

BURSA TEKNİK ÜNİVERSİTESİ
MÜHENDİSLİK VE DOĞA BİLİMLERİ FAKÜLTESİ



OTOPARK UYGULAMASI

LİSANS BİTİRME ÇALIŞMASI

Mehmet Mert Fidan

Bilgisayar Mühendisliği Bölümü

Haziran, 2024

BURSA TEKNİK ÜNİVERSİTESİ
MÜHENDİSLİK VE DOĞA BİLİMLERİ FAKÜLTESİ



OTOPARK UYGULAMASI

LİSANS BİTİRME ÇALIŞMASI

Mehmet Mert FİDAN
19360859018

Bilgisayar Mühendisliği Bölümü

Danışman: Dr. Öğr. Üyesi Mustafa Özgür CİNGİZ

Haziran, 2024

BTÜ, Mühendislik ve Doğa Bilimleri Fakültesi Bilgisayar Mühendisliği Bölümü'nün 19360859018 numaralı öğrencisi Mehmet Mert FİDAN, ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı “Otopark Uygulaması” başlıklı bitirme çalışmasını aşağıda imzaları olan jüri önünde başarı ile sunmuştur.

Danışmanı : **Dr. Öğr. Üyesi Mustafa Özgür CİNGİZ**
Bursa Teknik Üniversitesi

Jüri Üyeleri : **Dr. Öğr. Üyesi VOLKAN ALTUNTAŞ**
Bursa Teknik Üniversitesi

Arş. Gör. ŞEYMA DOĞRU
Bursa Teknik Üniversitesi

Arş. Gör. YUSUF KAYIPMAZ
Bursa Teknik Üniversitesi

Savunma Tarihi : 13 Haziran 2024

BM Bölüm Başkanı : Prof. Dr. Tugay Tugay BİLGİN
Bursa Teknik Üniversitesi/...../.....

İNTİHAL BEYANI

Bu bitirme alışmasında grsel, işitsel ve yazılı biçimde sunulan tüm bilgi ve sonuçların akademik ve etik kurallara uyularak tarafımdan elde edildiğini, bitirme alışması içinde yer alan ancak bu alışmaya özgü olmayan tüm sonuç ve bilgileri bitirme alışmasında kaynak göstererek belgelediğimi, aksinin ortaya ıkması durumunda her türlü yasal sonucu kabul ettiğimi beyan ederim.

Öğrencinin Adı Soyadı: Mehmet Mert FİDAN

İmzası :

ÖNSÖZ

Lisans bitirme projem olan otopark uygulaması, gerçek hayatta da uygulanabilir bir proje olması sebebiyle bana 4 yıllık emeğimin bir sonucunun olduğunu göstermiş oldu. Bu 4 yıllık süreçte beni geliştirmek için bana çeşitli projeler yaptıran ve bana bir şeyler katmak için emek harcayan bütün hocalarıma teşekkür etmek isterim. Proje sürecinde düşüncelerini benimle paylaşan Dr. Öğr. Üyesi Mustafa Özgür CİNGİZ hocama da ayrıca teşekkür ederim. Ayrıca 4 yıllık süreç boyunca arkamda olan ve yardımları bulunan aileme ve bütün arkadaşlarıma çok teşekkür ederim.

Haziran 2024

Mehmet Mert FİDAN

İÇİNDEKİLER

Sayfa

ÖNSÖZ	v
İÇİNDEKİLER.....	vi
ŞEKİL LİSTESİ.....	vii
ÖZET	viii
SUMMARY.....	ix
1. GİRİŞ	10
1.1 Literatür Araştırması	11
1.1.1 Plakaların tespiti.....	11
1.1.2 Resim üzerinde yapılan işlemler	12
1.1.3 Karakter tanıma	13
2. PLAKALARIN TESPİTİ	14
2.1 Yolo Kurulumu	14
2.2 Veri Seti	15
2.3 Colab Üzerinde YOLO Eğitimi	15
2.4 YOLO Versiyonlarının Karşılaştırması.....	17
2.5 Gerçek Verilerle YoloV7 Testi	20
3. RESİMLER ÜZERİNDE YAPILAN İŞLEMLER.....	22
3.1 Scale (Ölçekleme)	22
3.2 Plakada Gürültü Oluşturabilecek Kısımın Kesilmesi.....	23
3.3 Bilateral Filtre	24
3.4 Keskinleştirme Filtresi.....	25
3.5 Görüntüyü Gri Tonlarına Dönüştürme	26
3.6 Otsu ile Thresholding (Eşikleme)	26
3.7 Başarısız Görüntü İşleme Sonuçları	28
3.8 En Başarılı Görüntü İşleme Sonucu	29
4. RESİMDEKİ KARAKTERLERİN OKUNMASI	30
4.1 Gerekli kütüphaneler	30
4.2 Sonuçlar	31
5. SONUÇ VE ÖNERİLER	33
KAYNAKLAR.....	34
ÖZGEÇMİŞ	35

ŞEKİL LİSTESİ

Sayfa

Şekil 1: Anaconda Üzerinde Çerçeve Oluşturma	14
Şekil 2: Oluşturulan Çevrenin Aktive Edilmesi	14
Şekil 3: YOLO'nun İndirilmesi	15
Şekil 4: Gerekli Kütüphanelerin Kurulumu	15
Şekil 5: .yaml Uzantılı Dosya.....	16
Şekil 6: Colab'de Train İşleminin Başlatılması	17
Şekil 7: YoloV6 Genel Metrikler	18
Şekil 8: YoloV7 F1-Confidence Grafiği.....	18
Şekil 9: YoloV8 F1-Confidence Grafiği.....	19
Şekil 10: YoloV6 Test Sonuçları.....	19
Şekil 11: YoloV7 Test Sonuçları.....	19
Şekil 12: YoloV8 Test Sonuçları.....	19
Şekil 13: Örnek Gerçek Hayat Verisi	20
Şekil 17: Scale (Ölçekleme) İşlemi	23
Şekil 18: Plaka Resmi	24
Şekil 19: Resmin Kesilmesi	24
Şekil 20: Bilateral Filtre Kullanımı	25
Şekil 21: Keskinleştirme Filtresinin Uygulanması	25
Şekil 22: Görüntüyü Siyah-Beyaz Yapma	26
Şekil 23: Örnek Resim-1	27
Şekil 24: Örnek Resim-2	27
Şekil 25: Otsu Yönteminin Kullanılması	27
Şekil 26: Başarısız Sonuç-1.....	28
Şekil 27: Başarısız Sonuç-2.....	28
Şekil 28: Başarısız Sonuç-3.....	28
Şekil 29: Başarısız Sonuç-4.....	28
Şekil 30: Başarısız Sonuç-5.....	28
Şekil 31: Girdi Görseli	29
Şekil 32: Çıktı Görseli.....	29
Şekil 33: EasyOCR Kullanımı.....	30
Şekil 34: docTr Kullanımı.....	31
Şekil 35: PyTesseract Kullanımı	31

OTOPARK UYGULAMASI

ÖZET

Otoparklar modern şehir yaşamının vazgeçilmez bir parçası haline gelmiştir. İnsanlar, bir yere gittikleri zaman içlerinin rahat etmesi için arabalarını güvenli bir yere park etmek istemektedirler. Bunun en önemli nedenleri arabalarının çekileceğinden duyulan korku ve hırsızlıktır. Ayrıca sokakta bazı insanlar sırf zevkine başkalarının araçlarına çeşitli şekillerde hasar verebilmektedir. Bu sebeple şehirde yaşayan birçok insan, arabalarını kaldırıma veya güvensiz bir noktaya park etmek yerine otoparka park etmeyi tercih ediyor. Otoparklar insanların arabaların park etmeleri için güvenli alanlar oluştururlar. Bu sayede “Arabamın başına bir şey gelir mi?” endişesi de bitmiş olur. Ayrıca otoparklar araçların park etmeleri için, sokağa park ederken alınan riske göre oldukça cüzi rakamlar istemektedirler. Günümüzde her sektörde, teknolojinin sunduğu imkanlar sayesinde işler otomatik hale geliyor. Bilgisayarların girmediği neredeyse hiçbir sektör kalmadı. Bu uygulama otoparkların da bu teknolojilerden faydalanması ve gelişmesi için üretilmiştir. Uygulama, otoparka gelen arabaların resimlerini bir kamera yardımıyla kaydeder. Ardından yapay zeka, çekilen resimler üzerinden arabaların plakalarının bulunduğu konumu resim üzerinde tespit eder. Bunu yapmak için YOLO algoritmasından faydalanılmıştır. Daha sonra yapay zekanın plakayı algılayabilmesi için resimler üzerinde çeşitli oynamalar yapılır. Burada resme çeşitli filtreler, kırpımlar ve görsel efektler uygulanır. Plakanın tespitinde ise optik karakter tanıma (OCR) teknolojileri kullanılmıştır. Yapay zeka otoparka gelen aracın plakasını tespit ettikten sonra ise plakayı, arabanın otoparka giriş saatiyle birlikte kaydeder. Araçların otoparktan çıkışları da yine aynı şekilde sisteme kaydedilir. Plaka tanıma sistemi sayesinde, park alanına giren ve çıkan araçların takibi daha kolay ve hızlı bir şekilde yapılmış olur. Araçların çıkışı sırasında müşteriden alınacak olan tutar, sistem tarafından otomatik olarak belirlenir ve otopark çalışanına tahsis etmesi gereken tutar gösterilir. Bu sayede otopark için işler hızlı ve pratik bir biçimde tamamlanmış olur. Aynı zamanda da müşteri güvenli bir şekilde arabasını teslim almış olur. Uygulamanın, otoparkta kaliteli bir kamera kullanılması ve uygulamanın spesifik otoparklar için optimize edilmesi halinde oldukça başarılı olacağına yönelik vaatkar sonuçlar elde edilmiştir.

Anahtar kelimeler: otopark, yapay zeka, plaka

PARKING APPLICATION

SUMMARY

Parking lots have become an indispensable part of modern city life. People want to park their cars in a safe place for peace of mind when they go somewhere. The most important reasons for this are the fear that their cars will be towed and theft. In addition, some people on the street may damage other people's cars in various ways just for the fun. For this reason, many people living in the city prefer to park their cars in a parking lot rather than on the sidewalk or in an unsafe spot. Parking lots provide a safe place for people to park their cars. In this way, the worry of "Will something happen to my car?" is over. In addition, parking lots ask for very small amounts for the parking of vehicles compared to the risk taken when parking on the street. Today, in every sector, things are automated because of the opportunities offered by technology. There is almost no sector where computers are not used. This application has been produced for parking lots to benefit and develop from these technologies. The app uses a camera to take pictures of cars entering the parking lot. Then, the artificial intelligence detects the location of the license plates of the cars on the image. The YOLO algorithm is used to do this. Then, various manipulations are made on the images so that the AI can detect the license plate. Various filters, cropping and visual effects are applied to the image. Optical character recognition (OCR) technologies were used to detect the license plates. After the artificial intelligence detects the license plate of the vehicle entering the parking lot, it records the license plate along with the time the car enters the parking lot. The exit of the vehicles from the parking lot is also recorded in the same way. Thanks to the license plate recognition system, it is easier and faster to track vehicles entering and exiting the parking lot. The price to be collected from the customer during the exit of the vehicles is automatically determined by the system and the price to be allocated to the parking lot employee is shown. In this way, things are completed quickly and practically for the parking lot. At the same time, the customer receives his car safely. Promising results have been obtained that the application will be very successful if a quality camera is used in the parking lot and the application is optimized for specific parking lot.

Keywords: Parking lot, artificial intelligence, license plate

1. GİRİŞ

Günümüzde şehirlerde yaşanan hızlı nüfus artışı ve artan araç sayısı, park yeri bulma sorununu önemli bir mesele haline getirmiştir. Özellikle büyük şehirlerde, sürücülerin park yeri ararken harcadıkları zaman ve çaba, trafik yoğunluğunu artırmakta ve şehir içi ulaşımı olumsuz etkilemektedir. Ayrıca sürücülerin araçlarını sokağa park etmesi, sürücüler için ciddi bir stres kaynağıdır. Bunun en önemli sebebi arabaların çekilebilmesi ihtimali, hırsızlık ve kasten veya kaza sonucu başka insanların arabalara zarar vermesidir. Bu sebepten ötürü şehir merkezlerinde otoparklar git gide popülerite kazanmışlardır. Çünkü otoparklar müşterilerinden, sokağa park etmelerinden oluşacak riske kıyasla oldukça düşük ücretler talep etmektedir.

Son yıllarda teknolojik gelişmelerin hızının parabolik olarak artmasıyla birlikte teknoloji ve bilgisayarlar bütün iş sektörlerinde kullanılmaya başlanmıştır. Bu durum bütün iş kollarında güvenliğin artmasına, işlerin hızlanmasına, verimliliğin ve karlılığın artmasına yol açmıştır. Otoparklar da bu teknolojik gelişmelerden yararlanan iş kollarından birisidir. Otoparklarda yılların geçmesiyle birlikte farklı teknolojiler kullanılmıştır. Bunların bazıları kısıtlı bir alana daha fazla aracı sığdırmak için kullanılmışken bazıları da işleri hızlandırmak ve otomatik hale getirmek için kullanılmıştır. Bu yazıda ikinci kısma odaklanılmıştır. Zamanla işleri otomatikleştirmek için bazı yöntemler geliştirilmiştir. Bu yöntemlerden biri otoparka giriş ve çıkış yapan araçların otomatik olarak belirlenmesi ve fiyatlandırmayı otomatik olarak yapmasıdır. Bunun için otoparkların giriş ve çıkışlarına kameralar yerleştirilmiştir. Bu kameralar plakaların görüntüsünü kaydederek sisteme iletir. Sistem yapay zeka ve çeşitli görüntü işleme tekniklerini kullanarak aracın plakasını tanır ve giriş-çıkış zamanıyla birlikte veri tabanına kaydeder. Ayrıca sistem, müşteriden alınacak olan tutarı da belirli mevzuat kurallarına göre hesaplayarak, tahsis edilmesini sağlar. Bu yazıda böyle bir sistemin geliştirilmesi sürecinden bahsedilmiştir. Plakası olmayan ya da okunamayacak seviyede plakası kötü durumda

olan araçları tespit edebilmek için manyetik araç algılama cihazı gibi teknolojiler de vardır ancak bu yazıda bu tarz farklı yöntemlere yer verilmemiştir.

1.1 Literatür Araştırması

Konuyla alakalı birçok makale incelenmiştir. Bu makalelerin içerikleri büyük oranda benzerlik göstermektedir. Projeler arasında farklılıklar bulunsa da bu farklılıklar genel olarak oldukça minimal boyutta kalmıştır. Yine de bu minimal farklılıkların ve uygulamadaki farklılıkların, sonuç üzerinde önemli değişikliklere neden olabildiği gözlemlenmiştir. Bu farklara sonra değinilecektir.

1.1.1 Plakaların tespiti

Araştırma yapılan makalelerde en büyük farklılıklar plakaların konumunun tespiti sırasında gerçekleşmiştir. Bu makalelerde araçların plakalarını tespit etmek için sadece küçük bir kesim YOLO algoritmasını kullanmıştır. Makalelerin bir kısmında ise bu problemin etrafından dolaşmış ve plakaların resimdeki konumları tespit edilmemiştir. Kalan makalelerde ise benzer teknikler kullanılmıştır. Örneğin bir makalede resme Thresholding (Eşikleme) fonksiyonu kullanılarak, resim siyah-beyaz hale getirilmiştir. Ardından kenarlar algılanarak plaka tespiti yapılmıştır. Başka bir makalede ise resim üzerinde birtakım filtreler uygulandıktan sonra resim üzerindeki dikdörtgenler tespit edilmiş ve ardından bu dikdörtgenler içerisinde plaka olması en olası olan dikdörtgen seçilmiştir. Ancak bu konuda en başarılı makalelerde YOLO algoritmasının kullanıldığı belirlenmiştir. YOLO algoritmasını kullanan çalışmada plakanın resimdeki konumunun tespitinde %97'lik bir başarı oranı yakalanmıştır. Diğer makalelerin bazılarında ise %93,34, %95 ve %96,2'lik başarı oranları tutturulmuştur. Bazı makalelerde ise %87'nin üzerine çıkılamamıştır.

1.1.2 Resim üzerinde yapılan işlemler

Plakaların resim üzerindeki yerleri tespit edildikten sonra plakaların tanınması sırasında başarı oranının yükseltilmesi için resim üzerinde çeşitli oynamalar yapılması gerekmektedir. Bu aşama projenin en önemli kısmıdır çünkü plaka tanıma kısmındaki başarı oranı büyük oranda burada uygulanacak olan işlemlere bağlıdır. Bu kısımda bütün makalelerde uygulanan yöntemler neredeyse birebir aynıdır ancak çeşitli farklılıklar da bulunmaktadır. Bu makalelerde kullanılan filtre ve fonksiyonlarda kullanılan parametreler verilmemiştir. Bir filtrenin parametrelerindeki farklılıklar, diğer filtrelerle birleştğinde, ortaya çıkan sonucu oldukça değiştirebilmektedir.

Makalelere bakıldığında hepsinde grayscale (Gri tonlama) işlemi uygulanmıştır. Bu işlem sayesinde görüntüdeki renk detayları kaybolur ve resimdeki renkler siyahla beyaz arasında bir görünüme kavuşur. Bu durum plakalardaki karakterlerin okunması için yapılsa da bazı makalelerde ayrıca plakaların konumlarının tespitinde de kullanılmıştır. Bilateral filtre de problemin çözümünde sıkça kullanılan bir yöntemdir. Bilateral filtre görüntülerdeki gürültüleri azaltmaya yarayan bir yumuşatma filtresidir. Kullanılan farklı bir yöntem de Canny Edge (Kenar) tespiti yöntemidir. Bu yöntem resimde bulunan kenarları beyaz renge boyayarak plakaların okunmasını kolaylaştırır. Ayrıca bazı makalelerde plakaların konumunun tespitinde de kullanılmıştır. Gaussian Blur da kullanılan bir başka yöntemdir. Arka planı bulanıklaştırma, derinlik hissi yaratma veya seçilen nesneleri vurgulama amacıyla kullanılır. Bunların dışında threshold (Eşikleme) işlemleri de görüntülerdeki gürültüleri azaltmak için kullanılmıştır.

Makalelerde görüntülerle oynamak için genel olarak bu işlemler uygulanmıştır. Bazı makalelerde bu işlemlerin sıraları değişmiştir. Bazılarında ise bazı yöntemler kullanılmış, bazıları ise kullanılmamıştır. Ancak genel olarak benzer yöntemler kullanılmıştır.

1.1.3 Karakter tanıma

Resim üzerinde yapılan işlemlerden sonra ise geriye işlenmiş olan görüntülerin üzerinden plakaları okumak kalıyor. Bunun için “Optik Karakter Tanıma” (OCR) işlemi yapan çeşitli kütüphaneler bulunmaktadır. Ancak neredeyse bütün makalelerde PyTesseract kütüphanesi tercih edilmiştir. Bir çalışmada ise ekip, karakter tanınması için kendi yapay zeka modellerini geliştirmiştir. Kendi yapay zekalarını eğiten ekip plaka tanıma konusunda %96’lık bir başarı oranı yakalamıştır. PyTesseract kullanılan diğer makalelerde ise %96,7, %90 ve %91,67’lik başarı oranları yakalanmıştır.

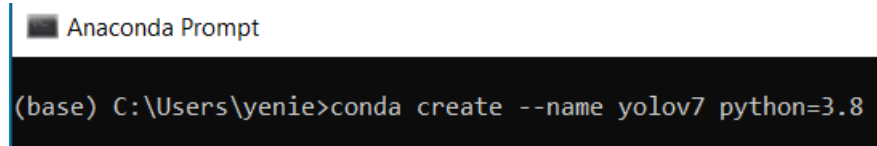
2. PLAKALARIN TESPİTİ

Projenin başlangıç aşaması, literatür araştırması kısmından da belirtildiği gibi resim üzerindeki plakaların tespit edilmesidir. Literatür araştırması kısmında bu konuda en başarılı yöntemin YOLO algoritması olduğu tespit edilmiştir. Bu projede YOLO'nun V7 sürümü kullanılmıştır.

YOLO (You only look once) Türkçe karşılığı “Yalnızca bir kez bak” olan, gerçek zamanlı nesne takibi için CNN kullanan en yaygın algoritmalarından birisidir. YOLO'dan daha iyi algoritmalar bulunmasına karşın çok azı YOLO kadar hızlı çalışabilmektedir. YOLO kısa süreler içerisinde büyük veri setlerini başarılı bir şekilde modelleyebilir.

2.1 Yolo Kurulumu

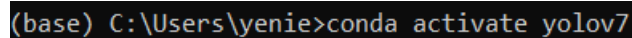
Yolo'yu kurmadan önce Anaconda'nın kurulması gerekli. Anaconda kurulduktan sonra Anaconda Prompt açılır ve kurulumları gerçekleştirmek için aşağıdaki koddaki gibi yeni bir environment (çevre) oluşturulur.



```
Anaconda Prompt
(base) C:\Users\yenie>conda create --name yolov7 python=3.8
```

Şekil 1: Anaconda Üzerinde Çerçeve Oluşturma

Ardından aşağıdaki şekildeki gibi, oluşturulmuş olan environment aktive edilir.



```
(base) C:\Users\yenie>conda activate yolov7
```

Şekil 2: Oluşturulan Çevrenin Aktive Edilmesi

Ardından YOLO'yu çalıştırmak için gerekli olan Pytorch ve Cuda'nın indirilmesi ve kurulumlarının tamamlanması gerekmektedir. Burada önemli olan nokta, kullanılan

bütün uygulamaların ve kütüphanelerin sürümlerinin birbiriyle uyumlu olmasıdır. Aksi taktirde YOLO gerektiği gibi çalışmayacaktır. Ardındansa YOLO'nun indirilmesi ve YOLO'nun çalışması için gerekli olan kütüphaneler indirilmelidir. Bu işlemler aşağıdaki görsellerde gerçekleştirilmiştir.

```
(yolov7) C:\Users\yenie>git clone https://github.com/WongKinYiu/yolov7
```

Şekil 3: YOLO'nun İndirilmesi

```
(yolov7) C:\Users\yenie>pip install -r requirements.txt
```

Şekil 4: Gerekli Kütüphanelerin Kurulumu

2.2 Veri Seti

Modelin eğitilebilmesi yeteri kadar büyüklükte bir veri setine ihtiyaç bulunmaktadır. Uygun veri seti Kaggle üzerinden bulunmuştur. Veri setinde binlerce araba, motosiklet ve kamyon gibi görseller bulunmaktadır. Veri setinde 25470 adet train, 1073 validation ve 386 adet test verisi bulunmaktadır. Toplamda ise 26929 adet veri bulunmaktadır. Görsellerde çeşitlilik sağlamak ve modelin overfitting (Aşırı öğrenme) olmaması için görsellere çeşitli manipülasyonlar uygulanmıştır. Görseller farklı açılarda rotate edilmiş ve çeşitli efektler uygulanmıştır.

2.3 Colab Üzerinde YOLO Eğitimi

Projenin geliştirildiği bilgisayarda 4 GB ekran kartı bulunmaktadır. YOLO, modelleri eğitmek için ciddi miktarda GPU alanı kullanmaktadır. İlgili bilgisayar gerekli özellikleri karşılamadığından dolayı eğitilmesi gerekli olan modelin Colab üzerinde eğitilip daha sonra eğitilmiş olan ağırlıkların bilgisayar üzerinde kullanılması uygun görülmüştür.

Google Colab, Google'ın sağladığı ücretsiz bir bulut tabanlı not defteridir. Araştırmacılar, veri bilimciler ve geliştiriciler, Python kodlarını yazmak, çalıştırmak ve paylaşmak için Colab'ı kullanabilirler. Colab, Jupyter Notebook benzeri bir arayüz sunar ve kullanıcıların python kodlarını bulut üzerinde çalıştırmalarına olanak sunar.

Colab'de YOLO eğitimi gerçekleştirebilmek için öncelikle bir Google hesabı gerekmektedir. Eğitim Colab'de ücretsiz olarak sunulan GPU'lar ile gerçekleştirilmiştir. Colab bu imkânı sınırlı olarak sunar. Colab'de günlük ücretsiz olarak ortalama 3 saat YOLO eğitimi gerçekleştirilebilmektedir. Çeşitli düzeylerde ücretler ödenerek bu süre uzatılabilmektedir. Ayrıca hesap bot olarak algılanırsa, Colab'den banlanma ihtimali de bulunmaktadır.

Colab'de YOLO modelleri sınırlı süre eğitilebildiği için veri setinin bölünmesi gerekmiştir. Bu sayede modelin daha fazla epoch ile eğitilmesi sağlanmıştır. Veri seti 4032 (%87) train, 200 (%4) validasyon ve 386 (%8) test verisi olacak şekilde ayrılmıştır. Test veri sayısı düşük olduğundan bu veriler bölünmemiştir. Veriler bölündükten sonra veri setiyle ilgili bilgileri içeren “.yaml” uzantılı bir dosyanın hazırlanması gerekmektedir. Dosya aşağıdaki şekildeki gibi hazırlanmıştır.

```
train: ../car_dataset/train/images
val: ../car_dataset/valid/images
test: ../car_dataset/test/images

nc: 1
names: ['plate']
```

Şekil 5: .yaml Uzantılı Dosya

Dosyada train, validasyon ve test verilerinin bulunduğu konumlar verilmiştir. Nc, veri seti üzerinde tespit edilmesi gereken sınıf sayısıdır. Names kısmında ise bu sınıfların isimleri girilmiştir.

Dosya veri seti klasörünün içine atılarak zip'lenir ve Google Drive'a yüklenir. Ardından eğitim gerçekleştirilir. Eğitimin gerçekleşmesi için gerekli kod satırları aşağıda verilmiştir.

```
!python train.py --batch-size 16 --data "/content/data.yaml" --img 640 640 --cfg cfg/training/yolov7.yaml --epochs 200 \
--weights "yolov7_training.pt" --name yolov71 --hyp "/content/yolov7/data/hyp.scratch.custom.yaml" --device 0
```

Şekil 6: Colab'de Train İşleminin Başlatılması

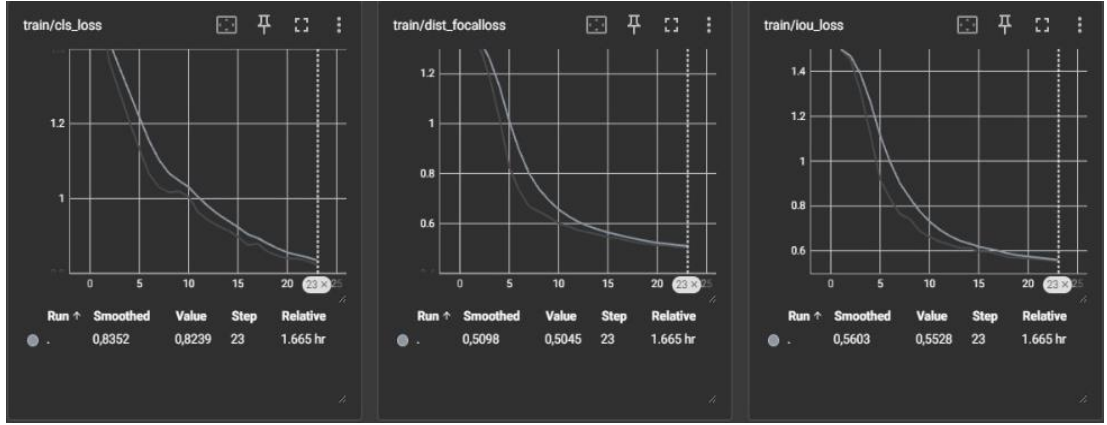
Eğitimi başlatmak için bazı parametrelerin girilmesi gerekmektedir. Bunlardan ilki batch size'dır. Batch size kısaca, her bir eğitim adımında modelin gördüğü veri örneklerinin sayısıdır. 2'nin katları olacak şekilde girilmelidir. Genellikle ne kadar büyük bir değer girilirse eğitim hem o kadar hızlı gerçekleşir hem de başarı oranı o kadar yüksek olur. Burada değeri 16 olarak seçilmiştir. Resimlerin boyutları eğitimi hızlandırmak için 640*640 boyutuna indirilmiştir. Model 200 epoch eğitilecek şekilde ayarlanmıştır fakat Colab'ın süre sınırı nedeniyle bu sayıya ulaşamayacaktır. Modelin GPU ile eğitilmesi için device 0 değeri girilmiştir ve eğitim başlatılmıştır.

Karşılaştırma yapabilmek için Yolo'nun v7 sürümünün yanı sıra v6 ve v8 modelleri de aynı veri setiyle eğitilmiştir.

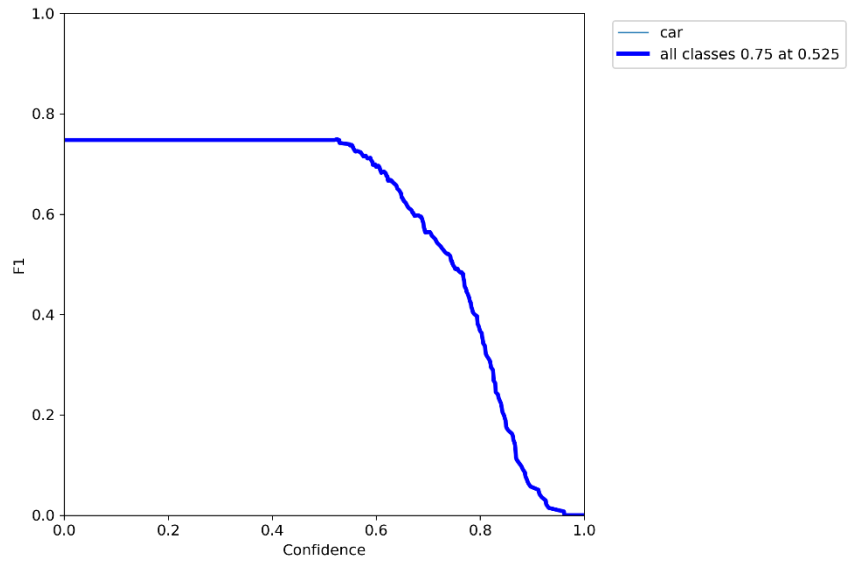
2.4 YOLO Versiyonlarının Karşılaştırması

İlgili YOLO versiyonlarının her biri yaklaşık 3 saat boyunca, yani Colab'ın izin verdiği süre müddetince eğitilmiştir. Daha adil bir sonuç elde edilebilmesi içinse her bir modelin eğitimi 23. epoch'da durdurulmuştur.

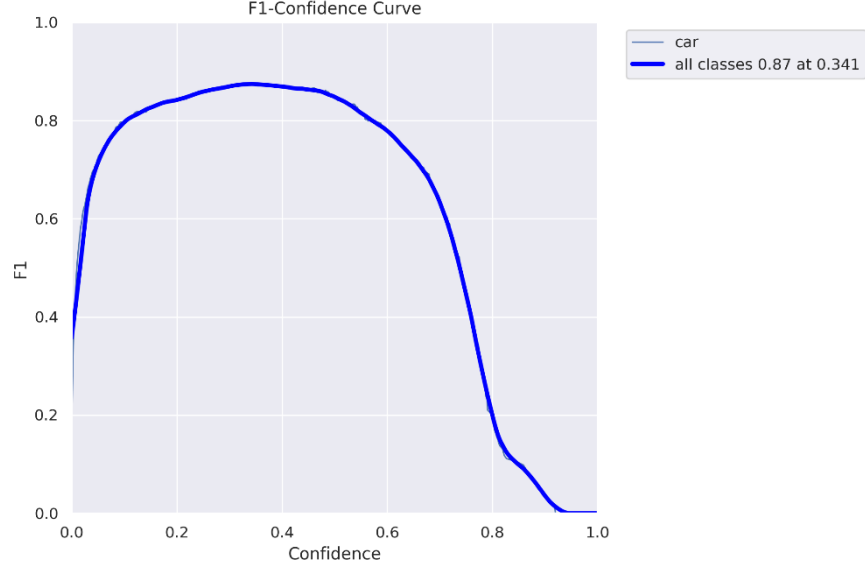
Yolo'nun precision (kesinlik) ve recall (duyarlılık) gibi metrikler dışında kendi başarı ölçüt yöntemleri de bulunmaktadır. Bunlar genellikle bir nesne tespit tahmininin yüzde kaç ihtimalle doğru olarak gerçekleştiğiyle ilgilidir. Aşağıda üç YOLO versiyonuna ait genel metrikler ve test verileri üzerindeki sonuçları gösterilmiştir.



Şekil 7: YoloV6 Genel Metrikler



Şekil 8: YoloV7 F1-Confidence Grafiği



Şekil 9: YoloV8 F1-Confidence Grafiği

Average Precision	(AP)	@[IoU=0.50:0.95	area= all maxDets=100]	= 0.358
Average Precision	(AP)	@[IoU=0.50	area= all maxDets=100]	= 0.722
Average Precision	(AP)	@[IoU=0.75	area= all maxDets=100]	= 0.312
Average Precision	(AP)	@[IoU=0.50:0.95	area= small maxDets=100]	= 0.158
Average Precision	(AP)	@[IoU=0.50:0.95	area=medium maxDets=100]	= 0.515
Average Precision	(AP)	@[IoU=0.50:0.95	area= large maxDets=100]	= 0.509
Average Recall	(AR)	@[IoU=0.50:0.95	area= all maxDets= 1]	= 0.369
Average Recall	(AR)	@[IoU=0.50:0.95	area= all maxDets= 10]	= 0.483
Average Recall	(AR)	@[IoU=0.50:0.95	area= all maxDets=100]	= 0.500
Average Recall	(AR)	@[IoU=0.50:0.95	area= small maxDets=100]	= 0.277
Average Recall	(AR)	@[IoU=0.50:0.95	area=medium maxDets=100]	= 0.653
Average Recall	(AR)	@[IoU=0.50:0.95	area= large maxDets=100]	= 0.684

Şekil 10: YoloV6 Test Sonuçları

Class	Images	Labels	P	R	mAP@.5	mAP@.5:.95:
all	200	271	0.872	0.657	0.634	0.3

Şekil 11: YoloV7 Test Sonuçları

Class	Images	Instances	Box(P	R	mAP50	mAP50-95):
all	200	271	0.883	0.862	0.919	0.514

Şekil 12: YoloV8 Test Sonuçları

Sonuçlar incelendiğinde bariz bir şekilde YOLO'nun v7 sürümünün v6'ya ciddi bir fark attığı ve v8 sürümünün v7'ye ciddi bir fark attığı gözükmektedir. Precision metriğinde v7 ve v8 sürümleri birbirine benzer değerler sonuçlar üretmiş ve v6 sürümüne ciddi bir fark atmışlardır. Recall metriğinde ise v8 sürümü %86, v7 sürümü %65 ve v6 sürümü %50'lik değerlere ulaşmıştır. Ayrıca test verileri üzerinde v8

sürümü %91, v7 sürümü %82 ve v6 sürümü %62'lik doğruluk oranlarına ulaşmıştır. Bunların dışındaki diğer metriklere de bakıldığı zaman v8 sürümünün v7'den ve v7 sürümünün ise v6 sürümünden çok daha doğru bir şekilde nesne tespiti yapabildiği ayrıca da yaptığı tahminleri çok daha yüksek bir confidence değeriyle gerçekleştirdikleri gözlemlenmiştir.

2.5 Gerçek Verilerle YoloV7 Testi

Yolo v7 test verileri üzerinde yaklaşık %82'lik bir başarı oranı yakalamıştır. Ancak asıl önemli olan nokta, modelin projede karşılaşacağı resimler üzerinde ne kadar başarılı olacaktır. Bu sebeple otoparkta farklı zamanlarda çekilmiş 20 fotoğraf üzerinden modelin gerçek hayattaki başarısı test edilmiştir. Modelin testinde kullanılan örnek bir resim aşağıdaki şekilde gösterilmiştir.



Şekil 13: Örnek Gerçek Hayat Verisi

Model bu resimlerde bulunan plakaların tamamını doğru bir şekilde tespit ederek %100'lük bir başarı oranı yakalamıştır. Uygulama geliştirilirken yapılan testler sırasında nadiren de olsa farklı nesneleri plaka olarak algıladığı da görülmüştür. Ancak modelin genel olarak plaka tespiti konusunda oldukça başarılı olduğu görülmektedir.

3. RESİMLER ÜZERİNDE YAPILAN İŞLEMLER

Plaka tespit edildikten sonra karakter tanıma kısmına geçilmeden önce, plakadaki karakterlerin daha iyi bir şekilde tespit edilebilmesi için resim üzerinde bazı oynamaların yapılması gerekmektedir. Bu oynamaların yapılması için OpenCV (Open Computer Vision Library) kullanılmıştır. OpenCV bilgisayarla görme (computer vision), görüntü işleme, video analizi ve makine öğrenimi (machine learning) yapmak için kullanılan açık kaynaklı devasa ve oldukça hünerli bir yazılım kütüphanesidir.

Bu kısımda genel olarak literatür araştırması kısmındaki işlemler uygulanmıştır. Ancak bunların dışında farklı yollar da denenmiştir ve bazılarının projede kalmasına karar verilmiştir. Ayrıca bu işlemleri uygulamak için pek çok parametrenin girilmesi gerekmektedir. Bir filtrede bulunan parametrelerin değiştirilmesi diğer filtrelerin girişlerini de değiştirmektedir. Dolayısıyla herhangi bir filtrede bulunan bir parametrenin değiştirilmesi, çıkış olarak alınan resimde büyük değişikliklere sebep olabilmektedir. Bu sebepten dolayı filtrelerin parametreleri defalarca kez değiştirilerek en uygun sonuç elde edilmeye çalışılmıştır. Gerektiğinde bazı filtreler çıkarılmış, başkaları eklenmiş ve yine defalarca kez filtre ve benzeri işlemlerin sıraları değiştirilmiştir.

3.1 Scale (Ölçekleme)

“Scale” işlemi bir resmin boyutlarını değiştirmek için kullanılır. Genellikle resmin boyutlarını küçültmek resimdeki piksel sayısının azaltılmasına yardımcı olur ve bunun çeşitli avantajları bulunmaktadır. Örneğin bir yapay zeka modelinin eğitiminde eğitim süresini kısaltması bunlardan biridir. Ancak burada küçültme değil büyütme için kullanılmıştır. Bunun en önemli nedeni, karakter tanıma kısmında plakaların üzerindeki küçük karakterlerin OCR kütüphaneleri tarafından okunamamasıdır. Bir diğer nedeni ise bu uygulamada resimlerin webcam üzerinden çekilmesidir.

Webcam'den çekilen resimlerin kalitesi oldukça düşük olduğundan resmin boyutları büyütülerek, plakada bulunan karakterlerin daha iyi okunabilmesi amaçlanmaktadır. Bu yöntem beraberinde bir sorunu da getirmektedir. Bu da zaten düşük kalitede olan resimlerdeki ayrıntıların daha da yok olmasıdır. Daha sonradan kullanılacak olan filtreler bu sorunu bir miktar çözeceğinden ve götürülerinin yanında getirileri daha fazla olacağı için kullanılmasına karar verilmiştir.

```
scale=4
width=int(pic.shape[1]*scale)
height=int(pic.shape[0]*scale)
pic_resized=cv2.resize(pic,(width,height))
```

Şekil 14: Scale (Ölçekleme) İşlemi

Uygulamada görseller 4 katına genişletilmiştir. Aslında 2-2,5 katına genişletmek de oldukça yeterlidir. Resmin bu kadar büyütülmesinin sebebi kameradan biraz daha uzakta bulunan araçların plakalarının da okunabilmesidir. Ancak gerçek bir otoparkta, bir arabanın kameradan ne kadar uzakta duracağı aşağı yukarı bellidir. Bu sebeple bu sayının kameranın kalitesine ve otoparkın yapısına göre optimize edilmesi yararlı olacaktır.

3.2 Plakada Gürültü Oluşturabilecek Kısımın Kesilmesi

Plakaların sol taraflarında şekil 18'de görüldüğü üzere bir "TR" yazısı bulunmaktadır. Ayrıca bazı plakalarda "TR" yazısının üzerinde bir sembol bulunabilmektedir. Karakter tanıma kısmında bu yazı ve sembol zaman zaman gürültü oluşturarak plakanın yanlış olarak okunmasına neden olabilmektedir. Bu sebeple bu kısmın resimden atılması gerekmektedir.



Şekil 15: Plaka Resmi

Gürültü oluşturabilen bu mavi kısmı resimden atabilmek için öncelikle mavi kısmın bütün plakaya oranı tespit edilerek resimden atılmıştır. Ancak sonradan fark edilmiştir ki YOLO, plakaları tespit ederken plakaların kenarlarını her zaman mükemmel bir biçimde tespit edemiyor. Bu yüzden resimden kesilmesi gereken alan her resimde küçük bir miktar da değişebilmektedir. Bundan dolayı resmin sol tarafından bütün resimlere uyacak şekilde bir oran kesilmektedir. Bu oran denemeler sonucunda %4,7 olarak tespit edilmiştir. Aşağıdaki şekilde kesme işleminin nasıl yapıldığı gözükmektedir.

```
pic_cropped=pic_resized[0:pic_resized.shape[0],int(pic_resized.shape[1]*0.047):pic_resized.shape[1]]
```

Şekil 16: Resmin Kesilmesi

3.3 Bilateral Filtre

Bilateral filtre, kenarları koruyarak bir görüntüyü yumuşatmak için kullanılan bir görüntü işleme tekniğidir. Bu filtre görüntülerdeki gürültüyü azaltırken kenarları keskin tutma yeteneği sayesinde yaygın olarak kullanılır. Bilateral filtre burada görüntüdeki gürültüleri çok iyi bir şekilde ortadan kaldırması sebebiyle kullanılmıştır. Filtre hem karakterlerin içerisinde bulunan gürültüleri hemde plakanın beyaz kısmında bulunan gürültüleri hem de bazı plakaların alt kısmında bulunan yazılardan kaynaklanan istenmeyen gürültüleri başarıyla ortadan kaldırmıştır.


```
filtered_image=cv2.bilateralFilter(pic_cropped,11,17,17)
```

Şekil 17: Bilateral Filtre Kullanımı

Filtrenin aldığı ilk değer, filtrenin uygulanacağı komşu piksel sayısını ifade eder. Filtre uygulanırken piksellerin uzaklıklarını hesaba katarak hesaplama yapar. Yakın piksellere daha yüksek, uzak olan piksellere ise düşük ağırlıklar verilir. Diğer iki değer ise filtrenin ne kadar fazla uygulanacağını belirler. Yüksek değerler verilirse resim daha bulanık görünür. Daha düşük değerler verilirse daha az bulanık görünür. Filtrenin parametreleri için en uygun değerleri bulmak için farklı değerlerle denemeler yapılmıştır. En sonunda bu değerlerin olası en uygun değerlerden biri olduğu tespit edilmiştir.

3.4 Keskinleştirme Filtresi

Bilateral filtrenin en önemli özelliği yumuşatma yaparken aynı zamanda kenarların keskinliğini korumasıdır. Filtre bu konuda genel anlamda başarılı olmuş olsa da bazı rakamların tespiti konusunda sıkıntılar çıkarmıştır. Örneğin 3 sayısının ortasındaki çıkıntının boyutunun plakalarda küçük olması sebebiyle görüntü işleme sırasında bazı sorunlar ortaya çıkarmıştır. Bu sorunları minimize edebilmek için keskinleştirme filtresi uygulanmıştır.

```
sharp_filter=np.array([
    [0,-1,0],
    [-1,4.96,-1],
    [0,-1,0]
])
image=cv2.filter2D(filtered_image,ddepth=-1,kernel=sharp_filter)
```

Şekil 18: Keskinleştirme Filtresinin Uygulanması

Filtrenin çalışabilmesi için yukarıdaki şekilde bulunan ortadaki değerin uygun bir biçimde seçilmesi önem teşkil etmektedir. Özellikle diğer filtrelerin sonuçlarını ciddi biçimde değiştirdiği tespit edilmiştir. Normal şartlarda filtrenin düzgün bir biçimde çalışabilmesi için ortadaki değere dört veya üzerinde bir değer verilmesi gerekmektedir. Daha yüksek değerler verildikçe resimde bulunan kenarlar daha da keskinleşmektedir. Ancak bu uygulamada keskinleştirme filtresi diğer filtrelerle birleştğinde çok farklı sonuçlar elde edilmiştir. Örneğin ortadaki değere sekiz sayısı atandığı durumda plakalar neredeyse tamamen siyah renge dönmüştür. Bu sebeple en uygun değerin bulunması için defalarca deneme yapılmıştır. Bu denemeler sonucunda en uygun değerin 4,93 ile 5 arasında bir değer olduğu belirlenmiştir. Bu uygulamada ise 4,96 değeri kullanılmıştır.

3.5 Görüntüyü Gri Tonlarına Dönüştürme

BGR'den gri tonlamalıya dönüştürme işlemi, birçok görüntü işleme ve bilgisayarla görme uygulamasında ön işleme adımı olarak kullanılır. Gri tonlamalı görüntüler, veri miktarını azaltarak işlemeyi hızlandırır ve kenar tespiti, nesne algılama, segmentasyon gibi algoritmaların daha verimli çalışmasını sağlar. OpenCV'de bir görüntüyü, gri tonlamalı bir hale dönüştürmek için `cv2.cvtColor` fonksiyonu kullanılır.

```
pic= cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
```

Şekil 19: Görüntüyü Siyah-Beyaz Yapma

3.6 Otsu ile Thresholding (Eşikleme)

Otsu yöntemi, görüntü işleme ve bilgisayarla görme alanında yaygın olarak kullanılan bir eşikleme (thresholding) tekniğidir. Otsu ismi, bu tekniği geliştiren Nobuyuki Otsu'nun adından gelmektedir. Otsu metodu gri tonlamalı görüntüleri, uygun bir eşik tespit ederek siyah-beyaz bir hale getirir.

Aslında bir görüntüyü siyah-beyaz hale getirmek oldukça basit bir yöntemdir. Bunun için sadece sabit bir eşik değerinin belirlenmesi gerekmektedir. Ancak sabit bir eşik değeri farklı resimler üzerinde farklı etkiler yaratacaktır. Ortamın o anki ışık durumu, havanın güneşli veya bulutlu olması görüntüyü tamamen değiştirebilmektedir. Aşağıda örnek olarak iki görüntü gösterilmiştir.



Şekil 20: Örnek Resim-1



Şekil 21: Örnek Resim-2

İlk başta eşik değeri Örnek Resim-1’de bulunan plaka için uygun olacak şekilde seçilmiştir. Ancak aynı eşik değeri Örnek Resim-2 üzerinde uygulandığında oldukça kötü sonuçlar elde edilmiştir. Bu durumu çözmek için eşik değerini her resme göre otomatik olarak belirleyen otsu yöntemi kullanılmıştır.

```
imw=cv2.threshold(pic,None,200,cv2.THRESH_OTSU)[1]
```

Şekil 22: Otsu Yönteminin Kullanılması

Burada kullanılan 200 parametresi, tespit edilen eşik değerinin altında bulunan piksellere verilecek olan rengi temsil eder. Bu sayı ne kadar yüksek girilirse, eşik değerinin altında bulunan pikseller o kadar beyaz olur. Ne kadar düşük girilirse de o kadar siyah olur. Burada plakalardaki yazıların beyaz kısımdan daha net bir şekilde ayrışması için yüksek bir değer girilmiştir.

3.7 Başarısız Görüntü İşleme Sonuçları

Resimler üzerinde yapılan işlemler başlığı altında buraya kadar olan kısımda projede kullanılan işlemler açıklanmıştır. Bu bölümde ise şu ana kadar denenmiş ve başarısız olan yöntemlerden bazılarının sonuçları belirtilmiştir.



Şekil 23: Başarısız Sonuç-1



Şekil 24: Başarısız Sonuç-2



Şekil 25: Başarısız Sonuç-3



Şekil 26: Başarısız Sonuç-4



Şekil 27: Başarısız Sonuç-5

3.8 En Başarılı Görüntü İşleme Sonucu

Bu bölümde şimdiye kadar yapılmış olan denemeler sonucunda en başarılı olmuş yöntemin sonuçları incelenmiştir. Şimdiye kadar anlatılmış olan görüntü işleme adımlarının sonuçları görsel olarak sunulmuştur. Aşağıda bulunan girdi görselinde YOLO'nun tespit etmiş olduğu görüntü gözükmemektedir. Girdi görseline, daha önce açıklanmış olan filtre ve yöntemler uygulanarak çıktı görseli elde edilmiştir.



Şekil 28: Girdi Görseli



Şekil 29: Çıktı Görseli

Şekillerde görüldüğü üzere plakadaki karakterlerin daha rahat bir şekilde okunabilmesi için plaka büyütülmüş ve daha okunaklı bir hal almıştır.

4. RESİMDEKİ KARAKTERLERİN OKUNMASI

Görüntü işleme sonucu elde edilen görüntüler üzerinden plakaların okunması gerekmektedir. Resimdeki karakterleri okumak için optik karakter tanıma (OCR) teknolojisini kullanılmıştır. Optik Karakter Tanıma (Optical Character Recognition, OCR), basılı veya el yazısıyla yazılmış metinleri dijital formata dönüştürmek için kullanılan bir teknolojidir. OCR, bir belgeyi tarayarak veya bir görüntüden karakterleri tanıyarak dijital metin oluşturur. Böylece bu metin düzenlenebilir, aranabilir ve elektronik olarak saklanabilir hale getirilir.

4.1 Gerekli kütüphaneler

Bu projede üç farklı OCR kütüphanesi denenerek performansları karşılaştırılmıştır. Bu kütüphanelerden ilki PyTesseract'dır. Literatür araştırması bölümünde incelenmiş olan makalelerin biri hariç hepsinde karakter tanınması için PyTesseract kütüphanesi kullanılmıştır ve genel olarak güzel sonuçlar elde edilmiştir. Bunun yanı sıra kurulum kısmı, diğer kütüphanelere kıyasla çok daha zahmetli olduğundan performans konusunda yüksek bir beklenti oluşturmaktadır. Diğer iki kütüphane ise EasyOCR ve docTR (Document Text Recognition)'dir. Her iki kütüphane de belgeleri ve resimleri işlemek için derin öğrenme tekniklerini kullanır. Resimlerden metin çıkarmak için modern görüntü işleme ve makine öğrenimi tekniklerini kullanarak oldukça doğru sonuçlar sağlarlar. Aşağıdaki görsellerde her üç kütüphanenin de uygulanması gösterilmiştir.

```
#easyocr
reader=easyocr.Reader(['en'],gpu=True)
result=reader.readtext(path)
print(result)
```

Şekil 30: EasyOCR Kullanımı

```
#docTR
model=ocr_predictor(pretrained=True)
Document=DocumentFile.from_images(path)
result=model(Document)
print(result)
```

Şekil 31: docTr Kullanımı

```
#tesseract
pyt.pytesseract.tesseract_cmd= "C:/Program Files/Tesseract-OCR/tesseract.exe"
result=pyt.image_to_string(image)
print(result)
```

Şekil 32: PyTesseract Kullanımı

4.2 Sonuçlar

DocTR ve EasyOCR'ın ürettikleri sonuçların, projede görüntü işleme tekniklerinin değişmesiyle birlikte sürekli olarak değiştiği tespit edilmiştir. docTR, okunması nispeten zor olan ve çeşitli gürültüler içeren görseller üzerinde EasyOCR'a göre, karakter tanıma konusunda daha başarılı olduğu tespit edilmiştir. docTR ayrıca tespit etmiş olduğu karakteri daha yüksek confidence (güven) değerleriyle tespit etmektedir. Bunun yanında docTR'nin bu tür okunması zor olan görseller üzerinde bir handikapı bulunmaktadır. docTr tespit etmiş olduğu karakterleri doğru bir şekilde ve yüksek confidence oranlarıyla tespit edebilse bile zaman zaman bu karakterlerin sıralarını karıştırdığı tespit edilmiştir. Bu durum plaka tespiti konusunda bir sıkıntı oluşturmaktadır. Diğer yandan EasyOCR'da bu tarz sorunlarla karşılaşılmamıştır.

PyTesseract'ın, görüntü işleme tekniklerinin değişimiyle birlikte sonuçlarda nasıl değişiklikler olduğu tespit edilememiştir. Bunun nedeni PyTesseract'ın config ayarlarının değiştirilmeden önce hiçbir sonuç üretememesidir. Config ayarlarının daha sonra yeniden yapılandırılmasıyla birlikte PyTesseract kütüphanesi de sonuçlar üretmeye başlamıştır. Ancak bu durum projenin son halini almasıyla birlikte gerçekleşmiştir.

Projenin son halini almasıyla birlikte her üç OCR kütüphanesinin sonuçları karşılaştırılmıştır. Her üç kütüphaneye de toplam 20 resim gösterilmiştir. Gösterilen resimlerin bazıları güneşli bazıları ise bulutlu havada çekilmiştir. Elde edilen sonuçlar ise şu şekildedir: EasyOCR kendisine gösterilen 20 plakanın 11(%55)'ini doğru bir şekilde tahmin etmiştir. Plakalarda bulunan 169 karakterden ise 157'sini doğru bir şekilde tahmin etmiştir. docTR ise kendisine gösterilen 20 plakadan yine aynı şekilde 11(%55)'ini doğru bir biçimde tahmin etmiştir. Plakalardaki 169 karakterden ise 160'ını doğru bir biçimde tahmin etmiştir. Beklentinin en yüksek olduğu PyTesseract ise 20 plakadan 12(%60)'sini doğru bir şekilde tahmin etmiştir. 169 karakterden ise 159'unu doğru biçimde tahmin etmiştir.

Başarı oranlarındaki bu düşüklüğün en önemli sebebinin, plakanın sol kısmında bulunan mavi alan ve o alanda bulunan "TR" yazısının ve simgelerden kaynaklandığı tespit edilmiştir. Her üç kütüphane de buradaki gürültüleri zaman zaman birer karakter olarak algılamıştır. Bu hatanın ortadan kaldırılmış olması farz edilerek elde edilen sonuçlar ise şu şekilde olacaktır: Başarı oranı EasyOCR için %75'e, docTR için %70'e, PyTesseract içinse %85'e çıkmaktadır. Buradan karakter okumadaki başlıca sorunun bu mavi alandan kaynaklandığı tespit edilmiştir.

5. SONUÇ VE ÖNERİLER

Genel olarak programın başarısına tekrar bakmak gerekirse uygulamanın resimler üzerindeki plakaları oldukça başarılı bir şekilde tespit ettiği görülmektedir. Görüntülerin işlenmesi sırasında ise fena olmayan sonuçlar elde edilmiştir. Ancak plakada bulunan karakterlerin okunması sırasında sorun yaşanmaktadır. Şimdiye kadar denenmiş olan en başarılı kütüphane olan PyTesseract ile %60'lık bir başarı elde edilmiştir. Ayrıca sorunun kaynağına inildiğinde, plakaların sol tarafında bulunan mavi alanın görüntüden atılması gerektiği ve atıldığı takdirde %85'lik bir başarı yakalanacağı tespit edilmiştir.

KAYNAKLAR

- 1kodum*. (tarih yok). 1kodum: <https://1kodum.com/gaussian-blur/> adresinden alındı
- A. Mohammed Shariff, R. B. (2021, 04 20). Vehicle Number Plate Detection Using Python and Open CV. *IEEE*.
- Aksu, B. (2022, 03 03). *siberegitmen*. siberegitmen: <https://www.siberegitmen.com/opencv-nedir-neler-yapilabilir/> adresinden alındı
- Atasoy, H. (2011, 07 17). *atasoyweb*. atasoyweb: <https://www.atasoyweb.net/Otsu-Esik-Belirleme-Metodu> adresinden alındı
- İLERİ, A. (2018, 08 28). *Medium*. Medium: [https://abdulsamet-ileri.medium.com/g%C3%B6r%C3%BCnt%C3%BC-filtrelerini-uygulama-ve-kenarlar%C4%B1-alg%C4%B1lama-21d42f194db4#:~:text=Bilateral%20\(%C4%B0ki%20taraf%C4%B1\)%20Filtreling,g%C3%BCr%C3%BClt%C3%BC%20azalt%C4%B1c%C4%B1%20bir%20yumu%C5%9Fatma%20fi](https://abdulsamet-ileri.medium.com/g%C3%B6r%C3%BCnt%C3%BC-filtrelerini-uygulama-ve-kenarlar%C4%B1-alg%C4%B1lama-21d42f194db4#:~:text=Bilateral%20(%C4%B0ki%20taraf%C4%B1)%20Filtreling,g%C3%BCr%C3%BClt%C3%BC%20azalt%C4%B1c%C4%B1%20bir%20yumu%C5%9Fatma%20fi) adresinden alındı
- Manoj Kumar, S. S. (2022, 07 29). High-Security Registration Plate Detection using OpenCV and Python. *IEEE*.
- Mesci, Y. (2019, 04 29). *Medium*. Medium: <https://medium.com/deep-learning-turkiye/yolo-algoritmas%C4%B1n%C4%B1-anlamak-290f2152808f> adresinden alındı
- Michael Kročka, P. D. (2022, 10 12). Automatic License Plate Recognition Using OpenCV. *IEEE*.
- Milan Samantaray, A. K. (2022, 01 20). Optical Character Recognition (OCR) based Vehicle's License Plate Recognition System Using Python and OpenCV. *IEEE*.
- V. Gnanaprakash, N. K. (tarih yok). Automatic number plate recognition using deep learning.

ÖZGEÇMİŞ



Ad-Soyad : Mehmet Mert Fidan
Doğum Tarihi ve Yeri : 12.07.2000 Bahçelievler/İstanbul
E-posta : mehmetmertfidan@gmail.com

BİTİRME ÇALIŞMASINDAN TÜRETİLEN MAKALE, BİLDİRİ VEYA SUNUMLAR:

-
-
-