outputShape1[3]
112             }
113         }
114     }
115
```

Εικόνα 5.68: initBlock

Στις γραμμες 116-118 υπάρχει η close() η οποία κλείνει τον **TensorFlow Lite Interpreter** και απελευθερώνει πόρους. Πρέπει πάντα να καλείται όταν τελειώνει η χρήση του μοντέλου

```
115
116     fun close() {  new *
117         interpreter.close()
118     }
119
```

Εικόνα 5.69: Μέθοδος close

Στην invoke(frame: Bitmap) **(γραμμες 120-183)** γίνονται τα παρακάτω:

1. Στις **(γραμμές 121-126)** γίνεται έλεγχος αν το interpreter είναι σωστά αρχικοποιημένο. Αν δεν έχουν σωστά φορτωθεί τα σχήματα (input/output tensors), δεν μπορεί να τρέξει
2. Στις **(γραμμες 128,130)** έχουμε την προεπεξεργασία εικόνας. Μετατρέπει το Bitmap σε buffer που το μοντέλο μπορεί να καταλάβει (σωστό μέγεθος, normalization, κλπ). Μετράει και τον χρόνο pre-processing.
3. Στις **(γραμμές 132,145) δημιουργούνται buffers για τα outputs.**
   a) Το μοντέλο επιστρέφει δύο outputs:
      i. **coordinatesBuffer**: τα bounding boxes + confidence.
      ii. **maskProtoBuffer**: τα πρωτότυπα segmentation masks.
   b) Αυτά μπαίνουν σε outputBuffer ώστε το interpreter να ξέρει που να γράψει.
4. Στην **γραμμή 151** τρέχει το μοντέλο.
5. Στις **(γραμμές 157-171) γίνεται μετα-επεξεργασία**
   a) Φιλτράρει τα αποτελέσματα από το coordinatesBuffer για να βρει τα καλύτερα boxes (ανάλογα με threshold & max results).
   b) Αν δεν βρεθεί τίποτα επιστρέφει onEmpty().
   c) **Γραμμή 162** Αναδιατάσσει το raw output των μασκών σε σωστή μορφή (π.χ. [mask, width, height]).
   d) **Γραμμές 164-171** Για κάθε επιλεγμένο box, φτιάχνει ένα SegmentationResult που περιέχει:
      i. **Box**: το bounding box με label + confidence.
      ii. **Mask**: την τελική μάσκα (προσαρμοσμένη στο μέγεθος του frame).
      iii. **Distance**: εδώ είναι placeholder (0.0), αλλά μπορεί να γεμίσει με απόσταση αντικειμένου αν έχει υπολογισμό
6. Τελος, γίνεται επιστροφή αποτελεσμάτων **στις γραμμές 175-182**.

```
120     fun invoke(frame: Bitmap) {  new *
121         if (tensorWidth == 0 || tensorHeight == 0
122             || numChannel == 0 || numElements == 0
123             || xPoints == 0 || yPoints == 0 || masksNum == 0) {
124             instanceSegmentationListener.onError( error: "Interpreter not initialized properly")
125             return
126         }
127
128         var preProcessTime = SystemClock.uptimeMillis()
129
130         val imageBuffer = preProcess(frame)
131
132         val coordinatesBuffer = TensorBuffer.createFixedSize(
133             intArrayOf(1 , numChannel, numElements),
134             OUTPUT_IMAGE_TYPE
135         )
136
137         val maskProtoBuffer = TensorBuffer.createFixedSize(
138             intArrayOf(1, xPoints, yPoints, masksNum),
139             OUTPUT_IMAGE_TYPE
140         )
141
142         val outputBuffer = mapOf<Int, Any>(
143             0 to coordinatesBuffer.buffer.rewind(),
144             1 to maskProtoBuffer.buffer.rewind()
145         )
146
147         preProcessTime = SystemClock.uptimeMillis() - preProcessTime
148
149         var interfaceTime = SystemClock.uptimeMillis()
```

Εικόνα 5.70: Μέθοδος invoke

```
150
151         interpreter.runForMultipleInputsOutputs(imageBuffer, outputBuffer)
152
153         interfaceTime = SystemClock.uptimeMillis() - interfaceTime
154
155         var postProcessTime = SystemClock.uptimeMillis()
156
157         val bestBoxes = bestBox(coordinatesBuffer.floatArray) ?: run {
158             instanceSegmentationListener.onEmpty()
159             return
160         }
161
162         val maskProto = reshapeMaskOutput(maskProtoBuffer.floatArray)
163
164         val segmentationResults = bestBoxes.map {
165             SegmentationResult(
166                 box = it,
167                 mask = getFinalMask(frame.width, frame.height, it, maskProto),
168                 distance = 0.0
169
170             )
171         }
172
173         postProcessTime = SystemClock.uptimeMillis() - postProcessTime
174
175         instanceSegmentationListener.onDetect(
176             preProcessTime = preProcessTime,
177             interfaceTime = interfaceTime,
178             postProcessTime = postProcessTime,
179             results = segmentationResults,
180             bitmap_width = frame.width,
181             bitmap_height = frame.height
182         )
183     }
```

Εικόνα 5.71: Μέθοδος invoke

Η getFinalMask παίρνει τις πρώτες εξόδους μάσκας του μοντέλου και το αντίστοιχο bounding box ενός αντικειμένου, εφαρμόζει τα βάρη κάθε prototype mask, συνδυάζει τα αποτελέσματα μόνο μέσα στα όρια του box και τελικά επανακλιμακώνει τη μάσκα στις επιθυμητές διαστάσεις, επιστρέφοντας έτσι την τελική 2D μάσκα του αντικειμένου.

```
185     private fun getFinalMask(  new *
186         width: Int,
187         height: Int,
188         output0: Output0,
189         output1: List<Array<FloatArray>>
190     ): Array<FloatArray> {
191         val output1Copy = output1.clone()
192         val relX1 = output0.x1 * xPoints
193         val relY1 = output0.y1 * yPoints
194         val relX2 = output0.x2 * xPoints
195         val relY2 = output0.y2 * yPoints
196
197         val zero: Array<FloatArray> = Array(yPoints) { FloatArray(xPoints) { 0F } }
198         for ((index, proto) in output1Copy.withIndex()) {
199             for (y in 0 ≤ until  < yPoints) {
200                 for (x in 0 ≤ until  < xPoints) {
201                     proto[y][x] *= output0.maskWeight[index]
202                     if (x + 1  > relX1 && x + 1 < relX2 && y + 1 > relY1 && y + 1 < relY2) {
203                         zero[y][x] += proto[y][x]
204                     }
205                 }
206             }
207         }
208
209         return zero.scaleMask( targetWidth: 450 ,  targetHeight: 600)
210     }
211
```

Εικόνα 5.72: Μέθοδος getFinalMask

Η reshapeMaskOutput μετατρέπει το μονοδιάστατο πίνακα floats που επιστρέφει το μοντέλο σε μια πιο εύχρηστη δομή τριών διαστάσεων [mask][x][y], διαχωρίζοντας κάθε μάσκα από τις συνολικά masksNum και οργανώνοντας τα δεδομένα ανά σημεία x και y.

```
213     private fun reshapeMaskOutput(floatArray: FloatArray): List<Array<FloatArray>> {  new *
214         return List(masksNum) { mask ->
215             Array(xPoints) { r ->
216                 FloatArray(yPoints) { c ->
217                     floatArray[masksNum * yPoints * r + masksNum * c + mask]
218                 }
219             }
220         }
221     }
222
```

Εικόνα 5.73: Μέθοδος reshapeMaskOutput

Η bestBox παίρνει τον πίνακα εξόδου του μοντέλου που περιέχει πληροφορίες για όλα τα πιθανά κουτιά (bounding boxes) και τις πιθανότητες κατηγοριών, εντοπίζει τα κουτιά που έχουν εμπιστοσύνη

(confidence) πάνω από το τρέχον όριο (currentThreshold), υπολογίζει τις συντεταγμένες τους (x1, y1, x2, y2) και αποθηκεύει τα βάρη μάσκας για κάθε κουτί. Στο τέλος εφαρμόζει μη μέγιστη καταστολή (NMS) για να αφαιρέσει επικαλυπτόμενα κουτιά και επιστρέφει τα καλύτερα αποτελέσματα μέχρι τον μέγιστο αριθμό που έχει οριστεί (currentMaxResults).

```
223     private fun bestBox(array: FloatArray) : List<Output0>? {  new *
224         val output0List = mutableListOf<Output0>()
225         for (c in 0 ≤  until  < numElements) {
226             var maxConf = currentThreshold
227             var maxIdx = -1
228             var currentInd = 4
229             var arrayIdx = c + numElements * currentInd
230
231             while (currentInd < (numChannel - masksNum)){
232                 if (array[arrayIdx] > maxConf) {
233                     maxConf = array[arrayIdx]
234                     maxIdx = currentInd - 4
235                 }
236                 currentInd++
237                 arrayIdx += numElements
238             }
```

Εικόνα 5.74: Μέθοδος bestBox

```
            if (maxConf > currentThreshold) {
                val clsName = labels[maxIdx]
                val cx = array[c] // 0
                val cy = array[c + numElements] // 1
                val w = array[c + numElements * 2]
                val h = array[c + numElements * 3]
                val x1 = cx - (w/2F)
                val y1 = cy - (h/2F)
                val x2 = cx + (w/2F)
                val y2 = cy + (h/2F)
                if (x1 < 0F || x1 > 1F) continue
                if (y1 < 0F || y1 > 1F) continue
                if (x2 < 0F || x2 > 1F) continue
                if (y2 < 0F || y2 > 1F) continue
                val maskWeight = mutableListOf<Float>()
                while (currentInd < numChannel){
                    maskWeight.add(array[arrayIdx])
                    currentInd++
                    arrayIdx += numElements
                }
                output0List.add(
                    Output0(
                        x1 = x1, y1 = y1, x2 = x2, y2 = y2,
                        cx = cx, cy = cy, w = w, h = h,
                        cnf = maxConf, cls = maxIdx, clsName = clsName,
                        maskWeight = maskWeight
                    )
                )
            }
        }
        if (output0List.isEmpty()) return null
        return applyNMS(output0List).take(currentMaxResults).toMutableList()
    }
```

Εικόνα 5.75: Μέθοδος bestBox

Η **applyNMS** εφαρμόζει τη μη μέγιστη καταστολή (Non-Maximum Suppression, NMS) σε μια λίστα από κουτιά (Output0). Αρχικά ταξινομεί τα κουτιά κατά φθίνουσα εμπιστοσύνη (cnf). Στη συνέχεια, επιλέγει το πιο "σίγουρο" κουτί και αφαιρεί από τα υπόλοιπα όλα τα κουτιά που επικαλύπτονται πολύ με αυτό (όπου ο δείκτης IoU ≥ IOU_THRESHOLD). Επαναλαμβάνει τη διαδικασία μέχρι να μην μείνουν άλλα κουτιά, επιστρέφοντας τη λίστα με τα τελικά, επιλεγμένα κουτιά.

```
273     private fun applyNMS(output0List: List<Output0>) : MutableList<Output0> {  new *
274         val sortedBoxes = output0List.sortedByDescending { it.cnf }.toMutableList()
275         val selectedBoxes = mutableListOf<Output0>()
276
277         while(sortedBoxes.isNotEmpty()) {
278             val first = sortedBoxes.first()
279             selectedBoxes.add(first)
280             sortedBoxes.remove(first)
281
282             val iterator = sortedBoxes.iterator()
283             while (iterator.hasNext()) {
284                 val nextBox = iterator.next()
285                 val iou = calculateIoU(first, nextBox)
286                 if (iou >= IOU_THRESHOLD) {
287                     iterator.remove()
288                 }
289             }
290         }
291
292         return selectedBoxes
293     }
294
```

Εικόνα 5.76: Μέθοδος applyNMS

Η **calculateIoU** υπολογίζει τον δείκτη επικαλυπτόμενης περιοχής (Intersection over Union, IoU) μεταξύ δύο κουτιών (Output0). Πρώτα βρίσκει την περιοχή που επικαλύπτονται τα δύο κουτιά, έπειτα υπολογίζει τις συνολικές περιοχές τους και τελικά επιστρέφει τον λόγο της κοινής περιοχής προς την ένωση των περιοχών τους. Αυτό χρησιμοποιείται για να μετρήσει πόσο πολύ επικαλύπτονται δύο κουτιά.

```
295     private fun calculateIoU(box1: Output0, box2: Output0): Float {  new *
296         val x1 = maxOf(box1.x1, box2.x1)
297         val y1 = maxOf(box1.y1, box2.y1)
298         val x2 = minOf(box1.x2, box2.x2)
299         val y2 = minOf(box1.y2, box2.y2)
300         val intersectionArea = maxOf( a: 0F,  b: x2 - x1) * maxOf( a: 0F,  b: y2 - y1)
301         val box1Area = box1.w * box1.h
302         val box2Area = box2.w * box2.h
303         return intersectionArea / (box1Area + box2Area - intersectionArea)
304     }
305
```

Εικόνα 5.77: Μέθοδος calculateIoU

Η **preProcess** προετοιμάζει ένα Bitmap για είσοδο στο μοντέλο: αλλάζει το μέγεθος της εικόνας ώστε να ταιριάζει με τις απαιτούμενες διαστάσεις του μοντέλου, τη φορτώνει σε ένα TensorImage, εφαρμόζει τις απαραίτητες μετατροπές και κανονικοποιήσεις μέσω του imageProcessor και τελικά επιστρέφει τα δεδομένα της εικόνας σε μορφή ByteBuffer, έτοιμα για επεξεργασία από τον TFLite interpreter.

```
306     private fun preProcess(frame: Bitmap): Array<ByteBuffer> {  new *
307         val resizedBitmap = Bitmap.createScaledBitmap(frame, tensorWidth, tensorHeight,  filter: false)
308         val tensorImage = TensorImage(INPUT_IMAGE_TYPE)
309         tensorImage.load(resizedBitmap)
310         val processedImage = imageProcessor.process(tensorImage)
311         return arrayOf(processedImage.buffer)
312     }
```

Εικόνα 5.78: Μέθοδος preProcess

Το **InstanceSegmentationListener** είναι μια διεπαφή που ορίζει τρεις μεθόδους για να χειρίζεται τα αποτελέσματα ή σφάλματα της ανίχνευσης αντικειμένων: η onError καλείται αν εμφανιζτεί κάποιο σφάλμα, η onEmpty όταν δεν ανιχνεύεται κανένα αντικείμενο και η onDetect παρέχει τα αποτελέσματα της ανίχνευσης μαζί με χρόνους προ-επεξεργασίας, μετα-επεξεργασίας και διαστάσεις της εικόνας. Το companion object ορίζει σταθερές που χρησιμοποιούνται σε όλη την κλάση, όπως οι παράμετροι κανονικοποίησης της εικόνας, οι τύποι δεδομένων εισόδου/εξόδου του μοντέλου, το αρχικό όριο εμπιστοσύνης για ανίχνευση και το όριο για την συνάρτηση μη-μέγιστης καταστολής (NMS).

```
314 interface InstanceSegmentationListener {  new *
315     fun onError(error: String)  new *
316     fun onEmpty()  new *
317     fun onDetect(  new *
318         interfaceTime: Long,
319         results: List<SegmentationResult>,
320         preProcessTime: Long,
321         postProcessTime: Long,
322         bitmap_width: Int,
323         bitmap_height: Int
324     )
325 }
326 companion object {  new *
327     private const val INPUT_MEAN = 0f
328     private const val INPUT_STANDARD_DEVIATION = 255f
329     private val INPUT_IMAGE_TYPE = DataType.FLOAT32
330     private val OUTPUT_IMAGE_TYPE = DataType.FLOAT32
331     private const val CONFIDENCE_THRESHOLD = 0.3F
332     private const val IOU_THRESHOLD = 0.5F
333 }
334 }
```

Εικόνα 5.79: Interface InstanceSegmentationListener

5.7.10 Περιγραφή Κλάσης “Metadata”

Η κλάση **MetaData** παρέχει βοηθητικές μεθόδους για την εξαγωγή των ονομάτων των κατηγοριών (labels) από το μοντέλο YOLO ή από ένα αρχείο ετικετών. Η extractNamesFromMetadata διαβάζει τα

metadata του μοντέλου και προσπαθεί να εξάγει τα ονόματα των κατηγοριών από ένα συσχετισμένο αρχείο temp_meta.txt. Η extractNamesFromLabelFile διαβάζει τα ονόματα από ένα απλό αρχείο κειμένου που βρίσκεται στα assets. Επιπλέον, η TEMP_CLASSES παρέχει μια προκαθορισμένη λίστα 1000 εικονικών κλάσεων ως fallback σε περίπτωση που δεν υπάρχουν πραγματικά labels.

Δήλωση Πακέτου(γραμμή 1): Δηλώνεται το πακέτο στο οποίο ανήκει η κλάση InstanceSegmentation.

Imports(γραμμές 3-9): Σε αυτό το τμήμα, ο κώδικας εισάγει απαραίτητες βιβλιοθήκες που χρειάζεται η εφαρμογή για να λειτουργήσει.

```
1  package org.tensorflow.lite.examples.objectdetection
2
3  import android.content.Context
4  import org.tensorflow.lite.support.metadata.MetadataExtractor
5  import java.io.BufferedReader
6  import java.io.IOException
7  import java.io.InputStream
8  import java.io.InputStreamReader
9  import java.nio.MappedByteBuffer
10
```

Εικόνα 5.80: Imports

Η συνάρτηση **extractNamesFromMetadata** που βρίσκεται στις **(γραμμές 13-32)** προσπαθεί να διαβάσει τα ονόματα των κλάσεων από τα metadata ενός μοντέλου TensorFlow Lite. Ανοίγει το συσχετισμένο αρχείο "temp_meta.txt", εντοπίζει την ενότητα που περιέχει τα ονόματα με regex, εξάγει κάθε όνομα και τα επιστρέφει σε μια λίστα. Αν δεν υπάρχει αρχείο ή προκύψει σφάλμα, επιστρέφει άδεια λίστα.

```
11  object MetaData {  new *
12
13      fun extractNamesFromMetadata(model: MappedByteBuffer): List<String> {  new *
14          try {
15              val metadataExtractor = MetadataExtractor(model)
16              val inputStream = metadataExtractor.getAssociatedFile( fileName: "temp_meta.txt")
17              val metadata = inputStream?.bufferedReader()?.use { it.readText() } ?: return emptyList()
18
19              val regex = Regex( pattern: "'names': \\{(.*?)\\}", RegexOption.DOT_MATCHES_ALL)
20
21              val match = regex.find(metadata)
22              val namesContent = match?.groups?.get(1)?.value ?: return emptyList()
23
24              val regex2 = Regex( pattern: "\"([^\"]*)\"|'([^']*)'")
25              val match2 = regex2.findAll(namesContent)
26              val list = match2.map { it.groupValues[1].ifEmpty { it.groupValues[2] }}.toList()
27
28              return list
29          } catch (_: Exception) {
30              return emptyList()
31          }
32      }
```

Εικόνα 5.81: Μέθοδος extractNamesFromMetadata

Η συνάρτηση **extractNamesFromLabelFile** που βρίσκεται στις **(γραμμές 34-56)** διαβάζει τα ονόματα των κλάσεων από ένα αρχείο κειμένου στο φάκελο assets της εφαρμογής, γραμμή-γραμμή, και τα επιστρέφει σε μια λίστα. Αν παρουσιαστεί σφάλμα κατά το άνοιγμα ή την ανάγνωση του αρχείου, επιστρέφει άδεια λίστα. Επιπλέον, η μεταβλητή TEMP_CLASSES παρέχει μια προεπιλεγμένη λίστα 1000 ονομάτων κλάσεων (“class1”, “class2”, …, “class1000”) για χρήση όταν δεν υπάρχουν διαθέσιμα metadata ή αρχείο ετικετών.

```
34      fun extractNamesFromLabelFile(context: Context, labelPath: String): List<String> {  new *
35          val labels = mutableListOf<String>()
36          try {
37              val inputStream: InputStream = context.assets.open(labelPath)
38              val reader = BufferedReader(InputStreamReader(inputStream))
39
40              var line: String? = reader.readLine()
41              while (line != null && line != "") {
42                  labels.add(line)
43                  line = reader.readLine()
44              }
45
46              reader.close()
47              inputStream.close()
48              return labels
49          } catch (e: IOException) {
50              return emptyList()
51          }
52      }
53
54
55      val TEMP_CLASSES = List( size: 1000) { "class${it + 1}" }
56  }
```

Εικόνα 5.82: Μέθοδος extractNamesFromLabelFile

5.7.11 Περιγραφή Κλάσης “Output0”

Η **Output0** χρησιμεύει ως δομή δεδομένων για την αναπαράσταση κάθε αντικειμένου που ανιχνεύει το μοντέλο. Περιέχει τις συντεταγμένες του πλαισίου, την κλάση, την πιθανότητα ανίχνευσης και βάρη μάσκας, διευκολύνοντας την επιπλέον επεξεργασία όπως η εφαρμογή μάσκας, η εμφάνιση αποτελεσμάτων στην οθόνη ή η φιλτραρισμένη επιλογή των σημαντικότερων ανιχνεύσεων. Δηλαδή, συγκεντρώνει όλα τα απαραίτητα δεδομένα για να χειριστούμε και να αξιοποιήσουμε τα αποτελέσματα της ανίχνευσης αντικειμένων.

```
1   package org.tensorflow.lite.examples.objectdetection
2
3   data class Output0(  new *
4           val x1: Float,
5           val y1: Float,
6           val x2: Float,
7           val y2: Float,
8           val cx: Float,
9           val cy: Float,
10          val w: Float,
11          val h: Float,
12          val cnf: Float,
13          val cls: Int,
14          val clsName: String,
15          val maskWeight: List<Float>,
16          val distance: Double? = -1.0
17
18
19  )
```

Εικόνα 5.83: Κλάση data Output0

5.7.12 Περιγραφή Κλάσης “SegmentationResult”

Η κλάση **SegmentationResult** είναι μια δομή δεδομένων που αντιπροσωπεύει το αποτέλεσμα της ανίχνευσης και της απομόνωσης (segmentation) ενός αντικειμένου. Αποθηκεύει το πλάισιο (box) από την Output0, τη μάσκα (mask) του αντικειμένου και την απόσταση (distance). Παρέχει επίσης υλοποιήσεις για equals και hashCode βασισμένες στη μάσκα, ώστε να μπορούμε να συγκρίνουμε ή να χρησιμοποιούμε τα αποτελέσματα σε συλλογές χωρίς να επαναλαμβάνονται. Ουσιαστικά, συνδυάζει όλες τις πληροφορίες που χρειάζονται για την εμφάνιση ή επεξεργασία των segmentations.

```kotlin
package org.tensorflow.lite.examples.objectdetection

data class SegmentationResult(  new *
    val box: Output0,
    val mask: Array<FloatArray>,
    val distance: Double
) {
    override fun equals(other: Any?): Boolean {  new *
        if (this === other) return true
        if (javaClass != other?.javaClass) return false

        other as SegmentationResult

        return mask.contentDeepEquals(other.mask)
    }

    override fun hashCode(): Int {  new *
        return mask.contentDeepHashCode()
    }
}
```

Εικόνα 5.84: Κλάση data SegmentationResult

5.7.13 Περιγραφή Κλάσης "YoloSegmentationPipeline"

Η κλάση **YoloSegmentationPipeline** χρησιμεύει ως σωλήνας για ανίχνευση και απομόνωση αντικειμένων χρησιμοποιώντας το YOLO μοντέλο. Αρχικοποιεί ένα αντικείμενο InstanceSegmentation με το προκαθορισμένο μοντέλο και τις ετικέτες, και παρέχει μεθόδους για ανίχνευση σε εικόνες bitmap. Επιπλέον, επιτρέπει την ενημέρωση παραμέτρων όπως το όριο εμπιστοσύνης (threshold), τον αριθμό νημάτων (threads) και τον μέγιστο αριθμό αποτελεσμάτων (maxResults). Η μέθοδος clear κλείνει τον segmenter για να απελευθερωθούν οι πόροι. Ουσιαστικά, η κλάση αυτή τυλίγει και διαχειρίζεται την λειτουργία του YOLO segmentation μοντέλου, προσφέροντας ευκολία στη χρήση του σε εφαρμογές Android.

Δήλωση Πακέτου(γραμμή 1): Δηλώνεται το πακέτο στο οποίο ανήκει η κλάση YoloSegmentationPipeline.

Imports(γραμμές 3-5): Σε αυτό το τμήμα, ο κώδικας εισάγει απαραίτητες βιβλιοθήκες που χρειάζεται η εφαρμογή για να λειτουργήσει.

```
package org.tensorflow.lite.examples.objectdetection

import android.content.Context
import android.graphics.Bitmap
import android.util.Log
```

Εικόνα 5.85: Imports

Το παρακάτω κομμάτι της **YoloSegmentationPipeline** αφορά την αρχικοποίηση της κλάσης. Αρχικά βλέπουμε τις παραμέτρους κατασκευαστή. Στην συνέχεια το πεδίο segmenter το οποίο αποθηκεύει το αντικείμενο InstanceSegmentation που εκτελεί πραγματικά την ανίχνευση και τον υπολογισμό των μασκών. Παρακάτων έχουμε το init block που καταγράφει ένα μήνυμα log ότι ξεκινά η αρχικοποίηση της pipeline και προσπαθεί να δημιουργήσει ένα νέο InstanceSegmentation, φορτώνοντας το YOLO μοντέλο (yolo11n-seg_float16.tflite) και τις ετικέτες (labels.txt) και συνδέοντας τον listener. Αν η αρχικοποίηση πετύχει, καταγράφεται επιτυχές μήνυμα, σε διαφορετική περίπτωση καταγράφεται σφάλμα και καλείται η onError.

```
class YoloSegmentationPipeline( new*
    private val context: Context,
    private val listener: InstanceSegmentation.InstanceSegmentationListener
) : DetectionPipeline {

    private var segmenter: InstanceSegmentation? = null


    init { new*
        Log.d( tag: "YoloSegmentationPipelin", msg: "Initializing YoloSegmentationPipeline")
        try {
            segmenter = InstanceSegmentation(
                context = context,
                modelPath = "yolo11n-seg_float16.tflite",
                labelPath = "labels.txt",
                instanceSegmentationListener = listener,
                message = { Log.d( tag: "YOLO_PIPELINE", it) }
            )
            Log.d( tag: "YoloSegmentationPipelin", msg: "YoloSegmentationPipeline initialized successfully")
        } catch (e: Exception) {
            Log.e( tag: "YoloSegmentationPipelin", msg: "Error initializing InstanceSegmentation", e)
            listener.onError( error: "Failed to initialize segmentation model.")
        }
    }
```

Εικόνα 5.86: initBlock

Στις **(γραμμές 33-57)** έχουμε τις λειτουργίες ανίχνευσης και διαχείρισης του YoloSegmentationPipeline:

1. **detect(bitmap: Bitmap, rotation: Int):** Καλείται για να εκτελεστεί η ανίχνευση αντικειμένων και segmentation πάνω σε ένα Bitmap. Επίσης καταγράφει ενα log και τέλος εκκινεί το μοντέλο segmentation μόνο αν το segmenter δεν ειναι null.
2. **Ρυθμίσεις segmenter**:

a) **updateThreshold(threshold: Float):** Προσαρμόζει το κατώφλι εμπιστοσύνης για τις ανιχνεύσεις, ώστε να φιλτράρονται τα αντικείμενα με χαμηλή πιθανότητα.

b) **updateThreads(threads: Int):** Αλλάζει τον αριθμό των νήματων (threads) που χρησιμοποιούνται για την εκτέλεση του μοντέλου, επιτρέποντας καλύτερη απόδοση ή οικονομία πόρων.

c) **updateMaxResults(maxResults: Int):** Περιορίζει τον μέγιστο αριθμό αποτελεσμάτων που επιστρέφει το μοντέλο.

3. **Clear():** Αυτή η μέθοδος τερματίζει και καθαρίζει το segmenter, απελευθερώνοντας πόρους όταν η pipeline δεν χρειάζεται πλέον

```
32
33      override fun detect(bitmap: Bitmap, rotation: Int) {  new *
34          Log.d( tag: "YoloSegmentationPipelin",  msg: "Detecting with YoloSegmentationPipeline")
35
36
37
38          segmenter?.invoke(bitmap)
39      }
40
41      fun updateThreshold(threshold: Float) {  new *
42          segmenter?.updateThreshold(threshold)
43      }
44
45      fun updateThreads(threads: Int) {  new *
46          segmenter?.updateThreads(threads)
47      }
48
49      fun updateMaxResults(maxResults: Int) {  new *
50          segmenter?.updateMaxResults(maxResults)
51      }
52
53
54      override fun clear() {  new *
55          segmenter?.close()
56      }
57  }
58
```

Εικόνα 5.87: Μέθοδος detect, updateThreshold, updateThreds, updateMaxResults & clear

5.7.14 Περιγραφή Script “activity_main.xml”

Το **activity_main.xml** ορίζει το Layout της κύριας δραστηριότητας της εφαρμογής και έχει τα εξής βασικά στοιχεία:

1. **CoordinatorLayout**:

   a) Ορίζεται ως το βασικό container για όλη την οθόνη.

   b) Συντονίζει τα εσωτερικά view components, π.χ. scrolling με toolbar.

2. **RelativeLayout**:

a) Περιέχει όλα τα υπόλοιπα στοιχεία του layout και επιτρέπει την τοποθέτηση view σε σχέση με άλλα.

b) Ορίζεται να καταλαμβάνει όλο το μέγεθος της οθόνης.

3. **FragmentContainerView**:

a) Χρησιμοποιείται για να φιλοξενήσει το NavHostFragment, που διαχειρίζεται τη navigation μεταξύ των fragments της εφαρμογής.

b) Ορίζεται με app:navGraph="@navigation/nav_graph" για να γνωρίζει το navigation graph της εφαρμογής.

c) android:keepScreenOn="true" διασφαλίζει ότι η οθόνη δεν θα σβήσει κατά τη χρήση.

d) Τοποθετείται κάτω από το toolbar λόγω του android:layout_marginTop="?android:attr/actionBarSize".

4. **Toolbar**:

a) Προσφέρει ένα προσαρμοσμένο action bar στην κορυφή της οθόνης.

b) Στυλιζέται με @color/toolbar_background και έχει ύψος ?attr/actionBarSize.

```
<androidx.coordinatorlayout.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:background="@android:color/transparent"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

    <androidx.fragment.app.FragmentContainerView
        android:id="@+id/fragment_container"
        android:name="androidx.navigation.fragment.NavHostFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="@android:color/transparent"
        android:keepScreenOn="true"
        app:defaultNavHost="true"
        app:navGraph="@navigation/nav_graph"
        android:layout_marginTop="?android:attr/actionBarSize"
        tools:context=".MainActivity"/>

        <androidx.appcompat.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:layout_alignParentTop="true"
            android:background="@color/toolbar_background">


        </androidx.appcompat.widget.Toolbar>

    </RelativeLayout>
</androidx.coordinatorlayout.widget.CoordinatorLayout>
```

Εικόνα 5.88: Activity_main.xml

5.7.15 Περιγραφή Script “fragment_camera.xml”

Το **fragment_camera.xml** ορίζει το layout για την οθόνη κάμερας της εφαρμογής και περιλαμβάνει τα εξής κύρια στοιχεία:

1. **CoordinatorLayout:**
    a) Βασικό container που φιλοξενεί όλα τα υπόλοιπα στοιχεία.
    b) Επιτρέπει καλύτερη διαχείριση του overlay και του bottom sheet.
2. **PreviewView (view_finder):**
    a) Προσφέρει το view της ζωντανής ροής της κάμερας.

b) Το app:scaleType="fillStart" καθορίζει πώς το preview γεμίζει τον χώρο του.

3. **OverlayView (overlay):**

a) Custom view που χρησιμοποιείται για να ζωγραφίζει πάνω στο preview αντικείμενα ανίχνευσης, μάσκες ή bounding boxes.

4. **ImageView (imageViewResult):**

a) Χρησιμοποιείται για να εμφανίζει επεξεργασμένες εικόνες ή αποτελέσματα ανίχνευσης.

b) scaleType="fitCenter" και adjustViewBounds="true" διασφαλίζουν ότι η εικόνα εμφανίζεται σωστά χωρίς παραμόρφωση.

5. **<include> (bottom_sheet_layout):**

a) Ενσωματώνει το layout info_bottom_sheet, που πιθανότατα περιέχει πληροφορίες σχετικά με τα ανιχνευθέντα αντικείμενα σε μορφή bottom sheet.

```
<androidx.coordinatorlayout.widget.CoordinatorLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/camera_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <androidx.camera.view.PreviewView
        android:id="@+id/view_finder"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:scaleType="fillStart"/>

    <org.tensorflow.lite.examples.objectdetection.OverlayView
        android:id="@+id/overlay"
        android:layout_height="match_parent"
        android:layout_width="match_parent" />

    <ImageView
        android:id="@+id/imageViewResult"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:scaleType="fitCenter"
        android:layout_gravity="center"
        android:adjustViewBounds="true" />



    <include
        android:id="@+id/bottom_sheet_layout"
        layout="@layout/info_bottom_sheet" />
</androidx.coordinatorlayout.widget.CoordinatorLayout>
```

Εικόνα 5.89: fragment_camera.xml

5.7.16 Περιγραφή Script “info_bottom_sheet.xml”

Το **info_bottom_sheet.xml ο**ρίζει το layout για το bottom sheet της εφαρμογής, το οποίο εμφανίζει ρυθμίσεις και πληροφορίες για το μοντέλο ανίχνευσης αντικειμένων:

1. Κύριο container:

a) **NestedScrollView**: Επιτρέπει το scrolling όταν οι ρυθμίσεις ξεπερνούν το διαθέσιμο ύψος.

b) **BottomSheetBehavior**: Καθορίζει ότι είναι bottom sheet με συγκεκριμένο peek height.

c) **background και padding:** Δίνουν στυλ και απόσταση περιεχομένου.

```
<androidx.core.widget.NestedScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/bottom_sheet_layout"
    android:layout_width="match_parent"
    android:layout_gravity="center_horizontal"
    android:background="@color/bottom_sheet_background"
    android:gravity="center_horizontal"
    android:orientation="vertical"
    android:padding="16dp"
    android:layout_height="wrap_content"

    app:behavior_hideable="false"
    app:behavior_peekHeight="50dp"
    app:layout_behavior="com.google.android.material.bottomsheet.BottomSheetBehavior">
```

Εικόνα 5.90: info_bottom_sheet.xml

2. **LinearLayoutCompat**

a) Είναι ένας κατακόρυφος (vertical) γραμμικός διατάκτης που καλύπτει όλο το πλάτος και ύψος του γονικού στοιχείου (match_parent).

b) Χρησιμοποιείται για να τοποθετήσει τα περιεχόμενα του bottom sheet σε στήλη.

3. **RelativeLayout**

a) Έχει wrap_content ύψος και πλήρες πλάτος (match_parent).

b) Περιλαμβάνει ένα εικονίδιο (ImageView) που εμφανίζει ένα κλείδωμα (ic_lock_48), κεντραρισμένο οριζόντια και κατακόρυφα μέσα στο RelativeLayout.

c) Το ImageView έχει περιγραφή για accessibility (contentDescription) ώστε να είναι κατανοητό από screen readers.

```
    <androidx.appcompat.widget.LinearLayoutCompat
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
        <!-- Chevron indicating that the bottom sheet is expandable -->
        <RelativeLayout
            android:layout_height="wrap_content"
            android:layout_width="match_parent">
            <ImageView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_centerHorizontal="true"
                android:layout_centerVertical="true"
                android:src="@drawable/ic_lock_48"
                android:contentDescription="Bottom sheet expandable indicator" />
        </RelativeLayout>
```

Εικόνα 5.91: info_bottom_sheet.xml

4. **LinearLayoutCompat**
   a) Οριζόντια διάταξη (orientation="horizontal") που καταλαμβάνει όλο το πλάτος και έχει ύψος ανάλογο με το περιεχόμενο (wrap_content).
   b) Έχει επάνω περιθώριο (layout_marginTop) για να ξεχωρίζει από την προηγούμενη σειρά.
5. Περιέχει δύο **TextView** στοιχεία:
   a) **inference_time_label:** Εμφανίζει το κείμενο "Inference time" (ή ό,τι καθορίζεται στο string resource).
      i. Σταθερό πλάτος ανάλογα με το περιεχόμενο (wrap_content).
      ii. Χρώμα και μέγεθος κειμένου προσαρμοσμένα στο στυλ του bottom sheet.
   b) **inference_time_val**: Εμφανίζει την τιμή του χρόνου εκτίμησης (π.χ., "0ms").
      i. Καταλαμβάνει το υπόλοιπο πλάτος (match_parent) και στοιχίζεται στο τέλος (gravity="end").

```
<!-- Inference time row -->
<androidx.appcompat.widget.LinearLayoutCompat
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="16dp"
    android:orientation="horizontal">
    <TextView
        android:id="@+id/inference_time_label"
        android:text="Inference Time"
        android:textColor="@color/bottom_sheet_text_color"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerVertical="true"
        android:textSize="20sp" />
    <TextView
        android:id="@+id/inference_time_val"
        android:text="0ms"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="end"
        android:layout_centerVertical="true"
        android:textColor="@color/bottom_sheet_text_color"
        android:textSize="20sp" />
</androidx.appcompat.widget.LinearLayoutCompat>
```

Εικόνα 5.92: info_bottom_sheet.xml

Το επόμενο τμήμα κώδικα δημιουργεί το όριο σιγουριάς για τις προβλέψεις) του μοντέλου.

- Ολόκληρη η σειρά βρίσκεται σε **RelativeLayout**, ώστε να μπορεί να τοποθετεί στοιχεία αριστερά και δεξιά.

- Στα αριστερά υπάρχει ένα TextView με την ετικέτα *Confidence threshold*.
- Στα δεξιά υπάρχει ένα **LinearLayout** με τρία στοιχεία:
  - **Κουμπί "-"** (threshold_minus) → μειώνει την τιμή του threshold.
  - **TextView (**threshold_value**)** → εμφανίζει την τρέχουσα τιμή (π.χ. 0.50).
  - **Κουμπί "+"** (threshold_plus) → αυξάνει την τιμή του threshold.

Ο χρήστης μπορεί εύκολα να αυξομειώσει το κατώφλι εμπιστοσύνης του αλγορίθμου, ρυθμίζοντας πόσο σίγουρες πρέπει να είναι οι προβλέψεις για να εμφανιστούν.

```
<!-- ML confidence threshold adjustment row -->
<RelativeLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="@dimen/bottom_sheet_default_row_margin"
    android:gravity="center"
    android:orientation="horizontal">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/label_confidence_threshold"
        android:layout_centerVertical="true"
        android:textSize="@dimen/bottom_sheet_text_size"
        android:textColor="@color/bottom_sheet_text_color" />
    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:gravity="center_vertical"
        android:orientation="horizontal"
        >
```

Εικόνα 5.93: info_bottom_sheet.xml

```
96              <androidx.appcompat.widget.AppCompatImageButton
97                  android:id="@+id/threshold_minus"
98                  android:layout_width="@dimen/bottom_sheet_control_btn_size"
99                  android:layout_height="@dimen/bottom_sheet_control_btn_size"
100                 android:contentDescription="@string/alt_bottom_sheet_threshold_button_minus"
101                 android:src="@drawable/ic_minus" />
102
103             <TextView
104                 android:id="@+id/threshold_value"
105                 android:layout_width="wrap_content"
106                 android:layout_height="wrap_content"
107                 android:layout_marginLeft="@dimen/bottom_sheet_control_text_side_margin"
108                 android:layout_marginRight="@dimen/bottom_sheet_control_text_side_margin"
109                 android:gravity="center"
110                 android:minEms="@integer/bottom_sheet_control_text_min_ems"
111                 android:text="0.50"
112                 android:textColor="@color/bottom_sheet_text_color"
113                 android:textSize="@dimen/bottom_sheet_text_size" />
114
115             <androidx.appcompat.widget.AppCompatImageButton
116                 android:id="@+id/threshold_plus"
117                 android:layout_width="@dimen/bottom_sheet_control_btn_size"
118                 android:layout_height="@dimen/bottom_sheet_control_btn_size"
119                 android:contentDescription="@string/alt_bottom_sheet_threshold_button_plus"
120                 android:src="@drawable/ic_plus"  />
121         </LinearLayout>
122     </RelativeLayout>
123
```

Εικόνα 5.94: info_bottom_sheet.xml

Το παρακάτω τμήμα αφορά τον έλεγχο της παραμέτρου **Max Results**, δηλαδή τον μέγιστο αριθμό αντικειμένων/ανιχνεύσεων που θα εμφανίζει το μοντέλο.

- Στα αριστερά υπάρχει ένα TextView με την ετικέτα *Max results*.
- Στα δεξιά (μέσα σε ένα **LinearLayout**) υπάρχουν:
  - **Κουμπί "-"** (max_results_minus) → μειώνει το όριο των αποτελεσμάτων.
  - **TextView (**max_results_value**)** → δείχνει τον τρέχοντα αριθμό αποτελεσμάτων (π.χ. 3).
  - **Κουμπί "+"** (max_results_plus) → αυξάνει το όριο.

Δίνει στον χρήστη τη δυνατότητα να ελέγχει πόσα αποτελέσματα θέλει να επιστρέφει το μοντέλο από κάθε εκτέλεση. Για παράδειγμα, αν το όριο είναι χαμηλό, εμφανίζονται μόνο οι πιο σίγουρες ανιχνεύσεις· αν είναι υψηλό, εμφανίζονται περισσότερες.

```
<!-- ML max results adjustment row -->
<RelativeLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="@dimen/bottom_sheet_default_row_margin"
    android:gravity="center"
    android:orientation="horizontal">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/label_max_results"
        android:textSize="@dimen/bottom_sheet_text_size"
        android:layout_centerVertical="true"
        android:textColor="@color/bottom_sheet_text_color" />

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:gravity="center"
        android:orientation="horizontal">
        <androidx.appcompat.widget.AppCompatImageButton
            android:id="@+id/max_results_minus"
            android:layout_width="@dimen/bottom_sheet_control_btn_size"
            android:layout_height="@dimen/bottom_sheet_control_btn_size"
            android:contentDescription="@string/alt_bottom_sheet_max_results_button_minus"
            android:src="@drawable/ic_minus" />
```

Εικόνα 5.95: info_bottom_sheet.xml

```
            android:src="@drawable/ic_minus" />
        <TextView
            android:id="@+id/max_results_value"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:minEms="@integer/bottom_sheet_control_text_min_ems"
            android:gravity="center"
            android:layout_marginLeft="@dimen/bottom_sheet_control_text_side_margin"
            android:layout_marginRight="@dimen/bottom_sheet_control_text_side_margin"
            android:text="3"
            android:textColor="@color/bottom_sheet_text_color"
            android:textSize="@dimen/bottom_sheet_text_size" />
        <androidx.appcompat.widget.AppCompatImageButton
            android:id="@+id/max_results_plus"
            android:layout_width="@dimen/bottom_sheet_control_btn_size"
            android:layout_height="@dimen/bottom_sheet_control_btn_size"
            android:contentDescription="@string/alt_bottom_sheet_max_results_button_plus"
            android:src="@drawable/ic_plus" />
    </LinearLayout>
</RelativeLayout>
```

Εικόνα 5.96: info_bottom_sheet.xml

Αυτό το κομμάτι του layout προσθέτει μία γραμμή στο bottom sheet που επιτρέπει στον χρήστη να ρυθμίσει τον αριθμό των threads απο το μενού που θα χρησιμοποιεί το μοντέλο κατά την εκτέλεση.

- Στα αριστερά υπάρχει ένα **TextView** με την ετικέτα Threads.
- Στα δεξιά, μέσα σε ένα **LinearLayout**, βρίσκονται:
  - Ένα κουμπί **"–"** (threads_minus) → μειώνει τον αριθμό threads.
  - Ένα **TextView (**threads_value**)** που δείχνει την τρέχουσα τιμή (εδώ, 2).
  - Ένα κουμπί **"+"** (threads_plus) → αυξάνει τον αριθμό threads.

```
172     <!-- Number of threads adjustment row -->
173     <RelativeLayout
174         android:layout_width="match_parent"
175         android:layout_height="wrap_content"
176         android:layout_marginTop="@dimen/bottom_sheet_default_row_margin"
177         android:gravity="center"
178         android:orientation="horizontal">
179
180         <TextView
181             android:layout_width="wrap_content"
182             android:layout_height="wrap_content"
183             android:text="@string/label_threads"
184             android:layout_centerVertical="true"
185             android:textSize="@dimen/bottom_sheet_text_size"
186             android:textColor="@color/bottom_sheet_text_color" />
187         <LinearLayout
188             android:layout_width="wrap_content"
189             android:layout_height="wrap_content"
190             android:layout_alignParentRight="true"
191             android:gravity="center_vertical"
192             android:orientation="horizontal"
193             >
```

Εικόνα 5.97: info_bottom_sheet.xml

```
194
195             <androidx.appcompat.widget.AppCompatImageButton
196                 android:id="@+id/threads_minus"
197                 android:layout_width="@dimen/bottom_sheet_control_btn_size"
198                 android:layout_height="@dimen/bottom_sheet_control_btn_size"
199                 android:contentDescription="@string/alt_bottom_sheet_thread_button_minus"
200                 android:src="@drawable/ic_minus" />
201
202             <TextView
203                 android:id="@+id/threads_value"
204                 android:layout_width="wrap_content"
205                 android:layout_height="wrap_content"
206                 android:layout_marginLeft="@dimen/bottom_sheet_control_text_side_margin"
207                 android:layout_marginRight="@dimen/bottom_sheet_control_text_side_margin"
208                 android:gravity="center"
209                 android:minEms="@integer/bottom_sheet_control_text_min_ems"
210                 android:text="2"
211                 android:textColor="@color/bottom_sheet_text_color"
212                 android:textSize="@dimen/bottom_sheet_text_size" />
213
214             <androidx.appcompat.widget.AppCompatImageButton
215                 android:id="@+id/threads_plus"
216                 android:layout_width="@dimen/bottom_sheet_control_btn_size"
217                 android:layout_height="@dimen/bottom_sheet_control_btn_size"
218                 android:contentDescription="@string/alt_bottom_sheet_thread_button_plus"
219                 android:src="@drawable/ic_plus"  />
220         </LinearLayout>
221     </RelativeLayout>
```

Εικόνα 5.98: info_bottom_sheet.xml

Το παρακάτω τμήμα του info_bottom_sheet.xml προσθέτει μια γραμμή που επιτρέπει στον χρήστη να επιλέξει το delegate που θα χρησιμοποιηθεί από το TensorFlow Lite μοντέλο.

- Στα αριστερά υπάρχει ένα TextView με την ετικέτα Delegate.
- Στα δεξιά υπάρχει ένα Spinner, δηλαδή ένα dropdown μενού, που γεμίζει με επιλογές από τον πίνακα @array/delegate_spinner_titles (π.χ. CPU, GPU, NNAPI).

```
223        <!-- Delegate selection row -->
224        <RelativeLayout
225            android:layout_width="match_parent"
226            android:layout_height="wrap_content"
227            android:layout_marginTop="@dimen/bottom_sheet_default_row_margin">
228
229            <TextView
230                android:text="@string/label_delegate"
231                android:layout_width="wrap_content"
232                android:layout_height="wrap_content"
233                android:textSize="@dimen/bottom_sheet_text_size"
234                android:textColor="@color/bottom_sheet_text_color" />
235
236            <androidx.appcompat.widget.AppCompatSpinner
237                android:id="@+id/spinner_delegate"
238                android:layout_width="wrap_content"
239                android:layout_height="wrap_content"
240                android:minWidth="@dimen/bottom_sheet_spinner_delegate_min_width"
241                android:spinnerMode="dropdown"
242                android:theme="@style/BottomSheetSpinnerItemStyle"
243                android:layout_alignParentRight="true"
244                android:entries="@array/delegate_spinner_titles"/>
245
246        </RelativeLayout>
247
```

Εικόνα 5.99: info_bottom_sheet.xml

Στο τελευταίο τμήμα του info_bottom_sheet.xml δημιουργείτε μια γραμμή ώστε να διαλέγει ο χρήστης μοντέλο.

- Στα αριστερά υπάρχει ένα **TextView** με την ετικέτα Models.
- Στα δεξιά υπάρχει ένα **Spinner** που εμφανίζει τις διαθέσιμες επιλογές μοντέλων, οι οποίες ορίζονται στον πίνακα @array/models_spinner_titles

```
248         <!-- Model selection row -->
249         <RelativeLayout
250             android:layout_width="match_parent"
251             android:layout_height="wrap_content"
252             android:layout_marginTop="@dimen/bottom_sheet_default_row_margin">
253
254             <TextView
255                 android:text="@string/label_models"
256                 android:layout_width="wrap_content"
257                 android:layout_height="wrap_content"
258                 android:textSize="@dimen/bottom_sheet_text_size"
259                 android:textColor="@color/bottom_sheet_text_color" />
260
261             <androidx.appcompat.widget.AppCompatSpinner
262                 android:id="@+id/spinner_model"
263                 android:layout_width="wrap_content"
264                 android:layout_height="wrap_content"
265                 android:minWidth="@dimen/bottom_sheet_spinner_model_min_width"
266                 android:spinnerMode="dropdown"
267                 android:theme="@style/BottomSheetSpinnerItemStyle"
268                 android:layout_alignParentRight="true"
269                 android:entries="@array/models_spinner_titles"/>
270
271         </RelativeLayout>
272     </androidx.appcompat.widget.LinearLayoutCompat>
273 </androidx.core.widget.NestedScrollView>
```

Εικόνα 5.100: info_bottom_sheet.xml

5.8 Επίλογος

Στο παρόν κεφάλαιο έγινε αναλυτική παρουσίαση της δομής και της λειτουργικότητας της εφαρμογής. Αναλύθηκαν τα βασικά τμήματα του κώδικα και του γραφικού περιβάλλοντος, καθώς και ο τρόπος με τον οποίο αυτά συνεργάζονται για να επιτύχουν την υλοποίηση της ανίχνευσης και τμηματοποίησης αντικειμένων σε πραγματικό χρόνο. Παράλληλα, είδαμε την χρησιμότητα της εφαρμογής ως εργαλείο υποστήριξης ατόμων με προβλήματα όρασης, μέσα από φωνητική περιγραφή και απτική ανατροφοδότηση. Τέλος, έγινε παρουσίαση των ρυθμίσεων και των δυνατοτήτων προσαρμογής, οι οποίες προσφέρουν στον χρήστη μεγαλύτερη ευελιξία και έλεγχο κατά τη χρήση της εφαρμογής.

Κεφάλαιο 6ο: Δοκιμές και Αξιολόγηση της Εφαρμογής

6.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα δούμε την διαδικασία δοκιμών και αξιολόγησης της εφαρμογής με βάση ένα ερωτηματολόγιο. Ο στόχος ήταν να διερευνηθεί η αποτελεσματικότητα και η χρηστικότητα της εφαρμογής στην υποστήριξη ατόμων με προβλήματα όρασης. Για τον σκοπό αυτό σχεδιάστηκαν σενάρια χρήσης, πραγματοποιήθηκαν δοκιμές με συμμετέχοντες.

6.2 Ερωτηματολόγιο και Στατιστικά Στοιχεία

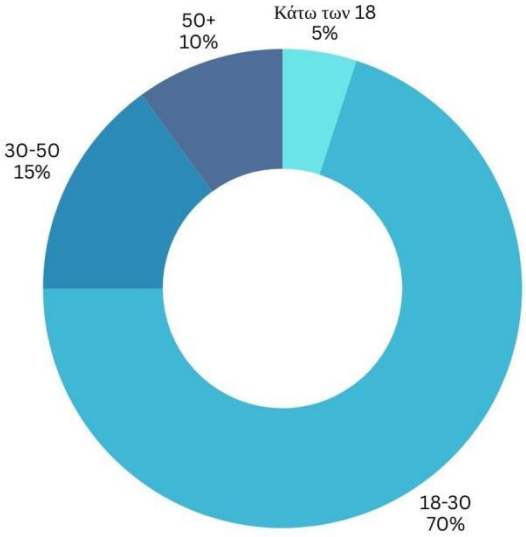
Από τούς συμμετέχοντες ζητήθηκε να δοκιμάσουν την εφαρμογή με κλειστά μάτια, ώστε να προσομοιωθεί η εμπειρία ενός τυφλού χρήστη. Αφού δοκίμασαν την εφαρμογή με κλειστά μάτια, τους ζητήθηκε να την αξιολογήσουν βάσει εικοσιέξι ζευγών χαρακτηριστικών λέξεων (π.χ. «παρελκιστικό-υποστηρικτικό», «δυσνόητο - κατανοητό»).

Η αξιολόγηση πραγματοποιήθηκε σε κλίμακα από 1 έως 8, όπου η χαμηλότερη τιμή αντιστοιχούσε στο αρνητικό και η υψηλότερη στο θετικό. Με αυτό τον τρόπο, συγκεντρώθηκαν κάποια δεδομένα που αποτυπώνουν τις εντυπώσεις των χρηστών για τη χρηστικότητα, την ευχρηστία και τη συνολική εμπειρία με την εφαρμογή.

Παρακάτω έχουμε το ερωτηματολόγιο καθώς και ορισμένα στατιστικά στοιχεία που προέκυψαν από την αξιολόγηση.

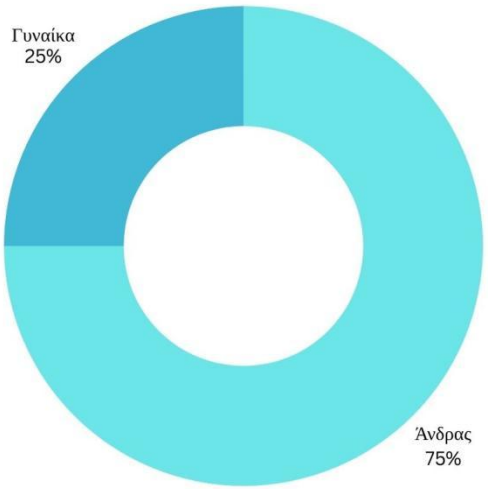
παρελκυστικό	o o o o o o o	υποστηρικτικό
περίπλοκο	o o o o o o o	εύκολο
ανεπαρκές	o o o o o o o	επαρκές
μπερδεμένο	o o o o o o o	σαφές
βαρετό	o o o o o o o	συναρπαστικό
αδιάφορο	o o o o o o o	ενδιαφέρον
συμβατικό	o o o o o o o	εφευρετικό
συνηθισμένο	o o o o o o o	πρωτοπόρο

Στην συνέχεια έχουμε το διάγραμμα που παρουσιάζεται η κατανομή των ηλικιών των συμμετεχόντων στο ερωτηματολόγιο. Όπως φαίνεται, η πλειοψηφία των χρηστών (70%) ανήκει στις ηλικίες τών 18–30 ετών, κάτι που δείχνει ότι η αξιολόγηση βασίστηκε κυρίως σε νέους ενήλικες με εξοικείωση στη χρήση κινητών συσκευών. Οι υπόλοιπες ηλικιακές κατηγορίες είχαν μικρότερη συμμετοχή, προσφέροντας ωστόσο μια πιο σφαιρική εικόνα για την εμπειρία χρήσης της εφαρμογής.



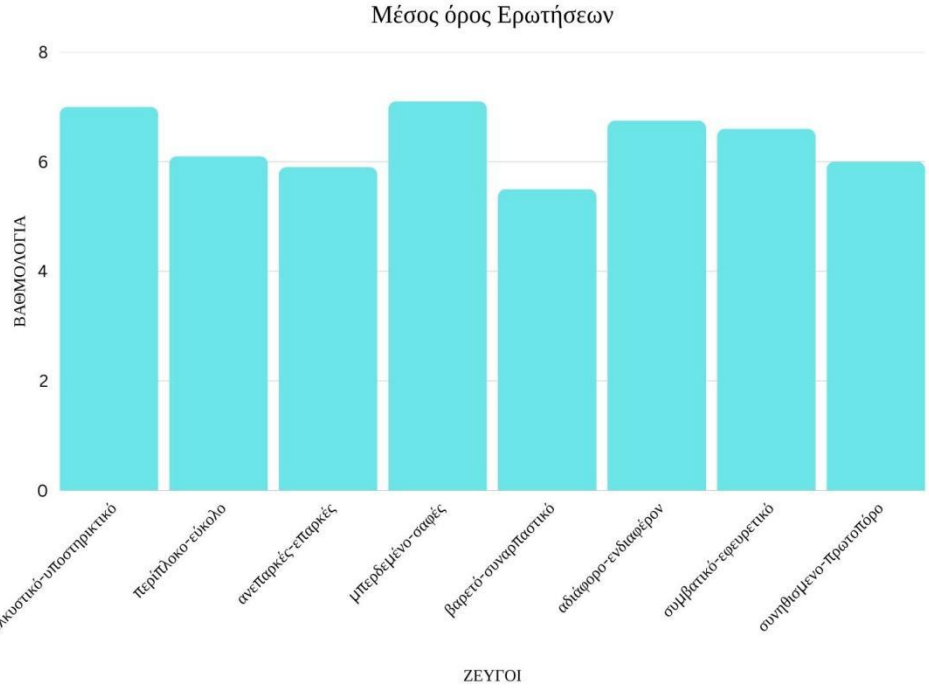
Διάγραμμα 6.1: Διάγραμμα με ποσοστά ηλικίας

Στην συνέχεια έχουμε το διάγραμμα με το τι φύλλο είναι οι συμμετέχοντες, με το 75% περίπου του δείγματος να αποτελείται από άντρες και το 25% περίπου να αποτελείται από γυναίκες σύμφωνα με το διάγραμμα παρακάτω:



Διάγραμμα 6.2: Διάγγραμα με ποσοστό φύλλου.

Στο επόμενο διάγραμμα παρουσιάζονται οι μέσοι όροι των απαντήσεων των χρηστών για καθεμία από τις ερωτήσεις του ερωτηματολογίου. Οι τιμές αυτές αποτυπώνουν συνοπτικά τη γενική άποψη των συμμετεχόντων απέναντι στην εφαρμογή και δίνουν μια εικόνα για το πώς αξιολογήθηκε σε ζητήματα χρηστικότητας, κατανοητότητας, φιλικότητας και καινοτομίας. Η ανάλυση των μέσων όρων διευκολύνει την εξαγωγή συμπερασμάτων σχετικά με τα δυνατά σημεία της εφαρμογής, αλλά και τις περιοχές που ενδεχομένως χρειάζονται βελτίωση.



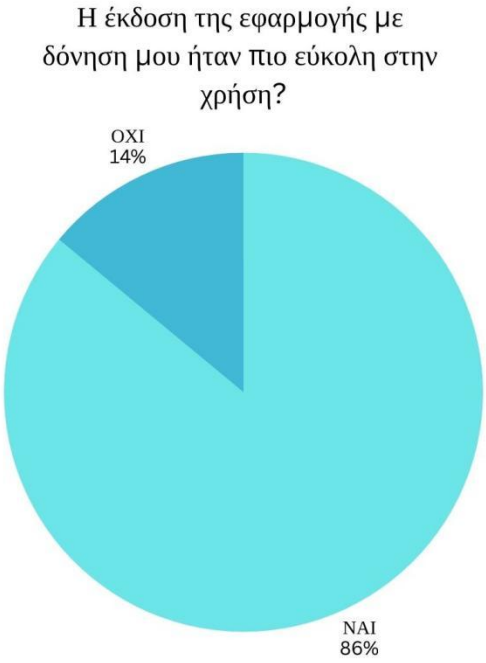
Διάγραμμα 6.3: Διάγραμμα με τις κατηγορίες ερωτήσεων της αξιολόγησης της εφαρμογής

6.2.1 Δοκιμές με και χωρίς δόνηση

Για την καλύτερη αξιολόγηση, οι χρήστες δοκίμασαν την εφαρμογή σε δύο διαφορετικές συνθήκες:

- Με δόνηση
- Χωρίς δόνηση

Η σύγκριση έδειξε ότι στην πλειοψηφία, η ύπαρξη δόνησης βελτίωσε την ταχύτητα εκτέλεσης των ενεργειών και μείωσε τα λάθη. Τα αποτελέσματα της έρευνας έδειξαν πως το 86% προτιμάει την εφαρμογή με δόνηση. Ωστόσο, υπήρξαν και κάποιοι που προτίμησαν τη χρήση μόνο με ήχο, κρίνοντας ότι η δόνηση μερικές φορές τους αποσπούσε την προσοχή ή τους μπέρδευε.



Διάγραμμα 6.4: Διάγραμμα το οποίο δείχνει το ποσοστό των ερωτηθέντων εάν η δόνηση κάνει την εφαρμογή πιο εύκολη στην χρήση

6.3 Επίλογος

Στο παρόν κεφάλαιο είδαμε την αξιολόγηση της εφαρμογής απο 20 άτομα και προκύπτει ότι η πλειοψηφία των χρηστών είχε θετική γνώμη και εμπειρία. Οι απαντήσεις τους δείχνουν ότι η εφαρμογή είναι σε μεγάλο βαθμό κατανοητή, ευχάριστη και υποστηρικτική. Επιπλέον, η σύγκριση της λειτουργίας με και χωρίς δόνηση κατέδειξε ότι η προσθήκη δόνησης βελτίωσε σημαντικά την πλοήγηση και την αίσθηση ασφάλειας των χρηστών.

Κεφάλαιο 7ο: Συμπεράσματα και Μελλοντικές Επεκτάσεις

7.1 Ανακεφαλαίωση

Στην εργασία αυτή παρουσιάστηκε η ανάπτυξη της android εφαρμογής που αξιοποιεί τεχνολογίες τεχνητής νοημοσύνης, με στόχο την βοήθεια ατόμων με προβλήματα όρασης. Αναλύθηκε η αρχιτεκτονική της εφαρμογής, η διεπαφή χρήστη, καθώς και ο τρόπος με τον οποίο συνδυάζονται η φωνητική περιγραφή και η απτική ανατροφοδότηση.

Έγιναν δοκιμές με χρήστες που τους ζητήθηκε να χρησιμοποιήσουν την εφαρμογή με δεμένα μάτια, και μέσα από αυτές τις δοκιμές αποτυπώθηκε η εμπειρία τους μέσα από ερωτηματολόγιο. Τα αποτελέσματα έδειξαν ότι η εφαρμογή είναι κατανοητή, φιλική και χρήσιμη, με την απτική ανατροφοδότηση να αυξάνει την ευκολία χρήσης.

7.2 Συμπεράσματα

Από τη διαδικασία αξιολόγησης προέκυψαν ορισμένα σημαντικά συμπεράσματα:

- Ευκολία χρήσης και κατανόηση:
  - Οι περισσότεροι χρήστες θεώρησαν την εφαρμογή απλή και εύκολη στη χρήση. Αυτό αποδεικνύει ότι ο σχεδιασμός της διεπαφής είναι κατανοητός και προσιτός.
- Σημαντικός ρόλος της δόνησης:
  - Η λειτουργία δόνησης βελτίωσε την εμπειρία χρήσης. Οι συμμετέχοντες ανέφεραν ότι με τη δόνηση μπορούσαν να αντιληφθούν πιο γρήγορα την παρουσία αντικειμένων.
- Φωνητική ανατροφοδότηση:
  - Η φωνητική περιγραφή αντικειμένων βελτίωσε την αίσθηση υποστήριξης. Το χαρακτηριστικό αυτό θεωρήθηκε ιδιαίτερα πολύτιμο, καθώς μετέτρεψε την κάμερα σε "εργαλείο όρασης".
- Συνολική εμπειρία:
  - Οι χρήστες αξιολόγησαν την εμπειρία ως ευχάριστη, κατανοητή και πρακτική.

Συνολικά, τα αποτελέσματα δείχνουν ότι η εφαρμογή μπορεί να προσφέρει ουσιαστική βοήθεια σε πραγματικές συνθήκες, αλλά απαιτούνται περαιτέρω μελέτες με άτομα που αντιμετωπίζουν όντως προβλήματα όρασης.

7.3 Ιδέες για Εμπλουτισμό της Εφαρμογής

Παρά τα θετικά αποτελέσματα, υπάρχουν αρκετές προοπτικές για μελλοντικές βελτιώσεις και εμπλουτισμό της εφαρμογής. Κάποιες προσωπικές ιδέες ειναι:

- Βελτίωση αναγνώρισης αντικειμένων μέσω πιο προηγμένων μοντέλων (π.χ. YOLO νεότερης γενιάς)
- Εμπλουτισμός του μοντέλου αναγνώρισης: Ενσωμάτωση περισσότερων κατηγοριών αντικειμένων, ειδικά αυτών που είναι κρίσιμα για την καθημερινότητα (π.χ. φαρμακευτικά προϊόντα, οικιακές συσκευές, πινακίδες).
- Συνδυασμός με GPS/χάρτες για πλοήγηση σε εξωτερικούς χώρους.
- Υποστήριξη σε πολλαπλές γλώσσες ώστε να είναι διαθέσιμη σε περισσότερους χρήστες.

Με αυτά τα βήματα, η εφαρμογή μπορεί να εξελιχθεί σε ένα πιο ολοκληρωμένο και καινοτόμο εργαλείο υποβοήθησης της καθημερινότητας των ατόμων με προβλήματα όρασης.

7.4 Επίλογος

Η εφαρμογή απέδειξε πως η τεχνητή νοημοσύνη μπορεί να είναι ενα πολύτιμο εργαλείο για την βοήθεια και την ποιότητας ζωής ατόμων με προβλήματα όρασης. Η θετική αποδοχή από τους συμμετέχοντες δείχνει τη χρησιμότητα και τη δυναμική της. Στο μέλλον κάποιες αναβαθμίσεις θα μπορούν να ενισχύσουν ακόμα περισσότερο την αποτελεσματικότητα και τη χρήση της εφαρμογής, κάνοντας μια πιο ολοκληρωμένη λύση υποστήριξης.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] World Health Organization, *World report on vision*, 2019.

[2] World Health Organization, “Eye care, vision impairment and blindness,” *WHO*, 2025.

[3] A. Pratap, S. Kumar, S. Chakravarty, “Adaptive Object Detection for Indoor Navigation Assistance: A Performance Evaluation of Real-Time Algorithms,” *arXiv*, 2025.

[4] J. Shukurov, “Improve accessibility for Low Vision and Blind people using Machine Learning and Computer Vision,” *arXiv*, 2024.

[5] J. Redmon, S. Divvala, R. Girshick, A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” *arXiv*, 2015.

[6] *IBM*, “What is computer vision?”, Think AI, 27 July 2021.

[7] Stanford University, *CS231n: Deep Learning for Computer Vision*, Course Webpage (2025).

[8] *IBM*, “What is a neural network?”, Think AI, 6 October 2021.

[9] T. T. Toche Tchio *et al.*, “A Comprehensive Review of Supervised Learning Algorithms for the Diagnosis of Photovoltaic Systems…,” *Applied Sciences*, vol. 14, no. 5, Art. 2072, 2024.

[10] L. T. Ramos, A. D. Sappa, “A Decade of You Only Look Once (YOLO) for Object Detection: A Review,” *arXiv preprint* arXiv:2504.18586, 2024.

[11] N. Surantha, N. Sutisna, “Key Considerations for Real-Time Object Recognition on Edge Computing Devices,” *Applied Sciences*, vol. 15, no. 13, Art. 7533, 2025.

[12] A. Kumar, P. Singh, et al., "Accessible Applications for People with Visual Disabilities through Voice Assistants: A Systematic Review," *ResearchGate*, 2023.

[13] M. Smith, E. Johnson, "Double Tap Gesture and Haptic Feedback in Assistive Technologies," *arXiv preprint arXiv:2410.20545*, 2024.

[14] Haptic Designing Assistive Technology for People who are Blind - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Terminology-of-haptics-Oakley-McGee-Brewster-Gray-2000_tbl2_280777071 [Online] Accessed May 2024

[15] J. Lee and H. Park, “Evaluation of Haptic Feedback for Enhancing Mobile Accessibility,” *IEEE Access*, vol. 9, pp. 122314–122325, 2021.

[16] P. Gupta et al., “Gesture-Based Interaction with Haptic Feedback for Visually Impaired Users,” *2022 IEEE International Conference on Human-Computer Interaction*, pp. 334–342, 2022.

[17] S. Kim et al., “Latency and Perception in Haptic Feedback for Mobile Devices,” *IEEE Transactions on Haptics*, vol. 14, no. 4, pp. 709–720, Oct.–Dec. 2021.

[18] Ultralytics, “YOLOv11: Fast & Accurate Vision AI,” Ultralytics, 2024. [Online]. Available: https://www.ultralytics.com/blog/all-you-need-to-know-about-ultralytics-yolo11-and-its-applications

[19] Ultralytics, “YOLOv11 Documentation,” 2024. [Online]. Available: https://docs.ultralytics.com/models/yolo11

[20] J. Redmon et al., “You Only Look Once: Unified, Real-Time Object Detection,” in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788.

[21] Limelight Vision, “Estimating Distance,” docs.limelightvision.io, 2025.

[22] Android Developers, “Position sensors,” Google, 2025.

[23] Android Developers, “CameraCharacteristics | camera2 API,” Google, 2025.

[24] World Wide Web Consortium (W3C), “Web Content Accessibility Guidelines (WCAG) 2.2,” W3C, 2023.

[25] Android Developers, “Text-to-Speech Overview,” Google, 2025.

[26] Artificial neural network – Scientific Figure on Wikimedia Commons. Available from: https://commons.wikimedia.org/wiki/File:Artificial_neural_network.svg

[27] Reinforcement learning diagram – Scientific Figure on Wikimedia Commons. Available from: https://commons.wikimedia.org/wiki/File:Reinforcement_learning_diagram.svg

[28] COCO: Common Objects in Context – Dataset Homepage. Available from: https://cocodataset.org/#home

[29] Speech recognition pipeline – Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/speech-recognition-pipeline_fig1_362987298

[30] Gestures Double Tap – Scientific Figure on Wikimedia Commons. Available from: https://commons.wikimedia.org/wiki/File:Gestures_Double_Tap.png

[31] Haptic Designing Assistive Technology for People who are Blind - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Terminology-of-haptics-Oakley-McGee-Brewster-Gray-2000_tbl2_280777071

[32] Howard, A.G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; Adam, H. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv* 2017, arXiv:1704.04861. [Google Scholar]

[33] Lin, T.Y.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; Zitnick, C.L. Microsoft coco: Common objects in context. In Proceedings of the Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, 6–12 September 2014; Springer International Publishing: New York, NY, USA, 2014.[Google Scholar]

[34] Jiang, C.; Kuang, E.; Fan, M. How Can Haptic Feedback Assist People with Blind and Low Vision (BLV): A Systematic Literature Review. *ACM Trans. Access. Comput.* 2025, *18*, 1–57. [Google Scholar] [CrossRef]

[35] Kuriakose, B.; Shrestha, R.; Sandnes, F.E. Multimodal navigation systems for users with visual impairments—A review and analysis. *Multimodal Technol. Interact.* 2020, *4*, 73. [Google Scholar] [CrossRef]

[36] Kuriakose, B.; Shrestha, R.; Sandnes, F.E. DeepNAVI: A deep learning based smartphone navigation assistant for people with visual impairments. *Expert Syst. Appl.* 2023, *212*, 118720. [Google Scholar] [CrossRef]

[37] Rakkolainen, I.; Farooq, A.; Kangas, J.; Hakulinen, J.; Rantala, J.; Turunen, M.; Raisamo, R. Technologies for multimodal interaction in extended reality—A scoping review. *Multimodal Technol. Interact.* 2021, *5*, 81. [Google Scholar] [CrossRef]

[38] Jiménez, M.F.; Mello, R.C.; Bastos, T.; Frizera, A. Assistive locomotion device with haptic feedback for guiding visually impaired people. *Med. Eng. Phys.* 2020, *80*, 18–25. [Google Scholar] [CrossRef]

[39] See, A.R.; Costillas, L.V.M.; Advincula, W.D.C.; Bugtai, N.T. Haptic Feedback to Detect Obstacles in Multiple Regions for Visually Impaired and Blind People. *Sens. Mater.* 2021, *33*, 1799–1808. [Google Scholar] [CrossRef]

[40] Palani, H.P.; Fink, P.D.; Giudice, N.A. Comparing map learning between touchscreen-based visual and haptic displays: A behavioral evaluation with blind and sighted users. *Multimodal Technol. Interact.* 2021, *6*, 1. [Google Scholar] [CrossRef]

[41] Raynal, M.; Ducasse, J.; Macé, M.J.M.; Oriola, B.; Jouffrais, C. The FlexiBoard: Tangible and Tactile Graphics for People with Vision Impairments. *Multimodal Technol. Interact.* 2024, *8*, 17. [Google Scholar] [CrossRef]

[42] Tzimos, N.; Voutsakelis, G.; Kontogiannis, S.; Kokkonis, G. Evaluation of Haptic Textures for Tangible Interfaces for the Tactile Internet. *Electronics* 2024, *13*, 3775. [Google Scholar] [CrossRef]

[43] Billi, M.; Burzagli, L.; Catarci, T.; Santucci, G.; Bertini, E.; Gabbanini, F.; Palchetti, E. A unified methodology for the evaluation of accessibility and usability of mobile applications. *Univers. Access Inf. Soc.* 2010, *9*, 337–356. [Google Scholar] [CrossRef]

[44] Ballantyne, M.; Jha, A.; Jacobsen, A.; Hawker, J.S.; El-Glaly, Y.N. Study of accessibility guidelines of mobile applications. In Proceedings of the 17th International Conference on Mobile and Ubiquitous Multimedia, Cairo, Egypt, 25–28 November 2018; Association for Computing Machinery: New York, NY, USA, 2018. [Google Scholar]

[45] Park, E.; Han, S.; Bae, H.; Kim, R.; Lee, S.; Lim, D.; Lim, H. Development of automatic evaluation tool for mobile accessibility for android application. In Proceedings of the IEEE International Conference on Systems of Collaboration Big Data, Internet of Things Security (SysCoBIoTS), Casablanca, Morocco, 12–13 December 2019; IEEE: New York, NY, USA, 2019. [Google Scholar]

[46] Acosta-Vargas, P.; Salvador-Ullauri, L.; Jadán-Guerrero, J.; Guevara, C.; Sanchez-Gordon, S.; Calle-Jimenez, T.; Lara-Alvarez, P.; Medina, A. Accessibility assessment in mobile applications for android. In *Advances in Human Factors and Systems Interaction*; Springer: Cham, Switzerland, 2020; pp. 279–288. [Google Scholar]

[47] Mateus, D.A.; Silva, C.A.; Eler, M.M.; Freire, A.P. Accessibility of mobile applications: Evaluation by users with visual impairment and by automated tools. In Proceedings of the 19th Brazilian Symposium on Human Factors in Computing Systems, Diamantina, Brazil, 26–30 October 2020; Association for Computing Machinery: New York, NY, USA, 2020. [Google Scholar]

[48] Acosta-Vargas, P.; Salvador-Acosta, B.; Salvador-Ullauri, L.; Villegas-Ch, W.; Gonzalez, M. Accessibility in native mobile applications for users with disabilities: A scoping review. *Appl. Sci.* 2021, *11*, 5707. [Google Scholar] [CrossRef]

[49] Zheng, Q.; Tian, X.; Yu, Z.; Elhanashi, Y.M.A.; Saponara, S. Robust automatic modulation classification using asymmetric trilinear attention net with noisy activation function. *Eng. Appl. Artif. Intell.* 2025, *141*, 1098. [Google Scholar] [CrossRef]

ΠΑΡΑΡΤΗΜΑ Α : ΤΙΤΛΟΣ ΠΑΡΑΡΤΗΜΑΤΟΣ

Τα παραρτήματα μπορούν να είναι περισσότερα από ένα. Αριθμούνται με γράμματα του Ελληνικού αλφάβητου (Α, Β, Γ, …).

Στα παραρτήματα παρουσιάζονται πληροφορίες που δεν είναι κρίσιμες για την εργασία, αλλά σημαντικές για την απόδειξη συμπερασμάτων που αναπτύχθηκαν στην εργασία. Περιέχουν κώδικα λογισμικού, ερωτηματολόγια και απαντήσεις σε ερωτηματολόγια, κτλ.