SÜRÜCÜ DAVRANIŞ ANALİZİ UYGULAMASI

...

Sürücü Davranış analizi uygulaması

Uygulamanın amacı, gündelik hayatımızda sıkça kullandığımız ulaşım araçlarını kullanan sürücülerin, sürüş esnasında kazaya sebep olup yolcuların ve sürücünün hayatlarını tehlikeye atacak üç davranışın analiz edilmesidir. Bu davranışlar, sürücünün telefonla konuşması, sigara içmesi ve iki elini aynı anda direksiyondan çekmesi durumlarıdır.

Uygulama, daha önceden belirlenen objelerin bulunduğu görüntülerin, CNN tabanlı Darknet mimarisinde bulunan katmanlarda gerçekleşen derin öğrenme olayları sonucunda bilgisayara öğretilmesi ve eğitim sonucunda elde edilen ağırlık dosyasının Python dilinde yazılan ve sürücünün davranışlarını analiz etmemizi sağlayan script dosyasının kullanılmasıyla gerçekleştirilmiştir.

Convolutional Neural Network (CNN) Nedir?

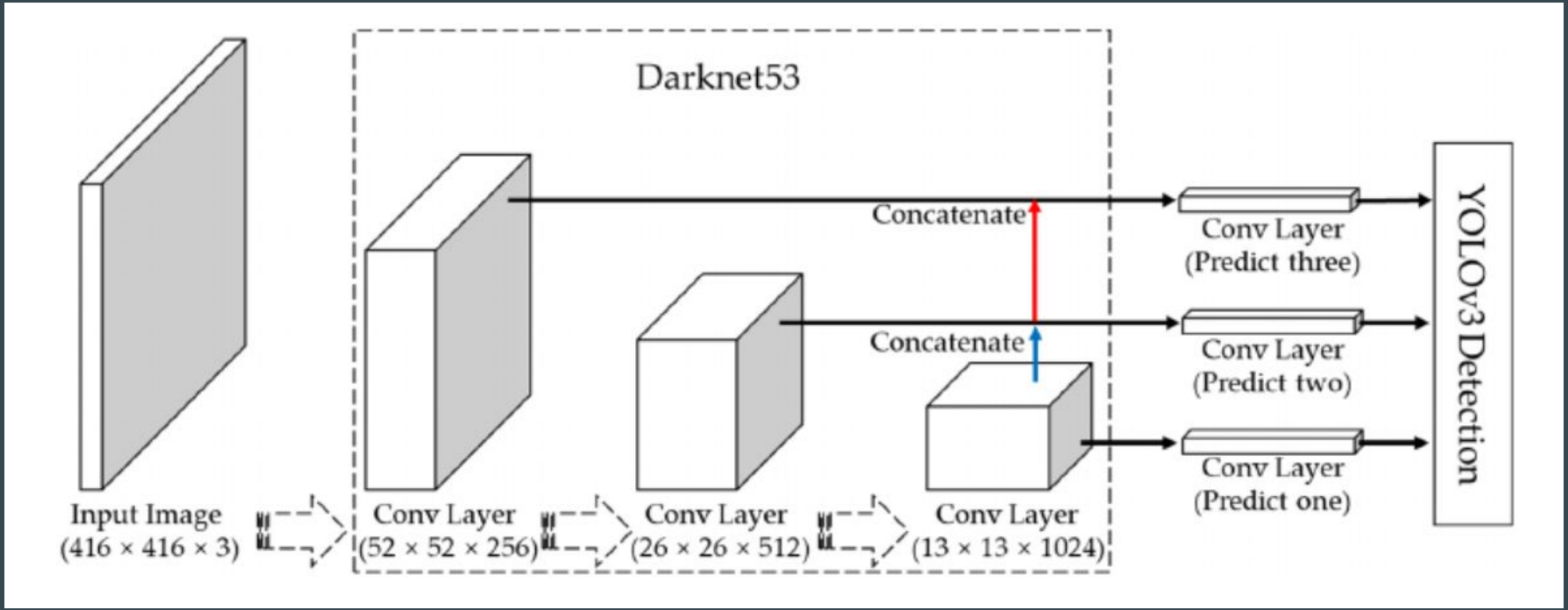
CNN genellikle görüntü işlemede kullanılan ve girdi olarak görselleri alan bir derin öğrenme algoritmasıdır. Sinir ağlarında Evrişimli sinir ağı (CNN), görüntü tanıma, görüntü sınıflandırmaları yapmak için ana kategorilerden biridir.

Farklı CNN mimarileri, farklı katmanlardan ve bu katmanlarda gerçekleşen farklı işlemlerden oluşmaktadır. Bu mimariler, input layer, Convolutional layers, pooling layer, fully connected layer, activation layer, padding, dropout, flattening gibi katmanlardan oluşabilir.

Darknet-53 mimarisi

YOLO algoritmaları serisi (Yolov1, Yolov2, Yolov3 vb.) regresyon problemleriyle görüntüde nesne tanımayı amaçlayan yöntemlerdir. Bu serilerden biri olan YOLOv3, küçük ve yoğun hedefler için yüksek algılama hızı ve yüksek algılama doğruluğuna sahip olarak geliştirilmiştir.

Darknet-53, YOLO'nun temel mimarisinin adıdır. Yapısı 106 katmandan oluşmaktadır, yalnızca evrişimli katmanları (convolutional layers) kullanır (örneğin pooling katmanı Darknet-53 mimarisinde bulunmaz). Aktivasyon fonksiyonu olarak Leakly ReLu, optimiser olarak SGD kullanır. 82, 94 ve 106. Katmanlarda gerçekleşen scale (ölçekleme) işlemleriyle birlikte gerçekleşen işlemler ile görüntüde tespitler yapmaya çalışır.



YOLOv3 ağ mimarisi

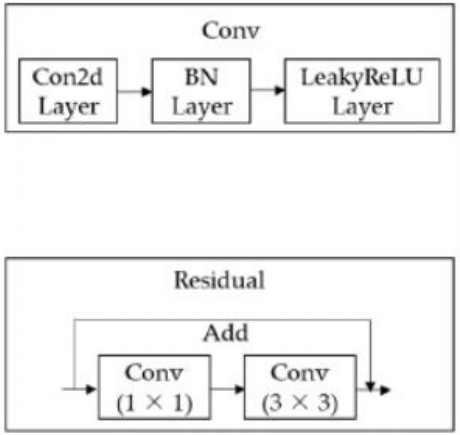
Darknet-53 mimarisi

Darknet53 evirişim ağı, özellik çıkarma işlevi görür. 416 x 416 boyutlarındaki görüntüler üzerinde işlemler yapar. Esas olarak 1x1 ve 3x3 boyutlarında bir dizi evrişim çekirdeği katmanından oluşur. Her evirişim katmanı Batch normalization ve Leaky ReLu katmanını takip eder. Resudial layer katmanıyla ağdaki gradyan kaybolması veya gradyan patlaması problemleri çözülür, böylece gradyanın yayılması daha kolay kontrol edebilir ve ağ eğitimi gerçekleştirebiliriz.

```
[convolutional]
batch_normalize=1
filters=32
size=3
stride=1
pad=1
activation=leaky
```

YOLOv3-voc modelinde bulunan bir convolutional layer örneği.

Layer	Filters size	Repeat	Output size
Image			416 × 416
Conv	32 3 × 3/1	1	416 × 416
Conv	64 3 × 3/2	1	208 × 208
Conv Conv Residual	32 1 × 1/1 64 3 × 3/1	Conv Conv Residual × 1	208 × 208 208 × 208 208 × 208
Conv	128 3 × 3/2	1	104 × 104
Conv Conv Residual	64 1 × 1/1 128 3 × 3/1	Conv Conv Residual × 2	104 × 104 104 × 104 104 × 104
Conv	256 3 × 3/2	1	52 × 52
Conv Conv Residual	128 1 × 1/1 256 3 × 3/1	Conv Conv Residual × 8	52 × 52 52 × 52 52 × 52
Conv	512 3 × 3/2	1	26 × 26
Conv Conv Residual	256 1 × 1/1 512 3 × 3/1	Conv Conv Residual × 8	26 × 26 26 × 26 26 × 26
Conv	1024 3 × 3/2	1	13 × 13
Conv Conv Residual	512 1 × 1/1 1024 3 × 3/1	Conv Conv Residual × 4	13 × 13 13 × 13 13 × 13



Darknet-53 mimarisi

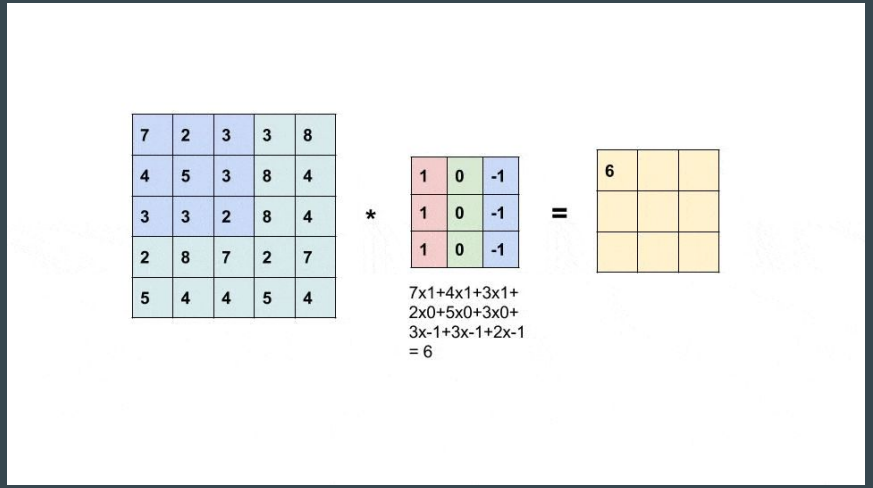
Convolutional Layer

Convolutional layer, (evrişim katmanı) CNN algoritmalarında görüntüyü ele alan ilk katmandır. Görseller içlerinde belirli değerler taşıyan piksellerden oluşan matrislerdir. Evrişim katmanında da (Convolution Layer) orijinal görsel boyutlarından daha küçük bir filtre görselin üzerinde gezer ve bu görsellerden belirli özellikleri yakalamaya çalışır.

Convolutional Layer

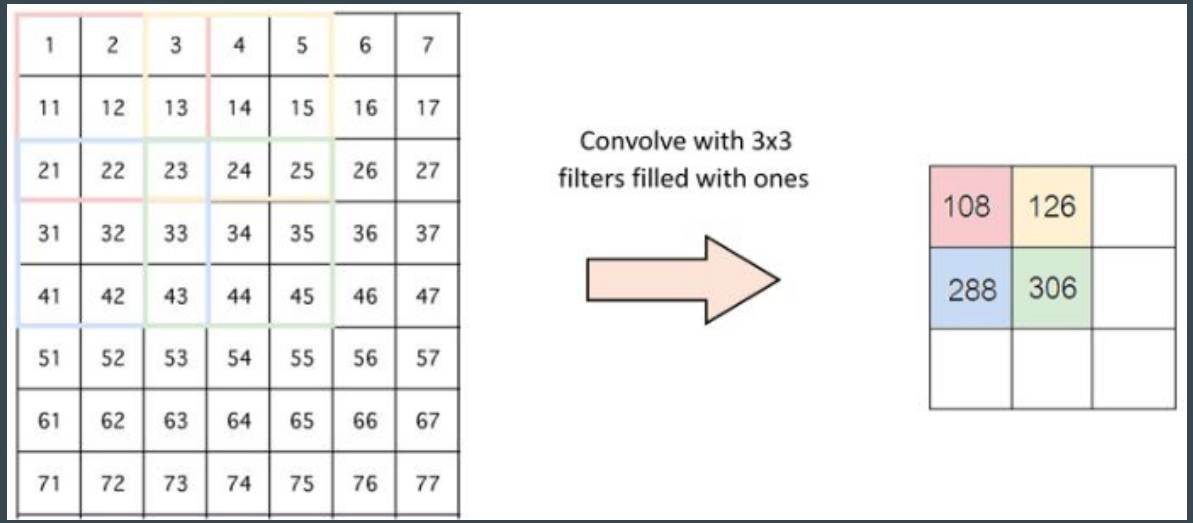
yanda görüldüğü üzere 3×3'lük bir filtre, 5×5'lik bir görsel üzerinde gezdiriliyor. Çıkan sonuçlar eşitliğin sağ tarafındaki yeni matrisimiz olan feature map üzerine yazılıyor.

CNN algoritmalarında öğrenilen parametreler bu filtrelerdeki değerlerdir. Model sürekli olarak bu değerleri günceller ve özellikleri daha da iyi tespit etmeye başlar



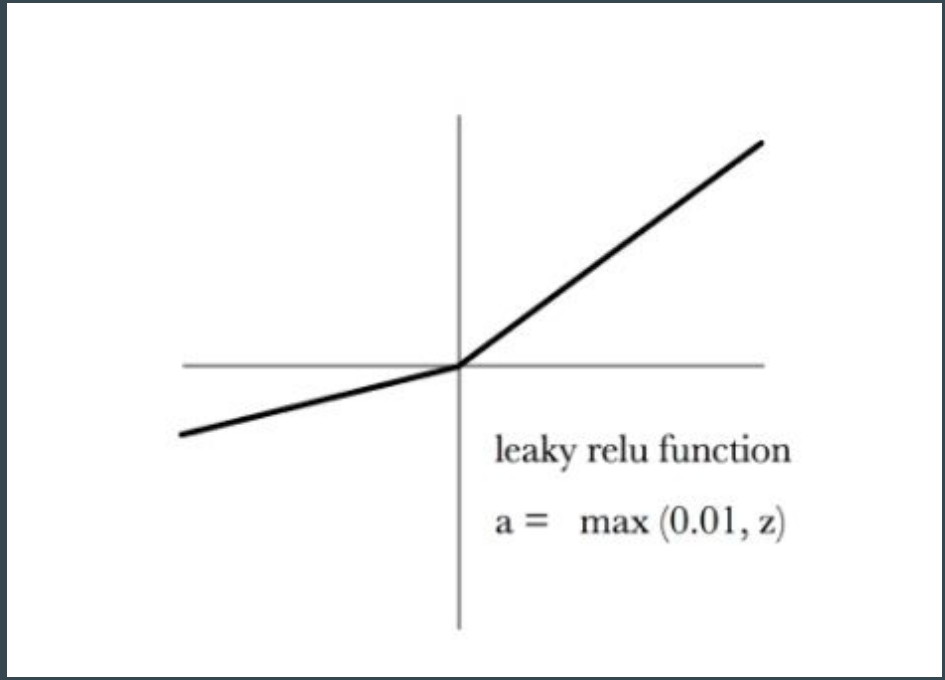
Stride

Stride, piksellerin giriş matrisi üzerindeki kayma sayısıdır. Adım 1 olduğunda, filtreleri bir seferde 1 piksele taşırız. Adım 2 olduğunda, filtreleri bir seferde 2 piksele taşırız ve bu böyle devam eder. yandaki şekil, evrişimin 2 adımda çalışacağını göstermektedir.



Leaky ReLU

Elimizde çok katmanlı bir yapay sinir ağı varsa ve bu sinir ağında büyük bir veriseti daha kesin güven aralığı değerlerini elde edecek şekilde eğitilmek istenirse, Leaky ReLu fonksiyonu bizim için kullanılabilir. CNN tabanlı YOLO algoritmasının kullanıldığı Darknet53 modelinde Leaky ReLu aktivasyon fonksiyonu kullanılmaktadır. YOLO algoritmasıyla oluşturulmuş mimarilerde eğitilen COCO veri setinin sonuçlarının incelenmesiyle, hız ve doğruluk grafiğine bakıldığı zaman bu aktivasyon fonksiyonunun başarısı gözlemlenebilir



SGD (Stochastic Gradient Descent) optimiser

Gradient descent, bir maliyet fonksiyonunu en aza indiren bir fonksiyonun parametrelerinin (katsayılarının) değerlerini bulmak için kullanılan bir optimizasyon algoritmasıdır. Her yinelemede (iteration)bir tahmin bekler. Bu durum çok büyük veri setlerinde kullanımı yavaşlattığı için tercih edilmez.

SGD optimiser yapısında da GD yürütülür fakat farklı olarak katsayıların güncellenmesi eğitimin sonunda gerçekleşir. Bu durum büyük veri setlerinde kullanımı hızlandırır.

SGD (Stochastic Gradient Descent) optimiser

1. Uygulaması kolay
2. Hesaplama açısından verimli
3. Bellek gereksinimi az
4. Gradyanların çapraz yeniden ölçeklendirmesi ile değişmez
5. Büyük veri setleri için uyumludur.

SGD (Stochastic Gradient Descent) optimizer

```
learning_rate=0.001
burn_in=1000
max_batches = 50200
policy=steps
steps=40000,45000
scales=.1,.1
```

YOLOv3-VOC modelinde SGD optimizer kullanımı

YOLO algoritması ve çalışma mantığı

...

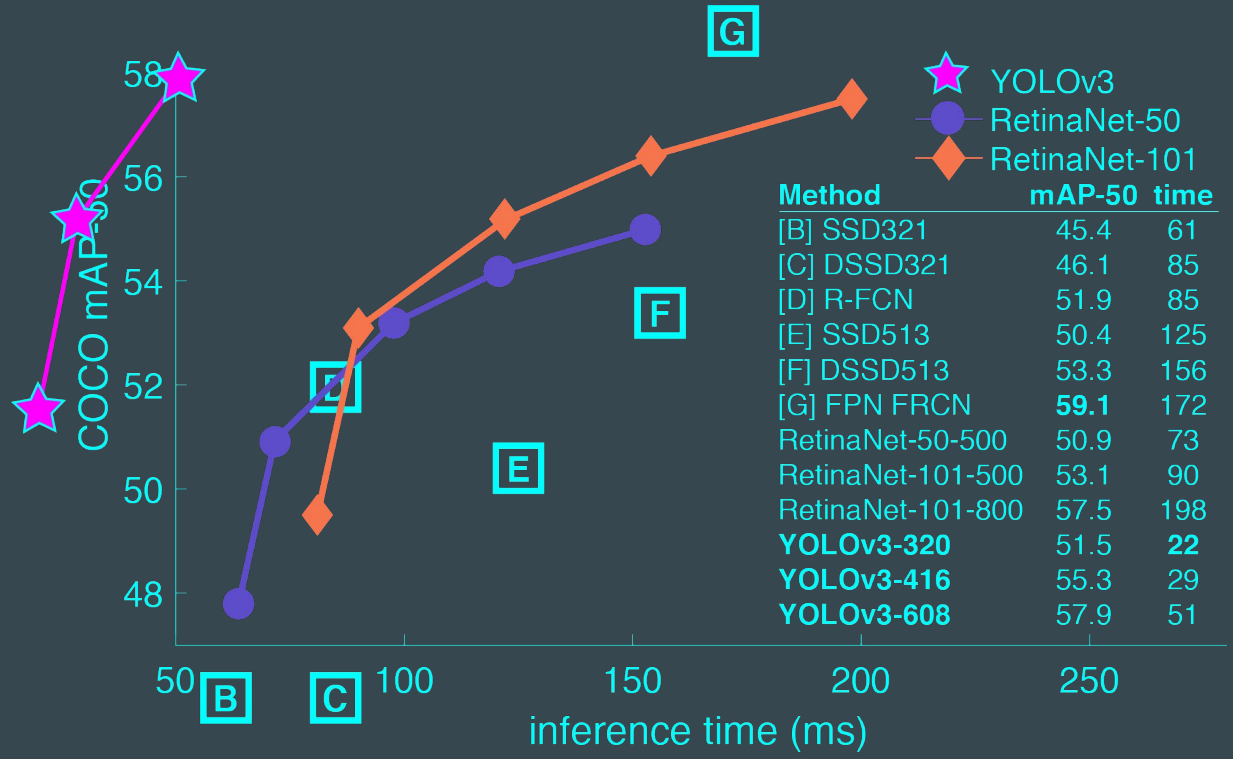
YOLO algoritması nedir?

YOLO (you look only once) en etkili gerçek zamanlı nesne tespit algoritmalarından biridir.

YOLO görüntüye tek bir sinir ağı uygular. Bu ağ görüntüyü bölgelere ayırır ve her bölge için sınırlayıcı kutuları ve olasılıkları tahmin eder.

YOLO algoritmasını diğer algoritmalardan ayıran en önemli özelliği rakiplerine göre hızlı olması ve yine rakiplerine göre gerçek zamanlı nesne tespitinde kesinlik değerli (mAP) değerinin çok yüksek olmasıdır (anlık olarak tespit ettiği objelerin doğruluğu).

Yolo algoritması ve rakiplerinin karşılaştırılması



Method	mAP-50	time
[B] SSD321	45.4	61
[C] DSSD321	46.1	85
[D] R-FCN	51.9	85
[E] SSD513	50.4	125
[F] DSSD513	53.3	156
[G] FPN FRCN	**59.1**	172
RetinaNet-50-500	50.9	73
RetinaNet-101-500	53.1	90
RetinaNet-101-800	57.5	198
YOLOv3-320	51.5	**22**
YOLOv3-416	55.3	29
YOLOv3-608	57.9	51

Yolo algoritması ve rakiplerinin karşılaştırılması

Bir önceki slaytta 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. .

YOLO algoritması nasıl çalışır ?

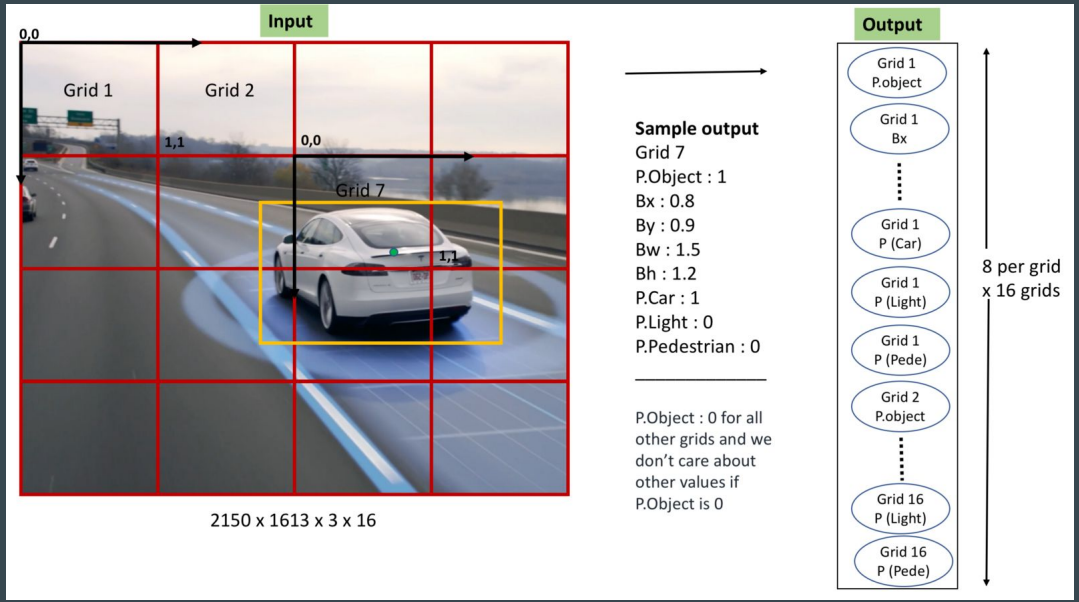
Yolo algoritmasının hızlı ve büyük doğruluk oranında tespit yapabilmesinin sebebi resmi tek bir seferde nöral ağdan geçirerek resimdeki tüm nesnelerin sınıfını ve koordinatlarını tahmin edebiliyor olmasıdır.

Yani bu tahmin işleminin temeli, nesne tespitini tek bir regresyon problemi olarak ele almalarında yatıyor.

Bu işlemleri bir örnek üzerinde inceleyelim,

YOLO algoritması nasıl çalışır ?

İlk olarak girdi resmini SxS'lik ızgaralara bölüyor. Bu ızgaraların herbirinde, tespit edilmesi istenen objenin orta noktası bulunmaya çalışılır.



YOLO algoritması nasıl çalışır ?

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.

Bir önceki 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, Bx, By, Bw, Bh, Bağlı Sınıf Olasılığı gibi değerli elde ederiz. Bu değerler algoritmanın nesne tespiti için önemli değerlerdir.

YOLO algoritması nasıl çalışır ?

1. Güven skoru : Bu skor modelin geçerli ızgara içinde nesne bulunup bulmadığını 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.
2. Bx: Nesnenin orta noktasının x koordinatı
3. By: Nesnenin orta noktasının y koordinatı
4. Bw: Nesnenin genişliği
5. Bh: Nesnenin yüksekliği
6. Bağlı Sınıf Olasılığı: Modelimizde kaç farklı sınıf varsa o kadar sayıda tahmin değeri.

YOLO algoritması nasıl çalışır ?

Güven skoru şu şekilde hesaplanır:

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?)

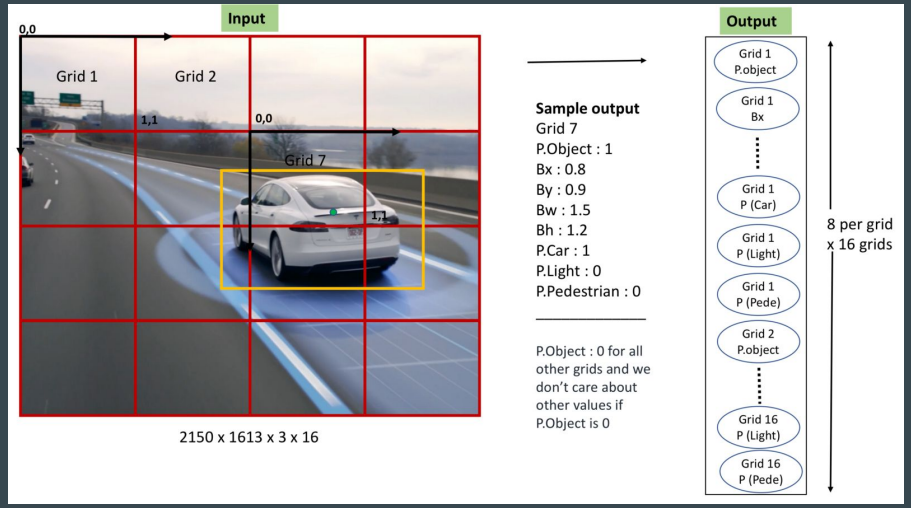
IoU = Ground truth ile tahmin edilmiş kutu arasındaki değerdir.

YOLO algoritması nasıl çalışır ?

Yandaki resimde Grid 7' ye baktığımızda eğer araba olduğundan kesin olarak eminsek:

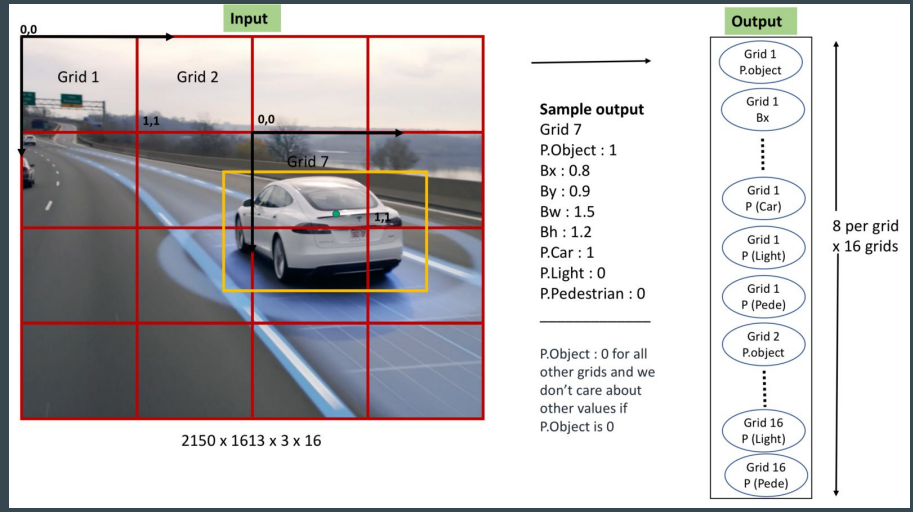
Araba: 1, Yaya: 0 olacaktır.

(olasılıkta, doğruluk değeri 0 - 1 arasında değer alır. 1 ise kesinlikle doğrudur, 0 ise kesinlikle yanlıştır.)



YOLO algoritması nasıl çalışır ?

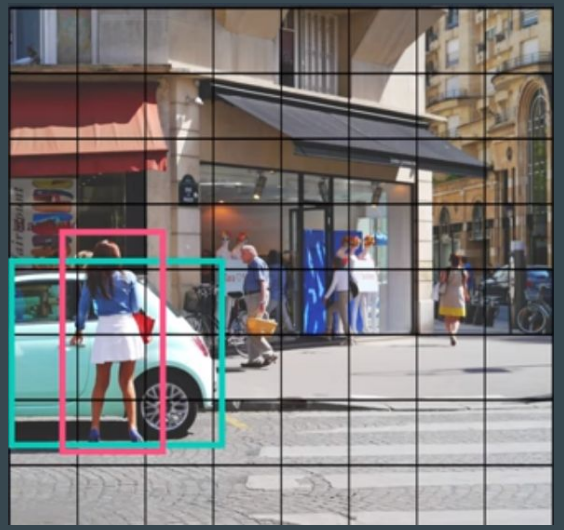
Yandaki çıktı vektörüne göre her bir ızgara sadece 1 tane nesne tanımlayabiliyor. Örneğin sadece 3x3'lük bir ızgara kullansaydık 9 tane nesne tahmini yapabilirdik. Peki bir ızgarada birden fazla nesne varsa Hatta bir ızgarada 2 farklı nesnenin orta noktası bulunursa bu durumu çözmek için "Anchor Box" yapıları kullanılır.



Anchor Box nedir?

Anchor Box yapılarının amacı, merkezi aynı hücrede bulunan farklı boyutlardaki birden fazla nesneyi tespit etmektir.

Sağdaki görselde, görselde örtüşen bir kişi ve arabamız olduğunu görüyoruz. Araba ve yayanın merkezleri aynı ızgara hücresine düştüğünden, gerekli bağlantı kutularını tanımlayarak (anchor box) daha uzun bir ızgara hücre vektörü oluşturulup her ızgara hücresiyle birden çok sınıfı ilişkilendirebiliriz.



Anchor Box nedir?

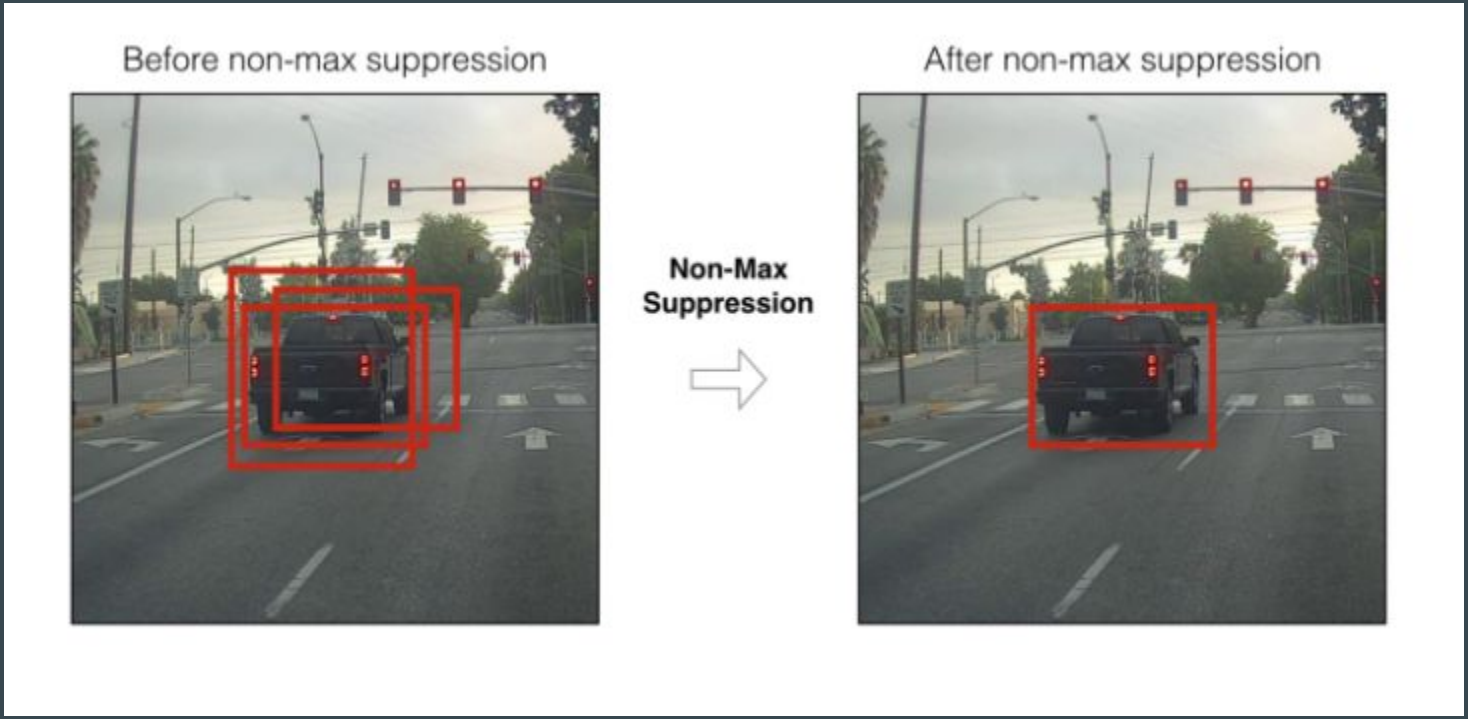
1. Görüntü ilk olarak ızgaralara bölünür ve ağ, her ızgara hücresi için bir tane çıktı vektörü üretir.

2. Bu vektörler bize bir hücrenin içinde bir nesne varsa, nesnenin hangi sınıfa ait olduğunu ve nesnenin sınırlayıcı kutularını gösterir.

3. İki bağlantı kutusu kullandığımızdan, her ızgara hücresi için iki tahmini bağlantı kutusu alacağız. Aslında tahmin edilen bağlantı kutularının çoğu çok düşük bir PC (nesnenin içinde bulunma olasılığı) değerine sahip olacaktır.

4. Bu çıkış vektörlerini ürettikten sonra olası olmayan sınırlı kutulardan kurtulmak için Non-Maximal Suppression kullanıyoruz. Her sınıf için, Non-Maximal Suppression(maksimal olmayan tüm sınırlayıcı) ,belirli eşiklerden daha düşük bir PC değeri olan sınırlayıcı kutulardan kurtulur.

Non Maximal Suppression



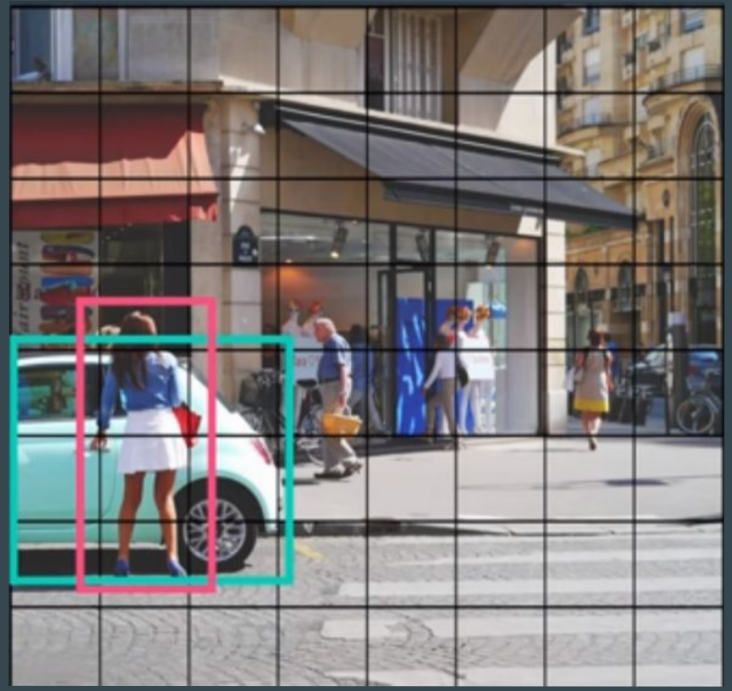
Anchor Box nedir?

En Son olarak, en yüksek nesnenin içinde bulunma olasılığı değerine sahip sınırlayıcı kutuları seçer ve buna çok benzeyen sınırlayıcı kutuları kaldırır.

Her sınıf için maksimal olmayan tüm sınırlayıcı kutular kaldırılıncaya kadar bunu tekrarlayacaktır.

Sonuç sağdaki resimdeki gibi görünecek, mavi bölgedeki arabayı ve kırmızı bölgedeki insanı etkili bir şekilde tespit ettiğini görebiliriz.

Anchor Box yapıları, Darknet-53 mimarisinde 82, 94, 106. Katmanlarında devreye girer.



Anchor Boxların YOLOv3-VOC modelinde gösterilmesi

```
[yolo]
mask = 6,7,8
anchors = 10,13,  16,30,  33,23,  30,61,  62,45,  59,119,  116,90,  156,198,  373,326
classes=5
num=9
jitter=.3
ignore_thresh = .5
truth_thresh = 1
random=1
```

82. KATMAN

```
[yolo]
mask = 3,4,5
anchors = 10,13,  16,30,  33,23,  30,61,  62,45,  59,119,  116,90,
classes= 5
num=9
jitter=.3
ignore_thresh = .5
truth_thresh = 1
random=1
```

94. KATMAN

```
[yolo]
mask = 0,1,2
anchors = 10,13,  16,30,  33,23,  30,61,  62,45,  59,119,  116,90,  156,198,  373,326
classes=5
num=9
jitter=.3
ignore_thresh = .5
truth_thresh = 1
random=1
```

106. KATMAN

YOLO algoritmasını nasıl kullandık ?

...

Sürücü Davranış analizi uygulamasında kullanılan teknolojiler

- **CUDA**
  Eğitimin GPU üzerinde gerçekleştirilmesi için kullanıldı
- **YOLOv3**
  Gerçek zamanlı nesne tespiti için kullanıyoruz
- **Kendi datasetimizi oluşturduk ve kendi eğitimimizi yaptık**
  1700 resim ile 5 farklı nesne için eğitim yaptık.
- **OpenCV**
  Web kamerasıyla görüntü işleme ve görüntü yakalamak için kullanıldı
- **PyGame**
  Ses dosyalarını çalıştırmak için kullanıldı
- **Camera**

CUDA Nedir

CUDA, GPU için NVIDIA'nın sunduğu C programlama dili üzerinde eklenti olarak kullanıma sunulan bir mimari ve teknolojidir. PathScale tabanlı bir C derleyicisi ve C ile yazılmış algoritmaların GPU üzerinde çalışmasını sağlayan geliştirme araçları kümesidir.

Cuda ile eğitimimiz yaklaşık 12 saat sürdü. Cuda olmadan 10 katına kadar çıkabilirdi.

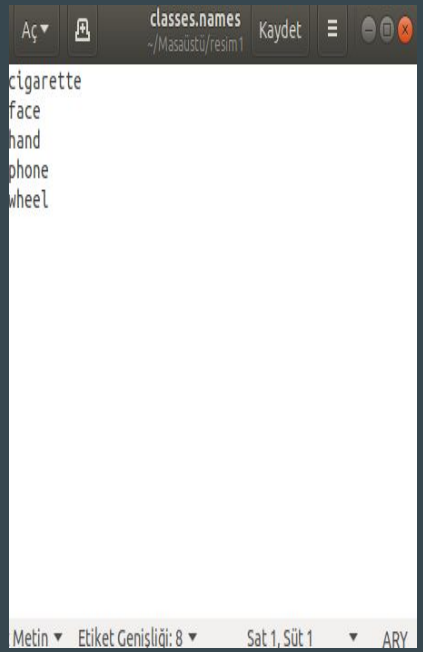
YoloV3 kullanarak eğitim için YOLO bizden 4 adet dosya istiyor

1. Train.txt > Bizden Eğitim setindeki eğitim resimlerinin dosya yollarını belirten her resmin satır satır yolunun verildiği txt dosyasıdır.
2. Test.txt > Bizden Eğitim setindeki test resimlerinin dosya yollarını belirten her resmin satır satır yolunun verildiği txt dosyasıdır.
3. Classes.names > Tespit edilecek nesnelerin isimlerinin satır satır yazıldığı dosya türü
4. Custom.data > Formatı aşağıda belirttildiği gibidir:
   classes = Class sayımız (Bizim class sayımız 5 idi)
   train = Train.txt dosyasının dosya yolunu veriyoruz
   valid = Test.txt dosyasının dosya yolunu veriyoruz
   names = Classes.names dosyasının dosya yolunu veriyoruz
   backup = Eğitimde weights dosyalarını nereye kaydetmek istiyorsak dosya yolunu veriyoruz

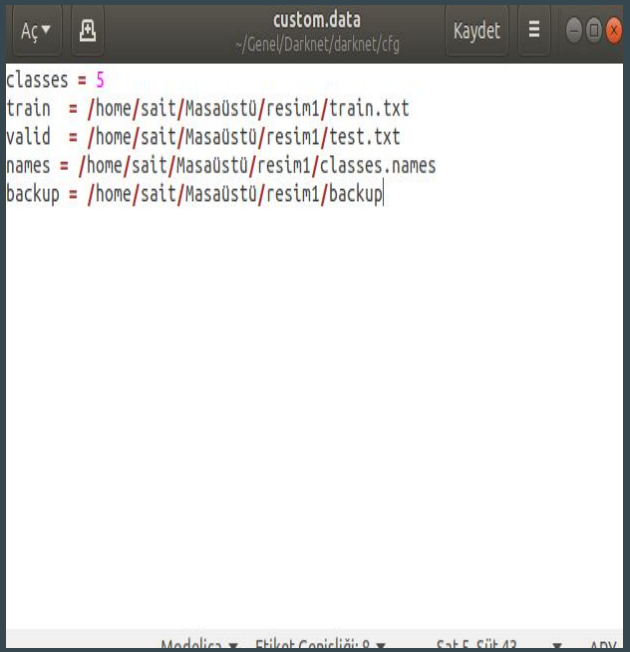
Örnek resimler

Train.txt

test.txt



classes.names



custom.data

Resimlerde ki ayarlamaları yaptıktan sonra eğitimi başlatıyoruz:

Önceden eğitilmiş ağırlık dosyasını indirmemiz gerekecek

```
wget https://pjreddie.com/media/files/yolov3.weights
```

Sonrasında ise hazırladığımız dosyaların yollarını Yolo'ya veriyoruz

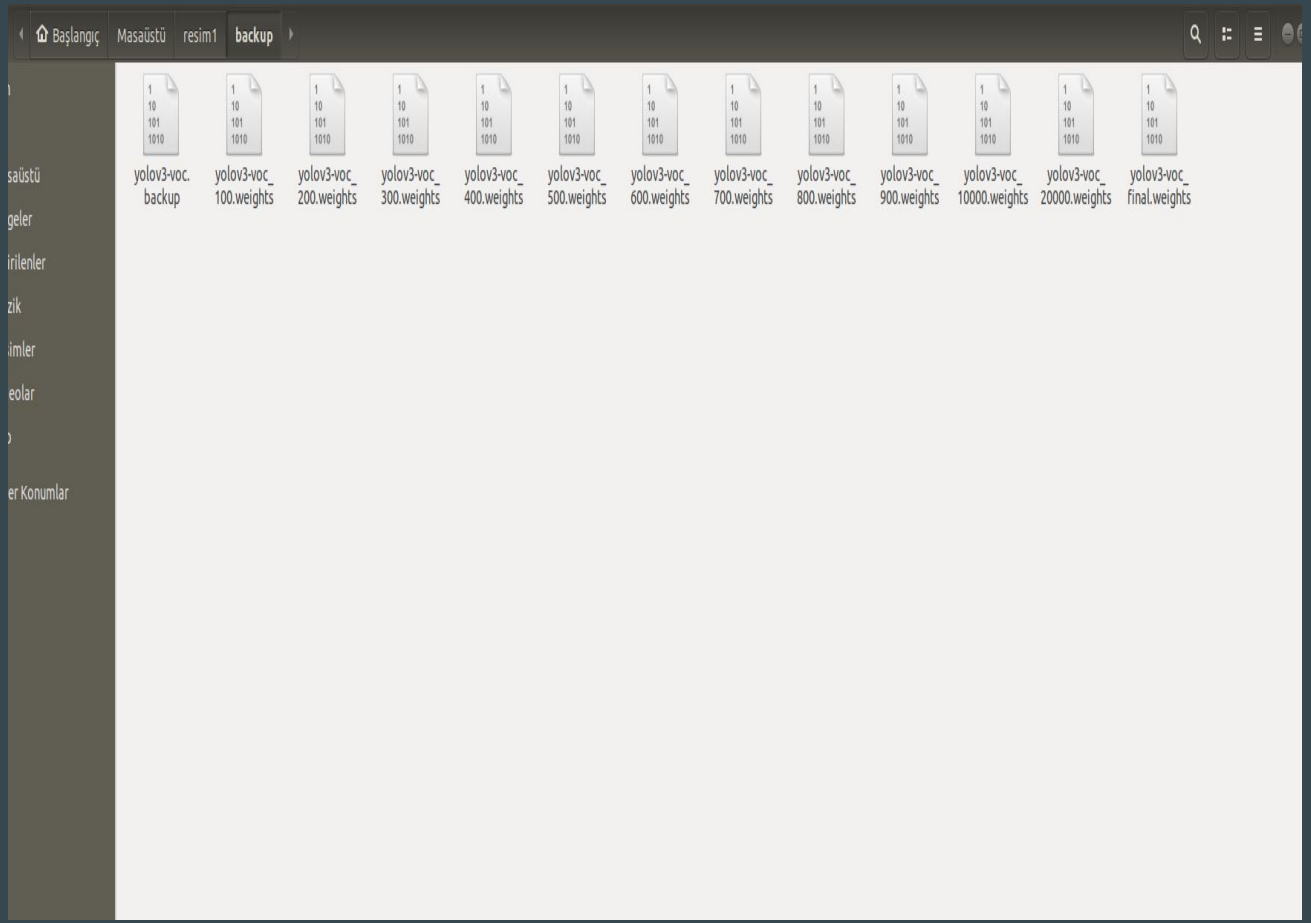
```
./darknet detector train cfg/custom.data cfg/yolov3-voc.cfg darknet53.conv.74
```

Sonrasında ise Yolo bize şöyle bir çıktı verecek...

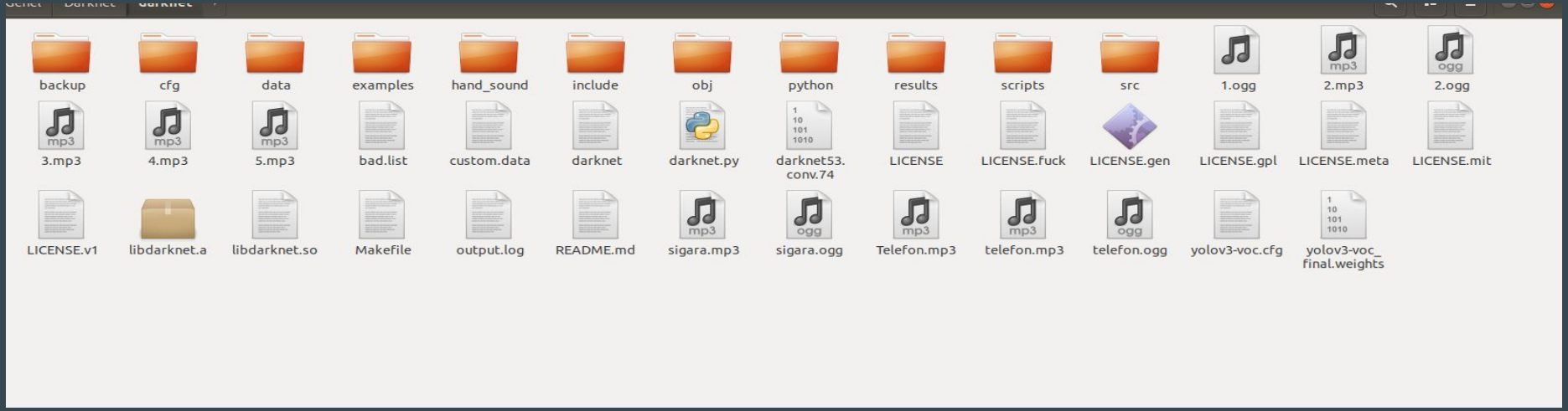
```
2: 1829.618896, 1829.447510 avg, 0.000000 rate, 2.371014 seconds, 32 images
Loaded: 0.000036 seconds
Region 82 Avg IOU: 0.207410, Class: 0.828103, Obj: 0.550666, No Obj: 0.546425, .5R: 0.000000, .75R: 0.000000,  count: 2
Region 94 Avg IOU: -nan, Class: -nan, Obj: -nan, No Obj: 0.514322, .5R: -nan, .75R: -nan,  count: 0
Region 106 Avg IOU: -nan, Class: -nan, Obj: -nan, No Obj: 0.459193, .5R: -nan, .75R: -nan,  count: 0
Region 82 Avg IOU: 0.168502, Class: 0.469798, Obj: 0.365848, No Obj: 0.546912, .5R: 0.000000, .75R: 0.000000,  count: 2
Region 94 Avg IOU: 0.113319, Class: 0.550174, Obj: 0.507061, No Obj: 0.514963, .5R: 0.000000, .75R: 0.000000,  count: 3
Region 106 Avg IOU: 0.002968, Class: 0.448119, Obj: 0.974438, No Obj: 0.459792, .5R: 0.000000, .75R: 0.000000,  count: 1
Region 82 Avg IOU: -nan, Class: -nan, Obj: -nan, No Obj: 0.547821, .5R: -nan, .75R: -nan,  count: 0
Region 94 Avg IOU: 0.064193, Class: 0.288444, Obj: 0.157448, No Obj: 0.514789, .5R: 0.000000, .75R: 0.000000,  count: 1
Region 106 Avg IOU: -nan, Class: -nan, Obj: -nan, No Obj: 0.464118, .5R: -nan, .75R: -nan,  count: 0
Region 82 Avg IOU: 0.125972, Class: 0.653920, Obj: 0.260737, No Obj: 0.545247, .5R: 0.000000, .75R: 0.000000,  count: 2
Region 94 Avg IOU: 0.326054, Class: 0.161348, Obj: 0.419836, No Obj: 0.513872, .5R: 0.500000, .75R: 0.000000,  count: 2
Region 106 Avg IOU: -nan, Class: -nan, Obj: -nan, No Obj: 0.458489, .5R: -nan, .75R: -nan,  count: 0
Region 82 Avg IOU: 0.149062, Class: 0.749391, Obj: 0.392735, No Obj: 0.545576, .5R: 0.000000, .75R: 0.000000,  count: 2
Region 94 Avg IOU: 0.152293, Class: 0.721992, Obj: 0.297125, No Obj: 0.512278, .5R: 0.000000, .75R: 0.000000,  count: 3
Region 106 Avg IOU: 0.216486, Class: 0.466384, Obj: 0.730098, No Obj: 0.457367, .5R: 0.000000, .75R: 0.000000,  count: 1
Region 82 Avg IOU: -nan, Class: -nan, Obj: -nan, No Obj: 0.547191, .5R: -nan, .75R: -nan,  count: 0
Region 94 Avg IOU: 0.442323, Class: 0.223433, Obj: 0.083663, No Obj: 0.514478, .5R: 0.000000, .75R: 0.000000,  count: 1
Region 106 Avg IOU: -nan, Class: -nan, Obj: -nan, No Obj: 0.466667, .5R: -nan, .75R: -nan,  count: 0
Region 82 Avg IOU: 0.273501, Class: 0.363910, Obj: 0.291987, No Obj: 0.545000, .5R: 0.000000, .75R: 0.000000,  count: 2
Region 94 Avg IOU: 0.428641, Class: 0.839665, Obj: 0.137141, No Obj: 0.513618, .5R: 0.000000, .75R: 0.000000,  count: 1
Region 106 Avg IOU: 0.000349, Class: 0.857632, Obj: 0.042048, No Obj: 0.461277, .5R: 0.000000, .75R: 0.000000,  count: 1
Region 82 Avg IOU: 0.160371, Class: 0.709659, Obj: 0.063509, No Obj: 0.549210, .5R: 0.000000, .75R: 0.000000,  count: 1
Region 94 Avg IOU: 0.107595, Class: 0.971558, Obj: 0.988802, No Obj: 0.515492, .5R: 0.000000, .75R: 0.000000,  count: 1
Region 106 Avg IOU: -nan, Class: -nan, Obj: -nan, No Obj: 0.460164, .5R: -nan, .75R: -nan,  count: 0
Region 82 Avg IOU: -nan, Class: -nan, Obj: -nan, No Obj: 0.548789, .5R: -nan, .75R: -nan,  count: 0
Region 94 Avg IOU: 0.174455, Class: 0.220483, Obj: 0.600401, No Obj: 0.514425, .5R: 0.000000, .75R: 0.000000,  count: 1
Region 106 Avg IOU: -nan, Class: -nan, Obj: -nan, No Obj: 0.466401, .5R: -nan, .75R: -nan,  count: 0
Region 82 Avg IOU: 0.229878, Class: 0.745300, Obj: 0.733990, No Obj: 0.546974, .5R: 0.000000, .75R: 0.000000,  count: 1
Region 94 Avg IOU: 0.569546, Class: 0.842108, Obj: 0.170090, No Obj: 0.512763, .5R: 1.000000, .75R: 0.000000,  count: 1
Region 106 Avg IOU: -nan, Class: -nan, Obj: -nan, No Obj: 0.456666, .5R: -nan, .75R: -nan,  count: 0
Region 82 Avg IOU: 0.048929, Class: 0.404007, Obj: 0.904662, No Obj: 0.546760, .5R: 0.000000, .75R: 0.000000,  count: 2
Region 94 Avg IOU: 0.158880, Class: 0.212324, Obj: 0.631945, No Obj: 0.513828, .5R: 0.000000, .75R: 0.000000,  count: 3
Region 106 Avg IOU: 0.222601, Class: 0.908435, Obj: 0.473639, No Obj: 0.463986, .5R: 0.000000, .75R: 0.000000,  count: 1
Region 82 Avg IOU: 0.246336, Class: 0.590457, Obj: 0.412009, No Obj: 0.550207, .5R: 0.200000, .75R: 0.000000,  count: 5
Region 94 Avg IOU: -nan, Class: -nan, Obj: -nan, No Obj: 0.515382, .5R: -nan, .75R: -nan,  count: 0
Region 106 Avg IOU: -nan, Class: -nan, Obj: -nan, No Obj: 0.462789, .5R: -nan, .75R: -nan,  count: 0
Region 82 Avg IOU: 0.261491, Class: 0.579703, Obj: 0.080441, No Obj: 0.545202, .5R: 0.000000, .75R: 0.000000,  count: 1
Region 94 Avg IOU: -nan, Class: -nan, Obj: -nan, No Obj: 0.513039, .5R: -nan, .75R: -nan,  count: 0
Region 106 Avg IOU: -nan, Class: -nan, Obj: -nan, No Obj: 0.460418, .5R: -nan, .75R: -nan,  count: 0
Region 82 Avg IOU: 0.085725, Class: 0.675183, Obj: 0.583687, No Obj: 0.545310, .5R: 0.000000, .75R: 0.000000,  count: 2
Region 94 Avg IOU: 0.322056, Class: 0.609898, Obj: 0.376546, No Obj: 0.512992, .5R: 0.333333, .75R: 0.000000,  count: 3
Region 106 Avg IOU: -nan, Class: -nan, Obj: -nan, No Obj: 0.457888, .5R: -nan, .75R: -nan,  count: 0
Region 82 Avg IOU: 0.371791, Class: 0.457519, Obj: 0.598735, No Obj: 0.546651, .5R: 0.500000, .75R: 0.000000,  count: 2
Region 94 Avg IOU: 0.248187, Class: 0.471107, Obj: 0.693366, No Obj: 0.515690, .5R: 0.000000, .75R: 0.000000,  count: 3
Region 106 Avg IOU: 0.013163, Class: 0.228523, Obj: 0.993441, No Obj: 0.462508, .5R: 0.000000, .75R: 0.000000,  count: 1
Region 82 Avg IOU: 0.224153, Class: 0.716591, Obj: 0.754158, No Obj: 0.548121, .5R: 0.000000, .75R: 0.000000,  count: 3
```

Eğitim sırasında
Yolo'nun çıktıları

82-94-106 katmanlardaki nesne algılama yaptığı görebiliriz



Eğitim bittikten sonra custom.data belirttiğimiz backup klasörümüzün içi



YOLO'yu indirdiğinizde

```
git clone https://github.com/pjreddie/darknet
cd darknet
Make
```

Adımlarını izleyerek libdarknet.so dosyasını elde ediyoruz.

Yazılan Python Scripti

```
for i in range (objects.shape[0]):

    if (names[i].decode() == "face"):
        face_l = Point(int(objects[i][0]) - int(objects[i][2]/2), int(objects[i][1]) - int(objects[i][3]/2))
        face_r = Point(int(objects[i][0]) + int(objects[i][2]/2), int(objects[i][1]) + int(objects[i][3]/2))
    elif (names[i].decode() == "cigarette"):
            cigarette_l = Point(int(objects[i][0]) - int(objects[i][2]/2), int(objects[i][1]) - int(objects[i][3]/2))
            cigarette_r = Point(int(objects[i][0]) + int(objects[i][2]/2), int(objects[i][1]) + int(objects[i][3]/2))
    elif (names[i].decode() == "phone"):
            phone_l = Point(int(objects[i][0]) - int(objects[i][2]/2), int(objects[i][1]) - int(objects[i][3]/2))
            phone_r = Point(int(objects[i][0]) + int(objects[i][2]/2), int(objects[i][1]) + int(objects[i][3]/2))
    elif (names[i].decode() == "wheel"):
            wheel_l = Point(int(objects[i][0]) - int(objects[i][2]/2), int(objects[i][1]) - int(objects[i][3]/2))
            wheel_r = Point(int(objects[i][0]) + int(objects[i][2]/2), int(objects[i][1]) + int(objects[i][3]/2))
    elif (names[i].decode() == "hand"):
        if hand_counter %2==0:
            hand1_l = Point(int(objects[i][0]) - int(objects[i][2]/2), int(objects[i][1]) - int(objects[i][3]/2))
            hand1_r = Point(int(objects[i][0]) + int(objects[i][2]/2), int(objects[i][1]) + int(objects[i][3]/2))
            hand_counter = hand_counter + 1
        elif hand_counter %2 ==1:
            hand2_l = Point(int(objects[i][0]) - int(objects[i][2]/2), int(objects[i][1]) - int(objects[i][3]/2))
            hand2_r = Point(int(objects[i][0]) + int(objects[i][2]/2), int(objects[i][1]) + int(objects[i][3]/2))
            hand_counter+=1

    rx = (int(objects[i][0]) - int(objects[i][2]/2), int(objects[i][1]) - int(objects[i][3]/2))
    ry = (int(objects[i][0]) + int(objects[i][2]/2), int(objects[i][1]) + int(objects[i][3]/2))
    tx = int(objects[i][0])
    ty = int(objects[i][1])
    cv2.putText(frame, names[i].decode(), (tx, ty), cv2.FONT_HERSHEY_SIMPLEX, 0.5,(0,255,0), 1, 1)
    cv2.rectangle(frame, rx, ry, (0,255,0), 2)
```

YOLO’nun mantığına göre nesnelerin sol üst noktasını ve Sağ alt noktasını belirlediğimiz for döngümüz

```
if (face_l != None and face_r != None):
    if (phone_l != None and phone_r != None):
        if(overlap(face_l, face_r, phone_l, phone_r)):
            now = time.time()
            if now - play_time > 2.5:
               print('')
if (face_l != None and face_r != None):
    if (cigarette_l != None and cigarette_r != None):
        if(overlap(face_l, face_r, cigarette_l, cigarette_r)):
            now = time.time()
            if now - play_time > 2.5:
                print("Sigara")
if ((hand1_l != None and hand1_r != None)):
    if (wheel_l != None and wheel_r != None):
            if(overlap(wheel_l, wheel_r, hand1_l, hand1_r)):
                    print("")
            else:
                    print("1 Elde çalışması")
if ((hand2_l != None and hand2_r != None)):
    if (wheel_l != None and wheel_r != None):
            if(overlap(wheel_l, wheel_r, hand2_l, hand2_r) or (overlap(wheel_l, wheel_r, hand1_l, hand1_r))):
                    print("")
            else:
                    print("2 Elde çalışması")
```

Döngüde noktalarını atadığımız cisimlerin noktaların karşılaştırması için overlap foksiyonuna gönderiyoruz

```python
def overlap(l1, r1, l2, r2):
    Rect1_x1 = l1.x
    Rect1_y1 = l1.y
    Rect1_x2 = r1.x
    Rect1_y2 = r1.y

    Rect2_x1 = l2.x
    Rect2_y1 = l2.y
    Rect2_x2 = r2.x
    Rect2_y2 = r2.y

    if (Rect2_x2 > Rect1_x1 and Rect2_x2 < Rect1_x2) or \
        (Rect2_x1 > Rect1_x1 and Rect2_x1 < Rect1_x2):
        x_match = True
    else:
        x_match = False
    if (Rect2_y2 > Rect1_y1 and Rect2_y2 < Rect1_y2) or \
        (Rect2_y1 > Rect1_y1 and Rect2_y1 < Rect1_y2):
        y_match = True
    else:
        y_match = False

    if x_match and y_match:
        return True

    else:
        return False
```

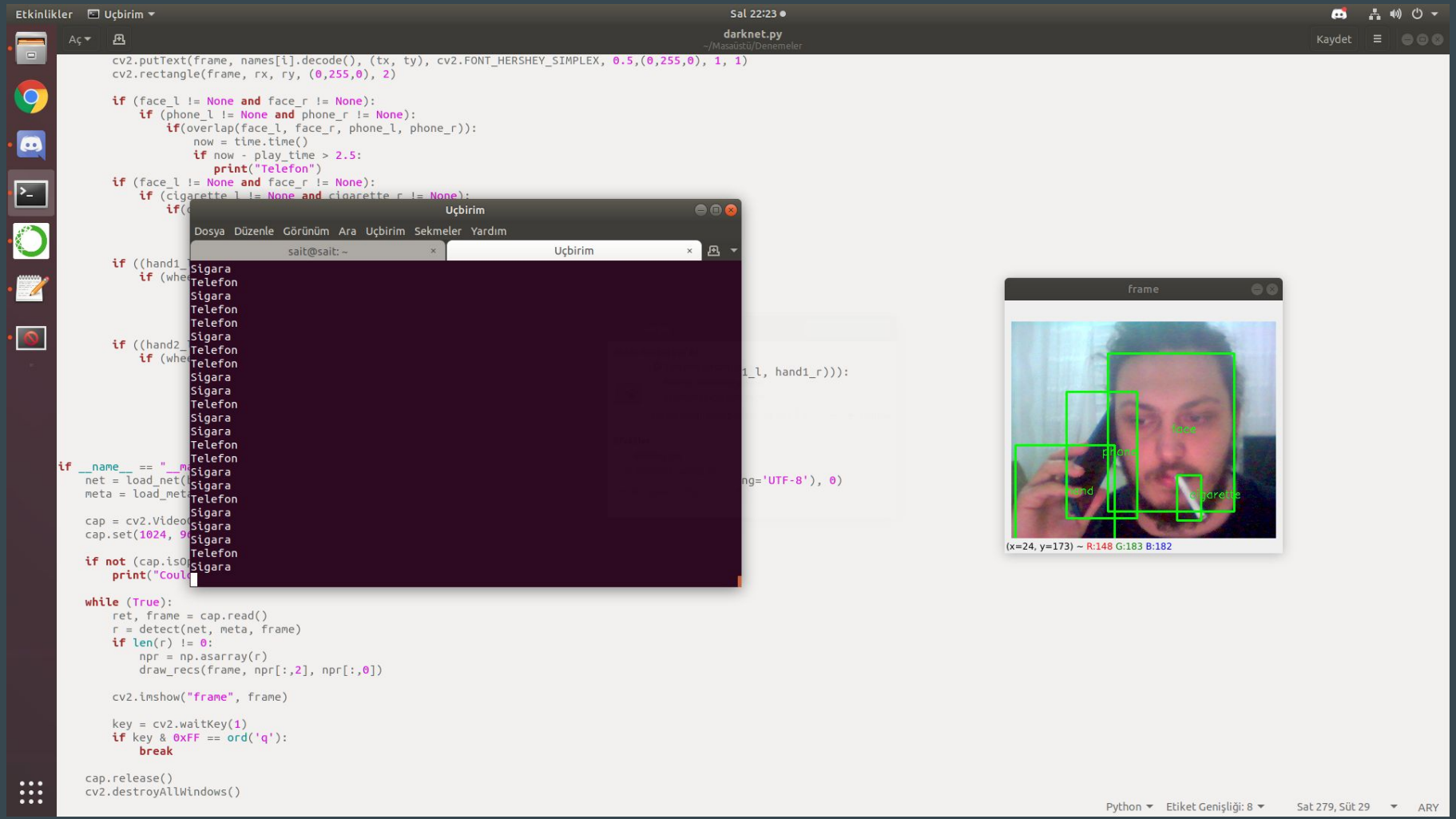
Büyük çerçevede küçük çerçeveyi arayan Overlap fonksiyonu

```python
if __name__ == "__main__":
    net = load_net(bytes("yolov3-voc.cfg", encoding='UTF-8'), bytes("yolov3-voc_final.weights",encoding='UTF-8'), 0)
    meta = load_meta(bytes("custom.data",encoding='UTF-8'))

    cap = cv2.VideoCapture('deneme.mp4')
    cap.set(1024, 960)

    if not (cap.isOpened()):
        print("Could not open video device")

    while (True):
        ret, frame = cap.read()
        r = detect(net, meta, frame)
        if len(r) != 0:
            npr = np.asarray(r)
            draw_recs(frame, npr[:,2], npr[:,0])

        cv2.imshow("frame", frame)

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

    cap.release()
    cv2.destroyAllWindows()
```

Main

```python
        return False

def detect(net, meta, image, thresh=.5, hier_thresh=.5, nms=.45):
    im = nparray_to_image(image)
    num = c_int(0)
    pnum = pointer(num)
    predict_image(net, im)
    dets = get_network_boxes(net, im.w, im.h, thresh, hier_thresh, None, 0, pnum)
    num = pnum[0]
    if (nms): do_nms_obj(dets, num, meta.classes, nms);
    res = []
    for j in range(num):
        for i in range(meta.classes):
            if dets[j].prob[i] > 0:
                b = dets[j].bbox
                res.append((meta.names[i], dets[j].prob[i], (b.x, b.y, b.w, b.h)))
    res = sorted(res, key=lambda x: -x[1])
    free_image(im)
    free_detections(dets, num)
    return res
```

Nesne tespit fonksiyonu


Etkinlikler
Uçbirim
Sal 22:23
darknet.py
~/Masaüstü/Denemeler
Aç
Kaydet
cv2.putText(frame, names[i].decode(), (tx, ty), cv2.FONT_HERSHEY_SIMPLEX, 0.5,(0,255,0), 1, 1)
cv2.rectangle(frame, rx, ry, (0,255,0), 2)
if (face_l != None and face_r != None):
if (phone_l != None and phone_r != None):
if(overlap(face_l, face_r, phone_l, phone_r)):
now = time.time()
if now - play_time > 2.5:
print("Telefon")
if (face_l != None and face_r != None):
if (cigarette_l != None and cigarette_r != None):
if ((hand1_
if (whe
if ((hand2_
if (whe
1_l, hand1_r))):
if __name__ == "__m
net = load_net(
meta = load_met
ng='UTF-8'), 0)
cap = cv2.Video
cap.set(1024, 9
if not (cap.isO
print("Coul
while (True):
ret, frame = cap.read()
r = detect(net, meta, frame)
if len(r) != 0:
npr = np.asarray(r)
draw_recs(frame, npr[:,2], npr[:,0])
cv2.imshow("frame", frame)
key = cv2.waitKey(1)
if key & 0xFF == ord('q'):
break
cap.release()
cv2.destroyAllWindows()
Uçbirim
Dosya Düzenle Görünüm Ara Uçbirim Sekmeler Yardım
sait@sait: ~
Uçbirim
Sigara
Telefon
Sigara
Telefon
Telefon
Sigara
Telefon
Telefon
Sigara
Sigara
Telefon
Sigara
Sigara
Telefon
Telefon
Sigara
Sigara
Telefon
Sigara
Sigara
Sigara
Telefon
Sigara
frame
face
phone
hand
cigarette
(x=24, y=173) ~ R:148 G:183 B:182
Python
Etiket Genişliği: 8
Sat 279, Süt 29
ARY

Kayıp grafiği