

**TEKNOLOJİ FAKÜLTESİ**

**BİLGİSAYAR MÜHENDİSLİĞİ**

**YAPAY ÖĞRENMENİN TEMELLERİ**

**PROJE RAPORU**

**Semih ÖZENÇ – B200109008**

**Yavuz Selim ŞAHİN – B200109031**

**Yapay Zeka ile Nesne Tespiti – Object Detection**

1. **Projenin Tanımı**
2. **Kullanılan Algoritma**
3. **Uygulama Adımları**
4. **Projenin Tanımı**

Toplumumuzda bilinçsiz ilaç kullanımı yaygın olarak görülmektedir. Bir hastalık için yanlış ilaç kullanımı, gereğinden fazla ilaç alımı gibi durumlara sık rastlanmaktadır [1]. Yapmayı planladığımız bu projede kullanıcıların ilaçlar hakkında doğru bilgilere ulaşabilmesi planlanmakta ve gereksiz ilaç̧ kullanımının önüne geçmeyi hedeflemekteyiz [2].

Bu uygulamada toplumsal sağlık acısından yanlış ilaç kullanımının önüne geçmeye çalışacağız. Uygulamada resimden Yapay Zeka ile ilaç görüntüsü alınacak ve ilaç hakkında genel bilgiler verilip kullanım durumu açıklanacaktır.

Uygulamayı hazırlarken öncelikle kullanacağımız ilaçların veri seti elde edilecektir. Ardından veriler etiketlenecek ve YOLO formatında çıktısı alınacaktır. Uygulama YOLOv4 algoritması ile çalışacaktır. Veri setinin hazırlanması ve etiketlenme işlemlerinin bitmesinin ardından veriler train verileri ve test verileri olmak üzere bölünecek, sınıflar belirlenecek ve eğitime başlanacaktır. Eğitim sonunda kabul edilebilir bir ağırlık dosyasına ulaşıldıktan sonra eğitim sonlanacak, bu ağırlık dosyası alınacak ve projenin doğruluğu kontrol edilecektir. Gerekli kodlar yazıldıktan bu ağırlık dosyası ve algoritmanın config dosyasıyla uygulama tamamlanacaktır. Sonuç olarak ilacın resmini yolladığımız zaman bize hangi ilaç olduğunu belli bir doğruluk oranıyla gösterecektir.

Projemizde Kullandıklarımız

1. Her ilaç için 60 farklı resim üzere toplam 720 resim
2. Resimleri etiketlemek için kullandığımız web sitesi ‘https://www.makesense.ai/’
3. Eğitimi gerçekleştirdiğimiz ortam Google Colab Pro
4. Depolama yeri Google Drive
5. Eğitim sonucu oluşan modelin kullanım ortamı Anaconda Spyder
6. **Kullanılan Algoritma**

Yolo (You Only Look Once)

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. YOLO algoritmasının diğer algoritmalardan daha hızlı olmasının sebebi resmin tamamını tek seferde nöral bir ağdan geçiriyor olmasıdır.

YOLO algoritması görüntüler üzerinde tespit ettiği nesnelerin çevresini bounding box ile çevreler.

YOLO kendisine girdi olarak verilen görüntüyü NxN’lik ızgaralara böler. Bu ızgaralar 5×5,9×9,17×17… olabilir.

Her ızgara kendi içerisinde nesne olup olmadığını ve nesne var olduğunu düşünüyorsa merkez noktasının kendi alanında olup olmadığını düşünür.

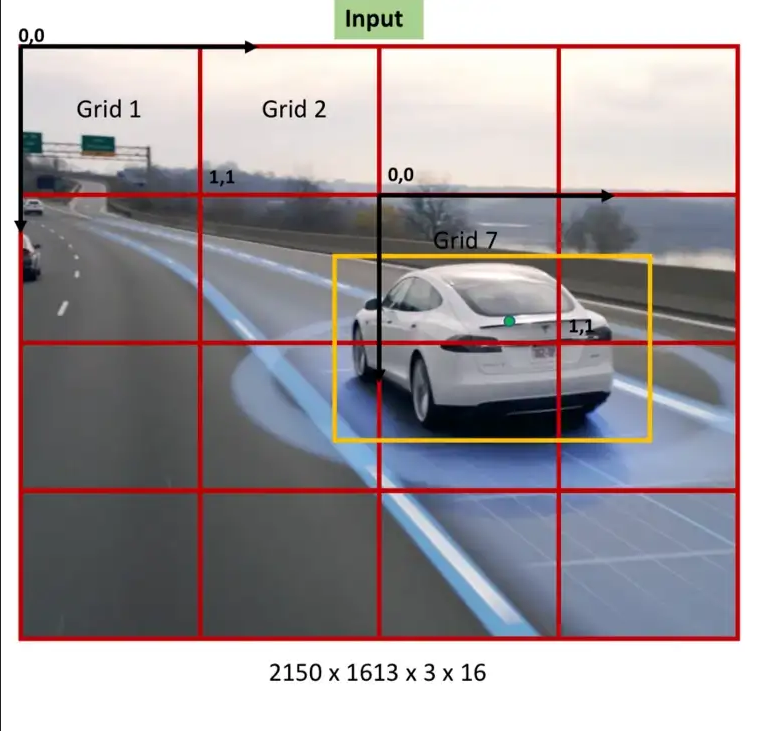
Nesnenin merkez noktasına sahip olduğuna karar veren ızgara o nesnenin sınıfını, yüksekliğini ve genişliğini bulup o nesnenin çevresine bounding box çizmelidir.

Bütün bounding boxların güven skoru vardır.

Birden fazla ızgara, nesnenin kendi içerisinde olduğunu düşünebilir. Bu durumda ekranda gereksiz bounding box’lar oluşur.

Birden çok bounding box olma durumunu engellemek için Non-Maximum Suppression algoritması kullanılır.

Kısaca Non-max Suppression algoritması görüntü üzerinde tespit edilen nesneler için çizilen bounding boxlardan güven değeri en yüksek olanı ekrana çizer.



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.

**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.

**Eğitim İşleminin Başlaması için gereken bazı dosyalar**

* YOLO algoritmasının çalıştırılabilmesi için Darknet bizden bazı dosyalar beklemektedir. Bunlardan bir tanesi de .cfg dosyasıdır.
* Config dosyası YOLO içerisindeki Sinir ağının başarısını, hızını vs. etkileyecek özellikleri bizden talep eder.
* Train.txt, test.txt dosyaları
* .data dosyası. Gereken dosyaların konumlarını barındırır.
* YOLO’nun bizden beklediği bir başka dosya ise .names dosyası. Names içerisinde sınıfların adlarını tutar.
* Bir eğitimin başarısını eğitim sonrası oluşan loss grafiğinden anlaşılabilir.
* Eğitim aşamasını tamamlayıp. kabul edilebilir bir loss değere sahip .weights dosyasını elde ettiğimizde duruyoruz. Artık modelimiz hazır.

1. **Uygulama Adımları**

**3.1) Veri Setinin elde edilmesi**

Yapacağımız proje için hazırlanmış bir veri seti olmadığı için verileri kendimiz elde ettik. Bunun için 12 farklı ilaca ait, ilacın tanesinde 60 resim olmak üzere toplam 720 farklı resim kullandık.

İlaç resimlerini elde ettikten sonra bu resimleri etiketleme işlemine geçtik. Makesense.ai sitesine resimlerimizi attıktan ilaçları isimlerine göre sınıflandırdık.

Örneğin Majezik adlık ilacın sınıfı => Majezik

Parol ilacı => Parol sınıfı

ibuCold ilacı => ibuCold sınıfı

720 tane ilaç için ayrı ayrı etiketleme işlemi yaptıktan sonra bu verileri Yolo Formatında kaydettik.

1.jpg(ibuCold) ilacına ait Yolo formatında dosya.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**3.2) Darknet ‘in elde edilmesi ve dosya düzenlemeleri**

Github’dan Darknet dosyalarını elde ettik. İçinde YoloV4 algoritmamız var.

Ardından eğitim işlemine başlayabilmemiz için bazı düzenlemeler yapmamız gerekiyor.

* 1. İlacimages klasörü = içinde 720 tane resim ve Yolo formatında etiketlediğimiz dosyalar bulunuyor.
  2. Labels klasörü = İçerisinde her resim için Yolo formatında çıktı bulunuyor
  3. İlaç.names = Etiketlediğimiz sınıflarla aynı isimde ve sırada 12 tane sınıf adı bulunuyor.
  4. ilac\_training = Eğitim verilerinin konumu bulunuyor. 1.jpg, 2.jpg. 3.jpg… 48.jpg (720 tane resmin %80 i burda bulunuyor.)
  5. ilac\_testing = Test verilerinin konumu bulunuyor. 48.jpg, 49.jpg, 50.jpg… 60.jpg (720 tane resmin %20 si burada bulunuyor.)
  6. ilac.data = test , train , images , data ve ağırlık dosyalarının kaydedileceği konumları barındıran dosya.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

İlac.data görünümü.



Eğitim işlemi için gerekli 6 dosyanın görünümü

İşlemlerimiz bitti bu 6 dosyayı bir klasöre atıp Github’dan elde ettiğimiz Darknet klasörüne atıyoruz.

**3.2) Config ve Makefile dosyalarının düzenlenmesi**

Darknet’in içindeki config ve Makefile dosyalarını eğitime başlamadan önce düzenlememiz gerekiyor.

Google Colab, GPU ve OPENCV kullanacağımız için Makefile içerisindeki

GPU = 0 GPU = 1

CUDNN=0 CUDNN = 1

OPENCV=0 bilgilerini OPENCV = 1 olarak değiştiriyoruz.

g

**3.3) Dosyaların Colab’a yüklenmesi ve Eğitim İşlemi**

Veri setimizden elde ettiğimiz 6 dosyayı, Github’dan elde ettiğimiz Darknet dosyamızın içine atmıştık. Darknet dosyamızı zipleyip Google Drive’a atıyoruz ve Google Colab ile Drive bağlantımızı sağlıyoruz.

Google Colab kullanmamızın nedeni Google’nin sağladığı Güçlü bir GPU olduğu için eğitim işlemi çok daha hızlı sürüyor.

**Colab’daki bazı komutlar.**

!unzip "/content/drive/MyDrive/Object\_Detection/darknet.zip" Attığımız zip dosyasını çıkarıyoruz

%cd /content/darknet Dosyamızın içine giriyoruz.

!find . -type f -print0 | xargs -0 dos2unix Dosyaları Unix formatına çeviriyoruz

!chmod +x /content/darknet Gerekli izinleri alma işlemi

!make Makefile dosyamızı çalıştırıyoruz.

!./darknet detector train IlacData/ilac.data yolov4.cfg yolov4.conv.137 -map -dont\_show Eğitime başlıyoruz. Hazırladığımız verilerdeki ilac.data , düzenlediğimiz yolov4.cfg dosyaları ve daha önce hazırlanmış ağırlık dosyası olan yolov4.conv.137 dosyalarını kullanarak eğitime başlıyoruz. Daha önceden eğitilmiş bir ağırlık dosyası kullanmamızın nedeni eğitime 0’dan başlarsak eğitimin çok uzun sürebileceğindendir.

**Eğitim İşlemi**

Eğitimin başlamasının ardından bize adım sayıları ve loss değerini veriyor. Loss değerinde modelin hata payını görüyoruz. Loss değerinin ne kadar düşükse bizim için o kadar önemli. Her iterasyonda loss değerinin düşmesi beklenir.

Ayrıca Her 100 adımda bir bize ağırlık dosyamızı backup klasörü içinde veriyor.

İlk Adımda Loss değerimiz 3121 iken 78.adımda 1155.63 olmuş bu azalma modelin eğitiminin devam ettiğini gösteriyor.

105 Adım = 331.231 Loss değeri

177 Adım = 11.30 Loss değeri

447 Adım = 4.33 Loss Değeri

1920 Adım = 0.52 Loss Değeri

2100 Adım = 0.6 Loss Değeri

Görüldüğü gibi modelin loss değeri belirli bir noktadan sonra değişmemeye başlıyor yani model daha fazla eğitilemiyor bu noktada eğitimi durdurup ağırlık dosyamızı alıyoruz.

**Modelin Denenmesi**

Elde ettiğimiz last.weights dosyasını , modeli eğittiğimiz yolov4.cfg dosyasını kullanarak bir resim üzerinde modelimizi deneyeceğiz.

Kodun içinde opencv, labellarımız , kutu renkleri , bounding box çizim paremetreleri , güven skoru ayarı gibi bilgiler bulunmakta.

Anaconda Spyder üzerindeki kodlarla ve elde ettiğimiz dosyaları kullanarak bir deneme yaptık ve şöyle bir sonuç elde ettik.

metin, beyaz tahta içeren bir resim

Açıklama otomatik olarak oluşturuldu

Görüldüğü gibi tespit işlemini başarıyla yapıyor ancak birden fazla bounding box oluşturuyor. Bunun önüne geçmek içinde sadece en yüksek skorlu bounding boxun oluşturulması için Non-Maximum Supression yöntemi kullandık ve en son olarak böyle bir sonuç aldık.

metin, beyaz tahta içeren bir resim

Açıklama otomatik olarak oluşturuldu