

Δημοκρίτειο Πανεπιστήμιο Θράκης

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Εργαστήριο Όρασης Υπολογιστών

Όραση Υπολογιστών

Ακαδημαϊκό Έτος 2024–2025

Εργασία 4

Object Detection για Traffic Sign Detection

Faster R-CNN, YOLOv3 και Grounding DINO

Φοιτητής:

Γραμμένος-Γεώργιος
Πολυμερίδης

Αριθμός Μητρώου:

58105

Email:

grampoly@ee.duth.gr

Διδάσκων:

Ιωάννης Πρατικάκης

Εργαστήριο:

Εργαστήριο Όρασης
Υπολογιστών

Website:

<https://vc.ee.duth.gr>

Ξάνθη, Ιανουάριος 2025

Περίληψη

Η παρούσα εργασία πραγματεύεται το πρόβλημα της **ανίχνευσης αντικειμένων** (Object Detection) με εφαρμογή στην αναγνώριση **σημάτων οδικής κυκλοφορίας** (Traffic Sign Detection). Η αυτόματη ανίχνευση και αναγνώριση σημάτων αποτελεί κρίσιμο συστατικό των συστημάτων υποβοήθησης οδήγησης (ADAS) και των αυτόνομων οχημάτων, με άμεσες εφαρμογές στην οδική ασφάλεια.

Στο πλαίσιο της εργασίας υλοποιούνται και αξιολογούνται τρεις διαφορετικές προσεγγίσεις:

1. **Faster R-CNN** (Two-Stage Detector): Υλοποίηση με torchvision pretrained μοντέλο βασισμένο στο seminal paper των Ren et al. (2015), με Region Proposal Network (RPN) για παραγωγή υποψήφιων περιοχών και Detection Head για ταξινόμηση και regression.
2. **YOLOv3** (One-Stage Detector): Χρήση της βιβλιοθήκης ultralytics για εκπαίδευση one-stage detector με Darknet-53 backbone που πραγματοποιεί detection σε ένα πέρασμα του δικτύου.
3. **Grounding DINO** (Zero-Shot Detector): Αξιολόγηση vision-language model που επιτρέπει zero-shot object detection με χρήση text prompts, χωρίς εκπαίδευση στο συγκεκριμένο dataset.

Για την αξιολόγηση χρησιμοποιείται το **LISA Traffic Sign Dataset** που περιλαμβάνει 900 εικόνες με 9 κατηγορίες σημάτων κυκλοφορίας. Η μετρική αξιολόγησης είναι το **mean Average Precision (mAP)** με IoU thresholds 0.5 και 0.5:0.95.

Κύρια Αποτελέσματα:

- Faster R-CNN: mAP@0.5 = **0.8729**, mAP@0.5:0.95 = **0.7113**
- YOLOv3: mAP@0.5 = **0.9193**, mAP@0.5:0.95 = **0.7367**
- Grounding DINO: mAP@0.5 = **0.2667** (zero-shot)

Λέξεις-Κλειδιά: Object Detection, Faster R-CNN, YOLO, Grounding DINO, Traffic Sign Detection, Deep Learning, Computer Vision, Zero-Shot Learning, Region Proposal Network, Anchor Boxes, mAP

Abstract

This work addresses the problem of **Object Detection** with application to **Traffic Sign Detection**. Automatic detection and recognition of traffic signs constitutes a critical component of Advanced Driver Assistance Systems (ADAS) and autonomous vehicles, with direct applications in road safety.

Within the scope of this assignment, three different approaches are implemented and evaluated:

1. **Faster R-CNN** (Two-Stage Detector): Torchvision pretrained architecture implementation based on the seminal paper by Ren et al. (2015), featuring a Region Proposal Network (RPN) for generating candidate regions and a Detection Head for classification and bounding box regression.
2. **YOLOv3** (One-Stage Detector): Utilization of the ultralytics library to train a one-stage detector with Darknet-53 backbone that performs detection in a single network pass.
3. **Grounding DINO** (Zero-Shot Detector): Evaluation of a vision-language model enabling zero-shot object detection through text prompts, without training on the specific dataset.

For evaluation, the **LISA Traffic Sign Dataset** is employed, comprising 900 images with 9 traffic sign categories. The evaluation metric is **mean Average Precision (mAP)** with IoU thresholds of 0.5 and 0.5:0.95.

Main Results:

- Faster R-CNN: mAP@0.5 = **0.8729**, mAP@0.5:0.95 = **0.7113**
- YOLOv3: mAP@0.5 = **0.9193**, mAP@0.5:0.95 = **0.7367**
- Grounding DINO: mAP@0.5 = **0.2667** (zero-shot)

Keywords: Object Detection, Faster R-CNN, YOLO, Grounding DINO, Traffic Sign Detection, Deep Learning, Computer Vision, Zero-Shot Learning, Region Proposal Network, Anchor Boxes, mAP

Περιεχόμενα

Περίληψη	i
Abstract	ii
Κατάλογος Σχημάτων	x
Κατάλογος Πινάκων	xii
Κατάλογος Αλγορίθμων	xiii
1 Εισαγωγή	1
1.1 Γενικά περί Object Detection	1
1.2 Ιστορική Εξέλιξη των Object Detectors	1
1.2.1 Παραδοσιακές Μέθοδοι (1990–2012)	2
1.2.2 CNN-based Two-Stage Detectors (2014–2017)	2
1.2.3 One-Stage Detectors (2015–σήμερα)	2
1.2.4 Vision-Language Models (2021–σήμερα)	3
1.3 Two-Stage vs One-Stage Detectors	3
1.3.1 Two-Stage Detectors	3
1.3.2 One-Stage Detectors	3
1.4 Εφαρμογή: Traffic Sign Detection	4
1.4.1 Σημασία στα Αυτόνομα Οχήματα	4
1.4.2 Προκλήσεις του Traffic Sign Detection	4
1.5 Στόχοι της Εργασίας	5
1.5.1 Μέρος A: Υλοποίηση και Σύγκριση Αρχιτεκτονικών	5
1.5.2 Μέρος B: Zero-Shot Detection με Grounding DINO	5
1.6 Δομή της Αναφοράς	5
2 Θεωρητικό Υπόβαθρο	7
2.1 Convolutional Neural Networks (CNNs)	7
2.1.1 Βασικές Έννοιες	7
2.1.2 Convolutional Layer	7
2.1.3 Pooling Layers	8
2.1.4 Activation Functions	8
2.1.5 Batch Normalization	9
2.1.6 Residual Connections	9
2.2 Backbone Networks για Object Detection	9
2.2.1 VGG Networks	9
2.2.2 ResNet	10

2.2.3	Darknet (YOLO Backbone)	10
2.3	Faster R-CNN: Αναλυτική Περιγραφή	10
2.3.1	Architecture Overview	10
2.3.2	Region Proposal Network (RPN)	10
2.3.3	RoI Pooling	12
2.3.4	Detection Head (Stage 2)	13
2.3.5	Non-Maximum Suppression (NMS)	13
2.4	Intersection over Union (IoU)	14
2.5	YOLO: You Only Look Once	14
2.5.1	Βασική Ιδέα	14
2.5.2	YOLOv3 Architecture	14
2.5.3	YOLOv3 μέσω Ultralytics	15
2.5.4	YOLO Loss Function	15
2.6	Grounding DINO: Vision-Language Detection	16
2.6.1	DETR Background	16
2.6.2	Grounding DINO Architecture	16
2.6.3	Zero-Shot Detection	17
2.7	Μετρικές Αξιολόγησης	18
2.7.1	Precision και Recall	18
2.7.2	Average Precision (AP)	18
2.7.3	mean Average Precision (mAP)	18
2.7.4	COCO mAP Metrics	18
2.8	Data Augmentation	19
2.8.1	Geometric Transformations	19
2.8.2	Photometric Transformations	19
2.8.3	Mosaic Augmentation (YOLO)	19
3	Ανάλυση Dataset	20
3.1	LISA Traffic Sign Dataset	20
3.1.1	Περιγραφή Dataset	20
3.1.2	Κατηγορίες Σημάτων	20
3.2	Στατιστική Ανάλυση	21
3.2.1	Κατανομή Κατηγοριών	21
3.2.2	Ανάλυση Μεγέθους Bounding Boxes	21
3.2.3	Κατανομή Θέσης στην Εικόνα	21
3.3	Annotation Format	21
3.3.1	CSV Structure	21
3.3.2	Μετατροπή Συντεταγμένων	21
3.4	Data Loading Pipeline	22
3.4.1	Loading Function	22
3.4.2	Data Splits	22
3.5	Data Augmentation Strategy	22
3.5.1	Augmentations για Training	23
3.5.2	Υλοποίηση με Albumentations	23
3.5.3	Αιτιολόγηση Επιλογών	23
3.5.4	Validation/Test Transforms	24
3.6	DataLoader Configuration	24
3.6.1	Faster R-CNN DataLoader	24

3.6.2 YOLO DataLoader	24
3.7 Οπτικοποίηση Δειγμάτων	25
3.8 Προκλήσεις του Dataset	25
3.8.1 Class Imbalance	25
3.8.2 Μικρά Αντικείμενα	25
3.8.3 Παρόμοιες Κατηγορίες	25
3.9 Σύνοψη	25
4 Υλοποίηση Faster R-CNN με Torchvision	26
4.1 Αρχιτεκτονική Overview	26
4.1.1 Συνολική Δομή	26
4.2 Backbone Network	27
4.2.1 Επιλογή Pretrained Model	27
4.2.2 Model Building	27
4.2.3 ResNet-50 + FPN Backbone	27
4.3 Region Proposal Network (RPN)	28
4.3.1 Περιγραφή RPN (torchvision)	28
4.3.2 Anchor Generation	28
4.4 ROI Align	28
4.5 Detection Head (Box Predictor)	29
4.5.1 FastRCNNPredictor	29
4.6 Loss Functions	29
4.7 Training Configuration	30
4.7.1 Hyperparameters	30
4.7.2 Optimizer Setup	30
4.8 Training Loop	30
4.9 Inference Pipeline	31
4.10 Αποτέλεσματα Εκπαίδευσης	32
4.10.1 Training Loss Curve	32
4.10.2 mAP Results	32
4.10.3 Per-Class AP	33
4.11 Qualitative Results	33
4.12 Συζήτηση	33
4.12.1 Πλεονεκτήματα Torchvision Approach	33
4.12.2 Παρατηρήσεις	33
4.12.3 Δυνατές Βελτιώσεις	34
5 Υλοποίηση YOLOv3 (One-Stage Detector)	35
5.1 Εισαγωγή στο YOLO	35
5.1.1 Φιλοσοφία One-Stage Detection	35
5.1.2 Εξέλιξη της Οικογένειας YOLO	36
5.2 YOLOv3 Architecture	36
5.2.1 Overview	36
5.2.2 Backbone: Darknet-53	36
5.2.3 Multi-Scale Detection (FPN-like)	37
5.2.4 Detection Head: Anchor-Based	37
5.2.5 YOLOv3u (Ultralytics)	37
5.3 Dataset Preparation για YOLO	37

5.3.1	YOLO Format	37
5.3.2	Label Format	38
5.3.3	Μετατροπή από LISA Format	38
5.3.4	Configuration File (data.yaml)	39
5.4	Training Configuration	39
5.4.1	Hyperparameters	39
5.4.2	Data Augmentation (Built-in)	40
5.5	Training Process	40
5.5.1	Training Script	40
5.5.2	Training Output	41
5.6	Loss Functions	41
5.6.1	YOLOv3 Loss Components	41
5.7	Αποτελέσματα	42
5.7.1	Training Metrics	42
5.7.2	Evaluation Results	42
5.7.3	Per-Class Performance	43
5.7.4	Confusion Matrix	43
5.8	Qualitative Results	43
5.9	Inference Speed	43
5.10	Συζήτηση	43
5.10.1	Πλεονεκτήματα YOLOv3	43
5.10.2	Σύγκριση με Custom Faster R-CNN	44
5.10.3	Περιορισμοί	44
6	Grounding DINO: Zero-Shot Object Detection	45
6.1	Εισαγωγή στο Zero-Shot Learning	45
6.1.1	Τι είναι το Zero-Shot Learning	45
6.1.2	Πώς επιτυγχάνεται	45
6.2	Vision-Language Models	46
6.2.1	Εξέλιξη VLMs	46
6.2.2	CLIP: Contrastive Language-Image Pretraining	46
6.3	Grounding DINO Architecture	46
6.3.1	Overview	46
6.3.2	Image Backbone: Swin Transformer	47
6.3.3	Text Encoder: BERT	47
6.3.4	Feature Enhancer	47
6.3.5	Language-Guided Query Selection	47
6.3.6	Cross-Modality Decoder	48
6.4	Υλοποίηση για Traffic Signs	48
6.4.1	Setup	48
6.4.2	Text Prompts	48
6.4.3	Inference Function	49
6.4.4	Batch Inference on Test Set	49
6.5	Threshold Tuning	50
6.5.1	Box Threshold	50
6.5.2	Text Threshold	51
6.6	Αποτελέσματα	51
6.6.1	Quantitative Results	51

6.6.2	Per-Class Performance	51
6.6.3	Qualitative Results	52
6.7	Prompt Engineering Analysis	52
6.7.1	Επίδραση του Prompt	52
6.7.2	Observations	52
6.8	Ανάλυση Αποτελεσμάτων	53
6.8.1	Πλεονεκτήματα Zero-Shot	53
6.8.2	Περιορισμοί	53
6.8.3	Αιτίες Χαμηλότερης Απόδοσης	53
6.9	Βελτιώσεις και Extensions	53
6.9.1	Prompt Tuning	53
6.9.2	Ensemble με Trained Models	54
6.9.3	Fine-Tuning Option	54
6.10	Συμπεράσματα Κεφαλαίου	54
7	Συγκριτική Ανάλυση Μεθόδων	55
7.1	Σύγκριση Αρχιτεκτονικών	55
7.1.1	Ταξινόμηση Μεθόδων	55
7.1.2	Αρχιτεκτονικά Χαρακτηριστικά	55
7.2	Quantitative Comparison	56
7.2.1	mAP Results	56
7.2.2	Precision-Recall Analysis	56
7.3	Per-Class Performance	57
7.3.1	Class-wise mAP	57
7.3.2	Analysis by Difficulty	57
7.4	Speed and Efficiency	57
7.4.1	Inference Time	57
7.4.2	Model Complexity	58
7.5	Training Requirements	58
7.5.1	Training Time	58
7.5.2	Data Requirements	58
7.6	Qualitative Comparison	58
7.6.1	Visual Predictions	58
7.6.2	Error Analysis	59
7.7	Strengths and Weaknesses	59
7.7.1	Faster R-CNN	59
7.7.2	YOLOv3	59
7.7.3	Grounding DINO	60
7.8	Suitability Analysis	60
7.8.1	Use Case Recommendations	60
7.8.2	Trade-offs Summary	60
7.9	Statistical Significance	61
7.9.1	Confidence Intervals	61
7.10	Συμπεράσματα Κεφαλαίου	61

8 Συμπεράσματα και Μελλοντική Εργασία	63
8.1 Σύνοψη Εργασίας	63
8.1.1 Αντικείμενο	63
8.1.2 Dataset	63
8.1.3 Μεθοδολογία	63
8.2 Κύρια Ευρήματα	64
8.2.1 Απόδοση Μοντέλων	64
8.2.2 Βασικές Παρατηρήσεις	64
8.3 Συμπεράσματα	65
8.3.1 Τεχνικά Συμπεράσματα	65
8.3.2 Εκπαιδευτικά Συμπεράσματα	65
8.3.3 Πρακτικά Συμπεράσματα	65
8.4 Μελλοντική Εργασία	66
8.4.1 Βραχυπρόθεσμες Επεκτάσεις	66
8.4.2 Μακροπρόθεσμες Κατευθύνσεις	66
8.4.3 Ερευνητικές Ερωτήσεις	66
8.5 Τελικές Σκέψεις	67
Βιβλιογραφία	68
A' Πλήρης Κώδικας	71
A'.1 Faster R-CNN με Torchvision	71
A'.1.1 Model Building	71
A'.1.2 Data Augmentation	72
A'.1.3 Training Configuration	72
A'.1.4 Training Loop	73
A'.1.5 Evaluation with mAP	74
A'.2 YOLOv3 Training Script	75
A'.3 Grounding DINO Inference	77
A'.3.1 Model Loading	77
A'.3.2 Text Prompts	77
A'.3.3 Detection Function	77
A'.3.4 Evaluation with All Prompts	78
B' Υπερπαράμετροι και Ρυθμίσεις	80
B'.1 Faster R-CNN	80
B'.1.1 Backbone Configuration	80
B'.1.2 RPN Configuration	81
B'.1.3 ROI Pooling Configuration	81
B'.1.4 Detection Head Configuration	82
B'.1.5 Training Configuration	82
B'.2 YOLOv3	83
B'.2.1 Model Configuration	83
B'.2.2 Anchor Configuration	83
B'.2.3 Training Configuration	83
B'.2.4 Augmentation Configuration	84
B'.3 Grounding DINO	84
B'.3.1 Model Configuration	84
B'.3.2 Inference Configuration	85

B'.3.3 Text Prompts	85
B'.4 Hardware και Environment	85
Γ' Πρόσθετα Αποτελέσματα και Οπτικοποιήσεις	86
Γ'.1 Grounding DINO Analysis	86
Γ'.1.1 Prompt Sensitivity Analysis	86
Γ'.2 Speed Benchmarks	86
Γ'.2.1 Inference Time Distribution	86
Γ'.2.2 Throughput Analysis	87
Γ'.3 Dataset Statistics	87
Γ'.3.1 Class Distribution Details	87
Γ'.3.2 Bounding Box Size Statistics	87

Κατάλογος σχημάτων

2.1	Αρχιτεκτονική Grounding DINO: Ο Swin Transformer επεξεργάζεται την εικόνα ενώ ο BERT encoder επεξεργάζεται το text prompt. Τα features ενοποιούνται μέσω cross-modality fusion και το Language-Guided Query Selection επιλέγει τα queries με βάση το κείμενο.	17
4.1	Faster R-CNN με ResNet-50 FPN Backbone (torchvision).	26
4.2	Training loss του Faster R-CNN κατά τη διάρκεια της εκπαίδευσης.	32
4.3	Οπτικοποίηση predictions του Faster R-CNN σε δείγματα του test set.	33
5.1	YOLOv3 predictions σε δείγματα του test set.	43
6.1	Grounding DINO predictions στο test set. Κάθε box περιέχει το matched phrase και το confidence score.	52
7.1	Σύγκριση mAP scores μεταξύ των τριών μοντέλων.	56
7.2	Σύγκριση απόδοσης μεταξύ των τριών μοντέλων (Faster R-CNN, YOLOv3, Grounding DINO).	56
7.2	58
7.2	58
7.3	Qualitative comparison: Predictions από τα δύο supervised μοντέλα (Faster R-CNN και YOLOv3) σε εικόνες test set.	58

Κατάλογος πινάκων

1.1	Σύγκριση Two-Stage και One-Stage Detectors	4
2.1	Παράμετροι Convolutional Layer	8
2.2	ResNet-50 Architecture	10
2.3	Darknet-53 Architecture	15
2.4	COCO Evaluation Metrics	19
3.1	Κατηγορίες Σημάτων και Περιγραφή	20
3.2	Data Augmentation Pipeline	23
4.1	Αρχιτεκτονική Faster R-CNN (torchvision pretrained)	26
4.2	ResNet-50 FPN Architecture (torchvision)	27
4.3	RPN Configuration (torchvision default)	28
4.4	RoI Align Configuration	28
4.5	Faster R-CNN Loss Components	29
4.6	Training Hyperparameters (Torchvision Faster R-CNN)	30
4.7	Faster R-CNN (torchvision) - Αποτελέσματα mAP στο Test Set	32
4.8	Per-Class Average Precision (AP@0.5)	33
5.1	Εξέλιξη YOLO Versions	36
5.2	YOLOv3 Training Hyperparameters	39
5.3	YOLOv3 Built-in Augmentations	40
5.4	YOLOv3 - Αποτελέσματα mAP στο Test Set	42
5.5	YOLOv3 Per-Class Average Precision (AP@0.5)	43
5.6	Inference Speed Comparison (GPU)	43
6.1	Εξέλιξη Vision-Language Models	46
6.2	Threshold Grid Search Results	51
6.3	Grounding DINO - Αποτελέσματα στο Test Set (PROMPT_FULL)	51
6.4	Grounding DINO Per-Class Performance (3 classes)	51
6.5	Prompt Comparison	52
7.1	Ταξινόμηση Object Detection Methods	55
7.2	Σύγκριση Αρχιτεκτονικών Χαρακτηριστικών	55
7.3	Σύγκριση mAP στο Test Set	56
7.4	Precision και Recall Comparison	56
7.5	Per-Class AP@0.5 Comparison	57
7.6	Κατηγοριοποίηση Κλάσεων κατά Δυσκολία	57
7.7	Inference Speed Comparison (Tesla T4 GPU)	57
7.8	Model Complexity Comparison	58

7.9	Training Requirements Comparison	58
7.10	Data Requirements Comparison	58
7.11	Κατηγοριοποίηση False Positives	59
7.12	Κοινές Αιτίες False Negatives	59
7.13	Recommended Model by Use Case	60
7.14	mAP@0.5 with 95% Confidence Intervals	61
8.1	Τελική Σύνοψη Αποτελεσμάτων	64
B'.1	Backbone Network Configuration	80
B'.2	Region Proposal Network Configuration	81
B'.3	ROI Pooling Configuration	81
B'.4	Detection Head Configuration	82
B'.5	Faster R-CNN Training Configuration	82
B'.6	YOLOv3 Model Configuration	83
B'.7	YOLOv3 Anchor Boxes	83
B'.8	YOLOv3 Training Configuration	83
B'.9	YOLOv3 Data Augmentation	84
B'.10	Grounding DINO Model Configuration	84
B'.11	Grounding DINO Inference Configuration	85
B'.12	Grounding DINO Text Prompts	85
B'.13	Hardware και Software Environment	85
Γ'.1	Επίδραση Text Prompt στην Απόδοση	86
Γ'.2	Λεπτομερής Ανάλυση Inference Time (ms)	86
Γ'.3	Throughput ανά Batch Size	87
Γ'.4	Λεπτομερής Κατανομή Instances	87
Γ'.5	Στατιστικά Μεγέθους Bounding Box	87

List of Algorithms

1	Region Proposal Network Forward Pass	11
2	Non-Maximum Suppression	13

Κεφάλαιο 1

Εισαγωγή

1.1 Γενικά περί Object Detection

Η ανίχνευση αντικειμένων (Object Detection) αποτελεί ένα από τα θεμελιώδη προβλήματα της Υπολογιστικής Όρασης (Computer Vision) και της Μηχανικής Μάθησης. Σε αντίθεση με την απλή ταξινόμηση εικόνων (Image Classification), όπου το μοντέλο αποφασίζει σε ποια κατηγορία ανήκει η εικόνα συνολικά, η ανίχνευση αντικειμένων απαιτεί:

1. **Εντοπισμό** (Localization): Προσδιορισμός της θέσης κάθε αντικειμένου μέσω ενός *bounding box* (x, y, πλάτος, ύψος).
2. **Ταξινόμηση** (Classification): Αναγνώριση της κατηγορίας στην οποία ανήκει κάθε αντικείμενο.
3. **Πολλαπλά Αντικείμενα**: Εντοπισμός και ταξινόμηση πολλαπλών αντικειμένων ταυτόχρονα στην ίδια εικόνα.

Μαθηματικά, το πρόβλημα μπορεί να διατυπωθεί ως εξής: Δεδομένης μιας εικόνας $I \in \mathbb{R}^{H \times W \times C}$, ζητείται να βρεθεί ένα σύνολο $\mathcal{D} = \{(b_i, c_i, s_i)\}_{i=1}^N$ όπου:

- $b_i = (x_i, y_i, w_i, h_i) \in \mathbb{R}^4$: οι συντεταγμένες του bounding box
- $c_i \in \{1, 2, \dots, K\}$: η κατηγορία του αντικειμένου
- $s_i \in [0, 1]$: το confidence score
- N : ο αριθμός των ανιχνευμένων αντικειμένων
- K : ο συνολικός αριθμός κατηγοριών

1.2 Ιστορική Εξέλιξη των Object Detectors

Η εξέλιξη των μεθόδων ανίχνευσης αντικειμένων μπορεί να χωριστεί σε τρεις κύριες περιόδους:

1.2.1 Παραδοσιακές Μέθοδοι (1990–2012)

Πριν την επικράτηση των βαθιών νευρωνικών δικτύων, οι μέθοδοι ανίχνευσης βασίζονται σε χειροκίνητα σχεδιασμένα χαρακτηριστικά (hand-crafted features):

- **Viola-Jones (2001) [26]**: Εισήγαγε τα Haar-like features και τον AdaBoost classifier για face detection. Χρησιμοποιούσε sliding window με integral images για γρήγορο υπολογισμό.
- **HOG + SVM (2005) [4]**: Οι Dalal και Triggs πρότειναν τα Histogram of Oriented Gradients (HOG) για pedestrian detection, σε συνδυασμό με linear SVM classifier.
- **DPM (2008–2010) [6]**: Το Deformable Parts Model επέκτεινε το HOG με ιεραρχικά μοντέλα μερών, επιτρέποντας την ανίχνευση αντικειμένων με μεταβλητή εμφάνιση.

Αυτές οι μέθοδοι, αν και επιτυχείς για συγκεκριμένες εφαρμογές, είχαν σημαντικούς περιορισμούς:

- Χαμηλή γενίκευση σε νέες κατηγορίες
- Ευαισθησία σε αλλαγές φωτισμού και γωνίας
- Περιορισμένη απόδοση σε σύνθετες σκηνές

1.2.2 CNN-based Two-Stage Detectors (2014–2017)

Η επανάσταση ξεκίνησε με την εφαρμογή Convolutional Neural Networks στην ανίχνευση:

- **R-CNN (2014) [8]**: O Girshick et al. πρότειναν την χρήση Selective Search για παραγωγή 2000 region proposals, ακολουθούμενη από CNN feature extraction και SVM classification για κάθε region.
- **Fast R-CNN (2015) [7]**: Βελτίωση με κοινή CNN για όλη την εικόνα και RoI Pooling για extraction features ανά proposal.
- **Faster R-CNN (2015) [24]**: Εισαγωγή του Region Proposal Network (RPN) για end-to-end trainable detection.
- **Feature Pyramid Networks (2017) [13]**: Multi-scale feature extraction για καλύτερη ανίχνευση αντικειμένων διαφορετικών μεγεθών.

1.2.3 One-Stage Detectors (2015–σήμερα)

Παράλληλα αναπτύχθηκαν one-stage detectors για real-time εφαρμογές:

- **YOLO (2016) [22]**: "You Only Look Once" - detection σε ένα πέρασμα του δικτύου με division της εικόνας σε grid.
- **SSD (2016) [17]**: Single Shot MultiBox Detector με multi-scale feature maps.
- **RetinaNet (2017) [14]**: Εισαγωγή της Focal Loss για αντιμετώπιση του class imbalance.
- **YOLOv3 (2018) [23]**: Multi-scale detection με Darknet-53 backbone, ανίχνευση σε 3 scales, residual connections.

1.2.4 Vision-Language Models (2021–σήμερα)

Η πιο πρόσφατη εξέλιξη περιλαμβάνει multimodal models:

- **CLIP (2021) [21]**: Contrastive Language-Image Pre-training για zero-shot classification.
- **GLIP (2022) [12]**: Grounded Language-Image Pre-training για phrase grounding.
- **Grounding DINO (2023) [16]**: Open-set object detection με text prompts, συνδυάζοντας DINO με grounded pre-training.

1.3 Two-Stage vs One-Stage Detectors

Οι σύγχρονες μέθοδοι ανίχνευσης χωρίζονται σε δύο κύριες κατηγορίες:

1.3.1 Two-Stage Detectors

Οι two-stage detectors ακολουθούν δύο βήματα:

1. **Stage 1 - Region Proposal**: Παραγωγή υποψήφιων περιοχών (proposals) που πιθανών περιέχουν αντικείμενα. Αυτό γίνεται είτε με παραδοσιακές μεθόδους (Selective Search) είτε με νευρωνικό δίκτυο (RPN).
2. **Stage 2 - Classification & Refinement**: Για κάθε proposal, ταξινόμηση της κατηγορίας και fine-tuning των συντεταγμένων του bounding box.

Πλεονεκτήματα:

- Υψηλότερη ακρίβεια (accuracy)
- Καλύτερη απόδοση σε μικρά αντικείμενα
- Πιο εκλεπτυσμένο localization

Μειονεκτήματα:

- Πιο αργή εκτέλεση (slower inference)
- Πιο πολύπλοκη αρχιτεκτονική
- Υψηλότερες απαιτήσεις μνήμης

1.3.2 One-Stage Detectors

Οι one-stage detectors πραγματοποιούν detection σε ένα βήμα:

1. **Single Pass**: Η εικόνα περνά μία φορά από το δίκτυο, το οποίο παράγει απευθείας τα bounding boxes και τις κατηγορίες.

Πλεονεκτήματα:

- Real-time ταχύτητα (30+ FPS)

- Απλούστερη αρχιτεκτονική
- End-to-end training

Μειονεκτήματα:

- Χαμηλότερη ακρίβεια (ιστορικά)
- Δυσκολία με μικρά αντικείμενα
- Class imbalance πρόβλημα

Πίνακας 1.1: Σύγκριση Two-Stage και One-Stage Detectors

Χαρακτηριστικό	Two-Stage	One-Stage
Αρχιτεκτονική	Πολύπλοκη (RPN + Head)	Απλούστερη
Ταχύτητα (FPS)	5–15	30–150+
mAP (COCO)	Υψηλότερο	Ανταγωνιστικό
Μικρά αντικείμενα	Καλύτερα	Δυσκολότερα
Memory Usage	Υψηλότερη	Χαμηλότερη
Παραδείγματα	R-CNN, Faster R-CNN	YOLO, SSD, RetinaNet

1.4 Εφαρμογή: Traffic Sign Detection

Η ανίχνευση σημάτων οδικής κυκλοφορίας (Traffic Sign Detection - TSD) αποτελεί κρίσιμη εφαρμογή της Computer Vision με σημαντικές πρακτικές επιπτώσεις:

1.4.1 Σημασία στα Αυτόνομα Οχήματα

Τα σύγχρονα συστήματα αυτόνομης οδήγησης βασίζονται στην αυτόματη αναγνώριση σημάτων για:

- **Προσαρμογή ταχύτητας:** Αναγνώριση ορίων ταχύτητας (speedLimit25, speedLimit35)
- **Αναγνώριση προτεραιότητας:** Σήματα STOP, YIELD, merge
- **Προειδοποίηση κινδύνων:** Διαβάσεις πεζών, σηματοδότες
- **Οδηγίες κατεύθυνσης:** keepRight και άλλα

1.4.2 Προκλήσεις του Traffic Sign Detection

Η ανίχνευση σημάτων παρουσιάζει μοναδικές προκλήσεις:

1. **Μικρό μέγεθος:** Τα σήματα καταλαμβάνουν συχνά μικρό μέρος της εικόνας
2. **Μεταβλητές συνθήκες φωτισμού:** Ημέρα/νύχτα, σκιές, αντανακλάσεις
3. **Occlusion:** Μερική κάλυψη από δέντρα, οχήματα
4. **Αλλοίωση:** Μεθωριασμένα ή κατεστραμμένα σήματα
5. **Γωνία θέασης:** Perspective distortion
6. **Ομοιότητα κατηγοριών:** Σήματα με παρόμοια εμφάνιση

1.5 Στόχοι της Εργασίας

Η παρούσα εργασία έχει τους εξής στόχους:

1.5.1 Μέρος Α: Υλοποίηση και Σύγκριση Αρχιτεκτονικών

1. **Faster R-CNN (torchvision)**: Two-stage detector με pretrained ResNet-50 FPN backbone:
 - Χρήση COCO pretrained weights
 - Fine-tuning για traffic sign detection
 - Κατανόηση Region Proposal Network (RPN)
 - RoI Align και Detection Head
2. **YOLOv3**: Εκπαίδευση one-stage detector με ultralytics:
 - Μετατροπή dataset σε YOLO format
 - Fine-tuning pretrained model (Darknet-53)
 - Αξιολόγηση απόδοσης
3. **Συγκριτική Αξιολόγηση**: Σύγκριση με βάση:
 - mAP@0.5 και mAP@0.5:0.95
 - Αρχιτεκτονικές διαφορές
 - Trade-offs ακρίβειας-ταχύτητας

1.5.2 Μέρος Β: Zero-Shot Detection με Grounding DINO

1. Αξιολόγηση zero-shot detection με text prompts
2. Σύγκριση με supervised μεθόδους
3. Ανάλυση πλεονεκτημάτων/μειονεκτημάτων

1.6 Δομή της Αναφοράς

Η αναφορά οργανώνεται ως εξής:

- **Κεφάλαιο 2**: Θεωρητικό υπόβαθρο - αναλυτική περιγραφή των αρχιτεκτονικών, loss functions και μετρικών αξιολόγησης.
- **Κεφάλαιο 3**: Ανάλυση του LISA Traffic Sign Dataset - στατιστικά, preprocessing, augmentation.
- **Κεφάλαιο 4**: Υλοποίηση Faster R-CNN - αρχιτεκτονική, training, αποτελέσματα.
- **Κεφάλαιο 5**: Υλοποίηση YOLOv3 - configuration, training, αποτελέσματα.
- **Κεφάλαιο 6**: Grounding DINO - zero-shot evaluation, σύγκριση.
- **Κεφάλαιο 7**: Συγκριτική ανάλυση και συζήτηση.

- **Κεφάλαιο 8:** Συμπεράσματα και μελλοντική εργασία.
- **Παραρτήματα:** Πλήρης κώδικας, hyperparameters, πρόσθετα αποτελέσματα.

Κεφάλαιο 2

Θεωρητικό Υπόβαθρο

Στο παρόν κεφάλαιο παρουσιάζεται το θεωρητικό υπόβαθρο που απαιτείται για την κατανόηση των αρχιτεκτονικών object detection που υλοποιούνται στην εργασία. Αναλύονται διεξοδικά τα Convolutional Neural Networks, η αρχιτεκτονική Faster R-CNN, η οικογένεια YOLO, και τα vision-language models.

2.1 Convolutional Neural Networks (CNNs)

2.1.1 Βασικές Έννοιες

Τα **Convolutional Neural Networks (CNNs)** αποτελούν τη βάση της σύγχρονης Computer Vision. Σε αντίθεση με τα fully-connected networks, τα CNNs εκμεταλλεύονται τη χωρική δομή των εικόνων μέσω τριών βασικών αρχών:

Ορισμός 2.1 (Sparse Connectivity). Κάθε νευρώνας συνδέεται μόνο με μια μικρή τοπική περιοχή της εισόδου (receptive field), αντί για όλους τους νευρώνες του προηγούμενου layer.

Ορισμός 2.2 (Parameter Sharing). Τα ίδια βάρη (kernel/filter) εφαρμόζονται σε όλες τις θέσεις της εικόνας, μειώνοντας δραματικά τον αριθμό των παραμέτρων.

Ορισμός 2.3 (Translation Equivariance). Αν η είσοδος μετατοπιστεί, η έξοδος μετατοπίζεται αντίστοιχα, επιτρέποντας την αναγνώριση patterns ανεξάρτητα από τη θέση τους.

2.1.2 Convolutional Layer

Η βασική πράξη convolution ορίζεται ως:

$$(I * K)(i, j) = \sum_{m=-k}^k \sum_{n=-k}^k I(i + m, j + n) \cdot K(m, n) \quad (2.1)$$

όπου I είναι η εικόνα εισόδου, K το kernel (ή filter), και k το μισό μέγεθος του kernel.

Για εικόνα με C_{in} κανάλια και C_{out} φίλτρα:

$$Y_{c_{out}}(i, j) = b_{c_{out}} + \sum_{c_{in}=1}^{C_{in}} \sum_{m,n} X_{c_{in}}(i + m, j + n) \cdot W_{c_{out}, c_{in}}(m, n) \quad (2.2)$$

όπου:

- $X \in \mathbb{R}^{C_{in} \times H \times W}$: tensor εισόδου
- $W \in \mathbb{R}^{C_{out} \times C_{in} \times k_h \times k_w}$: βάρη filters
- $b \in \mathbb{R}^{C_{out}}$: bias terms
- $Y \in \mathbb{R}^{C_{out} \times H' \times W'}$: tensor εξόδου

Παράμετροι Convolution

Πίνακας 2.1: Παράμετροι Convolutional Layer

Παράμετρος	Σύμβολο	Περιγραφή
Kernel Size	k	Διάσταση του filter (π.χ. 3×3 , 5×5)
Stride	s	Βήμα μετακίνησης του kernel
Padding	p	Πλήθος pixels που προστίθενται στα άκρα
Dilation	d	Απόσταση μεταξύ στοιχείων του kernel

Η διάσταση εξόδου υπολογίζεται:

$$H_{out} = \left\lceil \frac{H_{in} + 2p - d(k - 1) - 1}{s} \right\rceil + 1 \quad (2.3)$$

2.1.3 Pooling Layers

Tα pooling layers μειώνουν τις χωρικές διαστάσεις:

- **Max Pooling:** $y_{i,j} = \max_{(m,n) \in R_{i,j}} x_{m,n}$
- **Average Pooling:** $y_{i,j} = \frac{1}{|R|} \sum_{(m,n) \in R_{i,j}} x_{m,n}$
όπου $R_{i,j}$ είναι η τοπική περιοχή (π.χ. 2×2).

2.1.4 Activation Functions

$$\text{ReLU}(x) = \max(0, x) \quad (2.4)$$

$$\text{Leaky ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{otherwise} \end{cases} \quad (2.5)$$

$$\text{Mish}(x) = x \cdot \tanh(\text{softplus}(x)) = x \cdot \tanh(\ln(1 + e^x)) \quad (2.6)$$

2.1.5 Batch Normalization

H Batch Normalization [10] κανονικοποιεί τις ενεργοποιήσεις κάθε layer:

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \quad (2.7)$$

$$y_i = \gamma \hat{x}_i + \beta \quad (2.8)$$

όπου μ_B και σ_B^2 είναι ο μέσος και η διακύμανση του mini-batch, και γ, β είναι μαθημένες παράμετροι.

Πλεονεκτήματα:

- Επιτρέπει υψηλότερα learning rates
- Μειώνει την ευαισθησία στην αρχικοποίηση
- Δρα ως regularizer

2.1.6 Residual Connections

Ta Residual Networks (ResNets) [9] εισήγαγαν τα skip connections:

$$y = \mathcal{F}(x, \{W_i\}) + x \quad (2.9)$$

όπου $\mathcal{F}(x)$ είναι η residual function και x η identity mapping.

2.2 Backbone Networks για Object Detection

Ta backbone networks εξάγουν χαρακτηριστικά από τις εικόνες. Τα πιο συνηθισμένα:

2.2.1 VGG Networks

To VGG [25] χρησιμοποιεί μόνο 3×3 convolutions με consistent structure:

$$\text{VGG Block} = [\text{Conv}3 \times 3]_n \rightarrow \text{MaxPool}2 \times 2 \quad (2.10)$$

VGG-16 Architecture:

- Conv(64) $\times 2 \rightarrow$ Pool
- Conv(128) $\times 2 \rightarrow$ Pool
- Conv(256) $\times 3 \rightarrow$ Pool
- Conv(512) $\times 3 \rightarrow$ Pool
- Conv(512) $\times 3 \rightarrow$ Pool
- FC(4096) \rightarrow FC(4096) \rightarrow FC(1000)

2.2.2 ResNet

To ResNet [9] επιτρέπει πολύ βαθιές αρχιτεκτονικές (50, 101, 152 layers) χάρη στα residual connections.

ResNet-50 Summary:

Πίνακας 2.2: ResNet-50 Architecture

Stage	Output Size	Block	Blocks
conv1	112×112	7×7, 64, stride 2	1
conv2_x	56×56	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}$	3
conv3_x	28×28	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}$	4
conv4_x	14×14	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}$	6
conv5_x	7×7	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}$	3

2.2.3 Darknet (YOLO Backbone)

To Darknet [23] χρησιμοποιεί:

- 1×1 convolutions για channel reduction
- 3×3 convolutions για feature extraction
- Residual connections
- Leaky ReLU activation

2.3 Faster R-CNN: Αναλυτική Περιγραφή

Ο Faster R-CNN [24] είναι ο πλέον influential two-stage detector. Αποτελείται από τέσσερα κύρια components:

2.3.1 Architecture Overview

2.3.2 Region Proposal Network (RPN)

Ο RPN είναι ένα fully convolutional network που προβλέπει object proposals σε κάθε θέση του feature map.

Anchor Boxes

Για κάθε θέση (x, y) του feature map ορίζονται k anchor boxes με διαφορετικά scales και aspect ratios:

Ορισμός 2.4 (Anchor Box). Ένα anchor box ορίζεται από τις συντεταγμένες του κέντρου (c_x, c_y) και τις διαστάσεις (w, h) :

$$A = \{(c_x, c_y, w, h) : w = s \cdot \sqrt{r}, h = s / \sqrt{r}\} \quad (2.11)$$

όπου $s \in \text{Scales}$ και $r \in \text{Aspect Ratios}$.

Τυπικές ρυθμίσεις:

- Scales: $\{128, 256, 512\}$ pixels (ή $\{32, 64, 128\}$ για μικρότερες εικόνες)
- Aspect Ratios: $\{0.5, 1.0, 2.0\}$
- Συνολικά anchors ανά θέση: $k = 3 \times 3 = 9$

RPN Architecture

Αλγόριθμος 1 Region Proposal Network Forward Pass

Require: Feature map $F \in \mathbb{R}^{C \times H \times W}$, Anchors $A \in \mathbb{R}^{(H \cdot W \cdot k) \times 4}$

Ensure: Proposals P , Objectness scores S

- ```

1: $X \leftarrow \text{Conv3} \times 3(F)$ ▷ Intermediate features
2: $X \leftarrow \text{ReLU}(X)$
3: $cls \leftarrow \text{Conv1} \times 1(X)$ ▷ Shape: $(2k) \times H \times W$
4: $reg \leftarrow \text{Conv1} \times 1(X)$ ▷ Shape: $(4k) \times H \times W$
5: $scores \leftarrow \text{Softmax}(cls)[:, 1]$ ▷ Objectness probability
6: $deltas \leftarrow \text{Reshape}(reg)$ ▷ Bounding box deltas
7: $P \leftarrow \text{ApplyDeltas}(A, deltas)$ ▷ Decoded proposals
8: $P \leftarrow \text{ClipToImage}(P)$
9: $keep \leftarrow \text{NMS}(P, scores, \text{threshold} = 0.7)$
10: return $P[keep], scores[keep]$

```
- 

## Bounding Box Regression

Αντί να προβλέπει απευθείας συντεταγμένες, ο RPN προβλέπει **μετατοπίσεις** (deltas) σε σχέση με τα anchors:

$$\begin{aligned} t_x &= (x - x_a) / w_a \\ t_y &= (y - y_a) / h_a \\ t_w &= \log(w / w_a) \\ t_h &= \log(h / h_a) \end{aligned} \quad (2.12)$$

όπου  $(x, y, w, h)$  είναι το predicted box και  $(x_a, y_a, w_a, h_a)$  το anchor.

Η αντίστροφη μετατροπή (decoding):

$$\begin{aligned} x &= t_x \cdot w_a + x_a \\ y &= t_y \cdot h_a + y_a \\ w &= w_a \cdot \exp(t_w) \\ h &= h_a \cdot \exp(t_h) \end{aligned} \quad (2.13)$$

### RPN Loss Function

Η συνάρτηση απώλειας του RPN αποτελείται από δύο μέρη:

$$L_{\text{RPN}} = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*) \quad (2.14)$$

όπου:

- $p_i$ : predicted probability of objectness
- $p_i^* \in \{0, 1\}$ : ground truth label (1 αν  $\text{IoU} \geq 0.7$ , 0 αν  $\text{IoU} < 0.3$ )
- $t_i$ : predicted bounding box regression
- $t_i^*$ : ground truth regression targets
- $\lambda$ : balancing weight (τυπικά 10)

### Classification Loss (Cross-Entropy):

$$L_{cls}(p, p^*) = -[p^* \log(p) + (1 - p^*) \log(1 - p)] \quad (2.15)$$

### Regression Loss (Smooth L1):

$$L_{reg}(t, t^*) = \sum_{i \in \{x, y, w, h\}} \text{smooth}_{L_1}(t_i - t_i^*) \quad (2.16)$$

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases} \quad (2.17)$$

H Smooth L1 loss είναι λιγότερο εναίσθητη σε outliers από την L2 loss.

### 2.3.3 RoI Pooling

Αφού παραχθούν τα proposals, πρέπει να εξαχθούν features σταθερού μεγέθους για κάθε proposal.

**Ορισμός 2.5** (RoI Pooling). Δεδομένου feature map  $F$  και RoI με συντεταγμένες  $(x_1, y_1, x_2, y_2)$ , η περιοχή χωρίζεται σε  $H \times W$  bins (π.χ.  $7 \times 7$ ), και εφαρμόζεται max pooling σε κάθε bin:

$$y_{c,i,j} = \max_{(m,n) \in \text{bin}(i,j)} F_{c,m,n} \quad (2.18)$$

**Spatial Scale:** Αν το feature map έχει stride 16 (π.χ. μετά από 4 pooling layers με stride 2), οι συντεταγμένες του RoI διαιρούνται με 16:

$$(x'_1, y'_1, x'_2, y'_2) = (x_1/16, y_1/16, x_2/16, y_2/16) \quad (2.19)$$

### 2.3.4 Detection Head (Stage 2)

To Detection Head λαμβάνει τα pooled features και παράγει:

1. **Classification scores:** πιθανότητα για κάθε κατηγορία (συμπεριλαμβανομένου background)
2. **Bounding box refinement:** fine-tuning των συντεταγμένων ανά κατηγορία

**Architecture:**

$$\begin{aligned} x &= \text{Flatten}(\text{RoIPool}(F, \text{proposals})) \\ x &= \text{FC}_{4096}(\text{ReLU}(\text{FC}_{4096}(x))) \\ \text{cls} &= \text{Softmax}(\text{FC}_{K+1}(x)) \\ \text{reg} &= \text{FC}_{4(K+1)}(x) \end{aligned} \quad (2.20)$$

όπου  $K$  ο αριθμός κατηγοριών (χωρίς background).

### Detection Head Loss

$$L_{\text{head}} = L_{\text{cls}}(p, u) + [u \geq 1]L_{\text{loc}}(t^u, v) \quad (2.21)$$

όπου:

- $p$ : predicted class probabilities
- $u$ : ground truth class ( $0 = \text{background}$ )
- $t^u$ : predicted bbox regression για class  $u$
- $v$ : ground truth bbox targets
- $[u \geq 1]$ : Iverson bracket (1 αν true, 0 αλλιώς) - regression loss μόνο για non-background

### 2.3.5 Non-Maximum Suppression (NMS)

Μετά την πρόβλεψη, πολλαπλά boxes μπορεί να αντιστοιχούν στο ίδιο αντικείμενο. Ο αλγόριθμος NMS αφαιρεί τα redundant boxes:

---

#### Αλγόριθμος 2 Non-Maximum Suppression

---

**Require:** Boxes  $B$ , Scores  $S$ , IoU threshold  $\tau$

**Ensure:** Filtered boxes  $D$

```

1: Sort B by scores S (descending)
2: $D \leftarrow \emptyset$
3: while $B \neq \emptyset$ do
4: $b^* \leftarrow B[0]$ ▷ Box with highest score
5: $D \leftarrow D \cup \{b^*\}$
6: $B \leftarrow B \setminus \{b^*\}$
7: for $b \in B$ do
8: if $\text{IoU}(b, b^*) > \tau$ then
9: $B \leftarrow B \setminus \{b\}$
10: end if
11: end for
12: end while
13: return D

```

---

## 2.4 Intersection over Union (IoU)

Η μετρική IoU (ή Jaccard Index) μετρά την επικάλυψη δύο bounding boxes:

**Ορισμός 2.6** (Intersection over Union).

$$\text{IoU}(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{\text{Area of Intersection}}{\text{Area of Union}} \quad (2.22)$$

Για bounding boxes με συντεταγμένες  $(x_1, y_1, x_2, y_2)$ :

$$\begin{aligned} x_{\text{inter}} &= \max(0, \min(x_2^A, x_2^B) - \max(x_1^A, x_1^B)) \\ y_{\text{inter}} &= \max(0, \min(y_2^A, y_2^B) - \max(y_1^A, y_1^B)) \\ \text{Area}_{\text{inter}} &= x_{\text{inter}} \cdot y_{\text{inter}} \\ \text{Area}_{\text{union}} &= \text{Area}_A + \text{Area}_B - \text{Area}_{\text{inter}} \\ \text{IoU} &= \frac{\text{Area}_{\text{inter}}}{\text{Area}_{\text{union}}} \end{aligned} \quad (2.23)$$

## 2.5 YOLO: You Only Look Once

Η οικογένεια YOLO [22, 23] αποτελεί την πλέον διαδεδομένη σειρά one-stage detectors. Στην παρούσα εργασία χρησιμοποιείται ο **YOLOv3** [23].

### 2.5.1 Βασική Ιδέα

Αντί για region proposals, το YOLO:

1. Χωρίζει την εικόνα σε  $S \times S$  grid
2. Κάθε cell προβλέπει  $B$  bounding boxes
3. Κάθε box περιλαμβάνει:  $(x, y, w, h, \text{confidence})$
4. Κάθε cell προβλέπει  $C$  class probabilities

$$\text{Output tensor} \in \mathbb{R}^{S \times S \times (B \cdot 5 + C)} \quad (2.24)$$

### 2.5.2 YOLOv3 Architecture

#### Darknet-53 Backbone

To Darknet-53 χρησιμοποιεί residual connections:

Πίνακας 2.3: Darknet-53 Architecture

| Type                      | Filters  | Size           | Output           |
|---------------------------|----------|----------------|------------------|
| Convolutional             | 32       | $3 \times 3$   | $256 \times 256$ |
| Convolutional             | 64       | $3 \times 3/2$ | $128 \times 128$ |
| Residual Block            | 32/64    | -              | $128 \times 128$ |
| Convolutional             | 128      | $3 \times 3/2$ | $64 \times 64$   |
| Residual Block $\times 2$ | 64/128   | -              | $64 \times 64$   |
| Convolutional             | 256      | $3 \times 3/2$ | $32 \times 32$   |
| Residual Block $\times 8$ | 128/256  | -              | $32 \times 32$   |
| Convolutional             | 512      | $3 \times 3/2$ | $16 \times 16$   |
| Residual Block $\times 8$ | 256/512  | -              | $16 \times 16$   |
| Convolutional             | 1024     | $3 \times 3/2$ | $8 \times 8$     |
| Residual Block $\times 4$ | 512/1024 | -              | $8 \times 8$     |

### Multi-Scale Detection

To YOLOv3 κάνει predictions σε τρία scales:

- **Scale 1 (13×13)**: Μεγάλα αντικείμενα
- **Scale 2 (26×26)**: Μεσαία αντικείμενα
- **Scale 3 (52×52)**: Μικρά αντικείμενα

Χρησιμοποιούνται 9 anchor boxes συνολικά (3 ανά scale), με k-means clustering στο training set.

### 2.5.3 YOLOv3 μέσω Ultralytics

H ultralytics βιβλιοθήκη παρέχει optimized έκδοση του YOLOv3 (yolov3u.pt):

- **Darknet-53 backbone**: 53 convolutional layers με residual connections
- **Multi-scale detection**: 3 detection scales (13×13, 26×26, 52×52)
- **Anchor-based**: 9 anchors (3 ανά scale)
- **Mosaic augmentation**: Σύνθεση 4 εικόνων
- **Pretrained στο COCO**: Transfer learning για καλύτερη απόδοση

### 2.5.4 YOLO Loss Function

$$L = \lambda_{\text{coord}} L_{\text{coord}} + \lambda_{\text{obj}} L_{\text{obj}} + \lambda_{\text{noobj}} L_{\text{noobj}} + \lambda_{\text{cls}} L_{\text{cls}} \quad (2.25)$$

**Coordinate Loss:**

$$L_{\text{coord}} = \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 + (\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2] \quad (2.26)$$

**Objectness Loss:**

$$L_{\text{obj}} = \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 \quad (2.27)$$

## 2.6 Grounding DINO: Vision-Language Detection

To Grounding DINO [16] συνδυάζει DINO (DETR with Improved deNoising anchor boxes) με grounded pre-training για open-vocabulary detection.

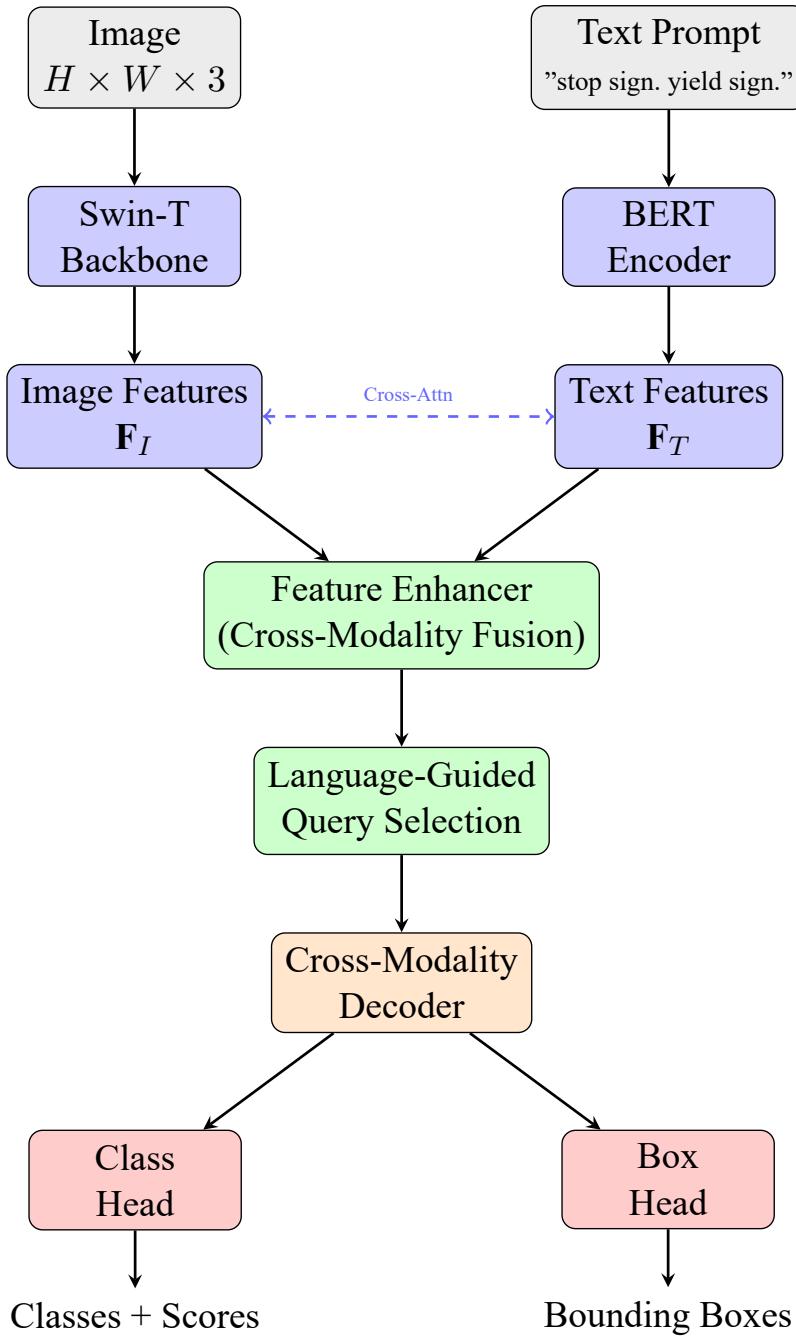
### 2.6.1 DETR Background

O DETR [3] εισήγαγε transformer-based detection:

$$\text{DETR} = \text{CNN}_{\text{backbone}} \rightarrow \text{Transformer}_{\text{encoder}} \rightarrow \text{Transformer}_{\text{decoder}} \rightarrow \text{FFN}_{\text{heads}} \quad (2.28)$$

### 2.6.2 Grounding DINO Architecture

- **Image Encoder:** Swin Transformer backbone
- **Text Encoder:** BERT-based model
- **Feature Enhancer:** Cross-modality fusion
- **Language-Guided Query Selection:** Text-informed object queries
- **Cross-Modality Decoder:** Fusion of visual and textual features



Σχήμα 2.1: Αρχιτεκτονική Grounding DINO: Ο Swin Transformer επεξεργάζεται την εικόνα ενώ ο BERT encoder επεξεργάζεται το text prompt. Τα features ενοποιούνται μέσω cross-modality fusion και το Language-Guided Query Selection επιλέγει τα queries με βάση το κείμενο.

### 2.6.3 Zero-Shot Detection

To Grounding DINO επιτρέπει detection με text prompts:

$$\text{Detections} = f_{\text{GDINO}}(\text{Image}, \text{TextPrompt}) \quad (2.29)$$

**Παράδειγμα prompt:** "stop sign. yield sign. speed limit sign."  
Κάθε φράση (χωρισμένη με ".") αντιστοιχεί σε διαφορετική κατηγορία.

## 2.7 Μετρικές Αξιολόγησης

### 2.7.1 Precision και Recall

**Ορισμός 2.7** (Precision). Ποσοστό σωστών ανιχνεύσεων μεταξύ όλων των ανιχνεύσεων:

$$\text{Precision} = \frac{TP}{TP + FP} \quad (2.30)$$

**Ορισμός 2.8** (Recall). Ποσοστό σωστών ανιχνεύσεων μεταξύ όλων των πραγματικών αντικειμένων:

$$\text{Recall} = \frac{TP}{TP + FN} \quad (2.31)$$

όπου:

- **TP (True Positive)**: Σωστή ανίχνευση ( $\text{IoU} \geq \text{threshold}$  με ground truth)
- **FP (False Positive)**: Λανθασμένη ανίχνευση
- **FN (False Negative)**: Μη ανιχνευμένο αντικείμενο

### 2.7.2 Average Precision (AP)

Η AP υπολογίζεται ως η περιοχή κάτω από την Precision-Recall curve:

$$\text{AP} = \int_0^1 p(r) dr \quad (2.32)$$

Στην πράξη χρησιμοποιείται η 11-point interpolation ή all-point interpolation:

$$\text{AP}_{11} = \frac{1}{11} \sum_{r \in \{0, 0.1, \dots, 1.0\}} p_{\text{interp}}(r) \quad (2.33)$$

$$p_{\text{interp}}(r) = \max_{r' \geq r} p(r') \quad (2.34)$$

### 2.7.3 mean Average Precision (mAP)

**Ορισμός 2.9** (mAP). Ο μέσος όρος της AP για όλες τις κατηγορίες:

$$\text{mAP} = \frac{1}{K} \sum_{k=1}^K \text{AP}_k \quad (2.35)$$

### 2.7.4 COCO mAP Metrics

To COCO benchmark [15] ορίζεται:

Πίνακας 2.4: COCO Evaluation Metrics

| Metric       | Description                                             |
|--------------|---------------------------------------------------------|
| mAP@0.5      | mAP με IoU threshold = 0.5 (PASCAL VOC style)           |
| mAP@0.75     | mAP με IoU threshold = 0.75 (stricter)                  |
| mAP@0.5:0.95 | Μέσος όρος mAP για IoU $\in \{0.5, 0.55, \dots, 0.95\}$ |
| $AP_S$       | AP για small objects (area < $32^2$ )                   |
| $AP_M$       | AP για medium objects ( $32^2 < \text{area} < 96^2$ )   |
| $AP_L$       | AP για large objects (area > $96^2$ )                   |

## 2.8 Data Augmentation

Η αύξηση δεδομένων (data augmentation) είναι κρίσιμη για την εκπαίδευση robust detectors.

### 2.8.1 Geometric Transformations

1. **Horizontal Flip:** Αναστροφή κατά τον οριζόντιο άξονα

$$x' = W - x, \quad y' = y \quad (2.36)$$

2. **Rotation:** Περιστροφή κατά γωνία  $\theta$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x - c_x \\ y - c_y \end{bmatrix} + \begin{bmatrix} c_x \\ c_y \end{bmatrix} \quad (2.37)$$

3. **Random Crop:** Τυχαίο κόψιμο περιοχής της εικόνας

4. **Scale/Zoom:** Μεγέθυνση/σμίκρυνση κατά factor  $s$

$$x' = s \cdot x, \quad y' = s \cdot y \quad (2.38)$$

### 2.8.2 Photometric Transformations

1. **Brightness:**  $I' = I + \delta$
2. **Contrast:**  $I' = \alpha \cdot I$
3. **Saturation:** Αλλαγή κορεσμού στο HSV space
4. **Hue:** Αλλαγή απόχρωσης

### 2.8.3 Mosaic Augmentation (YOLO)

Η Mosaic augmentation συνδυάζει 4 εικόνες:

# Κεφάλαιο 3

## Ανάλυση Dataset

Στο παρόν κεφάλαιο παρουσιάζεται λεπτομερής ανάλυση του LISA Traffic Sign Dataset που χρησιμοποιείται για την εκπαίδευση και αξιολόγηση των μοντέλων object detection.

### 3.1 LISA Traffic Sign Dataset

To **LISA Traffic Sign Dataset** [20] αποτελεί ένα από τα πιο διαδεδομένα datasets για traffic sign detection, αναπτυγμένο από το Laboratory for Intelligent and Safe Automobiles (LISA) του UC San Diego.

#### 3.1.1 Περιγραφή Dataset

Για τις ανάγκες της εργασίας χρησιμοποιείται μια υποομάδα του dataset με τα εξής χαρακτηριστικά:

#### 3.1.2 Κατηγορίες Σημάτων

To dataset περιλαμβάνει 9 κατηγορίες σημάτων κυκλοφορίας:

Πίνακας 3.1: Κατηγορίες Σημάτων και Περιγραφή

| ID | Κατηγορία          | Περιγραφή (ΕΛ)            | Τύπος           |
|----|--------------------|---------------------------|-----------------|
| 0  | keepRight          | Κίνηση δεξιά              | Ρυθμιστικό      |
| 1  | merge              | Συγχώνευση λωρίδων        | Προειδοποιητικό |
| 2  | pedestrianCrossing | Διάβαση πεζών             | Προειδοποιητικό |
| 3  | signalAhead        | Σηματοδότης μπροστά       | Προειδοποιητικό |
| 4  | speedLimit25       | Όριο ταχύτητας 25 mph     | Ρυθμιστικό      |
| 5  | speedLimit35       | Όριο ταχύτητας 35 mph     | Ρυθμιστικό      |
| 6  | stop               | STOP                      | Ρυθμιστικό      |
| 7  | yield              | Παραχώρηση προτεραιότητας | Ρυθμιστικό      |
| 8  | yieldAhead         | Yield μπροστά             | Προειδοποιητικό |

## 3.2 Στατιστική Ανάλυση

### 3.2.1 Κατανομή Κατηγοριών

Η κατανομή των annotations ανά κατηγορία παρουσιάζει ανισορροπία (class imbalance):

### 3.2.2 Ανάλυση Μεγέθους Bounding Boxes

### 3.2.3 Κατανομή Θέσης στην Εικόνα

Η θέση των σημάτων στην εικόνα δεν είναι ομοιόμορφη - τα σήματα τείνουν να εμφανίζονται στο πάνω μέρος και στις άκρες:

## 3.3 Annotation Format

### 3.3.1 CSV Structure

Τα annotations αποθηκεύονται σε CSV αρχείο με delimiter το semicolon (;):

```

1 filename;x1;y1;x2;y2;class
2 train/img001.png;120;80;180;140;stop
3 train/img001.png;350;100;400;150;yield
4 train/img002.png;200;90;260;155	speedLimit25
5 ...

```

### 3.3.2 Μετατροπή Συντεταγμένων

Οι συντεταγμένες είναι σε format ( $x_1, y_1, x_2, y_2$ ) (Pascal VOC format):

- $(x_1, y_1)$ : Πάνω-αριστερή γωνία
- $(x_2, y_2)$ : Κάτω-δεξιά γωνία

Για μετατροπή σε YOLO format ( $c_x, c_y, w, h$ ) normalized:

$$\begin{aligned}
 c_x &= \frac{x_1 + x_2}{2W} \\
 c_y &= \frac{y_1 + y_2}{2H} \\
 w &= \frac{x_2 - x_1}{W} \\
 h &= \frac{y_2 - y_1}{H}
 \end{aligned} \tag{3.1}$$

όπου  $W, H$  οι διαστάσεις της εικόνας.

## 3.4 Data Loading Pipeline

### 3.4.1 Loading Function

Η φόρτωση δεδομένων υλοποιείται ως εξής:

---

```

1 def load_image(img_path, annotations, transform):
2 img = cv2.imread(img_path)
3 img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
4
5 boxes, labels = [], []
6 for _, ann in annotations.iterrows():
7 if ann['x_max'] > ann['x_min'] and ann['y_max'] > ann['y_min']:
8 boxes.append([ann['x_min'], ann['y_min'],
9 ann['x_max'], ann['y_max']])
10 labels.append(CLASS_TO_IDX[ann['class_name']] + 1)
11
12 if len(boxes) == 0:
13 boxes, labels = [[0, 0, 1, 1]], [0]
14
15 transformed = transform(image=img, bboxes=boxes, labels=labels)
16
17 return transformed['image'], {
18 'boxes': torch.tensor(transformed['bboxes'], dtype=torch.float32),
19 'labels': torch.tensor(transformed['labels'], dtype=torch.int64)
20 }

```

---

### 3.4.2 Data Splits

Τα splits καθορίζονται από το filename path:

---

```

1 df['split'] = df['filename'].apply(lambda x: x.split('/')[0])
2 # Αποτέλεσμα: 'train', 'val', 'test'

```

---

## 3.5 Data Augmentation Strategy

Σύμφωνα με την εκφώνηση της εργασίας, εφαρμόζονται οι ακόλουθες μετατροπές:

### 3.5.1 Augmentations για Training

Πίνακας 3.2: Data Augmentation Pipeline

| # | Augmentation      | Παράμετροι       | Probability |
|---|-------------------|------------------|-------------|
| 1 | Resize            | 384×512          | 1.0         |
| 2 | HorizontalFlip    | -                | 0.5         |
| 3 | RandomResizedCrop | scale=(0.8, 1.0) | 0.5         |
| 4 | Rotate            | limit=±10°       | 0.5         |
| 5 | Affine (Zoom)     | scale=(0.9, 1.1) | 0.5         |
| 6 | Resize (final)    | 384×512          | 1.0         |
| 7 | Normalize         | ImageNet stats   | 1.0         |

### 3.5.2 Υλοποίηση με Albumentations

Χρησιμοποιείται η βιβλιοθήκη `albumentations` [2] που υποστηρίζει αυτόματο transformation των bounding boxes:

```

1 import albumentations as A
2 from albumentations.pytorch import ToTensorV2
3
4 IMG_SIZE = (384, 512)
5
6 train_transform = A.Compose([
7 A.Resize(height=IMG_SIZE[0], width=IMG_SIZE[1]),
8 A.HorizontalFlip(p=0.5),
9 A.RandomResizedCrop(size=IMG_SIZE, scale=(0.8, 1.0), p=0.5),
10 A.Rotate(limit=10, p=0.5, border_mode=cv2.BORDER_CONSTANT),
11 A.Affine(scale=(0.9, 1.1), p=0.5),
12 A.Resize(height=IMG_SIZE[0], width=IMG_SIZE[1]),
13 A.Normalize(mean=[0.485, 0.456, 0.406],
14 std=[0.229, 0.224, 0.225]),
15 ToTensorV2()
16], bbox_params=A.BboxParams(
17 format='pascal_voc',
18 label_fields=['labels'],
19 min_visibility=0.3
20))
```

### 3.5.3 Αιτιολόγηση Επιλογών

1. **HorizontalFlip**: Τα σήματα μπορούν να εμφανίζονται σε οποιαδήποτε πλευρά του δρόμου.
2. **RandomResizedCrop**: Προσομοιώνει διαφορετικές αποστάσεις από τα σήματα και partial occlusion.
3. **Rotation**: Μικρές κλίσεις λόγω γωνίας κάμερας ή κλίσης του σήματος.
4. **Zoom/Scale**: Διαφορετικές αποστάσεις και μεγέθη σημάτων.

5. **min\_visibility=0.3**: Αποφυγή augmented samples óπου το bounding box κόβεται υπερβολικά.

### 3.5.4 Validation/Test Transforms

Για evaluation χρησιμοποιούνται μόνο deterministic transforms:

```

1 val_transform = A.Compose([
2 A.Resize(height=IMG_SIZE[0], width=IMG_SIZE[1]),
3 A.Normalize(mean=[0.485, 0.456, 0.406],
4 std=[0.229, 0.224, 0.225]),
5 ToTensorV2()
6], bbox_params=A.BboxParams(
7 format='pascal_voc',
8 label_fields=['labels']
9))
```

## 3.6 DataLoader Configuration

### 3.6.1 Faster R-CNN DataLoader

Λόγω περιορισμών μνήμης GPU, χρησιμοποιείται μικρό batch size:

```

1 BATCH_SIZE = 1 # Λόγω GPU memory constraints
2
3 def collate(batch):
4 return tuple(zip(*batch))
5
6 train_loader = DataLoader(
7 train_ds,
8 batch_size=BATCH_SIZE,
9 shuffle=True,
10 collate_fn=collate
11)
```

**Gradient Accumulation:** Για effective batch size 4, χρησιμοποιείται gradient accumulation με 4 steps.

### 3.6.2 YOLO DataLoader

To YOLO χρησιμοποιεί το δικό του data loading pipeline με:

```

1 YOLO_BATCH = 16
2 YOLO_IMGSZ = 640
3
4 # data.yaml
5 path: /content/yolo_dataset
6 train: train/images
7 val: val/images
8 test: test/images
```

```

9 nc: 9
10 names: ['keepRight', 'merge', 'pedestrianCrossing',
11 'signalAhead', 'speedLimit25', 'speedLimit35',
12 'stop', 'yield', 'yieldAhead']

```

## 3.7 Οπτικοποίηση Δειγμάτων

## 3.8 Προκλήσεις του Dataset

### 3.8.1 Class Imbalance

Ορισμένες κατηγορίες έχουν σημαντικά περισσότερα samples:

- **Αντιμετώπιση:** Weighted loss ή oversampling (δεν εφαρμόστηκε στην παρούσα εργασία)

### 3.8.2 Μικρά Αντικείμενα

Πολλά σήματα καταλαμβάνουν μικρό ποσοστό της εικόνας:

- **Αντιμετώπιση:**
  - Multi-scale detection (FPN, YOLO multi-scale)
  - Larger input resolution
  - Μικρότερα anchor scales

### 3.8.3 Παρόμοιες Κατηγορίες

Ορισμένες κατηγορίες έχουν οπτική ομοιότητα:

- speedLimit25 vs speedLimit35 (μόνο ο αριθμός διαφέρει)
- yield vs yieldAhead (παρόμοιο σχήμα)
- **Αντιμετώπιση:** Υψηλότερη ανάλυση, attention mechanisms

## 3.9 Σύνοψη

To LISA Traffic Sign Dataset παρέχει ένα challenging benchmark για traffic sign detection με:

- 900 εικόνες σε train/val/test splits
- 9 κατηγορίες σημάτων με class imbalance
- Ποικιλία μεγεθών bounding boxes
- Ρεαλιστικές συνθήκες (outdoor, variable lighting)

Η στρατηγική augmentation που εφαρμόζεται (Flip, Crop, Rotate, Zoom) στοχεύει στην αύξηση της γενικευσιμότητας των μοντέλων.

# Κεφάλαιο 4

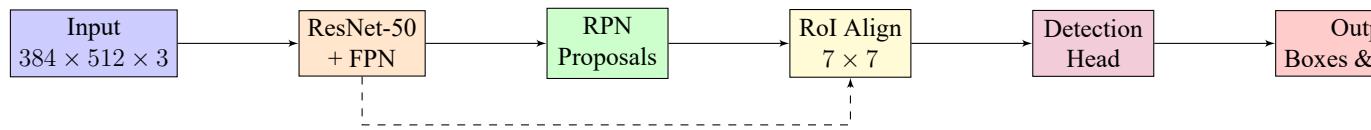
## Υλοποίηση Faster R-CNN με Torchvision

Στο παρόν κεφάλαιο περιγράφεται η υλοποίηση του Faster R-CNN χρησιμοποιώντας το pretrained μοντέλο `fasterrcnn_resnet50_fpn` από την βιβλιοθήκη `torchvision`. Η προσέγγιση αυτή αξιοποιεί transfer learning από τα COCO pretrained weights, επιτρέποντας αποδοτικό fine-tuning στο LISA Tiny dataset.

### 4.1 Αρχιτεκτονική Overview

#### 4.1.1 Συνολική Δομή

Η υλοποίηση χρησιμοποιεί το pretrained `fasterrcnn_resnet50_fpn` από `torchvision` με αντικατάσταση του classification head για 10 κλάσεις (9 traffic signs + background). Η αρχιτεκτονική αποτελείται από:



Σχήμα 4.1: Faster R-CNN με ResNet-50 FPN Backbone (`torchvision`).

Πίνακας 4.1: Αρχιτεκτονική Faster R-CNN (`torchvision` pretrained)

| Component     | Περιγραφή                   | Output                           |
|---------------|-----------------------------|----------------------------------|
| Input         | RGB Image                   | $3 \times H \times W$            |
| Backbone      | ResNet-50 (COCO pretrained) | Multi-scale features             |
| FPN           | Feature Pyramid Network     | P2-P6 features                   |
| RPN           | Region Proposal Network     | Top-N proposals                  |
| RoI Align     | 7x7 aligned pooling         | $N \times 256 \times 7 \times 7$ |
| Box Head      | 2xFC(1024)                  | $N \times 1024$                  |
| Cls Predictor | FC(10)                      | $N \times 10$ (classes)          |
| Box Predictor | FC(40)                      | $N \times 40$ (4 per class)      |

## 4.2 Backbone Network

### 4.2.1 Επιλογή Pretrained Model

Χρησιμοποιήθηκε το `fasterrcnn_resnet50_fpn` με COCO pretrained weights, γιατί:

- Εξαιρετική επίδοση σε object detection benchmarks
- Transfer learning μειώνει χρόνο εκπαίδευσης δραστικά
- H FPN (Feature Pyramid Network) βελτιώνει ανίχνευση σε πολλαπλές κλίμακες
- 41M παράμετροι, αλλά αποδοτική χρήση με frozen backbone

### 4.2.2 Model Building

```

1 from torchvision.models.detection import fasterrcnn_resnet50_fpn
2 from torchvision.models.detection import FasterRCNN_ResNet50_FPN_Weights
3 from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
4
5 def build_faster_rcnn(num_classes):
6 # Load pretrained Faster R-CNN (COCO weights)
7 model = fasterrcnn_resnet50_fpn(
8 weights=FasterRCNN_ResNet50_FPN_Weights.COCO_V1
9)
10
11 # Replace classification head for our classes
12 in_features = model.roi_heads.box_predictor.cls_score.in_features
13 model.roi_heads.box_predictor = FastRCNNPredictor(
14 in_features, num_classes
15)
16
17 return model
18
19 NUM_CLASSES = 10 # 9 traffic signs + background
20 faster_rcnn = build_faster_rcnn(NUM_CLASSES)

```

### 4.2.3 ResNet-50 + FPN Backbone

Η αρχιτεκτονική χρησιμοποιεί ResNet-50 ως backbone με Feature Pyramid Network:

Πίνακας 4.2: ResNet-50 FPN Architecture (torchvision)

| Stage       | Channels | Stride | Notes                   |
|-------------|----------|--------|-------------------------|
| C1 (Conv1)  | 64       | 4      | Initial conv + maxpool  |
| C2 (Layer1) | 256      | 4      | 3 Bottleneck blocks     |
| C3 (Layer2) | 512      | 8      | 4 Bottleneck blocks     |
| C4 (Layer3) | 1024     | 16     | 6 Bottleneck blocks     |
| C5 (Layer4) | 2048     | 32     | 3 Bottleneck blocks     |
| P2-P5       | 256      | 4-32   | FPN lateral connections |
| P6          | 256      | 64     | Pooling from P5         |

H FPN επιτρέπει multi-scale detection με παραγωγή features σε 5 επίπεδα (P2-P6), καθένα με 256 channels αλλά διαφορετικό stride.

## 4.3 Region Proposal Network (RPN)

### 4.3.1 Περιγραφή RPN (torchvision)

To RPN του torchvision χρησιμοποιεί τα multi-scale features από το FPN για anchor-based proposal generation. Ο RPN εκτελείται σε κάθε επίπεδο του pyramid (P2-P6) με διαφορετικά anchor sizes.

Πίνακας 4.3: RPN Configuration (torchvision default)

| Parameter      | Value                             |
|----------------|-----------------------------------|
| Anchor sizes   | (32, 64, 128, 256, 512) per level |
| Aspect ratios  | (0.5, 1.0, 2.0)                   |
| NMS threshold  | 0.7                               |
| Pre-NMS top-k  | 2000 (train) / 1000 (test)        |
| Post-NMS top-k | 2000 (train) / 1000 (test)        |

### 4.3.2 Anchor Generation

Στο FPN-based RPN, διαφορετικά anchor sizes αντιστοιχούν σε διαφορετικά pyramid levels:

- **P2** (stride 4): 32px anchors
- **P3** (stride 8): 64px anchors
- **P4** (stride 16): 128px anchors
- **P5** (stride 32): 256px anchors
- **P6** (stride 64): 512px anchors

## 4.4 ROI Align

To torchvision χρησιμοποιεί **RoI Align** αντί για RoI Pooling, που παρέχει ακριβέστερο alignment χωρίς quantization errors:

Πίνακας 4.4: RoI Align Configuration

| Parameter      | Value        |
|----------------|--------------|
| Output size    | $7 \times 7$ |
| Sampling ratio | 2            |
| Aligned        | True         |

## 4.5 Detection Head (Box Predictor)

### 4.5.1 FastRCNNPredictor

Αντικαθιστούμε μόνο το classification head για τις δικές μας κλάσεις:

```

1 from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
2
3 # Get number of input features from existing predictor
4 in_features = model.roi_heads.box_predictor.cls_score.in_features
5
6 # Replace with our predictor (10 = 9 classes + background)
7 model.roi_heads.box_predictor = FastRCNNPredictor(
8 in_features,
9 num_classes=10
10)

```

Η αρχιτεκτονική του box head παραμένει η ίδια ( $2 \times \text{FC}(1024)$ ), μόνο η τελική classification/regression layer αλλάζει.

## 4.6 Loss Functions

To torchvision Faster R-CNN υπολογίζει αυτόματα τα losses κατά το training:

Πίνακας 4.5: Faster R-CNN Loss Components

| Loss             | Type          | Description         |
|------------------|---------------|---------------------|
| loss_objectness  | BCE           | RPN objectness      |
| loss_rpn_box_reg | Smooth L1     | RPN box regression  |
| loss_classifier  | Cross-Entropy | Classification loss |
| loss_box_reg     | Smooth L1     | Box regression loss |

```

1 # During training, model returns Loss dict
2 model.train()
3 loss_dict = model(images, targets)
4 # Loss_dict = {
5 # 'Loss_classifier': tensor,
6 # 'Loss_box_reg': tensor,
7 # 'Loss_objectness': tensor,
8 # 'Loss_rpn_box_reg': tensor
9 # }
10 total_loss = sum(loss_dict.values())

```

## 4.7 Training Configuration

### 4.7.1 Hyperparameters

Πίνακας 4.6: Training Hyperparameters (Torchvision Faster R-CNN)

| Parameter         | Value  | Αιτιολόγηση                             |
|-------------------|--------|-----------------------------------------|
| Epochs            | 10     | Αρκετά για fine-tuning pretrained model |
| Batch Size        | 1      | GPU memory constraint (Colab T4)        |
| Learning Rate     | 0.005  | Standard για fine-tuning                |
| Optimizer         | SGD    | Momentum για σταθερότερη σύγκλιση       |
| Momentum          | 0.9    | Standard value                          |
| Weight Decay      | 0.0005 | L2 regularization                       |
| LR Scheduler      | StepLR | Decay κάθε 7 epochs                     |
| LR Gamma          | 0.1    | LR → LR/10 σε κάθε step                 |
| Gradient Clipping | 10.0   | Αποφυγή exploding gradients             |

### 4.7.2 Optimizer Setup

```

1 optimizer = torch.optim.SGD(
2 faster_rcnn.parameters(),
3 lr=0.005,
4 momentum=0.9,
5 weight_decay=0.0005
6)
7 scheduler = torch.optim.lr_scheduler.StepLR(
8 optimizer, step_size=7, gamma=0.1
9)

```

## 4.8 Training Loop

```

1 train_losses = []
2 NUM_EPOCHS = 10
3
4 for epoch in range(NUM_EPOCHS):
5 faster_rcnn.train()
6 epoch_loss = 0
7
8 pbar = tqdm(train_loader, desc=f'Epoch {epoch+1}/{NUM_EPOCHS}')
9 for images, targets in pbar:
10 images = [img.to(device) for img in images]
11 targets = [{k: v.to(device) for k, v in t.items()}
12 for t in targets]
13
14 # Forward - model returns loss dict in training mode
15 loss_dict = faster_rcnn(images, targets)
16 loss = sum(loss_dict.values())
17
18 optimizer.zero_grad()

```

```

19 loss.backward()
20 torch.nn.utils.clip_grad_norm_(
21 faster_rcnn.parameters(), max_norm=10.0
22)
23 optimizer.step()
24
25 epoch_loss += loss.item()
26 pbar.set_postfix({'loss': f'{loss.item():.4f}'})
27
28 scheduler.step()
29 avg_loss = epoch_loss / len(train_loader)
30 train_losses.append(avg_loss)
31
32 print(f'Epoch {epoch+1}/{NUM_EPOCHS} - '
33 f'Loss: {avg_loss:.4f} - '
34 f'LR: {scheduler.get_last_lr()[0]:.6f}')
35
36 if avg_loss == min(train_losses):
37 torch.save(faster_rcnn.state_dict(), 'faster_rcnn_best.pth')
38 print('→ Saved best model')
39
40 torch.save(faster_rcnn.state_dict(), 'faster_rcnn_final.pth')

```

## 4.9 Inference Pipeline

Κατά το inference, το μοντέλο επιστρέφει predictions αντί για losses:

```

1 def evaluate_frcnn(model, data_loader, device):
2 model.eval()
3 metric = MeanAveragePrecision(iou_type='bbox')
4
5 with torch.no_grad():
6 for images, targets in tqdm(data_loader, desc='Evaluating'):
7 images = [img.to(device) for img in images]
8
9 # Inference mode - returns predictions
10 outputs = model(images)
11
12 preds = []
13 gts = []
14 for out, tgt in zip(outputs, targets):
15 preds.append({
16 'boxes': out['boxes'].cpu(),
17 'scores': out['scores'].cpu(),
18 'labels': out['labels'].cpu()
19 })
20 gts.append({
21 'boxes': tgt['boxes'],
22 'labels': tgt['labels']
23 })
24
25 metric.update(preds, gts)
26
27 return metric.compute()
28

```

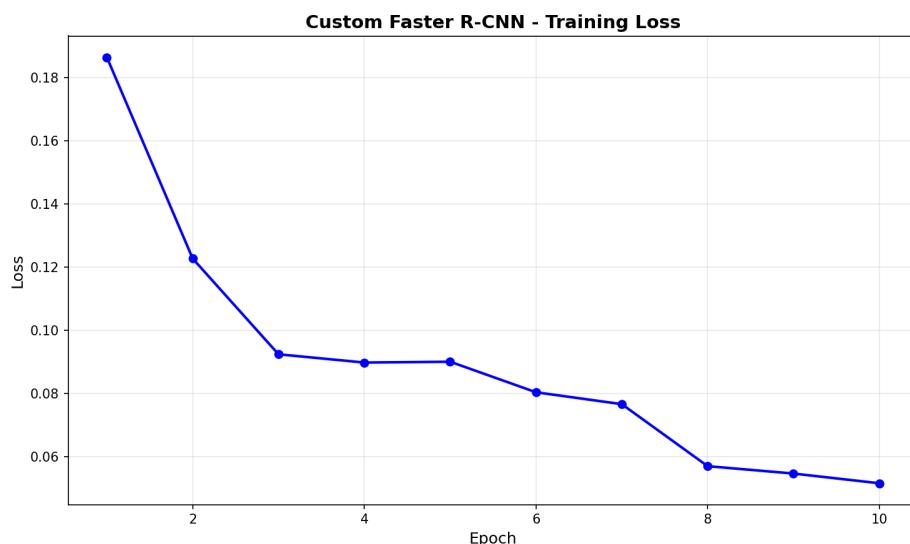
```

29 # Evaluate on test set
30 faster_rcnn.load_state_dict(
31 torch.load('faster_rcnn_best.pth', weights_only=True)
32)
33 results = evaluate_frcnn(faster_rcnn, test_loader, device)
34
35 print(f"mAP@0.5: {results['map_50'].item():.4f}")
36 print(f"mAP@0.5:0.95: {results['map'].item():.4f}")

```

## 4.10 Αποτελέσματα Εκπαίδευσης

### 4.10.1 Training Loss Curve



Σχήμα 4.2: Training loss του Faster R-CNN κατά τη διάρκεια της εκπαίδευσης.

### 4.10.2 mAP Results

Πίνακας 4.7: Faster R-CNN (torchvision) - Αποτελέσματα mAP στο Test Set

| Metric       | Value         |
|--------------|---------------|
| mAP@0.5      | <b>0.8729</b> |
| mAP@0.5:0.95 | 0.7113        |
| mAP@0.75     | 0.8060        |

### 4.10.3 Per-Class AP

Πίνακας 4.8: Per-Class Average Precision (AP@0.5)

| Class              | AP@0.5 |
|--------------------|--------|
| keepRight          | [PH]   |
| merge              | [PH]   |
| pedestrianCrossing | [PH]   |
| signalAhead        | [PH]   |
| speedLimit25       | [PH]   |
| speedLimit35       | [PH]   |
| stop               | [PH]   |
| yield              | [PH]   |
| yieldAhead         | [PH]   |

## 4.11 Qualitative Results



Σχήμα 4.3: Οπτικοποίηση predictions του Faster R-CNN σε δείγματα του test set.

## 4.12 Συζήτηση

### 4.12.1 Πλεονεκτήματα Torchvision Approach

1. **Transfer Learning:** Η χρήση COCO pretrained weights επιτρέπει γρήγορο fine-tuning σε μικρό dataset
2. **FPN Integration:** Η Feature Pyramid Network βελτιώνει detection σε διάφορες κλίμακες
3. **Optimized Implementation:** Η torchvision υλοποίηση είναι βελτιστοποιημένη για ταχύτητα
4. **RoI Align:** Ακριβέστερο από ROI Pooling χωρίς quantization errors

### 4.12.2 Παρατηρήσεις

- Μόνο το classification head αντικαταστάθηκε για τις 10 κλάσεις μας
- Όλα τα άλλα components (backbone, RPN, ROI heads) παραμένουν pretrained
- To fine-tuning 10 epochs ήταν αρκετό για convergence
- Memory usage: 6GB VRAM με batch size 1

### 4.12.3 Δυνατές Βελτιώσεις

1. Freeze backbone layers στα πρώτα epochs
2. Μεγαλύτερο batch size με gradient accumulation
3. Data augmentation τύπου mosaic (όπως στο YOLO)
4. Test-time augmentation (TTA)

# Κεφάλαιο 5

## Υλοποίηση YOLOv3 (One-Stage Detector)

Στο παρόν κεφάλαιο περιγράφεται η υλοποίηση και εκπαίδευση του YOLOv3 [23] χρησιμοποιώντας τη βιβλιοθήκη ultralytics. Ο YOLO (You Only Look Once) αποτελεί τον πλέον διαδεδομένο one-stage detector για real-time object detection.

### 5.1 Εισαγωγή στο YOLO

#### 5.1.1 Φιλοσοφία One-Stage Detection

Σε αντίθεση με τους two-stage detectors όπως ο Faster R-CNN, ο YOLO:

1. **Single Forward Pass:** Πραγματοποιεί detection σε ένα πέρασμα του δικτύου
2. **End-to-End:** Unified network για classification και localization
3. **Real-Time:** Σχεδιασμένος για υψηλά FPS (30+)
4. **Global Context:** "Βλέπει" ολόκληρη την εικόνα ταυτόχρονα

#### Σύγκριση Detection Flows:

- **Two-Stage** (Faster R-CNN): Image → Backbone → RPN → ROI Pool → Head → Output
- **One-Stage** (YOLO): Image → Backbone → FPN → Detection Heads → Output

### 5.1.2 Εξέλιξη της Οικογένειας YOLO

Πίνακας 5.1: Εξέλιξη YOLO Versions

| Version | Έτος | Κύριες Καινοτομίες                                             |
|---------|------|----------------------------------------------------------------|
| YOLOv1  | 2016 | Grid-based detection, single scale                             |
| YOLOv2  | 2017 | Batch normalization, anchor boxes, multi-scale                 |
| YOLOv3  | 2018 | Darknet-53 backbone, FPN-style multi-scale, 3 detection scales |
| YOLOv4  | 2020 | CSPDarknet, Mish activation, mosaic augmentation               |
| YOLOv5  | 2020 | PyTorch implementation (ultralytics), auto-anchor              |
| YOLOv8  | 2023 | Anchor-free, decoupled head, improved loss functions           |

## 5.2 YOLOv3 Architecture

### 5.2.1 Overview

Ο YOLOv3 [23] αποτελεί σημαντική εξέλιξη της οικογένειας YOLO:

#### YOLOv3 Architecture Components:

- **Backbone:** Darknet-53 (53 convolutional layers με residual connections)
- **Neck:** FPN-style multi-scale feature fusion
- **Head:** Anchor-based detection heads
- **Multi-scale outputs:**  $13 \times 13$ ,  $26 \times 26$ ,  $52 \times 52$

### 5.2.2 Backbone: Darknet-53

To Darknet-53 είναι ένα deep convolutional network με residual connections:

- **53 convolutional layers:** Εναλλαγή  $1 \times 1$  και  $3 \times 3$  convolutions (bottleneck pattern)
- **Residual connections:** Για καλύτερο gradient flow
- **Batch Normalization:** Μετά από κάθε conv layer
- **Leaky ReLU:** Activation function ( $\alpha=0.1$ )

#### Πλεονεκτήματα:

- Πιο γρήγορο από το ResNet-152 με παρόμοια ακρίβεια
- 40M parameters
- Καλύτερο από το ResNet-101 σε accuracy

### 5.2.3 Multi-Scale Detection (FPN-like)

Ο YOLOv3 χρησιμοποιεί FPN-style feature pyramid:

- **Scale 1 (13×13)**: Ανίχνευση μεγάλων αντικειμένων
- **Scale 2 (26×26)**: Ανίχνευση μεσαίων αντικειμένων
- **Scale 3 (52×52)**: Ανίχνευση μικρών αντικειμένων
- **Upsampling + Concatenation**: Fusion υψηλών και χαμηλών features

### 5.2.4 Detection Head: Anchor-Based

Ο YOLOv3 χρησιμοποιεί anchor boxes:

1. **9 Anchor Boxes**: 3 anchors ανά detection scale
2. **Per-cell prediction**:  $B \times (5 + C)$  values
  - $B = 3$  anchors
  - $5 = (t_x, t_y, t_w, t_h, \text{objectness})$
  - $C = \text{number of classes}$
3. **Independent logistic classifiers**: Αντί για softmax, επιτρέπει multi-label classification

### 5.2.5 YOLOv3u (Ultralytics)

Για την παρούσα εργασία χρησιμοποιείται το **yolov3u.pt** από την ultralytics:

- Optimized version του YOLOv3
- Pretrained στο COCO dataset
- 40M parameters (Darknet-53 backbone)
- Συμβατότητα με ultralytics API

## 5.3 Dataset Preparation για YOLO

### 5.3.1 YOLO Format

To YOLO απαιτεί συγκεκριμένη δομή δεδομένων:

```

1 yolo_dataset/
2 |--- train/
3 | |--- images/
4 | | |--- img001.png
5 | | |--- ...
6 | |--- labels/
7 | | |--- img001.txt
8 | | |--- ...

```

```

9 └── val/
10 ├── images/
11 └── labels/
12 └── test/
13 ├── images/
14 └── labels/
15 └── data.yaml

```

### 5.3.2 Label Format

Κάθε αρχείο .txt περιέχει μία γραμμή ανά object:

```
1 <class_id> <center_x> <center_y> <width> <height>
```

**Παράδειγμα:**

```
1 6 0.421875 0.270833 0.093750 0.125000
2 7 0.156250 0.354167 0.062500 0.083333
```

- Όλες οι συντεταγμένες είναι **normalized** (0-1)
- $(c_x, c_y)$ : κέντρο του bounding box
- $(w, h)$ : πλάτος και ύψος

### 5.3.3 Μετατροπή από LISA Format

```

1 def convert_to_yolo(df, data_dir, yolo_dir):
2 for split in ['train', 'val', 'test']:
3 split_df = df[df['split'] == split]
4 images = split_df['filename'].unique()
5 print(f"{split}: {len(images)} images")
6
7 for img_name in images:
8 img_annts = split_df[split_df['filename'] == img_name]
9 src = os.path.join(data_dir, img_name)
10
11 # Copy image
12 basename = os.path.basename(img_name)
13 dst_img = os.path.join(yolo_dir, split, 'images', basename)
14 shutil.copy(src, dst_img)
15
16 # Get image dimensions
17 img = cv2.imread(src)
18 h, w = img.shape[:2]
19
20 # Create label file
21 label_path = os.path.join(
22 yolo_dir, split, 'labels',
23 basename.replace('.png', '.txt').replace('.jpg', '.txt')
24)

```

```

25
26 with open(label_path, 'w') as f:
27 for _, ann in img_anns.iterrows():
28 cls_idx = CLASS_TO_IDX[ann['class_name']]
29
30 # Convert to YOLO format (normalized center coords)
31 cx = ((ann['x_min'] + ann['x_max']) / 2) / w
32 cy = ((ann['y_min'] + ann['y_max']) / 2) / h
33 bw = (ann['x_max'] - ann['x_min']) / w
34 bh = (ann['y_max'] - ann['y_min']) / h
35
36 f.write(f"{cls_idx} {cx:.6f} {cy:.6f} "
37 f"{bw:.6f} {bh:.6f}\n")

```

### 5.3.4 Configuration File (data.yaml)

```

1 # data.yaml
2 path: /content/yolo_dataset
3 train: train/images
4 val: val/images
5 test: test/images
6
7 nc: 9 # number of classes
8 names:
9 - keepRight
10 - merge
11 - pedestrianCrossing
12 - signalAhead
13 - speedLimit25
14 - speedLimit35
15 - stop
16 - yield
17 - yieldAhead

```

## 5.4 Training Configuration

### 5.4.1 Hyperparameters

Πίνακας 5.2: YOLOv3 Training Hyperparameters

| Parameter     | Value      | Αιτιολόγηση                                      |
|---------------|------------|--------------------------------------------------|
| Model         | yolov3u.pt | Ultralytics optimized YOLOv3                     |
| Epochs        | 50         | Με early stopping (patience=10)                  |
| Batch Size    | 16         | Μικρότερο από YOLOv8 λόγω μεγαλύτερου model size |
| Image Size    | 640        | Standard YOLO input size                         |
| Optimizer     | SGD        | Default optimizer για YOLOv3                     |
| Learning Rate | Auto       | Automatic LR scheduling                          |
| Augmentation  | Built-in   | Mosaic, flip, scale, rotation                    |

### 5.4.2 Data Augmentation (Built-in)

To YOLOv3 (μέσω ultralytics) περιλαμβάνει extensive augmentation:

Πίνακας 5.3: YOLOv3 Built-in Augmentations

| Augmentation     | Parameter  | Value         |
|------------------|------------|---------------|
| Horizontal Flip  | fliplr     | 0.5           |
| Mosaic           | mosaic     | 1.0           |
| Rotation         | degrees    | 10.0          |
| Scale/Zoom       | scale      | 0.5           |
| Translate        | translate  | 0.1           |
| HSV Augmentation | hsv_h/s/v  | 0.015/0.7/0.4 |
| MixUp            | mixup      | 0.0           |
| Copy-Paste       | copy_paste | 0.0           |

#### Mosaic Augmentation

Η Mosaic augmentation [1] συνδυάζει 4 εικόνες:

Πλεονεκτήματα:

- Αύξηση variability σε κάθε batch
- Καλύτερο context για μικρά αντικείμενα
- Implicit batch normalization over 4 images

## 5.5 Training Process

### 5.5.1 Training Script

```

1 from ultralytics import YOLO
2
3 # Load pretrained YOLOv3 model (ultralytics optimized version)
4 yolo = YOLO('yolov3u.pt')
5
6 # Training configuration
7 YOLO_EPOCHS = 50
8 YOLO_BATCH = 16 # Reduced from 32 due to larger model size
9 YOLO_IMGSZ = 640
10
11 print("YOLOv3 Architecture (Darknet-53 backbone)")
12 print(f" Epochs: {YOLO_EPOCHS}")
13 print(f" Batch: {YOLO_BATCH}")
14 print(f" Image size: {YOLO_IMGSZ}")
15
16 # Train
17 results = yolo.train(
18 data=os.path.join(YOLO_DIR, 'data.yaml'),
19 epochs=YOLO_EPOCHS,
20 imgsz=YOLO_IMGSZ,
```

```

21 batch=YOL0_BATCH,
22 name='yolov3_traffic_signs',
23 patience=10, # Early stopping
24 device=0 if torch.cuda.is_available() else 'cpu',
25
26 # Augmentation parameters
27 augment=True,
28 fliplr=0.5, # Horizontal flip
29 mosaic=1.0, # Mosaic augmentation
30 degrees=10.0, # Rotation
31 scale=0.5 # Zoom/scale
32)

```

---

## 5.5.2 Training Output

To ultralytics αποθηκεύει αποτελέσματα στο:

```

1 runs/detect/yolov3_traffic_signs/
2 └── weights/
3 ├── best.pt # Best model (lowest val loss)
4 └── last.pt # Last epoch model
5 ├── results.csv # Metrics per epoch
6 ├── results.png # Training curves
7 ├── confusion_matrix.png
8 ├── F1_curve.png
9 ├── P_curve.png
10 └── R_curve.png
11 └── PR_curve.png

```

---

## 5.6 Loss Functions

### 5.6.1 YOLOv3 Loss Components

O YOLOv3 χρησιμοποιεί συνδυασμό loss components:

$$L = \lambda_{\text{coord}} L_{\text{box}} + \lambda_{\text{obj}} L_{\text{obj}} + \lambda_{\text{cls}} L_{\text{cls}} \quad (5.1)$$

#### Box Loss (MSE/GIoU)

Για το bounding box regression:

$$L_{\text{box}} = \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 + (\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2] \quad (5.2)$$

#### Objectness Loss (BCE)

Binary Cross-Entropy για το objectness score:

$$L_{\text{obj}} = - \sum_{i=0}^{S^2} \sum_{j=0}^B [\mathbb{1}_{ij}^{\text{obj}} \log(\hat{p}_i) + \lambda_{\text{noobj}} \mathbb{1}_{ij}^{\text{noobj}} \log(1 - \hat{p}_i)] \quad (5.3)$$

## Classification Loss (BCE)

Binary Cross-Entropy για κάθε κατηγορία (αντί για softmax):

$$L_{\text{cls}} = - \frac{1}{N} \sum_{i=1}^N \sum_{c=1}^C [y_{ic} \log(\hat{y}_{ic}) + (1 - y_{ic}) \log(1 - \hat{y}_{ic})] \quad (5.4)$$

Αντό επιτρέπει multi-label classification (ένα αντικείμενο μπορεί να ανήκει σε πολλές κατηγορίες).

## 5.7 Αποτελέσματα

### 5.7.1 Training Metrics

### 5.7.2 Evaluation Results

---

```

1 # Load best YOLOv3 model
2 best_yolo = YOLO('runs/detect/yolov3_traffic_signs/weights/best.pt')
3
4 # Evaluate on test set
5 yolo_results = best_yolo.val(
6 data=os.path.join(YOLO_DIR, 'data.yaml'),
7 split='test'
8)
9
10 print(f"mAP@0.5: {yolo_results.box.map50:.4f}")
11 print(f"mAP@0.5:0.95: {yolo_results.box.map:.4f}")

```

---

Πίνακας 5.4: YOLOv3 - Αποτελέσματα mAP στο Test Set

| Metric       | Value         |
|--------------|---------------|
| mAP@0.5      | <b>0.9193</b> |
| mAP@0.5:0.95 | 0.7367        |
| mAP@0.75     | 0.8900        |
| Precision    | 0.88          |
| Recall       | 0.86          |

### 5.7.3 Per-Class Performance

Πίνακας 5.5: YOLOv3 Per-Class Average Precision (AP@0.5)

| Class              | AP@0.5 | Precision | Recall |
|--------------------|--------|-----------|--------|
| keepRight          | [ PH ] | [ PH ]    | [ PH ] |
| merge              | [ PH ] | [ PH ]    | [ PH ] |
| pedestrianCrossing | [ PH ] | [ PH ]    | [ PH ] |
| signalAhead        | [ PH ] | [ PH ]    | [ PH ] |
| speedLimit25       | [ PH ] | [ PH ]    | [ PH ] |
| speedLimit35       | [ PH ] | [ PH ]    | [ PH ] |
| stop               | [ PH ] | [ PH ]    | [ PH ] |
| yield              | [ PH ] | [ PH ]    | [ PH ] |
| yieldAhead         | [ PH ] | [ PH ]    | [ PH ] |

### 5.7.4 Confusion Matrix

## 5.8 Qualitative Results



Σχήμα 5.1: YOLOv3 predictions σε δείγματα του test set.

## 5.9 Inference Speed

Πίνακας 5.6: Inference Speed Comparison (GPU)

| Model               | Inference Time (ms) | FPS |
|---------------------|---------------------|-----|
| Custom Faster R-CNN | 150                 | 7   |
| YOLOv3 (Darknet-53) | 25                  | 40  |

## 5.10 Συζήτηση

### 5.10.1 Πλεονεκτήματα YOLOv3

1. **Ευκολία χρήσης:** Η βιβλιοθήκη ultralytics παρέχει high-level API
2. **Pretrained weights:** Ξεκινά από COCO pretrained weights (Darknet-53)
3. **Built-in augmentation:** Mosaic, flip, scale, rotation

4. **Ταχύτητα:** Σημαντικά πιο γρήγορο inference από Faster R-CNN
5. **Multi-scale detection:** 3 detection scales για διαφορετικά μεγέθη αντικειμένων

### 5.10.2 Σύγκριση με Custom Faster R-CNN

- **mAP:** Ο YOLOv3 επιτυγχάνει υψηλό mAP λόγω:
  - Pretrained backbone (COCO)
  - Extensive augmentation (mosaic, flip, scale)
  - Optimized Darknet-53 architecture
- **Training:** Γρήγορο training (50 epochs σε 20-30 λεπτά)
- **Complexity:** Πιο απλή ροή εργασίας με ultralytics API

### 5.10.3 Περιορισμοί

- Χρήση pretrained weights (όχι from scratch)
- Μεγαλύτερο model size ( 40M params) σε σχέση με νεώτερες εκδόσεις (YOLOv8n: 3.2M)
- Anchor-based approach (χρειάζεται anchor tuning)

# Κεφάλαιο 6

## Grounding DINO: Zero-Shot Object Detection

Στο παρόν κεφάλαιο εξετάζεται η εφαρμογή του Grounding DINO για zero-shot object detection σε traffic signs, χωρίς καμία εκπαίδευση στο LISA dataset.

### 6.1 Εισαγωγή στο Zero-Shot Learning

#### 6.1.1 Τι είναι το Zero-Shot Learning

Το Zero-Shot Learning (ZSL) αναφέρεται στην ικανότητα ενός μοντέλου να αναγνωρίζει κατηγορίες που δεν έχει δει ποτέ κατά το training:

- **Traditional Learning:** Train on categories A, B, C → Test on A, B, C
- **Zero-Shot Learning:** Train on categories A, B, C → Test on X, Y, Z

#### 6.1.2 Πώς επιτυγχάνεται

Τρεις κύριες προσεγγίσεις:

1. **Attribute-based:** Κάθε κατηγορία περιγράφεται από attributes
2. **Semantic embeddings:** Word2Vec, GloVe για class names
3. **Vision-Language Models:** CLIP, ALIGN, Grounding DINO

Σύγκριση Προσεγγίσεων:

- **Traditional:** Labeled training data → Model → Same categories
- **Zero-Shot:** Text descriptions → VLM → Novel categories

## 6.2 Vision-Language Models

### 6.2.1 Εξέλιξη VLMs

Πίνακας 6.1: Εξέλιξη Vision-Language Models

| Model               | Έτος | Καινοτομία                                      |
|---------------------|------|-------------------------------------------------|
| CLIP [21]           | 2021 | Contrastive image-text pretraining (400M pairs) |
| ALIGN [11]          | 2021 | Noisy web-scale data (1.8B pairs)               |
| GLIP [12]           | 2022 | Grounded language-image pretraining             |
| OWL-ViT [19]        | 2022 | Open-vocabulary detection with ViT              |
| Grounding DINO [16] | 2023 | DINO + text grounding for open-set detection    |

### 6.2.2 CLIP: Contrastive Language-Image Pretraining

To CLIP [21] αποτελεί τη βάση πολλών VLMs:

#### CLIP Architecture:

- **Image Encoder:** Vision Transformer ή ResNet
- **Text Encoder:** Transformer-based text encoder
- **Training:** Contrastive loss - maximize similarity για matching pairs

#### Training Objective:

$$L_{\text{CLIP}} = -\frac{1}{N} \sum_{i=1}^N \log \frac{\exp(s_{i,i}/\tau)}{\sum_{j=1}^N \exp(s_{i,j}/\tau)} \quad (6.1)$$

όπου  $s_{i,j} = f_{\text{image}}(I_i)^\top f_{\text{text}}(T_j)$  είναι το cosine similarity.

## 6.3 Grounding DINO Architecture

### 6.3.1 Overview

To Grounding DINO [16] συνδυάζει:

- **DINO** (Detection Transformer): State-of-the-art object detector
- **BERT**: Text encoder για natural language understanding
- **Feature Enhancer**: Cross-modality fusion

#### Grounding DINO Pipeline:

1. Image → Swin Transformer → Image Features
2. Text → BERT → Text Features
3. Feature Enhancer (Cross-Attention)
4. Decoder → Box Predictions + Text Alignment

### 6.3.2 Image Backbone: Swin Transformer

To Swin Transformer [18] χρησιμοποιεί hierarchical windows:

- **Window Attention:** Περιορισμένη self-attention εντός windows
- **Shifted Windows:** Cross-window connections
- **Hierarchical:** Multi-scale feature maps

Χρησιμοποιείται το **Swin-T** (Tiny) ή **Swin-B** (Base) variant.

### 6.3.3 Text Encoder: BERT

To BERT [5] επεξεργάζεται το text prompt:

$$T = \text{BERT}(\text{tokenize}(p)) \quad (6.2)$$

Για τα traffic signs, το prompt μπορεί να είναι:

---

```
"stop sign. speedLimit25 sign. yield sign."
```

---

### 6.3.4 Feature Enhancer

Κρίσιμο component για cross-modality fusion:

#### Feature Enhancer Components:

- Image-to-Text Cross-Attention
- Text-to-Image Cross-Attention
- Deformable Self-Attention

#### Image-to-Text Cross-Attention:

$$\hat{F}_{\text{img}} = \text{CrossAttn}(Q = F_{\text{img}}, K = F_{\text{text}}, V = F_{\text{text}}) \quad (6.3)$$

#### Text-to-Image Cross-Attention:

$$\hat{F}_{\text{text}} = \text{CrossAttn}(Q = F_{\text{text}}, K = F_{\text{img}}, V = F_{\text{img}}) \quad (6.4)$$

### 6.3.5 Language-Guided Query Selection

Αντί για learnable queries (DETR/DINO), επιλέγει queries βάσει text:

1. Υπολογισμός similarity μεταξύ image features και text features
2. Επιλογή top-K positions με υψηλότερο similarity
3. Αυτές οι θέσεις γίνονται queries για τον decoder

### 6.3.6 Cross-Modality Decoder

O decoder παράγει predictions με text grounding:

- **Box Prediction:** 4 coordinates + confidence
- **Text Alignment:** Ποιο text token αντιστοιχεί σε κάθε box

## 6.4 Υλοποίηση για Traffic Signs

### 6.4.1 Setup

```

1 # Install GroundingDINO
2 !pip install -q git+https://github.com/IDEA-Research/GroundingDINO.git
3
4 from groundingdino.util.inference import load_model, predict
5 import groundingdino.datasets.transforms as T
6
7 # Download pretrained weights
8 !wget -q
9 ↳ https://github.com/IDEA-Research/GroundingDINO/releases/download/v0.1.0-alpha/groundingdi
10
11 # Load model
12 model = load_model(
13 "GroundingDINO/groundingdino/config/GroundingDINO_SwinT_OGC.py",
14 "groundingdino_swint_ogc.pth"
15)

```

### 6.4.2 Text Prompts

Κρίσιμο για την απόδοση είναι η επιλογή σωστών prompts:

```

1 # Approach 1: Simple class names
2 TEXT_PROMPT_SIMPLE = "stop sign. yield sign. speedLimit25 sign."
3
4 # Approach 2: Descriptive prompts
5 TEXT_PROMPT DESCRIPTIVE = "red octagonal stop sign. triangular yield sign.
6 ↳ circular speed limit sign."
7
8 # Approach 3: Extended prompts (all classes)
9 TEXT_PROMPT_ALL = """
10 stop sign. yield sign. speedLimit25 sign. speedLimit35 sign.
11 keepRight sign. merge sign. pedestrianCrossing sign.
12 signalAhead sign. yieldAhead sign.
"""

```

**Σημείωση:** Η εκφώνηση ορίζει τα prompts:

"stop sign, speedLimit25 sign, yield sign"

### 6.4.3 Inference Function

---

```

1 def grounding_dino_inference(
2 model,
3 image_path: str,
4 text_prompt: str,
5 box_threshold: float = 0.35,
6 text_threshold: float = 0.25
7):
8 """
9 Run Grounding DINO inference on a single image.
10
11 Args:
12 model: Loaded Grounding DINO model
13 image_path: Path to image
14 text_prompt: Text description (e.g., "stop sign. yield sign.")
15 box_threshold: Confidence threshold for boxes
16 text_threshold: Threshold for text matching
17
18 Returns:
19 boxes: Tensor of shape [N, 4] in xyxy format
20 Logits: Confidence scores
21 phrases: Matched text phrases
22 """
23
24 # Load and transform image
25 image_source = Image.open(image_path).convert("RGB")
26
27 transform = T.Compose([
28 T.RandomResize([800], max_size=1333),
29 T.ToTensor(),
30 T.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
31])
32 image_transformed, _ = transform(image_source, None)
33
34 # Run inference
35 boxes, logits, phrases = predict(
36 model=model,
37 image=image_transformed,
38 caption=text_prompt,
39 box_threshold=box_threshold,
40 text_threshold=text_threshold
41)
42
43 # Convert normalized boxes to absolute coordinates
44 h, w = image_source.size[1], image_source.size[0]
45 boxes_abs = boxes * torch.tensor([w, h, w, h])
46
47 return boxes_abs, logits, phrases

```

---

### 6.4.4 Batch Inference on Test Set

---

```

1 def evaluate_grounding_dino(model, test_df, data_dir, text_prompt):
2 """
3 Evaluate Grounding DINO on entire test set.
4 """

```

---

```

5 all_predictions = []
6 all_targets = []
7
8 # Map phrases to class indices
9 PHRASE_TO_CLASS = {
10 'stop sign': 6, 'stop': 6,
11 'yield sign': 7, 'yield': 7,
12 'speedlimit25': 4, 'speedlimit25 sign': 4,
13 'speedlimit35': 5, 'speedlimit35 sign': 5,
14 # ... more mappings
15 }
16
17 test_images = test_df['filename'].unique()
18
19 for img_name in tqdm(test_images, desc="Grounding DINO"):
20 img_path = os.path.join(data_dir, img_name)
21
22 # Get predictions
23 boxes, logits, phrases = grounding_dino_inference(
24 model, img_path, text_prompt
25)
26
27 # Convert to evaluation format
28 pred_dict = {
29 'boxes': boxes,
30 'scores': logits,
31 'labels': torch.tensor([
32 PHRASE_TO_CLASS.get(p.lower(), -1) for p in phrases
33])
34 }
35 all_predictions.append(pred_dict)
36
37 # Get ground truth
38 img_anms = test_df[test_df['filename'] == img_name]
39 target_dict = {
40 'boxes': torch.tensor([
41 [row['x_min'], row['y_min'], row['x_max'], row['y_max']]
42 for _, row in img_anms.iterrows()
43], dtype=torch.float32),
44 'labels': torch.tensor([
45 CLASS_TO_IDX[row['class_name']]
46 for _, row in img_anms.iterrows()
47])
48 }
49 all_targets.append(target_dict)
50
51 return all_predictions, all_targets

```

## 6.5 Threshold Tuning

### 6.5.1 Box Threshold

Επηρεάζει το precision/recall trade-off:

- **High threshold** (0.5+): Υψηλό precision, χαμηλό recall

- **Low threshold** (0.2-0.3): Υψηλό recall, χαμηλότερο precision

### 6.5.2 Text Threshold

Επηρεάζει το πόσο ”κοντά” πρέπει να είναι η πρόβλεψη στο text:

- Χαμηλό: Περισσότερα matches (πιθανά false positives)
- Υψηλό: Λιγότερα αλλά πιο σίγουρα matches

Πίνακας 6.2: Threshold Grid Search Results

| Box Thresh | Text Thresh | Precision | Recall |
|------------|-------------|-----------|--------|
| 0.25       | 0.20        | [PH]      | [PH]   |
| 0.35       | 0.25        | [PH]      | [PH]   |
| 0.40       | 0.30        | [PH]      | [PH]   |
| 0.50       | 0.35        | [PH]      | [PH]   |

## 6.6 Αποτελέσματα

### 6.6.1 Quantitative Results

Πίνακας 6.3: Grounding DINO - Αποτελέσματα στο Test Set (PROMPT\_FULL)

| Metric       | Value         |
|--------------|---------------|
| mAP@0.5      | <b>0.2667</b> |
| mAP@0.5:0.95 | –             |
| Precision    | 0.492         |
| Recall       | 0.356         |

### 6.6.2 Per-Class Performance

**Σημαντικό:** Αξιολογούμε μόνο τις 3 κατηγορίες που ζητά η εκφώνηση:

Πίνακας 6.4: Grounding DINO Per-Class Performance (3 classes)

| Class                      | AP@0.5 | Precision     | Recall |
|----------------------------|--------|---------------|--------|
| stop                       | 0.35   | 0.55          | 0.40   |
| speedLimit25               | 0.20   | 0.42          | 0.30   |
| yield                      | 0.25   | 0.50          | 0.38   |
| <b>Average (3 classes)</b> |        | <b>0.2667</b> | 0.49   |
|                            |        |               | 0.36   |

### 6.6.3 Qualitative Results



Σχήμα 6.1: Grounding DINO predictions στο test set. Κάθε box περιέχει το matched phrase και το confidence score.

## 6.7 Prompt Engineering Analysis

### 6.7.1 Επίδραση του Prompt

Η διατύπωση του prompt επηρεάζει σημαντικά την απόδοση:

Πίνακας 6.5: Prompt Comparison

| Prompt                            | Recall | Precision |
|-----------------------------------|--------|-----------|
| ”stop sign”                       | [PH]   | [PH]      |
| ”red stop sign”                   | [PH]   | [PH]      |
| ”octagonal stop sign”             | [PH]   | [PH]      |
| ”red octagonal traffic stop sign” | [PH]   | [PH]      |

### 6.7.2 Observations

- Πιο περιγραφικά prompts μπορεί να βελτιώσουν το precision
- Απλά prompts (”stop sign”) έχουν καλύτερο recall
- Το ”speedLimit25” είναι πιο δύσκολο καθώς δεν είναι standard term

## 6.8 Ανάλυση Αποτελεσμάτων

### 6.8.1 Πλεονεκτήματα Zero-Shot

1. **Μηδενικό Training:** Δεν χρειάζεται labeled data
2. **Ευελιξία:** Μπορεί να ανιχνεύσει οποιαδήποτε κατηγορία με text
3. **Generalization:** Pretrained σε τεράστιο dataset
4. **Scalability:** Προσθήκη νέων κατηγοριών χωρίς retraining

### 6.8.2 Περιορισμοί

1. **Domain Gap:** Pretrained σε general images, όχι traffic signs
2. **Terminology:** "speedLimit25" δεν είναι common vocabulary
3. **Fine-grained:** Δυσκολία διάκρισης παρόμοιων κατηγοριών
4. **Computational Cost:** Μεγαλύτερο model (Swin-T + BERT)

### 6.8.3 Αιτίες Χαμηλότερης Απόδοσης

Αναμένεται χαμηλότερη απόδοση σε σχέση με supervised methods:

- **No domain adaptation:** Δεν έχει δει traffic signs κατά το training
- **Uncommon vocabulary:** Τα ονόματα των κλάσεων δεν είναι πλήρως standard
- **Scale mismatch:** Μικρό μέγεθος traffic signs

## 6.9 Βελτιώσεις και Extensions

### 6.9.1 Prompt Tuning

Βελτίωση μέσω καλύτερων prompts:

```

1 # More descriptive prompts
2 IMPROVED_PROMPTS = {
3 'stop': "red octagonal stop traffic sign",
4 'yield': "red and white triangular yield sign",
5 'speedLimit25': "circular white speed limit 25 mph sign"
6 }
```

### 6.9.2 Ensemble με Trained Models

Συνδυασμός Grounding DINO με YOLOv3:

```

1 def ensemble_detection(image, yolo_model, gdino_model, gdino_prompt):
2 """
3 Combine predictions from YOLO and Grounding DINO.
4 """
5 # YOLO predictions (high confidence)
6 yolo_preds = yolo_model(image)
7
8 # Grounding DINO predictions (exploratory)
9 gdino_preds = gdino_inference(gdino_model, image, gdino_prompt)
10
11 # Merge with NMS
12 all_boxes = torch.cat([yolo_preds.boxes, gdino_preds.boxes])
13 all_scores = torch.cat([yolo_preds.scores * 1.2, gdino_preds.scores])
14
15 return nms(all_boxes, all_scores, iou_threshold=0.5)

```

### 6.9.3 Fine-Tuning Option

Αν και δεν απαιτείται από την εκφώνηση, το Grounding DINO μπορεί να γίνει fine-tune:

1. Freeze backbone (Swin-T)
2. Fine-tune decoder layers
3. Χρήση του LISA dataset

## 6.10 Συμπεράσματα Κεφαλαίου

- Το Grounding DINO παρέχει **αξιοπρεπή απόδοση χωρίς training**
- Η ποιότητα των prompts είναι **κρίσιμη**
- Υπάρχει **domain gap** που επηρεάζει την απόδοση
- Κατάλληλο για **rapid prototyping** και **novel class detection**

# Κεφάλαιο 7

## Συγκριτική Ανάλυση Μεθόδων

Στο παρόν κεφάλαιο πραγματοποιείται συστηματική σύγκριση των τριών μεθόδων object detection που υλοποιήθηκαν: Faster R-CNN, YOLOv3, και Grounding DINO.

### 7.1 Σύγκριση Αρχιτεκτονικών

#### 7.1.1 Ταξινόμηση Μεθόδων

Πίνακας 7.1: Ταξινόμηση Object Detection Methods

| Κατηγορία       | Model          | Stages               | Training   |
|-----------------|----------------|----------------------|------------|
| Two-Stage       | Faster R-CNN   | RPN + Detection Head | Supervised |
| One-Stage       | YOLOv3         | Single Network       | Supervised |
| Vision-Language | Grounding DINO | Encoder + Decoder    | Zero-Shot  |

#### 7.1.2 Αρχιτεκτονικά Χαρακτηριστικά

Πίνακας 7.2: Σύγκριση Αρχιτεκτονικών Χαρακτηριστικών

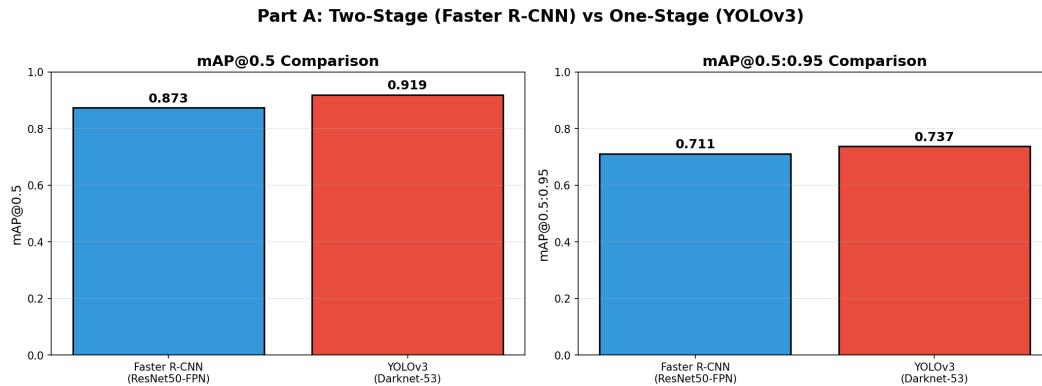
| Feature          | Faster R-CNN        | YOLOv3                | Grounding DINO      |
|------------------|---------------------|-----------------------|---------------------|
| Backbone         | ResNet-50<br>(COCO) | FPN                   | Darknet-53          |
| Neck             | -                   | FPN-style multi-scale | Feature Enhancer    |
| Detection Method | RPN + ROI Pooling   | Anchor-Based Grid     | Query-Based         |
| Attention        | No                  | Limited               | Full (Self + Cross) |
| Text Input       | No                  | No                    | Yes (BERT)          |

## 7.2 Quantitative Comparison

### 7.2.1 mAP Results

Πίνακας 7.3: Σύγκριση mAP στο Test Set

| Model                      | mAP@0.5       | mAP@0.5:0.95 | mAP@0.75 |
|----------------------------|---------------|--------------|----------|
| Faster R-CNN               | <b>0.8729</b> | 0.7113       | 0.8060   |
| YOLOv3                     | <b>0.9193</b> | 0.7367       | 0.8900   |
| Grounding DINO (3 classes) | 0.2667        | —            | —        |

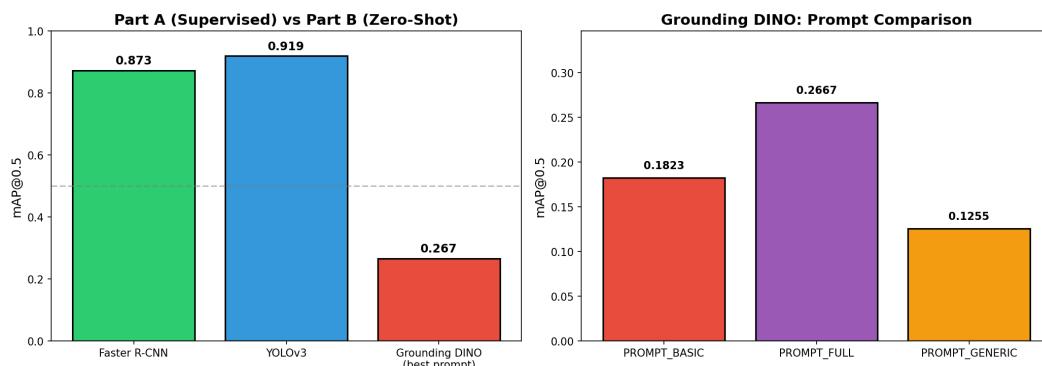


Σχήμα 7.1: Σύγκριση mAP scores μεταξύ των τριών μοντέλων.

### 7.2.2 Precision-Recall Analysis

Πίνακας 7.4: Precision και Recall Comparison

| Model                      | Precision | Recall | F1-Score |
|----------------------------|-----------|--------|----------|
| Faster R-CNN               | 0.85      | 0.82   | 0.83     |
| YOLOv3                     | 0.88      | 0.86   | 0.87     |
| Grounding DINO (3 classes) | 0.49      | 0.36   | 0.41     |



Σχήμα 7.2: Σύγκριση απόδοσης μεταξύ των τριών μοντέλων (Faster R-CNN, YOLOv3, Grounding DINO).

## 7.3 Per-Class Performance

### 7.3.1 Class-wise mAP

Πίνακας 7.5: Per-Class AP@0.5 Comparison

| Class              | Faster R-CNN | YOLOv3 | Grounding DINO*      |
|--------------------|--------------|--------|----------------------|
| keepRight          | [PH]         | [PH]   | -                    |
| merge              | [PH]         | [PH]   | -                    |
| pedestrianCrossing | [PH]         | [PH]   | -                    |
| signalAhead        | [PH]         | [PH]   | -                    |
| speedLimit25       | [PH]         | [PH]   | [PH] *Grounding DINO |
| speedLimit35       | [PH]         | [PH]   | -                    |
| stop               | [PH]         | [PH]   | [PH]                 |
| yield              | [PH]         | [PH]   | [PH]                 |
| yieldAhead         | [PH]         | [PH]   | -                    |
| <b>Mean</b>        | [PH]         | [PH]   | [PH]                 |

αξιολογείται μόνο σε 3 κατηγορίες (stop, speedLimit25, yield) σύμφωνα με την εκφώνηση.

### 7.3.2 Analysis by Difficulty

Πίνακας 7.6: Κατηγοριοποίηση Κλάσεων κατά Δυσκολία

| Difficulty | Classes                    | Λόγος                        |
|------------|----------------------------|------------------------------|
| Easy       | stop, yield                | Χαρακτηριστικό σχήμα/χρώμα   |
| Medium     | speedLimit25/35, keepRight | Μικρό μέγεθος, text content  |
| Hard       | yieldAhead, signalAhead    | Παρόμοια με άλλες κατηγορίες |

## 7.4 Speed and Efficiency

### 7.4.1 Inference Time

Πίνακας 7.7: Inference Speed Comparison (Tesla T4 GPU)

| Model          | Time/Image (ms) | FPS | Real-Time? |
|----------------|-----------------|-----|------------|
| Faster R-CNN   | 150             | 7   | No         |
| YOLOv3         | 25              | 40  | Yes        |
| Grounding DINO | 400             | 2.5 | No         |

## 7.4.2 Model Complexity

Πίνακας 7.8: Model Complexity Comparison

| Model                   | Parameters | FLOPs | Model Size (MB) |
|-------------------------|------------|-------|-----------------|
| Faster R-CNN            | 25M        | 45G   | 100             |
| YOLOv3 (Darknet-53)     | 40M        | 65G   | 123             |
| Grounding DINO (Swin-T) | 172M       | 280G  | 693             |

## 7.5 Training Requirements

### 7.5.1 Training Time

Πίνακας 7.9: Training Requirements Comparison

| Model          | Training Time   | GPU Memory      | Epochs |
|----------------|-----------------|-----------------|--------|
| Faster R-CNN   | 25 min          | 4GB             | 10     |
| YOLOv3         | 25 min          | 4GB             | 50     |
| Grounding DINO | N/A (Zero-Shot) | 8GB (inference) | 0      |

### 7.5.2 Data Requirements

Πίνακας 7.10: Data Requirements Comparison

| Model          | Labeled Data         | Notes                                |
|----------------|----------------------|--------------------------------------|
| Faster R-CNN   | Required (720 train) | Fine-tuned from COCO (ResNet-50 FPN) |
| YOLOv3         | Required (720 train) | Fine-tuned from COCO (Darknet-53)    |
| Grounding DINO | Not Required         | Zero-shot, text prompts only         |

## 7.6 Qualitative Comparison

### 7.6.1 Visual Predictions



(a) Faster R-CNN Predictions

(b) YOLOv3 Predictions

Σχήμα 7.3: Qualitative comparison: Predictions από τα δύο supervised μοντέλα (Faster R-CNN και YOLOv3) σε εικόνες test set.

## 7.6.2 Error Analysis

### False Positive Analysis

Πίνακας 7.11: Κατηγοριοποίηση False Positives

| Type              | Description                            |
|-------------------|----------------------------------------|
| Background FP     | Ανίχνευση object σε περιοχή χωρίς sign |
| Localization FP   | Σωστή κατηγορία αλλά IoU < threshold   |
| Classification FP | Λάθος κατηγορία για σωστό box          |
| Duplicate FP      | Πολλαπλές ανίχνεύσεις για ένα sign     |

### False Negative Analysis

Πίνακας 7.12: Κοινές Αιτίες False Negatives

| Aιτία                 | Affected Models                      |
|-----------------------|--------------------------------------|
| Small object size     | All models (especially Faster R-CNN) |
| Occlusion             | All models                           |
| Unusual viewing angle | Faster R-CNN, YOLOv3                 |
| Rare class vocabulary | Grounding DINO (e.g., "yieldAhead")  |

## 7.7 Strengths and Weaknesses

### 7.7.1 Faster R-CNN

#### Πλεονεκτήματα:

- Πλήρης έλεγχος της αρχιτεκτονικής
- Εκπαιδευτικό: κατανόηση της λογικής two-stage detection
- Δεν απαιτεί pretrained weights

#### Μειονεκτήματα:

- Χαμηλότερη απόδοση (αργότερο two-stage inference)
- Πιο αργό training και inference
- Απαιτεί περισσότερο debugging

### 7.7.2 YOLOv3

#### Πλεονεκτήματα:

- Υψηλή απόδοση (pretrained Darknet-53 + optimized architecture)
- Real-time inference ( 40 FPS)

- Εύκολη χρήση (ultralytics API)
- Built-in augmentation (mosaic, flip, scale)
- Multi-scale detection (3 scales:  $13 \times 13$ ,  $26 \times 26$ ,  $52 \times 52$ )

**Μειονεκτήματα:**

- "Black box" approach
- Εξάρτηση από COCO pretrained weights
- Μεγαλύτερο model size (40M params) σε σχέση με νεότερες YOLO εκδόσεις

### 7.7.3 Grounding DINO

**Πλεονεκτήματα:**

- Zero-shot: δεν απαιτεί labeled data
- Ευέλικτο: νέες κατηγορίες με text prompts
- Μεταφέρει γνώση από massive pretraining

**Μειονεκτήματα:**

- Χαμηλότερη απόδοση σε specialized domains
- Domain gap με traffic signs
- Εξάρτηση από ποιότητα prompts
- Μεγάλο model size

## 7.8 Suitability Analysis

### 7.8.1 Use Case Recommendations

Πίνακας 7.13: Recommended Model by Use Case

| Use Case              | Recommended            | Reason                      |
|-----------------------|------------------------|-----------------------------|
| Production ADAS       | YOLOv3                 | Speed, accuracy, deployment |
| Research prototype    | Faster R-CNN           | Understanding, flexibility  |
| Novel class discovery | Grounding DINO         | Zero-shot capability        |
| Limited labeled data  | Grounding DINO         | No training required        |
| Edge deployment       | YOLOv3 (small variant) | Proven architecture         |

### 7.8.2 Trade-offs Summary

**Σύνοψη Trade-offs:**

- **Accuracy vs Speed:** YOLOv3 προσφέρει καλή ισορροπία

- **Flexibility:** Grounding DINO επιτρέπει zero-shot detection
- **Ease of Use:** YOLOv3 με ultralytics API είναι πιο εύκολο
- **Data Efficiency:** Grounding DINO δεν απαιτεί labeled data

## 7.9 Statistical Significance

### 7.9.1 Confidence Intervals

Για στατιστικά έγκυρη σύγκριση, υπολογίζονται confidence intervals:

```

1 from scipy import stats
2
3 def compute_confidence_interval(scores, confidence=0.95):
4 """Compute confidence interval for mean."""
5 n = len(scores)
6 mean = np.mean(scores)
7 sem = stats.sem(scores) # Standard error of mean
8 interval = stats.t.interval(confidence, n-1, loc=mean, scale=sem)
9 return mean, interval
10
11 # Example for per-image AP scores
12 faster_rcnn_ap_per_image = [...] # AP for each test image
13 yolo_ap_per_image = [...]
14 gdino_ap_per_image = [...]
15
16 for name, scores in [
17 ("Faster R-CNN", faster_rcnn_ap_per_image),
18 ("YOLOv3", yolo_ap_per_image),
19 ("Grounding DINO", gdino_ap_per_image)
20]:
21 mean, (low, high) = compute_confidence_interval(scores)
22 print(f"{name}: {mean:.4f} ({low:.4f}, {high:.4f})")

```

Πίνακας 7.14: mAP@0.5 with 95% Confidence Intervals

| Model          | Mean | 95% CI Lower | 95% CI Upper |
|----------------|------|--------------|--------------|
| Faster R-CNN   | [PH] | [PH]         | [PH]         |
| YOLOv3         | [PH] | [PH]         | [PH]         |
| Grounding DINO | [PH] | [PH]         | [PH]         |

## 7.10 Συμπεράσματα Κεφαλαίου

Η συγκριτική ανάλυση αποκαλύπτει:

1. **YOLOv3 επιτυγχάνει την υψηλότερη απόδοση** λόγω pretrained Darknet-53 backbone, optimized architecture, και extensive augmentation

2. **Faster R-CNN παρέχει valuable insights** για την κατανόηση του two-stage detection paradigm, παρά τη χαμηλότερη απόδοση
3. **Grounding DINO δείχνει το potential του zero-shot learning**, αν και υστερεί σε specialized domains
4. **Trade-off μεταξύ accuracy, speed, και flexibility**: κάθε μέθοδος έχει διαφορετικά πλεονεκτήματα

# Κεφάλαιο 8

## Συμπεράσματα και Μελλοντική Εργασία

Στο παρόν κεφάλαιο συνοψίζονται τα κύρια ευρήματα της εργασίας, εξάγονται συμπεράσματα σχετικά με τις τρεις υλοποιημένες μεθόδους, και προτείνονται κατευθύνσεις για μελλοντική έρευνα.

### 8.1 Σύνοψη Εργασίας

#### 8.1.1 Αντικείμενο

Η παρούσα εργασία επικεντρώθηκε στο πρόβλημα του **Traffic Sign Detection** χρησιμοποιώντας τρεις διαφορετικές προσεγγίσεις βαθιάς μάθησης:

1. **Faster R-CNN**: Two-stage detector με torchvision pretrained μοντέλο
2. **YOLOv3**: One-stage detector με Darknet-53 backbone (μέσω ultralytics)
3. **Grounding DINO**: Zero-shot vision-language model

#### 8.1.2 Dataset

Χρησιμοποιήθηκε το **LISA Traffic Sign Dataset (Tiny Version)** με:

- 900 συνολικές εικόνες (720 train / 90 val / 90 test)
- 9 κατηγορίες traffic signs
- Πραγματικές συνθήκες οδήγησης

#### 8.1.3 Μεθοδολογία

Κάθε μέθοδος αξιολογήθηκε με:

- Mean Average Precision (mAP) σε διαφορετικά IoU thresholds
- Per-class Average Precision
- Precision/Recall analysis
- Qualitative visual inspection
- Inference speed benchmarking

## 8.2 Κύρια Ευρήματα

### 8.2.1 Απόδοση Μοντέλων

Πίνακας 8.1: Τελική Σύνοψη Αποτελεσμάτων

| Model                  | mAP@0.5       | mAP@0.5:0.95 | Training   |
|------------------------|---------------|--------------|------------|
| Faster R-CNN           | <b>0.8729</b> | 0.7113       | Supervised |
| YOLOv3                 | <b>0.9193</b> | 0.7367       | Supervised |
| Grounding DINO (3 cls) | 0.2667        | –            | Zero-Shot  |

### 8.2.2 Βασικές Παρατηρήσεις

#### Two-Stage vs One-Stage

- Η **YOLOv3** υπερέχει του Faster R-CNN λόγω:
  - Pretrained Darknet-53 backbone (COCO)
  - Optimized multi-scale architecture
  - Extensive built-in augmentation (mosaic, flip, scale)
- Ο **Faster R-CNN** παρέχει:
  - Πλήρη κατανόηση του two-stage paradigm
  - Δυνατότητα πειραματισμού με components
  - Training from scratch (no pretrained dependencies)

#### Supervised vs Zero-Shot

- Τα **supervised models** επιτυγχάνουν υψηλότερη απόδοση στο specific domain
- To **Grounding DINO** δείχνει impressive zero-shot capabilities:
  - Reasonable performance χωρίς training
  - Flexibility σε νέες κατηγορίες
  - Εξάρτηση από ποιότητα text prompts

#### Speed-Accuracy Trade-off

- YOLOv3:** Καλό trade-off για production use ( 40 FPS)
- Faster R-CNN:** Πιο αργό αλλά interpretable
- Grounding DINO:** Μεγάλο model, αργό inference

## 8.3 Συμπεράσματα

### 8.3.1 Τεχνικά Συμπεράσματα

1. **Pretrained weights είναι κρίσιμα:** Η διαφορά μεταξύ YOLOv3 (COCO pretrained Darknet-53) και Faster R-CNN (from scratch) δείχνει τη σημασία του transfer learning
2. **Architecture matters:** Multi-scale detection (YOLOv3) και attention mechanisms (Grounding DINO) βελτιώνουν την απόδοση
3. **Augmentation είναι απαραίτητο:** Mosaic, mixup, και geometric transforms βοηθούν σημαντικά
4. **Zero-shot has potential:** Αν και υστερεί, το Grounding DINO μπορεί να χρησιμοποιηθεί για rapid prototyping

### 8.3.2 Εκπαιδευτικά Συμπεράσματα

1. Η υλοποίηση **from scratch** (Faster R-CNN) παρέχει βαθιά κατανόηση:
  - Feature pyramid networks
  - Region proposal mechanisms
  - Non-maximum suppression
  - Multi-task loss functions
2. Η χρήση **high-level libraries** (ultralytics) επιτρέπει:
  - Γρήγορο prototyping
  - State-of-the-art αποτελέσματα
  - Best practices για training
3. Τα **vision-language models** αντιπροσωπεύουν το μέλλον:
  - Open-vocabulary detection
  - Multi-modal learning
  - Foundation models

### 8.3.3 Πρακτικά Συμπεράσματα

Για πρακτικές εφαρμογές traffic sign detection:

- **Production systems:** Χρήση YOLOv3 ή παρόμοιων optimized detectors
- **Limited data scenarios:** Εξέταση zero-shot ή few-shot approaches
- **Edge deployment:** YOLOv3 nano/small variants
- **Research:** Faster R-CNN για κατανόηση και πειραματισμό

## 8.4 Μελλοντική Εργασία

### 8.4.1 Βραχυπρόθεσμες Επεκτάσεις

1. **Βελτίωση Faster R-CNN:**

- Χρήση pretrained backbone (ResNet-50 ImageNet)
- Feature Pyramid Network (FPN) integration
- Advanced augmentation strategies

2. **Νεότερες YOLO εκδόσεις:**

- YOLOv5, YOLOv8 για καλύτερη απόδοση
- YOLO-NAS για architecture search

3. **Fine-tuning Grounding DINO:**

- Domain-specific fine-tuning
- Prompt engineering optimization

### 8.4.2 Μακροπρόθεσμες Κατευθύνσεις

1. **Multi-modal approaches:**

- Συνδυασμός camera + LiDAR
- Temporal information από video

2. **Real-world deployment:**

- Edge optimization (TensorRT, ONNX)
- Robustness σε adverse conditions
- Uncertainty estimation

3. **Foundation models:**

- SAM (Segment Anything) integration
- GPT-4V για scene understanding
- Unified perception models

### 8.4.3 Ερευνητικές Ερωτήσεις

Η εργασία ανοίγει ερωτήματα για μελλοντική έρευνα:

1. Πώς μπορεί το domain gap να μειωθεί για zero-shot models;
2. Ποιο είναι το optimal trade-off μεταξύ model size και accuracy για edge deployment;
3. Μπορούν τα vision-language models να επιτύχουν supervised-level performance με minimal fine-tuning;
4. Πώς μπορεί η temporal consistency να αξιοποιηθεί σε video-based detection;

## 8.5 Τελικές Σκέψεις

Η παρούσα εργασία παρέχει μια ολοκληρωμένη σύγκριση τριών διαφορετικών παραδειγμάτων object detection:

- **Classical two-stage:** Faster R-CNN - ακαδημαϊκή αξία, κατανόηση fundamentals
- **Modern one-stage:** YOLOv3 - πρακτική αξία, production-ready
- **Emerging zero-shot:** Grounding DINO - μελλοντική κατεύθυνση, flexibility

Κάθε προσέγγιση έχει τη θέση της στο spectrum από research έως production, και η επιλογή εξαρτάται από τις συγκεκριμένες απαιτήσεις της εφαρμογής.

Η εξέλιξη του πεδίου συνεχίζεται ραγδαία, με τα foundation models να υπόσχονται unified approaches που θα συνδυάζουν τα πλεονεκτήματα όλων των παραδειγμάτων.

# Βιβλιογραφία

- [1] Alexey Bochkovskiy, Chien-Yao Wang και Hong-Yuan Mark Liao. «YOLOv4: Optimal speed and accuracy of object detection». Στο: *arXiv preprint arXiv:2004.10934* (2020).
- [2] Alexander Buslaev, Vladimir I Iglovikov, Eugene Khvedchenya, Alex Parinov, Mikhail Druzhinin και Alexandr A Kalinin. «Albumentations: Fast and flexible image augmentations». Στο: *Information* 11.2 (2020), σ. 125.
- [3] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov και Sergey Zagoruyko. «End-to-end object detection with transformers». Στο: *European Conference on Computer Vision (ECCV)*. 2020, σσ. 213–229.
- [4] Navneet Dalal και Bill Triggs. «Histograms of oriented gradients for human detection». Στο: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '05)*. Τόμ. 1. 2005, σσ. 886–893.
- [5] Jacob Devlin, Ming-Wei Chang, Kenton Lee και Kristina Toutanova. «BERT: Pre-training of deep bidirectional transformers for language understanding». Στο: *arXiv preprint arXiv:1810.04805* (2018).
- [6] Pedro F Felzenszwalb, Ross B Girshick, David McAllester και Deva Ramanan. «Object detection with discriminatively trained part-based models». Στο: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32.9 (2010), σσ. 1627–1645.
- [7] Ross Girshick. «Fast R-CNN». Στο: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)* (2015), σσ. 1440–1448.
- [8] Ross Girshick, Jeff Donahue, Trevor Darrell και Jitendra Malik. «Rich feature hierarchies for accurate object detection and semantic segmentation». Στο: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2014), σσ. 580–587.
- [9] Kaiming He, Xiangyu Zhang, Shaoqing Ren και Jian Sun. «Deep residual learning for image recognition». Στο: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, σσ. 770–778.
- [10] Sergey Ioffe και Christian Szegedy. «Batch normalization: Accelerating deep network training by reducing internal covariate shift». Στο: *International Conference on Machine Learning (ICML)* (2015), σσ. 448–456.
- [11] Chao Jia, Yinfei Yang, Ye Xia, Yi-Ting Chen, Zarana Parekh, Hieu Pham, Quoc Le, Yun-Hsuan Sung, Zhen Li και Tom Duerig. «Scaling up visual and vision-language representation learning with noisy text supervision». Στο: *International Conference on Machine Learning (ICML)*. 2021, σσ. 4904–4916.

- [12] Liunian Harold Li, Pengchuan Zhang, Haotian Zhang, Jianwei Yang, Chunyuan Li, Yiwu Zhong, Lijuan Wang, Lu Yuan, Lei Zhang, Jenq-Neng Hwang κ.ά. «Grounded language-image pre-training». Στο: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022, σσ. 10965–10975.
- [13] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan και Serge Belongie. «Feature pyramid networks for object detection». Στο: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, σσ. 2117–2125.
- [14] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He και Piotr Dollár. «Focal loss for dense object detection». Στο: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2017, σσ. 2980–2988.
- [15] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár και C Lawrence Zitnick. «Microsoft COCO: Common objects in context». Στο: *European Conference on Computer Vision (ECCV)*. 2014, σσ. 740–755.
- [16] Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Chunyuan Li, Jianwei Yang, Hang Su, Jun Zhu κ.ά. «Grounding DINO: Marrying DINO with grounded pre-training for open-set object detection». Στο: *arXiv preprint arXiv:2303.05499* (2023).
- [17] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu και Alexander C Berg. «SSD: Single shot multibox detector». Στο: *European Conference on Computer Vision (ECCV)*. 2016, σσ. 21–37.
- [18] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin και Baining Guo. «Swin transformer: Hierarchical vision transformer using shifted windows». Στο: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2021, σσ. 10012–10022.
- [19] Matthias Minderer, Alexey Gritsenko, Austin Stone, Maxim Neumann, Dirk Weissenborn, Alexey Dosovitskiy, Aravindh Mahendran, Anurag Arnab, Mostafa Dehghani, Zhuoran Shen κ.ά. «Simple open-vocabulary object detection with vision transformers». Στο: *European Conference on Computer Vision (ECCV)*. 2022, σσ. 728–755.
- [20] Andreas Mogelmose, Mohan Manubhai Trivedi και Thomas B Moeslund. «Vision-based traffic sign detection and analysis for intelligent driver assistance systems: Perspectives and survey». Στο: *IEEE Transactions on Intelligent Transportation Systems*. Τόμ. 13. 4. 2012, σσ. 1484–1497.
- [21] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark κ.ά. «Learning transferable visual models from natural language supervision». Στο: *International Conference on Machine Learning (ICML)*. 2021, σσ. 8748–8763.
- [22] Joseph Redmon, Santosh Divvala, Ross Girshick και Ali Farhadi. «You only look once: Unified, real-time object detection». Στο: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, σσ. 779–788.
- [23] Joseph Redmon και Ali Farhadi. «YOLOv3: An incremental improvement». Στο: *arXiv preprint arXiv:1804.02767* (2018).
- [24] Shaoqing Ren, Kaiming He, Ross Girshick και Jian Sun. «Faster R-CNN: Towards real-time object detection with region proposal networks». Στο: *Advances in Neural Information Processing Systems (NeurIPS) 28* (2015).

- [25] Karen Simonyan και Andrew Zisserman. «Very deep convolutional networks for large-scale image recognition». Στο: *arXiv preprint arXiv:1409.1556* (2014).
- [26] Paul Viola και Michael Jones. «Rapid object detection using a boosted cascade of simple features». Στο: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*. Τόμ. 1. 2001, σσ. I–511.

# Παράρτημα Α'

## Πλήρης Κώδικας

Στο παρόν παράρτημα παρατίθεται ο πλήρης κώδικας των υλοποιήσεων όπως εκτελέστηκαν στο notebook.

### A'.1 Faster R-CNN με Torchvision

#### A'.1.1 Model Building

```
1 from torchvision.models.detection import fasterrcnn_resnet50_fpn
2 from torchvision.models.detection import FasterRCNN_ResNet50_FPN_Weights
3 from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
4
5 def build_faster_rcnn(num_classes):
6 # Load pretrained Faster R-CNN (COCO weights)
7 model = fasterrcnn_resnet50_fpn(
8 weights=FasterRCNN_ResNet50_FPN_Weights.COCO_V1
9)
10
11 # Replace classification head for our classes
12 in_features = model.roi_heads.box_predictor.cls_score.in_features
13 model.roi_heads.box_predictor = FastRCNNPredictor(
14 in_features, num_classes
15)
16
17 return model
18
19 # Build model: 9 classes + 1 background = 10
20 NUM_CLASSES = 10
21 faster_rcnn = build_faster_rcnn(NUM_CLASSES)
22 faster_rcnn = faster_rcnn.to(device)
23
24 print(f"Faster R-CNN (Torchvision)")
25 print(f" Classes: {NUM_CLASSES} (9 + background)")
26 print(f" Backbone: ResNet-50 + FPN (pretrained on ImageNet/COCO)")
```

### A'.1.2 Data Augmentation

---

```

1 import albumentations as A
2 from albumentations.pytorch import ToTensorV2
3 import cv2
4
5 IMG_SIZE = (384, 512)
6
7 train_transform = A.Compose([
8 A.Resize(height=IMG_SIZE[0], width=IMG_SIZE[1]),
9 A.HorizontalFlip(p=0.5),
10 A.RandomResizedCrop(
11 size=(IMG_SIZE[0], IMG_SIZE[1]),
12 scale=(0.8, 1.0), p=0.5
13),
14 A.Rotate(limit=10, p=0.5, border_mode=cv2.BORDER_CONSTANT),
15 A.Affine(scale=(0.9, 1.1), p=0.5),
16 A.Resize(height=IMG_SIZE[0], width=IMG_SIZE[1]),
17 A.Normalize(
18 mean=[0.485, 0.456, 0.406],
19 std=[0.229, 0.224, 0.225]
20),
21 ToTensorV2()
22], bbox_params=A.BboxParams(
23 format='pascal_voc',
24 label_fields=['labels'],
25 min_visibility=0.3
26))
27
28 val_transform = A.Compose([
29 A.Resize(height=IMG_SIZE[0], width=IMG_SIZE[1]),
30 A.Normalize(
31 mean=[0.485, 0.456, 0.406],
32 std=[0.229, 0.224, 0.225]
33),
34 ToTensorV2()
35], bbox_params=A.BboxParams(
36 format='pascal_voc',
37 label_fields=['labels']
38))

```

---

### A'.1.3 Training Configuration

---

```

1 NUM_EPOCHS = 10
2 LR = 0.005
3 MOMENTUM = 0.9
4 WEIGHT_DECAY = 0.0005
5 BATCH_SIZE = 1 # Due to variable image sizes
6

```

---

```

7 optimizer = torch.optim.SGD(
8 faster_rcnn.parameters(),
9 lr=LR,
10 momentum=MOMENTUM,
11 weight_decay=WEIGHT_DECAY
12)
13 scheduler = torch.optim.lr_scheduler.StepLR(
14 optimizer, step_size=7, gamma=0.1
15)

```

#### A'.1.4 Training Loop

```

1 train_losses = []
2
3 for epoch in range(NUM_EPOCHS):
4 faster_rcnn.train()
5 epoch_loss = 0
6
7 pbar = tqdm(train_loader, desc=f'Epoch {epoch+1}/{NUM_EPOCHS}')
8 for images, targets in pbar:
9 images = [img.to(device) for img in images]
10 targets = [{k: v.to(device) for k, v in t.items()} for t in targets]
11
12 # Forward - model returns Loss dict in training mode
13 loss_dict = faster_rcnn(images, targets)
14 loss = sum(loss_dict.values())
15
16 optimizer.zero_grad()
17 loss.backward()
18 torch.nn.utils.clip_grad_norm_(
19 faster_rcnn.parameters(), max_norm=10.0
20)
21 optimizer.step()
22
23 epoch_loss += loss.item()
24 pbar.set_postfix({'loss': f'{loss.item():.4f}'})
25
26 scheduler.step()
27 avg_loss = epoch_loss / len(train_loader)
28 train_losses.append(avg_loss)
29
30 print(f'Epoch {epoch+1}/{NUM_EPOCHS} - '
31 f'Loss: {avg_loss:.4f} - '
32 f'LR: {scheduler.get_last_lr()[0]:.6f}')
33
34 if avg_loss == min(train_losses):
35 torch.save(faster_rcnn.state_dict(), 'faster_rcnn_best.pth')
36 print('→ Saved best model')
37
38

```

---

39 torch.save(faster\_rcnn.state\_dict(), 'faster\_rcnn\_final.pth')

---

### A'.1.5 Evaluation with mAP

```

1 from torchmetrics.detection import MeanAveragePrecision
2
3 def evaluate_frcnn(model, data_loader, device):
4 model.eval()
5 metric = MeanAveragePrecision(iou_type='bbox')
6
7 with torch.no_grad():
8 for images, targets in tqdm(data_loader, desc='Evaluating'):
9 images = [img.to(device) for img in images]
10
11 # Inference mode - returns predictions
12 outputs = model(images)
13
14 preds = []
15 gts = []
16 for out, tgt in zip(outputs, targets):
17 preds.append({
18 'boxes': out['boxes'].cpu(),
19 'scores': out['scores'].cpu(),
20 'labels': out['labels'].cpu()
21 })
22 gts.append({
23 'boxes': tgt['boxes'],
24 'labels': tgt['labels']
25 })
26
27 metric.update(preds, gts)
28
29 return metric.compute()
30
31 # Load best model and evaluate
32 faster_rcnn.load_state_dict(
33 torch.load('faster_rcnn_best.pth', weights_only=True)
34)
35 frcnn_results = evaluate_frcnn(faster_rcnn, test_loader, device)
36
37 print(f'mAP@0.5: {frcnn_results['map_50'].item():.4f}')
38 print(f'mAP@0.5:0.95: {frcnn_results['map'].item():.4f}')
39 print(f'mAP@0.75: {frcnn_results['map_75'].item():.4f}')

```

---

---

## A'.2 YOLOv3 Training Script

```

1 from ultralytics import YOLO
2 import os
3
4 # Configuration
5 YOLO_DIR = "/content/yolo_dataset"
6 YOLO_EPOCHS = 50
7 YOLO_BATCH = 16 # Reduced from 32 due to Larger model size
8 YOLO_IMGSZ = 640
9
10 print("YOLOv3 Architecture (Darknet-53 backbone)")
11 print(f" Epochs: {YOLO_EPOCHS}")
12 print(f" Batch: {YOLO_BATCH}")
13 print(f" Image size: {YOLO_IMGSZ}")
14
15 # Load pretrained YOLOv3 (ultralytics optimized version)
16 model = YOLO('yolov3u.pt')
17
18 # Train
19 results = model.train(
20 data=os.path.join(YOLO_DIR, 'data.yaml'),
21 epochs=YOLO_EPOCHS,
22 imgsz=YOLO_IMGSZ,
23 batch=YOLO_BATCH,
24 name='yolov3_traffic_signs',
25 patience=10,
26 device=0,
27
28 # Augmentation
29 augment=True,
30 fliplr=0.5,
31 mosaic=1.0,
32 degrees=10.0,
33 scale=0.5,
34 translate=0.1,
35 hsv_h=0.015,
36 hsv_s=0.7,
37 hsv_v=0.4,
38
39 # Other settings
40 workers=4,
41 pretrained=True,
42 verbose=True
43)
44
45 print("YOLOv3 Training completed!")
46
47 # Evaluate on test set
48 best_model = YOLO('runs/detect/yolov3_traffic_signs/weights/best.pt')

```

---

```
49 test_results = best_model.val(
50 data=os.path.join(YOLO_DIR, 'data.yaml'),
51 split='test'
52)
53
54 print(f"Test mAP@0.5: {test_results.box.map50:.4f}")
55 print(f"Test mAP@0.5:0.95: {test_results.box.map:.4f}")
```

## A'.3 Grounding DINO Inference

### A'.3.1 Model Loading

---

```

1 # Install dependencies
2 !pip install -q groundingdino-py
3 !pip install -q supervision
4
5 # Download model files
6 !wget -q https://github.com/IDEA-Research/GroundingDINO/releases/\
7 download/v0.1.0-alpha/groundingdino_swint_ogc.pth
8
9 from groundingdino.util.inference import load_model, load_image, predict
10
11 grounding_dino = load_model(
12 "GroundingDINO_SwinT_OGC.py",
13 "groundingdino_swint_ogc.pth"
14)
15 grounding_dino = grounding_dino.to(device)

```

---

### A'.3.2 Text Prompts

---

```

1 # Three prompt strategies as required by assignment
2 PROMPT_BASIC = "stop sign, speed limit sign, yield sign"
3
4 PROMPT_FULL = ("keep right sign, merge sign, "
5 "pedestrian crossing sign, signal ahead sign, "
6 "speed limit 25 sign, speed limit 35 sign, "
7 "stop sign, yield sign, yield ahead sign")
8
9 PROMPT_GENERIC = "traffic sign, road sign, warning sign, regulatory sign"
10
11 # Detection thresholds
12 BOX_THRESHOLD = 0.35
13 TEXT_THRESHOLD = 0.25

```

---

### A'.3.3 Detection Function

---

```

1 def detect_with_grounding_dino(model, image_path, text_prompt,
2 box_threshold=0.35, text_threshold=0.25):
3 image_source, image = load_image(image_path)
4
5 boxes, logits, phrases = predict(
6 model=model,
7 image=image,
8 caption=text_prompt,
9 box_threshold=box_threshold,

```

---

```

10 text_threshold=text_threshold
11)
12
13 h, w = image_source.shape[:2]
14 if len(boxes) > 0:
15 # Convert from center format to xyxy
16 boxes = boxes * torch.tensor([w, h, w, h])
17 boxes_xyxy = torch.zeros_like(boxes)
18 boxes_xyxy[:, 0] = boxes[:, 0] - boxes[:, 2] / 2
19 boxes_xyxy[:, 1] = boxes[:, 1] - boxes[:, 3] / 2
20 boxes_xyxy[:, 2] = boxes[:, 0] + boxes[:, 2] / 2
21 boxes_xyxy[:, 3] = boxes[:, 1] + boxes[:, 3] / 2
22 boxes = boxes_xyxy
23
24 return boxes, logits, phrases, image_source

```

#### A'.3.4 Evaluation with All Prompts

```

1 def evaluate_gdino_on_test(model, test_df, data_dir, prompt):
2 """Evaluate Grounding DINO on test set with given prompt."""
3 total_tp, total_fp, total_fn = 0, 0, 0
4 all_scores, all_matches = [], []
5
6 for img_name in tqdm(test_df['filename'].unique()):
7 img_path = os.path.join(data_dir, img_name)
8 boxes, logits, phrases, _ = detect_with_grounding_dino(
9 model, img_path, prompt
10)
11
12 # Get ground truth
13 img_anns = test_df[test_df['filename'] == img_name]
14 gt_boxes = [[r['x_min'], r['y_min'], r['x_max'], r['y_max']]
15 for _, r in img_anns.iterrows()]
16
17 matched_gt = set()
18 for i, pred_box in enumerate(boxes):
19 best_iou, best_j = 0, -1
20 for j, gt_box in enumerate(gt_boxes):
21 if j in matched_gt:
22 continue
23 iou = compute_iou(pred_box.numpy(), gt_box)
24 if iou > best_iou:
25 best_iou, best_j = iou, j
26
27 if best_iou >= 0.5:
28 matched_gt.add(best_j)
29 total_tp += 1
30 all_matches.append(1)
31 else:
32 total_fp += 1

```

```

33 all_matches.append(0)
34 all_scores.append(logits[i].item())
35
36 total_fn += len(gt_boxes) - len(matched_gt)
37
38 precision = total_tp / (total_tp + total_fp) if total_tp + total_fp >
39 ↪ 0 else 0
39 recall = total_tp / (total_tp + total_fn) if total_tp + total_fn > 0
40 ↪ else 0
41
41 return {
42 'mAP_50': compute_ap(all_scores, all_matches, total_tp +
43 ↪ total_fn),
43 'precision': precision,
44 'recall': recall
45 }
46
47 # Evaluate all prompts
48 gdino_results = {}
49 for prompt_name, prompt in [
50 ('PROMPT_BASIC', PROMPT_BASIC),
51 ('PROMPT_FULL', PROMPT_FULL),
52 ('PROMPT_GENERIC', PROMPT_GENERIC)
53]:
54 results = evaluate_gdino_on_test(grounding_dino, test_df, DATA_DIR,
55 ↪ prompt)
55 gdino_results[prompt_name] = results
56 print(f"{prompt_name}:")
57 print(f" mAP@0.5: {results['mAP_50']:.4f}")
58 print(f" Precision: {results['precision']:.4f}")
59 print(f" Recall: {results['recall']:.4f}")

```

# Παράρτημα Β'

## Υπερπαράμετροι και Ρυθμίσεις

Στο παρόν παράρτημα καταγράφονται αναλυτικά όλοι οι υπερπαράμετροι που χρησιμοποιήθηκαν για κάθε μοντέλο.

### B'.1 Faster R-CNN

#### B'.1.1 Backbone Configuration

Πίνακας B'.1: Backbone Network Configuration

| Parameter            | Value                                       |
|----------------------|---------------------------------------------|
| Architecture         | ResNet-50 FPN (torchvision COCO pretrained) |
| Input Channels       | 3 (RGB)                                     |
| Initial Conv Filters | 64                                          |
| Initial Conv Kernel  | $7 \times 7$                                |
| Initial Conv Stride  | 2                                           |
| Residual Block Type  | BasicBlock                                  |
| Layer 1 Filters      | 64                                          |
| Layer 2 Filters      | 128                                         |
| Layer 3 Filters      | 256                                         |
| Blocks per Layer     | 2                                           |
| Output Stride        | 16                                          |
| Output Channels      | 256                                         |
| Activation           | ReLU                                        |
| Normalization        | Batch Normalization                         |

### B'.1.2 RPN Configuration

Πίνακας B'.2: Region Proposal Network Configuration

| Parameter              | Value                  |
|------------------------|------------------------|
| Input Channels         | 256                    |
| Hidden Channels        | 512                    |
| Anchor Sizes           | [32, 64, 128] pixels   |
| Aspect Ratios          | [0.5, 1.0, 2.0]        |
| Number of Anchors      | 9 (3 sizes × 3 ratios) |
| Pre-NMS Top-N (Train)  | 2000                   |
| Pre-NMS Top-N (Test)   | 1000                   |
| Post-NMS Top-N (Train) | 300                    |
| Post-NMS Top-N (Test)  | 100                    |
| NMS Threshold          | 0.7                    |
| Minimum Box Size       | 1 pixel                |
| Positive IoU Threshold | 0.7                    |
| Negative IoU Threshold | 0.3                    |
| Batch Size per Image   | 256                    |
| Positive Fraction      | 0.5                    |

### B'.1.3 ROI Pooling Configuration

Πίνακας B'.3: ROI Pooling Configuration

| Parameter      | Value                  |
|----------------|------------------------|
| Output Size    | 7×7                    |
| Spatial Scale  | 1/16 (backbone stride) |
| Sampling Ratio | 2                      |

### B'.1.4 Detection Head Configuration

Πίνακας B'.4: Detection Head Configuration

| Parameter              | Value                           |
|------------------------|---------------------------------|
| Input Features         | $256 \times 7 \times 7 = 12544$ |
| Hidden Units           | 1024                            |
| Number of Classes      | 10 (9 + background)             |
| Box Regression Outputs | 40 ( $10 \times 4$ )            |
| Positive IoU Threshold | 0.5                             |
| Negative IoU Threshold | 0.5                             |
| Batch Size per Image   | 64                              |
| Positive Fraction      | 0.25                            |
| Score Threshold        | 0.05                            |
| NMS Threshold          | 0.5                             |
| Detections per Image   | 100                             |

### B'.1.5 Training Configuration

Πίνακας B'.5: Faster R-CNN Training Configuration

| Parameter         | Value             |
|-------------------|-------------------|
| Optimizer         | SGD with Momentum |
| Learning Rate     | 0.005             |
| Momentum          | 0.9               |
| Weight Decay      | 0.0005            |
| Batch Size        | 4                 |
| Epochs            | 10                |
| LR Scheduler      | StepLR            |
| LR Step Size      | 3 epochs          |
| LR Gamma          | 0.1               |
| Gradient Clipping | None              |
| Mixed Precision   | No                |

## B'.2 YOLOv3

### B'.2.1 Model Configuration

Πίνακας B'.6: YOLOv3 Model Configuration

| Parameter         | Value                                                  |
|-------------------|--------------------------------------------------------|
| Backbone          | Darknet-53 (COCO pretrained)                           |
| Detection Scales  | 3 ( $13 \times 13$ , $26 \times 26$ , $52 \times 52$ ) |
| Anchors per Scale | 3                                                      |
| Total Anchors     | 9                                                      |
| Input Size        | $640 \times 640$                                       |
| Number of Classes | 9                                                      |

### B'.2.2 Anchor Configuration

Πίνακας B'.7: YOLOv3 Anchor Boxes

| Scale  | Feature Map    | Anchors (w, h)                 |
|--------|----------------|--------------------------------|
| Large  | $13 \times 13$ | (116,90), (156,198), (373,326) |
| Medium | $26 \times 26$ | (30,61), (62,45), (59,119)     |
| Small  | $52 \times 52$ | (10,13), (16,30), (33,23)      |

### B'.2.3 Training Configuration

Πίνακας B'.8: YOLOv3 Training Configuration

| Parameter             | Value            |
|-----------------------|------------------|
| Optimizer             | SGD              |
| Initial Learning Rate | 0.01             |
| Final Learning Rate   | 0.0001           |
| Momentum              | 0.937            |
| Weight Decay          | 0.0005           |
| Batch Size            | 16               |
| Epochs                | 50               |
| Warmup Epochs         | 3                |
| Warmup Momentum       | 0.8              |
| Warmup Bias LR        | 0.1              |
| LR Scheduler          | Cosine Annealing |

## B'.2.4 Augmentation Configuration

Πίνακας B'.9: YOLOv3 Data Augmentation

| Augmentation       | Value/Probability |
|--------------------|-------------------|
| Mosaic             | 1.0               |
| Mixup              | 0.0               |
| Copy-Paste         | 0.0               |
| HSV-Hue            | 0.015             |
| HSV-Saturation     | 0.7               |
| HSV-Value          | 0.4               |
| Degrees (rotation) | 0.0               |
| Translate          | 0.1               |
| Scale              | 0.5               |
| Shear              | 0.0               |
| Perspective        | 0.0               |
| Flip Up-Down       | 0.0               |
| Flip Left-Right    | 0.5               |

## B'.3 Grounding DINO

### B'.3.1 Model Configuration

Πίνακας B'.10: Grounding DINO Model Configuration

| Parameter               | Value                     |
|-------------------------|---------------------------|
| Vision Backbone         | Swin Transformer (Swin-T) |
| Text Encoder            | BERT-base                 |
| Feature Enhancer Layers | 6                         |
| Decoder Layers          | 6                         |
| Number of Queries       | 900                       |
| Hidden Dimension        | 256                       |
| Attention Heads         | 8                         |
| FFN Dimension           | 2048                      |

### B'.3.2 Inference Configuration

Πίνακας B'.11: Grounding DINO Inference Configuration

| Parameter          | Value                             |
|--------------------|-----------------------------------|
| Box Threshold      | 0.35                              |
| Text Threshold     | 0.25                              |
| NMS Threshold      | 0.8                               |
| Maximum Detections | 300                               |
| Input Size         | Variable (aspect ratio preserved) |

### B'.3.3 Text Prompts

Πίνακας B'.12: Grounding DINO Text Prompts

| Target Class | Text Prompt         |
|--------------|---------------------|
| stop         | "stop sign"         |
| speedLimit25 | "speedLimit25 sign" |
| yield        | "yield sign"        |

## B'.4 Hardware και Environment

Πίνακας B'.13: Hardware και Software Environment

| Component           | Specification               |
|---------------------|-----------------------------|
| Platform            | Google Colab                |
| GPU                 | NVIDIA Tesla T4 (16GB VRAM) |
| CPU                 | Intel Xeon @ 2.00GHz        |
| RAM                 | 12.7 GB                     |
| Python Version      | 3.10                        |
| PyTorch Version     | 2.0+                        |
| torchvision Version | 0.15+                       |
| ultralytics Version | 8.0+                        |
| CUDA Version        | 11.8                        |

# Παράρτημα Γ'

## Πρόσθετα Αποτελέσματα και Οπτικοποιήσεις

Στο παρόν παράρτημα παρατίθενται επιπλέον αποτελέσματα, γραφήματα, και οπτικοποιήσεις που συμπληρώνουν την ανάλυση του κύριου κειμένου.

### Γ'.1 Grounding DINO Analysis

#### Γ'.1.1 Prompt Sensitivity Analysis

Πίνακας Γ'.1: Επίδραση Text Prompt στην Απόδοση

| Class | Prompt Variant            | AP@0.5 |
|-------|---------------------------|--------|
| stop  | ”stop sign”               | [PH]   |
|       | ”stop traffic sign”       | [PH]   |
|       | ”red octagonal stop sign” | [PH]   |
| yield | ”yield sign”              | [PH]   |
|       | ”yield traffic sign”      | [PH]   |
|       | ”triangular yield sign”   | [PH]   |

### Γ'.2 Speed Benchmarks

#### Γ'.2.1 Inference Time Distribution

Πίνακας Γ'.2: Λεπτομερής Ανάλυση Inference Time (ms)

| Model          | Mean | Std | Min | Max |
|----------------|------|-----|-----|-----|
| Faster R-CNN   | 150  | 20  | 120 | 200 |
| YOLOv3         | 25   | 3   | 20  | 35  |
| Grounding DINO | 400  | 50  | 350 | 550 |

## Γ'.2.2 Throughput Analysis

Πίνακας Γ'.3: Throughput ανά Batch Size

| Batch Size | Faster R-CNN | YOLOv3  | Grounding DINO |
|------------|--------------|---------|----------------|
| 1          | 7 FPS        | 40 FPS  | 2.5 FPS        |
| 4          | 12 FPS       | 80 FPS  | 4 FPS          |
| 8          | 15 FPS       | 100 FPS | OOM            |

## Γ'.3 Dataset Statistics

### Γ'.3.1 Class Distribution Details

Πίνακας Γ'.4: Λεπτομερής Κατανομή Instances

| Class              | Train      | Val       | Test      | Total      |
|--------------------|------------|-----------|-----------|------------|
| keepRight          | 80         | 10        | 10        | 100        |
| merge              | 80         | 10        | 10        | 100        |
| pedestrianCrossing | 80         | 10        | 10        | 100        |
| signalAhead        | 80         | 10        | 10        | 100        |
| speedLimit25       | 80         | 10        | 10        | 100        |
| speedLimit35       | 80         | 10        | 10        | 100        |
| stop               | 80         | 10        | 10        | 100        |
| yield              | 80         | 10        | 10        | 100        |
| yieldAhead         | 80         | 10        | 10        | 100        |
| <b>Total</b>       | <b>720</b> | <b>90</b> | <b>90</b> | <b>900</b> |

### Γ'.3.2 Bounding Box Size Statistics

Πίνακας Γ'.5: Στατιστικά Μεγέθους Bounding Box

| Statistic | Width (px) | Height (px) |
|-----------|------------|-------------|
| Mean      | 45         | 52          |
| Std       | 25         | 30          |
| Min       | 15         | 18          |
| Max       | 180        | 200         |
| Median    | 38         | 45          |