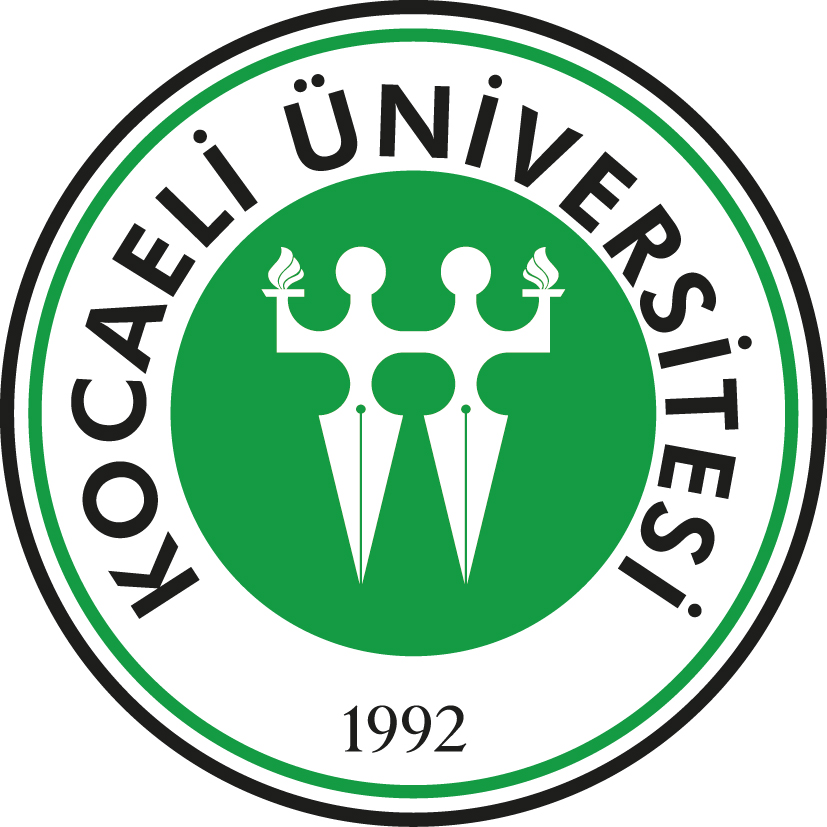
**T.C.**

**KOCAELİ ÜNİVERSİTESİ**

**TEKNOLOJİ FAKÜLTESİ**

**BİYOMEDİKAL MÜHENDİSLİĞİ**

**BİRİNCİ DÖNEM BİTİRME PROJESİ RAPORU**



**Görüntü İşleme Temelli Maske Kontrol Sistemi**

**HAZIRLAYAN: DOĞUKAN BİÇER**

**ÖĞRENCİ NO:191305052**

**YIL: 2022**

**Özet:**

Korona virüsün (COVID-19) hızlı bulaşması nedeniyle dünya büyük bir sağlık kriziyle karşı karşıya kalmıştır. Korona virüsün yayılmasını engellemek için Dünya Sağlık Örgütüne (WHO) göre en etkili tedbir, halka açık yerlerde ve kalabalık alanlarda maske takmaktır. Ancak kalabalık ortamlarda uzun süre kalan kişilerde sıkılma, boş verme ve umursamazlık gibi nedenlerle insanlar bu kuralı ihlal edebilmektedir. Bu nedenle kalabalık alanlarda insanların izlenmesi ve gerektiğinde ilgililerin uyarılarak toplum sağlığını korumak önem arz etmektedir. Bu çalışmada maske takmayan, maskesini yanlış takan ve maskesini doğru takan kişileri belirleme sürecini otomatikleştirmek için derin öğrenme modeli kullanan bir yazılım geliştirilmiştir. İnternetten elde edilen veri setleri ve çevreden alınan fotoğraflar kullanılarak özgün bir veri seti oluşturulmuştur. Çalışmada gerçek zamanlı olarak maskeli, maskesiz ve maskesini yanlış takan kişilerin tespiti gerçekleştirilmiş sesli olarak kişilere dönütler verilmiştir.

Hız ve eğitim süresi açısından YOLOv7-tiny algoritmasıyla geliştirilen modelin daha başarılı olduğu tespit edilmiştir. Çalışma kapsamında geliştirilen yapay zekâ modellerinin farklı donanımlar üzerinde gerçek zamanlı çalıştırılması için MKS (Maske Kontrol Sistemi) olarak adlandırılan etkileşimli yazılım önerilmektedir.

Geliştirilen yazılımın COVID-19 tedbirlerinin kontrolü ve toplum bilincinin arttırılmasında yararlı olacağı düşünülmektedir.

**3. Giriş**

Görüntü İşleme Temelli Maske Kontrol Sisteminin hedefi, gerçek zamanlı maske tespiti yapıp 3 farklı durumla ilgili sesli geri bildirim vermesidir. Bu üç durum:

1.Maske takıldığında

2.Maske takılmadığında

3.Kişi tespit edilmediğinde

Olarak sıralanabilir. Bu üç duruma göre çeşitli senaryolar uygulanabilir. (örn: Maskesini takan kişilere kapıyı otomatik olarak açma, maske takan ve takmayan kişileri sayma ve kaydetme gibi…)

Yapılan benzer çalışmalar incelendiğinde:

1376 adet görsel üzerinde destek vektörleri (SVM) ve MobileNetV2 birleştirilmesi ile yapılan başka bir

çalışmada maske takan kişilerin sınıflandırılması yapılmış ve başarı oranı %97,1 elde edildiği görülmüştür [9].

Yapılan başka bir çalışmada 1415 adet veri seti üzerinde özellik çıkarma için ResNet50, nesne algılama için YOLOv2 algoritmaları kullanılarak %81 hassasiyet yüzdesine ulaşıldığı belirtilmektedir [10].

YOLOv3 algoritması ile maske tespiti yapan bir çalışmada 4000 iterasyon sonunda %96 başarı elde ettiği görülmüştür [11].

Maskeli ve maskesiz kişilerin tespiti için ilk aşamada yüzleri tespit eden bir modelden geçirildikten sonra ikinci aşamada tespit edilen yüzde maske olup olmadığı sınıflandıran CNN tabanlı bir ağ kullanılmış ve %99,5 doğruluk oranı elde edilmiştir [12].

Yüzde maske tespiti için MobileNetV2, VGG16 ve ResNet50 derin öğrenme algoritmalarının kullanıldığı çalışmada en iyi başarımın %97,82 ile VGG16 algoritması ile elde ifade edilmektedir [13].

YOLOv5 derin öğrenme algoritması ile maske tespiti yapan başka bir çalışmada 300 iterasyon sonunda %96,5 doğruluğa erişildiği görülmüştür [14].

SSD (Single Shot Detector) ve MobileNetV2 sınıflandırıcının birleştirilmesi ile yapılan başka bir çalışma

%92,64 başarı oranı ulaştığı ve gömülü sistemlerde kullanılabilecek düzeyde olduğu belirtilmiştir [15].

4098 adet görsel ile oluşturulmuş bir ver seti üzerinde maskesini takan ve takmayan kişileri sınıflandırmak için MobileNetV2 derin öğrenme ağı üzerinden transfer öğrenme tekniği kullanılarak yapılan başka bir çalışmada %98’lik doğruluğa ulaşıldığı görülmüştür [16].

25.000 görüntüden oluşturulan maskesiz ve maskeli kişileri gerçek zamanlı Raspberry pi donanımı üzerinde ayrıştıran bir çalışmada MobileNetV2 ve SSD mimarileri kullanılarak %96 doğruluk elde edildiği rapor edilmiştir [17].

Maskesini takan kişilere kapıyı otomatik olarak açmasını hedefleyen başka bir çalışmada MobileNetV2 kullanılmış %95,85 doğruluk elde edildiği görülmüştür [18].

Görüldüğü gibi son yıllarda literatürde maske kontrolü yapabilen derin öğrenme yöntemleri ile geliştirilen

Değişik çalışmaların yapılmaya başlandığı görülmektedir.

**4. Kavramlar**

**Makine Öğrenmesi nedir?**

Makine öğrenmesi, insanların öğrenme şekillerini taklit etmek için veri ve algoritmaların kullanımına odaklanıp doğruluğunu kademeli olarak artıran bir [yapay zeka (AI)](https://www.ibm.com/tr-tr/cloud/learn/what-is-artificial-intelligence) ve bilgisayar bilimi dalıdır.

**Derin öğrenme nedir?**

Derin öğrenme, yapay sinir ağlarının (insan beyni gibi çalışacak şekilde modellenen algoritmalar) büyük miktarda veriden öğrendiği makine öğreniminin (ML) bir altkümesidir.

**Derin öğrenme nasıl çalışır?**

Derin öğrenme, insan beyninin çalışma şekline göre genel hatlarıyla modellenen algoritmalar olan sinir ağlarının katmanları tarafından desteklenir. Büyük miktarlarda veri ile eğitim, sinir ağındaki nöronları konfigüre etmektir. Sonuç, eğitildikten sonra yeni verileri işleyen derin öğrenme modelidir. Derin öğrenme modelleri, birden fazla veri kaynağından bilgi alır ve bu verileri insan müdahalesine gerek kalmadan gerçek zamanlı olarak analiz eder. Derin öğrenmede, grafik işleme birimleri (GPU'lar), aynı anda birden fazla hesaplamayı işleyebildikleri için eğitim modellerine yönelik olarak optimize edilmiştir.

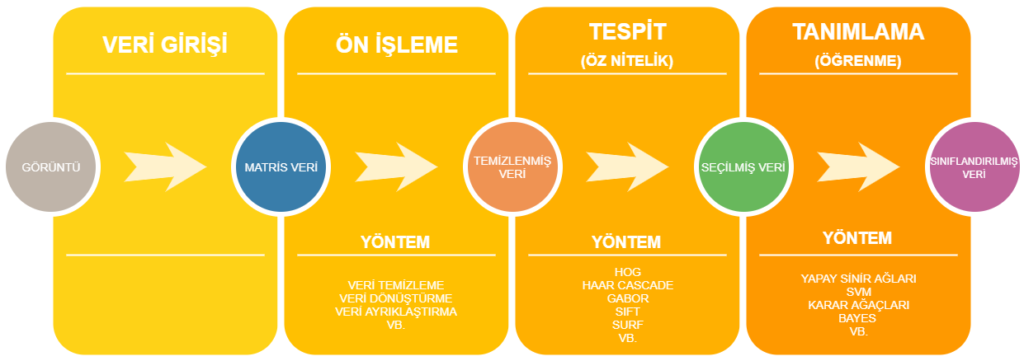
**Nöral ağlar nedir ve nasıl çalışır?**

Yapay nöral ağlar (ANN) ya da benzetimli nöral ağları (SNN) olarak da bilinen nöral ağlar, [makine öğrenmesinin](https://www.ibm.com/tr-tr/cloud/learn/machine-learning) bir alt kümesidir ve [derin öğrenme](https://www.ibm.com/cloud/learn/deep-learning) algoritmalarının temelini oluşturur. Biyolojik nöronların birbirine sinyal göndermesini taklit ettiğinden, ismini ve yapısını insan beyninden alır.

**Görüntü işleme algoritmaları nasıl çalışır?**

Görüntü işleme, her biri belirli bir konuma ve değere sahip olan adına resim öğeleri, görüntü öğeleri veya piksel de dediğimiz öğelerin sınırlı sayıda bir araya gelmesiyle oluşan dijital görüntünün bazı bilgisayar algoritmaları ve görsel teknikler kullanılarak amaca uygun hale getirme yöntemine denir. Görüntü bu bilgisayar algoritmaları ile bir fonksiyona atanır ve işlemler bu fonksiyon üzerinden yapılır. Bir görüntü, iki boyutlu bir fonksiyon f(x, y) olarak tanımlanabilir. (x,y) görüntünün o noktadaki yoğunluğu veya gri seviyesi olarak adlandırılır.

**Nesne tanıma ve makineye nesne öğretme**



**Şekil 1 –** Nesne tanıma ve makineye nesne öğretme diyagramı

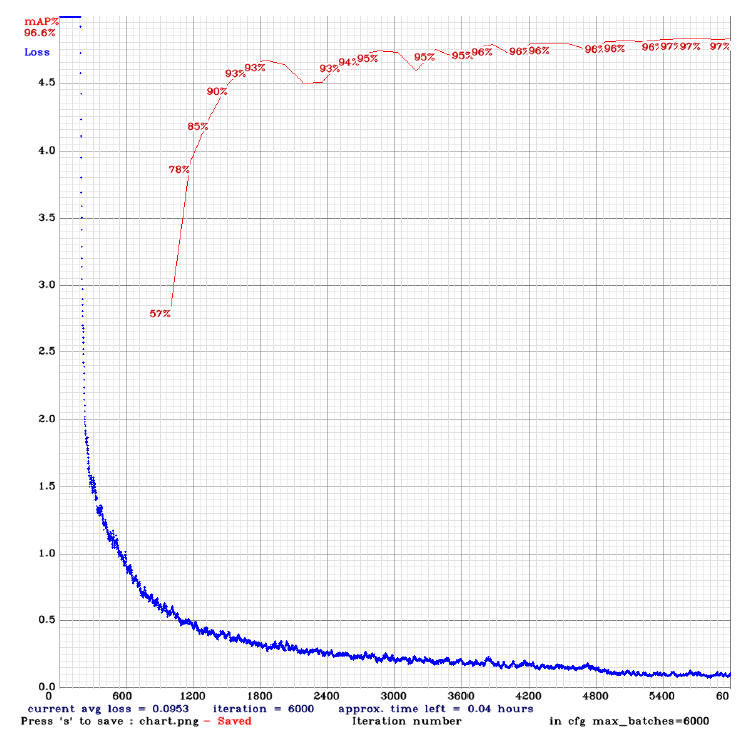
**Gerçek zamanlı obje tespiti**

Nesne tespiti, dijital görüntülerde ve videolarda belirli bir sınıftaki (insanlar, binalar veya arabalar gibi) anlamsal nesnelerin örneklerini algılamakla ilgilenen, bilgisayarla görme ve [görüntü işleme](https://tr.wikipedia.org/wiki/Say%C4%B1sal_g%C3%B6r%C3%BCnt%C3%BC_i%C5%9Fleme) ile ilgili bir bilgisayar teknolojisidir.[[1]](https://tr.wikipedia.org/wiki/Nesne_tespiti) Nesne tespiti, bilgisayarla görme ve görüntü işlemeden farklı olarak algılanan nesnenin görüntü üzerinde koordinatlarının bulunmasını içerir. Bulunan koordinatlar ile nesnenin bir çerçeve ile içine alınacağı alan da tespit edilmiş olur. Nesne tespiti, gerçek zamanlı (anlık) ve gerçek zamanlı olmayan olarak ikiye ayrılır. Üzerinde iyi araştırma yapılmış alanlar [yüz tespiti](https://tr.wikipedia.org/wiki/Y%C3%BCz_tan%C4%B1ma), [yaya tespiti](https://tr.wikipedia.org/w/index.php?title=Yaya_alg%C4%B1lama&action=edit&redlink=1) ve [araç tespitidir](https://tr.wikipedia.org/w/index.php?title=Ara%C3%A7_tespiti&action=edit&redlink=1). Nesne tespiti, görüntü alma ve [video gözetimi](https://tr.wikipedia.org/wiki/Kapal%C4%B1_devre_televizyon) dahil olmak üzere bilgisayarla görmenin birçok alanında uygulamaya sahiptir.

**5. Proje Adımları**

Bu çalışmanın temelinde derin öğrenme algoritmaları kullanılmıştır. Yapay zekânın alt kümelerinden olan derin öğrenme tekniği klasik makine öğrenmesi tekniklerinden farklı olarak dış müdahale olmadan modellerin yanlış karar verip vermediği algılayabilen bir tekniktir [21]. Çalışmada derin öğrenme yöntemlerinden YOLOv7 ve YOLOv7-tiny nesne tespiti algoritmaları kullanılmıştır. YOLOv7-tiny YOLOv7’e göre katman sayısı ve modüller değiştirilerek gömülü sistemler, mobil cihazlar ve düşük donanıma sahip cihazlar için daha iyi FPS değerleri elde etmek için tasarlanmıştır [22].

Çalışmada kullanılan veri seti üzerinde iki farklı derin öğrenme algoritması (YOLOv7-tiny ve YOLOv7) kullanılarak yapay zekâ modelleri Google Colab platformu üzerinde Tesla P100-PCIE-16GB GPU kullanılarak eğitilmiştir. Her iki model için geliştirilen ortam ve iterasyon sayısı 6000 olarak belirlenmiştir. Model 1 ve Model 2 olarak adlandırılan modeller sırasıyla YOLOv7-tiny ve YOLOv7 algoritmalarının COCO veri seti üzerinde eğitilen ağırlıkları kullanılarak transfer öğrenmesi yöntemiyle eğitilmiştir.



**Şekil 2 –** YOLOV7-tiny ile geliştirilen modele ait mAP, kayıp (loss) / İterasyon Grafiği

Şekil 1’de YOLOv7-tiny ile geliştirilen Model 1’e ait mAP, kayıp (loss) ve iterasyon değerlerine ait grafik

Görülmektedir. Bu grafik Google Colab üzerinde eğitim sonunda elde edilmiştir. Şekil 6’da 6000 iterasyon sonunda mavi renk ile gösterilen kayıp (loss) değerlerinin azaldığı ve kırmızı renk ile gösterilen mAP değerlerinin arttığı görülmüştür.

**Çalışmada kullanılan bilgisayar donanımı:**

|  |  |
| --- | --- |
| **Donanım** | **Özellikleri** |
| Asus Zenbook 13 | **CPU:** AMD Ryzen 5 5500U 2.10 GHz  **Ram:** 16 GB  **GPU:** Radeon Graphics |

Görüntü İşleme Temelli Maske Kontrol Sistemi:

Python dilinde yazılmıştır (kullanılan sürüm: 3.7) ve pycharm ide’sinde geliştirilmiştir.

Python’da Kullanılan kütüphaneler ise.

**1.YOLOv7-tiny**

YOLO konvolüsyonel sinir ağları kullanarak nesne tespiti yapan bir algoritmadır. Açılımı ‘’You Only Look Once’’ demektir.Sebebi ise algoritmanın nesne tespitini oldukça hızlı bir şekilde ve tek seferde yapabiliyor olmasıdır[2].

**2.** **pyttsx3**

Python’da yazılı metini sese çevirmeye yarayan kütüphanedir. Windows’un içerisindeki ses sentezleyici kullanır (Türkçe desteği mevcuttur) ve çevrimdışı çalışabilir.

**3.Opencv**

OpenCV (Open Source Computer Vision) açık kaynak kodlu görüntü işleme kütüphanesidir [3]

**4.Tensorflow**

Açık kaynak kodlu bir deep learning(derin öğrenme) kütüphanesidir. Esnek yapısı sayesinde, tek bir API ile platform farketmeksizin hesaplamaları, bir veya birden fazla CPU, GPU kullanarak deploy etmenize olanak sağlar. Temelinde Python kullanılarak geliştirilen bu framework, günümüzde Python‘ın yanısıra C++, Java, C#, Javascript ve R gibi birçok dili desteklemektedir[4].

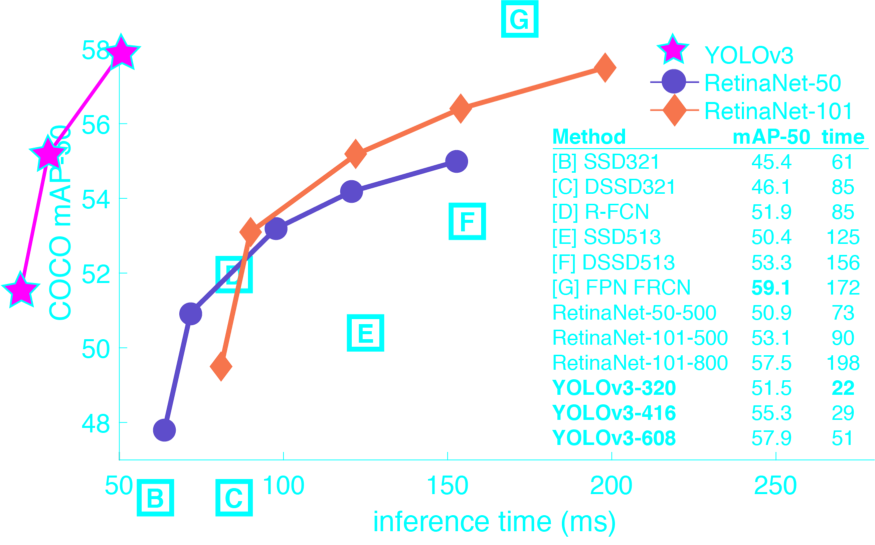
Geliştirilen yazılım yolov7 tiny kütüphanesindeki farklı nesnelerinde tespit edebilmektedir (Örn: insan ,bisiklet, araba, motosiklet, uçak gibi).

MKS (Maske Kontrol Sistemi) yazılımı kişilerin yüzünde maskesiz, maskeli ve yanlış maske kullanımı yapan kişilerin gerçek zamanlı

Tespitini yaparak kişileri sesli olarak uyarabilmektedir. MKS yazılımının uygulanması için Asus Zenbook 13 bilgisayarda Windows 11 işletim sistemi kullanılmıştır.

**YOLO Algoritması nasıl çalışır**

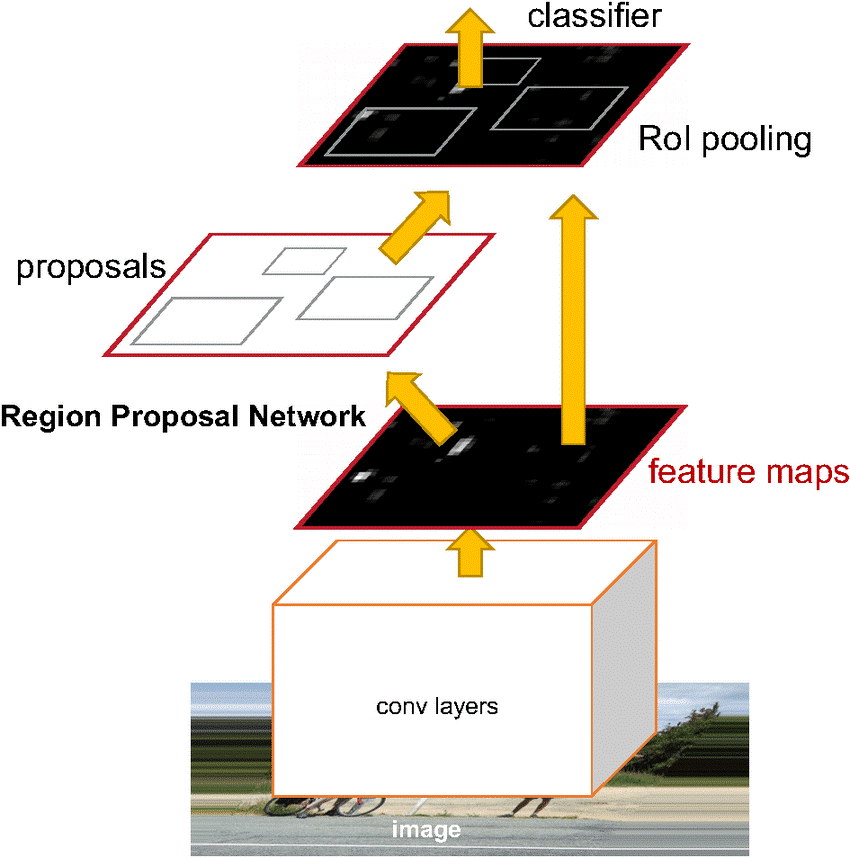
Son yıllarda nesne tespiti alanında revaçta olan YOLO (You Only Look Once) algoritmasını ister istemez duymuşsunuzdur. Bu algoritma neden bu kadar popüler? Sebebi tahmin edebileceğiniz gibi tabii ki iyi olması. Peki bu zamana kadar olan algoritmalar kötü müydü? Hayır, YOLO’dan önce de şu anda da YOLO’dan iyi tahmin yapabilen algoritmalar vardı fakat temel sorun yavaş olmalarıydı. YOLO’yu diğer algoritmalardan ayıran en önemli özelliği gerçek zamanlı nesne tespiti yapabilmesi oldu. Peki YOLO’dan önce hiç gerçek zamanlı nesne tespiti yapabilen bir algoritma yok muydu? Vardı fakat genel ortalama kesinlik (mAP) değerleri yeterli değildi.



**Şekil 3 –** YOLOv3 vs. Diğer Algoritmalar (1)

Yukarıdaki resimde YOLOv3 ve diğer algoritmaların COCO veri setinde 0.5 IoU (mAP-50) ile karşılaştırmasını görüyorsunuz. Grafikten de anlaşılacağı üzere YOLO rakiplerine karşı süre ve doğruluk açısından çok iyi durumda. Ayrıca istersek doğruluk ve hız arasında rahatlıkla takas yapabiliyoruz.

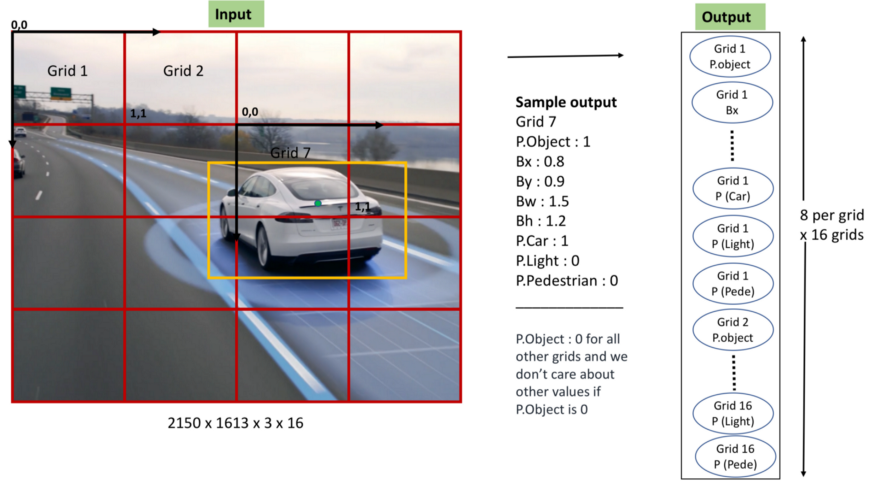
Peki YOLO nasıl hem bu kadar hızlı tahminler yaparken aynı zamanda çok iyi sonuçlar verebiliyor? Öncelikle isterseniz diğer algoritmaların neden yavaş olduğuna bakalım. Örneğin R-CNN gibi bölge bazlı nesne tespit algoritmaları önce nesne bulunması muhtemel alanları belirleyip ardından oralarda ayrı ayrı CNN (Convolutional Neural Network, Evrişimsel Sinir Ağları) sınıflandırıcıları yürütüyor. Bu yöntem her ne kadar iyi sonuçlar verse de bir resim iki ayrı işleme tabi tutulduğu için resim üzerindeki işlem sayımız artıyor ve düşük bir FPS (Frames per second, saniye başına kare) almamıza sebep oluyor.



**Şekil 4 –** Faster R-CNN Yapısı

Eğer isterseniz R-CNN’den sonra gelişmiş versiyonları olan Fast R-CNN ve Faster R-CNN algoritmalarında ayrı ayrı segmentasyon ile belirlenmiş bölgeler yerine tüm resmi tek seferde CNN yapısından geçirip ardından önceden yazılmış Region Proposal Network’e gönderen yapısıyla bu işlemi nasıl hızlandırdığını araştırabilirsiniz fakat algoritmaların içine bu yazıda girmeyeceğim.

Sonuç olarak Faster R-CNN algoritması bile gerçek zamanda ortalama 7 FPS ile çalışıyor. YOLO algoritmasının bu kadar hızlı olmasının sebebi resmi tek bir seferde nöral ağdan geçirerek resimdeki tüm nesnelerin sınıfını ve koordinatlarını tahmin edebiliyor. Yani bu tahmin işleminin temeli, nesne tespitini tek bir regresyon problemi olarak ele almalarında yatıyor. Bunu yapmak için ilk önce girdi resmini SxS‘lik ızgaralara bölüyor. Bu ızgaralar 3x3 5x5 19x19 vs. olabilir. Buna bağlı olarak isterseniz resim nöral ağdan geçtikten sonra elimize geçen vektörü inceleyelim:



**Şekil 5 –** Örnek YOLO algoritması çıktısı

Her bir ızgara kendi içinde, alanda nesnenin olup olmadığını, varsa orta noktasının içinde olup olmadığını, orta noktası da içindeyse uzunluğunu, yüksekliğini ve hangi sınıftan olduğunu bulmakla sorumlu. Daha açık anlatmak gerekirse örneğin yukarıdaki resimde arabanın orta noktası 7. ızgaraya denk geldiği için arabanın tespit edilmesinden/etrafına kutucuk çizmesinden o ızgara sorumlu.

Buna göre YOLO her ızgara için ayrı bir tahmin vektörü oluşturur. Bunların her birinin içinde:

**Güven skoru:** Bu skor modelin geçerli ızgara içinde nesne bulunup bulunmadığından ne kadar emin olduğunu gösterir. (0 ise kesinlikle yok 1 ise kesinlikle var) Eğer nesne olduğunu düşünürse de bu nesnenin gerçekten o nesne olup olmadığından ve etrafındaki kutunun koordinatlarından ne kadar emin olduğunu gösterir.

**Bx:** Nesnenin orta noktasının x koordinatı

**By:** Nesnenin orta noktasının y koordinatı

**Bw:** Nesnenin genişliği

**Bh:** Nesnenin yüksekliği

**Bağlı Sınıf Olasılığı:** Modelimizde kaç farklı sınıf varsa o kadar sayıda tahmin değeri. Örn;

Yukarıdaki resimde Grid 7’ ye baktığımızda eğer araba olduğundan kesin olarak eminse:

Araba: 1, Yaya: 0 olacaktır.

Devam etmeden önce güven skoru nasıl hesaplanıyor onu görelim:

**Güven skoru =** Kutu Güven Skoru *x* Bağlı Sınıf Olasılığı

**Kutu Güven Skoru =** P(nesne) . IoU

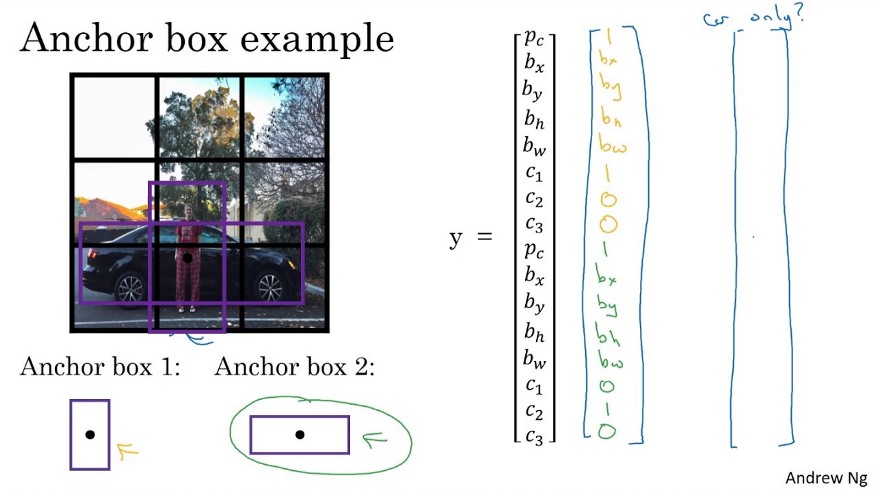
**P(**nesne**) =** Kutunun nesneyi kapsayıp kapsamadığının olasılığı. (Yani nesne var mı yok mu?)

**IoU =** Ground truth ile tahmin edilmiş kutu arasındaki IoU değeri

Yani aslında hiçbir nesne olmayan ızgaralarda bağlı sınıf olasılığı 0 olması gerektiği için (aslında arka plan olarak tespit ediliyor) güven skoru 0 olacaktır.

Yukarıdaki çıktı vektörüne göre her bir ızgara sadece 1 tane nesne tanımlayabiliyor. Örneğin sadece 3x3’lük bir ızgara kullansaydık 9 tane nesne tahmini yapabilirdik. Peki bir ızgarada birden fazla nesne varsa ne olacak? Hatta bir ızgarada 2 farklı nesnenin orta noktası bulunursa ne olacak diye sorsak daha mantıklı olur.

Bu sorun ise YOLOv2’ de algoritmaya monte edilen Anchor Box’ları ile çözülüyor. Anchor Boxes methodu ilk olarak Faster R-CNN’ de kullanılmış ve mantığında el ile seçilmiş belli kalıpların yardımıyla nesnenin etrafındaki kutuyu tahmin etmemiz yatıyor. Ayrıca her bir ızgarada önceden belirlenmiş anchor box sayısı kadar tahmin yapıyoruz.



**Şekil 6 –** Andrew Ng’nin C4W3L08 dersinden Anchor box oluşturması

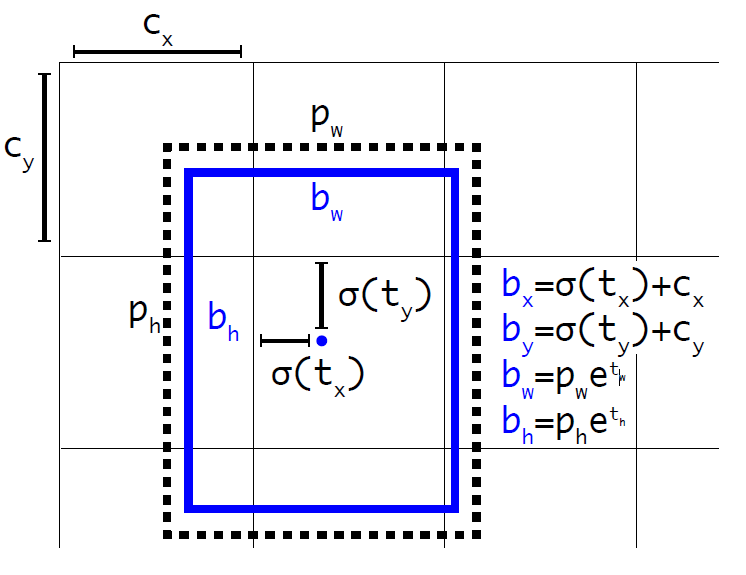
Anchor Box’larının da gelmesiyle çıktı vektörümüz şu şekilde şekilleniyor:

**S x S x ( #A x (5 + #C )**

Yani S x S toplam ızgara sayısını belirttiğine göre her bir ızgara için anchor box sayısı kadar Güven, x, y, w, h ve diğer sınıflar için olasılık hesaplayacağız. Yukarıdaki resimde 2 tane anchor box ve orta noktası aynı ızgaraya denk gelmiş iki tane nesne görüyorsunuz. İnsan şekli birinci anchor box’a araba ise ikincisine daha çok benzediği için de o ızgaranın vektörü de resimdeki gibi oluyor diyebiliriz. (Sarı kısım 1. Anchor box yeşil ise 2. si için) Anchor box’a ne kadar benzediğini hesaplamak için de IoU kullanıyoruz.

Amacımız tespit edilen nesnenin etrafına bir kutu çizmek olduğunu ve aynı zamanda bu kutuların belirli yükseklik-genişlik oranları olduğunu biliyoruz. YOLO v1’de nesnelerin etrafındaki kutuları ağın sonunda kurulu olan birbirine bağlanmış katmanlar ile direkt tahmin ediyorduk. Fakat artık anchor box’lar da elimizde olduğuna göre direkt olarak nesne kutucuğunun en benzediği anchor box’u bulup o kutucuğun anchor box’a göre yükseklik ve genişlik çıkıntılarını/farkını tahmin edebiliriz. Bu şekilde de YOLO v1’deki erken eğitme safhalarında karşımıza çıkan kararsızlık problemini azaltıyoruz. (Daha fazla bilgi için YOLO paper’ına bakabilirsiniz)

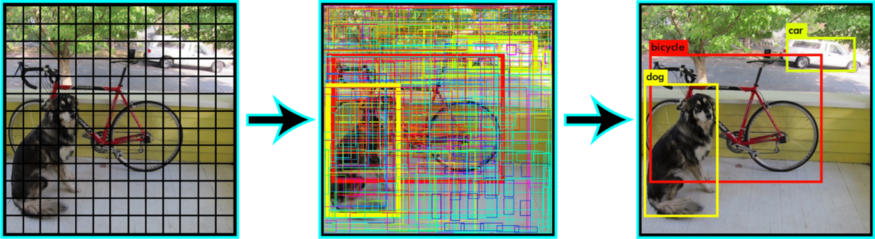
Bu tahminlerin pratikte yerini bulması için göreceli bir sistem kurulması gerekiyor. Elimizde neler var? **Tx, Ty, Tw** ve **Th** zaten her bir kutucuk için ağ tarafından tahmin ediliyor. Aynı zamanda işlem yaptığımız ızgaranın hangi ızgara olduğunu da biliyoruz, buna göre de ızgaranın sol üst köşeye olan uzaklığını bulabiliriz. Bu uzaklıklara da **Cx** ve **Cy** diyelim. Önceden belirlediğimiz anchor box’un genişlik ve yüksekliklerine **Pw** ve **Ph** dersek sistemimiz şu şekilde kuruluyor:



**Şekil 7 –** Yolo algoritmasının tespit edilen nesnenin etrafında kutucuk oluşturması

Ayrıca bu gördüğümüz sistemle parametreleri 0 ile 1 arasında normalleştirdiğimiz için ağımız daha kararlı oluyor ve bu parametreleri daha kolay öğreniyor.

Buraya kadar YOLO’nun her bir ızgara için nesne kutularını nasıl tahmin ettiğini ve bir kutu için kaç tane tahmin yaptığı hakkında genel bir fikrimiz oldu. Fakat algoritma çalışırken çok fazla gereksiz kutular çıkacaktır hatta sadece bir nesne için birkaç farklı kutu bile çıkabilir. Gereksiz kutuları atmak kolay olacaktır, zaten elimizde o ızgaranın içinde nesne olup olmadığını tahmin eden bir parametre var fakat ızgaranın içinde nesne varsa ve aynı nesne için birden fazla ızgara o nesnenin orta noktası olduğunu düşünürse ne olacak?



**Şekil 8 –** Yolo algoritmasının görüntü ayrımı yapması için ızgara oluşturması.

Burada devreye Non max Suppression algoritması devreye giriyor. İsterseniz gelin algoritmaya bakalım:

**1-)** Güven skoru belli bir seviyenin altında olan tüm kutuları at (örn. 0.5)

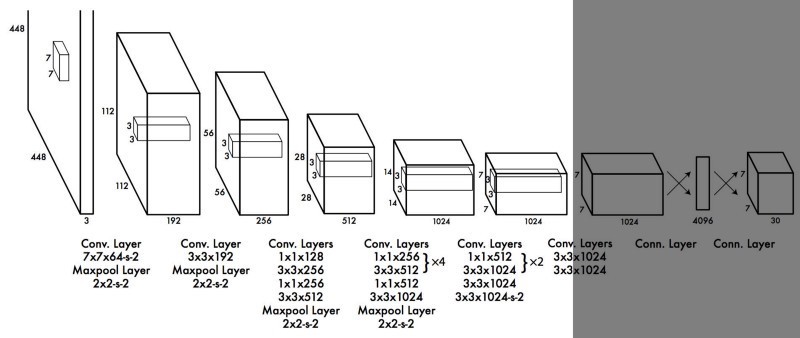
*Kutu kaldığı sürece:*

**1-)** En yüksek güven skorlu kutuyu seç ve onu çıktı olarak ver. Bu kutuya A diyelim.

**2-)** A ile IoU değeri 0.5’ten fazla olan diğer tüm kutuları at

Bu işlem sonucunda da elimizde her nesne için bir tane kutucuk kalmış oluyor.

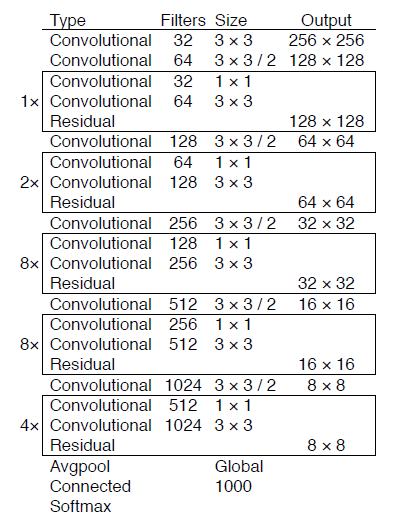
Genel olarak algoritmanın nasıl çalıştığı hakkında belli bir bilgi sahibi olduğumuza göre son olarak gelin biraz yapısını inceleyelim. İlk olarak ağın yapısına bakalım:



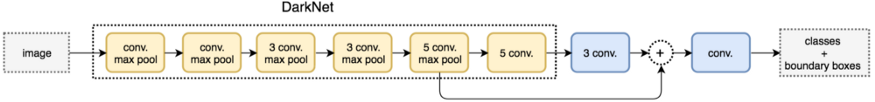
**Şekil 9 –** Andrew Ng’nin C4W3L08 dersinden

Dediğimiz gibi, anchor box’ların gelmesiyle YOLOv1’deki birbirine bağlı katmanlar kaldırıldı. Ayrıca CNN’ler yapısı gereği girdi resmini küçülterek ilerliyor ve buda küçük nesnelerin tanınmasını zorlaştırıyor. Buna çözüm olarak örneğin SSD(Single Shot MultiBox Detector) algoritması farklı katmanlardaki feature map’lerden küçük nesneleri belirliyor. YOLO ise farklı bir yöntem kullanmış. Örneğin boyutu 28x28x512 olan bir katmanı 14x14x2048 boyutuna getirip bunu 14x14x1024’lük çıktı katmanının arkasına ekliyor.

En son YOLOv3’te geliştirilmiş olan Darknet-53’ ün yapısı ise şu şekilde (<https://pjreddie.com/darknet/>) :



**Şekil 10** Darknet-53 Konvolüsyon işlemleri



YOLO’da hata fonksiyonlarını üç temel ana başlıkta inceleyebiliriz:

**1-)** Sınıflandırma kaybı: Tahmin edilen nesnenin ne kadar yanlış olduğu.

**2-)** Konum kaybı: Tahmin edilen kutunun ne kadar yanlış olduğu.

**3-)** Güven kaybı: Izgaranın içinde nesne olup olmadığının ne kadar yanlış olduğu.

Bunların hepsi genel kaybımızı etkileyen faktörler. Gelin üçünü kısaca inceleyelim.

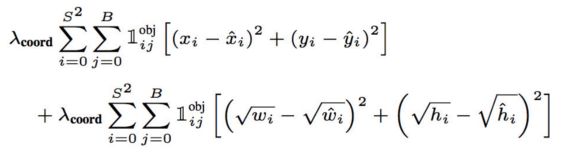
**Sınıflandırma Kaybı**

Eğer geçerli ızgarada nesne varsa, her sınıf için:



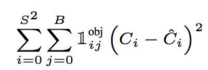
**Konum Kaybı**

Eğer geçerli ızgarada nesne varsa:

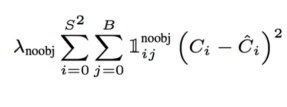


**Güven Kaybı**

Eğer nesne varsa:

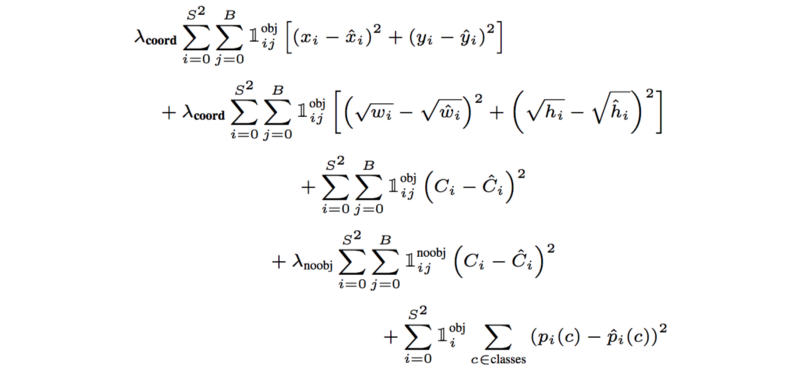


Eğer nesne yoksa:



Resmin ızgaralarında gezerken çoğu ızgara hiçbir nesne içermeyecektir. Yani arka plan olarak tespit ettiğimiz sınıflar çok fazla olacaktır. Bu durum sınıflarımız arasında bir dengesizliğe yol açıyor. Bunu azaltmak içinse eğer ızgarada nesne yoksa, yukarıdaki formülde de gördüğünüz gibi bu kaybı belirli bir sabitle (noobj ağırlığı) çarpıyoruz. (Genelde 0.5 olarak tanımlanır)

Bu üç hata fonksiyonun toplamı ise genel hata fonksiyonumuzu oluşturuyor:



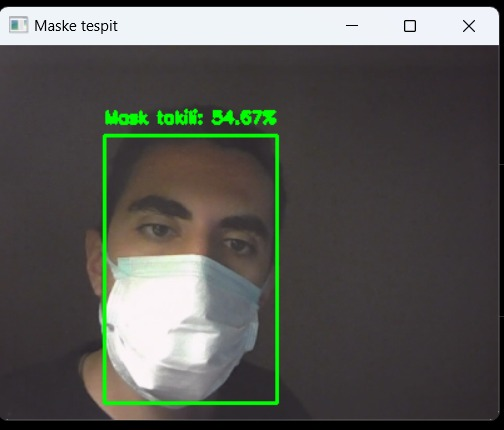
**5. Sonuçlar**

Bu çalışmada 2 sınıflı (maskeli ve maskesiz) maske kontrolünü gerçekleştiren bir yapay zekâ modeli geliştirilmiştir. Geliştirilen yapay zekâ modelleri Google Colab platformu üzerinde GPU’lu bilgisayarlar üzerinden eğitilmiştir.

Geliştirilen yazılım Windows 11 ortamında test edilmiştir Yazılım Python dilinde yazılmıştır (kullanılan sürüm: 3.7) ve pycharm ide’sinde geliştirilmiştir.

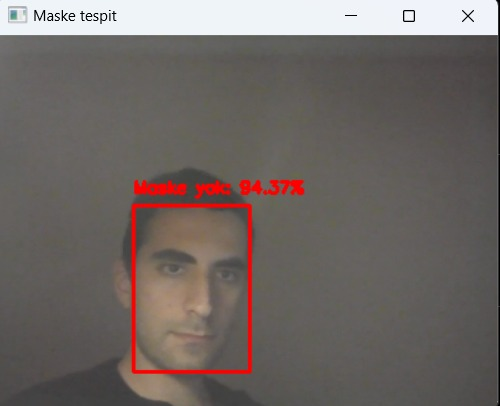
Gerçek zamanlı olarak maskeli, maskesiz ve maskesini yanlış takan kişilerin tespiti gerçekleştirilmiş sesli olarak kişilere geri bildirimler verilmiştir. Projenin ileriki versiyonları için gerekli altyapı oluşturulmuştur.

**Sonuçlar:**



**Şekil 11 –** Maske takıldığında programın verdiği geri bildirim

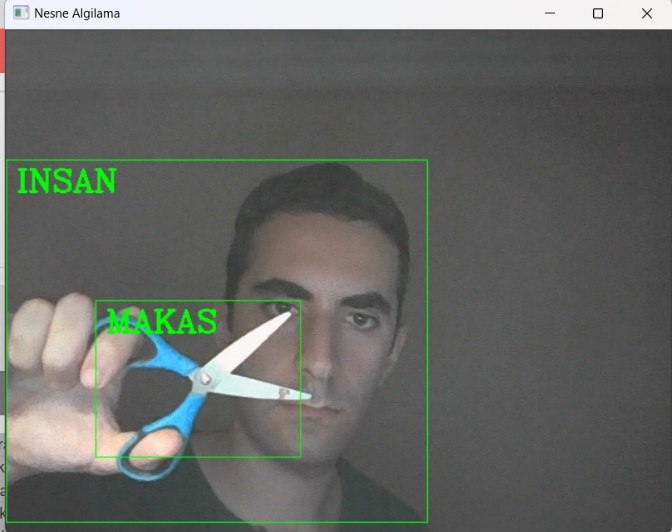
Yukarıda tespit edilen nesnenin doğruluk oranı ile birlikte tespit edilen alanın geri bildirimi vermiştir. Tespit edilen nesnenin maske olma ihtimali yolov7-tiny algoritmasına göre yüzde 54.67% dir.



**Şekil 12 –** Maske takılmadığında programın verdiği geri bildirim

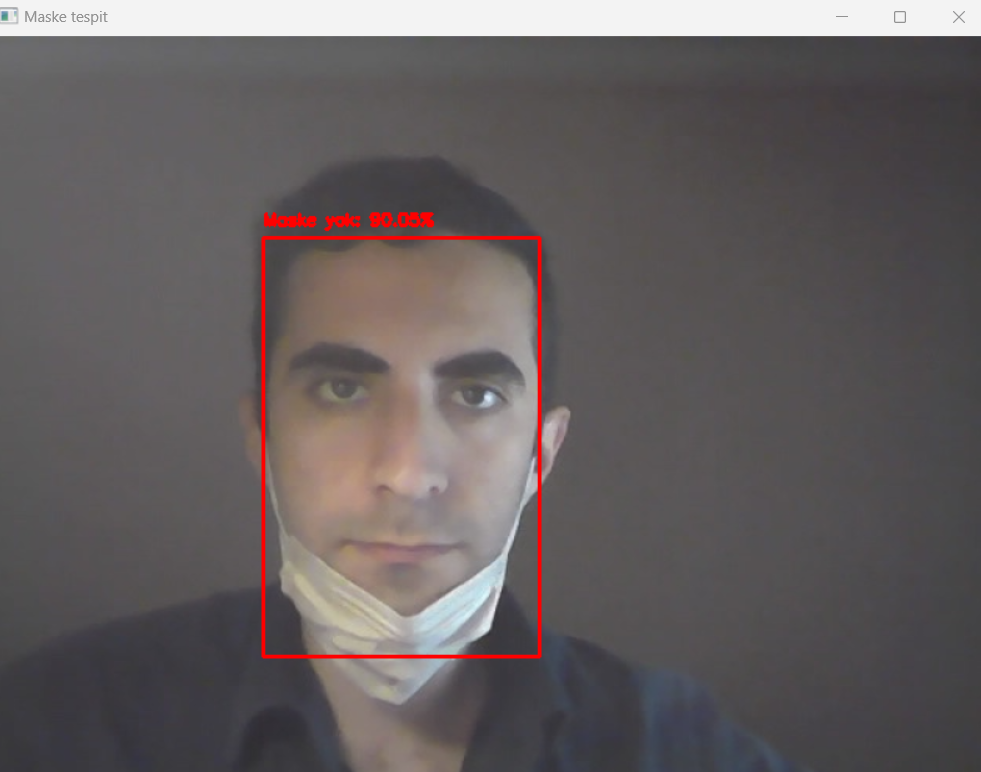
Yukarıda tespit edilen nesnenin doğruluk oranı ile birlikte tespit edilen alanın geri bildirimi vermiştir. Tespit edilen nesnenin maske olmama ihtimali yolov7-tiny algoritmasına göre yüzde 94.37% dir.

Farklı nesneleri tespit etme



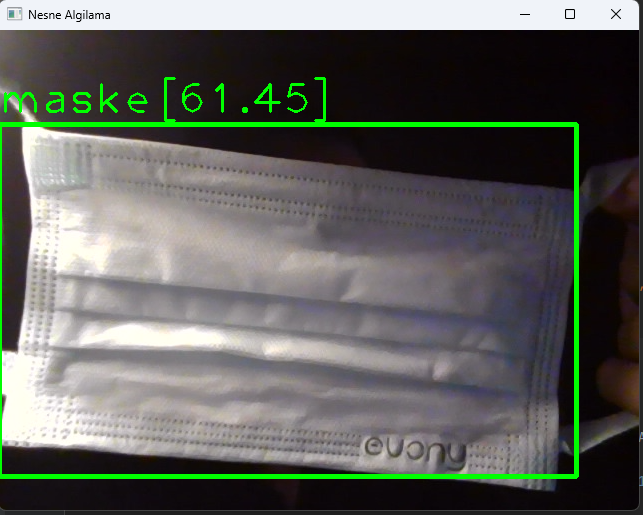
**Şekil 13 –** iki farklı nesnenin uygulama tarafından tespit edilmesi.

Yukarıdaki örnekte Yolov7-tiny algoritması ile iki farklı nesne (İnsan ve makas) eş zamanlı olarak doğruluk oranı olmadan tespit edilmiştir.



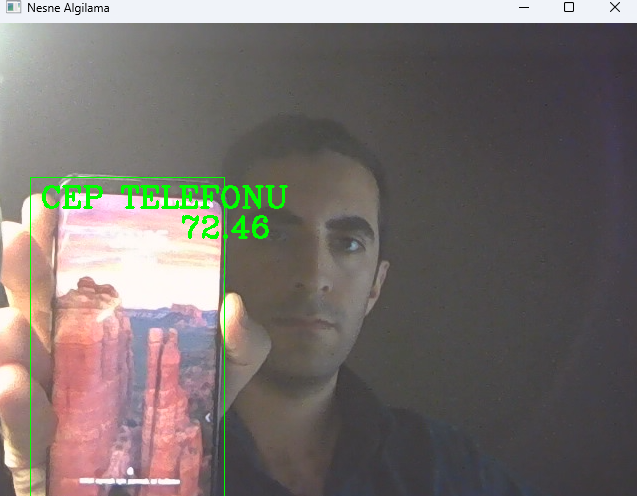
**Şekil 14 –** Maske yanlış takıldığında programın verdiği geri bildirim

Yukarıda tespit edilen nesnenin doğruluk oranı ile birlikte tespit edilen alanın geri bildirimi vermiştir. Maske yanlış takıldığında yolov7-tiny algoritmasının bunu tespit etme ihtimali yüzde 90.05% dır.



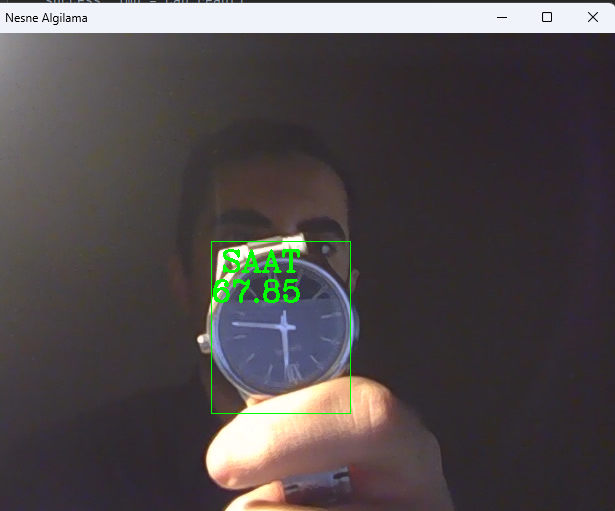
**Şekil 15 –** Uygulama tarafından sadece maskenin tespit edilmesi.

Yukarıda tespit edilen nesnenin doğruluk oranı ile birlikte tespit edilen alanın geri bildirimi vermiştir. Tespit edilen nesnenin sadece maske olma ihtimali yolov7-tiny algoritmasına göre yüzde 61.45% dir.



**Şekil 16 –** Uygulama tarafından cep telefonu nesnesinin tespiti

Yukarıda tespit edilen nesnenin doğruluk oranı ile birlikte geri tespit edilen alanın bildirimi vermiştir. Tespit edilen nesnenin sadece cep telefonu ihtimali yolov7-tiny algoritmasına göre yüzde 74.46% dır



**Şekil 17 –** Uygulama tarafından saat nesnesinin tespiti

Yukarıda tespit edilen nesnenin doğruluk oranı ile birlikte geri tespit edilen alanın bildirimi vermiştir. Tespit edilen nesnenin sadece saat olma ihtimali yolov7-tiny algoritmasına göre yüzde 67.85% dır

**KAYNAKÇA**

[2] <https://www.odakarge.com/yolo-nedir.html> Yolo nedir? (2021)

[3] <https://mesutpiskin.com/blog/opencv-nedir.html> OpenCv nedir? (2016)

[4] <https://devnot.com/2019/tensorflow-nedir-nasil-kullanilir/> TensorFlow Nedir? Nasıl Kullanılır? (2019)

[5] Mercaldo F., Santone A., “Transfer learning for mobile real-time face mask detection and localization,” Journal of the American Medical Informatics Association, vol. 28, no. 7, (2021).

[6] Kumar G. P., “Face Mask Detection with Raspberry Pi,” International Journal for Research in Applied Science and Engineering Technology, vol. 9, no. VI, (2021).

[7] Rani N. ve diğerleri, “Real-Time Face Mask Detection Using Raspberry Pi and Camera,” in Lecture Notes in Networks and Systems, 2022, vol. 300 LNNS.

[8] Mahurkar R. R., Gadge N. G., “Real-time Covid-19 Face Mask Detection with YOLOv4,” in Proceedings of the 2nd International Conference on Electronics and Sustainable Communication Systems, ICESC 2021, 2021.

[9] Oumina A. ve diğerleri, “Control the COVID-19 Pandemic: Face Mask Detection Using Transfer Learning,” in 2020

IEEE 2nd International Conference on Electronics, Control, Optimization and Computer Science, ICECOCS 2020, 2020.

[10] Loey M. ve diğerleri, “Fighting against COVID-19: A novel deep learning model based on YOLO-v2 with ResNet-50 for medical face mask detection,” Sustainable Cities and Society, vol. 65, (2021).

[11] Bhuiyan M. R. ve diğerleri, “A Deep Learning Based Assistive System to Classify COVID-19 Face Mask for Human Safety with YOLOv3,” in 2020 11th International Conference on Computing, Communication and Networking Technologies, ICCCNT 2020, 2020.

[12] Abbasi S. ve diğerleri, “A Face-Mask Detection Approach based on YOLO Applied for a New Collected Dataset,” in 26th International Computer Conference, Computer Society of Iran, CSICC 2021, 2021.

[13] Gedik O., Demirhan A., “Comparison of the effectiveness of deep learning methods for face mask detection,” Traitement du Signal, vol. 38, no. 4, (2021).

[14] Ieamsaard J. ve diğerleri, “Deep Learning-based Face Mask Detection Using YoloV5,” in Proceeding of the 2021 9th International Electrical Engineering Congress, iEECON 2021, 2021.

[15] Nagrath P. ve diğerleri, “SSDMNV2: A real time DNN-based face mask detection system using single shot multibox detector and MobileNetV2,” Sustainable Cities and Society, vol. 66, (2021).

[16] Mercaldo F., Santone A., “Transfer learning for mobile real-time face mask detection and localization,” Journal of the American Medical Informatics Association, vol. 28, no. 7, (2021).

[17] Kumar G. P., “Face Mask Detection with Raspberry Pi,” International Journal for Research in Applied Science and Engineering Technology, vol. 9, no. VI, (2021).

[18] Rani N. ve diğerleri, “Real-Time Face Mask Detection Using Raspberry Pi and Camera,” in Lecture Notes in Networks and Systems, 2022, vol. 300 LNNS.