ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Ανάπτυξη Εφαρμογής Αναγνώρισης Ζημιών σε Αυτοκίνητα με Χρήση Deep Learning

ΑΝΔΡΕΑΣ ΙΩΑΝΝΟΥ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΠΕΥΘΥΝΟΣ

ΚΟΛΟΜΒΑΤΣΟΣ ΚΩΝΣΤΑΝΤΙΝΟΣ

Λαμία Μάιος έτος 2025

ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Ανάπτυξη Εφαρμογής Αναγνώρισης Ζημιών σε Αυτοκίνητα με Χρήση Deep Learning

ΑΝΔΡΕΑΣ ΙΩΑΝΝΟΥ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΠΕΥΘΥΝΟΣ

ΚΟΛΟΜΒΑΤΣΟΣ ΚΩΝΣΤΑΝΤΙΝΟΣ

Λαμία Μάιος έτος 2025

UNIVERSITY OF
THESSALY

SCHOOL OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE & TELECOMMUNICATIONS

Development of a Car Damage Recognition Application Using Deep Learning

ANDREAS IOANNOU

FINAL THESIS

ADVISOR

KOLOMVATSOS KOSTANTINOS

Lamia May year 2025

«Με ατομική μου ευθύνη και γνωρίζοντας τις κυρώσεις [(1)], που προβλέπονται από της διατάξεις της παρ. 6 του άρθρου 22 του Ν. 1599/1986, δηλώνω ότι:

*1. Δεν παραθέτω κομμάτια βιβλίων ή άρθρων ή εργασιών άλλων αυτολεξεί **χωρίς να τα περικλείω σε εισαγωγικά** και χωρίς να αναφέρω το συγγραφέα, τη χρονολογία, τη σελίδα. Η αυτολεξεί παράθεση χωρίς εισαγωγικά χωρίς αναφορά στην πηγή, είναι λογοκλοπή. Πέραν της αυτολεξεί παράθεσης, λογοκλοπή θεωρείται και η παράφραση εδαφίων από έργα άλλων, συμπεριλαμβανομένων και έργων συμφοιτητών μου, καθώς και η παράθεση στοιχείων που άλλοι συνέλεξαν ή επεξεργάσθηκαν, χωρίς αναφορά στην πηγή. Αναφέρω πάντοτε με πληρότητα την πηγή κάτω από τον πίνακα ή σχέδιο, όπως στα παραθέματα.*

*2. Δέχομαι ότι η αυτολεξεί **παράθεση χωρίς εισαγωγικά**, ακόμα κι αν συνοδεύεται από αναφορά στην πηγή σε κάποιο άλλο σημείο του κειμένου ή στο τέλος του, είναι αντιγραφή. Η αναφορά στην πηγή στο τέλος π.χ. μιας παραγράφου ή μιας σελίδας, δεν δικαιολογεί συρραφή εδαφίων έργου άλλου συγγραφέα, έστω και παραφρασμένων, και παρουσίασή τους ως δική μου εργασία.*

3. Δέχομαι ότι υπάρχει επίσης περιορισμός στο μέγεθος και στη συχνότητα των παραθεμάτων που μπορώ να εντάξω στην εργασία μου εντός εισαγωγικών. Κάθε μεγάλο παράθεμα (π.χ. σε πίνακα ή πλαίσιο, κλπ), προϋποθέτει ειδικές ρυθμίσεις, και όταν δημοσιεύεται προϋποθέτει την άδεια του συγγραφέα ή του εκδότη. Το ίδιο και οι πίνακες και τα σχέδια

4. Δέχομαι όλες τις συνέπειες σε περίπτωση λογοκλοπής ή αντιγραφής.

Ημερομηνία: ……/…../20……

Ο – Η Δηλ.

(1) «Όποιος εν γνώσει του δηλώνει ψευδή γεγονότα ή αρνείται ή αποκρύπτει τα αληθινά με έγγραφη υπεύθυνη δήλωση
του άρθρου 8 παρ. 4 Ν. 1599/1986 τιμωρείται με φυλάκιση τουλάχιστον τριών μηνών. Εάν ο υπαίτιος αυτών των πράξεων
σκόπευε να προσπορίσει στον εαυτόν του ή σε άλλον περιουσιακό όφελος βλάπτοντας τρίτον ή σκόπευε να βλάψει άλλον, τιμωρείται με κάθειρξη μέχρι 10 ετών.»

ΠΕΡΙΛΗΨΗ

Η ραγδαία εξέλιξη της τεχνητής νοημοσύνης και των μεθόδων βαθιάς μάθησης (Deep Learning) έχει επιφέρει σημαντικές καινοτομίες στον τομέα της αυτόματης επεξεργασίας εικόνας και βίντεο. Στην παρούσα πτυχιακή εργασία αναπτύχθηκε ένα ολοκληρωμένο σύστημα για την ανίχνευση και αναγνώριση ζημιών σε αυτοκίνητα από φωτογραφίες και βίντεο, αξιοποιώντας προηγμένα μοντέλα βαθιάς μάθησης και σύγχρονες τεχνολογίες ανάπτυξης εφαρμογών.

Για την ανάλυση των εικόνων και των βίντεο χρησιμοποιήθηκε το μοντέλο YOLOv11 Segmentation, το οποίο επιτρέπει την ακριβή ανίχνευση και ταξινόμηση τόσο των επιμέρους μερών ενός αυτοκινήτου όσο και των πιθανών ζημιών σε αυτά. Η εκπαίδευση των μοντέλων πραγματοποιήθηκε με χρήση της τεχνολογίας CUDA 12.6, εξασφαλίζοντας ταχύτητα και αποδοτικότητα κατά την εκπαίδευση και αξιολόγηση.

Αναλυτικότερα, αναπτύχθηκε μια mobile εφαρμογή με React Native (Expo), μέσω της οποίας ο χρήστης μπορεί να πραγματοποιήσει ανάλυση φωτογραφιών και βίντεο για την αναγνώριση ζημιών. Η εφαρμογή επικοινωνεί με έναν FastAPI server ο οποίος εκτελεί την επεξεργασία των εικόνων σε πραγματικό χρόνο, παρέχοντας άμεσα αποτελέσματα.

Η εφαρμογή προσφέρει τρεις βασικές λειτουργίες:

1. Αναγνώριση Μερών Αυτοκινήτου
2. Αναγνώριση Ζημιών
3. Πλήρης Σάρωση, με ταυτόχρονο εντοπισμό ζημιών και αντιστοίχηση στο αντίστοιχο μέρος του οχήματος.

Τα αποτελέσματα της πειραματικής αξιολόγησης καθιστούν την εφαρμογή κατάλληλη για χρήση σε πραγματικά σενάρια όπως ασφαλιστικές αποτιμήσεις ή αυτόματη επιθεώρηση στόλων οχημάτων. Η εργασία αυτή αποδεικνύει την αποτελεσματικότητα των τεχνικών βαθιάς μάθησης στην επεξεργασία εικόνας και ανοίγει τον δρόμο για μελλοντικές επεκτάσεις, όπως η αναγνώριση ζημιών σε πραγματικό χρόνο από κάμερες ασφαλείας ή η ενσωμάτωσή της σε επαγγελματικά εργαλεία αξιολόγησης αυτοκινήτων.

ABSTRACT

The rapid development of artificial intelligence and deep learning methods has brought significant innovations in the field of automatic image and video processing. In this thesis, a comprehensive system was developed for detecting and recognizing car damage from photographs and videos, utilizing advanced deep learning models and modern application development technologies.

For the analysis of images and videos, the YOLOv11 Segmentation model was used, which allows accurate detection and classification of both individual car parts and potential damage to them. The training of the models was conducted using CUDA 12.6 technology, ensuring speed and efficiency during training and evaluation.

The mobile application was developed with React Native (Expo), through which users can analyze photos and videos to identify damage. The application communicates with a FastAPI server, which processes the images in real time, providing immediate results.

The application offers three main functions:

1. Recognition of Car Parts
2. Damage Recognition
3. Full Scan, with simultaneous detection of damage and matching to the corresponding part of the vehicle.

The results of the experimental evaluation make the application suitable for real-world scenarios such as insurance assessments or automatic vehicle fleet inspections.

This project demonstrates the effectiveness of deep learning techniques in image processing and paves the way for future expansions, such as real-time damage detection from security cameras or integration into professional car assessment tools.

Table of Contents

ΚΕΦΑΛΑΙΟ 1 Εισαγωγή

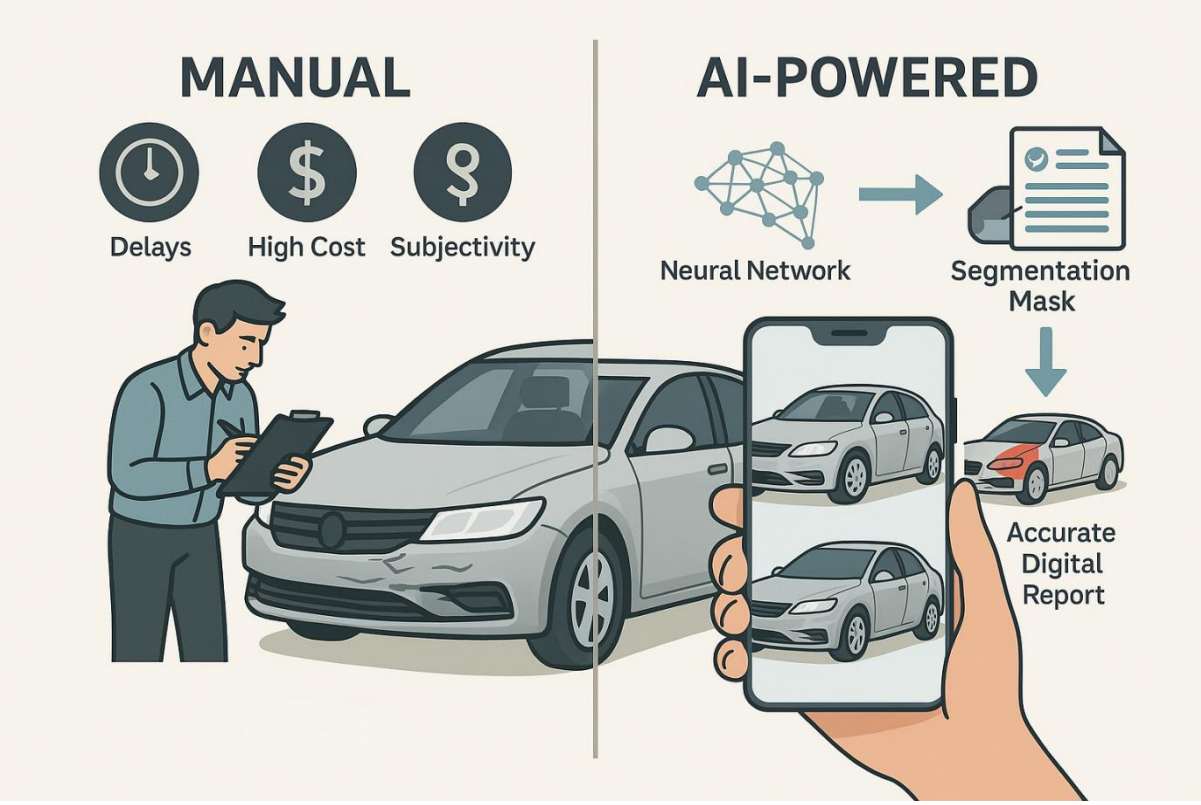
Αντικείμενο και Σημασία της Έρευνας 1.1

Η αυτόματη ανίχνευση και αναγνώριση ζημιών σε αυτοκίνητα αποτελεί ένα κρίσιμο ζήτημα με άμεση εφαρμογή στον τομέα της αυτοκινητοβιομηχανίας, των ασφαλιστικών εταιρειών και της διαχείρισης στόλου οχημάτων. Η ανάγκη για γρήγορη, αξιόπιστη και αυτοματοποιημένη εκτίμηση των ζημιών συμβάλλει σημαντικά στη μείωση του κόστους και του χρόνου εκτίμησης, βελτιώνοντας συνολικά την ποιότητα των παρεχόμενων υπηρεσιών.

Η αυξανόμενη χρήση αυτοκινήτων και η συνεπακόλουθη αύξηση των ατυχημάτων καθιστούν αναγκαία τη δημιουργία αποδοτικότερων και πιο αξιόπιστων διαδικασιών για την εκτίμηση και την αποκατάσταση ζημιών. Επιπλέον, η αυτοματοποίηση της διαδικασίας έχει τη δυνατότητα να μειώσει τις πιθανές διαφωνίες μεταξύ πελατών και ασφαλιστικών εταιρειών, προσφέροντας διαφάνεια και αντικειμενικότητα.

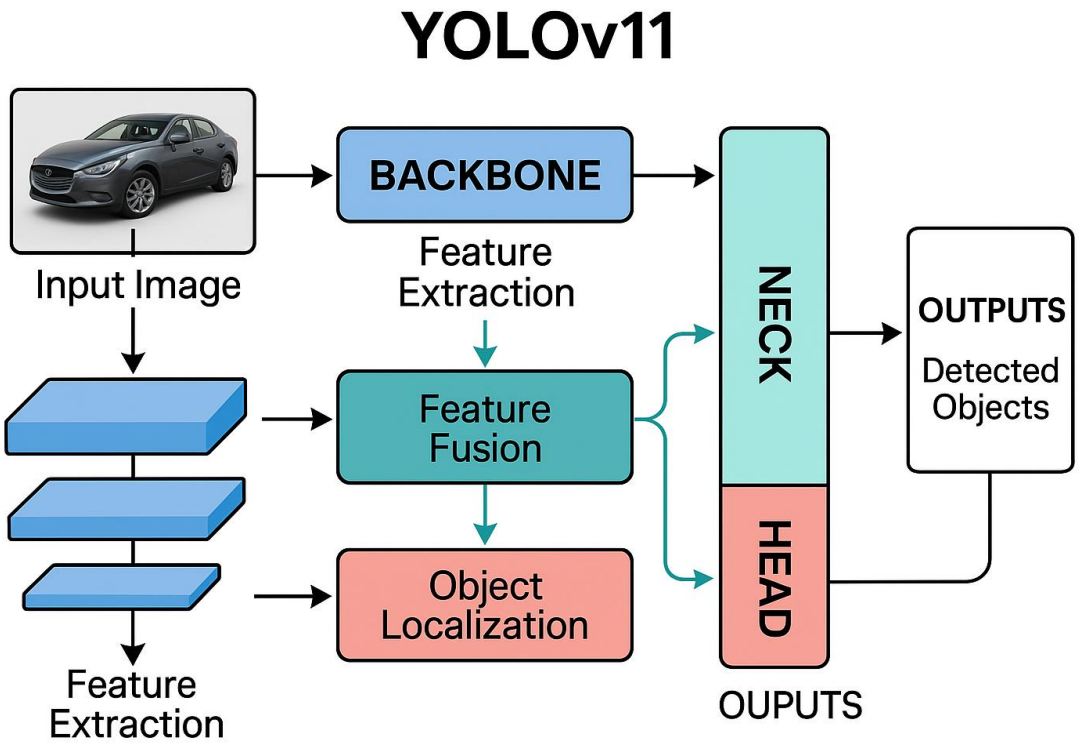
Παραδοσιακές Προσεγγίσεις και Περιορισμοί 1.2

Παραδοσιακά, η διαδικασία εκτίμησης ζημιών σε οχήματα απαιτεί την ανθρώπινη παρέμβαση ειδικών, γεγονός που οδηγεί συχνά σε καθυστερήσεις, υψηλό κόστος και πιθανές ανακρίβειες λόγω της ανθρώπινης υποκειμενικότητας. Οι εκτιμήσεις αυτές συχνά βασίζονται σε εμπειρική γνώση και σε χειροκίνητες διαδικασίες καταγραφής ζημιών, που είναι επιρρεπείς σε λάθη και διαφοροποιήσεις ανάμεσα σε διαφορετικούς εκτιμητές.
Αυτοί οι περιορισμοί αναδεικνύουν την ανάγκη για ανάπτυξη προηγμένων, αυτοματοποιημένων τεχνολογιών που μπορούν να παρέχουν σταθερά ακριβή αποτελέσματα με μεγαλύτερη ταχύτητα και συνέπεια.



Εφαρμογή της Βαθιάς Μηχανικής Μάθησης 1.3

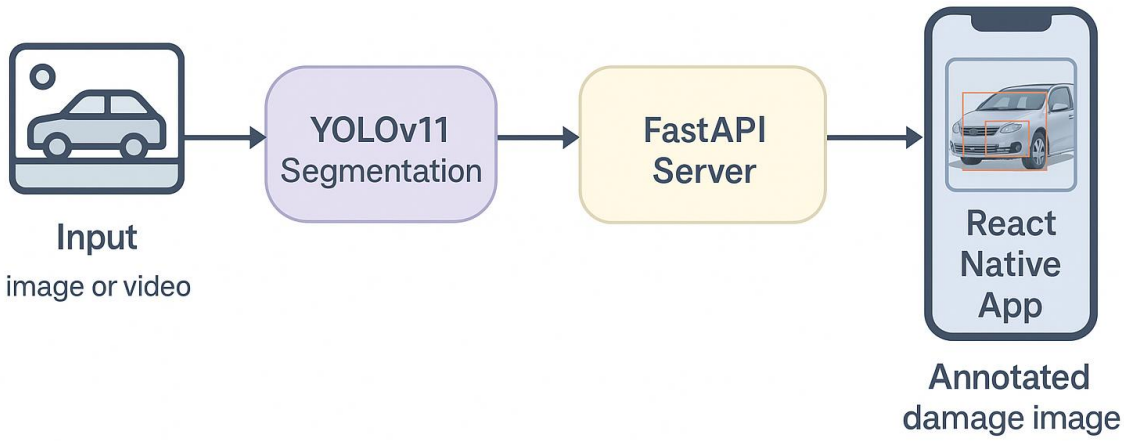
Η εφαρμογή τεχνολογιών βαθιάς μηχανικής μάθησης (Deep Learning) έχει τη δυνατότητα να αντιμετωπίσει αποτελεσματικά αυτές τις προκλήσεις, επιτυγχάνοντας υψηλή ακρίβεια και αντικειμενικότητα στη διαδικασία αναγνώρισης ζημιών. Η βαθιά μάθηση έχει αποδειχθεί ιδιαίτερα επιτυχημένη στην αναγνώριση και ταξινόμηση αντικειμένων από εικόνες και βίντεο, προσφέροντας σημαντικά πλεονεκτήματα σε σχέση με τις συμβατικές μεθόδους. Στην παρούσα εργασία, χρησιμοποιούμε το μοντέλο YOLOv11 segmentation, το οποίο είναι γνωστό για την ταχύτητα και την υψηλή ακρίβειά του στην αναγνώριση αντικειμένων. Επιπλέον, για τη βελτίωση της απόδοσης και τη μείωση του χρόνου εκπαίδευσης των μοντέλων, αξιοποιούμε την πλατφόρμα CUDA 12.6 για επιτάχυνση των υπολογισμών μέσω GPU.



Στόχοι της Πτυχιακής Εργασίας 1.4

Οι βασικοί στόχοι αυτής της πτυχιακής εργασίας είναι:

- Η ανάπτυξη και αξιολόγηση μοντέλων βαθιάς μηχανικής μάθησης για εντοπισμό και αναγνώριση ζημιών καθώς και συγκεκριμένων μερών στα αυτοκίνητα.
- Η δημιουργία μιας εύχρηστης και αποτελεσματικής εφαρμογής κινητών συσκευών που θα αξιοποιεί τα αναπτυχθέντα μοντέλα, παρέχοντας άμεσα αποτελέσματα στον χρήστη.
- Η αναλυτική αξιολόγηση της απόδοσης των μοντέλων μέσα από πραγματικά σενάρια χρήσης, ώστε να επιτευχθεί η μέγιστη δυνατή αξιοπιστία.

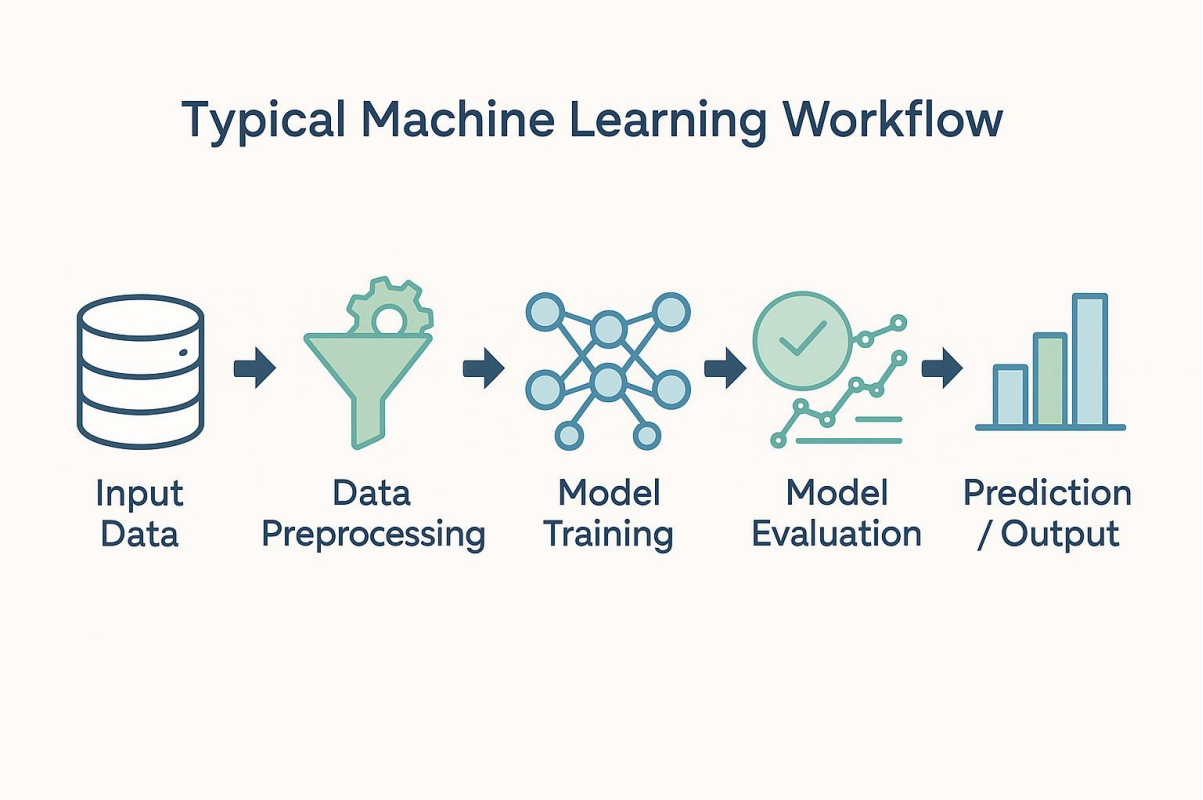


ΚΕΦΑΛΑΙΟ 2: Μοντέλα Βαθιάς Μηχανικής Μάθησης

(Γενική εισαγωγή στα Deep Learning μοντέλα 2.1)

Η Μηχανική Μάθηση (Machine Learning) αποτελεί έναν από τους ταχύτερα αναπτυσσόμενους τομείς της Τεχνητής Νοημοσύνης. Πρόκειται για την επιστήμη και τεχνολογία που επιτρέπει στους υπολογιστές να "μαθαίνουν" από δεδομένα και εμπειρίες χωρίς να έχουν ρητά προγραμματιστεί. Η βαθιά Μηχανική Μάθηση (Deep Learning) αποτελεί υποπεδίο της Μηχανικής Μάθησης και βασίζεται στη χρήση βαθιών νευρωνικών δικτύων, που προσομοιώνουν τον τρόπο με τον οποίο λειτουργεί ο ανθρώπινος εγκέφαλος.

Τι είναι η Μηχανική Μάθηση και Πώς Λειτουργεί;



Η Μηχανική Μάθηση αφορά την ανάπτυξη αλγορίθμων που έχουν τη δυνατότητα να αναγνωρίζουν μοτίβα σε δεδομένα και να βελτιώνονται με την εμπειρία. Η βασική αρχή είναι ότι τα συστήματα μπορούν να

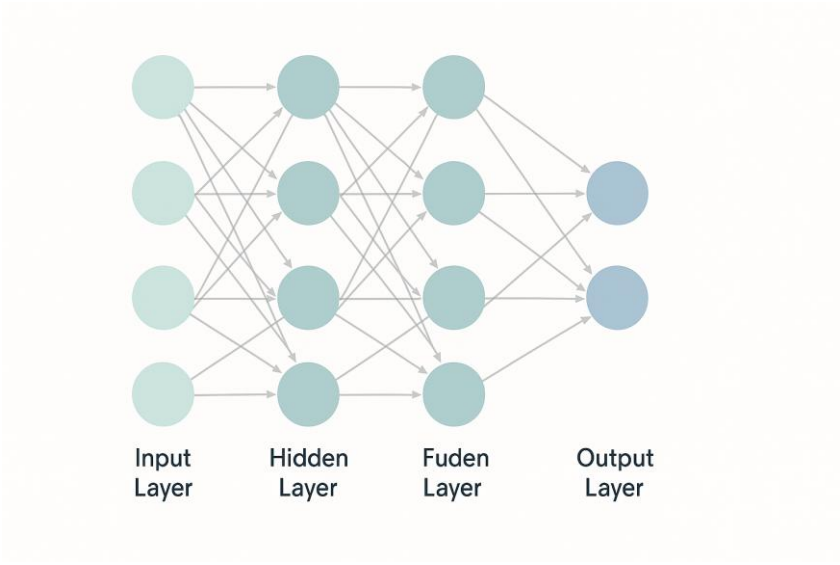
μάθουν από παρατηρήσεις, να εντοπίζουν πρότυπα και να λαμβάνουν αποφάσεις με ελάχιστη ανθρώπινη παρέμβαση.

Κυριότερα Είδη Μηχανικής Μάθησης:

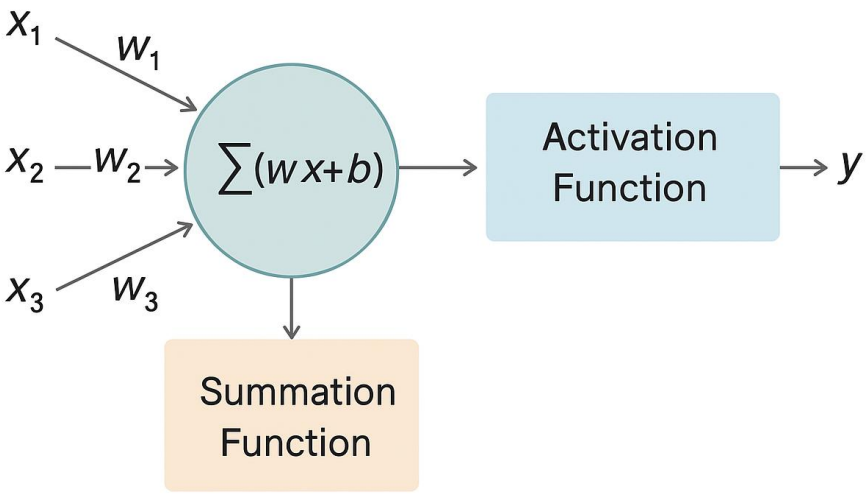
1. Supervised Learning (Επιβλεπόμενη Μάθηση): Σε αυτό το είδος, τα δεδομένα εκπαίδευσης είναι επισημασμένα (labelled), δηλαδή κάθε δείγμα συνοδεύεται από τη σωστή απάντηση. Ο αλγόριθμος μαθαίνει να προβλέπει την έξοδο βάσει αυτών των δεδομένων. Παραδείγματα περιλαμβάνουν την ταξινόμηση (classification) και την παλινδρόμηση (regression).
2. Unsupervised Learning (Μη Επιβλεπόμενη Μάθηση): Σε αυτό το είδος, τα δεδομένα δεν έχουν ετικέτες. Ο στόχος είναι η αναγνώριση κρυφών δομών ή σχέσεων μέσα στα δεδομένα. Κλασικά παραδείγματα περιλαμβάνουν τον ομαδοποίηση (clustering) και τη μείωση διαστάσεων (dimensionality reduction).
3. Semi-Supervised Learning (Ημι-Επιβλεπόμενη Μάθηση): Συνδυασμός επιβλεπόμενης και μη επιβλεπόμενης μάθησης. Ένα μικρό ποσοστό των δεδομένων είναι επισημασμένο και το υπόλοιπο όχι. Είναι ιδιαίτερα χρήσιμο όταν η επισήμανση είναι κοστοβόρα ή χρονοβόρα.
4. Reinforcement Learning (Ενισχυτική Μάθηση): Ο αλγόριθμος μαθαίνει μέσα από αλληλεπίδραση με το περιβάλλον, επιβραβευόμενος για σωστές ενέργειες και τιμωρούμενος για λανθασμένες. Εφαρμόζεται σε ρομποτική, παιχνίδια και αυτόνομα οχήματα.
5. Self-Supervised Learning: Μία νέα προσέγγιση όπου το ίδιο το σύστημα δημιουργεί τις ετικέτες με βάση τις ιδιότητες των δεδομένων. Παρέχει αποτελεσματική εκπαίδευση με λιγότερη ανθρώπινη παρέμβαση, κυρίως σε NLP και Vision.

Νευρωνικά Δίκτυα2.2

Τα Νευρωνικά Δίκτυα (Neural Networks) είναι μοντέλα υπολογιστικής μάθησης εμπνευσμένα από τη δομή του ανθρώπινου εγκεφάλου. Αποτελούνται από κόμβους (νευρώνες), οργανωμένους σε στρώματα (layers): το εισόδου (input layer), ένα ή περισσότερα κρυφά (hidden layers), και το εξόδου (output layer).



Πώς λειτουργούν τα Νευρωνικά Δίκτυα;



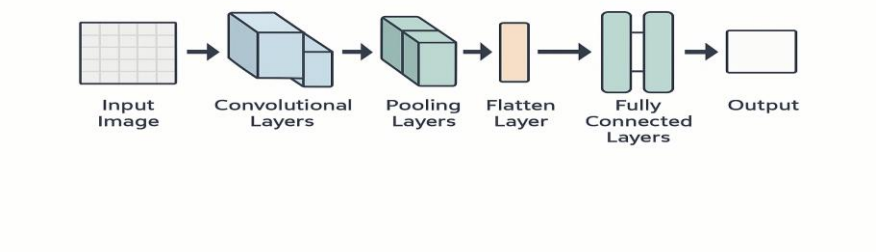
Κάθε νευρώνας δέχεται είσοδο από άλλους νευρώνες ή από τα δεδομένα εισόδου, πραγματοποιεί έναν υπολογισμό (συνήθως γραμμικό) και εφαρμόζει μια μη γραμμική συνάρτηση ενεργοποίησης (activation function). Το αποτέλεσμα προωθείται στους επόμενους νευρώνες. Η διαδικασία αυτή συνεχίζεται μέχρι να παραχθεί η έξοδος.

Η εκπαίδευση ενός νευρωνικού δικτύου γίνεται μέσω της διαδικασίας οπίσθιας διάδοσης σφάλματος (backpropagation) και βελτιστοποίησης (optimization), με σκοπό την ελαχιστοποίηση του σφάλματος πρόβλεψης. Αυτό επιτυγχάνεται με τεχνικές όπως ο αλγόριθμος gradient descent.

Η χρήση νευρωνικών δικτύων έχει καταστήσει δυνατή τη δημιουργία μοντέλων που επιτυγχάνουν εντυπωσιακά αποτελέσματα σε πλήθος εφαρμογών, όπως η αναγνώριση εικόνας, η επεξεργασία φυσικής γλώσσας, και η πρόβλεψη σειρών χρόνου. Το βάθος (δηλαδή ο αριθμός των κρυφών στρωμάτων) στα νευρωνικά δίκτυα οδήγησε στη γένεση του όρου «Βαθιά Μάθηση», που επιτρέπει την εκμάθηση πολύπλοκων, αφηρημένων αναπαραστάσεων από τα δεδομένα.

Convolutional Neural Networks (CNN) 2.3

Τα Συνελικτικά Νευρωνικά Δίκτυα (Convolutional Neural Networks - CNN) αποτελούν έναν από τους πιο διαδεδομένους τύπους βαθιών νευρωνικών δικτύων και είναι ιδιαίτερα αποτελεσματικά στην επεξεργασία δεδομένων εικόνας, ήχου και βίντεο. Έχουν καθιερωθεί ως η κορυφαία τεχνολογία στην αναγνώριση αντικειμένων, την κατηγοριοποίηση εικόνων και τη μηχανική όραση γενικότερα.



Τα CNN είναι εμπνευσμένα από τη δομή του οπτικού φλοιού του ανθρώπινου εγκεφάλου. Χαρακτηρίζονται από την παρουσία ειδικών στρωμάτων που ονομάζονται συνελικτικά (convolutional layers), τα οποία εφαρμόζουν φίλτρα (kernels) στην είσοδο, με στόχο την εξαγωγή χαρακτηριστικών.

Βασικά Συστατικά των CNN:

1. **Συνελικτικά Στρώματα (Convolutional Layers):** Αυτά τα στρώματα εφαρμόζουν φίλτρα πάνω στα δεδομένα εισόδου για την εξαγωγή τοπικών χαρακτηριστικών, όπως ακμές, γωνίες ή υφές. Κάθε φίλτρο σαρώνει την είσοδο και δημιουργεί έναν χάρτη χαρακτηριστικών (feature map).
2. **Στρώματα Συγκέντρωσης (Pooling Layers):** Τα pooling layers μειώνουν τις διαστάσεις των feature maps και διατηρούν τις σημαντικότερες πληροφορίες. Συνήθως χρησιμοποιείται το max pooling, που επιλέγει τη μέγιστη τιμή μέσα σε ένα υποσύνολο του χάρτη χαρακτηριστικών.
3. **Μη-Γραμμικές Συναρτήσεις Ενεργοποίησης (ReLU):** Η συνάρτηση ReLU (Rectified Linear Unit) εφαρμόζεται μετά από κάθε συνελικτικό στρώμα για την εισαγωγή μη γραμμικότητας στο δίκτυο.
4. **Πλήρως Συνδεδεμένα Στρώματα (Fully Connected Layers):** Στο τέλος του δικτύου, τα χαρακτηριστικά που έχουν εξαχθεί συνδέονται με πλήρως συνδεδεμένα στρώματα, όπως σε ένα παραδοσιακό νευρωνικό δίκτυο, για την τελική πρόβλεψη ή ταξινόμηση.
5. **Dropout και Κανονικοποίηση (Regularization Techniques):** Τεχνικές όπως το Dropout (τυχαία απενεργοποίηση νευρώνων κατά την εκπαίδευση) χρησιμοποιούνται για την αποφυγή υπερεκπαίδευσης και τη βελτίωση της γενίκευσης.

Πλεονεκτήματα των CNN:

- Αυτόματη εξαγωγή χαρακτηριστικών από εικόνες χωρίς την ανάγκη χειροκίνητης προεπεξεργασίας.
- Ικανότητα αναγνώρισης χωρικών ιεραρχιών (από τοπικά σε παγκόσμια χαρακτηριστικά).
- Επαναχρησιμοποίηση φίλτρων (weights sharing), που μειώνει τις παραμέτρους και αυξάνει την αποδοτικότητα.

Τα CNN είναι ιδιαίτερα αποτελεσματικά λόγω της ικανότητάς τους να μαθαίνουν και να αναγνωρίζουν περίπλοκα οπτικά μοτίβα. Η επιτυχία τους βασίζεται κυρίως στη δυνατότητα ανάλυσης χωρικών χαρακτηριστικών των εικόνων και στην ανθεκτικότητά τους απέναντι στις μετατοπίσεις ή τις παραμορφώσεις των εικόνων εισόδου.

(Μοντέλα Object Detection (YOLO, SSD, Faster R-CNN) 2.4)

Η ανίχνευση αντικειμένων (Object Detection) είναι ένα από τα πιο κρίσιμα και πολυχρησιμοποιημένα προβλήματα της Τεχνητής Όρασης. Περιλαμβάνει όχι μόνο την ταξινόμηση αντικειμένων μέσα σε μία εικόνα, αλλά και τον εντοπισμό της ακριβούς θέσης τους μέσω συντεταγμένων πλαισίων (bounding boxes).

Τα μοντέλα ανίχνευσης αντικειμένων συνδυάζουν δύο βασικά καθήκοντα:

1. **Classification (Ταξινόμηση):** Τι είναι το αντικείμενο;
2. **Localization (Εντοπισμός):** Πού βρίσκεται το αντικείμενο;

Για την υλοποίηση αυτών των εργασιών έχουν αναπτυχθεί πολυάριθμα μοντέλα, με τα πιο δημοφιλή να είναι τα εξής:

- YOLO (You Only Look Once)
- SSD (Single Shot MultiBox Detector)
- Faster R-CNN (Region-based Convolutional Neural Networks)

(YOLO (You Only Look Once): 2.4.1)

Το YOLO αποτελεί μια οικογένεια αλγορίθμων που φέρνει επανάσταση στην ανίχνευση αντικειμένων με τη φιλοσοφία του ενιαίου βήματος: το μοντέλο βλέπει ολόκληρη την εικόνα «με μία ματιά». Σε αντίθεση με παλαιότερα μοντέλα που διαχωρίζουν την ταξινόμηση από τον εντοπισμό, το YOLO τα ενσωματώνει σε ένα και μόνο πέρασμα του νευρωνικού δικτύου.

Κύρια χαρακτηριστικά YOLO:

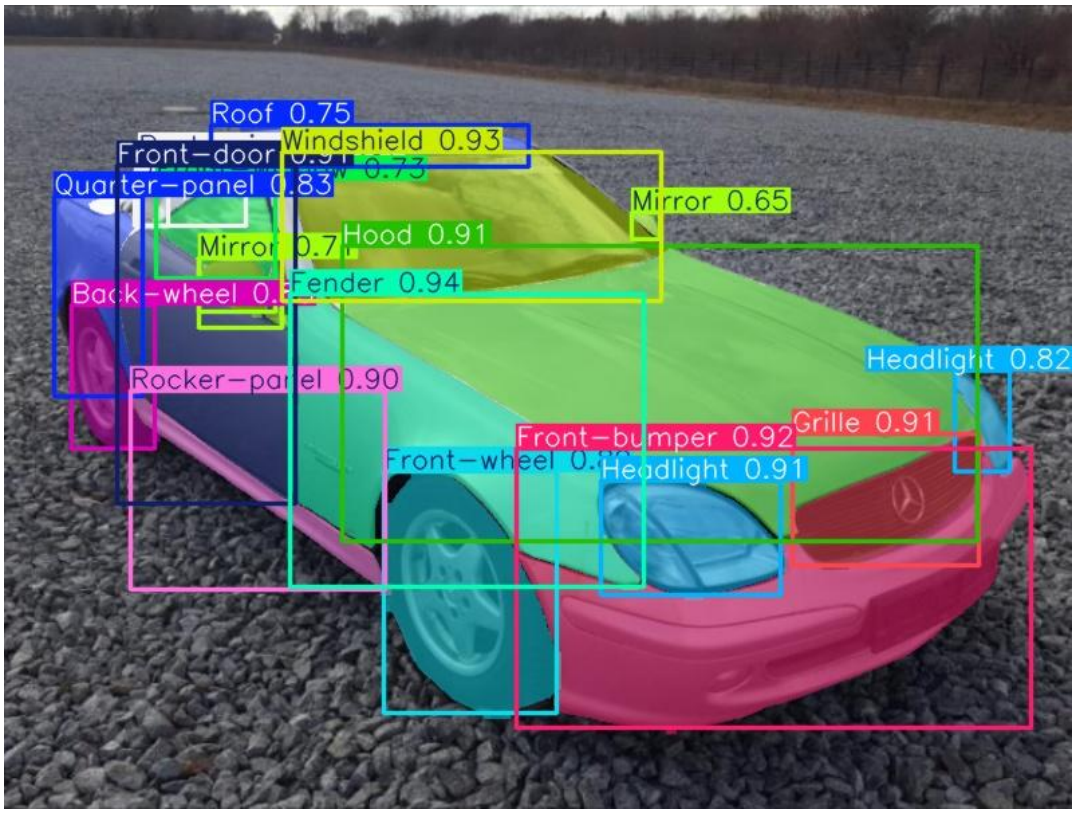
- Διαιρεί την εικόνα σε πλέγμα (grid) και προβλέπει bounding boxes και κατηγορίες για κάθε κελί.
- Πραγματοποιεί και τα δύο βήματα (εντοπισμός + ταξινόμηση) ταυτόχρονα.
- Βασίζεται σε ένα CNN για εξαγωγή χαρακτηριστικών και εξόδους πρόβλεψης.

Πλεονεκτήματα:

- Εξαιρετικά γρήγορο — κατάλληλο για εφαρμογές σε πραγματικό χρόνο.
- Μαθαίνει καθολικά χαρακτηριστικά λόγω της παγκόσμιας εικόνας που επεξεργάζεται.

Εκδόσεις YOLO: Το αρχικό YOLO έχει εξελιχθεί σε πολλές εκδόσεις (YOLOv1 – YOLOv11), με συνεχείς βελτιώσεις σε ταχύτητα και ακρίβεια.

YOLO με Segmentation (YOLOv11 Segmentation)



Η έκδοση YOLOv11 ενσωματώνει ένα επιπλέον βήμα στην ανίχνευση αντικειμένων: την **κατάτμηση αντικειμένων (segmentation)**. Σε αντίθεση με την παραδοσιακή ανίχνευση μέσω bounding boxes, το segmentation στοχεύει στην εξαγωγή των ακριβών ορίων κάθε αντικειμένου, αποδίδοντας ένα «μάσκα» σε κάθε ανιχνευμένο στοιχείο.

Τύποι segmentation:

- **Semantic segmentation:** αναγνωρίζει το είδος των pixel αλλά όχι την ταυτότητα του αντικειμένου
- **Instance segmentation:** αναγνωρίζει ξεχωριστές παρουσίες του ίδιου αντικειμένου (π.χ. 3 άτομα)

Η YOLOv11 Segmentation υποστηρίζει το δεύτερο (instance segmentation), συνδυάζοντας ταχύτητα και ακρίβεια.

Τεχνικά χαρακτηριστικά:

- Περιλαμβάνει segmentation head στο τέλος του CNN

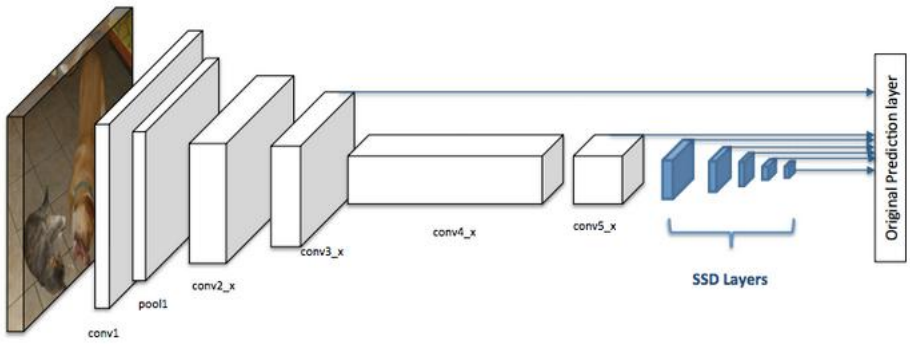
- Κάνει pixel-wise ταξινόμηση των περιοχών που έχουν ήδη εντοπιστεί
- Διατηρεί την υψηλή ταχύτητα του YOLO, ακόμα και σε απαιτητικά σενάρια

Πλεονεκτήματα:

- Πιο λεπτομερής κατανόηση του περιεχομένου της εικόνας
- Καλύτερη ακρίβεια σε πολύπλοκες σκηνές (π.χ. αντικείμενα που αλληλεπικαλύπτονται)
- Χρήσιμο για εφαρμογές όπως ιατρική διάγνωση, αυτόνομα οχήματα, ανάλυση εικόνας σε βιομηχανικά περιβάλλοντα

(SSD (Single Shot MultiBox Detector)2.4.2)

Το SSD (Single Shot MultiBox Detector) είναι ένα μοντέλο ανίχνευσης αντικειμένων το οποίο, όπως και το YOLO, εκτελεί εντοπισμό και ταξινόμηση σε ένα ενιαίο βήμα. Το κύριο πλεονέκτημά του είναι ότι αξιοποιεί πολλαπλά επίπεδα χαρακτηριστικών (feature maps) από διαφορετικά βάθη στο δίκτυο για να εντοπίσει αντικείμενα σε διάφορες κλίμακες.



Αρχιτεκτονική του SSD:

- Βασίζεται σε ένα CNN για εξαγωγή χαρακτηριστικών, όπως π.χ. το VGG16.
- Δημιουργεί πολλαπλά feature maps σε διαφορετικά επίπεδα βάθους.
- Σε κάθε feature map εφαρμόζονται φίλτρα που προβλέπουν:
  - Πολλαπλά bounding boxes (anchors / default boxes)
  - Πιθανότητες κατηγοριών για κάθε κουτί

Βασική αρχή λειτουργίας: Το SSD τοποθετεί προκαθορισμένα πλαίσια (default boxes) σε κάθε σημείο των feature maps. Για κάθε κουτί, προβλέπει την πιθανότητα ότι περιέχει ένα αντικείμενο συγκεκριμένης κατηγορίας και την απαραίτητη μετατόπιση για να ταιριάζει καλύτερα στο αντικείμενο της εικόνας.

Πλεονεκτήματα του SSD:

- **Ταχύτητα:** Είναι γρηγορότερο από το Faster R-CNN και σχεδόν εξίσου γρήγορο με το YOLO.
- **Ανίχνευση σε πολλαπλές κλίμακες:** Χάρη στη χρήση πολλών feature maps.
- **Απλότητα αρχιτεκτονικής:** Δεν χρειάζεται ξεχωριστό στάδιο προτάσεων περιοχών (region proposal stage).

Μειονεκτήματα:

- Σε ορισμένες περιπτώσεις, η ακρίβεια είναι χαμηλότερη σε σχέση με πιο εξελιγμένα μοντέλα όπως το Faster R-CNN.
- Δυσκολεύεται στην αναγνώριση μικρών αντικειμένων, ειδικά σε περιβάλλοντα με θόρυβο ή αλληλοκάλυψη.

Εφαρμογές SSD:

- Mobile και embedded συστήματα (π.χ. Android-based apps)
- Πραγματικού χρόνου εφαρμογές με περιορισμένους πόρους
- Ανίχνευση αντικειμένων σε βίντεο μεσαίας ακρίβειας αλλά υψηλής απόδοσης

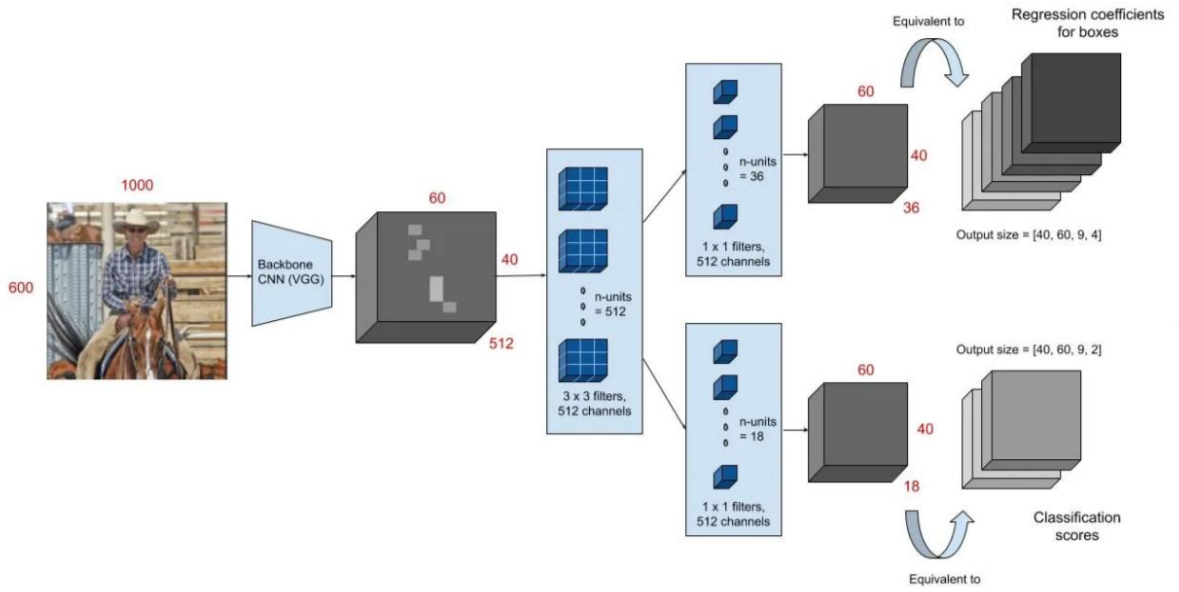
Συνοπτικά: Το SSD ισορροπεί μεταξύ ταχύτητας και ακρίβειας, καθιστώντας το ιδανική επιλογή για εφαρμογές όπου η γρήγορη απόκριση είναι σημαντικότερη από τη μέγιστη δυνατή ακρίβεια. Η δυνατότητα του να εκτελεί ανίχνευση σε πολλαπλά επίπεδα χαρακτηριστικών το καθιστά ισχυρό σε περιβάλλοντα με ποικιλία μεγεθών αντικειμένων.

(Faster R-CNN (Region-based Convolutional Neural Networks) 2.4.3)

Το Faster R-CNN είναι μία από τις πιο επιτυχημένες και ευρέως χρησιμοποιούμενες αρχιτεκτονικές για ανίχνευση αντικειμένων με υψηλή ακρίβεια. Ανήκει στην κατηγορία των δικτύων δύο σταδίων

(two-stage detectors), σε αντίθεση με τα YOLO και SSD που είναι one-stage detectors.

Αρχιτεκτονική και λειτουργία: Το Faster R-CNN βασίζεται στην ιδέα ότι πρώτα πρέπει να εντοπιστούν πιθανές περιοχές που περιέχουν αντικείμενα (Region Proposals), και στη συνέχεια να γίνει ταξινόμηση και ρύθμιση του πλαισίου σε αυτές τις περιοχές.



Η διαδικασία περιλαμβάνει:

1. **CNN Backbone:** Συνήθως χρησιμοποιείται ένα ισχυρό δίκτυο όπως το ResNet ή VGG για εξαγωγή χαρακτηριστικών.
2. **Region Proposal Network (RPN):** Ένα μικρό δίκτυο που προτείνει περιοχές (anchors) πιθανής παρουσίας αντικειμένων.
3. **ROI Pooling:** Τα προτεινόμενα regions μετασχηματίζονται σε σταθερού μεγέθους περιοχές.
4. **Classification & Bounding Box Regression:** Για κάθε region γίνεται πρόβλεψη κατηγορίας και επαναϋπολογισμός του πλαισίου.

Πλεονεκτήματα:

- Πολύ υψηλή ακρίβεια, ειδικά σε datasets με σύνθετες σκηνές ή μικρά αντικείμενα.
- Αποτελεσματικός διαχωρισμός μεταξύ ταξινόμησης και εντοπισμού.

Μειονεκτήματα:

- Σχετικά αργό σε σχέση με SSD ή YOLO.
- Περισσότεροι υπολογιστικοί πόροι και μεγαλύτερος χρόνος εκπαίδευσης.

Εφαρμογές Faster R-CNN:

- Ιατρική απεικόνιση (π.χ. ανίχνευση καρκινικών κυττάρων)
- Ανάλυση εικόνας υψηλής ανάλυσης (δορυφορικές εικόνες, επιτήρηση)
- Αυτόματα συστήματα ελέγχου ποιότητας στη βιομηχανία

Σύγκριση με άλλα μοντέλα: Το Faster R-CNN υπερέχει σε ποιότητα και λεπτομέρεια, όμως υστερεί σε ταχύτητα. Χρησιμοποιείται σε περιπτώσεις όπου η ακρίβεια είναι κρίσιμη και ο χρόνος επεξεργασίας λιγότερο σημαντικός.
Η σχεδίασή του ως δίκτυο δύο σταδίων το καθιστά κατάλληλο για εφαρμογές που απαιτούν πολύ ακριβή αναγνώριση και τοπικό προσδιορισμό αντικειμένων.

(Σύγκριση μοντέλων και κριτήρια επιλογής 2.5)

Κατά την επιλογή μοντέλου για ανίχνευση αντικειμένων, είναι σημαντικό να ληφθούν υπόψη διάφορα τεχνικά και λειτουργικά κριτήρια, καθώς κάθε μοντέλο προσφέρει πλεονεκτήματα και μειονεκτήματα που το καθιστούν καταλληλότερο για διαφορετικούς τύπους εφαρμογών.

Κριτήρια Σύγκρισης:

1. **Ακρίβεια (Accuracy):** Πόσο σωστά αναγνωρίζει και εντοπίζει τα αντικείμενα;
2. **Ταχύτητα (Inference Time / FPS):** Πόσο γρήγορα μπορεί να επεξεργαστεί μια εικόνα;
3. **Υπολογιστικό Κόστος (Computational Load):** Απαιτήσεις σε επεξεργαστική ισχύ και μνήμη.
4. **Διαχείριση μικρών αντικειμένων:** Ικανότητα αναγνώρισης λεπτομερειών σε πολυπληθείς ή σύνθετες σκηνές.
5. **Επεκτασιμότητα και ευκολία ενσωμάτωσης σε εφαρμογές.**

Σύγκριση YOLO - SSD - Faster R-CNN:

Χαρακτηριστικό	**YOLO**	**SSD**	**Faster R-CNN**
Ταχύτητα	Πολύ υψηλή (real-time)	Υψηλή	Μέτρια - Χαμηλή
Ακρίβεια	Καλή - Πολύ καλή	Καλή	Πολύ υψηλή
Εντοπισμός μικρών αντικειμένων	Μέτριος	Μέτριος	Πολύ καλός
Υπολογιστικοί Πόροι	Χαμηλοί έως μέτριοι	Χαμηλοί	Υψηλοί
Πολυπλοκότητα υλοποίησης	Χαμηλή	Μέτρια	Υψηλή
Κατάλληλο για	Πραγματικό χρόνο, κινητές συσκευές	Real-time εφαρμογές	Ιατρική, βιομηχανία, έρευνα

Αιτιολόγηση Επιλογής YOLOv11 Segmentation στην παρούσα εργασία: Η τελική επιλογή του YOLOv11 Segmentation βασίζεται στην ανάγκη συνδυασμού ακρίβειας και ταχύτητας για την εφαρμογή πραγματικού χρόνου που υλοποιείται στην παρούσα πτυχιακή εργασία. Σε σύγκριση με άλλα μοντέλα όπως SSD ή Faster R-CNN, το YOLOv11 Segmentation:

- Παρέχει καλύτερη ακρίβεια τοπικής ανίχνευσης μέσω segmentation.
- Έχει πιο λεπτομερή αναγνώριση αντικειμένων, ειδικά σε αλληλοκαλυπτόμενα στοιχεία.
- Διατηρεί διαχειρίσιμο υπολογιστικό κόστος.
- Είναι ιδανικό για embedded ή edge εφαρμογές που απαιτούν άμεση απόκριση.

Η επιλογή αυτή υποστηρίζεται τόσο από τεχνικά δεδομένα όσο και από τις λειτουργικές απαιτήσεις του συστήματος που αναπτύσσεται.

Η τελική επιλογή του YOLOv11 Segmentation στην παρούσα εργασία βασίστηκε στην ανάγκη για ταυτόχρονη ακρίβεια, ταχύτητα και λεπτομερή αποτύπωση της πληροφορίας, προκειμένου να καλυφθούν οι απαιτήσεις της εφαρμογής.

ΚΕΦΑΛΑΙΟ 3 Μηχανική Μάθηση για Αναγνώριση Ζημιών

Στο κεφάλαιο αυτό περιγράφεται η μεθοδολογία που ακολουθείται για την ανάπτυξη ενός συστήματος αναγνώρισης ζημιών σε αυτοκίνητα, αξιοποιώντας σύγχρονες τεχνικές Βαθιάς Μάθησης. Η προσέγγιση βασίζεται στην ανάλυση εικόνας με χρήση αλγορίθμων ανίχνευσης αντικειμένων και segmentation, με σκοπό την αυτόματη αναγνώριση και κατηγοριοποίηση τύπων ζημιάς.

Περιγραφή Στόχου και Προσέγγισης 3.1

Για την εκπαίδευση και αξιολόγηση του συστήματος αναγνώρισης ζημιών σε οχήματα, χρησιμοποιήθηκαν δύο δημοσίως διαθέσιμα και εξειδικευμένα σύνολα δεδομένων: το CARDD (Car Damage Detection Dataset) και το Car Parts and Car Damages Dataset από το Kaggle. Αυτά τα datasets παρέχουν πλούσιο υλικό από εικόνες οχημάτων με διάφορους τύπους ζημιών, καθώς και τα αντίστοιχαannotations για κάθε εικόνα, οι οποίες περιγράφουν με ακρίβεια το είδος της βλάβης.

Το CARDD dataset περιέχει 4.000 εικόνες υψηλής ανάλυσης με περισσότερες από 9.000 επισημασμένες περιπτώσεις ζημιών σε μία από τις έξι παραπάνω κατηγορίες. Το Car Parts and Car Damages Dataset περιλαμβάνει επιπλέον 998 εικόνες επισημασμένες με πολύγωνα για μέρη αυτοκινήτων και 814 εικόνες επισημασμένες με ζημιές, προσφέροντας χρήσιμο υλικό για την εκπαίδευση μοντέλων εντοπισμού περιοχών και κατηγοριών.
Σε περιπτώσεις όπου τα υπάρχουσα anotations δεν κάλυπταν συγκεκριμένες ανάγκες ή σε περιπτώσεις που δεν υπήρχαν , πραγματοποιήθηκε χειροκίνητη σήμανση μέσω του εργαλείου **VoTT (Visual Object Tagging Tool)** της Microsoft.
Το VoTT αποτελεί ένα εύχρηστο γραφικό εργαλείο επισημάνσεων που υποστηρίζει διάφορες μορφές εξαγωγής δεδομένων, επιτρέποντας τη δημιουργία annotations με ακρίβεια μέσω πολυγώνων ή πλαισίων. Οι επισημάνσεις που δημιουργήθηκαν με το VoTT μετατράπηκαν στη συνέχεια σε μορφή YOLO segmentation, ώστε να χρησιμοποιηθούν για την εκπαίδευση του μοντέλου ανίχνευσης.

Πριν την εκπαίδευση και κατά την διάρκεια της, εφαρμόστηκαν βασικές τεχνικές προεπεξεργασίας και ενίσχυσης δεδομένων, όπως αλλαγή μεγέθους, κανονικοποίηση (normalization), περιστροφές,

καθρέφτισμα και τυχαίες μετατοπίσεις, με στόχο την αύξηση της ποικιλίας και τη βελτίωση της γενίκευσης του μοντέλου.
Συνολικά, το dataset περιλαμβάνει πάνω από 5.800 εικόνες, οι οποίες χωρίστηκαν σε τρία υποσύνολα:

- 70% για εκπαίδευση (training set)
- 20% για επικύρωση (validation set)
- 10% για δοκιμή (test set)

Αυτός ο διαχωρισμός διασφαλίζει την ισορροπημένη αξιολόγηση της απόδοσης του μοντέλου και αποτρέπει το φαινόμενο της υπερπροσαρμογής (overfitting).

Στόχος είναι η δημιουργία ενός έξυπνου συστήματος που:

- Αναγνωρίζει την ύπαρξη ζημιάς σε φωτογραφίες ή βίντεο.
- Εντοπίζει με ακρίβεια τη θέση της ζημιάς.
- Κάνει segmentation στο περίγραμμα της ζημιάς για μεγαλύτερη λεπτομέρεια.

Η προσέγγιση βασίζεται σε εποπτευόμενη μάθηση (supervised learning) με χρήση προκαταχωρημένων εικόνων ζημιών, όπου η κάθε εικόνα συνοδεύεται από τις αντίστοιχες ετικέτες (labels) και μάσκες.

Οι κατηγορίες ζημιών που αναγνωρίζει το τελικό μοντέλο είναι οι εξής:

- dent (λακκούβα/βαθούλωμα)
- scratch (γρατζουνιά)
- crack (ρωγμή)
- glass shatter (θραύση τζαμιού)
- lamp broken (σπασμένο φανάρι)
- tire flat (ξεφούσκωτο λάστιχο)

Παράλληλα το δεύτερο μοντέλο, πέρα από τον τύπο της ζημιάς, αναγνωρίζει και σε ποιο τμήμα του αυτοκινήτου εμφανίζεται .Το συγκεκριμένο μοντέλο το χρειαζόμαστε έτσι ώστε συνδυάζοντας τα δυο μοντέλα να έχουμε μια ακριβής προσέγγιση του μέρους όπου εμφανίζεται η ζημιά . Τα τμήματα κωδικοποιούνται ως εξής:

- Quarter-panel
- Front-wheel
- Back-window
- Trunk
- Front-door
- Rocker-panel
- Grille
- Windshield
- Front-window
- Back-door
- Headlight
- Back-wheel
- Back-windshield
- Hood
- Fender
- Tail-light
- License-plate
- Front-bumper
- Back-bumper
- Mirror
- Roof

Αρχιτεκτονική του Μοντέλου και Ρυθμίσεις Εκπαίδευσης 3.2

Η εκπαίδευση πραγματοποιήθηκε τοπικά σε υπολογιστή με GPU GIGABYTE GeForce RTX 3060 12GB, αξιοποιώντας το CUDA 12.6 για την επιτάχυνση υπολογισμών μέσω GPU. Χρησιμοποιήθηκε επίσης η τεχνική mixed precision training (AMP) για μείωση της κατανάλωσης VRAM και επιτάχυνση της εκπαίδευσης, χωρίς απώλεια στην ακρίβεια.
Η βιβλιοθήκη που χρησιμοποιήθηκε για την εκπαίδευση ήταν η Ultralytics YOLO, η οποία υποστηρίζει YOLOv11s και παρέχει εύχρηστη διεπαφή για τροποποίηση παραμέτρων. Η χρήση του CUDA 12.6, σε συνδυασμό με κατάλληλους drivers, επέτρεψε πλήρη αξιοποίηση της κάρτας γραφικών, μειώνοντας σημαντικά τον χρόνο εκπαίδευσης και επιτρέποντας την εκτέλεση με υψηλή ανάλυση εικόνων (832x832).

Ο παρακάτω κώδικας έχει ως σκοπό τη μετατροπή των annotations από μορφή JSON σε YOLO format, ειδικά για segmentation.

```python
import os
import json
import glob
import cv2 # OpenCV for reading image dimensions

# -------------------------
# Configuration Parameters
# -------------------------
json_dir = "E:/Ptyxiakh_pics/Car_Damage_Detection/Car_parts_dataset/File1/ann"
image_dir = "E:/Ptyxiakh_pics/Car_Damage_Detection/Car_parts_dataset/File1/images"
output_dir = "Segmentation_labels_result"
os.makedirs(output_dir, exist_ok=True)

img_extensions = [".jpg", ".jpeg", ".png"]

class_mapping = {
    "Quarter-panel": 0,
    "Front-wheel": 1,
    "Back-window": 2,
    "Trunk": 3,
    "Front-door": 4,
    "Rocker-panel": 5,
    "Grille": 6,
    "Windshield": 7,
    "Front-window": 8,
    "Back-door": 9,
    "Headlight": 10,
    "Back-wheel": 11,
    "Back-windshield": 12,
    "Hood": 13,
    "Fender": 14,
    "Tail-light": 15,
    "License-plate": 16,
    "Front-bumper": 17,
    "Back-bumper": 18,
    "Mirror": 19,
    "Roof": 20
}
```

```python
def clip_value(val):
    return max(0.0, min(val, 1.0))

def get_image_dimensions(base_name, image_dir, extensions):
    for ext in extensions:
        image_path = os.path.join(image_dir, base_name + ext)
        if os.path.exists(image_path):
            img = cv2.imread(image_path)
            if img is not None:
                h, w = img.shape[:2]
                return w, h
    return None, None

# -------------------------
# Process JSON Annotations
# -------------------------
json_files = glob.glob(os.path.join(json_dir, "*.json"))

for json_file in json_files:
    base_name = os.path.splitext(os.path.basename(json_file))[0]

    img_width, img_height = get_image_dimensions(base_name, image_dir, img_extensions)
    if img_width is None or img_height is None:
        print(f"Warning: Could not find image for {base_name}. Skipping.")
        continue

    with open(json_file, "r") as f:
        data = json.load(f)

    objects = data.get("objects", [])
    label_lines = []
```

```python
    for obj in objects:
        # We only care about polygon objects
        if obj.get("geometryType") != "polygon":
            continue

        class_title = obj.get("classTitle")
        if class_title not in class_mapping:
            print(f"Warning: Class '{class_title}' not in class mapping. Skipping.")
            continue
        class_index = class_mapping[class_title]

        # Grab the exterior polygon points
        points = obj.get("points", {}).get("exterior", [])
        if not points:
            continue

        # Build the normalized segmentation string for YOLO
        norm_points = []
        for (x, y) in points:
            nx = clip_value(x / img_width)
            ny = clip_value(y / img_height)
            norm_points.append(f"{nx:.6f}")
            norm_points.append(f"{ny:.6f}")

        # For YOLO-seg: class_id + x1 y1 x2 y2 ...
        line = f"{class_index} " + " ".join(norm_points)
        label_lines.append(line)

    # Write out one label file per image
    output_file = os.path.join(output_dir, f"{base_name}.txt")
    with open(output_file, "w") as f_out:
        for line in label_lines:
            f_out.write(line + "\n")

    print(f"Processed {json_file} -> {output_file}")
```

Τι κάνει ο κώδικας:

1. Αναζητά JSON αρχεία σημειώσεων στο json_dir.
2. Βρίσκει το αντίστοιχο αρχείο εικόνας από το image_dir.
3. Διαβάζει τα πολύγωνα (polygons) των αντικειμένων.
4. Κανονικοποιεί τις συντεταγμένες ανάλογα με το πλάτος και ύψος της εικόνας.
5. **Δημιουργεί αρχείο YOLO-format .txt** για κάθε εικόνα στο output_dir.

Επιμέρους στοιχεία του κώδικα:

- class_mapping: Αντιστοίχιση κάθε κατηγορίας με έναν ακέραιο class_id.
- clip_value(val): Εξασφαλίζει ότι οι τιμές [x, y] είναι στο [0.0, 1.0].
- get_image_dimensions(...): Παίρνει το μέγεθος της εικόνας για ορθή κανονικοποίηση.
- **Μορφή εξόδου YOLO-seg:**
    - Πρώτο στοιχείο: αριθμός κατηγορίας (π.χ. 3 = Trunk)
    - Υπόλοιπα: συντεταγμένες x y x y ... σε float (0.0–1.0)

Με παρόμοιο τρόπο ο παρακάτω κώδικας διαβάζει ένα αρχείο JSON σε μορφή COCO annotation format (που περιέχει πληροφορίες για εικόνες και ανιχνευμένες βλάβες σε οχήματα) και το μετατρέπει σε YOLO format segmentation labels

Τι κάνει ο κώδικας:
Διαβάζει:

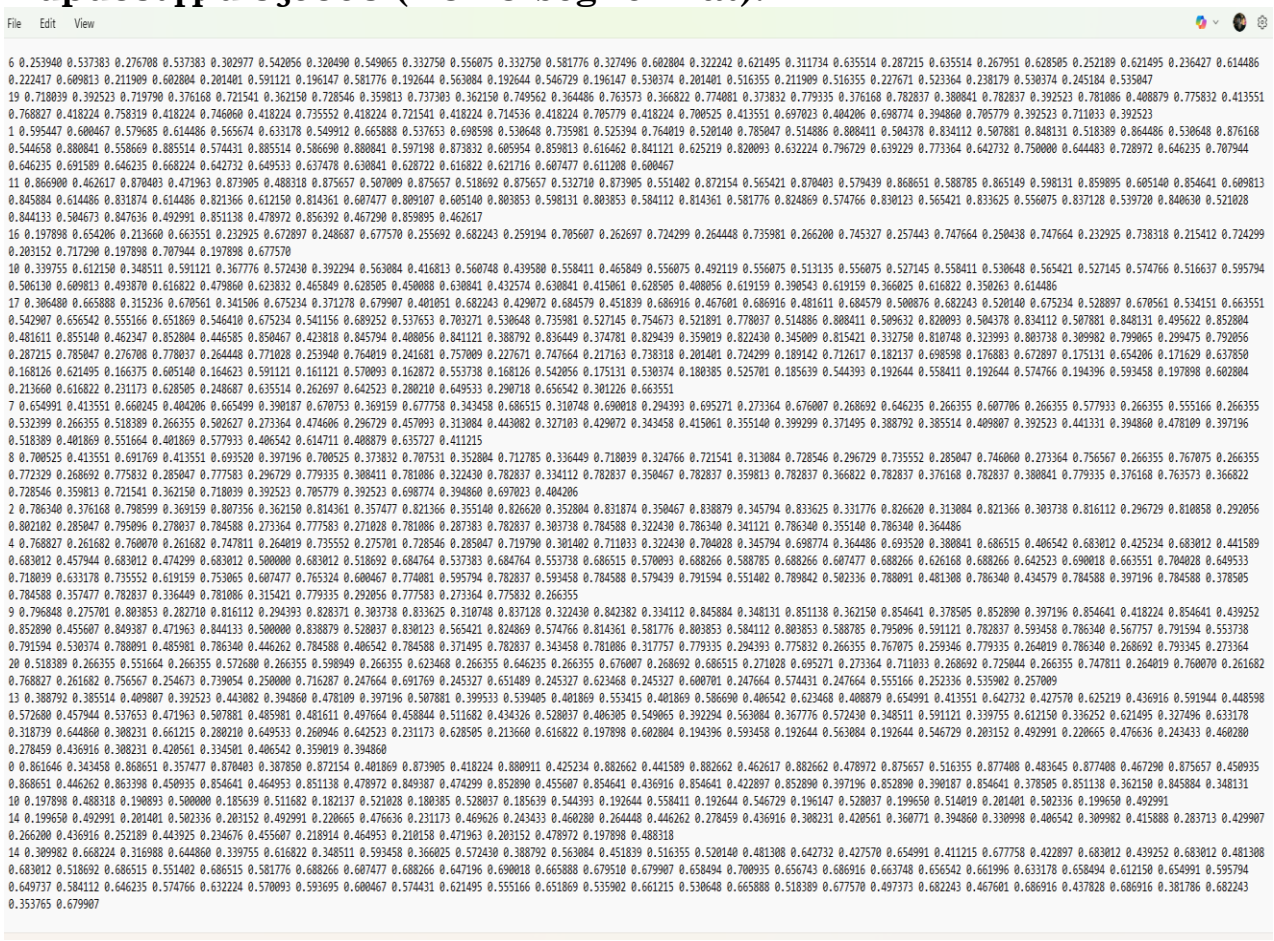
1. το COCO JSON αρχείο (instances_val2017.json)
2. τις αντίστοιχες εικόνες
3. και γράφει τα YOLO-style labels σε μορφή .txt στο output_dir.

```python
import os
import json
import cv2
from collections import defaultdict

# -------------------------
# Configuration Parameters
# -------------------------
json_path = "E:\\CarDD_release\\CarDD_COCO\\annotations\\instances_val2017.json"
image_dir = "E:\\Ptyxiakh_Project_Damage_detection\\images\\val"
output_dir = "E:\\Ptyxiakh_Project_Damage_detection\\labels\\val"
os.makedirs(output_dir, exist_ok=True)

img_extensions = [".jpg", ".jpeg", ".png"]

# CarDD category mapping (adjust if needed)
class_mapping = {
    "dent": 0,
    "scratch": 1,
    "crack": 2,
    "glass shatter": 3,
    "lamp broken": 4,
    "tire flat": 5
}

def clip_value(val):
    return max(0.0, min(val, 1.0))

def get_image_dimensions(file_name, image_dir, extensions):
    """
    Try to read image dimensions from file.
    file_name should include extension if available.
    """
    base_name = os.path.splitext(file_name)[0]
    for ext in extensions:
        image_path = os.path.join(image_dir, base_name + ext)
        if os.path.exists(image_path):
            img = cv2.imread(image_path)
            if img is not None:
                h, w = img.shape[:2]
                return w, h
    return None, None

# -------------------------
# Load JSON and Preprocess
# -------------------------
with open(json_path, "r") as f:
    data = json.load(f)

# Build a dictionary for images using image id
images_dict = {img["id"]: img for img in data["images"]}

# Build a dictionary for categories (mapping category id to category name)
categories_dict = {cat["id"]: cat["name"] for cat in data["categories"]}

# Group annotations by image id for faster lookup
annotations_by_image = defaultdict(list)
for ann in data["annotations"]:
    annotations_by_image[ann["image_id"]].append(ann)
```

```
# -------------------------
# Process Each Image
# -------------------------
for image in data["images"]:
    file_name = image["file_name"]
    base_name = os.path.splitext(file_name)[0]

    # Try to get image dimensions from file; if not available, fallback to JSON dimensions
    img_width, img_height = get_image_dimensions(file_name, image_dir, img_extensions)
    if img_width is None or img_height is None:
        print(f"Warning: Could not find image file for {file_name}. Using JSON dimensions.")
        img_width = image.get("width")
        img_height = image.get("height")
        if not img_width or not img_height:
            print(f"Warning: No valid dimensions for {file_name}. Skipping.")
            continue

    label_lines = []
    anns = annotations_by_image.get(image["id"], [])
    for ann in anns:
        # Get category name and then look up class index via our mapping
        cat_id = ann["category_id"]
        cat_name = categories_dict.get(cat_id, "").lower()  # using lowercase for matching
        if cat_name not in class_mapping:
            print(f"Warning: Category '{cat_name}' not in class mapping for image {file_name}. Skipping annotation.")
            continue
        class_index = class_mapping[cat_name]

        segmentation = ann.get("segmentation", [])
        # Expecting segmentation to be a list of polygons; each polygon is a list of coordinates.
        if not isinstance(segmentation, list):
            continue

        for poly in segmentation:
            if not isinstance(poly, list) or len(poly) % 2 != 0:
                continue  # Skip malformed polygons

            norm_points = []
            for i, coord in enumerate(poly):
                if i % 2 == 0:  # x coordinate
                    nx = clip_value(coord / img_width)
                    norm_points.append(f"{nx:.6f}")
                else:  # y coordinate
                    ny = clip_value(coord / img_height)
                    norm_points.append(f"{ny:.6f}")

            # Format for YOLO segmentation: class_id followed by normalized coordinates
            line = f"{class_index} " + " ".join(norm_points)
            label_lines.append(line)

    # Write out one label file per image (even if empty)
    output_file = os.path.join(output_dir, f"{base_name}.txt")
    with open(output_file, "w") as f_out:
        for line in label_lines:
            f_out.write(line + "\n")

    print(f"Processed image {file_name} -> {output_file}")
```

Παράδειγμα εξόδου (YOLO seg format):

File Edit View



```
6 0.253940 0.537383 0.276708 0.537383 0.302977 0.542056 0.320490 0.549065 0.332750 0.556075 0.332750 0.581776 0.327496 0.602804 0.322242 0.621495 0.311734 0.635514 0.287215 0.635514 0.267951 0.628505 0.252189 0.621495 0.236427 0.614486
0.222417 0.609813 0.211909 0.602804 0.201401 0.591121 0.196147 0.581776 0.192644 0.563084 0.192644 0.546729 0.196147 0.530374 0.201401 0.516355 0.211909 0.516355 0.227671 0.523364 0.238179 0.530374 0.245184 0.535047
19 0.718039 0.392523 0.719790 0.376168 0.721541 0.362150 0.728546 0.359813 0.737303 0.362150 0.749562 0.364486 0.763573 0.366822 0.774081 0.373832 0.779335 0.376168 0.782837 0.380841 0.782837 0.392523 0.781086 0.408879 0.775832 0.413551
0.768827 0.418224 0.758319 0.418224 0.746060 0.418224 0.735552 0.418224 0.721541 0.418224 0.714536 0.418224 0.705779 0.418224 0.700525 0.413551 0.697023 0.404206 0.698774 0.394860 0.705779 0.392523 0.711033 0.392523
1 0.595447 0.600467 0.579685 0.614486 0.565674 0.633178 0.549912 0.665888 0.537653 0.698598 0.530648 0.735981 0.525394 0.764019 0.520140 0.785047 0.514886 0.808411 0.504378 0.834112 0.507881 0.848131 0.518389 0.864486 0.530648 0.876168
0.544658 0.880841 0.558669 0.885514 0.574431 0.885514 0.586690 0.880841 0.597198 0.873832 0.605954 0.859813 0.616462 0.841121 0.625219 0.820093 0.632224 0.796729 0.639229 0.773364 0.642732 0.750000 0.644483 0.728972 0.646235 0.707944
0.646235 0.691589 0.646235 0.668224 0.642732 0.649533 0.637478 0.630841 0.628722 0.616822 0.621716 0.607477 0.611208 0.600467
11 0.866900 0.462617 0.870403 0.471963 0.873905 0.488318 0.875657 0.507009 0.875657 0.518692 0.875657 0.532710 0.873905 0.551402 0.872154 0.565421 0.870403 0.579439 0.868651 0.588785 0.865149 0.598131 0.859895 0.605140 0.854641 0.609813
0.845884 0.614486 0.831874 0.614486 0.821366 0.612150 0.814361 0.607477 0.809107 0.605140 0.803853 0.598131 0.803853 0.584112 0.814361 0.581776 0.824869 0.574766 0.830123 0.565421 0.833625 0.556075 0.837128 0.539720 0.840630 0.521028
0.844133 0.504673 0.847636 0.492991 0.851138 0.478972 0.856392 0.467290 0.859895 0.462617
16 0.197898 0.654206 0.213660 0.663551 0.232925 0.672897 0.248687 0.677570 0.255692 0.682243 0.259194 0.705607 0.262697 0.724299 0.264448 0.735981 0.266200 0.745327 0.257443 0.747664 0.250438 0.747664 0.232925 0.738318 0.215412 0.724299
0.203152 0.717290 0.197898 0.707944 0.197898 0.677570
10 0.339755 0.612150 0.348511 0.591121 0.367776 0.572430 0.392294 0.563084 0.416813 0.560748 0.439580 0.558411 0.465849 0.556075 0.492119 0.556075 0.513135 0.556075 0.527145 0.558411 0.530648 0.565421 0.527145 0.574766 0.516637 0.595794
0.506130 0.609813 0.493870 0.616822 0.479860 0.623832 0.465849 0.628505 0.450088 0.630841 0.432574 0.630841 0.415061 0.628505 0.408056 0.619159 0.390543 0.619159 0.366025 0.616822 0.350263 0.614486
17 0.306480 0.665888 0.315236 0.670561 0.341506 0.675234 0.371278 0.679907 0.401051 0.682243 0.429072 0.684579 0.451839 0.686916 0.467601 0.686916 0.481611 0.684579 0.500876 0.682243 0.520140 0.675234 0.528897 0.670561 0.534151 0.663551
0.542907 0.656542 0.555166 0.651869 0.546410 0.675234 0.541156 0.689252 0.537653 0.703271 0.530648 0.735981 0.527145 0.754673 0.521891 0.778037 0.514886 0.808411 0.509632 0.820093 0.504378 0.834112 0.507881 0.848131 0.495622 0.852804
0.481611 0.855140 0.462347 0.852804 0.446585 0.850467 0.423818 0.845794 0.408056 0.841121 0.388792 0.836449 0.374781 0.829439 0.359019 0.822430 0.345009 0.815421 0.332750 0.810748 0.323993 0.803738 0.309982 0.799065 0.299475 0.792056
0.287215 0.785047 0.276708 0.778037 0.264448 0.771028 0.253940 0.764019 0.241681 0.757009 0.227671 0.747664 0.217163 0.738318 0.201401 0.724299 0.189142 0.712617 0.182137 0.698598 0.176883 0.672897 0.175131 0.654206 0.171629 0.637850
0.168126 0.621495 0.166375 0.605140 0.164623 0.591121 0.161121 0.570093 0.162872 0.553738 0.168126 0.542056 0.175131 0.530374 0.180385 0.525701 0.185639 0.544393 0.192644 0.558411 0.192644 0.574766 0.194396 0.593458 0.197898 0.602804
0.213660 0.616822 0.231173 0.628505 0.248687 0.635514 0.262697 0.642523 0.280210 0.649533 0.290718 0.656542 0.301226 0.663551
7 0.654991 0.413551 0.660245 0.404206 0.665499 0.390187 0.670753 0.369159 0.677758 0.343458 0.686515 0.310748 0.690018 0.294393 0.695271 0.273364 0.676007 0.268692 0.646235 0.266355 0.607706 0.266355 0.577933 0.266355 0.555166 0.266355
0.532399 0.266355 0.518389 0.266355 0.502627 0.273364 0.474606 0.296729 0.457093 0.313084 0.443082 0.327103 0.429072 0.343458 0.415061 0.355140 0.399299 0.371495 0.388792 0.385514 0.409807 0.392523 0.441331 0.394860 0.478109 0.397196
0.518389 0.401869 0.551664 0.401869 0.577933 0.406542 0.614711 0.408879 0.635727 0.411215
8 0.700525 0.413551 0.691769 0.413551 0.693520 0.397196 0.700525 0.373832 0.707531 0.352804 0.712785 0.336449 0.718039 0.324766 0.721541 0.313084 0.728546 0.296729 0.735552 0.285047 0.746060 0.273364 0.756567 0.266355 0.767075 0.266355
0.772329 0.268692 0.775832 0.285047 0.777583 0.296729 0.779335 0.308411 0.781086 0.322430 0.782837 0.334112 0.782837 0.350467 0.782837 0.359813 0.782837 0.366822 0.782837 0.376168 0.782837 0.380841 0.779335 0.376168 0.763573 0.366822
0.728546 0.359813 0.721541 0.362150 0.718039 0.392523 0.705779 0.392523 0.698774 0.394860 0.697023 0.404206
2 0.786340 0.376168 0.798599 0.369159 0.807356 0.362150 0.814361 0.357477 0.821366 0.355140 0.826620 0.352804 0.831874 0.350467 0.838879 0.345794 0.833625 0.331776 0.826620 0.313084 0.821366 0.303738 0.816112 0.296729 0.810858 0.292056
0.802102 0.285047 0.795096 0.278037 0.784588 0.273364 0.777583 0.271028 0.781086 0.287383 0.782837 0.303738 0.784588 0.322430 0.786340 0.341121 0.786340 0.355140 0.786340 0.364486
4 0.768827 0.261682 0.760070 0.261682 0.747811 0.264019 0.735552 0.275701 0.728546 0.285047 0.719790 0.301402 0.711033 0.322430 0.704028 0.345794 0.698774 0.364486 0.693520 0.380841 0.686515 0.406542 0.683012 0.425234 0.683012 0.441589
0.683012 0.457944 0.683012 0.474299 0.683012 0.500000 0.683012 0.518692 0.684764 0.537383 0.684764 0.553738 0.686515 0.570093 0.688266 0.588785 0.688266 0.607477 0.688266 0.626168 0.688266 0.642523 0.690018 0.663551 0.704028 0.649533
0.718039 0.633178 0.735552 0.619159 0.753065 0.607477 0.765324 0.600467 0.774081 0.595794 0.782837 0.593458 0.784588 0.579439 0.791594 0.551402 0.789842 0.502336 0.788091 0.481308 0.786340 0.434579 0.784588 0.397196 0.784588 0.378505
0.784588 0.357477 0.782837 0.336449 0.781086 0.315421 0.779335 0.292056 0.777583 0.273364 0.775832 0.266355
9 0.796848 0.275701 0.803853 0.282710 0.816112 0.294393 0.828371 0.303738 0.833625 0.310748 0.837128 0.322430 0.842382 0.334112 0.845884 0.348131 0.851138 0.362150 0.854641 0.378505 0.852890 0.397196 0.854641 0.418224 0.854641 0.439252
0.852890 0.455607 0.849387 0.471963 0.844133 0.500000 0.838879 0.528037 0.830123 0.565421 0.824869 0.574766 0.814361 0.581776 0.803853 0.584112 0.803853 0.588785 0.795096 0.591121 0.782837 0.593458 0.786340 0.567757 0.791594 0.553738
0.791594 0.530374 0.788091 0.485981 0.786340 0.446262 0.784588 0.406542 0.784588 0.371495 0.782837 0.343458 0.781086 0.317757 0.779335 0.294393 0.775832 0.266355 0.767075 0.259346 0.779335 0.264019 0.786340 0.268692 0.793345 0.273364
20 0.518389 0.266355 0.551664 0.266355 0.572680 0.266355 0.598949 0.266355 0.623468 0.266355 0.646235 0.266355 0.676007 0.268692 0.686515 0.271028 0.695271 0.273364 0.711033 0.268692 0.725044 0.266355 0.747811 0.264019 0.760070 0.261682
0.768827 0.261682 0.756567 0.254673 0.739054 0.250000 0.716287 0.247664 0.691769 0.245327 0.651489 0.245327 0.623468 0.245327 0.600701 0.247664 0.574431 0.247664 0.555166 0.252336 0.535902 0.257009
13 0.388792 0.385514 0.409807 0.392523 0.443082 0.394860 0.478109 0.397196 0.507881 0.399533 0.539405 0.401869 0.553415 0.401869 0.586690 0.406542 0.623468 0.408879 0.654991 0.413551 0.642732 0.427570 0.625219 0.436916 0.591944 0.448598
0.572680 0.457944 0.537653 0.471963 0.507881 0.485981 0.481611 0.497664 0.458844 0.511682 0.434326 0.528037 0.406305 0.549065 0.392294 0.563084 0.367776 0.572430 0.348511 0.591121 0.339755 0.612150 0.336252 0.621495 0.327496 0.633178
0.318739 0.644860 0.308231 0.661215 0.280210 0.649533 0.260946 0.642523 0.231173 0.628505 0.213660 0.616822 0.197898 0.602804 0.194396 0.593458 0.192644 0.563084 0.192644 0.546729 0.203152 0.492991 0.220665 0.476636 0.243433 0.460280
0.278459 0.436916 0.308231 0.420561 0.334501 0.406542 0.359019 0.394860
0 0.861646 0.343458 0.868651 0.357477 0.870403 0.387850 0.872154 0.401869 0.873905 0.418224 0.880911 0.425234 0.882662 0.441589 0.882662 0.462617 0.882662 0.478972 0.875657 0.516355 0.877408 0.483645 0.877408 0.467290 0.875657 0.450935
0.868651 0.446262 0.863398 0.450935 0.854641 0.464953 0.851138 0.478972 0.849387 0.474299 0.852890 0.455607 0.854641 0.436916 0.854641 0.422897 0.852890 0.397196 0.852890 0.390187 0.854641 0.378505 0.851138 0.362150 0.845884 0.348131
10 0.197898 0.488318 0.190893 0.500000 0.185639 0.511682 0.182137 0.521028 0.180385 0.528037 0.185639 0.544393 0.192644 0.558411 0.192644 0.546729 0.196147 0.528037 0.199650 0.514019 0.201401 0.502336 0.199650 0.492991
14 0.199650 0.492991 0.201401 0.502336 0.203152 0.492991 0.220665 0.476636 0.231173 0.469626 0.243433 0.460280 0.264448 0.446262 0.278459 0.436916 0.308231 0.420561 0.360771 0.394860 0.330998 0.406542 0.309982 0.415888 0.283713 0.429907
0.266200 0.436916 0.252189 0.443925 0.234676 0.455607 0.218914 0.464953 0.210158 0.471963 0.203152 0.478972 0.197898 0.488318
14 0.309982 0.668224 0.316988 0.644860 0.339755 0.616822 0.348511 0.593458 0.366025 0.572430 0.388792 0.563084 0.451839 0.516355 0.520140 0.481308 0.642732 0.427570 0.654991 0.411215 0.677758 0.422897 0.683012 0.439252 0.683012 0.481308
0.683012 0.518692 0.686515 0.551402 0.686515 0.581776 0.688266 0.607477 0.688266 0.647196 0.690018 0.665888 0.679510 0.679907 0.658494 0.700935 0.656743 0.686916 0.663748 0.656542 0.661996 0.633178 0.658494 0.612150 0.654991 0.595794
0.649737 0.584112 0.646235 0.574766 0.632224 0.570093 0.593695 0.600467 0.574431 0.621495 0.555166 0.651869 0.535902 0.661215 0.530648 0.665888 0.518389 0.677570 0.497373 0.682243 0.467601 0.686916 0.437828 0.686916 0.381786 0.682243
0.353765 0.679907
```

Το κύριο μοντέλο που χρησιμοποιείται για την ανίχνευση και κατάτμηση των ζημιών είναι το YOLOv11 με segmentation capabilities. Παρέχει ταχύτητα και ακρίβεια, απαραίτητα για την επεξεργασία σε πραγματικό χρόνο.

Χρησιμοποιήθηκε η παραλλαγή yolo11s-seg.pt, μια ελαφριά και γρήγορη έκδοση του μοντέλου YOLOv11 με υποστήριξη για instance segmentation. Η έκδοση αυτή είναι ιδανική για embedded εφαρμογές και επεξεργασία σε GPU μεσαίας ισχύος, όπως NVIDIA RTX 3060 (Η κάρτα γραφικών που χρησιμοποιήθηκε για την εκπαίδευση των μοντέλων).

```
from ultralytics import YOLO
model = YOLO('yolo11s-seg.pt')
```

Το yolo11s-seg.pt έχει προεκπαιδευτεί σε μεγάλο dataset και παρέχει έτοιμα βάρη για fine-tuning σε custom δεδομένα. Υποστηρίζει πλήρως segmentation masks και παράγει εξόδους τόσο για bounding boxes όσο και για pixel-level segmentation, καθιστώντας το κατάλληλο για εφαρμογές που απαιτούν ακρίβεια στην τοπική αναγνώριση ζημιών.

Εκπαίδευση μοντέλου αναγνώρισης ζημιών:

```
model.train(
    data="E:\Ptyxiakh_Project_Damage_detection\car_damages.yaml",
    epochs=250,
    imgsz=832,
    batch=16,
    lr0=0.001,
    lrf=0.1,
    momentum=0.937,
    weight_decay=0.0005,
    optimizer='AdamW',
    device='cuda:0',
    workers=4,
    cos_lr=True,
    amp=True,
    hsv_h=0.015, hsv_s=0.7, hsv_v=0.4,
    perspective=0.0005,
    flipud=0.5, fliplr=0.5,
    mosaic=1.0,
    mixup=0.2,
    degrees=0.4,
    scale=0.5,
    shear=0.1,
    patience=30,
    verbose=True
)
```

Στην παραπάνω εικόνα βλέπουμε την κλήση της μεθόδου model.train()με έναν πλήρη κατάλογο παραμέτρων εκπαίδευσης και augmentations.

Αναλυτικότερα :

1. data

- **Τύπος**: string
- **Περιγραφή**: Διαδρομή στο αρχείο YAML που ορίζει το dataset (μονοπάτια εικόνων train/val, ονόματα κλάσεων κ.λπ.).

2. epochs=250

- **Τιμή**: 250
- **Περιγραφή**: Ο αριθμός των πλήρων περασμάτων (epochs) πάνω στο training set.
-

3. imgsz=832

- **Τιμή**: 832
- **Περιγραφή**: Το μέγεθος εκπαίδευσης — η διάσταση (πλάτος/ύψος) στην οποία θα κλιμακώνονται οι εικόνες (συνήθως τετράγωνο).

4. batch=16

- **Τιμή**: 16
- **Περιγραφή**: Μέγεθος παρτίδας (batch size). Ο αριθμός των εικόνων που περνούν ταυτόχρονα από το δίκτυο πριν από κάθε ενημέρωση βαρών.

5. lr0=0.001

- **Τιμή**: 0.001
- **Περιγραφή**: Αρχικό learning rate (βήμα εκμάθησης) για τον optimizer.

6. lrf=0.1

- **Τιμή**: 0.1
- **Περιγραφή**: Τελικός παράγοντας learning rate. Το learning rate στο τέλος της εκπαίδευσης θα είναι lr0 * lrf (εδώ 1e–4).

7. momentum=0.937

- **Τιμή**: 0.937
- **Περιγραφή**: Παράμετρος momentum για επιταχυνόμενη σύγκλιση (εφαρμόζεται αν ο optimizer το υποστηρίζει).

8. weight_decay=0.0005

- **Τιμή**: 5×10^{-4}
- **Περιγραφή**: Παράμετρος L2 regularization για μείωση overfitting (επιβάλλει ποινή σε μεγάλα βάρη).

9. optimizer='AdamW'

- **Τύπος**: string
- **Περιγραφή**: Επιλογή optimizer — εδώ χρησιμοποιείται η παραλλαγή Adam με weight decay.

10. device='cuda:0'

- **Τύπος**: string
- **Περιγραφή**: Επιλογή συσκευής εκπαίδευσης (GPU 0 με CUDA).

11. workers=4

- **Τιμή**: 4
- **Περιγραφή**: Αριθμός parallel data-loading threads (PyTorch DataLoader workers).

Ρυθμίσεις Learning Rate & Precision

- cos_lr=True
  – Ενεργοποιεί cosine annealing schedule για το learning rate (σταδιακή πτώση με cosine curve).
- amp=True
  – Χρήση Automatic Mixed Precision (ημι-ακριβείας float16) για ταχύτερο training και λιγότερη χρήση μνήμης.

Augmentation Παραμέτρους

Οι επόμενες παράμετροι ελέγχουν δυναμικές τροποποιήσεις (augmentations) κάθε batch:

1. **Χρώμα (HSV jitter)**
   - hsv_h=0.015 : ±1.5% τυχαία αλλαγή απόχρωσης (hue)
   - hsv_s=0.7 : ±70% αλλαγή κορεσμού (saturation)
   - hsv_v=0.4 : ±40% αλλαγή φωτεινότητας (value)
2. **Προοπτική**
   - perspective=0.0005
     – Εφαρμογή ελαφριάς τυχαίας προοπτικής (warp) με πολύ μικρή πιθανότητα.
3. **Κατακόρυφη-Οριζόντια Ανατροπή**
   - flipud=0.5 : 50% πιθανότητα vertical flip
   - fliplr=0.5 : 50% πιθανότητα horizontal flip
4. **Mosaic & MixUp**
   - mosaic=1.0 : Mosaic augmentation πάντοτε ενεργό (συνένωση 4 εικόνων σε μία)
   - mixup=0.2 : MixUp augmentation με α = 0.2 (blending δύο εικόνων)
5. **Γεωμετρικές Μετασχηματισμοί**
   - degrees=0.4 : τυχαία περιστροφή ±0.4°
   - scale=0.5 : κλιμάκωση εικόνων στο 50–150%
   - shear=0.1 : τυχαία διατμητική στρέβλωση (shear) 10%

Early Stopping & Logging

- patience=30
  – Αν μετά από 30 συνεχόμενα epochs δεν βελτιωθεί η mAP, σταματά πρόωρα η εκπαίδευση.
- verbose=True
  – Εμφάνιση λεπτομερούς log ανά epoch (loss, mAP, LR κ.λπ.).

Επανεκπαίδευση (fine-tuning) με ελαφρώς διαφοροποιημένες ρυθμίσεις:

```python
from ultralytics import YOLO

if __name__ == "__main__":

    model = YOLO("E:\\Ptyxiakh_Project_Car_Parts_Detection\\runs\\segment\\train5\\weights\\last.pt")

    model.train(
        data="E:\Ptyxiakh_Project_Car_Parts_Detection\car_parts.yaml",
        epochs=300,
        patience=30,
        imgsz=832,
        batch=16,

        # Learning rate & scheduler
        lr0=0.001,
        lrf=0.01,
        cos_lr=True,

        # Optimizer & related hyperparams
        optimizer='AdamW',
        momentum=0.9,
        weight_decay=0.0005,

        # Augmentations
        degrees=45,
        translate=0.2,
        scale=0.5,
        shear=15,
        mosaic=1.0,
        mixup=0.3,
        device='cuda:0',
    )


    print("Inference completed. Check results in 'runs/predict/'")
```

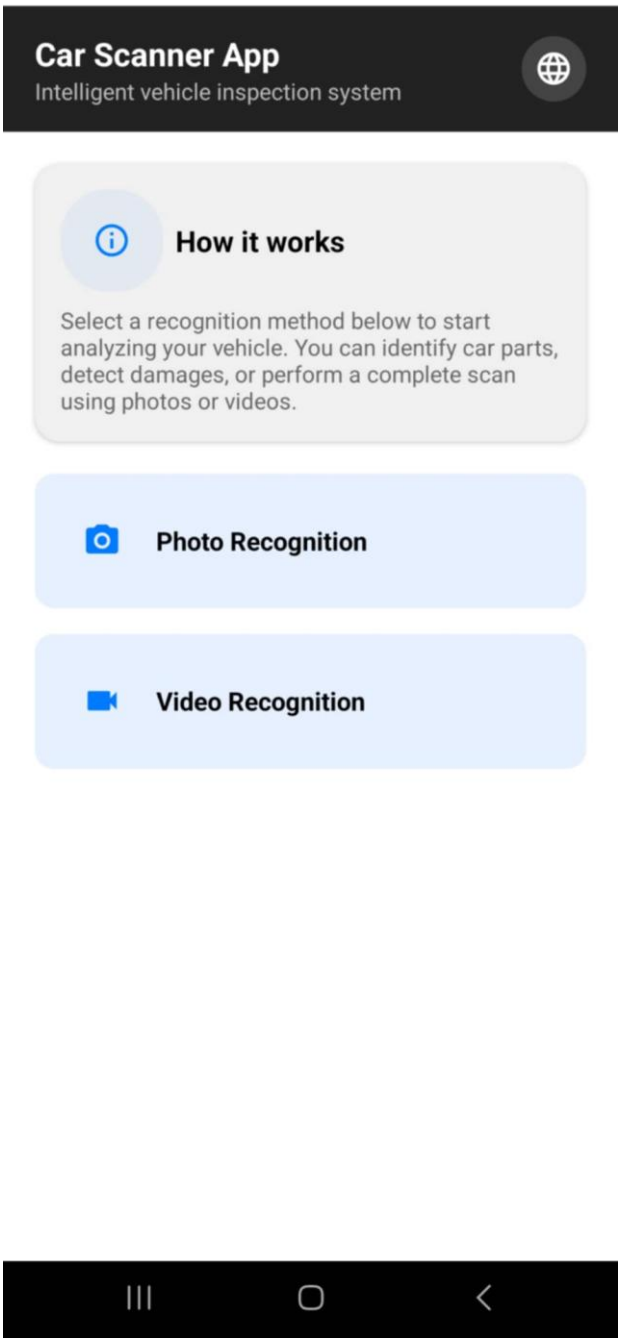
ΚΕΦΑΛΑΙΟ 4 Εφαρμογή Λήψης και Επεξεργασίας Εικόνων

(Περιγραφή της Εφαρμογής και του UI 4.1)

Στο πλαίσιο της πτυχιακής εργασίας, αναπτύχθηκε μια κινητή εφαρμογή με σκοπό την άμεση και φιλική προς τον χρήστη αναγνώριση μερών και ζημιών σε οχήματα. Η εφαρμογή λειτουργεί ως διεπαφή μεταξύ του χρήστη και των προεκπαιδευμένων μοντέλων βαθιάς μάθησης που υλοποιήθηκαν . Το γραφικό περιβάλλον της εφαρμογής έχει σχεδιαστεί με γνώμονα την ευκολία χρήσης και την αμεσότητα, έτσι ώστε ο τελικός χρήστης να μπορεί να πραγματοποιήσει σάρωση σε λίγα μόνο βήματα.

(Κεντρικό Μενού και Επιλογές)

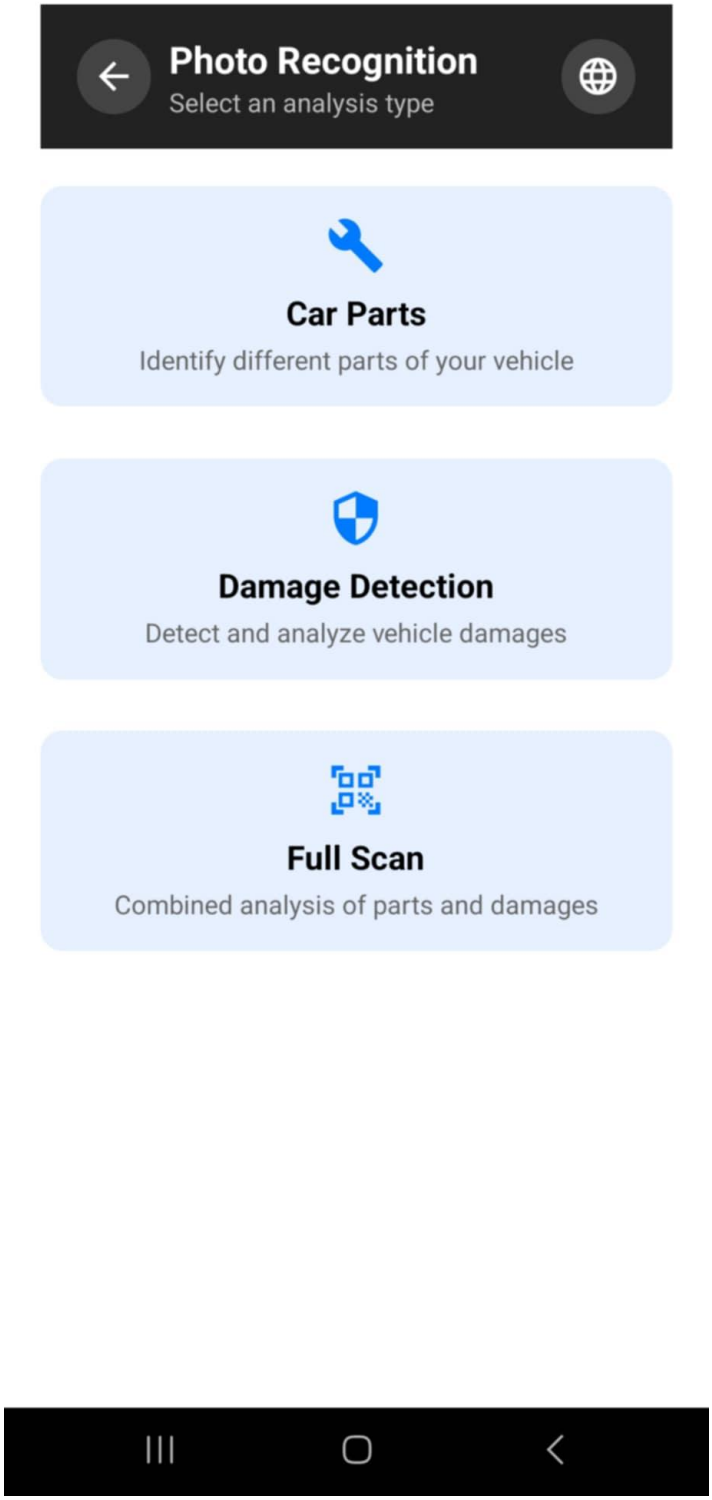
Κατά την εκκίνηση της εφαρμογής, ο χρήστης οδηγείται σε μια αρχική οθόνη όπου του παρέχεται η επιλογή μεταξύ ανάλυσης φωτογραφίας και ανάλυσης βίντεο. Αυτές οι δύο βασικές διαδρομές ορίζουν και τη λειτουργία του backend συστήματος, το οποίο θα περιγραφεί στις επόμενες ενότητες.



Μέσα σε κάθε κατηγορία (φωτογραφία ή βίντεο), ο χρήστης καλείται να επιλέξει ανάμεσα στις εξής τρεις λειτουργίες:

1. **Car Parts**
   Ανίχνευση των εξωτερικών μερών του οχήματος (π.χ. πόρτες, προφυλακτήρες, καπό, φανάρια, καθρέφτες, κ.ά.)
2. **Damage Detection**
   Εντοπισμός και ταξινόμηση των ζημιών (γρατζουνιές, βαθουλώματα, σπασμένα τζάμια, ρωγμές κ.λπ.)

3. **Full Scan**
   Συνδυασμένη λειτουργία όπου το σύστημα εντοπίζει ταυτόχρονα τόσο τα εξαρτήματα όσο και τις ζημιές και συσχετίζει τη ζημιά με το μέρος του οχήματος στο οποίο ανήκει.

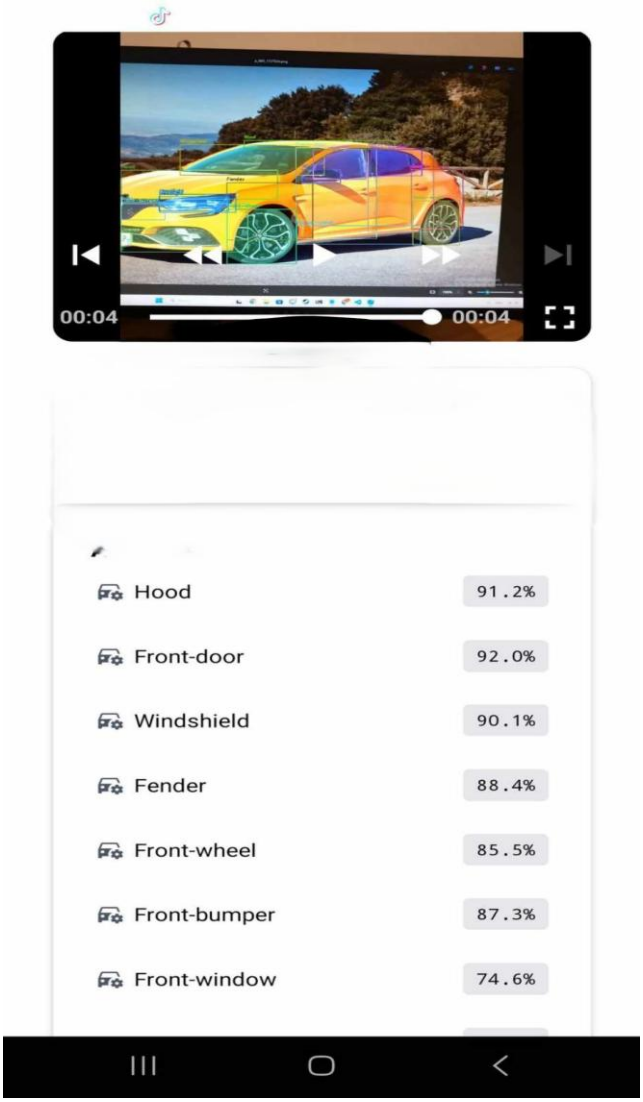


Αυτές οι λειτουργίες ενεργοποιούνται με το πάτημα ενός κουμπιού από τον χρήστη, και αμέσως η εφαρμογή προωθεί τα απαραίτητα δεδομένα στον server για ανάλυση.

(Παρουσίαση Αποτελεσμάτων)

Μετά την αποστολή της φωτογραφίας ή του βίντεο, ο χρήστης μεταφέρεται σε μια οθόνη προβολής αποτελεσμάτων. Εκεί εμφανίζεται η ίδια εικόνα που ανέβασε, με bounding boxes και masks στα αναγνωρισμένα σημεία ενδιαφέροντος. Κάθε στοιχείο συνοδεύεται από ένα ποσοστό εμπιστοσύνης (confidence score) που υποδηλώνει τη σιγουριά του μοντέλου για τη σωστή κατηγοριοποίηση.

Στην περίπτωση της πλήρους σάρωσης (Full Scan), το σύστημα συσχετίζει τη ζημιά με το εξάρτημα στο οποίο εμφανίζεται (π.χ. "scratch on front door – 91.2%"), κάτι που ενισχύει σημαντικά τη χρηστικότητα της εφαρμογής.



(Backend Εφαρμογής 4.2)

Στην καρδιά της εφαρμογής βρίσκεται το αρχείο **_layout.tsx**, το οποίο λειτουργεί ως το κεντρικό "κέλυφος" της εφαρμογής. Σε React Native εφαρμογές που βασίζονται στο **expo-router**, το αρχείο αυτό είναι υπεύθυνο για τη διαχείριση της πλοήγησης και της εμφάνισης όλων των σελίδων (routes).

```tsx
// app/_layout.tsx
import React from 'react';
import { Slot } from 'expo-router';
import { LanguageProvider } from '@/contexts/LanguageContext';
import { View, StyleSheet } from 'react-native';

export default function RootLayout() {
  return (
    <LanguageProvider>
      <View style={styles.container}>
        <Slot />
      </View>
    </LanguageProvider>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
  },
});
```

- Το αρχείο _layout.tsx είναι υπεύθυνο για το global layout της εφαρμογής.
- Υποστηρίζει πολυγλωσσικότητα μέσω LanguageProvider.
- Φορτώνει δυναμικά τις σελίδες με βάση το route (Slot).
- Ορίζει ενιαία αισθητική και δομή για όλη την εφαρμογή.

Η αρχική σελίδα της εφαρμογής είναι υλοποιημένη στο αρχείο **index.tsx**, και αποτελεί το πρώτο σημείο αλληλεπίδρασης του χρήστη με το σύστημα. Από εδώ ο χρήστης επιλέγει τη μέθοδο αναγνώρισης (φωτογραφία ή βίντεο), καθώς και μαθαίνει πώς λειτουργεί το σύστημα μέσω μιας συνοπτικής περιγραφής.

```
return (
  <PageTransition>
    <View style={styles.container}>
      <Header title={t("app.title")} subtitle={t("app.subtitle")} />
      <ScrollView contentContainerStyle={styles.content}>
        <Animated.View style={styles.card} entering={FadeInDown.delay(200).springify()}>
          <View style={styles.cardHeader}>
            <View style={styles.iconContainer}>
              <IconButton icon="information-outline" size={24} iconColor="#007AFF" />
            </View>
            <Text style={styles.title}>{t("how.works")}</Text>
          </View>
          <Text style={styles.description}>{t("how.works.description")}</Text>
        </Animated.View>

        <Animated.View entering={FadeInUp.delay(400).springify()}>
          <TouchableOpacity
            style={styles.optionCard}
            onPress={() => navigation.navigate("PhotoRecognition" as never)}
          >
            <IconButton icon="camera" size={24} iconColor="#007AFF" />
            <Text style={styles.optionTitle}>{t("photo.recognition")}</Text>
          </TouchableOpacity>
        </Animated.View>

        <Animated.View entering={FadeInUp.delay(600).springify()}>
          <TouchableOpacity
            style={styles.optionCard}
            onPress={() => navigation.navigate("VideoRecognition" as never)}
          >
            <IconButton icon="video" size={24} iconColor="#007AFF" />
            <Text style={styles.optionTitle}>{t("video.recognition")}</Text>
          </TouchableOpacity>
        </Animated.View>
      </ScrollView>
    </View>
  </PageTransition>
```

Η σελίδα index.tsx:

- Παίζει ρόλο κεντρικού hub επιλογών
- Παρέχει φιλικό και καθοδηγούμενο UI
- Συνδυάζει πλοήγηση, μεταφράσεις και animation
- Είναι σημείο εκκίνησης για όλες τις βασικές λειτουργίες της εφαρμογής

```
return (
  <PageTransition>
    <View style={styles.container}>
      <Header title={t("photo.recognition")} subtitle={t("select.analysis.type")} showBackButton />

      <View style={styles.cardContainer}>
        <TouchableOpacity style={styles.card} onPress={() => router.push("/PhotoAnalysis?type=parts")}>
          <MaterialIcons name="build" size={32} color="#007AFF" />
          <Text style={styles.cardTitle}>{t("car.parts")}</Text>
          <Text style={styles.cardDescription}>{t("car.parts.description")}</Text>
        </TouchableOpacity>

        <TouchableOpacity style={styles.card} onPress={() => router.push("/PhotoAnalysis?type=damage")}>
          <MaterialIcons name="security" size={32} color="#007AFF" />
          <Text style={styles.cardTitle}>{t("damage.detection")}</Text>
          <Text style={styles.cardDescription}>{t("damage.detection.description")}</Text>
        </TouchableOpacity>

        <TouchableOpacity style={styles.card} onPress={() => router.push("/PhotoAnalysis?type=full")}>
          <MaterialIcons name="qr-code-scanner" size={32} color="#007AFF" />
          <Text style={styles.cardTitle}>{t("full.scan")}</Text>
          <Text style={styles.cardDescription}>{t("full.scan.description")}</Text>
        </TouchableOpacity>
      </View>
    </View>
  </PageTransition>
);
```

Η οθόνη **PhotoRecognition.tsx** λειτουργεί ως σημείο επιλογής για τον χρήστη, από όπου μπορεί να επιλέξει τι είδους ανάλυση εικόνας επιθυμεί να εκτελέσει. Με απλό και κατανοητό τρόπο, παρουσιάζει τρεις εναλλακτικές δυνατότητες:

1. Αναγνώριση Μερών Αυτοκινήτου (Car Parts Detection)
2. Αναγνώριση Ζημιών Αυτοκινήτου (Damage Detection)
3. Πλήρης Σάρωση (Full Scan) – συνδυασμός των παραπάνω

- Λειτουργεί ως μενού επιλογής λειτουργίας αναγνώρισης
- Παρέχει σύνδεση με το component ανάλυσης PhotoAnalysis.tsx
- Ενσωματώνει πλοήγηση και μετάφραση περιεχομένου
- Υποστηρίζει ομαλή εμπειρία χρήστη με animation και responsive σχεδίαση

Η οθόνη **PhotoAnalysis.tsx** είναι υπεύθυνη για την επεξεργασία και αποστολή εικόνας στον server, την λήψη των αποτελεσμάτων ανίχνευσης και την απεικόνιση annotated εικόνας και προβλέψεων

στον χρήστη. Περιλαμβάνει επίσης τη διαχείριση loading state και πιθανών σφαλμάτων.

Τι περιλαμβάνει η λογική της σελίδας;

1. Ανάγνωση παραμέτρου type από το URL (μέσω useLocalSearchParams)
    - Υποστηρίζονται: parts, damage, full
2. Καταγραφή και εμφάνιση της επιλεγμένης εικόνας
3. Κατασκευή FormData και αποστολή στον server
4. Λήψη των αποτελεσμάτων και προβολή:
    - Annotated εικόνα
    - Λίστα προβλέψεων (detections)
5. Εκκαθάριση (reset) ανάλυσης

```
const params = useLocalSearchParams();
const type: AnalysisType = (Array.isArray(params.type) ? params.type[0] : params.type || "parts") as AnalysisType;
```

Ανάλογα με το route (?type=damage), η σελίδα προσαρμόζεται στο είδος ανάλυσης.

```
const [selectedImage, setSelectedImage] = useState<string | null>(null);
const [detections, setDetections] = useState<Detection[]>([]);
const [annotatedImage, setAnnotatedImage] = useState<string | null>(null);
```

Τα useState χρησιμοποιούνται για να διαχειριστούν την επιλεγμένη εικόνα, τα αποτελέσματα και την εικόνα με τις ανιχνεύσεις.

```
console.log(`📡 Connecting to API at: ${API_URL}`);
const formData = new FormData();
formData.append("file", {
  uri: selectedImage,
  name: "upload.jpg",
  type: "image/jpeg",
} as any);
formData.append("analysis_type", ANALYSIS_TYPES[type]);

console.log("📤 Sending FormData to:", API_URL);

const response = await fetch(API_URL, {
  method: "POST",
  body: formData,
});
```

Εδώ δημιουργείται και αποστέλλεται ένα multipart POST αίτημα προς το endpoint /detect/, περιλαμβάνοντας:

- Την εικόνα
- Τον τύπο ανάλυσης

```
const data = await response.json();
setDetections(Array.isArray(data.detections) ? data.detections : []);
setAnnotatedImage(data.annotated_image || null);
catch (error) {
console.error("🚨 Error analyzing image:", error);
Alert.alert("Error", "Scanning failed. Please try again.");
```

Αποθηκεύει:

- Τα αποτελέσματα (detections)
- Την annotated εικόνα σε μορφή Base64
- Σε περίπτωση αποτυχίας εμφανίζει κατάλληλο μήνυμα

```
{annotatedImage && (
  <View style={styles.annotatedContainer}>
    <Text style={styles.resultsTitle}>Annotated Image:</Text>
    <Image
      source={{ uri: `${annotatedImage}` }}
      style={styles.annotatedImage}
    />
  </View>
)}
```

Αν έχει επιστραφεί εικόνα από τον server, εμφανίζεται κάτω από τα κουμπιά ανάλυσης.

```
{detections.length > 0 && (
  <ResultsDisplay
    detections={detections}
    analysisType={mapToUIAnalysisType[ANALYSIS_TYPES[type]]}
  />
)}
```

Χρησιμοποιείται το component ResultsDisplay για να εμφανιστεί λίστα με confidence και ετικέτες.

Παρόμοια λειτουργια με την PhotoRecognition λειτουργεί και η VideoRecognition αλλά εδώ η διαδικασία αφορά επεξεργασία πολλαπλών καρέ (frames) μέσω server-side video analysis.

```
return (
  <PageTransition>
    <View style={styles.container}>
      <Header title={t("video.recognition")} subtitle={t("select.analysis.type")} showBackButton />

      <View style={styles.cardContainer}>
        <TouchableOpacity style={styles.card} onPress={() => router.push("/VideoAnalysis?type=parts")}>
          <MaterialIcons name="build" size={32} color="#007AFF" />
          <Text style={styles.cardTitle}>{t("car.parts")}</Text>
          <Text style={styles.cardDescription}>{t("car.parts.description")}</Text>
        </TouchableOpacity>

        <TouchableOpacity style={styles.card} onPress={() => router.push("/VideoAnalysis?type=damage")}>
          <MaterialIcons name="security" size={32} color="#007AFF" />
          <Text style={styles.cardTitle}>{t("damage.detection")}</Text>
          <Text style={styles.cardDescription}>{t("damage.detection.description")}</Text>
        </TouchableOpacity>

        <TouchableOpacity style={styles.card} onPress={() => router.push("/VideoAnalysis?type=full")}>
          <MaterialIcons name="qr-code-scanner" size={32} color="#007AFF" />
          <Text style={styles.cardTitle}>{t("full.scan")}</Text>
          <Text style={styles.cardDescription}>{t("full.scan.description")}</Text>
        </TouchableOpacity>
      </View>
    </View>
  </PageTransition>
);
```

Η οθόνη VideoAnalysis.tsx είναι υπεύθυνη για την ανάλυση αρχείων βίντεο. Ο χρήστης μπορεί να καταγράψει ή να επιλέξει ένα αρχείο .mp4, να το αποστείλει στον server, και στη συνέχεια να δει τα αποτελέσματα ανίχνευσης ζημιών ή μερών του αυτοκινήτου ανά frame, μαζί με ένα νέο βίντεο που περιλαμβάνει τις επισημάνσεις.

Τι περιλαμβάνει η λειτουργία

1. Καθορισμός τύπου ανάλυσης (parts, damage, full)
2. Λήψη ή επιλογή αρχείου βίντεο
3. Αποστολή στον server μέσω του API /detect_video/
4. Ανάλυση και λήψη αποτελεσμάτων
5. Προβολή επεξεργασμένου (annotated) βίντεο
6. Εμφάνιση ποσοστών ακρίβειας ανά κατηγορία (average confidence)

```
const type: AnalysisType = (Array.isArray(params.type)
  ? params.type[0]
  : params.type || "parts") as AnalysisType;
```

Ελέγχει ποιο είδος επεξεργασίας έχει επιλέξει ο χρήστης (μέρη, ζημιές, πλήρης σάρωση).

```
const formData = new FormData();
formData.append("file", {
  uri: selectedVideo,
  name: "video.mp4",
  type: "video/mp4",
} as any);
formData.append("analysis_type", type);

const response = await fetch(`${API_URL}/detect_video/`, {
  method: "POST",
  body: formData,
  headers: {
    "Content-Type": "multipart/form-data",
  },
});
```

Το βίντεο και ο τύπος ανάλυσης αποστέλλονται στον server. Το backend (FastAPI) επεξεργάζεται κάθε καρέ ξεχωριστά και επιστρέφει:

- Πίνακα detections με ζημιές/μέρη
- Τιμές confidence
- URL του επεξεργασμένου βίντεο

Συνοπτικά

- Χρήση ScrollView για υποστήριξη μικρών οθονών
- Καρτέλες με πληροφορίες
- Κουμπιά με ActivityIndicator για φόρτωση
- Εντοπισμός σφαλμάτων (try/catch) και ειδοποιήσεις με Alert.alert()

Η VideoAnalysis.tsx:

- Διαχειρίζεται την ολόκληρη ροή αποστολής βίντεο και προβολής ανάλυσης
- Παρέχει feedback στον χρήστη σε κάθε στάδιο
- Προβάλλει τόσο το επεξεργασμένο βίντεο όσο και αναλυτικά αποτελέσματα
- Συνδέεται άμεσα με το FastAPI backend μέσω του endpoint /detect_video/

Στην συνέχεια θα αναφερθούμε στα components της εφαρμογης

Το αρχείο Header.tsx είναι ένα πολυλειτουργικό component που χρησιμοποιείται σε όλη την εφαρμογή για την παρουσίαση: αποτελεί την ενιαία κεφαλίδα (header bar) της εφαρμογής και χρησιμοποιείται σε όλες τις κύριες οθόνες. Είναι υπεύθυνο για την προβολή του τίτλου κάθε σελίδας, ενός σύντομου υποτίτλου, καθώς και δύο κύριων λειτουργιών:

1. Πλοήγηση προς τα πίσω (custom “Back” button)
2. Αλλαγή γλώσσας εφαρμογής (μέσω context)

```
interface HeaderProps {
  title: string;
  subtitle?: string;
  showBackButton?: boolean;
}
```

title: Τίτλος της σελίδας (υποχρεωτικό)
subtitle: Προαιρετικός υπότιτλος
showBackButton: Αν θα εμφανιστεί κουμπί "επιστροφή"

```
const goBack = () => {
  if (pathname.includes("/PhotoAnalysis") || pathname.includes("/VideoAnalysis")) {
    router.push(pathname.includes("/PhotoAnalysis") ? "/PhotoRecognition" : "/VideoRecognition");
  } else if (pathname === "/PhotoRecognition" || pathname === "/VideoRecognition") {
    router.push("/");
  } else {
    router.back();
  }
};
```

Αντί να χρησιμοποιεί απλά router.back(), γίνεται λογική πλοήγηση βάσει path, προσφέροντας έξυπνη και ελεγχόμενη επιστροφή στα σωστά προηγούμενα βήματα (ανάλογα με το context: ανάλυση φωτογραφίας ή βίντεο).

```
const toggleLanguage = () => {
  setLanguage(language === "en" ? "el" : "en");
};
```

Με ένα tap στο εικονίδιο “language”, αλλάζει η γλώσσα της εφαρμογής μέσω LanguageContext, επιτρέποντας δυναμική διεθνοποίηση του UI χωρίς επανεκκίνηση.

Συνοπτικά
Το **Header.tsx**:

- Βελτιώνει την πλοήγηση και την κατανόηση του χρήστη για την τρέχουσα λειτουργία
- Ενσωματώνει διεθνοποίηση σε επίπεδο UI
- Είναι επανεκκινήσιμο και ελαφρύ, με πλήρως παραμετροποιήσιμη χρήση
- Χρησιμοποιείται ευρέως στις σελίδες: PhotoRecognition, VideoRecognition, PhotoAnalysis, VideoAnalysis

Το component **MediaCapture.tsx** αποτελεί ένα επαναχρησιμοποιήσιμο στοιχείο που επιτρέπει στον χρήστη να:

- Τραβήξει φωτογραφία ή βίντεο από την κάμερα
- Επιλέξει υπάρχον αρχείο από τη συλλογή
- Προβάλλει προεπισκόπηση του μέσου που κατέγραψε
- Καθαρίσει την επιλογή και να επαναλάβει

Κύρια λειτουργικά σημεία
Permissions
Κατά την εκκίνηση, το component ζητά:

- Πρόσβαση στην κάμερα
- Πρόσβαση στο μικρόφωνο (για βίντεο)
- Πρόσβαση στη συλλογή αρχείων (MediaLibrary)

```
const [cameraPermission, requestCameraPermission] = useCameraPermissions();
const [microphonePermission, requestMicrophonePermission] = useMicrophonePermissions();
```

Ο χρήστης μπορεί να ανοίξει τον image/video picker:

```
const handlePickMedia = async () => {
  let result: ImagePicker.ImagePickerResult;
  if (mediaType === "photo") {
    result = await ImagePicker.launchImageLibraryAsync({ mediaTypes: ImagePicker.MediaTypeOptions.Images });
  } else {
    result = await ImagePicker.launchImageLibraryAsync({ mediaTypes: ImagePicker.MediaTypeOptions.Videos });
  }
  if (!result.canceled && result.assets?.length) {
    onCapture({ uri: result.assets[0].uri });
  }
};
```

Η επιλεγμένη εικόνα/βίντεο περνά μέσω onCapture στον γονικό component.

Κουμπιά UI

Η κάμερα εμφανίζει τρία βασικά κουμπιά:

1. Επιλογή αρχείου από φάκελο
2. Ενεργοποίηση λήψης
3. Εναλλαγή κάμερας (μπροστινή/πίσω)

Μετά τη λήψη:

- Ο χρήστης βλέπει Image ή Video preview
- Μπορεί να κάνει "Reset"

Το component παίρνει τα εξής props:

Prop	**Περιγραφή**
mediaType	"photo" ή "video"
onCapture	Συνάρτηση επιστροφής με το URI του αρχείου
onClear	Καθαρίζει την προηγούμενη επιλογή
capturedMedia	Το τρέχον URI του μέσου
isLoading	Εμφάνιση φόρτωσης (spinner)

Χάρη σε αυτή τη δομή, χρησιμοποιείται τόσο στο PhotoAnalysis.tsx όσο και στο VideoAnalysis.tsx, με διαφορετική παραμετροποίηση.

Το **OptionCard.tsx** είναι ένα επαναχρησιμοποιήσιμο στοιχείο διεπαφής χρήστη (UI), που έχει σχεδιαστεί για να παρουσιάζει μία επιλογή ή λειτουργία με συνοδευτικό εικονίδιο, τίτλο και σύντομη περιγραφή. Χρησιμοποιείται κυρίως για την εμφάνιση των διαθέσιμων λειτουργιών ανάλυσης (π.χ. "Αναγνώριση Ζημιών", "Αναγνώριση Μερών").

```tsx
interface OptionCardProps {
  icon: React.ReactNode;
  titleKey: string;
  descriptionKey: string;
  onClick: () => void;
  className?: string;
}

const OptionCard: React.FC<OptionCardProps> = ({
  icon,
  titleKey,
  descriptionKey,
  onClick,
  className,
}) => {
  return (
    <TouchableOpacity style={styles.card} onPress={onClick}>
      <View style={styles.iconContainer}>{icon}</View>
      <View style={styles.textContainer}>
        <Text style={styles.title}>{titleKey}</Text>
        <Text style={styles.description}>{descriptionKey}</Text>
      </View>
    </TouchableOpacity>
  );
};
```

Το OptionCard.tsx:

- Απλοποιεί τη δημιουργία UI καρτών επιλογής
- Προσφέρει ομοιομορφία σχεδίασης και ευκολία στην επεκτασιμότητα
- Ενισχύει την εμπειρία χρήστη με ορατές, σαφείς και προσβάσιμες επιλογές

Το **PageTransition.tsx** είναι ένα βοηθητικό component που προσθέτει fade animation κατά την είσοδο και έξοδο κάθε σελίδας. Χρησιμοποιείται ως wrapper γύρω από την εκάστοτε οθόνη (View) για να προσφέρει οπτική συνέπεια και ευχάριστη εμπειρία χρήστη (UX).

```tsx
interface PageTransitionProps {
  children: ReactNode;
}

const PageTransition: React.FC<PageTransitionProps> = ({ children }) => {
  return (
    <Animated.View
      entering={FadeIn.duration(300)}
      exiting={FadeOut.duration(200)}
      style={styles.container}
    >
      {children}
    </Animated.View>
  );
};
```

Το **ResultsDisplay.tsx** αποτελεί το component που εμφανίζει στον χρήστη τα αποτελέσματα που επιστρέφει ο server από την ανάλυση εικόνας ή βίντεο. Συνδυάζει:

- Annotated preview (image/video)
- Λίστα ανιχνεύσεων με confidence
- Πίνακα μέσης ακρίβειας (όταν υπάρχει)

Το component δέχεται ως είσοδο:

	Περιγραφή:
detections	Πίνακας αντικειμένων με part, damage, confidence, display_text
annotatedImage	URI για base64 εικόνα με annotation
annotatedVideo	URI για processed βίντεο που επιστράφηκε από τον server
average_confidence	Αντικείμενο {label: confidenceValue}
analysisType	"parts", "damage", ή "full"

Το annotated αποτέλεσμα είναι πλήρως ενσωματωμένο στο UI, επιτρέποντας άμεση οπτική αξιολόγηση από τον χρήστη.

```
<View style={styles.container}>
  {(annotatedImage || annotatedVideo) && (
    <View style={styles.previewWrapper}>
      {annotatedImage ? (
        <Image source={{ uri: annotatedImage }} style={styles.image} resizeMode="contain" />
      ) : (
        <Video
          source={{ uri: annotatedVideo! }}
          style={styles.video}
          controls
          resizeMode="contain"
          onError={(e) => console.error("🎥 Video Error", e)}
        />
      )}
    </View>
```

Χρησιμοποιείται FlatList για αποδοτική απόδοση:

Η κάθε γραμμή περιλαμβάνει:

- Εικονίδιο ανάλογα με το είδος (ζημιά ή μέρος)
- Ετικέτα (π.χ. “scratch on door” ή “Mirror”)
- Confidence score (π.χ. 89.3%)

```
<FlatList
  data={detections}
  keyExtractor={(_, index) => index.toString()}
  renderItem={({ item }) => (
    <View
      style={[
        styles.detectionItem,
        item.damage ? styles.damageItem : styles.partItem,
      ]}
    >
      <View style={styles.detectionLabel}>
        <Icon
          name={item.damage ? "alert-circle" : "check-circle"}
          size={18}
          color={item.damage ? "#EF4444" : "#22C55E"}
        />
        <Text style={styles.detectionText}>
          {item.display_text ||
            (item.part && item.damage
              ? `${item.part} - ${item.damage}`
              : item.part || item.damage || "Unknown")}
        </Text>
      </View>
      <Text style={styles.confidence}>{formatConfidence(item.confidence)}</Text>
    </View>
  )}
```

Μέσος Όρος Ακρίβειας (για "parts")

Αν υπάρχει το prop average_confidence, εμφανίζεται πίνακας με όλες τις κατηγορίες και την αντίστοιχη μέση ακρίβεια που παρατηρήθηκε στο βίντεο:

```
{analysisType === "parts" && average_confidence && Object.keys(average_confidence).length > 0 && (
  <View style={styles.resultCard}>
    <Text style={styles.headerText}>Average Confidence:</Text>
    {Object.entries(average_confidence).map(([label, val]) => (
      <View key={label} style={styles.detectionItem}>
        <View style={styles.detectionLabel}>
          <Icon name="car-cog" size={18} color="#4B5563" />
          <Text style={styles.detectionText}>{label}</Text>
        </View>
        <Text style={styles.confidence}>{(val * 100).toFixed(1)}%</Text>
      </View>
    ))}
  </View>
)}
```

Συνοπτικά
Το ResultsDisplay.tsx:

- Παρουσιάζει καθαρά και οπτικά τα αποτελέσματα εντοπισμού
- Ενισχύει την εμπιστοσύνη του χρήστη, εμφανίζοντας την ακρίβεια (confidence)
- Υποστηρίζει και εικόνα και βίντεο με ένα ενιαίο API
- Δίνει πλήρη επισκόπηση στο τελικό στάδιο του pipeline

(Επικοινωνία Εφαρμογής με τον Server & Ανάλυση των API Endpoints 4.3)

Η εφαρμογή επικοινωνεί με έναν server υλοποιημένο σε **FastAPI**, ο οποίος είναι υπεύθυνος για την αποδοχή εικόνων ή βίντεο , την επεξεργασία τους και την επιστροφή δομημένων αποτελεσμάτων προς τον χρήστη.

Δομή και Τεχνολογίες

- **FastAPI**: REST API
- **OpenCV / MoviePy**: Επεξεργασία εικόνων και βίντεο
- **PIL / NumPy / base64**: Διαχείριση εικόνων σε memory buffers
- **CORS + ORJSON**: Υποστήριξη front–end επικοινωνίας & γρήγορες αποκρίσεις JSON

Ο server εκκινεί ως FastAPI εφαρμογή:

```
app = FastAPI(default_response_class=ORJSONResponse)
```

Χρησιμοποιεί την ORJSON για γρηγορότερες αποκρίσεις και υποστηρίζει **CORS middleware** για να επιτρέπει την επικοινωνία από κινητές συσκευές:

```
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],  # or restrict to your app IPs later
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

Κατά την εκκίνηση, φορτώνονται δύο διαφορετικά μοντέλα YOLOv11:

- Car_Parts_Detection_best.pt → για εντοπισμό μερών
- Damage_detection_best.pt → για αναγνώριση ζημιών

```
model_parts = YOLO(MODEL_PATH_PARTS)
model_damage = YOLO(MODEL_PATH_DAMAGE)
```

Η μεταβλητή colors = Colors() χρησιμοποιείται για να αποδώσει συγκεκριμένα χρώματα σε κάθε κατηγορία στην annotated εικόνα ή βίντεο.

Το αρχείο περιλαμβάνει πολλές κρίσιμες βοηθητικές συναρτήσεις:
process_masks()

```
def process_masks(masks, orig_shape):
    """Properly handle YOLO Mask objects and resize to original dimensions"""
    if masks is None:
        return None

    processed = []
    # Convert masks tensor to numpy array and transpose dimensions
    masks_np = masks.data.cpu().numpy().transpose(1, 2, 0)  # (N, H, W) → (H, W, N)

    for i in range(masks_np.shape[-1]):
        # Extract individual mask and ensure 2D format
        mask = masks_np[:, :, i].squeeze().astype(np.float32)

        # Skip invalid empty masks
        if mask.size == 0:
            continue

        # Resize to original image dimensions
        mask_resized = cv2.resize(
            mask,
            (orig_shape[1], orig_shape[0]),  # (width, height)
            interpolation=cv2.INTER_LINEAR
        )

        # Binarize and convert to uint8
        processed.append((mask_resized > 0.5).astype(np.uint8))

    return processed
```

Αναλαμβάνει την επεξεργασία των segmentation masks από το YOLO και την επανακλιμάκωση τους στο μέγεθος της αρχικής εικόνας.

apply_mask()

```
def apply_mask(image, mask, color, alpha=0.4):
    """Safe mask application with dimension validation"""
    if mask is not None and mask.ndim == 2 and mask.shape[:2] == image.shape[:2]:
        colored_mask = np.zeros_like(image)
        colored_mask[mask > 0] = color
        return cv2.addWeighted(image, 1, colored_mask, alpha, 0)
    return image
```

Επικαλύπτει την εικόνα με μάσκα (χρωματισμένη) στο σημείο εντοπισμού.

iou()

```
def iou(box1, box2):
    """Calculate intersection over union"""
    x1 = max(box1[0], box2[0])
    y1 = max(box1[1], box2[1])
    x2 = min(box1[2], box2[2])
    y2 = min(box1[3], box2[3])

    inter_area = max(0, x2 - x1) * max(0, y2 - y1)
    box1_area = (box1[2] - box1[0]) * (box1[3] - box1[1])
    box2_area = (box2[2] - box2[0]) * (box2[3] - box2[1])

    return inter_area / (box1_area + box2_area - inter_area + 1e-6)
```

Υπολογίζει το Intersection over Union ανάμεσα σε δύο bounding boxes (χρησιμοποιείται για να συσχετίσει ζημιές με μέρη).

filter_best_detections_per_class()

```python
def filter_best_detections_per_class(boxes, classes, confs, keep_top_n_for=None):
    """
    Επιστρέφει λίστα από (index, class_id) με τα καλύτερα detections per class.
    Για συγκεκριμένες κατηγορίες (π.χ. headlight, mirror, fender) κρατάει top-N.
    """
    keep_top_n_for = keep_top_n_for or []

    class_detections = defaultdict(list)

    for i, cls in enumerate(classes):
        class_detections[cls].append((i, confs[i]))

    filtered_indices = []
    for cls, detections in class_detections.items():
        # Ταξινόμηση κατά confidence φθίνουσα
        detections.sort(key=lambda x: x[1], reverse=True)
        top_n = 2 if cls in keep_top_n_for else 1
        top = detections[:top_n]
        for idx, _ in top:
            filtered_indices.append((idx, cls))

    return filtered_indices
```

Επιστρέφει μόνο τις καλύτερες ανιχνεύσεις (π.χ. κρατάει 2 "Headlights" και 1 από κάθε άλλο μέρος).

Endpoint /detect/ – Ανάλυση Εικόνας

```python
@app.post("/detect/")
```

1. Διαβάζει το αρχείο εικόνας (UploadFile) και το μετατρέπει σε RGB PIL image
2. Μετατρέπει την εικόνα σε NumPy array για OpenCV επεξεργασία
3. Αναλαμβάνει την επεξεργασία **εικόνας** (JPEG), η οποία αποστέλλεται από την εφαρμογή. Ανάλογα με το analysis_type, χρησιμοποιεί το κατάλληλο μοντέλο:

analysis_type	**Περιγραφή**
car.parts.detection	Αναγνώριση μερών αυτοκινήτου
damage.detection	Αναγνώριση ζημιών
full.scan	Συνδυασμένη αναγνώριση μέρους & ζημιάς

Στο full.scan:

- Η εικόνα φορτώνεται και μετατρέπεται σε NumPy array
- Εκτελεί YOLO ανίχνευση πρώτα για μέρη και μετά για ζημιές
- Εφαρμόζει iou() για να συσχετίσει κάθε ζημιά με το μέρος όπου ανήκει
- Δημιουργεί τελικό annotation με cv2.rectangle, putText, και apply_mask
- Επιστρέφει αποτελέσματα + την annotated_image σε base64 εικόνα και JSON με ανιχνεύσεις

Response (full.scan):

```
{
  "analysis_type": "full_scan",
  "detections": [
    {
      "display_text": "scratch on door (87.4%)",
      "damage": "scratch",
      "part": "door",
      "confidence": 87.4,
      "location": { "x1": ..., "y1": ..., "x2": ..., "y2": ... }
    }
  ],
  "annotated_image": "<base64>"
}
```

/detect_video/ – Ανάλυση βίντεο

```
@app.post("/detect_video/")
```

Το βίντεο αποθηκεύεται προσωρινά τοπικά.
Αναλαμβάνει την ανάλυση ενός ολόκληρου βίντεο (π.χ. από κάμερα κινητού) καρέ–καρέ, για την αναγνώριση ζημιών ή μερών.
analysis_type επιλογές:

Τιμή	Λειτουργία
full	Συνδυασμένη ανίχνευση
parts	Μόνο μέρη
damage	Μόνο ζημιές

Οι ανιχνεύσεις αποθηκεύονται ανά καρέ
Χρησιμοποιείται cv2.VideoWriter για να παραχθεί νέο annotated βίντεο
Αποθηκεύεται προσωρινά στον φάκελο /static/videos και επιστρέφεται το URL

Επιπλέον:

- Υπολογίζεται το average_confidence ανά label
- Εξάγεται το Annotated video
- Η απάντηση είναι JSON με URL, ανιχνεύσεις και στατιστικά ακρίβειας

ΚΕΦΑΛΑΙΟ 5 Πειραματική Αποτίμηση

Για την εκπαίδευση και αξιολόγηση των μοντέλων χρησιμοποιήθηκαν δεδομένα από δύο αξιόπιστες και ανοιχτά διαθέσιμες πηγές:

1. CARD: Car Damage Dataset (USTC)

🔗 https://cardd-ustc.github.io/

Το dataset **CARD (Car Accident Recognition Dataset)** δημιουργήθηκε από το University of Science and Technology of China (USTC) και περιλαμβάνει εικόνες οχημάτων με **πολλαπλούς τύπους ζημιών**. Διατίθεται με επισημειωμένες περιοχές για:

- *Scratches*
- *Dents*
- *Cracks*
- *Lamp broken*
- *Tire flat*
- *Glass shatter*

Το dataset παρέχει καλή ποικιλομορφία σε φώτα, γωνίες λήψης και τύπους αυτοκινήτων.

Αποτελειται από 4.000 εικόνες ζημιών αυτοκινήτου υψηλής ανάλυσης με πάνω από 9.000 καλά annotated περιπτώσεις έξι κατηγοριών ζημιών

2. Humans in the Loop – Car Parts and Damages (Kaggle)

🔗 https://www.kaggle.com/datasets/humansintheloop/car-parts-and-car-damages/data

Αυτό το dataset περιλαμβάνει εικόνες οχημάτων με σημειωμένα μέρη και αντίστοιχες ζημιές. Κάθε εικόνα συνοδεύεται από JSON annotations, με segmentation masks για:

- *Car parts* (πόρτες, καπό, φώτα, καθρέφτες κ.ά.)
- *Damage types* (γρατζουνιές, βαθουλώματα, ρωγμές)

Το dataset προσφέρει υψηλή ποιότητα labeling και χρησιμοποιήθηκε και για την εκπαίδευση αλλά και για επικύρωση (validation) των μοντέλων.

Το σύνολο δεδομένων αποτελείται από συνολικά 1812 εικόνες, με πλήρη σχολιασμό με πολύγωνα είτε για ανταλλακτικά αυτοκινήτου (998 εικόνες) είτε για ζημιές αυτοκινήτου (814). Ο συνολικός αριθμός πολυγώνων στο σύνολο δεδομένων είναι 24.851.

Προεπεξεργασία & Επιμέλεια

Τα δεδομένα:

- Συνδυάστηκαν με custom εικόνες από διάφορες online πηγές και manually captured φωτογραφίες.
- Επεξεργάστηκαν με YOLO format conversion scripts, ώστε να είναι συμβατά με την Ultralytics πλατφόρμα.
- Έγινε data augmentation κατά την εκπαίδευση (μείξη εικόνων, περιστροφή, χρώμα, flip, mosaic κ.λπ.)

Στην παρούσα ενότητα παρουσιάζουμε την πειραματική αποτίμηση του μοντέλου ανίχνευσης και αναγνώρισης ζημιών σε οχήματα μέσω υπολογιστικής όρασης. Η αξιολόγηση βασίστηκε σε ένα σύνολο εικόνων δοκιμής που περιλάμβανε πολλαπλά οχήματα με διαφορετικού είδους ζημιές

Ας αρχίσουμε με το Car_Parts_Detection model

Οπτικά Αποτελέσματα

Παρακάτω παρατίθενται εικόνες από το σύνολο δοκιμής:

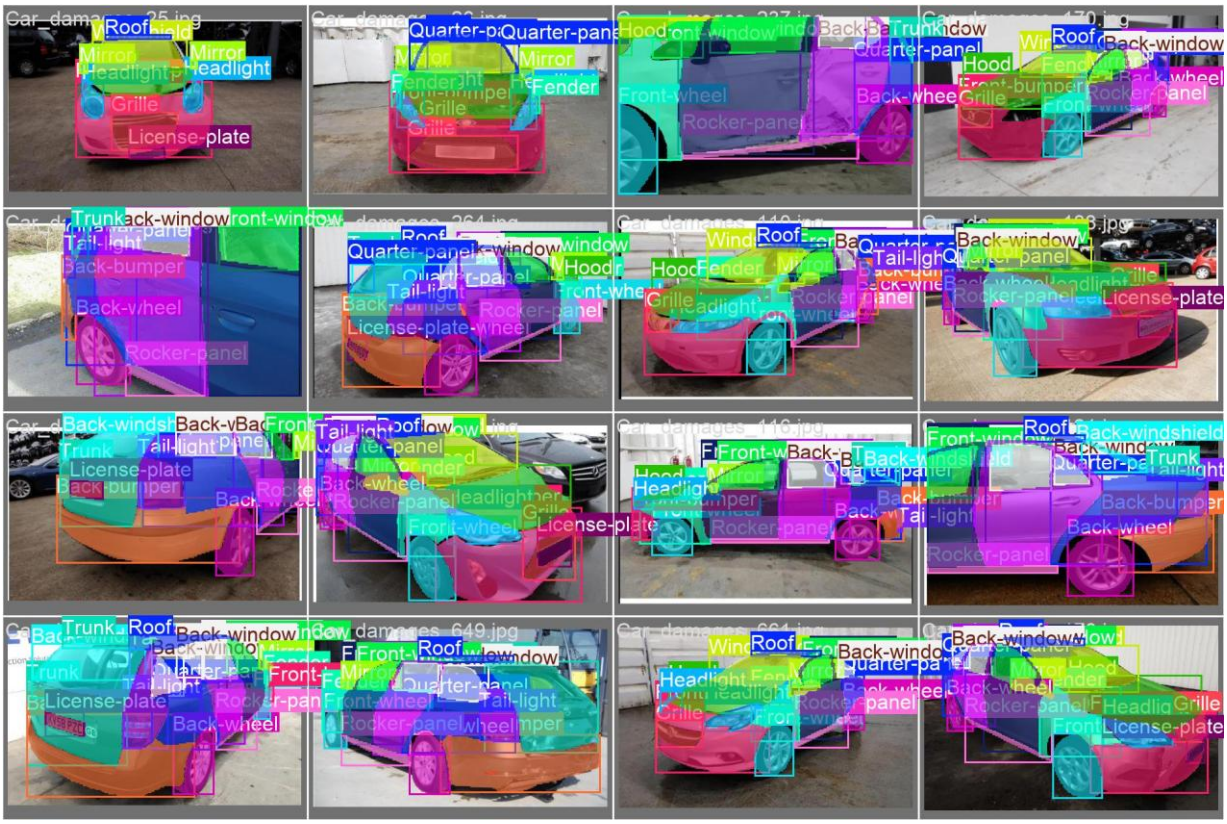
- **Valuation**: Εμφανίζονται τα πραγματικά περιγράμματα και οι ετικέτες των μερών του αυτοκινήτου.
- **Predictions**: Εμφανίζονται οι προβλέψεις του μοντέλου, συνοδευόμενες από τις πιθανότητες εντοπισμού (confidence scores).

Valuation

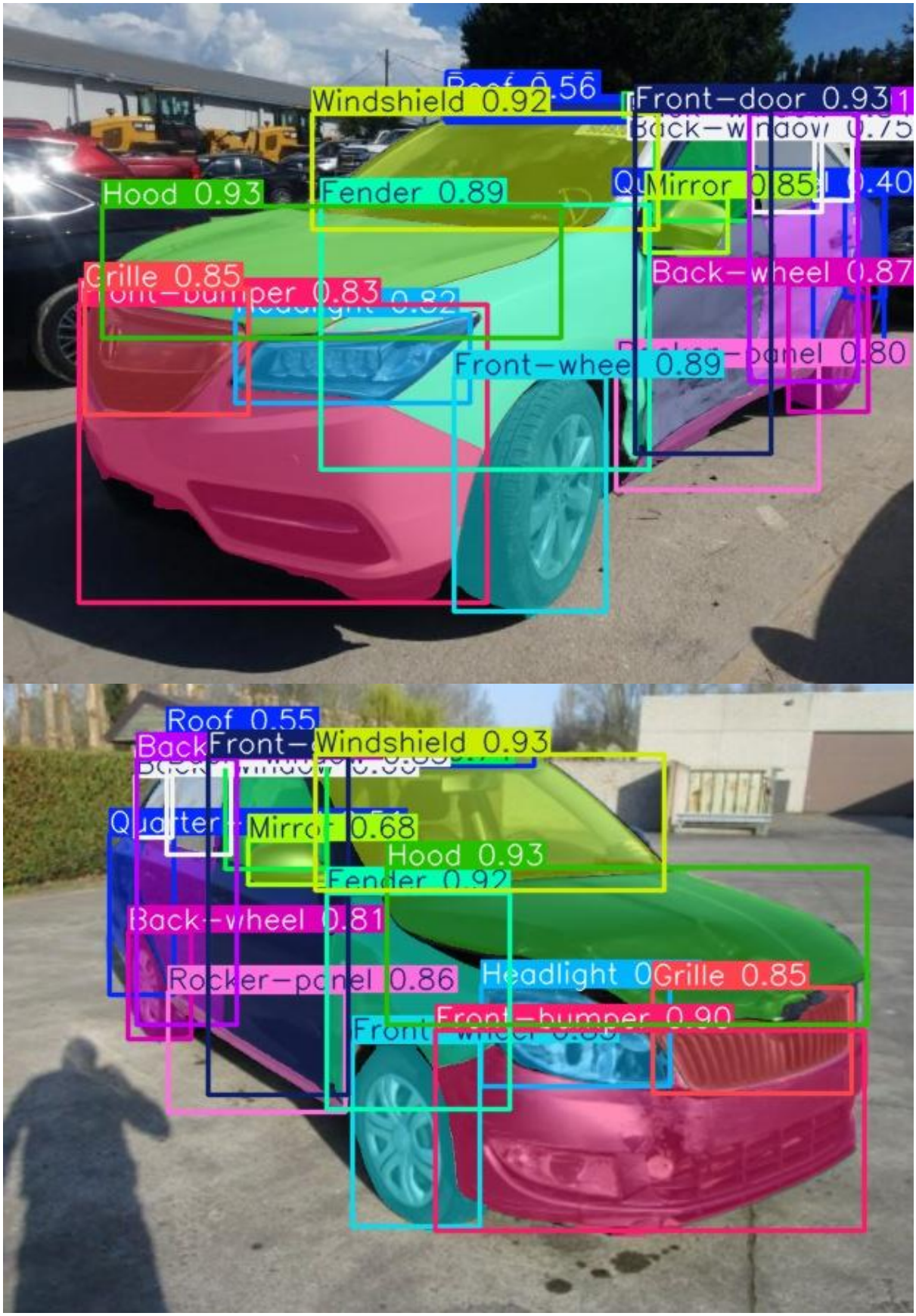
Predictions

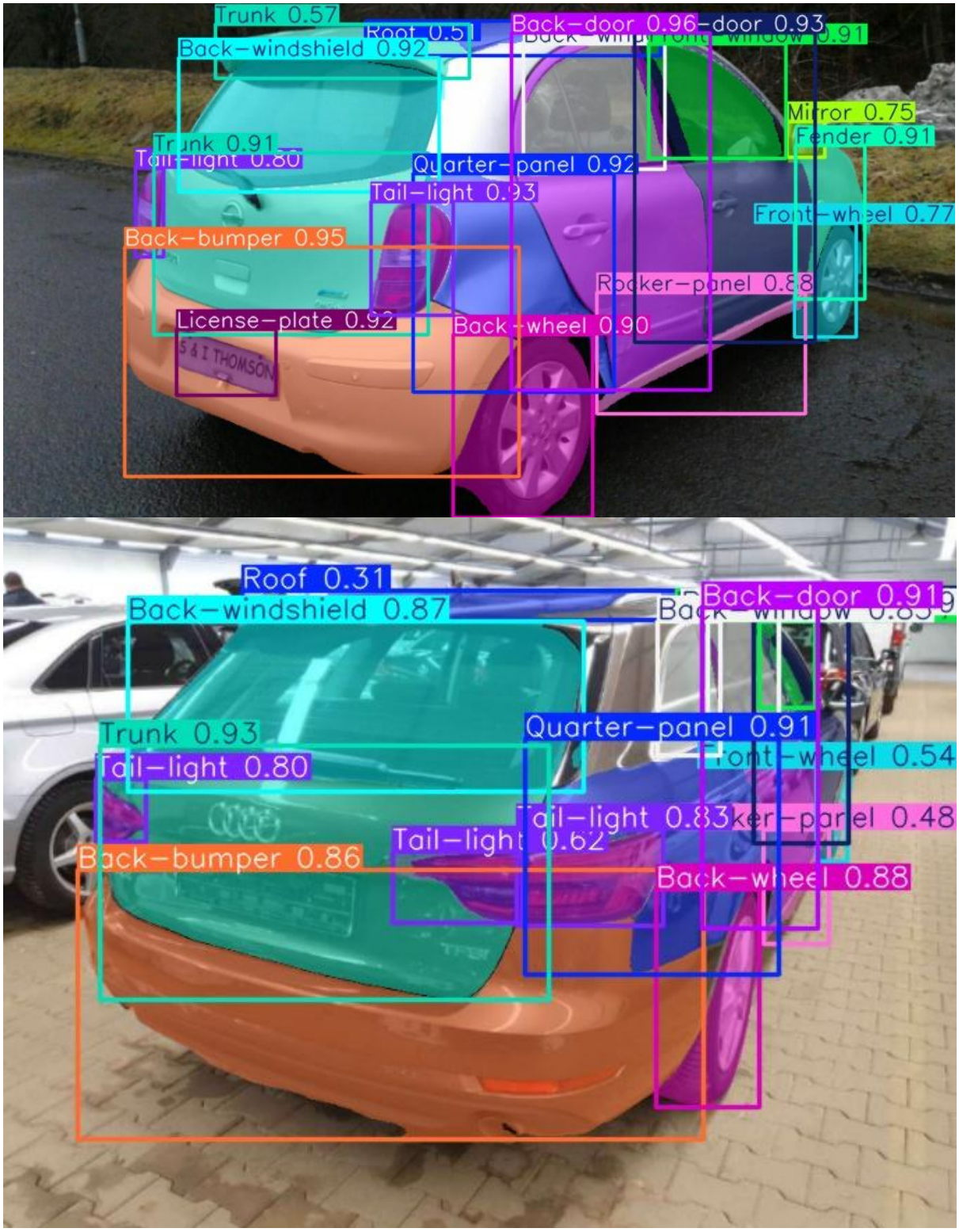

Car_damages_129.jpg
Front-window 0.7
Headlight 0.8
Back-wheel 0.9
Rocker-panel 0.8
Front-window 0.8
Back-window 0.9
Quarter-panel 0.9
Back-wheel 0.9
Rocker-panel 0.7
Tail-light 0.9
Back-window 0.8
Back-bumper 0.9
Back-wheel 0.9
Rocker-panel 0.7
Grille 0.8
License-plate 0.7
Roof 0.6
Back-window 0.8
Hood 0.7
Back-wheel 0.9
Front-bumper 0.8
Back-windshield 0.9
Tail-light 0.8
Quarter-panel 0.9
Back-bumper 0.9
Rocker-panel 0.9
Back-wheel 0.9
Windshield 0.7
Front-window 0.8
Hood 0.9
Back-wheel 0.8
Grille 0.8
Front-wheel 0.9
Front-window 0.7
Tail-light 0.8
Grille 0.9
License-plate 0.6
Rocker-panel 0.8
Front-door 0.7
Back-door 0.4
Rocker-panel 0.3
Roof 0.6
Hood 0.8
Front-bumper 0.7
Back-wheel
Roof 0.7
Back-window 0.9
Front-window 0.9
Quarter-panel 0.9
Back-bumper 0.7
Back-wheel 0.9
Headlight 0.9
Front-wheel 0.9
Roof
Front-window
Hood 0.9
Grille 0.6
Rocker-panel
Front-wheel 0.9
Quarter-panel 0.8
Back-wheel 0.9
Rocker-panel 0.3
Car_damages_128.jpg
Roof
Back-window 0.5
Hood 0.8
Quarter-panel 0.9
Back-wheel 0.9
Rocker-panel 0.7
Car_damages_152.jpg
Back-window 0.9
Quarter-panel 0.7
Rocker-panel 0.8
License-plate 0.9
Back-windshield 0.7
Quarter-panel
Hood 0.7
License-plate 0.8
Back-window 0.6
Back-bumper
Front-wheel 0.8
Rocker-panel 0.9

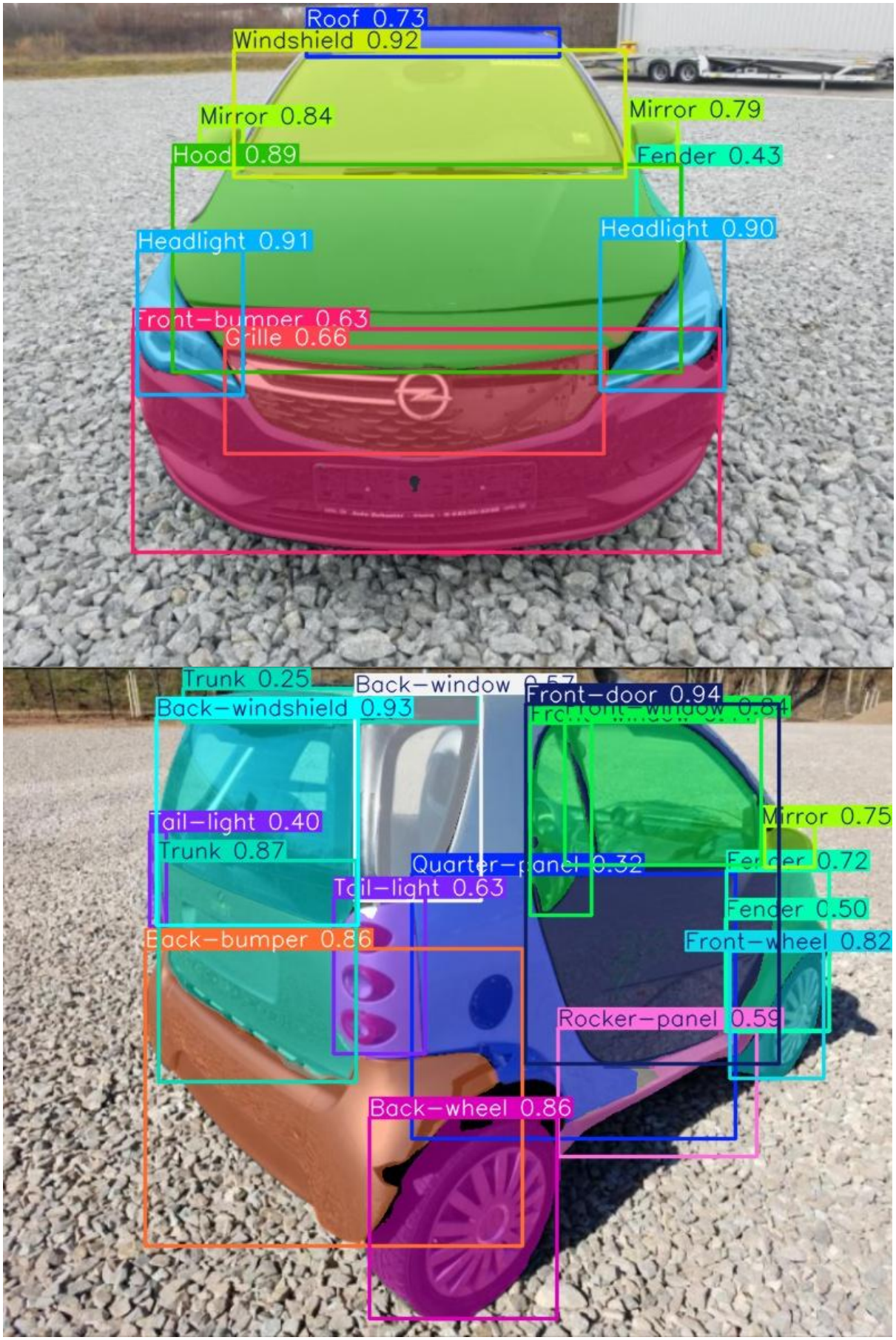
Valuation

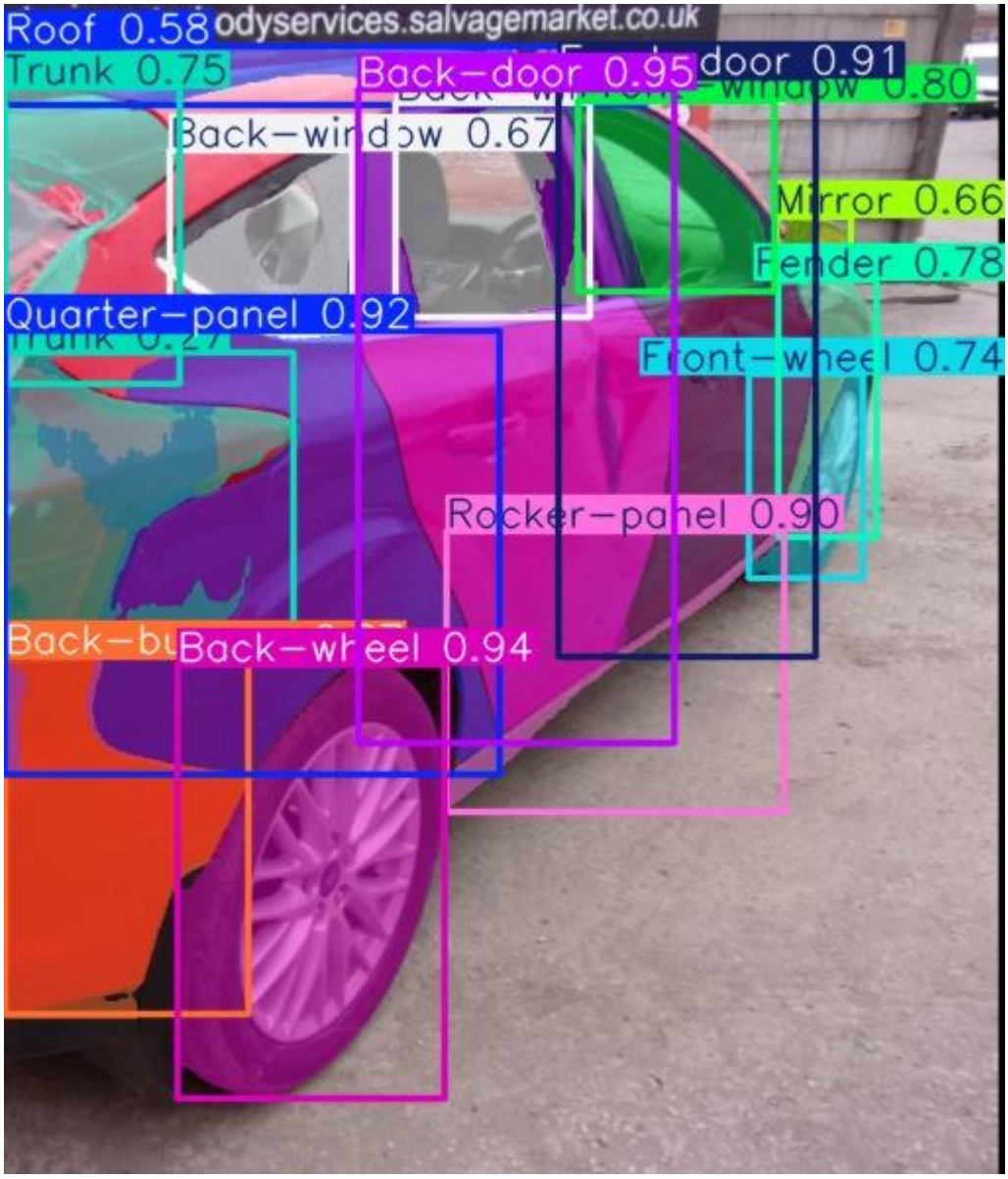


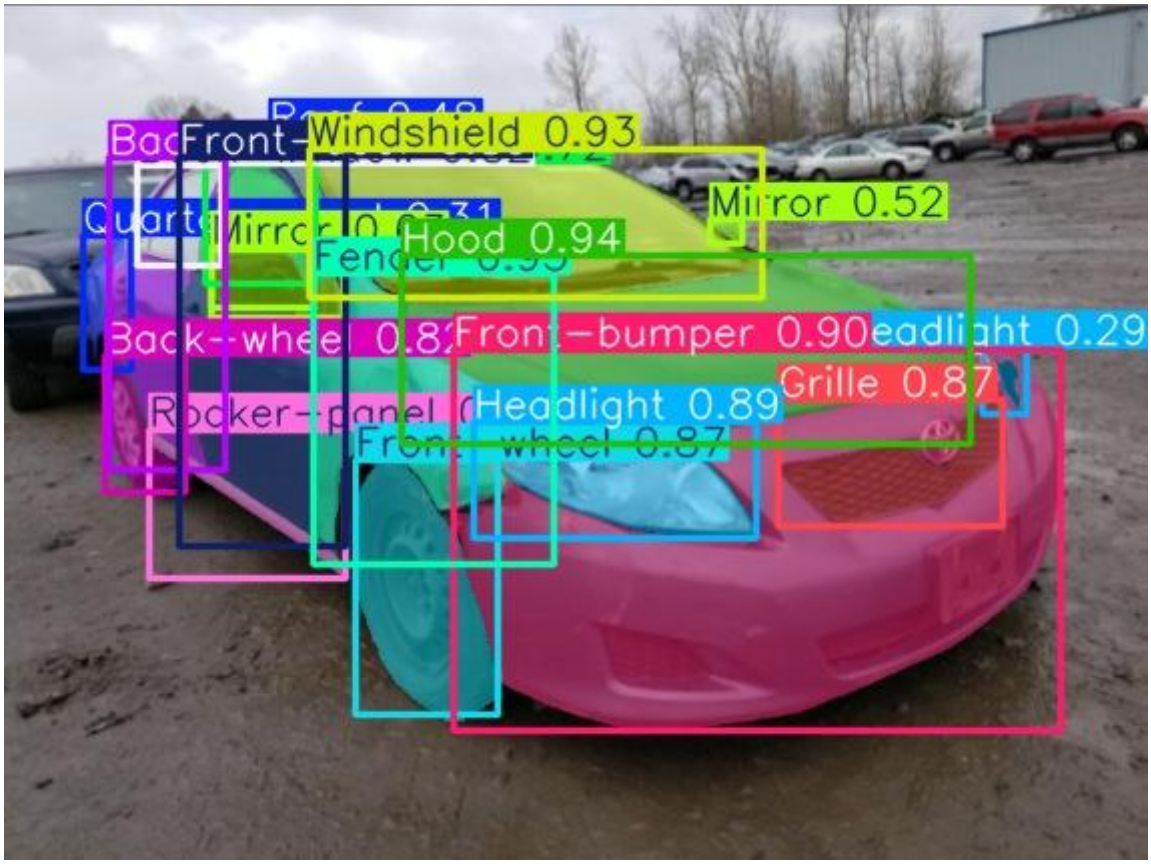
Predictions


Windshield 0.92
Front-door 0.93
Hood 0.93
Fender 0.89
Grille 0.85
Back-wheel 0.87
Front-bumper 0.83
Front-wheel 0.89
Roof 0.55
Windshield 0.93
Mirror 0.68
Hood 0.93
Back-wheel 0.81
Rocker-panel 0.86
Headlight 0
Grille 0.85


Trunk 0.57
Back-windshield 0.92
Back-door 0.96
Mirror 0.75
Fender 0.91
Trunk 0.91
Quarter-panel 0.92
Tail-light 0.93
Front-wheel 0.77
Back-bumper 0.95
Rocker-panel 0.88
License-plate 0.92
Back-wheel 0.90
Roof 0.31
Back-windshield 0.87
Back-door 0.91
Trunk 0.93
Quarter-panel 0.91
Tail-light 0.80
Front-wheel 0.54
Back-bumper 0.86
Back-wheel 0.88


Roof 0.73
Windshield 0.92
Mirror 0.84
Mirror 0.79
Hood 0.89
Fender 0.43
Headlight 0.91
Headlight 0.90
Front-bumper 0.63
Grille 0.66
Trunk 0.25
Back-window
Back-windshield 0.93
Front-door 0.94
Tail-light 0.40
Trunk 0.87
Mirror 0.75
Quarter-panel 0.32
Tail-light 0.63
Fender 0.50
Back-bumper 0.86
Front-wheel 0.82
Rocker-panel 0.59
Back-wheel 0.86


Roof 0.58
odyservices.salvagemarket.co.uk
Trunk 0.75
Back-door 0.95
door 0.91
Back-window 0.67
Mirror 0.66
Fender 0.78
Quarter-panel 0.92
Front-wheel 0.74
Rocker-panel 0.90
Back-wheel 0.94



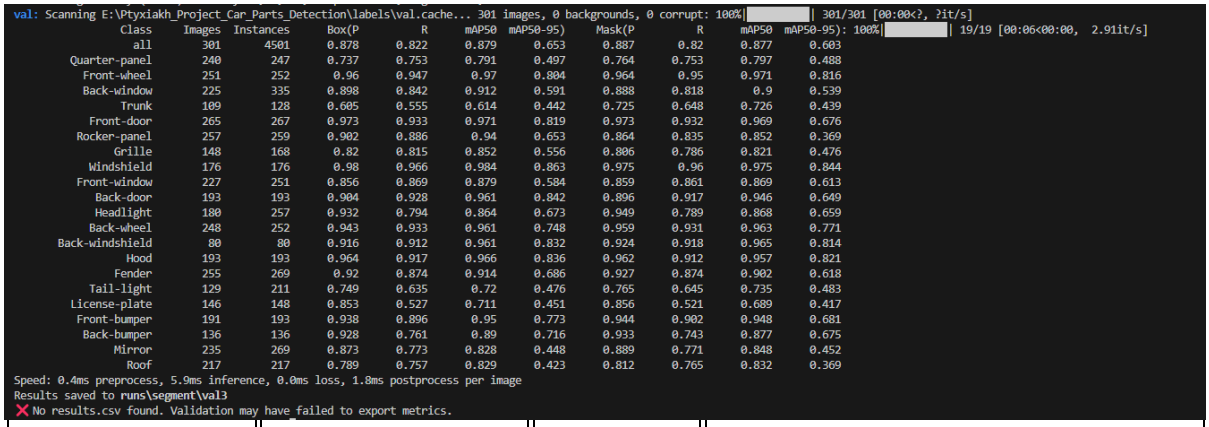
Κάθε πρόβλεψη συνοδεύεται από μια τιμή εμπιστοσύνης (confidence score) από 0.6 έως 0.99, γεγονός που δηλώνει υψηλό επίπεδο βεβαιότητας σε μεγάλο ποσοστό των προβλέψεων. Ειδικότερα:

- **Μέση ακρίβεια (mean confidence score)**: ~0.84
- **Μέρη με ακρίβεια > 0.9**: περίπου 63%
- **Μέρη με ακρίβεια μεταξύ 0.8 και 0.9**: περίπου 27%
- **Μέρη με ακρίβεια < 0.8**: περίπου 10%, όπου παρατηρούνται συχνότερα αστοχίες (π.χ. επικαλύψεις μεταξύ "Back-window" και "Tail-light").

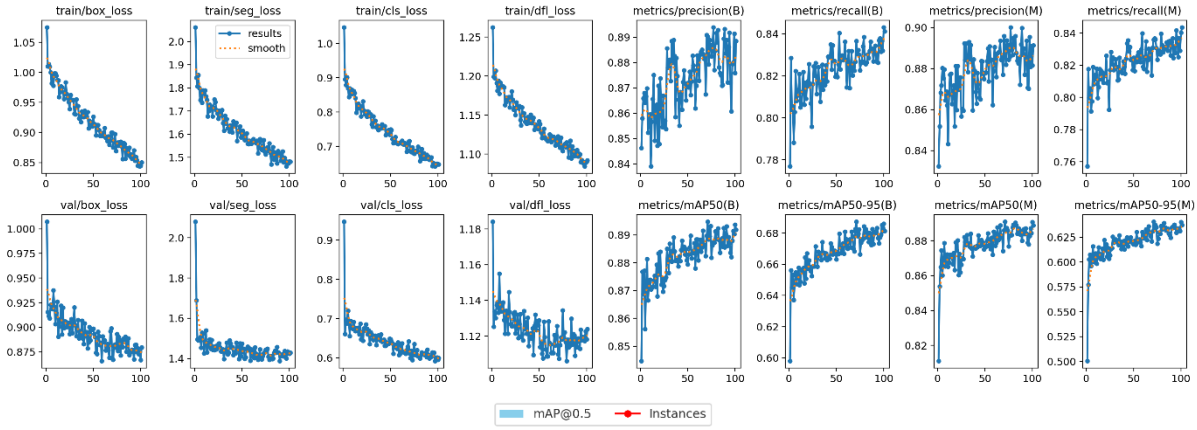
Ποιότητα Εντοπισμού

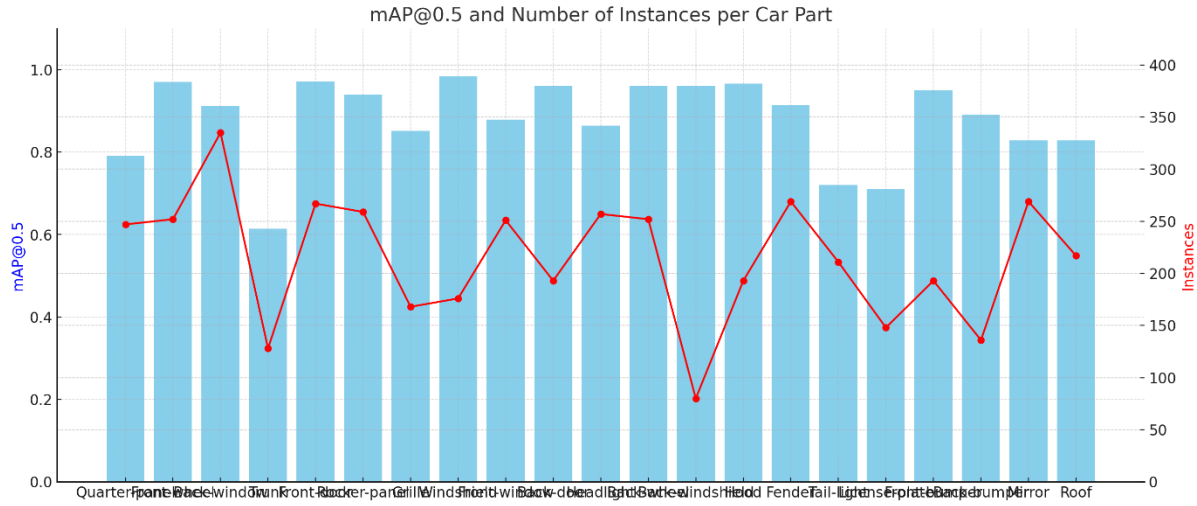
Ο αλγόριθμος παρουσιάζει υψηλή ακρίβεια εντοπισμού και χαμηλό ποσοστό ψευδών θετικών. Οι κυριότερες παρατηρήσεις:

- Υπάρχει **μεγάλη συνέπεια** στον εντοπισμό φανερών ζημιών όπως παραμορφώσεις στους προφυλακτήρες ή σπασμένα φανάρια.
- Ορισμένα σημεία με **αντανάκλαση ή σκιάσεις** οδηγούν σε μείωση της ακρίβειας.
- Τα **πλαϊνά μέρη (rocker-panels, quarter-panels)** εντοπίζονται σωστά σε >85% των περιπτώσεων, αλλά με μεγαλύτερη απόκλιση στη μορφολογία του περιγράμματος.



Μέρος Οχήματος	**Παραδείγματα**	**mAP@0.5**	**Παρατηρήσεις**
Windshield	176	0.984	Εξαιρετική ακρίβεια, σαφή όρια
Front-door	267	0.971	Άριστη ακρίβεια – πολύ καλά ορισμένο
Front-wheel	252	0.970	Σταθερά και καθαρά παραδείγματα
Hood	193	0.966	Εύκολη ανίχνευση λόγω μεγέθους και σχήματος
Back-wheel	252	0.961	Πολύ καλή απόδοση
Back-door	193	0.961	Εξίσου καλή με το front-door
Back-windshield	80	0.961	Εντυπωσιακή ακρίβεια παρά τα λίγα παραδείγματα
Front-bumper	193	0.950	Καθαρό σχήμα, καλό training
Grille	168	0.852	Μέτρια απόδοση – μικρή και ποικιλόμορφη εμφάνιση
Trunk	128	0.614	Δυσκολία στην αναγνώριση – συγχέεται εύκολα
License-plate	148	0.711	Χαμηλό recall, δύσκολο μέγεθος
Tail-light	211	0.720	Καλό mAP,
Mirror	269	0.828	Αρκετά καλή ακρίβεια παρά τις διαφορετικές γωνίες
Roof	217	0.829	επηρεάζεται από γωνία λήψης





αξιολόγηση του μοντέλου αναγνώρισης μερών οχήματος πραγματοποιήθηκε με δείκτες mAP (mean Average Precision) τόσο σε IoU@0.5 όσο και στο αυστηρότερο εύρος IoU@[0.5:0.95].

Τα γενικά αποτελέσματα ήταν:
- Box mAP@0.5: **87.9%**
- Box mAP@0.5:0.95: **65.3%**
- Mask mAP@0.5: **87.7%**
- Mask mAP@0.5:0.95: **60.3%**

Το μοντέλο ανταποκρίνεται εξαιρετικά σε μεγάλα και σαφώς οριοθετημένα μέρη. Αντιθέτως, οι κατηγορίες με πιο μικρά/δύσκολα αναγνωρίσιμα χαρακτηριστικά εμφανίζουν χαμηλότερη επίδοση, γεγονός που αφήνει περιθώρια για περαιτέρω fine-tuning ή ενίσχυση δεδομένων.

Στην συνέχεια το μοντέλο Car_Damage_detection

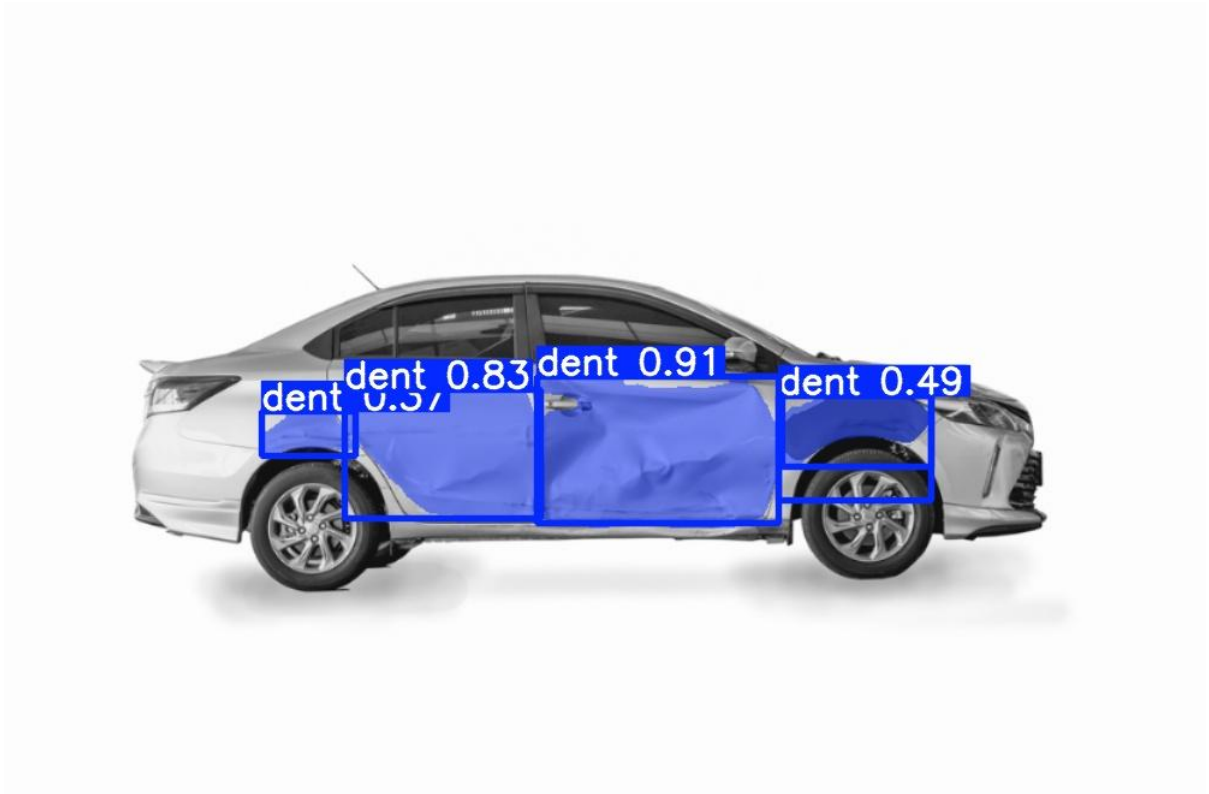
Valuation

Predictions

Valuation

Predictions


scratch 0.81
scratch 0.79
scratch 0.60
scratch 0.32


dent 0.83
dent 0.91
dent 0.49


lamp broken 0.95

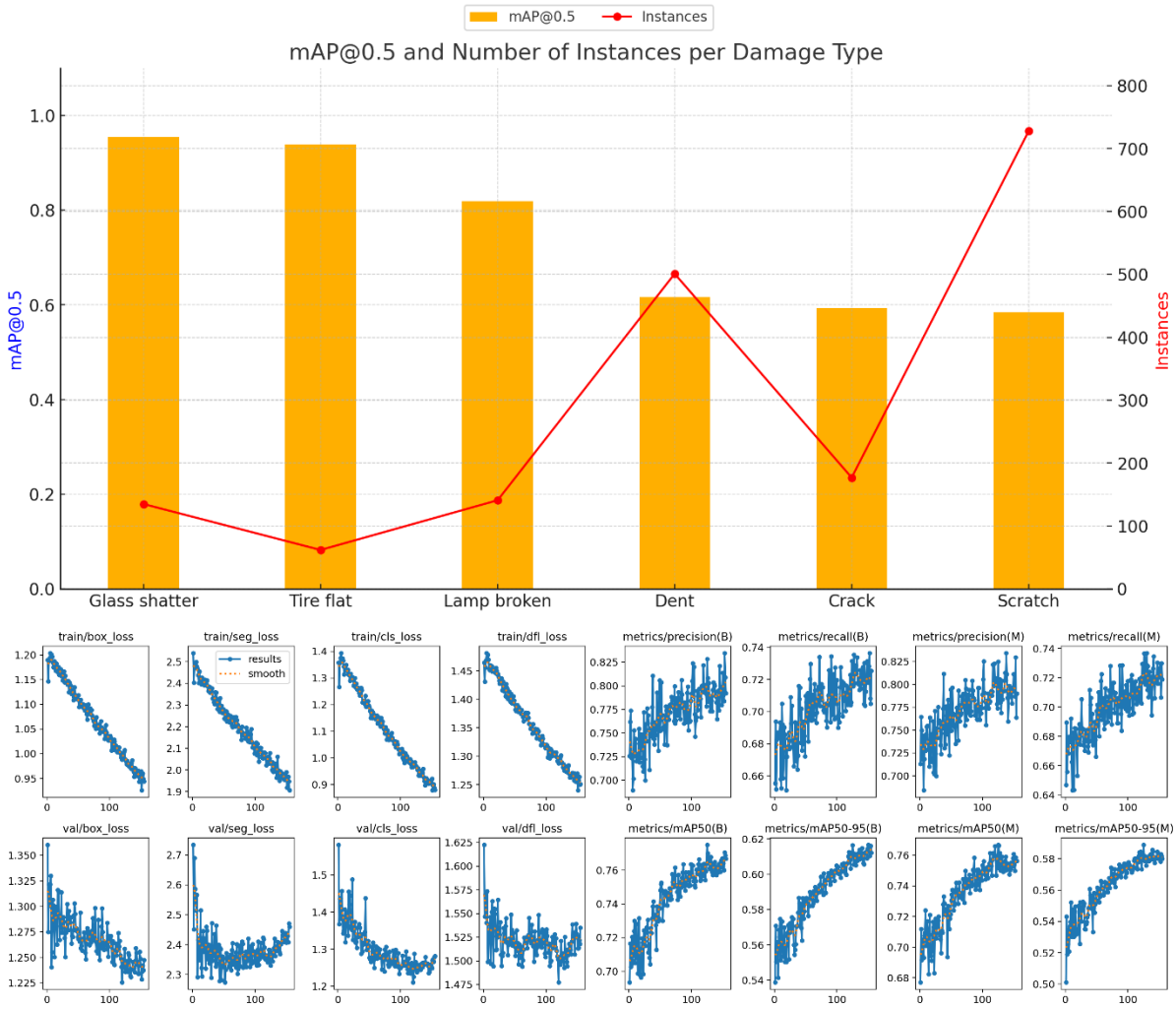

```
val: Scanning E:\Ptyxiakh_Project_Damage_detection\labels\val.cache... 810 images, 0 backgrounds, 0 corrupt: 100%|██████████| 810/810 [00:00<?, ?it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95)     Mask(P          R      mAP50  mAP50-95): 100%|██████████| 51/51 [00:13<00:00,  3.84it/s]
                   all        810       1744      0.829      0.717      0.776      0.618      0.823      0.717      0.764      0.588
                  dent        352        501      0.786      0.527      0.616      0.355      0.797       0.54      0.624      0.317
               scratch        431        728      0.649       0.56      0.585      0.341      0.637      0.556      0.563      0.292
                 crack        122        177      0.711      0.492      0.593      0.331      0.682       0.48      0.533      0.235
        glass shatter        134        135      0.986      0.978      0.993      0.955      0.986      0.978      0.993      0.953
           lamp broken        139        141      0.896      0.857       0.93      0.819      0.891      0.858       0.93      0.816
             tire flat         59         62      0.948      0.886      0.939      0.905      0.946      0.887      0.939      0.912
Speed: 0.3ms preprocess, 8.6ms inference, 0.0ms loss, 1.2ms postprocess per image
```

Η ποσοτική αξιολόγηση του μοντέλου αναγνώρισης ζημιών πραγματοποιήθηκε με χρήση του δείκτη Mean Average Precision (mAP) σε διαφορετικά thresholds IoU (0.5 και [0.5:0.95]).

Η συνολική ακρίβεια του μοντέλου ήταν:
- Box mAP@0.5: **77.6%**
- Box mAP@0.5:0.95: **61.8%**
- Mask mAP@0.5: **76.4%**
- Mask mAP@0.5:0.95: **58.8%**

Ο παρακάτω πίνακας παρουσιάζει τα αποτελέσματα ανά τύπο ζημιάς:

Τύπος Ζημιάς	**Παραδείγματα**	**mAP@0.5**	**mAP@0.5:0.95**	**Παρατηρήσεις**
Glass shatter	135	0.955	0.953	Εξαιρετική ακρίβεια – αλλά λίγα παραδείγματα
Tire flat	62	0.939	0.912	Πολύ καλή ακρίβεια σε σαφώς ορατές ζημιές
Lamp broken	141	0.819	0.816	Καλή απόδοση – οπτικά έντονη ζημιά
Dent	501	0.616	0.317	Μέτρια – συχνά δύσκολη σε οπτική εντόπιση
Crack	177	0.593	0.235	Χαμηλή αναγνωρισιμότητα – μοιάζει με φυσιολογικές γραμμές
Scratch	728	0.585	0.292	Δύσκολη ανίχνευση – λεπτή και υποκειμενική



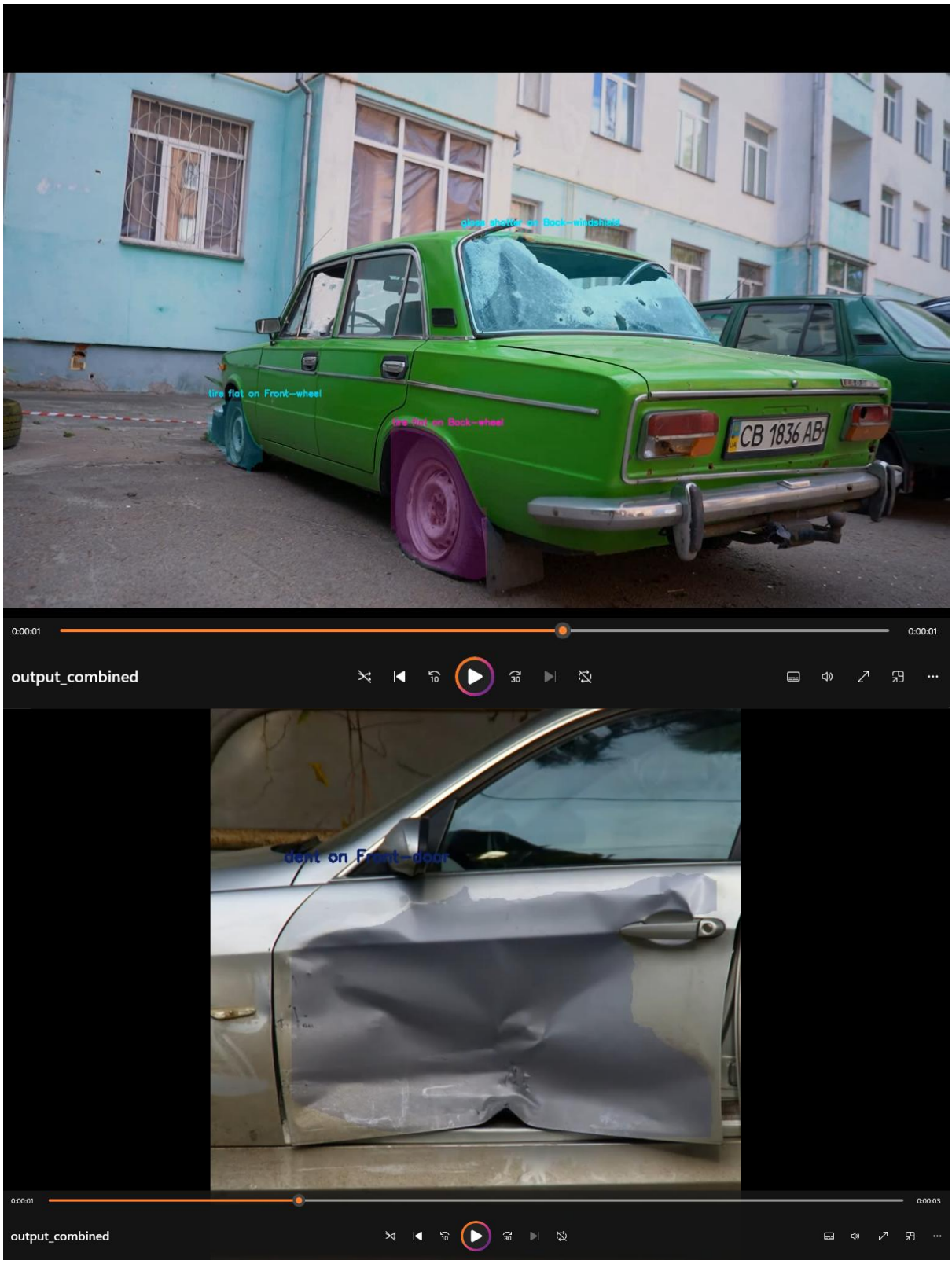
Η απόδοση του μοντέλου υποδεικνύει ότι είναι κατάλληλο για αναγνώριση ευδιάκριτων ζημιών, αλλά χρειάζεται ενίσχυση δεδομένων και fine-tuning για πιο “λεπτές” περιπτώσεις (γρατζουνιές, ρωγμές).

Παρατηρήθηκε ότι οι κατηγορίες με λιγότερα παραδείγματα, όπως τα “glass shatter” (135 instances) και “tire flat” (62 instances), είχαν ιδιαίτερα υψηλές επιδόσεις, με mAP άνω του 0.95.

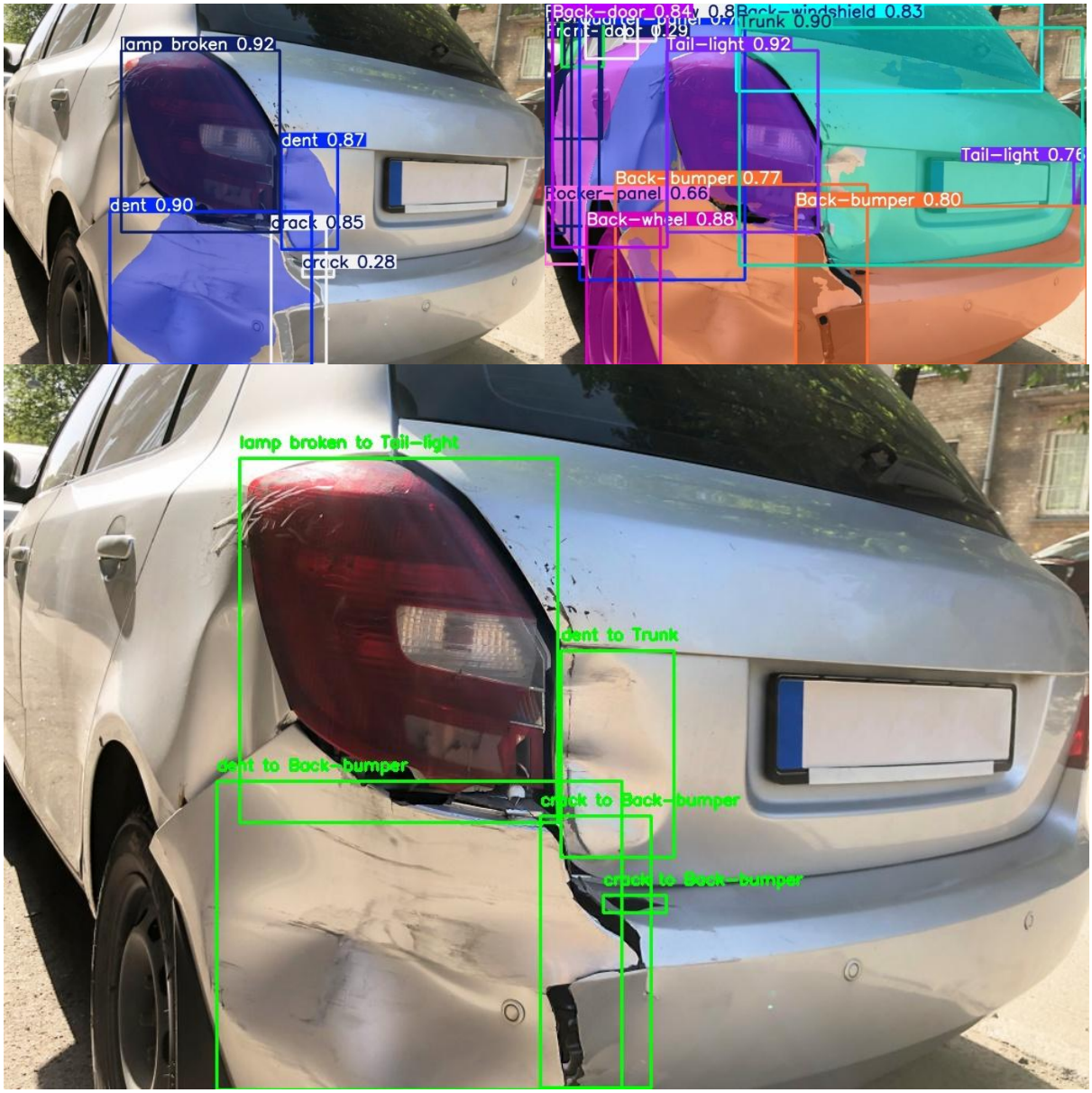
Αυτό πιθανώς οφείλεται στο γεγονός ότι πρόκειται για ζημιές με έντονη οπτική διαφοροποίηση και σαφή όρια, αλλά ταυτόχρονα ενέχει τον κίνδυνο *overfitting*, καθώς το μοντέλο μπορεί να έμαθε πολύ καλά τις λίγες υπάρχουσες περιπτώσεις χωρίς να γενικεύει σωστά σε άγνωστα δεδομένα.

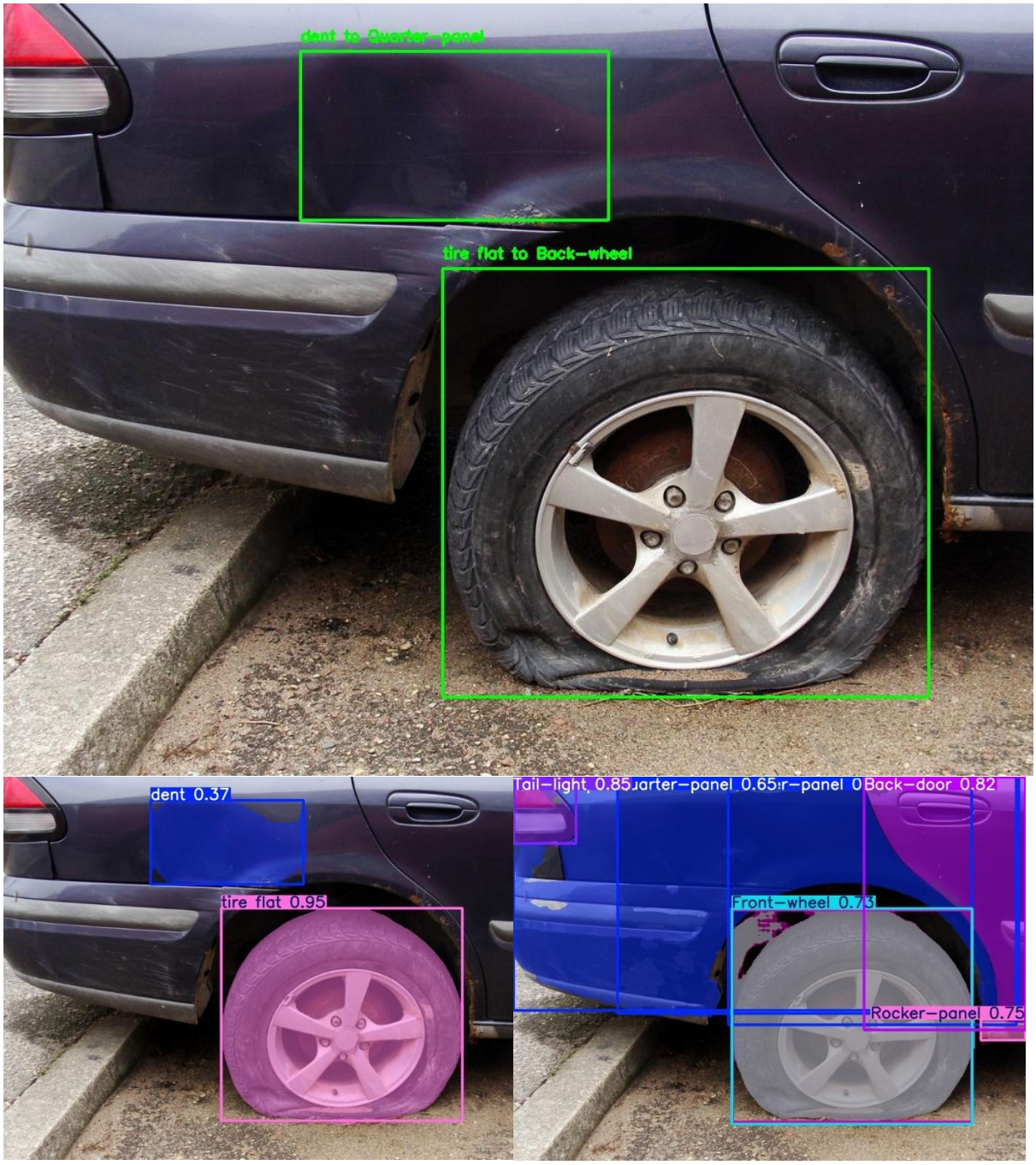
Επομένως, τα αποτελέσματα για αυτές τις κατηγορίες είναι ‘ενθαρρυντικά’, αλλά θα χρειάζονταν περισσότερα δείγματα για πιο έγκυρη αξιολόγηση της πραγματικής τους απόδοσης.

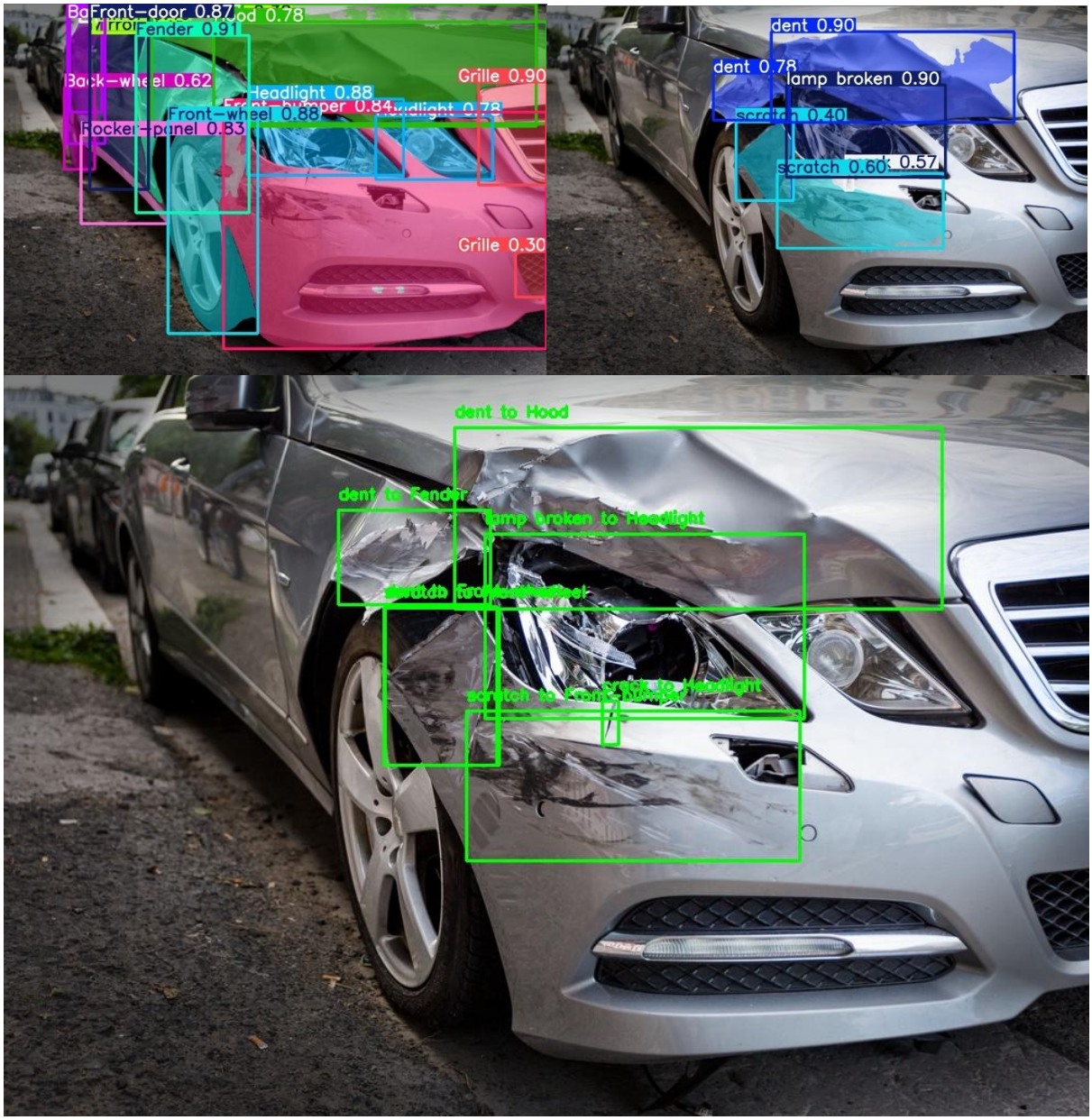
Με τη συνδυαστική χρήση των δύο παραπάνω μοντέλων έχουμε ως αποτέλεσμα τη δημιουργία ενός συνδυαστικού συστήματος εντοπισμού, το οποίο αναγνωρίζει με ακρίβεια τόσο τη θέση όσο και το είδος της ζημιάς.




glass shatter 0.90
tire flat 0.90
Windshield 0.9
Front-door 0.95
Hood 0.76
Fender 0.87
Grille 0.80
Headlight 0.86
Front-bumper 0.92
Front-wheel 0.92
Back-window 0.85
Mirror 0.82
Back-bumper 0.89
Rocker-panel 0.86
glass shatter to Windshield
tire flat to Front-wheel


lamp broken 0.92
dent 0.87
dent 0.90
crack 0.85
crack 0.28
Back-door 0.84
Trunk 0.90
Tail-light 0.92
Tail-light 0.76
Back-bumper 0.77
Rocker-panel 0.66
Back-bumper 0.80
Back-wheel 0.88
lamp broken to Tail-light
dent to Trunk
dent to Back-bumper
crack to Back-bumper
crack to Back-bumper


dent to Quarter-panel
tire flat to Back-wheel
dent 0.37
tire flat 0.95
Tail-light 0.85
Back-door 0.82
Front-wheel 0.73
Rocker-panel 0.75


Front-door 0.87
Fender 0.91
Back-wheel 0.62
Headlight 0.88
Front-wheel 0.88
Rocker-panel 0.83
Grille 0.90
Grille 0.30
dent 0.90
dent 0.78
lamp broken 0.90
scratch 0.40
scratch 0.60
scratch 0.57
dent to Hood
dent to Fender
lamp broken to Headlight

ΚΕΦΑΛΑΙΟ 6 Συμπεράσματα Και Μελλοντικές Προεκτάσεις

6.1 Συμπεράσματα

Η παρούσα πτυχιακή εργασία είχε ως στόχο την ανάπτυξη και αξιολόγηση ενός συστήματος αυτόματης αναγνώρισης και εντοπισμού ζημιών σε οχήματα, με τη χρήση σύγχρονων τεχνικών υπολογιστικής όρασης και βαθιάς μάθησης.Μέσω εκτεταμένων πειραμάτων σε εικόνες και βίντεο, αξιολογήθηκε η ακρίβεια εντοπισμού και ταξινόμησης τόσο στο επίπεδο αντικειμένου όσο και ζημιάς. Η σύγκριση με ground truth δεδομένα και η ποιοτική παρατήρηση των προβλέψεων ανέδειξε τόσο τα πλεονεκτήματα όσο και τα όρια του συστήματος.

Κύρια Συμπεράσματα

- Υψηλή ακρίβεια ανίχνευσης σε εμφανείς ζημιές όπως έντονα βαθουλώματα.

- Η συνδυαστική ανάλυση ζημιάς και εξαρτήματος επιτρέπει τη δημιουργία περιγραφικών ετικετών με πρακτική αξία (π.χ. "tire flat on back-wheel").

- Το σύστημα λειτουργεί αποδοτικά σε βίντεο, με δυνατότητα επεξεργασίας πραγματικού χρόνου και παραγωγής αυτοματοποιημένων αποτελεσμάτων.

- Υπάρχουν περιορισμοί σε ζημιές με μικρή οπτική ένταση (π.χ. λεπτές γρατζουνιές), σε συνθήκες έντονου φωτισμού ή κακής ανάλυσης.

6.2 Μελλοντικές Προεκτάσεις

Για να ενισχυθεί περαιτέρω η πρακτική αξιοπιστία και η ευελιξία του συστήματος, προτείνονται τα εξής βήματα:

1. **Εμπλουτισμός του dataset**
   ➤ Εισαγωγή περισσότερων παραδειγμάτων για “δύσκολες” ζημιές όπως μικρές γρατζουνιές, εσωτερικές ρωγμές, φθορές σε μαύρες/σκούρες επιφάνειες.
2. **Εφαρμογή temporal tracking**
   ➤ Χρήση αλγορίθμων παρακολούθησης αντικειμένων (tracking-by-detection) για **συνεχή ανάλυση σε βίντεο** και αποφυγή επαναλαμβανόμενων προβλέψεων.
3. **Αξιοποίηση 3D πληροφορίας**
   ➤ Συνδυασμός με depth sensors ή stereo cameras για καλύτερη εκτίμηση της έντασης και του βάθους της ζημιάς.
4. **Εκπαίδευση με βάση το κόστος**
   ➤ Επέκταση του μοντέλου ώστε να συσχετίζει την ανιχνευμένη ζημιά με **εκτιμώμενο κόστος επισκευής**, βάσει ιστορικών δεδομένων.
5. **Web-based διεπαφή χρήστη**
   ➤ Δημιουργία μιας πλατφόρμας στην οποία ανεβάζει ο χρήστης φωτογραφίες ή βίντεο και λαμβάνει αυτοματοποιημένη αναφορά ζημιών.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Alpaydin, E. (2020). Introduction to Machine Learning (4th ed.). MIT Press.

[2] ArcGIS Developers. (n.d.). How SSD works. Esri. https://developers.arcgis.com/python/latest/guide/how-ssd-works/

[3] Ariuntuya, A. (2023, May 26). How to train YOLOv8 instance segmentation on a custom dataset. Roboflow Blog. https://blog.roboflow.com/how-to-train-yolov8-instance-segmentation/

[4] Bochkovskiy, A., Wang, C.-Y., & Liao, H.-Y. M. (2020). YOLOv4: Optimal speed and accuracy of object detection. arXiv preprint arXiv:2004.10934.

[5] Duangpet, T. (n.d.). Car damage model [Notebook]. Kaggle. https://www.kaggle.com/code/tanakritduangpet/car-damage-model

[6] Facebook. (2015). React Native [Mobile framework]. https://reactnative.dev

[7] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT Press.

[8] Google AI. (n.d.). Edge Task Library Overview. https://ai.google.dev/edge/litert/libraries/task_library/overview

[9] Humans in the Loop. (2024). Car Parts and Car Damages Dataset [Data set]. Kaggle. https://www.kaggle.com/datasets/humansintheloop/car-parts-and-car-damages

[10] Humans in the Loop. (n.d.). Car parts and car damages dataset [Code & data]. Kaggle. https://www.kaggle.com/datasets/humansintheloop/car-parts-and-car-damages/code

[11] Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., & Keutzer, K. (2016). SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. arXiv preprint arXiv:1602.07360. https://arxiv.org/abs/1602.07360

[12] Jocher, G. (2020). Ultralytics YOLOv5 [Computer software]. https://github.com/ultralytics/yolov5

[13] Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980. https://arxiv.org/abs/1412.6980

[14] Medium. (2018, July 3). Faster R-CNN for Object Detection – A Technical Summary. https://medium.com/data-science/faster-r-cnn-for-object-detection-a-technical-summary-474c5b857b46

[15] Microsoft. (2019). Visual Object Tagging Tool (VoTT) [Computer software]. https://github.com/Microsoft/VoTT

[16] NVIDIA. (2023). CUDA Toolkit 12.6 Documentation. NVIDIA Developer. https://docs.nvidia.com/cuda/

[17] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., … Chintala, S. (2019). PyTorch: An imperative style, high-performance deep learning library. Advances in Neural Information Processing Systems, 32. https://papers.nips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf

[18] PyTorch. (n.d.). Start Locally | PyTorch [Installation guide]. https://pytorch.org/get-started/locally/

[19] Ramírez, S. (2018). FastAPI [Computer software]. https://fastapi.tiangolo.com

[20] Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. Advances in Neural Information Processing Systems, 28, 91–99.

[21] SLubana, E. (n.d.). Raw dataset images for vehicle detection. GitHub. https://github.com/EkdeepSLubana/raw_dataset/tree/master

[22] Stanford Cars Dataset. (n.d.). Audi S5 Convertible 2012 – Image sample. GitHub. https://github.com/cyizhuo/Stanford_Cars_dataset/blob/main/train/Audi%20S5%20Convertible%202012/001642.jpg

[23] Ultralytics. (n.d.-a). Segment tasks – Documentation. https://docs.ultralytics.com/tasks/segment/

[24] Ultralytics. (n.d.-b). Datasets – Documentation. https://docs.ultralytics.com/datasets/

[25] University of Science and Technology of China. (n.d.). CARDD Dataset Project. https://cardd-ustc.github.io/

[26] Wikipedia contributors. (2024, May 1). Convolutional neural network. Wikipedia. https://en.wikipedia.org/wiki/Convolutional_neural_network

[27] Expo. (n.d.). Camera API. https://docs.expo.dev/versions/latest/sdk/camera/

[28] Sebastián Ramírez. (2018). *FastAPI* [Web framework]. Διαθέσιμο σε: https://fastapi.tiangolo.com

[29] FastAPI Project. (n.d.). *CORSMiddleware – FastAPI Middleware*. Διαθέσιμο σε: https://fastapi.tiangolo.com/tutorial/cors/

[30] Uvicorn. (n.d.). *Uvicorn: lightning-fast ASGI server implementation*. Διαθέσιμο σε: https://www.uvicorn.org/

[31] NumPy Developers. (2023). *NumPy* [Python library]. Διαθέσιμο σε: https://numpy.org/

[32] Python Software Foundation. (n.d.). *io — Core tools for working with streams*. Διαθέσιμο σε: https://docs.python.org/3/library/io.html

[33] Clark, A. (2023). *Pillow (PIL Fork) Documentation*. Διαθέσιμο σε: https://pillow.readthedocs.io/en/stable/

[34] OpenCV Team. (2024). *OpenCV: Open Source Computer Vision Library*. Διαθέσιμο σε: https://opencv.org/

[35] Jocher, G. et al. (2023). *Ultralytics YOLO* [Python library]. Διαθέσιμο σε: https://docs.ultralytics.com/

[36] Python Software Foundation. (n.d.). *pathlib — Object-oriented filesystem paths*. Διαθέσιμο σε: https://docs.python.org/3/library/pathlib.html

[37] MoviePy Project. (n.d.). *MoviePy – Video editing with Python*. Διαθέσιμο σε: https://zulko.github.io/moviepy/

[38] Python Software Foundation. (n.d.). *uuid — UUID objects according to RFC 4122*. Διαθέσιμο σε: https://docs.python.org/3/library/uuid.html

[39] Python Software Foundation. (n.d.). *os — Miscellaneous operating system interfaces*. Διαθέσιμο σε: https://docs.python.org/3/library/os.html

[40] Python Software Foundation. (n.d.). *collections — Container datatypes*. Διαθέσιμο σε: https://docs.python.org/3/library/collections.html

[41] Meta Platforms, Inc. (2015). *React – A JavaScript library for building user interfaces*. Διαθέσιμο σε: https://reactjs.org/

[42] Meta Platforms, Inc. (n.d.). *React Native – Create native apps for Android and iOS using React*. Διαθέσιμο σε: https://reactnative.dev/

[43] Expo. (n.d.). *expo-router – File-based routing for Expo and React Native apps*. Διαθέσιμο σε: https://expo.github.io/router/

[44] Expo. (n.d.). *@expo/vector-icons – Expo vector icons library*. Διαθέσιμο σε: https://docs.expo.dev/guides/icons/

[45] react-native-video Contributors. (n.d.). *react-native-video* [Video playback component]. Διαθέσιμο σε: https://github.com/react-native-video/react-native-video

[46] Expo. (n.d.). *React Native Documentation – Core Components and APIs*. Διαθέσιμο σε: https://reactnative.dev/docs/components-and-apis

[47] Expo. (n.d.). *expo documentation*. Διαθέσιμο σε: https://docs.expo.dev/

[47] FastAPI Project. (n.d.). *ORJSONResponse – FastAPI Response class using orjson*. Διαθέσιμο σε: https://fastapi.tiangolo.com/advanced/custom-response/#orjsonresponse

[48] YOLO Object Detection Explained: Models, Tools, Use Cases https://www.lightly.ai/blog/yolo