ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΣΥΣΤΗΜΑΤΩΝ ΜΕΤΑΔΟΣΗΣ ΠΛΗΡΟΦΟΡΙΑΣ
ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΛΙΚΩΝ

Εντοπισμός κινούμενων οχημάτων στην πλατφόρμα Versal ACAP

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Γεώργιος Β. Κόικας

Επιβλέπων : Δημήτριος Ι. Σούντρης
Καθηγητής Ε.Μ.Π.

Αθήνα, Φεβρουάριος 2024

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΣΥΣΤΗΜΑΤΩΝ ΜΕΤΑΔΟΣΗΣ ΠΛΗΡΟΦΟΡΙΑΣ
ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΛΙΚΩΝ

Εντοπισμός κινούμενων οχημάτων στην πλατφόρμα Versal ACAP

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Γεώργιος Β. Κόικας

Επιβλέπων : Δημήτριος Ι. Σούντρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 22η Μαρτίου 2024.

.....	.....	.....
Σούντρης Δημήτριος	Τσανάκας Παναγιώτης	Ξύδης Σωτήριος
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Επίκουρος Καθηγητής Ε.Μ.Π.

Αθήνα, Φεβρουάριος 2024

.....
Γεώργιος Β. Κόικας

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Στην εποχή της ψηφιακής επανάστασης, όπου η τεχνολογία μετασχηματίζεται με εκθετικούς ρυθμούς, η παρούσα διπλωματική εργασία σπεύδει να εφαρμόσει και να αναπτύξει τις γνώσεις της μηχανικής μάθησης και των νευρωνικών δικτύων, εστιάζοντας συγκεκριμένα στην πρόκληση της ανίχνευσης και παρακολούθησης οχημάτων. Μέσα από μια προσεκτική διερεύνηση του θεωρητικού υπόβαθρου και την εφαρμογή προηγμένων τεχνολογικών εργαλείων, αποκαλύπτεται η πληθώρα των δυνατοτήτων που ανοίγονται μέσω της ενσωμάτωσης καινοτόμων λύσεων στον τομέα της τεχνητής νοημοσύνης.

Η εργασία ξεκινά με μια ολοκληρωμένη ανάλυση του θεωρητικού υπόβαθρου, καταδεικνύοντας την κρισιμότητα της κατανόησης των βασικών αρχών της μηχανικής μάθησης και της λειτουργίας των νευρωνικών δικτύων. Η χρήση τεχνολογικών εργαλείων όπως η γλώσσα προγραμματισμού Python, το Pytorch, καθώς και πλατφόρμες όπως το Google Colaboratory και το Vitis AI, αποτελεί θεμέλιο για την ανάπτυξη και την αποδοτική εκτέλεση της εφαρμογής.

Η πρακτική εφαρμογή αποκαλύπτει μια σύνθετη διαδικασία ανάπτυξης, από την προετοιμασία των δεδομένων εισόδου έως τη διασύνδεση προηγμένων αλγορίθμων ανίχνευσης και παρακολούθησης. Η εκτενής αξιολόγηση της απόδοσης των μοντέλων, μέσω συγκριτικής ανάλυσης και μετρήσεων κατανάλωσης ενέργειας, επιβεβαιώνει την αποδοτικότητα των προτεινόμενων λύσεων.

Το σημαντικότερο επίτευγμα της εργασίας είναι η επιτάχυνση των υπολογισμών των μοντέλων YOLO μέσω της χρήσης Field Programmable Gate Arrays (FPGAs), η οποία ανοίγει νέες δυνατότητες για την ανίχνευση οχημάτων σε πραγματικό χρόνο. Η ενσωμάτωση της σειράς Versal AI Core VCK190 υπογραμμίζει τη δυνατότητα για αυξημένη αποδοτικότητα και ταχύτητα επεξεργασίας, παρέχοντας ένα αξιόλογο πλεονέκτημα στην εφαρμογή της τεχνητής νοημοσύνης σε πρακτικά προβλήματα. Συγκεκριμένα, εντυπωσιακό αποτελέσματα της εργασίας είναι η επίτευξη σχεδόν 100 καρέ ανά δευτερόλεπτο (fps) – ταχύτητα πάνω από 5x καλύτερη από μία T4 GPU - με κατανάλωση ενέργειας μόλις 20-30 watt, καταδεικνύοντας την υψηλή αποδοτικότητα και την οικονομία στην ενέργεια των προτεινόμενων λύσεων.

Συνοψίζοντας, η παρούσα διπλωματική εργασία αποσκοπεί να αναδείξει πώς η συνεχής τεχνολογική εξέλιξη και η καινοτομία μπορούν να οδηγήσουν στην ανάπτυξη πρακτικών και αποδοτικών λύσεων για την αντιμετώπιση περίπλοκων προκλήσεων.

Λέξεις Κλειδιά: Μηχανική μάθηση, Συνελικτικά Νευρωνικά Δίκτυα, Ανίχνευση οχημάτων, Pytorch, FPGAs, Versal AI Core VCK190

Abstract

In the era of the digital revolution, where technology is transforming at exponential rates, the present thesis endeavors to apply and develop knowledge in machine learning and neural networks, focusing specifically on the challenge of vehicle detection and tracking. Through careful exploration of the theoretical background and the application of advanced technological tools, the plethora of possibilities opened up through the integration of innovative solutions in the field of artificial intelligence is revealed.

The thesis begins with a comprehensive analysis of the theoretical background, demonstrating the critical importance of understanding the basic principles of machine learning and neural network operation. The use of technological tools such as the Python programming language, Pytorch, as well as platforms like Google Colaboratory and Vitis AI, lays the foundation for the development and efficient execution of the application.

The practical implementation reveals a complex development process, from the preparation of input data to the integration of advanced detection and tracking algorithms. The extensive evaluation of model performance, through comparative analysis and energy consumption measurements, confirms the efficiency of the proposed solutions.

The most significant achievement of the work is the acceleration of YOLO model computations through the use of Field Programmable Gate Arrays (FPGAs), opening up new possibilities for real-time vehicle detection. The integration of the Versal AI Core VCK190 series underscores the potential for increased efficiency and processing speed, providing a significant advantage in the application of artificial intelligence to practical problems. Specifically, impressive results of the work include achieving nearly 100 frames per second (fps) - over 5x faster than a T4 GPU - with energy consumption as low as 20-30 watts, demonstrating high efficiency and energy economy of the proposed solutions.

In summary, the present thesis aims to highlight how continuous technological advancement and innovation can lead to the development of practical and efficient solutions for addressing complex challenges. Through the exploration of the possibilities of artificial intelligence, this work invites further research and development in the exciting journey of technological progress.

Keywords: Machine Learning, Convolutional Neural Networks, Vehicle Detection, Pytorch, FPGAs, Versal AI Core VCK190

Περιεχόμενα

1 Θεωρητικό Υπόβαθρο

1.1 Machine Learning

1.1.1 Ορισμός

Η Μηχανική Μάθηση (ML) αποτελεί πεδίο εντυπωσιακής εξέλιξης στην επιστήμη των υπολογιστών, επιτρέποντας στους υπολογιστές να μαθαίνουν αυτόματα από δεδομένα και να προβλέπουν μελλοντικές καταστάσεις. Αν και η ML υπάρχει εδώ και δεκαετίες, η εκθετική ανάπτυξη της υπολογιστικής ισχύος και η διαθεσιμότητα μεγάλων όγκων δεδομένων έχουν ενισχύσει τη σημασία της.

Κατηγορίες Μηχανικής Μάθησης:

- Επιβλεπόμενη Μάθηση (Supervised Learning): Εκπαίδευση μοντέλου με ετικετασμένα δεδομένα για πρόβλεψη ετικετών σε νέα δεδομένα.
- Ανεπιβλεπόμενη Μάθηση (Unsupervised Learning): Κατηγοριοποίηση δεδομένων χωρίς ετικέτες, ανακάλυψη προτύπων.
- Ενισχυτική Μάθηση (Reinforcement Learning): Εκπαίδευση με ανταμοιβές ή τιμωρίες για βέλτιστη συμπεριφορά σε ένα περιβάλλον.

1.1.2 Εφαρμογές

Η Μηχανική Μάθηση έχει επιφέρει επαναστατικές αλλαγές σε πολλούς τομείς, προσφέροντας πρωτοποριακές λύσεις και ανοίγοντας νέους δρόμους στην τεχνολογική πρόοδο. Ενδεικτικά, κάποια βασικά πεδία εφαρμογής που δείχνουν την ευρεία επιρροή και τη δυναμική της μηχανικής μάθησης είναι:

- Αναγνώριση Προτύπων: Η μηχανική μάθηση παίζει κρίσιμο ρόλο στην ανάπτυξη συστημάτων ικανών να αναγνωρίζουν και να ερμηνεύουν πρότυπα σε δεδομένα, είτε πρόκειται για εικόνες, ήχο, ή ακόμα και συμπεριφορικά μοτίβα. Αυτό ενισχύει τις δυνατότητες σε τομείς όπως η ιατρική απεικόνιση, η βιομετρία και η ανίχνευση απατών.
- Φυσική Γλώσσα και Μετάφραση: Η επεξεργασία φυσικής γλώσσας (Natural Language Processing, NLP), ένας κλάδος της μηχανικής μάθησης, ασχολείται με την ανάλυση, κατανόηση και γενέτειρα της ανθρώπινης γλώσσας. Αυτό

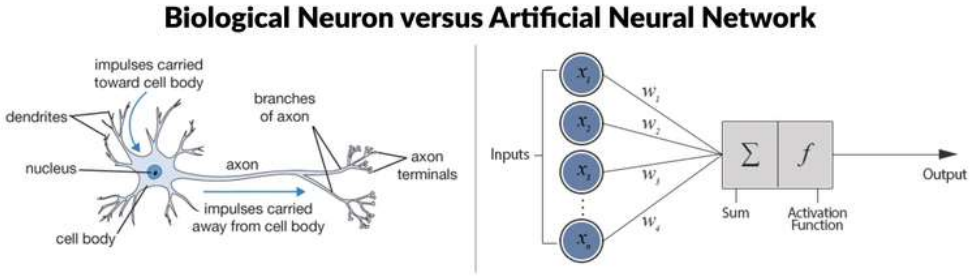
επιτρέπει την ανάπτυξη προηγμένων συστημάτων μετάφρασης και διαλογικών συστημάτων, βελτιώνοντας την επικοινωνία μεταξύ ανθρώπων και μηχανών.

- Αυτόνομα Συστήματα (Οδήγηση): Στον τομέα των αυτόνομων οχημάτων, η μηχανική μάθηση είναι καθοριστική για την ανάπτυξη οχημάτων που μπορούν να αντιλαμβάνονται το περιβάλλον, να λαμβάνουν αποφάσεις και να εκτελούν δράσεις με ελάχιστη ή καθόλου ανθρώπινη επέμβαση, αυξάνοντας την ασφάλεια και την αποδοτικότητα στον τομέα της μεταφοράς.

1.2 Νευρωνικά Δίκτυα

1.2.1 Νευρώνες

Οι νευρώνες σε ένα νευρωνικό δίκτυο είναι απλουστευμένες απομιμήσεις των βιολογικών νευρώνων. Σε ένα τυπικό νευρωνικό δίκτυο, ο κάθε νευρώνας λαμβάνει πολλαπλές εισόδους, είτε από τα αρχικά δεδομένα είτε από τις εξόδους άλλων νευρώνων. Κάθε είσοδος έχει ένα βάρος, το οποίο αποτελεί μέρος της "μάθησης" του δικτύου. Οι είσοδοι αθροίζονται, και η συνολική τιμή περνά από μια συνάρτηση ενεργοποίησης, η οποία καθορίζει την έξοδο του νευρώνα. Αυτή η διαδικασία επιτρέπει στο δίκτυο να κάνει πολύπλοκους υπολογισμούς και να εκπαιδευτεί για να αναγνωρίζει πρότυπα. [1] [2]



Σχήμα 1: Ομοιότητες μεταξύ Βιολογικών και Τεχνιτών Νευρώνων

1.2.2 Training & Inference

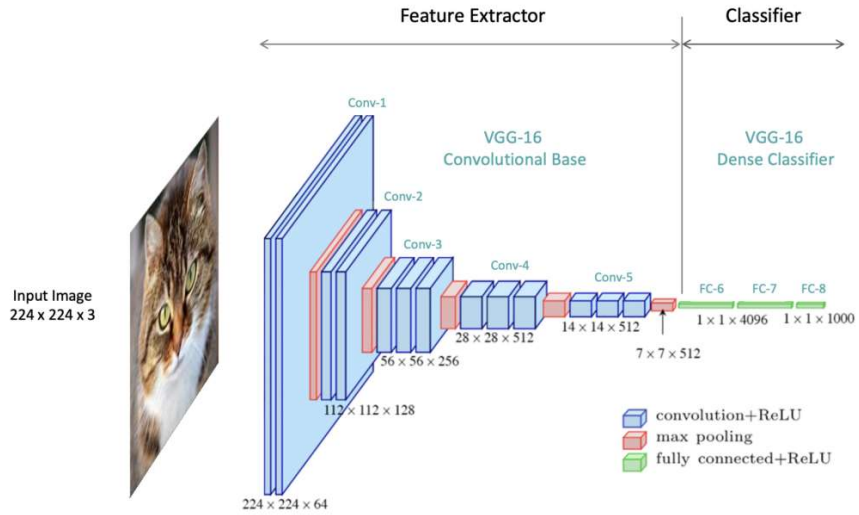
Η εκπαίδευση ενός νευρωνικού δικτύου είναι μια διαδικασία όπου το δίκτυο προσαρμόζεται για να αναγνωρίζει συγκεκριμένα πρότυπα στα δεδομένα. Κατά τη διάρκεια της εκπαίδευσης, το δίκτυο παρουσιάζεται με ένα σύνολο εκπαιδευτικών δεδομένων και προσπαθεί να κάνει προβλέψεις ή κατηγοριοποιήσεις. Όταν το δίκτυο κάνει μια λάθος πρόβλεψη, ένας αλγόριθμος, όπως η ανάδραση, ρυθμίζει τα βάρη του δικτύου για να μειώσει το λάθος. Αυτή η διαδικασία επαναλαμβάνεται

πολλές φορές, και με τον καιρό, το δίκτυο γίνεται καλύτερο στο να αναγνωρίζει τα πρότυπα που εκπαιδεύτηκε να αναγνωρίζει. [3]

Μετά την εκπαίδευση, το νευρωνικό δίκτυο χρησιμοποιείται για να κάνει προβλέψεις ή να αναγνωρίζει πρότυπα σε νέα δεδομένα, μια διαδικασία γνωστή ως συμπέρασμα. Σε αυτή τη φάση, το δίκτυο δεν μαθαίνει από τα νέα δεδομένα· αντίθετα, εφαρμόζει απλώς την "γνώση" που απέκτησε κατά την εκπαίδευση. Το συμπέρασμα μπορεί να γίνει σε διάφορες συσκευές και πλατφόρμες, από τεράστια διακομιστήρια μέχρι ενσωματωμένα συστήματα.

1.2.3 Convolutional Neural Networks (CNNs)

Τα Convolutional Neural Networks (CNNs) είναι μια ειδική κατηγορία νευρωνικών δικτύων που έχουν σχεδιαστεί για την επεξεργασία δεδομένων με γνωστή διαστατική δομή, όπως είναι οι εικόνες. Ένα CNN αποτελείται από ένα σύνολο στρωμάτων, κάθε ένα από τα οποία εκτελεί διαφορετικές λειτουργίες. [4]



Σχήμα 2: Απεικόνιση των σταδίων ενός Convolutional Neural Network (CNN)

Ακολουθούν τα βασικά στρώματα σε ένα CNN:

- Input Layer: Το εισαγωγικό στρώμα είναι το πρώτο στρώμα σε ένα νευρωνικό δίκτυο και έχει ως βασική λειτουργία την αποδοχή των αρχικών δεδομένων που θα επεξεργαστεί το δίκτυο. Οι διαστάσεις του στρώματος ταιριάζουν με τις διαστάσεις της εισαγόμενης εικόνας.

- Convolutional Layers: Τα στρώματα συνέλιξης είναι το κύριο χαρακτηριστικό των CNNs και είναι υπεύθυνα για την εξαγωγή χαρακτηριστικών από τις εικόνες. Κατά τη συνέλιξη, ένα μικρό παράθυρο (γνωστό ως φίλτρο ή πυρήνας) γλιστράει πάνω από την εικόνα, κάνοντας σημειακούς πολλαπλασιασμούς με τα υποκείμενα pixel και συνοψίζοντας τα αποτελέσματα σε μια νέα χαρτογράφηση χαρακτηριστικών. Αυτή η διαδικασία επιτρέπει στο CNN να μαθαίνει αυτόματα σημαντικά χαρακτηριστικά από τα δεδομένα, όπως άκρες, γωνίες ή πιο σύνθετα μοτίβα.

- Pooling Layers: Τα στρώματα συγκέντρωσης ακολουθούν συνήθως τα στρώματα συνέλιξης και είναι σχεδιασμένα για να μειώνουν τις διαστάσεις της χαρτογράφησης χαρακτηριστικών. Αυτό επιτυγχάνεται μέσω διαδικασιών όπως η μέγιστη συγκέντρωση (max pooling), η οποία διαλέγει τη μέγιστη τιμή από μια ομάδα γειτονικών pixel, μειώνοντας έτσι τον όγκο των δεδομένων που πρέπει να επεξεργαστεί το δίκτυο.

- Fully Connected Layers: Τα πλήρως συνδεδεμένα στρώματα χρησιμοποιούνται στο τέλος του δικτύου. Σε αυτή τη φάση, τα χαρακτηριστικά που εξάγονται από τα στρώματα συνέλιξης και συγκέντρωσης συνδυάζονται για να σχηματίσουν τελικές προβλέψεις ή κατηγοριοποιήσεις.

Τα CNNs είναι ιδιαίτερα δημοφιλή σε εφαρμογές όπως η αναγνώριση εικόνων, η ανάλυση βίντεο και ακόμα και σε πιο προηγμένες εφαρμογές όπως η αυτόνομη οδήγηση. Η ικανότητά τους να εξάγουν σημαντικά χαρακτηριστικά από πολύπλοκα δεδομένα τα καθιστά έναν ισχυρό εργαλείο στον τομέα της βαθιάς μάθησης.

2 Παρουσίαση Προβήματος και Εργαλείων

Το πρώτο μέρος της παρούσας εργασίας εστιάζει στη χρήση νευρωνικών δικτύων, με έμφαση στα σύγχρονα μοντέλα CNN, YOLOv5 και YOLOv8, για την ανίχνευση αντικειμένων (object detection). Ο κύριος στόχος είναι ο εντοπισμός και η καταμέτρηση κινούμενων οχημάτων στο οδικό δίκτυο, χρησιμοποιώντας βιντεογραφικό υλικό από κάμερες.

Στο δεύτερο μέρος της εργασίας, επιχειρείται η επιτάχυνση του μοντέλου μέσω της χρήσης ενός FPGA, καθώς και στη διασύνδεσή του με την υπόλοιπη εφαρμογή. Για αυτόν τον σκοπό, χρησιμοποιήθηκε η πλατφόρμα Versal AI Core Series VCK190 της Xilinx.

2.1 Python Programming Language

Η Python είναι μια δημοφιλής, υψηλού επιπέδου γλώσσα προγραμματισμού, που χαρακτηρίζεται από την ευκολία μάθησης και την καθαρή σύνταξή της. Είναι διασκευασμένη για δυναμικό προγραμματισμό και αντικειμενοστραφή σχεδίαση, κάνοντάς την ιδανική για γρήγορη ανάπτυξη εφαρμογών και ως συγκολλητική γλώσσα για τη σύνδεση υπαρχόντων συστατικών. Υποστηρίζει επίσης μονάδες και πακέτα, ενθαρρύνοντας τη μοναδικότητα του προγράμματος και την επανάχρηση κώδικα. [5]

2.2 Pytorch

Το PyTorch, αναπτυγμένο αρχικά από τη Meta AI και τώρα μέρος της Linux Foundation, είναι ένα ανοιχτού κώδικα πλαίσιο μηχανικής μάθησης, βασισμένο στη βιβλιοθήκη Torch. Είναι γνωστό για την ισχυρή υποστήριξη στην εκπαίδευση νευρωνικών δικτύων με γραφικές μονάδες επεξεργασίας (GPU), ενώ προσφέρει δυνατότητες όπως η ταχεία ανάπτυξη μοντέλων, γρήγοροι χρόνοι εκπαίδευσης και ένα ισχυρό οικοσύστημα εργαλείων και βιβλιοθηκών. Το PyTorch χρησιμοποιεί την αυτόματη διαφοροποίηση (automatic differentiation) για αποτελεσματική εκπαίδευση μοντέλων και έχει κερδίσει μεγάλη δημοτικότητα στην κοινότητα της έρευνας μηχανικής μάθησης. [6] [7]

2.3 Jupyter Notebook

Το Jupyter Notebook είναι ένας δημοφιλής διαδικτυακός επεξεργαστής που επιτρέπει τη δημιουργία και τον διαμοιρασμό υπολογιστικών εγγράφων. Αυτό το εργαλείο υποστηρίζει πολλές γλώσσες προγραμματισμού, όπως Python, R, Julia, και Scala, και παρέχει μια ευέλικτη διεπαφή για τη διαμόρφωση και διαχείριση εργασιών σε τομείς όπως η επιστήμη δεδομένων, η επιστημονική υπολογιστική, το δημοσιογραφικό υπολογισμό και η μηχανική μάθηση. Τα Jupyter Notebooks είναι ένα ισχυρό εργαλείο για τη διαδραστική ανάπτυξη και παρουσίαση ερευνητικών εργασιών στον τομέα της επιστήμης δεδομένων, προσφέροντας τη δυνατότητα να

ενσωματωθεί κώδικας, κείμενο, εικόνες, γραφήματα και άλλα στοιχεία σε ένα ενιαίο έγγραφο. [8]

2.4 Google Colaboratory

Το Google Colaboratory, γνωστό ως Colab, είναι μια δωρεάν υπηρεσία βασισμένη στο cloud από τη Google, η οποία παρέχει ένα περιβάλλον Jupyter Notebook χωρίς την ανάγκη εγκατάστασης τοπικού λογισμικού. Είναι ιδιαίτερα κατάλληλο για εφαρμογές στη μηχανική μάθηση, την επιστήμη δεδομένων και την εκπαίδευση. Το Colab παρέχει πρόσβαση σε υπολογιστικούς πόρους, όπως GPUs και CPUs, και επιτρέπει τη δημιουργία συνδυαστικών εγγράφων που περιλαμβάνουν εκτελέσιμο κώδικα Python, κείμενο, γραφήματα, εικόνες, HTML και LaTeX. [9]

2.5 Vitis AI

Το Vitis AI είναι μια πλατφόρμα ανάπτυξης AI από την AMD, η οποία προσφέρει μια ολοκληρωμένη λύση για την επιτάχυνση της AI από την άκρη (edge) έως το cloud. Υποστηρίζει δημοφιλή πλαίσια μηχανικής μάθησης και προσφέρει ισχυρά εργαλεία για τη βελτιστοποίηση της ακρίβειας και της αποδοτικότητας των μοντέλων. Είναι σχεδιασμένο για να απλοποιεί την ανάπτυξη AI εφαρμογών και να αξιοποιεί τις δυνατότητες της επιτάχυνσης AI σε πλατφόρμες της AMD.

Η πλατφόρμα περιλαμβάνει μια σειρά από βελτιστοποιημένα AI μοντέλα και παρέχει υψηλού επιπέδου βιβλιοθήκες και API για την αποτελεσματική ανίχνευση AI. Επιπλέον, το Vitis AI διαθέτει ευέλικτους πυρήνες DPU (Deep Learning Processor Unit) για διάφορες απαιτήσεις όσον αφορά τη ροή, την καθυστέρηση και την κατανάλωση ενέργειας. [10]

2.6 ByteTrack

Το ByteTrack είναι μια πρωτοποριακή τεχνολογία στον τομέα της υπολογιστικής όρασης, ειδικά σχεδιασμένη για την πολύπλοκη εργασία της πολυ-αντικειμενικής παρακολούθησης (Multi-Object Tracking, MOT). Αυτός ο καινοτόμος αλγόριθμος AI είναι σχεδιασμένος για να αναθέτει μοναδικούς αναγνωριστικούς σε αντικείμενα μέσα σε ένα βίντεο, επιτρέποντας τη συνεπή και ακριβή παρακολούθηση κάθε αντικειμένου με την πάροδο του χρόνου.

Το ByteTrack ξεπερνά τις παραδοσιακές μεθόδους παρακολούθησης χρησιμοποιώντας προηγμένες τεχνικές AI. Έχει χτιστεί βασισμένο σε μια βαθιά κατανόηση του πώς κινούνται και αλληλεπιδρούν τα αντικείμενα σε ένα δυναμικό περιβάλλον, κάνοντάς το ιδιαίτερα ικανό στο να χειρίζεται σενάρια που θα συγχέουν τα συμβατικά συστήματα παρακολούθησης.

Το ByteTrack χρησιμοποιεί τις ανιχνεύσεις σε συνεχόμενα καρέ για να παρακολουθεί τα αντικείμενα, εφαρμόζοντας βήματα όπως η ανίχνευση με υψηλή εμπιστοσύνη και η συσχέτιση των ανιχνεύσεων χαμηλότερης εμπιστοσύνης, οι οποίες συνήθως αγνοούνται. Αυτό είναι κρίσιμο για τη διατήρηση της ταυτότητας των αντικειμένων όταν είναι κρυμμένα ή δεν είναι ξεκάθαρα ορατά. [11]

2.7 Roboflow Supervision

Το Roboflow Supervision είναι ένα ισχυρό εργαλείο που σχεδιάστηκε για να ενισχύει τις δυνατότητες των μοντέλων ανίχνευσης και τμηματοποίησης αντικειμένων που χρησιμοποιούνται στην ανάλυση βίντεο. Αυτή η τεχνολογία επιτρέπει στους χρήστες να αναγνωρίζουν και να παρακολουθούν εύκολα αντικείμενα που έχουν εντοπιστεί από διάφορα μοντέλα, απλοποιώντας τη διαδικασία παρακολούθησης και ανάλυσης. Επίσης, προσφέρει προηγμένες δυνατότητες φιλτραρίσματος, δίνοντας στον χρήστη έναν ευέλικτο και αποδοτικό τρόπο για να περιορίσει και να διευκρινίσει τις ανιχνεύσεις αντικειμένων. Αυτό περιλαμβάνει διάφορες μεθόδους φιλτραρίσματος, όπως το φιλτράρισμα ανά συγκεκριμένη κλάση ή σύνολο κλάσεων, την εμπιστοσύνη, το εμβαδόν του αντικειμένου, το εμβαδόν του πλαισίου οριοθέτησης, το σχετικό εμβαδόν, τις διαστάσεις του πλαισίου και τις ορισμένες ζώνες. [12]

2.8 YOLO

Το YOLO (You Only Look Once) είναι μια σειρά από μοντέλα για ανίχνευση αντικειμένων σε εικόνες και βίντεο. Αυτά τα μοντέλα είναι γνωστά για την ταχύτητα και την ακρίβειά τους και έχουν εξελιχθεί σημαντικά με την πάροδο του χρόνου.
Ακολουθεί μια τεχνική ανάλυση των διάφορων εκδόσεων των YOLO μοντέλων: [13] [14]

- YOLOv1 (2016): Το αρχικό YOLO χρησιμοποιεί μια μοναδική convolutional neural network (CNN) για να προβλέψει κλάσεις και τοποθεσίες αντικειμένων. Η καινοτομία του ήταν η χρήση ενός single-pass algorithm, που το καθιστούσε σημαντικά ταχύτερο από πολλαπλά-pass systems όπως το R-CNN. Ωστόσο, είχε χαμηλότερη ακρίβεια στην ανίχνευση μικρών αντικειμένων και σε συνθήκες υπερκάλυψης (overlap).

- YOLOv2 και YOLO9000 (2017): Το YOLOv2, γνωστό και ως Darknet-19, βελτίωσε την ακρίβεια και εισήγαγε νέες τεχνικές όπως anchor boxes για καλύτερη πρόβλεψη των διαστάσεων των bounding boxes. Το YOLO9000 επεκτάθηκε στην ανίχνευση πολλαπλών κλάσεων, με δυνατότητα αναγνώρισης πάνω από 9000 διαφορετικών αντικειμένων.

- YOLOv3 (2018): Χρησιμοποιώντας το Darknet-53 ως backbone, το YOLOv3 παρουσίασε βελτιωμένη ακρίβεια, κυρίως μέσω της χρήσης τριών διαφορετικών κλιμάκων για ανίχνευση (multi-scale detection). Είχε καλύτερη απόδοση στην ανίχνευση μικρών αντικειμένων και συνέχισε να διατηρεί υψηλές ταχύτητες.

- YOLOv4 (2020): Το YOLOv4 έφερε πολλαπλές βελτιώσεις, όπως τη χρήση του CSPDarknet53 ως backbone, τεχνικές όπως Mish activation, cross-stage partial connections (CSP), και spatial pyramid pooling. Αυτές οι καινοτομίες βελτίωσαν την ακρίβεια και επιτρέπουν αποδοτικότερη χρήση σε μη εξειδικευμένο hardware.

- YOLOv5 (2020): Παρά την ανεπίσημη φύση του, το YOLOv5 έγινε δημοφιλές λόγω της ευκολίας της χρήσης και των γρήγορων χρόνων εκπαίδευσης. Ενσωματώνει τεχνικές όπως automatic mixed precision (AMP) και model pruning για αυξημένη αποδοτικότητα.

- YOLOv6 (2022): Η έκδοση αυτή, προσφέρει βελτιωμένη ακρίβεια και ταχύτητα, με έμφαση στην ευελιξία και την προσαρμοστικότητα σε διάφορα είδη υλικού και εφαρμογών.

- YOLOv7 (2022): Το YOLOv7 χρησιμοποιεί προηγμένες τεχνικές όπως ELAN (Efficient Layer Aggregation Network) για βελτιωμένη ανίχνευση, SPPCSPC (Spatial Pyramid Pooling Convolutional Spatial Pyramid), μια νέα δομή SPP για την ενοποίηση χαρακτηριστικών, και MP (Max Pooling και συνελίξεις με

stride=2) για τη μείωση διαστάσεων. Επιπλέον, συνδυάζει στρατηγικές εκπαίδευσης από το YOLOv5 και το YOLOX για την κατανομή δειγμάτων.

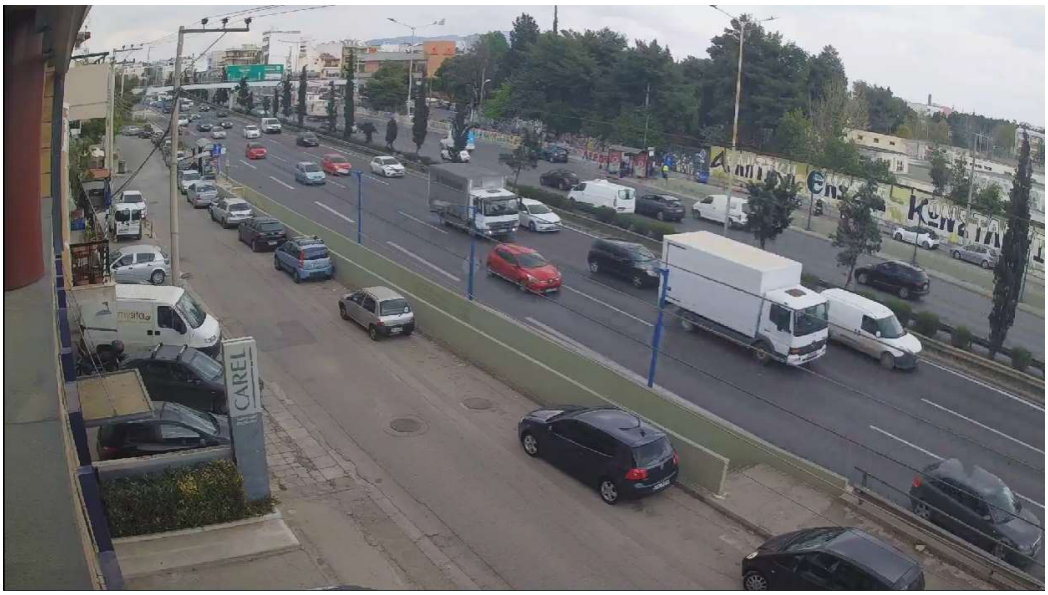
– YOLOv8 (2023): Το YOLOv8 χρησιμοποιεί έναν τροποποιημένο CSPDarknet53 ως τη βάση του (backbone) και ενσωματώνει τεχνικές αυτό-προσοχής στο κεφάλι του δικτύου, επιτρέποντας την εστίαση σε διαφορετικά μέρη της εικόνας και την προσαρμογή της σημασίας διαφορετικών χαρακτηριστικών ανάλογα με τη σημασία τους στο εκάστοτε καθήκον. Το μοντέλο υποστηρίζει επίσης ανίχνευση πολλαπλών κλιμάκων, χρησιμοποιώντας ένα δίκτυο πυραμίδας χαρακτηριστικών για την ανίχνευση αντικειμένων διαφόρων μεγεθών και κλιμάκων.

Σε σχέση με άλλα μοντέλα, όπως το Faster R-CNN ή το SSD, τα YOLO μοντέλα ξεχωρίζουν για την εκπληκτική ταχύτητά τους, πράγμα που τα καθιστά ιδανικά για εφαρμογές πραγματικού χρόνου. Με τις τελευταίες εκδόσεις, η ακρίβεια των YOLO μοντέλων έχει πλησιάσει ή ακόμη και υπερβεί αυτή των ανταγωνιστικών μοντέλων, ενώ παραμένουν εύκολα στην εκπαίδευση και την εφαρμογή.

3 Ανάπτυξη Εφαρμογής

3.1 Δεδομένα εισόδου

Για την εφαρμογή χρησιμοποιήθηκε βίντεο από έναν πολυσύχναστο δρόμο της Αθήνας, το οποίο αποκτήθηκε μέσω της ιστοσελίδας Skyline Webcams. [15] Το βίντεο αυτό περιέχει διάφορα είδη οχημάτων, όπως αυτοκίνητα, μηχανάκια και φορτηγά, που κινούνται μέσα στο οδικό δίκτυο της πόλης. Αυτή η επιλογή δεδομένων είναι ιδιαίτερα ενδεικτική για το σκοπό της εργασίας, καθώς παρέχει μια πλούσια ποικιλία από πραγματικά σενάρια κυκλοφορίας σε αστικό περιβάλλον. Ο πολυσύχναστος χαρακτήρας του δρόμου προσθέτει στην πολυπλοκότητα της εργασίας, απαιτώντας από το μοντέλο να είναι ακριβές και αξιόπιστο στον εντοπισμό και την καταμέτρηση των οχημάτων, παρά τις πολυάριθμες προκλήσεις όπως η συμφόρηση της κυκλοφορίας, οι διαφορετικοί τύποι οχημάτων και οι δυναμικές συνθήκες φωτισμού. Ένα ενδεικτικό καρέ φαίνεται παρακάτω.



Σχήμα 3: Παράδειγμα ενός καρέ από το βίντεο εισόδου

3.2 Διασύνδεση μοντέλου YOLO και ByteTrack

Το εργαλείο ByteTrack χρειάζεται τις συντεταγμένες των bounding boxes σε μορφή διαφορετική από αυτή που παράγεται από το μοντέλο YOLO. Συνεπώς χρησιμοποιείται το ακόλουθο κομμάτι κώδικα για την αντιστοίχιση των bounding boxes που επιστρέφονται από το μοντέλο ανίχνευσης αντικειμένων (YOLO) με τις περιοχές που παρακολουθούνται από τον αλγόριθμο παρακολούθησης (ByteTrack):

```
from typing import List
import numpy as np

# converts Detections into format that can be consumed by match_detec-
tions_with_tracks function
def detections2boxes(detections: Detections) -> np.ndarray:
    return np.hstack((
        detections.xyxy,
        detections.confidence[:, np.newaxis]
    ))

# converts List[STrack] into format that can be consumed by match_detec-
tions_with_tracks function
def tracks2boxes(tracks: List[STrack]) -> np.ndarray:
    return np.array([
        track.tlbr
        for track
        in tracks
```

```
    ], dtype=float)

# matches our bounding boxes with predictions
def match_detections_with_tracks(
    detections: Detections,
    tracks: List[STrack]
) -> Detections:
    if not np.any(detections.xyxy) or len(tracks) == 0:
        return np.empty((0,))

    tracks_boxes = tracks2boxes(tracks=tracks)
    iou = box_iou_batch(tracks_boxes, detections.xyxy)
    track2detection = np.argmax(iou, axis=1)

    tracker_ids = [None] * len(detections)

    for tracker_index, detection_index in enumerate(track2detection):
        if iou[tracker_index, detection_index] != 0:
            tracker_ids[detection_index] = tracks[tracker_index].track_id

    return tracker_ids
```

- Η συνάρτηση detections2boxes μετατρέπει τις ανιχνεύσεις (τύπου Detections) σε έναν πίνακα numpy. Κάθε ανίχνευση περιλαμβάνει τις συντεταγμένες του bounding box (xyxy) και το επίπεδο εμπιστοσύνης (confidence).
- Η συνάρτηση tracks2boxes μετατρέπει μια λίστα από αντικείμενα παρακολούθησης (List[STrack]) σε έναν πίνακα numpy. Κάθε STrack αντιπροσωπεύει μια παρακολουθούμενη περιοχή και περιέχει τις συντεταγμένες της (tlbr).
- Η συνάρτηση match_detections_with_tracks είναι η βασική λειτουργία για την αντιστοίχιση. Χρησιμοποιεί τις παραπάνω συναρτήσεις για να μετατρέψει τόσο τις ανιχνεύσεις όσο και τις παρακολουθήσεις σε κατάλληλες μορφές. Στη συνέχεια, χρησιμοποιεί τη συνάρτηση box_iou_batch για να υπολογίσει τον δείκτη επικάλυψης (Intersection over Union - IoU) μεταξύ των ανιχνεύσεων και των παρακολουθήσεων. Βάσει αυτού του δείκτη, γίνεται η αντιστοίχιση μεταξύ των δύο, και κάθε ανίχνευση αντιστοιχίζεται με την πιο κοντινή της παρακολούθηση.

3.3 Ανίχνευση οχημάτων

Το ακόλουθο τμήμα του κώδικα είναι υπεύθυνο για την πρόβλεψη και την επισημείωση (annotation) ενός ολόκληρου βίντεο.

```
VideoInfo.from_video_path(SOURCE_VIDEO_PATH)
from tqdm.notebook import tqdm

# create BYTETracker instance
byte_tracker = BYTETracker(BYTETrackerArgs())
# create VideoInfo instance
video_info = VideoInfo.from_video_path(SOURCE_VIDEO_PATH)
# create frame generator
generator = get_video_frames_generator(SOURCE_VIDEO_PATH)
# create LineCounter instance
line_counter = LineCounter(start=LINE_START, end=LINE_END)
# create instance of BoxAnnotator and LineCounterAnnotator
box_annotator = BoxAnnotator(color=ColorPalette(), thickness=2, text_thick-
ness=1, text_scale=0.5)
line_annotator = LineCounterAnnotator(thickness=2, text_thickness=1,
text_scale=1)

coord_transform = np.array([1920,1080,1920,1080])

# open target video file
with VideoSink(TARGET_VIDEO_PATH, video_info) as sink:
    # loop over video frames
    for frame in tqdm(generator, total=video_info.total_frames):
        # model prediction on single frame and conversion to supervision
Detections
        results = model(frame)
        yolo_coords = np.delete(results.xyxyn[0][:, :-1].cpu().numpy(), 4,
axis=1)
        xyxy_coords = yolo_coords*coord_transform
        detections = Detections(
            xyxy=xyxy_coords,
            confidence=results.xyxyn[0][:, :-1][:, -1].cpu().numpy(),
            class_id=results.xyxyn[0][:, -1].cpu().numpy().astype(int)
        )
        # filtering out detections with unwanted classes
        mask = np.array([class_id in CLASS_ID for class_id in detec-
tions.class_id], dtype=bool)
        detections.filter(mask=mask, inplace=True)
        # tracking detections
        tracks = byte_tracker.update(
            output_results=detections2boxes(detections=detections),
            img_info=frame.shape,
            img_size=frame.shape
        )
        tracker_id = match_detections_with_tracks(detections=detections,
tracks=tracks)
        detections.tracker_id = np.array(tracker_id)
        # filtering out detections without trackers
        mask = np.array([tracker_id is not None for tracker_id in detec-
tions.tracker_id], dtype=bool)
        detections.filter(mask=mask, inplace=True)
        # format custom labels
        labels = [
            f"#{tracker_id} {CLASS_NAMES_DICT[class_id]} {confidence:0.2f}"
            for _, confidence, class_id, tracker_id
```

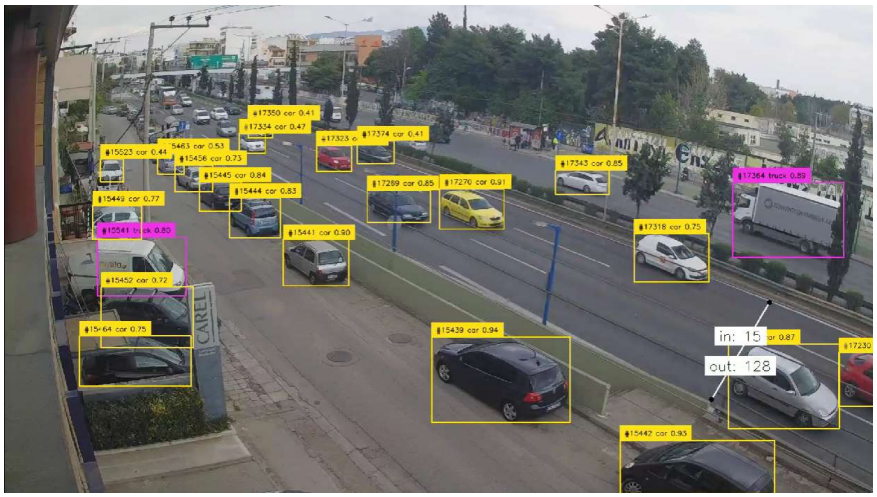
```
            in detections
        ]
        # updating line counter
        line_counter.update(detections=detections)
        # annotate and display frame
        frame = box_annotator.annotate(frame=frame, detections=detections,
labels=labels)
        line_annotator.annotate(frame=frame, line_counter=line_counter)
        sink.write_frame(frame)
```

Ακολουθεί μια σύντομη περιγραφή των βασικών βημάτων:

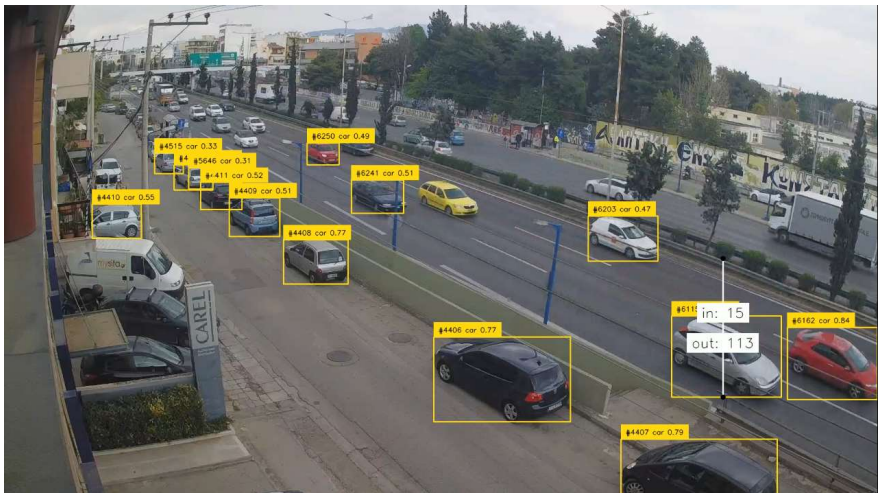
- Επεξεργασία Καρέ-Καρέ: Το βίντεο επεξεργάζεται καρέ-καρέ. Σε κάθε καρέ γίνεται πρόβλεψη των οχημάτων με χρήση του μοντέλου YOLO και τα αποτελέσματα μετατρέπονται στην κατάλληλη μορφή (detections).
- Φιλτράρισμα και Παρακολούθηση Ανιχνεύσεων: Οι ανιχνεύσεις φιλτράρονται βάσει των επιθυμητών κατηγοριών οχημάτων και συνδέονται με τα αντίστοιχα detections που διαχειρίζεται το ByteTrack.
- Επισημείωση και Καταγραφή: Κάθε καρέ επισημαίνεται με τις ανιχνεύσεις και τις πληροφορίες παρακολούθησης, και στη συνέχεια καταγράφεται στο τελικό βίντεο.

3.4 Βελτιστοποίηση παραμέτρων της εφαρμογής

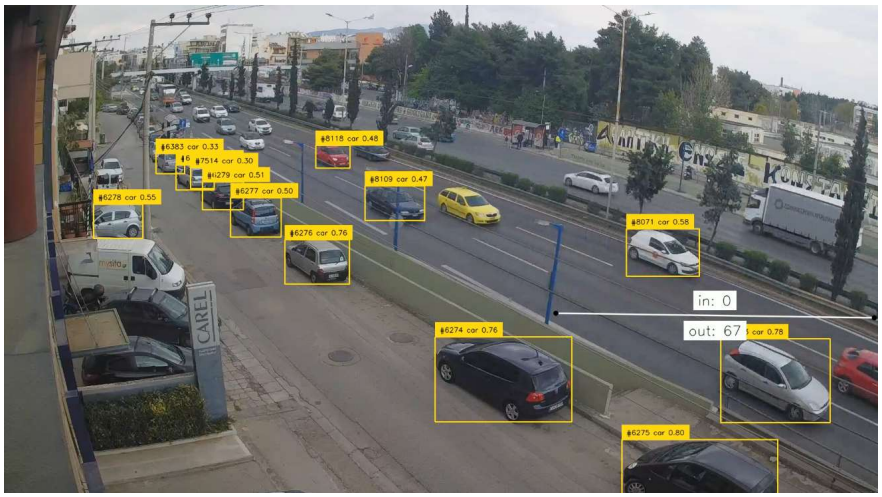
Ένας παράγοντας που αξίζει αναφορά ήταν η επιλογή της θέσης και του προσανατολισμού μιας εικονικής γραμμής καταμέτρησης. Αυτή η γραμμή λειτουργεί ως ένα είδος φραγμού, όπου τα οχήματα που διασχίζουν αυτήν τη γραμμή καταγράφονται και καταμετρώνται (αριθμός “out”). Η επιλογή των συντεταγμένων της γραμμής ήταν αποτέλεσμα εκτενούς δοκιμών και πειραματισμού. Πολλές διαφορετικές θέσεις και γωνίες εξετάστηκαν για να βρεθεί η βέλτιστη θέση που μεγιστοποιεί την ακρίβεια της καταμέτρησης, ενώ ταυτόχρονα ελαχιστοποιεί τα λάθη ή τις ψευδείς ανιχνεύσεις. Η τελική επιλογή των συντεταγμένων αυτών αντικατοπτρίζει την ιδανική ισορροπία μεταξύ αυτών των παραμέτρων, καθιστώντας την ανάλυση πιο αξιόπιστη και αποτελεσματική.



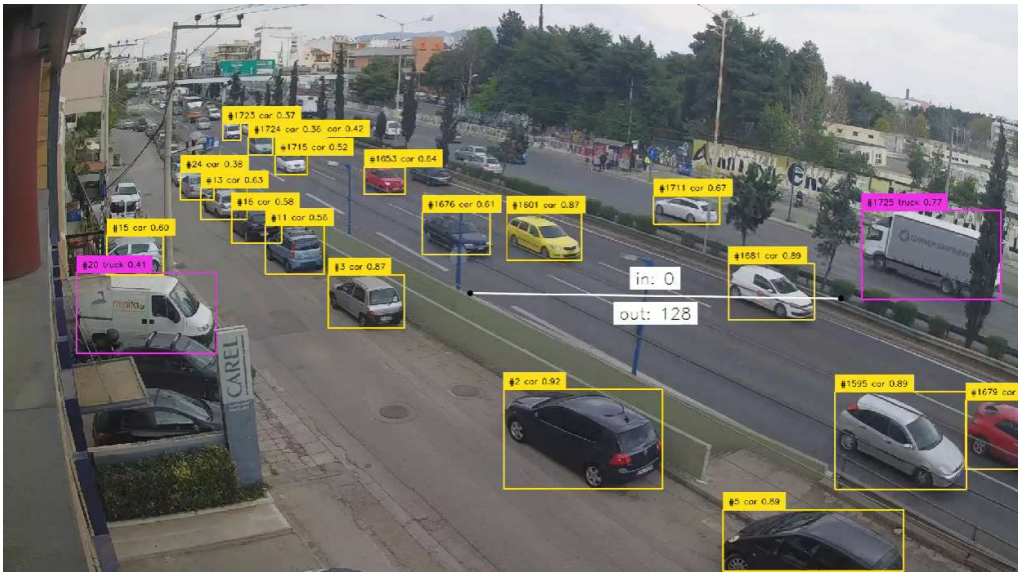
Σχήμα 4: Εναλλακτική Α



Σχήμα 5: Εναλλακτική Β



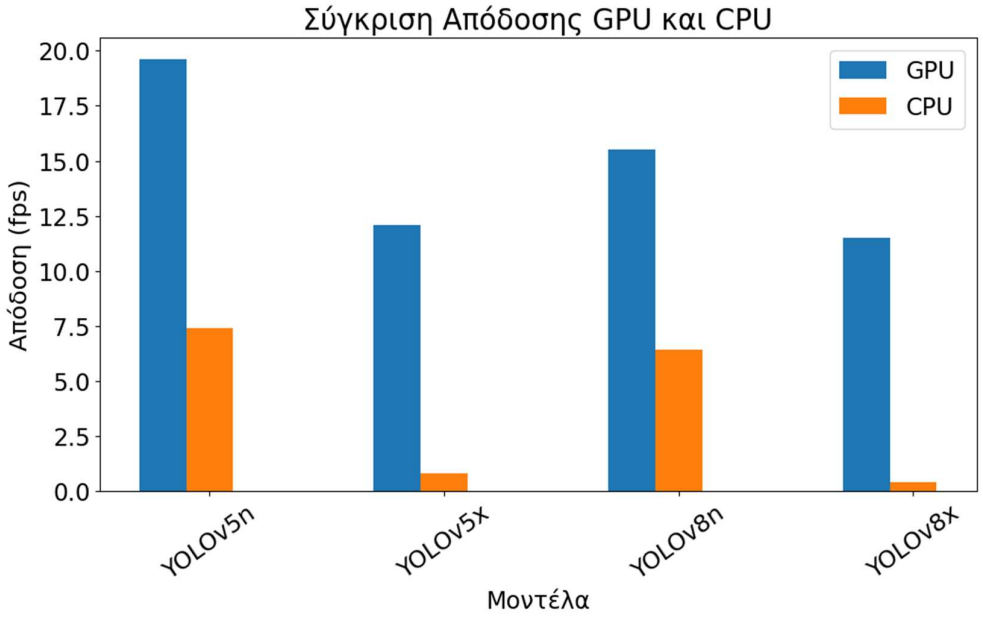
Σχήμα 6: Εναλλακτική Γ



Σχήμα 7: Τελική επιλογή

3.5 Σύγκριση μεταξύ GPU και CPU

Το παρακάτω γράφημα αποκαλύπτει κάποιες σημαντικές διαφορές στην απόδοση μεταξύ των διαφορετικών υπολογιστικών πόρων (GPU, CPU) για τα μοντέλα YOLOv5 και YOLOv8.



Σχήμα 8: Σύγκριση υπολογιστικών πόρων

Συγκεκριμένα:

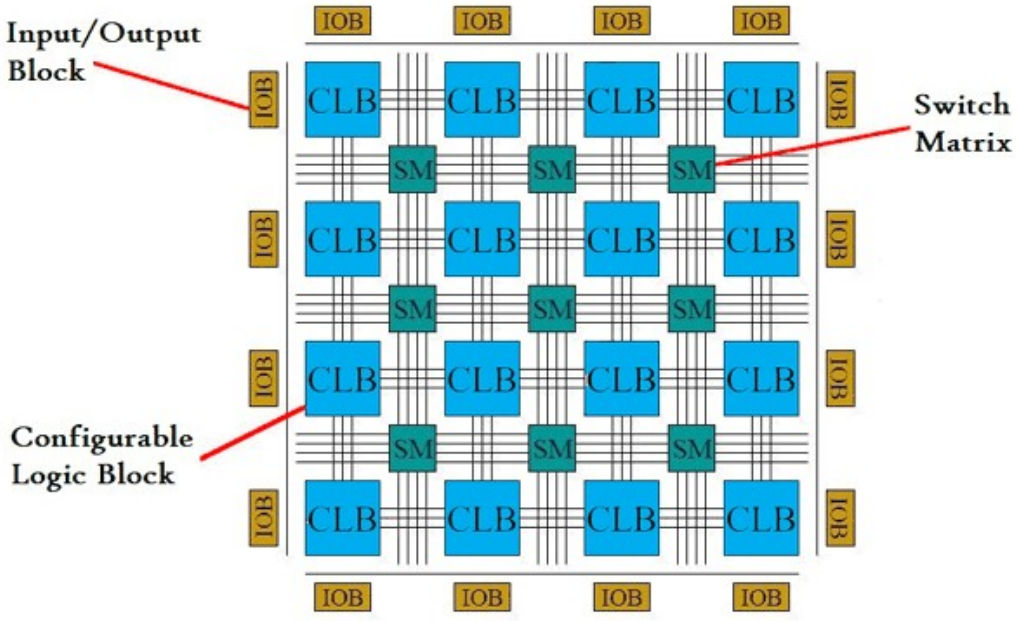
- GPU (Tesla T4): Σε όλα τα μοντέλα, η GPU παρουσιάζει την υψηλότερη απόδοση σε fps. Αυτό υποδηλώνει ότι η GPU μπορεί να επεξεργαστεί ταχύτερα τις εικόνες, καθιστώντας την ιδανική για πραγματικού χρόνου εφαρμογές.
- CPU: Η απόδοση της CPU είναι σημαντικά χαμηλότερη σε σύγκριση με την GPU. Αυτό είναι αναμενόμενο, καθώς δεν είναι εξειδικευμένες για παράλληλες εργασίες υψηλής επεξεργαστικής ισχύος όπως οι GPUs. Η μειωμένη απόδοση στα μεγαλύτερα μοντέλα (YOLOv5x, YOLOv8x) επιβεβαιώνει περαιτέρω τις περιορισμένες δυνατότητες της CPU για απαιτητικές εφαρμογές.

Για το υπόλοιπο της εργασίας θα εστιάσουμε μόνο στα αποτελέσματα που δόθηκαν από τη χρήση της T4 GPU.

4 Επιτάχυνση Υπολογισμών Μοντέλου

4.1 Field Programmable Gate Arrays (FPGAs)

Τα FPGA (Field-Programmable Gate Arrays) είναι εξαιρετικά ευέλικτα ηλεκτρονικά κυκλώματα που μπορούν να προγραμματιστούν για να εκτελέσουν οποιοδήποτε λογικό κύκλωμα ο χρήστης επιθυμεί. Αυτό τα καθιστά ιδανικά για πληθώρα εφαρμογών, από ψηφιακή σήμανση και επεξεργασία εικόνας έως πιο προηγμένα συστήματα όπως η επιταχυνόμενη Μηχανική Μάθηση και η κρυπτογράφηση. Ένα από τα κύρια πλεονεκτήματα των FPGA είναι η δυνατότητά τους για υψηλές επιδόσεις και χαμηλή κατανάλωση ενέργειας, ειδικά σε εφαρμογές που απαιτούν παράλληλη επεξεργασία (όπως εφαρμογές νευρωνικών δικτύων). Σε αντίθεση με τους παραδοσιακούς μικροεπεξεργαστές, ένα FPGA μπορεί να προσαρμοστεί στην ειδική λειτουργία που απαιτείται για μια δεδομένη εφαρμογή, παρέχοντας μεγαλύτερη ευελιξία και συχνά καλύτερη απόδοση. Επιπλέον, η δυνατότητα προγραμματισμού τους επιτρέπει την αναβάθμιση ή την αλλαγή λειτουργίας χωρίς την ανάγκη για φυσική αντικατάσταση του υλικού, διαφέροντας έτσι από τα Application Specific Integrated Circuits (ASICs), τα οποία είναι κατασκευασμένα για συγκεκριμένες σχεδιαστικές εργασίες. [16] [17]

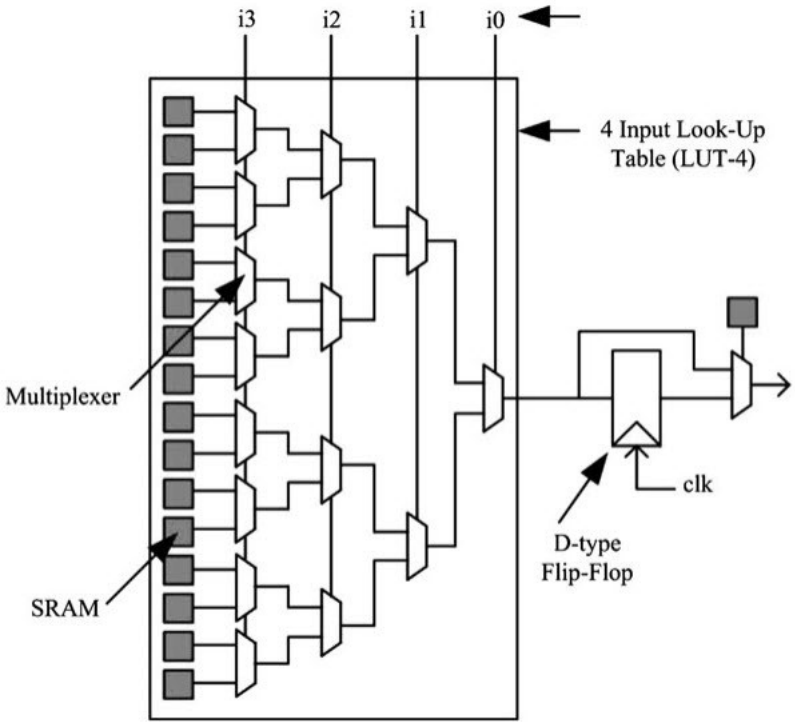


Σχήμα 9: Η δομή ενός FPGA

4.1.1 Εσωτερκή αρχιτεκτονική

Η βασική δομή ενός FPGA περιλαμβάνει τρία κύρια στοιχεία:

- Προγραμματιζόμενα Λογικά Κύτταρα (ή Λογικά Μπλοκ): Αποτελούν τα βασικά δομικά στοιχεία ενός FPGA και είναι υπεύθυνα για την εκτέλεση των βασικών λογικών λειτουργιών. Σε διάφορα FPGA, αυτά τα λογικά μπλοκ μπορεί να είναι γνωστά ως Configurable Logic Blocks (CLBs) ή Logic Array Blocks (LABs). Περιλαμβάνουν βασικά στοιχεία όπως τα Look-up Tables (LUTs), τα οποία αποτελούνται από SRAM και Multiplexors, και Flip-flops, επιτρέποντας την υλοποίηση διαφόρων λογικών λειτουργιών.



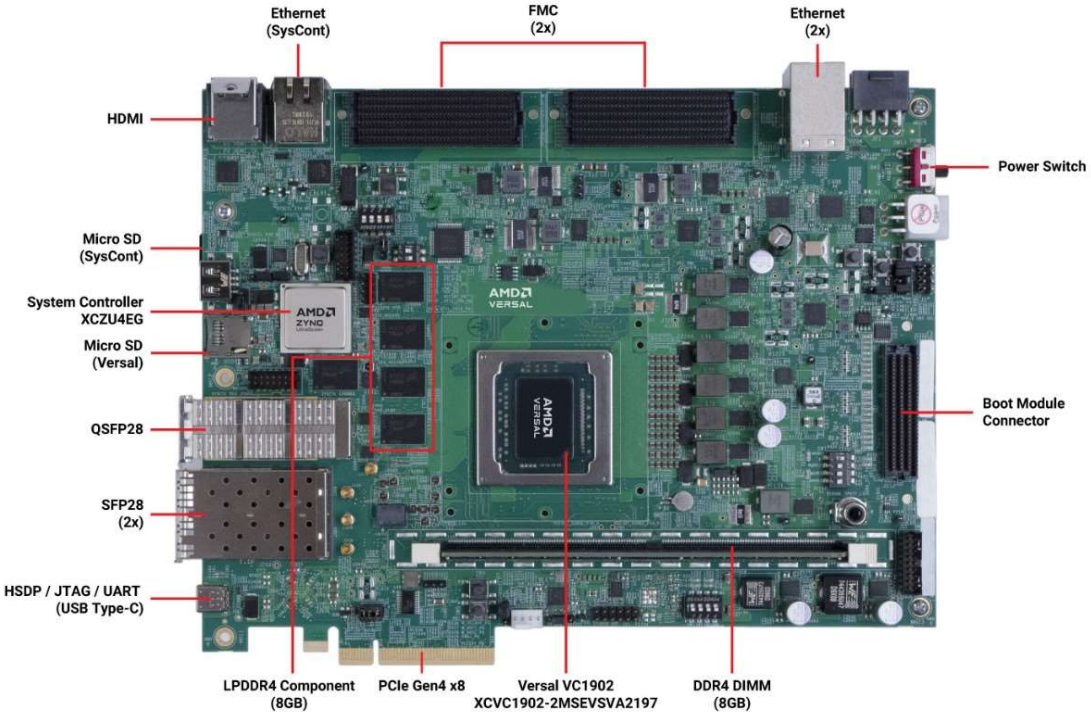
Σχήμα 10: Παράδειγμα ενός Configurable Logic Block (CLB)

- Προγραμματιζόμενο Δίκτυο Δρομολόγησης: Αυτό το συστατικό διασυνδέει τα Λογικά Μπλοκ εντός του FPGA. Αποτελείται από συνδετήρια καλώδια και προγραμματιζόμενους διακόπτες, οι οποίοι μπορούν να διαμορφωθούν για να εγκαθιδρύσουν συνδέσεις μεταξύ διαφόρων λογικών μπλοκ και I/O μπλοκ. Υπάρχουν δύο κύριοι τύποι αρχιτεκτονικής δρομολόγησης που χρησιμοποιούνται στα FPGA: το Island Style Routing, όπου τα λογικά μπλοκ τοποθετούνται σε δισδιάστατο πίνακα και διασυνδέονται μέσω ενός προγραμματιζόμενου δικτύου δρομολόγησης, και το Hierarchical Routing, που χωρίζει τα λογικά μπλοκ σε ομάδες ή συστάδες για συνδέσεις σε διαφορετικά επίπεδα ιεραρχίας.

- Μπλοκ Εισόδου/Εξόδου: Αυτά τα μπλοκ συνδέουν την εσωτερική αρχιτεκτονική του FPGA με τον εξωτερικό κόσμο. Περιλαμβάνουν συνήθως στοιχεία όπως flip-flops D και είναι υπεύθυνα για λειτουργίες εισόδου και εξόδου.

4.2 Versal AI Core Series VCK190

Το Versal VCK190 είναι μια καινοτόμος πλατφόρμα ανάπτυξης που προσφέρει ανώτερες δυνατότητες στον τομέα του AI και της επεξεργασίας σημάτων. Ανήκει στη σειρά Versal™ AI Core της Xilinx και βασίζεται στο Adaptive SoC VC1902,



Σχήμα 11: Versal™ AI Core XCVC1902-2MSEVSVA2197 ACAP

που συνδυάζει επεξεργαστική ισχύ που υπερβαίνει κατά πολύ τις συμβατικές server class CPU. Είναι εξοπλισμένο με Dual-Core Arm Cortex-A72 APU και Dual-Core Arm Cortex-R5F RTPU, προσφέροντας προηγμένες δυνατότητες για την ανάλυση και επεξεργασία μεγάλων όγκων δεδομένων.

Η πλατφόρμα VCK190 χαρακτηρίζεται από την ισχυρή της προγραμματιζόμενη λογική και την ικανότητά της να υποστηρίζει πολύπλοκες υπολογιστικές διαδικασίες, όπως την ανάλυση AI και την επεξεργασία σημάτων σε πραγματικό χρόνο. Η ενσωμάτωση της προγραμματιζόμενης λογικής με τον επεξεργαστή επιτρέπει την αποδοτική διαχείριση των υπολογιστικών πόρων, βελτιστοποιώντας την απόδοση για απαιτητικές εφαρμογές.

Το VCK190 διαθέτει ποικιλία διεπαφών συνδεσιμότητας, όπως PCIe Gen4, DDR4 και LPDDR4 για τη μνήμη, καθώς και είσοδο/έξοδο βίντεο μέσω HDMI. Η υποστήριξη των διασυνδέσεων Ethernet και οι οπτικές διεπαφές SFP28 και QSFP28 ενισχύουν την ευελιξία της πλατφόρμας για διάφορες εφαρμογές δικτύωσης. Η παρουσία επίσης των FMC expansion connectors προσφέρει επιπλέον δυνατότητες επέκτασης. [18]

AI Engines	400
DSP Engines	1,968
System Logic Cells (K)	1,968
LUTs	899,840
Application Processing Unit	Dual-Core Arm® Cortex®-A72
Real-Time Processing Unit	Dual-Core Arm Cortex-R5F
Maximum I/O Pins	770
Programmable NoC Ports	28
Integrated Memory Controllers	4

Σχήμα 12: Προδιαγραφές του Versal VCK190

4.3 Διαδικασία Επιτάχυνσης των Μοντέλων YOLO

4.3.1 Custom Training YOLOv5 και YOLOv8

Για την επιτυχή ενσωμάτωση των YOLOv5 και YOLOv8 στο Versal VCK190, απαιτήθηκαν κρίσιμες προσαρμογές λόγω των περιορισμών του hardware. Συγκεκριμένα, συμβουλεύοντας το έγγραφο οδηγιών του DPUCVDX8G (κωδική ονομασία του VCK190) για Versal ACAPs [19] παρατηρείται πως μόνο ένα περιορισμένο σετ ενεργοποιήσεων (ReLU, LeakyReLU, ReLU6, Hard-Swish, Hard-Sigmoid) υποστιρίζεται. Συνεπώς χρειάστηκε να αντικατασταθεί η λειτουργία ενεργοποίησης SiLU (Sigmoid-weighted Linear Unit), την οποία χρησιμοποιούν τα YOLOv5 και YOLOv8, με τη LeakyReLU. Οι συγκεκριμένες αλλαγές φαίνονται παρακάτω.

```
# changes in files models/experimental.py
# and models/common.py

# Before
self.act = nn.SiLU()

# After
self.act = nn.LeakyReLU(0.1, inplace=True)
```

Σχήμα 13: Αλλαγές στον πηγαίο κώδικα του YOLOv5

```
# changes in file ultralytics/nn/modules/conv.py

# Before
default_act = nn.SiLU()  # default activation
# After
default_act = nn.LeakyReLU(0.1, inplace=True)  # default activation


# changes in file ultralytics/nn/modules/block.py

# Before
self.act = nn.SiLU()
# After
self.act = nn.LeakyReLU(0.1, inplace=True)
```

Σχήμα 14: Αλλαγές στον πηγαίο κώδικα του YOLOv8

Έπειτα, έγινε training των μοντέλων στο Google Colab, για το οποίο παρατίθεται αναλυτικά ο κώδικας στο παράρτημα Α'.

4.3.2 Quantization των μοντέλων

Το Quantization (ή κβαντισμός) στο πεδίο των νευρωνικών δικτύων αποτελεί μια σημαντική τεχνική για την επιτάχυνση της εκτέλεσης των μοντέλων και τη μείωση τους απαιτήσεων σε αποθηκευτικό χώρο και επεξεργαστική ισχύ. Ειδικότερα, ο κβαντισμός μετατρέπει τα δεδομένα και τις παραμέτρους του μοντέλου από υψηλής ακρίβειας μορφές (όπως 32-bit κινητής υποδιαστολής) σε μορφές χαμηλότερης ακρίβειας (όπως 8-bit ακέραιοι), διατηρώντας παράλληλα την αποτελεσματικότητα του μοντέλου όσο το δυνατόν περισσότερο.

Οφέλη του Κβαντισμού:

- Μείωση Μεγέθους Μοντέλου: Ο κβαντισμός μπορεί να μειώσει σημαντικά το μέγεθος ενός νευρωνικού δικτύου, καθιστώντας το πιο ελκυστικό για

εφαρμογές σε περιβάλλοντα με περιορισμένο αποθηκευτικό χώρο και μνήμη, όπως οι ενσωματωμένες συσκευές και τα smartphones.

- Επιτάχυνση Inference: Με τη μετατροπή σε χαμηλότερη ακρίβεια, οι πράξεις είναι πιο γρήγορες, καθώς απαιτούνται λιγότεροι υπολογιστικοί πόροι. Αυτό συμβάλλει στη βελτίωση των χρόνων απόκρισης σε εφαρμογές πραγματικού χρόνου.

- Εξοικονόμηση Ενέργειας: Οι χαμηλότερες απαιτήσεις υπολογιστικής ισχύος μεταφράζονται σε μειωμένη κατανάλωση ενέργειας, κάτι ιδιαίτερα σημαντικό για φορητές και ενσωματωμένες συσκευές που λειτουργούν με μπαταρία.

Προκλήσεις:

- Διατήρηση Απόδοσης: Η μεγαλύτερη πρόκληση του κβαντισμού είναι η διατήρηση της απόδοσης του μοντέλου μετά τη μετατροπή των δεδομένων σε χαμηλότερη ακρίβεια. Η απώλεια πληροφορίας μπορεί να οδηγήσει σε μειωμένη ακρίβεια ή απόδοση του μοντέλου.

- Στρατηγικές Κβαντισμού: Η επιλογή της κατάλληλης στρατηγικής κβαντισμού και των παραμέτρων είναι κρίσιμη. Διαφορετικές τεχνικές κβαντισμού και ρυθμίσεις μπορεί να έχουν σημαντικά διαφορετικά αποτελέσματα στην απόδοση του μοντέλου.

Τεχνικές Κβαντισμού:

- Στατικός Κβαντισμός: Ολόκληρο το μοντέλο κβαντίζεται μετά την εκπαίδευση, χρησιμοποιώντας ακέραιες τιμές για την αντικατάσταση των κινητών υποδιαστολών.

- Δυναμικός Κβαντισμός: Η κβαντισμένη αναπαράσταση των παραμέτρων του μοντέλου ρυθμίζεται δυναμικά κατά τη διάρκεια του inference, επιτρέποντας μεγαλύτερη ευελιξία.

- Κβαντισμός Κατά την Εκπαίδευση: Ενσωματώνει τον κβαντισμό απευθείας στη διαδικασία εκπαίδευσης, επιτρέποντας στο μοντέλο να προσαρμόζεται στις αλλαγές που επιφέρει ο κβαντισμός.

Για αυτή την εργασία ο κβαντισμός έγινα σε ολόκληρο το μοντέλο (στατικός). Για τα δύο μοντέλα (YOLOv5 και YOLOv8) χρησιμοποιήθηκε συγκεκριμένα η παρακάτω συνάρτηση.

```python
def quantize(build_dir,quant_mode,batchsize):

  dset_dir = build_dir + '/dataset'
  float_model = build_dir + '/float_model'
  quant_model = build_dir + '/quant_model'


  device = torch.device('cpu')

  model = DetectMultiBackend(weights='best.pt')

  rand_in = torch.randn(1, 3, 640, 640)
  quantizer = torch_quantizer(quant_mode, model, (rand_in), bitwidth=8)
  quantized_model = quantizer.quant_model
  quantized_model = quantized_model.to(device)

  quantized_model.eval()
  # print(evaluate(model=quantized_model))
  results = quantized_model(rand_in)

  # export config
  if quant_mode == 'calib':
    quantizer.export_quant_config()
  if quant_mode == 'test':
    torch.save(quantized_model.state_dict(),"out.pt")
    quantizer.export_xmodel(deploy_check=False)

  return
```

Σχήμα 15: Quantization function, python

Η διαδικασία εξελίσσεται ως εξής:

- Το pre-trained μοντέλο φοτώνεται μέσω της κλάσης DetectMultiBackend.

- Η διαδικασία κβαντισμού ξεκινά με την δημιουργία ενός αντικειμένου torch_quantizer, καθορίζοντας τον τρόπο κβαντισμού (quant_mode), ένα δείγμα τανυστή εισόδου (rand_in) και το επιθυμητό πλάτος δυαδικών ψηφίων για τον κβαντισμό (8-bit σε αυτή την περίπτωση). Αυτή η ρύθμιση είναι κρίσιμη για την προσαρμογή του μοντέλου στους υπολογιστικούς περιορισμούς του στόχευμένου υλικού χωρίς να θυσιάζεται σημαντικά η προβλεπτική του απόδοση.

- Κβαντισμός Μοντέλου: Το μοντέλο υφίσταται κβαντισμό και στη συνέχεια μεταφέρεται στη συσκευή CPU. Αυτό το βήμα διασφαλίζει ότι το κβαντισμένο μοντέλο μπορεί να αξιολογηθεί ή να αναπτυχθεί σε συστήματα χωρίς την ανάγκη υποστήριξης GPU.

- Αξιολόγηση Μοντέλου και Εξαγωγή: Στη λειτουργία βαθμονόμησης (calib), εξάγεται το configuration που απαιτείται για το deployment του μοντέλου. Στη λειτουργία δοκιμής (test), αποθηκεύεται το state dictionary του κβαντισμένου μοντέλου και εξάγεται το μοντέλο σε μορφή XModel κατάλληλη για την ανάπτυξη σε πλατφόρμες Xilinx, συμπεριλαμβανομένου του Versal VCK 190.

Ενδεικτικά ακολουθεί συνοπτικά ένα μέρος του output του quantization για το YOLOv5 nano μοντέλο.

```
Fusing layers...
Model summary: 157 layers, 1867405 parameters, 0 gradients, 4.5 GFLOPs
[VAIQ_NOTE]: OS and CPU information:
    system --- Linux
    release --- 5.19.0-41-generic
    version --- #42~22.04.1-Ubuntu SMP PREEMPT_DYNAMIC Tue Apr 18 17:40:00
UTC 2
    machine --- x86_64
    processor --- x86_64
[VAIQ_NOTE]: Tools version information:
    GCC --- GCC 9.4.0
    python --- 3.7.12
    pytorch --- 1.12.1
    vai_q_pytorch --- 3.0.0+a44284e+torch1.12.1
[VAIQ_WARN][QUANTIZER_TORCH_CUDA_UNAVAILABLE]: CUDA (HIP) is not available,
change device to CPU
[VAIQ_NOTE]: Quant config file is empty, use default quant configuration
[VAIQ_NOTE]: Quantization test process start up...
[VAIQ_NOTE]: =>Quant Module is in 'cpu'.
[VAIQ_NOTE]: =>Parsing DetectMultiBackend...
[VAIQ_NOTE]: Start to trace and freeze model...
[VAIQ_NOTE]: The input model DetectMultiBackend is torch.nn.Module.
[VAIQ_NOTE]: Finish tracing.
[VAIQ_NOTE]: Processing ops...
| 0/144 [00:00<?, ?it/s, OpInfo: name = input_0, type = Param]
▌ | 1/144 [00:00<00:00, 12264.05it/s, OpInfo: name = DetectMultiBackend/De-
tectionModel[model]/Conv[model]/Conv[0]/Conv2d[conv]/input.3, type = _con-
volution]
▌ | 2/144 [00:00<00:00, 3449.26it/s, OpInfo: name = DetectMultiBackend/De-
tectionModel[model]/Conv[model]/Conv[0]/LeakyReLU[act]/input.5, type =
leaky_relu_]

...

████████████████████████████████████████████▌ | 142/144 [00:00<00:00,
```

```
4956.86it/s, OpInfo: name = DetectMultiBackend/DetectionModel[model]/De-
tect[model]/Detect[24]/Conv2d[m]/ModuleList[2]/9323, type = _convolution]
████████████████████████████████████████████████| 143/144 [00:00<00:00,
4951.79it/s, OpInfo: name = return_0, type = Return]
████████████████████████████████████████████████| 144/144 [00:00<00:00,
4966.86it/s, OpInfo: name = return_0, type = Return]
[VAIQ_NOTE]: =>Doing weights equalization...
[0;33m[VAIQ_WARN][QUANTIZER_TORCH_LEAKYRELU]: Force to change nega-
tive_slope of LeakyReLU from 0.1 to 0.1015625 because DPU only supports
this value. It is recommended to change all negative_slope of LeakyReLU to
0.1015625 and re-train the float model for better deployed model accuracy.
[VAIQ_NOTE]: =>Quantizable module is generated.(quantize_result/Detect-
MultiBackend.py)
[VAIQ_NOTE]: =>Get module with quantization.
[VAIQ_NOTE]: =>Converting to xmodel ...
[VAIQ_NOTE]: =>Successfully convert 'DetectMultiBackend' to xmodel.(quan-
tize_result/DetectMultiBackend_int.xmodel)
```

Σχήμα 16: Ενδεικτικό output, YOLOv5 (nano) quantization

4.3.3 Compilation των μοντέλων

Η διαδικασία του compiling, μετατρέπει το ανώτερου επιπέδου, αφηρημένο μοντέλο, που συνήθως αναπτύσσεται και εκπαιδεύεται σε περιβάλλοντα όπως η Python με τη χρήση βιβλιοθηκών όπως το TensorFlow ή το PyTorch, σε έναν πιο χαμηλού επιπέδου κώδικα που μπορεί να εκτελεστεί απευθείας στον hardware της πλακέτας (όπως του Versal VCK190 board). Το compilation λαμβάνει υπόψη τις αρχιτεκτονικές ιδιαιτερότητες και τους περιορισμούς του hardware, όπως την υποστήριξη για συγκεκριμένους τύπους δεδομένων ή τελεστών, και τη διαθέσιμη μνήμη ή την επεξεργαστική ισχύ. Στόχος είναι η μεγιστοποίηση της απόδοσης του μοντέλου στην ενσωματωμένη συσκευή, εξασφαλίζοντας ταυτόχρονα ότι η εφαρμογή μπορεί να λειτουργήσει εντός των διαθέσιμων πόρων.

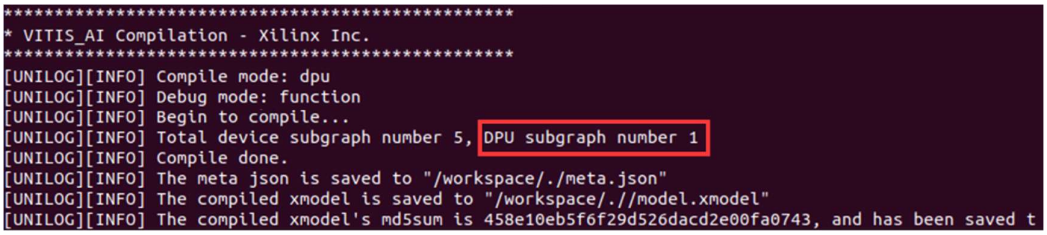
Για αυτό το βήμα χρησιμοποιήθηκε το Vitis AI compiler για μοντέλα pytorch, το vai_c_xir, για να γίνει το compile του αρχείου xmodel που δημιουργήθηκε στο προηγούμενο βήμα. Αυτό επιτυγχάνεται τρέχοντας την παρακάτω εντολή:

```
vai_c_xir -x ./DetectMultiBackend_int.xmodel -a
/opt/vitis_ai/compiler/arch/DPUCVDX8G/VCK190/arch.json -o ./ -n out-
put_model
```

Σχήμα 17: Εντολή compile του μοντέλο YOLO για το board Versal VCK190

Η παράμετρος -x καθορίζει το αρχείο xmodel που αποκτήθηκε στο προηγούμενο βήμα, η παράμετρος -a καθορίζει το αρχείο αρχιτεκτονικής του DPU και της πλακέτας ανάπτυξης, η παράμετρος -o καθορίζει τον τοποθεσία για τα αποτελέσματα εξόδου, και η παράμετρος -n καθορίζει το όνομα του εξαγόμενου

μοντέλου. Εάν δεν αναφερθεί κανένα σφάλμα σε αυτό το βήμα, θα πρέπει να ληφθεί μοντέλο με 1 υπογράφημα DPU, όπου έξοδος φαίνεται ως εξής:



Σχήμα 18: Compilation output

4.4 Διασύνδεση της εφαρμογής με το Versal

Για την ενσωμάτωση του Versal στην ευρύτερη εφαρμογή χρησιμοποιήθηκε μια τροποποιημένη έκδοση του κώδικα που αναπτύχθηκε στην παράγραφο 3. Η κύριες διαφορές σε αυτή την περίπτωση είναι αρχικά ο τρόπος εισαγωγής των δεδομένων, δηλαδή του βίντεο προς επεξεργασία, στο μοντέλο, και επίσης η εξαγωγή των αποτελεσμάτων από το Versal και η χρήση τους από το ByteTrack. Στη συνέχεια αναλύονται τα δύο αυτά σημεία.

4.4.1 Μετατροπή βίντεο σε εικόνες

Για την επεξεργασία βίντεο από το Versal, αναπτύχθηκε ένα script που μετατρέπει τα βίντεο σε εικόνες. Αυτή η διαδικασία είναι απαραίτητη καθώς το μοντέλο επεξεργάζεται εικόνες και όχι άμεσα βίντεο. Το συγκεκριμένο script χρησιμοποιεί τη βιβλιοθήκη OpenCV για να ανοίξει το βίντεο και να διαβάσει κάθε καρέ ξεχωριστά, αποθηκεύοντας το ως εικόνα JPEG. Στη συνέχεια, αυτές οι εικόνες τροφοδοτούνται στο μοντέλο σε πακέτα (batches) για την ανίχνευση αντικειμένων. Ακολουθεί ο σχετικός κώδικας:

```
import cv2
vidcap = cv2.VideoCapture('athens.mp4')
success,image = vidcap.read()
count = 0
while success:
    cv2.imwrite("/home/root/video_test/frames/frame%d.jpg" % count, image)
# save frame as JPEG file
    success,image = vidcap.read()
    print('Read a new frame: ', count, " ", success)
    count += 1
```

Σχήμα 19: Python script μετατροπής βίντεο σε εικόνες (καρέ)

4.4.2 Deployment των μοντέλων στο Versal

Στη συνέχεια, για την εκτέλεση των μοντέλων στο Versal VCK190 board αναπτύχθηκε ο παρακάτω κώδικας σε c++ :

```
int main(int argc, char* argv[]) {
  string model = argv[1];
  GLOBAL_ENABLE_NEW_IOU = 1;
  auto yolo = vitis::ai::YOLOv3::create(model);

  std::string filePath = "/home/root/video_test/frames/";

  string modelName = argv[1];
  std::filesystem::path modelPath(modelName);
  modelName = modelPath.stem().string();

  ofstream outfile;
  outfile.open ("/home/root/video_test/" + modelName.substr(0, mod-
elName.find(".")) + "_results.txt");
  auto start_time = std::chrono::high_resolution_clock::now();

  int i = 0;
  while (true) {
    std::string filenameStr = "frame" + std::to_string(i) + ".jpg";
    std::string fullPathStr = filePath + filenameStr;
    if (!fs::exists(fullPathStr)) {
        break;
    }
    Mat frame = cv::imread(fullPathStr);
    auto result_vec = yolo->run(frame);
      for (const auto &bbox : result_vec.bboxes) {
        int label = bbox.label;
        float xmin = bbox.x * frame.cols + 1;
        float ymin = bbox.y * frame.rows + 1;
        float xmax = xmin + bbox.width * frame.cols;
        float ymax = ymin + bbox.height * frame.rows;
        if (xmin < 0.) xmin = 1.;
        if (ymin < 0.) ymin = 1.;
        if (xmax > frame.cols) xmax = frame.cols;
        if (ymax > frame.rows) ymax = frame.rows;
        float confidence = bbox.score;
       outfile << label << " " << xmin << " " << ymin << " "
            << xmax << " " << ymax << " " << confidence << "\n";
      }
      outfile << "\n";
      i++;
  }

std::chrono::duration<double> duration_seconds = end_time - start_time;
```

```
cout << "Average Performance = " << i / duration_seconds.count() << " fps."
<< std::endl;

outfile.close();
return 0;
}
```

Σχήμα 20: Κώδικας εκτέλεσης του compiled xmodel στο Versal board

Εδώ τα καρέ (frames) που παράχθηκαν στο προηγούμενο βήμα, περνούν προς επεξεργασία από το μοντέλο, χρησιμοποιώντας τη βιβλιοθήκη Vitis AI της Xilinx. Τα αποτελέσματα εξάγονται σε αρχείο σε μορφή κειμένου με κατάλληλη μορφοποίηση όπως αναλύεται ακολούθως.

4.4.3 Μετατροπή αποτελεσμάτων

Για κάθε εικόνα που επεξεργάζεται, το μοντέλο επιστρέφει τις συντεταγμένες των περιοχών ενδιαφέροντος (bounding boxes) και την αντίστοιχη ετικέτα και βαθμολογία εμπιστοσύνης για κάθε ανιχνευμένο αντικείμενο. Αυτές οι πληροφορίες αποθηκεύονται στο αρχείο κειμένου με απλή (raw) μορφή ώστε να γίνει ευκολότερη η ανάγνωση (parsing) της πληροφορίας από την υπόλοιπη εφαρμογή.

```
# Class_ID, [Bounding_Box_Coordinates], Confidence_Score
2 943.07 740.487 1248.98 922.083 0.808009
2 613.936 526.687 759.392 616.611 0.783165
2 190.34 444.686 312.664 515.814 0.779049
2 733.842 290.646 817.133 337.402 0.745853
2 573.674 274.451 627.854 308.049 0.744585
```

Σχήμα 21: Ενδεικτική μορφή αποτελεσμάτων

Στη συνέχεια θα πρέπει να επεξεργαστούν τα δεδομένα στην μορφή κατάλληλη για να αξιοποιηθεί από το ByteTrack. Για αυτή την διαδικασία χρησιμοποιείται το ακόλουθο κομμάτι κώδικα, το οποίο διαβάζει το αρχείο κειμένου με τα αποτελέσματα, τα διαχωρίζει και τα μετατρέπει σε διανύσματα για την περαιτέρω χρήση τους στην εφαρμογή.

```
import torch
import numpy as np

# load Model just to extract class ids
model = torch.hub.load("ultralytics/yolov5", "yolov5n")

VERSAL_RESULTS = f"{DRIVE}/Versal_Results/yolov5_nano_pt_results.txt"

with open(VERSAL_RESULTS, 'r') as file:
content = file.read().strip()
```

```
results = content.split('\n\n')
frame_data = results[0]
lines = frame_data.split('\n')
predictions = [x.split() for x in lines]
predictions = np.array(predictions, dtype=float)
print(predictions)
class_id = predictions[:, 0].astype(int)
confidence = predictions[:, -1]
xyxy = predictions[:, 1:-1]
```

Σχήμα 22: Parsing των αποτελεσμάτων

5 Αποτελέσματα

Σε αυτό το μέρος της εργασίας θα αναλυθούν τα αποτελέσματα που παράχθηκαν μέσω του Google Colaboratory και του Vitis AI, και θα γίνει σύγκριση μεταξύ των μοντέλων YOLOv5 και YOLOv8, καθώς και μεταξύ των δύο βασικών υπολογιστικών πόρων: του Versal VCK190 board και της Tesla T4 GPU.

5.1 Συλλογή δεδομένων κατά το runtime

Η εντολή `vaitrace` παρέχεται από το Vitis AI και χρησιμοποιείται για τη συλλογή και την ανάλυση δεδομένων απόδοσης από εφαρμογές που τρέχουν στο σύστημα. Με χρήση της εντολής αυτής, με την παράμετρο `--txt_summary`, λήφθηκαν τα παρακάτω αποτελέσματα για το YOLOv5 (τα αποτελέσματα είναι παρόμοια και για το YOLOv8 οπότε τα συμπεράσματα ισχύουν και για τα 2 μοντέλα):

```
### YOLOV5 NANO ###

================================================================================================
DPU Id      | Bat | WL    | SW_RT | HW_RT | Effic | LdWB  | LdFM   | StFM   | AvgBw
------------+-----+-------+-------+-------+-------+-------+--------+--------+-----------
DPUCVDX8G_1 | 6   | 4.488 | 6.516 | 6.402 | 7.2   | 1.783 | 56.316 | 47.413 | 16480.026
================================================================================================

### YOLOV5 LARGE ###

================================================================================================
DPU Id      | Bat | WL     | SW_RT | HW_RT | Effic | LdWB  | LdFM   | StFM   | AvgBw
------------+-----+--------+-------+-------+-------+-------+--------+--------+----------
DPUCVDX8G_1 | 6   | 109.11 | 38.82 | 38.70 | 29.1  | 44.39 | 402.01 | 263.03 | 18330.65
================================================================================================

Notes:
"~0": Value is close to 0, Within range of (0, 0.001)
Bat: Batch size of the DPU instance
WL(Work Load): Computation workload (MAC indicates two operations), unit is GOP
SW_RT(Software Run time): The execution time calculate by software in milliseconds, unit
is ms
HW_RT(Hardware Run time): The execution time from hardware operation in milliseconds, unit
is ms
Effic(Efficiency): The DPU actual performance divided by peak theoretical performance,
unit is %
Perf(Performance): The DPU performance in unit of GOP per second, unit is GOP/s
LdFM(Load Size of Feature Map): External memory load size of feature map, unit is MB
LdWB(Load Size of Weight and Bias): External memory load size of bias and weight, unit is
MB
```

```
StFM(Store Size of Feature Map): External memory store size of feature map, unit is MB
AvgBw(Average bandwidth): External memory average bandwidth. unit is MB/s
```

Ένα ενδιαφέρον σημείο αποτελεί η διαφορά στην αποδοτικότητα (Effic) μεταξύ του nano και του large μοντέλου, με το YOLOv5 large να έχει σημαντικά υψηλότερη αποδοτικότητα (29.1%) σε σύγκριση με το YOLOv5 nano (7.2%), μπορεί να εξηγηθεί από πολλούς παράγοντες:

1. **Υπολογιστική Πυκνότητα**: Πιο πολύπλοκα μοντέλα όπως το YOLOv5 large μπορεί να έχουν υψηλότερη υπολογιστική πυκνότητα, δηλαδή περισσότερες υπολογιστικές εργασίες ανά μονάδα δεδομένων. Αυτό σημαίνει ότι μπορούν να εκμεταλλευτούν πιο αποτελεσματικά τη δυνατότητα υπολογισμού του DPU, αυξάνοντας την αποδοτικότητα.
2. **Συνδυαστική Εκτέλεση**: Πολύπλοκα μοντέλα μπορεί να επιτρέπουν πιο αποδοτική συνδυαστική εκτέλεση εργασιών (π.χ., παράλληλη επεξεργασία διαφορετικών σταδίων του μοντέλου), η οποία μεγιστοποιεί την αποδοτικότητα του υλικού.
3. **Κλιμάκωση Εργασιών**: Το YOLOv5 large μπορεί να κλιμακώνεται καλύτερα στις δυνατότητες του DPU, αξιοποιώντας πλήρως τους πόρους και επιτυγχάνοντας μια υψηλότερη "βαθμολογία" αποδοτικότητας σε σχέση με μικρότερα ή λιγότερο πολύπλοκα μοντέλα.
4. **Διαχείριση Δεδομένων και Μνήμης**: Τα μεγαλύτερα μοντέλα μπορεί να χρησιμοποιούν πιο αποδοτικά τη μνήμη και τα δεδομένα, μειώνοντας τις ανάγκες για συχνές προσβάσεις στη μνήμη ή τις επιβαρύνσεις από τη διαχείριση δεδομένων, που σε τελική ανάλυση μπορεί να βελτιώνει την αποδοτικότητα.

Συγκεκριμένα το τελευταίο επιβεβαιώνεται και από το μέσο εύρος ζώνης (AvgBw) όπου παρατηρούμε μια σχετικά μικρή διαφορά μεταξύ του nano και του large μοντέλου, παρά την τεράστια διαφορά στην χρήση μνήμης και στο φόρτο εργασίας. Φυσικά η διαφορά στο εύρος ζώνης μπορεί επίσης να αντανακλά περιορισμούς του υλικού ή του σχεδιασμού του συστήματος. Ακόμη και αν το YOLOv5 large απαιτεί σημαντικά περισσότερη μεταφορά δεδομένων, η δυνατότητα του συστήματος να παρέχει αυτό το εύρος ζώνης μπορεί να είναι κορεσμένη, με αποτέλεσμα μόνο ελαφρώς μεγαλύτερες τιμές AvgBw στην πράξη.

Στην έξοδο του εργαλείου `vaitrace` παρουσιάζονται επίσης διάφορες λειτουργίες που εκτελούνται στην CPU, όπως φαίνεται παρακάτω:

```
CPU Functions(Not in Graph, e.g.: pre/post-processing, vai-runtime):
============================================================================
Function                                | Device | Runs | AverageRunTime(ms)
----------------------------------------+--------+------+-------------------
cv::imread                              | CPU    | 1259 | 10.502
xir::XrtCu::run                         | CPU    | 1259 | 6.496
vitis::ai::DpuTaskImp::run              | CPU    | 1259 | 7.810
vitis::ai::ConfigurableDpuTaskImp::run  | CPU    | 1259 | 7.823
vitis::ai::YOLOv3Imp::run               | CPU    | 1259 | 13.964
============================================================================
```

Οι λειτουργίες που αναφέρονται στον πίνακα είναι οι εξής:

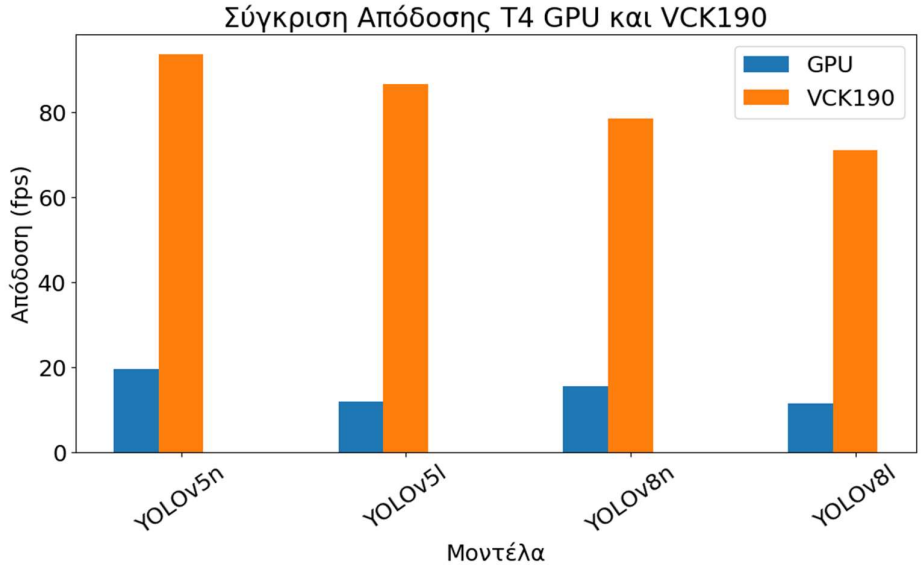
1. **cv::imread**: Αυτή η λειτουργία από τη βιβλιοθήκη OpenCV χρησιμοποιείται για τη φόρτωση εικόνων από αρχεία. Ο μέσος χρόνος εκτέλεσης ανά κλήση είναι 10.502 ms, κάτι που δείχνει τον χρόνο που απαιτείται για την ανάγνωση και την φόρτωση εικόνων στη μνήμη πριν από την επεξεργασία.
2. **xir::XrtCu::run**: Αναφέρεται σε μια εσωτερική λειτουργία της Xilinx Runtime (XRT) που εκτελείται στην CPU για τη διαχείριση και την εκκίνηση υπολογισμών στο DPU. Ο μέσος χρόνος είναι 6.496 ms ανά εκτέλεση.
3. **vitis::ai::DpuTaskImp::run**: Αυτή η λειτουργία αντιπροσωπεύει την εκτέλεση ενός DPU task. Ο μέσος χρόνος εκτέλεσης 7.810 ms υποδηλώνει τον χρόνο για την εκτέλεση της εργασίας στο DPU, περιλαμβανομένης της διαχείρισης των δεδομένων και της επικοινωνίας με το DPU.
4. **vitis::ai::ConfigurableDpuTaskImp::run**: Παρόμοια με την προηγούμενη, αυτή η λειτουργία αφορά την εκτέλεση ενός παραμετροποιήσιμου DPU task. Ο μέσος χρόνος είναι ελαφρώς υψηλότερος στα 7.823 ms.
5. **vitis::ai::YOLOv3Imp::run**: Παρά το ότι αναφέρεται στο YOLOv3 και όχι στο YOLOv5 ή YOLOv8, αυτή η λειτουργία δείχνει την εκτέλεση του αλγορίθμου ανίχνευσης αντικειμένων YOLO εντός του πλαισίου της Vitis AI. Ο μέσος χρόνος εκτέλεσης είναι 13.964 ms, ο οποίος είναι ο υψηλότερος στον πίνακα και αντανακλά την πολυπλοκότητα της επεξεργασίας του αλγορίθμου.

Οι διεργασίες που αναφέρονται στον πίνακα μπορεί να εκτελούνται με έναν συνδυασμό παράλληλης και σειριακής επεξεργασίας, ανάλογα με την αρχιτεκτονική του συστήματος . Κάθε μία από αυτές τις λειτουργίες συμβάλλει στον συνολικό χρόνο που απαιτείται για την επεξεργασία ενός frame, επηρεάζοντας τις συνολικές επιδόσεις του συστήματος σε fps. Οι χρόνοι που παρουσιάζονται εδώ δεν περιλαμβάνουν μόνο τον υπολογισμό στο DPU αλλά και τις επιπλέον διαδικασίες που πρέπει να εκτελεστούν στην CPU, παρέχοντας μια πιο

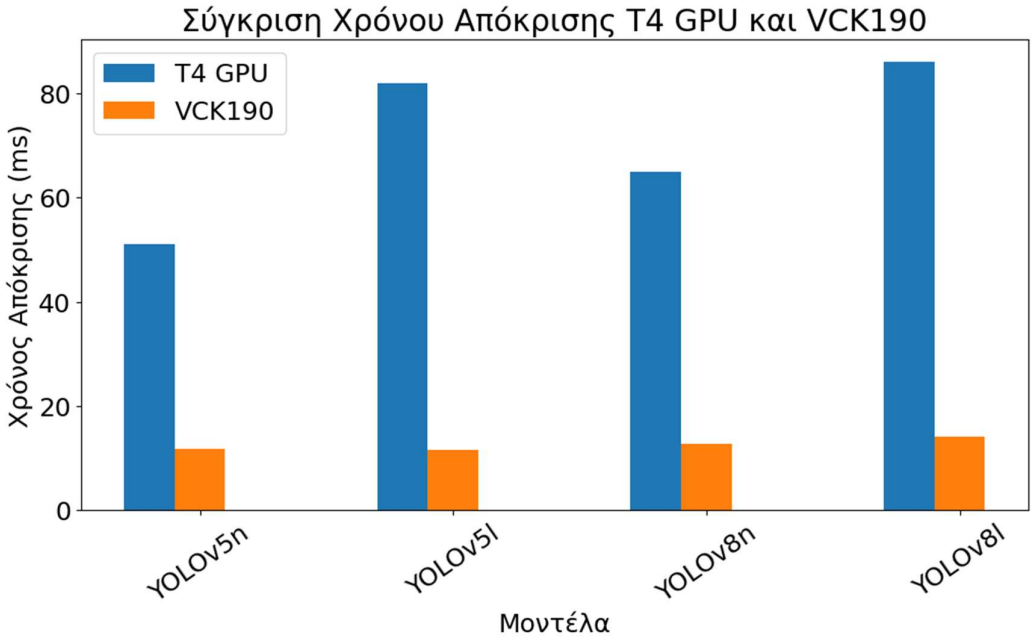
ολοκληρωμένη εικόνα του χρόνου απαιτούμενου για την επεξεργασία εικόνων με τη χρήση της πλατφόρμας Vitis AI.

5.2 Σύγκριση απόδοσης μεταξύ T4 GPU και VCK190

Αρχικά θα δοθεί βάρος στη σύγκριση των δύο διαφορετικών τεχνολογιών επεξεργασίας, T4 GPU και VCK190. Αυτή η αξιολόγηση θα μας επιτρέψει να εκτιμήσουμε τη διαφορά στην απόδοση μεταξύ των δύο τεχνολογιών και να αποφασίσουμε εάν η μετάβαση σε FPGA δικαιολογείται ως μια επικερδής αναβάθμιση, λαμβάνοντας υπόψη τόσο την αύξηση στην απόδοση όσο και τον αντίκτυπο στο κόστος.



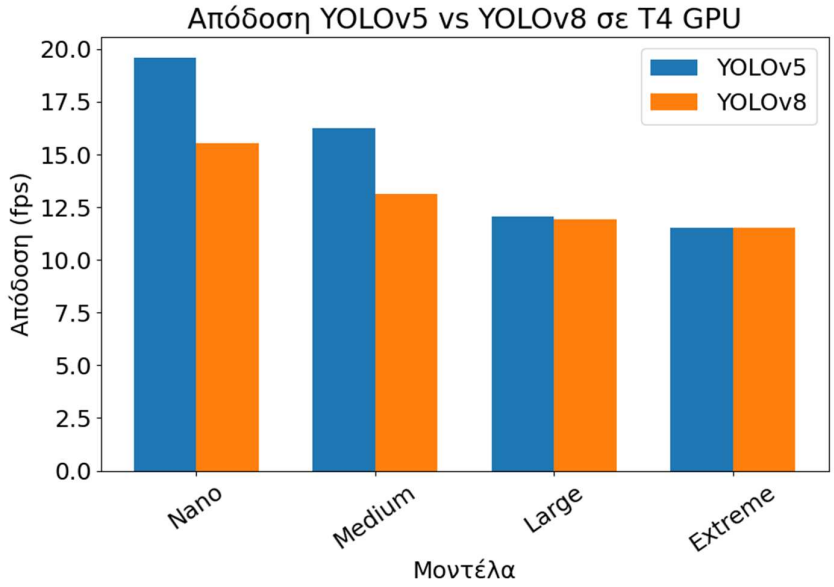
Σχήμα 23: Σύγκριση fps των T4 GPU και VCK190 Board



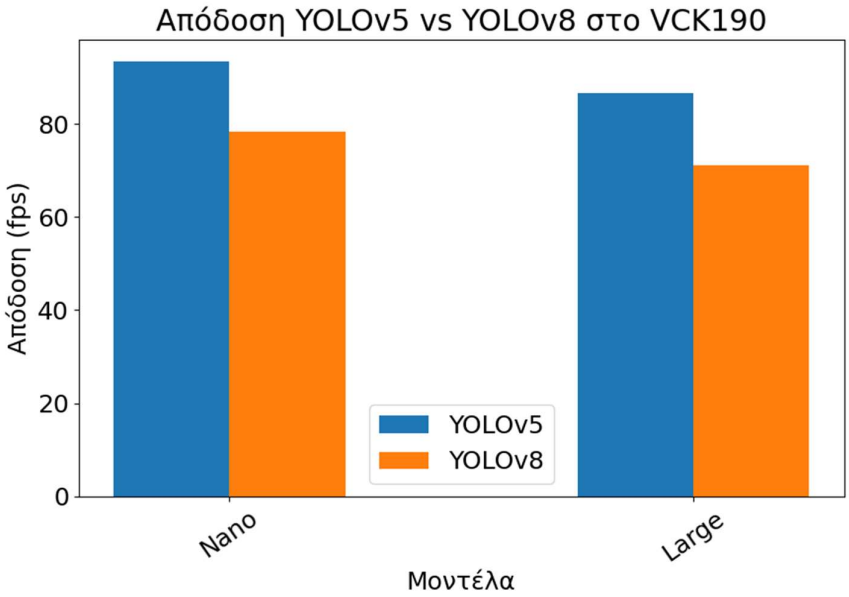
Σχήμα 24: Σύγκριση latency των T4 GPU και VCK190 Board

5.3 Σύγκριση απόδοσης μεταξύ των μοντέλων

Στα ακόλουθα γραφήματα γίνεται άμεση σύγκριση της απόδοσης (fps) μεταξύ των μοντέλων YOLOv5 και YOLOv8 σε διαφορετικά μεγέθη τους – n (nano), m (medium), l (large), x (extreme).



Σχήμα 25: Σύγκριση απόδοσης μεταξύ YOLOv5 και YOLOv8 στην T4 GPU



Σχήμα 26: Σύγκριση απόδοσης μεταξύ YOLOv5 και YOLOv8 στο VCK190

Αναλύοντας τα αποτελέσματα, παρατηρούμε τις εξής τάσεις:

- Συγκρίνοντας τα Μοντέλα YOLOv5 και YOLOv8: Στα μικρότερα μοντέλα 'n' και 'm', το YOLOv5 επιδεικνύει καλύτερη απόδοση σε σύγκριση με το YOLOv8. Ωστόσο, στα μεγαλύτερα μοντέλα 'l' και 'x', οι διαφορές στις

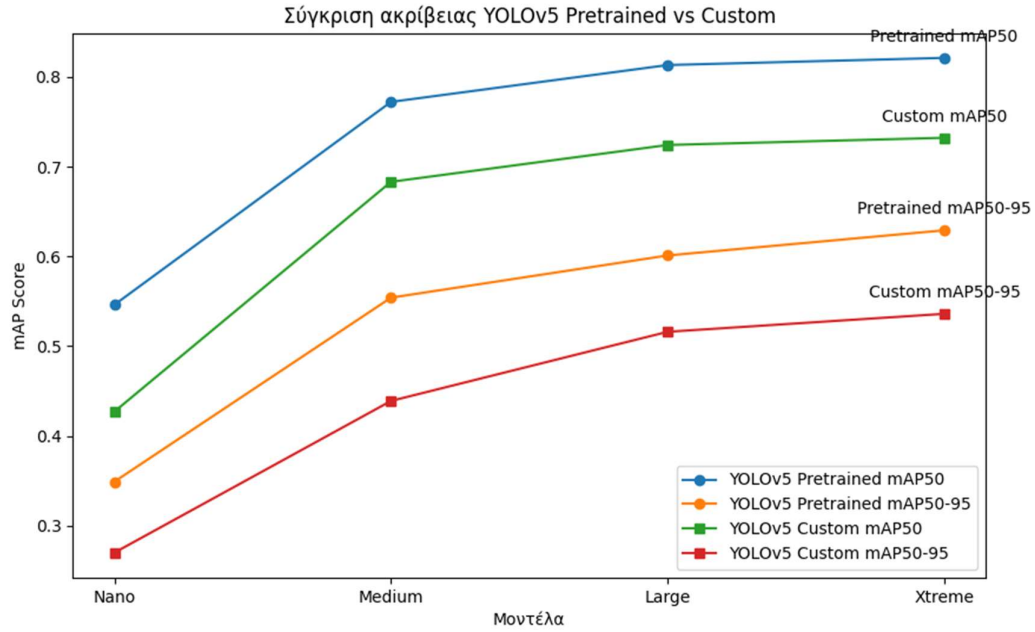
ταχύτητες είναι λιγότερο εμφανείς. Αυτό υποδηλώνει ότι το YOLOv5 μπορεί να είναι πιο αποδοτικό για ταχύτητα, ειδικά σε ελαφρύτερες εφαρμογές.

- Απόδοση ανά Έκδοση Μοντέλου: Και στα δύο μοντέλα YOLOv5 και YOLOv8, η απόδοση σε fps μειώνεται καθώς αυξάνεται το μέγεθος του μοντέλου (από n σε x). Τα μικρότερα μοντέλα (n, m) είναι γρηγορότερα, ενώ τα μεγαλύτερα (l, x) είναι αργότερα. Αυτό είναι λογικό, καθώς τα μεγαλύτερα μοντέλα έχουν περισσότερες παραμέτρους και απαιτούν περισσότερους υπολογισμούς για κάθε καρέ.

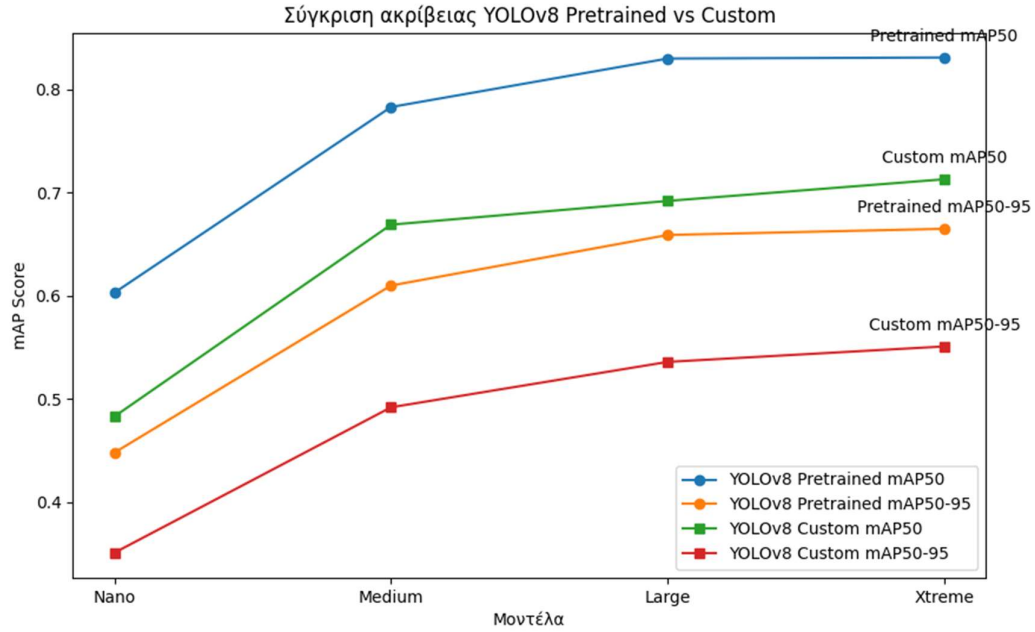
5.3.1 Σύγκριση ακρίβειας μοντέλων

Για την αξιολόγηση των αλγορίθμων ανίχνευσης, χρησιμοποιήθηκαν δύο δημοφιλείς μετρήσεις: το mAP@0.5 (ή αλλιώς mAP50) και το mAP@0.5:0.95 (ή αλλιώς mAP50-95). Το mAP@0.5 είναι η επίσημη μετρική για τον διαγωνισμό VOC (Visual Object Classes) και αναφέρεται στον μέσο όρο της Ακρίβειας (Average Precision - AP) για διάφορα κατηγορήματα, με ένα κατώφλι IoU (Intersection over Union) στο 0.5. Από την άλλη πλευρά, το mAP@0.5:0.95 είναι η επίσημη μετρική για τον διαγωνισμό COCO (Common Objects in Context) και παίρνει υπόψη το μέσο όρο των AP για κατώφλια IoU από 0.5 έως 0.95, με βήμα 0.05. Αυτό σημαίνει ότι στο mAP@0.5:0.95 λαμβάνονται υπόψη διαφορετικά επίπεδα ακρίβειας στην ταύτιση των αντικειμένων. Οι δοκιμές διεξήχθησαν στο σετ δεδομένων COCO128 με ανάλυση 640x640, προσφέροντας μια σαφή εικόνα της απόδοσης των μοντέλων σε διάφορα επίπεδα ακρίβειας και ποικιλία αντικειμένων.

Τα μοντέλα που εκπαιδεύτηκαν (ώστε να χρησιμοποιηθούν από το VCK190 Board) είχαν χαμηλότερα mAP σκορ σε σύγκριση με τα προ-εκπαιδευμένα μοντέλα. Αυτό οφείλεται αρχικά στον αριθμό εποχών (epochs) που χρησιμοποιήθηκαν ο οποίος περιορίστηκε λόγω χρόνου και υπολογιστικών πόρων που υπήρχαν διαθέσιμοι. Επίσης η έλλειψη εξειδικευμένης ρύθμισης των παραμέτρων των νευρωνικών δικτύων ενδέχεται να συνέβαλε στην μη βέλτιστη εκπαίδευση και απόδοση των μοντέλων. Ακολουθούν γραφήματα σύγκρισης των Custom και Pre-trained μοντέλων για τα Yolov5 και Yolov8:

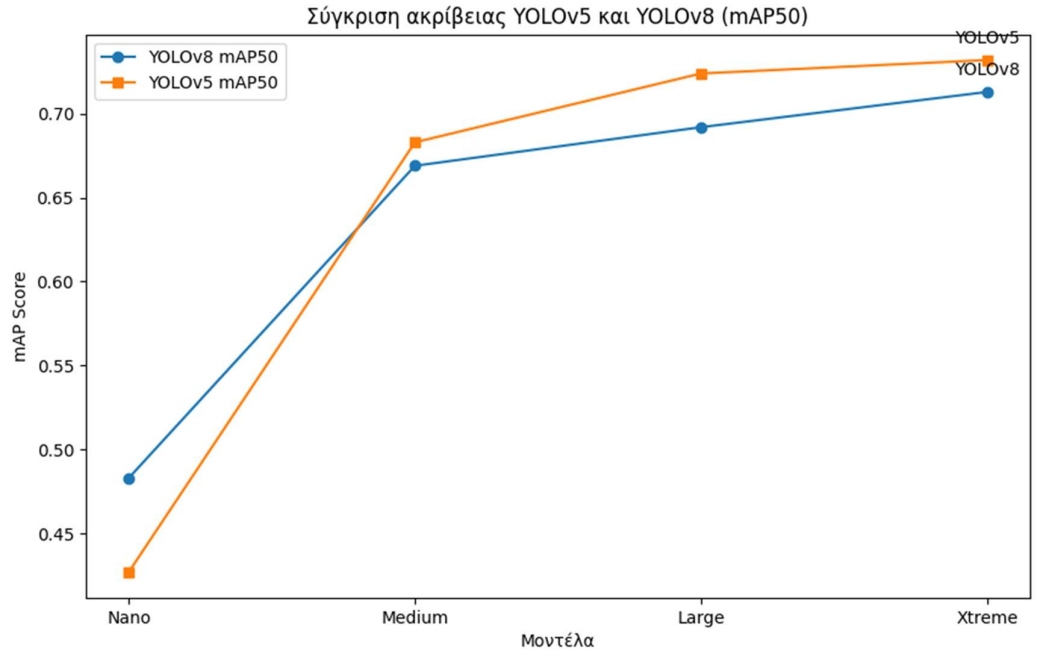


Σχήμα 27: Επίδραση του custom training στην ακρίβεια των μοντέλων (mAP50)

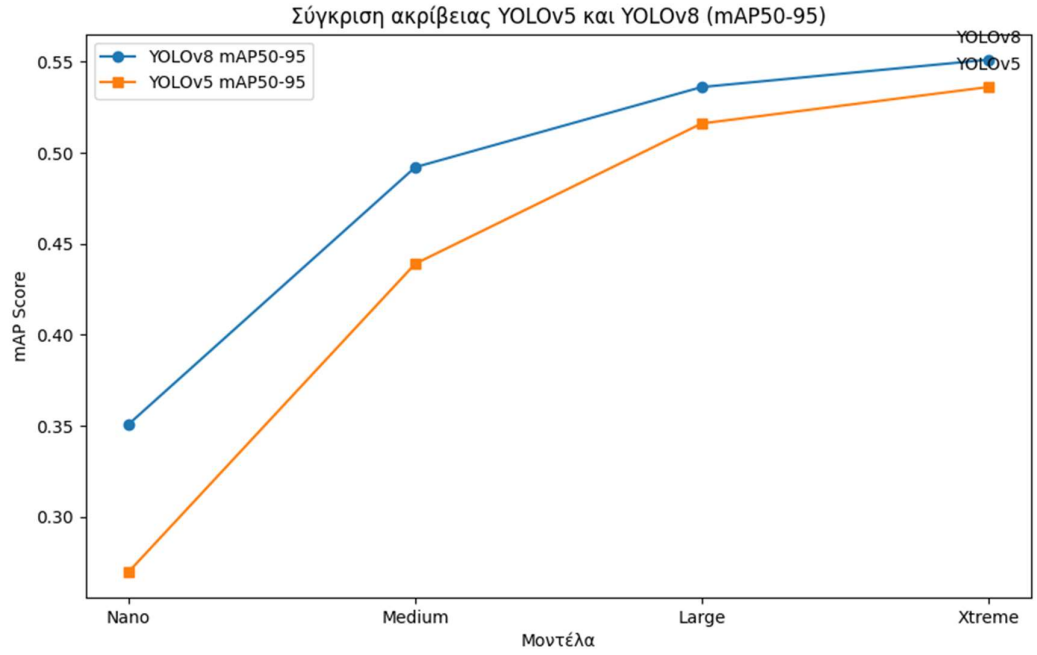


Σχήμα 28: Επίδραση του custom training στην ακρίβεια των μοντέλων (mAP50-95)

Εν τέλη, συγκρίνουμε τα μοντέλα που χρησιμοποιήθηκαν, ώστε να δούμε τις διαφορές μεταξύ των δυο εκδόσεων: Yolov5 και Yolov8



Σχήμα 29: Σύγκριση YOLOv5 και YOLOv8 σε ακρίβεια (mAP50)



Σχήμα 30: Σύγκριση YOLOv5 και YOLOv8 σε ακρίβεια (mAP50-95)

Η συνολική τάση δείχνει ότι υπάρχει ένας συμβιβασμός μεταξύ της ταχύτητας και του μεγέθους του μοντέλου. Τα μικρότερα μοντέλα είναι πιο γρήγορα αλλά μπορεί

να μην έχουν την ίδια ακρίβεια με τα μεγαλύτερα, ενώ τα μεγαλύτερα μοντέλα προσφέρουν καλύτερη ακρίβεια αλλά με χαμηλότερη ταχύτητα επεξεργασίας.

Αυτό που παρατηρείται και προκαλεί ενδιαφέρον είναι ότι το YOLOv5 υπερτερεί στη μέτρηση mAP50, ενώ το YOLOv8 δείχνει καλύτερα αποτελέσματα στη mAP50-95 μέτρηση. Αυτό μπορεί να εξηγηθεί μέσω των διαφορών στις αρχιτεκτονικές και τις εστιάσεις των δύο μοντέλων.

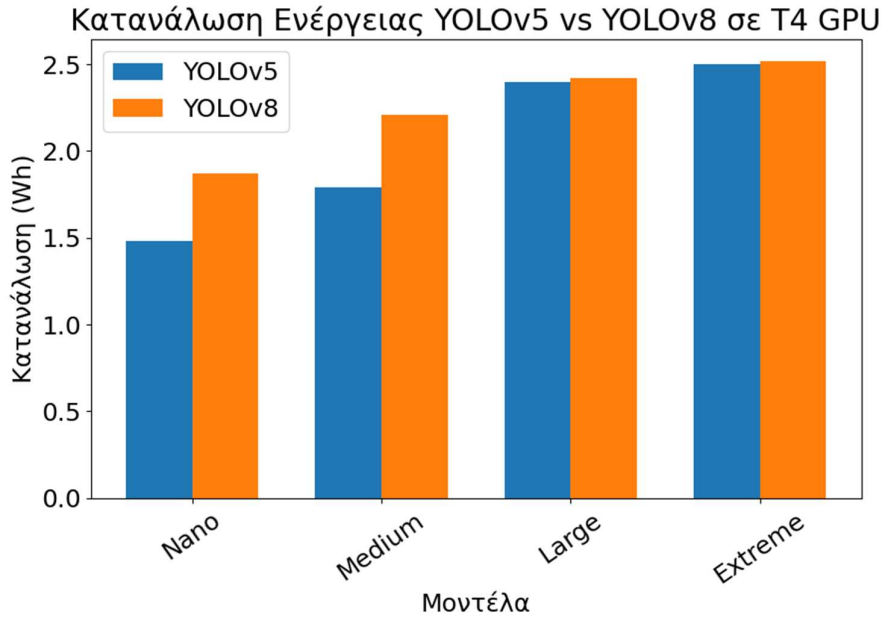
- Το mAP50 εστιάζει στην ακρίβεια ανίχνευσης με κατώφλι IoU (Intersection over Union) 0.5. Αυτό υποδηλώνει ότι το YOLOv5 είναι πιο ικανό στην ανίχνευση αντικειμένων με μέτρια ακρίβεια στη θέση και το μέγεθος τους. Η ευκρίνεια αυτή μπορεί να οφείλεται στις βελτιστοποιήσεις του YOLOv5 στην ανίχνευση γενικών χαρακτηριστικών αντικειμένων.

- Το mAP50-95 αντικατοπτρίζει την απόδοση σε πιο αυστηρά κατώφλια IoU από 0.5 έως 0.95. Αυτό δείχνει ότι το YOLOv8 είναι καλύτερο στην ακριβέστερη ανίχνευση αντικειμένων, με μεγαλύτερη ευαισθησία στο μέγεθος και την ακριβή θέση τους. Οι βελτιώσεις στο YOLOv8, όπως πιθανότατα πιο προηγμένοι αλγόριθμοι και αρχιτεκτονικές βαθιών νευρωνικών δικτύων, μπορεί να συμβάλλουν σε αυτή την αυξημένη ακρίβεια.

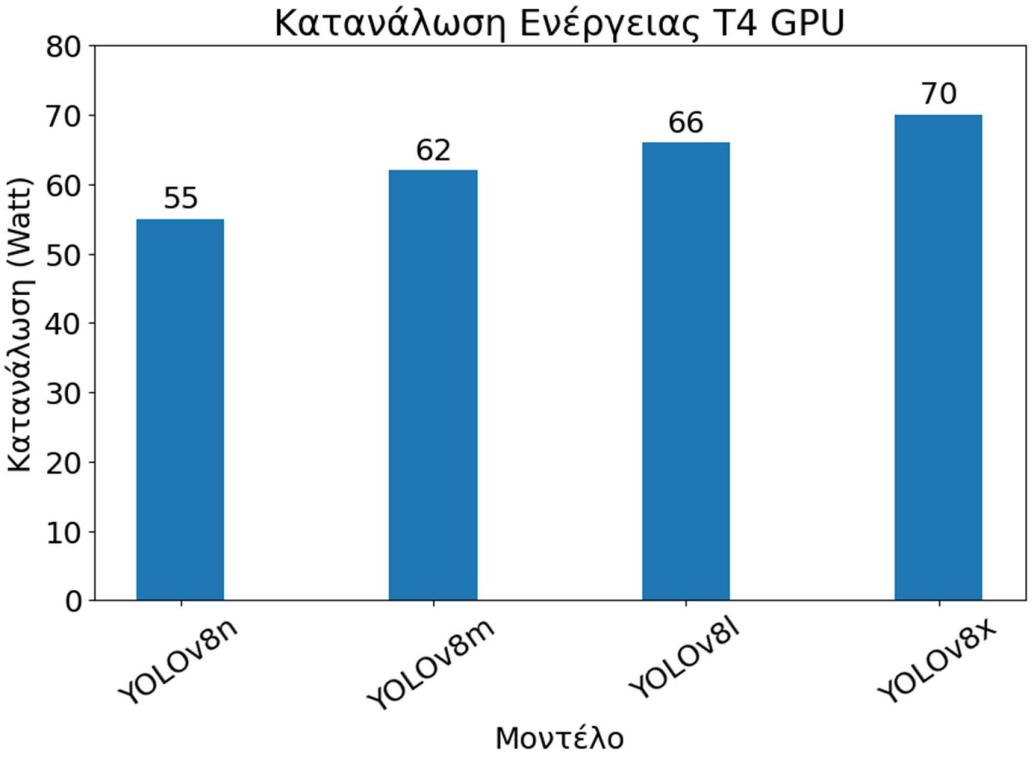
Από αυτά μπορούμε να βγάλουμε το συμπέρασμα ότι, για εφαρμογές που απαιτούν γρήγορη και γενική ανίχνευση, το YOLOv5 μπορεί να είναι ιδανικό, ενώ για πιο απαιτητικές συνθήκες με ανάγκη υψηλότερης ακρίβειας, το YOLOv8 μπορεί να προσφέρει καλύτερα αποτελέσματα.

5.4 Μέτρηση κατανάλωσης ενέργειας

Για τον υπολογισμό της ενεργειακής κατανάλωσης της T4 GPU στο Google Colaboratory, χρησιμοποιήθηκε η ιστοσελίδα Green Algorithms [20], ένα εργαλείο που προσφέρει μια εκτίμηση της περιβαλλοντικής επίπτωσης των υπολογιστικών διεργασιών. Με αυτή τη μεθοδολογία, κατέστη δυνατό να εκτιμηθεί η συνολική κατανάλωση ενέργειας για την επεξεργασία του βίντεο εισόδου, παρέχοντας έτσι μια εικόνα για την ενεργειακή αποδοτικότητα και το περιβαλλοντικό αποτύπωμα της εφαρμογής.

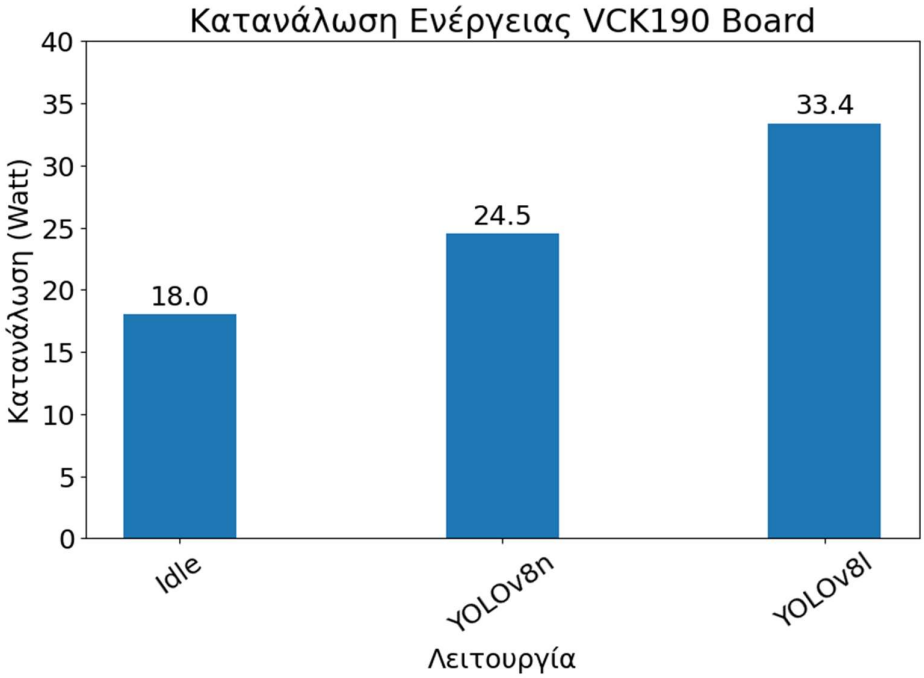


Σχήμα 31: Συνολική κατανάλωση ενέργειας μοντέλων με την T4 GPU για το βίντεο εισόδου (83 δευτερολέπτων)



Σχήμα 32: Κατανάλωση ενέργειας μοντέλων με την T4 GPU

Η μέτρηση της κατανάλωσης του VCK190 Board ήταν απλή και αποτελεσματική, χάρη στην άμεση πρόσβαση στα διαγνωστικά εργαλεία του συστήματος, που παρέχονται από το ίδιο το hardware.



Σχήμα 33: Κατανάλωση ενέργειας μοντέλων με το VCK190 Board

Επίλογος

Η παρούσα εργασία αναδεικνύει τη σπουδαιότητα και την αξία της συνεχούς εξέλιξης στον τομέα της τεχνητής νοημοσύνης και της μηχανικής μάθησης. Ενώ η ψηφιακή επανάσταση μας προσφέρει αμέτρητες δυνατότητες, είναι ο συνεχής αναστρέψιμος αυτός κύκλος της καινοτομίας που οδηγεί στην επίτευξη νέων υψηλών επιδόσεων και λύσεων.

Η παρούσα εργασία δείχνει ότι η συνδυασμένη χρήση προηγμένων τεχνολογικών εργαλείων και της εμβάθυνσης στη θεωρητική κατανόηση είναι καθοριστική για την ανάπτυξη αποδοτικών λύσεων. Η ενσωμάτωση προηγμένων τεχνικών όπως οι FPGAs αποτελεί ένα εξαιρετικό παράδειγμα πώς η τεχνολογία μπορεί να ανοίξει νέους δρόμους για την επίλυση προβλημάτων σε πραγματικό χρόνο με αποδοτικότητα και οικονομία.

Τέλος, η εργασία καλεί όχι μόνο σε περαιτέρω έρευνα και ανάπτυξη στον τομέα αυτόν, αλλά και στη συνεχή αναζήτηση νέων τρόπων για την ενσωμάτωση καινοτόμων λύσεων προκειμένου να ανταποκριθούμε στις αναπόφευκτες προκλήσεις του μέλλοντος. Με τη συνεχή ανάπτυξη και την ανταλλαγή ιδεών, μπορούμε να προωθήσουμε την τεχνολογία προς μια πιο αποδοτική, αξιόπιστη και βιώσιμη κατεύθυνση, επιτρέποντας μας να αντιμετωπίσουμε τις προκλήσεις της εποχής μας με αυξημένη επιτυχία και αποτελεσματικότητα.

Βιβλιογραφία

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

[2] “Learning Paradigms in Neural Networks” [Online]. Available: https://medium.com/swlh/learning-paradigms-in-neural-networks-30854975aa8d.

[3] LeCun Y., Bengio Y. & Hinton G. (2015). Deep learning. Nature, 521(7553), 436-444..

[4] “CS231n Convolutional Neural Networks for Visual Recognition” [Online]. Available: https://cs231n.github.io/convolutional-networks/#conv.

[5] [Online]. Available: https://www.python.org/doc/essays/blurb/.

[6] “About Pytorch” [Online]. Available: https://pytorch.org/.

[7] “Pytorch” [Online]. Available: https://en.wikipedia.org/wiki/PyTorch.

[8] “Project Jupyter Documentation” [Online]. Available: https://docs.jupyter.org/en/latest/.

[9] “Google Colaboratory” [Online]. Available: https://colab.google/.

[10] “Get Started with Vitis AI” [Online]. Available: https://www.xilinx.com/developer/products/vitis-ai.html#articles.

[11] “An Introduction to BYTETrack: Multi-Object Tracking by Associating Every Detection Box” [Online]. Available: https://www.datature.io/blog/introduction-to-bytetrack-multi-object-tracking-by-associating-every-detection-box.

[12] “Roboflow Supervision” [Online]. Available: https://supervision.roboflow.com/.

[13] “The History of YOLO Object Detection Models from YOLOv1 to YOLOv8” [Online]. Available: https://deci.ai/blog/history-yolo-object-detection-models-from-yolov1-yolov8/.

[14] “A Brief History of YOLO Object Detection Models From YOLOv1 to YOLOv5” [Online]. Available: https://machinelearningknowledge.ai/a-brief-history-of-yolo-object-detection-models/.

[15] “Skyline Webcams” [Online]. Available: https://www.skylinewebcams.com/.

[16] “Introduction to FPGA | Structure, Components, Applications” [Online]. Available: https://www.electronicshub.org/introduction-to-fpga/.

[17] “Introduction to FPGA and It's Programming Tools” [Online]. Available: https://circuitdigest.com/tutorial/what-is-fpga-introduction-and-programming-tools.

[18] “Versal AI Core Series VCK190 Evaluation Kit” [Online]. Available: https://www.xilinx.com/products/boards-and-kits/vck190.html.

[19] “DPUCVDX8G for Versal ACAPs Product Guide” [Online]. Available: https://docs.xilinx.com/r/en-US/pg389-dpucvdx8g/Feature-Support.

[20] “Green Algorithms” [Online]. Available: https://www.green-algorithms.org/.