GERÇEK ZAMANLI NESNE TANIMA SİSTEMLERİNDE MİCRO SERVİS YAKLAŞIMI İLE PERFORMANS ARTIRIMI

Mehmet Ali Serttaş

111450103

DOKTORA TEZİ

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Doktora Programı

Danışman: Prof. Dr. Emin Murat Esin

İstanbul

T.C. Maltepe Üniversitesi

Lisansüstü Eğitim Enstitüsü

Ekim,2021

GERÇEK ZAMANLI NESNE TANIMA SİSTEMLERİNDE MİCRO SERVİS YAKLAŞIMI İLE PERFORMANS ARTIRIMI

Mehmet Ali Serttaş

111450103

Orcid: 0000-0001-5833-0158

DOKTORA TEZİ

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Doktora Programı

Danışman**:** Prof. Dr. Emin Murat Esin

İstanbul

T.C. Maltepe Üniversitesi

Lisansüstü Eğitim Enstitüsü

Ekim,2021



JÜRİ VE ENSTİTÜ ONAYI

Bu belge, Yükseköğretim Kurulu tarafından 19.01.2021 tarihli "Lisansüstü Tezlerin Elektronik Ortamda Toplanması, Düzenlenmesi ve Erişime Açılmasına İlişkin Yönerge" ile bildirilen 6689 Sayılı Kişisel Verilerin Korunması Kanunu kapsamında gizlenmiştir.

ETİK İLKE VE KURALLARA UYUM BEYANI

Bu belge, Yükseköğretim Kurulu tarafından 19.01.2021 tarihli "Lisansüstü Tezlerin Elektronik Ortamda Toplanması, Düzenlenmesi ve Erişime Açılmasına İlişkin Yönerge" ile bildirilen 6689 Sayılı Kişisel Verilerin Korunması Kanunu kapsamında gizlenmiştir.

TEŞEKKÜR

Bu tez çalışması süresi boyunca her zaman anlayışla ve sabırla yol gösteren, bilimsel katkı sunan, desteklerini ve yardımlarını esirgemeyen değerli danışman hocam Sn. Prof. Dr. Emin Murat Esin'e çok teşekkür ederim. Ayrıca Maltepe Üniversitesi Bilgisayar Mühendisliği Bölümü hocalarıma ve arkadaşım Mustafa Çakır'a teşekkürlerimi sunarım.

Akademik hayatımın her aşamasında beni yalnız bırakmayan ve çalışmalarımda başarılı olmamı sağlayan sevgili eşime ve çocuklarıma, manevi desteklerini her daim hissettiğim anne ve babama sonsuz teşekkür ederim.

Mehmet Ali Serttaş

Ekim,2021

ÖZ

GERÇEK ZAMANLI NESNE TANIMA SİSTEMLERİNDE MİCROSERVİS YAKLAŞIMI İLE PERFORMANS ARTIRIMI

Mehmet Ali Serttaş
Doktora Tezi
Bilgisayar Mühendisliği Anabilim Dalı
Bilgisayar Mühendisliği Doktora Programı
Danışman**:** Prof. Dr. Emin Murat Esin
Maltepe Üniversitesi Lisansüstü Eğitim Enstitüsü, 2021

Bu çalışma ile belli bir güzergâhta yer alan binalara yerleştirilmiş tabelaların varlık ve boyutlarının tespitine yönelik gerçek zamanlı, mikro servis mimarisine dayalı yeni bir sistem tasarlanmış ve uygulaması gerçekleştirilmiştir.

Sistem ile gezici bir platforma yerleştirilmiş video kameradan elde edilen anlık görüntü çerçevelerinde sınıflandırma işlemi yapılarak belli sınıflardan birisine ait olma durumları belirlenmektedir. Bu amaçla geliştirilmiş SSD Mobile Net ve Fast R-CNN modeli, ekran kartının tahsisli bir alanında çalışarak amaç nesnelerin çerçevelenmesini sağlamakta ve görüntüyü base64stringe çevirmektedir. Bu veri Mask R-CNN modeline aktarılarak tabelanın genelleştirilmiş koordinatları ve piksel cinsinden alanı hesaplanmaktadır.

Bu işlem boyunca görüntü akışı sürdüğünden geleneksel olarak nesne tanıma, alan taraması ve boyutlandırılması işlemlerinin artarda yapılması yerine bağımsız mikroservis birimlerinde paralel işlenmesi ile zaman kazandıran ve çerçeve kaybettirmeyen yeni bir yapı gerçekleştirilmiştir. Böylece görüntü akışının kesintisiz sürmesi gereken sistemlerde süreç döngüsünün önemli ölçüde hızlandırılması mümkün olabilmiştir

Sonuç olarak bu çalışmada, hareketli kamera ile anlık olarak elde edilen görüntünün tanınması amacıyla görüntü işleme teknikleri ve derin öğrenme algoritmaları kullanılarak geliştirilen yazılımda, görüntü gerçek zamanlı olarak 24 FPS hızda işlenerek %90 doğruluk başarımı elde edilmiştir. Böylece süreç önemli ölçüde hızlandırılabilmiştir.

Anahtar kelimeler: Nesne tanıma, derin öğrenme, görüntü işleme, Tensorflow, SSD Mobile Net, Fast R-CNN, Mask R-CNN, Mikroservis, Redis.

ABSTRACT

REAL-TIME OBJECT DETECTION SYSTEMS PERFORMANCE INCREASE WITH MICROSERVICE APPROACH

Mehmet Ali Serttaş
PhD Thesis
Department of Computer Engineering
Computer Engineering Programme
Thesis Advisor: Prof. Dr. Emin Murat Esin
Maltepe University Graduate School, 2021

In this study, a new system based on real-time, micro service architecture was designed and implemented for the recognition of the signs placed in the buildings located on a certain route.

System; In the snapshot frames obtained from the video camera placed on a mobile platform, the classification process is performed and the status of belonging to one of the certain classes is determined. The SSD and Fast R-CNN model developed for this purpose works in a dedicated area of the graphics card, providing framing of the object objects and converting the image to base64 string. By transferring this data to the Mask R-CNN model, the generalized coordinates of the sign and its area in pixels are calculated.

Since the picture flow continues throughout this process, in order to save time and not to waste frames, the traditionally sequential object recognition, area scanning and sizing operations are processed in parallel in independent microservice units. Thus, it was possible to prevent frame losses and speed up the process cycle significantly in systems that require uninterrupted picture flow.

As a result, in this study, in the software developed by using image processing techniques and deep learning algorithms for the purpose of recognizing the image instantly obtained with the moving camera, the image is processed in real time at 24 FPS (Frame Per Second: the number of pictures per second), an accuracy of 90% was achieved. Thus, the completion time of the process was significantly accelerated.

Keywords: Object Detection, Deep Learning, İmage Processing, Tensorflow, SSD Mobile Net, Fast R-CNN, Mask R-CNN, Microservice, Redis.

İÇİNDEKİLER

JÜRİ VE ENSTİTÜ ONAYI ii
ETİK İLKE VE KURALLARA UYUM BEYANI iii
TEŞEKKÜR iv
ÖZ v
ABSTRACT vi
İÇİNDEKİLER vii
TABLOLAR LİSTESİ ix
ŞEKİLLER LİSTESİ x
KISALTMALAR xii
ÖZGEÇMİŞ xiii
BÖLÜM 1. GİRİŞ 1
1.2. Literatür Araştırması 3
BÖLÜM 2. ÖNBİLGİ VE TEMEL KAVRAMLAR 7
2.1. Derin Öğrenme 7
2.2. Derin Öğrenme Modelleri ve Mimarileri 8
2.2.1. Denetimli Öğrenme 8
2.2.1.1. Konvolüsyonel Sinir Ağları (Convolutional Neural Networks) (CNN) . 9
2.2.2. Tekrarlayan Sinir Ağı (Recurrent Neural Networks) (RNN) 12
2.2.3. Uzun Kısa Dönemli Bellek (Long / Short Term Memory) (Lstm) 14
2.2.4. ..... Kısıtlanmış (Derin) Boltzmann Makineleri (Boltzmann Machines) (DBM) 15
2.2.5. Derin İnanç Ağları (Deep Belief Networks) (DBN) 16
2.2.6. Oto Kodlayıcılar (Autoencoders)(AE) 17
2.3. Derin Öğrenme Kütüphaneleri 19
2.3.1. Tensorflow 19
2.3.2. Keras 19
2.3.3. Theano 19
2.3.4. Torch 20
2.3.5. Caffe 20
2.3.6. Deeplearnıng4j 20
2.3.7. Mxnet 20
2.3.8. Matconvnet 20
2.4. Nesne Tanıma 21
2.4.1. SSD MOBİLE NET 24
2.4.2. R-CNN 25
2.4.3. FAST R-CNN 25
2.4.4. FASTER R-CNN 26

2.4.5. MASK R-CNN 27
2.5. Mikroservis Mimarisi 29
2.5.1. Mikroservis Tanımı 29
2.5.2. Mikro Servislerin Temel Özellikleri 30
2.5.3. Mikroservis Mimarisinin Avantajları 31
2.5.3.1. Teknoloji Çeşitliliği 31
2.5.3.2. Esneklik 31
2.5.3.3. Ölçekleme 32
2.5.3.4. Deployment Kolaylığı 32
2.5.3.5. Organizasyonların Düzenlenmesi 33
2.5.4. Mikro servislerin eksiklikleri ve dikkat edilmesi gereken noktaları 33
2.5.5. REDIS ve CUDA Mimarileri 34
BÖLÜM 3. YÖNTEM 38
3.1. Veri Seti Oluşturulması 40
3.2. Veri Etiketleme 41
3.3. Eğitim Verisi Oluşturma 42
3.4. Label Map Oluşturma 43
3.5. Eğitim Ayarlarının Yapılması 43
3.6. Eğitimin Gerçekleştirilmesi 44
3.7. Uygulama Sonuçlarının Ekran Görüntüsü 44
3.8. Mimari Tasarım Ve Uygulama 48
3.8.1. Mikroservis Mimari Tasarımı 48
3.8.2. Mikroservis Mimarisinin Uygulanması 48
3.8.2.1. Redis Konfigürasyonu 48
3.8.2.2. CUDA Konfigürasyonu 51
3.9. Programın Çalışması ve Ekran Görüntüleri 52
3.10. Mimarinin Doğrulanması ve Geçerlilik Testleri 58
3.11. Deneysel Sonuçlar 60
3.11.1. Doğruluk Oranı Karşılaştırması 60
BÖLÜM 4. SONUÇ 62
KAYNAKÇA 64

TABLOLAR LİSTESİ

Tablo 1. Bilgisayar sistemi performans karşılaştırması 5
Tablo 2: YOLO ve SSD arasındaki farklılıklar 5
Tablo 3: Üç modelin performans analizi 6
Tablo 4: Derin Öğrenme Modelleri Doğruluk ve Parametre Karşılaştırması..... 11
Tablo 5: Mikroservis Mimarisi ile Klasik Mimari Arasındaki Farklılıklar Tablosu 60
Tablo 6: Mikroservis mimari sistemi performans karşılaştırması 60

ŞEKİLLER LİSTESİ

Şekil 1. Derin Öğrenme Mimarileri Şema Gösterimi 8
Şekil 2: Havuzlama İşlemi (Gürbüz, 2020) 9
Şekil 3: Evrişimsel Sinir Ağı Yapısı (Ergin, 2018) 10
Şekil 4: RNN Mimarisi (Gürbüz, 2020) 13
Şekil 5: RNN Uygulama Örneği (Gürbüz, 2020) 13
Şekil 6: LSTM Yapısı (Gökçe & Sonugür, 2016) 14
Şekil 7: Boltzman Makinesi (Hinton G. E., 2012) 15
Şekil 8: Kısıtlı Boltzmann Makinesi (Hinton G. E., 2012) 16
Şekil 9: Art Arda Eklenmiş RBM (Hinton G. E., 2012) 17
Şekil 10: AE Yapısı (KAYAALP, 2018) 18
Şekil 11 : DAE Yapısı (KAYAALP, 2018) 18
Şekil 12: Görüntü Tanıma ile Nesne Tanıma farklılığın gösterimi 22
Şekil 13: SSD Mobile Net mimarisi (Arabi, 2019) 24
Şekil 14 : R-CNN Mimarisi (Girshick R. D., 2014) 25
Şekil 15 : Fast R-CNN Mimarisi (Girshick R. , 2015) 26
Şekil 16: Faster R-CNN Mimarisi (Ren, 2016) 27
Şekil 17 : Mask R-CNN Mimarisi (Bharati, 2019) 28
Şekil 18: Monolitik Mimari Şeması (microsoft, 2019) 29
Şekil 19: Mikro servis Mimarisi Şeması (derek, 2019) 30
Şekil 20: Monolitik ve Mikroservis Mimarilerde Yazılım Dağıtım Şeması (Kızıltan, 2018) 32
Şekil 21 : GPU ve CPU mimarileri (Parsons, 2018) 36
Şekil 22: Temsili CPU ve GPU karşılaştırması (Cheng, 2014) 37
Şekil 23: Görüntü tanımada yeni nesil dağıtık mimari diyagramı 40
Şekil 24: Veri seti için toplanan resimler 41
Şekil 25: LabelImg programı ile etiketleme 41
Şekil 26: Via Labelling Program ile Etiketleme 42
Şekil 27: Via ile Etiketlemede x ve y koordinatlarının CSV gösterimi 42
Şekil 28: Mask R-CNN Eğitim Grafiği 44
Şekil 29: Video üzerinden yakalanan görüntünün SSD Mobile Net test sonucu, doğruluk oranları 45

Şekil 30: Fast R-CNN test veri girişi resim örneği 45
Şekil 31: Fast R-CNN test sonucu, doğruluk oranları 46
Şekil 32: Fast R-CNN test veri girişi resim örneği çapraz açı 46
Şekil 33: Fast R-CNN Test Veri Sonucu 47
Şekil 34: Mask R-CNN Test veri sonucu 47
Şekil 35: Mikroservis mimari tasarımı 48
Şekil 36: REDIS Mimarisi kod yapısı 50
Şekil 37: Redis'in aktif edilmesi 56
Şekil 38: Fast R-CNN modelinin servisinin çalıştırılması 56
Şekil 39: Mask R-CNN modelinin servisinin çalıştırılması 57
Şekil 40: SSD Mobile Net modelinin servisinin çalıştırılması 57
Şekil 41: Object Detection çıktı örneği 1 58
Şekil 42: Object Detection çıktı örneği 2 58
Şekil 43: Fast R-CNN Doğruluk (Accuracy) Değeri Grafiği 59
Şekil 44: Mask R-CNN Doğruluk (Accuracy) Değeri Grafiği 59
Şekil 45: Klasik Mimari ile Doğruluk Oranı Grafiği 61
Şekil 46: Mikroservis Mimarisi ile Doğruluk Oranı Grafiği 61

KISALTMALAR

CNN : Convolutional Neural Network

RNN : Recurrent neural network

CUDA : Compute Unified Device Architecture

GPU : Graphics Processing Unit

GPS : Global Positioning System

GDY : Geometrik Dönüşüm Yöntemi

ÇFY : Çerçeve Farkı Yönetimi

YSA : Yapay Sinir Ağı

DL : Deep Learning

MİB : Merkezi İşlem Birimi

MNIST : Modified National Institue of Standarts

LSTM : Long Short-Term Memory

BM : Boltzman Makinası

DBN : Deep Belief Networks

DAE : Denoising Autoencoders

DRL : Deep Reinforcement Learning

SVM : Destek Vektör Makinası

ÖZGEÇMİŞ

Mehmet Ali Serttaş

Bilgisayar Mühendisliği Anabilim Dalı

EĞİTİM

Derece	**Yıl Üniversite, Enstitü, Anabilim/ Anasanat Dalı**
Y.Lisans	2009 Haliç Üniversitesi Fen Bilimleri Enstitüsü/Yönetim Bilişim Sistemleri
Lisans	2006 Haliç Üniversitesi Mühendislik Fakültesi/Bilgisayar Mühendisliği
Önlisans	2002 İstanbul Üniversitesi Teknik Bilimler Meslek Yüksek Okulu Bilgisayar Programcılığı

İŞ DENEYİMİ

Kurum	**Görev**	**Yıl**
Sultangazi Belediyesi	Bilgi işlem Şefi	2017-Devam
Sultangazi Belediyesi	SOME Ekip Lideri	2016-2019
Sultangazi Belediyesi	Yazılım Birim Sorumlusu	2013-2017
İSMEK	Bilişim Teknolojileri Proje Sorumlusu	2010-2013
İSMEK	Bilgisayar Branşları Rehberi	2002–2010
Medyasoft	Eğitim Danışmanlığı	2010-2011
Haliç Üniversitesi	Veritabanı Yönetim Sistemleri Dersi Asistanlığı	2005-2006
Obje Bilgisayar	Web Master	2001–2002

BÖLÜM 1. GİRİŞ

Görüntü işleme, bilgisayarların görüntüler ve videolardaki nesneleri tanımasını sağlayan bir bilgisayar bilimidir. Görüntü işlemenin nesne tespiti ve nesne tanıma olarak 2 ana işlevi vardır. Nesne tespiti ve nesne tanıma konuları yıllardır üzerinde çalışılmakta olup görüntü işlemenin en önemli unsurlarındandır ve bu alanda birçok farklı algoritmalar ve yöntemler geliştirilmiştir. 2001 yılında Paul Viola ve Michael Jones tarafından dijital görüntülerdeki nesne sınıflarını tespit etmek için geliştirilen ilk algoritma Viola Jones algoritması olmuştur. Sonraki yıllarda daha fazla doğruluk oranı ile nesne tespiti ve tanımlama yapılabilmesi, derin öğrenme ve grafik işleme birimlerindeki gelişmeler sayesinde olmuştur.

Literatürde nesne takibi genel olarak dört farklı süreçte ele alınmaktadır. Bu süreçler ön analiz, nesne tespiti, nesne sınıflandırma ve nesne izlemenin belirlenmesinden oluşmaktadır. Bu aşamalardan nesne tespiti diğerlerine göre zorluk taşımaktadır ve bu aşamanın başarısı nesne sınıflandırma ile nesne takibi aşamalarının başarısını olumlu yönde etkilemektedir. Nesne tespiti tanımı; genel olarak bir görüntü dizisi veya videodaki nesnenin görüntü üzerindeki koordinatlarının bulunması, görüntünün belirginleşmesi ve işlenecek olan nesnenin arka plandan ayrılması olarak yapılabilir. Gerçek zamanlı ve gerçek zamanlı olmayan olarak ikiye ayrılır. Bölge önerileri (R-CNN, Fast R-CNN, Faster R-CNN, Mask R-CNN), tek atış çoklu kutu dedektörü (SSD) ve YOLO nesne tespiti ve tanımada en çok tercih edilen kütüphaneler arasındadır (Söylemez & Tarhan, 2018) .

Derin öğrenme, anormallik tespiti, görüntü tespiti, görüntü ve ses işleme, gelecek tahmini gibi uygulamalarda etkileyici başarılar elde eden, yaygın olarak kullanılan makine öğrenme tekniklerinden biridir. Derin öğrenme, genel olarak Yapay Zekâ ve Akıllı Sistemlere ilginç ve kavramsal ve aynı zamanda algoritmik olarak eldeki son teknolojiye uygun bir yaklaşım sunar. Bu değerler dizisi, makine çevirisi, bilgisayarla görme, doğal dil işleme, finansal ve sağlık gibi çok sayıda alana uygulanmıştır. Makine öğreniminin bir alt kümesi olarak kabul edilen derin öğrenme, makine öğrenimi sürecini verimli bir şekilde yürütmek için yapay sinir ağlarının bir hiyerarşi düzeyini kullanır.

Derin öğrenme, birden çok temsil düzeyine sahip bir grup makine öğrenimi yöntemleri sınıfıdır. Genel olarak, tüm derin öğrenme yöntemlerinin sınıflandırılmaları aşağıdaki gibi üç farklı kategoridedir.

1. Evrişimli Sinir Ağları (CNN'ler)
2. Önceden Eğitilmiş Denetimsiz Ağlar (PUN'lar)
3. Tekrarlayan / Özyinelemeli Sinir Ağları (RNN'ler)

Evrişim Sinir Ağı (CNN), görüntü analizi için tasarlanmış derin bir sinir ağıdır. CNN, görüntü sınıflandırma, nesne algılama, görüntü tanıma ve benzerlerinin ana kategorisidir. Girdi alır ve çıktıyı sınıflandırır. Bunun yanı sıra 2014'te nesne algılamada verimli nesne yerelleştirme sorununu çözmek için R-CNN, Ross Girshick ve diğerleri tarafından önerilmiştir. R-CNN'de görüntünün 2000 bölgeye bölünmesi tavsiyesi nedeniyle R-CNN de eğitim çok zaman alacak ve bunun maliyeti yüksek olacaktır. R-CNN'in bu sorununu çözmek için Fast R-CNN önerilmektedir. Fast R-CNN'ni farklılaştıran en önemli özelliği, 2000 bölge önerisini her seferinde evrişimli sinir ağına beslemek zorunda kalmamasıdır. Bu da R-CNN'den daha hızlı olmasını sağlamaktadır. Geri besleme yerine evrişim işlemi yalnızca bir kez yapılır ve ondan bir özellik haritası oluşturulur (Gandhi, 2018). Fast R-CNN, farklı nesnelerin konumlarını doğru ve hızlı bir şekilde tahmin edebilir. Bu nedenle özellikle derin öğrenme için kullanılmakta olan Fast R-CNN bu çalışmada da tercih edilmiştir.

Mask R-CNN görüntü segmentasyonu açısından kullanılan son teknolojidir. Bir görüntüdeki nesneleri algılar ve her örnek için yüksek kaliteli bir segmentasyon maskesi oluşturur. Nesnenin bulunduğu yere dikdörtgen çizerek sınırlayıcı kutuyu belirler. Ayrıca nesnenin resimde kapladığı tüm pikseller tespit ederek sınıflandırır. Resim üzerinde belirlenen nesneye ait tüm pikselleri işaretleyerek görselleştirmek için maskelemeyi yapar. Bu nedenle bu çalışmada amaçlanan nesnenin alan hesaplaması yapılabilmesi için Mask R-CNN modeli kullanılmıştır.

SSD Mobile Net model, nesne tespitinde diğer derin öğrenme modellerine göre çok daha hızlı çalışmaktadır. SSD mimarisi sınırlayıcı kutu konumlarını tahmin etmeyi ve bu konumları tek geçişte sınıflandırmayı öğrenen tek bir evrişim ağıdır. Bu nedenle n düşük donanımlı cihazlarda bile hızlı bir şekilde nesne tanıma yapabilmektedir. Bu çalışmada gezici bir platforma yerleştirilmiş video kameradan elde edilen anlık görüntü

çerçevelerinde hız önemli olduğu için gerçek zamanlı nesne tespiti ve tanıma işleminde SSD Mobile Net modeli tercih edilmiştir.

1.2. Literatür Araştırması

Literatürde ve uygulamada çok sayıda çalışmanın yapıldığı derin öğrenme ilk olarak Toronto Üniversitesi Profesörü Geoffrey Hinton tarafından ortaya atılmıştır. Hinton yayınlamış olduğu makalesindeki çalışmasında yapay sinir ağlarına getirdiği yeni bir yaklaşımı derin öğrenme olarak isimlendirmiştir. (Hinton G. E., 2006)

Bulut hibrit ortamdaki optimizasyon problemini çözmek için Derin Güçlendirmeli Öğrenme yöntemi olan (Multiple Buffer Deep Deterministic Policy Gradient) MB DDPG'yi öneren Lulu Chen ve arkadaşları MB DDPG'yi rastgele algoritma ve gerçek veri kümelerine ve bazı sentetik verilere dayanan genetik bir algoritma ile karşılaştırmışlardır. MB DDPG'nin ortamın karmaşık heterojenliğini belirleyebileceğini ve IoT cihazlarının ortalama bekleme süresini en aza indirebileceğini göstermeyi başardılar. Son olarak, ortamdaki taleplerin büyümesine rağmen, MB DDPG hala kıyaslamalardan daha iyi performans gösteriyor ve mükemmel ölçeklenebilirliği görülmüştür. (Chen, 2020)

Modular Microservice based GPU Utilization Manager with Gunicorn adlı çalışmada üretimde dağıtılan bir docker konteyner olarak entegre edilmiş bir GPU performans izleme aracı oluşturulmuştur. Mikroservis olarak geliştirildiğinden, herhangi bir kullanıcının görüntüyü docker sunucusundan çekmesi ve GPU performansını takip etmek için kendi makinesinde çalıştırmasının daha kolay hale getirilmiştir. Bu yöntem, derin sinir ağları için hiper parametre optimizasyonu arama sistemiyle ilgili iş yüklerinin dağıtılması ve yönetilmesinde kullanılan konteyner düzenleme araçlarından biri olan Kubernetes'e genişletilmeye olanak sağlamaktadır. Önerilen yöntem ile GPU kaynaklarının paylaşılmasında kilitlenme durumlarından kaçınarak paralel işlemeyi idare edecek şekilde genişletilebilirlik ortaya konmuştur. (Supriya, 2020)

2017 yılında Takuya Akiba ve arkadaşları, geliştirmiş olduğu ölçeklenebilir dağıtılmış derin öğrenme çerçevesi olan ChainerMN'nin tasarımını, uygulamasını ve değerlendirmesini yapmışlardır. Makalelerinde ChainerMN'nin ResNet-50 modelinin

öğrenme sürecini %90 paralel verimlilikle 128 GPU'ya kadar ImageNet veri kümesine ölçeklendirebileceğini göstermişlerdir. (Akiba, 2017)

Dijital kameraların gelişen sensör teknolojileri ile görüntülerin çözünürlüklerinin artması söz konusu olmuştur. Bu nedenle görüntü işlemede daha fazla işlemci gücüne ihtiyaç duyulmaktadır. 2020 yılında Refik Aydın ve arkadaşlarının "GPU programlamada CUDA platformu kullanılan paralel görüntü işleme çalışmalarının incelenmesi" adlı çalışmasında görüntü işleme sürecini hızlandırmak için paralel görüntü işleme teknikleri anlatılmaktadır. Bu çalışmada GPU ve CUDA teknolojilerini verimli bir şekilde kullanabilmesi açıklanmıştır. (Aydın, Refik, & Bay, 2020)

Mehmet Söylemez ve Ayça Tarhan'ın 2018 yılı bildirisi kapsamında mikroservis mimarisi ve temel faktörleri incelenmiş, mimarinin gerekleri ortaya konmuş ve monolitik mimariye göre faydaları ve zorlukları açıklanmıştır. Mikroservis mimarisini kullanacak ekiplerin faydalanabileceği faktörler literatürdeki birçok kaynak ve proje incelenerek belirlenmiştir. Bu mimari faktörler için teknoloji firmalarının ne gibi proje veya ürünler geliştirdikleri araştırılmış ve bu proje veya ürünler hakkında bilgi verilmiştir. Çalışmanın sonuçlarından biri de teknoloji firmalarının mikroservis mimarisi faktörlerinin çoğuyla ilgilendikleri ve bunlar için kapsamlı çözümler ürettikleri ortaya konulmuştur (Söylemez & Tarhan, 2018).

2018 yılında Ö. Mercanoğlu ve arkadaşlarının bildirisinde, hareketli bir kameradan elde edilen izleme videolarından hareketli nesneleri tespit etmek ve sınıflandırmak için aday bölge önerisi yöntemi sunulmuştur. Görüntü konumundaki ilgi noktaların hareketini yorumlanarak kamera hareketi tahmin edilmektedir. Ardından kamera hareketini ortadan kaldırarak hareketli nesnelere ait aday bölgeleri bulmaktadır. Önerilen yaklaşımın gözetim veri setinde değerlendirilmesi, yaklaşımın hareketli bir kamera ortamında hareketli nesnelerin sınıflarını belirlemede çok etkili olduğunu göstermektedir (Mercanoğlu Sincan, Yalım Keleş, & Tosun, 2018).

Barış Gökçe ve Güray Sonugür'ün 2016 yılındaki "GPS destekli imge çakıştırma yöntemi ile hareketli nesnelerin tespiti" adlı çalışmasında insansız kara araçlarının güzergâhları üzerinde bulunan hareket halindeki nesneleri bilgisayar görme sistemleri kullanarak tespit edebilmesi için GPS destekli bir sistem geliştirilmiştir. Bu çalışmanın deneysel sonuçlarında; sistemin çalışma hızı ile ilgili denemelerde bilgisayarın CPU

kullanım oranı %7 seviyelerine düştüğü gözlemlenmiştir. Deneysel çalışma sonunda elde edilen sonuçlar Tablo 1'de verilmiştir.

Tablo 1. Bilgisayar sistemi performans karşılaştırması

Yöntem	İmge çözünürlüğü	CPU yükü (%)	Çerçeve başına işlem süresi (sn)
GD[25]+ÇF[12]	3264x4928	%99	5.475
	480x640	%56	0.177
GD+Gauss [26]	3264x4928	%99	5.91
	480x640	%61	0.24
GY	3264x4928	%7	0.043

Kaynak: (Gökçe & Sonugür, 2016)

Tabloda GD; Geometrik Dönüşüm Yöntemini, ÇF; Çerçeve Farkı Yöntemini, GY ise geliştirilen yaklaşımı göstermektedir. (Gökçe & Sonugür, 2016)

YOLOv3 ve SSD algoritmalarının karşılaştırması çalışmasında video keşfi, yüz tanıma v.b. uygulamalarda hız, doğruluk, zaman, FPS, mAP ve bunların gerçek zamanlı uygulamalar için kullanılıp kullanılamayacaklarına ilişkin olarak YOLO ve SSD arasındaki karşılaştırmayı ortaya konulmuştur. Bu karşılaştırmayı tablo olarak gösterilmektedir. (Ambika Neelopant, 2021)

Tablo 2: YOLO ve SSD arasındaki farklılıklar

Parametre	**YOLO**	**SSD**
Hız	Düşük	Yüksek
Doğruluk	%80.3	%72.1
Süre	0.84~0.9	0.17~0.23
FPS	45	59
mAP	0.358	0.251

Kaynak : (Ambika Neelopant, 2021)

Rakkshab ve arkadaşları tarafından, YOLOv3, YOLOv5 ve SSD MobileNetV2 algoritmaları kullanılarak videolar ve gerçek zamanlı gözetim üzerinde maske tanıma uygulaması gerçekleştirilmiştir. Sonuçlar FPS cinsinden gerçek zamanlı olarak bir kişide yüz maskesi varlığını tespit etmek için üç algoritmanın performansı ortaya konmuştur.

Eğitilen modellerin her birinin performansı Tablo 3'de verilmiştir. (Rakkshab Varadharajan Iyer, 2021)

Tablo 3: Üç modelin performans analizi

Model	**mAP (%)**	**FPS**		
		TeslaT4	**1660 Ti**	**Jetson Nano**
YOLOv3	54.3	80	21	8
YOLOv5s	37.6	100	28	15
SSD MobileNet v2	33.7	94	26	15

Kaynak : (Rakkshab Varadharajan Iyer, 2021)

Tabloda görüldüğü üzere, YOLOv5s'nin hem doğruluk hem de fps'in optimal değerleri ile gerçek zamanlı durumlar için en uygun model olduğu sonucuna varılabilir. SSD Mobile Net v2'nin YOLOv5 benzer bir hız sağladığı görülmektedir. (Rakkshab Varadharajan Iyer, 2021)

BÖLÜM 2. ÖNBİLGİ VE TEMEL KAVRAMLAR

2.1. Derin Öğrenme

Derin öğrenme makine öğreniminin bir koludur. İnsan beyninin yapısal ve öğrenme özelliklerinden örnek alınarak tasarlanmıştır. Yapay sinir ağları üzerinde çalışan algoritmalar ve modeller kümesidir. Derin öğrenme algoritmaları artan karmaşıklık ve soyutlama hiyerarşisine dayanmaktadır. Ağırlıklı olarak belirli problemleri anlamaya odaklanılmış ve bu alanlarda birçok başarı elde edilmiştir. Derin öğrenme son yılların en önemli konuları olan görüntü işleme, nesne tanıma, ses tanıma ve doğal dil işleme gibi alanların endüstri, tıp, robotik, çeviri, gelecek tahmini ve finansal uygulamalarının akıllı çözümler üretilmektedir. Google, Facebook, Baidu, Microsoft gibi firmalar yapay zekâ konusunda yatırım yapmaktadır.

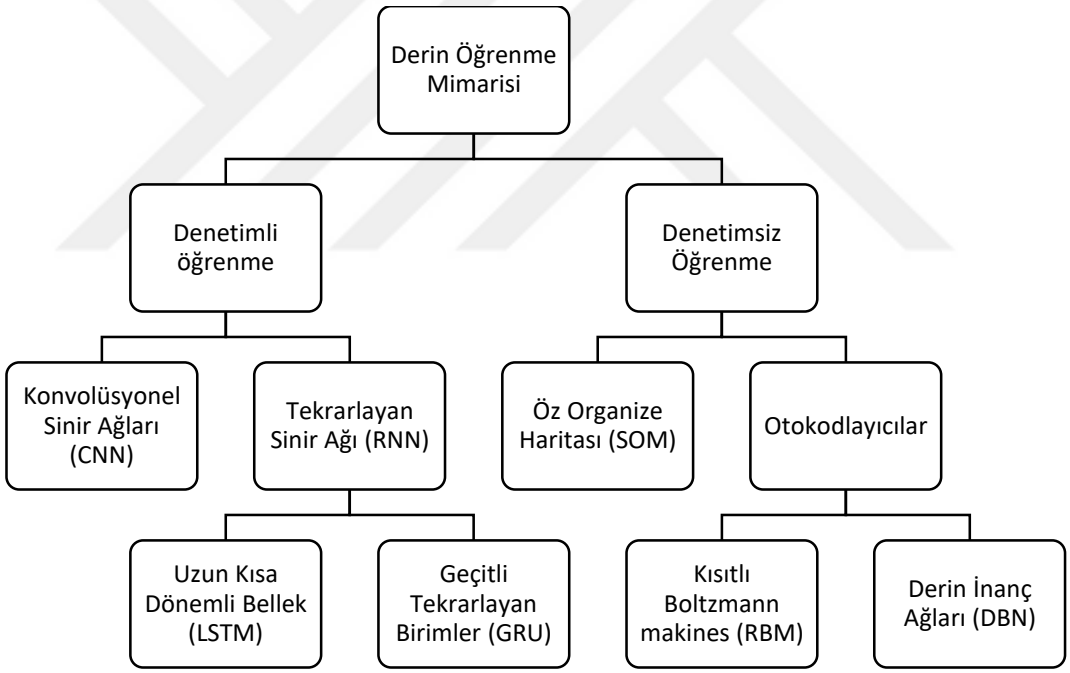
Derin öğrenme algoritmalarının, makine öğrenme algoritmalarından ayrılan en önemli yönü yüksek miktarda veriyi işleyecek, hesaplama gücü yüksek donanımlara ihtiyaç duymasıdır. Teknoloji alanında yaşanan gelişmeler derin öğrenmenin başarısını da artırmaktadır. Ancak yüksek işlem gücüne sahip cihazlar yüksek maliyetlere neden olmaktadır.

Derin öğrenme ile video analizi, görüntü sınıflandırma, ses tanıma v.b. gibi çeşitli uygulamalarda, yüksek bellek bant genişliği ve daha büyük veriyi aynı anda bellek birimleri arasında taşınmasına izin veren GPU tercih edilmektedir. Ancak paralel işlem gücünden faydalanmak adına tüm kodun GPU üzerine yüklenmesi dar boğaz oluşumuna neden olmaktadır. Bu durumda veri ön analizi için CPU tercih edilirken eğitim kodu GPU üzerinde çalıştırılmalıdır. Hem darboğaz oluşumunun önüne geçilmiş olunur hem de performans artışı görülecektir. Derin öğrenme, çok katmanlı algılayıcı yapısında bir yapay sinir ağı ile gerçekleştirilebilirken, konvolüsyonel yapay sinir ağları ile de gerçekleştirilebilir. Tüm bu açıklamalar ile birlikte, yapay sinir ağlarının eğitimi ve yürütülmesi yalnızca merkezi işlem biriminde değil grafik işlem biriminde de gerçekleştirilebilir (Supriya, 2020).

2.2. Derin Öğrenme Modelleri ve Mimarileri

Derin öğrenmenin arkasındaki temel mimari yapay sinir ağıdır (YSA). YSA ya bağlı olarak model ve mimarilerin çeşitli varyantları geliştirilmiştir. Geliştirilen yeni mimariler ve grafik işleme birimleri derin öğrenmeyi yapay zekânın ön saflarına getirmiştir. Derin öğrenmede kullanılan mimarilerin ve algoritmaların sayısı geniş ve çeşitlidir. Özellikle, uzun kısa süreli bellek (LSTM) ve evrişimli sinir ağları (CNN'ler) bu listedeki en eski yaklaşımlardan ikisidir.

Yapay sinir ağlarındaki katman sayılarının artırılmasıyla kurulan çok farklı türde derin öğrenme mimarileri bulunmaktadır. Derin öğrenme mimarilerini denetimli ve denetimsiz öğrenme olarak sınıflandırmak mümkündür. Kullanılan mimarilerin şema olarak aşağıdaki şekilde gösterilmektedir.



Şekil 1. Derin Öğrenme Mimarileri Şema Gösterimi

2.2.1. Denetimli Öğrenme

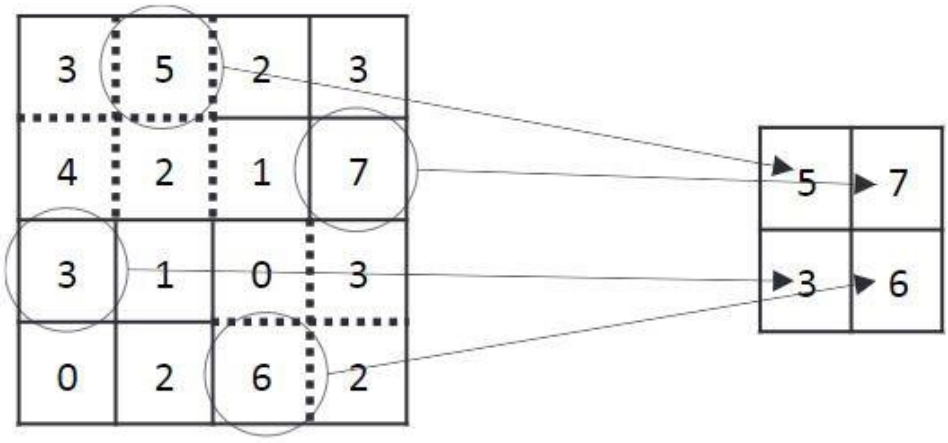
Denetimli öğrenme, tahmin edilecek hedefin eğitim için kullanılan veriler içinde açıkça etiketlendiği problem alanını ifade eder. Verileri sınıflandırmak veya sonuçları doğru bir şekilde tahmin etmek için algoritmaları eğitmek için etiketlenmiş veri kümelerinin kullanılmasıyla tanımlanır. Girdi verileri modele beslenirken, çapraz

doğrulama sürecinin bir parçası olarak gerçekleşen, model uygun şekilde takılana kadar ağırlıklarını ayarlamaktadır.

2.2.1.1. Konvolüsyonel Sinir Ağları (Convolutional Neural Networks) (CNN)

Görüntüdeki çeşitli obje veya nesneleri birbirinden ayrıştırmamıza yardımcı olan derin öğrenme algoritmasıdır. Hayvan görsel korteksinden biyolojik olarak ilham alan çok katmanlı bir sinir ağıdır. Mimari özellikle görüntü işleme uygulamalarında kullanışlıdır. İlk CNN, Yann LeCun tarafından oluşturulan, posta kodu yorumlama gibi el yazısı karakter tanımaya odaklanmış mimaridir. Konvolüsyonel sinir ağları, ilk katmanlar özellikleri (kenarlar gibi) tanır ve sonraki katmanlar bu özellikleri girdinin daha yüksek düzey niteliklerinde yeniden birleştirir.

CNN'ler daha az sayıda eğitime ve daha az parametreye gerek duymaktadırlar. Özellik çıkarma ve ardından sınıflandırma uygulayan birkaç katmandan oluşur. Örneğin bir konvolüsyonel katmandaki girdi x * y *z 3x3'lük bir resim düşünülebilir. Burada x görüntünün yüksekliğini, y görüntünün genişliğini, z ise kanalların sayısı göstermektedir. Konvolüsyonel katmanı ise x1 * y1 * z1 boyutlarında belirli bir sayıdaki filtreden oluşur. Filtre için görüntünün boyutundan daha küçük olan x1, y1 ve kanal değeriyle genellikle aynı seçilen z1 değeri seçilir. Bu filtreler ile her biri x-x1+1, y-y1+1 boyutunda k adet birbirine yerel olarak bağlı özellik haritaları üretir. Şekil 2'de verilen örnek gibi maksimum veya ortalama havuzlama ile 2x2 'lik alt örnekleme işlemi uygulanır (Gürbüz, 2020).

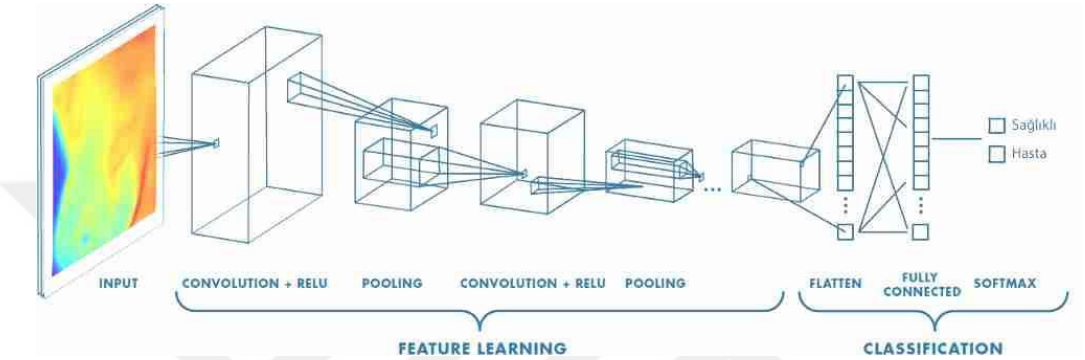


Şekil 2: Havuzlama İşlemi (Gürbüz, 2020)

İki boyutlu olan evrişimlerde birden fazla aşama vardır. Bu aşamalar şu şekilde sıralanır;

- Kenar Bulma
- Piksel Ekleme
- Kaydırma
- Havuzlama (Pooling)

Şekil 3'de bu aşamaların bulunduğu görsel verilmiştir.



Şekil 3: Evrişimsel Sinir Ağı Yapısı (Ergin, 2018)

Evrişimsel sinir ağlarından sonraki aşama model oluşturmadır. Modeller şu şekilde sıralanabilir;

- LeNet: 1998 yılında Yann LeCun , Leon Bottou , Yoshua Bengio ve Patrick Haffner tarafından " Gradient-Based Learning Applied To Document Recognition" adlı araştırma makalesinde tanıtıldı. LeNet'in amacı posta numaralarının üzerindeki sayıların okunmasıdır.
- AlexNet: Alex Krizhevsky tarafından tasarlanan evrişimli bir sinir ağı mimarisinin adıdır. AlexNet birbirini takip eden 11 katmandan oluşur.
  1) İlk katman, iki dolgulu 96 (11 × 11) kullanan bir evrişim katmanıdır.
  2) Daha sonra (55 × 55 × 96) bir çıktıyı girdi olarak kabul eden maksimum havuzlama katmanıdır.
  3) Üçüncü katman, (27 × 27 × 96) tensör olan çıktıyı girdi olarak kabul eden başka bir evrişimli katmandır.
  4) 2. Maksimum havuz katmanı
  5) 3. Evrişim Katmanı
  6) 4. Evrişim Katmanı

7) 5. Evrişim Katmanı

8) 3. Maksimum havuz katmanı

9) 1. Tamamen Bağlı Katman

10) 2. Tamamen Bağlı Katman

11) 3.Tamamen Bağlı Katman

- ZF Net: ZFNet klasik bir evrişimsel sinir ağıdır. Tasarım, ara özellik katmanlarını ve sınıflandırıcının çalışmasını görselleştirerek geliştirildi. AlexNet ile karşılaştırıldığında, daha küçük boyutlu filtre (7x7) kullanır. Nesne tanımada hata oranı %11,2 dir.
- GoogleNet: Google'daki araştırmacılar tarafından geliştirilen 22 katmanlı bir derin konvolüsyonel sinir ağıdır. Kümelerinde başarılı sonuçlar üreten ve veri hata oranı %5,7 modeldir.
- VGGNet: VGG16 ve VGG19 olarak iki farklı versiyonu vardır. VGG19, 5 ortaklama katmanı ve 3 tam bağlı katmanda sona eren 16 evrişim katmanından meydana gelen ve ReLU aktivasyon fonksiyonlarıyla tanımlı bir derin öğrenme modelidir. ImageNet Large Scale Visual Recognition Challenge'ı %7,3'lük hata oranı ile bitirmiştir.
- Microsoft ResNet: Microsoft Bing'deki Multimedya Grubu tarafından oluşturulan önceden eğitilmiş büyük bir görüntü modelidir. Model, görsel arama ve görsel arama'yı güçlendirmek için arama motorunun web ölçekli görsel verileri kullanılarak oluşturulmuştur. ResNet, sinyal gösterimini doğrudan öğrenmek yerine artık temsil fonksiyonlarını öğrenerek 152 katmana kadar çok derin bir ağa sahip olabilir. ResNet, girişte herhangi bir değişiklik yapmadan önceki katmandaki girişi bir sonraki katmana sığdırmak için atlamalı bağlantı sunmaktadır.

Tablo 4: Derin Öğrenme Modelleri Doğruluk ve Parametre Karşılaştırması

Model	**Yıl**	**İlk 5 Doğruluk**	**Parametreler**	**FLOP**
Alexnet	2012	84.70%	62M	1.5B
VGGNet	2014	92.30%	138M	19.6B
GoogleNet	2014	93.30%	6.4M	2B
ResNet-152	2015	95.51%	60.3M	11B

Kaynak : (Doğan & Türkoğlu, 2019)

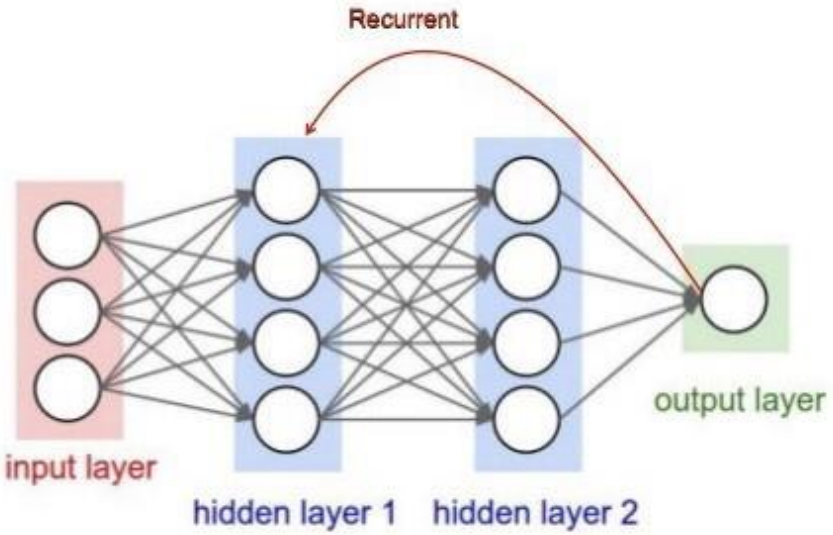
Tablo 4’de görüldüğü üzere derin öğrenme modellerinde, teknolojinin ve algoritmaların gelişimiyle doğruluk oranları da artmaktadır. CNN uygulamaları daha çok aşağıda belirtilen alanlarda kullanılmaktadır:

- Yüz Tanımlama
- Belgeleri Analiz Etme
- İklimi Tanımlama
- Gri Alanlar
- Reklamlar
- Tarihi Eserlerin Ayrıştırılması

CNN modelin tüm bu avantajlarının yanında eğitim ve test aşamasında yüksek kapasitede donanım ihtiyacı olmasıdır. Gerçek zamanlı çalıştırıldıklarında test süreleri göz ardı edilmemelidir. Eğitim süreci iyi gözlemlenmezse görünmeyen verileri genelleştiremeyecek kadar çok eğitilebilir.

2.2.2. Tekrarlayan Sinir Ağı (Recurrent Neural Networks) (RNN)

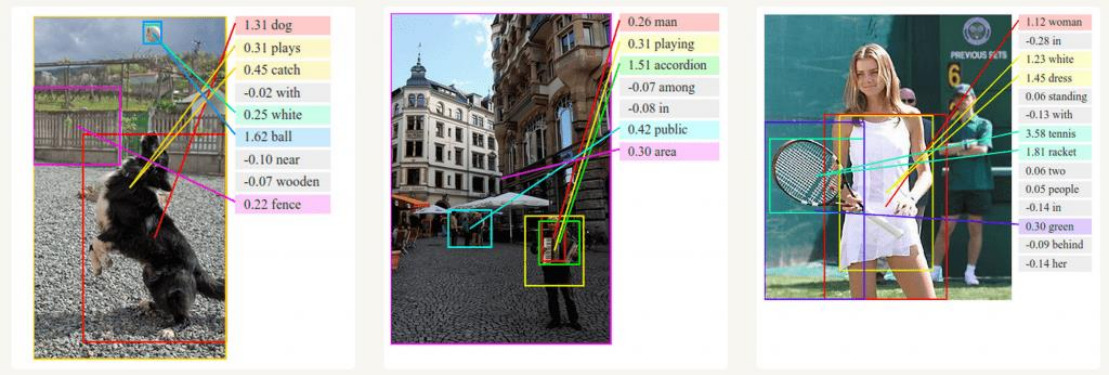
Basit Tekrarlayan Ağ (Simple Recurrent Network) 1980’lerde tekrarlayan sinir ağı fikri olarak ortaya atılmıştır. 1990’larda Jeff Elman tarafından RNN olarak önerilmiştir. Sıralı verileri kullanan bir tür yapay sinir ağıdır. Dil çevirisi, doğal dil işleme, konuşma tanıma ve resim yazısı gibi sıralı veriler için tercih edilir. Örneğin, “ağzını açıp gözünü yummak” deyiminin anlamlı olması için belirli bir sırada ifade edilmesi gerekir. RNN cümledeki her kelimenin konumu hesaba katması gerekir ve bu bilgiyi bir sonraki kelime tahmin etmek için kullanılır. Şekil 4’de RNN yapısı görülmektedir.



Şekil 4: RNN Mimarisi *(Gürbüz, 2020)*

RNN mimarisinde katmanların hepsi bağımsız şekilde çalışan ve her katmanın kendine göre olan bir işleve sahiptir. Ayrıntılı olarak açıklanacak olursa her katmandaki yapıların ağırlıkları vardır ve katmana özgü şekilde eşik değerleri belirlenir. Bu sayede sistemin daha olası sonuçlar vermesi amaçlanır.

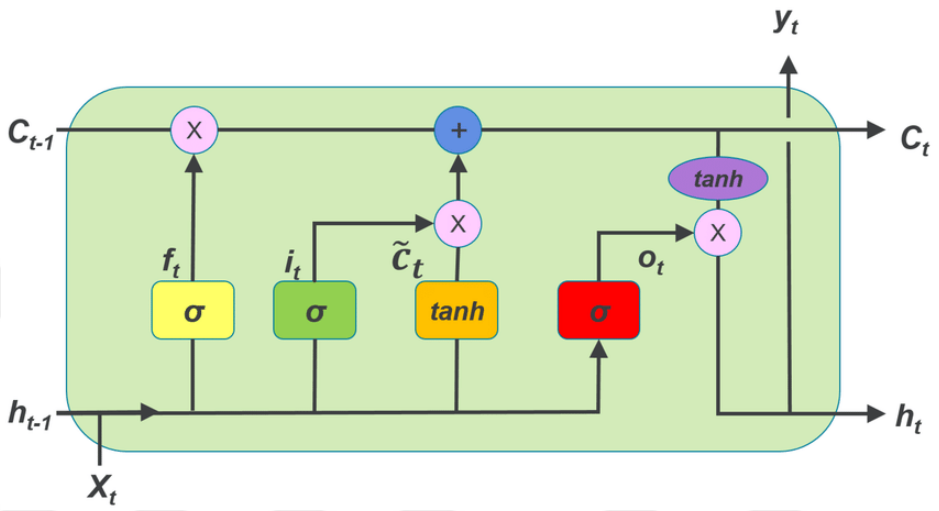
RNN uygulama alanlarının çoğunda kullanılabilir yapıdadırlar. Teknolojinin ilerlemesiyle birlikte başka birden fazla alanda çalışmalarda kullanılmışlardır. Kelime tahmini uygulaması bu alanların başında gelmektedir. Algoritmanın çıkış konusu kelime veya karakter tahmini üzerine olmuştur. Şekil 5'de RNN uygulaması örneği verilmiştir.



Şekil 5: RNN Uygulama Örneği (Gürbüz, 2020)

2.2.3. Uzun Kısa Dönemli Bellek (Long / Short Term Memory) (Lstm)

LSTM, değerleri rastgele aralıklarla hatırlayan ve gizli durumları hesaplamak için kullanılan bir RNN mimarisidir. LSTM, bilinmeyen süreli zaman gecikmeleri verilen zaman serilerini sınıflandırmak, işlemek ve tahmin etmek için çok uygundur. LSTM, içerisinde yer alan hafıza hücreleri sayesinde önceki durumlara ait bilgileri tutmaktadır. Bu sayede silinecek ve tutulacak verilere karar verir. Böylece uzun vadeli bağımlılıklar ortadan kaldırılmaktadır. Şekil 6'de LSTM yapısı verilmiştir.



Şekil 6: LSTM Yapısı (Gökçe & Sonugür, 2016)

LSTM yapısındaki kapılar sayesinde hücredeki bilginin akışı denetlenebilmektedir. Akışlar içeri ya da dışarı yöne doğru olabilir. Önceki durum bilgisi hafızasında olan kapı bilginin hafızaya girme zamanını belirler. Unutma kapısı silinecek olan bilgiyi ve yeni gelen bilgilerin hatırlanmasını denetler. Çıkış kapısı ise hücre içerisindeki bilginin çıkışını denetler. LSTM'ler veri kaybolmadan girdi ve çıktıların beraber çalışmasını sağlamaktadır. Birden fazla alanda bu konu üzerine çalışılmaktadır. Şu başlıklar altında konu incelenmiştir:

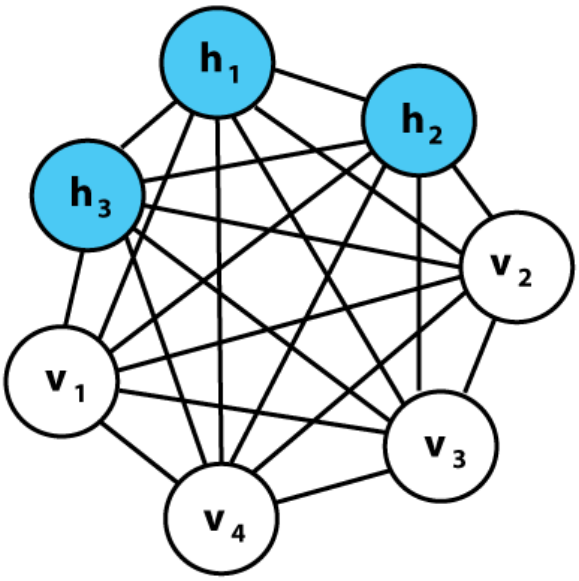
- Metin Üretimi: Harflerin öğretilmesi sonucunda otomatik olarak üretilen metin dizileridir.
- El Yazısı Tanıma: El yazısının harflerinin öğretilmesi ve sınıflandırılması sonucunda tanımlama yapılmasıdır.
- El yazısı Üretimi: Algoritma tarafından üretilen el yazılarıdır.

• Müzik Üretimi: daha önce çalınan notalar (veya nota kombinasyonları) dikkate alınarak LSTM tarafından da oluşturulabilen yapılardır.

• Dil Çevirisi: Kodlama-Kodlama Çözdürme işlemleri yapılarak uygulanan yöntem

2.2.4. Kısıtlanmış (Derin) Boltzmann Makineleri (Boltzmann Machines) (DBM)

Kısıtlanmış Boltzmann makinesi (DBM) düğümler arasında yönsüz bağlantılara sahip daha gizli katmanlara sahip bir modeldir. DBM'lerin temeli Hinton, Sejnowski ve Ackley tarafından yayınlanan "A Learning Algorithm for Boltzmann Machines" adlı makale ile öğrenme algoritmalarının prensiplerine dayandırılmaktadır. Şekil 7'de verilen yapıda Boltzmann makinesinin yapısı gösterilmiştir.

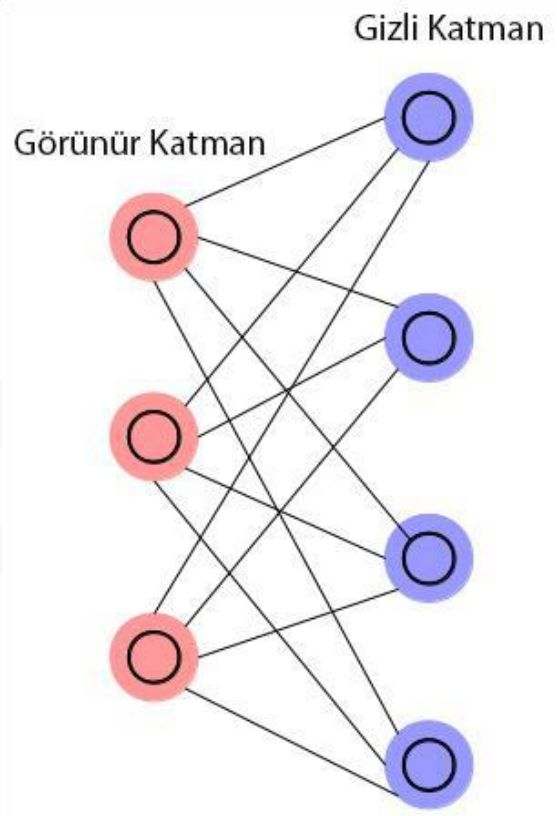


Şekil 7: Boltzman Makinesi *(Hinton G. E., 2012)*

DBM, özellikleri ham verilerden hiyerarşik olarak öğrenir ve bir katmanda çıkarılan özellikler, sonraki katmana girdi olarak gizli değişkenler olarak uygulanır. DBM, etiketlenmemiş büyük veriler için katman bazında ön eğitim için bir Markov rastgele alanı içerir ve ardından üst katmandan arka katmanlara geri bildirim sağlar. Geri yayılım yöntemi uygulanarak, eğitim algoritması ince ayarlanmıştır. DBM'deki eğitim sürecinin, eğitim bilgilerini, ağırlık başlatma ve ayarlama parametrelerini tanımlayacak şekilde uyarlanması gerekir. Bu DBM'den, parametreleri optimal olarak ayarlarken zaman

karmaşıklığı kısıtlamalarının ortaya çıkarmaktadır. Montavon ve diğerleri tarafından bir merkezleme optimizasyon yöntemi önerilmiştir.

Kısıtlı Boltzmann Makineleri iki parçalı yapıya sahiptir. Bu parçalar aralarında simetrik bağlantı bulunan görünür ve gizli olmak iki parçalı yapıdan oluşmaktadır. Şekil 8'de bu yapı gösterilmiştir.



Şekil 8: Kısıtlı Boltzmann Makinesi (Hinton G. E., 2012)

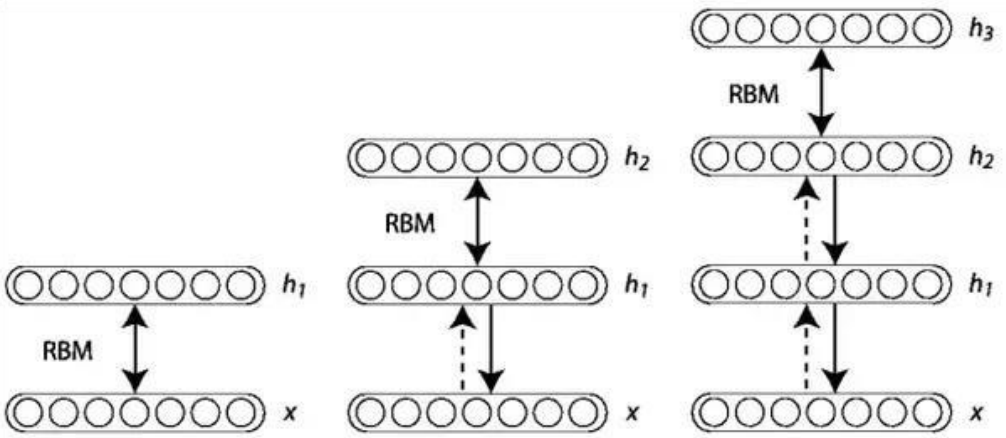
Şekil 8'deki grafikteki her daire, düğüm adı verilen nöron benzeri bir birimi temsil eder ve düğümler basitçe hesaplamaların gerçekleştiği yerlerdir. Düğümler birbirine katmanlar halinde bağlanır, ancak aynı katmandaki iki düğüm birbirine bağlanmaz. Bundan dolayı katman içi iletişim yoktur. RBM'lerin kısıtlaması budur. Her düğüm, girdiyi işleyen ve bu girişi iletip iletmeyeceği konusunda rastgele kararlar alan bir hesaplama odağıdır.

2.2.5. Derin İnanç Ağları (Deep Belief Networks) (DBN)

Derin inanç ağları, görüntüleri, video dizilerini ve hareket yakalama verilerini tanımak, kümelemek ve oluşturmak için kullanılır. Kısıtlı Boltzman makinalarının

yığınlarıyla oluşturulur. Sürekli bir derin inanç ağı, ikili veriler yerine sürekli bir ondalık sayıları kabul eden bir derin inanç ağının bir uzantısıdır. Tipik olarak ağın kodlama yarısını temsil eden dört veya beş sığ katmana ve kod çözme yarısını oluşturan ikinci dört veya beş katman kümesine sahip iki simetrik derin inanç ağından oluşur. 2006 yılında Geoff Hinton ve öğrencileri tarafından tanıtılmıştır.

Şekil 9'da verilen yapı art arda eklenmiş Kısıtlanmış Boltzmann Makinelerinin (RBM) yaklaşımıdır.



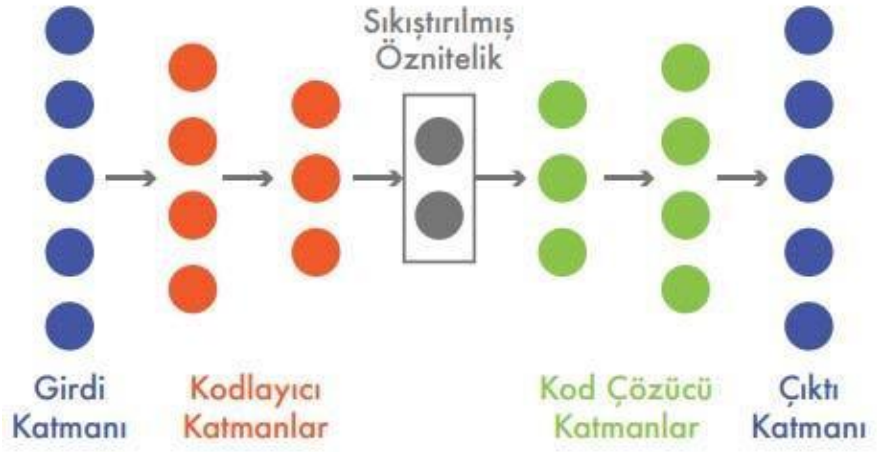
Şekil 9: Art Arda Eklenmiş RBM (Hinton G. E., 2012)

Derin inanç ağı modeli kullanılarak yapılan çalışmalar şu başlıklar altında toplanabilir:

- Görüntü Tanımlama
- Video Dizileri
- Hareket Yakalama
- Konuşma Tanıma

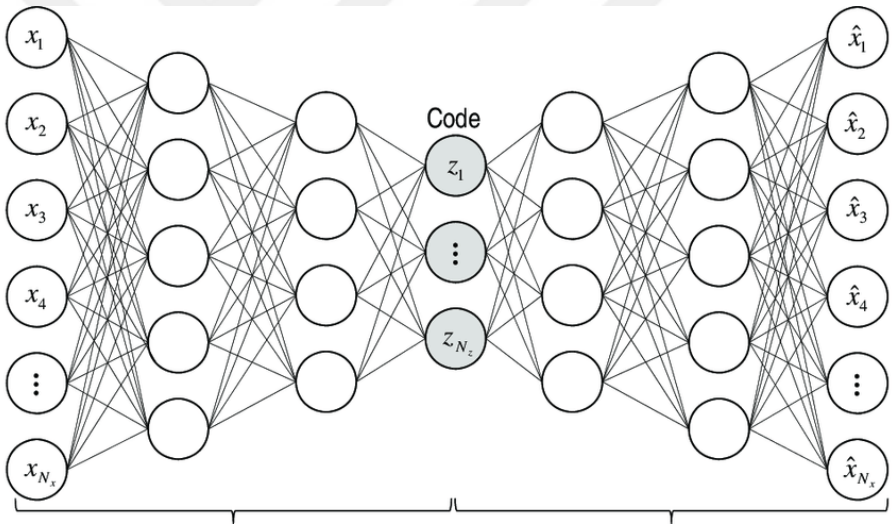
2.2.6. Oto Kodlayıcılar (Autoencoders)(AE)

Oto kodlayıcılar (AE), giriş ile çıkışın aynı ileri beslemeli sinir ağlarının özel bir türüdür (Bengio, 2009). Girdiyi daha düşük boyutlu bir koda sıkıştırırlar ve ardından çıktı yeniden yapılandırılır. AE, gizli uzay gösterimi olarak da adlandırılan girdinin kompakt bir özeti ve ya sıkıştırmasıdır (Krizhevsky & Geoffrey, 2011). Bir otomatik oto kodlayıcı, kodlayıcı, kod ve kod çözücü olmak üzere 3 bileşenden oluşur. Şekil 10'da verildiği üzere, kodlayıcı girişi sıkıştırır ve kodu üretir. Daha sonra kod çözücü girişi yalnızca bu kodu kullanarak yeniden oluşturur.



Şekil 10: AE Yapısı (KAYAALP, 2018)

Derin oto kodlayıcılar (Deep Auto Encoders) DAE ise oto kodlayıcıların çok katmanlı yapısından oluşan bir sinir ağıdır. DAE yapısı şekil 11'de gösterilmiştir.



Şekil 11 : DAE Yapısı (KAYAALP, 2018)

Şekil 11'de girdi dizisi ilk gizli katmanda şifrelenmiş dizi ile elde edilmiştir. Sonraki gizli katmana elde edilen girdi 2. gizli katmandaki şifreleme sonucunda elde edilmiştir. Bu aşamaların bitmesi sonucunda kodlama işlemi gerçekleşmiştir. Bu yapının devamında şifrenin çözülme aşaması bulunmaktadır. Söylenilen işlemleri tersi olarak da düşünülen bu yapı gizli katmanlardan geçen şifrenin çözülerek çıktı olarak verilmesi sonucunda biter. Uygulama alanları şu şekildedir.

- Doğrusal Olmayan Boyutların Azaltılmasın
- Tavsiye İnternet Motorlarının Tasarımın

- Özellikleri Sınıflandırma
- Görüntü Tanımlama

2.3. Derin Öğrenme Kütüphaneleri

Derin öğrenme modellerinin eğitimi için birçok açık kaynak kodlu yazılımlar geliştirilmiştir. Bu yazılımlar geliştirilmeye açık ve çeşitli üniversiteler ve şirketlerce geliştirilmiştir. Derin öğrenme çalışmalarını kolaylaştıran farklı özelliklerde kütüphaneler bulunmaktadır. Kütüphaneler ve API'ler sayesinde derin öğrenme ile geliştirilen uygulamaların yapılması de basit hale gelmektedir. Güncel olarak mevcut yazılım kütüphanelerinin bazıları aşağıdaki gibi listelenmektedir (İnik & Ülker, 2017).

2.3.1. Tensorflow

TensorFlow, 2011 yılında Google araştırma ekibi olan tarafından geliştirilen veri akışı grafikleriyle verimli sayısal hesaplamalarına olanak sağlayan Python ile geliştirilmiş açık kaynak kodlu bir derin öğrenme kütüphanesidir. Model oluşturmak için veri akış grafiklerini kullanarak makine öğrenimini daha etkin hale getirmektedir.

Tensorflow, geliştiricilerin birçok katmana sahip büyük ölçekli sinir ağları oluşturmasına olanak tanır. Tensorflow'un esas amacı sınıflandırma, algılama, anlama, keşfetme ve tahmin işlemlerini yapabilmesidir. Yüksek ya da düşük kapasiteli işleme sahip cihazlarda bile herhangi bir değişiklik yapılmaksızın kullanılabilmesi mümkündür (Abadi, 2016).

2.3.2. Keras

Keras, Tensorflow ve Theano'un üstünde çalışan python ile yazılmış açık kaynak kodlu sinir ağı kütüphanesidir. Google mühendisi Francois Chollet tarafından geliştirilmiştir.Modüler, hızlı ve kullanımı kolay olacak şekilde tasarlanmıştır. GPU ve CPU üzerinde sorunsuz olacak şekilde çalışmaktadır. Düşük seviyeli hesaplamayı işlememektedir. Evrişimli sinir ağlarını (CNN) ve yinelemeli sinir ağlarını (RNN) desteklemektedir.

2.3.3. Theano

LISA laboratuvarlarında yazılmış, çok boyutlu diziler de dâhil olmak üzere matematik ifadeleri etkili bir şekilde tanımlamayı ve değerlendirmeyi sağlayan bir Python kütüphanesidir. Daha çok derin öğrenme projeleri yapımında kullanılır. GPU'larda veya

CPU'larda verimli çalışabilen bir sistemdir (Theano Development Team, 2016). Theano ile uygulanan yaygın makine öğrenme algoritmaları, CPU için derlendiğinde rakip alternatiflerden (C / C ++, NumPy / SciPy ve MATLAB) 1.6x ile 7.5x kat daha hızlı, GPU ile derlendiğinde ise 6.5x ile 44x daha hızlıdır (Schmidhuber, 2012) . Birden fazla GPU desteklememesi dezavantajlarındandır (Bergstra, James, et a, 2010).

2.3.4. Torch

Torch, GPU için CUDA'nın yanı sıra C / C ++ kitaplıklarını kullanmaktadır. Torch'un popüler uygulamaları, görüntü işleme için CNN, ve daha karmaşık alanlarda ise derin takviyeli öğrenmedir (deep reinforcement learning-DRL).

2.3.5. Caffe

Caffe, hız ve modülerlik göz önünde bulundurularak yapılmış derin öğrenme framework'üdür. Berkeley öğrenme merkezi ve kullanıcı topluluğu tarafından geliştirilmiştir. Uygulamayı ve yeniliği teşvik eden etkileyici bir mimariye sahiptir. Araştırma deneyleri ve endüstri için hızlı çalışma sunar. Nvidia K40 GPU ile günde 60 milyondan fazla görüntüyü işleyebilir.

2.3.6. Deeplearnıng4j

Java için yazılmış açık kaynaklı dağıtılmış derin öğrenme kütüphanesidir. Hadoop ve Apache Spark ile entegre olabilecek bir yapıdadır. Dağıtılmış GPU ve CPU kullanımı için yapay zekayı ulaşılabilir kılmaktadır.

2.3.7. Mxnet

Derin öğrenme algoritmalarının geliştirilmesini kolaylaştırmak için oluşturulan MXNETi çoklu dil yapısına sahip güçlü bir kütüphane aracıdır. Derin sinir ağlarını tanımlamak, eğitmek ve dağıtmak için kullanılır. Yalın, esnek ve ultra ölçeklenebilir ve birden çok dili desteklemektedir. MXNet modelleri, çok küçük miktarlarda belleğe sığabilecek şekilde taşınabilir. Bu nedenle, modelini bulutta eğitebilir ve mobil veya bağlı cihaza dağıtabilir. MXNet ayrıca birden çok GPU'ya ve birden çok makineye ölçeklenebilir.

2.3.8. Matconvnet

MatConvNet derin öğrenme modellerinin eğitilmesi için Matlab tabanlı bir yazılım kütüphanesidir. Bu kütüphane derin ağların tasarımında kullanılan katmanların blok

halinde tasarlanmasına olanak sunar. Blok halindeki tasarlanan fonksiyonlar sayesinde, yeni bir derin öğrenme modelinin tasarlanması daha esnek, anlaşılır ve kolay olmaktadır. Hem GPU hem de CPU üzerinde verimli hesaplama yapılmasına olanak verir. Bu sayede oluşturulan bir ağın çok büyük veri setleri üzerinde çalışmasına olanak sunar. MatConvNet şu an her ne kadar MATLAB'a dayanıyor olsa da bu kütüphane MATLAB kodu ile C ++ ve CUDA çekirdeği arasında belli bir ayrımla geliştirilmektedir. Bu nedenle gelecekte kütüphane, MATLAB'dan bağımsız olarak derin ağların işlenmesine izin verecek şekilde genişletilebilir.

2.4. Nesne Tanıma

Nesne tanıma, görüntülerdeki ve videolardaki nesneleri tespit etmemizi, nesneleri tanımlamamızı, sınıflandırmamızı ve takibini yapabilmemizi sağlayan bilgisayarlı görü ve görüntü işleme tekniğidir. Bu tür bir tanımlama ve konum belirleme ile nesne algılama, bir sahnedeki nesneleri saymak ve bunları doğru bir şekilde etiketlerken tam konumlarını belirlemek ve izlemek için kullanılabilir. Nesne algılama genellikle görüntü tanıma ile karıştırılmaktadır. Görüntü tanıma, bir resme bir etiket atar. Bir köpeğin resmine “köpek” etiketi verir. Nesne algılama ise her köpeğin etrafına bir kutu çizer ve kutuyu “köpek” olarak etiketler. Model, her nesnenin nerede olduğunu ve hangi etiketin uygulanması gerektiğini tahmin eder. Bu şekilde, nesne algılama, bir görüntü hakkında tanımadan daha fazla bilgi sağlar.



Şekil 12: Görüntü Tanıma ile Nesne Tanıma farklılığın gösterimi

Nesne tespit ve tanıma süreçleri 4 başlıkta sıralanmaktadır.

1. Veri girişi (Görüntü)
2. Ön İşleme
3. Tespit (Öz nitelik)
4. Tanımlama

Gelişen teknoloji ile GPU'lar CPU'ların yaptığı işleri yapmaya başlamıştır. Hatta daha hızlı ve doğruluk oranı yüksek sonuçlar alınmaktadır. Bu sebeple Derin öğrenme ağlarının eğitimi için GPU'lar tercih edilmektedir. Bu durum yeni ve birbirinden farklı derin öğrenme modellerinin tasarlanmasına zemin hazırlamıştır. Derin öğrenme yöntemleri sayesinde de daha fazla doğruluk oranına sahip nesne tanıma metotları geliştirilmektedir.

GPU'ların gelişimine büyük katkı sağlayan Nvidia şirketi CUDA ismini verdiği GPU üzerinde paralel çalışmaya olanak sağlayan araçlar geliştirmektedir. CUDA'nın işlemleri ayırarak paralel çalıştırması sayesinde CPU'nun gerçekleştirmekte zorlandığı büyük işlemlerde GPU'lar avantaj sağlamaktadır (Durukan, 2016). Aynı zamanda CUDA, C, C++, C#, Fortran, Java, Python gibi programlama dilleri ile yazılmış

algoritmaların GPU yani “grafik işlem birimi” üzerinde çalışmasını sağlayan bir sistemdir. (Dinçer, 2020).

GPU üzerindeki güçlü ve verimli hesaplama büyük miktarda eğitim verisinin işlenmesine olanak sağladığı için uzun yıllardır var olan makine öğrenme yönteminin yaygın olarak kullanılmasını sağlamıştır. GPU’ların gelişimi sayesinde daha kısa sürelerde çok daha büyük veri setleri ile derin sinir ağları eğitilmektedir. Ayrıca daha güçlü alt yapısıyla sınıflandırma ve tahmin için eğitilmiş derin öğrenme modellerinin çalıştırılması için de kullanılmaktadır.

Google, Facebook, Microsoft ve Amazon gibi büyük kuruluşlar GPU’ların kullanılmasıyla birlikte makine öğrenmesi alanlarında çalışmalara başlamışlardır. Büyük veri analizlerinde merkezi işlem birimlerinden (CPU) 100 kata yakın performans sunan grafik işlemciler (GPU) bu kuruluşların tercih sebebi olmuştur. Carnegie Mellon Üniversitesi’nden Prof. Ian Lane ‘nin çalışmasında ses kayıtları içeriklerinin yazıya dönüştürülmesi uygulamasının performans karşılaştırmasında GPU’ların CPU’ya göre 33 kata kadar daha hızlı olduğunu ortaya koymuştur. (Kurt, 2015).

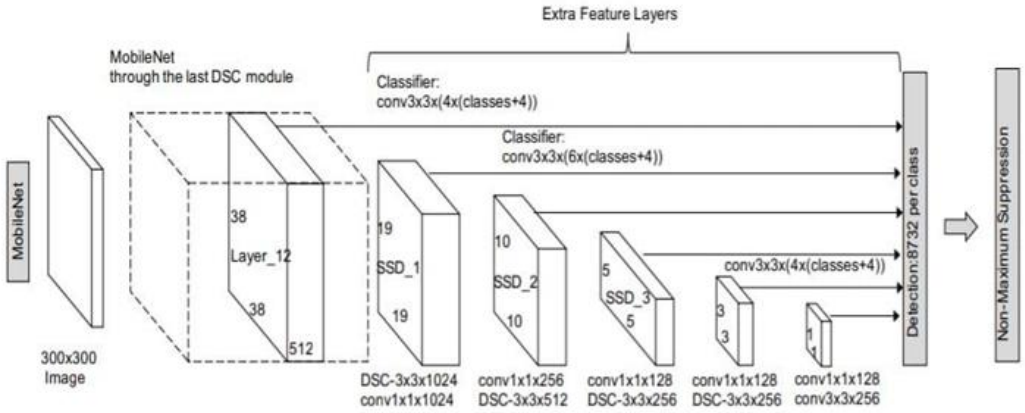
Nesne algılamayı makine öğrenimi tabanlı ve derin öğrenme tabanlı yaklaşımlar olarak ikiye ayırabiliriz. Daha geleneksel makine öğrenimi tabanlı yaklaşımlarda, bir nesneye ait olabilecek piksel gruplarını belirlemek için renk histogramı veya kenarlar gibi bir görüntünün çeşitli özelliklerine bakmak için bilgisayarla görme teknikleri kullanılır. Bu özellikler daha sonra nesnenin etiketiyle birlikte konumunu tahmin eden bir regresyon modeline beslenir. Diğer yandan derin öğrenmeye dayalı yaklaşımlar, özelliklerin ayrı ayrı tanımlanmasına ve çıkarılmasına gerek olmayan uçtan uca, denetimsiz nesne algılama gerçekleştirmek için evrişimli sinir ağlarını (CNN'ler) kullanır.

Derin öğrenme yöntemleri, nesne algılamaya yönelik en son teknoloji yaklaşımlar haline geldiğinden, son yıllarda R-CNN, Fast R-CNN, Faster R-CNN, Mask R-CNN ve SSD kütüphaneleri yaygın olarak kullanılmaya başlanmıştır. R-CNN model, bölge önerisi yaklaşımını tercih eden en temel derin öğrenme kütüphane modelidir. R-CNN serisinin arkasındaki anahtar kavram, bölge önerileridir. Bölge önerileri, bir görüntü içindeki nesneleri yerelleştirmek için kullanılır. 4 bölümden oluşan R-CNN’in birinci bölümünde görüntü içeri alınır. Ardından bölge önerisi yapılır. Üçüncü bölümde her bir bölge önerisi konvolüsyonel sinir ağı mimarisine iletilir. Son olarak CNN tarafından üretilen çıktı nesne değerini aldıysa seçici arama ile belirlenerek nihai sonuç üretilir. R-CNN

kütüphanelerinin geliştirilmesiyle Fast R-CNN, Faster R-CNN ve Mask R-CNN modelleri ortaya çıkmıştır

2.4.1. SSD MOBİLE NET

SSD'nin mimarisi, VGG-16 mimarisi üzerine kuruludur. SSD (Single Shot Dedector-Tek Atışlı Algılama) tek bir çekimdeki görüntünün içerisinden birden fazla nesneyi algılamak için kullanılan bölgesel teklif tabanlı bir yaklaşım sunmaktadır. R-CNN serisi bölge önerileri oluşturmak için iki çekime ihtiyaç duyarken SSD tek çekimde bunu gerçekleştirebilir. Bu yüzden R-CNN modellerine göre daha hızlı çalışır. Çalışma kapasitesi düşük işlemcilerde hızlı çalışacak şekilde tasarlanmıştır. (Howard, 2017). SSD model, resimde nesne olması beklenen bölgeler ile tam bağlı katman sınıflandırma işlemlerinin ikisini tek seferde gerçekleştirmektedir. Tüm evrişimsel modellerde olduğu gibi SSD Mobile Nette de girdi olarak bir görüntü alınır. Görüntü evrişimsel sinir ağından geçirilerek farklı boyutlarda özellik haritaları çıkarılmaktadır. 3x3 Evrişim filtreleri görüntüden belirli bir çıkarım elde ettik sonra bölge haritası yardımıyla farklı boyutlu sınırlayıcı kutular elde edilir. Sınırlayıcı kutular hem sınırları hem de sınıfları belirlemektedir ve hem küçük hem de büyük nesneleri tanımaktadır.



Şekil 13: SSD Mobile Net mimarisi (Arabi, 2019)

Avantajı

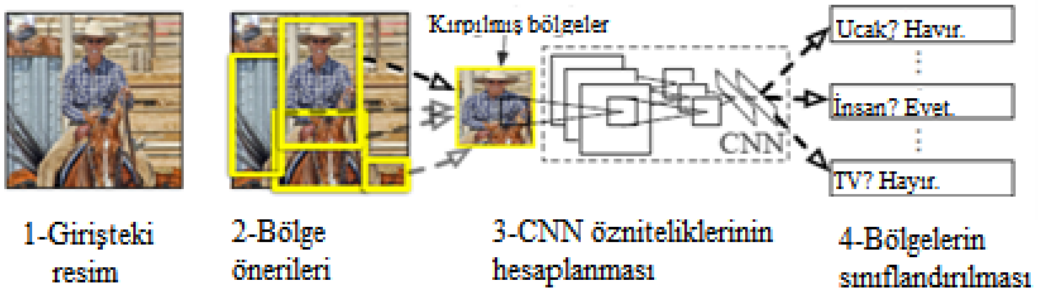
- Tahmin süresi çok düşük 1 sn altında
- Nesne tanıma hakkında genel bilgi sağlar
- Yönlendirici özellik olarak kullanılıyor
- Konum bilgisini yakalama bu modülde sağlanacaktır.

Dezavantajı

- Doğruluk düşük
- Özellik çıkarımının belirgin olduğu nesneleri rahat yakalıyor.

2.4.2. R-CNN

R-CNN bölge tabanlı Evrişimsel sinir ağıdır. R-CNN serisinin en önemli özelliği bölge önerileridir. Bölge önerileri, bir görüntüdeki nesneleri yerleştirmek için kullanılır. CNN ağının üzerinden bir görüntüyü geçirmeden önce seçici arama algoritması yardımıyla sınırlayıcı kutular bulunur. Daha sonra sınırlayıcı kutulardaki nesneleri bulmak için algılama algoritmasına sokularak tanıma gerçekleştiren bölge önerisi algoritmasıdır (Amidi, 2018). Şekil 14'de gösterildiği üzere R-CNN ağ mimarisi bölge tabanlı iki aşamalı bir nesne tanıma mimarisidir. (Girshick R. D., 2014).



Şekil 14 : R-CNN Mimarisi (Girshick R. D., 2014)

Avantajları

- R-CNN önce seçici-arama kullanarak olası nesne konumlarını tespit eder (Uijlings, 2012).

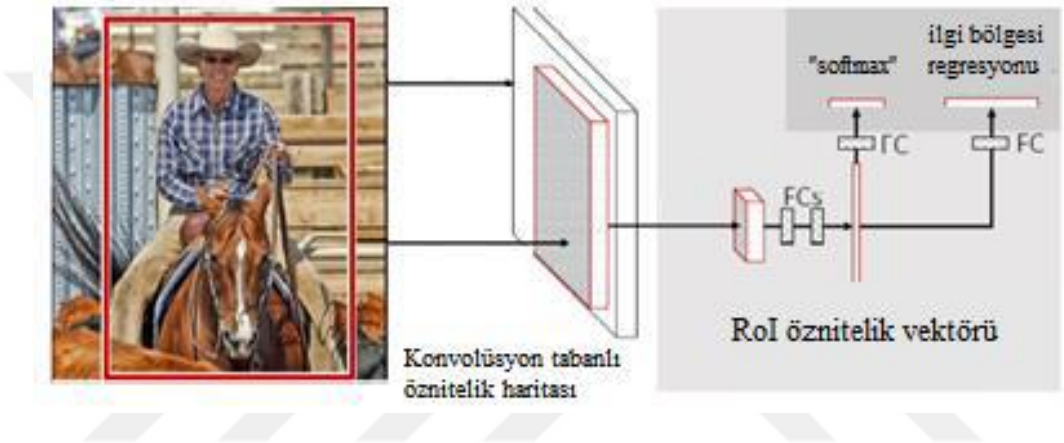
Dezavantajları

- Her görüntünün 2000 bölge önerisini sınıflandırması gerekir. Bu nedenle, ağı eğitmek çok zaman alır.
- GPU'daki bir görüntüdeki nesneleri algılamak için 47 saniye gerekir.
- Bölge önerisinin özellik haritasını saklamak için ayrıca çok fazla disk alanı gerekir.

2.4.3. FAST R-CNN

Gerçek zamanlı nesne tanıma işlemlerinde R-CNN yavaş kaldığı için kullanılması çok mümkün görülmemekteydi. Bu sebeple daha hızlı nesne tanıma algoritması oluşturmak için R-CNN modelinin dezavantajlı yönlerinin giderilmesiyle Fast R-CNN

mimarisi oluşturuldu. Fast R-CNN bölge önerisini CNN'den geçirmek yerine evrişimli özellik haritası oluşturmak için giriş görüntüsünü yani resimleri direk CNN'e beslemektedir. Tüm görsel CNN ile işlenerek özellik haritaları çıkarılır. Bir bölge önerisi özellik haritası oluşturulur. Roi havuzlama katmanı kullanılarak haritalar belirli boyutlara indirgenerek yeniden şekillendirilir. Roi özellik vektöründen önerilen bölgenin sınıfını ve sınırlayıcı kutuları (bounding box) tahmin etmek için softmax layer kullanılmaktadır. Fast R-CNN tek bir zayıf yönü vardır. Bu da zamanının çoğunu test aşamasında bölge önerisi yapmak için harcamasıdır. Fast R-CNN Şekil 14'de de görüldüğü gibi sınıflandırma için doğrusal SVM yerine "softmax" kullanarak doğruluğu arttırmıştır (Girshick R. , 2015).



Şekil 15 : Fast R-CNN Mimarisi (Girshick R. , 2015)

Avantajı

- Doğruluk oranı yüksek
- Fast R-CNN, eğitimi 8,75 saate kadar düşürmektedir ve R-CNN'den algılama süresini önemli ölçüde iyileştirir

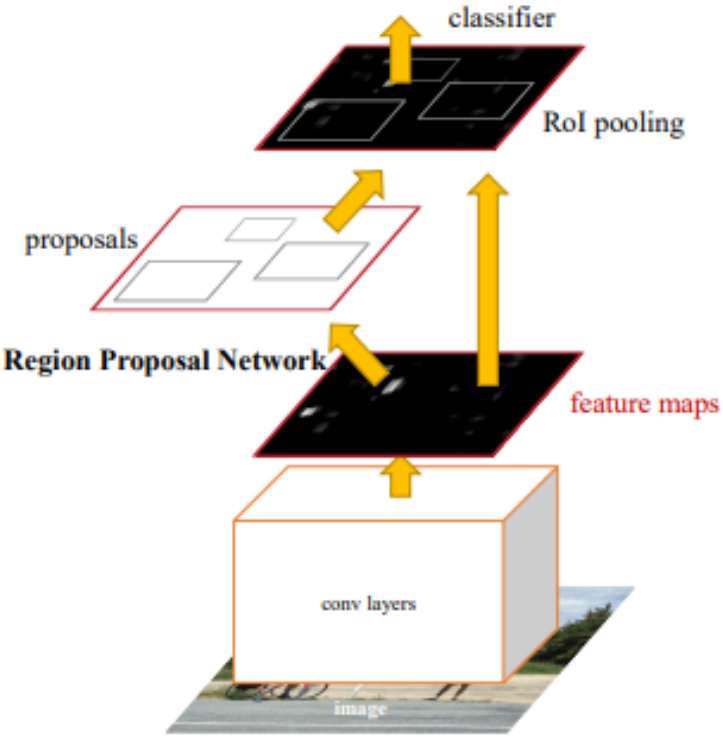
Dezavantajı

- Test aşamasında Bölge önerisini yapmak için çok fazla zaman harcamaktadır.
- Tahmin süresi 3 sn ile 5 sn arasında
- Gerçek piksel oranını veremiyor.

2.4.4. FASTER R-CNN

Faster R-CNN yüksek maliyetli seçici arama algoritması yerine daha kullanışlı olan ve ağın bölge önerilerini öğrenmesini sağlayan nesne tanıma algoritmasını kullanmaktadır. Bölge önerisini ağ içinde yaparak ciddi oranlarda hız kazanmaktadır. Faster R-CNN'de görüntü evrişimli özellik haritası sağlayan evrişimli bir ağa girdi olarak

sunulur ve bir özellik haritası çıkartılmaktadır. Daha sonra Fast R-CNN'e benzer şekilde bölge önerilerini tahmin etmek için bölge önerisi ağı oluşturulur. Tahmin edilen bölge önerileri alınıp sınıflandırmak ve sınırlayıcı kutular için ofset değerlerini tahmin etmek için kullanılan Roi havuzlama katmanıyla yeniden şekillendirilir. Faster R-CNN'in zor olan yanı hem bölge önerisini veren ağın eğitilmesi hem de Evrişimsel işlemlerin yapıldığı ağın eğitilmesidir. (Redmon J., 2016). Bu mimari ile çoklu obje tespiti hem yüksek doğruluk oranıyla hem de yüksek hızlı bir şekilde yapılmaktadır.



Şekil 16: Faster R-CNN Mimarisi (Ren, 2016)

2.4.5. MASK R-CNN

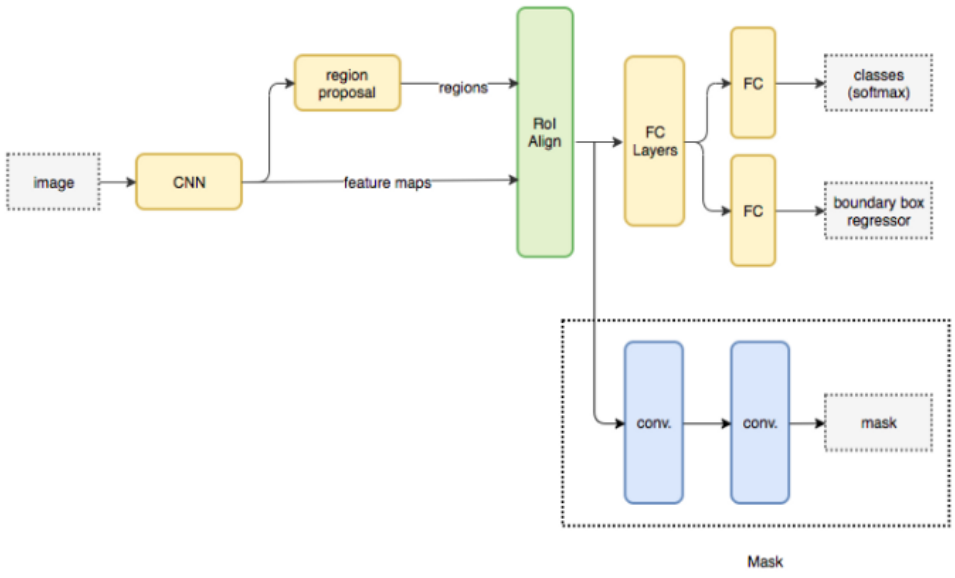
Mask R-CNN bir Evrişimsel sinir ağıdır ve görüntü segmentasyonu açısından son teknolojidir. Bir görüntüdeki nesneler algılar ve her örnek için yüksek kaliteli bir segmentasyon maskesi oluşturur. Görüntü segmentasyonu dijital bir görüntüyü birden çok bölüme yani piksel kümelerine bölme işlemidir. Bu segmentasyon nesneleri ve sınırları konumlandırmak için kullanılır. Mask R-CNN ağının kapsamına giren 2 ana segmentasyon vardır.

1. Semantik Segmentasyon
2. Örnek Segmentasyonu

Semantik segmentasyon nesne örneklerini ayırt etmeden her pikseli sabit bir kategoriler grubuna sınıflandırır. Görüntüdeki tüm nesneleri tek bir varlık olarak sınıflandırmaktadır. Bu yüzden görüntünün öznelerini arka plandan ayırdığı için arka plan segmentasyonu olarak da adlandırılır.

Örnek segmentasyon, bir görüntüdeki tüm nesnelerin doğru algılanmasıyla ilgilenirken aynı zamanda her bir örneği kesin olarak bölümlere ayırmaktadır. Bu nedenle nesne algılama, nesne lokalizasyonu ve nesne sınıflandırmanın birleşimi olarak ifade edilebilir. Görüntüdeki her bir nesneyi tek bir varlık olarak ayırmaktadır.

Mask R-CNN temel olarak Faster R-CNN kullanılarak oluşturulmuştur. Farklı olarak Faster R-CNN, her aday için 2 çıkışa, bir sınıf etiketine ve bir sınırlayıcı kutu uzaklığına sahipken, Mask R-CNN nesnenin maskesini oluşturan üçüncü bir dalın eklenmesidir. Eklenen kısım ikilik sayı sisteminde olacaktır ve sınıf ile kutu çıktılarından farklıdır (He, 2017). Maskeleme yapılacak ağda birler ve sıfırlar yardımıyla maskelemenin olup olmayacağında karar veren bir sistem kullanılmaktadır. Çıktı olarak 1 oluşuyorsa maskeleme yapılırken sıfırın çıktı olarak görüldüğü yerler maskeleme olmayacak yerler olarak belirlenir. Şekil 17'de Mask R-CNN mimarisi gösterilmiştir.



Şekil 17 : Mask R-CNN Mimarisi (Bharati, 2019)

Avantajları

- Yüzey alanın belirlenmesini sağlıyor.
- Boyut belirlemede kullanılır.

2.5. Mikroservis Mimarisi

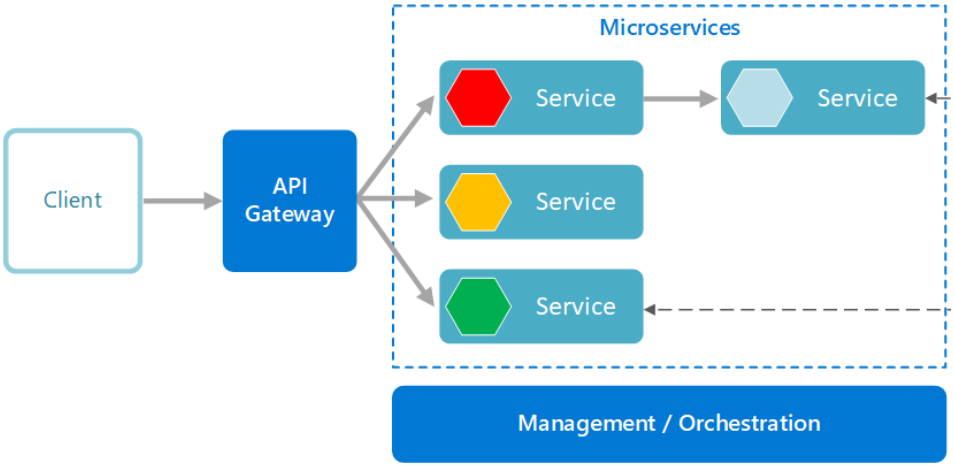
2.5.1. Mikroservis Tanımı

Mikroservis, modüler ve küçük bileşenlerden veya hizmetlerden büyük bir uygulamanın oluşturulduğu uygulama geliştirmeye yönelik mimari ve organizasyonel bir yaklaşımdır. Her modül belirli bir görevi veya iş hedefini destekler ve diğer hizmet gruplarıyla iletişim kurmak için uygulama programlama arabirimi (API) gibi basit, iyi tanımlanmış bir arabirim kullanır.

Mikroservis mimarileri, uygulamaları ölçeklendirmeyi ve geliştirmeyi daha hızlı hale getirerek yeniliğe olanak tanır ve yeni özellikler için sürüm çıkma kolaylığı sağlayarak süresini kazandırır.



Şekil 18: Monolitik Mimari Şeması (microsoft, 2019)



Şekil 19: Mikro servis Mimarisi Şeması (derek, 2019)

Yukarıdaki resimde de görüldüğü monolitik mimarilerle tüm süreçler sıkı bir şekilde birbirine bağlıdır ve tek bir hizmet olarak çalıştırılır. Bu, uygulamanın bir işlemin talebinde ani bir artış yaşanırsa, tüm mimarinin ölçeklendirilmesi gerektiği anlamına gelmektedir. Monolitik bir uygulamanın özelliklerini eklemek veya geliştirmek, kod tabanı büyüdükçe daha karmaşık hale gelir. Bu karmaşıklık, deneyleri sınırlar ve yeni fikirlerin uygulanmasını zorlaştırır. Monolitik mimariler, birbirine tam bağımlı bir şekilde bağlanmış süreçlere sahip olduğu için tek bir işlem ait hatanın uygulamanın tamamına etkisini olumsuz olarak artırır.

Bir mikroservis mimarisinde, bir uygulama, her uygulama sürecini bir hizmet olarak çalıştıran bağımsız bileşenler olarak oluşturulur. Bu hizmetler, küçük API'ler kullanarak iyi tanımlanmış bir arabirim aracılığıyla iletişim kurar. Hizmetler, iş yetenekleri için oluşturulmuştur ve her hizmet tek bir işlevi yerine getirir. Bağımsız olarak çalıştırıldıkları için, her hizmet bir uygulamanın belirli işlevlerine yönelik talebi karşılamak üzere güncellenebilir, dağıtılabilir ve ölçeklenebilir.

2.5.2. Mikro Servislerin Temel Özellikleri

Bağımsız Çalışma Bir sistemdeki servisler büyük ölçüde birbirinden bağımsızdır.

Bileşenleştirme: Mikroservisler, kolayca değiştirilebilen ve versiyonları artırılabilen bağımsız bileşenlerdir.

Yeteneklilik: Bir mikroservis basit bir yapıdadır ve tek bir göreve odaklanır.

Teknoloji Heterojenliği: Geliştiriciler ve ekipler birbirlerinden bağımsız olarak çalışabilir, böylece hızlıca geliştirme ve test süreçleri yürütülebilir.

Bağımsız ve Sürekli Dağıtım: Yazılım geliştirme, test etme ve onaylama sistematik otomasyonu ile sık sık yazılım sürümlerini canlıya almaya otomatik bir biçimde izin verir. Servisler ayrı ayrı güncellenebileceği için, yazılım dağıtımı süresi kısalır ve maliyet zamanla azalır.

Özgür Çalışabilme: Odak, doğru iş için doğru aracı kullanmaktır. Bu, standart bir teknoloji veya yapıyı zorunlu kılmadığı anlamına gelir. Geliştiriciler, sorunlarını çözmek için en iyi teknolojiyi seçme özgürlüğüne sahiptir.

Çeviklik: Mikroservisler çevikliği destekler. Herhangi bir yeni özellik hızla geliştirilip sisteme adapte edilebilir (Redmon J., 2016).

2.5.3. Mikroservis Mimarisinin Avantajları

2.5.3.1. Teknoloji Çeşitliliği

Bir mikro hizmet mimarisindeki her bileşen hizmeti, diğer hizmetlerin işleyişini etkilemeden geliştirilebilir, dağıtılabilir, çalıştırılabilir ve ölçeklenebilir. Hizmetlerin herhangi bir kodunu veya uygulamasını diğer hizmetlerle paylaşması gerekmez. Bireysel bileşenler arasındaki herhangi bir iletişim iyi tanımlanmış API'ler sayesinde gerçekleşir.

Her hizmet bir dizi yetenek için tasarlanmıştır ve belirli bir sorunu çözmeye odaklanır. Geliştiriciler zamanla bir hizmete daha fazla kod eklerse ve hizmet karmaşıklaşırsa, daha küçük hizmetlere bölünebilmektedir.

2.5.3.2. Esneklik

Mikro servisler, hizmetlerinin sahipliğini üstlenen küçük, bağımsız ekiplerden oluşan bir organizasyondur. Ekipler, küçük ve iyi anlaşılmış bir bağlamda hareket eder ve daha bağımsız ve daha hızlı çalışma yetkisine sahiptir. Bu, geliştirme döngüsü sürelerini kısaltır. Uygulamanın toplam çıktısından önemli ölçüde yararlanılmasını sağlar. Örneğin monolitik bir uygulamada, başarısız olan hizmetler tüm çalışmayı durdurur. Ancak mikro servis yaklaşımında bir uygulamanın durabilmesi için hizmetlerin tamamının toplam başarısızlığına bakılır. Hizmetlerdeki bağımsız hatalar uygulamanın sadece bir bölümünü etkileyecektir.

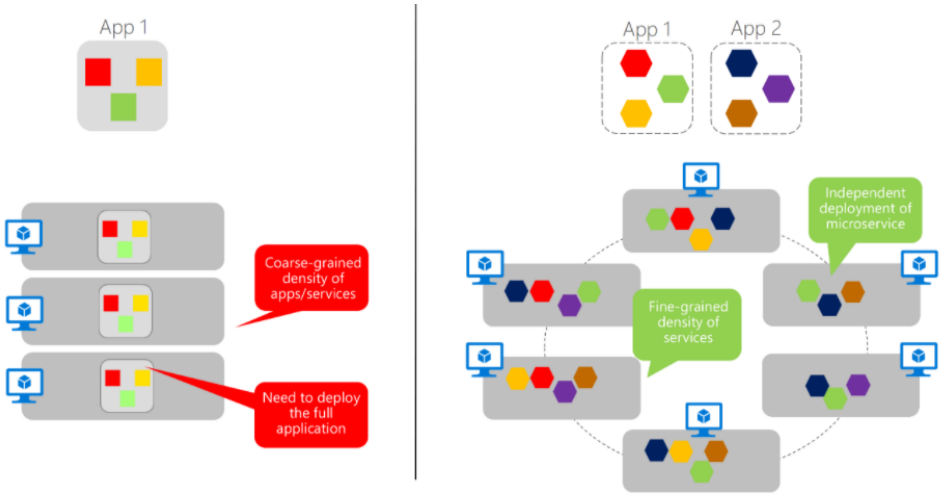
2.5.3.3. Ölçekleme

Mikro servisler, desteklediği uygulama özelliğine yönelik talebi karşılamak için her hizmetin bağımsız olarak ölçeklendirilmesine olanak tanır. Bu, ekiplerin altyapı ihtiyaçlarını doğru boyutlandırmasına, bir özelliğin maliyetini doğru bir şekilde ölçmesine ve bir hizmette talepte ani bir artış yaşanması durumunda kullanılabilirliği sürdürmesine olanak tanır.

Monolitik uygulamalarda ise, her şeyi birlikte düşünmek ve ölçeklemek zorundayız. Mesela aynı uygulamada, CPU üzerinde yürütülen bir hizmet ile GPU'da koşan bir hizmeti farklı şekillerde ölçeklendirip, kaynak kullanımını yönetmek oldukça kolay ve kullanılabilir olacaktır.

2.5.3.4. Deployment Kolaylığı

Monolitik yaklaşımda herhangi bir kod satırındaki değişikliğin güncelleme ve yayınlama işlemi için tüm uygulamanın durdurulmasına gerek duyulmaktadır. Mikro servisler, sürekli entegrasyon ve sürekli teslimat sağlayarak yeni fikirleri denemeyi ve bir şey işe yaramazsa geri dönmeyi kolaylaştırır. Başarısızlık maliyeti olduğundan birden çok denemeleri mümkün kılar, kodu güncellemeyi kolaylaştırır ve yeni özellikler için dağıtım süresini hızlandırır.



Şekil 20: Monolitik ve Mikroservis Mimarilerde Yazılım Dağıtım Şeması (Kızıltan, 2018)

2.5.3.5. Organizasyonların Düzenlenmesi

Mikro hizmet mimarileri, "herkese uyan tek bir boyut" yaklaşımını izlemez. Ekipler, belirli sorunlarını çözmek için en iyi aracı seçme özgürlüğüne sahiptir. Sonuç olarak, mikro servisler oluşturan ekipler her iş için en iyi aracı seçebilir ve ekip sayısının minimize edilmesine olanak sağlar. Ayrıca ekibe sonradan katılan bir kişinin projeye adaptasyonu kolaylıkla sağlanır. (Alshuqayran, Nour, & Roger, 2016)

2.5.4. Mikro servislerin eksiklikleri ve dikkat edilmesi gereken noktaları

- Her bir mikroservis için ayrı bir derleme, dağıtım ve sürüm iş akışı gerekir. Bu nedenle, kurulum iş akışının otomatik hale getirilmesini sağlamak önemlidir, aksi takdirde operasyon ekipleri için artan iş yükü demektir.
- Her bir servisi ayrı ayrı izlenebilir ve bakımı yapılabilir olmalıdır.
- Mikroservisler, yapılandırma sistemindeki iş yükünü arttırır. Harici servislerin yapılandırılması genellikle servisler arasında paylaşılır ve servis sayısı arttıkça süresi uzayabilir.
- Servisler arasındaki durum aktarımı trafiği performans açısından olumsuz bir yön alabilir. Bu sorunu çözmek amacıyla önbelleğe alma ve eşzamanlılık genellikle performansı artırılabilir.
- Mikroservisler üzerinde güvenlik, paralellik karmaşıklığı artıran etmenlerdir. Bu karmaşıklığı iyi yönetmek gerekir.
- Yeni özelliklerin mevcut hizmetlerin işlevselliği bozulmadığından emin olmak için uygulama düzeyinde testler önem vermek gerekir.
- Her hizmetin kendi sürümleri, yayın planı ve yayın döngüleri olduğu için dokümantasyon çalışması daha fazladır.
- Uygulamaların ve kod tabanlarının sayısı arttığında, onu düzgün bir şekilde korumak ve yönetmek için çaba harcanır.
- Bağımlılığı azaltmak için kod tekrarı yaşanabilir, ancak performans göz önüne alındığında bu kod tekrarları göz ardı edilebilir.

Mikroservis mimarisi, geliştirilen, pratikte uygulanan ve yazılım endüstrisinde DevOps uygulamaları ivme kazandığında popüler hale gelen ilk hizmet tabanlı mimari

tarzlardan biridir. Mikroservisler, dağıtılmış bir mimari sunmaktadır. Mikroservis mimarisinin faydalarını şu şekilde belirtebiliriz.

1) Büyük, karmaşık uygulamalarda kod dağıtımını sürekli hale getirir (Fowler, 2019) .
2) Test maliyetlerini düşürerek işlemleri daha küçük ve hızlı yapar ve test kalitesini artırır (Building Microservices, 2014).
3) Daha iyi konuşlanma – servisler bağımsız olarak dağıtılabilir (Thones, 2015).
4) Birden fazla takım etrafında geliştirme aşamalarını organize etmenizi sağlar (microservices, 2016).
5) Daha iyi ölçeklendirilebilir sistemler tasarlamamıza olanak sağlar (AWS, 2018).

Ayrıca tamamen işlevsel olması için hizmet keşfi, yük dengeleyici vb. gibi bazı destek hizmetlerine ihtiyaç duymaktadır. Uygulamamızda mikroservis mimarisinin sayılan faydalarından yararlanmak için REDIS teknolojisi kullanılmıştır.

2.5.5. REDIS ve CUDA Mimarileri

Remote Dictionary Server (uzak kütüphane sunucusu) ifadesinin kısaltması olan REDIS; veritabanı, önbellek ve ileti aracısı olarak kullanılan açık kaynak kodlu (BSD lisanslı), bellek içi veri yapısı deposudur . REDIS bir milisaniyenin altında yanıt süreleriyle gerçek zamanlı oyun, reklamcılık teknolojisi, finansal hizmetler, sağlık hizmetleri ve IoT (nesnelerin interneti) uygulamaları için saniyede milyonlarca isteğin işlenmesinde kullanılmaktadır. REDIS önbelleğe alma, oturum yönetimi, gerçek zamanlı analiz, jeo-uzamsal, sohbet/mesajlaşma, medya akışı ve yayımlama/abonelik uygulamaları için tercih edilmektedir. REDIS'in Caching (Önbellek) Mekanizması, Oturum Yönetimi, Pub/Sub (Yayım/Üye) ve Queues (Kuyruk) gibi bazı kullanım senaryoları bulunmaktadır. Pub/Sub işlevini destekleyen komutlara da sahip Redis, anlık yayın yapmasına olanak sağlar. Bu, mesajı tek bir istemcinin aynı kanal üzerinden çok sayıda istemciye iletilmesini kolaylaştırır. Aynı zamanda kuyruk mimarisi yapısı sayesinde tamamlanması zaman alacak işlerin kesintiye uğramaksızın gerçeklemesini sağlar (AWS, 2018). REDIS'in sağlamış olduğu avantajları aşağıdaki gibi ifade edilmektedir.

- Redis, saniyede 110.000 SET ve saniyede 81.000 GET gerçekleştirebileceği hıza sahiptir.
- Redis, liste, küme, sıralanmış küme ve karmalar gibi veri türlerinin çoğunu destekler.
- Rediste işlemlerin atomik olması, iki istemcinin aynı anda verilere erişmesi durumunda sunucunun güncellenmiş bir değer almasını sağlar.
- Çok yönlü kullanımlara uygundur. Redis, önbelleğe alma, mesajlaşma kuyrukları ve web uygulaması oturumları gibi kısa ömürlü veriler için kullanılabilir.
- Redis'in yapılandırılması kolay olduğundan kurulum kolaylığı sağlanmaktadır.

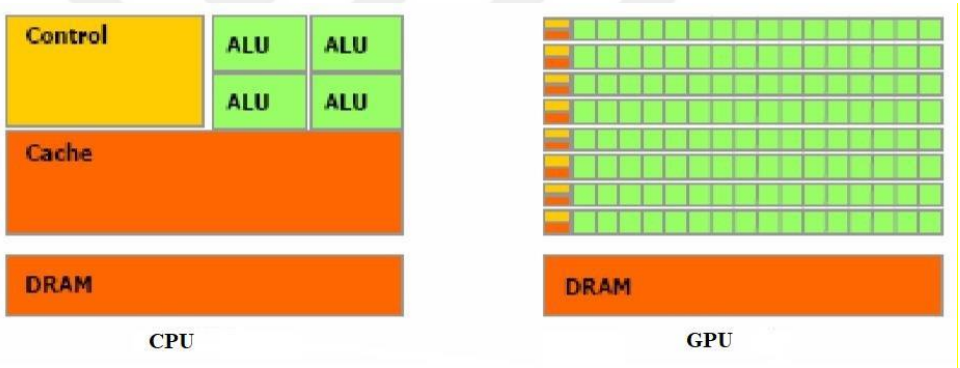
GPU mimarisi birden çok iş akışını tek bir emir ile aynı anda yapabilecek yapıdadır. GPU'lar birden çok akış işlemcisine (SM) sahiptir (Building Microservices, 2014). Her SM, ilişkili çekirdekleri ile ön bellekleme katmanını barındırır. Bu katman-1 olarak adlandırılır. Her bir akış işlemcisinin paylaşılan belleği ve kaydedicisi vardır. GPU daha az ve nispeten küçük önbellek katmanlarıyla çalışır. Fakat GPU'nun hesaplamaya ayrılmış daha fazla transistöre sahip olması belleğe erişimi hızlandırmaktadır. Kaynaklar aynı SM üzerinde bulunan iş parçacıkları arasında bölünmektedir (Thones, 2015).

CUDA, ilk olarak 2003 yılında Ian Buck liderliğindeki araştırma ekibi tarafından paralel yapılarla veri işlemek için C ile programlanan bir model olarak tasarlanmıştır. Buck'un Nvidia şirketine geçmesi ile ilk ticari çözüm 2006 yılında piyasaya sürülmüştür. Nvidia tarafından GPU'larda (grafik işlem birimleri) genel hesaplama için geliştirilmiş bir paralel hesaplama platformu ve programlama modelidir. CUDA, geliştiricilerin hesaplamanın paraleleştirilebilir kısmı için GPU'ların gücünden yararlanarak bilgi işlem açısından yoğun uygulamaları hızlandırmalarını sağlar. CUDA ve Nvidia GPU'ların kombinasyonu ile derin öğrenme de dâhil olmak üzere çeşitli uygulama alanlarına yaygın olarak kullanılmaktadır. GPU'lar, CUDA ile yazılmış "kernel" adı ile anılan seri bir programın binlerce kopyasını çoğaltarak paralelleştirmeyi sağlar. C dili ile geliştirildiği için genellikle CUDA kullanımını ve onları çoklu iş parçacığı haline dönüştürmek için yazılım mimarilerini değiştirmeye gerek duyulmamaktadır (Pahl, 2015).

CUDA ile geliştirilen program çalıştırıldığında GPU ve CPU farklı görevleri icra ederler. Hesaplama işlemlerini gerçekleştiren paralel kısmını GPU, kodun geri kalan seri kısmını CPU çalıştırır. Büyük verilerle çalışan yazılımcılar ve araştırmacılar, görüntü ve video işleme, hesaplamaya dayalı bilimsel çalışmalar gibi kullanım alanları bulunan uygulamalarda CUDA etkinleştirilmiş GPU tercih etmektedirler (microservices, 2016).

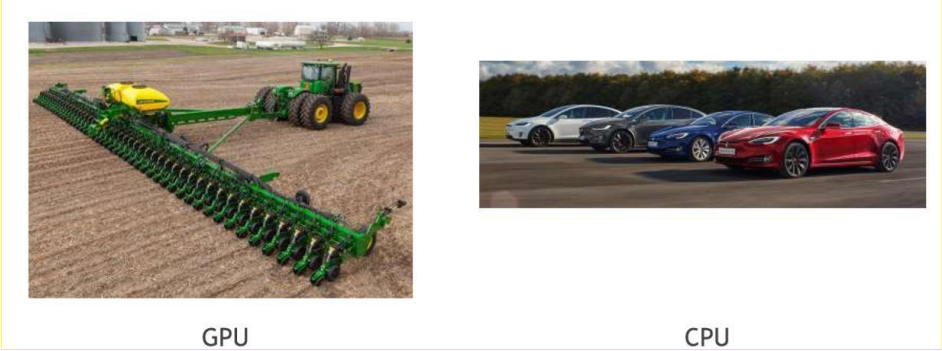
CPU'nun çalışma kapasitesi görüntü işleme gibi büyük boyutlu uygulamalarda yetersiz kaldığı bilinmektedir. GPU'lar üzerindeki kernellerin sayesinde grafiksel ve büyük veriler ile çalışmayı parallelleştirmeyi kullanarak kolayca yapmaktadır. GPU'ların görüntü işlemlerini yerine getiren donanımlara sahip olması yoğun hesaplamalarda büyük kolaylık sağlamaktadır (Parsons, 2018).

GPU'lar CPU ya göre daha çok sayıda aritmetik lojik birimine (ALU) sahiptir. Şekil 21'de GPU ve CPU mimarileri gösterilmektedir.



Şekil 21 : GPU ve CPU mimarileri (Parsons, 2018)

Genellikle GPU'lar donanım hızlandırıcısı olarak tercih edilmektedir. GPU ve CPU çekirdek karşılaştırması şekil 22'de temsili olarak gösterilmiştir.



Şekil 22: Temsili CPU ve GPU karşılaştırması (Cheng, 2014)

CireAan ve arkadaşları, GPU üzerinde, trafik işaretlerini %99,46'lık insandan daha iyi bir tanıma oranına derin ağların ön-eğitim olmaksızın eğitilebilmesi yaklaşımını kullanarak elde etmişlerdir. (CireAan, 2012)

BÖLÜM 3. YÖNTEM

Geleneksel olarak derin öğrenme ile gerçek zamanlı nesne tanıma, konumu belirleme ve değişenleri karşılaştırma işlemleri nispeten yavaş kaldığından işlemlerin paralelleştirilmesi bir çözüm olarak değerlendirilmiştir. Son zamanlarda pek çok farklı alanda aynı amaçla kullanılan mikroservis mimarisi bu bakımdan uygun bir seçenektir. Böylece; birden fazla R-CNN (SSD Mobile Net, Fast R-CNN, MASK R-CNN) ağının kullanılması ve bu R-CNN ağlarının senkron çalışması sağlanabilmektedir. Senkronizasyonun sağlanması ve ilave donanımlara gereksinim duyulmaması gerektiğinden aynı pc üzerinde performans kaybı yaşanmasını engellemek ve mevcut donanımlardan maksimum kazanç sağlanması için mikroservis mimarisi yaklaşımı sunulacak çalışma tasarlanmıştır.

Bu çalışmada, REDIS'in Pub/Sub ve kuyruk özelliklerinden faydalanılarak mikroservis mimari modeli geliştirilmiştir. Nesne tanıma için eğitilen derin öğrenme (SSD, Fast R-CNN, Mask R-CNN) modelleri arasında geçiş mimarisi tasarlama yoluna gidilmiştir.

Gezgin araç üzerindeki kameradan gelen görüntüler SSD Mobile Net, Fast R-CNN ve Mask R-CNN derin öğrenme modelleri ile geleneksel olarak ardışık yapılan nesne tanıma, alan taraması ve boyutlandırılması işlemlerinin, bağımsız mikroservis birimlerinde paralel işlenmesi sağlanacak bir uygulama geliştirilmiştir. Geliştirilen uygulamada tasarlanan model üzerinde testler yapılmış ve sonuçları ortaya konmuştur.

Gerçek ortamda çekilen fotoğraflar ile oluşturulan veri setinin bir bölümü eğitim verisi olarak ayrıldı. Diğer veriler ise test verisi olarak kullanıldı. Test verisi eğitim verisine eklenerek ve tüm veri sınıflandırılma işlemi için kullanıldı. Yarı gözetimli (semi supervised) öğrenme yöntemi ile Convolutional Neural Network CNN (Evrişimsel Sinir Ağı) yaklaşımı ile derin öğrenme modeli oluşturuldu.

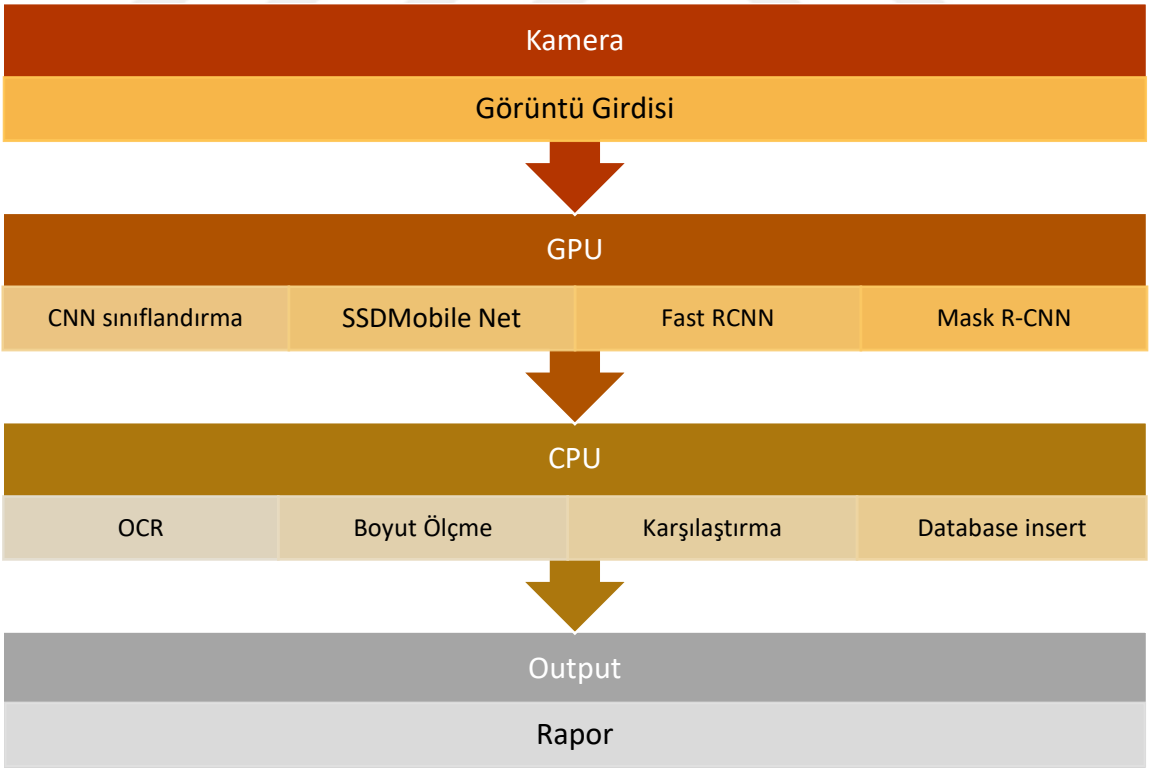
İlk olarak Fast R-CNN derin öğrenme modeli eğitildi. Buradaki amaç tabelayı bir kutu içine alarak nesne bazında tanıma yapıp uzaydaki koordinatlarını almaktı. Fast R-CNN ağında belirlenen tabelanın koordinatları hesaplandı. Tabela dışında başka nesne uzantılarının da görüntü içerisine gireceğinden tabela boyutunu tam olarak tespit edilmesi

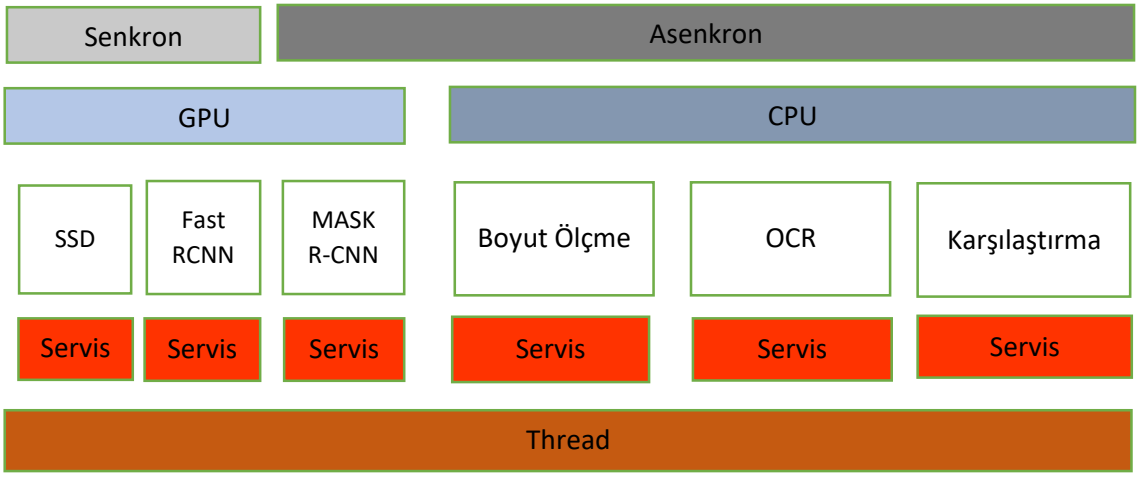
mümkün olmadı. Bu sebeple tabelanın boyutunu elde etmek için piksel bazında segmentasyona gidilerek yeni bir Mask RCNN temelli ağ kuruldu.

Buradaki amaç ise ilk başta tabela nesnesinin genelleştirilmiş koordinatlarını alıp görüntü içerisinden kesmek ve sonra Mask R-CNN modelimizle piksel bazında nesne algılama yaparak tabela nesnesinin hata payını minimize edip ana görüntüden ayırmak oldu. Bu işlemin amacı tabelanın boyutunu gerçeğe en uygun şekilde almak olduğundan Fast R-CNN ile eğittiğimiz verilerden dönecek olan görüntüyü genelleştirebileceğimiz için Mask R-CNN modelimiz kusursuza yakın öğrenmesi hedeflenmektedir.

Derin öğrenme modellerinin hazır hale getirilmesinin ardından her bir model mikroservisler halinde çalıştırılabilir duruma getirildi. REDİS ile her bir derin öğrenme modeli görevlerini ayrı ayrı icra edebilecek bir mikroservis mimari yapısı kuruldu.

Bu tez çalışmasında, Geleneksel mimaride yaşanan yavaşlık sorunu çözümü için mikroservis mimari yaklaşımı temel alınarak yeni bir çalışma modeli önerilmiştir. Yeni sistem önerisi şematik görünümü aşağıdaki gibidir.





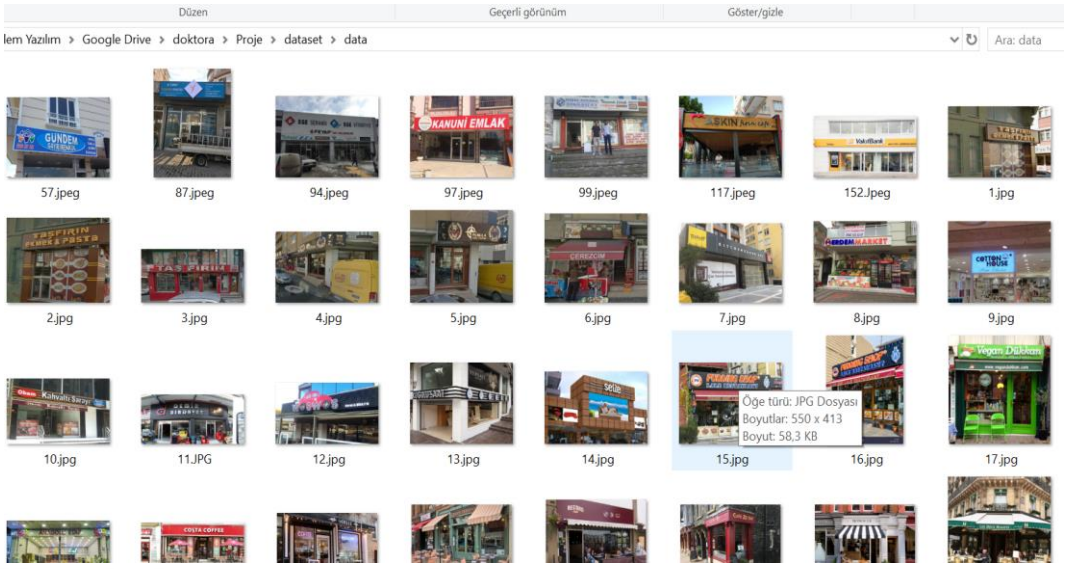
Şekil 23: Görüntü tanımada yeni nesil dağıtık mimari diyagramı

Şekil 23'de görüleceği üzere nesne tanıma, görüntü segmentasyonu senkron, boyut ölçme, OCR ve görüntü karşılaştırma işlemleri asenkron olacak bir şekilde çalışmaktadır. 80.000 adet talebi 1 milisaniyenin altında işleme kapasitesine sahip REDIS konfigürasyonu ile gerçek zamanlı nesne tanıma ve diğer tüm işlemlerin anlık olarak çalışmasının yolu açılmıştır. Tasarlanan model sayesinde tüm yapının senkron hızında çalışması sağlanmıştır.

Python programlama dili kullanılarak geliştirilen uygulamanın ve yapılan testlerin tüm aşamaları adım adım aşağıda verilmektedir.

3.1. Veri Seti Oluşturulması

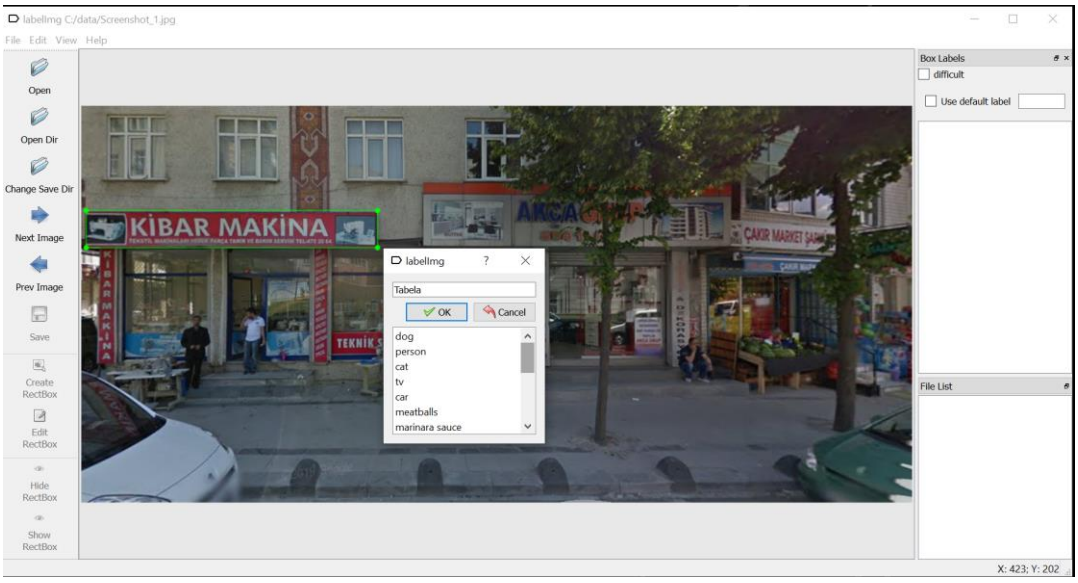
Tespit edilmesi istenen işyeri tabelaları nesnelerini örnekleyen 5000 adet görüntüden oluşan bir veri seti hazırlandı. Veri seti oluşturulmasına sahada çekilen fotoğraflar ile başlandı. Ardından Google Street View üzerinden ekran görüntüleri alınarak veri arşivi oluşturuldu. Veri arşivinin bir kısmı SSD Mobile Net ve Fast R-CNN ağı için labelling programı ile dikdörtgen alan içerisine alınarak tabela olarak etiketlendi. Bir kısmı Mask RCNN ağı için via etiketleme uygulaması ile tabela olarak etiketlenerek veri seti oluşturuldu. Böylece belirlenmiş probleme uygun gerçekçi bir veri seti oluşturulmuş oldu.



Şekil 24: Veri seti için toplanan resimler

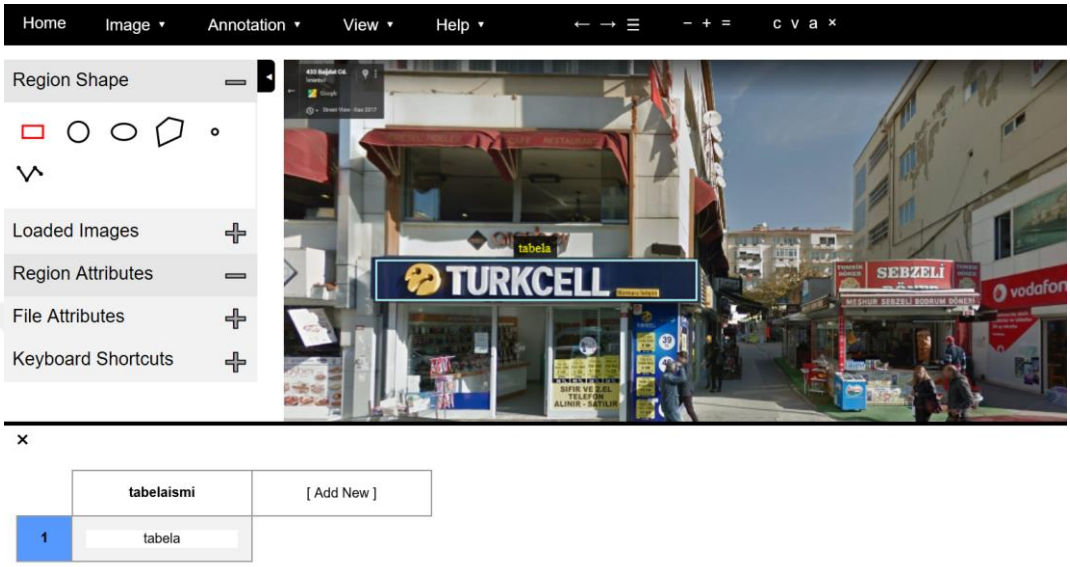
3.2. Veri Etiketleme

Seçilmiş olan görüntülerin SSD Mobile Net ve Fast R-CNN modeli eğitiminde kullanılabilmesi için gerekli etiketleme işini yapmak için açık kaynak kodlu bir uygulama olan labelImg isimli program kullanılmıştır. Uygulama ekran görüntüsü Şekil25 'de gösterilmiştir.



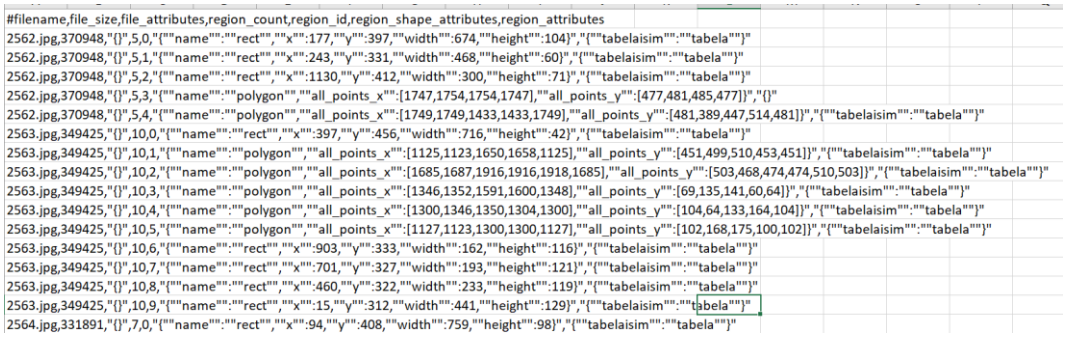
Şekil 25: LabelImg programı ile etiketleme

SSD Mobile Net ve Fast R-CNN modeli eğitiminde kullanılacak veri setleri ayrıca Mask R-CNN modeli eğitiminde kullanılabilmesi ve tabelaların köşe noktalarını tanımlamak için etiketleme işlemi Şekil26 ’da gösterilen Via Labelling Program isimli etiketleme programı ile yapılmıştır.



Şekil 26: Via Labelling Program ile Etiketleme

Sınıf ve koordinat bilgilerinin yer aldığı .csv dosyası oluşturulur.



```
#filename,file_size,file_attributes,region_count,region_id,region_shape_attributes,region_attributes
2562.jpg,370948,"{}",5,0,"{""name"":""rect"",""x"":177,""y"":397,""width"":674,""height"":104}","{""tabelaisim"":""tabela""}"
2562.jpg,370948,"{}",5,1,"{""name"":""rect"",""x"":243,""y"":331,""width"":468,""height"":60}","{""tabelaisim"":""tabela""}"
2562.jpg,370948,"{}",5,2,"{""name"":""rect"",""x"":1130,""y"":412,""width"":300,""height"":71}","{""tabelaisim"":""tabela""}"
2562.jpg,370948,"{}",5,3,"{""name"":""polygon"",""all_points_x"":[1747,1754,1754,1747],""all_points_y"":[477,481,485,477]}","{}"
2562.jpg,370948,"{}",5,4,"{""name"":""polygon"",""all_points_x"":[1749,1749,1433,1433,1749],""all_points_y"":[481,389,447,514,481]}","{""tabelaisim"":""tabela""}"
2563.jpg,349425,"{}",10,0,"{""name"":""rect"",""x"":397,""y"":456,""width"":716,""height"":42}","{""tabelaisim"":""tabela""}"
2563.jpg,349425,"{}",10,1,"{""name"":""polygon"",""all_points_x"":[1125,1123,1650,1658,1125],""all_points_y"":[451,499,510,453,451]}","{""tabelaisim"":""tabela""}"
2563.jpg,349425,"{}",10,2,"{""name"":""polygon"",""all_points_x"":[1685,1687,1916,1916,1918,1685],""all_points_y"":[503,468,474,474,510,503]}","{""tabelaisim"":""tabela""}"
2563.jpg,349425,"{}",10,3,"{""name"":""polygon"",""all_points_x"":[1346,1352,1591,1600,1348],""all_points_y"":[69,135,141,60,64]}","{""tabelaisim"":""tabela""}"
2563.jpg,349425,"{}",10,4,"{""name"":""polygon"",""all_points_x"":[1300,1346,1350,1304,1300],""all_points_y"":[104,64,133,164,104]}","{""tabelaisim"":""tabela""}"
2563.jpg,349425,"{}",10,5,"{""name"":""polygon"",""all_points_x"":[1127,1123,1300,1300,1127],""all_points_y"":[102,168,175,100,102]}","{""tabelaisim"":""tabela""}"
2563.jpg,349425,"{}",10,6,"{""name"":""rect"",""x"":903,""y"":333,""width"":162,""height"":116}","{""tabelaisim"":""tabela""}"
2563.jpg,349425,"{}",10,7,"{""name"":""rect"",""x"":701,""y"":327,""width"":193,""height"":121}","{""tabelaisim"":""tabela""}"
2563.jpg,349425,"{}",10,8,"{""name"":""rect"",""x"":460,""y"":322,""width"":233,""height"":119}","{""tabelaisim"":""tabela""}"
2563.jpg,349425,"{}",10,9,"{""name"":""rect"",""x"":15,""y"":312,""width"":441,""height"":129}","{""tabelaisim"":""tabela""}"
2564.jpg,331891,"{}",7,0,"{""name"":""rect"",""x"":94,""y"":408,""width"":759,""height"":98}","{""tabelaisim"":""tabela""}"
```

Şekil 27: Via ile Etiketlemede x ve y koordinatlarının CSV gösterimi

3.3. Eğitim Verisi Oluşturma

Her görüntü etiketlendikten sonra TFRecord kayıtları oluşturulmaktadır. Öncelikle etiket bilgisini barındıran xml dosyaları csv dosyasına çevirildi. Csv dosyaları oluşturulduktan sonra generate_tfrecord.py dosyası notepad++ metin editörü ile açılıp

sınıf bilgileri girildi. Id lerin sınıf sayısına eşit olmaları sağlandı. Veri setimizde 1 tane sınıf olduğu için 1 ID atandı. Geliştirilen uygulama için “Tabela” etiketi oluşturuldu.

```
def class_text_to_int(row_label):
if row_label == 'Tabela':
    return 1
else:
    None
```

TFRecord dosyası test ve eğitim için kullanılmaktadır. ID verdikten sonra aşağıdaki komutları çalıştırılır ve object_detection klasöründe test.record ve train.record dosyaları yaratılmaktadır.

```
python generate_tfrecord.py --csv_input=images\train_labels.csv --image_dir=images\train --output_path=train.record

python generate_tfrecord.py --csv_input=images\test_labels.csv --image_dir=images\test --output_path=test.record
```

TfRecord: Tensorflow'un veri depolamak için tercih ettiği dosya formatı, protobuf tabanlı TFRecord formatıdır. Diğer serileştirme formatları da işe yarayabilir ancak TFRecord dosyalarından veri setini doğrudan yazarak yükleme yapılabilmektedir.

3.4. Label Map Oluşturma

Oluşturulan modelde ID, sınıf tanımları label map dosyasında yapılmaktadır. Nesne tanıma eğitim klasörü içiresinde yeni bir dosya oluşturuldu. Oluşturulan dosyaya "labelmap.pbtxt" ismi verildi ve ID bilgileri girildi. Belirlenen her bir etiket için bir numara verilmektedir.

```
item {
  id: 1
  name: 'Tabela'
}
```

3.5. Eğitim Ayarlarının Yapılması

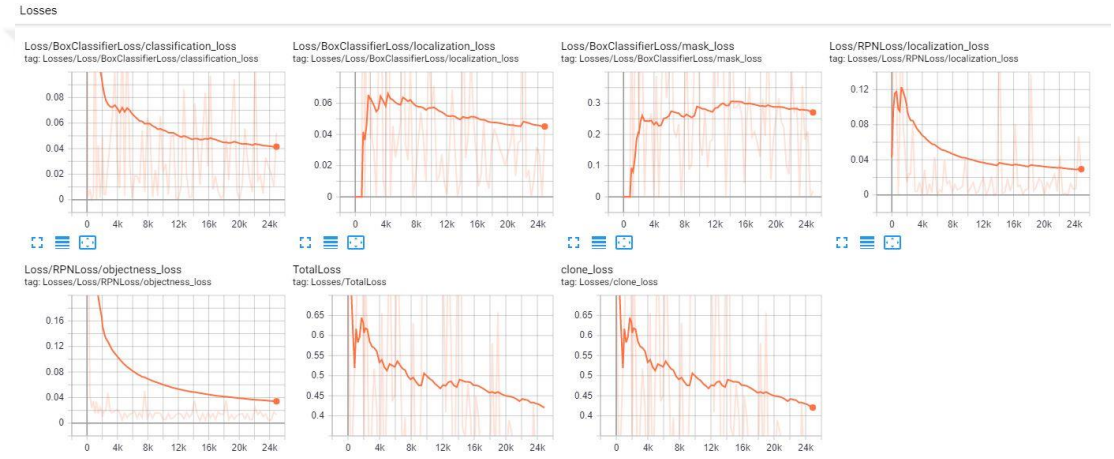
Tensorflow üzerinde kullanılacak model ve parametrelere karar verilir. İlk olarak çalışmak istenilen modelin yolu belirlenir. Ardından record ve labelmap dosyasının yolu

verilir. Daha sonra, test için ayrılan görüntü sayısı kadar yazılmalıdır. Son aşamada test için oluşturulan record ve label map dosya yolları belirtilmelidir.

3.6. Eğitimin Gerçekleştirilmesi

İlk olarak Tensorflow aktif edilir. Daha sonra python path ayarı yapılır. İşlemlerin ardından loss değerleri elde edilerek ekrana yazdırılır. Loss değeri tahmin edilen değerin gerçek değerden ne kadar uzak olduğunu belirtmektedir. Eğitim anında sıfıra yaklaşması beklenir (Pahl, 2015).

Mask R-CNN eğitimi 25.000 adımda gerçekleşmiştir.



Şekil 28: Mask R-CNN Eğitim Grafiği

3.7. Uygulama Sonuçlarının Ekran Görüntüsü

Bu bölümde sırasıyla SSD Mobile Net, Fast R-CNN, Mask R-CNN derin öğrenme modellerinin eğitimlerinin ardından yapılan test sonuçlarına yer verilmiştir.

SSD Mobile Net modelinin iş yeri tabelalarını tanıma oranının sonuçları aşağıdaki resimdeki gibidir. %99 oranına kadar doğrulukta tespit ettiği görülmüştür.



Şekil 29: Video üzerinden yakalanan görüntünün SSD Mobile Net test sonucu, doğruluk oranları

Fast R-CNN modelimizin eğitim gerçekleştikten sonra yapılan testlerde kameranın dikey olarak yakaladığı görüntülerde %99'a yakın başarı oranı yakalandığı şekil 31'de gösterilmiştir. Eğimli olarak gelen görüntülerde de Fast R-CNN modelimizin %99 oranında doğru tespit ettiği şekil 32'da gösterilmektedir.



Şekil 30: Fast R-CNN test veri girişi resim örneği



Şekil 31: Fast R-CNN test sonucu, doğruluk oranları

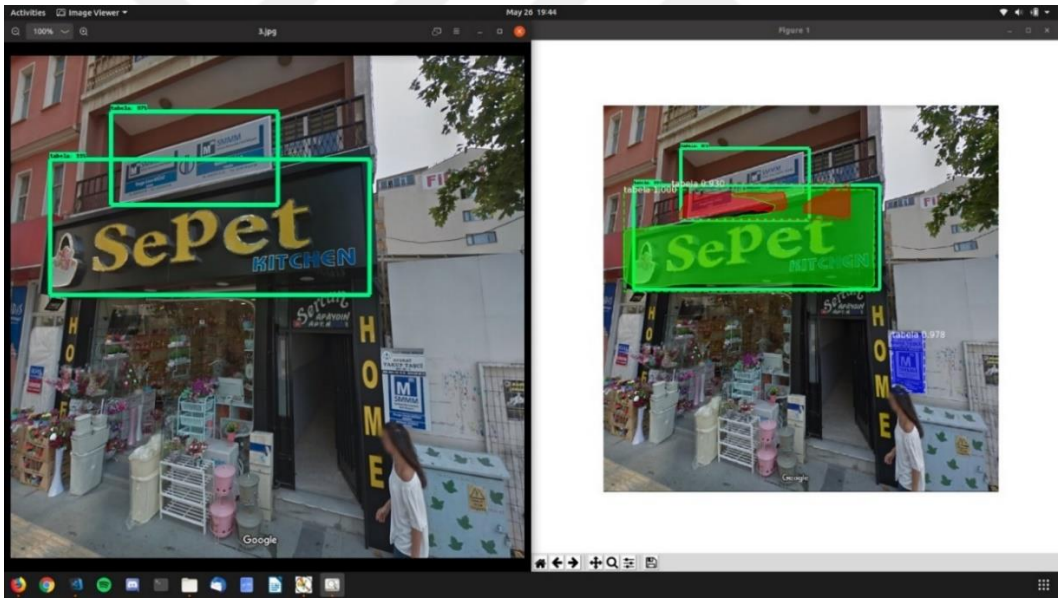


Şekil 32: Fast R-CNN test veri girişi resim örneği çapraz açı



Şekil 33: Fast R-CNN Test Veri Sonucu

Mask R-CNN modelimizin test sonuçlarımızda iş yeri tabela alanının taranabildiği ve alan hesaplaması yapılabilmesi için istenilen sonucu ürettiği görülmüştür.

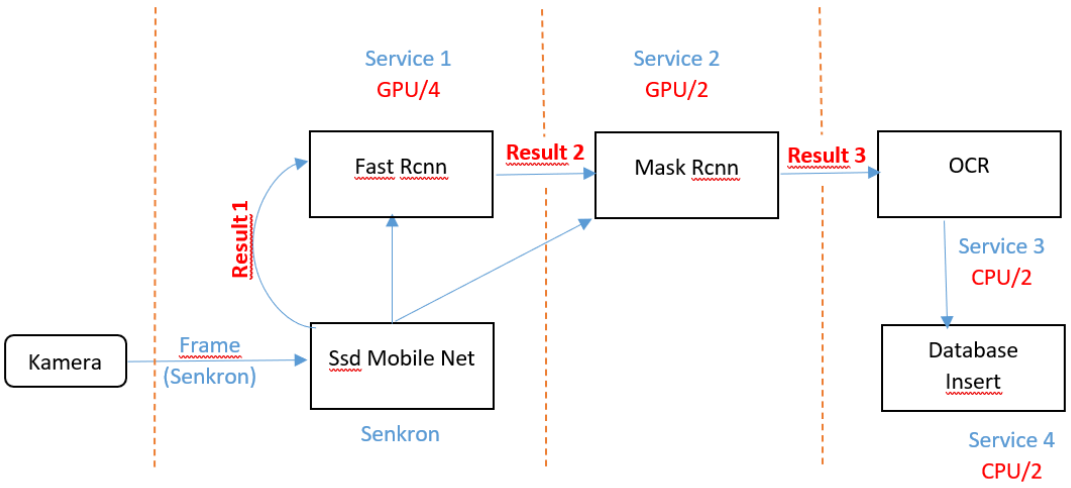


Şekil 34: Mask R-CNN Test veri sonucu

3.8. Mimari Tasarım Ve Uygulama

3.8.1. Mikroservis Mimari Tasarımı

Mikroservis mimarisi, kameradan gelen görüntülerin anlık olarak ilgili adımlardan geçerek bir sonraki nesne tanıma ve boyutlandırma adımlarına geçilmesini sağlayacak şekilde tasarlanmıştır. Şekil 35'de belirtildiği üzere önceden eğitilen SSD mobile Net ve Fast R-CNN modelimiz ekran kartının ayrılmış alanında çalışıp tabelayı çerçeveleyecek ve görüntüyü base64stringe çevirerek Mask R-CNN modele iletecektir. Mask R-CNN modeli ilk başta tabela nesnesinin genelleştirilmiş koordinatlarını alıp görüntü içerisinden keserek piksel bazında segmentasyon yaparak nesne tanıma tamamlanmaktadır. Mikroservis aracılığıyla diğer bir işlem adımı olan OCR ve Database insert gerçekleştirilerek tasarım süreci tamamlanmış olmaktadır.



Şekil 35: Mikroservis mimari tasarımı

3.8.2. Mikroservis Mimarisinin Uygulanması

Bu tez çalışmada mikroservis mimarisinin uygulanmasında kullanılan REDIS mimarisinin entegrasyonu aşağıda açıklanmaktadır.

3.8.2.1. Redis Konfigürasyonu

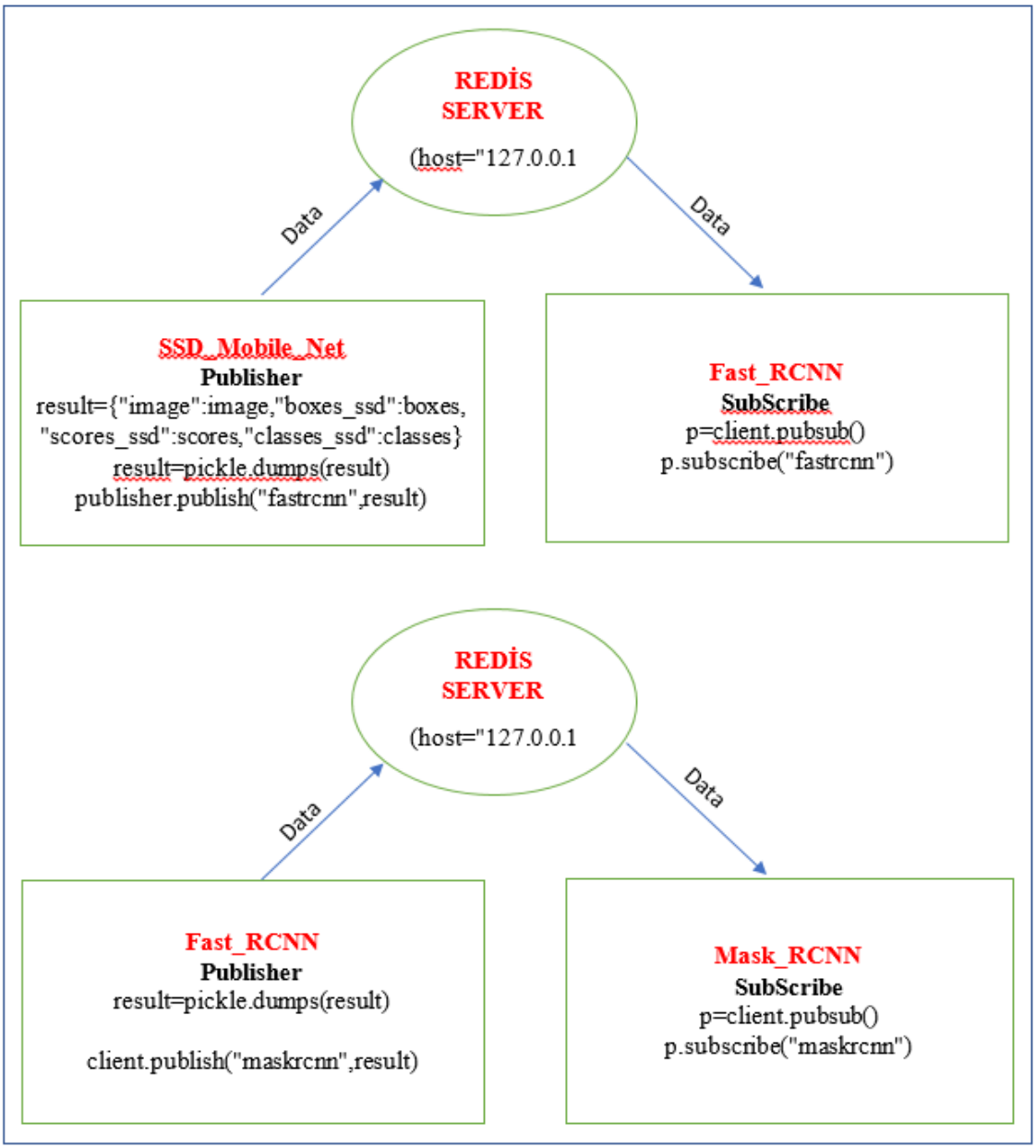
Gpu da çalışan bir derin öğrenme modelinin boyutu ne olursa olsun GPU'nun bütün belleğini işgal(allocate) edeceğinden GPU'yu fraksiyonlara bölüp derin öğrenme modellerinin ayrı ayrı servisler halinde çalıştırılması sağlandı.

Üç farklı model tercih edilmesinin sebebi tahmin sürelerine göre kıyaslarsak ssd_net<fast_rcnn<mask_rcnn dir. Bu bağlamda veri kaybetmemek adına önce ssd net te bir görüntüyü okuyup doğruluk sonucuna göre diğer modellere iletilmektedir. Mask R-CNN ise piksel bazında segmentasyon yaptığı için bounding box'ın dışında kalanı (ağaç, bina vb) almadığından tabelanın tam anlamıyla piksel bazında alanını dönmektedir. Bu sayede uzaklığı ölçümleyip;

tabelanın boyutu= (uzaklık*pikselgenişliği)/facollength

formülünden hesaplanabilmektedir. Son olarak tanıması yapılan tabelalar ve tabelanın boyutu kalmaktadır.

Redis mimarisinin kod kısmında uygulanması aşağıdaki şekillerde gösterilmektedir.



Şekil 36: REDIS Mimarisi kod yapısı

GPU çekirdeklerini kullanarak paralel hesaplama yapmak için CUDA yazılımı üzerinde Python kullanılmıştır. Bu tez uygulamasında CUDA 10.0 sürümü kullanılmıştır. CUDA 10.0 sürümü Windows 10 işletim sistemi üzerine kurulmuştur.

Bu tez çalışmasında kullanılan derin öğrenme algoritmalarının CUDA konfigürasyonları aşağıda gösterildiği gibi yapılmıştır.

3.8.2.2. CUDA Konfigürasyonu

SSD_Mobile_Net için GPU'da 0,10 oranında ayrım yapılmıştır. Kod gösterimi aşağıdaki gibidir.

```
def load_ssd_net_inference_graph(self):
    tf.device('/cpu:0')
    print("> ====== Loading frozen graph into memory")
    detection_graph = tf.Graph()
    config = tf.ConfigProto()
    config.gpu_options.per_process_gpu_memory_fraction = 0.10

    with detection_graph.as_default():
        od_graph_def = tf.GraphDef()
        with tf.gfile.GFile(self.PATH_TO_CKPT, 'rb') as fid:
            serialized_graph = fid.read()
            od_graph_def.ParseFromString(serialized_graph)
            tf.import_graph_def(od_graph_def, name='')
        sess = tf.Session(graph=detection_graph,config=config)

    print("> ====== Inference graph loaded.")

    return detection_graph, sess
```

Fast R-CNN modeli için GPU'da 0,10 oranında ayrım yapılmıştır. Kod gösterimi aşağıdaki gibidir.

```
def load_faster_cnn_inference_graph(self):

    print("> ====== Loading frozen graph into memory")
    detection_graph = tf.Graph()
    detection_graph = tf.Graph()
    config = tf.ConfigProto()
    config.gpu_options.per_process_gpu_memory_fraction = 0.10

    with detection_graph.as_default():
        od_graph_def = tf.GraphDef()
        with tf.gfile.GFile(self.PATH_TO_CKPT, 'rb') as fid:
            serialized_graph = fid.read()
            od_graph_def.ParseFromString(serialized_graph)
            tf.import_graph_def(od_graph_def, name='')
        sess = tf.Session(graph=detection_graph,config=config)

    print("> ====== Inference graph loaded.")

    return detection_graph, sess
```

Mask R-CNN Modeli için GPU'nun geri kalan 0,80 'i ile çalışması sağlanmıştır. Kod gösterimi aşağıdaki gibidir.

```
def load_mask_rcnn_inference_graph(self):
    print("> ====== Loading damage graph 50 into memory")

    detection_graph = tf.Graph()
    config = tf.ConfigProto()
    config.gpu_options.allow_growth = True
    with detection_graph.as_default():
       od_graph_def = tf.GraphDef()
       with tf.gfile.GFile(self.PATH_TO_CKPT, 'rb') as fid:
            serialized_graph = fid.read()
            od_graph_def.ParseFromString(serialized_graph)
            tf.import_graph_def(od_graph_def, name='')
       sess = tf.Session(graph=detection_graph,config=config)

    print(">  ====== Inference graph loaded.")

    return detection_graph, sess
```

3.9. Programın Çalışması ve Ekran Görüntüleri

Daha önce öğrenme süreçleri tamamlanan derin öğrenme modellerinin (SSD_Mobile_Net, Fas R-CNN ve Mask R-CNN) birbirleri ile REDIS üzerinden mikroservis mimarisi yaklaşımı ile haberleşebilmesi sağlanmıştır. Herbir derin öğrenme modeli için servis kodları geliştirildi. Herbirinin kod yapısı aşağıdaki gibidir.

Ssd_net_service.py dosyasının kodları aşağıdaki gibidir.

```
from ssd_net_models import ssd_net_models ,detector
import redis
import cv2
import numpy as np
import pickle
import imageio
import time

detection_graph,sess=ssd_net_models().load_ssd_net_inference_graph()

ssd_detect=detector(detection_graph=detection_graph, sess=sess)

publisher=redis.Redis(host="127.0.0.1",port=6379)

reader = imageio.get_reader('1.mov')

cap = cv2.VideoCapture("1.mov")

length = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))

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

```
for i ,image in enumerate(reader):

        boxes,scores,classes=ssd_detect.ssd_net(image)

 image=cv2.resize(image,(600,600))
 result={"image":image,"boxes_ssd":boxes,"scores_ssd":scores,"classes_ssd":classes}
  result=pickle.dumps(result)
  publisher.publish("fastrcnn",result)
  time.sleep(2)

      print(length-i)
```

Fast_rcnn_service.py dosyasının kodları aşağıdaki gibidir.

```
from fast_rcnn_models import fast_rcnn_models ,detector
import numpy as np
import cv2
import redis
import pickle
from time import sleep
from queue import Queue
import time

#models loaded
detection_graph,sess=fast_rcnn_models().load_faster_cnn_inference_graph()

#called function
fast_detect=detector(detection_graph=detection_graph, sess=sess)

#redis
client=redis.Redis(host="127.0.0.1",port=6379)
p=client.pubsub()
p.subscribe("fastrcnn")

#Queue

sira=Queue()

#post method

def post(q,client):
    if not q.empty():
        message=q.get()

        ssd_net_results=pickle.loads(message)
```

```
        image=ssd_net_results["image"]

        boxes, scores, classes=fast_detect.faster_cnn(image)

result={"image":image,"boxes_fast":boxes,"scores_fast":scores,"classes_fast":classes,"
boxes_ssd":ssd_net_results["boxes_ssd"],"scores_ssd":ssd_net_results["scores_ssd"],"cl
asses_ssd":ssd_net_results["classes_ssd"]}

        result=pickle.dumps(result)

        client.publish("maskrcnn",result)
        q.task_done()

while True:

   message=p.get_message()

   if message and not message["data"] ==1:
      try:

        sira.put(message["data"])
        post(sira,client)
        sira.join()
        print("işlem Tamam")
        time.sleep(3)

      except:
         print("Fast Rcnn Analiz Hatasi")
```

Mask_rcnn_service.py dosyasının kodları aşığıdaki gibidir.

```
from mask_rcnn_models import  mask_rcnn_models ,detector
import redis
import cv2
import base64
import numpy as np
import pickle
from time import sleep
from queue import Queue
import time
from tabela_visualize import visualize_box

detection_graph,sess=mask_rcnn_models().load_mask_rcnn_inference_graph()
```

```
mask=detector(detection_graph=detection_graph,sess=sess)

v=visualize_box()

#redis
client=redis.Redis(host="127.0.0.1",port=6379)
p=client.pubsub()
p.subscribe("maskrcnn")

while True:

   message=p.get_message()

          if message and not message["data"] ==1:
             try:
              fast_rcnn_results=pickle.loads(message["data"])
              image=fast_rcnn_results["image"]

              #dictionary ==sözlük içindekiler maske(array formatında ,bounding box,doğruluk oranı,sınıf id si)
              output_dict=mask.mask_rcnn(image)


image=v.draw_box_on_image(fast_rcnn_results["scores_fast"],fast_rcnn_results["boxes_fast"],fast_rcnn_results["classes_fast"],image)

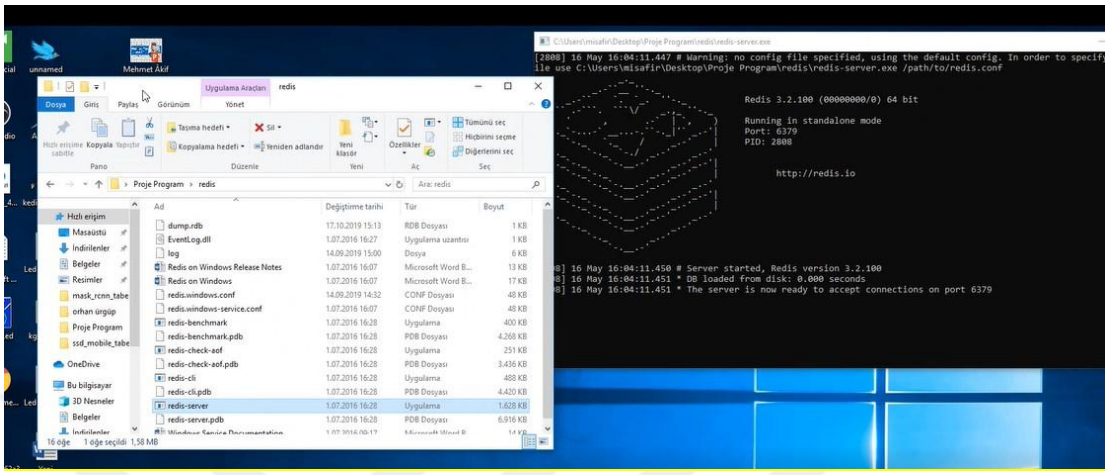
image=v.draw_box_on_image(fast_rcnn_results["scores_ssd"],fast_rcnn_results["boxes_ssd"],fast_rcnn_results["classes_ssd"],image)
              image=v.visualize_tf(image,output_dict)
              cv2.imwrite("deneme.jpg",image)
             except:
                print("mask_rcnn analiz hatası")
```

Öğrenme eğitimleri tamamlanan derin öğrenme modellerinin (SSD_Mobile_Net, Fas R-CNN ve Mask R-CNN) birbirleri ile REDİS üzerinden microservis mimarisi yaklaşımı ile haberleşebilmesi sağlanmıştır.

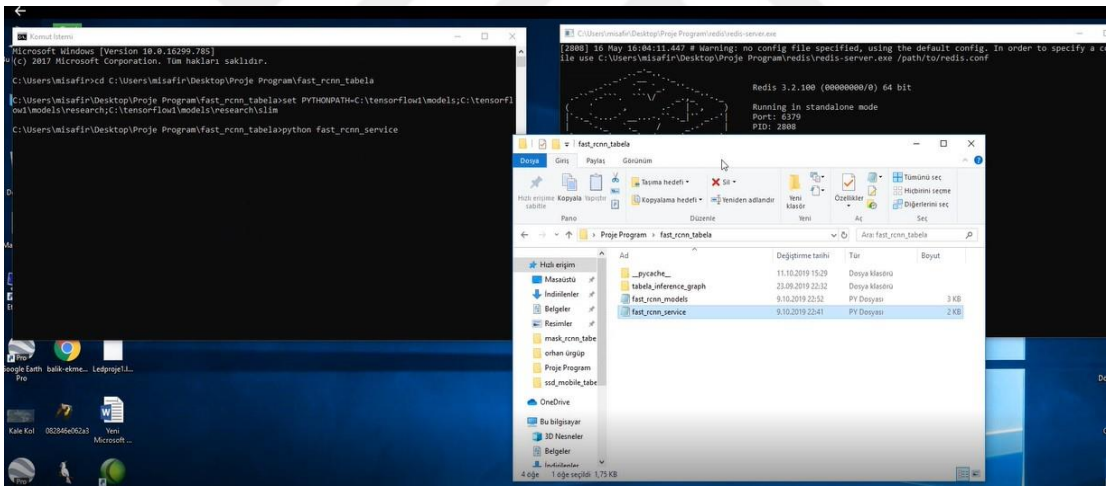
Microservis yaklaşımı ile geliştirilen modelin çalıştırılması ve sonuçlarının ekran görüntüleri aşağıda ifade edilmiştir.

İlk olarak REDIS'i aktif ediyoruz (Şekil 37). Ardından sırayla Fast R-CNN, Mask R-CNN ve SSD Mobile Net modelllerimizin servislerini ayağa kaldırıyoruz (Şekil 38).

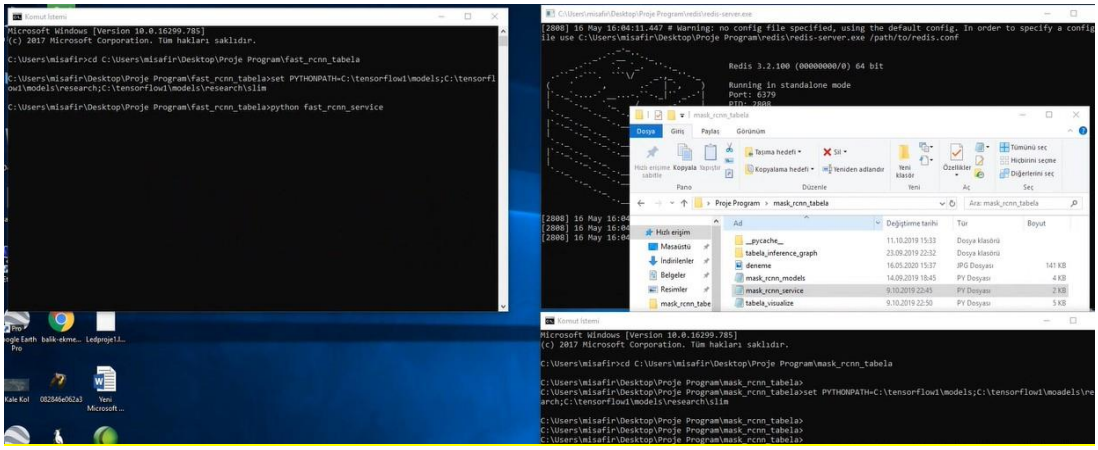
Kameradan gelen görüntüleri SSD_Mobile_Net okumaya başlar ve sırayla diğer derin öğrenme modellerine gönderek tanımlanan nesnelerin götüntüleri deneme.jpg adıyla sonuç olarak kayıt edilmektedir.



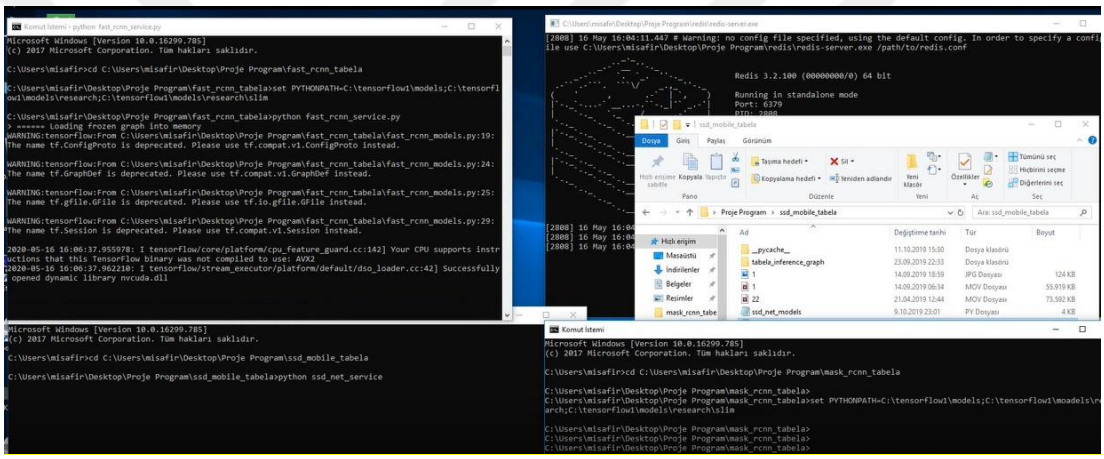
Şekil 37: Redis'in aktif edilmesi



Şekil 38: Fast R-CNN modelinin servisinin çalıştırılması

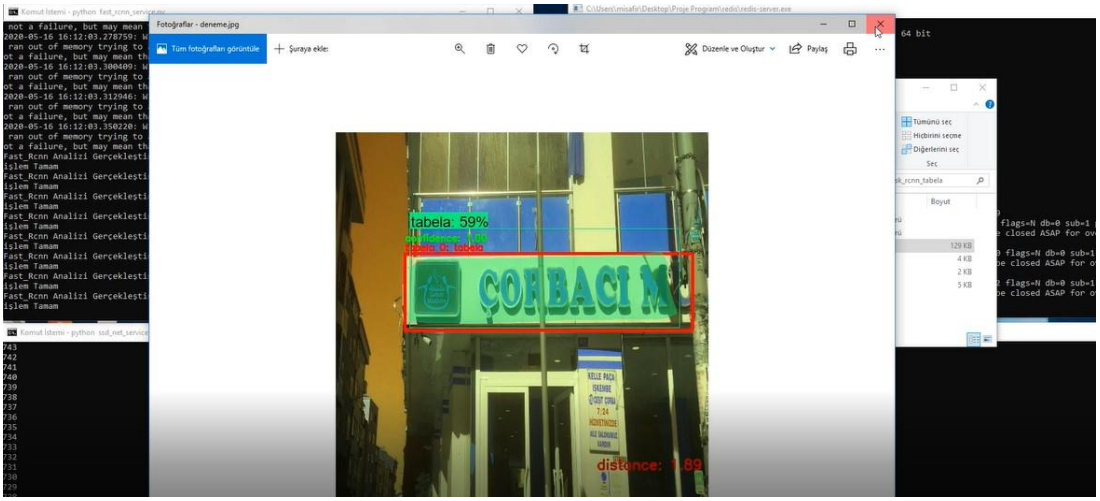


Şekil 39: Mask R-CNN modelinin servisinin çalıştırılması

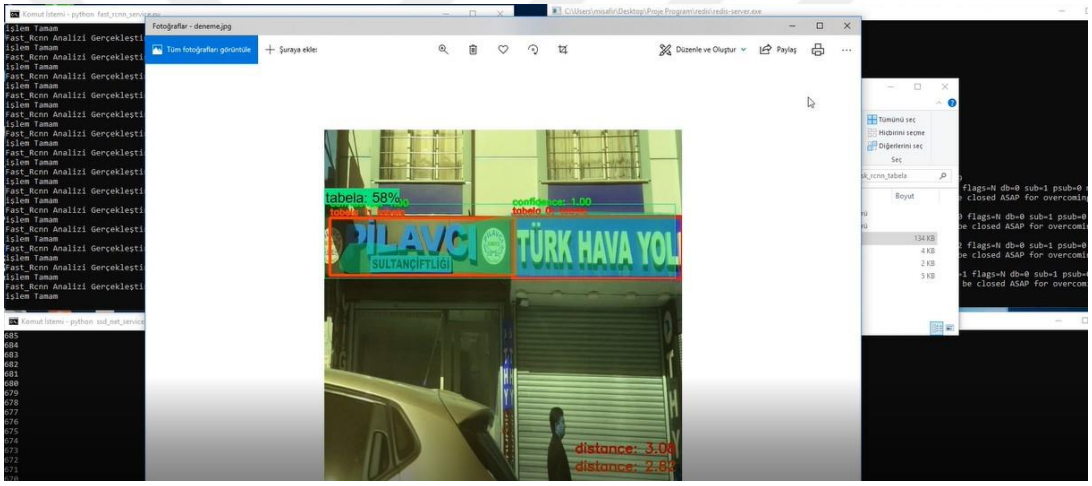


Şekil 40: SSD Mobile Net modelinin servisinin çalıştırılması

Hareket halindeki araçtan çekilen görüntülerin yeni geliştirilen sistemde anlık olarak işlenmesi ile elde edilen sonuçlar aşağıdaki resimler ile gösterilmiştir.



Şekil 41: Object Detection çıktı örneği 1



Şekil 42: Object Detection çıktı örneği 2

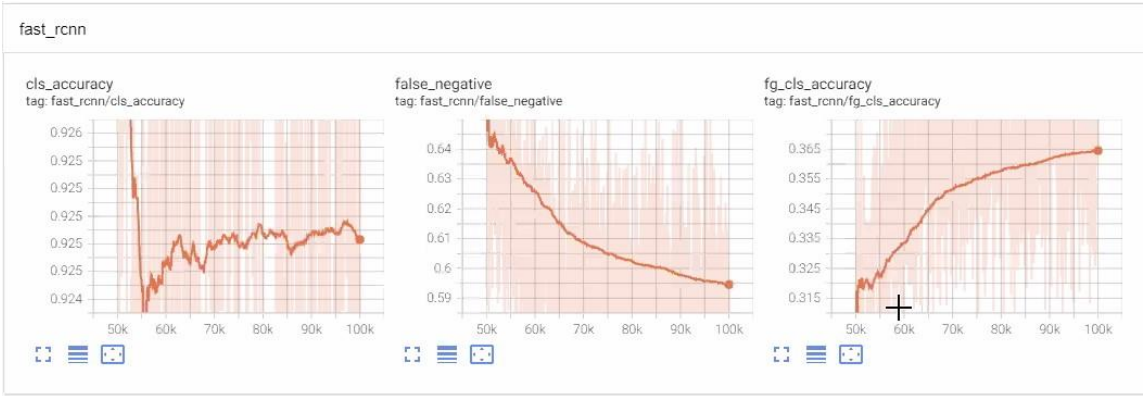
3.10. Mimarinin Doğrulanması ve Geçerlilik Testleri

Bu tez çalışmasında de mikroservis mimarisi nesne tanıma yapılabilmesinin yapısı kurularak araç içine konumlandırılmış bilgisayar ile görüntülerin kuyruğa alınarak farklı bir sunucuya aktarılması sağlanmış olup hareket halindeki araçtaki kameradan gelen çerçevelerin görüntü kaybına uğramadan nesne tanıması sağlanmıştır. Bu çalışmada 2 farklı test sonuç durumu belirlenebilmiştir.

1) Klasik mimari ile %65 oranında doğruluk sağlanmaktadır. Yeni önerilen mimari ile %90 ve üzeri doğruluk oranı yakalanmaktadır.

2) Klasik DL modelleri kullanılarak 50.000 veri seti ile %90 doğruluk oranına ulaşılabilecek iken mikroservis mimarisi sayesinde daha az (2500) veri seti ile %90 doğruluk (accuracy) oranına ulaşılmış oldu. Bu sayede daha az veri seti ile aynı doğruluk oranının yakalanması sağlanmış oldu.

Geliştirilen mimari ile Fast R-CNN ve Mask R-CNN modellerinin doğruluk değerleri aşağıdaki şekil43 ve şekil44 de gösterilmektedir.



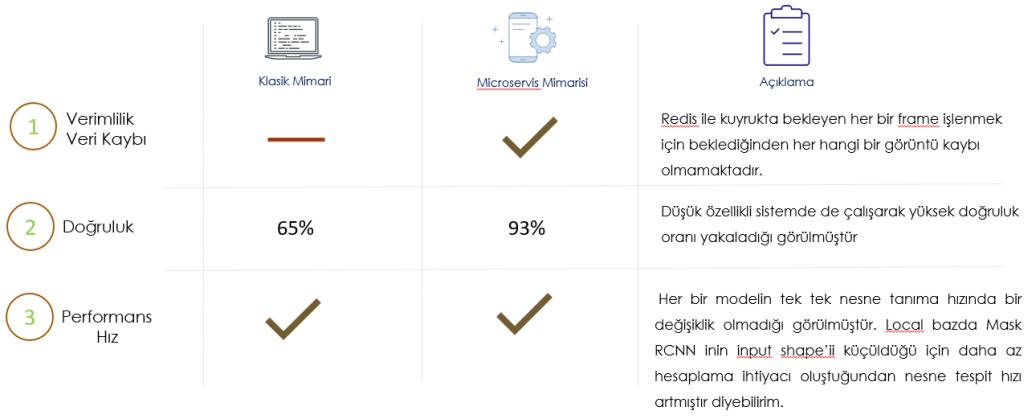
Şekil 43: Fast R-CNN Doğruluk (Accuracy) Değeri Grafiği



Şekil 44: Mask R-CNN Doğruluk (Accuracy) Değeri Grafiği

Gezgin araca konumlandırılmış kameradan saniyede 24 çerçeve gelmektedir. Derin öğrenme prosedürlerinde ise ortalama olarak 7 saniyede bir görüntü işlenebilmektedir. Kullanılan aracın hızı da hesaba katılırsa 7 sn de bir görüntü işlemek totalde binlerce görüntüyü kaybetmeye sebebiyet verecektir. Bu sebeple Redis teknolojisinden faydalanılarak oluşturulan kuyruk yapısı mikroservis mimarisi ile saniyede gelen 24 çerçevenin işlenmesi sağlanmıştır. Bu sayede her bir derin öğrenme modelinin sonuç üretebilmek için GPU belleğini tüketmesinin önüne geçilmiş oldu. Ayrıca veri kaybı önlenerek sistemin kararlı ve doğru çalışması sağlanarak performans artışı yakalanmıştır.

Tablo 5: Mikroservis Mimarisi ile Klasik Mimari Arasındaki Farklılıklar Tablosu



3.11. Deneysel Sonuçlar

Yapılan çalışma, Nvidia tarafından üretilen Nvidia Gpu gtx 1050 TI de ve Intel i7-7700 HQ 4 çekirdekli işlemciye ve 16GB RAM'e sahip masaüstü bilgisayarda uygulanmıştır. Elde edilen performans analizleri bu bölümden itibaren gösterilmiştir.

Sistemin çalışma hızı ile ilgili denemelerde öncelikle bilgisayarın CPU ve GPU kullanım oranı gözlemlenmiştir. Elde edilen sonuçlar Tablo 6'da verilmiştir.

Tablo 6: Mikroservis mimari sistemi performans karşılaştırması

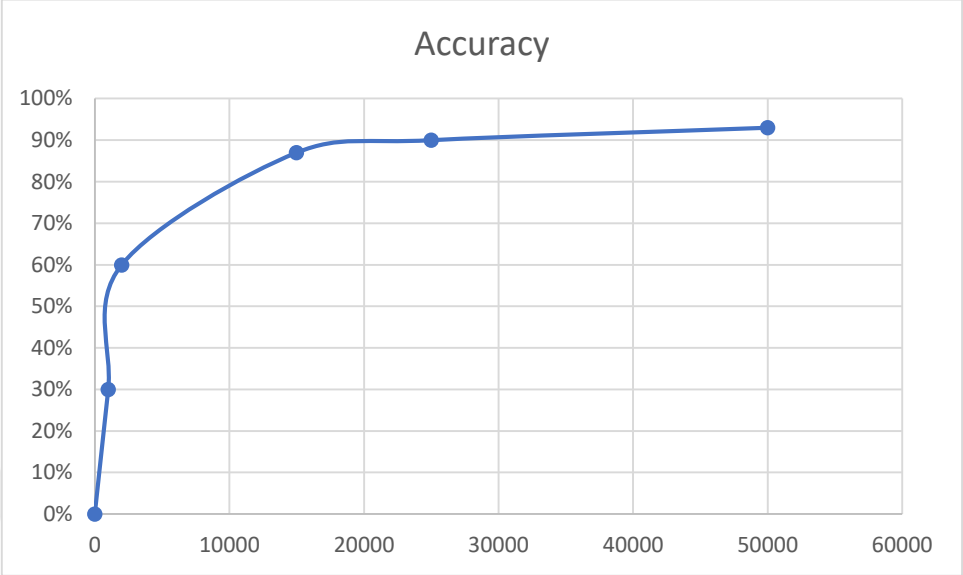
Araç Hız (km)	Donanım Türü	Nesne Çözünürlüğü	Çerçeve Başına İşlem Süresi (sn)
40	**CPU**	**576x768**	**1.220**
40	**GPU**	**576x768**	**0.160**
40	**Mikroservis mimarisi CPU +GPU**	**576x768**	**0.129**

Yeni tasarlanacak model ve gelişmiş özellikli grafik kartı ile (nvidia gtx 2070 ti) aracın hızı 60 km'ye ve çerçeve başına işlem süresi 0.015 saniyelere ulaşması hedeflenmektedir.

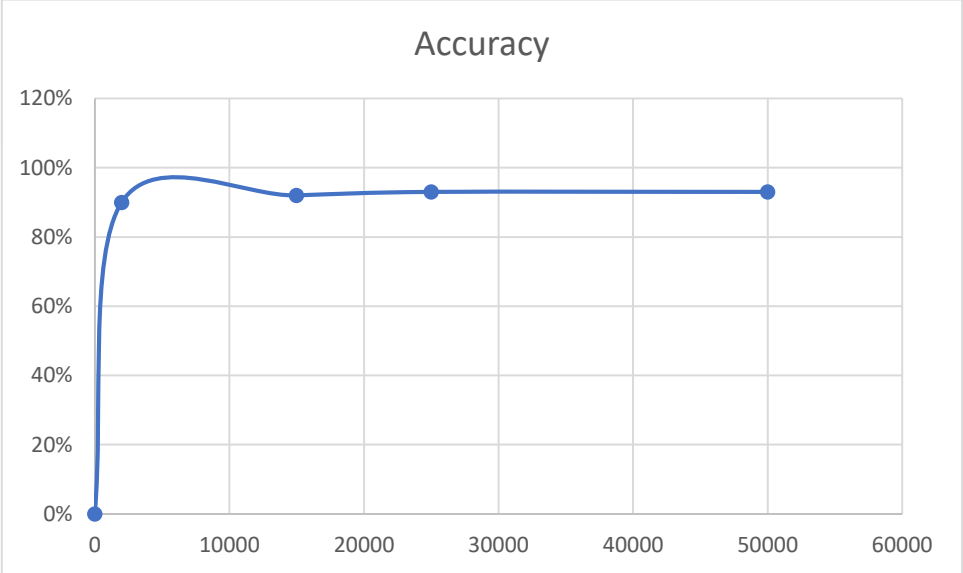
3.11.1. Doğruluk Oranı Karşılaştırması

Yapılan testlerde 10.000 adedin altındaki veri seti ile elde edilen sonuçlara göre klasik mimari doğruluk oranın %80'nin altında kaldığı bu tez kapsamında yeni önerilen

ve geliştirilen mikroservis mimarisi doğruluk oranın % 90'nın üzerine çıktığı görülmüştür. Şekil 45 ve şekil 46'da grafik olarak gösterilmektedir.



Şekil 45: Klasik Mimari ile Doğruluk Oranı Grafiği



Şekil 46: Mikroservis Mimarisi ile Doğruluk Oranı Grafiği

50000 veri ile ulaşabileceğimiz doğruluk oranına 2500 veri seti ile ulaşılmıştır.

BÖLÜM 4. SONUÇ

Nesne Tanıma tespitinde birçok yöntem mevcuttur. Bu yöntemlerden biri olan derin öğrenme mimarileri, nesne tespiti alanında son yıllarda sıklıkla kullanılan bir mimari olmuştur. Derin öğrenme mimarileri sayesinde daha büyük verilere sahip uygulamalarda çok daha hızlı ve başarılı sonuçlar elde edilmiştir. Klasik görüntü işleme metotları ile derin öğrenme mimarilerinin hibrit kullanımı durumunda nesne tespiti alanlarında daha başarılı sonuçlar ortaya konulabilir.

Bu tez çalışmasında, gezgin araca monte edilmiş bir kamera kullanarak belirlenmiş işyeri tabela nesnelerini tespit etmede performans artışı yakalanması ve veri kaybı oluşmaması için bir yöntem sunuldu. Yöntemde anlık yakalanan her bir görüntünün kuyrukta tutulmasıyla (CPU üzerinde) GPU üzerinde sadece nesne tanıma işleminin yapılması sağlanarak mikroservis mimarisi ile verimlik artımı sağlanmış ve veri kaybı oluşmasının önüne geçilmiştir. Geliştirilen mimari ile daha az veri ve daha az iterasyon sayısı ile daha yüksek doğruluk oranları yakalandığı görülmüştür.

Bu çalışma kapsamında gelecekteki araştırmalar için yine bu çalışma alanında kullanılmak üzere işyeri veri seti oluşturuldu. Tasarlanan mimari yapı ile gezgin araç içerisine yüksek performanslı donanıma ihtiyaç olmayan bir pc ile anlık nesne tanıma ve boyut ölçme işlemlerinin yapılmasına olanak sağlanmış oldu. Ayrıca mikroservis mimari yaklaşımı ile farklı sunucuların da nesne tanıma işlemi ile aynı anda boyut ölçme ve nesne üzerindeki yazının okunabilmesinin (OCR) önü açılmış oldu.

Geliştirilen uygulamanın yapılan testlerinde hem hız hem de doğruluk oranı artışının önemli olduğu durumlarda 3 farklı derin öğrenme modellerinin mikroservis mimarisi kurularak eşzamanlı çalıştırılması ile düşük donanımlı bilgisayarlarda gerçek zamanlı nesne tespiti ve görüntü analizi imkânı ortaya konulmuştur. Hata denetimi ve yönlendirme işlemlerinin kolaylıkla yapılmasına olanak sağlanmaktadır. Donanımsal olarak bir hata alındığında yönetimi daha kolay bir yapı kurulmuş oldu. Her bir modül ilk çalışmaya başladığı andan itibaren sıralı çalışmasının yerine sürekli çalışması sağlanarak her bir modülün her seferinde yeniden başlatılmasının önüne geçilmiş oldu.

Ayrıca tek aşamalı yöntemlerin (SSD Mobile Net) iki aşamalı yöntemler (RCNN) kadar yüksek doğrulukta nesne tespiti yapamaması söz konusudur. Tek aşamalı

yöntemler, çift aşamalı yöntemlere göre doğaları gereği genel olarak daha hızlıdır. Bu sebeple bu çalışmada çift aşamalı yöntem (Fast R-CNN, Mask R-CNN) ile tek aşamalı (SSD) yöntemlerin bir arada çalıştırılabildiği ve maksimum performans sağladığı da gösterilmiştir.

KAYNAKÇA

Abadi, M. A. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed system. *arXiv preprint arXiv:1603.*

Akdağ, A. (2017). Derin Öğrenme Algoritmaları Kullanılarak Gerçek Zamanlı Silah Tanıma Uygulaması. *Yüksek Lisans Tezi*.

Akiba, T. F. (2017). ChainerMN: Scalable Distributed Deep Learning. *arXiv preprint arXiv:1710.11351*.

Alshuqayran, N., Nour, A., & Roger, E. (2016). A Systematic Mapping Study in Microservice. *IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA.* IEEE.

Ambika Neelopant, D. S. (2021). Comparison of YOLOv3 and SSD Algorithms. *IJERT*.

Amidi, A. (2018). *Convolutional Neural Networks cheatsheet*. https://stanford.edu: https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks adresinden alındı

Arabi, S. A. (2019). A deep learning based solution for construction equipment detection: from development to deployment. *arXiv preprint*.

AWS. (2018). *Redis*. https://aws.amazon.com/tr/redis/: https://aws.amazon.com/tr/redis/ adresinden alındı

Aydın, S., Refik, S., & Bay, Ö. (2020). GPU programlamada CUDA platformu kullanılan paralel görüntü işleme çalışmalarının incelenmesi . *Politeknik Dergisi*, 737-754.

Bengio, Y. (2009). *Learning deep architectures for A.* Now Publishers Inc.

Bergstra, James, et a. (2010). Theano: A CPU and GPU math compiler in Python. *PROC. OF THE 9th PYTHON IN SCIENCE CON.*

Betts, T. (2020, 6 27). *The InfoQ eMag - Microservices: Testing, Observing, and Understanding*. https://www.infoq.com/microservices: https://www.infoq.com/microservices/minibooks/ adresinden alındı

Bharati, P. (2019). Deep Learning Techniques—R-CNN to Mask R-CNN: A Survey. *Computational Intelligence in Pattern Recognition*.

Building Microservices. (2014). S. Newman içinde, *Building Microservices: Designing Fine-Grained Systems.*

Chen, L. X. (2020). IoT Microservice Deployment in Edge-cloud Hybrid Environment Using Reinforcement Learning. *IEEE Internet of Things Journal.*

Cheng, J. M. (2014). *Professional CUDA c programming.* John Wiley & Sons.

CireAan, D. M. (2012). Multi-column deep neural network for traffic sign classification. *Neural Netwroks,* 333-338.

Derek, m. (2019, 10 30). *Azure'da mikro hizmetler oluşturma*. https://docs.microsoft.com/tr-tr/: https://docs.microsoft.com/tr-tr/azure/architecture/microservices/ adresinden alındı

Dinçer, B. (2020, 10 15). *Tensorflow Object Detection API ile Nesne Tanıma*. https://medium.com/: https://medium.com/deep-learning-turkiye/tensorflow-object-detection-api-ile-nesne-tan%C4%B1ma-c5162f8ddbc1 adresinden alındı

Doğan, F., & Türkoğlu, İ. (2019). Derin Öğrenme Modelleri ve Uygulama Alanlarına İlişkin Bir Derleme. *Dicle Üniversitesi Mühendislik Fakültesi Mühendislik Dergisi*, 409-445.

Durukan, E. (2016, 11 02). *TensorFlow ile Derin Öğrenmeye Giriş*. https://emredurukn.github.io/: 1. https://emredurukn.github.io/2016/11/02/tensorflow-ile-derin-ogrenmeye-giris.html adresinden alındı

Ergin, T. (2018, 1 2). *Convolutional Neural Network (ConvNet yada CNN) nedir, nasıl çalışır?* https://medium.com/: https://medium.com/@tuncerergin/convolutional-neural-network-convnet-yada-cnn-nedir-nasil-calisir-97a0f5d34cad adresinden alındı

Fowler, M. (2019). *Microservices Guide*. https://martinfowler.com: https://martinfowler.com/microservices/ adresinden alındı

Gandhi, R. (2018). *R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms.* https://towardsdatascience.com/.

Gers, F. A. (2020). Recurrent nets that time and count. *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks*.

Girshick, R. (2015). Fast R-CNN. *The IEEE International Conference on Computer Vision (ICCV)* (s. 1440-1448). Santiago, Şili:: IEEE.

Girshick, R. D. (2014). Rich feature hierarchies. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (s. 580-587). Columbus, Ohio, ABD: IEEE.

Gökçe, B., & Sonugür, G. (2016). GPS destekli imge çakıştırma yöntemi ile hareketli nesnelerin tespiti. *Pamukkale University Journal of Engineering Sciences*.

Gürbüz, F. (2020, 05 18). *Konvolüsyonel Sinir Ağları (Convolutional Neural Networks)(CNN)*. https://medium.com/: https://medium.com/@batincangurbuz/konvol%C3%BCsyonel-sinir-a%C4%9Flar%C4%B1-evri%C5%9Fimli-sinir-a%C4%9Flar%C4%B1-olarak-da-bilinir-convolutional-neural-4ecd2c5ad842 adresinden alındı

He, K. G. (2017). Mask R-CNN. *The IEEE* (s. 2961-2969). Venedik,İtalya: IEEE.

Hinton, G. E. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313(5786), 504-507.

Hinton, G. E. (2012). In Neural networks: Tricks of the trade. *A practical guide to training restricted Boltzmann machines.* (s. 599-619). Berlin,: In Neural networks: Tricks of the trade.

Hochreiter, S. &. (1997). Long short-term memory. *Neural computation*, 1735-1780.

Howard, A. G. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. . *arXiv*.

İnik, Ö., & Ülker, E. (2017). Derin Öğrenmede Kullanılan Veri Setleri ve Yazılım Kütüphaneleri. *1st International Symposium on Multidisciplinary Studies and Innovative Technologies Proceedings Book.* Tokat.

Joseph Redmon, S. D. (2016). You Only Look Once: Unified, Real-Time Object Detection. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, (s. 779-788).

Kayaalp, D. K. (2018). *Derin Öğrenme ve Türkiye'deki Uygulamaları.* İksad.

Kızıltan, Ç. (2018, 08 2). *Mikroservis Mimarisi ve Avantajları*. http://cagataykiziltan.net/mikroservis/: http://cagataykiziltan.net/ adresinden alındı

Krizhevsky, A., & Geoffrey, E. (2011). Using very deep autoencoders for content-based image retrieval. *ESANN*, (s. 2).

Kurt, F. (2015, 07 21). *Derin Öğrenme (Deep Learning) Nedir?* https://www.derinogrenme.com/: https://www.derinogrenme.com/2015/07/21/derin-ogrenme-deep-learning-nedir/ adresinden alındı

Mercanoğlu Sincan, Ö., Yalım Keleş, H., & Tosun, S. (2018). Moving object detection and classification in surveillance systems using moving cameras. *Communications Faculty of Sciences University of Ankara Series A2-A3 Physical Sciences and Engineering*.

microservices, W. a. (2016). http://microservices.io/: http://microservices.io/ adresinden alındı

microsoft. (2019, 11 4). *Etki alanı odaklı tasarım kullanarak mikro hizmetlere monolitik*. https://docs.microsoft.com/: https://docs.microsoft.com/tr-tr/azure/architecture/microservices/migrate-monolith adresinden alındı

Mikolov, T. (2010). Recurrent Neural Network Based Language Model. *11th Annual Conference of the International Speech Communication Association.* Chiba,Japan.

Pahl, C. (2015). Containerization and the paas cloud. *IEEE Cloud Computing*, 24-131.

Parsons, R. (2018, 5 15). *Microservices in Adopt*. https://www.thoughtworks.com/: https://www.thoughtworks.com/insights/blog/microservices-adopt adresinden alındı

Rakkshab Varadharajan Iyer, P. S. (2021). Comparison of YOLOv3, YOLOv5s and MobileNet-SSD V2 for Real-Time. *IRJET*.

Redmon J., D. S. (2016). You only look once: Unified, real-time object detection. *IEEE conference on computer vision and pattern recognition*, (s. 779-888).

Ren, S. H. (2016). FASTER R-CNN : Towards RealTime Object Detection with Region Proposal Networks. *Advances in Neural Information Processing Systems (NIPS).* Montreal, Kanada.

Resul DAŞ, B. P. (2019). Derin Öğrenme İle Resim ve Vidolarda Nesnelerin Tanınması ve Takibi. *Fırat Üniversitesi Müh. Bil. Dergisi*, 571-581.

Salakhutdinov, R., & Hinton, G. (2006). Reducing the Dimensionality of Data with Neural Networks,. *Science*.

Schmidhuber, J. (2012). Multi-column deep neural networks for image classification. *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR.*

Söylemez, M., & Tarhan, A. (2018). Mikroservis Mimarisi ve Mimari Faktörleri Üzerine Endüstriyel Bir İnceleme. *Proceedings of the 12th Turkish National Software Engineering Symposium.* İstanbul.

Supriya, M. H. (2020). Modular Microservice based GPU Utilization Manager with Gunicorn. *Journal of Science and Technology*, 230-237.

TECH INSIGHTS. (2019). *Microservices: How to deliver scalable software, faster*. https://tanzu.vmware.com/microservices: https://tanzu.vmware.com/microservices adresinden alındı

Theano Development Team. (2016). Theano: A Python framework for fast computation of mathematical expressions. *Computer Science*.

Thones, J. (2015). Microservices. *IEEE Software*, 116.

Uijlings, J. v. (2012). Rich feature hierarchies. *Selective Search for Object Recognition. International Journal*, 154-171.

Wallach, I., Dzamba, M., & Heifets, A. (2015). AtomNet: A Deep Convolutional Neural Network for Bioactivity Prediction in Structure-based Drug Discovery. *arXiv.org*.

Yang, Z. Y. (2008). Parallel image processing based on CUDA. *Parallel image processing based on CUDA* (s. 593-600). Shaanxi: International Conference on Computer Science and Software Engineering.