TBL 322 Makine Öğrenmesine Giriş

Plaka Algılama ve Tanıma

Kocaeli Üniversitesi
Teknoloji Fakültesi
Bilişim Sistemleri Mühendisliği
Barış Adıgüzel - 181307059

GİRİŞ

Günümüz modern toplumunda çokça kullanılan "yapay zeka" kavramı hayatımızın her noktasında karşımıza çıkmaktadır. İlerleyen teknoloji ile yapay zekadan beklentiler ve yapacağı işlemlerin beklentisi de bir hayli ile artmıştır. Bundan yaklaşık 20-30 yıl önce ye kadar ofis işlerinde gönderilen elektronik postalar, şirket içi haberleşmeler, muhasebe işlemleri, personel kayıt formları gibi çoğu ofis işi zaten bilgi sistemlerinin erken versiyonu ile tanışmış olup ofis işlerine entegrasyonları sağlanmıştı. İlerleyen teknoloji bize insanların talep ve ihtiyaçlarının hızlı bir şekilde değişerek arttığını ve bilgi sistemlerinin bile bu hızla artan talep ve ihtiyaçları yetişemediğini gözler önüne serdi. Artık bazı işlemlerin manuelden otomatik işlemlere geçerek firmalara zamandan tasarruf etmelerini sağlamalıdır.

Yapay zekanın, gelişen teknoloji ile birlikte hızla gelişmesi ile birlikte çoğu şirketler kendi bilgi sistemlerinde de kullanmaya başladı. Bu sayede, kendi bilgi sistemlerine yapay zekanın kendisini de dahil eden şirketlerde görüldü ki; manuel, sıkıcı ve tekrar gerektiren işlemlerin insan gücüne gerek kalmadan hızlı ve etkili bir şekilde yapıldığı gözlemlendi. Sağlık sektöründe hastaların normal insanla tespit edilemeyecek hastalıkların teşhisine, taşıma ve lojistik sektöründe depolarda kargo taşıyan, dolaşan yardımcı robotlara gibi çoğu sektörde gözlemleyebiliriz. Özellikle yolcu ve taşımacılık sektöründe de sık sık kullanıldığını görmekteyiz. Trafik kontrolü ve araç sahibi tespiti, her ülkede büyük bir sorun haline geldi. Bazen trafik kurallarını ihlal eden ve çok hızlı giden araç sahibini tespit etmek zorlaşıyor. Bu nedenle bu tür kişilerin yakalanması ve cezalandırılması mümkün değildir çünkü trafik polisleri aracın hızından dolayı hareket halindeki araçtan araç numarasını alamayabilir.

Benim yapacağım proje, araçların plakalarını otomatik olarak tespit etmek, plakalarını okumasını sağlayan bir derin öğrenme modeli geliştirmek. Projemi gerçekleştirebilmek için bir veri seti oluşturup, verilerin temizlemesini yapıp derin öğrenme modeline uygun olacak bir şekilde veri seti oluşturmak olucaktır. Bundan dolayı ilk olarak, kendi veri setimi oluşturabilmek için internetten bir veya birkaç siteden araç plakası içeren resimlerini çekmek olucaktır.

I KULLANDIĞIM PROGRAMLAMA DİLİ VE TEKNOLOJİLER

A. Python

Veri setini oluşturmak için internetten bana gerekli olan resimleri indirmem ve kendi bilgisayarımda depolamam gerekiyor. Python programlama dili bu konuda en kolay ve verimli olarak benim işimi görücektir. Python, birçok amaç için kullanılabilen, dinamik bir söz dizime sahip, yaygın olarak kullanılan, nesne yönelimli ve üst düzey bir programlama dilidir. Guido van Rossum tarafından 20 Şubat 1991'de piyasaya sürülen Python, günümüzde halen geliştiriliyor ve özellikle yapay zeka çalışmalarında çokça kullanılıyor.

Python, üst düzey ve genel amaçlı olarak kullanılan bir programlama dilidir. Açık kaynaklı bir dil olan Python; çoğunlukla Ruby, JavaScript ve Scheme ile karşılaştırılır. Python'u diğer programlama dillerinden ayıran en önemli özelliği, kullanımının basit olması, yeni başlayanlara kolay öğretilebilmesi, herhangi bir uygulamaya yerleştirilebilmesi ve Mac, Windows ve Linux dahil olmak üzere mevcut tüm işletim sistemlerinde çalışabilmesidir. Aynı zamanda bir programcının kullanabileceği en güçlü dillerden biridir ve sırasıyla JavaScript ve C ++ 'dan yaklaşık üç ila beş kat daha hızlı kodlama yapmanıza imkan tanımaktadır. Buna rağmen Python, diğer programlama dillerine göre biraz daha yavaştır fakat Python'ın standart kütüphanesi, derlenmiş C kütüphanelerinden oluştuğu için hız bakımından eksiğini kapatabiliyor. Python'ın en son çıkmış olan sürümü PEP 619 ile birlikte çıkan 3.10 sürümüdür. 3.10 sürümü ile gelen en dikkat çekici özellik PEP 634 – Structural Pattern Matching (Yapısal Model Eşleştirme) diğer programlama dillerinde olan switch-case yapısında match-case olarak Python'a entegre edilmiştir.

Projemde veri setini oluşturmak için Python'ı kullanarak basitçe yapmam gerekenleri listelemem gerekirse:

- Veri setini oluşturmak için uygun verileri barındıran bir web sitesi bulmak.

- Bulduğum web sitesinde benim için gerekli olan resimleri barındıran sitenin kaynak kodunu çekmek.
- Siteden çektiğim kaynak kodundan, resimlerin barındırıldığı linkleri ayırarak, linkler üzerinden GET isteği atarak resimleri indirmek.

Bunları yapabilmem için Python'ın standart kütüphanesi yeterli olamayacaktır. Bu yüzden bana gerekli olacak 3. parti kütüphaneleri indirmeliyim. (HTTP istekleri, web scraping.. vb.)

Gerekli modüller/kütüphaneler:

A.1 Requests

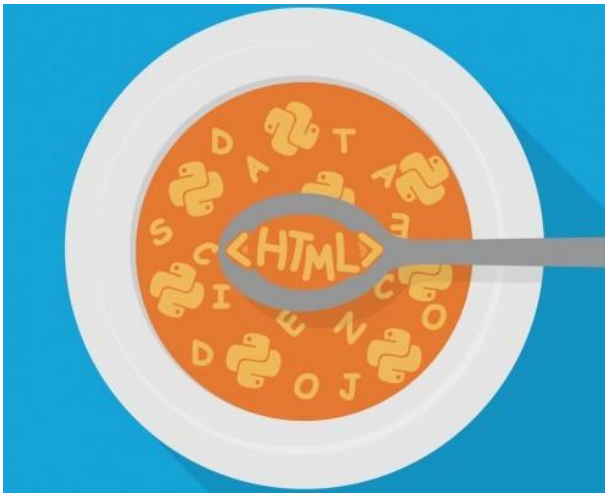
Python'ın internetteki verilerle etkileşimini sağlayan modüllerden biri olan Requests modülü, HTTP istekleri ile projeniz ve webdeki kaynak verileriniz arasında iletişim sağlar. Bu modül, Python ile beraber default olarak gelmez. Bu modülün temel yapı taşı olan en çok kullanılan HTTP istekleri aşağıda yer almaktadır:

- **GET:** En çok kullanılan istek türüdür. Web'deki verileri çekmenize yarar.
- **POST:** Web'deki sunucuya veri göndermenizi sağlar.
- **PUT:** Mevcut bilgileri değiştirmenize yarar.
- **DELETE:** Bilgileri siler.

Buna ek olarak yapılan istekler HTTP durum kodlarını döndürür. HTTP durum kodları aşağıdaki şekildedir:

- **200** : İstek başarıyla gerçekleşti.
- **204** : İşlem gerçekleştirildi, içerik bulunamadı.
- **301** : İstenen sayfanın taşındığını döndürür ve o adresi yönlendirir.
- **400** : Yanlış istek biçimi
- **401** : İşlem için yetkiniz yok.
- **403** : Erişim reddedildi.
- **404** : Belirtilen kaynak sunucuda yer almıyor.
- **500** : Sunucu hatası.

A.2 BeautifulSoup



Python'da HTML ve XML dosyalarını işlemek için, genelde acemi kullanıcılar, düzenli ifadeleri kullanır. Ancak düzenli ifadeler hem hata yapması kolay bir alandır, hem de bu iş için verimli değildir. Diğer yandan, BeautifulSoup gibi bu iş için tasarlanmış, performanslı ve kullanımı kolay bir kütüphanedir.

Beautiful Soup Python için bir HTML ve XML ayrıştırıcısıdır (parser). Beautiful Soup kütüphanesi kullanışlı olmasını şu özelliklerine borçludur:

- Beautiful Soup kötü girdi verseniz bile bozulmaz. Neredeyse orjinal belgenizle aynı anlama gelen bir ayrıştırma ağacı (parse tree) döndürür. Bu özellik çoğu zaman gereken bilgiyi almanız için yeterlidir.
- Beautiful Soup bir ayrıştırma ağacında kolayca gezinme (traversing), arama ve düzenleme yapmanıza olanak sağlayan birçok metot ve Python vari deyimler sağlar: her uygulama için baştan HTML veya XML ayrıştırıcı yazmanıza gerek kalmaz.
- Beautiful Soup gelen belgeleri Unicode'a, giden belgeleri de UTF-8'e kendiliğinden çevirir. Kodlamalarla uğraşmanıza gerek kalmaz.

A.3 Selenium

Selenium, Web uygulamalarının testini otomatikleştirmek için kullanılan açık kaynaklı, esnek yapıya sahip olan bir kütüphanedir. Selenium test komut dosyaları Java, Python, C# ve daha pek çok farklı programlama dillerinde yazılabilir. Bu test komut dosyaları Chrome, Safari, Firefox, Opera gibi çeşitli tarayıcılarda çalışabilir ve ayrıca Windows, Mac OS, Linux, Solaris gibi çeşitli platformları da destekler.

Selenium ana amacı, test amaçlı web uygulamalarını otomatikleştirmek içindir, ancak kesinlikle bununla sınırlı değildir. Seçtiğiniz bir tarayıcıyı açmanıza ve bir insanın yapacağı gibi görevleri gerçekleştirmenize olanak tanır.

Örneğin:

- Butonlara tıklama,
- Formlara girme,
- Web sayfalarda belirli bilgileri arama,
- Gerekli verileri çekme,

Selenium gibi kütüphaneler ile veri kazıma, çoğu sitenin hizmet şartlarına aykırı bir davranıştır. Çok sık veya kötü niyetli şekilde veri çekerseniz, IP adresiniz o web sayfasından geçici veya kalıcı bir şekilde yasaklanabilir.

Şuan ki kullandığım sürümü 4.10.0 sürümüdür.

A.4 Os Modülü

Python, standart dosya işleme işlemleri için OS (Operating Systems) modülünden yardım alır. Python'ın içinde standart bir kütüphanedir. Veri kazımından sonra verilerin belirlenen dizinlere kaydetmek için kullanacağız.

A.5 Time Modülü

Python, standart tarih ve zaman işlemleri için time modülünden yardım alır. Python'ın içinde standart bir kütüphanedir. GET isteklerinden sonra belirlir bir süre aralığında script'imizi durdurmak için kullanacağız.

B. PyCharm

PyCharm, en popüler Python IDE'lerinden biridir. Çapraz platform uygulaması olarak mevcut olan PyCharm, Linux, macOS ve Windows platformlarıyla uyumludur. Günümüzde en çok kullanılan python IDE'lerinden biridir. Hem Python 2 (2.7) hem de Python 3 (3.5 ve üstü) sürümleri için destek sağlar. Pycharm ilk kez 2010 yılının Şubat ayında halka yayınlandı.

Pycharm'ın 2 tane sürümü vardır:

- Community
- Professional

Community: ücretsizdir ve standart python geliştiricileri için tercih edilebilir.

Professional: Hem Bilimsel hem de Web Python geliştirme için. HTML, JS ve SQL desteği sunar ve daha profosyenel projeler yapmak isteyenlerin tercihlerinden biridir.

Şuan ki kullandığım sürümü 2021.3.3 sürümüdür.

II VERİ SETİNİN OLUŞTURULMASI

Bana gerekli olan kütüphaneleri de belirledikten sonra web scraping (web kazıması) için kendime bu verileri çekebileceğim ve işleyebileceğim bir site bulmam gerekiyor.

Kendime araba resimlerini çekmek için üç tane web sitesi belirledim. Bunlar:

- sahibinden.com
- Letgo
- arabam.com

Bu sitelerin hepsi ikinci veya sıfır araba satışları yapan e-ticaret siteleridir. Amacım bu sitelerdeki ilanlardaki araba resimlerini çekerek benim belirlediğim bir dizine kaydetmek.

Akış diyagramım bu şekilde ilerleyecek:

- İlk önce belirlediğim web sitelerdeki ilanların linklerini ayrıştıracağım.
- Ayrıştırdığım bu linkleri ya yerel bir yerde metin dosyasının içine kaydedeceğim ya da programımda bir diziye kaydeceğim.
- Ayrıştırdığım bu linkleri Selenium'dan üreteceğim WebDriver nesnesi ile ilanın sayfasının üzerinde gezerek, ilanda resmin linkini alacağım.
- Son olarak aldığım resmin linkini Requests kütüphanesini kullanarak GET isteği ile alıp belirlediğim dizinin içine kaydeceğim.

Öncellikle bunları yapmadan önce, kazıma yapacağım sitenin yapısını ayrıntılı olarak incelemem lazım. Örneğin sahibinden.com sitesindeki ilanlar nasıl sıralanmış.

Şekil 2.1 – sahibinden.com sitesinin HTML kaynak kodu

Sahibinden.com sitesindeki ilanlar, yukarıdaki bir şekilde sayfa kaynağını incelediğimizde görüyoruz ki HTML kodunda <table> etiketinin içinde <tr> etiketlerinde bulunuyor.

BeautifulSoup kütüphanesini kullanarak bulunduğu etiketlerin attribute(özellik) kısımlarını kullanarak ilanların linklerini çekebilirim.

```
searchURL = "https://www.sahibinden.com/kategori-vitrin?viewType=List&pagingSize=50&category=3530"
response = requests.get(searchURL, headers=usr_agent)
html = response.text
soup = BeautifulSoup(html, 'html.parser')
results = soup.find_all('td', {'class': "searchResultsLargeThumbnail"}, recursive=True)
```

Şekil 2.2 – sahibinden.com sitesinde belirlediğimiz etiketlerin BeautfitulSoup kütüphanesini kullanarak ayrımı

Yukarıdaki Şekil 2.2'de görüldüğü üzere önce Requests modülünün get() fonksiyonu ile web sitesine bir istekte bulunuyoruz. İstekten sonra cevap olarak bize bir Response objesi döndürmektedir. Response objesinin içeriğini unicode olarak BeautifulSoup kütüphanesinin kurucu fonksiyonunu çağırarak 'html.parser' ayrıştırıcı modunda bir nesne türetiyoruz. Bu nesne ile birlikte belirlediğimiz etiketlerin içindeki linklerin bulunduğu ebeveyn etiketi çekmek için BeautifulSoup nesnesinin find_all() kütüphanesinde çekilmesi istenen etiket ve özelliklerini belirterek bir web kazıma gerçekleştiriyoruz.

```
# region 1st Step - Fetching the links of the adverts from www.arabam.com

link_list_of_adverts = []
for i in range(1, 51):

    # Search URL to iterate through the pages of the website
    search_url = f"https://www.arabam.com/ikinci-el?take=50&page={i}"

    # GET request to the search URL
    response = requests.get(search_url, headers=usr_agent)

    # Getting the HTML of the response
    html = response.text

    # Creating a BeautifulSoup object
    soup = BeautifulSoup(html, 'html.parser')

    # Finding all tr elements with the class 'listing-list-item pr should-hover bg-white'
    results = soup.find_all('tr', {'class': "listing-list-item pr should-hover bg-white"}, recursive=True)

    # Parsing href attribute of the a tag in the tr element and adding it to the list
    link_list_of_adverts += ["https://www.arabam.com" + i.find('a')['href'] for i in results]# -> For List
    # link_list_of_adverts.update(["https://www.arabam.com" + i.find('a')['href'] for i in results]) # -> For Set

    # Progress bar to show the progress of how many adverts are being fetched
    print(f"Page {i} is done - {len(link_list_of_adverts)} adverts found")

#
advert_count = len(link_list_of_adverts)
print(f"All pages are done - {advert_count} adverts found")

# endregion
```

Şekil 2.3 – arabam.com sitesinden 1-50. sayfalar arasındaki ilanların linkini BeautifulSoup kütüphanesi alınması

Burada ilanların linki sabit olmak üzere sayfa numaralarından iterasyon yapabiliyoruz. GET isteğinde bulunarak Response objesi olarak ilgili web sayfamızın kaynak kodunu ayırabiliyoruz. Kaynak kodunu ayırdıktan sonra BeautifulSoup kütüphanesi kullanarak ayrıştırıyoruz. HTML kodundaki ilanın linkini içeren ilgili etiket ve özelliklerin bilgisini vererek ilanların linkini ayırarak bir listeye ekliyoruz. En sonunda ise bir progress bar (durum çubuğu) vasıtası ile ne kadar ilanın linkini topladığımızı gösteriyorum. Bittikten sonra toplam alınan linklerin sayısını bastırıyorum.

```
Page 44 is done - 2177 adverts found
Page 45 is done - 2227 adverts found
Page 46 is done - 2277 adverts found
Page 47 is done - 2327 adverts found
Page 48 is done - 2377 adverts found
Page 49 is done - 2427 adverts found
Page 50 is done - 2477 adverts found
All pages are done - 2477 adverts found
Press Enter to continue...
```

Şekil 2.4 – İlanların bulunduğu sayfalardaki linklerin çekildiğine ait bir durum göstergesi

Böylelikle akış diyagramında belirlediğim ilk ve ikinci aşamayı da gerçekleştirmiş oluyorum.

Üçüncü aşama için Selenium kütüphanesini kullanarak otomatik bizim yerimize web tarayıcıda gezmesi için bir nesne üretiyoruz. Bundan önce web tarayıcımızın, Selenium kütüphanesi tarafından kullanılabilmesi için WebDriver indirmemiz gerekiyor. Tarayıcınız için indirdiğiniz web driver'ı projede yolunu vererek bir WebDriver nesnesi oluşturabiliriz.

```
operaDriverPath = "C:\\Users\\Barış Adıgüzel\\Documents\\Python Scripts\\Drivers\\operadriver.exe"
```

Şekil 2.5 – Opera Web Tarayıcısının WebDriver'ının yolu

```
# Stackoverflow = https://stackoverflow.com/questions/31055124/drive-opera-with-selenium-python

options = webdriver.ChromeOptions()
# Solution for 'dict' has no attribute 'click' -> https://github.com/operasoftware/operachromiumdriver/issues/96
options.add_experimental_option('w3c', True)
# Path to the Opera executable file
options.binary_location = r"C:\Users\Barış Adıgüzel\AppData\Local\Programs\Opera\opera.exe"

# Creating an instance of Service class
webdriver_service = service.Service(operaDriverPath)

# Starting the service
webdriver_service.start()

# webdriver_service.service_url => http://localhost:57099 -> http://<IP_ADRESS>:<PORT>
driver = webdriver.Remote(webdriver_service.service_url, options=options)
```

Şekil 2.6 – Web Driver'ın Başlatılması

WebDriver'ımızı başlattıktan sonra artık istediğimiz şekilde driver'ı kullanabiliriz. Daha önceden ilanların linkini ayrıştırıp eklediğimiz dizinin üzerinden, Requests modülündeki gibi Driver'ı kullanarak linklerin üzerinden bir gezinti yapabiliriz.

Selenium'ı kullanarak gezinti yapacağımız ilanda belli başlı işlemler yapmak istiyoruz. Örneğin, ilan yüklendikten sonra hemen resmin linkini BeautifulSoup ile kazıyarak almak istiyoruz. Fakat, sonradan geriye dönüp baktığımızda resmin ön izlenmiş kısmını koparıp onu yüklediğimizi farkediyoruz. Bundan dolayı bir resme tıklayarak, yüksek çözünürlüklü versiyonun yüklenip onun linkini almayı hedefliyoruz. Selenium'da belli başlı etiketleri bularak onlar üzerinden işlem yapmamız için birkaç tane yöntem var fakat en iyi yöntem Xpath yöntemidir.

A. Xpath Nedir?

```
# -------
# XPath (XML Path Language) is an expression language designed to support the query or transformation of XML documents.
# It was defined by the World Wide Web Consortium (W3C)
# -------
# //*[@id="islrg"]/div[1]/div[1] -> Xpath location of 1st image on Google
# //*[@id="islrg"]/div[1]/div[2] -> Xpath location of 2nd image on Google
# //*[@id="islrg"]/div[1]/div[3] -> Xpath location of 3rd image on Google
# //*[@id="islrg"]/div[1]/div[4] -> Xpath location of 4th image on Google
```

XPath, XML etiketleri içerisinde yer alan verilere ulaşmak için kullanılan ve W3C tarafından geliştirilen bir standarttır.

XPath sadece XML içerisinde bulunan verilere erişmek için bir standart belirler.

XPath, Java, C#, C++, Python, PHP, JavaScript vb. programlama dilleri içerisinde yer alan hazır işlev veya programlama diline dahil edilebilen ek bileşenler ile birlikte kullanılır.

```
# Creating a directory to store the images
dataset_dir = r"D:\dataset\arabam_images"
create_a_directory(dataset_dir)
```

Şekil 2.7 – Verilerin kaydedileceği yol ve klasörün belirtilmesi

Selenium ile ilanların üzerinden geçmeden önce resimleri kaydeceğimiz yolu ve klasörü belirtiyoruz.

```
# region 2nd Step - Browsing the adverts and downloading the images

real_count = 0
for count, link in enumerate(link_list_of_adverts):

    # Is this file already downloaded?
    # if os.path.exists(f"{dataset_dir}/arabam_{real_count + 1}.jpg"):
    #     print(f"{count + 1}/{advert_count} - {round((real_count / advert_count) * 100, 1)}% - Already downloaded")
    #     real_count += 1
    #     continue

    try:
        driver.set_page_load_timeout(0.8)
        driver.get(link)
    except:
        # Timeout (0.8 seconds)
        print(f"{count+1}th link failed")
        continue

    # page_html = driver.page_source
    # soup = BeautifulSoup(page_html, 'html.parser')
    # images = soup.find_all('div', {'class': "w100 slider-main-gallery pr cf"})

    # Click on the first image preview
    try:
        driver.find_element(by=By.XPATH, value='//*[@id="js-hook-for-main-slider"]/div/div/a[1]/img').click()
    except selenium.common.exceptions.NoSuchElementException:
        print("No preview image found")
        continue
    except selenium.common.exceptions.TimeoutException:
        print("Timeout")
        continue
```

Şekil 2.8 – İlan linklerinin Selenium WebDriver'ı ile gidilerek vasıta resimlerinin üzerine tıklanması

İlan link linklerini, başlattığımız Selenium WebDriver'ı ile tarayıcı açarak ilgili ilan linkine gidiyoruz.

WebDriver'da XPath yolunu belirlerken ilgili web sayfasının kaynak kodunu inceleyip oradan XPath bilgisini de alabiliriz. Bir yerden sonra çoğu etiketin sıra sıra geldiğini böylelikle görebiliyoruz. Örneğin, <table> etiketindeki <tr> etiketlerinin XPath yolları ardışık olarak ilerlemekte olduğunu görebiliriz. Burada ise ben sadece açılan ilanın ilk resminin üzerine tıklayarak alma işlemini yaptım. Yaklaşık olarak 40.000 civarı ilanı olan letgo sitesinde zaman zaman ayırarak (2000-3500 ilan) Selenium ile gezinme de bulundum.

```
    try:
        # Fetch the image source
        element = driver.find_element(by=By.XPATH, value='/html/body/div[7]/div/div[1]/div[1]/div/img')
        # Getting the image source
        image_link = element.get_attribute('src')
    except:
        print("No image source found")
        continue

    # Downloading the image
    try:
        download_image(image_link, dataset_dir, f"arabam_{real_count + 1}")
        real_count += 1
        temp = count + 1
        # Progress bar
        print(f"Downloaded... {count + 1}/{advert_count} - {round((temp / advert_count) * 100, 1)}% - URL "
              f"{image_link}")
    except selenium.common.exceptions.TimeoutException:
        print("Timeout Exception - Downloading image")
        continue
    except:
        print("Image download failed")
        continue

    # time.sleep(.25)

# endregion
```

Şekil 2.9 - Selenium ile üzerine tıkladığımız resmin kaynağının linkini alarak indirilmesi ve olası hataların kontrolü

Üzerine tıkladığımız resmin kaynağını ayrıştırarak eğer herhangi bir sorun ile karşılaşılmazsa (İlan resminin olmaması, atılan isteğin uzun sürmesi vb.) daha önceden Şekil 2.7'de oluşturduğumuz klasörün yolu, indirilecek olan resmin kaynağı ve dosya ismi ile indirilmesi için bir fonksiyona parametre olarak gönderiyoruz.

```
def download_image(url, dir_name, file_name):
    """Downloading images from the given url and saving them in the given directory name

    :param str url: URL of the image
    :param str dir_name: directory name where the image will be saved
    :param str file_name: file name of the image
    :return: None
    """
    file_name = str(file_name) + '.jpg'
    response = requests.get(url)
    # status code 200 means OK status for HTTP response
    if response.status_code == 200:
        # os.path.join(dir_name, file_name) -> output string => dir_name/num.jpg, wb = write binary
        with open(os.path.join(dir_name, file_name), 'wb') as file:
            # write the image data to the file
            file.write(response.content)
```

Şekil 2.10 – İndirilecek resmin requests.get() metodu ile istekte bulunulup gelen içeriğin yazdırılması

İndirmek için oluşturulan fonksiyonda requests.get() metodundan bir GET isteği oluşturuyoruz. Eğer isteğimiz başarılı olursa (HTTP protokolünde 200 OK durum kodu) klasörde gönderilen dosya ismi ile gelen yanıtın (içerik bayt olarak döndürülüyor) içeriğini "write binary" modunda bir dosya oluşturup içeriğini yazdırarak verimizi kaydetmiş oluyoruz.

Şekil 2.11 – arabam.com sitesinden indirilen resimler, kırık linkler ve zaman aşımına uğraşmış linkler

Bu durumda terminalde bir durum göstergesi ayarlayarak nerelerde kırık link, nerelerde zaman aşımına uğradığını ve indirilen resimlerin linklerini görerek oluşturduğum veri setinin durumunu daha kolay bir şekilde takip edebiliyorum.

Sadece sahibinden.com sitesinden resimleri çekemedim. Çok hızlı bir şekilde istek attığımdan dolayı o yüzden iki e-ticaret sitesindeki ilanlardan resimlerini çekerek veri setinin açığını kapatmaya çalıştım. Sitelerde değişse bile kodlar neredeyse aynı sadece kazıyacağım HTML etiketleri ve Xpath yolları değişiklik göstermektedir.

III KARŞILAŞTIĞIM ZORLUKLAR VE BULDUĞUM ÇÖZÜMLER

Öncelikle web scraping keyifli olduğu gibi zaman zaman hatalarını çözmeniz gerektiği için yorucu bir iştir. Çoğu site web scraping kısmını belli bir seviyede izin verdiği gibi bazılarında ise direk hizmet şartlarına aykırı olduğu için belli başlı sınırlamalar koymaktadır. Veri setini oluştururken farkettiğim, zaman zaman bulmakta zorlandığım sonradan çözdüğüm veya çözemediğim hataları sırası ile bahsedeceğim.

A. *HTTP İstek ve Yanıtların Kontrolü*

Kesinlikle eğer HTTP istekleri/yanıtları ile uğraşılıyor ise sık sık karşılaşacağımız programdaki hatalardan birisidir. Yapacağınız işte kesinlikle ve özellikle web kısmında try-exception kullanımı sizi bir adım daha ileri taşıyacaktır. Web'de kazıma yaparken herhangi bir hatadan dolayı programınız durması sizin için vaktinizi önemli ölçüde harcayacaktır. Bir kırık linkin, 2-3 saatlik işinizi tekrara sokması kaynak ve zaman açısından çok ölümcül bir duruma sokabilir. En basit olarak ayrıştırılan linklerin çalışıp, çalışmadığından emin olun. Çalışmayacağı durumlarını da gözeterek algoritmanızı ona göre tekrardan şekillendirin.

B. *Resimlerin Düşük Çözünürlüklü Olması*

Bir başka karşılaştığım hatalardan bir tanesi linklerini aldığım resimlerin düşük çözünürlüklü olmasıydı. Bunun önüne geçebilmek için programda özellikle GET isteğinin yapıldığı, sayfanın yüklendiği kısımlara programı durdurmaya karar verdim. Sonuçta HTTP trafiği, programdan bağımsız olarak asenkron şekilde hareket etmekte ve bant genişliğine bağlı olduğundan dolayı bazı sayfalar hızlı yüklenirken bazı sayfaların yavaş yüklenmesi benim için sorun teşkil ediyordu. Bundan dolayı programda web sayfasına istek atılan yerlerin altında programı durdurdum ve resme tıklayıp belirli bir zaman aralığında dondurarak işlemlerime öyle devam ettim.

C. *Zaman Aşımı*

Selenium ile ilanların linkine giderken, bazı ilanların hızlı yüklendiğinden dolayı programımın hata vermeden donduğunu farkettim. Hata vermediğinden dolayı da bu sorunu bulmam bir epey zor oldu. Birkaç arkadaşıma danıştıktan sonra sorunun bazı ilanlardaki resimlerin açılmasının çok uzun sürdüğünü ve bundan dolayı programımın uzun süreler donduğunu farkettim. Bunun içinde bir try-exception komut bloğu koyarak sorunun önüne geçtim.

```
try:
    driver.find_element(by=By.XPATH, value='//*[@id="js-hook-for-main-slider"]/div/div/a[1]/img').click()
except selenium.common.exceptions.NoSuchElementException:
    print("No preview image found")
    continue
except selenium.common.exceptions.TimeoutException:
    print("Timeout")
    continue
```

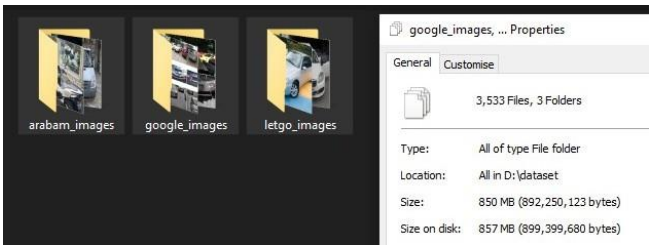
D. *Bot Engelleyici Siteler*

Selenium'ı kullanarak GET isteği atarak dolaştığımız ilanlarda bazı sitelerde uyarı, bazı sitelerde ise geçici IP yasağı yedim. Siteye atılan isteklerin yoğunluğu ve aralarındaki sürenin kısa olmasından dolayı ve hepsinin tek bir IP adresten geldiğini de gördüğü için sahibinden.com sitesinden herhangi bir veri çekemedim. Bunun önüne geçebilmek için belli zaman ve saat aralıklarında web sayfasına istekte bulunmak bir çözüm olabilir. Fakat bu çözümün veri çekimini uzatacağı için başka sitelere yöneldim.

E. *Verilerin Temizlenmesi*

Özellikle indirdiğim resimlerde alakasız çok fazla unsur bulunmaktaydı(kokpit iç görünüm, plakasız araçlar, yarım gözüken plakalar, çözünürlük vs.) fakat en azından alakasız verilerin silinebilmesi için otomatik elimine edilmesi ile alakalı herhangi bir kaynaktan birşey bulamadığım için manuel elle verileri silmek zorunda kaldım.

IV VERİSETİ SONUCU

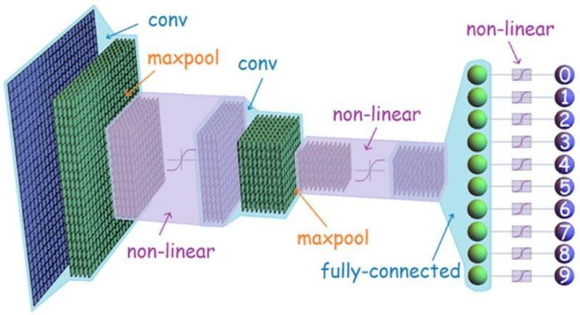


Şekil 4 – Topladığım resimlerden oluşturduğum veri setim

Bir tanesi Google Images olmak üzere, toplam internette 3 ayrı web sitesinden veri kazıma işleminde bulundum. 3553 tane ham veriden (raw data) oluşan bir veri setim var. Veri setimi daha da büyütmek adına aynı ve farklı sitelerin ilanlarının üzerinden gezinerek resimleri indiriyorum. Gereksiz ve düzensiz verileri temizledikten sonra geriye kalan veriler ile derin öğrenme yapabileceğimi düşünüyorum.

V KULLANILAN MODELLER

A. *Convolutional Neural Network (CNN) (Evrişimli Sinir Ağları)*



Şekil 5.1 – Basit bir CNN Mimarisi

CNN genellikle görüntü işlemede kullanılan ve girdi olarak görselleri alan bir derin öğrenme algoritmasıdır. Farklı operasyonlarla görsellerdeki featureları (özellikleri) yakalayan ve onları sınıflandıran bu algoritma farklı katmanlardan oluşmaktadır. Convolutional Layer, Pooling ve Fully Connected olan bu katmanlardan geçen görsel, farklı işlemlere tabii tutularak derin öğrenme modeline girecek kıvama gelir. CNN modelleri oluştururken, unstructural (düzensiz) veri ile uğraştığımızdan klasik makine öğrenmesi algoritmalarına kıyasla veri ön işleme kısmında çok uğraşmamaktayız.

Kendi projemde ise nesne tespiti yerine burada resim sınıflandırması probleminde yoğunlaştım. 200 tane valid_plate kategorisi, 200 tane non_valid_plate kategorisi olarak ayırdığım toplamda 400 resimden oluşan bir veriseti kendime oluşturdum.

valid_plate : Resimde araç plakası net bir şekilde gözükmekte olan resimleri ifade eder.

non_valid_plate : Resimde araç plakası olmayan, plakası kapatılmış olan resimleri ifade eder.

```
import numpy as np
import pandas as pd
import os
from sklearn.metrics import classification_report
from sklearn.utils import shuffle
import seaborn as sns; sns.set(font_scale=1.4)
import matplotlib.pyplot as plt
import cv2
import tensorflow as tf
```

Şekil 5.2 – Veriseti ve modeli için gerekli kütüphaneleri

Öncelikle verisetimizi projemize dahil edip model inşa etmeden önce gerekli kütüphanelerimizi ekliyoruz.

Verisetimi dahil etmeden önce etiketleri için bir ölçeklendirme yapıyorum. Bütün resimlerin 300x300 olucak şekilde ayarlamalarını yaptıktan sonra eğitim ve test klasörlerimden resimleri OpenCV ile okuyup tek tek karşısına gelen etiketlerini bir listenin içine ekliyorum.

```
DIRECTORY = r'/content/drive/MyDrive/datasets/dataset_cnn'
CATEGORY = ['train', 'test']

output = []

for category in CATEGORY:
    path = os.path.join(DIRECTORY, category)
    print(f'Current working directory : {path}')
    images = []
    labels = []

    print(f"Loading {category} images...")

    for folder in os.listdir(path):
        label = class_names_label[folder]

        # Iterate through each image in our folder
        for file in os.listdir(os.path.join(path, folder)):

            # Get the path name of the image
            img_path = os.path.join(os.path.join(path, folder), file)

            # Open and resize the img
            image = cv2.imread(img_path)
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            image = cv2.resize(image, IMAGE_SIZE)

            # Append the image and its corresponding label to the output
            images.append(image)
            labels.append(label)

    images = np.array(images, dtype='float32')
    labels = np.array(labels, dtype='int32')

    output.append((images, labels))
```

```
Current working directory : /content/drive/MyDrive/datasets/dataset_cnn/train
Loading train images...
Current working directory : /content/drive/MyDrive/datasets/dataset_cnn/test
Loading test images...
```

Şekil 5.3 – Verilerin eğitim ve test klasörlerinden alınarak etiketlendirmenin yapılması

Bu işlemlerden sonra verisetini eğitim ve test olarak bölüyorum. Böldüğüm verileri sonradan matplotlib yardımı ile bir çizelgede eğitim verilerine bakıyorum.

Şekil 5.4 – Eğitim verileri ile etiketlerini matplotlib kütüphanesi ile gösterimi

Burada görüldüğü üzere valid_plate etiketli olan resimlerde plakanın var olduğunu, non_valid_plate etiketli olan resimlerde ise plakanın olmadığı veya hiç görülmediğini görmüş oluyoruz.

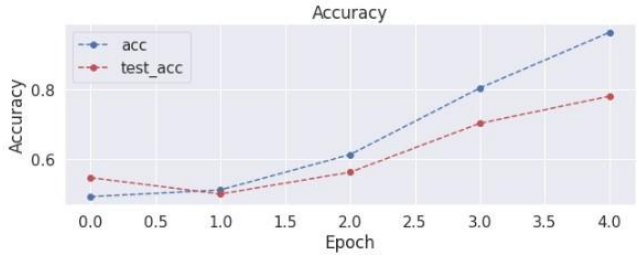
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 298, 298, 32)      896

 conv2d_1 (Conv2D)           (None, 295, 295, 32)      16416

 max_pooling2d (MaxPooling2D  (None, 147, 147, 32)     0
 )

 conv2d_2 (Conv2D)           (None, 145, 145, 128)     36992

 max_pooling2d_1 (MaxPooling  (None, 72, 72, 128)      0
 2D)

 flatten (Flatten)           (None, 663552)            0

 dense (Dense)               (None, 128)               84934784

 dense_1 (Dense)             (None, 2)                 258

=================================================================
Total params: 84,989,346
Trainable params: 84,989,346
Non-trainable params: 0
_________________________________________________________________
```

Şekil 5.5 – Üstünde eğitim yapılacak modelimizin boyutları, eğitileceği parametreleri ile katmanları

Conv2D, başlı başına içinde de ayrı ayrı katmanları olan bir katmandır. Modelimizde 7 katman olduğunu görüyoruz. Çıkış katmanında ise sadece 2 çıkışın olduğunu görüyoruz. Bunun sebebi ise 2 tane sınıfımız olduğundan dolayıdır.

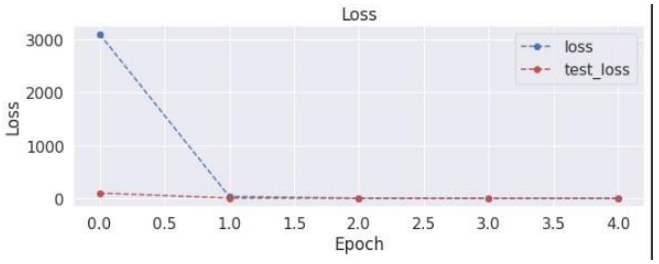
Modelimizin parametrelerini belirledikten sonra eğitime sokuyoruz. Bu model için batch boyutunu 32, epoch kısmını 5 olarak ayarladım. Batch boyutunu yüksek tuttuğum için yüksek miktarlarda sistem belleğini eğitim sırasında kullanmış oldu.

Eğitim sonrasında modelimizin eğitim sırasında ne kadar iyi bir şekilde eğitildiği ve eğitim sırasında kayıplarını görebilmek için metriklerini matplotlib ile çizdirdim.



Şekil 5.6 – 32 batch_size - 5 epoch eğitim verisi ile test verisinin doğruluk skoru

Burada da görüldüğü üzere model overfitting(aşırı öğrenme) uğradığını kolaylıkla görmekteyiz. Bunu düzeltmek için modelde katmanlarında düzenleştirme yapmak durumunda kaldım.



Şekil 5.7 – Eğitim verisi ile test verisinin zaman ile kayıpları

Daha sonra modelimi test verisi ile doğrulama yaptım. Doğrulama sonucu test doğruluğunun 77.5% olduğunu görmüş bulundum.

Tabi sadece loss fonksiyonu ve doğruluk skorları başlı başına bir modeli değerlendirmemizde yeterli metrikler değildir. Sınıflandırma metrikleri olan Precision, Recall, F1-Score gibi metrikleri de ölçümünü yaptım.

	precision	recall	f1-score	support
0	0.75	0.82	0.79	40
1	0.81	0.72	0.76	40
accuracy			0.78	80
macro avg	0.78	0.77	0.77	80
weighted avg	0.78	0.78	0.77	80

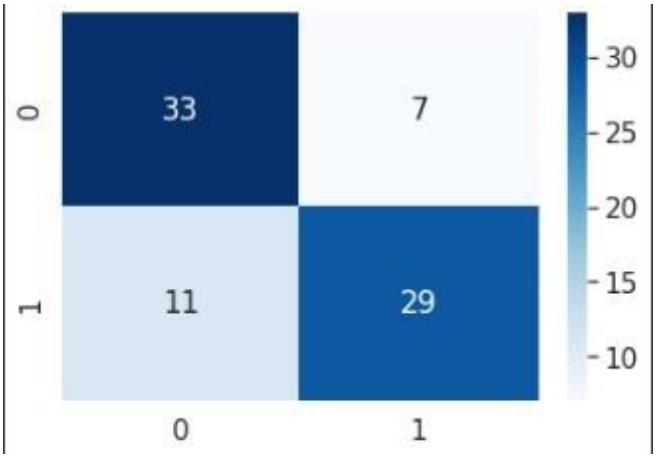
Şekil 5.8 – Sınıflandırma raporu

Buradaki 0 non_valid_plate sınıfını, 1 ise valid_plate sınıfnı temsil etmektedir. Buradaki metrikleri kısaca ifade etmek gerekirse:

Precision (Kesinlik) : Karmaşıklık matrisinde Pozitif olarak tahminlediğimiz değerlerin gerçekten kaç adedinin Pozitif olduğunu göstermektedir.

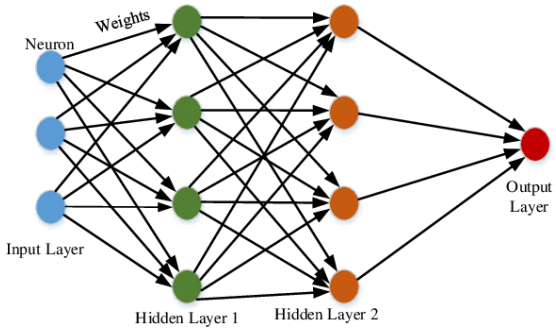
Recall (Duyarlılık) : Pozitif olarak tahmin etmemiz gereken işlemlerin ne kadarını Pozitif olarak tahmin ettiğimizi gösteren bir metriktir.

F1-Score (F1 Skoru) : Kesinlik (Precision) ve Duyarlılık (Recall) değerlerinin harmonik ortalamasını göstermektedir.



Şekil 5.9 – Modelimizin Karmaşıklık matrisi

B. Multi Layer Perceptron (MLP)
Çok Katmanlı Algılayıcılar



Çok Katmanlı Algılayıcılar (MLP), XOR Problemi'ni çözmek için yapılan çalışmalar sonucu ortaya çıkmıştır. MLP özellikle *sınıflandırma* ve *genelleme* yapma durumlarında etkin çalışır.

Birçok giriş için bir nöron yeterli olmayabilir. Paralel işlem yapan birden fazla nörona ihtiyaç duyulduğunda katman kavramı devreye girer. Görüldüğü üzere Single Perceptron Model'den farklı olarak arada gizli(hidden) katman bulunmaktadır. Giriş katmanı gelen verileri alarak ara katmana gönderir. Gelen bilgiler bir sonraki katmana aktarılırlar. Ara katman sayısı en az bir olmak üzere probleme göre değişir ve ihtiyaca göre ayarlanır. Her katmanın çıkışı bir sonraki katmanın girişi olmaktadır. Böylelikle çıkışa ulaşılmaktadır. Her işlem elemanı yani nöron bir sonraki katmanda bulunan bütün nöronlara bağlıdır. Ayrıca katmandaki nöron sayısı da probleme göre belirlenir. Çıkış katmanı önceki katmanlardan gelen verileri işleyerek ağın çıkışını belirler. Sistemin çıkış sayısı çıkış katmanında bulunan eleman sayısına eşittir.

İkinci modelimde yapay sinir ağı olan Çok Katmanlı Algılayıcıları seçtim. Bir önceki CNN derin öğrenme modelinde olduğu gibi tekrardan gerekli kütüphaneleri dahil projeme dahil edip sınıflarımı belirledim.

```
from sklearn.preprocessing import LabelEncoder
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score, recall_score, precision_score, f1_score
import keras
import tensorflow as tf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import cv2
import os


license_types = os.listdir('/content/drive/MyDrive/datasets/dataset')
print(license_types)

print(f"Types of classes found : {len(license_types)}")
```

```
['valid_plate', 'non_valid_plate']
Types of classes found : 2
```

Şekil 5.10 – Gerekli kütüphanelerin ve verisetimizin projeye dahil etmeden önce sınıfların belirlenmesi

Sınıflarımı CNN derin öğrenme algoritmasında yaptığım gibi valid_plate ve non_valid_plate şeklinde iki sınıfa böldüm. Verisetini oluşturmak için daha sonradan sınıflara ayırdığım klasörlerden resimlerin yolunu ve sınıf klasörlerini bir listeye ekleyerek bir Pandas DataFrame oluşturdum.

200 tane plakası olan resim ve 200 tane plakası olmayan resim olarak bir DataFrame oluşturduktan sonra resimleri OpenCV ile Numpy dizisine çevirerek her birini ilgili etiketi ile ayrı bir listede eklemelerini yaptım.

```
path = '/content/drive/MyDrive/datasets/dataset'

# Image size to be resized both for width and height
im_size = 300

images = []
labels = []

for item in license_types:
    # Get all the file names in the given directory
    data_path = path + '/' + str(item)
    filenames = [i for i in os.listdir(data_path)]

    for f in filenames:
        # Reading that image as array

        img = cv2.imread(data_path + '/' + f)
        img = cv2.resize(img, (im_size, im_size))       # Resizing 300x300
        images.append(img)
        labels.append(item)
```

Şekil 5.11 – Resimlerin OpenCV ile okunup tekrardan boyutlanıp listelere ayrılması

Bu işlemlerden sonra etiketleri Label Encoder ile her kategorik veriye bir sayısal değer atamasında bulundum.

Verilerimi artık model eğitimine hazırladıktan sonra test ve eğitim verilerine bölmeden önce sağlıklı bir eğitim olması adına karıştırdım ve verilerimi test ve eğitim veriseti olmak üzere ikiye böldüm.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 270000)            0

 dense (Dense)               (None, 256)               69120256

 dense_1 (Dense)             (None, 128)               32896

 dense_2 (Dense)             (None, 2)                 258

=================================================================
Total params: 69,153,410
Trainable params: 69,153,410
Non-trainable params: 0
_________________________________________________________________
```

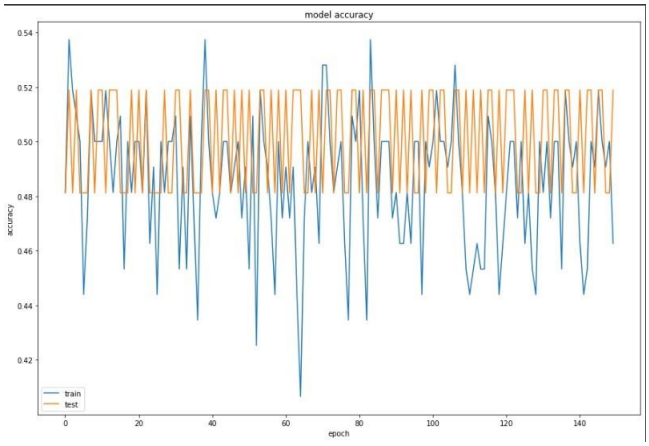
Şekil 5.12 – Modelimizin özeti

Modelimiz 3 katmandan oluşmaktadır. Sonuncu katmanda ise 2 çıkışımız bulunmaktadır.

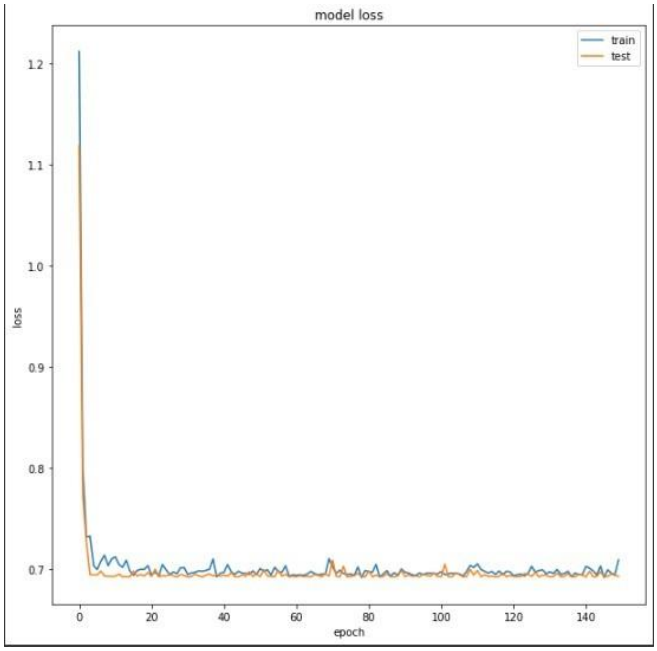
Modelimi epoch olarak 150, batch_size olarak 20 ayarlayarak eğitime başlattım.

```
11/11 [==============================] - 4s 329ms/step - loss: 0.6955 - accuracy: 0.5000 - val_loss: 0.6947 - val_accuracy: 0.4811
Epoch 107/150
11/11 [==============================] - 4s 331ms/step - loss: 0.6934 - accuracy: 0.5280 - val_loss: 0.6925 - val_accuracy: 0.5189
Epoch 108/150
11/11 [==============================] - 4s 328ms/step - loss: 0.6976 - accuracy: 0.5000 - val_loss: 0.6926 - val_accuracy: 0.5189
Epoch 109/150
11/11 [==============================] - 4s 329ms/step - loss: 0.7037 - accuracy: 0.4813 - val_loss: 0.6999 - val_accuracy: 0.4811
Epoch 110/150
11/11 [==============================] - 4s 327ms/step - loss: 0.7017 - accuracy: 0.4533 - val_loss: 0.6942 - val_accuracy: 0.5189
Epoch 111/150
11/11 [==============================] - 4s 328ms/step - loss: 0.7054 - accuracy: 0.4439 - val_loss: 0.6989 - val_accuracy: 0.4811
Epoch 112/150
11/11 [==============================] - 4s 324ms/step - loss: 0.6997 - accuracy: 0.4533 - val_loss: 0.6925 - val_accuracy: 0.5189
```

Şekil 5.13 – Modelin eğitim süreci



Şekil 5.14 – Modelin eğitim ve test doğruluğu



Şekil 5.15 – Modelin eğitim ve test kayıpları

Gördüğümüz üzere çokça başarılı bir model eğitiminin yapılamadığını görmüş oluyoruz. Eğitilen Verilerin arttırılması ve/veya katmanların arttırılması ile modelimizin doğruluğunu arttırılabileceğini düşünüyorum.

Modelimizi daha önce hiç veri setinde bulunmayan bir resim ile tahmin etmeye çalışacağım.

Şekil 5.17 – İnternetten aldığım herhangi bir araba resmi

```
[ ]  1  test_img = model.predict(new_image)
     2  test_img
     3
     4  print(f'''non_valid_license = {round(test_img[0][0] * 100, 2)}%
     5  \nvalid_license = {round(test_img[0][1] * 100, 2)}% ''')
     6  # first index = non_valid_plate percentage  -> 53.49% non_valid_license
     7  # second index = valid_plate percentage     -> 46.50% valid_license

non_valid_license = 53.49%

valid_license = 46.51%
```

Bu resmi OpenCV ile Numpy dizisine çevirerek bir tahminde bulundurm ve sonuç :

Şekil 5.18 – Rastgele bir resmin tahmin sonucu

Görüldüğü üzere non_valid_license sınıfına 53.49% doğruluk gösterdiğini görüyoruz. Gerçekten de bu resim üzerinde herhangi bir plaka bulunmamaktadır.

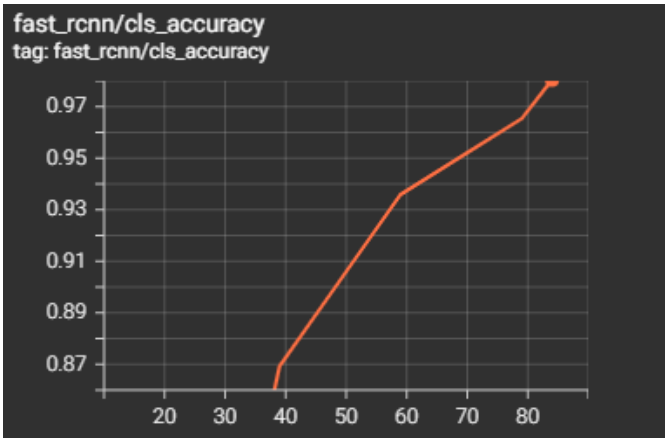
C. Faster R-CNN

Faster R-CNN maliyet açısından yük olan selective search yerine daha kullanışlı olan Region Proposal Network kullanmaktadır.

Burada Facebook'un geliştirmiş olduğu Detector2'nin geliştirmiş olduğu modeli kullandım. Öncellikle veri setlerimi projeme dahil ettim. Projeme dahil ettikten sonra kullanacağım modelimin kofigürasyon dosyasını ayarlarını yaptım.

```
# Creating Config File
cfg = get_cfg()
# Fetchs config file and adds into our config file
cfg.merge_from_file(model_zoo.get_config_file("COCO-Detection/faster_rcnn_R_50_C4_3x.yaml"))
# Adds training data into our config file
cfg.DATASETS.TRAIN = ("my_training_dataset",)
# Adds testing data into our config file
cfg.DATASETS.TEST = ("my_test_dataset",)
# Number of workers
cfg.DATALOADER.NUM_WORKERS = 4
# Fetches weight file and adds into our config file
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-Detection/faster_rcnn_R_50_C4_3x.yaml")
# Batch Size
cfg.SOLVER.IMS_PER_BATCH = 4
# Learning Rate
cfg.SOLVER.BASE_LR = 0.001
# Number of iterations required to decay Learning Rate  (zero by default)
cfg.SOLVER.STEPS = []

# Batch Size per Image
cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 128



cfg.MODEL.DEVICE = "cuda"
# Number of Iterations (epoch-like)
cfg.SOLVER.MAX_ITER = 500
# Number of classes (only for because we're only training to find license plate)
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 1
```

Konfigürasyon dosyalarını ekledikten sonra modelimi eğittim. Modelimin sonuçları ise bu şekildedir:



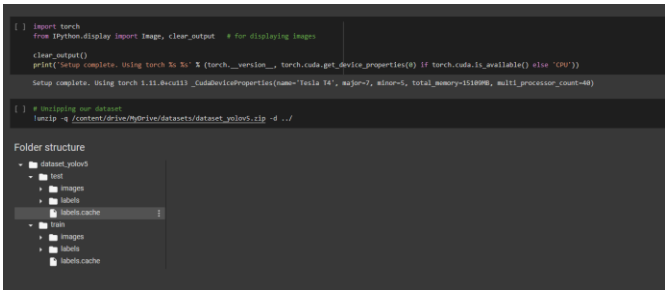
Görüldüğü üzere epoch sayısının 30'un sonlarında doğruluğu 97%'ye çıkmıştır. Modelimizi test ettiğimde ise gerçekten 90-95% başarı ile test verilermizin plakalarını bulmaktadır.

D. YOLOv5

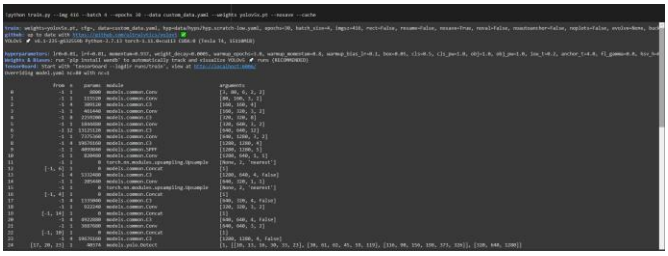
YOLOv5, diğer YOLO versiyonlarından farklı olarak bir PyThoch implementasyonudur. YOLOv4'teki gibi (Cross Stage Partial Networks)CSP backbone ve PA-NET neck kullanılır. Head kısmında ise YOLOv4'te kullanılan model kullanılır.

Aktivasyon fonksiyonu olarak, Leaky ReLU ve Sigmoid kullanılır. YOLO v5'te Leaky ReLU aktivasyon fonksiyonu orta/gizli katmanlarda kullanılır ve sigmoid aktivasyon fonksiyonu son tespit katmanında kullanılır.

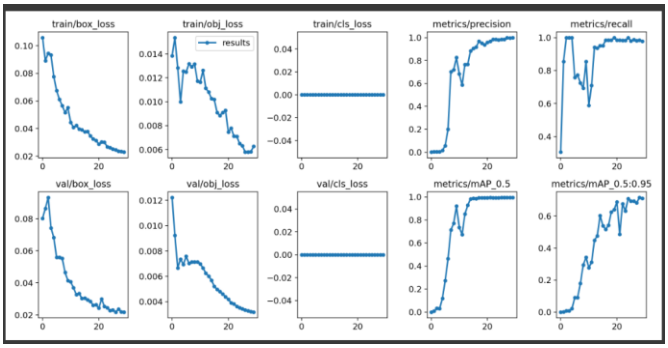
Son modelim olarak YOLOv5 algoritmasını kullandım. Öncellikle github sayfasında projeme klonladım. Klonladıktan sonra veri setimi projeme dahil ettim.



Veri setimin konfigürasyon dosyalarını vererek ve ağırlık dosyasını da belirledikten sonra modeli eğitime soktum.



Eğitim sonrası başarısı :



Modelimin testi:

Kaynakça

[1] *Youtube - Web Scraping (Downloading) Google Images using Python + Selenium Webdriver*

[2] *Youtube - How to Scrape and Download ALL images from a webpage with Python*

[3] *Youtube - Python Tutorial: Web Scraping with BeautifulSoup and Requests*

[4] *Youtube - How To Web Scrape & Download Images With Python*

[5] *https://github.com/ultralytics/yolov5*

[6] *https://github.com/facebookresearch/detectron2*

[7] *https://detectron2.readthedocs.io/en/latest/*

[8] *https://teknoloji.org/nesne-tanima-algoritmalari-r-cnn-fast-r-cnn-ve-faster-r-cnn-nedir/*

[9] *https://learnopencv.com/custom-object-detection-training-using-yolov5/*

[10] *https://medium.com/intelligentmachines/convolutional-neural-network-and-regularization-techniques-with-tensorflow-and-keras-5a09e6e65dc7#:~:text=Regularization%20is%20a%20method%20that,is%20assigned%20a%20certain%20weight*.

[11] *https://towardsdatascience.com/multilayer-perceptron-explained-with-a-real-life-example-and-python-code-sentiment-analysis-cb408ee93141*

[12] *https://www.kaggle.com/code/androbomb/simple-nn-with-python-multi-layer-perceptron/notebook*

[13] *https://medium.com/datatorch/how-to-create-a-custom-coco-dataset-from-scratch-7cd28f0a2c88*

[14] *https://labelstud.io/blog/Quickly-Create-Datasets-for-Training-YOLO-Object-Detection.html*

[15] *https://www.visiongeek.io/2019/10/preparing-custom-dataset-for-training-yolo-object-detector.html*