

KARABÜK ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ(U.E)
BİTİRME PROJESİ II TEZ RAPORU

DERİN ÖĞRENME İLE NESNE TESPİTİ

Projenin amacı:

Son yıllarda büyük oranda veritabanlarının kullanımı, iletişim teknolojilerinin artması, çoklu ortam teknolojilerinin hızlı bir şekilde hayatımıza girmesi ve mobil teknolojilerinin sıkça kullanılması ile içerik tabanlı görüntü tespiti önemli bir araştırma konusu haline gelmiştir. Görüntü tespiti için geliştirilen birçok yöntem bulunmaktadır. Bunlardan biride derin öğrenme yöntemleridir. Yapay zekâ konularından olan derin öğrenme yöntemleri son zamanlarda hızla gelişen ve birçok alanda kullanılan akıllı sistemler olarak karşımıza çıkmıştır. İçerik tabanlı görüntü çıkarımı ile birlikte kullanılan derin öğrenme yöntemleri, nesne tespit etme, sorgulama, indeksleme, kenar çıkarımı, sahne tespiti gibi birçok alanda çalışmalar bulunmaktadır.

Bu proje kapsamında derin öğrenme yöntemlerini kullanarak nesne tespiti yapacağız.

Kullanılacak Yazılımlar:

-Pyhton - Anaconda -Derin Öğrenme yazılımları
-pycram -Yapay sinir ağıları programları -OpenCV -Tensorflow

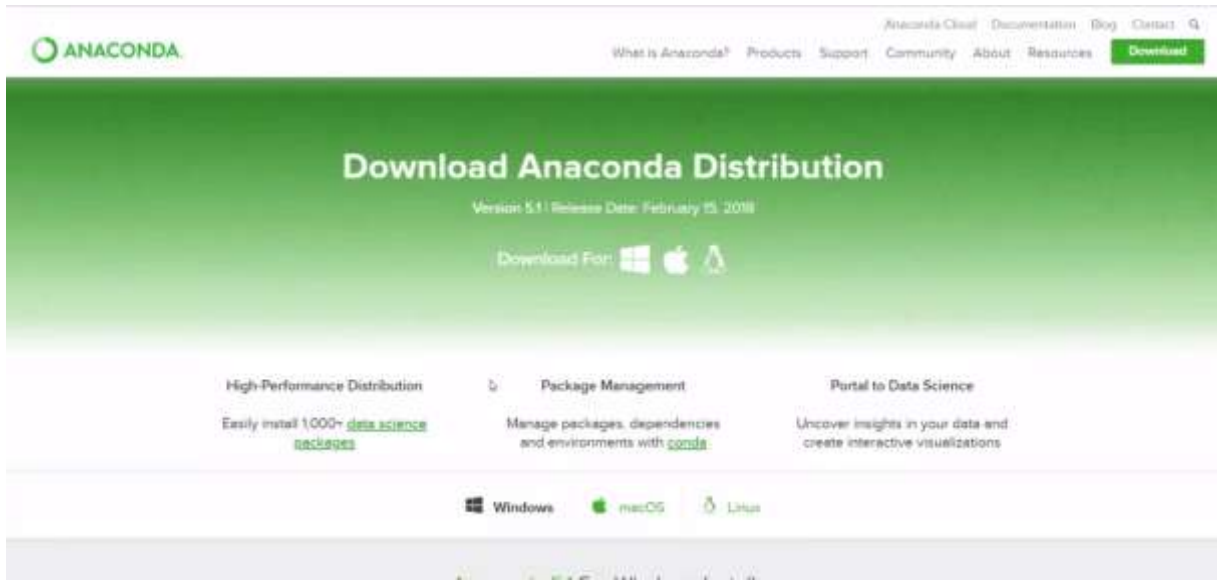
Bu aşamada derin öğrenme kütüphanelerinden yararlanmak ve projeyi çalıştırabilmek için gerekli programların eksiksiz indirilmesi gerekmektedir.

İLK AŞAMADA YAPILANLAN KURULUMLAR:

Anoconda yüklenmelidir. Neden? Bir python dağıtımdır, bununla birlikte makine öğrenimi kütüphaneleri , görüntü işleme kütüphaneleri, veri işleme kütüphaneleri gibi birçok kütüphane barındırır. Aşağıda gösterilen linkten kurulum yapılmalıdır;

ANACONDA

- <https://www.anaconda.com/download/>
- 64-bit Anaconda indirip kurun. Ayrıca Python kurulumu yapmanıza gerek yok. Python Anaconda'yla birlikte gelecek.
- Kurulum esnasında "add python to your PATH" seçeneğini işaretleyin.

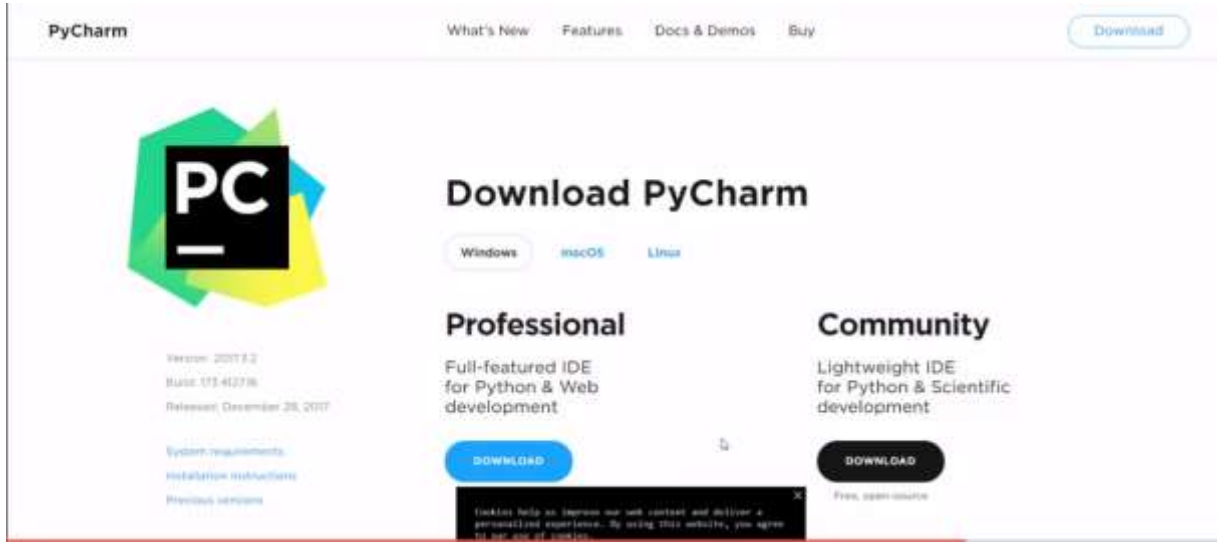


Kodların python için özel olarak hazırlanmış pycharm üzerinden yazılacak. Pycharm bir python IDE'sidir, yani üzerinde rahatlıkla kodları yazacağımız ortamdır. Bundan dolayı aşağıdaki link uzantısından pycharm kurulumu yapılmalıdır. Python için yazılım yazma ve geliştirme ortamıdır IDE sidir.

PYCHARM

- <https://www.jetbrains.com/pycharm/download/>
- Ben Pycharm Community Edition kullanıyorum. Siz isterseniz farklı bir IDE'de kullanabilirsiniz.
- Pycharm kurulumu yaptıysanız interpreter belirtin.
- File/Default Settings/Project Interpreter
- Sağdaki küçük butona tıklayıp "Add Local" seçin.
- Açılan pencereden "System Interpreter" seçeneğini seçip anaconda içerisinde python.exe yolunu verin. Bu yol şurası:
- C:\Users\KullanıcıAdı\Anaconda3\python.exe

PyCharm, çapraz platform bir Python geliştirme ortamı'dır. Kod analizleri, grafiksel hata ayıklamacısı, versiyon kontrol sistemi ile entegre ve Django ile Python web geliştirmeleri yapılmasını sağlamaktadır. Çapraz platformu Windows, OS X ve GNU/Linux işletim sistemleri üzerinde çalışır.



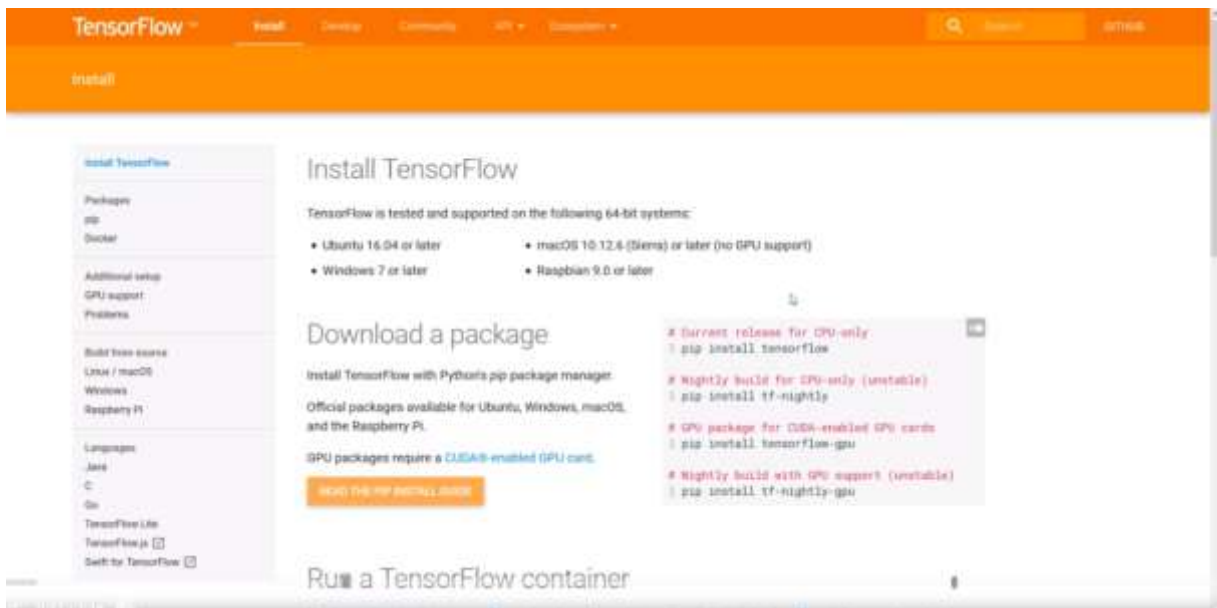
TensorFlow, bir dizi görev arasında veri akışı ve türevlenebilir programlama için kullanılan ücretsiz ve açık kaynaklı bir yazılım kütüphanesidir. Sembolik bir matematik kütüphanesidir ve sinir ağları gibi makine öğrenimi uygulamaları için de kullanılır. Google'da hem araştırma hem de üretim için kullanılır.

Proje için gerekli olan derin öğrenme kütüphaneleri için tensorflow kuruldu.

TENSORFLOW

Tensorflow kurarken iki seçeneğiniz var. CPU veya GPU. Eğer uygun ekran kartınız varsa kesinlikle GPU için kurulum yapın, CPU'ya göre daha hızlı eğitim gerçekleşecek. Nvidia'dan farklı bir ekran kartınız varsa CPU için kurulum yapmanız gerekiyor. Sadece Nvidia ekran kartları destekleniyor.

[About TensorFlow](#) [Documentation](#) [Blog](#) [Privacy](#)

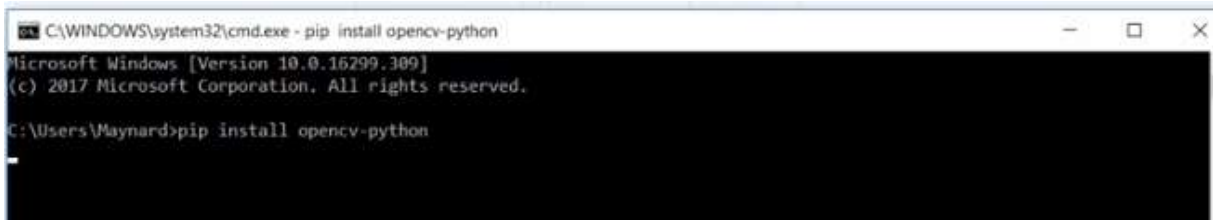


İki önemli kütüphane kurulması gerekiyor opencv için ;

OpenCV 47.000'den fazla kullanıcı tarafından yaygın olarak kullanılan ve 14 milyonu aşkın indirme sayısına sahip, açık kaynak kodlu 'Bilgisayarlı Görü' kütüphanesidir. Bu kütüphaneyi kullanarak, hem klasik hem de son teknoloji bilgisayarlı görme ve makine öğrenimi algoritmalarıyla ilgili çalışmalar gerçekleştirebilir.

```
1 | pip install opencv-python
2 | pip install imageio
```


Aşağıdaki gibi "cmd" ekranı açarak OpenCV kütüphanelerini indirmemiz gerekiyor.



```
C:\WINDOWS\system32\cmd.exe - pip install opencv-python
Microsoft Windows [Version 10.0.16299.309]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\Maynard>pip install opencv-python
_
```

OpenCV kurulduktan sonra Imageio kurulmalı, imageio resimler üzerinde işlem yapmamızı sağlıyor. Yine videolar için de imageio kullanacağız.



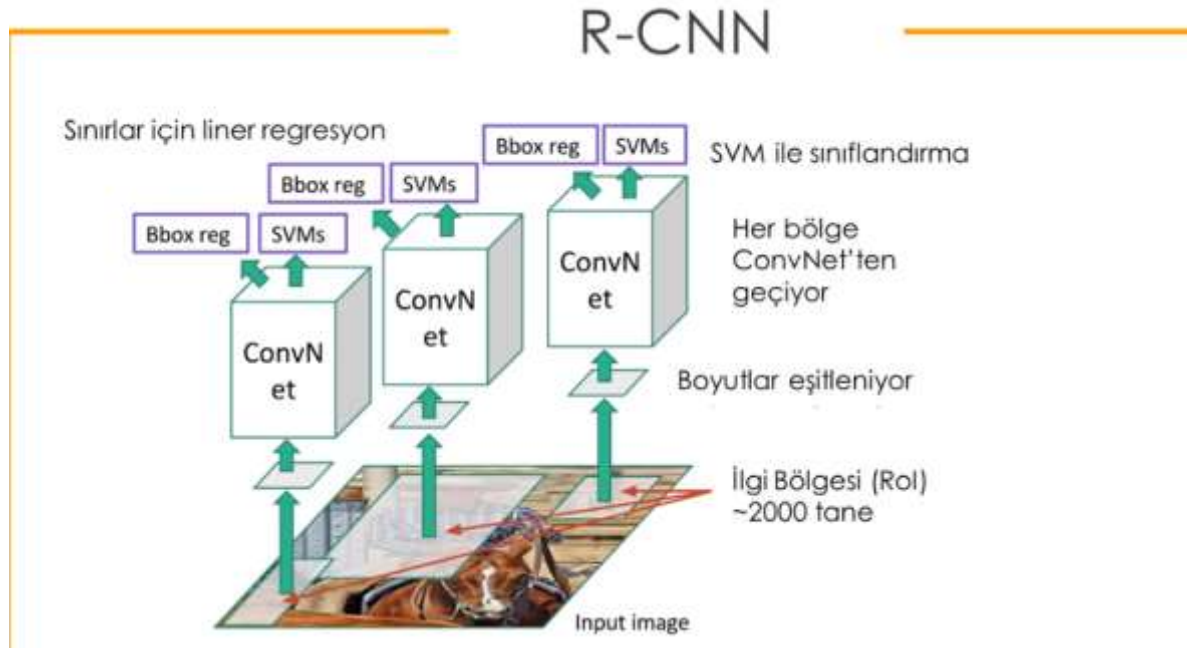
```
C:\Users\Maynard>pip install imageio
Requirement already satisfied: imageio in c:\users\maynard\anaconda3\lib\site-packages

C:\Users\Maynard>_
```

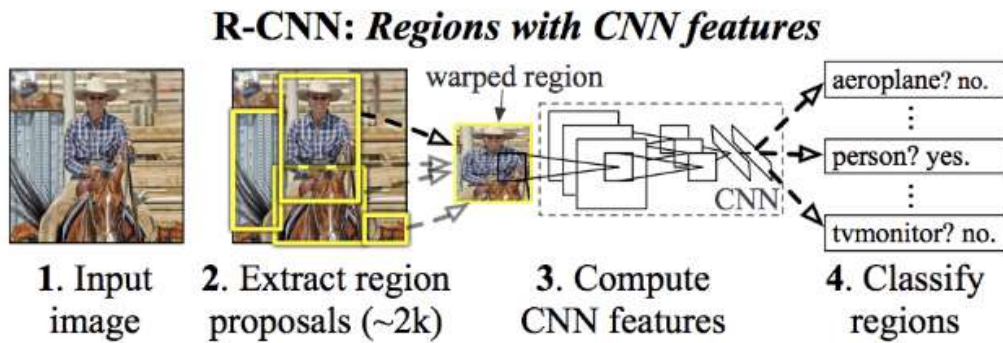
***Proje için kullanılacak algoritmalar öğrenildi :

Algotirmalar neden önemli? Başarılı sonuçlar elde etmek için resmin ya videonun(videolar aslında ard arda sıralanmış resimlerdir) kalitesi, boyutu, çözünürlüğü önemli etkenlerdir ancak en önemli parametre kullanılan algoritmadır şimdi bunları tek tek açıklayacağım;

1-R-CNN algoritması



Çalışma mantığı: nesne olması muhtemel 2000 bölge oluşturur bu bölgeleri sinir ağından geçirip sınıflandırma yapar. Olumsuz tarafı yavaş çalışan bir algoritmadır.



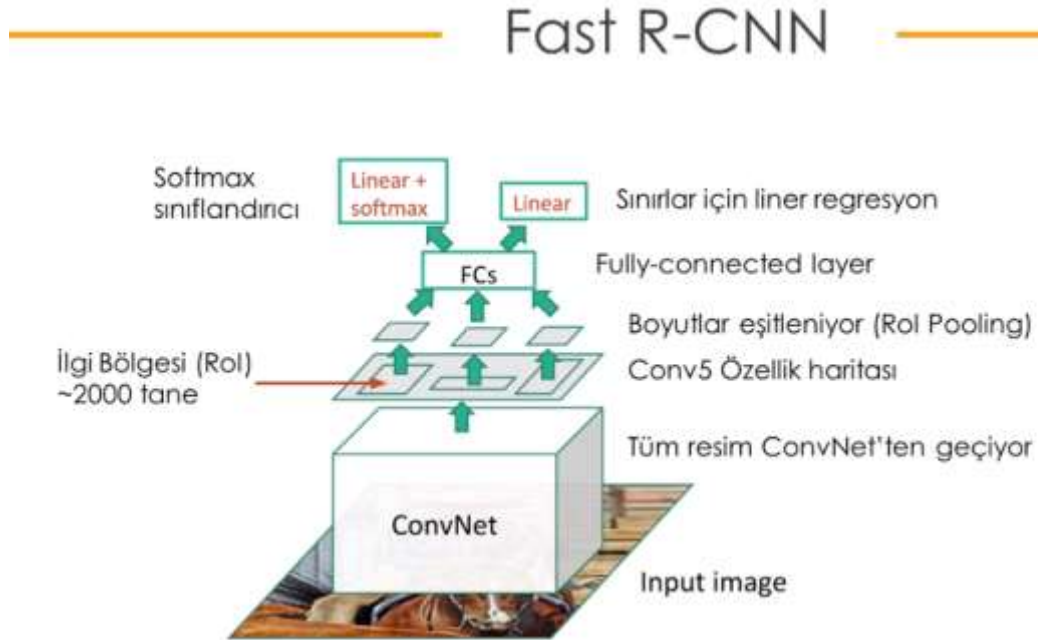
1. RCNN Algoritmasının Çalıştırılması

Bu tabloda 1. fotoğrafta işlem yapılacak giriş görüntümüz mevcut.2. görüntüde ise çıkartılabilecek 2000 tane aday bölge önerisi yer almakta ve bu bölgeler bir kareye çarpıtılmaktadır. Çıktı olarak ise 4096

(64x64) boyutlu bir özellik vektörü üreten evrişimli bir sinir ağına atandı. CNN bir özellik çıkarıcı olarak işlev görür ve çıktı yoğun katman, görüntüden çıkartılan özelliklerden oluşur ve elde edilen özellikler, o aday bölge teklifindeki nesnenin varlığını sınıflandırmak için bir SVM'ye beslenir.

3. görüntüde CNN özellikleri de devreye girerek kendine has algoritması ile hesaplama yapar ve ardından belirli bölgelere göre görüntü sınıflandırması yapar. Kesit olarak aldığı bölgeye uçak, tv monitörü veya insan vb. özellikleri sorgulayarak doğruyu tahmin eder.

2-Fast R-CNN algoritması



Çalışma mantığı: resmi sinir ağından geçiriyoruz. Daha sonra orijinal resime uyan bir özellik haritası çıkarılıyor, özellik haritası üzerinden bölge haritası çıkarılıp sinir ağına verilir. İlk resim 2000 bölgeye ayrıldığı için çok daha hızlı bir algoritmadır R-CNN'e göre.

3-Faster R-CNN:

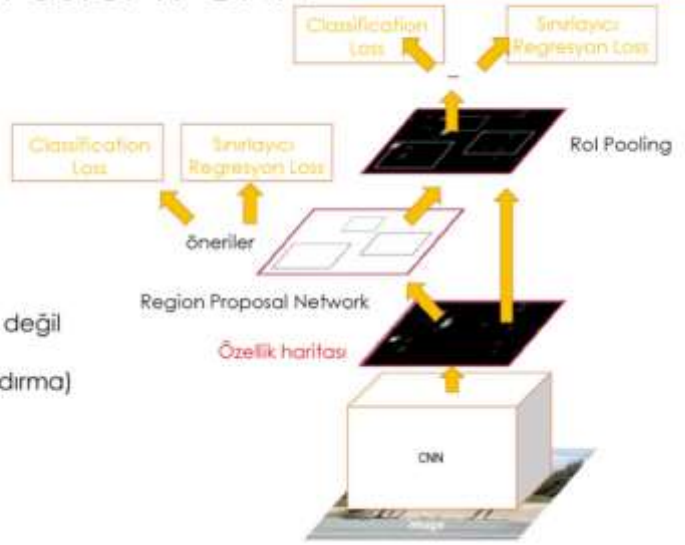
Bölge önerileri yapılı ve bu sayede daha isabetli nesne tespitleri yapılabilir .Ancak daha fazla işlem yapması gerektiği için SSD ye göre daha yavaştır.

Faster R-CNN

Region Proposal Network (RPN)
İle öneriler belirleniyor.

4 loss ile birlikte çalışır:

1. RPN sınıflandırıcı Nesne/Nesne değil
2. RPN sınırlayıcı
3. Son sınıf skorları (Nesne sınıflandırma)
4. Son sınırlar



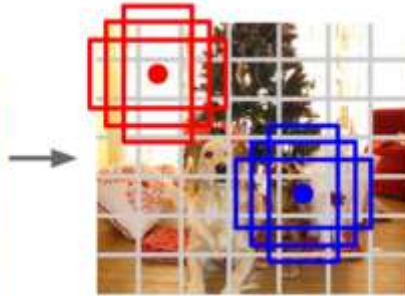
4-SSD Algoritması:

Tek seferde resimdeki nesneler tespit edilir. Resim sınır ağından sadece 1 defa geçtiği için işlem hızı oldukça iyi, Ancak isabet oranı Faster R-CNN'e göre daha düşük. Her bölge için farklı işlemler yapmak yerine bütün tahminleri tek bir konvansiyonel sinir ağı içinde yapıyoruz.

SSD



Input: 3 x H x W

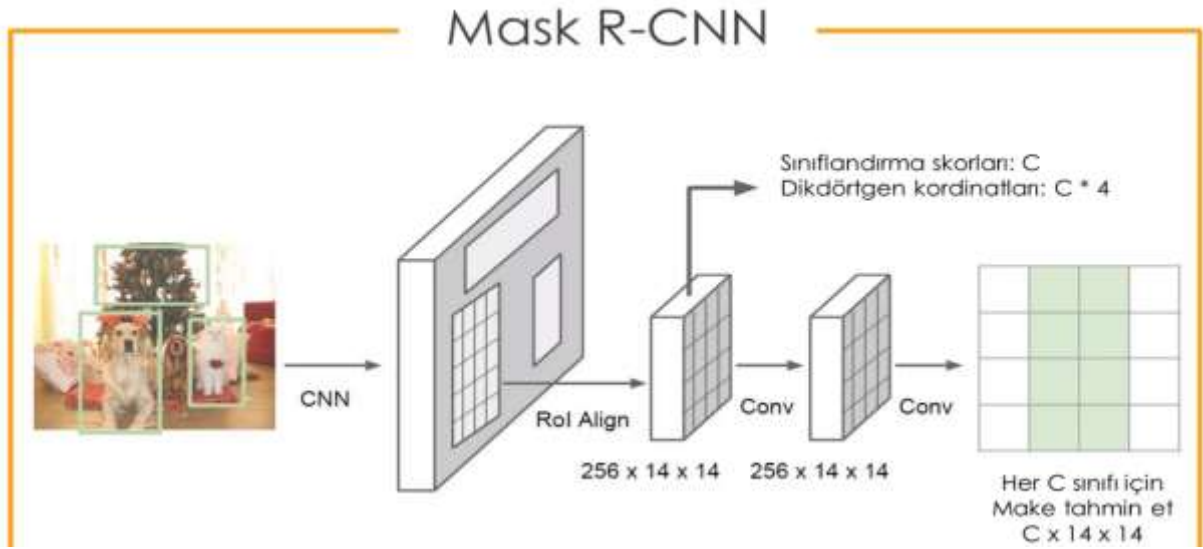


Resim 7x7'ye bölünüyor. Grid ortalanacak şekilde etrafında kareler oluşturuluyor. (B = 3)

Output: 7 x 7 x (5 * B + C)

B: Dikdörtgenler
C: Tahmin skorları

5- Mask R-CNN



İnput olarak bir resim alıyoruz. Daha sonra bu resmi konvansiyonel sinir ağından geçirerek bölge önerileri alıyoruz bu noktadan sonra sinir ağı ikiye ayrılıyor ilkinde bir sınıflandırma yapılır ve bulunan nesnenin sınırları belirlenir ikinci de ise mask ile eklenen kısım var burada sinir ağı özellik haritasını alıp tespit edilen nesnelerin resimde hangi pixsellerde bulunduğunu tahmin ediyor bu şekilde nesne bulunan pixseller maskeleniyor kısacası maskeleme yapmak için eklenen bir sinir ağı var Faster R-CNN den farklı olarak.

***Burada öğrendiğim modelleri nasıl kullanıldığı bilgileri var. Burada hazır olarak google tarafından eğitilmiş modeller var.Sıradan bilgisayarlar ile bir modelin eğitilmesi çok zor tek bir modeli bile eğitmek ortalama bir CPU ile aylar sürebilir. Tensorflow bize böyle bir kolaylık sağlıyor.

Tensorflow Object Detection API

Creating accurate machine learning models capable of localizing and identifying multiple objects in a single image remains a core challenge in computer vision. The TensorFlow Object Detection API is an open source framework built on top of TensorFlow that makes it easy to construct, train and deploy object detection models. At Google we've certainly found this codebase to be useful for our computer vision needs, and we hope that you will as well.



Daha önce eğitilmiş bir nesnenin python kodu:

SSD modeliyle ;

```
# In[4]:

# What model to download.
MODEL_NAME = 'ssd_mobilenet_v1_coco_2017_11_17'
MODEL_FILE = MODEL_NAME + '.tar.gz'
DOWNLOAD_BASE = 'http://download.tensorflow.org/models/object_detection/'

# Path to frozen detection graph. This is the actual model that is used for the object detection.
PATH_TO_CKPT = MODEL_NAME + '/frozen_inference_graph.pb'

# List of the strings that is used to add correct label for each box.
PATH_TO_LABELS = os.path.join('data', 'mscoco_label_map.pbtxt')

NUM_CLASSES = 90

# ## Download Model

# In[5]:
```

%93-94 oranında başarılı bir şekilde sonuç dönderdi.



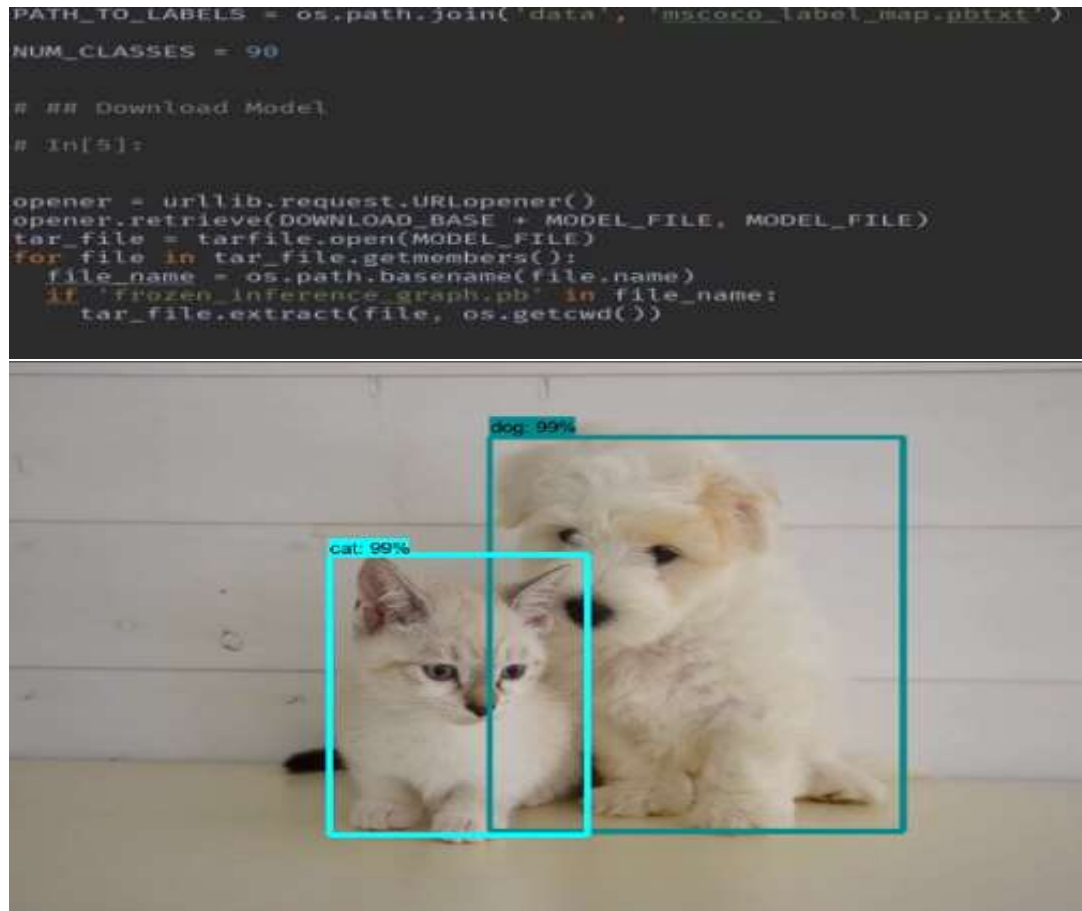
Eğer resim değilse aynısını bir video üzerinde uygulayacaksak bu şekilde webcam de nesne tanıma yapılıyor:

Kod:

```
with tf.Session(graph=detection_graph) as sess:
    while True:
        ret, image_np = cap.read()
        # Expand dimensions since the model expects images to have shape: [1, None, None, 3]
        image_np_expanded = np.expand_dims(image_np, axis=0)
        # Actual detection.
        output_dict = run_inference_for_single_image(image_np, detection_graph)
        # Visualization of the results of a detection.
        vis_util.visualize_boxes_and_labels_on_image_array(
            image_np,
            output_dict['detection_boxes'],
            output_dict['detection_classes'],
            output_dict['detection_scores'],
            category_index,
            instance_masks=output_dict.get('detection_masks'),
            use_normalized_coordinates=True,
            line_thickness=8)
        cv2.imshow('Video', image_np)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            cv2.destroyAllWindows()
            break
```

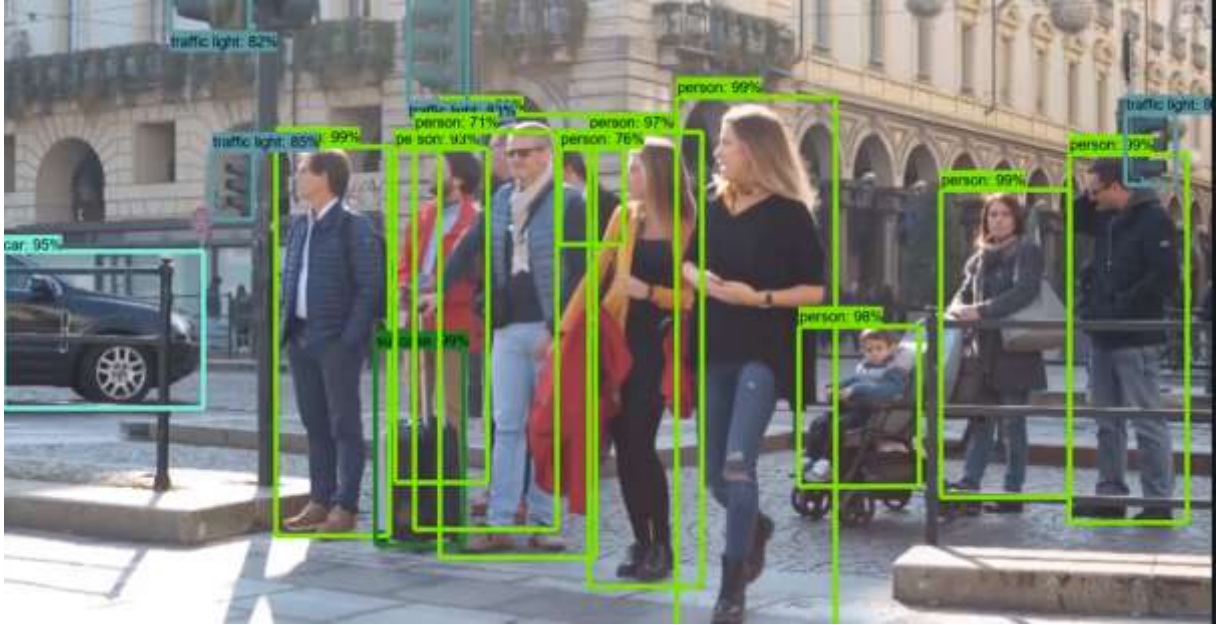
Faster R-CNN 'e göre hazırlanmış kodlar:

Başarı % 99 'a çıktı.



Faster R-CNN ile video için nesne tanıma kodu:SSD den daha başarılı nesne tanıma konusunda, videodaki nesneleri büyük oranda tanıdı.

```
with tf.Session(graph=detection_graph) as sess:
    for i, image_np in enumerate(reader):
        # Expand dimensions since the model expects images to have shape: [1, None, None, 3]
        image_np_expanded = np.expand_dims(image_np, axis=0)
        # Actual detection.
        output_dict = run_inference_for_single_image(image_np, detection_graph)
        # Visualization of the results of a detection.
        vis_util.visualize_boxes_and_labels_on_image_array(
            image_np,
            output_dict['detection_boxes'],
            output_dict['detection_classes'],
            output_dict['detection_scores'],
            category_index,
            instance_masks=output_dict.get('detection_masks'),
            use_normalized_coordinates=True,
            line_thickness=8)
        writer.append_data(image_np)
        print(i)
    writer.close()
```



COCO DATA SETİ:

Coco data seti günlük hayatta yaygın olarak bulunan 90 tane sınıftan oluşuyor. Coco datasetinde eğitilen bir modelin tanıyabileceği 90 tane nesne var fakat bizim istediğimiz nesne bunların arasında olmayabilir. Bu durumda kendi oluşturduğumuz bir data seti üzerinde yeni bir model oluşturmamız gerekir. Sıfırdan model oluşturmak hem çok zaman alır hem de bunun için güçlü bilgisayarlara ihtiyaç var. Tensorflow'un hazır modelleri üzerinden kendi nesnemizi tanıtmamız gerekiyor.

**COCO**
Common Objects in Context

[Home](#) [People](#) [Dataset](#) [Tasks](#) [Evaluate](#)

info@cocodataset.org

News

- 2017 Challenge Winners for Detection, Keypoint, & Stuff tasks have been announced! Please visit the Joint COCO and Places Recognition ICCV workshop page for details.
- This website is now hosted on Github, which provides page source and history.
- Keypoint analysis tools are now available, see [keypoints evaluation](#), Section 4.

What is COCO?

COCO is a large-scale object detection, segmentation, and captioning dataset. COCO has several features:

- Object segmentation
- Recognition in context
- Superpixel stuff segmentation
- 330K images (>200K labeled)
- 1.5 million object instances
- 80 object categories
- 91 stuff categories
- 5 captions per image
- 250,000 people with keypoints

Collaborators

Tsingyi Lin (Google Brain)
Genevieve Patterson (MIT)
Matteo R. Rospocher (Catalyst)
Yin Cui (Google Tech)
Michael Maire (TUM/Chicago)
Serge Belongie (Google Tech)
Lubomir Bourdev (Yahoo/Stan, Inc.)
Ross Girshick (FMR)
James Hays (Google Tech)
Pietro Perona (Catalyst)
Orra Ramanan (CMU)
Larry Zitnick (FMR)
Piotr Dollar (FMR)

Sponsors

 CVDF
 Microsoft
 facebook
 Mighty Ai

Research Paper

Download the paper that describes the Microsoft COCO dataset.



 [Download paper here](#)

COCO-trained models (#coco-models)

Model name	Speed (ms)	COCO mAP[^1]	Outputs
ssd_mobilenet_v1_coco	30	21	Boxes
ssd_inception_v2_coco	42	24	Boxes
faster_rcnn_inception_v2_coco	58	28	Boxes
faster_rcnn_resnet50_coco	89	30	Boxes
faster_rcnn_resnet50_lowproposals_coco	64		Boxes
rfcn_resnet101_coco	92	30	Boxes
faster_rcnn_resnet101_coco	106	32	Boxes
faster_rcnn_resnet101_lowproposals_coco	82		Boxes
faster_rcnn_inception_resnet_v2_atrous_coco	620	37	Boxes
faster_rcnn_inception_resnet_v2_atrous_lowproposals_coco	241		Boxes
faster_rcnn_nas	1833	43	Boxes
faster_rcnn_nas_lowproposals_coco	540		Boxes
mask_rcnn_inception_resnet_v2_atrous_coco	771	36	Masks
mask_rcnn_inception_v2_coco	79	25	Masks
mask_rcnn_resnet101_atrous_coco	470	33	Masks
mask_rcnn_resnet50_atrous_coco	343	29	Masks

Bu modelleri kendi datasetimiz üzerinden tekrar eğitirsek yüklediğimiz resim ve videoları tespit edebiliriz. Coco datasetinde 200 bin tane resim bulunuyor. Bu nesneler üzerinde eğitilmiş model işimizi görecektir. Bizim yapacağımız zaten eğitilmiş olan modeli kendi verilerimiz üzerinde tekrar tanımlamak olacaktır.

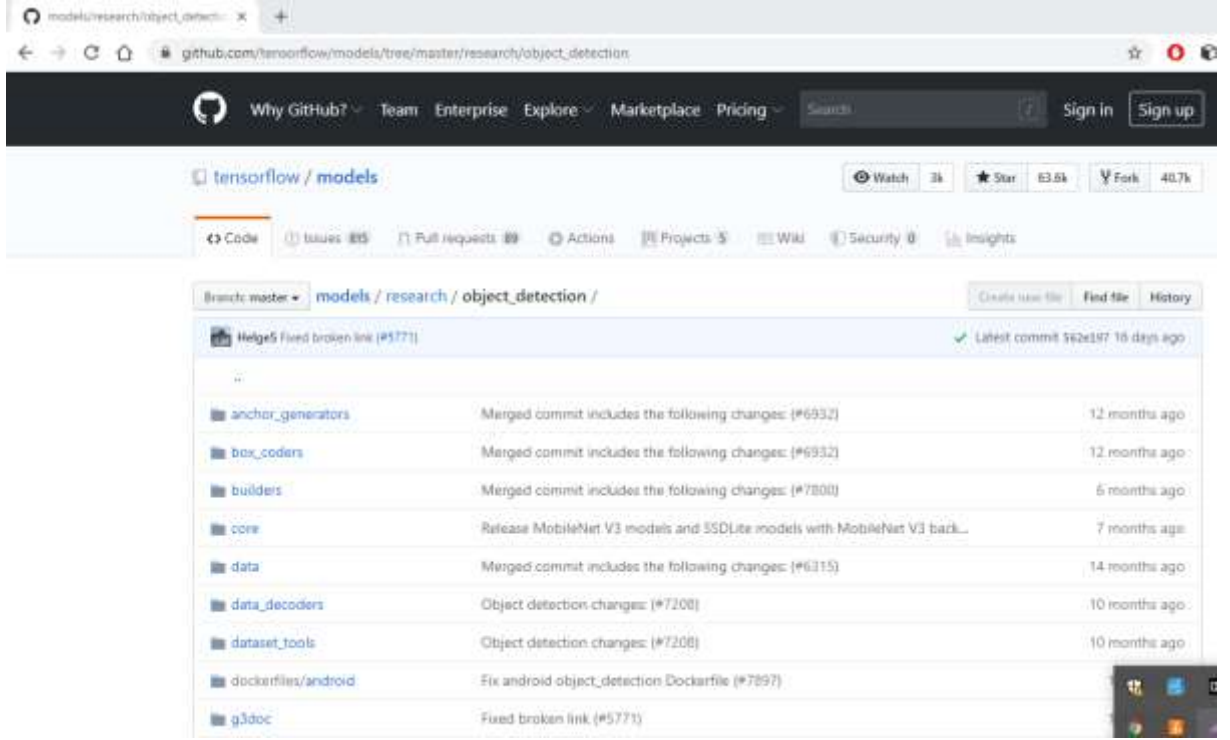
PROGRAM KURULUMLARI, PYTHON KODLARI VE BİLGİSAYARDA NESNE TESPİTİ

UYGULAMASI: Bu kısımda adımları eksiksiz olarak gerçekleştirmemiz gerekir yoksa modelimiz hata verir

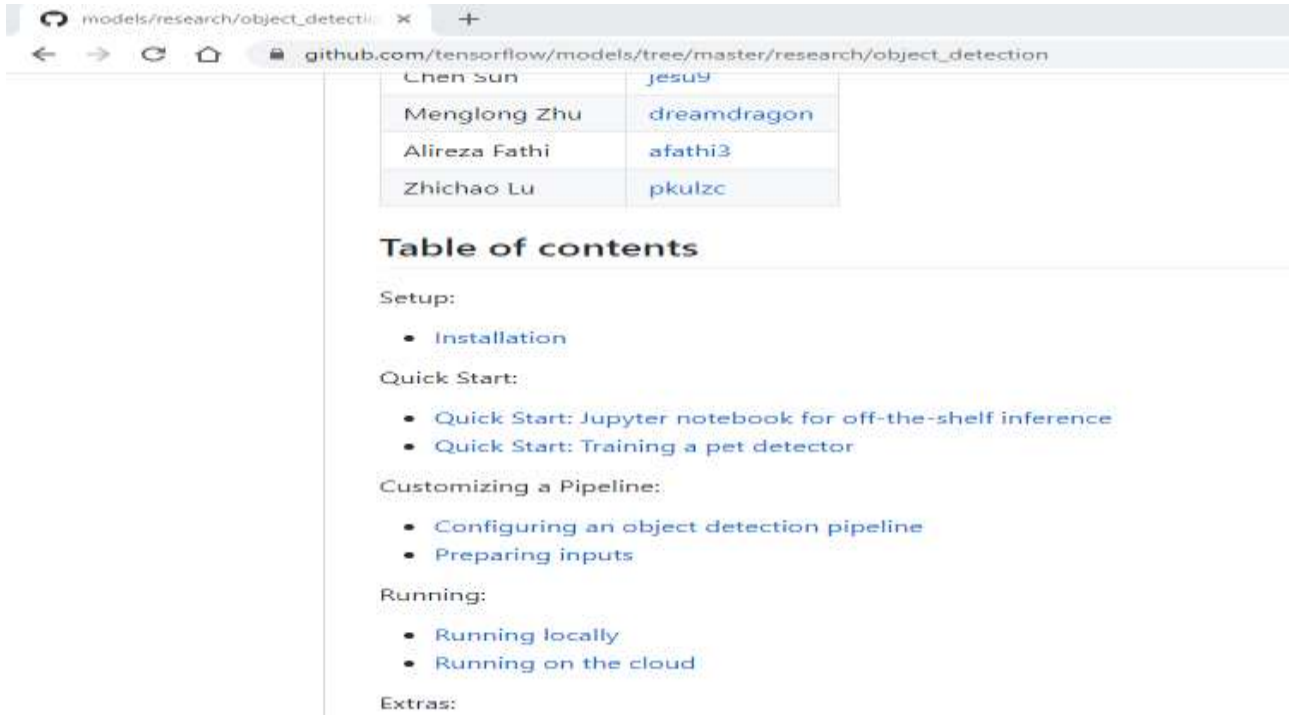
1-Google Tensorflow object detection yazıyoruz.

The screenshot shows a Google search page for 'tensorflow object detection'. The search bar contains the text 'tensorflow object detection'. Below the search bar, there are tabs for 'Tüm', 'Videolar', 'Görseller', 'Haberler', 'Akıllı', 'Daha fazla', 'Ayarlar', and 'Araçlar'. The search results show approximately 3,400,000 results found in 0.47 seconds. The first result is 'tensorflow object detection için bulunan akademik makaleler' (academic articles found for tensorflow object detection). Below this, there are links to 'github.com / object_detection', 'tensorflow/models', and 'Object detection | TensorFlow Lite'. The 'Object detection | TensorFlow Lite' link is highlighted, and it shows a date of '31 Mar 2020' and a description: 'An object detection model is trained to detect the presence and location of multiple classes of objects. For example, a model might be trained to detect faces, cars, and airplanes. What is object detection? Performance Benchmarks Starter model'.

Tensorflowün git_up ı açmamız gerekiyor



2-Git-up da linux için nasıl kurul yapılacağı detaylı anlatılıyor bunu windows a uyarlamak için bazı değişiklikler yapılacak bunun için [intsallation](#) linkini tıklayyoruz



3-Protobuf veri transfer protokolüdür.windows uyarlamak için object detection indirilmelidir.

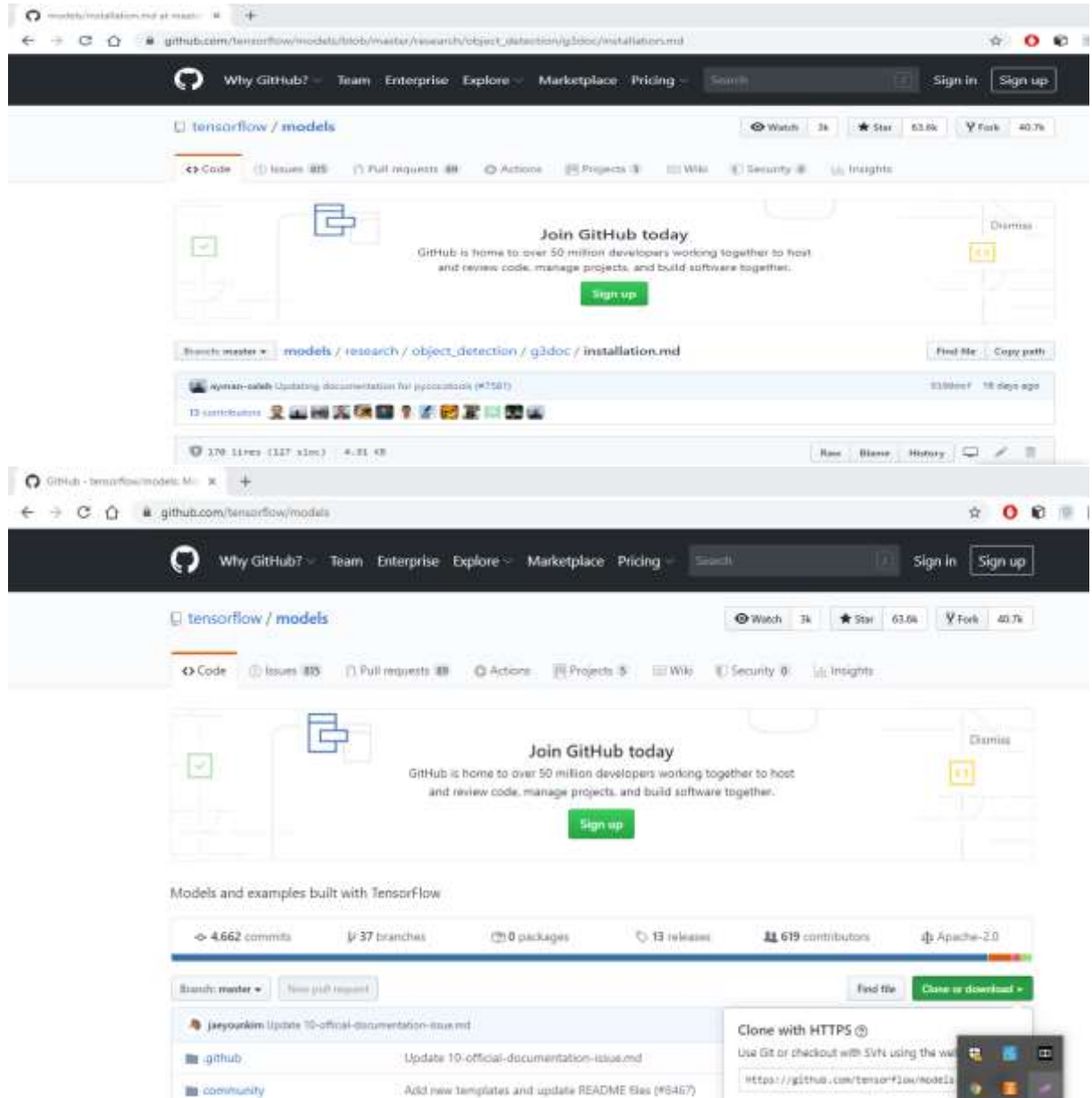
Protobuf Compilation

The Tensorflow Object Detection API uses Protobufs to configure model and training parameters. Before the framework can be used, the Protobuf libraries must be compiled. This should be done by running the following command from the [tensorflow/models/research/](#) directory:

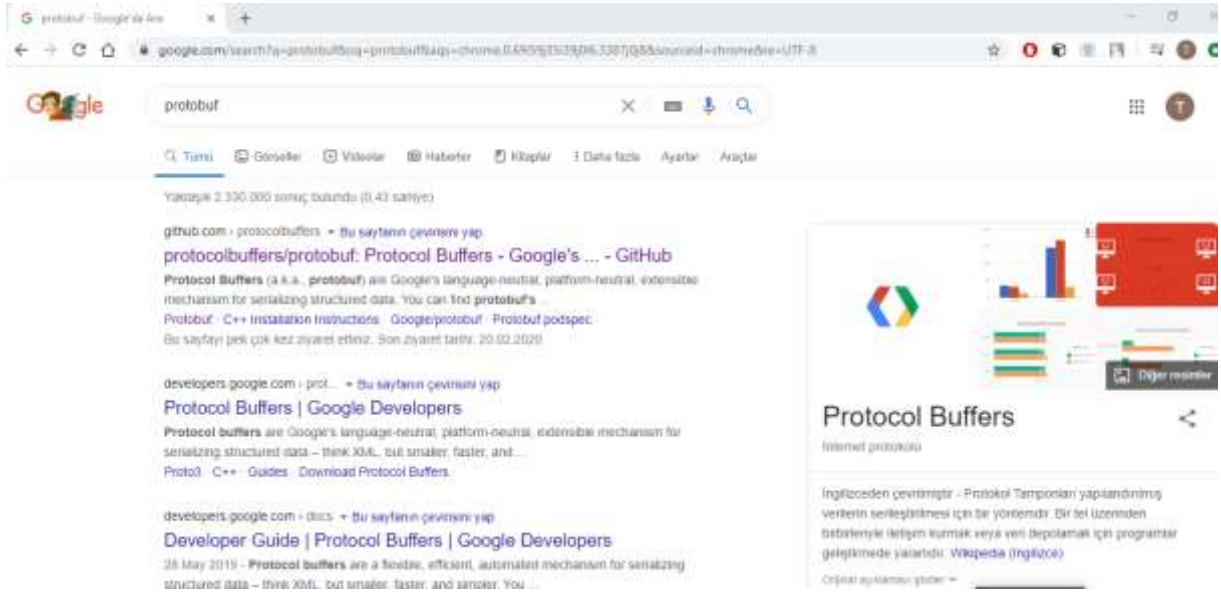
```
# From tensorflow/models/research/  
protoc object_detection/protos/*.proto --python_out=.
```

Note: If you're getting errors while compiling, you might be using an incompatible protobuf compiler. If that's the case, use the following manual installation

Bunun için models ' a giriyoruz.Models'ta "clone on download" tıklanır 400 MB boyutundaki model zip dosyasını indirmemiz gerekir



5-Bununla beraber protobuf indirmemiz gerekiyor. Bunun için google protobuf yazıyoruz.



Github linkine giriyoruz. Releases linkini tıklıyoruz.tıkladığımızda farklı sürümler çıkacaktır.

Protocol Compiler Installation

The protocol compiler is written in C++. If you are using C++, please follow the [C++ Installation Instructions](#) to install protoc along with the C++ runtime.

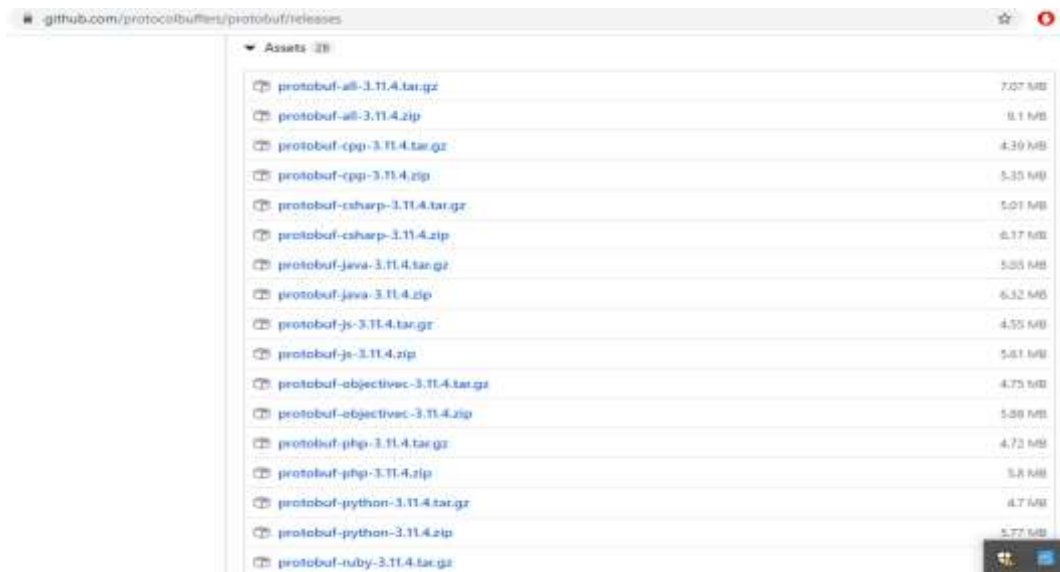
For non-C++ users, the simplest way to install the protocol compiler is to download a pre-built binary from our release page:

<https://github.com/protocolbuffers/protobuf/releases>

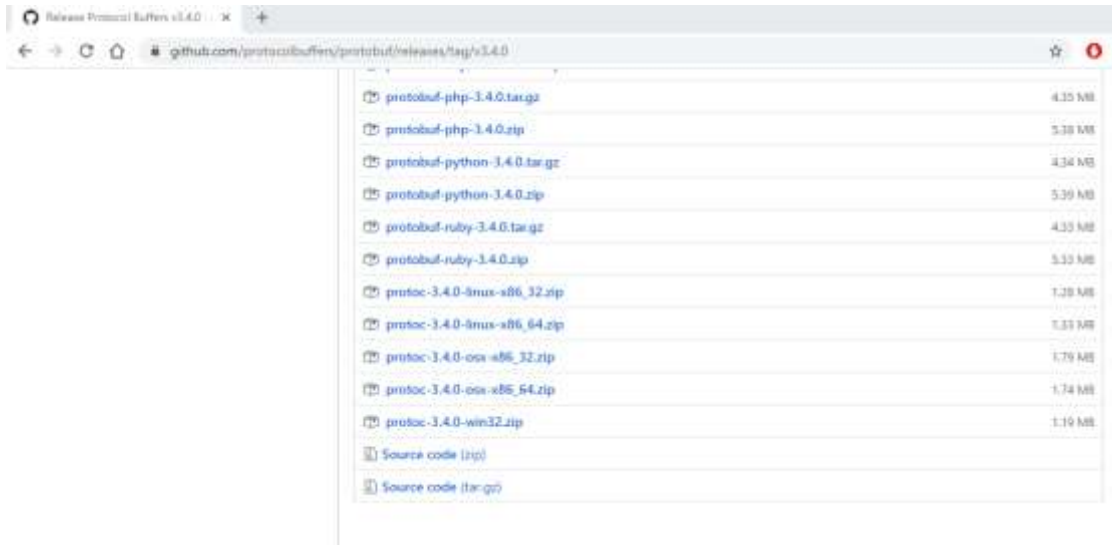
In the downloads section of each release, you can find pre-built binaries in zip packages: protoc-\$VERSION-\$PLATFORM.zip. It contains the protoc binary as well as a set of standard .proto files distributed along with protobuf.

If you are looking for an old version that is not available in the release page, check out the maven repo here:

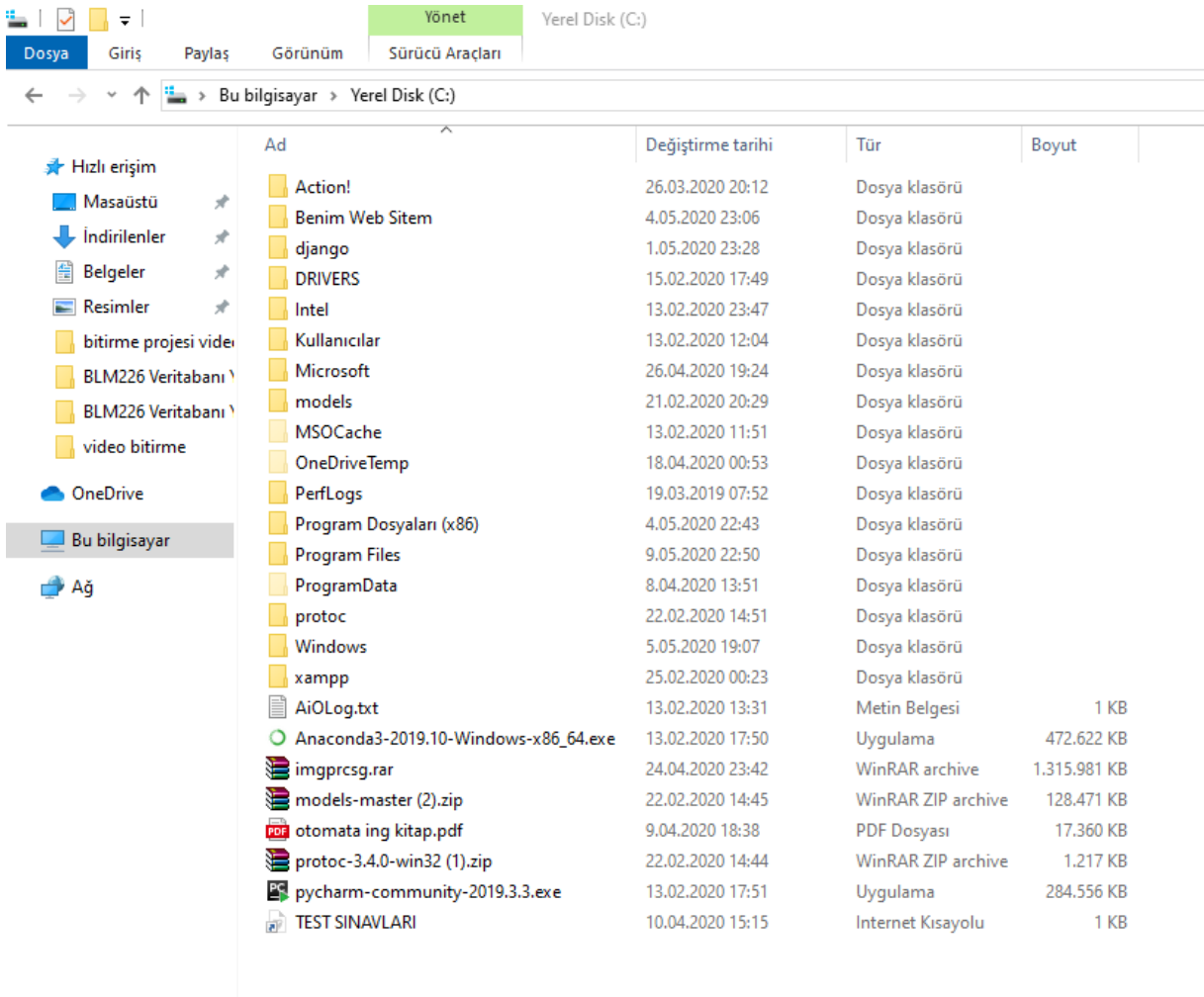
son sürüm bende çalışmadı daha eski bir sürüm denedim tek tek denenebilir. 3.4.0 sürümü bende çalıştı onu indirdim.



3.4.0 in windows için olanını seçip indirdim.



İkisinde indirdiğimde C sürücüne çıkardım.

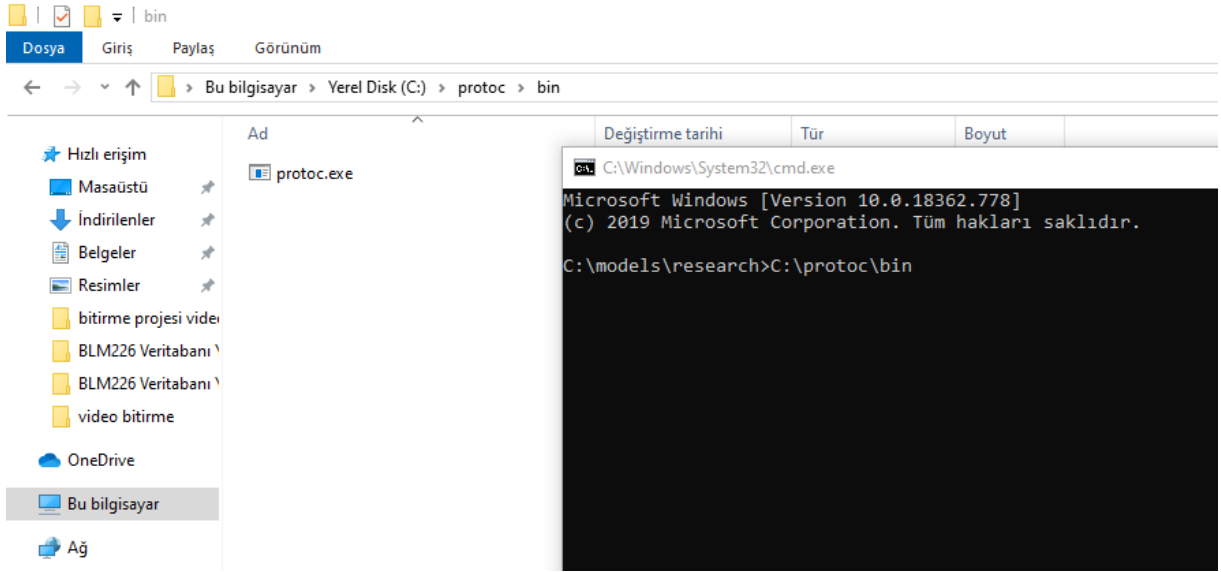


Buraya Kurulumları
yapacağız.

45 öge

Bu yolu kopyalamamız gerekiyor.

1 öge



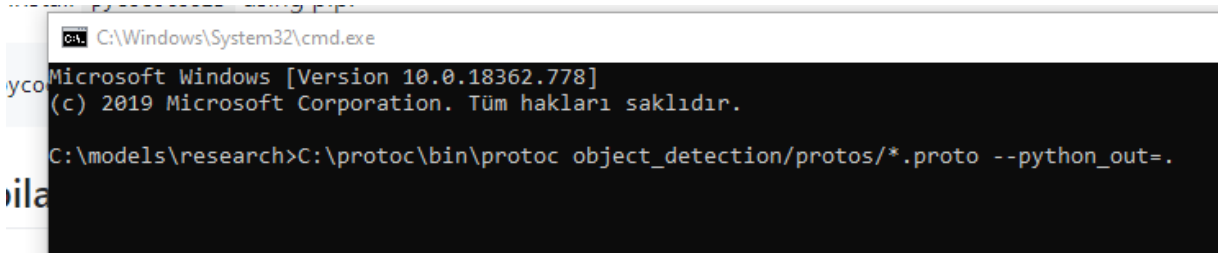
6-Artık tekrar instalation sayfasına gidip probuf Complation da kodu almam gerekiyor.

Protobuf Compilation

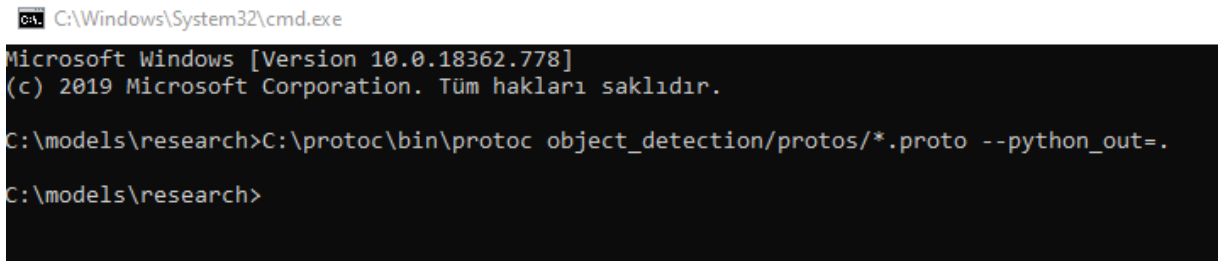
The Tensorflow Object Detection API uses Protobufs to configure model and training parameters. Before the framework can be used, the Protobuf libraries must be compiled. This should be done by running the following command from the [tensorflow/models/research/](#) directory:

```
# From tensorflow/models/research/  
protoc object_detection/protos/*.proto --python_out=.
```

Note: If you're getting errors while compiling, you might be using an incompatible protobuf compiler. If that's the case, use the following manual installation



Bu kodu çalıştırdığımızda tüm proto dosyaları derlenecektir. Aşadaki gibi çıkması gerekir



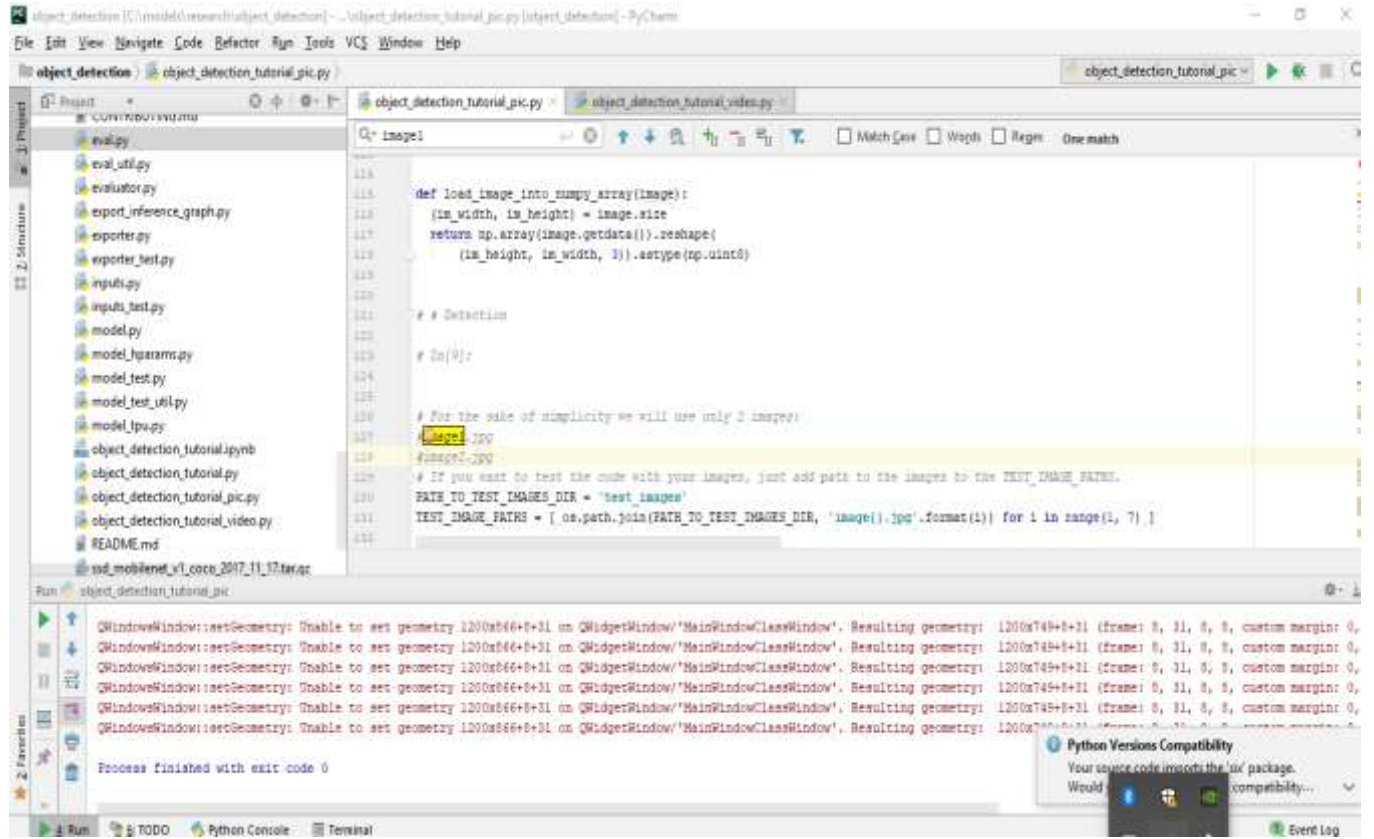
Yanlış yapıldığında bu kısımda hata verir. Bu kısımda yolu düzgün belirtmediğim için çok defa hata aldım. Eğer CMD ekranına bunlar yazıldığında `C:\models\research>` bu sonucu elde ediyorsanız hatasız tanıttığınızı gösterir.

Bu aşamaya kadar Tensorflow object detection kurduk ve derledik. Şimdi kendi resimlerimizde , videolarımızda ya da webcam de nasıl nesne tanıma oluyor onu göreceğiz. Object_detection_tutorial.py python dosyasından 2 tane oluşturduk isimler verdik

1-resimleri : “Object_detection_tutorial_pic.py “

2-videoları : “Object_detection_tutorial_video.py “

Fotografaları ve videoları bu dosyalarda kodları yazılıp üzerinde değişiklikler yapılacaktır.



Projede özellikle kod kısmı üzerinde çok duracağız çünkü kodu değiştirerek gerek algoritmaları değiştirerek gerek resim ve videolarda yapılacak değişiklikler uyarılama açısından yeterlidir.çünkü videolar ard arda gelen resimler olduğu için kod için yazdığım koddan video için belli yerlerde değişiklikler yeterli olacaktır. Tekrar kodu yazmak yerine resim ve videoda kodda nasıl bir değişiklik yapılması gerektiğini resimlerle göstereceğim.

RESİM İÇİN KOD KISMINDA ÖNEMLİ NOKTALAR

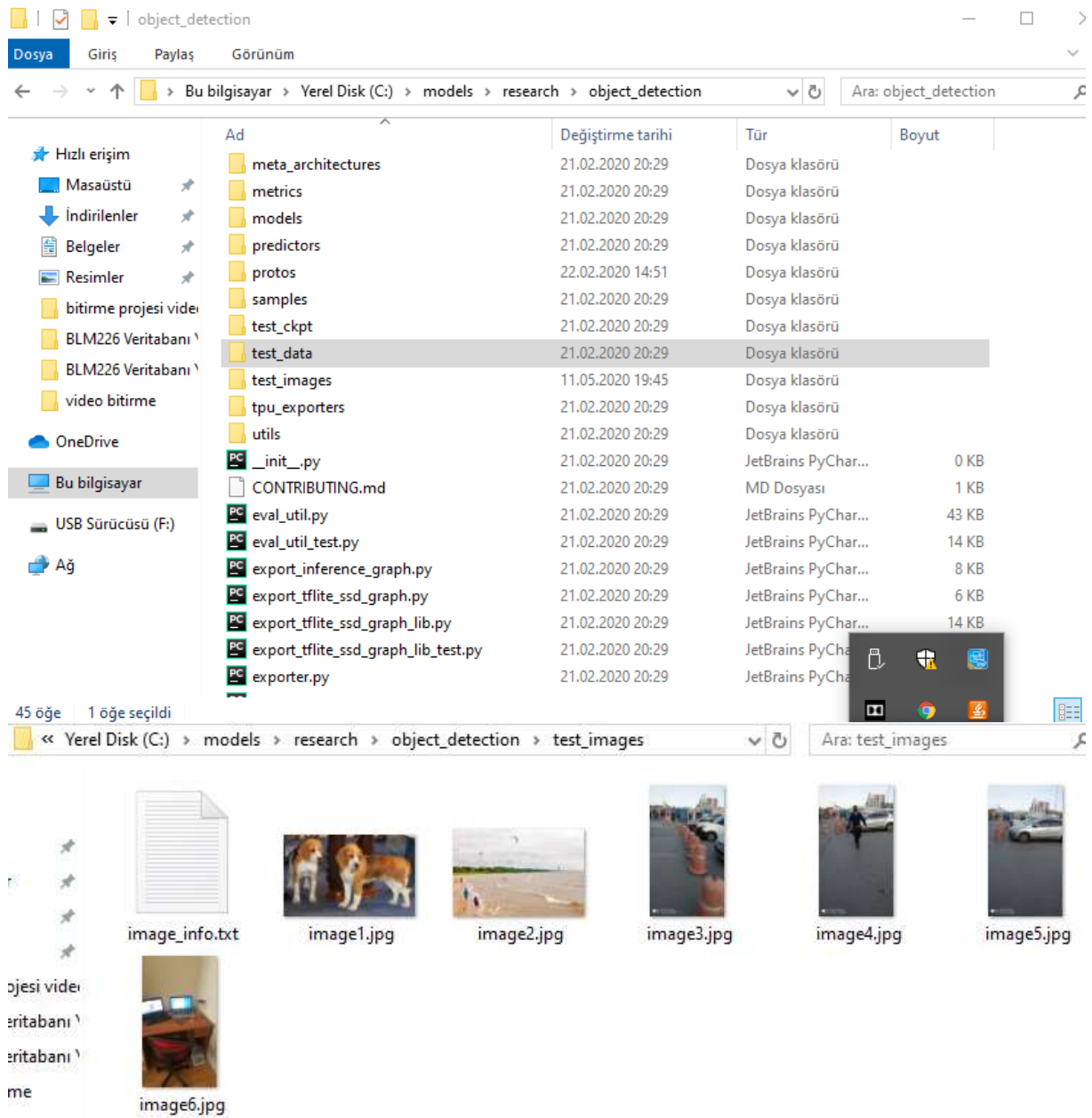
```
TEST_IMAGE_PATHS = [ os.path.join(PATH_TO_TEST_IMAGES_DIR,  
'image{}.jpg'.format(i)) for i in range(1, 7) ]
```

```
IMAGE_SIZE = (12, 8)
```

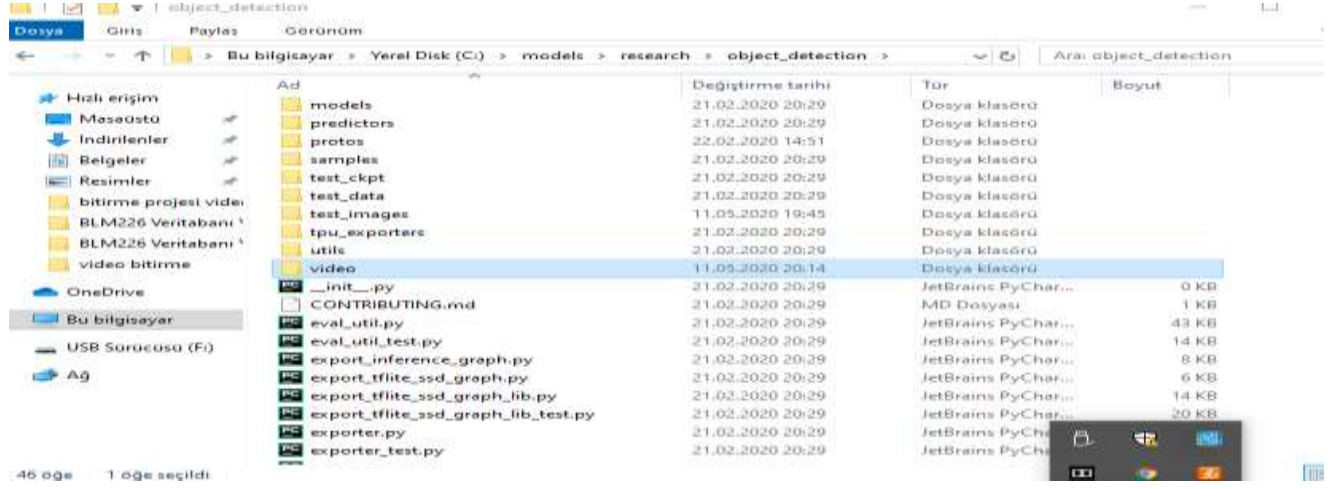
```
"PATH_TO_TEST_IMAGES_DIR = 'test_images'"
```

```
TEST_IMAGE_PATHS = [ os.path.join(PATH_TO_TEST_IMAGES_DIR,  
'image{}.jpg'.format(i)) for i in range(1, 7) ]"
```

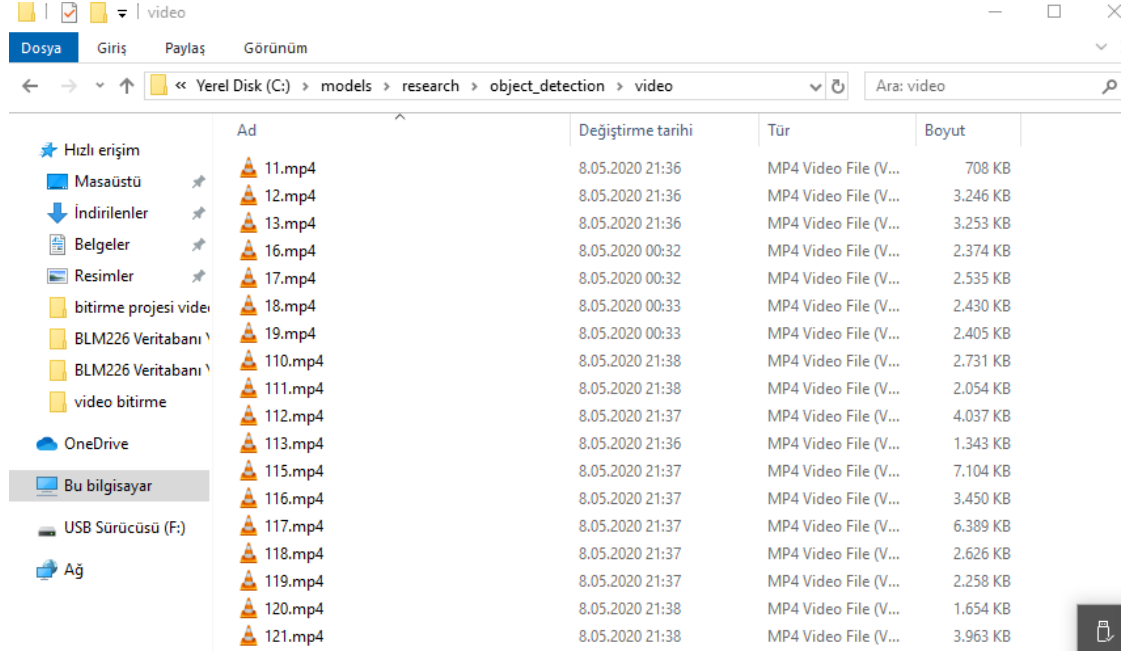
***kodda yukarda sarı renkli gösterdiğim kısmı açıklamam gerekiyor: Kodun bu kısmında test images kısmından klasörleri alıyor burada 6 resim klasörde olduğu için 1' den 7'ye kadar döngü kurmalıyız bu döngüye sırayla resim alıyor. Bu resimler üzerinden nesne tanıma yapılacaktır. Bu klasöre istediğimiz kadar resim atıp ona göre kodda döngüyü düzenleyeceğiz.



VIDEO İÇİN KOD KISMINDA ÖNEMLİ NOKTALAR



Yukardaki gösterilen yerde kendi videolarımız için video isminde bir klasör oluşturduk. İçine nesne tanıma yapmak istediğimiz videoları attık. Şimdi resim için oluşturduğumuz kodda bazı değişiklikler yapıp videoya uyarlayacağız.



```
import imageio
```

```
reader = imageio.get_reader('video/1.mp4')
```

```
fps = reader.get_meta_data()['fps']
```

```
writer = imageio.get_writer('video/output.mp4', fps=fps)
```



```

MODEL_NAME = 'ssd_mobilenet_v1_coco_2017_11_17'
MODEL_FILE = MODEL_NAME + '.tar.gz'
DOWNLOAD_BASE = 'http://download.tensorflow.org/models/object_detection/'

for i, image_np in enumerate(reader):
    # Expand dimensions since the model expects images to have shape: [1,
    None, None, 3]
    image_np_expanded = np.expand_dims(image_np, axis=0)
    # Actual detection.
    output_dict = run_inference_for_single_image(image_np,
    detection_graph)
    # Visualization of the results of a detection.
    vis_util.visualize_boxes_and_labels_on_image_array(
        image_np,
        output_dict['detection_boxes'],
        output_dict['detection_classes'],
        output_dict['detection_scores'],
        category_index,
        instance_masks=output_dict.get('detection_masks'),
        use_normalized_coordinates=True,
        line_thickness=8)
    writer.append_data(image_np)
    print(i)
writer.close()

```

// for i, image_np in enumerate(reader): Video resimlerden oluşur bu döngü videodaki resimler için bir döngü oluşturur. i ise döngünün kaçınıcı framede olduğunu gösterir. writer.append_data(image_np) bu kısım ile videodan bir frame alıp üzerinde nesne tanıma yapıp döngüye girmesini sağlıyoruz

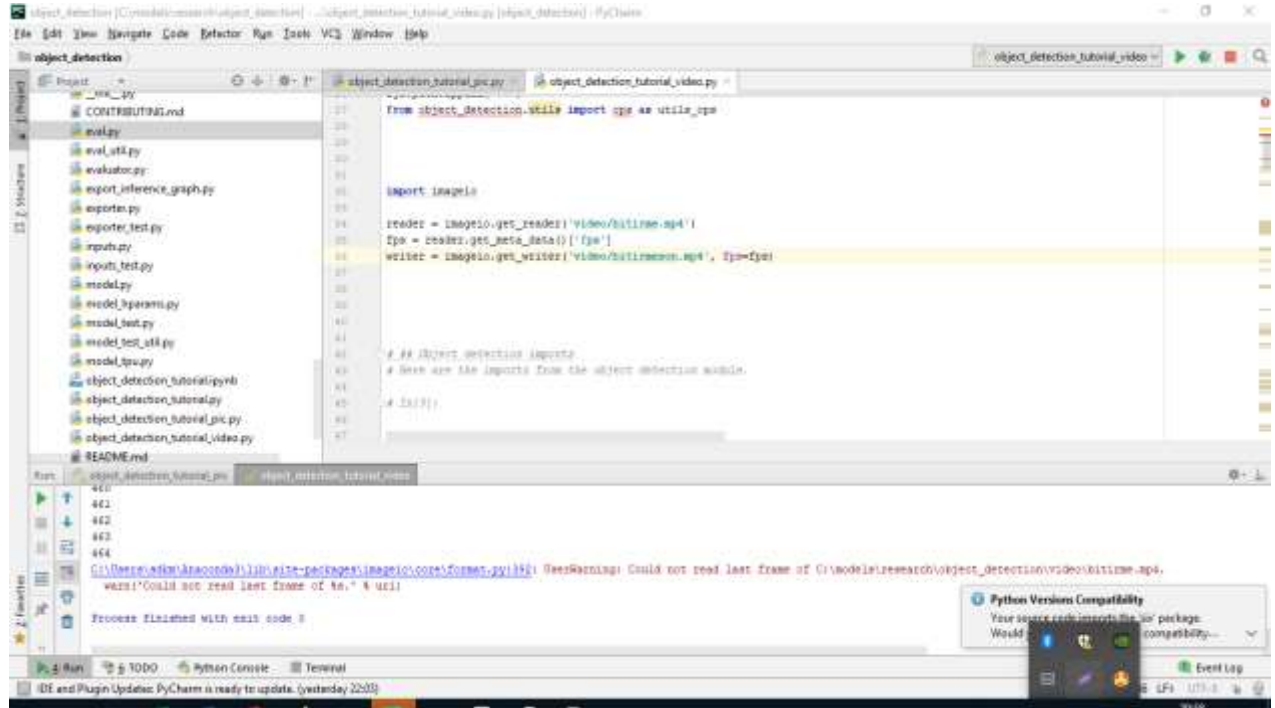
// önce klasörün ismini sonra reader = imageio.get_reader('video/1.mp4') önce klasörün daha sonra fps alıyoruz amacımız input videosuyla output videosunun aynı olmasını sağlamak. fps = reader.get_meta_data()['fps'] daha sonra output yanı çıktı veren videoyu yazdırmak

writer = imageio.get_writer('video/output.mp4', fps=fps) fps=fps hem input hem output uzunluklarının aynı olmasını sağlar.

with tf.Session(graph=detection_graph) as sess: Burada ssd modeli graph e yükleniyor ve nesne tanıma kodunu çalıştırabiliriz.

Bu bilgisayar > Yerel Disk (C:) > models > research > object_detection > video					
	Ad	Tarih	Tür	Boyut	Uzunluk
Hızlı erişim	output.mp4	25.04.2020 18:14	MP4 Video	7.983 KB	00:00:13
Masaüstü	original.mp4	2.03.2016 09:48	MP4 Video	6.353 KB	00:00:13
İndirilenler	odam.mp4	8.05.2020 22:51	MP4 Video	1.322 KB	00:00:21
Belgeler	oda.mp4	8.05.2020 22:49	MP4 Video	5.346 KB	00:00:21
Resimler	bitirme.mp4	8.05.2020 21:38	MP4 Video	3.963 KB	00:00:15
django-2020	benson1.mp4	9.05.2020 12:36	MP4 Video	1.190 KB	00:00:22
object_detection	benson.mp4	9.05.2020 12:35	MP4 Video	5.544 KB	00:00:22
test_images	ben12.mp4	8.05.2020 23:03	MP4 Video	1.130 KB	00:00:17
video	ben11.mp4	8.05.2020 23:01	MP4 Video	737 KB	00:00:13
Bu bilgisayar	ben2.mp4	8.05.2020 22:58	MP4 Video	4.469 KB	00:00:17
Ağ	ben1.mp4	8.05.2020 22:57	MP4 Video	3.410 KB	00:00:13
	a21.mp4	8.05.2020 22:38	MP4 Video	1.680 KB	00:00:15
	a20.mp4	8.05.2020 22:33	MP4 Video	417 KB	00:00:06
	a16.mp4	8.05.2020 22:27	MP4 Video	1.755 KB	00:00:13
	a13.mp4	8.05.2020 22:23	MP4 Video	537 KB	00:00:04
	a11.mp4	8.05.2020 22:20	MP4 Video	554 KB	00:00:07
	a10.mp4	8.05.2020 22:19	MP4 Video	1.371 KB	00:00:10
	a9.mp4	8.05.2020 22:16	MP4 Video	1.026 KB	00:00:09

Kodu çalıştırdığımızda



Videoyu 464 frame bölerek ağdan geçirildi ve nesne tanıma gerçekleştirildi.

Klasörde görüldüğü gibi “bitirmeson” diye output(çıkış) yani nesne tespiti yapan bir video oluştu.

Bu bilgisayar > Yerel Disk (C:) > models > research > object_detection > video					
	Ad	Tarih	Tür	Boyut	Uzunluk
Hızlı erişim	original.mp4	2.03.2016 09:48	MP4 Video	6.353 KB	00:00:13
Masaüstü	odam.mp4	8.05.2020 22:51	MP4 Video	1.322 KB	00:00:21
İndirilenler	oda.mp4	8.05.2020 22:49	MP4 Video	5.346 KB	00:00:21
Belgeler	bitirmeson.mp4	11.05.2020 20:57	MP4 Video	1.680 KB	00:00:15
Resimler	bitirme.mp4	8.05.2020 21:38	MP4 Video	3.963 KB	00:00:15

Yukardaki işlemler bittiğinde aşağıda örnek video ve resimler nesne tespiti sonucu oluşan örneklerdir, daha kaliteli sonuçlar(yüksek tespit oranlar) elde etmek için resmin kalitesi, resmin boyutu , kullanılan bilgisayarın CPU hızı ve kullanılan algoritma önemlidir. Bu projede CPU ile çalıştığım için SSD Algoritmasını kullandım kodun

```
# What model to download.
```

```
MODEL_NAME = 'ssd_mobilenet_v1_coco_2017_11_17'
```

Yukarıda gösterilen kısma SSD yerine R-CNN, Fast R-CNN, Faster R-CNN , Mask R-CNN gibi çok daha hızlı ve daha verimli algoritmalar kullanılabilir. Benim Özellikle SSD yani yazılanlar arasında en verimsiz algoritmayı kullanmamın sebebi bilgisayarındaki CPU'nun işlem gücünün zayıf olmasından kaynaklıdır. R-CNN ni kullandığımda bilgisayarında donmalar oldu ve nesne tespiti çok uzun sürdü. Başka bir bilgisayarda bu algoritmalar aynı kod üzerinden denenebilir.

Eğer projeyi CPU yerine GPU üzerinden yapsaydık ekranlardaki paralel işlemlerden dolayı hem bütün algoritmaları çalıştırırdı hem yüksek tespit gerçekleştirirdi.

Ekran kartını kullanarak benzer projeler yapmak için NVIDIA 3.5 üzeri ekran kartları ile çok daha başarılı ve hızlı sonuçlar elde edilebilir.

