



T.C.
FATİH SULTAN MEHMET VAKIF ÜNİVERSİTESİ
Mühendislik Fakültesi
Bilgisayar Mühendisliği Bölümü

Lisans Bitirme Projesi II

**Otonom Araç İçin Derin Öğrenme Tabanlı Çözüm
Yaklaşımları**

Şerafettin Doruk SEZER - Dilara ÇELİK
2021221038 - 2021221042
Doç. Dr. Berna KİRAZ

İstanbul, Haziran 2025



T.C.
FATİH SULTAN MEHMET VAKIF ÜNİVERSİTESİ
Mühendislik Fakültesi
Bilgisayar Mühendisliği Bölümü

Lisans Bitirme Projesi II

**Otonom Araç İçin Derin Öğrenme Tabanlı Çözüm
Yaklaşımları**

Şerafettin Doruk SEZER - Dilara ÇELİK

2021221038 - 2021221042

Bitirme Projesi Danışmanı: Doç. Dr. Berna KİRAZ

Jüri Üyeleri:

İmza:

Dr. Öğr. Üyesi

Dr. Öğr. Üyesi

Dr. Öğr. Üyesi

İstanbul, Haziran 2025

ÖNSÖZ

18.06.2025

Celik

Şerafettin Doruk SEZER - Dilara

Otonom araç teknolojileri, modern mühendisliğin en heyecan verici ve yenilikçi alanlarından biri olarak, trafikte insan hatalarını azaltma, güvenliği artırma ve ulaşımı daha verimli hale getirme amacıyla taşımaktadır. Bu çalışma, sürücüsüz araçların çevresel algılama gereksinimlerini ele alarak, şerit takibi, trafik işaretleri algılama, araç ve yaya tespiti gibi temel görevlerin çözümüne odaklanmaktadır. Derin öğrenme algoritmalarını içeren bu projede, çoklu görev öğrenme yaklaşımı benimsenmiştir. Çalışmalar, CARLA simülasyon platformunda gerçekleştirilmiş olup, teorik ve uygulamalı yaklaşımlarla otonom araç teknolojilerine katkı sunmayı hedeflemiştir.

Bu projeyi hayata geçirirken bilgi ve deneyimleriyle bizi yönlendiren danışman hocamız Berna Kiraz'a en içten teşekkürlerimizi sunarız.

Bu süreç, yalnızca teknik bilgi birikimimizi artırmakla kalmayıp, aynı zamanda mühendislik problemlerine çözüm üretmenin sistematik bir yaklaşım ve disiplin gerektirdiğini bize bir kez daha göstermiştir. Bu çalışmanın, otonom araç teknolojilerinin daha güvenilir ve verimli hale gelmesine yönelik gelecekteki araştırmalara ışık tutmasını dileriz.

Saygılarımlızla

İçindekiler

ÖNSÖZ	ii
ÖZET	vi
SUMMARY	vii
1 GİRİŞ	1
1.1 Problem Tanımı	1
1.2 Çalışmanın Amacı	1
1.3 Çalışmanın Günlük Hayatta Önemli Etkileri	1
2 TEMEL KAVRAMLAR	3
2.1 Otonom Araç	3
2.2 Derin Öğrenme (Deep Learning)	3
2.3 Nesne Tespiti (Object Detection)	3
2.4 YOLO (You Only Look Once)	3
2.5 Detectron2 Nedir?	3
2.6 mAP (Mean Average Precision)	3
2.7 Precision ve Recall	3
2.8 CARLA Simülasyonu	4
2.9 COCO Formатı	4
3 İLGİLİ ÇALIŞMALAR/LİTERATÜR ARAŞTIRMASI	5
3.1 Detectron2 ile ilgili Yapılan Çalışmalar	5
3.2 Yolo ile ilgili Yapılan Çalışmalar	6
4 ÇÖZÜM YÖNTEMLERİ	7
4.1 Veri Kümesi	7
4.1.1 Veri Setinin Oluşturulması	7
4.1.2 Etiketleme Formatları	7
4.1.3 Sınıflar (Etiketlenen Nesneler)	7
4.1.4 Kullanım Amacı	8
4.2 Detectron2 Yöntemleri	8
4.2.1 Detectron2'nin Tercih Edilme Sebebi	8
4.2.2 Kullanılan Model Yapılandırırmaları	8
4.2.3 Veri Seti Hazırlığı	10
4.2.4 Model Eğitim Stratejileri ve Parametreleri	10
4.2.5 Zorluklar ve Karşılaşılan Problemler	11
4.3 Yolo Yöntemleri	11
4.3.1 Neden YOLOv8 ile Başladık?	11
4.3.2 YOLO Sürümüleri ve Basit Model Seçimi	12
4.3.3 Model Eğitim Stratejileri ve Parametreleri	12
4.3.3.1 Eğitim Ortamı	13
4.3.3.2 Optimizasyon Algoritması	13
4.3.3.3 Eğitim Döngüsü (Epochs) ve Batch Boyutu	13

4.3.3.4 Öğrenme Oranı Çizelgeleme (Learning Rate Scheduling) .	13
5 Bulgular ve Tartışma (Results and Discussion)	14
5.1 Detectron2 Bulgular ve tartışma	14
5.1.1 Model Tahminleri ile Gerçek Etiketlerin Karşılaştırılması	16
5.1.2 Model Performans Grafikleri	18
5.2 Yolo Bulgular ve tartışma	20
5.2.1 YOLOv8-YOLOv12 Sürümleri Üzerindeki Bulgular ve Karşılaştırmalar	20
5.2.2 Model Performans Grafikleri	21
5.2.2.1 YOLOv9s	21
5.2.2.2 YOLOv9c	27
5.2.3 Sonuçların Tablolu Karşılaştırması:	32
5.2.3.1 CARLA Simülasyon Kısıtlamaları, Yeni model denemeleri Ve Çözüm Yolları	33
5.2.3.2 YOLOv5x Modelinin Simülasyona Entegrasyonu	34
5.2.3.3 Görsel Çıktılar	36
6 Sonuç ve Gelecek Çalışmalar	39
Kaynakça	40

Şekil Listesi

Şekil 5.1	Faster R-CNN R_50_FPN_1x modelinin test verisi üzerindeki tahmin sonuçları ile gerçek etiketlerin karşılaştırılması	16
Şekil 5.2	Faster R-CNN R_50_FPN_3x modelinin test verisi üzerindeki tahmin sonuçları ile gerçek etiketlerin karşılaştırılması	17
Şekil 5.3	Faster R-CNN R_101_FPN_3x modelinin test verisi üzerindeki tahmin sonuçları ile gerçek etiketlerin karşılaştırılması	17
Şekil 5.4	Faster R-CNN R_101_FPN_3x modelinin performans metriklerinin grafiği	18
Şekil 5.5	Faster R-CNN R_101_FPN_3x modelinin karmaşıklık matrisi	19
Şekil 5.6	Faster R-CNN R_101_FPN_3x modelinin sınıf bazında ortalama hassasiyet (AP) performansı	20
Şekil 5.7	Yolov8'e göre veri kümelerindeki her bir nesne sınıfının dağılımı	21
Şekil 5.8	Eğitim Sonuçları ve Kayıp Grafikleri	21
Şekil 5.9	karışıklık Matrisi	22
Şekil 5.10	Normalize edilmiş karışıklık matrisi	23
Şekil 5.11	Modelin Hassasiyet (Precision) Eğrisi	23
Şekil 5.12	Tüm Sınıflar için Geri Çağırma Performansı Eğrisi	24
Şekil 5.13	Modelin Sınıf Bazında Hassasiyet ve Geri Çağırma Dengesi	24
Şekil 5.14	F1 Skoru Eğrisi Yolov9s için	25
Şekil 5.15	Eğitim Süreci Boyunca mAP, Hassasiyet, Geri Çağırma ve Kayıp Değişimi	26
Şekil 5.16	Eğitim Sonuçları ve Kayıp Grafikleri	27
Şekil 5.17	karışıklık Matrisi	27
Şekil 5.18	Normalize edilmiş karışıklık matrisi	28
Şekil 5.19	Modelin Hassasiyet (Precision) Eğrisi	29
Şekil 5.20	üm Sınıflar için Geri Çağırma Performansı Eğrisi	29
Şekil 5.21	Modelin Sınıf Bazında Hassasiyet ve Geri Çağırma Dengesi	30
Şekil 5.22	F1 Skoru Eğrisi Yolov9s için	31
Şekil 5.23	Eğitim Süreci Boyunca mAP, Hassasiyet, Geri Çağırma ve Kayıp Değişimi	32
Şekil 5.24	Carla araç tesbit resmi	37
Şekil 5.25	Carla trafik ışığı tesbit resmi	37
Şekil 5.26	Carla motorsiklet ve bisiklet resmi	38

Tablo Listesi

Tablo 5.1 Object Detection Model Performance Metrics	14
Tablo 5.2 YOLO Test Results Tablosu	32

Otonom Araç İçin Derin Öğrenme Tabanlı Çözüm Yaklaşımları

ÖZET

Bu çalışmada, otonom araçların çevrelerindeki nesneleri algılayabilme yeteneklerini artırmaya yönelik olarak derin öğrenme tabanlı nesne tespiti algoritmaları kullanılmıştır. Proje kapsamında iki farklı yaklaşım benimsenmiştir: YOLO (You Only Look Once) ve Detectron2. Her iki yöntem de görüntülerde yer alan trafik nesnelerinin (araç, bisiklet, yaya vb.) tespiti amacıyla kullanılmış olup, Carla simülasyon ortamında üretilmiş görsel verilerle eğitilmiştir. YOLO, tek adımlı ve hızlı bir algılama yapısı sunarken; Detectron2, daha modüler ve esnek bir yapı sağlayarak yüksek doğrulukta sonuçlar elde etmiştir. Projenin bir üyesi YOLO algoritmasını kullanarak nesneleri gerçek zamanlı olarak yüksek hızla tanımlayabilecek bir model geliştirmiştir; diğer üye ise Detectron2 kullanarak daha derin özellik çıkarımı yapan, daha hassas tespitler yapabilen bir model tasarlamıştır. Her iki yöntem de, kendi avantajları ve sınırlılıkları çerçevesinde otonom araçların çevre algılamasını gerçekleştirmiştir. Eğitim süreci boyunca modellerin başarımı, ortalama hassasiyet (mAP), doğruluk (precision), geri çağrıma (recall) gibi metriklerle değerlendirilmiştir. Elde edilen sonuçlar, her iki yaklaşının da otonom sürüsistemlerinde kullanılabilir olduğunu, ancak uygulama senaryosuna göre tercihlerin değişimini ortaya koymuştur. Geliştirilen modeller, Carla simülasyon ortamına entegre edilerek görsel testler yapılmış ve modellerin pratikteki başarımı da gözlemlenmiştir. Bu bütünsel çalışma, otonom araçlarda nesne tespiti alanında farklı derin öğrenme tekniklerinin nasıl uygulanabileceğini ve birbirlerini nasıl tamamlayabileceğini ortaya koymaktadır.

Otonom Araç İçin Derin Öğrenme Tabanlı Çözüm Yaklaşımları

SUMMARY

In this study, deep learning-based object detection algorithms were used to improve the ability of autonomous vehicles to detect objects in their surroundings. Two different approaches were adopted within the scope of the project: YOLO (You Only Look Once) and Detectron2. Both methods were used to detect traffic objects (such as vehicles, bicycles, and pedestrians) in images and were trained using visual data generated in the Carla simulation environment. YOLO offers a single-step, fast detection structure, while Detectron2 provides a more modular and flexible structure, achieving high accuracy results. One member of the project developed a model that can identify objects in real time at high speed using the YOLO algorithm. In contrast, another member designed a model that performs more in-depth feature extraction and more accurate detection using Detectron2. Both methods have enabled autonomous vehicles to perceive their surroundings within the framework of their advantages and limitations. Throughout the training process, the performance of the models was evaluated using metrics such as mean average precision (mAP), accuracy (precision), and recall. The results obtained revealed that both approaches are usable in autonomous driving systems, but preferences may vary depending on the application scenario. The developed models were integrated into the Carla simulation environment, and visual tests were conducted to observe their performance in practice. This comprehensive study demonstrates how different deep learning techniques can be applied and complement each other in the field of object detection in autonomous vehicles.

1 GİRİŞ

1.1 Problem Tanımı

Teknolojideki hızlı ilerlemeler, son yıllarda otonom sistemlerin özellikle de sürücüsüz araçların gelişimine önemli katkılar sağlamıştır. Otonom araçlar, insan müdahalesi olmadan çevrelerini algılayabilen, karar verebilen ve hareket edebilen sistemler olarak tanımlanır. Bu araçların işleyişinde en kritik bileşenlerden biri, çevre algılama (perception) yeteneğidir. Araç, çevresindeki nesneleri (araçlar, yayalar, bisikletliler, trafik işaretleri vb.) doğru bir şekilde tespit edemezse, güvenli ve verimli bir sürüş deneyimi sunması mümkün olmaz. Dolayısıyla otonom araçların etkinliği, sahip oldukları nesne tespiti algoritmalarının başarısıyla doğrudan ilişkilidir.

Geleneksel görüntü işleme teknikleriyle geliştirilen sistemler, dinamik trafik koşullarında ve gerçek zamanlı uygulamalarda sınırlı başarı göstermektedir. Farklı ışık koşulları, karmaşık arka planlar ve ani hareketler gibi faktörler bu sistemlerin performansını olumsuz etkileyebilmektedir. Bu nedenle, son yıllarda derin öğrenme temelli nesne tespiti yaklaşımları, otonom sistemlerin vazgeçilmez bir parçası haline gelmiştir. Bu çalışmada da bu doğrultuda, derin öğrenmeye dayalı modern nesne tespiti yöntemlerinin kullanımı incelenmiş ve iki farklı yaklaşım olan YOLOv5 ve Detectron2, otonom sürüş senaryolarında test edilmiştir.

1.2 Çalışmanın Amacı

Bu bitirme projesinin temel amacı, otonom araçlarda çevre algılamaya yönelik olarak kullanılan güncel derin öğrenme algoritmalarını uygulamak, değerlendirmek ve karşılaştırmaktır. Proje kapsamında, YOLOv5 ve Detectron2 algoritmalarıyla nesne tespiti sistemleri geliştirilmiştir. Her iki yaklaşım, CARLA 0.9.13 sürümüne sahip açık kaynak simülasyon platformundan elde edilen görsel veriler üzerinde eğitilmiş ve test edilmiştir. Eğitim sürecinde elde edilen performans metrikleri kullanılarak modellerin doğruluğu, hızı ve uygulama açısından uygunluğu analiz edilmiştir. Ayrıca modeller, Carla ortamına entegre edilerek pratikteki performansları da gözlemlenmiş ve sistemlerin gerçek zamanlı kullanım potansiyelleri değerlendirilmiştir.

1.3 Çalışmanın Günlük Hayatta Önemli Etkileri

Bu çalışma kapsamında geliştirilen derin öğrenme tabanlı nesne tespiti sistemlerinin, yalnızca akademik bir problem çözümünün ötesinde, günlük yaşamda ciddi etkileri bulunmaktadır. Otonom araç teknolojileri, trafik güvenliği, enerji verimliliği ve şehir içi ulaşım sistemlerinde devrim yaratabilecek potansiyele sahiptir. Nesne tespiti sistemlerinin başarımı, bu araçların karayolu üzerinde diğer araçlar, yayalar ve çevresel unsurlar ile güvenli bir şekilde etkileşime girebilmesinde belirleyici bir rol oynamaktadır.

Özellikle şehir içi trafiğinde, aniden yola çıkan yayalar, bisikletliler ve diğer dinamik engellerin doğru tespiti, olası kazaların önlenmesinde kritik önem taşır. Bu bağlamda, bu çalışmada test edilen YOLOv5 ve Detectron2 gibi güncel derin öğrenme algoritmaları, bu tür senaryolarda otonom araçların daha güvenli kararlar alabilmesini sağlamaktadır.

Ayrıca, bu teknolojilerin gelişmesiyle birlikte:

Trafik kazalarının önemli ölçüde azaltılması,

İnsan hatasından kaynaklanan sürüs problemlerinin minimize edilmesi,

Engelli bireyler ve yaşlılar gibi ulaşımda dezavantajlı gruplara daha fazla bağımsızlık sağlanması,

Akıllı şehir altyapılarının gelişimine katkı sunulması beklenmektedir.

Simülasyon ortamında elde edilen sonuçların, gerçek dünya uygulamalarına aktarılabilir olması, bu tür sistemlerin daha güvenli ve sürdürülebilir ulaşım çözümlerine katkıda bulunacağını göstermektedir. Bu yönyle, çalışma yalnızca teknik bir başarı değil, aynı zamanda toplum yararına hizmet eden bir adım niteliği taşımaktadır.

Çalışma, otonom sürüs sistemlerinin geleceğini şekillendirecek teknolojilerin temel yapı taşlarını irdelemekte; algoritmaların gerçek dünya koşullarında nasıl performans gösterdiğine ışık tutmaktadır.

2 TEMEL KAVRAMLAR

2.1 Otonom Araç

Otonom araçlar, insan sürücüsüne ihtiyaç duymadan çevrelerini algılayabilen, bu algılamalara göre kararlar alabilen ve hareket edebilen sistemlerdir. Bu araçlar, çeşitli sensörler (kamera, radar, lidar), yapay zeka algoritmaları ve kontrol sistemleri ile donatılmıştır. Otonom araçların başarısı, çevreyi doğru algılayabilme ve doğru karar verebilme kabiliyetiyle doğrudan ilişkilidir.[1]

2.2 Derin Öğrenme (Deep Learning)

Derin öğrenme, çok katmanlı yapay sinir ağlarının kullanıldığı, büyük veri kümelerinden anlamlı temsiller çıkarabilen bir makine öğrenimi alanıdır. Görüntü, ses ve metin gibi karmaşık veriler üzerinde yüksek başarı oranlarına sahiptir. Görüntü işleme alanında sıkılıkla tercih edilen yöntemlerden biridir.[2]

2.3 Nesne Tespiti (Object Detection)

Nesne tespiti, bir görüntüde yer alan nesnelerin sınıfını belirleme ve bu nesnelerin konumlarını (bounding box) tahmin etme işlemidir. Bu işlem; sınıflandırma, lokalizasyon ve çoklu nesne algılama bileşenlerini aynı anda içerir. Otonom sürüste, bu görev yol üzerindeki nesnelerin fark edilmesi için kritik önemdedir.[3]

2.4 YOLO (You Only Look Once)

YOLO, nesne tespiti işlemini tek bir sinir ağı ileri geçiş ile gerçekleştiren, hızlı ve gerçek zamanlı çalışabilen bir algoritmadır. YOLOv5, bu serinin güncel ve optimize edilmiş sürümlerinden biridir. YOLO algoritmaları, genellikle hızın önemli olduğu uygulamalarda tercih edilir.[4]

2.5 Detectron2 Nedir?

Detectron2, Facebook AI Research (FAIR) tarafından geliştirilen açık kaynaklı bir nesne tespiti kütüphanesidir. PyTorch tabanlı olup, Mask R-CNN, Faster R-CNN gibi gelişmiş tespit algoritmalarını destekler. Yüksek doğruluk ve modüler yapı gerektiren projelerde tercih edilmektedir.[5]

2.6 mAP (Mean Average Precision)

mAP, bir nesne tespit sisteminin başarısını ölçmek için kullanılan yaygın bir metriktir. Precision-recall eğrileri üzerinden hesaplanır ve modelin farklı sınıflardaki ortalama hassasiyetini gösterir. mAP değeri ne kadar yüksekse, modelin tespit performansı o kadar iyidir.[6]

2.7 Precision ve Recall

Precision (Doğruluk): Tespit edilen nesnelerden kaç tanesinin gerçekten doğru olduğunu gösterir.

Recall (Geri Çağırma): Gerçek nesnelerden kaç tanesinin doğru tespitini edildiğini gösterir. Bu iki değer, modelin doğruluk ve duyarlılık dengesini anlamada önemlidir.[7]

2.8 CARLA Simülasyonu

CARLA, açık kaynak kodlu bir otonom sürüş simülasyon platformudur. Gerçekçi trafik senaryoları, sensör verisi üretimi ve özelleştirilmiş veri kümeleri oluşturmak için kullanılır. Bu proje kapsamında, nesne tespiti için gerekli olan eğitim ve test verileri CARLA 0.9.13 sürümü kullanılarak üretilmiştir.[8]

2.9 COCO Formatı

COCO (Common Objects in Context), bilgisayarla görme uygulamaları için yaygın olarak kullanılan bir veri seti formatıdır. Nesnelerin sınıf bilgisi, bounding box koordinatları ve görsel ID'leri gibi yapılar içerir. YOLO ve Detectron2 modelerin eğitimi için veriler bu formata dönüştürülmüştür.[9]

3 İLGİLİ ÇALIŞMALAR/LİTERATÜR ARAŞTIRMA-SI

Nesne tespiti, son yıllarda derin öğrenme tabanlı yöntemlerin gelişimiyle birlikte bilgisayarla görüş alanında en yoğun çalışılan konulardan biri haline gelmiştir. Otonom araçlar ve çevresel algılamaya yönelik yapılan güncel çalışmalar incelenerek, Detectron2 ve YOLO gibi yaygın olarak kullanılan modeller üzerine yürütülen literatür araştırmalarının neler olduğunu anlatacağız.

3.1 Detectron2 ile ilgili Yapılan Çalışmalar

Changhao Dong (2023) tarafından yazılan “Exploring visual techniques for indoor intrusion detection using detectron2 and Faster RCNN” başlıklı makale, iç mekan izinsiz giriş tespitinde görsel yöntemlerin kullanımını araştırmıştır. Amaç: Makale, iç mekan güvenliği için gözetim videolarında izinsiz giriş tespiti yapmayı amaçlamıştır.[10]

Yaklaşım: Video fark haritası ön işleme sonrası , Detectron2 ve Faster R-CNN’ın birleşik kullanımına dayalı bir yaya algılama algoritması geliştirilmiştir. Faster R-CNN, RPN ve Fast R-CNN’ı birleştirirken , Detectron2 platformu üstün performans ve esneklik sunar.

Bulgular: Geliştirilen algoritma, gözetim videolarında (özellikle ciddi hedef engellenmesi olmayan durumlarda) yüksek doğruluk, sağlamlık ve gerçek zamanlı performans göstermiştir. Küçük hedefleri algılamada ve farklı ışık koşullarına dayanıklılıkta başarılıdır.

Sınırlamalar ve Öneriler: Yoğun insan kalabalığı veya ciddi engellenme durumlarda modelin performansı sınırlıdır. Bu durumlar için özellik çıkarma ve birleştirme tekniklerinin geliştirilmesi, ayrıca gerçek zamanlı performansı artırmak amacıyla YOLOv3 ile hibrit yaklaşım önerilmiştir.

Joonho Byun, Jungjoon Kim ve Siwoo Byun (2024) tarafından yazılan “Object Detection with Detectron2 for Pothole Detection” başlıklı makale, çukur algılama için Detectron2 tabanlı bir derin öğrenme modelini önermektedir.[11]

Amaç: Makale, yol yüzeyi hasarlarının ana nedenlerinden biri olan çukurların hızlı bir şekilde tespit edilmesi ihtiyacını vurgulayarak, bu tespiti gerçekleştirmek için bir derin öğrenme modeli geliştirmeyi amaçlamıştır. İnsan denetiminin sınırlamalarının üstesinden gelmeyi hedeflemektedir.

Yaklaşım: Çalışma, PyTorch tabanlı nesne algılama ve anlamsal segmentasyon için bir eğitim çıkışım platformu olan Detectron2’ye dayalı bir model önermektedir.

Bulgular: Deneyde toplam 1334 görüntü eğitilmiştir. Elde edilen sınıf doğruluğu 0.94 ve maske doğruluğu 0.89 olup, bu durum iyi bir genel performans göstermektedir. Bir görüntünün işleme süresi ortalama 0.11 saniye olarak kaydedilmiştir, bu da gerçek hayatı yeterince uygulanabilir kabul edilmektedir. 199 test görüntüsü 21.42 saniyede işlenmiştir, ancak işleme süresinin örnek sayısına göre değiştiği belirtilmiştir.

Sonuç: Önerilen modelin, binlerce kilometrelik yolun insan denetimi sınırlamalarını aşabileceği ifade edilmiştir.

3.2 Yolo ile ilgili Yapılan Çalışmalar

Bu çalışmanın odak noktası olan YOLO (You Only Look Once) mimarisi, gerçek zamanlı nesne tespiti için çığır açan bir yaklaşım sunmuştur. Joseph Redmon ve arkadaşları tarafından 2016'da tanıtılan ilk YOLO modeli [12], tüm görüntü üzerinde tek bir regresyon problemi olarak nesne tespitini formüle ederek, diğer modellere göre önemli ölçüde daha yüksek hızlar elde etmiştir. Bu hız, özellikle otonom araçlar gibi düşük gecikme süresi gerektiren uygulamalar için hayatı önem taşımaktadır.

YOLO serisi, geçen yıllar içinde önemli evrimler geçirmiştir. YOLOv3 [13], daha iyi doğruluk için çoklu ölçekli tahminler ve daha karmaşık bir ağ mimarisi sunmuştur. Ultralytics tarafından geliştirilen YOLOv5 [14], PyTorch tabanlı hafif yapısıyla geliştirme ve dağıtım kolaylığı sağlamış, aynı zamanda yüksek performans sunmaya devam etmiştir. Sonraki sürümler olan YOLOv8, YOLOv9 [15] ve diğerleri ise mimari iyileştirmeler, yeni kayıp fonksiyonları ve eğitim stratejileriyle hem doğruluk hem de verimlilik açısından sürekli olarak iyileştirmeler getirmiştir. Özellikle YOLOv9 [15], enerji verimli bir mimarı ve yeni öğrenme teknikleri (örneğin, Generalized ELAN (GELAN) mimarisi ve Kapsayıcı Bilgi Bütünlüğü Ağrı (PANI)) ile parametre verimliliğini artırırken üstün performans sergilemektedir.

Bu çalışmada, hem YOLO serisinin farklı sürümleri hem de CARLA simülasyon ortamından elde edilen veri kümeleri üzerinde gerçekleştirilen nesne tespiti uygulamaları, literatürdeki bu gelişmelerin pratik bir yansımı niteliğindedir.

4 ÇÖZÜM YÖNTEMLERİ

4.1 Veri Kümesi

Bu çalışmada, otonom araçlar için nesne tespiti üzerine geliştirilen derin öğrenme modellerinin eğitimi ve değerlendirilmesi amacıyla Carla Object Detection Dataset adlı özel bir veri seti kullanılmıştır [16]. Bu veri seti, CARLA simülasyon ortamı içerisinde otonom sürüş senaryolarına uygun şekilde oluşturulmuştur ve otonom araçların çevresel farkındalıklarını geliştirmeye yönelik olarak tasarlanmıştır.

4.1.1 Veri Setinin Oluşturulması

Veri seti, CARLA simülatörünün autopilot (otomatik sürüş) modu kullanılarak oluşturulmuştur. Bu mod sayesinde araç, önceden tanımlanmış yollar üzerinde otomatik olarak ilerlerken simülasyon ortamından düzenli aralıklarla görseller (frame) kaydedilmiştir. Görüntüler; Town01, Town02, Town03, Town04 ve Town05 gibi farklı sanal şehir haritalarında elde edilmiştir. Bu sayede, farklı ışık, yol ve çevre koşulları simüle edilerek çeşitli senaryolar için zengin ve dengeli bir veri havuzu oluşturulmuştur.

Toplanan veriler, modelin etkin şekilde eğitilebilmesi için eğitim (train), doğrulama (validation) ve test olmak üzere üç ayrı alt kümeye ayrılmıştır. Bu kapsamda toplamda 1018 görüntüden 669'u eğitim, 100'u doğrulama ve 249'u ise test verisi olarak kullanılmıştır. Bu dağılım, modelin genelleme yeteneğini değerlendirebilmek amacıyla dengeli bir biçimde yapılmıştır.

4.1.2 Etiketleme Formatları

Toplanan görüntüler, her biri nesne tespiti modellerine uygun şekilde etiketlenmiş olup üç farklı formatta etiketleme dosyası içerir:

YOLO Formatı: YOLO algoritmasına uygun şekilde hazırlanmış .txt uzantılı etiketler, labels_yolo_format/ klasöründe yer almaktadır.

MS COCO Formatı: JSON tabanlı COCO formatında etiketlenmiş dosyalar, annotations/ klasöründe yer almaktak ve özellikle Detectron2 gibi framework'ler için kullanılmaktadır.

Bu çoklu format desteği sayesinde veri seti hem YOLO hem de Detectron2 gibi farklı yapay zeka modellerine kolayca entegre edilebilecek esneklikte tasarlanmıştır.

4.1.3 Sınıflar (Etiketlenen Nesneler)

Veri seti, otonom sürüş senaryolarında sık karşılaşılan nesneleri kapsayan beş farklı sınıf içermektedir:

Vehicle (Araçlar – otomobil, kamyon)

Bike (Bisiklet)

Motorbike (Motosiklet)

Traffic Light (Trafik ışığı)

Traffic Sign (Trafik levhası)

Her bir sınıf, simülasyon ortamında bulunan nesnelere karşılık gelmektedir ve tespit modellerinin bu sınıflar üzerinde eğitim alması sağlanmıştır.

4.1.4 Kullanım Amacı

Carla Object Detection Dataset, otonom araç geliştirme süreçlerinde nesne tespiti yeteneklerini test etmek amacıyla özel olarak geliştirilmiştir. Bu çalışmada, veri seti hem YOLOv5 hem de Detectron2 modellerinin eğitilmesi ve değerlendirilmesinde kullanılmış; her modelin performansı, bu veri seti üzerinde test edilmiştir.

Buna ek olarak, veri seti:

Derin öğrenme temelli bilgisayarla görme araştırmaları için uygun bir temel sağlamaktadır.

Gerçek dünya senaryolarını simüle eden ortamlar sayesinde genellenebilir modeller üremeye imkân tanımaktadır.

Bu yönyle, çalışmada kullanılan veri seti hem akademik araştırmalar hem de endüstriyel uygulamalar açısından önemli bir katkı sunmaktadır.

4.2 Detectron2 Yöntemleri

Detectron2, Facebook AI Research (FAIR) tarafından PyTorch altyapısı kullanılarak geliştirilmiş, modüler ve güncel bir bilgisayarla görme kütüphanesidir. Nesne tespiti, ayırmacı segmentasyon, panoptic segmentasyon gibi çok çeşitli görevleri destekler. Model Zoo desteği sayesinde Faster R-CNN, Mask R-CNN ve RetinaNet gibi ileri düzey mimarileri hızla entegre etme imkânı sağlar.

4.2.1 Detectron2'nin Tercih Edilme Sebebi

Bu çalışmada Detectron2 tercih edilmesinin başlıca nedenleri şunlardır:

- Modüler Yapı:** Farklı backbone mimarileri, veri kümeleri ve eğitim stratejileri kolayca entegre edilebilir.
- Güncel Model Desteği:** Faster R-CNN, Mask R-CNN, RetinaNet gibi modern nesne tespiti mimarileri hızlıca uygulanabilir.
- PyTorch Tabanlı:** Araştırma topluluğu tarafından yaygın olarak desteklenen PyTorch ile tam uyumludur.
- Gerçek Zamanlı Uygulamalara Uygunluk:** Hem doğruluk hem de hız açısından denge sağlayan modeller içerir.
- Simülasyon Verisi ile Uyum:** CARLA gibi simülasyon ortamlarından elde edilen özel veri kümeleri üzerinde kolayca çalıştırılabilir.

Bu avantajlar göz önüne alındığında, nesne tespiti görevinde Detectron2, hem esneklik hem de performans açısından güçlü bir çözüm sunmaktadır.

4.2.2 Kullanılan Model Yapılandırmaları

Bu çalışmada, Detectron2 Model Zoo'da yer alan üç farklı Faster R-CNN yapılandırması değerlendirilmiştir. Modeller; ağı derinliği, eğitim süresi ve performans

açısından farklılık göstermekte olup, özellikle gerçek zamanlı otonom sürüş senaryolarında doğruluk ve hız arasındaki dengeyi gözlemlemek amacıyla seçilmiştir.

- **1. faster_rcnn_R_50_FPN_3x.yaml**

Backbone: ResNet-50

Özellik Ağı: Feature Pyramid Network (FPN)

Eğitim Süresi: 3x (270k iterasyon)

Bu yapı, orta derinlikte bir ağ olan ResNet-50 üzerine kuruludur ve Feature Pyramid Network (FPN) ile daha farklı ölçeklerdeki nesnelerin tespitinde başarılı sonuçlar verir. 3x eğitim döngüsü, modelin daha fazla veri görerek daha iyi genelleme yapmasını sağlar. Bu nedenle hem doğruluk hem de stabilitet açısından dengeli bir yapı sunar.

Kullanım Alanı:

- Gerçek zamanlı sistemlerde doğruluk öncelikli olduğunda tercih edilir.
- Otonom araçlar gibi orta seviyede donanım desteği olan sistemlerde ideal bir çözümüdür.
- **2. faster_rcnn_R_50_FPN_1x.yaml**

Backbone: ResNet-50

Özellik Ağı: FPN

Eğitim Süresi: 1x (90k iterasyon)

Yine ResNet-50 ve FPN yapısını kullanan bu model, daha kısa sürede eğitilmiştir. Eğitim iterasyonu 1x olduğu için model daha hızlı prototip geliştirme ve sınırlı donanım kapasitesi gerektiren durumlar için uygundur. Ancak doğruluk seviyesi 3x versiyonuna kıyasla biraz daha düşüktür.

Kullanım Alanı:

- Hızlı prototipleme, araştırma testleri, düşük donanım koşulları
- Edge cihazlar veya ilk aşama testler
- **3. faster_rcnn_R_101_FPN_3x.yaml**

Backbone: ResNet-101

Özellik Ağı: FPN

Eğitim Süresi: 3x

ResNet-101, ResNet-50'ye kıyasla daha derin bir yapıya sahiptir ve bu sayede daha karmaşık özelliklerini öğrenebilir. Uzun süreli eğitim (3x) ile birleştiğinde modelin doğruluk seviyesi artar. Fakat bu gelişmişlik, daha yüksek işlem süresi ve donanım ihtiyacı anlamına gelir. Gerçek zamanlılık gereksinimi olan sistemlerde kullanımını sınırlı olabilir.

Kullanım Alanı:

- Doğruluğun ön planda olduğu senaryolar
- Offline analizler, gelişmiş veri setleri üzerinde ayrıntılı model değerlendirmesi
- Donanım kaynaklarının güçlü olduğu sistemler

4.2.3 Veri Seti Hazırlığı

Bu çalışmada nesne tespiti için kullanılan görüntüler, CARLA 0.9.13 sürümüne sahip açık kaynaklı otonom araç simülasyon ortamında elde edilmiştir. Simülasyon sırasında araçlar, autopilot modunda farklı senaryolar (gündüz/gece, yoğun trafik, açık/kent içi yollar) altında sürülerek çeşitli şehir haritalarında (Town01–Town05) sürülmüştür. Araç kamerasından elde edilen görüntüler belirli kare aralıklarıyla kaydedilmiş ve anlamlı sahnelerden oluşacak şekilde filtrelenmiştir. Toplanan görüntüler elle etiketlenmiş ve her bir nesneye ait sınıf ile konum bilgisi (bounding box) tanımlanmıştır. Etiketleme sürecinde yayalar, arabalar, bisikletler ve trafik levhaları gibi farklı nesne sınıfları dikkate alınmıştır. Etiketlenen veriler, Detectron2'nin uyumlu çalışabilmesi için COCO formatına dönüştürülmüştür. Bu dönüşüm sırasında, her bir görsele ait image_id, category_id, bbox ve segmentation gibi bilgileri içeren JSON dosyaları oluşturulmuştur.

4.2.4 Model Eğitim Stratejileri ve Parametreleri

(Training Strategies and Hyperparameters)

Bu projede, Detectron2 kütüphanesi altında yer alan Faster R-CNN tabanlı üç farklı model yapılandırması kullanılmıştır. Bu yapılandırmalar, Detectron2'nin Model Zoo arşivinden alınmış ve COCO veri seti formatına uygun olarak yeniden eğitilmiştir. Eğitim stratejileri aşağıdaki gibi özetlenebilir:

- **Öğrenme Oranı (Learning Rate):** Modeller, varsayılan olarak Detectron2'nin önerdiği base learning rate = 0.00025 değeriyle başlatılmış, eğitim süreci boyunca learning rate decay (azaltma) uygulanarak daha istikrarlı bir öğrenme sağlanmıştır. Bu strateji, overfitting (aşırı öğrenme) riskini azaltmayı hedeflemiştir.
- **Optimizer:** Model eğitiminde Stochastic Gradient Descent (SGD) algoritması kullanılmıştır. Momentum değeri 0.9, weight decay (ağırlık çürümesi) ise 0.0001 olarak ayarlanmıştır. Bu parametreler, öğrenme sürecini dengeleyerek daha sağlam bir konverjans sağlamıştır.
- **Batch Size:** Eğitim sırasında kullanılan IMS_PER_BATCH değeri 2 olarak belirlenmiştir. Bu değer, GPU bellek sınırları göz önünde bulundurularak optimize edilmiştir.
- **Iterasyon Sayısı:** Model konfigürasyonlarına göre farklı iterasyon sayıları uygulanmıştır:

faster_rcnn_R_50_FPN_1x.yaml: 90,000 iterasyon (1x schedule)

faster_rcnn_R_50_FPN_3x.yaml: 270,000 iterasyon (3x schedule)

faster_rcnn_R_101_FPN_3x.yaml: 270,000 iterasyon (3x schedule)

4.2.5 Zorluklar ve Karşılaşılan Problemler

Veri hazırlığı ve model eğitimi sürecinde çeşitli teknik ve pratik zorluklarla karşılaşmıştır. Bu zorluklar ve çözüm yolları aşağıda özetlenmiştir:

- **COCO Formatına Dönüşürme Süreci**: Roboflow platformu her ne kadar etiketleme sürecini kolaylaştırırsa da, verilerin COCO formatına uygun hale getirilmesi sırasında `image_id` ve `category_id` gibi alanlarda uyumsuzluk hataları oluşmuştur. Bu sorunlar, JSON dosyalarının elle düzenlenmesi ve doğrulama script'leriyle test edilmesi yoluyla giderilmiştir.
- **CUDA ve Detectron2 Uyum Problemleri**: Detectron2 kurulumu sırasında CUDA ve PyTorch sürümleri arasında uyumsuzluklar yaşanmıştır. Özellikle bazı Detectron2 sürümleri, sistemdeki mevcut CUDA sürümüyle uyumlu olmadığından dolayı model eğitimi başlamadan hata alınmıştır. Bu problem, Detectron2'nin uygun sürüm kombinasyonlarının araştırılması ve manuel kurulumlarla çözüme kavuşturulmuştur.
- **Eğitim Süresinin Uzunluğu**: Özellikle `faster_rcnn_R_101_FPN_3x` gibi daha derin yapılara sahip modellerde eğitim süresi oldukça uzun sürmüştür. Bu durum, GPU belleğinin yetersiz kalmasıyla birleşince eğitim süreçlerinde gecikmelere neden olmuştur. Bu sorun, batch size değerlerinin azaltılması ve eğitim sırasında geçici ağırlık dosyalarının daha sık kaydedilmesi ile aşılmıştır.

4.3 Yolo Yöntemleri

Nesne tespiti, otonom sürüş gibi gerçek zamanlı uygulamalar için kritik bir görevdir. Bu çalışmada, nesne tespiti için özellikle YOLO (You Only Look Once) mimarisi kullanılmıştır. YOLO, tek bir ağ ile hem nesne konumunu hem de sınıfını doğrudan tahmin ederek diğer yaklaşılara kıyasla yüksek hız ve doğruluk sunar. [4]

4.3.1 Neden YOLOv8 ile Başladık?

YOLO ailesi, zaman içinde birçok farklı versiyonla gelişmiştir. Bu çalışmada, YOLO modellerinin incelenmesine YOLOv8 ile başlanmıştır [17]. Bunun temel nedenleri şunlardır:

- **Güncel ve Gelişmiş Performans**: YOLOv8, önceki sürümlere kıyasla daha güncel bir mimariye sahip olup, genellikle daha iyi tespit doğruluğu ve daha verimli işlem kapasitesi sunar. Bu, başlangıç için güçlü ve rekabetçi bir temel sağlamıştır. [17], [18]
- **Kullanım Kolaylığı ve Topluluk Desteği**: Ultralytics tarafından geliştirilen YOLOv8, kullanıcı dostu bir arayüze ve kapsamlı dokümantasyona sahiptir. Geniş bir kullanıcı topluluğuna sahip olması, olası sorunların çözümünde ve bilgi paylaşımında önemli avantajlar sunar. [18]

- **İleriye Dönük Uyumluluk ve Karşılaştırma Temeli:** YOLOv8'den başlayarak YOLOv12'ye kadar olan sürümlerle ilerlemek, bu yeni nesil modellerin evrimini ve performans farklılıklarını sistematik bir şekilde incelememizi sağlamıştır. YOLOv8, bu karşılaştırma serisi için sağlam bir başlangıç noktası teşkil etmiştir. [18]

4.3.2 YOLO Sürümleri ve Basit Model Seçimi

Çalışma kapsamında, YOLOv8'den YOLOv12'ye kadar çıkan tüm ana YOLO sürümleri araştırılmış ve uygulanmıştır. Bu kapsamlı karşılaştırmanın yanı sıra, her bir YOLO modelinin en basit (nano/small) versiyonlarının seçilmesinin temel nedenleri şunlardır:

- **Kaynak Verimliliği:** Daha büyük modeller (medium, large) daha yüksek doğruluk potansiyeli sunsa da, eğitim ve çıkışım süreçlerinde çok daha fazla hesaplama kaynağı (GPU belleği, işlem süresi) gerektirirler. Projenin kaynak kısıtlamaları göz önüne alındığında, basit modeller daha ulaşılabilir bir karşılaştırma ortamı sağlamıştır. [19],
- **Gerçek Zamanlı Uygulama Odaklılık:** Otonom araçlar gibi gerçek zamanlı nesne tespiti gerektiren uygulamalarda, hız kritik bir faktördür. Basit modeller, genellikle daha az parametreye sahip oldukları için daha hızlı çıkışım yapabilirler [20]. Bu durum, modellerin gerçek dünya senaryolarındaki uygulanabilirliğini değerlendirmek için önemlidir.
- **Temel Performansın Anlaşılması:** Her modelin “temel” veya “varsayılan” performansını anlamak, ileriye dönük optimizasyonlar veya daha karmaşık modellerle çalışmadan önce önemlidir. En basit sürümler, mimarının çekirdek yeteneklerini ve verimliliğini ortaya koymada daha net bir resim sunar. [21]
- **Sistematik Karşılaştırma:** Farklı modellerin en basit versiyonlarını karşılaştırmak, sadece mimariler arasındaki temel farklara odaklanmamızı ve performansındaki artışların gerçekten model evriminden mi yoksa sadece parametre sayısındaki artıştan mı kaynaklandığını daha iyi anlamamızı sağlamıştır. [22], [23]

4.3.3 Model Eğitim Stratejileri ve Parametreleri

Bu bölümde, seçilen YOLO modellerinin eğitimi için kullanılan teknik detaylar sunulmuştur. Bu ayrıntılar, çalışmanın tekrarlanabilirliğini sağlamak ve elde edilen sonuçların bağlamını açıklamak için önemlidir.

4.3.3.1 Eğitim Ortamı

Modellerin eğitimi, **NVIDIA GeForce RTX 3060 Laptop GPU** (6144MiB bellek ile) kullanılarak **CUDA 11.8** tabanlı bir ortamda gerçekleştirilmiştir. Yazılım ortamı olarak **Python 3.11.9** ve **PyTorch 2.6.0+cu118** kullanılmıştır. **Ultralytics** kütüphanesinin **8.3.115 sürümü**, eğitim ve doğrulama süreçlerini yönetmek için kullanılmıştır.

4.3.3.2 Optimizasyon Algoritması

Model eğitimi için optimizasyon algoritması otomatik olarak belirlenmiştir (**optimizer=auto**). Sistem tarafından belirlenen algoritma **AdamW** olmuştur. Bu optimizasyon algoritması için başlangıç **öğrenme oranı (lr0) 0.001111** ve **momentum değeri 0.9** olarak ayarlanmıştır. **Ağırlık azaltma (weight decay)** değeri ise parametre gruplarına göre belirlenmiş olup, **0.0005** olarak uygulanmıştır. Bu parametreler, modelin eğitim sırasında kayıp fonksiyonunu etkin bir şekilde minimize etmesini sağlamak üzere otomatik olarak optimize edilmiştir.

4.3.3.3 Eğitim Döngüsü (Epochs) ve Batch Boyutu

Her bir model, **30 epoch** boyunca eğitilmiştir. Eğitim sürecinde kullanılan **batch boyutu 4** olarak belirlenmiştir. Bu, her bir eğitim adımda GPU belleğine 4 görüntünün yüklediği ve model ağırlıklarının bu 4 görüntüdeki hatalara göre güncellendiği anlamına gelir. Eğitim süreci boyunca **tek bir sınıf (single_cls=False)** yerine, veri kümesinde tanımlı 5 farklı sınıf (**Vehicle, Bike, Motorbike, Traffic Light, Traffic Sign**) için tespit yapılmıştır.

4.3.3.4 Öğrenme Oranı Çizelgeleme (Learning Rate Scheduling)

Başlangıç öğrenme oranı ve momentum değerleri için **warmup_epochs=1.0** ve **cos_lr=False** parametreleri kullanılmıştır. Bu durum, eğitimin ilk 1 epoch'unda öğrenme oranının kademeli olarak artırılarak “ısınma” (warmup) periyodu uygulanmıştır. Bu ısınma süreci, modelin eğitim başlangıcında büyük gradyan güncellemelerinden kaynaklanabilecek istikrarsızlıkları önlemeye ve daha kararlı bir yakınsama sağlamaya yardımcı olmaktadır.

5 Bulgular ve Tartışma (Results and Discussion)

5.1 Detectron2 Bulgular ve tartışma

Bu çalışmada, otonom araçlar için çevre algılamaya yönelik derin öğrenme tabanlı nesne tespiti amacıyla Detectron2 altyapısı kullanılmıştır. Faster R-CNN tabanlı üç farklı model — R_50_FPN_1x, R_50_FPN_3x ve R_101_FPN_3x — CARLA simülasyonundan elde edilen özel bir veri kümesiyle eğitilmiş ve karşılaştırılmıştır. Eğitim süresi ve model derinliği arttıkça doğruluk performansında anlamlı iyileşmeler gözlemlenmiştir. En iyi sonuçları R_101_FPN_3x modeli verirken, R_50_FPN_1x daha hızlı ancak daha düşük doğrulukta sonuçlar üretmiştir. Üç modelin temel metrik sonuçları aşağıdaki tabloda sunulmuştur.

Tablo 5.1: Object Detection Model Performance Metrics

Model	AP50:95 (AP)	AP50	AP75	AP_small	AP_medium	AP_large
Faster R-CNN R50-FPN_1x	48,707	77,286	46,987	21,985	51,993	84,065
Faster R-CNN R50-FPN_3x	51,48	81,512	44,21	25,734	52,734	81,512
Faster R-CNN R101-FPN_3x	52,401	91,426	45,971	32,421	53,631	91,426

Genel ortalama başarı (AP50:95) açısından en yüksek mAP değeri %52.40 ile R_101_FPN_3x modeline aittir.

R_50_FPN_3x modeli ise %51.48 ile bu başarıya oldukça yakın bir performans sergilemiştir.

Buna karşılık, R_50_FPN_1x modeli %48.70 mAP ile diğer modellerin gerisinde kalmıştır.

Bu durum, daha derin mimariye sahip ve daha uzun süre eğitilen modellerin (örneğin R_101, 3x eğitim süresi) daha başarılı sonuçlar verdiği ortaya koymaktadır.

IoU=0.50 eşiğinde ölçülen AP50 değerinde, R_101_FPN_3x modeli %91.42 ile çok yüksek bir başarı göstermiştir.

Bu değer, modelin nesneleri genel olarak doğru bölgelerde tespit etme yeteneğini öne çıkarır.

Kategori bazlı AP karşılaştırmalarına göre, “vehicle” sınıfında en yüksek başarıyı %70.72 ile R_50_FPN_1x modeli göstermiştir.

“Bike” ve “motobike” sınıflarında en başarılı model, sırasıyla %64.56 ve %54.82 AP ile R_101_FPN_3x olmuştur.

“Traffic light” ve “traffic sign” kategorilerinde ise en iyi sonuçlar R_50_FPN_3x modelinden gelmiş, AP değerleri sırasıyla %51.48 ve %53.01 olarak ölçülmüştür.

R_101_FPN_3x modeli, özellikle küçük nesne tespitinde (AP_small = %32.42) diğer modellerin önüne geçmiştir.

Bu durum, daha derin modellerin daha ayrıntılı görsel özelliklerini öğrenebilmesi sayesinde küçük nesneleri daha başarılı şekilde tespit edebildiğini göstermektedir.

Sonuç olarak, eğer yüksek doğruluk hedefleniyor ve karmaşık sahnelerde küçük nesne tespiti kritikse, Faster R-CNN R_101_FPN_3x modeli iyi tercih olacaktır.

Eğer daha hafif bir model isteniyor ama yine de makul bir doğruluk bekleniyorsa, Faster R-CNN R_50_FPN_3x dengeli en iyi bir çözüm sunar.

Hızlı ve basit bir prototipleme yapılacaksa, düşük doğruluk düzeyine rağmen R_50_FPN_1x modeli yeterli olabilir.

5.1.1 Model Tahminleri ile Gerçek Etiketlerin Karşılaştırılması

Test sürecinde elde edilen tahminler, gerçek etiketlemelerle birlikte aşağıda görsel olarak sunulmuştur. Bu sayede her modelin çevresel nesneleri ne kadar doğru tespit ettiği karşılaştırılmalı biçimde değerlendirilebilir.

- **Faster R-CNN R_50_FPN_1x:**



Şekil 5.1: Faster R-CNN R_50_FPN_1x modelinin test verisi üzerindeki tahmin sonuçları ile gerçek etiketlerin karşılaştırılması

- Faster R-CNN R_50_FPN_3x:



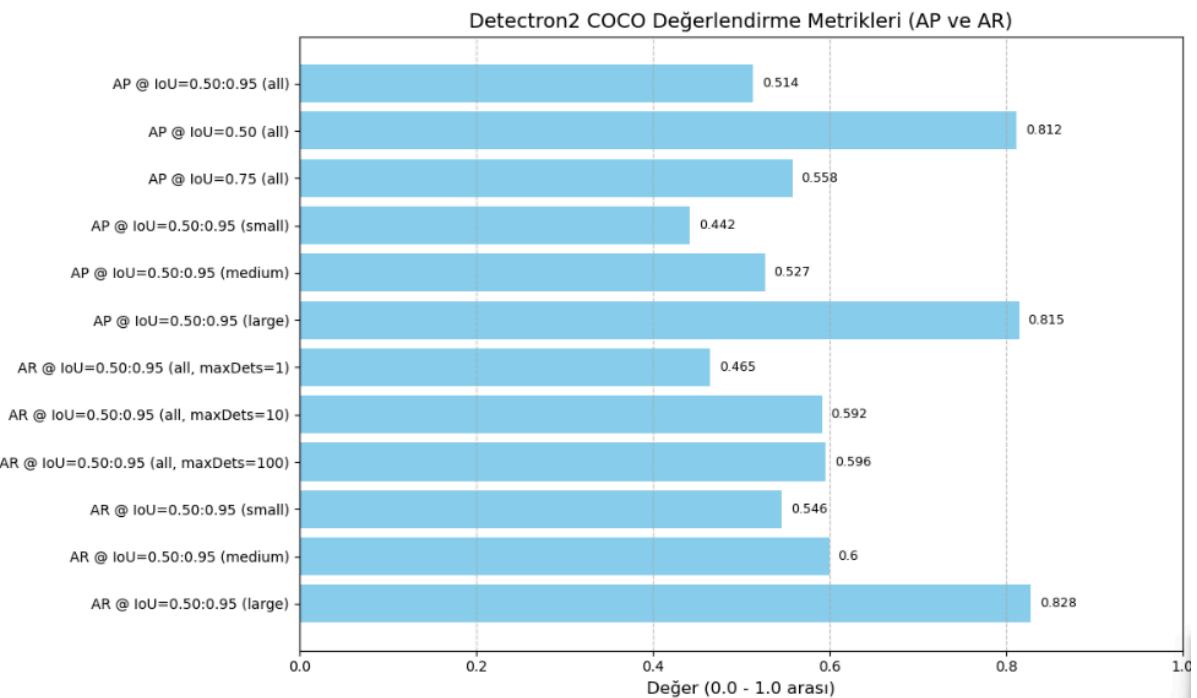
Şekil 5.2: Faster R-CNN R_50_FPN_3x modelinin test verisi üzerindeki tahmin sonuçları ile gerçek etiketlerin karşılaştırılması

- Faster R-CNN R_101_FPN_3x:



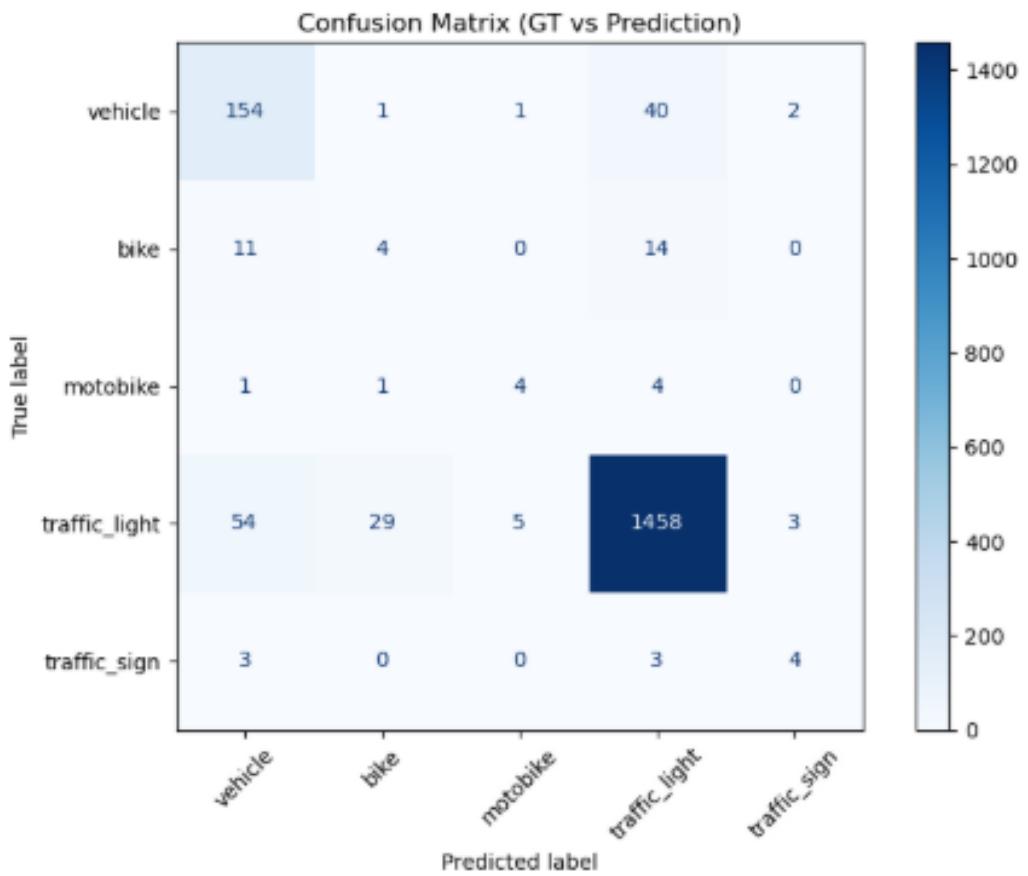
Şekil 5.3: Faster R-CNN R_101_FPN_3x modelinin test verisi üzerindeki tahmin sonuçları ile gerçek etiketlerin karşılaştırılması

5.1.2 Model Performans Grafikleri



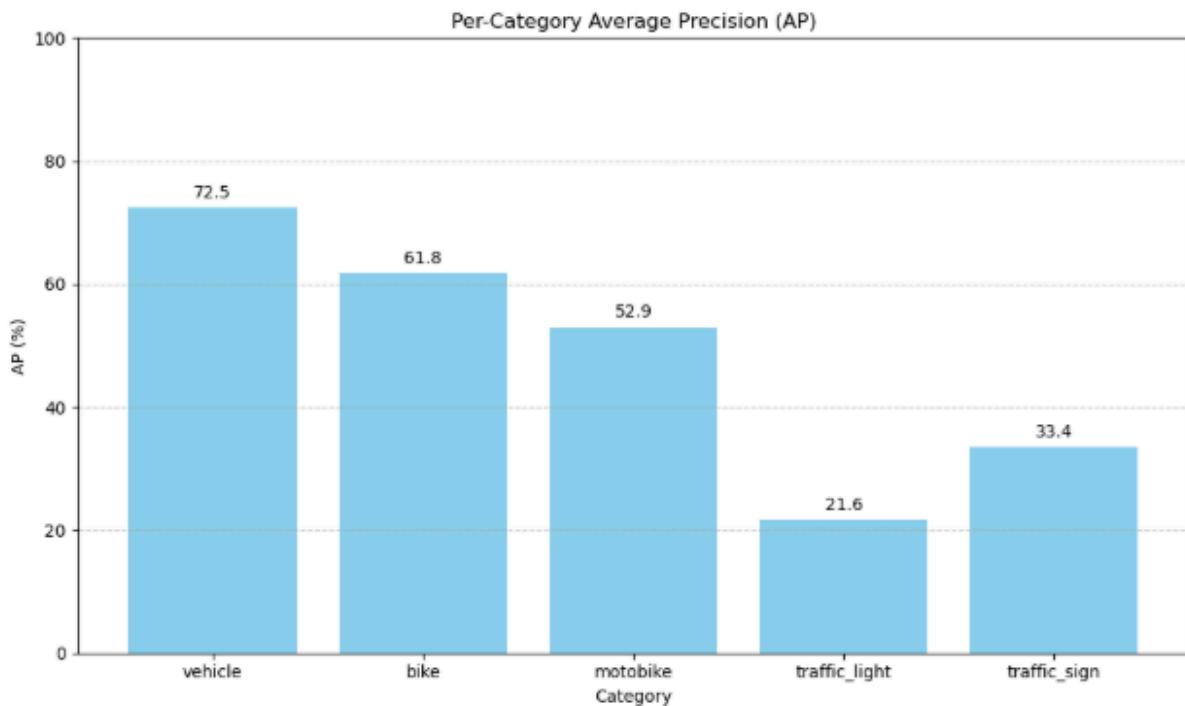
Şekil 5.4: Faster R-CNN R_101_FPN_3x modelinin performans metriklerinin grafiği

Detectron2 ile eğitilen model, genel olarak nesne algılama görevinde yeterli bir performans sergilemektedir. Özellikle büyük nesneleri ve %50 IoU eşiginde nesnelerin varlığını tespit etmede çok başarılıdır. Modelin uzamsal hassasiyeti (AP75) de kabul edilebilir düzeydedir. Ancak, modelin en belirgin zayıf noktası küçük boyutlu nesnelerin algılanmasıdır (AP_small). Bu durum, modelin küçük nesneler üzerindeki performansını artırmak için daha fazla optimizasyon (örneğin, daha yüksek çözünürlüklü girişler, küçük nesnelere özel veri artırma teknikleri veya FPN mimarisi üzerinde iyileştirmeler) gerektirebileceğini düşündürmektedir.



Şekil 5.5: Faster R-CNN R_101_FPN_3x modelinin karmaşıklık matrisi

Detectron2 modeliniz, trafik lambalarını mükemmel bir şekilde tespit ederken, araçlarda da iyi bir başarı göstermektedir. Ancak, bisiklet, motosiklet ve trafik işaretleri gibi sınıflarda ciddi algılama zorlukları yaşamaktadır, bu nesneleri sıkılıkla diğer sınıflarla karıştırma eğilimindedir. Bu zayıf performans gösteren sınıflar için veri artırma veya model iyileştirmeleri kritik önem taşımaktadır.



Şekil 5.6: Faster R-CNN R_101_FPN_3x modelinin sınıf bazında ortalama hassasiyet (AP) performansı

Faster R-CNN R_101_FPN_3x modeli, “Vehicle” sınıfında güçlü bir algılama performansı sergilemektedir. “Bike” sınıfında da kabul edilebilir bir başarıya sahiptir. Ancak, modelin “Motobike”, “Traffic_light” ve “Traffic_sign” gibi sınıfları algılamada belirgin zorluklar yaşadığı ve bu alanlarda iyileştirmelere ihtiyaç duyduğu açıklar. Bu sınıf bazındaki performans farklılıklarını, modelin eğitim veri setinin dağılımı, sınıfların görsel karmaşaklılığı veya boyut varyasyonları gibi faktörlerden kaynaklanabilir. Bu zayıf performans gösteren sınıflara yönelik daha fazla veri artırma veya model optimizasyonları, genel algılama kalitesini artırmak için faydalı olabilir.

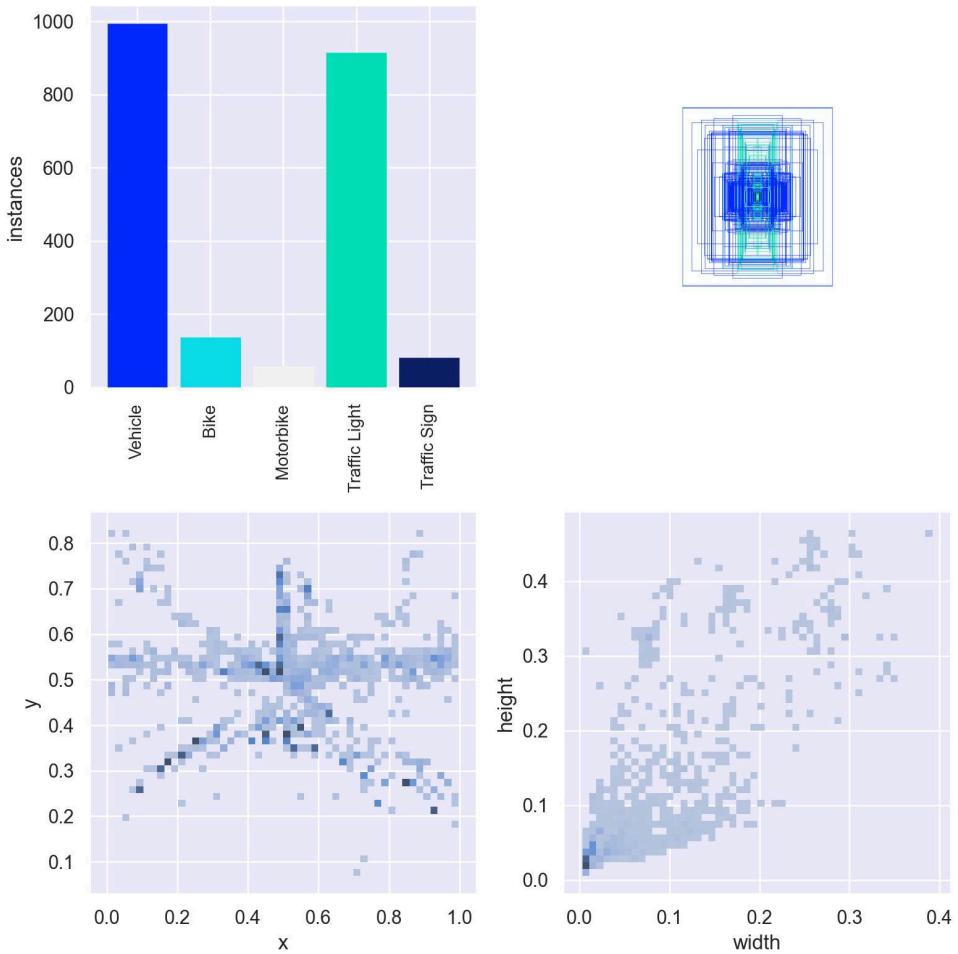
5.2 Yolo Bulgular ve tartışma

Bu bölümde, çalışmamız kapsamında YOLO mimarisi ile gerçekleştirilen nesne tespiti deneylerinin sonuçları detaylı olarak sunulmakta ve bu sonuçlar tartışılmaktadır

5.2.1 YOLOv8-YOLOv12 Sürümeleri Üzerindeki Bulgular ve Karşılaştırmalar

Çalışmanın ilk aşamasında, YOLOv8'den başlayarak YOLOv12'ye kadar olan güncel YOLO sürümlerinin Carla Object Detection Dataset üzerinde sergilediği performans incelenmiştir. Bu inceleme, her bir modelin temel yeteneklerini ve mimarının evrimini anlamak amacıyla en basit (nano/small) versiyonları kullanılarak gerçekleştirılmıştır. Yolov9'un diğer modellere göre (YOLOv8'den başlayarak YOLOv12'ye kadar) Precision ve Inference daha iyi çıktıği için, Yolov9 modülünün diğer sürümleri de (en basitten en karmaşağa doğru) test edilip karşılaştırılmıştır.

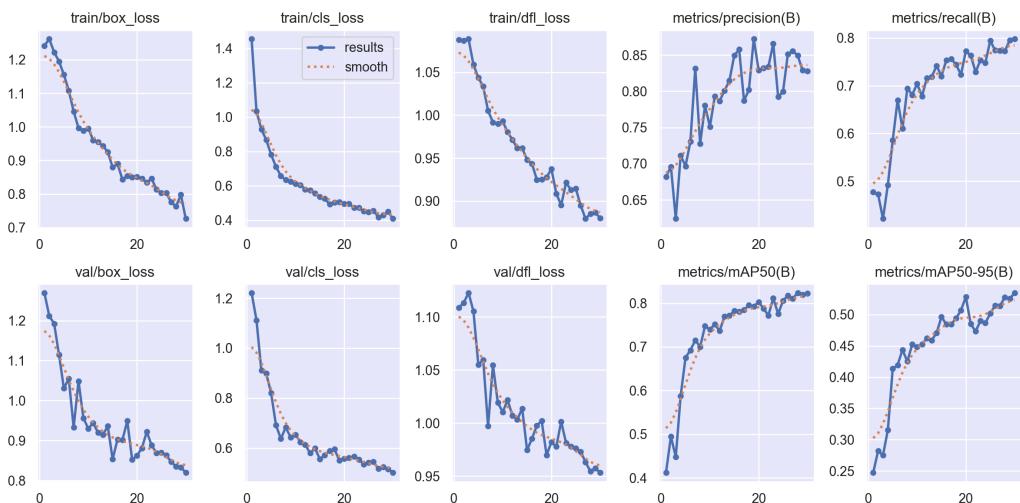
5.2.2 Model Performans Grafikleri



Şekil 5.7: Yolov8'e göre veri kümelerindeki her bir nesne sınıfının dağılımı

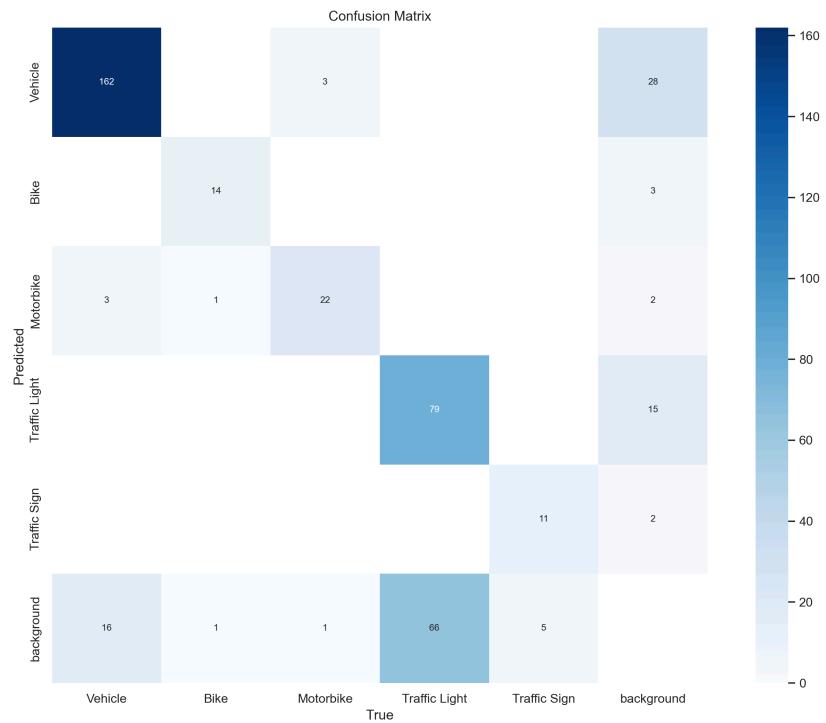
Şekil 7'den veri kümelerinde sınıf dengesizliği olduğu açıkça görülüyor. Bazı sınıfların ("Traffic light" "Vehicle") çok sayıda örneği varken, bazı sınıfların ("Bike", "Motorbike", "Traffic Sign") daha az sayıda örneği olduğu görülmüyor.

5.2.2.1 YOLOv9s

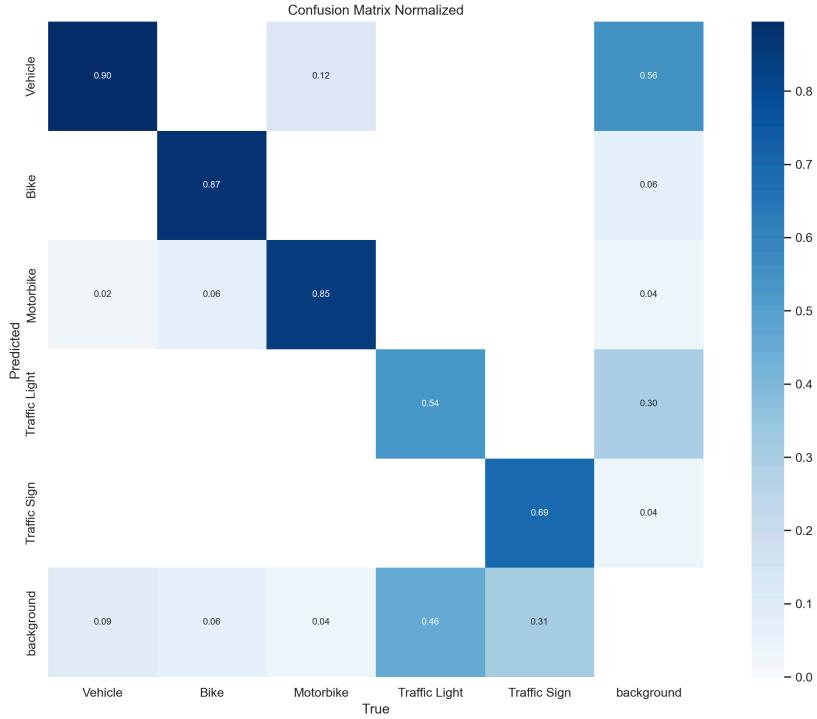


Şekil 5.8: Eğitim Sonuçları ve Kayıp Grafikleri

- Şekil 8'de eğitim süresince hem kutu (box_loss), sınıflandırma (cls_loss) hem de DFL (dfl_loss) kayıpları tutarlı ve hızlı bir şekilde düşüş göstermiştir. Bu, Yolov9s'in eğitim veri kümelerindeki nesnelerin konumlarını, sınıflarını ve boyut dağılımlarını doğru bir şekilde öğrenmede büyük ilerleme kaydettiğini göstermektedir.
- Ortalama ortalama hassasiyet (mAP50) ve genel ortalama hassasiyet (mAP50-95) değerleri eğitim ilerledikçe sürekli olarak artmış ve yüksek seviyelerde kararlılık göstermiştir. Bu, Yolov9s'in genelleme yeteneğinin yüksek olduğunu ve doğrulama seti üzerinde başarılı bir performans sergilediğini kanıtlar.

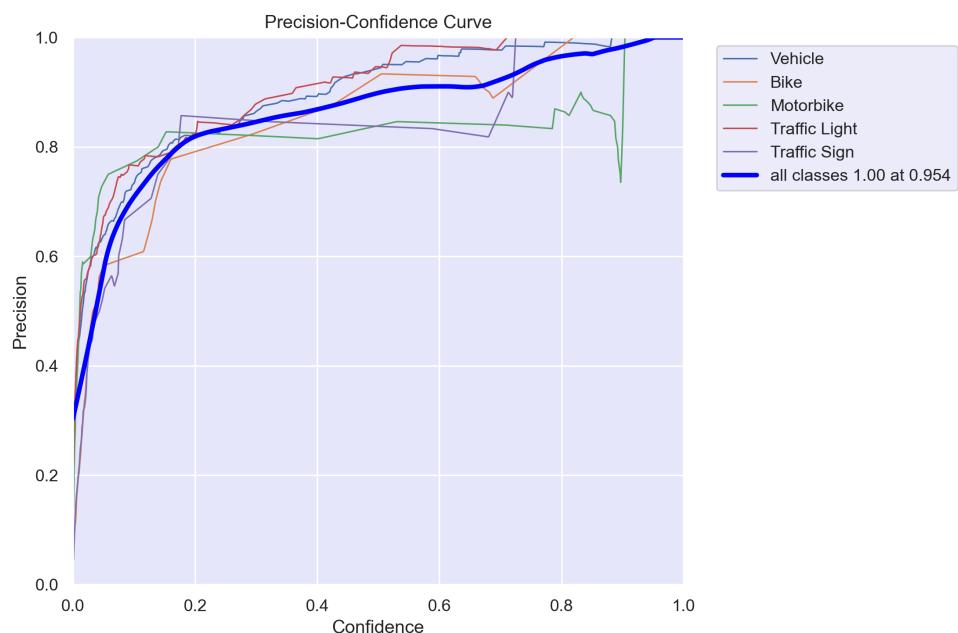


Şekil 5.9: karışıklık Matrisi



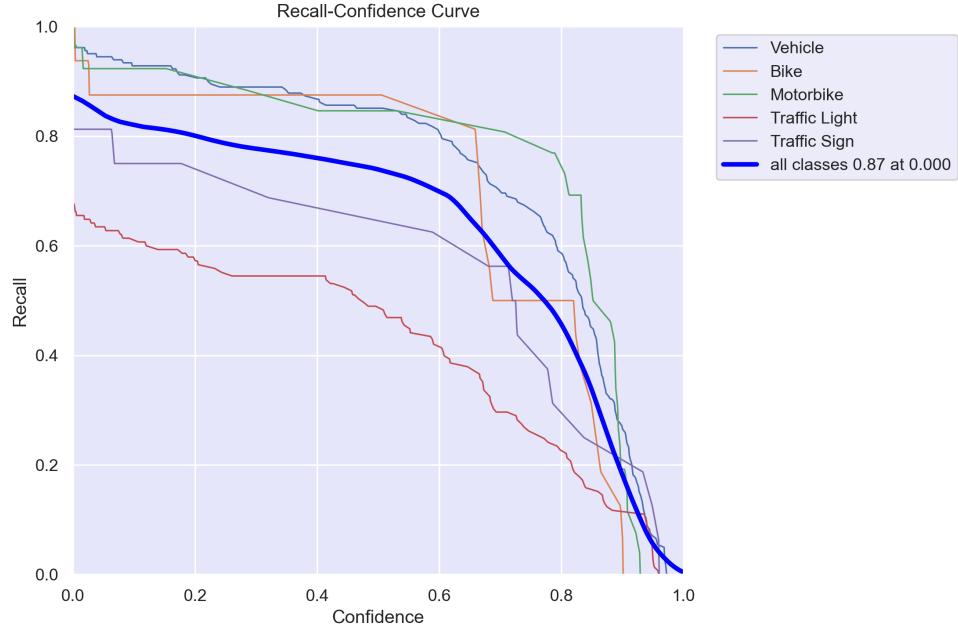
Şekil 5.10: Normalize edilmiş karışıklık matrisi

- Şekil 9 ve Şekil 10'de ana köşegenindeki değerlerin (0.90 ve üzeri) çok yüksek olması, Yolov9s'in sınıfların büyük çoğunluğunu yüksek bir doğrulukla doğru bir şekilde tespit ettiğini kesin olarak göstermektedir.
- “Traffic Light” sınıfı için diagonal dışındaki küçük değerler (örneğin “Traffic Sign” ile karmaşma oranı) gözlemlenmiştir. Bu, yolov9s'in bu iki sınıfı çok düşük bir oranda da olsa bazen birbirile karşıştırdığını ve bu sınıf için potansiyel olarak ek veri veya odaklanmış iyileştirme alanlarının olabileceğini belirtir.



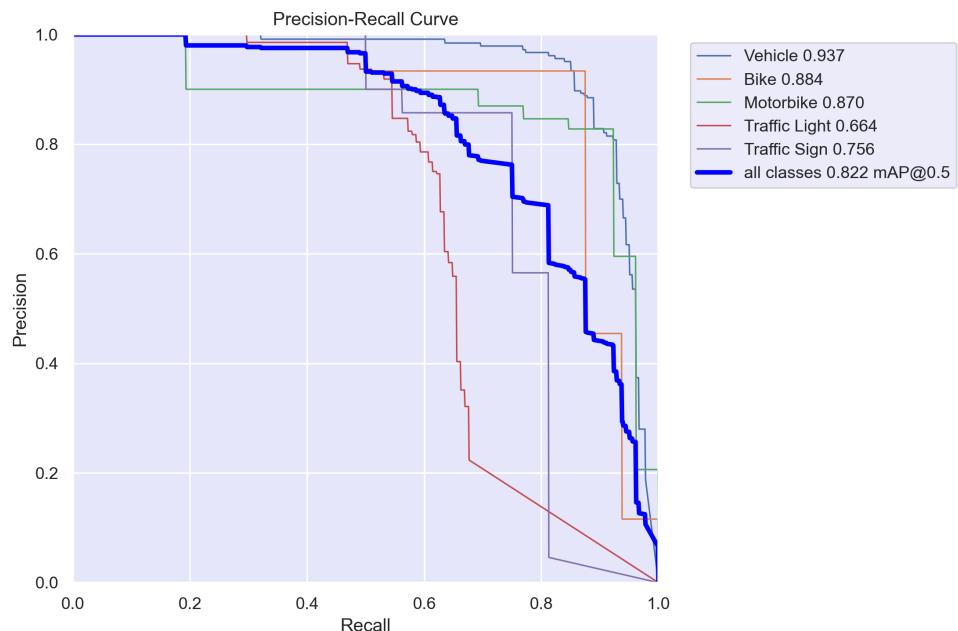
Şekil 5.11: Modelin Hassasiyet (Precision) Eğrisi

- Şekil 11'de tüm sınıflar için hassasiyet eğrisi, yüksek güven eşiklerinde çok yüksek hassasiyet değerleri göstermiştir. Bu, Yolov9s'in yaptığı pozitif tahminlerin büyük çoğunluğunun doğru olduğunu ve yanlış pozitif (hatalı tespit) oranının son derece düşük olduğunu göstermektedir.



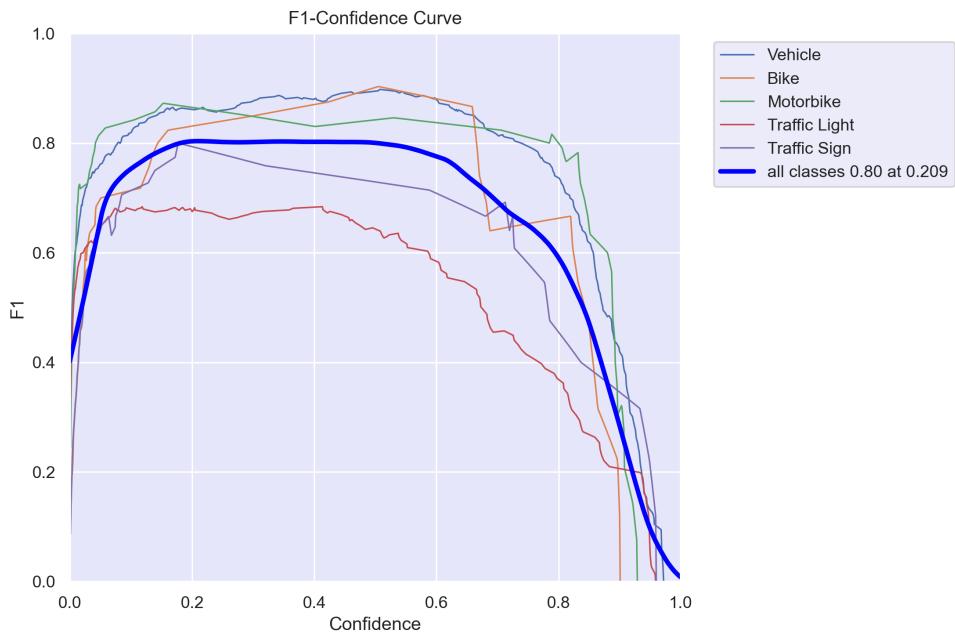
Şekil 5.12: Tüm Sınıflar için Geri Çağırma Performansı Eğrisi

- Bu Şekil 12'de, tüm sınıflar için geri çağrıma eğrileri, düşük güven eşiklerinde çok yüksek geri çağrıma değerlerine ulaşmıştır. Bu, Yolov9s'in veri kümelerindeki mevcut gerçek nesnelerin büyük bir kısmını başarıyla tespit edebildiğini ve yanlış negatif (gözden kaçırma) oranının düşük olduğunu göstermektedir.



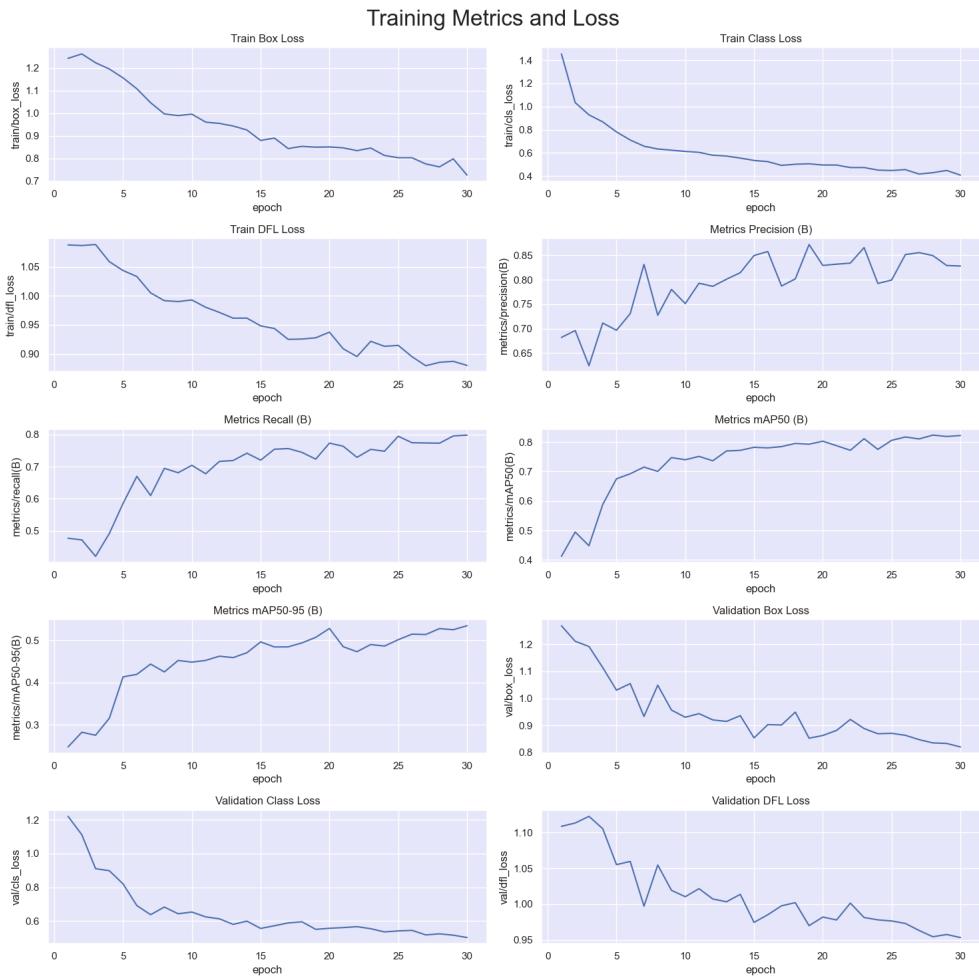
Şekil 5.13: Modelin Sınıf Bazında Hassasiyet ve Geri Çağırma Dengesi

- “Vehicle”, “Bike” ve “Motorbike” sınıfları için eğrilerin alanları (AP değerleri) oldukça yüksek ve Şekil 13’ün sağ üst köşesine yakın konumlanmıştır. Bu, Yolov9s’ın bu sınıflarda hem yüksek hassasiyet hem de yüksek geri çağrıma ile mükemmel bir performans sergilediğini gösterir.
- “Traffic Light” ve “Traffic Sign” sınıflarının AP değerleri de yüksek olmakla birlikte, diğer baskın sınıflara göre hafifçe daha düşüktür. Bu, yine de iyi bir performans sergilediklerini ancak diğer sınıflara kıyasla küçük bir performans farkı olduğunu belirtir.



Şekil 5.14: F1 Skoru Eğrisi Yolov9s için

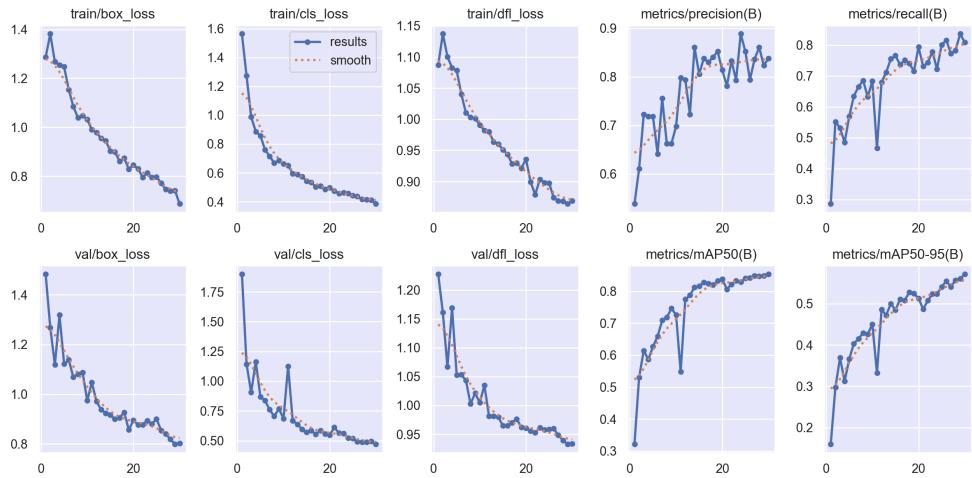
- Şekil 14’de Yolov9s, hassasiyet ve geri çağrıma arasında mükemmel bir denge kurmuştur; yüksek F1 skorları, Yolov9s’ın genel performansının güçlü olduğunu gösterir.



Şekil 5.15: Eğitim Süreci Boyunca mAP, Hassasiyet, Geri Çağırma ve Kayıp Değişimi

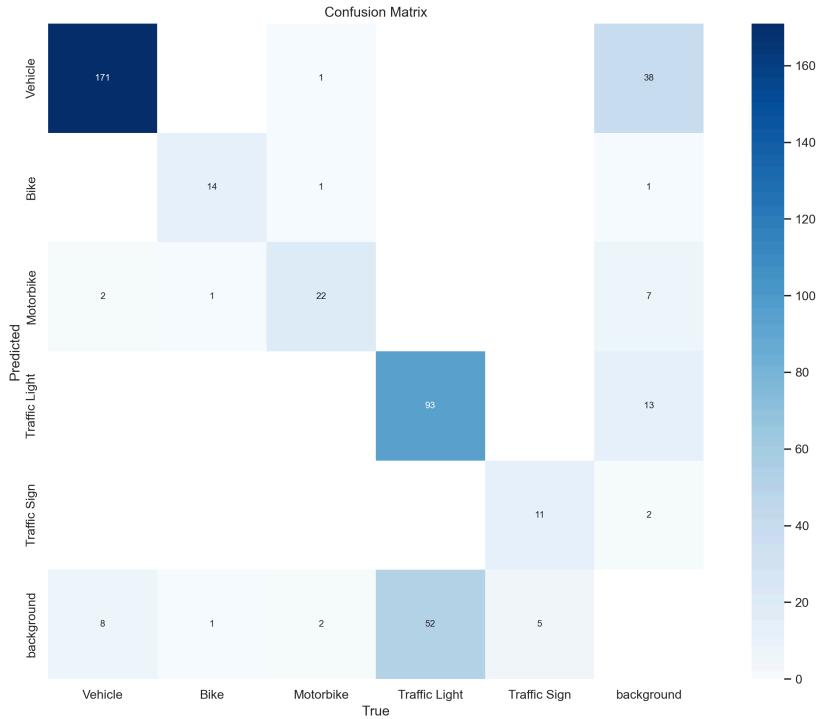
- Bu Şekil 15'de önceden eğitilmiş ağırlıklarla başlatılmış olup, belirli bir veri kümesi (Carla Traffic Sign Detection için yapılandırılmış) üzerinde nesne tespiti görevi için 30 epokluk bir eğitime başlamıştır. Eğitim süreci, 640x640 görüntü boyutu ve 4'lük toplu iş (batch) boyutu gibi optimize edilmiş temel parametrelerle kararlılık ve verimlilik hedeflenerek tasarlanmıştır. Erken durdurma (50 sabır epoku) ve deterministik eğitim gibi mekanizmaların aktif olması, Yolov9s'in aşırı öğrenmeye karşı korunmasını ve sonuçların tekrarlanabilirliğini sağlamayı amaçlamaktadır.

5.2.2.2 YOLOv9c



Şekil 5.16: Eğitim Sonuçları ve Kayıp Grafikleri

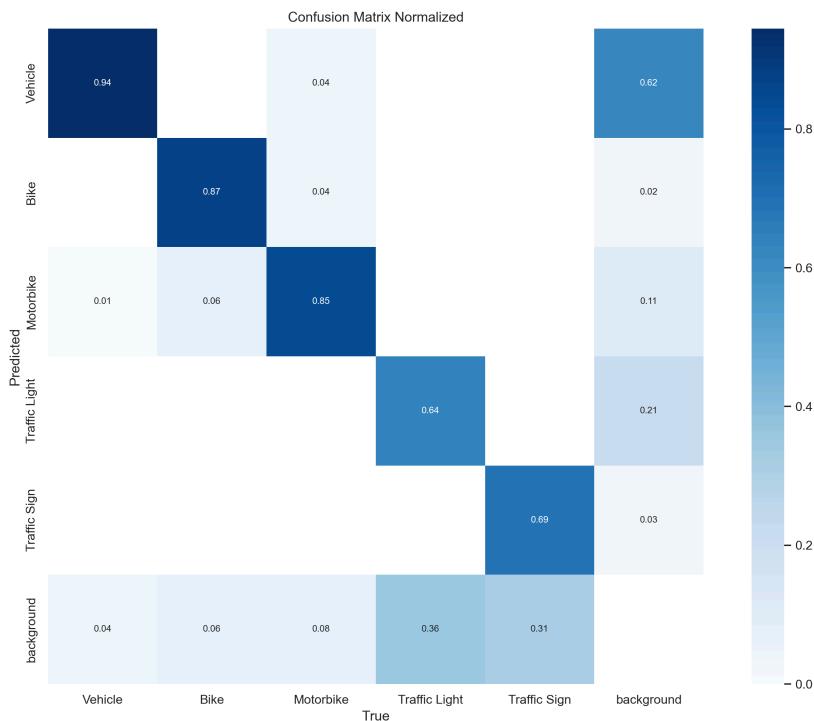
- Bu Şekil 16'de YOLOv9c Yolov9c'nin eğitim sürecinin genel olarak başarılı olduğunu gösteriyor. Tüm kayıp değerleri düşüyor, hassasiyet, geri çağrıma ve mAP değerleri ise artıyor. Doğrulama setindeki performansın da benzer eğilimleri göstermesi, Yolov9c'in iyi bir şekilde genellenebilir olduğunu ve aşırı öğrenme eğilimi göstermediğini gösteriyor.



Şekil 5.17: karışıklık Matrisi

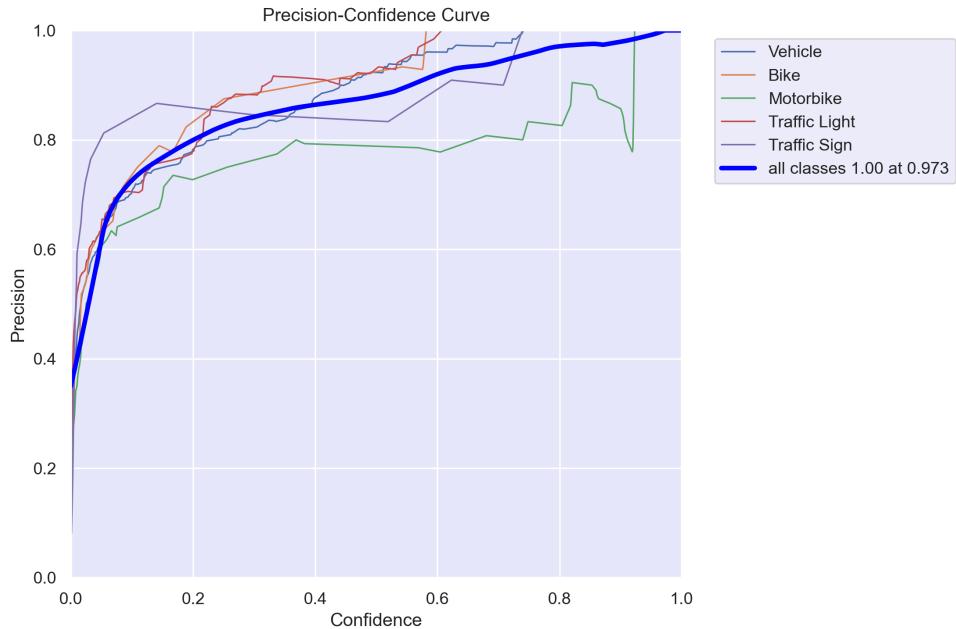
- Şekil 17'de, "Vehicle" ve "Traffic Light" sınıflarında oldukça iyi performans gösteriyor. Ancak, "Bike", "Motorbike" ve özellikle "Traffic Sign" sınıflarında Yolov9c'nin performansı daha zayıf. "Bike" ve "Traffic Sign" için doğru tahmin sayıları oldukça düşük.

- “background” sınıfını da karıştırma eğilimi var. Özellikle “Vehicle” ve “Traffic Light” nesnelerinin önemli bir kısmını “background” olarak sınıflandırmış, bu da recall (geri çağırma) değerlerini düşürecektilir.
- Bazı sınıflar arasında karışıklık var: Örneğin, “Bike”lar “Vehicle” ve “Motorbike” ile, “Traffic Sign”lar ise “Traffic Light” ile karıştırılabilir.



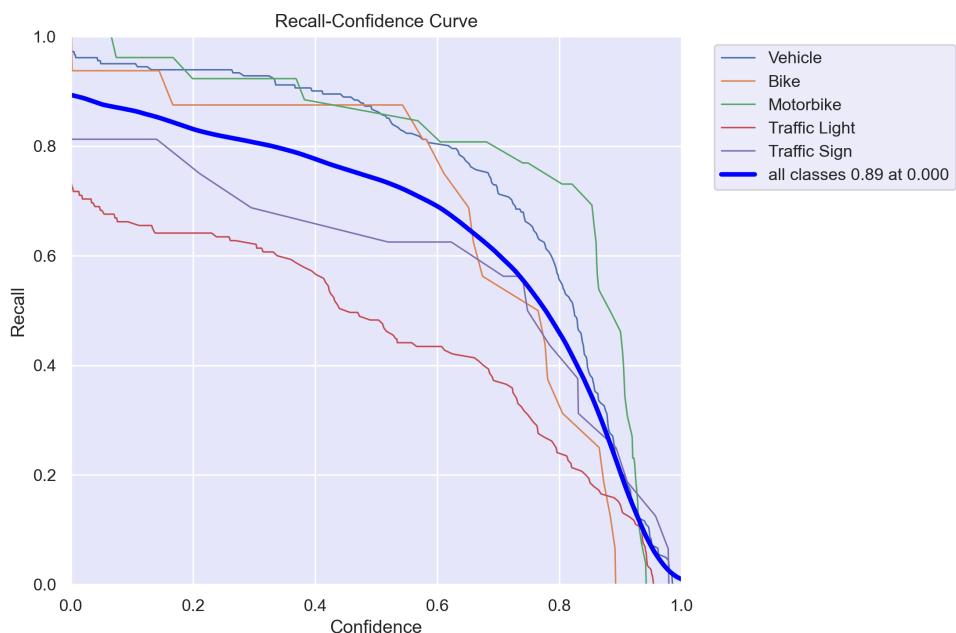
Şekil 5.18: Normalize edilmiş karışıklık matrisi

- Şekil 18'de “Vehicle” sınıfı hala en iyi performansı gösteriyor. “Bike” ve “Motorbike” sınıfları için performans fena değil (%87 ve %85), ancak iyileştirme alanı var. “Traffic Light” ve “Traffic Sign” sınıfları için geri çağırma (recall) oranları sırasıyla %64 ve %69 ile nispeten düşük.



Şekil 5.19: Modelin Hassasiyet (Precision) Eğrisi

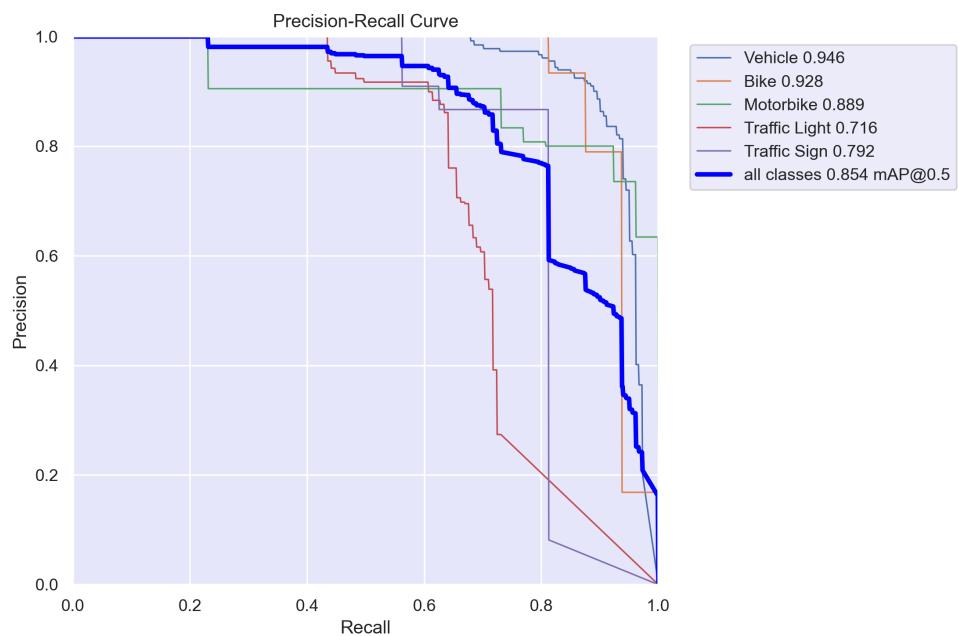
- Şekil 19 tüm sınıflar için eğriler, güven eşiği arttıkça hassasiyetin genel olarak arttığını gösteriyor. Özellikle yüksek güven eşiklerinde hassasiyet 1.0'a (mükemmel) yaklaşıyor. "All classes" (kalın mavi çizgi), 0.973 güven eşigidinde hassasiyetin 1.0 olduğunu gösteriyor. Bu, Yolov9c'nin çok yüksek güvenle yaptığı tahminlerin neredeyse hepsinin doğru olduğu anlamına gelir.
- Düşük güven eşiklerinde (örneğin 0.2'nin altında) hassasiyette belirgin düşüşler görülüyor, bu da Yolov9c'nin daha az emin olduğu tahminlerde daha fazla hata yapabileceğini gösterir.



Şekil 5.20: üm Sınıflar için Geri Çağırma Performansı Eğrisi

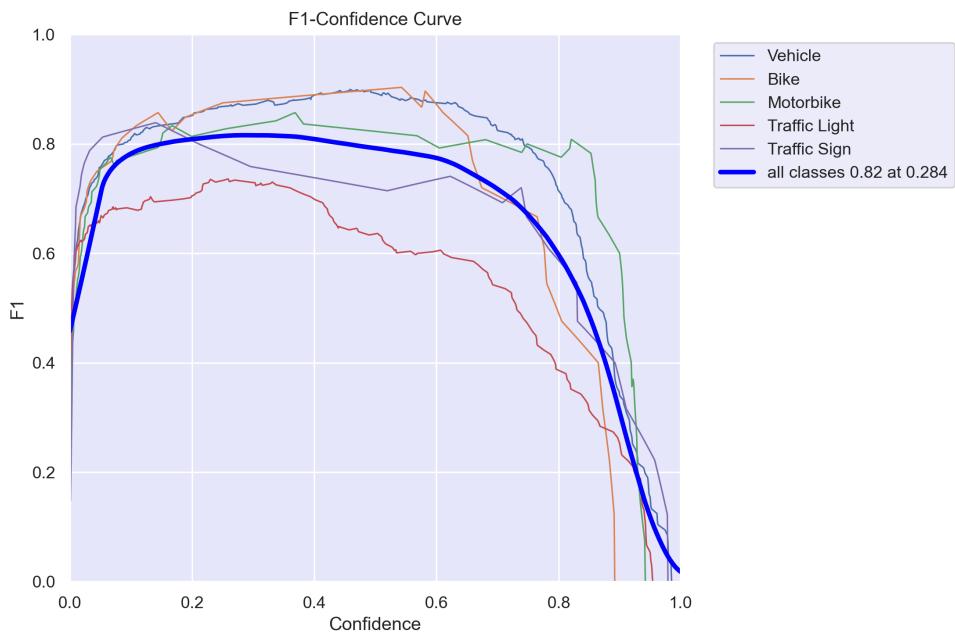
- R_curve9c'de tüm sınıflar için eğriler, güven eşiği arttıkça geri çağrımanın genel olarak azaldığını gösteriyor.

- “All classes” (kalın mavi çizgi), güven eşiği 0.0’da (yani tüm tahminler kabul edildiğinde) geri çağrımanın 0.89 olduğunu gösteriyor. Bu, gerçek nesnelerin büyük bir kısmının Yolov9c tarafından düşük güvenle bile olsa tespit edilebilğini gösterir.
- “Vehicle” ve “Bike” sınıfları için geri çağrıma eğrileri diğerlerine göre daha yüksek seviyelerde kalıyor, bu da bu sınıflardaki nesnelerin daha kolay tespit edildiğini gösterir.
- “Traffic Light” sınıfının geri çağrıma eğrisi (kırmızı) diğerlerine göre daha düşük seviyelerde seyrediyor ve daha hızlı düşüyor. Bu, Yolov9c’nin “Traffic Light” nesnelerini diğerlerine göre daha sık gözden kaçardığını (yani yanlış negatif ürettiğini) gösterir.



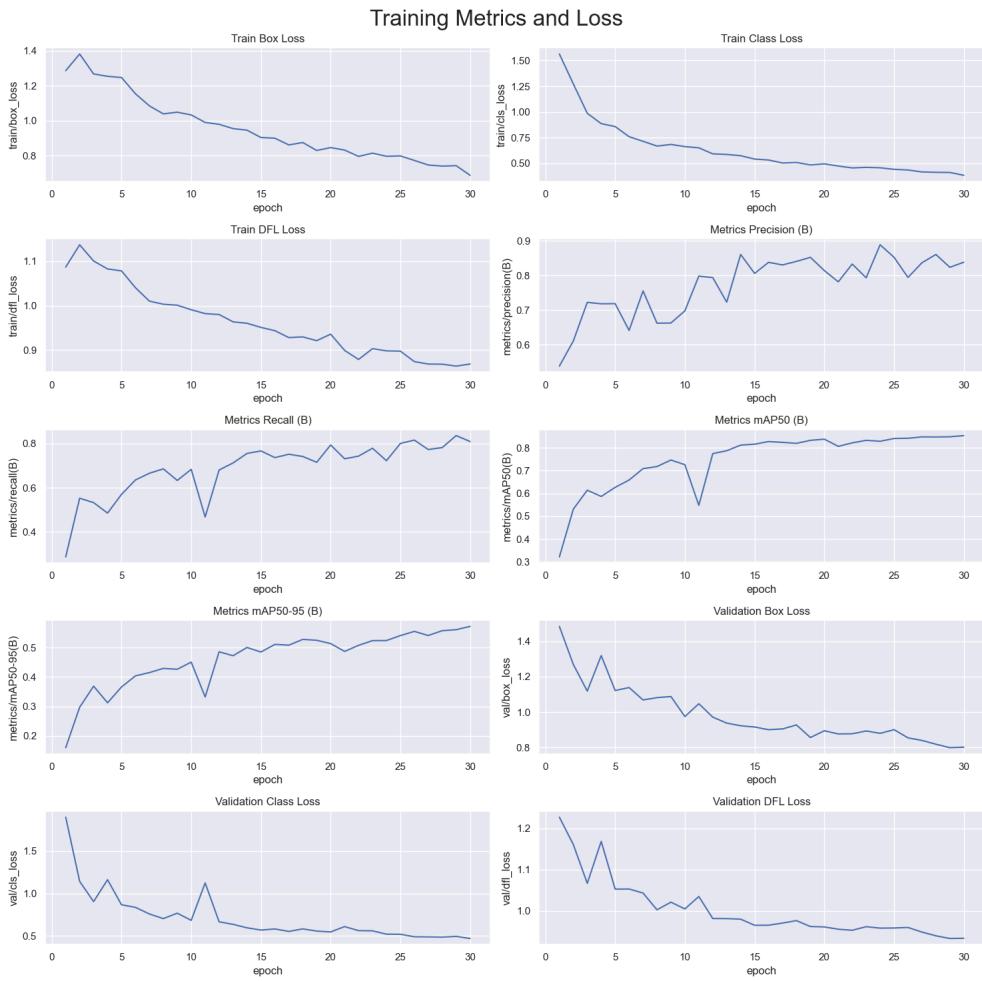
Şekil 5.21: Modelin Sınıf Bazında Hassasiyet ve Geri Çağırma Dengesi

- Şekil 21’de, Yolov9c “Vehicle”, “Bike” ve “Motorbike” algılamasında oldukça başarılı. “Traffic Sign” ve özellikle “Traffic Light” sınıfları için Yolov9c’nin performansını artırmak adına daha fazla iyileştirme yapılması gerekiyor. Bu, daha fazla eğitim verisi, veri artırma teknikleri veya Yolov9c mimarisinde ayarlamalar gerektirebilir.



Şekil 5.22: F1 Skoru Eğrisi Yolov9s için

- Şekil 22'de eğriler, belirli bir güven eşiginde (yaklaşık 0.2-0.3 arasında) F1-skorunun zirve yaptığını, bu eşliğin altında ve üstünde ise düşüğünü gösteriyor.
- “All classes” (kalın mavi çizgi), 0.284 güven eşiginde 0.82 F1-skoru ile zirve yapıyor. Bu, genel olarak Yolov9c için en iyi performans noktasını temsil eden güven eşigidir.
- “Vehicle”, “Bike”, “Motorbike” sınıfları için F1 eğrileri daha yüksek zirvelere ulaşırken, “Traffic Light” ve “Traffic Sign” için zirveler daha düşüktür. Özellikle “Traffic Light” (kırmızı çizgi) belirgin şekilde daha düşük bir F1 performansı sergiliyor.



Şekil 5.23: Eğitim Süreci Boyunca mAP, Hassasiyet, Geri Çağırma ve Kayıp Değişimi

5.2.3 Sonuçların Tablolu Karşılaştırması:

Tablo 5.2: YOLO Test Results Tablosu

Model	Precisi-on(B)	Recall(B)	mAP50(B)	mAP50-95(B)	Inference(ms)
YOLOv8n	0.71	0.43	0.47	0.31	4.83
YOLOv9t	0.86	0.44	0.52	0.34	5.23
YOLOv9s	0.78	0.52	0.64	0.41	6.40
YOLOv9m	0.73	0.60	0.66	0.41	7.65
YOLOv9c	0.76	0.60	0.67	0.45	8.39
YOLOv9e	0.72	0.55	0.58	0.37	18.52
YOLOv10n	0.66	0.47	0.46	0.30	3.65
YOLOv11n	0.80	0.38	0.47	0.30	3.88
YOLOv12n	0.81	0.43	0.50	0.33	4.35

- Tablo 2'a göre:

- **mAP50-95: Gerçek Performans Kriteri:**
- YOLOv9c: $mAP50-95 = 0.4517$ (en yüksek doğruluk)

- YOLOv9s: mAP50-95 = 0.4144 (ikinci en yüksek doğruluk)
- **Inference Süresi: Gerçek Zamanlılık:**
- YOLOv9s: 6.39 ms (hızlı)
- YOLOv9c: 8.38 ms (orta seviye hız)
- Eğer bir model hem daha hızlı hem de daha doğruysa, başka modellere kıyasla avantajlıdır. Bu noktada YOLOv9s öne çıkıyor çünkü:
 - YOLOv9m modeline çok yakın doğrulukta (0.4144 vs 0.4517) Ama daha hızlı (6.39 ms vs 8.38 ms)
 - Bir arabayı seçerken sadece maksimum hızı (doğruluk) değil, aynı zamanda yakıt verimliliğini (inference süresi) de düşünürüz. YOLOv9s, hem hızlı hem de az yakıt tüketiyor. YOLOv9c ise daha büyük motor gücüne (doğruluk) sahip ama biraz daha çok yakıyor.
- **Precision ve Recall:**
 - YOLOv9s
 - Precision: 0.7807 (nispeten iyi)
 - Recall: 0.5150 (daha fazla nesne tespit ediyor)
 - YOLOv9c:
 - Precision: 0.7590
 - Recall: 0.6042 (daha fazla nesne kaçırmadan buluyor)
 - **Sonuç:** YOLOv9s hem Yüksek doğruluk (mAP50-95: 0.4144) ve Çok hızlı (6.39 ms) bakımında dengeli sonuç sağladığı için en iyi model olarak seçilmiştir. Gerçek zamanlı sistemlerde harika çalışır. YOLOv9c En yüksek mAP50-95 değeri (0.4517) ve Yüksek Recall (0.6042) daha az nesne kaçırıyor. Orta seviye hız ile detaylı ve güvenli analiz gereken sistemler için idealdir.

5.2.3.1 CARLA Simülasyon Kısıtlamaları, Yeni model denemeleri Ve Çözüm Yolları

Her ne kadar yapılan ön karşılaştırmalı analizlerde **YOLOv9s**, Carla simülatörü için en iyi model adayı olarak belirlenmiş olsa da, CARLA simülasyon ortamının **Python 3.7** kısıtlaması nedeniyle **YOLOv9s** gibi daha yeni modellerin doğrudan entegrasyonu ve kullanımı mümkün olmamıştır. Bu durum, seçilen ideal modeli gerçek zamanlı simülasyon ortamında test etme planlarımızı değiştirmemize neden olmuştur.

Bu kısıtlamayı aşmak amacıyla, **Python 3.7** ile uyumlu olan **YOLOv5** mimarisine geçiş yapılmıştır. Başlangıçta **Ultralytics kütüphanesi** ile **YOLOv5** modellerinin eğitimi gerçekleştirilmiş, ancak bu modellerin CARLA ortamında **torch.hub.load** kaynaklı entegrasyon hataları nedeniyle yüklenemediği görülmüştür. Bu sorunu çözmek için alternatif bir strateji izlenmiştir: **YOLOv5'in** resmi GitHub deposu kullanılarak ve **Python 3.7** ile uyumlu olan **yolov5x.pt** önceden eğitilmiş modeli indirilerek, kendi özel veri kümemiz (Carla Object Detection Dataset) üzerinde yeniden eğitim (fine-tuning) gerçekleştirilmiştir. Bu yaklaşım, CARLA ortamında gerçek zamanlı nesne tespiti için stabil ve uyumlu bir çözüm sunmuştur.

5.2.3.2 YOLOv5x Modelinin Simülasyona Entegrasyonu

```
# Eğitilmiş YOLOv5x modelini yükle
model = torch.hub.load('C:/CARLA_0.9.15/yolov5', 'custom',
                       path='C:/CARLA_0.9.15/Yolo_train/carla_yolov5x/
weights/best.pt',
                       source='local')
model.conf = 0.5 # minimum güven skoru
model.iou = 0.45 # NMS eşiği
model.classes = None # Tüm sınıflar kullanılacak
model.eval() # inference moduna al

torch.hub.load() fonksiyonu kullanılarak, ince ayar yapılmış yolov5x/weights/best.pt model dosyası yerel olarak yüklenmiştir. Modelin güven eşiği (conf) 0.5, NMS (Non-Maximum Suppression) eşiği (iou) 0.45 olarak ayarlanmıştır.

print("Yüklenen sınıflar:", model.names) # Kontrol amaçlı

# CARLA simülatörüne bağlan
client = carla.Client("localhost", 2000)
client.set_timeout(10.0)
world = client.get_world()
print("Harita:", world.get_map().name)
```

Model adları yazdırılarak modelin yüklenip yüklenmediği kontrol edilmiştir. carla.Client kullanılarak simülatöre bağlantı kurulmuş ve client.set_timeout ile bağlantı zaman aşımı belirlenmiştir. Simülasyonun sanal dünyası (world) elde edilmiş ve harita bilgisi kontrol edilmiştir.

```
# Kamera sensörünü oluştur
CAMERA_POS_Z = 1.6
CAMERA_POS_X = 0.9
camera_bp = world.get_blueprint_library().find("sensor.camera.rgb")
camera_bp.set_attribute("image_size_x", "640")
camera_bp.set_attribute("image_size_y", "360")
camera_bp.set_attribute("fov", "110")

camera_init_trans = carla.Transform(carla.Location(z=CAMERA_POS_Z,
x=CAMERA_POS_X))
camera = world.spawn_actor(camera_bp, camera_init_trans,
attach_to=vehicle)
```

Aracın önüne, belirli bir konumda (**z=1.6, x=0.9**) bir RGB kamera sensörü (**sensor.camera.rgb**) eklenmiştir. Kamera çözünürlüğü 640x360 piksel olarak ayarlanmış ve görüş alanı (**FoV**) **110 derece** olarak belirlenmiştir. Kamera verilerini işlemek üzere camera_data adında bir NumPy dizisi (**np.zeros**) oluşturulmuştur.

```

# Görüntü tutucu (paylaşılacak veri)
image_w = int(camera_bp.get_attribute("image_size_x"))
image_h = int(camera_bp.get_attribute("image_size_y"))
camera_data = {"image": np.zeros((image_h, image_w, 3),
                                 dtype=np.uint8)}

# Kamera callback fonksiyonu
def camera_callback(image, data_dict):
    global is_braking, brake_end_time

    # RGBA → BGR ve writable hale getir
    array = np.frombuffer(image.raw_data,
                          dtype=np.uint8).reshape((image.height, image.width, 4))
    frame = np.array(array[:, :, :3][:, :, ::-1], copy=True)

    # YOLO tahmini
    results = model(frame)
    det = results.xyxy[0] # [x1, y1, x2, y2, conf, cls]

    for *xyxy, conf, cls in det:
        cls = int(cls)
        conf = float(conf)
        label = model.names[cls] if model.names else str(cls)
        x1, y1, x2, y2 = map(int, xyxy)

        # Renk: varsa kullan, yoksa yeşil
        color = color_map.get(label, (0, 255, 0))

        # Kutu çiz
        cv2.rectangle(frame, (x1, y1), (x2, y2), color, 2)
        cv2.putText(frame, f"{label} {conf:.2f}", (x1, y1 - 10),
                   cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)

        # Fren tetikleme örneği (sadece araçlar için)
        if label == "Vehicle" and conf > 0.6 and not is_braking:
            print("Araç tespit edildi, frenleme başlıyor...")
            is_braking = True
            brake_end_time = time.time() + brake_duration

    data_dict["image"] = frame

```

Görüntü İşleme: Kameradan gelen ham **RGBA** görüntüsünü alır, NumPy dizisine dönüştürür, **BGR** formatına çevirir ve üzerinde işlem yapılabılır hale getirir.

YOLO Tahmini: İşlenen bu görüntü üzerinde eğitilmiş YOLO modelini çalıştırarak nesne tespiti yapar. Tespit edilen nesnelerin konumlarını, güven skorlarını ve sınıflarını (araç, trafik ışığı vb.) alır.

Görselleştirme: Tespit edilen her nesne için görüntü üzerine sınırlayıcı kutular ve etiketler (**örneğin “Vehicle 0.95”**) çizer. Her sınıf için önceden tanımlanmış renklere göre kutuların rengini ayarlar.

Görüntü Güncelleme: İşlenmiş ve etiketlenmiş görüntüyü **data_dict** adlı paylaşılan bir sözlüğe kaydeder, böylece ana döngüde bu görüntü gösterilebilir.

```

# Kamerayı dinlemeye başla
camera.listen(lambda image: camera_callback(image, camera_data))

# Ana döngü
try:
    while True:
        if is_braking and time.time() >= brake_end_time:
            print(" Fren bırakıldı, araç devam ediyor.")
            is_braking = False

        world.tick()
        cv2.imshow("CARLA Object Detection", camera_data["image"])
        if cv2.waitKey(1) & 0xFF == ord("q"):
            break

    finally:
        print(" Simülasyon sonlandırılıyor...")
        cv2.destroyAllWindows()
        camera.stop()
        vehicle.destroy()

```

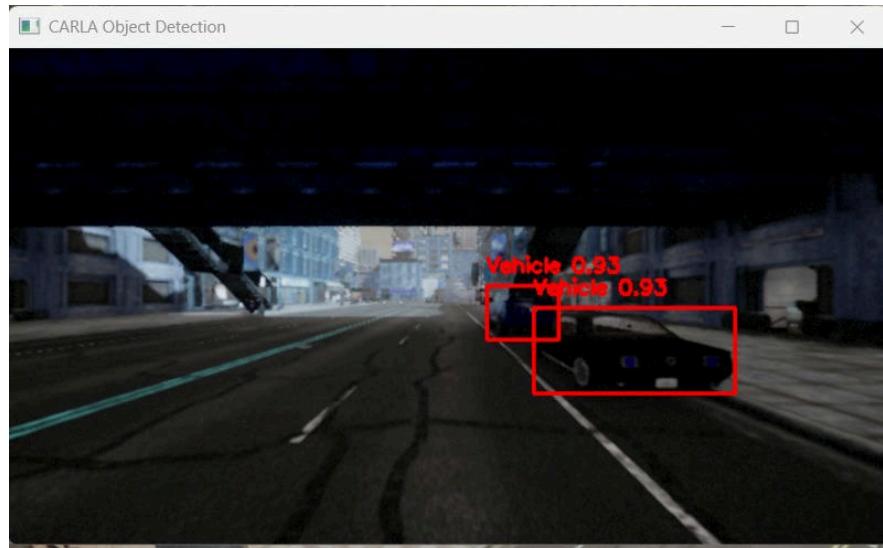
Kamera Dinleme: `camera.listen()` komutu ile kameradan gelen görüntüleri sürekli olarak `camera_callback` fonksiyonuna gönderir. Yani, her yeni görüntü geldiğinde `camera_callback` otomatik olarak çalıştırılır.

Ana Simülasyon Döngüsü: Bir `while True` döngüsü ile simülasyonu çalışır halde tutar: `world.tick()` ile CARLA simülasyon dünyasını bir adım ilerletir, böylece her şey (araç hareketi, sensör verileri vb.) güncel kalır. `cv2.imshow()` ile `camera_callback`'ten gelen işlenmiş (nesnelerin tespit edildiği) görüntüyü ekranda gösterir. Kullanıcının ‘Q’ tuşuna basıp basmadığını kontrol eder; basarsa döngüyü kırar ve simülasyonu sonlandırır.

Güvenli Kapanış: `finally` bloğu, simülasyon sonlandığında (normalde veya bir hata nedeniyle) tüm pencereleri kapatır (`cv2.destroyAllWindows()`), kamera dinlemesini durdurur (`camera.stop()`) ve oluşturulan aracı simülasyon dünyasından kaldırır (`vehicle.destroy()`). Bu, kaynakların düzgün bir şekilde serbest bırakılmasını sağlar.

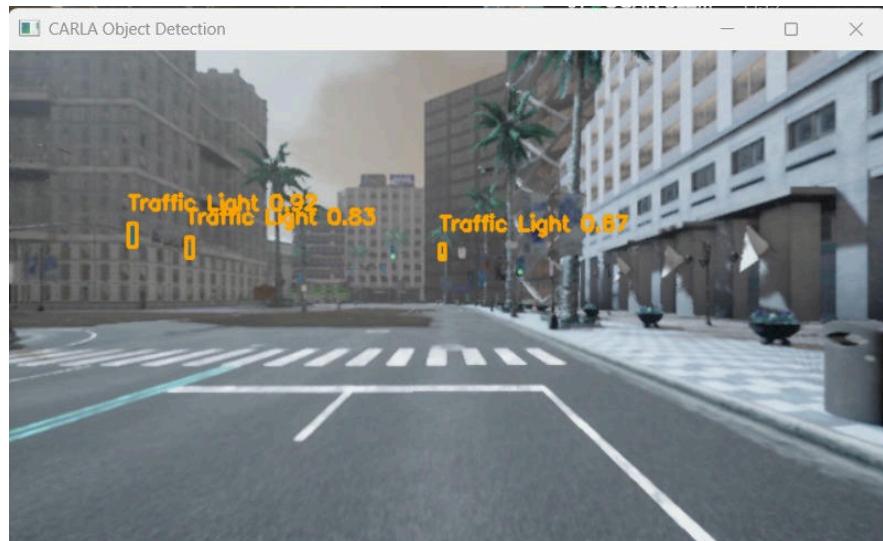
5.2.3.3 Görsel Çıktılar

CARLA simülasyon ortamında eğitilmiş YOLOv5x modelinin gerçek zamanlı nesne tespiti performansını gösteren temsili görseller sunulmuştur. Bu görseller, modelin karmaşık simülasyon senaryolarında farklı nesne sınıflarını (araçlar, trafik ışıkları, trafik işaretleri) başarıyla tespit edebildiğini ve otonom sürüsüz uygulamaları için potansiyelini açıkça ortaya koymaktadır. Her bir görselde, tespit edilen nesneler sınırlayıcı kutular ve ilgili sınıf etiketleriyle birlikte gösterilmiştir.



Şekil 5.24: Carla araç tesbit resmi

Şekil 24, modelin yoğun trafik akışı içinde birden fazla aracı aynı anda ve yüksek doğrulukla tespit edebildiği görülmektedir. Farklı boyut ve uzaklıktaki araçların doğru bir şekilde algılanması, modelin genelleyici yeteneğini ve çeşitli senaryolara adaptasyonunu göstermektedir. Özellikle, modelin birden fazla şeritteki araçları ayırt edebilmesi ve güven skorlarını (örn. “Vehicle 0.94”) belirterek tespitin güvenilirliğini sunması dikkat çekicidir.



Şekil 5.25: Carla trafik ışığı tesbit resmi

Şekil 25'de modelin hem hareketli nesneleri (araçlar) hem de statik çevresel unsurları (trafik ışıkları) aynı anda algılama kabiliyetini ortaya koymaktadır. Görüldüğü gibi, modelin bir trafik ışığını (“Traffic Light”) başarıyla tespit etmiştir.



Şekil 5.26: Carla motorsiklet ve bisiklet resmi

Şekil 26 CARLA simülasyon ortamında YOLOv5x modelinin gerçek zamanlı nesne tespiti çıktıları. Görüntüde araç, motosiklet ve bisiklet nesneleri başarıyla tespit edilmiş ve sınırlayıcı kutularla işaretlenmiştir. Her nesneye ait sınıf adı ve modelin güven skoru (%78–%85 arası) kutu üzerine yazılmıştır. Şekil 26 de, modelin farklı sınıfları yüksek doğrulukla tanıdığını ve otonom sürüş uygulamaları için pratik olarak kullanılabileceğini göstermektedir.

6 Sonuç ve Gelecek Çalışmalar

Bu çalışmada, otonom araçlar için çevre algılamasında nesne tespiti problemini çözmeye yönelik olarak, yaygın ve güçlü derin öğrenme tabanlı iki farklı yaklaşım olan Detectron2 ve YOLO mimarileri kullanılarak kapsamlı deneyler gerçekleştirilmiştir. Eğitim ve test süreçleri, CARLA simülasyon ortamından elde edilen özel bir veri kümlesi üzerinde yürütülmüştür.

Detectron2 altyapısı altında Faster R-CNN mimarisine sahip üç farklı model (R_50_FPN_1x, R_50_FPN_3x, R_101_FPN_3x) detaylı biçimde eğitilmiş ve değerlendirilmiştir. Bu modeller arasında R_101_FPN_3x modeli, hem genel başarı oranı (mAP) hem de küçük nesneleri tespit etme kabiliyeti açısından en yüksek performansı göstermiştir. Eğitim süresi ve donanım gereksinimi daha düşük olan R_50_FPN_1x ise hızlı prototipleme için uygun ancak doğruluk açısından sınırlı kalmıştır.

YOLO mimarileriyle yapılan analizlerde, başlangıçta yüksek performans vaat eden YOLOv9s modelinin CARLA'nın Python 3.7 kısıtlaması nedeniyle entegre edilemediği görülmüştür. Bu nedenle, uyumluluk sorununu aşmak için YOLOv5'e geçiş yapılmıştır. Ultralytics kütüphanesiyle yapılan ilk eğitim denemelerinde torch.hub.load kaynaklı hatalar yaşanmış, ancak YOLOv5'in resmi GitHub deposu kullanılarak ve yolov5x.pt önceden eğitilmiş modelin veri kümemiz üzerinde yeniden eğitilmesiyle sorun başarılı bir şekilde çözülmüştür. Bu strateji, YOLOv5x modelinin CARLA simülasyon ortamında gerçek zamanlı nesne tespiti için stabil ve etkili bir çözüm sunmasını sağlamıştır. Gerçek zamanlı testler, modelin çeşitli sürüs senaryolarında (yoğun trafik, trafik ışıkları, trafik işaretleri) nesneleri doğru ve hızlı bir şekilde tespit edebildiğini göstermiştir.

Bu çalışmada elde edilen sonuçlar, otonom araçlar için nesne tespiti konusunda sağlam bir temel oluşturmaktadır. Gelecekteki çalışmalar, bu temel üzerine inşa edilerek modelin gerçek dünya senaryolarındaki sağlamlığını ve uygulama alanlarını genişletmeyi hedefleyecektir. Bu kapsamında öne çıkan başlıklar şunlardır:

- **Çoklu Sensör Füzyonu:** Yalnızca kamera verilerine bağlı kalmayıp, lidar ve radar gibi farklı sensörlerden gelen verilerin entegrasyonuyla nesne tespiti performansının artırılması hedeflenecektir.
- **Gerçek Zamanlı Takip ve Davranış Tahmini:** Sadece nesne tespiti yapmakla kalmayıp, tespit edilen nesnelerin hareket yörüngelerini tahmin eden ve davranışlarını analiz eden (örneğin, yayaların anı hareketleri, araçların şerit değiştirmesi) ileri düzey takip algoritmalarının entegrasyonu hedeflenmektedir. Bu, otonom aracın daha güvenli ve öngörülü kararlar almasını sağlayacaktır.
- **Hesaplama Verimliliği ve Kenar Cihazlara Optimizasyon:** Gerçek otonom araçlarda kısıtlı işlem gücü ve enerji tüketimi göz önünde bulundurularak, eğitilmiş modellerin daha hafif ve hızlı sürümlerinin (quantization, pruning) geliştirilmesi ve kenar bilişim cihazlarına (edge devices) optimize edilmesi önemli bir araştırma alanı olacaktır. Bu, sistemin gerçek zamanlı performansını ve ticari uygulanabilirliğini artıracaktır.

Kaynakça

- [1] V. De Jong Yeong, G. Velasco-Hernandez, J. Barry, ve J. Walsh, “Sensor and sensor fusion technology in autonomous vehicles: A review”, Sensors, c. 21, sy 6, s. 2133, 2021, [Çevrimiçi]. Erişim adresi: <https://www.mdpi.com/1424-8220/21/6/2133>
- [2] X.-Y. Zhang ve J. Wu, “Deep learning: A comprehensive overview on techniques, taxonomy, applications and research directions”, SN Applied Sciences, c. 3, sy 3, s. 349, 2021, doi: 10.1007/s42452-021-04391-7.
- [3] J. Murel ve E. Kavlakoglu, “What is object detection?”, IBM, Oca. 2024, Erişim: 18 Haziran 2025. [Çevrimiçi]. Erişim adresi: <https://www.ibm.com/topics/object-detection>
- [4] J. Redmon, S. Divvala, R. Girshick, ve A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection”, CoRR, 2015, [Çevrimiçi]. Erişim adresi: <http://arxiv.org/abs/1506.02640>
- [5] Meta AI, “Detectron2 – Meta AI”. [Çevrimiçi]. Erişim adresi: <https://ai.meta.com/resources/models-and-libraries/detectron2/>
- [6] A. Sharma, “Mean Average Precision (mAP) Using the COCO Evaluator”, PyImageSearch, 2022, Erişim: 18 Haziran 2025. [Çevrimiçi]. Erişim adresi: <https://pyimagesearch.com/2022/03/14/mean-average-precision-map-using-the-coco-evaluator/>
- [7] Google Developers, “Classification: Accuracy, precision, recall, and related metrics”. [Çevrimiçi]. Erişim adresi: <https://developers.google.com/machine-learning/crash-course/classification/accuracy>
- [8] CARLA Simulator, “CARLA 0.9.13 Release”. Erişim: 18 Haziran 2025. [Çevrimiçi]. Erişim adresi: <https://carla.org/2021/08/17/release-0.9.13/>
- [9] Ultralytics, “COCO Dataset – Ultralytics YOLO Docs”. Erişim: 18 Haziran 2025. [Çevrimiçi]. Erişim adresi: <https://docs.ultralytics.com/datasets/detect/coco/>
- [10] C. Dong, “Exploring visual techniques for indoor intrusion detection using detectron2 and Faster RCNN”, Proceedings of the 2023 International Conference on Machine Learning and Automation, 2023, doi: 10.54254/2755-2721/36/20230452.
- [11] J. Byun, J. Kim, ve S. Byun, “Object Detection with Detectron2 for Pothole Detection”, Journal of Harbin Engineering University, c. 45, sy 4, 2024.
- [12] J. Redmon, S. Divvala, R. Girshick, ve A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection”, içinde Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, ss. 779-788.
- [13] J. Redmon ve A. Farhadi, “YOLOv3: An Incremental Improvement”, CoRR, 2018, [Çevrimiçi]. Erişim adresi: <https://arxiv.org/abs/1804.02767>

- [14] Ultralytics, “YOLOv5 GitHub Repository”. 2020.
- [15] C.-Y. Wang, H.-Y. M. Liao, C.-H. Wu, P.-Y. Chen, J.-W. Hsieh, ve I.-H. Yeh, “YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information”, CoRR, 2024, [Çevrimiçi]. Erişim adresi: <https://arxiv.org/abs/2402.13616>
- [16] Suraj520, “Carla Object Detection Dataset”. 2021.
- [17] C.-Y. Wang, A. Bochkovskiy, ve H.-Y. M. Liao, “YOLOv7: Trainable bag-of-freebies for real-time object detection”, CoRR, 2022, [Çevrimiçi]. Erişim adresi: <https://arxiv.org/abs/2207.02696>
- [18] Ultralytics, “YOLOv8 Modelleri Dokümantasyonu”. 2024.
- [19] Z. Huang, L. Li, G. C. Krizek, ve L. Sun, “Research on Traffic Sign Detection Based on Improved YOLOv8”, içinde 2nd International Conference on Civil, Mechanical and Automotive Engineering (CMAE 2023), IOP Publishing, 2023, s. 12023. doi: 10.1088/1742-6596/2642/1/012023.
- [20] J. Redmon ve A. Farhadi, “YOLOv3: An Incremental Improvement”, CoRR, 2018, [Çevrimiçi]. Erişim adresi: <http://arxiv.org/abs/1804.02767>
- [21] Y. Zhang, M. Yao, H. Zhang, X. Li, ve Y. Lv, “Lightweight Object Detection Algorithm Based on Improved YOLOv5”, Sensors, c. 22, sy 16, s. 6140, 2022, doi: 10.3390/s22166140.
- [22] Ultralytics, “Ultralytics GitHub Deposu”. 2024.
- [23] Ultralytics, “Ultralytics Modelleri Dokümantasyonu”. 2024.