GÜLÜMSEME PROJEM ☺

**Bilgisayar Görmesi (Computer Vision)?**

Bir insan görüntüleri ve hareketli nesneleri algılayıp yorumlayabilir. Aynı işlemleri bir makinenin yapabilmesi için bilgisayarlı görü çalışmalarına başlanmıştır. Bu çalışmalar sonucunda geliştirilmiş olan algoritmalar sayesinde bilgisayarlar tıpkı bir insan gibi dijital görüntülerden veya video görüntüleri üzerinden öğrenme, tanıma ve yorumlama yapabilmektedirler. Özetle bilgisayarların gördüklerini anlamalarını sağlayan alt bilim dalına “Bilgisayar Görmesi” denir.

**Görüntü Nedir (Image)?**

Bir imaj, herhangi bir şeyin görsel şekilde yeniden yansıtılmasıdır. Görüntü işleme açısından değerlendirildiğinde elektronik olarak oluşturulan, kopyalanan, işlenen ve saklanan bir resimdir. Görüntüler piksellerden oluşur. Piksel sayısı arttıkça görüntünün kalitesi de artar.

**Görüntü işleme,** dijital görüntülerin manipülasyonu ve analizi ile ilgilenen bir bilgisayar bilimi alanıdır. Görüntülerden yararlı bilgiler çıkarmak ve bunları bir şekilde dönüştürmek için bir dizi matematiksel ve hesaplama tekniğinin uygulanmasını içerir.

Görüntü işlemedeki bazı yaygın görevler şunları içerir:

* Boyutlarını veya çözünürlüğünü değiştirmek için bir görüntüyü yeniden boyutlandırma veya yeniden örnekleme
* Bir görüntünün parlaklığını, kontrastını veya renk dengesini ayarlama
* Bir görüntünün netliğini veya keskinliğini artırma
* Bir görüntüdeki gürültüyü veya kusurları giderme
* Bir görüntüdeki kenarlar, çizgiler veya şekiller gibi özellikleri veya desenleri algılama ve çıkarma
* Dosya boyutunu küçültmek için bir görüntüyü sıkıştırma
* Bir görüntüyü JPEG'den PNG'ye veya tersi gibi bir biçimden diğerine dönüştürme
* Görüntü işleme teknikleri genellikle tıbbi görüntüleme, uydu görüntüleri, güvenlik ve gözetleme ve bilgisayar görüşü gibi çok çeşitli uygulamalarda kullanılmaktadır.
* Görüntülerin kalitesini ve görünümünü iyileştirmek için fotoğrafçılık, baskı ve grafik tasarım gibi alanlarda da kullanılırlar.

**Video Nedir?**

Arka arkaya sıralanmış birçok imajın hareketli gibi algılanan bir görsel oluşturmasıdır. Videolar için FPS(Frame per second) değeri çok önemlidir. Aynı zamanda görüntünün iletimi sırasında sıkıştırılması da günümüz teknolojilerine çok büyük kolaylık sağlıyor. Örneğin;

1 Piksel = 8 bit \* 3(RGB) => 24 bit

1024×1024 piksel bir görüntü yaklaşık 24 Mbit yer kaplar.(1K\*1K\*24)

Bu görüntüyü sıkıştırmadan/kodlamadan iletiyor olsaydık saniyede 1 kare için(1 FPS ) 24Mbit bant genişliğine ihtiyacımız olurdu. Bu hareketli görüntüler için kabul edilemez bir FPS değeridir. Şimdi Görüntü ve Video hakkında bilgi edindiğimize göre nesne algılamanın algoritmik tarafına geçebiliriz.

**YOLO Algoritması**

Nesne Tespiti(Object Detection); fotoğraflardaki, videolardaki ve gerçek zamanlı görüntülerdeki nesneleri tespit etmeye odaklanan bilgisayarlı görü ve görüntü işleme ile ilgili bir bilgisayar teknolojisidir. Faster R-CNN, Single Shot Detector (SSD) ve YOLO(You Only Look Once) gibi birçok nesne tespiti algoritmaları günümüzde kullanılmaktadır. Son yıllarda nesne tespiti alanında revaçta olan YOLO (You Only Look Once) algoritmasını duymuşsunuzdur. YOLO’dan önce de şu anda da YOLO’dan iyi tahmin yapabilen algoritmalar vardı fakat temel sorun yavaş olmalarıydı.

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 resim düşünün (araba ortada) 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;

**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.

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?

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:

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.

import cv2

// Gerçek zamanlı işlemlerde rol oynamaktadır.

OpenCV kütüphanesi sayesinde web kameraları, video dosyaları veya diğer aygıt türleri

tarafından bir bilgisayara bağlanan görsel bilgilerin yakalanmasını, analiz edilmesini

ve değiştirilmesini destekleyen yüzlerce işlev içerir.

Sınıflandırıcıları yükleriz

face\_cascade = cv2.CascadeClassifier('haarcascade\_frontalface\_default.xml')

smile\_cascade = cv2.CascadeClassifier('haarcascade\_smile.xml')

cap = cv2.VideoCapture(0)

while True:

    fotoğrafları kare kare oku

    ret, frame = cap.read()

    gri renge çevir

    gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

    yüzleri arar

    faces = face\_cascade.detectMultiScale(gray, 1.3, 5)

    yüzün x,y kordinatı; genişliği ve yüksekliği dörtgenin ayarlanması

    for (x, y, w, h) in faces:

        cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)

        gülümsemeyi bulduktan sonra çıkarırız

        roi\_gray = gray[y:y+h, x:x+w]

        roi\_color = frame[y:y+h, x:x+w]

        çıkardıktan sonra gülümsemeyi algılarız

        smiles = smile\_cascade.detectMultiScale(roi\_gray, 1.8, 20)

        gülümsemeleri getirip etrafında dörtgen çizeriz

        for (sx, sy, sw, sh) in smiles:

            cv2.rectangle(roi\_color, (sx, sy), (sx+sw, sy+sh), (0, 255, 0), 2)

    görüntüyü listeleriz

    cv2.imshow('frame', frame)

    bitirmek için ‘q’ ya basalım

    if cv2.waitKey(1) & 0xFF == ord('q'):

        break

kamerayı bırakırız ve pencereleri kapatırız

cap.release()

cv2.destroyAllWindows()

haarcascade\_frontalface\_default.xmlve haarcascade\_smile.xml dosyalarını yükledikten sonra projemiz çalışacaktır. Yükleyeceğimiz hesap :

**https://github.com/opencv/opencv/tree/master/data/haarcascades**

<https://medium.com/@akdenizz7/yolov5-ile-nesne-tespiti-8aa370febfc0>

<https://medium.com/deep-learning-turkiye/yolo-algoritmas%C4%B1n%C4%B1-anlamak-290f2152808f>

<https://smartera.com.tr/gercek-zamanli-nesne-takibireal-time-object-detection-w-yolo-python/>

<https://suleymanogrekci.wordpress.com/python_dersleri-5-dosya-okuma-ve-yazma/>

https://www.youtube.com/watch?v=c73mmFGyNYU

<https://www.youtube.com/watch?v=STbx-2ERTAY>

https://www.youtube.com/watch?v=UClsx9Ltjk4

https://www.youtube.com/watch?v=ETNiRZtIeyg