**T.C.**
**İNÖNÜ ÜNİVERSİTESİ**
**FEN BİLİMLERİ ENSTİTÜSÜ**

# YAPAY ZEKA TEKNOLOJİLERİ İLE MOBİL UYGULAMALARDA NESNE TANIMA

**YÜKSEK LİSANS TEZİ**

**Batuhan KARADAĞ**

**Bilgisayar Mühendisliği Anabilim Dalı**

**Tez Danışmanı: Dr. Öğr. Üyesi Ali ARI**

**AĞUSTOS 2023**

**T.C**
**İNÖNÜ ÜNİVERSİTESİ**
**FEN BİLİMLERİ ENSTİTÜSÜ**
**İNÖNÜ ÜNİVERSİTESİ**
**FEN BİLİMLERİ ENSTİTÜSÜ MÜDÜRLÜĞÜ**

# YAPAY ZEKA TEKNOLOJİLERİ İLE MOBİL UYGULAMALARDA NESNE TANIMA

**YÜKSEK LİSANS TEZİ**

**Batuhan KARADAĞ**
**36193619022**

**Bilgisayar Mühendisliği Anabilim Dalı**

**Tez Danışmanı: Dr. Öğr. Üyesi Ali ARI**

**AĞUSTOS 2023**

**İNÖNÜ ÜNİVERSİTESİ**
**FEN BİLİMLERİ ENSTİTÜSÜ MÜDÜRLÜĞÜ**

# YAPAY ZEKA TEKNOLOJİLERİ İLE MOBİL UYGULAMALARDA NESNE TANIMA

**Tezi Hazırlayan: Batuhan KARADAĞ**
**Yüksek Lisans Tezi**

## TEŞEKKÜR

Bu tez çalışmasının her aşamasında yardım, öneri, bilgi, tecrübe ve desteklerini esirgemeden beni her konuda yönlendiren danışman hocam Sayın Dr. Öğr. Üyesi Ali Arı'ya,

Çalışmalarımın yanı sıra tüm hayatım boyunca her türlü desteklerini sonsuz şekilde bana sunan ve hep arkamda duran sevgili aileme,

Bilgisini ve tecrübesini benden esirgemeyen, her zaman maddi manevi yanımda olan hocam Sayın Prof. Dr. Davut Hanbay'a,

Lisans eğitimim boyunca hiç ayrılmadığımız ve çalışmalarım sürecinde beni destekleyen sevgili arkadaşlarım Mehmet Beşe'ye, Ahmet Ceylan'a, Osman Beyazoğlu'na ve kıymetli eşi Burcu Beyazoğlu'na, Akif Demirci'ye ve kıymetli eşi Esra Demirci'ye, Ahmet Singan'a ve kıymetli eşi İrem Singan'a,

Motivasyonları ve bana kattıklarıyla çok sevdiğim canım arkadaşlarım Sayın Dr. Öğr. Üyesi Merve Nilay Aydın ve Sayın Arş. Gör. Sami Karakul'a,

Çalışmalarımda yanımdan ayrılmayan ve mırıltısıyla huzur veren biricik dostum olan kedim Peri'ye,

teşekkür ederim.

# ÖNSÖZ

*06.02.2023 tarihinde gerçekleşen Kahramanmaraş merkezli depremden etkilenen tüm canlara geçmiş olsun diliyor ve hayatını kaybedenlerin mekanlarının cennet olmasını temenni ediyorum…*

Batuhan Karadağ

## ONUR SÖZÜ

Yüksek lisans tezi olarak sunduğum “Yapay Zeka Teknolojileri ile Mobil Uygulamalarda Nesne Tanıma” başlıklı bu çalışmanın bilimsel ahlak ve geleneklere aykırı düşecek bir yardıma başvurmaksızın tarafımdan yazıldığına ve yararlandığım bütün kaynakların hem metin içinde hem de kaynakçada yöntemine uygun biçimde gösterilenlerden oluştuğunu belirtir, bunu onurumla doğrularım.

Batuhan Karadağ

# İÇİNDEKİLER

# ÇİZELGELER DİZİNİ

# ŞEKİLLER DİZİNİ

## SEMBOLLER VE KISALTMALAR

**FPS** : Saniyedeki kare sayısı
**GPU** : Grafik işleme birimi
**YZ** : Yapay zeka
**SL** : Gözetimli öğrenme
**UL** : Gözetimsiz öğrenme
**RL** : Takviyeli öğrenme
**DL** : Derin öğrenme
**TL** : Transfer öğrenme
**DT** : Karar ağaçları
**DVM** : Destek vektör makineleri
**YSA** : Yapay sinir ağları
**HOG** : Yönlendirilmiş gradyan histogramı
**DPM** : Deforme edilebilir parça modeli
**ESA** : Evrişimli sinir ağı
**BESA** : Bölge tabanlı evrişimsel sinir ağı
**mAP** : Ortalama hassasiyet
**HBESA** : Hızlı bölge tabanlı evrişimsel sinir ağı
**RoI** : İlgi alanı havuzu
**DHBESA** : Daha hızlı bölge tabanlı evrişimsel sinir ağı
**BÖA** : Bölge öneri ağı
**FPN** : Özellik piramit ağı
**YOLO** : Yalnızca bir kere bak
**SSD** : Tek atış çoklu kutu dedektörü
**MP** : Maksimum havuzlama
**GS** : Güven skoru
**BoF** : Megabits per second
**BoS** : Station
**SPP** : Uzamsal piramit havuzlama
**SPPF** : Uzamsal piramit hızlı havuzlama
**EDH** : Etkili ayrılmış kafa
**TP** : Doğru pozitif
**FN** : Yanlış negatif
**FP** : Yanlış pozitif

**TN** : Doğru negatif

**E-ELAN** : Genişletilmiş verimli katman toplama ağı

**ELAN** : Verimli katman toplama ağı

**CSPNET** : Aşamalar arası kısmi ağ

**MS** : Microsoft

**COCO** : Bağlamda yaygın nesneler

**RN** : React-Native

**COLAB** : Google Colaboratory

**TPU** : Tensor işleme birimi

**TF** : TensorFlow

**TFLite** : TensorFlow Lite

**ONNX** : Açık sinir ağı değişimi

**IoT** : Nesnelerin interneti

# ÖZET

Yüksek Lisans Tezi

YAPAY ZEKA TEKNOLOJİLERİ İLE
MOBİL UYGULAMALARDA NESNE TANIMA

Batuhan KARADAĞ

İnönü Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

58+xi sayfa

2023

Danışman: Dr. Öğr. Üyesi Ali ARI

Teknolojinin ilerlemesiyle birlikte, neredeyse herkesin sahip olduğu akıllı bir mobil cihaz gözlemlenmektedir. Yapılan bu tez çalışmasında, akıllı mobil cihazların donanım imkanlarının yüksek olup olmadığına bakılmaksızın, nesne tanıma ve nesne tespiti yapılması istenilen görseli, çeşitli teknikler kullanarak içerisinde bulunan nesnelerin tanınması ve tespit edilmesi amaçlanmıştır. Derin öğrenmeye dayalı güncel nesne tespit algoritmalarından biri olan YOLO, ilk sunulduğu dönemden itibaren geliştirilmeye devam eden bir nesne tespit aracıdır. YOLO, hız ve doğruluk açısından etkili olması sebebiyle ticari alandaki nesne tespit problemlerinde tercih edilen bir araç olmuştur. Yapılan bu tez çalışmasında, akıllı mobil cihaz için bir arayüz ve YOLOv7 modelinin nesne tespit problemini gerçekleştirebilmesi için bir sunucu geliştirilmiştir. YOLO'nun 7. versiyonu YOLOv7, Microsoft COCO veri seti ile eğitiminin ardından %51.2'lik ortalama hassasiyet başarısı elde etmesi sebebiyle yapılan bu çalışmada tercih edilmiştir. Genellikle derin öğrenmeye dayalı sistemlerin çalıştırılabilmesi için yüksek kapasiteli donanımlara ihtiyaç duyulurken, tasarlanan bu sunucu ile ağ bağlantısına sahip akıllı bir mobil cihazın grafik işlemci birimine (GPU) bakılmaksızın nesne tespit problemi çözümü başarılı bir şekilde gerçekleştirilmiştir.

**Anahtar Kelimeler:** YOLOv7, Mobil Nesne Tespiti, Mobil YOLOv7.

## ABSTRACT

Master Thesis

OBJECT RECOGNITION IN MOBILE APPLICATIONS WITH ARTIFICIAL INTELLIGENCE TECHNOLOGIES

Batuhan KARADAĞ

Inonu University
Graduate School of Nature and Applied Sciences
Department of Computer Science

58+xi sayfa

2023

Supervisor: Asst. Prof. Ali ARI

With the advancement of technology, a smart mobile device is observed that almost everyone has. In this thesis study, it is aimed to recognize and detect the objects in the image by using various techniques, regardless of whether the hardware capabilities of smart mobile devices are high or not. YOLO, one of the current object detection algorithms based on deep learning, is an object detection tool that has been under development since it was first introduced. YOLO has become the preferred tool for object detection problems in the commercial field due to its speed and accuracy. In this thesis, an interface for the smart mobile device and a server have been developed for the YOLOv7 model to realize the object detection problem. In this study, the 7th version of YOLO, YOLOv7, achieved an average accuracy of 51.2% after training with the Microsoft COCO dataset. has been preferred. While high-capacity hardware is generally needed to run deep learning-based systems, the object detection problem has been successfully solved regardless of the graphics processor unit (GPU) of a smart mobile device with a network connection with this designed server.



**Keywords:** YOLOv7, Mobile Object Detection, Mobile YOLOv7.

# 1. GİRİŞ

Nesne tanıma ve nesne tespiti, dijital görüntü işleme uygulamalarının başında gelen bir bilgisayarlı görü tekniğidir [1]. Nesne tanıma ve nesne tespiti temelde birbirlerine çok yakın iki kavram olmakla birlikte ince bir çizgi ile birbirlerinden ayrılmaktadır. Görüntü veya video içerisinde bulunan nesnelerin bilgisayar tarafından tanınıp sınıflandırılması işlemine nesne tanıma denilmekte olup, nesnelerin konumlarının (koordinatlarının) ve boyutlarının bilgisayar tarafından bulunması işlemine de nesne tespiti denilmektedir. Nesne tanıma ve nesne tespit kavramlarının amacı, tıpkı bir insanın görüntü veya video içerisinde bulunan nesneleri tanıdığı gibi, bilgisayara da bu tanıma özelliğini öğretmektir [2]. Nesne tanıma işlemi, birden çok nesne için sınıflandırma amacında olduğundan ötürü nesne tespitine göre daha kapsamlı bir süreçtir. Derin öğrenme tekniklerinin büyük verileri işleyebilme kapasitesi ile nesne tanıma ve nesne tespit kavramları gelişim kaydetmiştir [2].

Derin sinir ağlarının uygulanabilirlik ve verimlilik bakımından avantajlı olması ve gerçek hayatta karşılaşılan problemlerin çözümünde sağlamış olduğu başarı ile bilgisayarlı görü ve makine öğrenmesi algoritmalarının popülerliği giderek artmaktadır [3-5]. Nesne tanıma ve nesne tespit algoritmaları özünde geleneksel yöntemler ve modern yöntemler derin öğrenmeye dayalı modern yöntemler olarak isimlendirilerek ikiye ayrılmaktadır [3]. Nesne tanıma ve nesne tespit problemlerinde kullanılan derin öğrenmeye dayalı yöntemler, hız ve doğruluk bakımından geleneksel yöntemlerden daha iyi sonuç vermektedir [6]. Nesne tanıma ve nesne tespit problemlerinde kullanılan derin öğrenmeye dayalı modern yöntemler literatürde incelendiğinde farklı yaklaşımları olduğu görülmektedir [3, 7]. Genellikle nesne tanıma ve nesne tespit araçları, asıl problemi sınıflandırma problemine veya regresyon problemine dönüştürmektedir [3]. Yöntem olarak ayrılsalar dahi nesne tanıma ve tespit problemlerinde kullanılan her iki yaklaşım da başarılı sonuçlar vermektedir [3, 8].

Sınıflandırma yaklaşımları, tamamıyla bölgelere dayalı olması sebebiyle, öncelikle verilen girdi görüntüleri için bölge önerileri üretmektedir [3]. Önerilen bölgelerde ağ, girdi görüntüsü üzerinde hangi konumda nesne içerildiğini önceden kontrol etmesi sebebiyle doğruluk açısından daha başarılıdır ama bununla beraber işlem maliyetini artırmaktadır [3, 9]. Regresyon yaklaşımları ise, öncelikle girdi görüntüsü içerisinde nesnenin var olup

olmadığı kontrolunü ardından nesne varsa bu nesnenin hangi sınıfa ait olduğuna odaklanmaktadır [3].

Regresyona dayalı nesne tanıma ve nesne tespit algoritmalarından biri olan YOLO günümüzde, gerçek hayatta karşılaşılan problemlerde yaygın olarak kullanılan bir derin öğrenme algoritmasıdır [3, 10]. Redmon ve arkadaşları [11] tarafından 2015 yılında öne sürülen YOLO algoritması, günümüze kadar birçok nesne tanıma ve nesne tespit problemlerinin çözümünde doğru ve hızlı bir şekilde sonuca ulaşabildiği için kullanılmaktadır [3]. YOLO algoritması, bilgisayar bilimcileri tarafından geliştirilerek daha düşük maliyetle daha kesin sonuçlar elde edecek şekilde iyileştirilmeye devam etmektedir [3].

Wang ve arkadaşlarının tanıtmış olduğu YOLOv7 [12] nesne tanıma ve nesne tespit algoritması 5-160 FPS (Saniyedeki kare sayısı) aralığında almış olduğu %51.2'lik başarımı sayesinde literatürde bulunan diğer nesne tanıma ve nesne tespit algoritmalarından daha başarılı olduğunu ispatlamıştır [3]. Yapılan bu tez çalışmasında, YOLOv7 nesne tanıma ve nesne tespit algoritmasının akıllı mobil cihazlara uygulanması ile mobil cihazlardan seçilen bir görüntünün içerisinde bulunan nesnelerin kısa bir sürede tanındığı görülmüştür. Akıllı mobil cihazın yalnızca kameraya ve ağ bağlantısına sahip olmasıyla birlikte grafik işlemci birimi (Graphics processing unit - GPU) gücüne bakılmaksızın, kullanıcının tasarlanan arayüzü kullanarak görüntü içerisinde bulunan nesnelerin belirlenmesi için istediği görseli uygulamaya yüklemesinin ardından nesne tanıma sonucunun hızlı bir şekilde alınması başarılmıştır. Uygulamaya yüklenen görsel, geliştirilen sanal sunucuya aktarılmakta ve tespit işlemi sanal sunucuda gerçekleşmektedir. Sanal sunucudan görsel içerisinde bulunan nesnelerin konumlarına ve büyüklüklerine bağlı olarak sınırlayıcı kutuların ve kutular üzerinde nesnelerin hangi sınıfa ait olduğunu belirten etiketler yazılarak sonuç elde edilmiştir. Elde edilen sonuç, mobil uygulumanın arayüzünde kullanıcıya gösterilmektedir. Bu tez çalışmasının sonucu olarak, başarılı bir şekilde nesne tanıma probleminin ağ bağlantısına sahip akıllı bir mobil cihaz üzerinde gerçekleştirildiği gösterilmiştir.

### 1.1 Tezin Önemi ve Hedefleri

Yapılan bu tez çalışması ile akıllı mobil cihazlar üzerinden nesne tanınması yapılması amaçlanmıştır. Bu amaç çerçevesinde, yalnızca kamera ve ağ bağlantısına sahip bir akıllı mobil cihazın GPU'ya ihtiyaç duyulmadan sanal sunucuya aktarılan görsel içerisinde bulunan nesnelerin tespit edilmesi sağlanmıştır. Yapılan bu tez çalışması için mobil arayüz

ve sanal sunucu geliştirilmiştir. Kullanıcı ağ bağlantısı olan mobil cihazı üzerinden, bu tez çalışması için geliştirilen mobil arayüzü kullanarak nesne tanıma yapmak istediği görseli sunucuya aktarır ve dönüt bekler. Sunucudan dönüt olarak gelen yanıt, aktarılan görsel içerisinde tanınan nesnelerin üzerine çizilen sınırlayıcı kutulardır. Çizilen sınırlayıcı kutular üzerinde, tanınan nesnelerin isimlendirilmiş etiketleri ve doğruluk sonuçları yer almaktadır. Sanal sunucu tarafında ise, bu nesnelerin etiketleri bir yazı dosyasında (TEXT) tutulmaktadır. Bu sayede, yalnızca ağ bağlantısı ve kamera gibi temel özelliklere sahip olan bir akıllı mobil cihazın GPU gücüne ihtiyaç duymadan görsel içerikte bulunan nesnelerin tanınması ve tespit edilmesi başarıyla gerçekleştirilmektedir.

Yapılan bu çalışmanın hedefi, teknolojinin gelişmesiyle birlikte hacim kazanan e-ticaret sektöründe kullanılan mobil uygulamalarda, kullanıcıların arayacakları ürünleri yazmak yerine, görsel yükleyerek hızlı bir şekilde arama yapabilmeleridir. Geliştirilen arayüz ve sanal sunucu sayesinde görsel içerisindeki nesnelerin tanınma işlemi, akıllı bir mobil cihaz için hızlı ve az maliyetli olduğu görülmüştür.

### 1.2 Tezin Organizasyonu

Bölüm 2'de yapay zeka ve makine öğrenmesi konularına yer verilmiştir. Bölüm 3'te ise literatürde bulunan nesne tespit araçlarından bahsedilmiş ve nesne tespit problemlerinin performansını ölçmek ve değerlendirmek için kullanılan metrikler ele alınmıştır. Bölüm 4’te yapılan çalışmada kullanılan algoritma, veri seti, geliştirilen uygulamanın ortamlarından ve edinilen modelin çalışmaya uyarlanması ayrıntılı bir şekilde açıklanılmıştır. Bölüm 5’te uygulamada elde edilen sonuçlara yer verilmiştir.

# 2. YAPAY ZEKA

## 2.1 Yapay Zeka Nedir

İnsanoğlu kendi zekasının sağlamış olduğu algılama, öğrenme, fikir yürütme, sorun çözme, iletişim kurma gibi yeteneklere sahiptir. Bu özellikleri sayesinde insan diğer bütün canlılardan ayrılmaktadır. İnsanoğlunun aletleri kullanabilme yeteneği geliştikçe hayatı kolaylaştırabilmek için icatlar yapmaya başlamıştır. Modern çağın gelişmesi ile makinelerin hayatımızdaki yeri günden güne artmaktadır. Kullanılan bu makinelerin hepsinin temelde tek bir amacı bulunmaktadır. Bu amaç ise, insanoğluna yardımcı olup işlerini kolaylaştırmaktır. Geçtiğimiz on yıl içerisinde popülerlik kazanan "Yapay Zeka" kavramı özünde, insanın sahip olduğu bazı nitelikleri bir nevi taklit ederek tıpkı insan gibi algılayabilme, öğrenebilme, fikir yürütebilme, sorun çözebilme, iletişim kurabilme vb. özelliklere sahip olabilen bir makine olarak tanımlanabilmektedir. Bugün yapay zeka (Artificial intelligence - YZ), biz farkında olmasak dahi insanoğlunun hayatına büyük ölçüde nüfuz etmektedir.

## 2.2 Yapay Zeka Tarihçesi

İngiliz matematikçi Charles Babbage, insana ait zihinsel özelliklerin taklit etmeyi hedeflemiş ve hesaplama konusundaki insani hataların (Gözden kaçırma sonucundan kaynaklı ortaya çıkan yanlış hesaplamaların) önüne geçmeyi düşünmüştür [13]. Geliştirdiği "Fark Motoru", basit matematiksel işlemlerin yapılabilmesinin yanı sıra işlem sonuçlarının saklanabileceği bir hafızaya sahipti.

YZ kavramı, 1950'lerde Alan M. Turing'in "Makineler düşünebilir mi? [14] hipotezi ile ortaya çıkmıştır. 1956'da Dartmouth Konferansı ile YZ ismen resmi olarak dile getirilmiştir. Öncülüğünü John McCarthy, Marvin Minsky, Nathaniel Rochester ve Claude Shannon gibi bilim insanlarının başlattığı YZ hareketi ilerleyen zamanlarda gelişme göstermiştir. John McCarthy, YZ alanında Lisp [15] dilini geliştirmiştir ve doğal dil işleme ile ilgili çalışmalar gerçekleştirmiştir. İlerleyen zamanlarda IBM tarafından geliştirilen "Deep Blue" adlı makinenin dünya satranç şampiyonu Garry Kasparov'u yenmesi ile, Turing'in yıllar önce ortaya attığı hipotezin ne kadar doğru olduğu ortaya çıkarılmıştır.

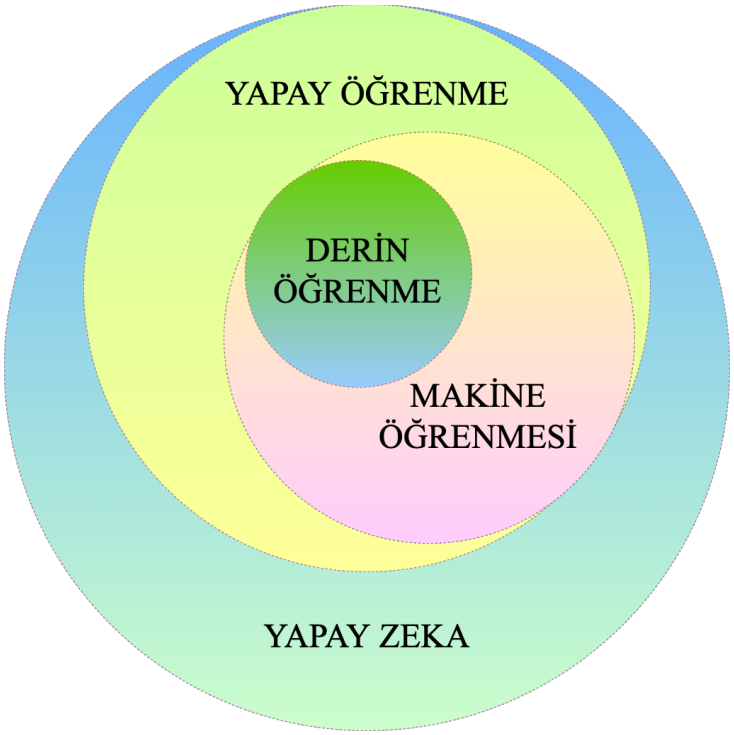
1960-1980 yılları YZ için karanlık çağ olarak adlandırılmaktadır. Lakin 1990'larda makine öğrenmesi alanındaki geliştirmeler bilim insanlarına umut olmuş ve YZ çalışmaları

tekrar hız kazanmaya başlamıştır. Günümüzde ise YZ, sağlık alanında hastalık tespiti yapılabilmesi, trafikte araçların sürücüsüz (otonom) kullanılabilmesi, finansta risk yönetimi ve verilerin analiz edilebilmesi, reklamcılıkta dijital pazarlama faaliyetlerinin geliştirilebilmesi, silah ve savunma sanayide insansız araçların/hava araçlarının kullanılabilmesi, dil işleme yöntemleri ile farklı dilleri konuşabilen insanların veyahut işitsel-duyusal engelli bireylerin birbirleriyle iletişim kurabilmesi, robotik faaliyetlerin geliştirilmesi vb. birçok alanda kullanılmaktadır [16].

## 2.3 Yapay Öğrenme

Yapay zeka alanının önemli bir alt disiplini olarak karşımıza çıkan yapay öğrenme temel olarak, bir bilgisayarın veri kullanarak (işleyerek) kendini geliştirebilmesidir. Bilgisayar bu gelişimini istatistiksel verilerle ve/veya bir dizi algoritmalar yardımıyla gerçekleştirmektedir.

Sınıflandırma, regresyon, kümeleme, doğal dil işleme, görüntü ve ses işleme, kullanıcı davranışına bağlı olarak öneri sistemi vb. gibi alanlarda tercih edilen bir yapay zeka disiplinidir. Yapay öğrenme ile makinelerin, belirli bir görevi tamamlaması veya problem çözme yeteneğini otomatik olarak kazanması hedeflenmektedir. Yapay zeka disiplinine ve alt dallarına ait hiyerarşi grafiği Şekil 2.1'de gösterilmiştir.




**Şekil 2.1 :** Yapay zeka ve alt dalları

## 2.4 Makine Öğrenmesi

Makine öğrenmesi, bilgisayarın belirli bir görevi gerçekleştirmesi veya verilen problemi çözebilmesi için veriye dayalı olarak eğitilmesinin ardından bilgisayarın öğrenme yeteneği kazandırılmasını sağlayan bir yapay zeka disiplinidir. Temel de yapay öğrenmeye çok benzemektedir fakat Şekil 2.1'de görüldüğü üzere makine öğrenmesi yapay öğrenmenin bir alt dalıdır. Yapay öğrenme, makinenin veri ve deneyimlerden öğrenme yeteneği olarak ifade edilirken makine öğrenmesi, bu sürecin çeşitli algoritmalar veya tekniklerle gerçekleştirmektedir. Makine öğrenmesi algoritmaları, veri analizi ve desen tanıma yöntemlerini kullanarak veriler arasındaki ilişkileri ve örüntüleri öğrenir ve bu bilgilere dayanarak gelecekteki veriler üzerinde analizler gerçekleştirebilir.

Makine öğrenmesinin uygulama alanı fark etmeksizin, büyük miktardaki verilerin analiz edilerek gelecek ile ilgili tahminlerde bulunabiliyor oluşu bu disiplinin önemini günden güne artırmıştır [17].

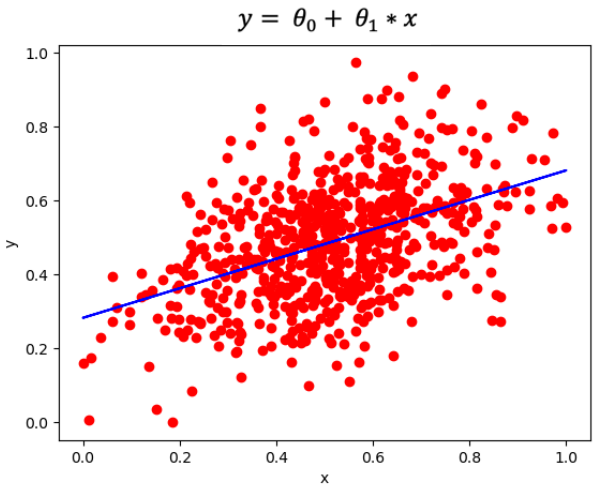
Makine öğrenmesi temelinde 5 ana başlıkla incelenmektedir. Bunlar: Gözetimli Öğrenme (Supervised Learning - SL), Gözetimsiz Öğrenme (Unsupervised Learning - UL), Takviyeli Öğrenme (Reinforcement Learning - RL), Derin Öğrenme (Deep Learning - DL) ve Transfer Öğrenme (Transfer Learning - TL)'dir.

### 2.4.1 Gözetimli öğrenme

Eğitim veri seti kullanılarak model eğitilmektedir ve model, bu eğitim sonucunda çıktı tahmin etmeyi öğrenmektedir. Eğitimin bir gözlemci tarafından (Önceden etiketlendirilmiş veri seti) kontrol edilmesini sağlayan öğrenme modelidir. Literatüre bakıldığında, gözetimli öğrenmenin yaygın olarak kullanılan bir makine öğrenmesi yöntemi olduğu görülmektedir. Bazı çok bilinen gözetimli öğrenme yöntemleri: Lineer Regresyon, Karar Ağaçları (Decision Trees - DT), Destek Vektör Makineleri (Support Vector Machines - DVM), Naif Bayes (Naive Bayes) ve Yapay Sinir Ağları (Artificial Neural Networks - ANNs)'dır.

#### 2.4.1.1 Lineer regresyon

Lineer regresyon, bağımlı bir değişkenin bir veye birden fazla bağımsız değişkenle ilişkisini analiz etmek ve bu ilişkiyi doğrusal bir modelle temsil etmek için kullanılan istatiksel bir yaklaşımdır. Veri setinde bulunan değişkenlere bakılarak gelecekteki değerleri tahmin etmek amaçlamaktadır. Lineer Regresyona ait örnek Şekil 2.2'de sunulmuştur.

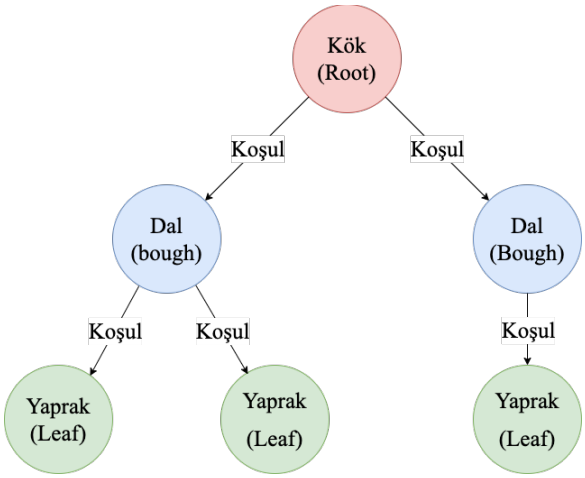



**Şekil 2.2 :** Lineer regresyon

### 2.4.1.2 Karar ağaçları

Ağaç yapıları anlaşılabilir olması sebebiyle yaygın olarak kullanılan bir yöntem olarak karşımıza çıkmaktadır [18]. Karar ağaçları, veriyi belirli özelliklerin değerlerine dayalı bir şekilde alt kümelere ayırarak çalışan ağaç biçiminde bir yapıdır [19]. Kök düğüm (root node) en üstte bulunur ve dallanarak ilerlemektedir.

Karar ağaçlarının her bir düğümü, bir kararı veya koşulu temsil etmektedir. Bu kararlar, veri setindeki ilişkileri ifade etmektedir. Her düğüm, bir özelliğe veya değişkene bağlı olarak verileri alt düğümlere bölmektedir. Bölme işlemi yaprak düğümlere (En alt kademede bulunan düğüm olan çocuklar) kadar devam etmektedir. Yaprak düğümleri eğer çözülmesi istenilen problem bir sınıflandırma ise sınıf etiketlerini temsil etmekte, eğer istenilen problem bir regresyon ise sayısal tahminleri temsil etmektedir. Karar ağacı örneği Şekil 2.3'te gösterilmiştir.



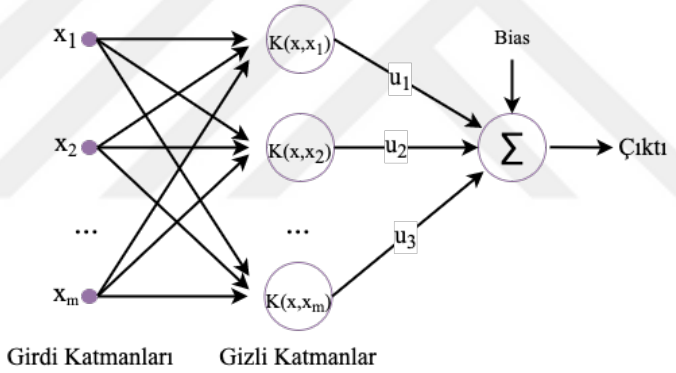

**Şekil 2.3 :** Karar ağacı yapısı

### 2.4.1.3 Destek vektör makineleri

Destek vektör makineleri (DVM), istatiksel öğrenmeye dayalı bir sınıflandırma algoritmasıdır [20]. DVM, sınıflandırma ve regresyon problemlerinde uygulanabilmektedir [21]. DVM ile yüz algılama, metin ve görüntü sınıflandırma, biyoinformatik çalışmaları, el yazısı tanıma, tahmine dayalı kontrol vb. olmak üzere birçok alanda kullanılmaktadır [22].

DVM özellikle doğrusal olarak birbirinden ayrılabilen sınıfların bulunduğu durumlarda etkilidir. İki sınıf arasında bir karar sınırı oluşturarak veri noktalarını en optimal şekilde ayırmaya çalışmaktadır.

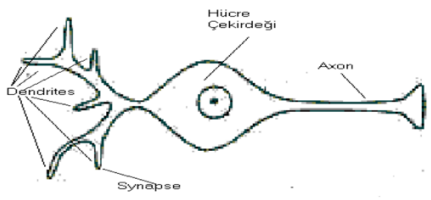
DVM, çekirdek (kernel) adı verilen fonksiyonları kullanarak doğrusal olmayan veriler için sınıflandırma veya regresyon problemlerini çözebilmektedir. Çekirdek fonksiyonları, veri noktalarını doğrusal olarak ayırabilmektedir. Çekirdek fonksiyonları ile DVM esneklik kazanmaktadır. Literatüre bakıldığında, yüksek performansı nedeniyle tercih edilen bir yöntemdir. Şekil 2.4'te örnek bir DVM yapısı gösterilmiştir.



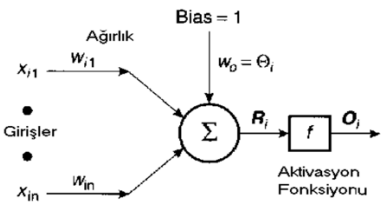

**Şekil 2.4 :** DVM yapısı [23]

### 2.4.1.4 Yapay sinir ağları

Yapay sinir ağları (YSA), insan beynin biyolojik yapısından esinlenilerek geliştirilmiş bir ağ modelidir. Şekil 2.5'te nöron (insan sinir hücresi), 2.6'da yapay nöron gösterilmektedir.




**Şekil 2.5 :** Nöron [24]




**Şekil 2.6 :** Yapay sinir ağı [24]

YSA çok katmanlı yapılardan oluşmaktadır. Bu katmanlar: Giriş katmanı, gizli katman(lar) (Yoğunluk katmanları olarakta bilinmektedir) ve çıkış katmanıdır. YSA ile sınıflandırma, kontrol sistemleri ve regresyon problemleri gibi sorunların üstesinden gelinmektedir [25].

YSA'daki temel yaklaşım eğitilmemiş verilerin ağa aktarılmasından sonra, ağ girdi verilerini belirli ağırlıklar ile güncellemektedir. Ağ eğitimini bu güncelleme iterasyonları sonunda tamamlamaktadır. Tıpkı canlıların deneme yanılma yöntemi ile bir şeyleri öğrenebilmesi gibi YSA'da yanılgılarını azaltmaya çalışarak (hatayı minimize ederek) öğrenmeyi başarmaktadır. Katman sayısının artması eğitim başarısını artırmaktadır [26]. Ama katman yoğunluğunun artması da eğitim sürecine ve maliyete olumsuz etkiye sebep olabilir.

### 2.4.2 Gözetimsiz öğrenme

Gözetimsiz öğrenme, makine öğrenmesi alanında kullanılan bir tekniktir. Gözetimsiz öğrenmede etiketlenmiş veriye ihtiyaç duyulmaz. Çünkü gözetimsiz öğrenme algoritmaları, veri setindeki örüntüleri keşfetmeye yönelik çalışır ve bu şekilde benzerlikleri gruplandırmayı hedeflemektedir. Bu gruplandırma işlemine de kümeleme (clustering) adı verilmektedir.

### 2.4.3 Takviyeli öğrenme

Makine öğrenmesi alanında kullanılan bir başka teknik, takviyeli öğrenmedir. Genellikle robotik ve oyun programlama da kullanılan bu tekniğin amacı, sistemin bulunan ortamı algılayıp kendi başına karar alıp bu kararlar sonucunda doğru sonuca ulaşmasını sağlamaktır [27]. Ödül ve ceza sistemine dayalı olarak geliştirilen bu yöntem ile sistemin doğru karar alabileceği bir duruma getirilmesi hedeflenmektedir.

### 2.4.4 Derin öğrenme

Derin öğrenme, büyük miktardaki veriler üzerinde karmaşık modellerin eğitilmesini sağlayan ve yüksek seviyede öznitelik çıkarımını gerçekleştirebilen bir makine öğrenimi yaklaşımıdır. Şekil 2.5 ve Şekil 2.6'da görüldüğü üzere, insan sinir ağlarını ilham alan YSA kullanmaktadır.

Derin öğrenme, önceki katmanlarda öğrenilen temel özellikleri sonraki katmanlarda birleştirerek, geri yayılım (Back propagation) algoritmasının kullanımı ile daha karmaşık

özniteliklerin çıkarılmasını sağlamaktadır. Derin öğrenme elde etmiş olduğu başarı ile özellikle görüntü tanıma, doğal dil işleme, ses tanıma, otonom sürüş vb. gibi alanlarda tercih edilmektedir.

### 2.4.5 Transfer öğrenme

Eğitilmiş bir ağın, ilgili ikinci bir görevde yeniden tasarlanma yaklaşımı olan makine öğrenmesi tekniğine transfer öğrenme denir [28].

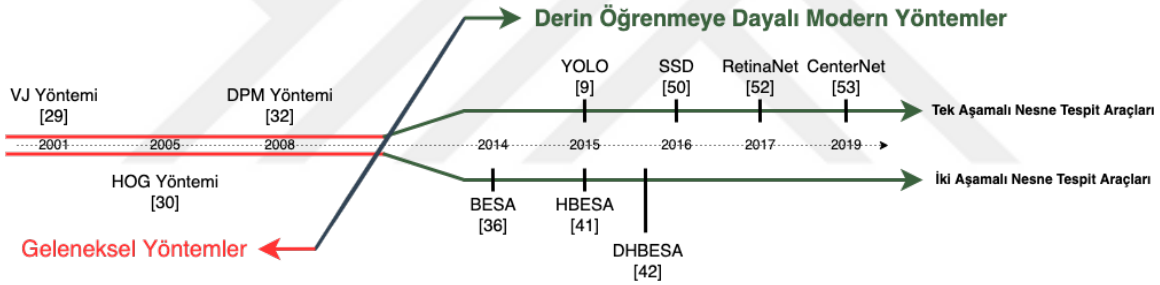
Geleneksel makine öğrenmesi yöntemleri, problemin çözülmesi için o probleme uygun bir model dizaynı (tasarımı) ve model eğitimini önermekteydi. Ancak transfer öğrenme, herhangi bir problem için oluşturulan modelin bir diğer problemin çözümünde kullanılmasına olanak sağlamaktadır. Nesne tanıma ve nesne tespit yöntemleri için literatür incelendiğinde, birçok yöntemin ImageNet gibi büyük görüntü veri setlerinden elde edilen özelliklerle nesne tespiti yaptığı gözlemlenmektedir. Transfer öğrenme çeşitli şekillerde gerçekleştirilir:

- Önceden eğitilmiş modelin kullanımı: Herhangi bir problem için eğitilmiş olan bir model, ağırlık ve parametreleriyle birlikte başka bir problem çözümüne uyarlanabilir.
- Özellik çıkarımı (Feature extraction): Herhangi bir problem için eğitilmiş bir modelin özellik çıkarım katmanları, başka bir problem çözümüne aktarılabilir.
- Özellik inceleme (Fine-Tuning): Herhangi bir problem için eğitilmiş olan modelin ağırlıkları, hedef görev için tekrar ayarlanabilir. Modelin bazı katmanları hedef probleme özgü veriyle yeniden eğitilirken, diğer katmanlar kaynak problemin bilgilerini koruyabilir.

# 3. NESNE TAKİP VE TESPİT YÖNTEMLERİ

## 3.1 Literatürde Bulunan İlgili Çalışmalar

Video veya görüntüdeki nesne tanıma ve nesne tespiti son yıllarda sağlık, trafik, ticaret, güvenlik vb. gibi alanlarda popüler bir çalışma alanı olmuştur [29]. Literatür incelendiğinde geliştirilen nesne tespit algoritmaları, iki ana başlık altında incelenmektedir. Bunlar: geleneksel yöntemler ve derin öğrenmeye dayalı modern yöntemler olarak adlandırılmaktadır. Geleneksel yöntemler, nesne tanıma ve nesne tespit problemlerinin çözülmesi fikrinin temelini oluşturmaktadır. İlk nesne algılama algoritmalarının çoğu, etkili bir şekilde görüntüden öznitelik çıkarımı yapamamaları sebebiyle el yapımı özelliklere dayalı olarak çözüm üretmek zorunda kalmışlardır [30]. Gelişen teknoloji ile daha rahat erişilebilen GPU donanımlar sayesinde derin öğrenmeye dayalı modern nesne tanıma ve tespit algoritmaları günümüzde gelişimine devam etmektedir. Modern nesne tespit algoritmaları sayesinde nesne algılama problemleri hızlı bir şekilde sonuca ulaşmaktadır.




**Şekil 3.1 :** Nesne tespit algoritmaları [30]

### 3.1.1 Geleneksel yöntemler

Viola ve Jones [31] 2001 yılında, yüz tespiti yapabilmek için önceden elle etiketlendirilip hizalandırılan $24 * 24$ boyutlarında biçimlendirdikleri yüz görüntüleri ile ağa sunulan görüntü üzerinde bulunan yüzleri tespit etmeyi başarmışlardır.

Dalal ve Triggs [32] 2005 yılında yaptıkları çalışma ile, "Yönlendirilmiş Gradyan Histogramı" (Histogram of Oriented Gradients - HOG) adını verdikleri yöntemi literatüre sunmuşlardır. HOG algoritması basitçe, görüntüyü gradyanlarına ayırarak gereksiz özellikleri göz ardı edip asıl görevi olan nesne tespit işlemine odaklanmayı hedefleyen bir yöntem olarak açıklanmaktadır. Nesne tespiti yaparken 3. boyut olan renkleri göz ardı etmektedir. Eski bir algoritma olmasına rağmen hassasiyeti sebebiyle günümüzde hala popülerliğini koruyan bu algoritma, özellikle sürücüsüz kullanılabilen araçlarda ve X-Ray

görüntülerdeki tümör tespiti gibi sağlık alanındaki çalışmalarda kullanılmaya devam etmektedir [33].

Felzenszwalb ve arkadaşları [34] tarafından 2008 yılında HOG algoritmasının bir uzantısı olarak ortaya çıkarılan Deforme Edilebilir Parça Modeli (Deformable Part Model - DPM), esasen "böl ve fethet" mantığını taşımaktaydı [30]. 2009 yılında PASCAL VOC yarışmasının galibi olan DPM, öncelikle görüntülerdeki nesneleri tek tek ve parça parça algılar [33]. Çalışma mantığını örneklemek gerekirse: ağa verilen görüntünün içerisinde, eğer bir kuyruk, dört bacak tespit edilirse bu görüntüde 'hayvan olma ihtimali daha yoğundur' söylemi yapılabilir. Nesne algılamadaki başarısı sebebiyle derin öğrenme yöntemlerinin temelini oluşturmaktadır.

### 3.1.2 Derin öğrenmeye dayalı modern yöntemler

Literatürde bakıldığında, modern nesne tespit algoritmaları iki başlıkta incelenmektedir: Tek aşamalı tespit araçları ve iki aşamalı tespit araçları.

#### 3.1.2.1 İki aşamalı tespit araçları

Derin öğrenmenin gelişimiyle ortaya çıkan Evrişimli Sinir Ağları (ESA), derin ağların temel yapı taşı haline gelmiştir [35]. İlk kez ESA modeli, 1989 yılında LeCun ve arkadaşları [36] tarafından el yazımı harflerin tahminindeki başarı ile ortaya çıkan LeNet ağıdır [37]. ESA, harf tahminindeki başarımı sebebiyle modern nesne tanıma ve tespit yöntemlerine öncülük etmiştir.

2014 yılında Girshick ve arkadaşları [38] Bölge Tabanlı Evrişimsel Sinir Ağı'nı (BESA) önermişlerdir. BESA, ağa verilen görüntü içerisinde Seçici Algoritma [39] kullanılarak nesne olmaya aday parçaların üzerinde bölge önerileri (~2000) oluşturmaktadır [40]. Oluşturulan bu bölge önerileri özellik çıkarımı yapılabilmesi için AlexNet [41] üzerinde önceden eğitilmiş ESA ağına aktarılır ve sınıflandırılır. BESA modeli ile nesne tanıma ve tespit problemi aslında bir sınıflandırma problemine dönüşmektedir. PASCAL VOC07'de ortalama hassasiyet (mAP) skoru %58.5 olan BESA ile nesne tanıma ve tespit problemi büyük ilerleme kaydetmiş olsada bölge öneri kutuları oluşturulurken gereksiz özellik haritaları ortaya çıkmasından ötürü maliyetli bir yöntem olmuştur [30].

Girshick 2015 yılında, BESA [38] ve SPPNet'in [42] daha gelişmiş versiyonu olan Hızlı Bölge Tabanlı Evrişimsel Sinir Ağı (HBESA) [43] önermiştir. BESA modelinde ~2000 adet aday bölgenin tek tek bir ESA ağ modeline girdi olarak verilmesi oldukça maliyetli bir

şekilde sonuca ulaştırmaktaydı. HBESA modelinde, SPPNet'te bulunan havuzlama katmanlarındaki yapı, ilgi alanı havuzu (Region of Interest - RoI) katmanı ile değiştirilmektedir. HBESA ağında RoI katmanı sayesinde, görüntü üzerinde yalnızca bir kez özellik haritaları çıkartılır ve sınıflandırıcı bir kez çalıştırılır. Bu yöntem sayesinde maliyet oldukça azalmaktadır. BESA ağının PASCAL VOC07'de mAP skoru %58.5 iken, HBESA ağı ile mAP skoru %70'lere kadar çıkarılmıştır.

2015 yılında Ren ve arkadaşları, HBESA'dan kısa bir süre sonra Daha Hızlı Bölge Tabanlı Evrişimsel Sinir Ağı (DHBESA) [44] ağını önermişlerdir. DHBESA ağı, özünde daha hızlı sonuca ulaşıp maliyeti daha da düşürmek için bölge öneri ağını (Region proposal network - BÖA) önermiştir. BÖA, görüntüye bağlı olarak nesne önerileri üretmektedir. Bu öneriler daha sonra sınıflandırılma ve sınırlayıcı kutu (bounding box) regresyonu sağlanması için ESA ağına aktarılmaktadır [45]. DHBESA modelinin sağladığı avantaj, BÖA ile doğrudan özellik haritasının bulunduğu katmana bağlanabilmesidir [46]. Katmanların karmaşıklığı sebebiyle eğitim süresinin uzamasının yanı sıra başarım oranı önceki nesne tanıma ve tespit algoritmalarına göre artmıştır.

O zamanlarda görüntü üzerindeki küçük nesneleri tespit etmek oldukça karmaşık bir problemdi. Lin ve arkadaşları tarafından önerilen özellik piramit ağları (Feature Pyramid Networks - FPN) [47] ile bu problemin çözüme ulaşması sağlanmıştır. FPN, farklı ölçekteki özellik haritalarını oluşturmakta ve birleştirmektedir. Bu sayede nesnelerin farklı ölçeklerdeki detaylarını ele almak ve nesne tespit modelinin hem küçük hem de büyük nesneleri doğru bir şekilde tespit etmesini sağlamak mümkün kılınmıştır. FPN, bilgiyi yüksek seviyelerden alt seviyelere yaymak amacıyla yukarıdan aşağı piramit biçiminde bir ağ modelidir ve ismini mimarisinden almaktadır [48]. FPN ağ modeli nesne tespiti yapabilmek için ResNet-101 [49] tabanlı DHBESA ağını kullanmaktadır [33]. ResNet [49] yapısında "artık" (residual) adı verilen bloklardan oluşmakta olup, bu bloklar sayesinde başarısız eğitim ile karşılaşsa bile en kötü durumda bir önceki eğitiminden devam etmektedir [50].

#### 3.1.2.2 Tek aşamalı tespit araçları

YOLO (Yalnızca Bir Kere Bak – You Only Look Once) ilk kez 2015 yılında Redmon ve arkadaşları [11] tarafından önerilmiştir. YOLO, bölge önerisi algılama sistemini değiştirerek nesne tanıma ve tespit problemlerini yeniden biçimlendirmiştir. BESA serisinin (BESA, HBESA, DHBESA) bölge önerileri oluştururken önemli sorunlarından biri bölge

önerilerinin örtüşmesiydi ve bu sebeple modelin tekrar tekrar çalışması gerekmekteydi [51]. İki aşamalı tespit araçları genellikle nesne tespit problemini bir sınıflandırma problemine dönüştürüp çözmeyi hedeflerken YOLO, nesne tanıma ve tespit problemini bir regresyon problemine dönüştürmeyi hedeflemektedir. YOLO'nun en büyük avantajı ise sahip olduğu hızıdır. Bunun sebebi ise görüntüyü bir kez ağdan geçirmesidir. Ağdan geçen görüntü, önce bölgelere ayrılır ardından herbir bölge içerisindeki sınırlayıcı kutuları ve sınıfların olasılık vektörlerini tek seferde tahmin etmeyi sağlamaktadır. Küçük nesnelerin tespiti konusunda dezavantajlı olması sebebiyle, YOLO'nun ilerleyen versiyonlarında bu sorunu gidermeye yönelik çalışmalar yapılmıştır.

Liu ve arkadaşları tarafından 2016 yılında SSD (Single Shot Multibox Detector – Tek Atış Çoklu Kutu Dedektörü) [52] önerilmiştir. SSD, performansını artırmak için VGG-16 [53] ağ modeli üzerine inşa edilmiştir [33]. VGG-16 ağ modelinin kullanılmasının sebebi, yüksek çözünürlüğe sahip görüntüleri sınıflandırmadaki performansıdır. SSD, YOLO ve DHBESA gibi iki farklı türdeki derin öğrenmeye dayalı nesne tespit araçlarından daha hızlı ve doğru bir araç olduğunu göstermiştir. Küçük nesneleri tespit etme problemi ilerleyen zamanlarda VGG-16'dan daha büyük mimariye sahip ResNet [49] ağına geçilerek düzeltilmiştir.

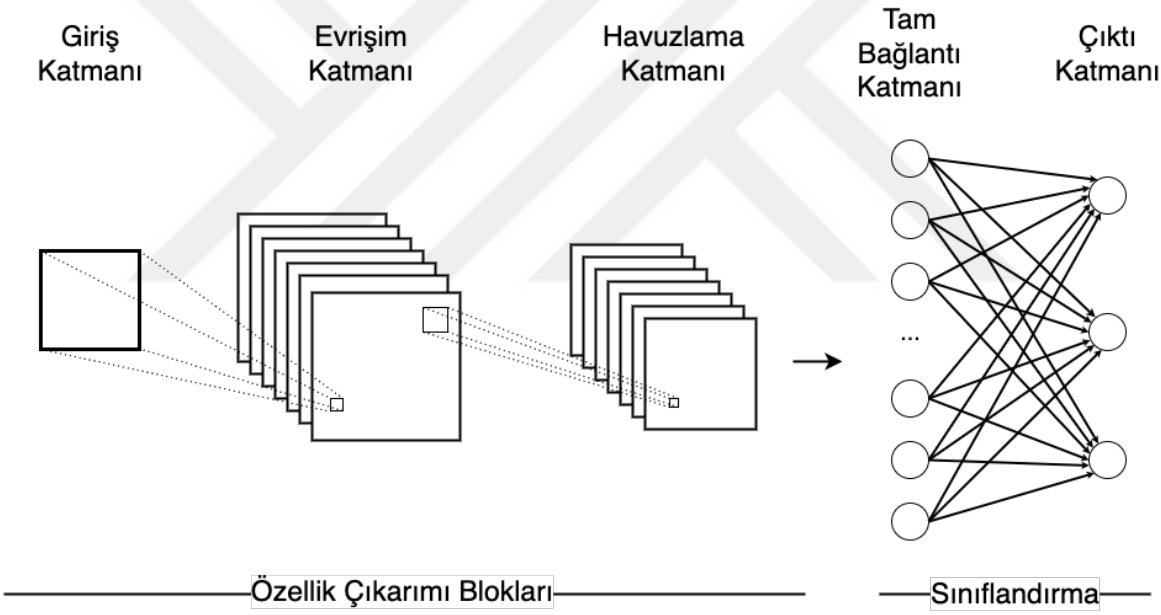
Nesne tespit yöntemlerinde yıllarca karşılaşılan hatanın küçük nesneler olmasından dolayı, Lin ve arkadaşları 2017 yılında RetinaNet'i [54] önermişlerdir. RetinaNet'in ana bileşeni "Focal Loss" olarak adlandırılan odak kaybıdır. Tek aşamalı tespit araçlarında, nesne tanıma ve tespit problemi çözülmeye çalışılırken pozitif ve negatif örnek sayılarında dengesizlik meydana gelmektedir (Sınıf dengesizliği). Bu sorunun çözülmesi için odak kaybı etkili olmuştur. Odak kaybı, baskın örneklere farklı ağırlıklar atamakta olup, sınıf dengesizliği sorununu bu şekilde çözmeyi başarmaktadır. O dönemlerde RetinaNet, bazı tek aşamalı tespit araçlarında kullanılan algoritmaların hız ve doğruluk açısından önüne geçmiştir [33].

2019 yılına gelindiğinde ise Zhou ve arkadaşları [55], o zamana kadar nesne tanıma ve tespit algoritmalarında bulunan ve gelenekselleşmiş hale gelen "sınıflayıcı kutu" kavramını değiştirerek, nesneleri noktalarla modellemeyi önermişlerdir. CenterNet [55], nesne tespiti yaparken merkez tabanlı bir yaklaşım sergiler ve ismini buradan almaktadır. CenterNet, gereksiz tahminleri ortadan kaldırıp, nesneyi daha hızlı ve doğru bulabilmek için ısı haritası kullanmaktadır. Kullandığı ısı haritası ile nesnenin merkezini bulmayı

amaçlamaktadır. Özellikle 3 Boyutlu görüntülerde nesne tespiti yapılması için tercih edilen bir yöntem olmuştur.

### 3.2 Evrişimsel Sinir Ağları

Derin öğrenme, bir makine öğrenmesi yaklaşımıdır ve son yıllarda bilgisayar bilimcilerin çalışmalarını yoğunlaştırdığı popüler bir çalışma alanıdır [56]. Derin öğrenme yöntemlerinden biri olan evrişimsel sinir ağları (ESA), insan beyninin görsel işleme sisteminden esinlenilerek geliştirilmiştir. ESA, özellikle görüntü işleme ve desen tanıma gibi alanlarda sahip olduğu başarılı sonuçlar ile literatürde sıklıkla karşılaşılan bir modeldir [57]. ESA modeli 5 adet katmandan oluşur. Bu katmanlar giriş katmanı, evrişim katmanı, havuzlama katmanı, tam bağlantı katmanı ve çıkış katmanıdır [50]. Klasik bir ESA yapısı Şekil 3.2'de gösterilmiştir.




**Şekil 3.2 :** ESA mimari

- Giriş katmanı:

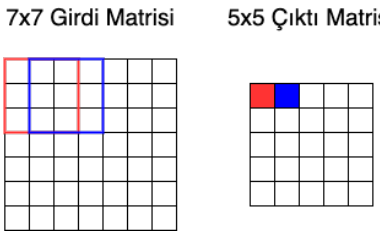
  İşlenmek istenilen ham verinin modele aktarıldığı katmandır.

- Evrişim katmanı:

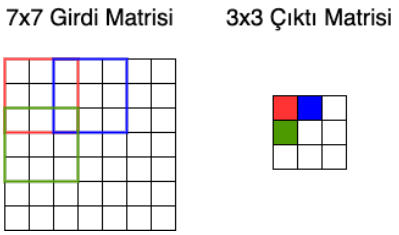
ESA'nın temelini oluşturan katmandır [58]. Bu katmanda girdi verisi üzerine filtreler (Belirli boyutlara sahip ağırlık matrisleri) uygulanarak özellik haritaları elde edilir. Genellikle $2 * 2$ ya da $3 * 3$ boyutlarında olan bu filtreler, girdi verisi üzerinde hangi şekilde gezinmesi gerektiğini iki parametre ile belirlenmektedir. Bu parametrelerden biri adım

(stride) diğeri ise dolgudur (padding). Bu iki teknik ile modelin daha iyi performans ve doğru sonuçlar vermesi sağlanmaktadır.

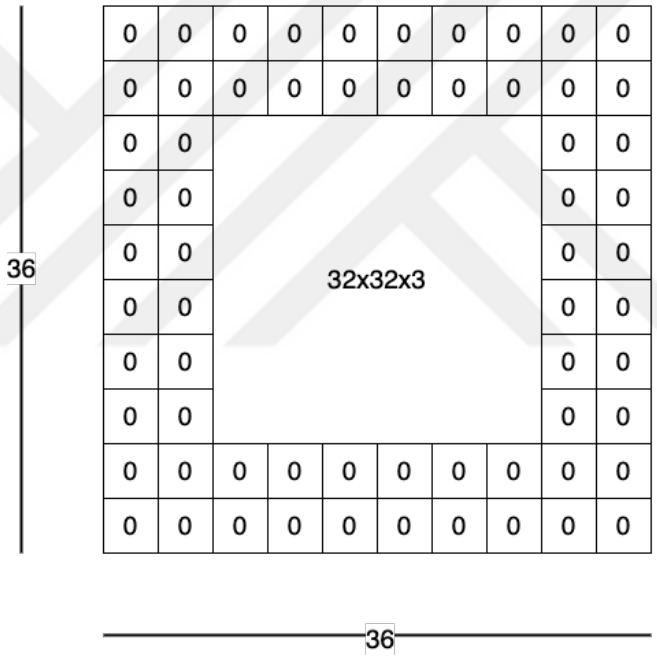
Adım tekniği için örnekler Şekil 3.3'te ve Şekil 3.4'te gösterilmiştir, dolgu tekniği için örnek şekil 3.5'te gösterilmiştir.




**Şekil 3.3 :** Adım 1 [59]




**Şekil 3.4 :** Adım 2 [59]

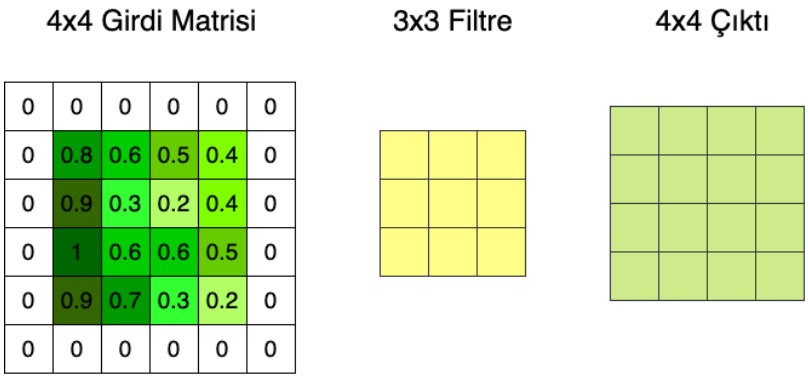



**Şekil 3.5 :** Dolgu [60]

Adım (Stride): Filtrenin girdi verisi üzerinde kaç birim kaydırılacağını belirler. Şekil 3.3'te adım sayısı 1 olarak belirlenmiştir. $7 * 7$'lik bir görüntü üzerinde adım 1 olarak belirlendiğinde oluşacak çıktı görüntüsünün boyutu $5 * 5$ olacaktır.

Dolgu (Padding): Girdi verisinin boyutunu korumak veya istenilen boyutta çıktı elde edebilmek için kullanılan bir tekniktir. Dolgunun iki yaygın türü bulunmaktadır. Bunlar: sıfır dolgusu (Zero padding) ve kenar dolgusudur (Edge padding).

Sıfır Dolgusu: Girdi görüntüsünün boyutunu sabit tutmak için kullanılan yöntemdir. Bu yöntemle girdi görüntüsünün çevresine Şekil 3.6'da gösterildiği gibi sıfır (0) değerli pikseller eklenerek yapılır.



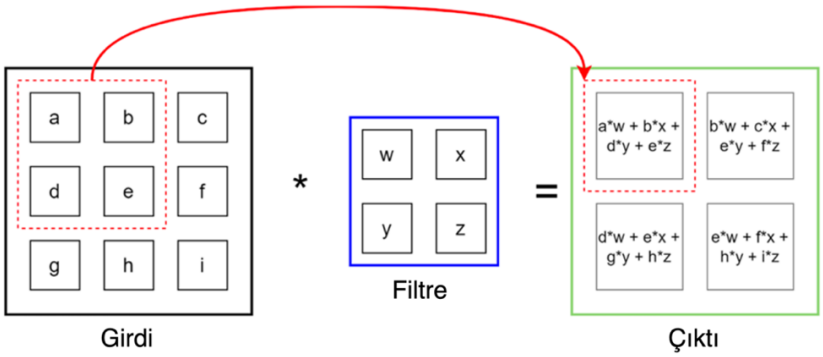

**Şekil 3.6 :** Zero padding [61]

Kenar Dolgusu: Girdi görüntüsünün çevresine, genellikle girdi verisinin kenarlarında bulunan piksel değerlerinin aynıları eklenerek uygulanmaktadır. Bu şekilde girdi görüntüsünün boyutu artmakta olup içerdiği bilgi aynı kalmaktadır. Kenar dolgusu, çıktı verisinin boyutunu korumak veya istenilen boyutta çıktı verisi elde etmek için kullanılmaktadır.

Girdi verisinin matris boyutu $n * n$, uygulanan filtrenin matris boyutu $k * k$, kaydırılacak adım sayısı $s$ ve uygulanan dolgu $p$ ile gösterilirse elde edilecek olan çıktı verisinin matris boyutu Denklem 3.1'deki gibi hesaplanmaktadır.

$$\left[\frac{(n+2p-f+1)}{s}\right] * \left[\frac{(n+2p-f+1)}{s}\right] \tag{3.1}$$

Girdi verisi üzerine filtreler kartezyen çarpılarak çıktı verisi elde edilir. Bu işleme evrişim adı verilmektedir. Şekil 3.7'de bu işlemin nasıl yapıldığı gösterilmiştir.




**Şekil 3.7 :** Evrişim işlemi [5]

$3 * 3$ 'lük bir girdi verisi matrisine uygulanan $2 * 2$'lik filtre ile elde edilen çıktı verisi matris boyutu Denklem 3.1'de verilen denklem ile hesaplandığında $2 * 2$ boyutu elde edilmektedir.

Evrişim işleminden sonra çıktı verisi üzerinde belirlenen bir aktivasyon fonksiyonu uygulanır. Bunun sebebi çıktı verilerini belirli (sonlu) bir aralığa getirip anlamlı sonuçlar elde etmektir. Literatür incelendiğinde, yaygın olarak Sigmoid, Tanh, ReLU ve Softmax aktivasyon fonksiyonlarının kullanıldığı görülmektedir [62, 63].

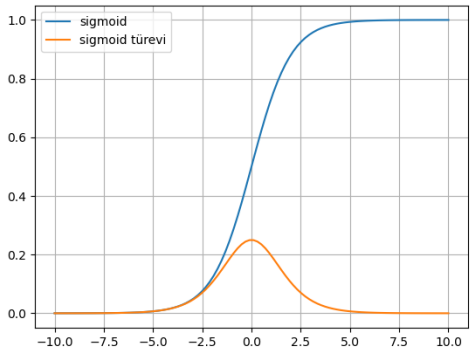
Sigmoid Aktivasyon Fonksiyonu: Doğrusal olmayan bir fonksiyondur. Bu sebeple en yaygın olarak kullanılan aktivasyon fonksiyonudur [64]. Sigmoid Fonksyionu, [0,1] aralığında çıktı üretir. Eşitliği Denklem 3.2'de gösterilmiştir.

$$f(x) = 1/(1 + e^{-x}) \tag{3.2}$$

Sigmoid Fonksiyonu türevlenebilir bir fonksiyondur ve türevlenmiş eşitliği Denklem 3.3'te gösterilmiştir.

$$f'(x) = 1 - sigmoid(x) \tag{3.3}$$

Sigmoid Fonksiyonun grafiği Şekil 3.8'de, Python dili ile kodları Şekil 3.9'da gösterilmiştir.




**Şekil 3.8 :** Sigmoid grafik

```python
import numpy as np
import matplotlib.pyplot as plt

def sigmoid(x):
    return 1/(1 + np.exp(-x))

def derivative_sigmoid(x):
    return sigmoid(x) * (1- sigmoid(x))

x_data = np.linspace(-10,10,100)
y_data = sigmoid(x_data)
dy_data = derivative_sigmoid(x_data)

plt.plot(x_data, y_data, x_data, dy_data)
plt.legend(['sigmoid','sigmoid türevi'])
plt.grid()
plt.show()
```

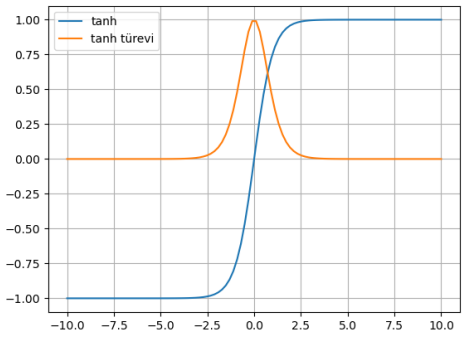
**Şekil 3.9 :** Sigmoid kod

Tanh Aktivasyon Fonksiyonu: Sigmoid Fonksiyonuna benzerdir fakat Tanh Fonksiyonunun üreteceği değer aralığı (-1,1)'dir. Tanh Fonksiyonu sürekli ve türevlenebilirdir [8]. Tanh aktivasyon fonksiyonunun eşitliği Denklem 3.4'te, türevlenmiş eşitliği Denklem 3.5'te sunulmuştur.

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{3.4}$$

$$f'(x) = (1 - g(x)^2) \tag{3.5}$$

Tanh Fonksiyonun grafiği Şekil 3.10'da, Python dili ile kodları Şekil 3.11'de gösterilmiştir.




**Şekil 3.10 :** Tanh grafik

```python
import numpy as np
import matplotlib.pyplot as plt

def tanh(x):
  return (np.exp(x) - np.exp(-x))/(np.exp(x) + np.exp(-x))

def derivative_tanh(x):
  return 1 - tanh(x) * tanh(x)

x_data = np.linspace(-6,6,100)
y_data = tanh(x_data)
dy_data = derivative_tanh(x_data)

plt.plot(x_data, y_data, x_data, dy_data)
plt.legend(['tanh','tanh türevi'])
plt.grid()
plt.show()
```

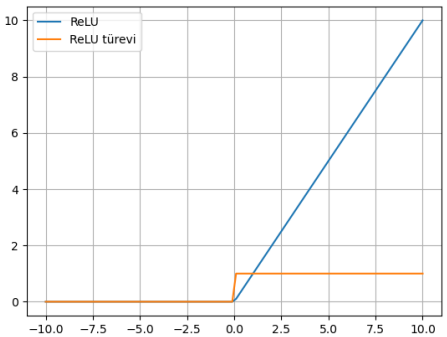
Şekil 3.11 : Tanh kod

ReLU Aktivasyon Fonksiyonu: Gelen girdi değeri 0'dan büyük ise aynı değer değişmeden çıktı olarak kabul edilir, 0'dan büyük değilse çıktı değeri 0 kabul edilir. ReLU fonksiyonun eşitliği Denklem 3.6'da, türevlenmiş eşitliği Denklem 3.7'de sunulmuştur.

$$f(x) = max(0, x) \tag{3.6}$$

$$f'(x) = \begin{cases} 1, x > 0 \\ 0, diğer \end{cases} \tag{3.7}$$

ReLU Fonksiyonun grafiği Şekil 3.12'de, Python dili ile kodları Şekil 3.13'te gösterilmiştir.




**Şekil 3.12 :** Relu grafik

```python
import numpy as np
import matplotlib.pyplot as plt

def ReLU(x):
  data = [max(0,value) for value in x]
  return np.array(data, dtype=float)

def derivative_ReLU(x):
  data = [1 if value>0 else 0 for value in x]
  return np.array(data, dtype=float)

x_data = np.linspace(-10,10,100)
y_data = ReLU(x_data)
dy_data = derivative_ReLU(x_data)

plt.plot(x_data, y_data, x_data, dy_data)
plt.legend(['ReLU','ReLU türevi'])
plt.grid()
plt.show()
```

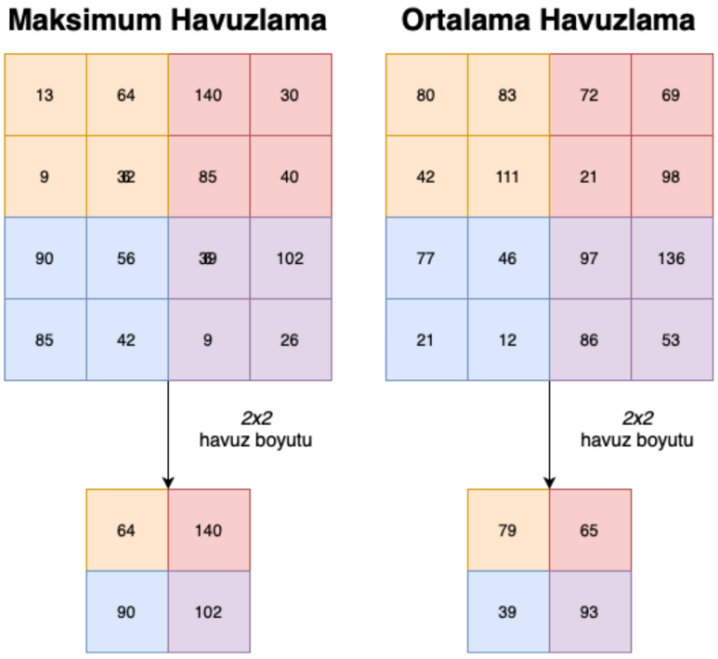
**Şekil 3.13 :** Relu kod

Softmax Aktivasyon Fonksiyonu: Makine öğrenmesindeki çok sınıflı sınıflandırma problemlerinde yaygın olarak kullanılan bir aktivasyon fonksiyonudur. Softmax genellikle sinir ağlarının son katmanlarında, çıktıların etiketlenmesi için kullanılmaktadır [65]. Eşitliği Denklem 3.8'de sunulmuştur.

$$softmax(Z_i) = \frac{exp(Z_i)}{\sum exp(Z_i)} \tag{3.8}$$

- Havuzlama katmanı:

Ağ karmaşıklığını ve hesaplama maliyetini azaltmak için evrişim katmanlarından elde edilen özellik haritalarının boyutunu küçültmeyi amaçlayan yapıdır. Bunun sebebi maliyeti azaltmaktır. Görüntü içerisindeki önemsiz özellikler elenir ve odaklanılması gereken önemli özelliklere odaklanma sağlanır. Havuzlama Katmanında, bir önceki katman olan evrişim katmanında olduğu gibi girdi verisi üzerinde belirli boyutlarda filtreler uygulanmaktadır. Bu filtrelerin evrişim katmanında bulunan filtrelerden farkı havuzlama yapılmasıdır [66]. Havuzlama yapabilmek için kullanılan iki yöntem bulunmaktadır. Bunlardan birisi maksimum havuzlama (Max pooling - MP), diğeri ortalama havuzlamadır (Average pooling - OP). Şekil 3.14'te bu iki yöntem belirtilmiştir.



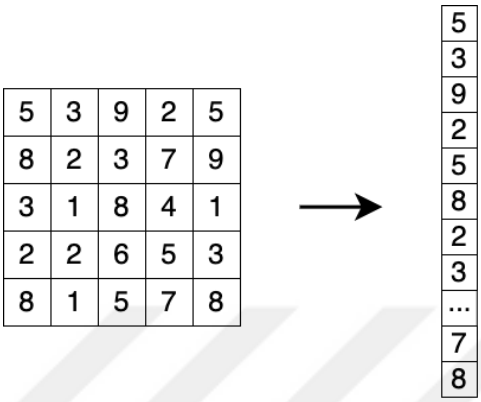

**Şekil 3.14 :** Havuzlama [67]

Maksimum Havuzlama: Geçerli matris üzerindeki en büyük değeri alır.

Ortalama Havuzlama: Geçerli matris üzerindeki ortalama değeri alır.

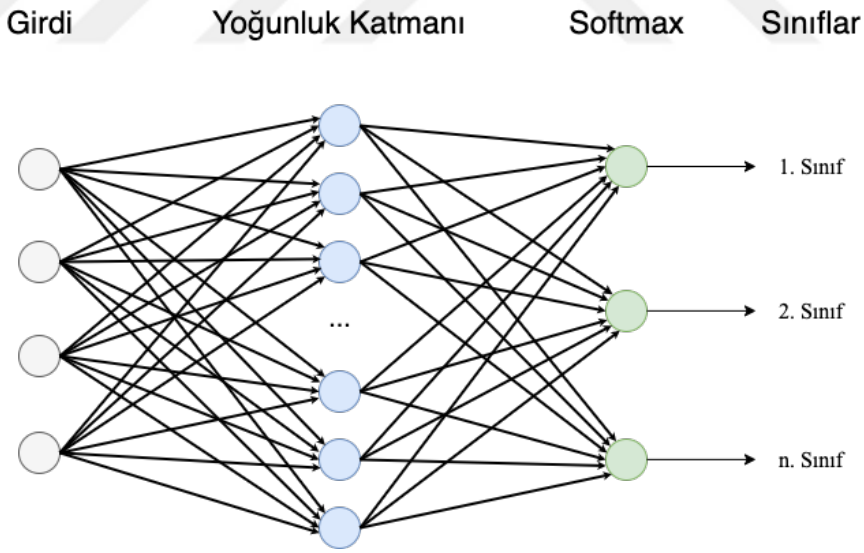
- Tam bağlantı katmanı:

Tam bağlantı katmanı öncelikle, önceki katmanlardan elde edilen öznitelik matrislerini bir vektöre dönüştürmek için düzleştirme işlemi gerçekleştirmektedir [68]. Elde edilen vektör ile sınıflandırma, tahmin veya başka bir çıktı hesaplaması yapılmaktadır. Şekil 3.15'te düzleştirme işlemi gösterilmiştir.




**Şekil 3.15 :** Düzleştirme [67]

Özellik matrislerinin düzleştirilmesiyle artık nöronların eğitimi gerçekleştirilebilmektedir. Şekil 3.16'da tam bağlantı katmanı gösterilmektedir.




**Şekil 3.16 :** Tam bağlantı katmanı [67]

Tam bağlantı katmanı, nöronların geleneksel YSA biçimlerinde düzenlenmesine benzemektedir.

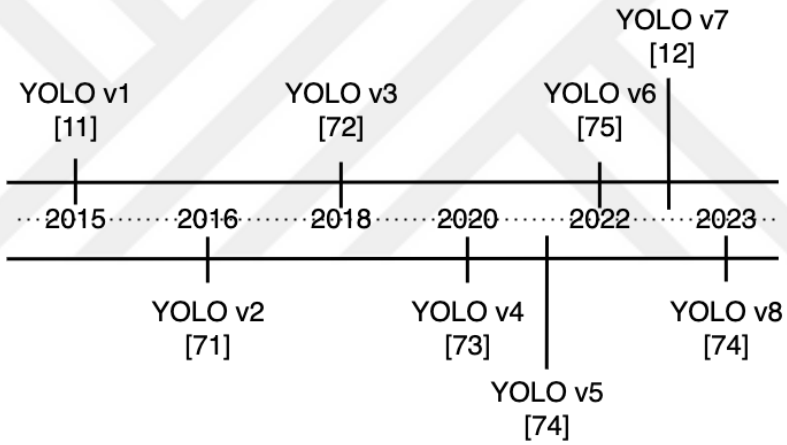
Şekil 3.16'da girdiler, düzleştirilmiş özellik matrisleridir. Bu yapı geleneksel YSA modeline benzemektedir [69]. Yoğunluk katmanlarında bulunan değerler ağırlık vektörleri ve bias değeri ile işlem görerek, eski değerler üzerinde güncelleme yapılmaktadır.

- Çıkış katmanı:

Gerçekleşen güncellemeler sonucunda ağ modeli girdi verisi üzerinde yapılacak işleme göre son katmanda bulunan Softmax aktivasyon fonksiyonu ile çıktı üretebilmektedir.

### 3.3 Yalnızca Bir Kere Bak

Yalnızca Bir Kere Bak (You Only Look Once - YOLO), bir ESA modeline verilen veri üzerinde sınırlayıcı kutuları doğrudan tahmin edebilen tek aşamalı nesne tespit aracıdır [10]. İlk kez 2015 yılında Redmon ve arkadaşları tarafından önerilen YOLOv1 [11], nesne tanıma ve nesne tespit problemlerinin çözümündeki hızı ile dikkat çekmiştir. Sahip olduğu bu hız, girdi verisinin yalnızca bir kere ESA modeline aktarılmasından kaynaklanmaktadır. YOLO yöntemine ait versiyonların hiyerarşisi Şekil 3.17'de gösterilmiştir.




**Şekil 3.17 :** 2023 itibariyle YOLO sürümlerinin hiyerarşisi [70]

YOLO ilk önerildiğinde, karmaşık görüntülerden (İçerisinde çok sayıda nesne bulunan) nesne tespiti problemlerinde ve küçük nesnelerin bulunmasında yetersiz kalmaktaydı. 2015'ten günümüz 2023'e kadar geldiğimiz süreç içerisinde bilgisayar bilimcileri tarafından YOLO geliştirilmeye devam etmiştir. YOLOv1'in [11] ardından YOLOv2 [71], YOLOv3 [72], YOLOv4 [73], YOLOv5 [74], YOLOv6 [75], YOLOv7 [12] ve son olarak YOLOv8 [76] geliştirilmiştir. Yapılan bu tez çalışmasının ilerleyen bölümlerinde herbir YOLO sürümü ayrıca açıklanacaktır. Çalışmamızda YOLOv7 tercih edilmiştir. Bunun sebebi, çalışmaya başladığımız zamanlardaki en son YOLO sürümünün YOLOv7 olmasından ve YOLOv7 tanıtıldığı dönemde COCO veri seti üzerinde yüksek ortalama hassasiyet ortalaması (Mean Average Presicison - mAP) skoru vermesinden kaynaklanmaktadır.

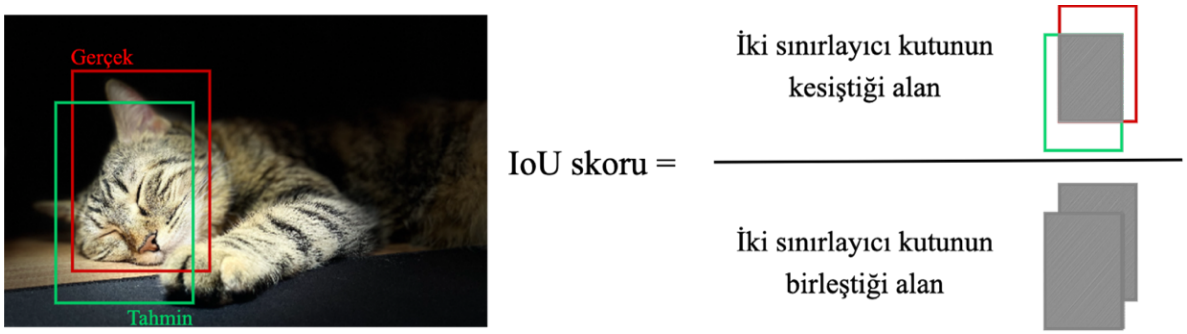
### 3.3.1 Yolo v1

Redmon ve arkadaşlarının 2015 yılında önerdikleri YOLO'nun [11] ilk versiyonu, o dönemde nesne tespit problemlerinde geleneksel yöntemlerin önüne geçmekteydi. Tek aşamalı olarak nesne tespit yapmayı amaçlayan bu yöntem aynı zamanda hız ve doğruluk açısından da başarılı sonuçlar elde etmekteydi. İlk sürüm olmasından kaynaklı dezavantajları vardı: görüntü veya video içerisinde bulunan küçük nesnelerin tespitinde başarılı değildi.

YOLOv1'in nesne tespit problemini çözebilmesi için öncelikle, girdi görüntüsünü $s * s$ lik ızgaralara (Genellikle $s = 7$ kabul edilir) ayırmaktadır. Herbir ızgara hücresi için güven skoru tahmini yapılmaktadır. Güven skoru ($GS$) tahmini için kullanılan eşitlik Denklem 3.9'da gösterilmiştir.

$$GS = p(Nesne) * IoU_{tahmin}^{gerçek} \tag{3.9}$$

Denklem 3.9'da gösterilen, $p(nesne)$ ifadesi nesnenin olma olasılığını ([0,1]) temsil etmektedir. Bu tahmini yaparken kaç adet sınıf olduğu önemlidir. Eğer birden fazla sınıf için $GS$ tahmin edilecekse, her sınıf için bu olasılık hesaplanmaktadır. $IoU_{tahmin}^{gerçek}$ ifadesi ise, nesnenin gerçekte var olduğu bölge ile tahmin edilen bölgenin kesişiminin bu iki bölgenin tamamına bölünmesiyle elde edilen değerdir. $IoU_{tahmin}^{gerçek}$ hesaplanması Şekil 3.18'de gösterilmiştir.




**Şekil 3.18 :** IoU hesaplaması

Her sınırlayıcı kutu ($B$), $GS$ ile birlikte, $x$ (Nesnenin orta noktasındaki yatay düzlem koordinatı), $y$ (Nesnenin orta noktasındaki yatay düzlem koordinatı), $w$ (Nesnenin genişliği), $h$ (Nesnenin yüksekliği) parametrelerini içermektedir. $(x, y)$ koordinatları, ızgara hücresindeki sınırlayıcı kutuya göre kutunun merkezini temsil etmektedir [11].

Her ızgara hücresi aynı zamanda C koşullu sınıf olasılıklarını ($tahmin(Sınıf_i\,|Nesne)$) da tahmin eder. Test zamanında ızgara sayısına bakılmaksızın

koşullu sınıf olasıkları ile sınırlayıcı kutu $GS$'leri çarpılır. Bu eşitlik Denklem 3.10'da gösterilmiştir.

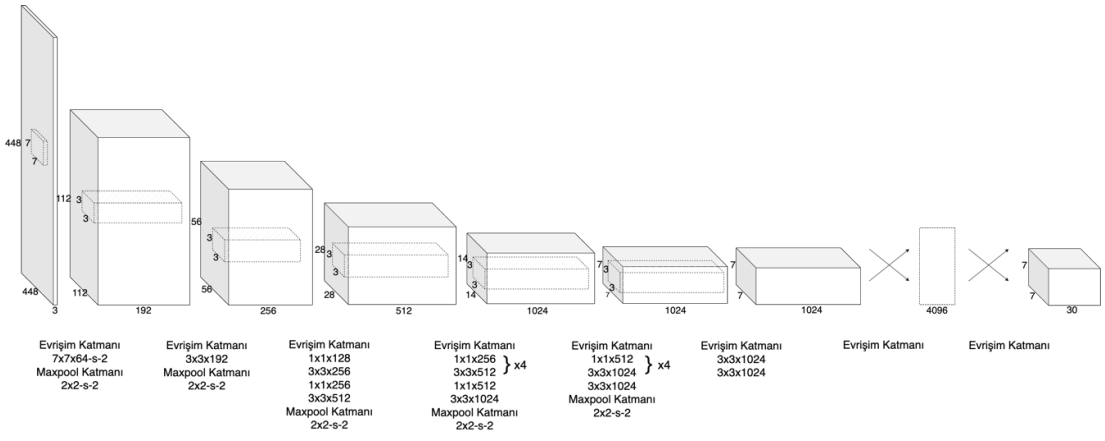
$$tahmin(Sınıf_i \mid Nesne) * tahmin(Nesne) * IoU_{tahmin}^{gerçek} = tahmin(Sınıf_i) * IoU_{tahmin}^{gerçek} \quad (3.10)$$

Denklem 3.10 bize her sınırlayıcı kutu için o sınıfa ait ($Sınıf_i$) $GS$ vermektedir. Bu $GS$, hem o sınıfın sınırlayıcı kutu içerisinde görünme olasılığını hem de tahmin edilen sınırlayıcı kutunun nesne için ne kadar doğru olduğunu göstermektedir.

Giriş görüntüsünün $s * s$ boyutlarında ızgara bölünmesi $A$, sınırlayıcı kutular $B$ ve sınıf sayısı $C$ ile temsil edildiğinde elde edilecek çıktı vektörü boyutu Denklem 3.11'de gösterilmiştir.

$$A * (B * 5 + C) \quad (3.11)$$

YOLO için kullanılan ESA modeli, GoogLeNet [77] ağından esinlenilerek hazırlanmıştır [11]. ESA ağları yapısı gereği girdi görüntülerini küçülterek ilerlemekte olduğu için YOLO'da girdi görüntüsü boyutları küçültülüp çözünürlük artırılmaktadır. YOLO, özellik haritalarının sayısını azaltmak ve parametre sayısını kısmen düşük tutmak için $1 * 1$ boyutlarında evrişim katmanlarını kullanmaktadır [70]. 24 evrişim katmanının ardından 2 tamamen bağlantı katmanına sahip olan tespit ağının mimarisi Şekil 3.19'de gösterilmiştir.




**Şekil 3.19 :** YOLOv1 mimarisi [11]

Redmon ve arkadaşları YOLO [11] ile yeni bir kayıp fonksiyonu önerdiler. Kayıp fonksiyonu nesne içermeyen kutuların önemini model üzerinde azaltmak için kullanılmaktadır. $\lambda_{koordinat}$ sınırlayıcı kutu tahminlerine önem veren bir ölçektir ve $\lambda_{zayıfj}$ ise nesne içermeyen kutuların önemini azaltan bir eşik ölçeği değeridir. $\lambda_{koordinat}$, tahmin

edilen sınırlayıcı kutu konumlarındaki $(x, y)$ koordinatlarını, boyutlarını $(w, h)$, $GS$ ve sınıf olasılığındaki hatayı hesaplar. Kayıp fonksiyonunun ilk kısmı Denklem 3.11 üzerinde incelenirse:

$$\alpha = \lambda_{koordinat} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{nesne} \left[ \left(x_i - \hat{x}_i\right)^2 + \left(y_i - \hat{y}_i\right)^2 \right] \qquad (3.11)$$

Denklem 3.11'de bulunan $\lambda_{koordinat}$ değeri tahmin edilen sınırlayıcı kutuların kayıp değerini artırmak için 5 olarak alınmaktadır. $\sum_{i=0}^{s^2}$ ifadesi her ızgara hücresi için, $\sum_{j=0}^{B}$ ifadesi her ızgarada bulunan sınırlayıcı kutu için değer hesaplamasını yapmaktadır. $\mathbb{1}_{ij}^{nesne}$ ifadesi nesnenin i'nci hücrede görünüyor ve j'inci sınırlayıcı kutu onu algılıyorsa 1, diğer tüm durumlarda 0 kabul edilmektedir. $x_i$, i'nci hücrede bulunan gerçek sınırlayıcı kutunun yatay düzlemdeki koordinatını temsil ederken, $y_i$ ise i'nci hücrede bulunan gerçek sınırlayıcı kutunun dikey düzlemdeki koordinatını temsil etmektedir. $\hat{x}_i$, i'nci hücrede bulunan tahmin edilen sınırlayıcı kutunun yatay düzlemdeki koordinatı temsil ederken, $\hat{y}_i$ ise i'nci hücrede bulunan tahmin edilen sınırlayıcı kutunun dikey düzlemdeki koordinatı temsil etmektedir. Denklem 3.11 en büyük IoU değerine sahip bir nesneyi tahmin etmekte kullanılmaktadır.

Kayıp fonksiyonun bir sonraki adımı, Denklem 3.11'e benzer biçimde sınırlayıcı kutuların hatasını hesaplamaktadır. $w$ ve $h$ (Genişlik ve yükseklik) değerlerinin 0 ve 1 aralığında normalleştirilmesi ve karekök farklarının daha yüksek değerlerde çıktığının bilinmesi ile bir sonraki adımda kullanılmaktadır. Kayıp fonksiyonunun ikinci kısmını Denklem 3.12'de incelenirse:

$$\beta = \lambda_{koordinat} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{nesne} \left[ \left(\sqrt{w_i} - \sqrt{\hat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2 \right] \qquad (3.12)$$

$w_i$, i'nci hücrede bulunan gerçek sınırlayıcı kutunun genişliğini temsil ederken, $h_i$ ise i'nci hücrede bulunan gerçek sınırlayıcı kutunun yüksekliğini temsil etmektedir. $\hat{w}_i$, i'nci hücrede bulunan tahmin edilen sınırlayıcı kutunun genişliğini temsil ederken, $\hat{h}_i$ ise i'nci hücrede bulunan tahmin edilen sınırlayıcı kutunun yüksekliğini temsil etmektedir. Karakök farklarının karesini almak yukarıda bahsedildiği üzere yüksek hata değerinin çıkmasından kaynaklanmaktadır. Bunun avantajı ise değer aralığını azaltmaktır.

Bir sonraki aşamada, nesnenin sınırlayıcı kutu içerisine varlığına bağlı olarak $GS$ hesaplanmaktadır. Denklem 3.13'te gösterilen eşitlik üzerinden incelenirse:

$$\gamma = \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{nesne} \left[ \left( c_i - \hat{c}_i \right)^2 \right] + \lambda_{zayıfj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{zayıfj} \left[ \left( c_i - \hat{c}_i \right)^2 \right] \quad (3.13)$$

$c_i$, gerçek $GS$'nu temsil ederken, $\hat{c}_i$ tahmin edilen $GS$'nu temsil etmektedir. $\gamma$ eşitliği iki parçadan oluşmaktadır. İlk parça, i'nci hücrede bir nesne algılandığında oluşan güven hatasını, ikinci parçada ise i'nci hücrede bir nesne algılanmadığında oluşan güven hatası belirtilmektedir. $\lambda_{zayıfj}$ ifadesi nesne içermeyen boş sınırlayıcı kutuların kaybını azaltmak için 0.5 olarak ayarlanmaktadır.

Kayıp fonksiyonunun son aşamasında da sınıflandırma kaybı hesaplanmaktadır. Denklem 3.14'te gösterilen eşitlik üzerinden incelenirse:

$$\delta = \sum_{i=0}^{S^2} \mathbb{1}_{ij}^{nesne} \sum_{c \in Sınıflar} [(p_i(c) - \hat{p}_i(c))^2] \quad (3.14)$$

Denklem 3.14'te gösterilen $p_i(c)$, i'nci hücrede görünen c sınıfının gerçek koşullu olasılığını temsil ederken, $\hat{p}_i(c)$ i'nci hücrede görünen c sınıfının tahmini koşullu olasılığını temsil etmektedir.

$\alpha$ ve $\beta$ eşitlikleri yerelleştirme kaybı fonksiyonları, $\gamma$ eşitliği güven kaybı fonksiyonları ve $\delta$ eşitliği sınıflandırma kaybı fonksiyonları olarak adlandırılmaktadır [70]. Elde edilen yerelleştirme kaybı fonksiyonu, güven kaybı fonksiyonu ve sınıflandırma kaybı fonksiyonunun toplamı, kayıp fonksiyonunu vermektedir ve eşitliği Denklem 3.15'te gösterilmiştir.

$$Kayıp\ Fonksiyonu = \alpha + \beta + \gamma + \delta \quad (3.15)$$

YOLOv1 [11], PASCAL VOC veri seti [78, 79] üzerinde gerçekleştirilen eğitim sonucunda 45 saniyedeki kare sayısı (Frequency per Second - FPS) eşiğinde aldığı mAP %63,4 almıştır [80].

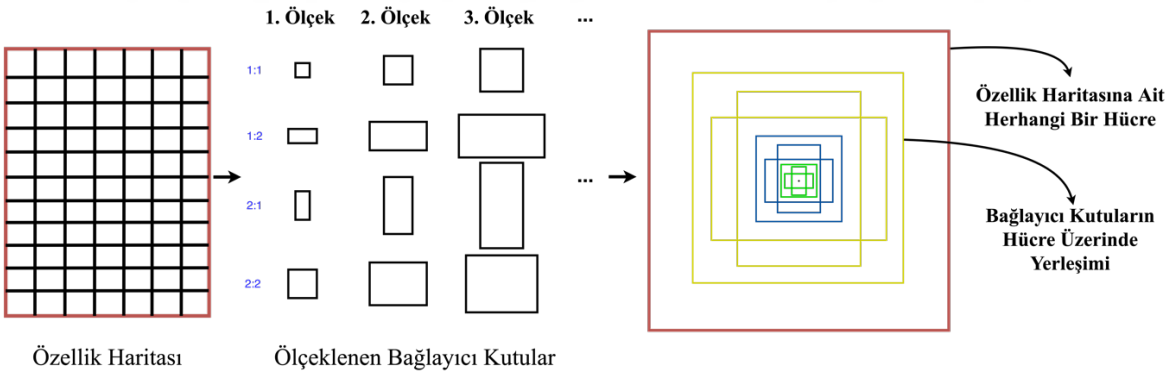
### 3.3.2 Yolo v2

YOLO'yu performans açısından daha etkili bir hale getirmek için Redmon ve arkadaşları tarafından YOLO9000 [71] adıyla YOLO'nun ikinci versiyonu yayınlanmıştır.

YOLO'nun ilk sürümü karmaşık görüntülerde nesne tespitinde ve küçük nesnelerin tespit edilmesinde başarılı değildi. YOLOv1'in eğitimi için kullanılan ImageNet, $224 * 224$ boyutlarında girdi görüntüleri almaktaydı ancak YOLOv2'nin eğitimi 10 adım (epochs) $448 * 448$ boyutlarında girdi görüntüleri ile sağlanmıştı [70]. Bu sayede girdi görüntülerine esneklik kazandırılmış olup farklı boyutlarda modele aktarılmaya imkan sağlamaktaydı.

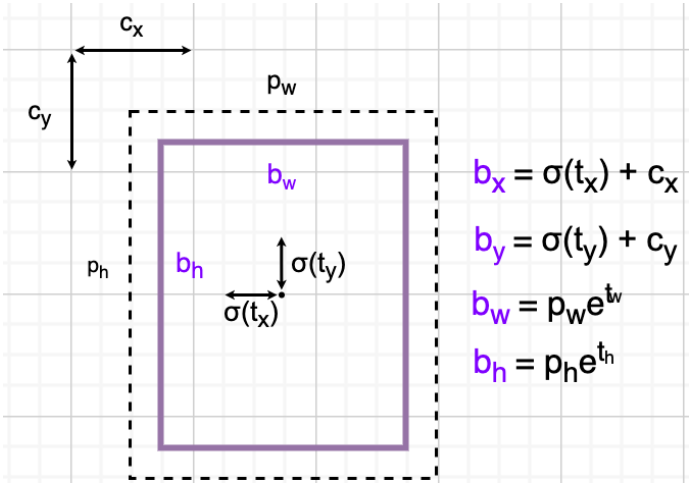
YOLOv2, VGG16 [53] ağ modeli yerine ImageNet üzerinde 1 milyondan fazla veri seti ile eğitilmiş ve özelleştirilmiş DarkNet-19 [71] ağını kullanmaktadır. DarkNet-19 ile YOLOv2 daha hızlı ve düşük maliyetle çalışabilmektedir.

YOLOv2, bağlayıcı kutular (Anchor boxes) kullanarak sınırlayıcı kutuların tahmin edilmesini önermiştir. Bağlayıcı kutular her bir özellik haritası hücreleri için birden fazla kez kullanılmaktadır. Birer çerçeve olarak düşünülen bağlayıcı kutular, düşük boyutlu özellik haritası hücreleri için düşük en-boy oranında kullanılırken, yüksek boyutlu özellik haritası hücreleri için yüksek en-boy oranına sahip bağlayıcı kutular kullanılmaktadır. Bağlayıcı kutular, merkez koordinatları, genişlik-yükseklik bilgilerini içermektedir. Bu sayede herhangi bir nesnenin, hücrenin/hücrelerin hangi koordinatında yer aldığı ve ne kadarlık bir alan kapladığı bilinmektedir. Bağlayıcı kutular ve bağlayıcı kutuların en-boy oranında ölçeklenmesi Şekil 3.20'de gösterilmiştir.




**Şekil 3.20 :** Bağlayıcı kutular [81]

Bağlayıcı kutuların sınıflayıcı kutuları tahmin edebilmesi için kullanılan yöntem Şekil 3.21'de gösterilmiştir.



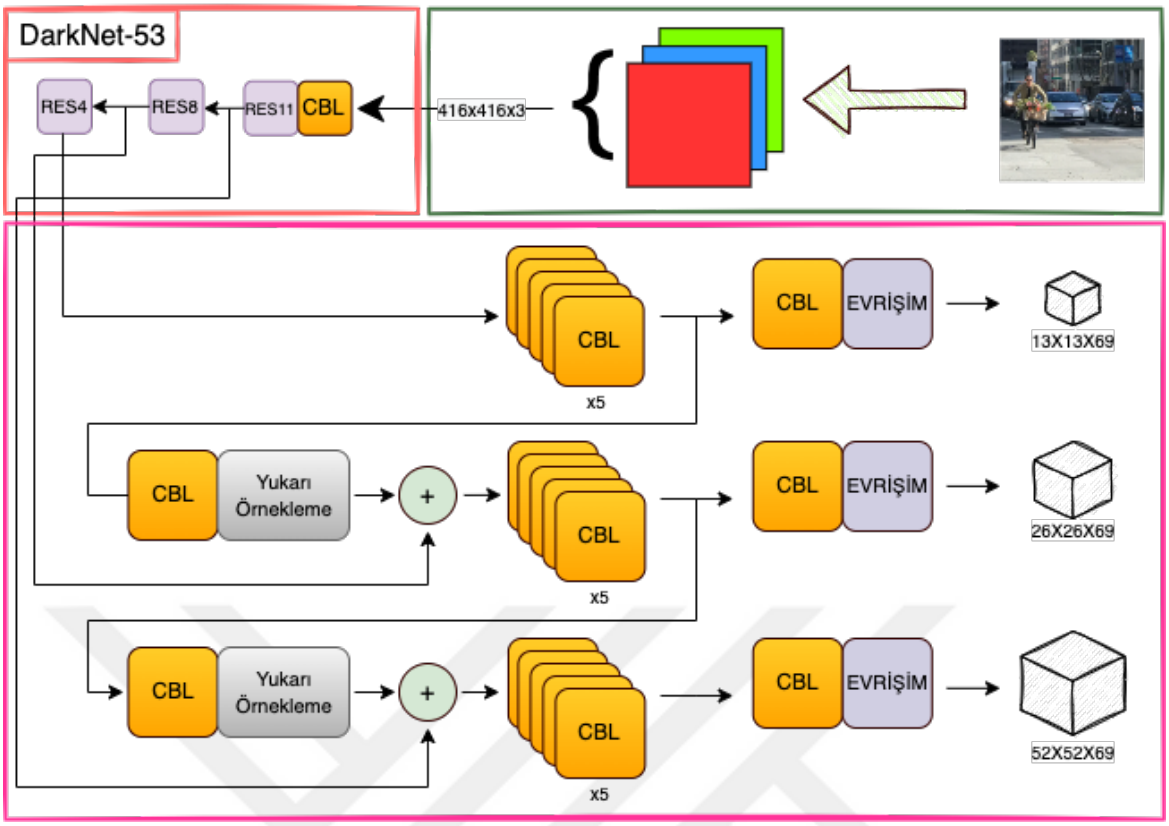

**Şekil 3.21 :** Sınırlayıcı kutu tahmini [70, 72]

### 3.3.3 Yolo v3

2018 yılında Redmon ve arkadaşları tarafından, YOLOv2'nin üzerine performans açısından geliştirilen YOLOv3 [72] önerilmiştir. YOLOv3, YOLOv2'ye kıyasla daha büyük bir mimariye sahipti.

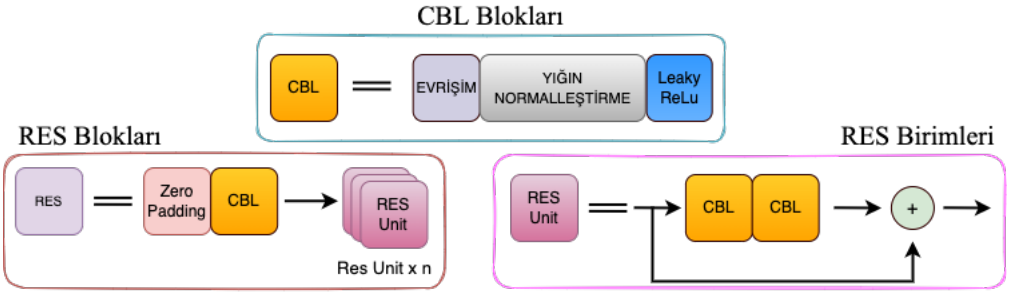
YOLOv2 tespit aracında, sınırlayıcı kutular Şekil 3.21'de gösterildiği üzere $t_x$, $t_y$, $t_w$ ve $t_h$ koordinatlarının tahmin edilmesi gerekmekteydi. YOLOv3'te ise bu yöntem değiştirilerek lojistik regresyon yöntemi kullanarak her sınırlayıcı kutular için nesnellik puan tahmini yapılmaktadır [70].

YOLOv2'de kullanılan DarkNet-19 ağı mimarisinde 19 evrişim katmanı içermekteydi. Ancak mimaride aşağı doğru inildikçe küçük ölçekli özelliklerin kaybolması kaçınılmazdı. YOLOv3'te bunun önüne geçilmesi istendi ve ResNet mimarisinde artık blokları (Residual blocks) kullanılması önerilmişti. ResNet mimarisinin özelliği olarak, sığ katmanlara inildikçe gradyan kaybını azaltmak için artık ağlar kullanmaktadır. Bu yüzden YOLOv3'te, küçük nesnelerin özellik çıkarımı kaybının ortadan kaldırılması istendiği için artık bloklarla geliştirilen DarkNet-53 mimarisini kullanılmıştır. YOLOv3'ün mimarisi Şekil 3.22'de sunulmuştur.




**Şekil 3.22 :** YOLOv3 mimarisi [72]

YOLOv3'ün mimarisi, her biri toplu normalleştirme (Batch normalization) ve Leaky ReLU aktivasyonuna sahip 53 evrişimli katmandan oluşur. Ayrıca artık bağlantılar, tüm ağ boyunca $1 * 1$ evrişimlerin girişini $3 * 3$ evrişimlerin çıkışıyla birleştirir. YOLOv3 mimarisinde kullanılan kısaltmalara ait bloklar Şekil 3.23'te sunulmuştur.

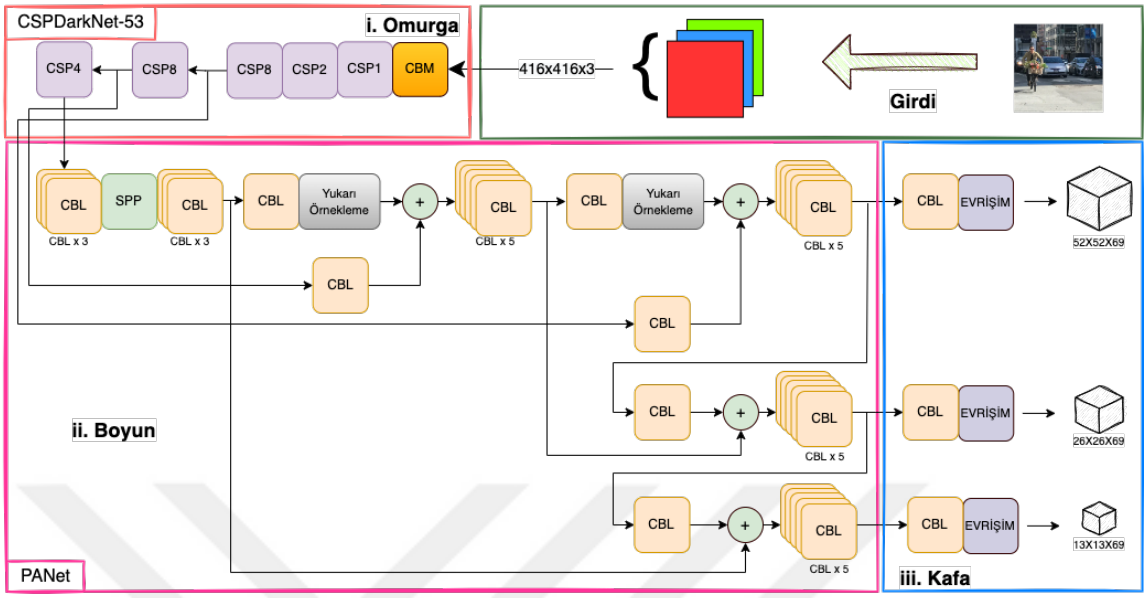



**Şekil 3.23 :** YOLOv3 mimarisine ait DBL, RES blokları ve RES birimleri

### 3.3.4 Yolo v4

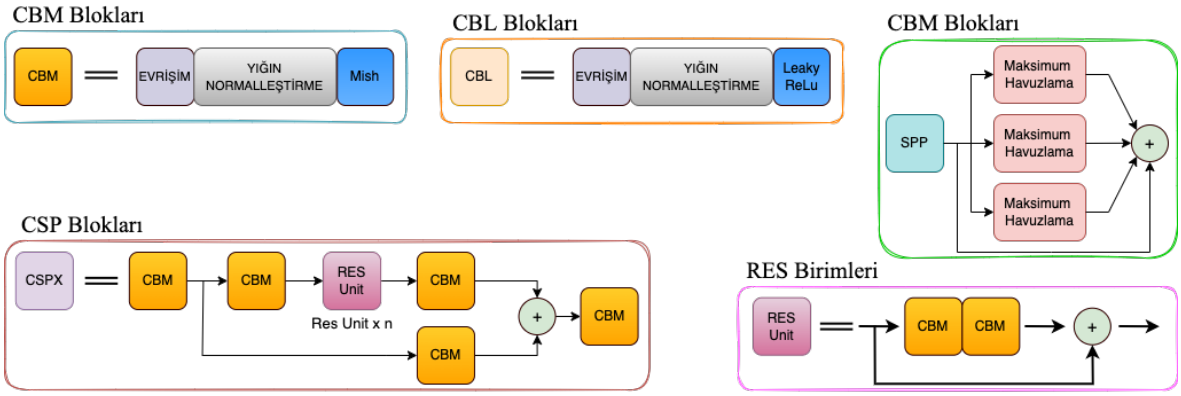
Bochkovskiy ve arkadaşları tarafından geliştirilen YOLOv4 [73] modeli o zamanların en son derin öğrenme tekniklerini kullanarak, resmi olmadan geliştirilen ilk YOLO sürümüdür. YOLOv3 ile kıyaslandığında aralarındaki en önemli fark, ağ yapısında bulunan ek yapılardır. YOLOv4 literatüre, Bag of Freebies (BoF) ve Bag of Speacials (BoS)

diye adlandırılan iki farklı terim kazandırmıştır. Bu iki terim sayesinde elde edilen sonuçlardaki doğruluk oranı artmıştır [82]. YOLOv4 mimarisi Şekil 3.24'te gösterilmiştir.




**Şekil 3.24 :** YOLOv4 mimarisi [73]

YOLOv4 ağı 3 ana bölümden oluşmaktadır: Omurga (Backbone), Boyun (Neck) ve Kafa (Head). Şekil 3.24'te sunulan mimaride bulunan blok kısaltmaları Şekil 3.25'te verilmiştir.

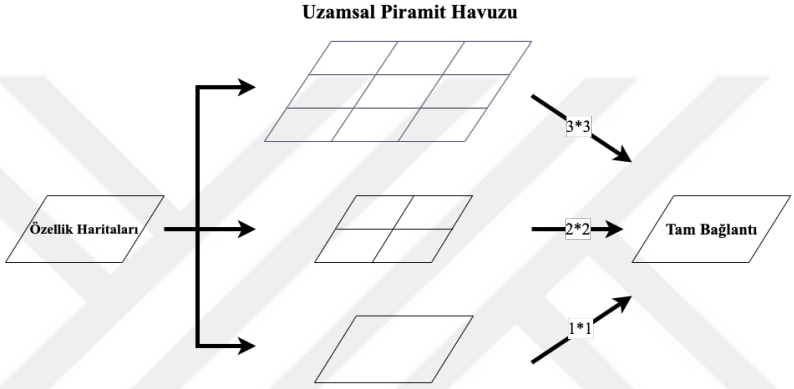



**Şekil 3.25 :** YOLOv4 mimarisine ait CBM, CBL, SPP ve RES blokları

Omurga bölümü DarkNet-53'ten esinlenilerek geliştirilen CSPDarkNet-53 ağını içermektedir. Yazarlar, omurga için birçok mimariyi denemiş olsalar dahi en iyi performansı CSPDarkNet-53 olduğu için tercih etmişlerdir. CSPDarkNet-53 ağı $3 * 3$ filtre ve 29 evrişim katmanından oluşmaktadır. CSPDarkNet-53, yoğun bağlantı modeli olarak adlandırılan bir yapıdadır. Yoğun katmanlara geçmeden, hiyerarşinin yardımıyla (önceden sonraya doğru)

girdiler birleştirilmektedir. Bu sayede daha fazla özellik geçişinin aktarılmasına imkan sağlanmaktadır. BoF ve BoS kavramları ile veri zenginleştirilmesi sağlanmaktadır [83].

Farklı ölçeklerdeki özellik haritalarını birleştirerek daha kapsamlı algılama yapabilmeyi sağlayan PANet (Path aggregarion network - Yol toplama sğı) boyun bölümünde yer almaktadır. Omurganın farklı aşamalarından toplanan özellik haritalarını bir sonraki adıma hazırlamak için karıştırır ve birleştirir. Omurgada bulunan CSPDarkNet-53 ile boyunda bulunan PANet arasında Uzamsal Piramit Havuzlama (Spatial pyramid pooling - SPP) olarak adlandırılan ek bir blok bulunmaktadır. SPP bloğunun yapısı Şekil 3.26'da sunulmuştur.




**Şekil 3.26 :** Uzamsal piramit havuzu [84]

SPP ağı, farklı boyutlardaki özellik haritaları oluşturmak ve bu haritalardan sabit boyutlu özellik vektörlerini elde etmek amacıyla kullanılmaktadır. Ayrıca bu blok sayesinde görüntü içerisinde bulunan en önemli özelliklerin saklanması sağlanmaktadır. Bu sayede görüntü içerisinde bulunan küçük boyutlu nesnelerin tespit edilmesinde, önceki modellere göre daha başarılı sonuçlar elde edilmiştir.

### 3.3.5 Yolo v5

YOLOv4'ten kısa bir süre sonra Jocher ve arkadaşları tarafından tanıtılan YOLOv5 [74] modeli PyTorch [85] tarafından geliştirilen ilk YOLO sürümüdür. YOLOv5 beş farklı model boyutu ile ortaya çıkmıştır: YOLOv5n (en küçük model boyutu, YOLOv5s (küçük model boyutu), YOLOv5m (orta model boyutu), YOLOv5l (büyük model boyutu) ve YOLOv5x (en büyük model boyutu). YOLOv5 modellerinin karşılaştırılması Çizelge 3.1'de sunulmuştur.

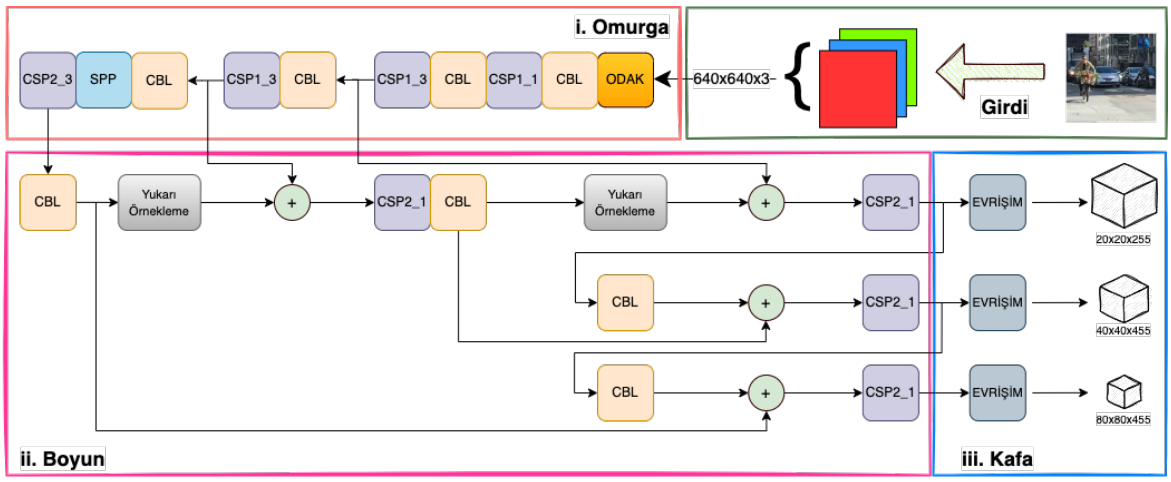
**Çizelge 3.1** : YOLOv5 modellerinin karşılaştırılması [86].

| Model | Boyut | Doğruluk (mAP 0.5) | Parametre (Milyon) |
|---|---|---|---|
| YOLOv5n | 640 | 45.7 | 1.9 |
| YOLOv5s | 640 | 56.8 | 7.2 |
| YOLOv5m | 640 | 64.1 | 21.2 |
| YOLOv5l | 640 | 67.3 | 46.5 |
| YOLOv5x | 640 | 68.9 | 86.7 |

YOLOv5, PyTorch üzerinde geliştirilmiştir ve bu sebeple önceki YOLO sürümlerinden ayrılmaktadır. Bu sayede model boyutu kendinden önce tanıtılan YOLOv4'ten daha küçük boyutlu olmasına karşın hız ve doğruluk performansları açısından daha etkili olduğu görülmektedir. PyTorch sayesinde YOLOv5 gömülü sistemler üzerinde gerçekleştirilmesi sağlanabilmektedir.

YOLOv5'in en önemli gelişmelerinden biri, tek bir katmanla temsil edilen Odak (Focus) katmanının ağa eklenmesidir. Odak katmanı, ağın başlangıcında bulunan ve özellik haritasındaki pikselleri birleştirerek hesaplama maliyetini azaltmaya yarayan bir yapıdır. Odak katmanı sayesinde model, daha küçük nesneleri tespit edebilmektedir.

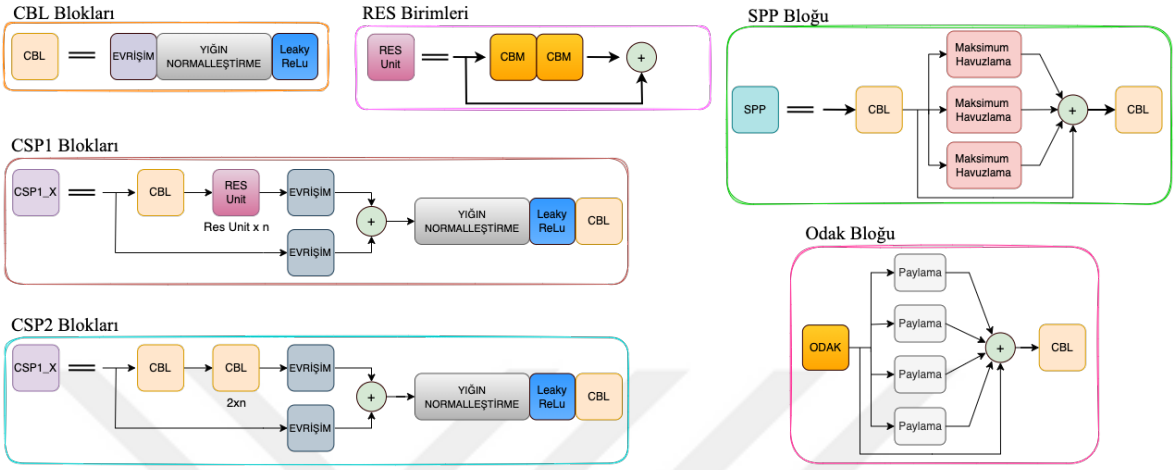
YOLOv4 gibi üç ana bölümden (omurga, boyun ve kafa) oluşan YOLOv5'in mimarisi Şekil 3.27'de gösterilmiştir.




**Şekil 3.27 :** YOLOv5 mimarisi [74]

YOLOv5, temelde YOLOv4'e benzer bir yapıya sahiptir. Ancak yazıldığı çerçeve ve geliştirilen ağ yapıları ile farklılık göstermektedir. YOLOv5'in omurgasında, odak katmanı ve geliştirilmiş CSPDarkNet-53 yapısı bulunmaktadır. Boyun kısmında ise, farklı

ölçeklerdeki özellikleri sabit boyutlu bir özellik haritasında toplayarak ağın hızını artırmayı amaçlayan geliştirilmiş SPP katmanı ve geliştirilmiş CSP-PAN ağı yer almaktadır [70]. Kafa kısmı ise YOLOv3'e benzemektedir. YOLOv5 mimarisinde bulunan blok kısaltmaları Şekil 3.28'te verilmiştir.




**Şekil 3.28 :** YOLOv5 mimarisine ait bloklar

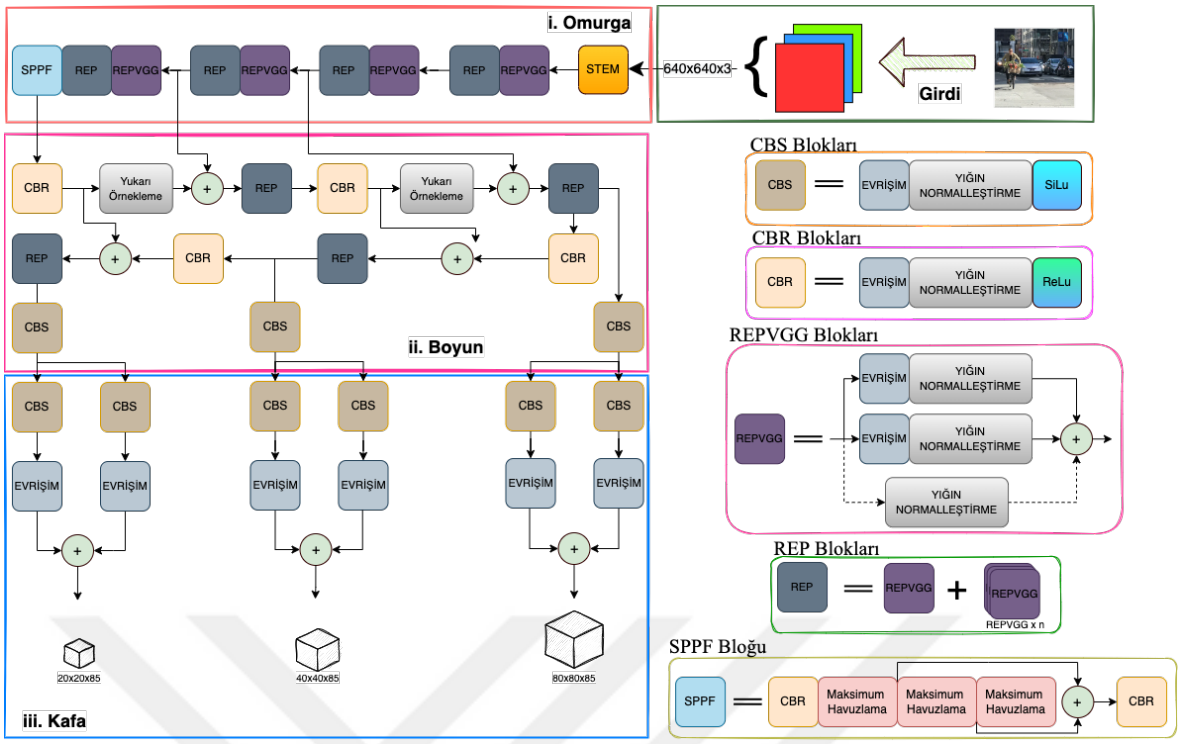
### 3.3.6 Yolo v6

Chuyi ve arkadaşları tarafından Meituan şirketinin Yapay Zeka Bölümü çatısı altında geliştirilen YOLOv6 [75] 2022 tarihinde tanıtılmıştır. YOLOv6'nın çıkış amacı, YOLO'nun endüstriyel alanda kullanımının artırılmasıdır. Bu sebeple tıpkı YOLOv5'te olduğu gibi farklı model boyutlarında YOLOv6 modelleri oluşturulmuştur. YOLOv6 modelleri Çizelge 3.2'de gösterilmiştir.

**Çizelge 3.2** : YOLOv6 modellerinin karşılaştırılması [73].

| Model | Boyut | Doğruluk (mAP 0.5) | Parametre (Milyon) |
|---|---|---|---|
| YOLOv6n | 640 | 37.5 | 4.7 |
| YOLOv6s | 640 | 45.0 | 18.5 |
| YOLOv6m | 640 | 50.0 | 34.9 |
| YOLOv6l | 640 | 52.8 | 59.6 |

YOLOv6'da kendinden önceki sürümler gibi 3 ana bölümden oluşmaktadır: Omurga, boyun ve kafa. YOLOv6 mimarisi ve mimarisinde yer alan kısaltılmış blokları Şekil 3.29'da gösterilmiştir.




**Şekil 3.29 :** YOLOv6 mimarisi ve mimariye ait bloklar [75]

Omurga kısmında, önceki YOLO sürümlerinde olduğu gibi özellik çıkarımı yapılmaktadır. YOLOv6 modeli yeniden parametrelenmiş ağları kullanmaktadır. Lakin model boyutuna göre kullanılan ağlar değişiklik göstermektedir. YOLOv6'nın Nano, Tiny ve Small sürümlerinde yeniden parametrelenmiş VGG ağları kullanılmaktadır. Ancak eğitim ve çıkarım sırasında ağ yapıları değişmektedir [87].

Eğitim için RepVGG blokları kullanılırken, özellik çıkarımı için RepConv blokları kullanılmaktadır. Orta ve Büyük modeller için ise, CSPStackRep adı verilen CSP'nin yeniden parametrelenmiş sürümlerini kullanmaktadır. YOLOv6'nın omurgasını oluşturan ağın tümüne de EfficientRep denilmektedir [88]. Boyun kısmı kendinden önceki YOLOv4 ve YOLOv5 modeline benzemektedir. YOLOv6'da bulunan PANet, EfficientRep bloğundaki gibi yeniden parametrelenmiş yapıdadır. Bu sebeple diğerlerinden ayrılır ve RepPAN olarak olarak adlandırılır. Kafa kısmında, etkili ayrılmış kafa (Efficient decoupled head - EDH) adı verilen bir yapı bulunmaktadır ve kendinden önceki sürümlerden ayrılmaktadır.

### 3.4 Nesne Tespit Problemlerinde Kullanılan Ölçüt Birimleri

Nesne tespiti problemlerinde kullanılan makine öğrenmesi modellerinin performansını değerlendirmek için istatistiksel ölçüt birimlerine ihtiyaç duyulmaktadır. Bu ölçüt birimlerinden bazıları: doğruluk (accuracy), hassasiyet (precision), duyarlılık (recall), F1-skoru, birleşim üzerindeki kesişim (intersection over union), hassasiyet-duyarlılık eğrisi (precision-recall curve), ortalama hassasiyet (average precision - AP). Çizelge 3.3'te sunulan Karmaşıklık Matrisi (Confussion Matrix) ölçüm metriklerinde kullanılmaktadır.

**Çizelge 3.3 :** Karmaşıklık Matrisi [89].

| | | **Algoritma Tahmini** | |
|---|---|---|---|
| | | **Doğru (True)** | **Yanlış (False)** |
| **Gerçek Değer** | **Doğru (True)** | Doğru Pozitif (TP) | Doğru Negatif (TN) |
| | **Yanlış (False)** | Yanlış Pozitif (FP) | Yanlış Negatif (FN) |

Çizelge 3.3'te sunulan karmaşıklık matrisine göre;

**TP:** Modelin nesne olarak bulduğu ve nesnenin var olduğu durum sayısı,

**FN:** Modelin nesne olarak bulamadığı fakat nesnenin var olduğu durum sayısı,

**FP:** Modelin nesne olarak bulduğu fakat nesnenin var olmadığı durum sayısı,

**TN:** Modelin nesne olarak bulamadığı ve nesnenin de var olmadığı durum sayısı olarak hesaplanır.

#### 3.4.1 Doğruluk

Doğru tahmin edilen nesne sayısının, toplam nesne sayısına oranlanması ile ifade edilmektedir. Eşitliği Denklem 3.16'da gösterilmiştir.

$$doğruluk = \frac{TP + TN}{TP + FP + TN + FN} \tag{3.16}$$

#### 3.4.2 Hassasiyet

Doğru pozitif tahminlerin, toplam pozitif tahminlere oranlanması ile ifade edilmektedir. Eşitliği Denklem 3.17'de gösterilmiştir.

$$hassasiyet = \frac{TP}{TP + FP} \tag{3.17}$$

### 3.4.3 Duyarlılık

Doğru pozitif tahminlerin doğru pozitif tahminler ile yanlış negatif tahminlerin toplamıyla oranlanması ile ifade edilmektedir. Eşitliği Denklem 3.18'de gösterilmiştir.

$$duyarlılık = \frac{TP}{TP + FN} \tag{3.18}$$

### 3.4.4 F1-skoru

Hassasiyet ile duyarlılığın harmonik ortalaması alınarak hesaplanmaktadır. Eşitliği Denklem 3.19'da gösterilmiştir.

$$F1 = 2 * \frac{hassasiyet * duyarlılık}{hassasiyet + duyarlılık} \tag{3.19}$$

### 3.4.5 Birleşim üzerinde kesişim

Gerçek ile tahmin edilen bölgeler arasındaki kesişimin, gerçek ve tahmin edilen bölgelerin toplamına oranlanması oranlanması ile ifade edilmektedir. Hesaplanması şekil 3.18'de gösterilmiş olup eşitliği Denklem 3.20'de sunulmuştur.

$$IoU = \frac{gerçek\ bölge\ \cap tahmin\ bölge}{gerçek\ bölge\ \cup tahmin\ bölge} \tag{3.20}$$

### 3.4.6 Hassasiyet-duyarlılık eğrisi

Değişen hassasiyet ve duyarlılık eşikleriyle hesaplanan, hassasiyet ve duyarlılık değerlerinin grafiğidir.

### 3.4.7 Ortalama hassasiyet

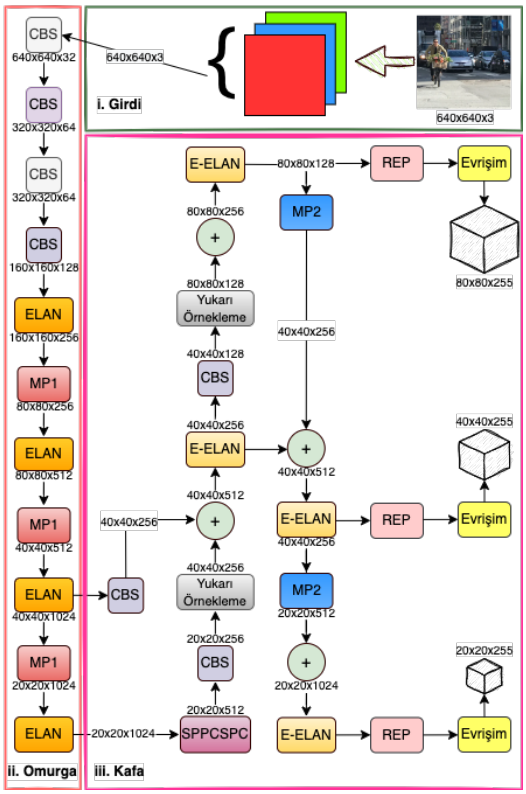
AP, nesne tespiti problemlerinde kullanılan önemli bir metriktir. Hassasiyet-duyarlılık eğrisi altında kalan alanın hesaplanması ile elde edilmektedir.

# 4. MATERYAL VE METOT

## 4.1 YOLO v7

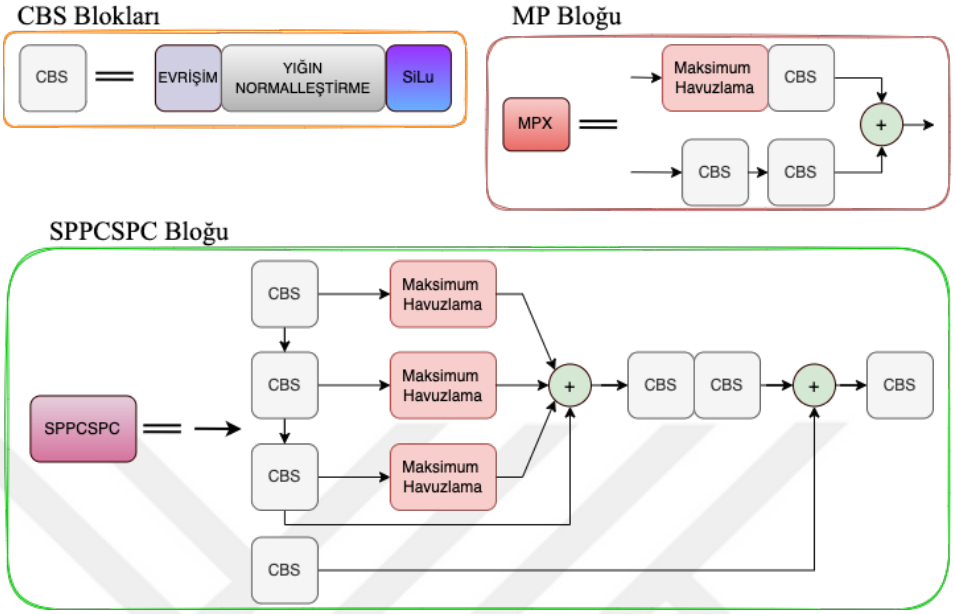
Wang ve arkadaşları tarafından 2022 tarihinde YOLO'nun 7. Sürümü olarak önerilen YOLOv7 [12] ile nesne tanıma ve nesne tespiti problemlerinde doğruluk, hız ve bellek verimliliği açısından YOLO'nun kendinden önceki sürümlerine göre daha iyi sonuç verdiği görülmüştür [90]. 5-160 FPS aralığında %51.2'lik mAP skoru almıştır [3].

YOLOv7 kendinden önceki YOLO sürümleri gibi 3 ana bölümden oluşmaktadır [3]. Bunlar: Girdi, Omurga ve Kafa. Girdi bölümü, modele verilen görüntü veya videolardan oluşmaktadır [3]. Omurga bölümü, önceki YOLO sürümlerinde olduğu gibi YOLOv7 modelinde de özellik çıkarımında kullanılan önceden eğitilmiş ağ kullanan bir yapıdır. Omurga bölümünde, evrişim katmanları, genişletilmiş verimli katman toplama ağı (Extended efficient layer aggregation networks – E-ELAN) ve maksimum havuzlama (Maximum Pooling - MP) katmanlarından oluşmaktadır [1]. Kafa bölümü, tespit işlemlerini gerçekleştiren ve çeşitli çıktıları üreten kısmı temsil eder. Bu çıktılar, genellikle nesne etiketleri, sınırlayıcı kutu çizimleri ve nesne tespit skorları gibi bilgiler içermektedir [3]. YOLOv7 mimarisi Şekil 4.1'de gösterilmiştir.




**Şekil 4.1 :** YOLOv7 mimarisi

YOLOv7 algoritmasın, genellikle girdi görüntüsünü $640 * 640$ biçiminde yeniden boyutlandırır, ardından bu görüntü omurga bölümünde bulunan ağa aktarır [3]. YOLOv7 mimarisinde yer alan bloklar Şekil 4.2'de sunulmuştur.



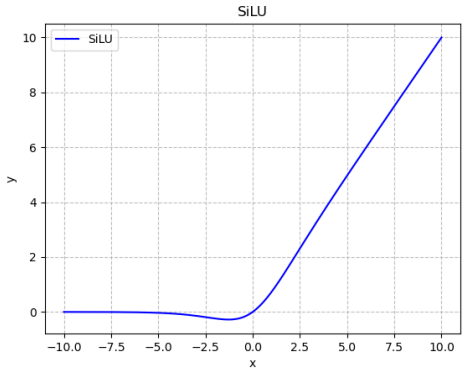

**Şekil 4.2 :** YOLOv7 mimarisine ait CBS, MPx ve SPPCSPC blokları

CBS bloğu, farklı katmanlardan elde edilen özellik haritalarını birleştiren ve bu özellik haritalarını detaylandırarak yeniden oluşturan bir ağ yapısıdır [3].

Omurga bölümü genelde verimli katman toplama ağı (Efficient layer aggregation networks - ELAN), MP yapılarını ve SiLU aktivasyon fonksiyonunu kullanır. SiLU aktivasyon fonksiyonu Denklem 4.1 ile gösterilmiştir.

$$f(x) = \frac{x}{1+e^{-x}} \tag{4.1}$$

SiLU aktivasyon fonksiyonunun grafiği Şekil 4.3'te gösterilmiştir.




**Şekil 4.3 :** SiLU aktivasyon fonksiyonu

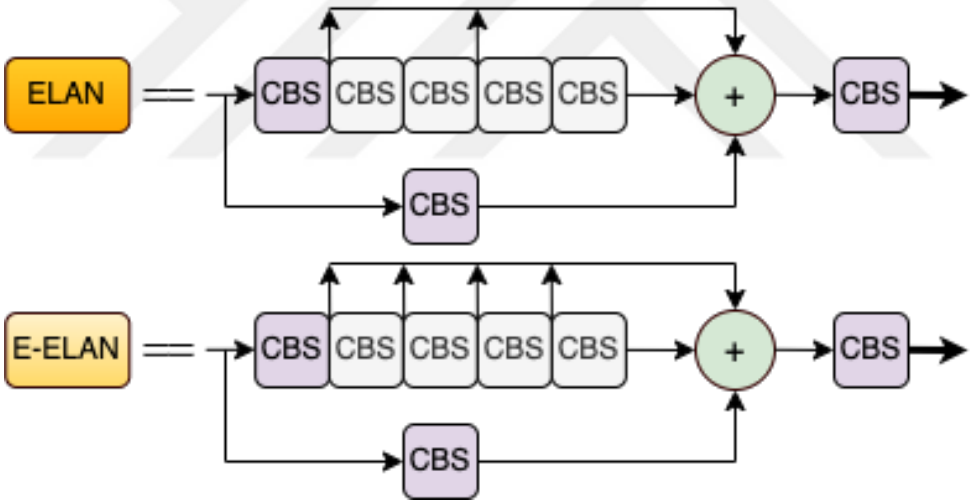
SPPCSPC ağı, SPP bloklu CSPNet'tir (Cross stage partial network - Aşamalar arası kısmi ağ). CSPNet ağ modeli, tekrarlanan gradyan sorununu minimize etmek için tasarlanan bir ağ modelidir [3, 91]

### 4.1.1 Genişletilmiş verimli katman toplama ağı

Modern nesne tespit araçları geliştirilirken, asıl önemli olan maliyeti azaltıp, doğruluğu artırmak için algoritmaların optimize edilmesidir [3, 12].

YOLOv7, mimarisinde yer alan E-ELAN yapısı ile derin ağların etkili bir şekilde kullanımını sağlamakta olup kendinden önceki YOLO sürümleri ile kıyaslandığında yenilikle güçlendirilmiş bir nesne tanıma ve nesne tespit algoritmasıdır [3].

YOLOv7 modeli, YOLOv4 temel alınarak geliştirilmiş bir modeldir ve bu geliştirme sürecinde özellikle ELAN yapısı üzerinde çalışılmıştır. Bu yapı daha fazla genişletilmiş, böylece daha fazla parametre kullanarak hızlı ve kesin sonuçlar elde edilmesi hedeflenmiştir [3, 92, 93]. ELAN ve E-ELAN mimarileri Şekil 4.4'te sunulmuştur.




**Şekil 4.4 :** YOLOv7 mimarisinde yer alan ELAN ve E-ELAN [3, 12, 91]

E-ELAN yapısında, her CBS ağı tamamlandıktan sonra birleştirme (concatenate) işlemi gerçekleştirilir. E-ELAN yapısı ile ağ modeli, daha çok parametre kullanımı sağlamaktadır. Bu sebeple performans açısından iyileştirme görülmüştür [3].

YOLOv7'nin kullanım amacına uygun olarak hem düşük boyutlu hem de yüksek boyutlu birkaç versiyonu bulunmaktadır. Bu modeller, MS COCO test veri seti ile eğitiminin ardından elde edilen sonuçlar karşılaştırıldığında, FPS değerleri ve test ile doğrulama veri setlerinde ortalama hassasiyet ($AP^{test}$ ve $AP^{val}$) skorları açısından performanslarına dair veriler Çizelge 4.1'de sunulmuştur.

**Çizelge 4.1** : YOLOv7 modellerinin AP skorları [3, 12, 94, 95].

| Model | Parametre Sayısı (Milyon) | FPS | $AP^{test}$ (%) | $AP^{val}$ (%) |
|---|---|---|---|---|
| **YOLOv7-Tiny** | 6.2 | 286 | 38.7 | 38.7 |
| **YOLOv7** | 36.9 | 161 | 51.4 | 51.2 |
| **YOLOv7-X** | 71.3 | 114 | 53.1 | 52.9 |
| **YOLOv7-W6** | 70.04 | 84 | 54.9 | 54.6 |
| **YOLOv7-E6** | 97.2 | 56 | 56 | 55.9 |
| **YOLOv7-D6** | 154.7 | 44 | 56.6 | 56.3 |
| **YOLOv7-E6E** | 151.7 | 36 | 56.8 | 56.8 |

### 4.1.2 Yolov7 eğitim parametreleri

Wang ve arkadaşlarının önerdikleri YOLOv7 [12] modeli, girdi görüntüsünü $640 * 640$ piksel olarak yeniden boyutlandırmaktadır [3]. Önerilen YOLOv7 modeli için belirlenen eğitim parametreleri aşağıdaki gibidir:

- Yığın boyutu (Batch Size): 16,
- Devir (Epoch): 100,
- Eğitim oranı: 0,0001,
- Aktivasyon fonksiyonu: SiLU,
- Kayıp fonksiyonu: Pürüzsüz L1 Kaybı (Smooth L1 Loss)

Wang ve arkadaşları, YOLOv7 [12] modelinin belirlenen parametrelerle MS COCO veri setinde en iyi performansı gösterdiğini ifade etmiştir [3].

## 4.2 Coco Veri Seti

2017'de Microsoft (MS) tarafından oluşturulan açık kaynaklı Common Objects in Context (Bağlamda Yaygın Nesneler - COCO) [96] veri seti, nesne tespiti ve takibi model eğitiminde yaygın olarak tercih edilmektedir. COCO veri seti, 80 farklı nesne sınıfını içeren yaklaşık 330,000 etiketli görüntü barındırmaktadır. COCO veri setindeki nesne sınıfları Şekil 4.5'te gösterilmiştir.

```
names = ['insan', 'bisiklet', 'araba', 'motosiklet', 'ucak', 'otobus', 'tren', 'kamyon', 'tekne', 'trafik lambasi',
         'yangin muslugu', 'dur isareti', 'parkmetre', 'bank', 'kus', 'kedi', 'kopek', 'at', 'koyun', 'inek',
         'fil', 'ayi', 'zebra', 'zurafa', 'sirt cantasi', 'semsiye', 'canta', 'kravat', 'bavul', 'frizbi',
         'kayak', 'kar kayagi', 'spor topu', 'ucurtma', 'beyzbol sopasi', 'beyzbol eldiveni', 'kaykay',
         'sorf tahtasi','tenis raketi', 'sise', 'sarap kadehi', 'fincan', 'catal', 'bicak', 'kasik', 'kase',
         'muz', 'elma', 'sandvic', 'portakal', 'brokoli', 'havuc', 'sosisli sandvic', 'pizza', 'çörek', 'kek',
         'sandalye', 'kanepe', 'bitki', 'yatak', 'yemek masasi', 'tuvalet', 'tv', 'laptop', 'fare', 'uzak', 'klavye',
         'cep telefonu', 'mikrodalga', 'firin', 'ekmek kizartma makinesi', 'lavabo', 'buzdolabi', 'kitap', 'saat',
         'vazo', 'makas', 'oyuncak ayi', 'sac kurutma makinesi', 'dis fircasi']
```

**Şekil 4.5 :** MS COCO veritesinde bulunan etiketler

### 4.3 Geliştirilen Uygulamalarda Kullanılan Kütüphaneler

Yapılan bu çalışma için iki farklı uygulama geliştirilmiştir. Birincisi, nesne tespit probleminin çözümü için kullanılan sanal sunucu (virtual server), ikincisi ise kullanıcının nesne tespit problemini akıllı mobil cihazında gerçekleştirmek istediği için gerekli olan mobil arayüzdür.

Sunucu tarafı için arka-uç (back-end) kodlaması Python dilinde geliştirilmiştir. Mobil arayüz için ön-uç (front-end) kodlaması React-Native (RN) dilinde geliştirilmiştir. Arka-uç ve ön-uç yazılımlarında kullanılan kütüphaneler ve sürümleri Çizelde 4.2'de sunulmuştur.

**Çizelge 4.2** : Kodlamada kullanılan kütüphaneler ve sürümleri.

| | **ARKA-UÇ (SUNUCU)** | **ÖN-UÇ (MOBİL ARAYÜZ)** |
|---|---|---|
| **Programlama Dili** | **Python** | **React-Native** |
| Kütüphaneler ve Sürümleri | numpy ~1.24.1 | expo ~47.0.14 |
| | opencv-python ~4.7.0.68 | expo-status-bar ~1.4.2 |
| | matplotlib ~3.6.2 | react 18.1.0 |
| | flask ~2.2.2 | react-native 0.70.8 |
| | pillow ~9.4.0 | expo-image-picker ~14.0.2 |

### 4.4 Google Colaboratory

Google Colaboratory (Colab) [97], Google Research tarafından sunulan yapay zeka, makine öğrenmesi, derin öğrenme vb. disiplinlerin, Python dilinde eğitim yapabilme, kod yazabilme, test edebilme gibi işlemlerin yapılabildiği bulut tabanlı bir platformdur [98, 9]. 2017 senesinde sunulan Colab, kullanıcılarına ücretsiz bir şekilde GPU ve TPU (Tensor işleme birimi - Tensor processing unit) desteği vermektedir.

Colab platformunda, herhangi bir araç yüklemeksizin, içerisinde bulunan Jupyter Notebook [100] ile Python kodları derlenebilir. Google'ın kullanıcılarına sunmuş olduğu ücretsiz sanal sürücüleri (Google Drive) ile Colab arasında veri akışı sağlanmaktadır. Kullanıcılar yapay zeka (makine öğrenmesi, derin öğrenme vb.) model eğitimlerinde kullanmak istedikleri veri setlerini ve/veya test sonuçlarında elde ettikleri sonuçları sanal sürücülerinde saklayabilmektedir. Colab'da, Python dilinde özellik yapay zeka uygulamalarında yaygın olarak kullanılan TensorFlow [101], Keras [102], Caffe [103] vb. kütüphaneler kuruludur fakat kullanıcılar ihtiyaç duydukları takdirde kullanmak istedikleri diğer kütüphaneleri de kurabilmektedir [104].

Colab'ın ücretli olarak kullanıcılarına sunduğu paketleri mevcuttur. Colab ücretsiz kullanımda, kullanıcılarına yaklaşık 12 saat çalışma süresi sağlamaktadır. Ancak ücretli paketler ile bu süre uzatılabilir. Colab, Çalışma süresi boyunca yarım kalan eğitimleri daha sonrasında devam ettirmeye izin vermektedir. Colab'ın sunduğu çalışma süresi, sağlanan GPU ve TPU gereksiz yere meşgul edilmemesi adına alınan bir şirket politikasıdır. Colab'ın paketleri Çizelge 4.3'te sunulmuştur.

**Çizelge 4.3** : Colab paketleri [99].

| | **COLAB** | **COLAB PRO** | **COLAB PRO+** |
|---|---|---|---|
| **GPU** | K80 | K80, T4, T100 | K80, T4, T100 |
| **RAM** | 16 GB | 32 GB | 52 GB |
| **Çalışma Ortamı** | 12 Saat | 24 Saat | 24 Saat |
| **Arka Planda Çalıştırma** | Uygun Değil | Uygun Değil | Uygun |
| **Ücretsiz** | Evet | Hayır | Hayır |

Yapılan bu çalışmada Colab'ın ücretsiz paketinden faydalanılmıştır.

### 4.5 React Native

React Native (RN) [105], 2015 yılında Meta (eski adıyla Facebook) tarafından geliştirilen açık kaynaklı bir yazılım yapısıdır. Bu yapı, Android ve iOS işletim sistemleri üzerinde çapraz platformda çalışabilen mobil uygulamalar geliştirmeyi mümkün kılan ve JavaScript betiğini temel alan bir teknolojidir. Bugün, özellikle Facebook, Microsoft, Uber, Skype, Shopify gibi tanınmış şirketler tarafından tercih edilmektedir.

Mobil uygulama geliştiricileri arasında popülerlik kazanmış olan React Native, açık kaynaklı olması ve geniş bir yazılımcı topluluğuna sahip olması sayesinde gelişmiş bir kütüphane ve ekosistem sunar. Bu nedenle React Native, popülerliğini sürdüren ve yaygın olarak kullanılan bir yazılım yapısıdır.

### 4.5.1 React native kütüphaneleri

React Native'in geliştiriciler tarafından tercih edilmesinin en önemli nedenlerinden biri, büyüyen açık kaynaklı kütüphaneleri ve geliştirici topluluğu sayesinde oluşturduğu ekosistemdir. Yazılımcılar, ihtiyaçlarına uygun kütüphanelere erişebilir ve bu kütüphaneleri istedikleri şekilde özelleştirebilirler.

Yapılan bu tez çalışmasında, Expo [106] kütüphanesinden yararlanılmıştır. Expo, RN için geliştirilen uygulamaları yerel ağ bağlantısı üzerinden gerçek cihazlarda görüntüleme ve kullanma imkanı sağlayan bir araçtır. Expo sayesinde, emülatör gereksinimi olmadan gerçek fiziksel cihazlar üzerinde uygulamaları test etmek mümkün hale gelmektedir. Expo aynı zamanda küçük boyutlu ve önceden geliştirilmiş bileşenlere sahip olmasıyla dikkat çekmektedir.

### 4.5.2 Eğitilen modelin tensorflowlite modeline dönüşümü

YOLOv7, YOLOv5 ve YOLOv6 gibi, PyTorch [85] kütüphanesi kullanılarak geliştirilmiştir. Ancak PyTorch [85] ile eğitilen bir modelin mobil cihazlarda kullanılabilmesi için bazı düzenlemeler yapılması gerekmektedir [3]. Bunun nedeni, YOLOv7'nin test edilmesi için kullanılan platform olan React Native (RN)'in TensorFlow (TF) [101] kütüphanesi ile uyumlu olan TensorFlow Lite (TFLite)'ı desteklemesidir. PyTorch modelin TFLite modele dönüşüm haritası Şekil 4.6'da gösterilmiştir [3].


**Şekil 4.6 :** Model dönüşüm haritası[3, 107].

Eğitilen PyTorch modelinin, Tensorflow Lite modeline dönüştürülmesi için gerekli adımlar:

i. PyTorch modelinin ONNX modeline dönüştürülmesi

ONNX (Açık Sinir Ağı Değişimi - Open Neural Network Exchange), makine öğrenimi modellerinin eğitildiği bir ortamdan farklı bir platformda modelin kullanılabilmesi

için kullanılan açık kaynaklı bir araç ve formattır. ONNX modelleri, TensorFlow [101], PyTorch [85], Caffe2, MXNet, MATLAB gibi birçok derin öğrenme çerçevesi tarafından desteklenmektedir. Bu sayede model geliştiricileri, farklı çerçevelerde eğitilmiş modelleri dönüştürerek ve uyumlu hale getirerek, bu modelleri farklı platformlarda veya uygulamalarda kullanabilirler. ONNX, model paylaşımını ve taşınabilirliğini artıran bir standart olarak büyük önem taşır [3].

PyTorch modeli ONNX modele dönüştürüldüğünde, modelin taşınabilirliği artar ve farklı ortamlarda daha hızlı ve daha ekonomik bir şekilde kullanılabilir [3].

ii. ONNX modelinin, TensorFlow modele dönüştürülmesi

TFLite modeline dönüştürme işlemi için öncelikle modelin TensorFlow (TF) modele dönüştürülmesi gerekmektedir [3].

iii. TensorFlow modelinin, TensorFlow Lite modele dönüştürülmesi

TensorFlow Lite, mobil ve nesnelerin interneti (Internet of Things - IoT) cihazlarında düşük kaynak tüketimi gerektiren derin öğrenme modellerini çalıştırmak için tercih edilen bir derin öğrenme kütüphanesidir. TensorFlow Lite, bu tür cihazlar için ekstra optimize edilmiş modelleri destekler, böylece modeller daha hızlı sonuçlar üretebilir ve daha az güç tüketebilir. Bu sayede eğitilen modeller, iOS ve Android gibi mobil cihazlarda sorunsuz bir şekilde kullanılabilir hale gelir. TensorFlow Lite, bu tür cihazlarda derin öğrenme tabanlı uygulamaların daha verimli ve etkili bir şekilde çalışmasına olanak tanır [3].

### 4.5.3 Tflite modelin mobil cihaza entegrasyonu

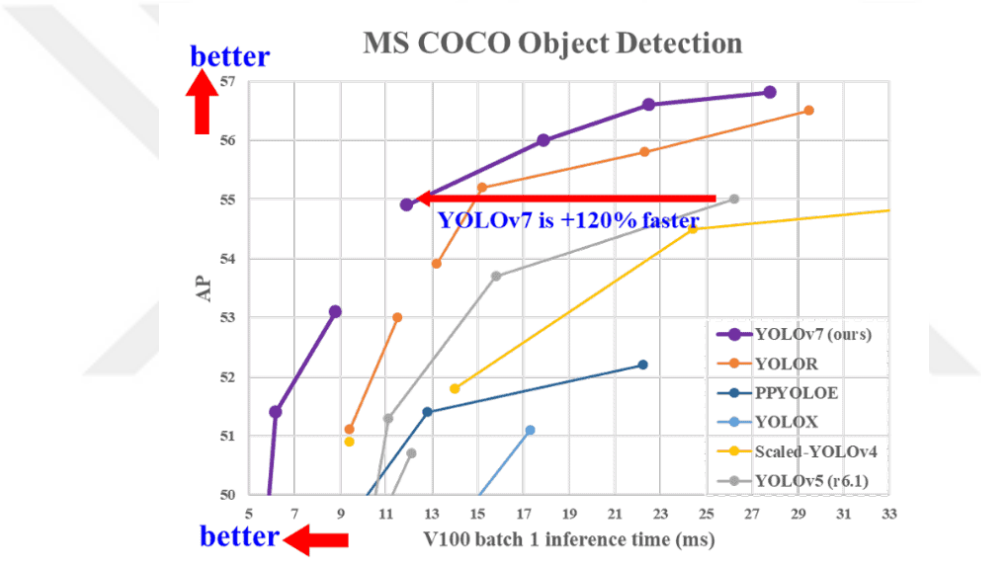
React Native'de modelin çalıştırılabilmesi için bir arka uç (backend) yapıya ihtiyaç duyulduğu bilinmektedir. Bu nedenle Python'da Flask kütüphanesi kullanılarak bir sunucu oluşturulmuştur. Oluşturulan bu sunucu üzerinde temelde iki işlem gerçekleştirilmektedir. İlk işlem, girdi görüntülerini sunucuya aktarmak için kullanılan POST işlemidir. İkinci işlem ise sunucuya aktarılan görüntüler üzerinde nesne tanıma ve/veya nesne tespiti yapılıp sonuçları cihaza geri göndermek ve görüntülemek için kullanılan GET işlemidir. Bu şekilde, mobil uygulama ile sunucu arasında görüntü verileri iletilip, işlenip sonuçlar geri döndürülerek modelin etkili bir şekilde kullanılması sağlanmaktadır.

# 5. SONUÇ VE DEĞERLENDİRME

## 5.1 Çalışmanın Tanıtımı

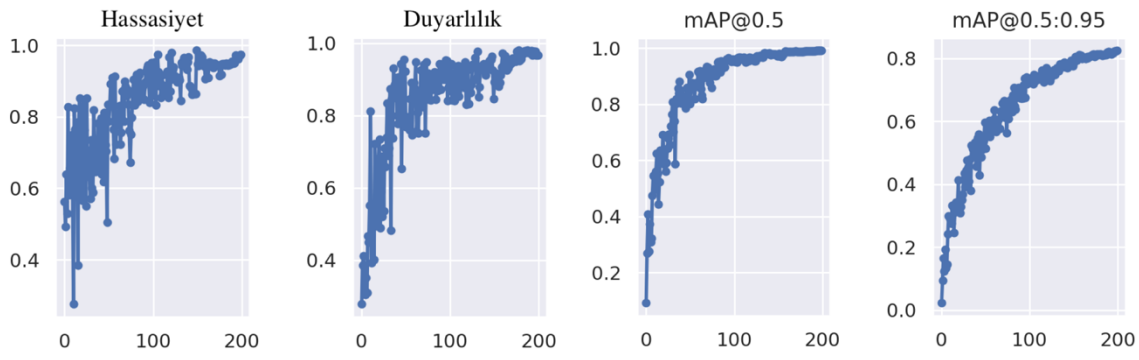
YOLOv7, 5-160 FPS aralığında diğer tüm nesne tespit algoritmalarının üzerinde bir doğruluk vermektedir [12]. Diğer bazı nesne tespit algoritmaları ile YOLOv7'nin MS COCO veri setindeki başarımı Şekil 5.1'de sunulmuştur.

YOLOv7, diğer tüm nesne tespit algoritmalarının üzerinde bir doğruluk sunarak 5 ila 160 FPS aralığında çalışabilme yeteneğine sahiptir [12] . YOLOv7'nin MS COCO veri setindeki başarımının, diğer bazı nesne tespit algoritmalarının başarımları ile karşılaştırılması Şekil 5.1'de gösterilmiştir.



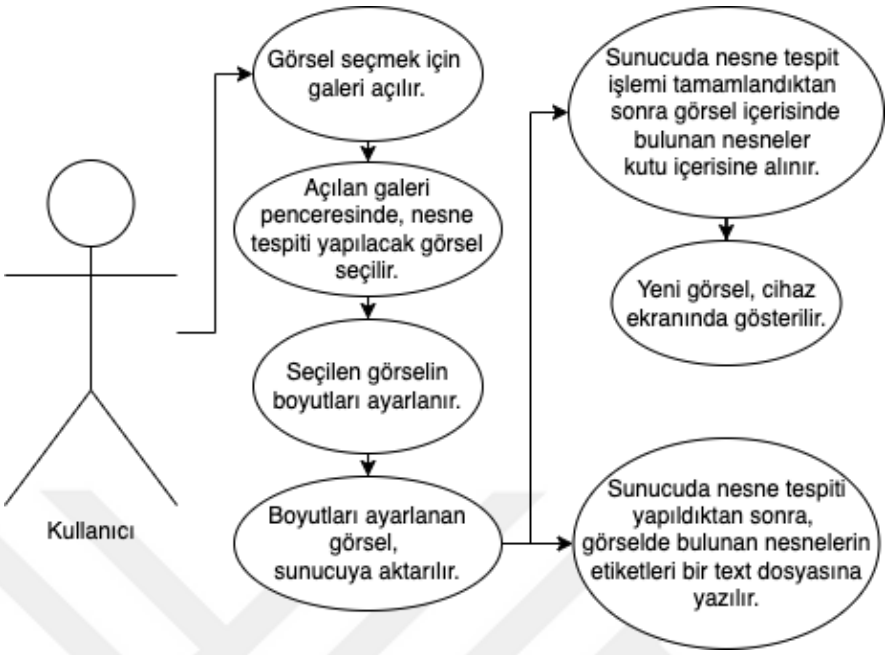

**Şekil 5.1 :** MS COCO veri seti ile eğitilen diğer nesne tespit algoritmaları [12]

Bu tez çalışması için eğitilen YOLOv7 modelinin, eğitim sonucunda elde ettiği performansı değerlendiren başarım metrikleri, Şekil 5.2'de sunulmuştur.




**Şekil 5.2 :** Eğitilen YOLOv7 modelinin başarım metrikleri

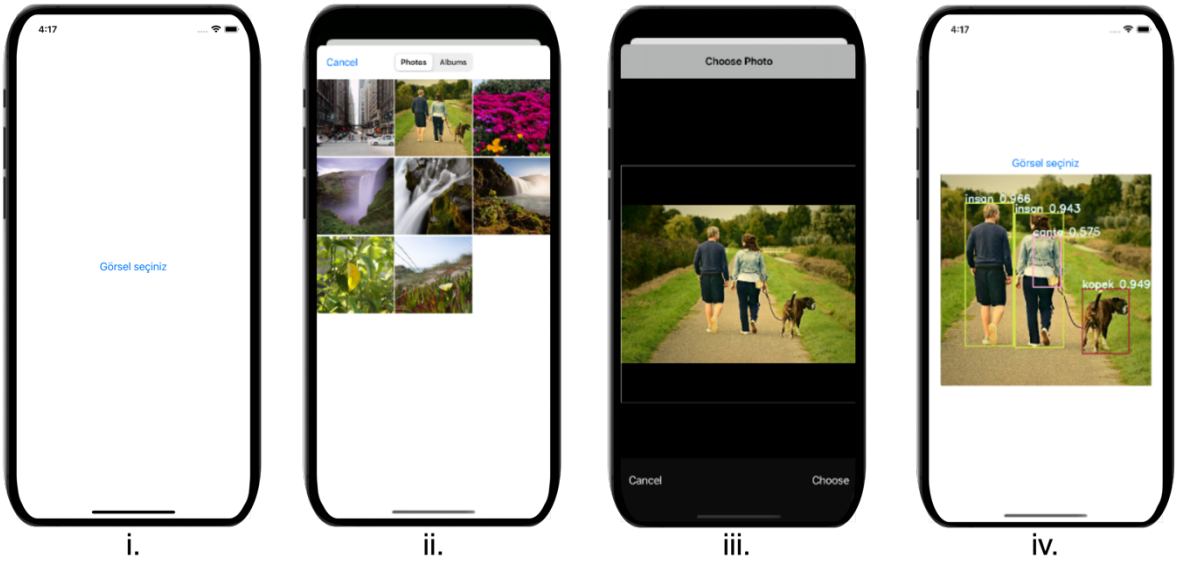
Yapılan bu tez çalışması için geliştirilen mobil arayüzün, Kullanıcı Senaryosu Diyagramı (Use Case Diagram) Şekil 5.3'te sunulmuştur.




**Şekil 5.3 :** Geliştirilen uygulamanın kullanıcı senaryosu diyagramı [3]

## 5.2 Çalışma için Tasarlanan Arayüz

MacOS işletim sistemi üzerinden Visual Studio Code editörü üzerinden geliştirmeler tasarlanmıştır. RN dili için geliştirilen Expo kütüphanesinden faydalanılmıştır. Uygulamanın testleri MacOS işletim sistemi üzerinde oluşturulan iOS işletim sistemine sahip iPhone 14 emülatöründen sağlanmıştır. Geliştirilen arayüze ait ekran görüntüleri Şekil 5.4'te sunulmuştur.




**Şekil 5.4 :** Çalışmada geliştirilen uygulamanın ekran görüntüleri [3]

Şekil 5.3’te yer alan i., ii., iii. ve iv. adımlar:

i. Uygulamanın anasayfası, kullanıcıdan bir görsel seçmesi beklenmektedir.

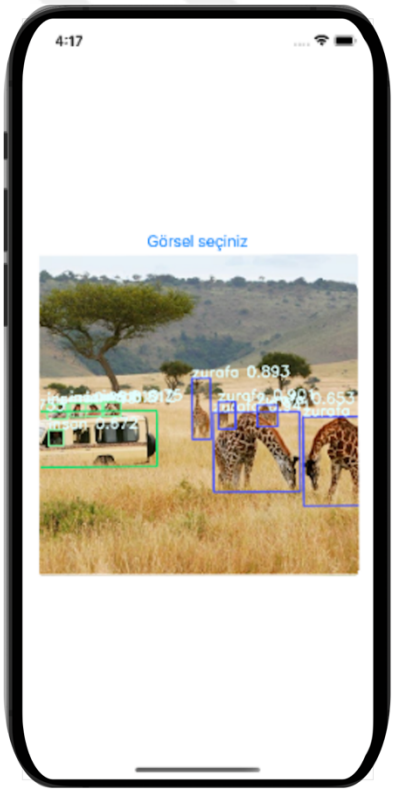
ii. Kullanıcı, bir görsel seçmek için galeri sayfasına yönlendirilmektedir ve nesne tespiti yapmak istediği görseli seçmesi beklenmektedir.

iii. Görsel, nesne tespiti yapılacak bölgeye göre kırpılır. Eğer görselin tamamında tespit işlemi yapılacaksa kırpma işlemine gerek yoktur.

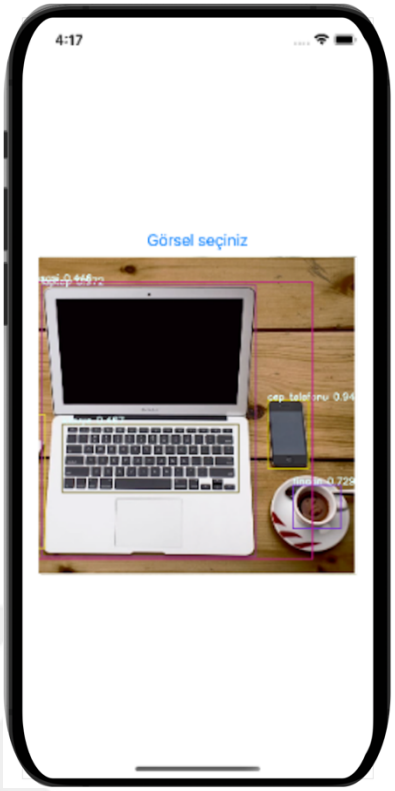
iv. Nesne tespiti işlemi tamamlandığında, görselde bulunan nesneler kutu içine alınır ve doğruluk değerleri ile kullanıcıya yansıtılır.

Yapılan bu tez çalışması için geliştirilen uygulama ile Şekil 5.4’te bulunan i., ii., iii. ve iv. adımları takip ederek elde edilen deneysel sonuçları içeren birkaç örnek sonuç Çizelge 5.1’de sunulmuştur.

**Çizelge 5.1** : Mobil uygulama ile görüntülerden nesne tanıma örnekleri [3].

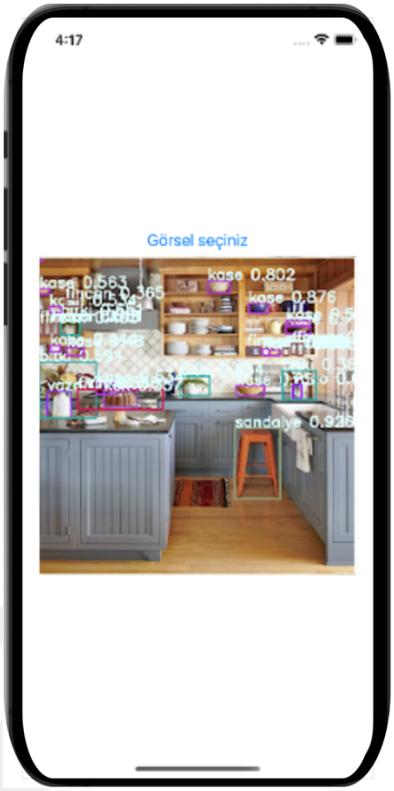
| No. | Mobil Uygulamadan Sunucuya Gönderilen Görüntü | Sunucudan Mobil Uygulamaya Gönderilen Görüntü |
|---|---|---|
| 1. |  |  |

2.

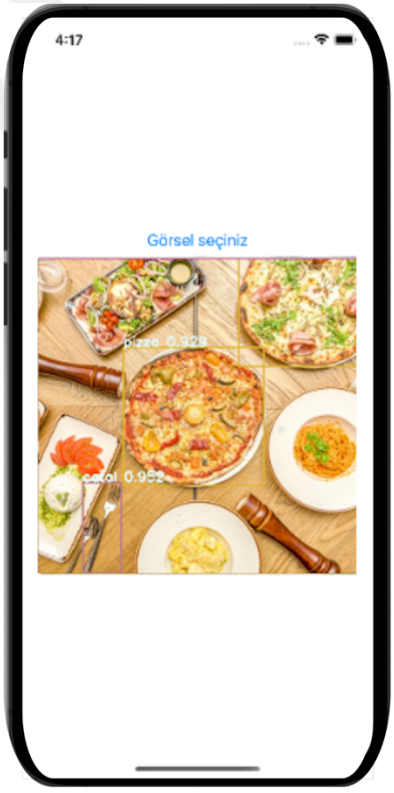



3.

4.


4:17
Görsel seçiniz
kase 0.802
kase 0.876
sandalye 0.92

5.


4:17
Görsel seçiniz
pizza 0.929

## 5.3 Çalışmaya Ait Değerlendirme

Ön-eğitimli YOLOv7 modeli, geliştirilen ön ve arka uçlar sayesinde, akıllı mobil cihazın GPU'suna ihtiyaç duymadan nesne tanınması yapabilir. Görüntü uygulamaya yüklendikten sonra, kısa bir süre içinde nesne tanınması gerçekleştirilebilir. Yapılan bu tez çalışması ile YOLOv7'nin ticari kullanımına olumlu katkı sağlayabileceği düşünülmektedir.

YOLOv7, doğruluk ve hız açısından diğer rakiplerine göre daha yüksek skorlar elde edebilen bir nesne tespit aracıdır [3]. Ancak YOLOv7, tıpkı kendinden önceki sürümlerde olduğu gibi,

- Küçük boyutlu görseller içerisinde nesne tanınması ve tespiti,
- Görsel içerisinde bulunan küçük boyutlu nesneler,
- Aydınlık-Karanlık gibi çevre koşulları

vb. durumlarda nesne tanıma ve nesne tespiti problemlerinde zayıf kalabilmektedir. Buna rağmen, literatüre bakıldığında 160 FPS'e kadar hızlı ve doğru sonuç verebilmesi sebebiyle diğer nesne tespit yöntemlerine göre video içerisinde nesne tanıma ve nesne tespitinde öne çıkmaktadır.

YOLO serisi geçtiğimiz on yıl içerisinde gelişmeye devam etmektedir [3]. İçinde bulunduğumuz 2023 senesinde Reis ve arkadaşları tarafından önerilen YOLOv8 [76] ile her zamanki gibi, bir önceki sürümün hız ve doğruluk açısından geliştirmeler yapılmıştır. Bu nedenle YOLO'nun gelecekteki gelişmelerini takip edeceğiz ve bu çalışmayı ilerletmeye devam edeceğiz.

## 5.4 Çalışmaya Ait Kaynak Kodları

Çalışmada geliştirilen her kodlama GitHub [108] platformunda, Batukar [109] profilinde "Tez" adıyla gizli depo olarak saklanmaktadır. Talep halinde paylaşım sağlanabilir.

## KAYNAKLAR


[1] **Resul, D. A. Ş., Polat, B., & Tuna, G.** (2019). Derin öğrenme ile resim ve videolarda nesnelerin tanınması ve takibi. Fırat Üniversitesi Mühendislik Bilimleri Dergisi, 31(2), 571-581.

[2] **Wang, N., Wang, Y., & Er, M. J.** (2022). Review on deep learning techniques for marine object recognition: Architectures and algorithms. Control Engineering Practice, 118, 104458.

[3] **Karadağ, B. & Arı, A.** (2023). Akıllı Mobil Cihazlarda YOLOv7 Modeli ile Nesne Tespiti . Politeknik Dergisi , , 1-1 . DOI: 10.2339/politeknik.1296541.

[4] **Cai, Y., Li, H., Yuan, G., Niu, W., Li, Y., Tang, X., ... & Wang, Y**. (2021, May). Yolobile: Real-time object detection on mobile devices via compression-compilation co-design. In Proceedings of the AAAI conference on artificial intelligence (Vol. 35, No. 2, pp. 955-963).

[5] **Ari, A.** (2023). Multipath feature fusion for hyperspectral image classification based on hybrid 3D/2D CNN and squeeze-excitation network. Earth Science Informatics, 16(1), 175-191.

[6] **Tao, J., Wang, H., Zhang, X., Li, X., & Yang, H.** (2017, October). An object detection system based on YOLO in traffic scene. In 2017 6th International Conference on Computer Science and Network Technology (ICCSNT) (pp. 315-319). IEEE.

[7] **Dersuneli, M., Gündüz, T., & Kutlu, Y.** (2021). Bul-Tak Oyuncağı Şekillerinin Klasik Görüntü İşleme ve Derin Öğrenme Yöntemleri ile Tespiti. Bitlis Eren Üniversitesi Fen Bilimleri Dergisi, 10(4), 1290-1303.

[8] **Liu, C., Tao, Y., Liang, J., Li, K., & Chen, Y.** (2018, December). Object detection based on YOLO network. In 2018 IEEE 4th information technology and mechatronics engineering conference (ITOEC) (pp. 799-803). IEEE.

[9] **Ömer, E. R., & BİLGE, H. Ş.** (2021). Bir Küçük Nesne Tespit Zorluğu Olarak Hava Görüntülerinden Araç Tespiti. Veri Bilimi, 4(1), 73-83.

[10] **Sultana, F., Sufian, A., & Dutta, P.** (2020). A review of object detection models based on convolutional neural network. Intelligent computing: image processing based applications, 1-16.

[11] **Redmon, J., Divvala, S., Girshick, R., & Farhadi, A.** (2016). You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 779-788).

[12] **Wang, C. Y., Bochkovskiy, A., & Liao, H. Y. M.** (2023). YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 7464-7475).

[13] **Coşkun, F., & GÜLLEROĞLU, H. D.** (2021). Yapay zekânın tarih içindeki gelişimi ve eğitimde kullanılması. Ankara University Journal of Faculty of Educational Sciences (JFES), 54(3), 947-966.

[14] **Turing, A. M.** (1950). Mind. Mind, 59(236), 433-460.

[15] **McCarthy, J.** (1959, September). Lisp: a programming system for symbolic manipulations. In Preprints of papers presented at the 14th national meeting of the Association for Computing Machinery (pp. 1-4).

[16] **Şahin, E., & Talu, M. F.** (2021). Bıyık Deseni Üretiminde Çekişmeli Üretici Ağların Performans Karşılaştırması. Bitlis Eren Üniversitesi Fen Bilimleri Dergisi, 10(4), 1575-1589.

[17] **Kutlugün, M. A.** (2017). Gözetimli makine öğrenmesi yoluyla türe göre *metinden ses sentezleme* (Yüksek Lisans Tezi), İstanbul Sabahattin Zaim Üniversitesi, Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği Anabilim Dalı, İstanbul.

[18] **Kavzoğlu, T., & Çölkesen, İ.** (2010). Karar ağaçları ile uydu görüntülerinin sınıflandırılması. Harita Teknolojileri Elektronik Dergisi, 2(1), 36-45.

[19] **Tohma, K., Okur, H. I., Kutlu, Y., & Sertbas, A.** (2023). Sentiment Analysis in Turkish Question Answering Systems: An application of Human-Robot Interaction. IEEE Access.

[20] **Kavzoğlu, T., & Çölkesen, İ.** (2010). Destek vektör makineleri ile uydu görüntülerinin sınıflandırılmasında kernel fonksiyonlarının etkilerinin incelenmesi. Harita Dergisi, 144(7), 73-82.

[21] **Yakut, E. , Elmas, D. & Yavuz, Y.** (2014). YAPAY SİNİR AĞLARI VE DESTEK VEKTÖR MAKİNELERİ YÖNTEMLERİYLE BORSA ENDEKSİ TAHMİNİ . Süleyman Demirel Üniversitesi İktisadi ve İdari Bilimler Fakültesi Dergisi , 19 (1) , 139-157.

[22] **Metlek, S., & Kayaalp, K.** (2020). *Makine Öğrenmesinde, Teoriden Örnek MATLAB Uygulamalarına Kadar Destek Vektör Makineleri*. İksad Yayınevi.

[23] **Ruiz-Gonzalez, R., Gomez-Gil, J., Gomez-Gil, F. J., & Martínez-Martínez, V.** (2014). An SVM-based classifier for estimating the state of various rotating components in agro-industrial machinery with a vibration signal acquired from a single point on the machine chassis. Sensors, 14(11), 20713-20735.

[24] **PİRİM, H.** (2006). Yapay zeka. Yaşar Üniversitesi E-Dergisi, 1(1), 81-93.

[25] **Uğur, A., & Kınacı, A. C.** (2006). Yapay zeka teknikleri ve yapay sinir ağları kullanılarak web sayfalarının sınıflandırılması. XI. Türkiye'de İnternet Konferansı (inet-tr'06), Ankara, 1(4).

[26] **Ersoy, E., & Karal, Ö.** (2012). Yapay sinir ağları ve insan beyni. İnsan ve Toplum Bilimleri Araştırmaları Dergisi, 1(2), 188-205.

[27] **Çukurova Üniversitesi, Bilgisayar Mühendisliği Bölümü, Makine Öğrenmesine Giriş.** (2000). *Makine Öğrenmesi 8. Hafta* [PowerPoint slaytı]. Retrieved from https://ceng.cu.edu.tr/uorhan/DersNotu/Ders08.pdf

[28] **Elmas, B.** (2021). Evrişimli sinir ağları ile ağaç kabuğu görüntülerinden ağaç türlerinin transfer öğrenme yöntemiyle tanımlanması. Gazi Üniversitesi Mühendislik Mimarlık Fakültesi Dergisi, 36(3), 1253-1270.

[29] **Zhao, Z. Q., Zheng, P., Xu, S. T., & Wu, X.** (2019). Object detection with deep learning: A review. IEEE transactions on neural networks and learning systems, 30(11), 3212-3232.

[30] **Zou, Z., Chen, K., Shi, Z., Guo, Y., & Ye, J.** (2023). Object detection in 20 years: A survey. Proceedings of the IEEE.

[31] **Viola, P., & Jones, M.** (2001). Rapid object detection using a boosted cascade of simple features. In Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001 (Vol. 1, pp. I-I). Ieee.

[32] **Dalal, N., & Triggs, B.** (2005). Histograms of oriented gradients for human detection. In 2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05) (Vol. 1, pp. 886-893). Ieee.

[33] **Zaidi, S. S. A., Ansari, M. S., Aslam, A., Kanwal, N., Asghar, M., … & Lee, B.** (2022). A survey of modern deep learning based object detection models. Digital Signal Processing, 126, 103514.

[33] **Felzenszwalb, P., McAllester, D., & Ramanan, D.** (2008). A discriminatively trained, multiscale, deformable part model. In 2008 IEEE conference on computer vision and pattern recognition (pp. 1-8). Ieee.

[35] **Li, Z., Liu, F., Yang, W., Peng, S., & Zhou, J.** (2021). A survey of convolutional neural networks: analysis, applications, and prospects. IEEE transactions on neural networks and learning systems.

[36] **Le Cun, Y., Jackel, L. D., Boser, B., Denker, J. S., Graf, H. P., Guyon, I., ... & Hubbard, W.** (1989). Handwritten digit recognition: Applications of neural network chips and automatic learning. IEEE Communications Magazine, 27(11), 41-46.

[37] **Donuk, K., Arı, A., & Hanbay, D.** (2021). Yüz İmgelerinden Göz Bölgelerinin Tespitinde ESA Tabanlı Alternatif Bir Yaklaşım. Fırat Üniversitesi Mühendislik Bilimleri Dergisi, 33(2), 735-743.

[38] **Girshick, R., Donahue, J., Darrell, T., & Malik, J.** (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 580-587).

[39] **Uijlings, J. R., Van De Sande, K. E., Gevers, T., & Smeulders, A. W.** (2013). Selective search for object recognition. International journal of computer vision, 104, 154-171.

[40] **Çay, T., Ölmez, E., & Er, O.** (2023). Bölgesel Tabanlı Evrişimli Sinir Ağı ile Araç Plaka Tanıma. Düzce Üniversitesi Bilim ve Teknoloji Dergisi, 11(1), 10-20.

[41] **Krizhevsky, A., Sutskever, I., & Hinton, G. E.** (2012). Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 25.

[42] **He, K., Zhang, X., Ren, S., & Sun, J.** (2015). Spatial pyramid pooling in deep convolutional networks for visual recognition. IEEE transactions on pattern analysis and machine intelligence, 37(9), 1904-1916.

[43] **Girshick, R.** (2015). Fast r-cnn. In Proceedings of the IEEE international conference on computer vision (pp. 1440-1448).

[44] **Ren, S., He, K., Girshick, R., & Sun, J.** (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. Advances in neural information processing systems, 28.

[45] **Şenol, H. İ.** (2023). Gelişmiş Deniz Gözlemi: SAR Tabanlı Gemi Tespiti için CNN Algoritmalarının Kullanımı. Türkiye Lidar Dergisi , 5 (1) , 1-7 .

[46] **Altaş, Z., Özgüven, M. M., & Adem, K.** (2023). Bazı Bağ Hastalıklarının Faster R-CNN Modeli ile Otomatik Tespit Edilmesi ve Sınıflandırılması. Turkish Journal of Agriculture-Food Science and Technology, 11(1), 97-103.

[47] **Lin, T. Y., Dollár, P., Girshick, R., He, K., Hariharan, B., & Belongie, S.** (2017). Feature pyramid networks for object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2117-2125).

[48] **Chen, S., Zhao, J., Zhou, Y., Wang, H., Yao, R., Zhang, L., & Xue, Y.** (2023). Info-FPN: An Informative Feature Pyramid Network for object detection in remote sensing images. Expert Systems with Applications, 214, 119132.

[49] **He, K., Zhang, X., Ren, S., & Sun, J.** (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[50] **Karadağ, B., Arı, A., & Karadağ, M.** (2021). Derin öğrenme modellerinin sinirsel stil aktarımı performanslarının karşılaştırılması. Politeknik Dergisi, 24(4), 1611-1622.

[51] **Lu, J., Ma, C., Li, L., Xing, X., Zhang, Y., Wang, Z., & Xu, J.** (2018). A vehicle detection method for aerial image based on YOLO. Journal of Computer and Communications, 6(11), 98-107.

[52] **Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., … & Berg, A. C.** (2016). Ssd: Single shot multibox detector. In Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14 (pp. 21-37). Springer International Publishing.

[53] **Simonyan, K., & Zisserman, A.** (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

[54] **Lin, T. Y., Goyal, P., Girshick, R., He, K., & Dollár, P.** (2017). Focal loss for dense object detection. In Proceedings of the IEEE international conference on computer vision (pp. 2980-2988).

[55] **Zhou, X., Wang, D., & Krähenbühl, P.** (2019). Objects as points. arXiv preprint arXiv:1904.07850.

[56] **Arı, A., & Hanbay, D.** (2019). Tumor detection in MR images of regional convolutional neural networks. Journal of the Faculty of Engineering and Architecture of Gazi University, 34(3), 1395-1408.

[57] **Gao, X., Li, W., Loomes, M., & Wang, L.** (2017). A fused deep learning architecture for viewpoint classification of echocardiography. Information Fusion, 36, 103-113.

[58] **Kemaloğlu, N., & Sevli, O.** (2019). Evrişimsel Sinir Ağları ile İşaret Dili Tanıma. In Proceedings on 2nd International Conference on Technology and Science (pp. 942-948).

[59] **Stride Example**. Erişim tarihi 01.07.2023, https://ai.stackexchange.com/questions/5991/is-my-understanding-of-how-the-convolution-with-stride-2-works-in-this-example-c

[60] **A Beginner's Guide To Understanding Convolutional Neural Networks Part 2**. Erişim tarihi 01.07.2023, https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/

[61] **Deep Learning Fundamentals - Classic Edition**. Erişim tarihi 01.07.2023, https://deeplizard.com/learn/video/qSTv_m-KFk0

[62] **Karataş, A. F., Mercan, Ö. B., Özdil, U., & Şükrü, O**. (2023). Çağrı Merkezlerinde Olumsuzluk İçeren Çağrıların Evrişimsel Sinir Ağları ile Tespiti. Bilişim Teknolojileri Dergisi, 16(1), 13-19.

[63] **Wang, Y., Li, Y., Song, Y., & Rong, X.** (2020). The influence of the activation function in a convolution neural network model of facial expression recognition. Applied Sciences, 10(5), 1897.

[64] **Sharma, S., Sharma, S., & Athaiya, A.** (2017). Activation functions in neural networks. Towards Data Sci, 6(12), 310-316.

[65] **Ther, A., Pandit, B. K., Ganguly, A., Chakraborty, A., & Banerjee, A**. (2023). Resource-efficient VLSI Architecture of Softmax Activation Function for Real-time Inference in Deep Learning Applications. In 2023 International Symposium on Devices, Circuits and Systems (ISDCS) (Vol. 1, pp. 01-05). IEEE.

[66] **Bayram, F.** (2020). Derin öğrenme tabanlı otomatik plaka tanıma. Politeknik Dergisi, 23(4), 955-960.

[67] **Evrişimsel Sinir Ağları ( Convolutional Neural Network).** Erişim tarihi 01.07.2023, https://ece-akdagli.medium.com/evrişimsel-sinir-ağları-convolutional-neural-network-a6c9d8e666c4

[68] **ASLAN, M.** (2022). Derin Öğrenme Tabanlı Otomatik Beyin Tümör Tespiti. Fırat Üniversitesi Mühendislik Bilimleri Dergisi, 34(1), 399-407.

[69] **O'Shea, K., & Nash, R.** (2015). An introduction to convolutional neural networks. arXiv preprint arXiv:1511.08458.

[70] **Terven, J., & Cordova-Esparza, D.** (2023). A comprehensive review of YOLO: From YOLOv1 to YOLOv8 and beyond. arXiv preprint arXiv:2304.00501.

[71] **Redmon, J., & Farhadi, A.** (2017). YOLO9000: better, faster, stronger. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 7263-7271).

[72] **Redmon, J., & Farhadi, A.** (2018). Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767.

[73] **Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. M.** (2020). Yolov4: Optimal speed and accuracy of object detection. arXiv preprint arXiv:2004.10934.

[74] **YOLOv5.** Erişim tarihi 01.05.2023, https://github.com/ultralytics/yolov5

[75] **Li, C., Li, L., Jiang, H., Weng, K., Geng, Y., Li, L., & Wei, X.** (2022). YOLOv6: A single-stage object detection framework for industrial applications. arXiv preprint arXiv:2209.02976.

[76] **Reis, D., Kupec, J., Hong, J., & Daoudi, A.** (2023). Real-Time Flying Object Detection with YOLOv8. arXiv preprint arXiv:2305.09972.

[77] **Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., & Rabinovich, A.** (2015). Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9).

[78] **Everingham, M., Van Gool, L., Williams, C. K., Winn, J., & Zisserman, A.** (2010). The pascal visual object classes (voc) challenge. International journal of computer vision, 88, 303-338.

[79] **Shetty, S.** (2016). Application of convolutional neural network for image classification on Pascal VOC challenge 2012 dataset. arXiv preprint arXiv:1607.03785.

[80] **Hussain, M.** (2023). YOLO-v1 to YOLO-v8, the Rise of YOLO and Its Complementary Nature toward Digital Manufacturing and Industrial Defect Detection. Machines, 11(7), 677.

[81] **Stanford Üniversitesi, Bilgisayar Mühendisliği Bölümü, Object Detection.** *Object Detection* [PowerPoint slaytı]. Retrieved from http://cs231n.stanford.edu/2021/slides/2021/discussion_6_detection.pdf

[82] **Ku, B., Kim, K., & Jeong, J.** (2022). Real-time ISR-YOLOv4 based small object detection for safe shop floor in smart factories. Electronics, 11(15), 2348.

[83] **Mehmet, S., AYDIN, İ., & Erhan, A.** (2023). YOLOv5 ile Topluluk Öğrenmesine Dayalı Olarak Ray Yüzeyindeki Kusurların Tespiti. Demiryolu Mühendisliği, (17), 115-132.

[84] **Yang, W., Chen, Y., Huang, C., & Gao, M.** (2018). Video-based human action recognition using spatial pyramid pooling and 3D densely convolutional networks. Future Internet, 10(12), 115.

[85] **PyTorch**. Erişim tarihi 01.05.2023, https://pytorch.org

[86] **Introduction to YOLOv5**. Erişim tarihi 01.07.2023, https://pythonistaplanet.com/yolov5/

[87] **YOLOv6 Object Detection – Paper Explanation and Inference**. Erişim tarihi 01.07.2023, https://learnopencv.com/yolov6-object-detection/

[88] **YOLO: Algorithm for Object Detection Explained [+Examples]**. Erişim tarihi 01.07.2023, https://www.v7labs.com/blog/yolo-object-detection

[89] **HANBAY, K., & Hüseyin, Ü.** (2017). Nesne tespit ve takip metotları: Kapsamlı bir derleme. Türk Doğa ve Fen Dergisi, 6(2), 40-49.

[90] **Build a Weapon Detection Algorithm using YOLOv7**. Erişim tarihi 01.07.2023, https://medium.com/the-modern-scientist/build-a-weapon-detection-algorithm-using-yolov7-8d1787c93f96

[91] **Jiang, K., Xie, T., Yan, R., Wen, X., Li, D., Jiang, H., & Wang, J.** (2022). An attention mechanism-improved YOLOv7 object detection algorithm for hemp duck count estimation. Agriculture, 12(10), 1659.

[92] **Yunus, E.** (2023). YOLO V7 and Computer Vision-Based Mask-Wearing Warning System for Congested Public Areas. Journal of the Institute of Science and Technology, 13(1), 22-32.

[93] **Hussain, M., Al-Aqrabi, H., Munawar, M., Hill, R., & Alsboui, T.** (2022). Domain feature mapping with YOLOv7 for automated edge-based pallet racking inspections. Sensors, 22(18), 6927.

[94] **YOLOv7 Object Detection – Paper Explanation and Inference**. Erişim tarihi 01.05.2023, https://learnopencv.com/yolov7-object-detection-paper-explanation-and-inference/

[95] **What is YOLOv7?**. Erişim tarihi 01.05.2023, https://roboflow.com/model/yolov7#:~:text=YOLOv7%20is%20a%20state%20of%20the%20art%20object%20detection%20model.

[96] **Lin, T. Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., & Zitnick, C. L.** (2014). Microsoft coco: Common objects in context. In Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13 (pp. 740-755). Springer International Publishing.

[97] **COLAB.** Erişim tarihi 01.05.2023, https://colab.research.google.com

[98] **COLAB Sık Sorulan Sorular.** Erişim tarihi 01.07.2023, https://research.google.com/colaboratory/intl/tr/faq.html#:~:text=Temel%20Bilgiler&text=Colaboratory%20(ya%20da%20kısaca%20%22Colab,rastgele%20Python%20kodu%20yazıp%20yürütebilir.

[99] **Google Colab Nedir ve Nasıl Kullanılır?.** Erişim tarihi 01.07.2023, https://globalaihub.com/google-colab-nedir-ve-nasil-kullanilir/

[100] **Jupyter.** Erişim tarihi 01.05.2023, https://jupyter.org

[101] **TensorFlow.** Erişim tarihi 01.05.2023, https://www.tensorflow.org/?hl=tr

[102] **Keras.** Erişim tarihi 01.05.2023, https://keras.io

[103] **Caffe.** Erişim tarihi 01.05.2023, https://caffe.berkeleyvision.org

[104] **Temiz, H., Bilge, H. Ş., & Uyğur, S.** (2019). Colaboratory: Makine Öğrenmesi ve Derin Öğrenme Çalışmalarında Donanım Gereksinimini Karşılamada Alternatif Bir Çözüm. In SETSCI Conference Proceedings (Vol. 4, No. 1, pp. 568-571).

[105] **React Native.** Erişim tarihi 01.05.2023, https://reactnative.dev

[106] **Expo.** Erişim tarihi 01.05.2023, https://expo.dev

[107] **Journey putting YOLO v7 model into TensorFlow Lite (Object Detection API) model running on Android.** Erişim tarihi 01.05.2023, https://medium.com/geekculture/journey-putting-yolo-v7-model-into-tensorflow-lite-object-detection-api-model-running-on-android-e3f746a02fc4

[108] **Github.** Erişim tarihi 01.05.2023, https://github.com

[109] **Batukar Github.** Erişim tarihi 01.07.2023, https://github.com/batukar/tez

## ÖZGEÇMİŞ

**Ad-Soyad** **:** Batuhan KARADAĞ

**ÖĞRENİM DURUMU:**

- **Lisans** **:** 2019, Munzur Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü.

**MESLEKİ DENEYİM:**

- 2022'den itibaren İskenderun Teknik Üniversitesi Mühendislik ve Doğa Bilimleri Fakültesi Bilgisayar Mühendisliği Bölümü'nde Araştırma Görevlisi olarak çalışmaktadır.

**YÜKSEK LİSANS VEYA DOKTORA TEZİNDEN TÜRETİLEN ÇALIŞMALAR (Makaleler, Bildiriler, Patentler v.b.)**

- **Karadağ, B. & Arı, A.** (2023). Akıllı Mobil Cihazlarda YOLOv7 Modeli ile Nesne Tespiti . Politeknik Dergisi , , 1-1 . DOI: 10.2339/politeknik.1296541