GRAFİK İŞLEME ÜNİTESİ (GPU) TABANLI ÖĞRENME KULLANARAK OTONOM ARAÇLAR İÇİN ALGILAMA SİSTEMİNİN GELİŞTİRİLMESİ

Mehmet Safa BİNGÖL

Yüksek Lisans Tezi

Mekatronik Mühendisliği Anabilim Dalı

Danışman: Doç. Dr. Ayşegül UÇAR

AĞUSTOS-2018

T.C.
FIRAT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

GRAFİK İŞLEME ÜNİTESİ (GPU) TABANLI ÖĞRENME KULLANARAK OTONOM ARAÇLAR İÇİN ALGILAMA SİSTEMİNİN GELİŞTİRİLMESİ

YÜKSEK LİSANS TEZİ

Mehmet Safa BİNGÖL
(151134106)

Tezin Enstitüye Verildiği Tarih : 16 Temmuz 2018
Tezin Savunulduğu Tarih : 2 Ağustos 2018

Tez Danışmanı : Doç. Dr. Ayşegül UÇAR (F.Ü.)
Diğer Jüri Üyeleri : Doç. Dr. Mehmet KARAKÖSE (F.Ü.)
Dr. Öğr. Üyesi Özal YILDIRIM (M.Ü.)

AĞUSTOS-2018

ÖNSÖZ

Lisans ve yüksek lisans eğitimimde beraber çalışma fırsatı bulduğum değerli hocam Sayın Doç. Dr. Ayşegül UÇAR'a sonsuz teşekkürlerimi sunarım.

Bu çalışma, Fırat Üniversitesi Bilimsel Araştırma Projeleri (FÜBAP) komisyonu tarafından MF.17.05 proje numarasıyla desteklenmiştir. FÜBAP komisyonuna teşekkür ederim.

Çalışmalarımda bana yardımcı olan değerli arkadaşım Arş. Gör. Çağrı KAYMAK'a, parkurun ve çalışma ortamının hazırlanmasında yardımcı olan bölüm teknisyeni Ali Rıza DERMANLI'ya, malzeme bağışları ve yazılım desteği için Ferhat KURT'a teşekkür ederim.

Hayatım boyunca bana olan desteklerinden ve bu çalışmam süresince bana gösterdikleri hoşgörüden ötürü değerli aileme teşekkür ederim.

Mehmet Safa BİNGÖL

ELAZIĞ - 2018

İÇİNDEKİLER

ÖZET

GRAFİK İŞLEME ÜNİTESİ (GPU) TABANLI ÖĞRENME KULLANARAK OTONOM ARAÇLAR İÇİN ALGILAMA SİSTEMİNİN GELİŞTİRİLMESİ

Otonom araçlar çevre koşullarını algılayarak kararlar alan ve aldıkları kararlar doğrultusunda hareket eden araçlardır. Günümüzde otonom araçlara olan ilgi hızla artmaktadır. Gelişen, sensör, Grafik İşleme Birimi (GPU - Graphics Process Unit) teknolojisi ve yapay öğrenme yöntemlerindeki yenilikler ile birlikte otonom araç teknolojisi de gelişmektedir.

Bu çalışmada ilk olarak, otonom araçlar ile ilgili genel bilgiler verilmiştir. İkinci olarak, Linux ortamında Caffe kullanılarak, MNIST veri kümesi ile rakam tanıma yapılmıştır. Bu uygulama için Nvidia TK1 ve Nvidia TX1 kartlarına ek olarak iki adet bilgisayar kullanılmış ve Merkezi İşlem Birimi (CPU - Central Process Unit) ile GPU'lar hız ve doğruluk bakımından değerlendirilmiştir. Üçüncü olarak, nesne tanıma uygulamaları yapılmıştır. Bu amaçla ilk olarak, Windows ortamında Matlab'da yaya geçidi trafik tabelası algılama uygulaması yapılmıştır. Bu uygulama için Elazığ, Niğde ve Kayseri şehirlerinin yaya geçitlerinde çekilen görüntüler ile oluşturulan veri kümesi kullanılmıştır. Algılama için Bölgesel Konvolüsyonel Sinir Ağı (B-KSA), Daha Hızlı Bölgesel Konvolüsyonel Sinir Ağı (Daha Hızlı B-KSA) kullanılmış ve sonuçlar incelenmiştir. Sonrasında, Linux ortamında temel anlamda YOLO kullanılarak nesne tanıma uygulaması yapılmış ve çıktılara yer verilmiştir. Son olarak, küçük bir yer aracı ile yapay öğrenme yöntemlerini kullanan otonom bir araç yapılmıştır. Bu amaçla, yer aracı üzerine çeşitli sensörler, kamera ve Nvidia TX2 kartı yerleştirilmiştir. Otonom araç, tasarlanan parkur üzerinde test edilmiştir.

Tüm uygulamalar başarılı bir şekilde gerçekleştirilmiştir. Elde edilen sonuçlar karşılaştırmalı olarak ve şekiller ile verilmiştir.

Anahtar Kelimeler: Otonom Araçlar, GPU, Konvolüsyonel Sinir Ağı, Bölgesel Konvolüsyonel Sinir Ağı, YOLO

SUMMARY

DEVELOPMENT OF PERCEPTION SYSTEM FOR AUTONOMOUS VEHICLES BY USING GPU-BASED LEARNING

Autonomous vehicles are that perceive environmental conditions and act in accordance with decisions. At present, interest in autonomous vehicles is increasing rapidly. With the development of sensor and GPU (Graphics Process Unit) technology, and innovations in artificial learning methods, autonomous vehicle technology is also developing.

In this study, firstly, general information about autonomous vehicles was given. Secondly, using the Caffe in the Linux environment, the MNIST dataset was used for digit recognition. For this application, two computers were used in addition to the Nvidia TK1 and Nvidia TX1 cards, and the CPU (Central Process Unit) and GPUs were evaluated for speed and accuracy. Thirdly, object recognition applications have been implemented. For this purpose, firstly, in Matlab in Windows environment, pedestrian crossings traffic sign detection application was done. The originally created dataset was used for this application. For detection, Region Convolution Neural Network (R-CNN), Faster R-CNN was used and the results were examined. Later in the Linux environment, object recognition was implemented using Yolo in a basic sense and output was given. Finally, a small autonomous land vehicle was built using artificial learning methods. For this purpose, various sensors, camera and Nvidia TX2 card were installed on the land vehicle. The autonomous vehicle were tested on the designed racecourse.

All applications were realized successfully. The results were given comparatively in tables and figures.

Keywords: Autonomous Vehicles, GPU, Convolutional Neural Network, Region Convolutional Neural Network, YOLO

ŞEKİLLER LİSTESİ

Sayfa No

TABLOLAR LİSTESİ

Sayfa No

KULLANILAN TERİMLER LİSTESİ

Pooling : Öznitelik seçici
Padding : Doldurma
Stride : Kaydırma
Subsampling : Alt örnekleme
Batch Size : Yığın boyutu
Flatten : Vektöre dönüştürme
Dense : Sık bağlı katman

KISALTMALAR LİSTESİ

KSA **:** Konvolüsyonel Sinir Ağı

B-KSA **:** Bölgesel KSA (R-CNN - Region Convolution Neural Network)

CPU **:** Merkezi İşlem Birimi (Central Process Unit)

GPU **:** Grafik İşlem Birimi (Graphics Process Unit)

YOLO **:** Sadece Bir Defa Bak (You Only Look Once)

HOG **:** Gradyanın Histogramı (Histogram of Gradient**)**

OBOD **:** Ölçekten Bağımsız Öznitelik Dönüşümü (SIFT - Scale Invariant Feature Transform)

HGO **:** Hızlandırılmış Gürbüz Öznitelikler (SURF - Speeded Up Robust Features)

IGTO **:** İkili Gürbüz Temel Öznitelikler (BRIEF - Binary Robust Independent Elementary Feature)

YHDB **:** Yönlendirilmiş Hızlı ve Döndürülmiş Brief (Oriented Fast and Rotated BRIEF)

IGDOAN **:** İkili Gürbüz Değişmez Ölçeklendirilebilir Anahtar Noktalar (BRISK - Binary Robust Invariant Scalable Keypoints)

HRN **:** Hızlı Retina Noktası (Fast Retina Keypoint)

DVM **:** Destek Vektör Makine

AOM **:** Aşırı Öğrenme Makinesi

KA **:** Karar Ağacı

DDB **:** Doğrultulmuş Doğrusal Birim (ReLU - Rectified Linear Unit)

1. GİRİŞ

Otonom araçlara olan ilgi dünya çapında giderek artmaktadır. Bu teknolojideki gelişim potansiyeli çok açık bir şekilde görülmektedir. Öyle tahmin ediliyor ki çok yakın zamanda ulaşım ve taşımacılık çarpıcı bir biçimde değişecektir. Otonom sürüş, trafik akışı ve park problemlerine yardımcı olarak, sürüş özelliklerini geliştirerek ve yakıtı daha verimli kullanarak şehirlerdeki kirlenmeyi azaltmaya yardımcı olmaktadır. Bunlara ek olarak, otonom araçlar ile birlikte insan ve nakliye taşımacılığında insan kaynaklı hatalar azaldığından güvenlik oldukça artmaktadır [1-2].

DARPA Türkçe adıyla ABD Savunma Bakanlığı İleri Araştırma Projeleri Ajansı 1958 yılında ARPA (Advanced Research Projects Agency) adıyla Virginia'da kurulmuştur. Ordu tarafından kullanılmak üzere, yeni teknolojiler üretmekten sorumlu kuruluştur. Sürücüsüz araç teknolojilerinin gelişmesi ile birlikte DARPA tarafından otonom araç yarışları düzenlenmeye başlanmıştır. 2002 yılında ABD Savunma Bakanlığı "Büyük Mücadele" anlamına gelen Grand Challenge adıyla bir yarışma başlatmıştır. Sadece Amerikalıların katılabileceği bu yarışmada, otonom bir aracın Amerika'daki bir çölde belirlenmiş hedefe kendi kendine gitmesini sağlayabilecek bir sistem geliştirmek amaçlanmıştır. 2004 yılında sonlanan bu yarışmanın ödülü 1 milyon dolar olmasına rağmen katılan 25 takımdan bu ödülü alan takım çıkmamıştır. Ancak 2005 yılında yinelenen yarışta beş takım 217 km'lik rotayı tamamlamayı başarmıştır, böylece Stanford Üniversitesi'ni temsil eden takım 2 milyon dolarlık büyük ödülü kazanmıştır. 2007 yılında 3,5 milyon dolara çıkarılan toplam ödül yarışmayı tamamlayabilen altı sürücüsüz araç tarafından paylaşılmıştır. DARPA Şehir Mücadelesi adıyla bilinen bu yarışmada hedef 89 km'lik şehir içi trafiğinde, ralli stili bir yarış sonunda hedefe ulaşmaktı. Şekil 1.1'de DARPA'ya katılan otonom araçlardan bir tanesi gösterilmektedir.

Şekil 1.1. DARPA'ya katılan otonom bir araç [3]

Son yıllarda, otonom sürüşün kullanım alanlarının çokluğu nedeniyle, hükümetler ve araştırma enstitüleri hem büyük miktarlarda çalışma gücü hem de finansal yatırım yapmışlardır. Tamamen otonom araçlar gerçek hayatta kullanılır hale gelmiştir, örneğin Kasım 2015 de Google'ın otonom arabası gerçek yaşam alanlarında 1.9 milyon kilometreden fazla sürüş yapmıştır. Bu da bir insanın yaklaşık 90 yıllık sürüş deneyimine karşılık gelmektedir [4].

Teknolojiler arası transfer sağlanmış ve araştırmalardaki gelişmeler mevcut ticari araçlarda İleri Sürücü Destek Sistemleri adı altında yavaş yavaş tanıtılmıştır. Dahası, birçok şirket (Nissan, Mercedes, Volvo, Tesla, vb.) 2020'ye kadar otonom ticari araçların farklı modellerini üretme niyetlerini duyurmuştur. Diğer taraftan, hükümetler yasa çıkarmaya başlamış ve otonom araçların yollarda test edilmesi ve sürülmesine müsaade edilmiştir. Örneğin, California'da 2012 yılından beri hükümet bu çalışmalara izin vermiştir.

Otonom araçların git gide daha önemli hale gelmesine rağmen, bu teknoloji henüz olgunlaşmış değildir. Bizim şehirlerimiz ve yollarımız yayalar, hayvanlar, sokak mobilyaları veya diğer araçların bir arada oluşu gibi öngörülemeyen dinamik ortamlar içerir. Bu nedenle, otonom araçları çevresini doğru bir şekilde anlaması için sağlam algı sistemleri ile donatmak gerekir ve bu yüzden doğru bir şekilde hareket edebilmeleri için çevresinde ne olduğunu yorumlayabilmelidirler.

Bir otonom araç için çevresinden bilgi almanın ilk yolu bir elektronik kart üzerindeki çeşitli sensörlerdir. Günümüzde, otomotiv radarları, çok katmanlı lazer tarayıcılar, görüntü ve derinlik yakalayan kameralar gibi yeni uygun sensörler pazara girmiş ve mobil robotlara ve araçlara dahil edilmiştir.

Yazılımdaki gelişmelere bağlı olarak, geçtiğimiz yıllarda Bilgisayar Görmesi, Makine öğrenmesi ve Mobil Robot araştırmaları alanlarında önemli gelişmeler yaşanmıştır. Özellikle derin öğrenme alanındaki gelişmeler umut verici sonuçlar içermektedir.

Yukarıda ifade edilen yenilikler bizi araştırmalara ve yeni gelişmelere katılmak için motive etmiştir. Standart bilgisayar görmesi yaklaşımlarının yerine; derin öğrenme tabanlı algoritmaları, otonom araçların algılama sistemlerinde kullanmaya yöneltmiştir.

Temelde doğal gelişim ve eniyileme süreçlerini modellemeye yönelik yaklaşımlar içeren evrimsel algoritmalar, eniyileme problemlerinin makul sürelerde ve kabul edilebilir sonuçlarla çözülmesi konusunda ciddi bir hızlandırma sağlamaktadırlar. Bunun yanında problem sahası genişledikçe bu hızlanma dahi uygun bir zaman diliminde çözüm üretmekte yetersiz kalmaktadır.

İşlem gücünün artması ise sistemin maliyetiyle doğru orantılıdır. Buna bağlı olarak çözüm süresi yine kabul edilir seviyenin altında kalabilmektedir. Bu nedenle performansı artırmak ve gerekirse gerçek zamanlı sistemlerde kullanabilmek açısından problemlerin paralel ortamda gerçeklenmesi etkin bir yöntem olarak karşımıza çıkmaktadır.

Sistemler iki temel donanım üzerinde gerçeklenmektedirler. Bunlar, çok çekirdekli Merkezi İşlem Birimleri (CPU - Central Process Unit) ve Grafik İşlem Birimleridir (GPU - Graphics Process Unit). Geleneksel CPU mimarisine kıyasla GPU'lar, yoğun ve kolay ulaşılabilir bir paralel hesaplama gücü sağlamaktadırlar. Bundaki en önemli faktör ise yüksek maliyeti olan ve artık limitlerine yaklaşan yüksek frekanslı birkaç çekirdekten ziyade daha düşük frekansa, dolayısıyla düşük maliyete sahip binlerce çekirdeğin aynı ünite üzerinde toplanmasıdır. Paralel yaklaşımlarda amaç, genelde problem sahasının paralel çözülebilecek parçalara ayrılması ve işlem parçacıklarının ilgili kısımları paralel işleyip ortak çözümler üretmesi, temelde zaman alan hesaplamaların paralel çözülmesi veya bunların birlikte ele alındığı yöntemler üretmektir [5]. Bugünlerde ise yüksek hesaplama gücü, paralel programlanabilir, çok çekirdekli işlemcili, çok iş parçacıklı, programlanabilir, çok yüksek bellek bant genişliği sunan GPU'lar piyasada mevcuttur [6].

Nesne algılama ve tanıma, otonom sürüş uygulamalarının en önemli araştırma konularından biridir. Çünkü, bir kontrol hareketi öncelikle nesneyi algılar ve sonra o nesneyi tanımlar. Son zamanlarda, gerçek hayatta kullanılan araçlar için nesne tanıma uygulamaları hızla büyümüştür. Bu uygulamalara örnek olarak, şerit algılama asistanı, stereo görüntü kullanarak engel algılama, kızılötesi görüntü kullanarak yaya algılama uyarı sistemi [7-10], lazer radar ve tek lensli kamera sistemi kombinasyonu kullanarak araç algılama sistemleri verilebilir [11].

Otonom araç uygulamalarının ana amacı trafik işaretleri, hayvan, araba, kamyon, motorsiklet ve insan gibi statik veya dinamik nesneleri algılamak, takip etmek ve tanımaktır. Nesne tanıma bilgisayar görmesinin zorlu konularından biridir. Bilimsel yazında, çeşitli şekillerde özellik çıkarma algoritmaları geliştirilmiştir. Bu algoritmalara örnek olarak, Gradyanın Histogramı (HOG - Histogram of Gradient) [12], Ölçekten Bağımsız Öznitelik Dönüşümü (SIFT - Scale Invariant Feature Transform) [13,14], Hızlandırılmış Gürbüz Öznitelikler (SURF - Speeded up Robust Feature) [15], İkili Gürbüz Temel Öznitelikler (BRIEF - Binary Robust Independent Elementary Feature) [16], Yönlendirilmiş Hızlı ve Döndürülmiş BRİEF (ORB - Oriented Fast and Rotated BRIEF) [17], İkili Gürbüz Değişmez Ölçeklendirilebilir Anahtar Noktalar (BRISK - Binary Robust Invariant Scalable Keypoints) [18] ve Hızlı Retina Noktası (FREAK - Fast Retina Keypoint) [19] verilebilir.

Diğer yandan nesne algılama ve tanıma için Yapay Sinir Ağları (YSA'lar) [30]; Küresel / Eliptik sınıflayıcı [20], Destek Vektör Makineleri (DVM'ler) [20], Aşırı Öğrenme Makineleri (AÖM'ler) [21, 22], Adaboost, Naïve Bayes ve Karar Ağaçları (KA'lar) [23, 24] gibi birçok sınıflayıcı vardır.

Son yıllarda, derin öğrenme yöntemleri nesne algılama ve tanıma için güçlü makine öğrenmesi yöntemleri olarak ortaya çıkmıştır [25-27]. Derin öğrenme yöntemleri, bütün geleneksel yaklaşımlardan farklıdır. Derin öğrenme yöntemleri, özellikleri ham piksellerden doğrudan öğrenir. Derin öğrenme yöntemlerinde, yerel algılama alanları katman katman büyür. Çizgi, sınır ve köşe gibi basit özellikleri, düşük seviyeli katmanlar; yaya, araba veya trafik işaretleri gibi yüksek özellikleri yüksek seviyeli katmanlar algılar. Diğer bir ifadeyle derin öğrenme yöntemleri, nesneleri farklı ayrıntı düzlemlerinde temsil etmeye müsaade eder. Derin öğrenme yöntemlerinin başarıları ImageNet sınıflama yarışmalarında kendini göstermiştir [28, 29]. Bu yarışmada derin sinir ağlarının bir türü olan DKSA'lar kullanılmıştır.

Yeni nesil otonom araçlarda araç içerisindeki tüm çoğul ortamlı sistemler ve durumsal farkındalığı sağlayan algılayıcılar, tek birim tarafından komuta edilmektedir. Kamera ve sensörler vasıtasıyla çevresinden aldığı bilgileri, derin öğrenme yöntemlerini kullanarak değerlendiren otonom bir araç örneği Şekil 1.2'de verilmiştir. Araç yönetim sistemi sahip olduğu derin öğrenme yapısı sayesinde kameralar vasıtasıyla aldığı görüntüleri eş zamanlı sınıflandırarak sürücü destek sistemini oluşturmaktadır. Bu sayede özellikle kısıtlı görüş şartları gibi birçok durumda kazaların önüne geçilebileceği değerlendirilmektedir.

Bu tez çalışmasında, küçük bir yer aracı üzerine yerleştirilen kamera ve sensörler yardımıyla, derin öğrenme algoritmaları kullanılarak otonom bir araç yapılması amaçlanmıştır. Bu amaca uygun araç ve aracın üzerinde hareket edeceği parkur hazırlanmıştır. Öncelikle parkur üzerinde manuel olarak hareket ettirilen araçtan kamera görüntüsü ve sensör verileri alınmıştır. Daha sonra elde edilen bu veriler kullanılarak Konvolüsyonel Sinir Ağı (KSA) eğitilmiştir.

Sonuç olarak, bu tez çalışması için hazırlanan otonom araç derin öğrenme algoritması kullanılarak parkur üzerinde başarılı bir şekilde hareket ettirilmiştir.

1.1. Tezin Yapısı

Tezin geri kalan bölümleri şu şekilde düzenlenmiştir.

İkinci bölümde, yapay zekâ ve öğrenme yöntemleri ile ilgili genel bilgiler verilmiştir. Uygulamalarda kullanılan KSA, B-KSA, Daha Hızlı B-KSA ve YOLO yöntemleri incelenmiştir.

Üçüncü bölümde, çeşitli CPU ve GPU'lar kullanılarak rakam tanıma yapılmıştır. Elde edilen sonuçlar sayesinde CPU ve GPU'lar hız ve doğruluk açısından karşılaştırılmıştır.

Dördüncü bölümde, yaya geçidi trafik tabelası için kendi oluşturduğumuz veri kümesi ile B-KSA ve Daha Hızlı B-KSA yöntemleri kullanılarak yaya geçidi trafik tabelası tanıma uygulaması yapılmış ve sonuçlar değerlendirilmiştir. Buna ek olarak temel anlamda YOLO kullanılarak nesne tanıma uygulaması yapılmıştır.

Beşinci bölümde, küçük bir yer aracı üzerine kamera ve sensörler yerleştirilmiştir. Daha sonra aracın üzerinde hareket edebileceği bir parkur hazırlanmıştır. Hazırlanan parkur üzerinde manuel olarak hareket ettirilen araçtan kamera görüntüsü ve sensör verileri alınmıştır. Elde edilen bu veriler kullanılarak konvolüsyonel yapay sinir ağı eğitilmiştir.

Altıncı bölümde, tezde elde edilen sonuçlar ve öneriler sunulmuştur.

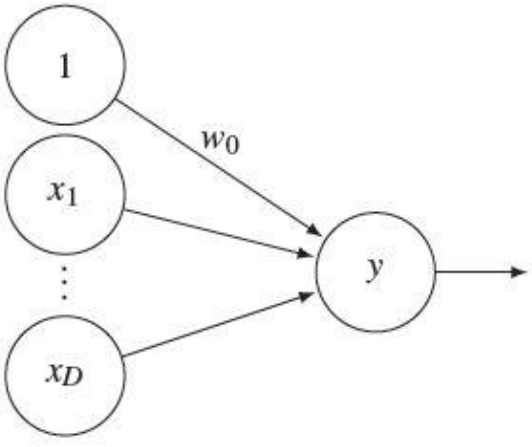
2. YAPAY SİNİR AĞLARI VE ÖĞRENME YÖNTEMLERİ

YSA'lar, sinasplar ile birbirine bağlanan sinirlerden oluşan insan beyninin öğrenme kabiliyetinden esinlenilerek oluşturulmuştur. Aslında YSA'lar, kuramsal olarak verilen herhangi bir dönüşüm görevini belli bir doğruluk oranıyla öğrenme yeteneğine sahiptir [31]. Ek olarak, YSA'lar kendisine verilen önceki görevlerle ilgili bilgileri, ağ mimarisi ile kolayca birleştirir.

YSA'lar, birbirine bağlanmış işlem birimi kümelerinden oluşur [33]. x1,…,x_D giriş ve w_0 harici giriş ağırlığı verildiği zaman, bir işlem birimi kendi çıkışını y=f(z) olarak hesaplar. Burada f, aktivasyon fonksiyonu olarak isimlendirilir ve y yayılma kuralları uygulanarak elde edilen çıkıştır. Bahsedilen açıklama [34]'de tanımlanan ve bir tane işlem birimi içeren modeli anlatmaktadır.

YSA'lar, ağ grafiği olarak adlandırılan grafikler ile gösterilebilir [33]. Her birim, kendi çıkışı ile isimlendirilmiş bir düğüm ile gösterilir ve birimler kendi arlarında bağlıdırlar. Şekil 2.1'de bir işlem birimi için örnek gösterilmiştir [33]. Burada harici giriş olan w_0, 1 girişli bias olarak düşünülebilir.

Bütün olarak ağ, y(x) fonksiyonunu gösterir. y(x) fonksiyonu giriş birimi ve çıkış birimi ile düzenlenir. Bunun anlamı ağın girişi, giriş birimleri tarafından kabul edilirken; ağın çıkışı, çıkış birimlerinden elde edilir.



Şekil 2.1. YSA için bir işlem birimi [34]

2.1. Çok Katmanlı Algılayıcılar

(L+1) katmanlı algılayıcının katmanları Şekil 2.2'de gösterilmiştir. Aslında bu ağda giriş ile birlikte (L+2) tane katman vardır. Bu ağda, giriş katmanında D tane giriş birimi, çıkış katmanında C tane çıkış birimi ve çok sayıda gizli katman vardır. Birimler, çok katmanlı bir algılayıcı olarak, bir giriş katmanı, bir çıkış katmanı ve L tane gizli katman olarak düzenlenmiştir. l katmanındaki i. birim çıkışı (2.1)'deki gibi hesaplanır:

$$y_i^{(l)} = f\,(z_i^{(l)}) \text{ ile } z_i^{(l)} = \sum_{k=1}^{m^{(l-1)}} w_{i,k}{}^{(l)} y_k{}^{(l-1)} + w_{i,0}{}^{(l)}. \quad (2.1)$$

Burada; $w_{i,k}^{(l)}$, (l-1). katmandaki k. birimini, l. katmandaki i. birime bağlayan ağırlıklandırılmış bağlantıyı göstermektedir ve $w_{i,0}^{(l)}$ harici giriş olarak bilinir ve biası gösterir. $m^{(l)}$; l katmanındaki birimlerin sayısını gösterir, $D=m^{(0)}$ ve $C=m^{(L+1)}$ ise sırasıyla giriş ve çıkıştaki birim sayısını gösterir. Basitlik için, bias, $y_0^{(l)}:=1$ olarak tanımlanan yalancı girişinin ağırlığı olarak düşünülebilir:

$$y_i^{(l)} = \sum_{k=0}^{m^{(l-1)}} w_{i,k}{}^{(l)} y_k{}^{(l-1)} \quad \text{veya} \quad z^{(l)} = w^{(l)} y^{(l-1)}. \quad (2.2)$$

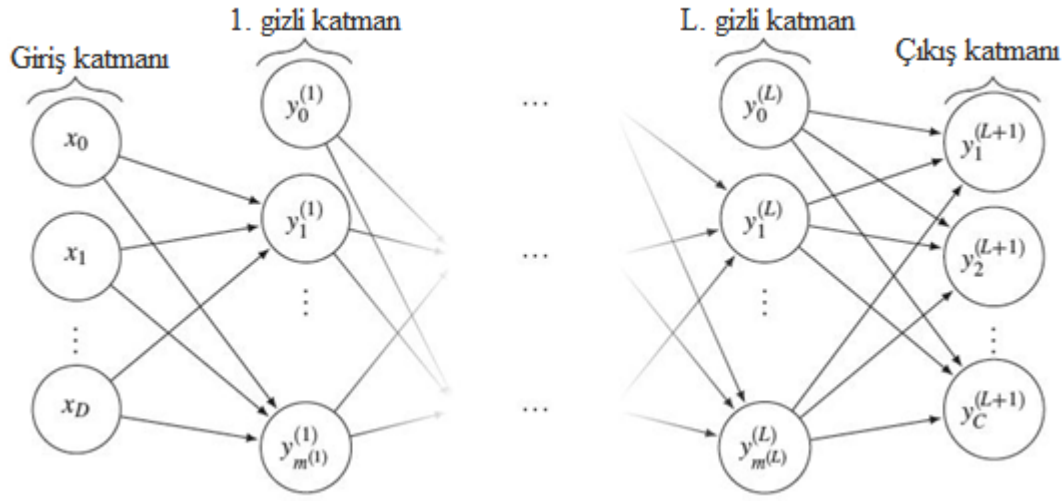
(2.2)'deki eşitlikteki $z^{(l)}$, $w^{(l)}$ ve $y^{(l-1)}$; sırasıyla $z_i^{(l)}$ gerçek girişleri, $y_k^{(l-1)}$ çıkışları, $w_{i,k}^{(l)}$ ağırlıkları göstermektedir.

Çok katmanlı algılayıcılar (2.3)'deki genel formda gösterilir:

$$y(.,w): \mathbb{R}^D \rightarrow \mathbb{R}^C, x \rightarrow y(x,w). \quad (2.3)$$

Çıkış vektörü olan $y(x,w)$, çıkış değerleri olan $y_i(x,w) := y_i^{(L+1)}$'leri içerir ve w, ağdaki bütün ağırlıkları gösteren vektördür.

Derin sinir ağlarında gizli katman sayısı 3'ten fazladır. Bu nedenle derin öğrenme yani derin sinir ağlarının eğitimi zor bir iş olarak düşünülmektedir [32].



Şekil 2.2. D tane giriş birimi, C tane çıkış birimi ve birinci gizli katmanda m(l) tane gizli birim içeren, (L+1) katmanlı algılayıcı [34]

2.2 Aktivasyon Fonksiyonları

Bu bölümde [34]'deki üç çeşit aktivasyon fonksiyonu incelenmiştir. Bunlar, basamak fonksiyonu, parça parça lineer fonksiyon ve sigmoid fonksiyondan oluşur.

a) Basamak fonksiyonu: Bu aktivasyon fonksiyonu Heaviside fonksiyonu olarak da bilinir ve (2.4)'deki gibi ifade edilir

$$h(z) = \begin{cases} 1 & z \geq \frac{1}{2} \\ z & \frac{1}{2} > z > -\frac{1}{2} \\ 0 & z \leq -\frac{1}{2} \end{cases}. \qquad (2.4)$$

b) Parça parça doğrusal fonksiyon: Bu aktivasyon fonksiyonu (2.8)'deki gibi ifade edilir.

c) Sigmoid fonksiyon: Bu aktivasyon fonksiyonu (2.5)'deki gibi ifade edilir.

Basamak fonksiyonu ve parça parça doğrusal fonksiyonun iki tane sakıncası vardır. Birincisi, doğrusal fonksiyonların türevlenemez olmasıdır. Ama ağın eğitiminde, türevlenebilir aktivasyon fonksiyonlarına ağın eğitiminde ihtiyaç duyulur. İkincisi, doğrusal olmayan aktivasyon fonksiyonları, doğrusal aktivasyon fonksiyonlarına göre daha fazla hesapsal güce sahiptir. En yaygın kullanılan aktivasyon fonksiyonu sigmoid fonksiyondur. Sigmoid fonksiyon (2.5) deki gibi verilir:

$$\sigma(z) = \frac{1}{1+\exp(-z)}. \quad (2.5)$$

Bu fonksiyonun grafiği s şeklindedir ve türevlenebilir. Hiperbolik tanjant tanh(z), lojistik sigmoid fonksiyonun [-1,1] aralığında doğrusal dönüşümü olarak düşünülebilir. İki aktivasyon fonksiyonu da doyumludur [35].

YSA'lar, sınıflandırma için kullanıldığında softmax aktivasyon fonksiyonu, çıkış birimleri için çıkış değerlerini sonraki ihtimaller olarak yorumlar. Sonra, çıkış katmanındaki i. çıkış birimi (2.6)'daki gibi verilir:

$$\sigma(z^{(L+1)},i) = \frac{\exp(z_i^{(L+1)})}{\sum_{k=1}^{C}\exp(z_k^{(L+1)})}. \quad (2.6)$$

[45]'de ki sonuçlar derin öğrenme uygulamalarında, hem lojistik sigmoid fonksiyonun hem de hiperbolik sigmoid fonksiyonun pek iyi sonuç vermediğini göstermektedir. Daha iyi sonuçlar softsign aktivasyon fonksiyonu kullanılarak alınmıştır:

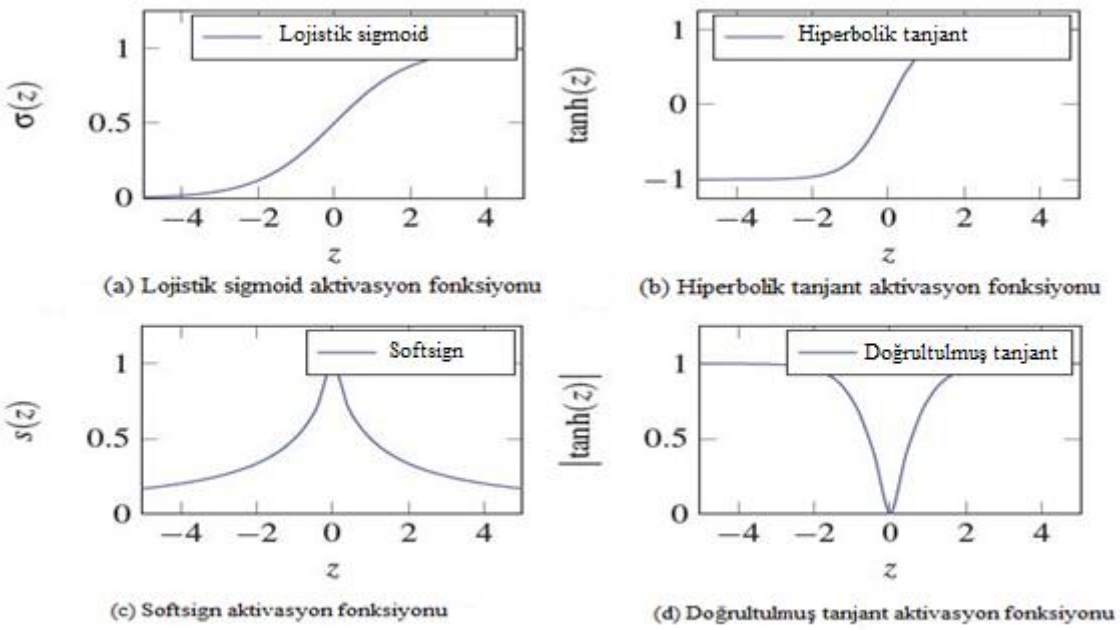
$$s(z) = \frac{1}{1+|z|}. \quad (2.7)$$

[46]' da (2.8)'deki doyumsuz aktivasyon fonksiyonu:

$$r(z) = \max(0, z) \quad (2.8)$$

kullanılmıştır. Bu aktivasyon fonksiyonunda kullanılan gizli birimler doğrultulmuş doğrusal birim olarak adlandırılmıştır.

Ayrıca, [36]'da hiperbolik tanjant aktivasyonuna ek olarak doğrultma daha iyi sonuçlar vermiştir. Yukarıda bahsedilen aktivasyon fonksiyonun bazıları Şekil 2.3'de gösterilmiştir.



Şekil 2.3. Yaygın olarak kullanılan aktivasyon fonksiyonları

2.3. Eğiticili Öğrenme

Eğiticili öğrenmede, giriş ve çıkış çiftlerinden oluşan eğitim bilgileri vardır. Ağ; giriş bilgisine göre ürettiği çıkış değerini, istenen değerle kıyaslayarak ağırlıkların değiştirilmesinde kullanılacak bilgiyi elde eder. Hata değeri, girilen değer ile istenen değer arasındaki farka verilen isimdir ve hata değeri önceden belirlenen değerden küçük oluncaya kadar eğitim işlemine devam edilir. Hata değeri istenen değerin altına düştüğünde, ağırlıklar sabitlenerek eğitim tamamlanır.

YSA'nın önemli olan özellik ve ilişkileri öğrenmesi isteniyorsa, eğitim kümesinin ihtiyaç duyulan bütün bilgileri içermesi gerekir. Ağ sadece bir örnek ile eğitilirse, bir olay için hassas olan ağırlıklar kümesi, farklı bir olayda yetersiz kalabilir. Sonuç olarak, YSA'lar gerekli ve yeterli bilgiler ile eğitilmelidir.

2.3.1. Öğrenme Oranı

YSA'ların öğrenme oranının seçimi önemlidir. Düşük bir öğrenme oranı ile YSA'ların eğitimi için için çok fazla zaman harcanır. Daha yüksek öğrenme oranıyla eğitilmiş bir ağ, daha yavaş eğitilen bir ağa göre daha düşük bir başarım gösterebilir. Birçok öğrenme

işleminde, ağırlıkların ayarlanma işleminde öğrenme sabiti veya öğrenme oranı kullanılmaktadır.

Öğrenme sabiti genel olarak sıfır ile bir arasında bir değer olmaktadır. Eğer birden büyük bir öğrenme sabiti seçilirse öğrenme algoritmasını ayarlamak kolay olmasına rağmen ağda salınımlar meydana gelir. Bu yüzden öğrenme oranı genellikle 0 ile 1 arasında seçilir. Küçük öğrenme sabiti, anlık hataları hızlı bir şekilde düzeltememesine karşın en iyi sonuca ulaşma şansını yükseltmektedir.

2.4. Eğiticisiz Öğrenme

Eğiticisiz öğrenmede, YSA'ların doğru çıkış hakkında bilgisi yoktur. Eğiticisiz olarak eğitilebilen ağlar, hedef çıkış olmadan giriş bilgilerinin özelliklerine göre ağırlık değerlerini ayarlar. Bu yöntemde nöronlar eğitilmek için seçilen durum ya da ölçütleri güncellemek içi yarışırlar. En büyük çıkış ile işlenen nöron, kazananı belirler ve komşularına bağlantı boyutlarını güncellemelerine izin verir.

2.5. Derin Öğrenme

Derin öğrenme, son yılların güncel çalışma konularından birisidir. Derin Öğrenme ilk olarak 2012 yılında nesne sınıflandırma için yapılan, ImageNet yarışmasında elde ettiği başarı ile adını duyurmuştur. Derin öğrenmenin kullanımının günümüzde artmasının iki temel sebebi vardır. Bunlardan birincisi, derin öğrenmede eğitim için gerekli olan verilerin artmasıdır. Günümüzde büyük veri kavramı gelişmiştir. Bunun sonucunda, bu veriler etiketlenerek derin öğrenmede eğitim için rahatlıkla kullanılabilmektedir. İkincisi ise donanımdaki gelişmelerdir. Çünkü, etiketlenen bu büyük veriler makul sürelerde işlenebilmelidir. GPU'ların yaygınlaşması ve güçlenmesi derin öğrenme uygulamalarının önünü açmaktadır [37-40].

Derin öğrenmenin en büyük getirilerinden birisi ise el ile öznitelik çıkarma işlemi yapılmamasıdır. Diğer yapay öğrenme yöntemlerinde uzman bir kişi tarafından öznitelik çıkarma işleminin yapılması gerekmektedir. Bu da zaman ve iş gücü kaybına neden olmaktadır [41,42].

Derin öğrenme, görüntü işleme (yüz tanıma, plaka tanıma, parmak izi tanıma, iris okuyucular, otonom araçlar), ses tanıma ve doğal dil işleme gibi birçok alanda kullanılmaktadır. Google, Baidu, Microsoft ve Facebook gibi firmalar derin öğrenme üzerine çalışmalar yapmaktadır.

Derin öğrenmede oluşturulan ağ, verilen çok sayıda girişten öznitelikleri kendisi öğrenir. Öznitelik öğrenme işleminin doğru bir şekilde gerçekleşmesi için ağ yeterince eğitilmelidir. Ağ yapısında alt katmanlar daha düşük öznitelikleri öğrenirken, üst katmanlar daha yüksek seviyeli öznitelikleri öğrenir [43,44].

2.6. Konvolüsyonel Sinir Ağları

KSA'ların amacı, görüntünün pikselleri arasındaki uzamsal bilgileri kullanmaktır. O yüzden, ayrık konvolüsyon dikkate alınır. Aşağıdaki alt bölümlerde, ayrık konvolüsyon ve KSA'ların temel bileşenleri tanımlanmıştır [36,37,45].

2.6.1. Konvolüsyon

Gri renk ölçekli bir görüntü (2.9)'daki gibi bir fonksiyon ile tanımlanırsa:

$$I:\{1,\ldots,n_1\}\times\{1,\ldots,n_2\}\rightarrow W\subseteq\mathbb{R},(i,j)\rightarrow I_{i,j} \tag{2.9}$$

böyle bir görüntü n1×n2 satır şeklinde gösterilebilir. K filtresi $K\in\mathbb{R}^{2h_1+1x_2h_2+1}$ şeklinde verildiğinde, I görüntüsünün K filtresi ile ayrık konvolüsyonu (2.10)'daki gibi hesaplanır

$$(I*K)_{r,s}\coloneqq\sum_{u=-h_1}^{h_1}\sum_{v=-h_2}^{h_2}K_{u,v}I_{r+u,s+v}. \tag{2.10}$$

Burada; K (2.11)'deki gibi ifade edilir

$$K=\begin{pmatrix}K_{-h1,-h2} & \cdots & K_{-h1,h2}\\ \vdots & \ddots & \vdots\\ K_{h1,-h2} & \cdots & K_{h1,h2}\end{pmatrix}. \tag{2.11}$$

Bu işlem için görüntünün sınırları düzgün bir şekilde belirlenmelidir. Örnek olarak, n1×n2 boyutunda gri renk ölçekli bir görüntüyü göz önüne alalım. (1,1) numaralı pikseline $2h_1+1\times h_2+1$ boyutunda rastgele bir filtre uygulandığında, (2.10) numaralı eşitliğin toplamı negatif piksel değerleri içerir. Bu problemi çözmek için çeşitli yaklaşımlar göz önüne alınabilir. Mesela, görüntü bazı yönlerde doldurulabilir veya filtre sadece bu işlemin düzgünce sonuç vereceği yerlere uygulanarak görüntüden daha küçük bir çıkış dizisi elde edilebilir.

Düzgünleştirme için yaygın olarak kullanılan ayrık Gaussian filtre (2.12)'deki gibi tanımlanır [46]:

$$(K_{G(\sigma)})_{r,s} = \frac{1}{\sqrt{2\pi}\sigma^2} \exp\left(\frac{r^2+s^2}{2\sigma^2}\right). \quad (2.12)$$

Burada; σ, Gaussian dağılımının standart sapmasıdır.

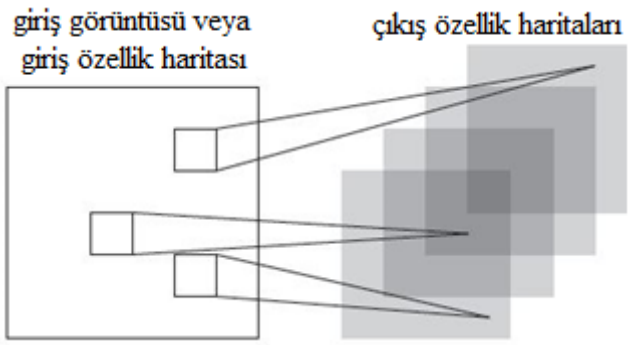
2.6.2. Konvolüsyon Katmanı

Konvolüsyonun temel amacı giriş görüntüsüne ait özniteliklerin belirlenmesidir. Konvolüsyon, giriş görüntüsünün piksellerini kullanarak görüntünün özniteliklerini çıkartır.

l, konvolüsyon katmanı olsun. *l* katmanının girişi, önceki katmanın öznitelik haritası olan ve her birinin boyutu $m_2^{(l-1)}$x $m_3^{(l-1)}$ olan $m_1^{(l-1)}$'den oluşur. *l*=1 olması durumunda, giriş bir veya daha fazla kanal içeren bir I görüntüsüdür. Bu bakımdan KSA'lar, ham görüntüyü giriş olarak kabul ederler. *l* katmanının çıkışı, $m_2^{(l-1)}$x $m_3^{(l-1)}$ boyutunda olan $m_1^{(l)}$ özellik haritasından oluşur. *l* katmanında ki i. öznitelik haritası, $Y_i^{(l)}$ olarak gösterilir ve (2.13)'deki gibi hesaplanır:

$$Yi^{(l)} = Bi^{(l)} + \sum_{j=1}^{m1^{(l-1)}} K_{i,j}{}^{(l)} * Yj^{(l-1)}. \quad (2.13)$$

l katmanı bir konvolüsyon katmanı ise, giriş görüntüsü (eğer *l*=1 ise) yada önceki katmanın öznitelik vektörü, farklı filtreler ile konvolüsyon edilerek, *l* katman çıkışının öznitelik haritasını oluşturur. Şekil 2.4'te tekli konvolüsyon katmanı gösterilmektedir.



Şekil 2.4. Tekli konvolüsyon katmanı [32]

$B_i^{(l)}$ bias matrisidir. $K_{i,j}^{(l)}$; l katmanındaki i. öznitelik haritası ile (l-1) katmanındaki j. öznitelik haritasının bağlanması ile oluşan ve boyutu $2h_1(l)+1$ x $2h_2(l)+1$ olan filtredir [45]. $m_2^{(l)}$ ve $m_3^{(l)}$'nin birimlerinden oluşan iki boyutlu $Y_i^{(l)}$ öznitelik haritası ve tek birim olan $y_i^{(l)}$ arasındaki fark, $Y_i^{(l)}$'nin çok katmanlı algılayıcı olarak kullanılmasıdır. Yukarıda bahsedildiği gibi, $m_2^{(l)}$ ve $m_3^{(l)}$, sınır etkilerinden etkilenirler. Çıkışta ki öznitelik haritalarının boyutları (2.14)'deki gibidir:

$$m_2^{(l)} = m_2^{(l-1)} - 2h_1^{(l)} \quad \text{ve} \quad m_3^{(l)} = m_3^{(l-1)} - 2h_2^{(l)}. \tag{2.14}$$

Konvolüsyonel katmanı ve onun işlemlerini anlatmak için, çok katmanlı algılayıcı olarak bilinen eşitlik (2.13) yeniden düzenlenir. (r,s) pozisyonundaki işlem birimi çıkışı (2.15)'deki gibi hesaplar:

$$(Y_i^{(l)})_{r,s} = (B_i^{(l)})_{r,s} + \sum_{j=1}^{m_1^{(l-1)}} (K_{i,j}^{(l)} * Y_j^{(l-1)})_{r,s}$$

$$= (B_i^{(l)})_{r,s} + \sum_{j=1}^{m_1^{(l-1)}} \sum_{u=-h_1^{(l)}}^{h_1^{(l)}} \sum_{v=-h_2^{(l)}}^{h_2^{(l)}} (K_{i,j}^{(l)})_{u,v} (Y_j^{(l-1)})_{r+u,s+v}. \tag{2.15}$$

Ağın eğitilebilir ağırlıkları, $K_{i,j}^{(l)}$ filtresinde ve $B_i^{(l)}$ bias matrisinde bulunabilir. Alt örnekleme, gürültü ve bozulmaların etkisini azaltmak için kullanılır. [36]'da anlatıldığı gibi; alt örnekleme, $s_1^{(l)}$ ve $s_2^{(l)}$ şeklinde atlama faktörü olarak adlandırılarak kullanılabilir. Temel fikir, filtreyi tekrar uygulamadan önce, dikey ve yatayda ki piksel sayılarını sabit tutmaktır. Yukarıda anlatılan atlama faktörü kullanılarak hesaplanan çıkış özellik haritasının boyutu (2.16)'daki gibidir:

$$m_2^{(l)} = \frac{m_2^{(l-1)} - 2h_1^{(l)}}{s_1^{(l)}+1} \quad \text{ve} \quad m_3^{(l)} = \frac{m_3^{(l-1)} - 2h_2^{(l)}}{s_2^{(l)}+1}. \tag{2.16}$$

2.6.3. Doğrultulmuş Doğrusal Birim

Eğer l katmanı doğrultulmuş doğrusal birim ise, bu katmanın girişi $m_1^{(l)}$ olarak verilen özellik haritasıdır. Bu katmanın çıkışı $m_1^{(l)}=m_1^{(l-1)}$ özellik vektörünü içerir ve her birinin boyutu $m_2^{(l-1)}xm_3^{(l-1)}$'dur. Ayrıca, $m_2^{(l)}=m_2^{(l-1)}$ ve $m_3^{(l)}=m_3^{(l-1)}$'dir:

$$Y_i^{(l)} = f(Y_i^{(l-1)}). \tag{2.17}$$

Bu denkleme [11]'de kazanç katsayısı eklenmiştir:

$$Y_i^{(l)} = g_i f(Y_i^{(l-1)}). \quad (2.18)$$

2.6.4. Doğrultma

l katmanı doğrultma katmanı olsun. O halde *l* katmanının girişi $m_1^{(l-1)}$ tane ve boyutları $m_2^{(l-1)}$x $m_3^{(l-1)}$ olan özellik haritalarından oluşur ve özellik haritasında ki her bileşen için kesin değer aşağıdaki gibi hesaplanır:

$$Y_i^{(l)} = |Y_i^{(l)}|. \quad (2.19)$$

(3.11) numaralı eşitlik 3.2.4 deki tam bağlı katmana kolay bir şekilde uygulanabilir.

2.6.5. Tam Bağlı Katman

Tam bağlı ifadesi önceki katmandaki tüm nöronların sonraki katmandaki tüm nöronlara bağlı olduğunu ifade etmektedir.

l tam bağlı katman olsun. Eğer (*l*-1)'de tam bağlı katman ise (2.20) numaralı eşitliği kullanabiliriz. Aksi halde, *l* katmanı $m_2^{(l-1)}$x $m_3^{(l-1)}$ boyutunda $m_1^{(l-1)}$ tane özellik haritasını giriş olarak bekler. *l* katmanındaki i. birim (2.20)'deki gibi hesaplama yapar:

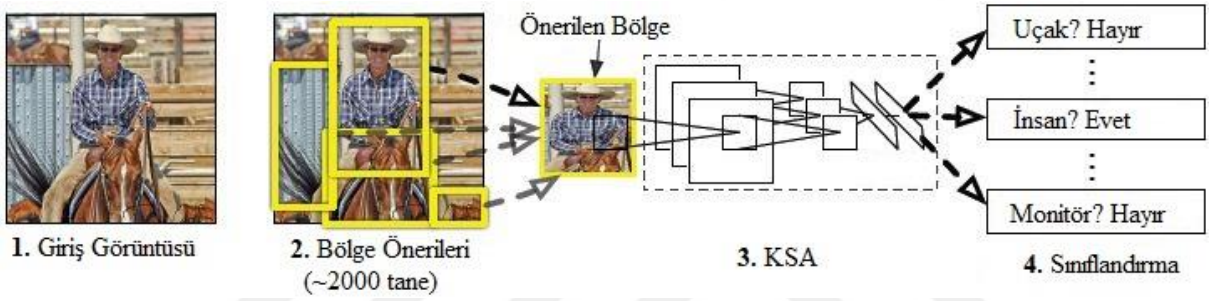
$$y_i^{(l)} = f(z_i^{(l)}) \quad \text{ile} \quad z_i^{(l)} = \sum_{j=1}^{m_1^{(l-1)}} \sum_{r=1}^{m_2^{(l-1)}} \sum_{s=1}^{m_3^{(l-1)}} w_{i,j,r,s}^{(l)} (Y_j^{(l-1)})_{r,s}. \quad (2.20)$$

2.6.6. Öznitelik Seçici Katman

Öznitelik seçici katman ile her bir filtrenin boyutu azaltılarak işlem hızı arttırılmış olur. Burada dikkat edilmesi gereken önemli özelliklere ait bilgilerin kaybolmaması için ağın başlarında öznitelik seçici katmanın dikkatli kullanılmasıdır. Farklı tiplerde öznitelik seçici katman olmakla birlikte, uygulamalarda en iyi sonucu en büyük öznitelik seçici katmanın verdiği görülmüştür.

2.7. Bölgesel Konvolüsyonel Sinir Ağları ve Çeşitleri

B-KSA, bir nesnenin görüntünün neresinde olduğunu algılamak ve sınırlarını belirlemek için 2014 yılında geliştirilmiş bir yöntemdir [47]. Bu yöntem temel olarak 4 bölümden oluşur. Yöntem ilk olarak giriş görüntüsünü alır. İkinci bölümde, giriş görüntüsü üzerinde nesne olma ihtimali olan yaklaşık 2000 bölgeyi seçer. Üçüncü bölümde, ikinci bölümde seçilen bu bölgeler sırasıyla KSA'nın girişine verilir. Dördüncü bölümde ise, KSA'nın çıkışı sınıflandırma işlemine tabi tutulur ve nesnenin sınıfına karar verilir. Bu yapı Şekil 2.5'de gösterilmiştir.

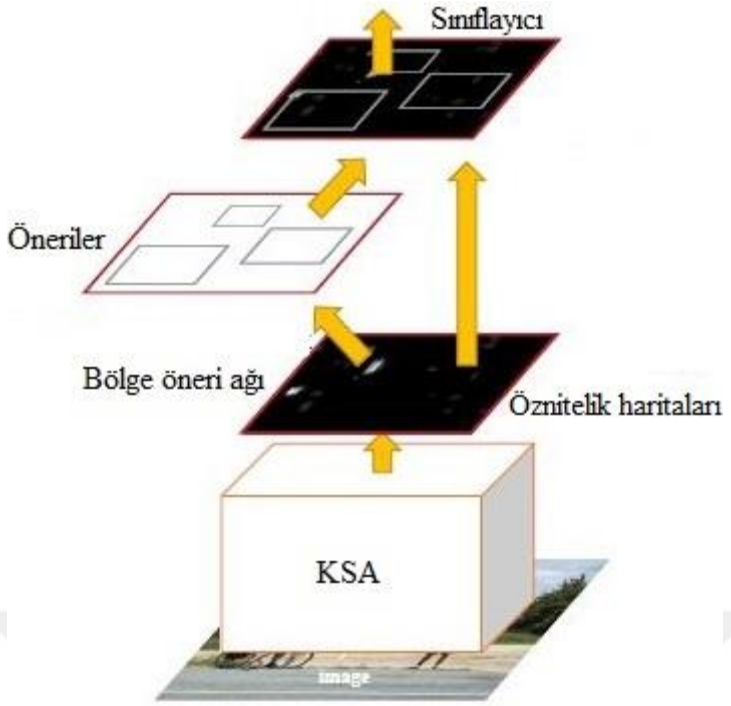


Şekil 2.5. B-KSA algılama sistemi [47]

B-KSA, giriş görüntüsünü ayırdığı 2000 bölgenin tamamını KSA'ya giriş olarak ayrı ayrı uygular. Bu durum zaman kaybına neden olmaktadır. Ayrıca, B-KSA üç farklı yapıyı eğitmek zorundadır. Bunlar; bölge önerisini sunan regresyon modeli, KSA ve sınıflandırma işlemini yapan sınıflayıcıdır. Bu üç yapının eğitimi işlem yükünü arttırmaktadır.

Hızlı B-KSA, KSA'ları kullanarak nesne algılama için 2015 yılında geliştirilmiştir [48]. Hızlı B-KSA, eğitim ve test hızını artırırken, aynı zamanda algılama doğruluğunu artırmak için B-KSA'ya göre çeşitli yenilikler kullanmaktadır. Hızlı B-KSA, her bir görüntü için yaklaşık 2000 defa KSA çalıştırmak yerine tek bir kez KSA çalıştırarak öneriler arasında bu hesaplamanın dağılımı yapılmaktadır. Bu nedenle Hızlı B-KSA'in, eğitim ve test süresi B-KSA'ya göre daha kısadır.

B-KSA'nın yukarıda anlatılan sakıncalarını gidermek için 2016 yılında Daha Hızlı B-KSA yöntemi geliştirilmiştir [49]. Bu yöntemin diğerlerinden farkı, bölge öneri ağı yapısını içermesidir. Bu yapı, Şekil 2.6'da gösterilmiştir.



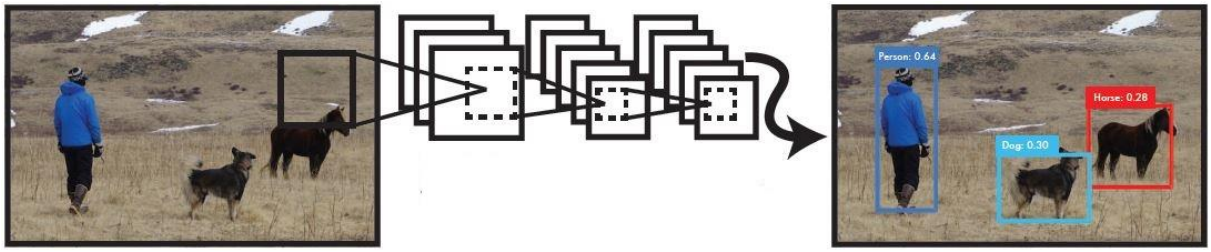
Şekil 2.6. Daha hızlı B-KSA algılama sistemi [49]

2.8. YOLO

Sadece Bir Defa Bak (YOLO - You Only Look Once), nesne algılama için kullanılan bir yaklaşımdır. Nesne algılama ile ilgili daha önce yapılan çalışmalar, algılamayı gerçekleştirmek için sınıflandırıcıları yeniden tasarlar. YOLO; bunun yerine nesne tespitini, sınırlayıcı kutular ve ilişkili sınıf olasılıkları için bir regresyon problemi olarak ele almaktadır. YOLO'da tek bir sinir ağı, bütün görüntüdeki nesneleri ve bu nesnelerin olasılık dağılımlarını sadece bir değerlendirme ile hesaplar. Tüm algılama, tek bir ağ üzerinden gerçekleştiği için, algılama performansı çok hızlıdır [50]. Bu nedenle gerçek zamanlı sistemlerde rahatlıkla kullanılabilir.

Temel YOLO modeli, görüntüleri gerçek zamanlı olarak saniyede 45 kare hızında işler. Ağın daha küçük bir versiyonu olan Hızlı YOLO ise saniyede 155 kare işleyebilmektedir. Son teknoloji ürünü algılama sistemleriyle karşılaştırıldığında; YOLO daha fazla yerelleştirme hatası yapar, ancak arka planda yanlış pozitifleri tahmin etme olasılığı daha azdır. YOLO, nesnelerin çok genel temsillerini öğrenir. Bu nedenle, YOLO'nun genelleştirme kabiliyeti yüksektir.

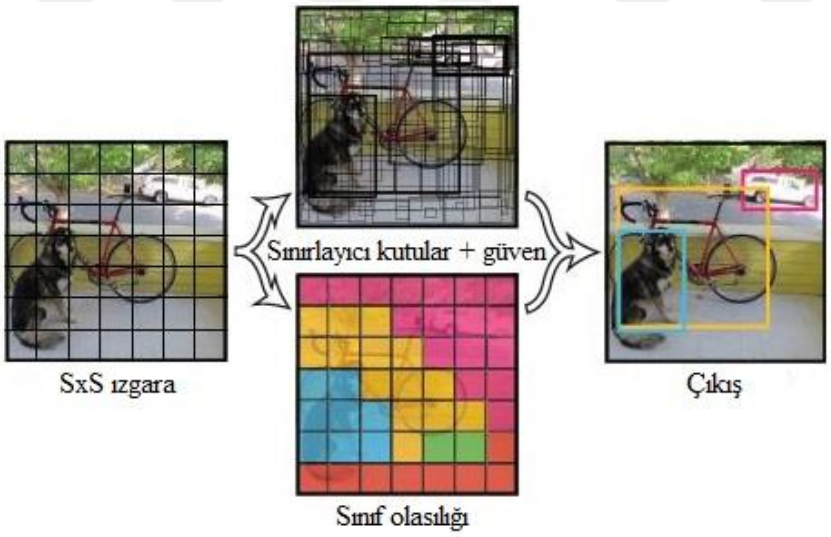
YOLO'nun algılama sistemi Şekil 2.7'de verilmiştir. Burada ilk olarak görüntü 448x448 boyutuna dönüştürülür. Daha sonra görüntü üzerinde bir tane KSA çalıştırılır. Son olarak, belirlenen eşik değerine göre bulunan algılamalar gösterilir.



Şekil 2.7. YOLO algılama sistemi [50]

YOLO'da giriş görüntüsünü S×S boyutunda ızgaraya bölünür. Sonra her bir ızgaraya B tane sınırlayıcı kutu uygulanır ve bu sınırlayıcı kutulara ait güven değerleri belirlenir. Bu güven değerleri, sınırlayıcı kutunun bir nesne içerip içermediğini ve sınırlayıcı kutunun tahmin ettiği nesnenin ne kadar doğru olduğunu gösterir. Eğer bir sınırlayıcı kutunun içinde nesne yok ise güven değeri 0 olarak atanır. Daha sonra, her bir ızgara için sınıf olasılığı belirlenir.

Son olarak, bulunan sınıflar üzerinden algılama tahmini yapılır. Şekil 2.8'de bu durum gösterilmiştir. 2016 yılında çıkarılan YOLO9000 modeli, 9000'den fazla nesneyi algılayabilmektedir [51].



Şekil 2.8. YOLO modeli [50]

3. CPU-GPU'LAR VE RAKAM TANIMA UYGULAMASI

YSA'lar, bilgisayar görmesinden konuşma tanımaya kadar birçok uygulamada kullanılmaktadır. YSA'lar, geleneksel hesaplama yöntemlerinden daha üstün sonuçlar vermektedir. Ancak veri sayısının büyük olduğu uygulamalarda, çok fazla hesaplama gerektirmektedir. Günümüzde veri kümelerinin büyümesi ile Merkezi İşlem Birimleri kısaca CPU'lar üzerinde yapılan hesaplamalar yetersiz kalarak yerini verileri paralel işleyen Grafik İşleme Birimlerine kısaca GPU'lara bırakmıştır. Bu nedenle verileri paralel işleyerek sistemlere hız kazandıran GPU'ların kullanımını zorunlu hale gelmiştir. Özellikle, çok sayıda eğitim verisinden öznitelik çıkarma işlemi yapabilen sistemler oluşturmak için ileri teknoloji çalışmalarında, derin ileri beslemeli YSA'ların kullanımı artmıştır. Son yıllarda, YSA'ların bir çeşidi olan derin öğrenme algoritmaları GPU'lar sayesinde, gerçek hayattaki birçok uygulamada başarılı olarak uygulanmış ve GPU içeren gömülü sistemlere talep artmıştır.

Tezin bu bölümünde, derin öğrenme yöntemlerinden birisi olan KSA'lar kullanılarak rakam tanıma uygulaması yapılmış ve düşük maliyetli Nvidia Jetson TK1/TX1 gömülü sistemleri genel olarak incelenmiştir. Derin öğrenme algoritmalarını uygulamak için Kaliforniya Üniversitesi, Berkeley tarafından geliştirilen Caffe programı kullanılmıştır. Son olarak paralel işleme gücüne sahip olan Nvidia Jetson TK1/TX1 kartları üzerinde MNIST veri kümesi kullanılarak LeNet ağının eğitimi gerçekleştirilmiştir. Eğitim için kartların hem CPU'ları hem de GPU'ları kullanılmıştır. Deneysel sonuçlar için, Nvidia Jetson TK1/TX1 kartlarına ek olarak Nvidia GTX550 ve GTX960 ekran kartlarına sahip iki bilgisayar da kullanılmıştır. Sonuçlar, hız ve doğruluk açısından değerlendirilmiştir.

3.1. Nvidia Jetson Geliştirici Kitleri

Günümüz endüstrisinde, işlerin büyük çoğunluğu robot benzeri sistemler tarafından yapılmaktadır. Bu sistemlerde, görüntü işleme temel görevlerden biridir. Bu yüzden GPU'ların kullanımı artmıştır. Ancak, Genel Amaçlı GPU (GPGPU - General Purpose GPU)'lar hem yüksek enerji tüketimine sahiptir hemde gömülü sistemler için fazla büyüktür.

Jetson TK1/TX1, Nvidia tarafından bilgisayar görmesi, otomotiv, tıp ve robotik uygulamalarında kullanılmak için geliştirilen kartlardır. Nvidia Jetson TK1/TX1 geliştirme kartları, tümleşik sistem uygulamalarında, GPU'nun gücünden ve getirdiği kazançlardan yararlanmak için gereksinim duyulan her şeyi sunmaktadır. Bilgisayarla görme ve derin

öğrenme uygulamalarına yönelik yüksek performanslı ve düşük güç kullanan, yüksek hesaplama kapasitesi sayesinde gerçek zamanlı yapay zeka uygulamalarında kullanılabilir.

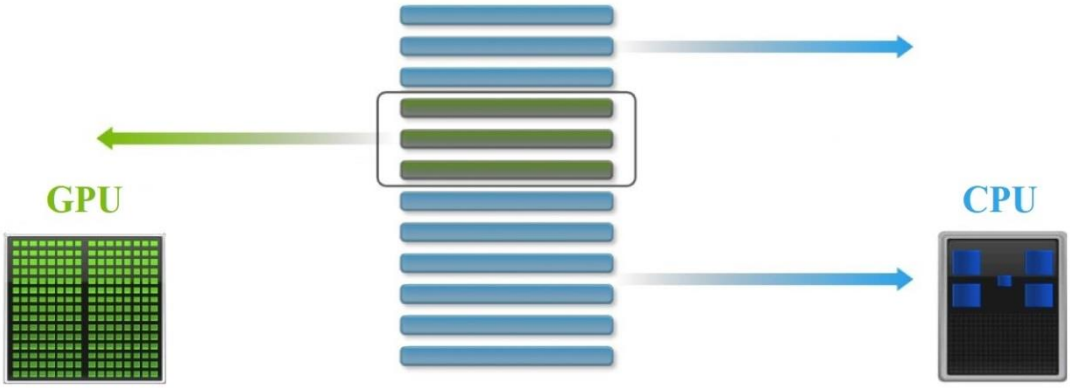
3.1.1. Nvidia Jetson TK1

Jetson TK1, 2014 yılının mayıs ayında Nvidia tarafından kullanıma sunulmuştur. Bu kart, hibrit işlemci olan Tegra K1 tabanlıdır ve Nvidia Kepler mimarisine sahiptir. Nvidia Jetson TK1, batarya koruma çekirdeği, dört çekirdekli ARM Cortex A15 CPU ve 192 CUDA çekirdeğine sahiptir. Kart üzerindeki bağlantılar aşağıdaki gibidir:

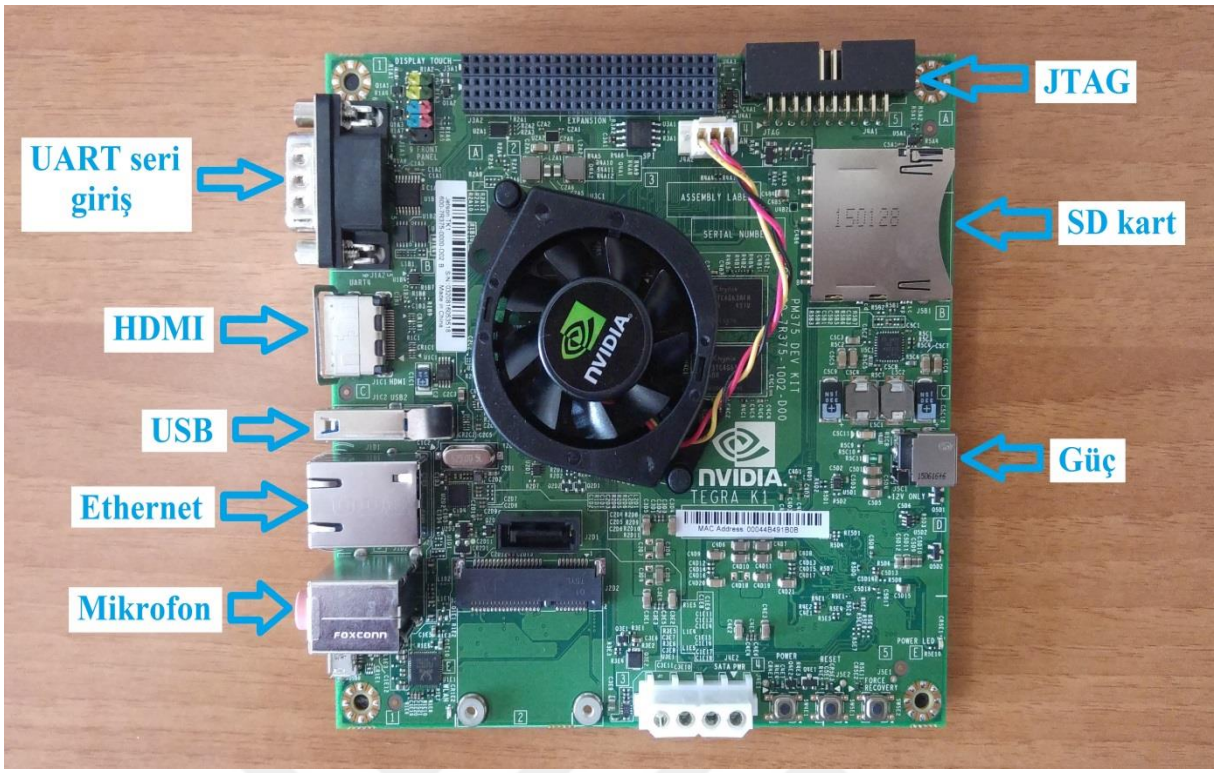
- USB 3.0 girişi,
- HDMI girişi,
- Mini-PCIe yuva,
- SD/MMC bağlayıcı,
- RS232 seri bağlantı,
- GigE LAN,
- SATA girişi,
- SPI 4 MB boot flash.

Şekil 3.2'de Nvidia Jetson TK1 ve bağlantıları verilmiştir.

GPU hesaplama, CPU ve GPU'nun heterojen bir hesaplamasıdır. Programcı, GPU'nun paralel işlem gücü ile CPU'nun tek satır işlem performansı ve düşük gecikme süresi arasında bir seçim yapar. Yani uygulama, seri hesaplama ve paralel hesaplama olarak ikiye ayrılır. Bu heterojen hesaplama Şekil 3.1'de gösterilmiştir.



Şekil 3.1. Heterojen hesaplama [64]



Şekil 3.2. Nvidia Jetson TK1 ve bağlantıları

CUDA genel amaçlı paralel hesaplama platformu ve programlama modelidir. CUDA ile NVIDIA GPU'lardaki paralel hesaplama motoru birçok karmaşık hesaplama problemini CPU'dan daha verimli bir şekilde çözebilir.

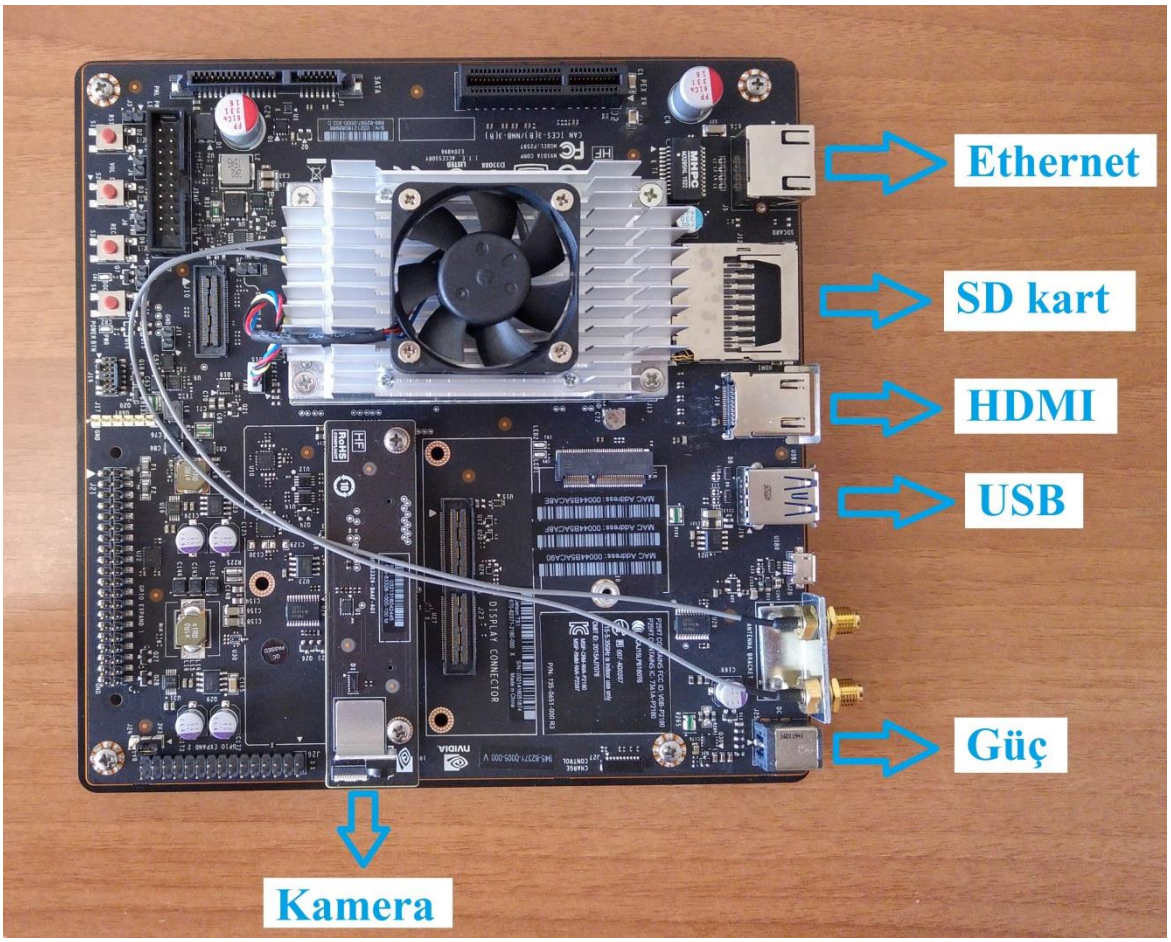
3.1.2. Nvidia Jetson TX1

Jetson TX1, bir modül üzerindeki ilk süper bilgisayardır. Görsel hesaplama uygulamaları için performans ve güç verimliliği sağlamaktadır. Nvidia Maxwell mimarisine sahiptir ve 256 CUDA çekirdeği vardır. 64-bit CPU'lar sayesinde 4K video kodlama ve kod çözme işlemleri yapılabilmektedir. Üzerindeki 1400 MPix/s kapasiteli kamera ve yüksek performanslı GPU'lar sayesinde görüntü işleme içeren derin öğrenme uygulamalarını gömülü olarak yapmaya izin verir. Kart üzerindeki bağlantılar aşağıdaki gibidir:

- USB 3.0 girişi,
- USB 2.0 Micro AB (kurtarma ve ana bilgisayar modunu destekler) girişi,
- HDMI girişi,
- M.2 Key E,
- PCI-E x4,

- Gigabit Ethernet,
- Tam Boyut SD,
- SATA Verileri ve Güç,
- GPIOs, I2C, I2S, SPI,
- TTL UART ve Akış Kontrolü,
- Ekran Genişletme Başlığı,
- Kamera Genişletme Başlığı.

Şekil 3.3'te Nvidia Jetson TX1 ve bağlantıları verilmiştir.



Şekil 3.3. Nvidia Jetson TX1 ve bağlantıları

Gömülü geliştirme kitleri arasında, Jetson TX1 geliştirici kiti dört açıdan öne çıkmaktadır. İlk olarak, Jetson TX1 kartındaki uygulamaların geliştirilmesi, tipik bir gömülü geliştirme kartı üzerinde geliştirmekten ziyade bir bilgisayar üzerinde geliştirmeye benzemektedir. İkinci olarak Jetson JetPack yükleyici, kartta özel olarak yapılandırılmış veya güncellenmiş sistem

görüntüsünü yüklemeyi kolaylaştırır. Üçüncü olarak, CUDA için kapsamlı bir yazılım paketi seti (cuDNN, OpenCV4Tegra ve VisionWorks) sunmaktadır. Böylece, geliştiricilerin GPU programlamanın karmaşıklıklarını incelemek zorunda kalmadan uygulamalarını hızlandırmak için GPU hesaplamasını sorunsuz bir şekilde kullanabilmelerini sağlar. Son olarak, bilgisayar görüsü ve derin öğrenim uygulamaları üzerindeki Jetson TX1 performansı etkileyicidir [47]. OpenCV (OpenCV4Tegra aracılığıyla) ve Caffe (cuDNN aracılığıyla) GPU hızlandırılmış sürümlerini kullanabilmektedir. Böylece kodun kendisini eniyilemeden gerçek zamanlı başarım elde etmek mümkün hale gelmektedir.

3.2. Caffe

Caffe, derin öğrenme algoritmalarını uygulamak için Kaliforniya Üniversitesi, Berkeley tarafından geliştirilen açık kaynaklı bir platformdur. Kodlar, GPU hesaplama için kullanılan CUDA kütüphanesi ile C++'da yazılır [56-62]. CUDA genel amaçlı paralel hesaplama platformu ve programlama modelidir. CUDA ile NVIDIA GPU'larda ki paralel hesaplama motoru birçok karmaşık hesaplama problemini CPU'dan daha verimli bir şekilde çözebilir. CUDA paralel programlama modeli, paralelliğini, üç boyutlu grafik uygulamaları gibi artan işlemci çekirdeklerine kadar ölçeklendirmek için tasarlanmıştır. Geliştiriciler CUDA ortamında yüksek seviyeli programlama dili olarak C ve C++ kullanabilirler. Programcılar tarafından C ve C++, düşük öğrenme eğrisini sürdürmek için kullanılır [57]. Ayrıca CUDA kütüphanesi, Python/Numpy ve MATLAB'ı destekler. Hızlı CUDA kodu ve GPU'nun hesaplama gücü sayesinde, K40 veya Titan GPU kullanılarak her gün 40 milyonun üzerinde görüntü işlenebilmektedir. Bu da yaklaşık olarak her görüntü için 2.5 ms zaman ayrıldığı anlamına gelmektedir [55].

Caffe, ilk olarak görüntü işleme için tasarlanmasına rağmen daha sonradan ses tanıma, robotik, astronomi ve sinir bilimi için kullanılmıştır. Caffe'nin, bu hızlı yayılımını sürdürmesiyle farklı bilim alanlarında ve endüstride yaygın olarak kullanılarak derin öğrenmenin getirilerinden faydalanılması beklenmektedir.

3.3. LeNet Ağı

LeNet, konvolüsyonel ağların başarılı olarak uygulandığı ilk ağ yapısıdır. LeNet, 1989 yılında Yann LeCun tarafından geliştirilmiştir [63]. LeNet mimarisi başka uygulamalar için

kullanılmakla birlikte, daha çok rakamlar için kullanılmaktadır. Şekil 3.6'da LeNet ağının örnek bir gösterimi verilmiştir.

LeNet ağlarının ilki olan LeNet 1, 28×28 boyutunda giriş katmanına sahiptir ve ağın MNIST verileri için hata oranı %1.7 olmuştur. Daha sonra geliştirilen LeNet 4 ağ yapısında, LeNet 1'e göre daha fazla özellik haritası ve gizli katman bulunmaktadır. LeNet 4'ün giriş boyutu 32×32'dir ve ağın MNIST verileri için hata oranı %1.1'e düşmüştür. Son olarak geliştirilen LeNet 5 ise LeNet 4 ağ yapısına benzemektedir. Ancak LeNet 5'te, daha fazla özellik haritası ve daha büyük tam bağlı katmanlar bulunmaktadır. Ağın MNIST verileri için hata oranı %0.9 olmuştur [54].

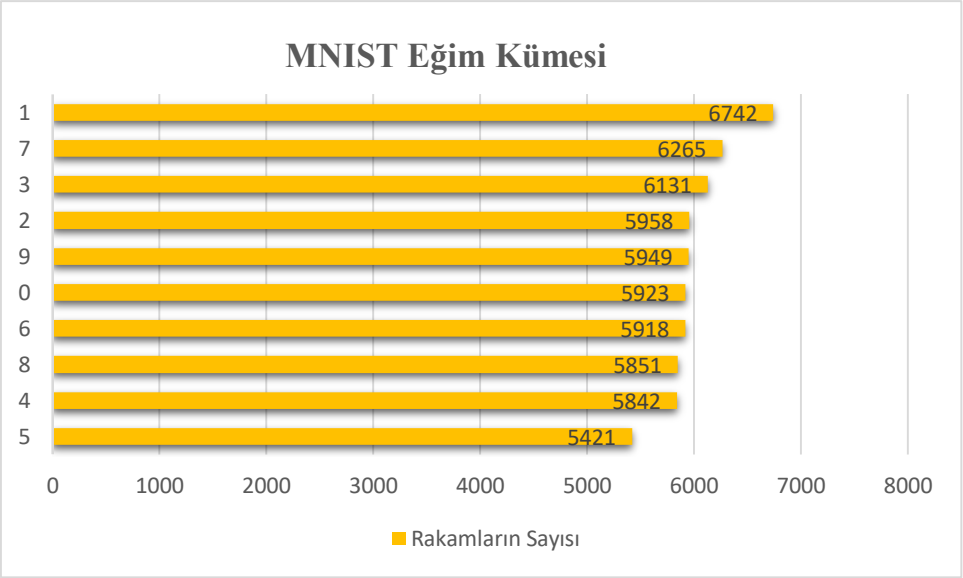
3.4. MNIST Veri Kümesi

MNIST veri kümesi LeCun, Cortes ve Burges tarafından el yazısı rakamları sınıflandırma problemi üzerine yapay öğrenme modellerini değerlendirmek için geliştirilmiş bir veri kümesidir [60]. MNIST veri kümesi araştırmalarda ve yeni geliştirilen rakam tanıma uygulamalarında yaygın bir şekilde kullanılmaktadır. MNIST veri kümesi 60000 eğitim görüntüsü ve 10000 test görüntüsü içermektedir [58]. 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 el yazısı rakamlarından oluşan MNIST veri kümesinden örnek bir görüntü Şekil 3.4'te gösterilmiştir.

Şekil 3.4. MNIST veri kümesinden bir örnek [59]

El yazısı rakamlar, 28x28 boyutuna indirgenmiş ve görüntünün ortasında yerleştirilmiştir. Her görüntü gri tonlamalı (0-255) olan 784 pikselden oluşmaktadır. MNIST veri kümesi 10

sınıftan oluşmaktadır. Bu sınıflardaki örnek sayısı Şekil 3.5'te ve yüzdelik oranları Tablo 3.1'de verilmiştir.



Şekil 3.5. MNIST eğitim kümesindeki rakamların sayıları [60]

Tablo 3.1. MNIST eğitim verilerinin yüzdelik olarak dağılımları [60]

Sınıf	Veri Sayısı	Yüzde (%)
1	6742	11,237
7	6265	10,442
3	6131	10,218
2	5958	9,93
9	5949	9,915
0	5923	9,872
6	5918	9,863
8	5851	9,752
4	5842	9,737
5	5421	9,035

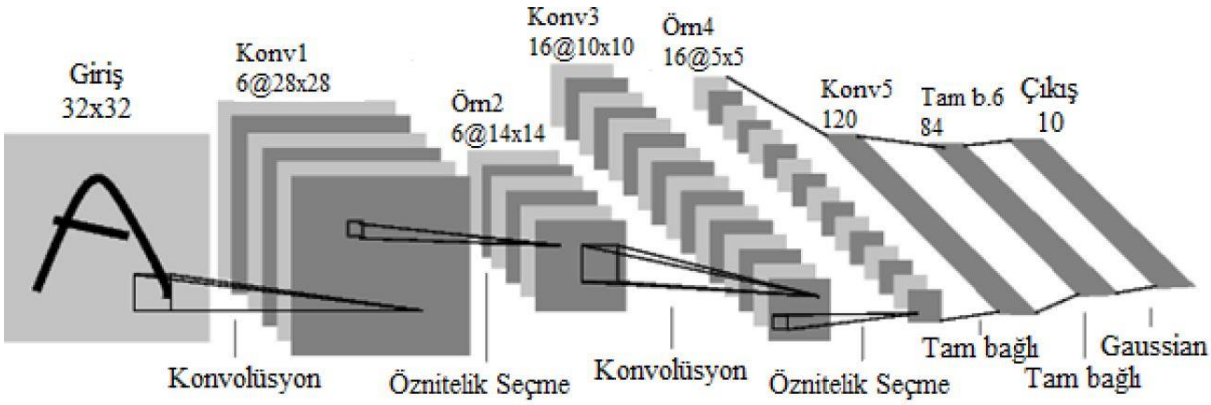
3.5. Rakam Tanıma Uygulaması ve Sonuçlar

Bu bölümde, derin öğrenme algoritmalarını uygulamak için Kaliforniya Üniversitesi, Berkeley tarafından geliştirilen Caffe programı kullanılmıştır. Paralel işlem gücüne sahip olan Nvidia Jetson TK1/TX1 kartları üzerinde MNIST verileri kullanılarak LeNet ağının eğitimi gerçekleştirilmiştir. Eğitim için kartların hem CPU'ları hem de GPU'ları kullanılmıştır. Deneysel sonuçlar için, Nvidia Jetson TK1/TX1 kartlarına ek olarak Nvidia GTX550 ve GTX960 ekran kartlarına sahip iki bilgisayar da kullanılmıştır. Sonuçlar, hız ve doğruluk açısından değerlendirilmiştir.

Caffe'de, MNIST veri kümesi ile LeNet ağı eğitilirken Tablo 3.2'deki parametreler kullanılmıştır. MNIST verileri için özelleştirilmiş ağ yapısı Şekil 3.6'da verilmiştir. MNIST veri kümesinde 10 tane sınıf olduğu için ağ çıkışı da 10 tanedir. Bu çalışmada kullanılan ağ parametrelerinin değerleri Tablo 3.2'de verilmiştir.

Tablo 3.2. Kullanılan ağ parametreleri

Parametre	**Değeri**
Temel öğrenme oranı (Base learning rate)	0.01
Momentum	0.9
Ağırlık gecikmesi (Weight decay)	0.0005
Yığın boyutu (Batch size)	64
Gama (Gamma)	0.0001
Güç (Power)	0.75
En büyük adım sayısı (Maximum iteration)	10000
Test işleminin kaç adımda bir gerçekleştirileceği	500



Şekil 3.6. Kullanılan ağ yapısı [59]

Tablo 3.3'te elde edilen sonuçlar gösterilmiştir. Bu sonuçlara göre; ağ eğitimini en hızlı, Nvidia GTX960 içeren dizüstü bilgisayarın GPU işlemcisi yapmıştır. En yavaş ağ eğitimini ise, Nvidia Jetson TK1'in CPU işlemcisi yapmıştır. Diğer taraftan, en yüksek doğruluk oranı Jetson TK1 ile elde edilmiştir.

Tablo 3.3. Deneysel sonuçlar

Cihaz	İşlemci Türü	Süre (dakika)	Doğruluk (%)
Nvidia Jetson TK1	CPU	65	99.20
	GPU	13	99.07
Nvidia Jetson TX1	CPU	24	99.08
	GPU	3	98.98
Nvidia GTX 550 Masaüstü Bilgisayar	CPU	14	99.02
	GPU	5	99.08
Nvidia GTX960 Dizüstü Bilgisayar	CPU	12	99.02
	GPU	1	99.14

Nvidia Jetson TX1 ve TK1 birbiriyle karşılaştırtıldığında, GPU ve CPU kullanıldığı zaman, TX1'in TK1'den sırasıyla 4.33 ve 2.7 kat daha hızlı olduğu görülmüştür. Doğruluk oranlarının karşılaştırılmasında ise Nvidia Jetson TK1'in biraz daha yüksek doğruluk oranlarına sahip olduğu görülmüştür. Genel anlamda, GPU'ların CPU'lara göre doğruluk oranlarının yakın olmasına karşın daha hızlı olduğunu görülmüştür. Sonuç olarak, grafik işleme kartları kullanılarak elde edilen hız ve doğruluk değerleri, bu gömülü sistemlerin gerçek zamanlı uygulamalarda başarıyla kullanılabileceğini kanıtlanmıştır.

Algoritmalar, 10 defa çalıştırılarak ortalama sonuç verilmiştir. CPU ve GPU modlarında hızların farklı olması beklenmesine rağmen kararlık acısından da farklılık görülmüştür. Bunun olası 4 nedeni aşağıda belirtilmiştir.

1- Eğitim algoritmasının rastgele ağırlıklar ile başlatılması,

2- Seçilen CPU/GPU modellerine bağlı olarak, farklı noktalı sayılı (floating point) sonuçlar ortaya çıkması,

3- Jetson TX1 ve TK1'deki Jetpack yazılımların ve hatta caffenin kullandığı yazılımların sürümlerinin farklı olması,

4- Kullanılan hızlandırılmış eniyileme algoritmalarının farklı sürümlerde olması.

4. NESNE ALGILAMA UYGULAMALARI

Trafik işaretlerinin iki temel rolü vardır. Birincisi, trafiği düzenler. İkincisi, yolun durumu hakkında sürücü ve yayalara rehberlik eder. Trafik işareti tanıma son zamanlarda büyük ilgi görmektedir. Trafik işaretlerinin algılanması ve sınıflandırılması sürücü asistan sistemlerinin ve tamamen otonom araçların geliştirilmesi için önemli bir konudur.

Bu bölümde, kendimizin oluşturduğu yaya geçidi trafik tabelası veri kümesini kullanarak B-KSA ve Daha Hızlı B-KSA yöntemlerini kullanarak yaya geçidi trafik tabelası algıma uygulaması gerçekleştirilmiştir ve sonuçlar değerlendirilmiştir.

4.1. Yaya Geçidi Trafik Tabelası Algılama Uygulaması

4.1.1. Yaya Geçidi Trafik Tabelası Veri Kümesi

Yaya geçidi trafik tabelası veri kümesinde toplamda 172 adet görüntüde 185 tane yaya geçidi trafik tabelası bulunmaktadır. Yaya geçidi trafik tabelası veri kümesi için Elazığ, Niğde ve Kayseri illerinde LG G4 telefonunun 16 megapiksellik kamerası ile fotoğraflar çekilmiştir. Bu fotoğraflar MATLAB ile 1200x800 piksel boyutuna normalize edilmiştir. Görüntülerdeki tabelalar etiketlenmiştir ve sonrasında görüntüler eğitim kümesi ve test kümesi olarak ikiye ayrılmıştır. Eğitim kümesinde 95 tane görüntü bulunurken, test kümesinde 77 tane görüntü bulunmaktadır. Şekil 4.1, Şekil 4.2 ve Şekil 4.3'te eğitim kümesine, Şekil 4.4, Şekil 4.5 ve Şekil 4.6'da test kümesine ait görüntüler bulunmaktadır.

Şekil 4.1. Eğitim kümesinden bir görüntü

Şekil 4.2. Eğitim kümesinden bir görüntü

Şekil 4.3. Eğitim kümesinden bir görüntü

Şekil 4.4. Test kümesinden bir görüntü

Şekil 4.5. Test kümesinden bir görüntü

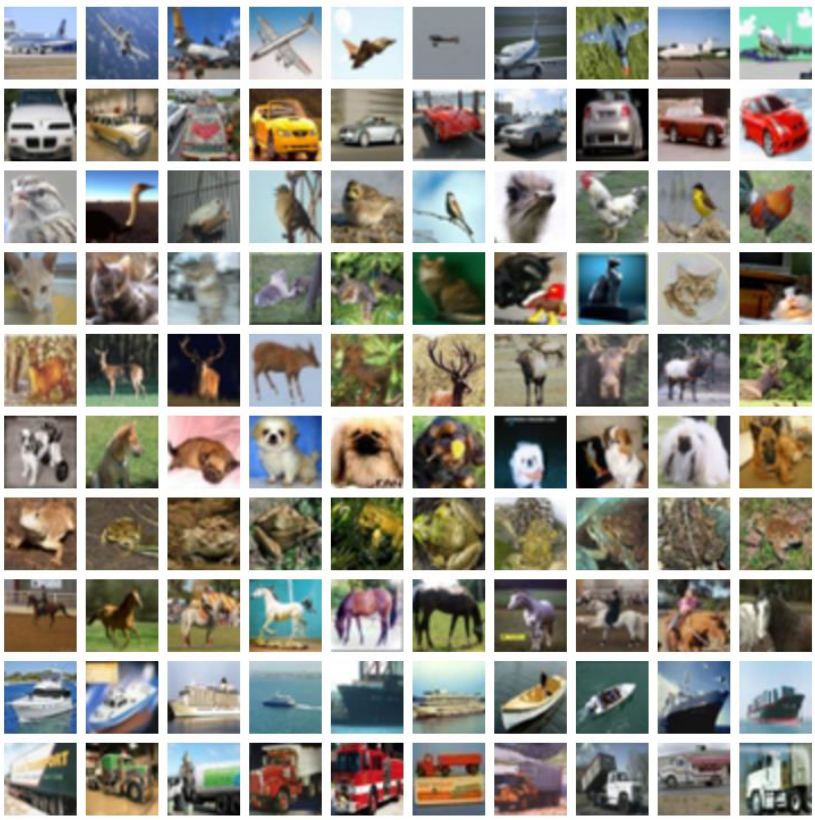
Şekil 4.6. Test kümesinden bir görüntü

4.1.2. Bölgesel Konvolüsyonel Sinir Ağları ile Yaya Geçidi Trafik Tabelası Algılama

Bu uygulamada, CIFAR-10 veri kümesi kullanılarak önceden eğitilmiş bir ağın üzerinde değişiklik yapılarak, B-KSA ile yaya geçidi trafik tabelası algılayıcı olarak tasarlanmış ve sonuçlar değerlendirilmiştir.

4.1.2.1. CIFAR-10 Veri Kümesi

CIFAR-10 veri kümesi Alex Krizhevsky, Vinod Nair ve Geoffrey Hinton tarafından oluşturulmuştur. CIFAR-10 veri kümesi, 10 sınıfa ait 60000 tane 32×32 boyutunda renkli görüntü içerir. Bu görüntülerin 50000 tanesi eğitim kümesini oluştururken, kalan 10000 tanesi test kümesindedir [65,66]. Bu veri seti her biri 10000 görüntü içeren 5 eğitim bölümüne ve 1 test kümesine ayrılmıştır. Test kümesi her sınıftan rastgele 1000 görüntü içerir. Eğitim kümesi her sınıfa ait 5000 görüntü içerir. Ancak, eğitim kümesi 5 bölüme ayrıldığı için bölümler içinde bir sınıfa ait görüntü sayısı diğerinden biraz fazla olabilir. Şekil 4.7'de CIFAR-10 veri kümesinin uçak, otomobil, kuş, kedi, geyik, köpek, kurbağa, at, gemi ve kamyon sınıflarına ait örnek görüntüler bulunmaktadır.



Şekil 4.7. CIFAR-10 veri kümesine ait görüntüler [65]

4.1.2.2. Yaya Geçidi Trafik Tabelası Algılama

Bu uygulama için Windows ortamında MATLAB kullanılmıştır. Eğitim için CIFAR-10 veri kümesinin 32×32 boyutunda 50000 tane renkli görüntüsü kullanılmıştır. KSA'yı oluşturmak için şu katmanlar tanımlanmıştır: giriş katmanı, konvolüsyon katmanı, Doğrultulmuş Doğrusal Birim (Rectified Linear Unit - ReLU) katmanı, öznitelik seçici katman, tam bağlı katman, softmax katmanı ve ağ çıkışı için sınıflayıcı katman.

Öncelikle 32×32×3 boyutundaki CIFAR-10 eğitim kümesinin görüntüleri için giriş katmanı oluşturulmuştur. Daha sonra ağın orta katmanı oluşturulmuştur. Orta katman; konvolüsyon, DDB ve öznitelik seçici katmanlarının tekrarlanmasından oluşturulmuştur. Bu 3 katman konvolüsyonel sinir ağlarının temelini oluşturmaktadır. Konvolüsyonel katman, ağın eğitimi sırasında güncellenen filtrelerin ağırlıklarını tanımlar. DDB katmanı, ağı lineer olmaktan kurtarır. Öznitelik seçici katmanı, verileri alt örneklemeye yarar. Büyük ağlarda, verileri ağın başında kaybetmemek için öznitelik seçici katmanı ilk başlarda dikkatli kullanılmalıdır.

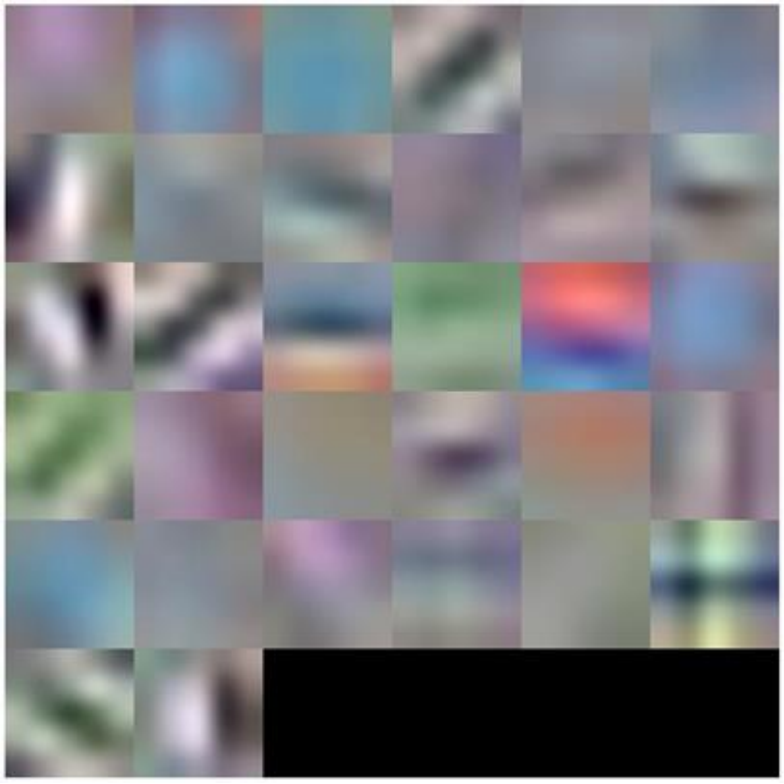
Birinci konvolüsyon katmanı 5×5×3 boyutunda 32 tane filtreden oluşur. Doldurma değeri 2 olarak seçilmiştir. Doldurma, sınırdaki bilgilerin henüz ağın başlarındayken kaybolmasını önlemektedir. Filtrenin 3. boyutu, ağın girişinin 3. boyutu ile aynı olmalıdır ve ikisi de 3'tür. Konvolüsyon, DDB ve öznitelik seçici katmanları tekrarlanarak daha derin ağlar oluşturulabilir. Erkenden alt örneklemeden kaçınmak için öznitelik seçici katmanının sayısı azaltılabilir. Ağın başlarında yapılan alt örnekleme, öğrenme için kullanışlı olan bilgilerin kaybolmasına neden olabilir.

Ağın son bölümü tipik olarak tam bağlı katman ve softmax katmanından oluşur. Bu noktada ağın 10 tane çıkışı bulunmaktadır. Son olarak sınıflandırma katmanını ekleriz. Sınıflandırma katmanı, tam bağlı katmanın çıkışını kullanarak görüntü sınıfları üzerinden kategorik olasılık dağılımını hesaplar. Eğitim yapılırken, ağın bütün ağırlıkları bu kategorik dağılım üzerindeki kayıpları en aza indirmek için ayarlanır. Son olarak oluşturulan giriş katmanı, orta katman ve son katman birleştirilir. Standart sapması 0.0001 olan ve normal dağılım gösteren rastgele sayılar kullanılarak ilk konvolüsyon katmanın ağırlıkları başlatılır. Bu eğitimin daha çabuk yakınsamasına yardımcı olur.

Ağ mimarisi oluşturulduktan sonra, ağın CIFAR-10 veri kümesi ile eğitilmesi gerekmektedir. İlk olarak, |trainingOptions| fonksiyonunun kullanarak ağın eğitim algoritması ayarlanmalıdır. Eğitim algoritması olarak, Momentum olasılıksal gradyan iniş yöntemi kullanılmıştır. Başlangıçta öğrenme oranı 0.001'dir. Eğitim sırasında her 8 özyinelemede bir,

başlangıçtaki öğrenme oranı azaltılır. 1 özyineleme, tüm eğitim verilerinin bir sefer ağdan geçmesine denir. Eğitim algoritması 40 özyineleme için çalıştırılmıştır. Eğitim algoritması en küçük yığın boyutu olarak 128 görüntü kullanılmıştır. Eğer eğitim için GPU kullanılıyorsa, GPU'nun kısıtlı hafızasından dolayı bu en küçük yığın boyutu değerinin düşürülmesi gerekebilir.

Birinci konvolüsyon tabakasının filtre ağırlıklarının hızlı bir şekilde görselleştirilmesi, eğitim ile ilgili acil sorunları belirlememize yardımcı olabilir. Birinci katmanın ağırlıkları belirli bir düzene sahip olmalıdır. Eğer ağırlıklar halen rastgele gözüküyor ise ağın ek olarak biraz daha eğitime ihtiyacı olabilir. Genel olarak birinci katman, CIFAR-10 eğitim verilerinden kenar özelliklerini öğrenir ve ağırlıklarını buna göre ayarlar. Birinci konvolüsyon katmanının ağırlıklarının görselleştirilmiş hali Şekil 4.8'de verilmiştir.



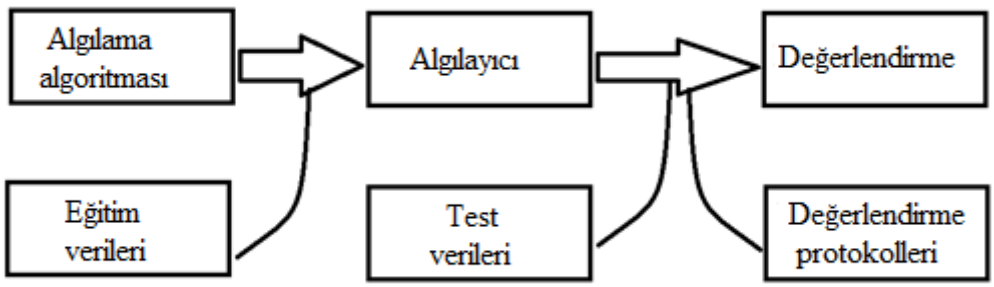
Şekil 4.8. Birinci konvolüsyon katmanın ağırlıkları

Eğitim sonuçlarını doğrulamak için, ağın sınıflandırma doğruluğunu ölçmek üzere CIFAR-10 test verilerini kullanılır. Doğruluk oranı düşük ise, eğitimin devam etmesi gerektiğini veya ek eğitim verilerine ihtiyaç duyulduğunu gösterir. Bu uygulamanın amacı, test setinde %100 doğruluk elde etmek değil, bir nesne detektörünün eğitiminde kullanılmak üzere bir ağı yeteri kadar eğitmektir.

Daha fazla eğitim doğruluğu arttıracaktır, ancak bu B-KSA ile nesne algılama için gerekli değildir. Şuan yukarıda anlatılan ağ yapısı CIFAR-10 veri kümesinde sınıflandırma yapmak için iyi bir şekilde çalışmaktadır. Trasnfer öğrenme yaklaşımı, yaya geçidi trafik tabelası algılamada ince ayar yapmak için kullanılabilir. Şimdi, kendi veri kümemiz olan yaya geçidi trafik tabelası veri kümesi yüklenir. Veri kümesi yüklenirken görüntünün ismi ve trafik tabelasının etiketi belirtilir. Etiket, trafik tabelasının etrafını çevreleyecek dikdörtgenin koordinat bilgisini içerir.

Bizim veri kümemizde eğitim için 95 tane görüntü bulunmaktadır. Sıfırdan bir B-KSA'yı trafik tabelası algılayıcı olarak eğitmek için 95 tane eğitim görüntüsü kullanmak yeterli ve güvenilir olmayabilir. Burada, CIFAR-10 (50000 eğitim görüntüsü) gibi büyük bir veri kümesiyle önceden eğitilen ağ yapısına ince ayar yaparak çok daha küçük bir veri kümesi kullanmamız mümkün olmuştur. Eğitim için en küçük yığın boyutu değeri 128, en büyük özyineleme değeri 100 olarak ayarlanmıştır.

Eğitim tamamlandıktan sonra test işlemine geçilir. Test için, yaya geçidi trafik tabelası test kümesi kullanılmıştır. Uygulama, çıktı olarak trafik tabelasını kutu içerisine alır ve bir puan üretir. Bu puan 0 ile 1 arasındadır ve algılamadaki güveni belirtir. Böylelikle düşük puandaki algılamalar göz ardı edilebilir. Algılama sisteminin çalışma ilkesi Şekil 4.9'da verilmiştir.

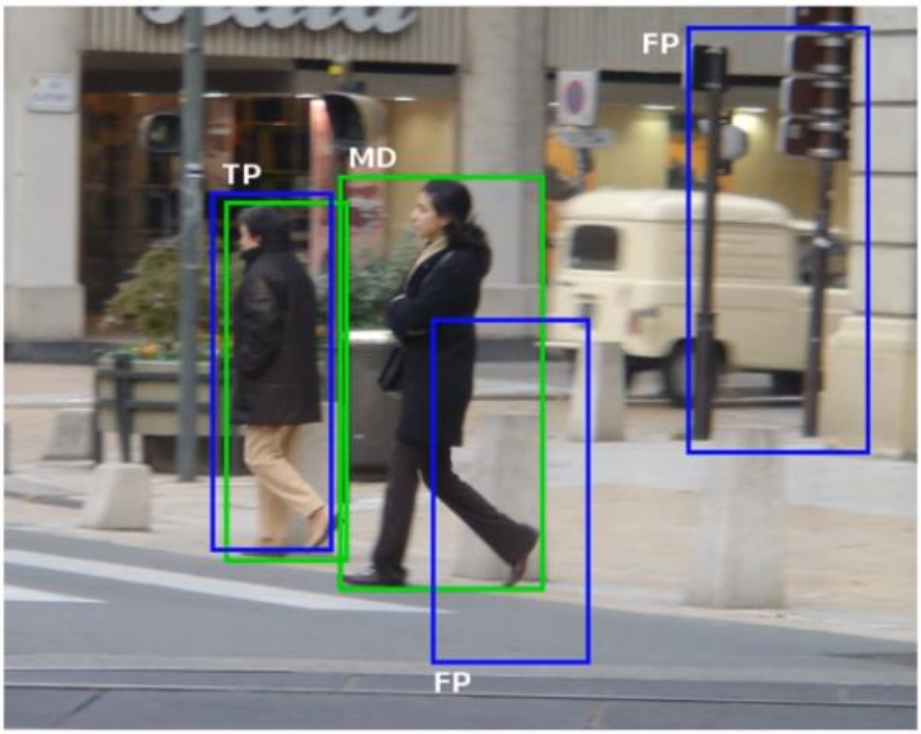


Şekil 4.9. Algılama sisteminin çalışma prensibi

Algılamanın doğru şekilde yapılıp yapılmadığını tespit etmek için belirli kriterler vardır. Bunlardan bir tanesi Pascal VOC kriteridir [67]. Pascal VOC'a göre, algılanan yaya (Det - Detection) ile test verisindeki yaya (GT - Ground Truth)'nın kesişim kümesi, iki görüntünün birleşim kümesinin %50'sinden büyük ise algılama doğru kabul edilir. Bu ölçütü gösteren örnek Şekil 4.10'da verilmiştir.

Şekil 4.10. INRIA veri kümesi üzerinde Pascal VOC kriteri [70]

Algılama sisteminin değerlendirirken 3 tane çıkış bilgisi kullanılır. Bu bilgiler; Doğru Pozitif (TP - True Positive), Yanlış Pozitif (FP - False Positive) ve Kaçırılan Algılama (MD - Missed Detection)'dır. Bu çıkışlara ait örnek bir görüntü Şekil 4.11'de verilmiştir.



Şekil 4.11. INRIA veri kümesinden alınan bir örnek algılama sisteminin çıkış görüntüsü [70]

Yaya geçidi trafik tabelası algılama sisteminin sonuçları değerlendirilirken Pascal VOC ölçütünden faydalanılmıştır. Uygulamamız, test kümesinde bulunan 77 görüntüden 73 tanesini doğru bir şekilde bularak %94,8 başarı oranı ile çalışmıştır. Uygulamamızın örnek çıktıları Şekil 4.12, Şekil 4.13 ve Şekil 4.14'te verilmiştir.

Şekil 4.12. Yaya geçidi tabelası algılama uygulaması örnek bir çıkış

Şekil 4.13. Yaya geçidi tabelası algılama uygulaması örnek bir çıkış

Şekil 4.14. Yaya geçidi tabelası algılama uygulaması örnek bir çıkış

4.1.3. Daha Hızlı Bölgesel Konvolüsyonel Sinir Ağı ile Yaya Geçidi Trafik Tabelası Algılama

Bu uygulamada, kendi oluşturduğumuz yaya geçidi trafik tabelası veri kümesi kullanılarak, Daha Hızlı B-KSA ile yaya geçidi trafik tabelası algılayıcı tasarlanmış ve sonuçlar değerlendirilmiştir.

4.1.3.1. Yaya Geçidi Trafik Tabelası Algılama

Bu uygulama için Windows ortamında MATLAB kullanılmıştır. Uygulama için öncelikle veri kümesi yüklenmiştir. Daha sonra sırasıyla KSA oluşturulmuş, eğitim parametreleri ayarlanmış, Daha Hızlı B-KSA ile algılayıcı eğitilmiştir. Son olarak, eğitilen ağ yapısı test kümesi üzerinde değerlendirilmiştir.

KSA, giriş katmanı, orta katman ve çıkış katmanı olmak üzere üç bölümden oluşmaktadır. KSA'nın giriş katmanı 32×32×3 boyutunda oluşturulmuştur. Daha sonra ağın orta katmanı oluşturulmuştur. Orta katman; konvolüsyon, DDB ve öznitelik seçici katmanlarının

tekrarlanmasından oluşturulmuştur. Birinci ve ikinci konvolüsyon katmanı için 3×3 boyutunda 32 tane filtre kullanılmıştır. Konvolüsyon katmanları için doldurma değeri ve kaydırma değeri 1 olarak ayarlanmıştır öznitelik seçici katmanının kaydırma değeri 2 olarak ayarlanmıştır.

Ağın son bölümü olan çıkış katmanı için tipik olarak tam bağlı katman, DDB ve softmax katmanından oluşur. Son olarak sınıflandırma katmanını ekleriz. Sınıflandırma katmanı çıkışında, giriş görüntüsünde herhangi bir sınıfa ait nesne olup olmadığına karar verilir. Daha sonra anlatılan giriş katmanı, orta katman ve çıkış katmanı sırasıyla birleştirilir.

Daha Hızlı B-KSA ile eğitim yaparken ağ 4 adımda eğitilir. Bu yüzden ağın eğitim parametreleri ayarlanırken 4 adım için farklı eğitim parametreleri kullanılabilir. Burada ilk iki adım ile Daha Hızlı B-KSA'da, kullanılan önerilen bölge ve algılama ağlarını eğitilmiştir. Son iki adım ise ilk iki adımda eğitilen ağları birleştirerek, algılama için tek bir ağ oluşturulmuştur. Bu dört adım için de en büyük özyineleme sayısı 10'dur. İlk iki adım için öğrenme oranı 0.00001 iken son iki adım için öğrenme oranı 0.000001 olarak ayarlanmıştır. Burada ilk iki adımın öğrenme oranı son iki adımın öğrenme oranından büyük seçilmiştir. Bunun nedeni son iki adımdaki ağırlıkların değişiminin ilk iki adımdaki kadar hızlı olması istenmemektedir. Çünkü son iki adımda ince ayar yapılmaktadır.

Eğitim tamamlandıktan sonra değerlendirme için yaya geçidi trafik tabelası test kümesi kullanılmıştır. Uygulama, çıktı olarak trafik tabelasını kutu içerisine alır ve bir puan üretir. Bu puan 0 ile 1 arasındadır ve algılamadaki güveni belirtir. Değerlendirme önceki uygulamada olduğu gibi Pascal VOC kriterine göre yapılmıştır. Uygulamamız, test kümesinde bulunan 77 görüntüden 63 tanesini doğru bir şekilde bularak %81,8 başarı oranı ile çalışmıştır. Uygulamamızın örnek çıktıları Şekil 4.15, Şekil 4.16 ve Şekil 4.17'te verilmiştir.

Şekil 4.15. Yaya geçidi tabelası algılama uygulaması örnek bir çıkış

Şekil 4.16. Yaya geçidi tabelası algılama uygulaması örnek bir çıkış

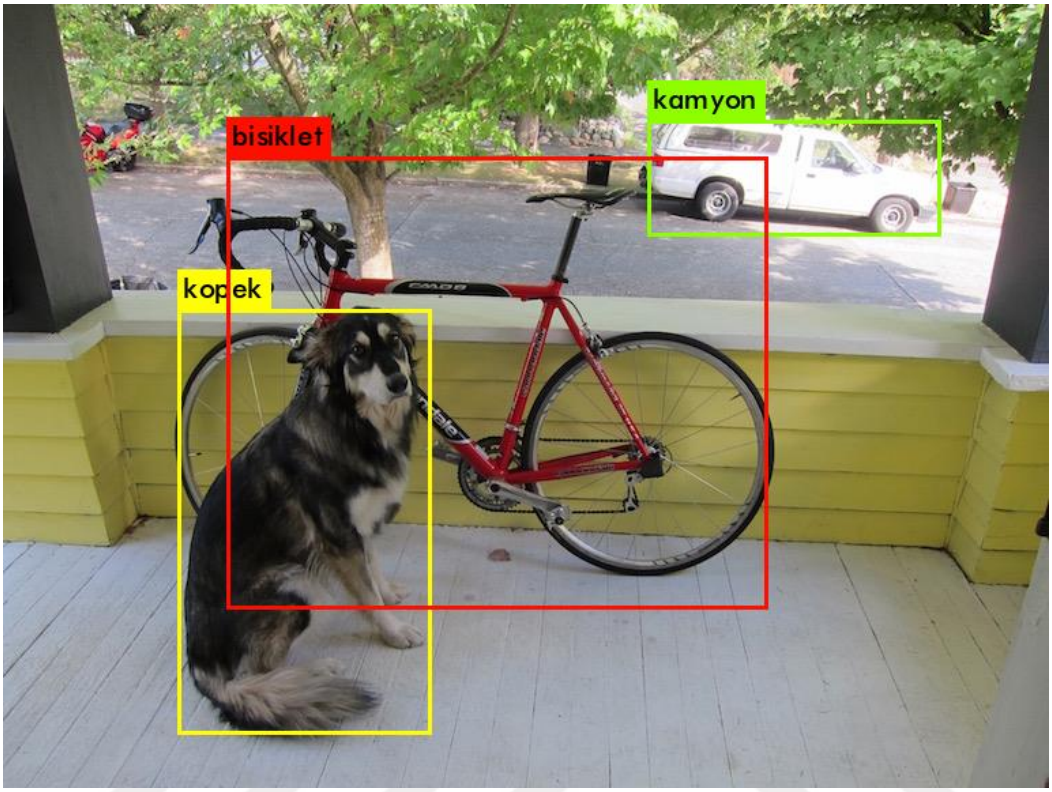
Şekil 4.17. Yaya geçidi tabelası algılama uygulaması örnek bir çıkış

4.2. YOLO ile Nesne Tespiti Uygulaması

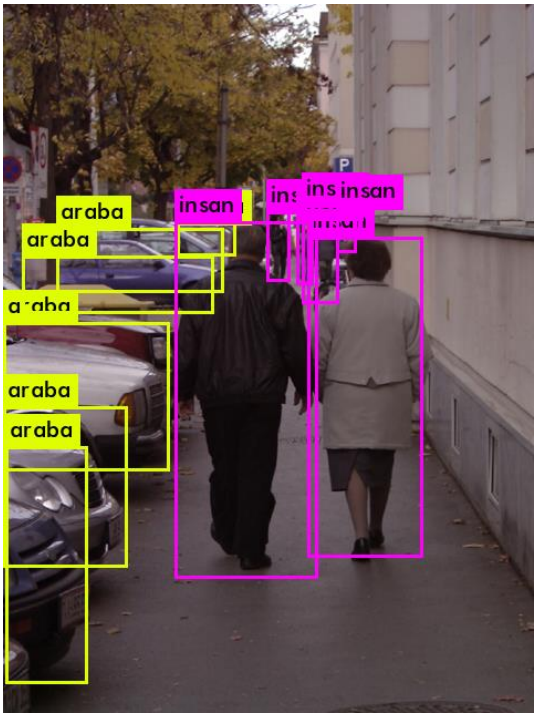
Nesne tespiti uygulamasının ilk zamanlarında kedi ile köpek arasında ayrım yapmak çok zordu. Günümüzde ise bu ayrım yaklaşık %99 oranında yapılabilmektedir. Otonom araç veya robot sistemi gibi bilgisayarlı görünün kullanıldığı bir sistem tasarlanmak istenildiğinde nesne algılamaya ihtiyaç duyulmaktadır. Darknet kütüphanesi, görüntü sınıflama ve nesne tespiti amacıyla kurulmuş bir kütüphanedir [68]. Darknet, C ve CUDA tabanlıdır ve açık kaynak kodludur.

Bu uygulamada, Darknet kütüphanesi ve YOLO yöntemi kullanılarak basit anlamda nesne tespiti uygulaması yapılmıştır. Bunun için Linux ortamında çalışılmıştır. Uygulama için öncelikle Darknet kütüphanesi indirilmiştir. İndirilen dosyalar, ekran kartının GPU desteği olup olmaması durumuna bağlı olarak GPU veya CPU üzerinde derlenebilir. Ağ yapısı, önceden Microsoft COCO (Common Objects in Context) veri kümesi ile eğitilmiştir. COCO veri kümesi 91 sınıfa ait 328,000 görüntü içinde toplam 2.500,000 etiketlenmiş nesneye sahiptir [69]. Uygulamaya geçmek için önceden eğitilmiş ağın, ağırlık dosyaları indirilmiştir. Nesnelerin etiketleri Türkçe olarak değiştirilmiştir. Son olarak, test işlemine geçilmiştir. Test

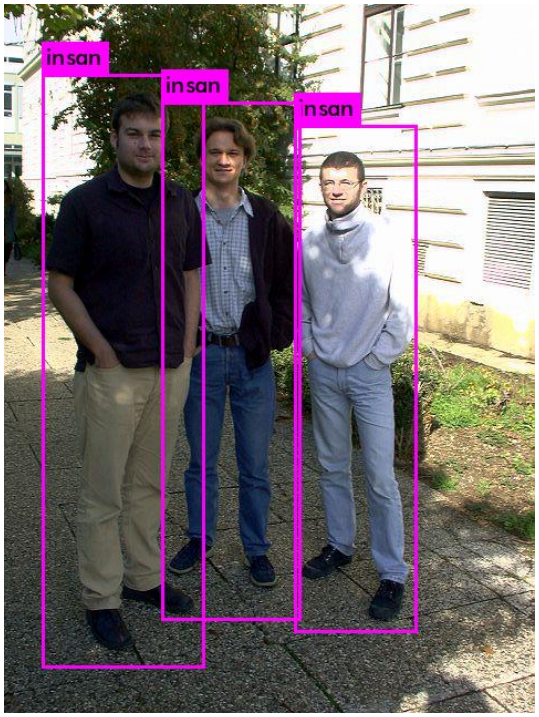
işleminde yayalar için hazırlanmış INRIA veri kümesi de kullanılmıştır [70]. YOLO ile nesne tespitine ait görüntüler Şekil 4.18, Şekil 4.19 ve Şekil 4.20'de verilmiştir.



Şekil 4.18. YOLO ile nesne tespiti [50]



Şekil 4.19. YOLO ile nesne tespiti [70]



Şekil 4.20. YOLO ile nesne tespiti [70]

5. OTONOM ARAÇ UYGULAMASI

Tezin bu bölümünde, otonom araçlar ile ilgili yapılan çalışmalar doğrultusunda, bir otonom araç yapılmaya çalışılmıştır [71-74]. İlk olarak uygulamada kullanılan araç ve sensörler tanıtılmış, sonrasında yapılan çalışmalar ayrıntılı olarak anlatılmıştır.

5.1. Donanım

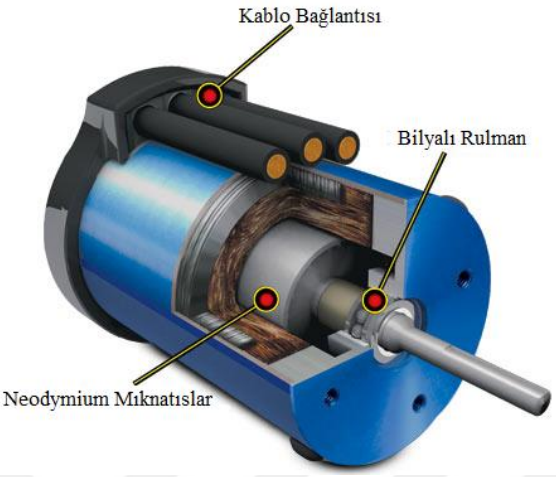
5.1.1. Araç Platformu

Otonom araç uygulaması için, Traxxas firmasının ürettiği, radyo kontrollü ve uzaktan kumandalı Traxxas Slash 4×4 Platinyum versiyonlu araç kullanılmıştır [75]. Aracın görüntüsü Şekil 5.1'de verilmiştir.

Şekil 5.1. Traxxas slash 4x4 platinum edition

Günümüzdeki araçlarda genellikle Ackerman direksiyon sistemi kullanılmaktadır. Ackermann direksiyon sistemi, Rudolph Ackermann tarafından 1817 yılında geliştirilmiştir [76]. Ackerman direksiyon sistemi, aracın dönüşü sırasında içteki tekerleğin dıştaki tekerleğe göre daha keskin bir dönüş yapmasını sağlar. Traxxas'ın kullanılan aracında, Ackerman direksiyon sistemi kullanılmıştır. Traxxas aracında direksiyon kontrolü için bu çalışmada servo motor kullanılmıştır. Servo motoru sürmek için Darbe Genişlik Modülasyonu (PWM - Pulse Width Modulation) servo sürücü kullanılmıştır.

Araçta Velineon 3500 fırçasız motor kullanılmıştır. Bu motor güç ve uzun ömür için, yüksek sıcaklıkta sinterlenmiş Neodymium mıknatıslar ve yüksek hızlı bilyalı rulmanlar ile üretilmiştir [77]. Araçta kullanılan motor Şekil 5.2'de verilmiştir.



Şekil 5.2. Velineon 3500 fırçasız motor

Uygulamada aracın üzerindeki orijinal Elektronik Hız Kontrolör (ESC - Electronic Speed Controller) çıkarılarak onun yerine VESC (Vedder Electronic Speed Controller) takılmıştır. Aracın üzerindeki ESC'nin en düşük hızı saatte yaklaşık 9 kilometredir. VESC kullanımının ana sebebi hızı düşürerek aracın kontrolünü arttırmaktır. VESC mimari olarak, STM32 ARM Cortex işlemciye sahiptir ve açık kaynak kodludur [78]. VESC'ye ait görüntü Şekil 5.3'de verilmiştir.

Şekil 5.3. VESC

Araçta 7 hücreli 8.4 Volt 3000 mAh NiMH (Nikel-Metal Hidrit) batarya kullanılmıştır. Kullanılan batarya Şekil 5.4'te gösterilmiştir. Kullanılan batarya türüne göre aracın çıkabileceği en yüksek hız değişmektedir. Örneğin, 3000 mAh NiMH batarya ile saatte

yaklaşık 55 kilometre hıza çıkabilirken, 5000 mAh 3S LiPo batarya ile saatte yaklaşık 95 kilometre hıza çıkabilmektedir.

Şekil 5.4. 3000 mAh NiMH batarya

Veri toplama aşamasında, aracı kontrol etmek için Logitech F710 kablosuz oyun kolu kullanılmıştır. Bu oyun kolu, radyo kontrol ile haberleşen 2.4 GHz hızında nano alıcıya sahiptir. Kullanılan oyun kolu Şekil 5.5'te verilmiştir.

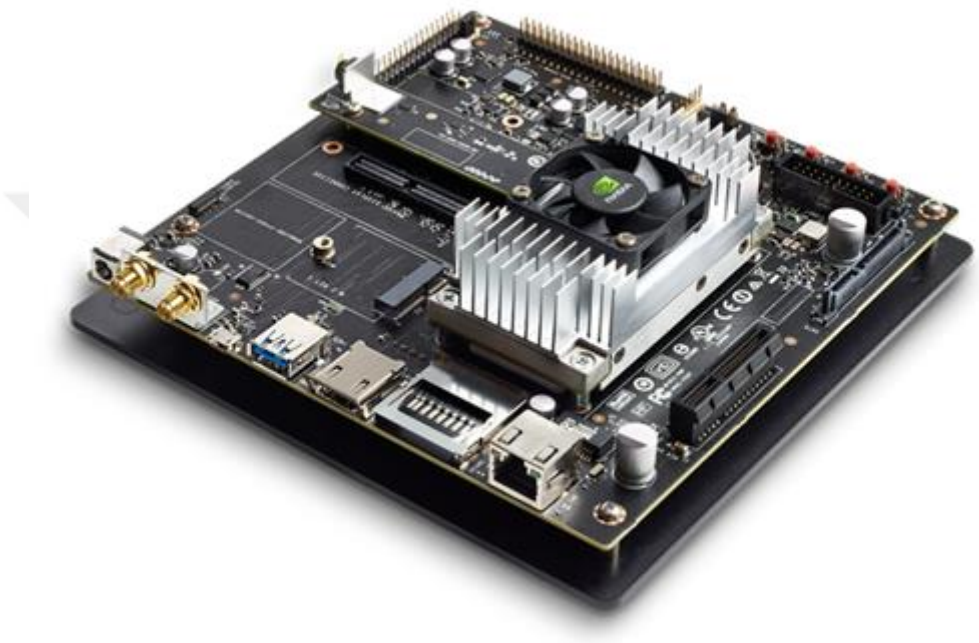
Şekil 5.5. Logitech F710 oyun kolu

5.1.2. Nvidia Jetson TX2

Nvidia Jetson TX2, düşük güç ve yüksek performans ile gerçek zamanlı yapay zeka ve görüntü işleme uygulamaları için kullanılabilir. Jetson TX2, GPU olarak NVIDIA Pascal™, 256 CUDA çekirdeğine sahipken, CPU olarak HMP Dual Denver 2/2 MB L2 + Quad ARM® A57/2 MB L2 sahiptir [79]. Nvidia Jetson TX2 geliştirme kitinin üzerinde sabit odaklı 5 MP çözünürlüğe sahip kamera bulunmaktadır. Nvidia Jetson TX2 geliştirme kitinin görüntüsü Şekil 5.6'da verilmiştir. Kart üzerindeki bağlantılar aşağıdaki gibidir:

- USB 3.0 girişi,
- USB 2.0 mikro girişi,
- HDMI girişi,
- M.2 Key,

- PCI-E x4,
- Gigabit Ethernet,
- Tam Boyut SD,
- SATA Verileri ve Güç,
- GPIOs, I2C, I2S, SPI,
- TTL UART.



Şekil 5.6. Nvidia Jetson TX2

Otonom araçlarda bulunan kamera, mesafe sensörü ve diğer sensörlerden anlık olarak birçok veri gelmektedir. Bu nedenle, GPU'lar otonom sürüş uygulamaları için kilit rol oynamaktadır. GPU'lar tarafından sağlanan yüksek paralellik, aracın hareket planlaması için sensörlerden gelen verileri hızlı bir şekilde değerlendirerek karar vermesini sağlar.

5.1.3. Stereo Kamera

ZED kamera, derinlik algılama ve hareket izleme için dünyanın ilk 3 boyutlu kamerası olarak Stereolabs firması tarafından geliştirilmiştir [80]. ZED kamerada, insan görüsünü temel alan gelişmiş algıma sistemleri kullanılmıştır. ZED kamera, derinlik algılama, hareket izleme ve 3 boyutlu haritalama gibi uygulamalar için kullanılabilir. 2.2K video çekerken saniyede 15 görüntü alma kapasitesi ile dünyanın en hızlı kamerası olmuştur. Geniş açılı lensleri sayesinde

algılama açısı 110 derecedir. 0,5 metre ile 20 metre arasında algılama yapabilmektedir. ZED kameranın görüntüsü Şekil 5.7'de verilmiştir.

Şekil 5.7. ZED kamera

5.1.4. İki Boyutlu Lazer Tarayıcı

İki boyutlu (2B) lazer tarayıcı (Lidar) olarak A2M6 isimli 2B-Lidar kullanılmıştır. Kullanılan 2B-Lidar 360 derece tarama özelliğine sahiptir. Bir saniyede 4000 adet örnek alabilmektedir. Menzili 20 metredir. Normal tarama hızı 10 Hz olmasına karşın, istenilen durumlarda 5-15 Hz arasında kullanılabilir [81]. 2B-Lidar'ın görüntüsü Şekil 5.8'de verilmiştir.

Şekil 5.8. 2B-Lidar

5.1.5. Dokuz Serbestlik Dereceli İvmeölçer

SparkFun firmasının 9 serbestlik dereceli ivmeölçer (IMU - Inertial Measurement Unit) kullanılmıştır [82]. 9 serbestlik dereceli IMU, 3 eksen jiroskop, ivmeölçer ve manyetometre özelliği sayesinde açısal hız, ivme ve hareket yönünü algılamada kullanılabilir. İvmeölçer ± 2, 4, 8 ve 16 g, jiroskop ± 245, 500 ve 2000 °/s, manyetometre ± 4, 8, 12 ve 16 gauss aralığında ölçüm yapabilmektedir. Çalışmada kullanılan 9 serbestlik dereceli IMU sensör Şekil 5.9'da gösterilmiştir.

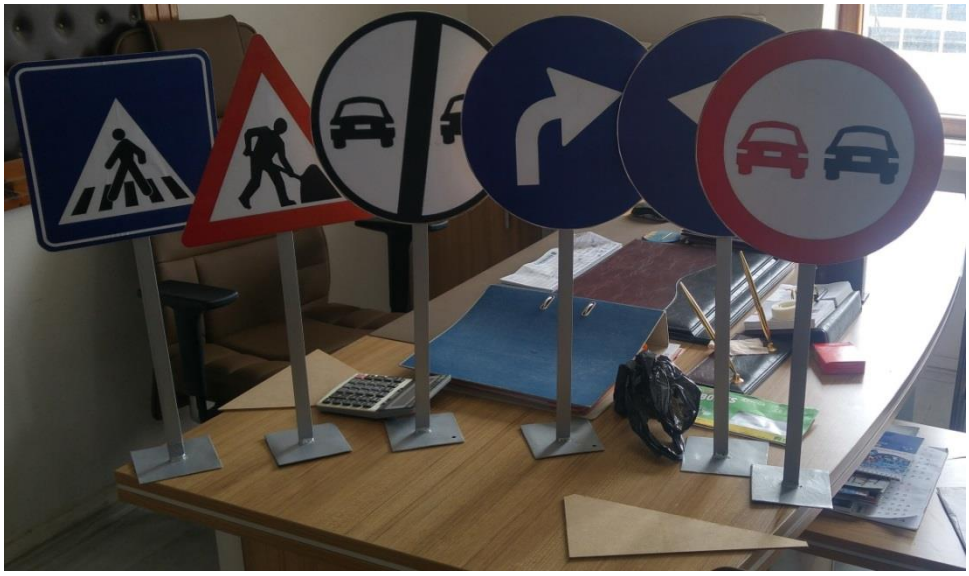
Şekil 5.9. 9 serbestlik dereceli IMU

5.2. Yazılım

Aracın işletim sistemi Linux for Tegra (L4T) isimli bir Linux sürümüdür. Bu işletim sistemine, CUDA, cuDNN, VisionWorks, OpenCV4Tegra yazılımları Nvidia'nın JetPack isimli yazılım paketi kurularak yüklenmiştir. Ayrıca, IMU, VESC, LIDAR ve kameranın çalışması için sürücüleri yüklenmiştir. Mobil robotun benzetimi ve kontrolü için Robot İşletim Sistemi (ROS- Robot Operating System) yüklenmiştir.

5.3. Parkur

Aracın üzerinde hareket etmesi için eni 4 metre, boyu 5,5 metre olan bir parkur hazırlanmıştır. Parkur üzerinde yol çizgileri yapılmıştır. Parkura yerleştirmek için çeşitli trafik levhaları hazırlanmıştır. Hazırlanan trafik levhaları Şekil 5.10'da verilmiştir. Şekil 5.11, Şekil 5.12 ve Şekil 5.13'te parkura ait fotoğraflar verilmiştir.



Şekil 5.10. Trafik levhaları

Şekil 5.11. Parkur görüntüsü - 1

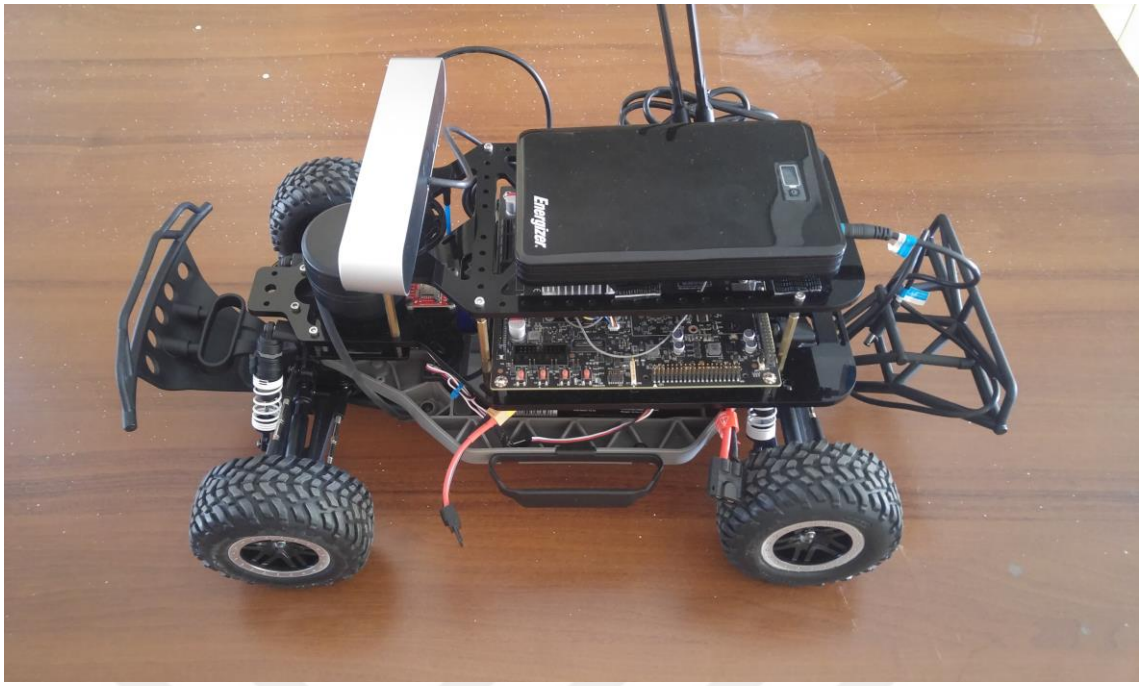
Şekil 5.12. Parkur görüntüsü - 2

Şekil 5.13. Parkur görüntüsü - 3

5.4. Uygulama

Uygulamada kullanmak için Traxxas aracının kaporta kısmı sökülmüştür. Otonom araç için gerekli olan kamera ve sensörler aracın şasesine sabitlenmiştir. Aracın önden görüntüsü Şekil 5.14'te, yandan görüntüsü Şekil 5.15'te verilmiştir.

Şekil 5.14. Aracın önden görünüşü



Şekil 5.15. Aracın yandan görünüşü

Sonra araç parkura konularak Logitech F710 oyun kolu yardımıyla el ile kontrol edilmiştir. Araç, öğrenmesi istenilen alanlarda sürülmüştür. Bunun amacı, otonom aracın eğitiminde kullanılmak üzere kamera ve sensörlerden eğitim verileri toplayarak eğitim kümesi oluşturmaktır. El ile sürüş sonucunda, kameradan 1243 tane görüntüye ek olarak, araca ait hız ve direksiyon açılarının değerleri de kayıt altına alınmıştır. Kameradan alınan ilk 30 görüntü düzgün olmadığı için eğitim için kullanılmamıştır. Elde edilen görüntülerin %80'i eğitim kümesi, %20'si test kümesi olacak şekilde ayrılmıştır. Kameradan alınan görüntülere ait örnekler Şekil 5.16 ve Şekil 5.17'de verilmiştir.

Araç parkur üzerinde sürülürken kameradan alınan görüntülerin, açıya göre dağılımını görmek için, Şekil 5.18'deki gibi bir histogram çizilmiştir. Açıların 0-1 aralığında düzgün dağılması için veri çoğaltma işlemi uygulanmıştır. Sayısı az olan açı değerleri veri kümesine yeni bir veri olarak tekrar eklenmiştir. Böylece, dengeli bir histogram Şekil 5.19'daki gibi elde edilmiştir.

Aracın her ortamda doğru bir şekilde hareket edebilmesi için, veriye parlaklık değişimi uygulanmıştır. Araçtan alınan görüntünün tüm pikselleri gerekli değildir. Bu nedenle üstten 144, alttan 76, soldan 0 ve sağdan 672 piksel alınmıştır. Bu nedenle; ilk olarak, ağa kırpıcı bir katman oluşturulmuştur. Tüm ağ yapısı Tablo 5.1'deki gibidir. Daha sonra elde edilen bu veriler ile oluşturulan ağ yapısı eğitilmiştir. Eğitim sonuçları Şekil 5.20'de verilmiştir. Sonuçta elde edilen direksiyon açısı, ağ sonuç verdikten sonra -0.3 ila 0.3 aralığından -1 ila 1

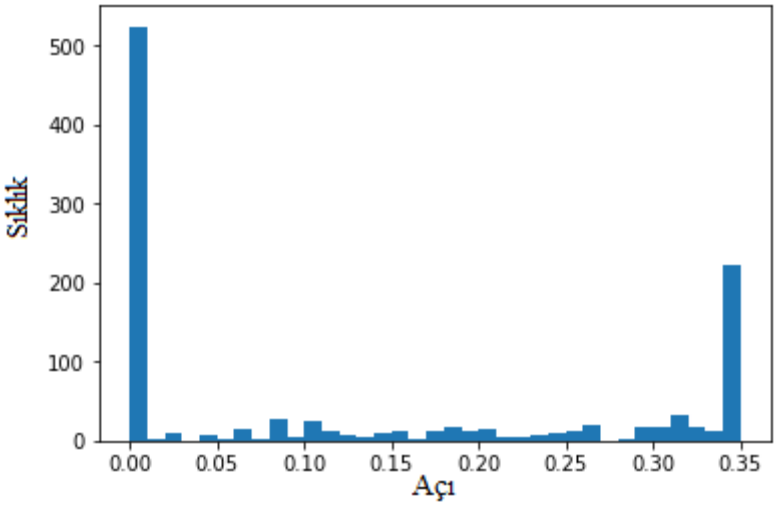
aralığına çevirmek için 3 ile çarpılmıştır. Aracın parkur üzerindeki görüntüsü Şekil 5.21 ve Şekil 5.22'de verilmiştir.

Tablo 5.1. Ağ yapısı

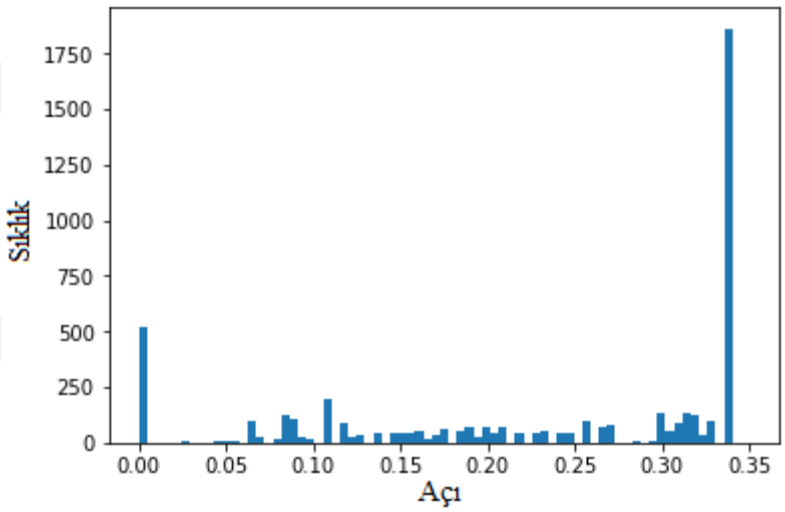
Katman	**Çıkış Biçimi**	**Parametre Sayısı**
cropping2d_1 (Kırpma2B)	(None, 156, 672, 3)	0
lambda_1 (Lambda)	(None, 156, 672, 3)	0
conv2d_1 (Konvolüsyon2B)	(None, 76, 334, 24)	1824
conv2d_2 (Konvolüsyon2B)	(None, 36, 165, 36)	21636
conv2d_3 (Konvolüsyon2B)	(None, 16, 81, 48)	43248
conv2d_4 (Konvolüsyon2B)	(None, 14, 79, 64)	27712
conv2d_5 (Konvolüsyon2B)	(None, 12, 77, 64)	36928
flatten_1 (Vektöre dönüştürme)	(None, 59136)	0
dense_1 (Sık bağlı katman)	(None, 100)	5913700
dense_2 (Sık bağlı katman)	(None, 50)	5050
dense_3 (Sık bağlı katman)	(None, 10)	510
dense_4 (Sık bağlı katman)	(None, 1)	11

Şekil 5.16. Araç kamerasından alınan bir görüntü - 1

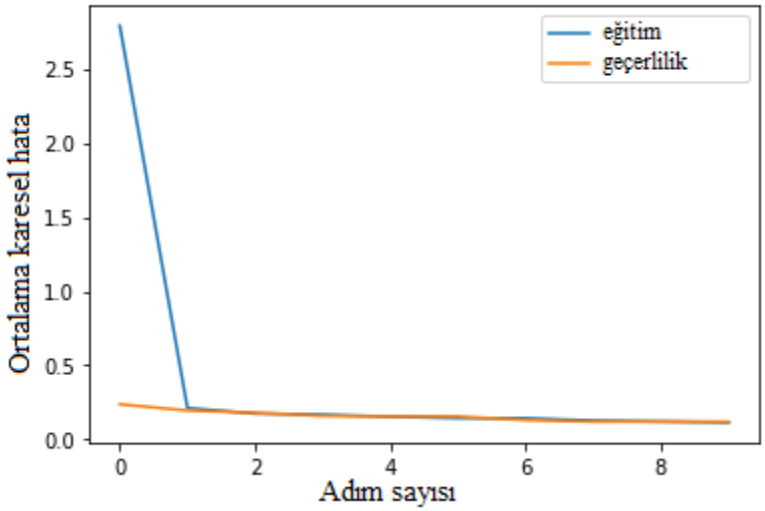
Şekil 5.17. Araç kamerasından alınan bir görüntü - 2



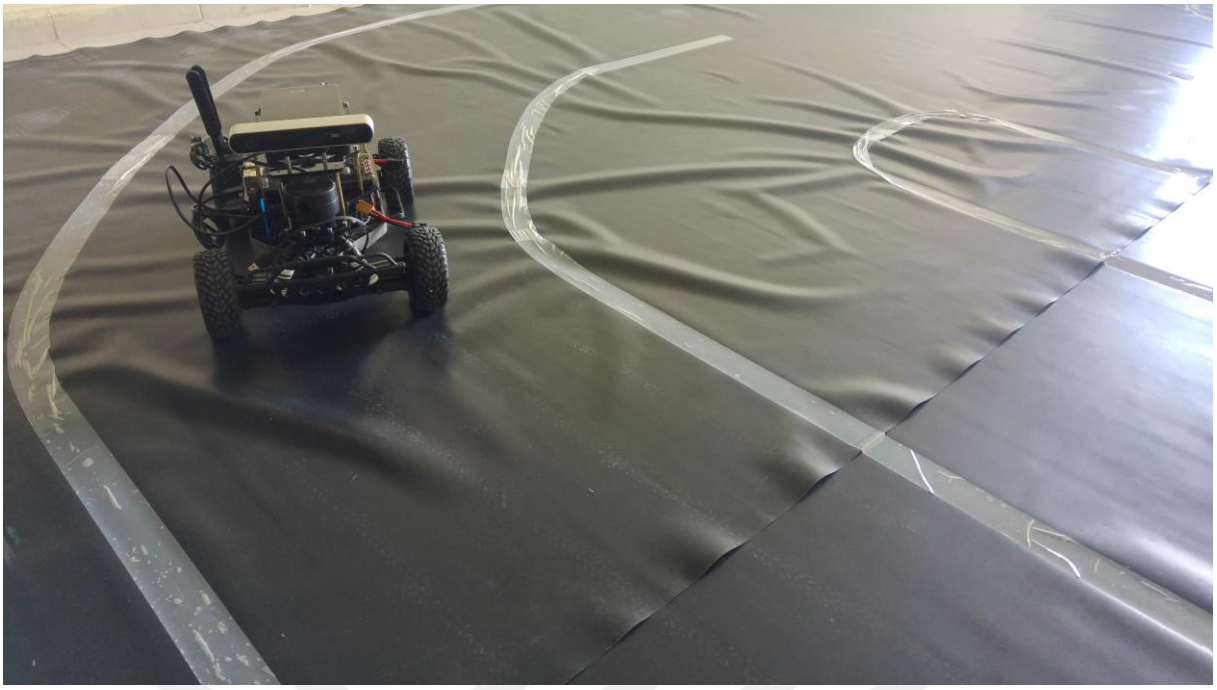
Şekil 5.18. Tekerin açı değerlerine ait histogram grafiği



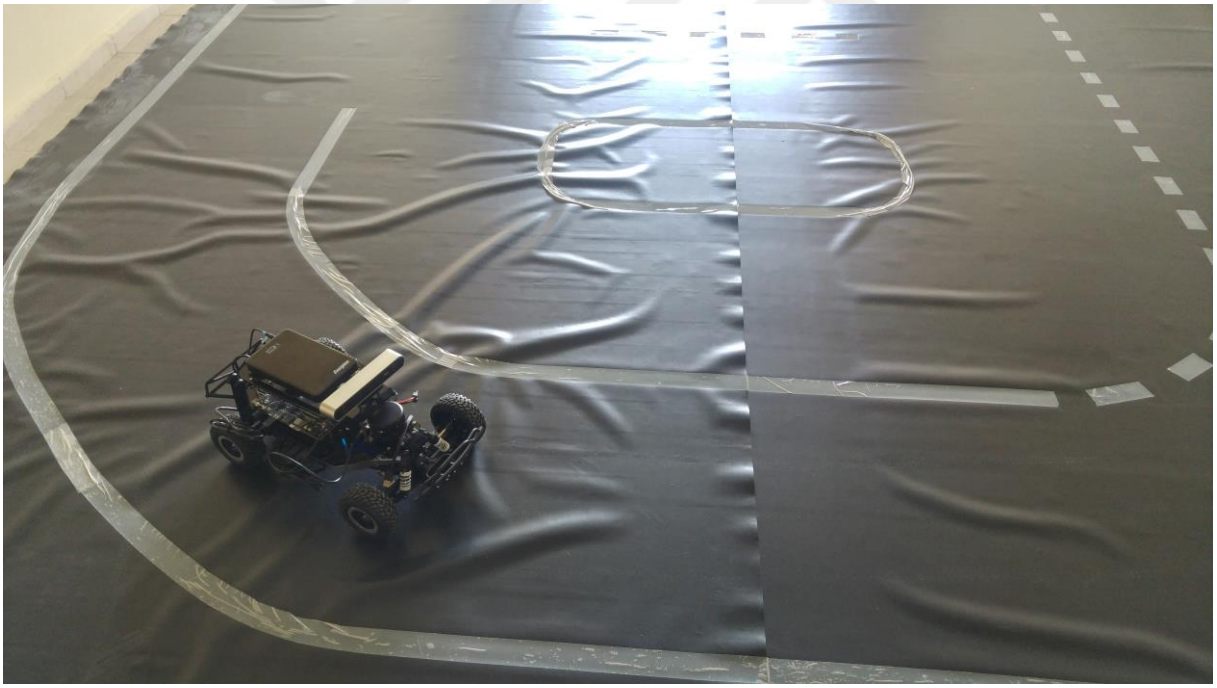
Şekil 5.19. Veri arttırma yöntemi ile elde edilmiş histogram grafiği



Şekil 5.20. Eğitim sonuçları



Şekil 5.21. Aracın parkur üzerindeki görüntüsü - 1



Şekil 5.22. Aracın parkur üzerindeki görüntüsü - 2

6. SONUÇLAR

Otonom araçlar, yol koşullarını ve çevrelerini insan yardımı olmadan kamera ve sensör bilgileri ile kendileri algılayan araçlardır. Otonom araçlar, çevrelerini algılamak için kamera, mesafe sensörü, küresel konumlandırma sistemi gibi çeşitli sensör verilerini farklı yöntemlerle bir araya getirerek kullanırlar. Otonom araçlar için gelişmiş kontrol sistemleri, navigasyondan aldığı uygun yol bilgisinin yanı sıra yollardaki engeller, tarfik akışı ve trafik tabelalarını tanımlamak için elde ettiği verileri değerlendirir.

Bu çalışmada, genel olarak otonom araçlar ile ilgili derin öğrenme yöntemleri incelenmiş ve bir otonom yer aracı üzerinde uygulamalar yapılmıştır. Bunun için öncelikle, Linux ortamında Caffe kullanılarak, MNIST veri kümesi ile rakam tanıma uygulaması yapılmıştır. Bu uygulama için Nvidia TK1 ve Nvidia TX1 kartlarına ek olarak iki adet bilgisayar kullanılarak, CPU ile GPU'lar hız ve doğruluk bakımından değerlendirilmiştir. Bu uygulama ile birlikte GPU'ların, CPU'lara göre üstün yönleri gözlemlenmiş ve sonuçlar tablo halinde sunulmuştur. İkinci olarak nesne algılama uygulamaları yapılmıştır. Bu amaçla ilk olarak, Windows ortamında Matlab'da yaya geçidi trafik tabelası algılama uygulaması yapılmıştır. Bu uygulama için Elazığ, Niğde ve Kayseri şehirlerinde yaya geçitlerinde çekilen görüntüler ile oluşturulan veri kümesi kullanılmıştır. Bu veri kümesinde 172 fotoğrafta toplam 185 tane yaya geçidi tabelası bulunmaktadır. Algılama için B-KSA, Daha Hızlı B-KSA kullanılmış ve sonuçlar incelenmiştir. Sonrasında, Linux ortamında temel anlamda YOLO kullanılarak nesne algılama uygulaması yapılmış ve çıktılara yer verilmiştir. Son olarak, otonom bir yer aracı tasarlanmıştır. Bu amaçla ilk olarak, aracın üzerinde hareket edebileceği parkur oluşturulmuştur. Araç, oluşturulan bu parkur üzerinde el ile hareket ettirilmiş ve ağı eğitmek için gerekli veriler toplanmıştır. Sonrasında toplanan veriler ile ağ eğitilmiştir. Son olarak, tasarlanan otonom araç parkur üzerinde test edilmiştir.

Tüm uygulamalar başarılı bir şekilde gerçekleştirilmiştir. Elde edilen sonuçlar karşılaştırmalı olarak verilmiştir.

Gelecek çalışma olarak daha kapsamlı bir algılama sistemi tasarlanarak daha gelişmiş bir otonom araç gerçekleştirilebilir. Algılamadaki yanlışlıkları en aza indirmek, aracın hızını arttırmak ve aracın daha verimli çalışmasını sağlamak gelecek araştırmalara konu olabilir.

KAYNAKLAR

[1] Gomez, V., 2015. Object detection for autonomous driving using deep learning, PhD Thesis, Universitat At Politecnica De Catalunya.

[2] Nikbay, K., 2015. Otonom araçların güzergah takibi için bir uygulama, Yüksek Lisans Tezi, Okan Üniversitesi, Fen Bilimleri Enstitüsü, İstanbul.

[3] URL-1, http://www.extremetech.com/extreme/105117-inventing-our-world-darpas-top-inventions/4

[4] URL-2, Google Self-Driving Car Project Montly Report, 2015. https://www.google.com/selfdrivingcar/reports/

[5] Çekmez, U., 2014. İnsansız hava araçlarında büyük ölçekli yol planlama problemlerinin GPU üzerinde CUDA yardımı ile çözümü, Yüksek Lisans Tezi, Hava Harp Okulu, Havacılık ve Uzay Teknolojileri Enstitüsü, İstanbul.

[6] URL-3, http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-980-ti

[7] URL-4, Mobileye pedestrian collision warning system, www.mobileye.com/technology/applications/pedestriandetection/pedestrian-collision-warning/

[8] Coelingh, E., Eidehall, A. and Bengtsson, M., 2010. Collision warning with full auto brake and pedestrian detection - a practical example of automatic emergency braking. In Proceedings of the 13th International IEEE Conference on Intelligent Transportation Systems (ITSC), 155–160, Funchal.

[9] URL-5, BMV Driving Assistance Package, http://www.bmw.com/

[10] URL-6, VW Emergency Assistance System, http://safecarnews.com/

[11] URL-7, http://www.toyota-global.com/innovation/safety_technology/toyota-safety-sense/

[12] Dalal, N. and Triggs, B., 2005. Histograms of oriented gradients for human detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 886–893, San Diego.

[13] Ke, Y. and Sukthankar, R., 2004. Pca-sift: A more distinctive representation for local image descriptors. In Proceedings of the Computer Vision and Pattern Recognition (CVPR), 398–400, Washington.

[14] Lowe, D., 1999. Object recognition from local scale-invariant features. In Proceedings of the Seventh IEEE International Conference on Computer Vision (ICCV), **2**, 1150–1157, Kerkyra.

[15] Bay, H., Tuytelaars, T. and Gool, V., 2006. Surf: Speeded up robust features. In Proceedings of the European Conference on Computer Vision (ECCV), 404–417, Graz.

[16] Calonder, M., Lepetit, V., Strecha, C., and Fua, P., 2010. Brief: Binary robust independent elementary features. In Proceedings of the European Conference on Computer Vision (ECCV), 778–792 Greece.

[17] Rublee, E., Rabaud, V., Konolige, K. and Bradski, G., 2011. Orb: an efficient alternative to sift or surf. In Proceedings of the International Conference on Computer Vision (ICCV), 2564–2571, Barcelona.

[18] Leutenegger, S., Chli, M. and Siegwart, R., 2011. Binary robust invariant scalable keypoints. In Proceedings of IEEE International Conference on Computer Vision (ICCV), 2548–2555, Barcelona.

[19] Alahi, A., Ortiz, R. and Vandergheynst, P., 2012. Freak: Fast retina keypoint. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 510–517, Rhode Island.

[20] Uçar, A., Demir, Y. and Güzeliş, C., 2014. A penalty function method for designing efficient robust classifiers with input–space optimal separating surfaces, Turkish Journal of Electrical Engineering and Computer Science, **22**, 1664–1685.

[21] Uçar, A., Demir, Y. and Güzeliş, C., 2016. A new facial expression recognition based on curvelet transform and online sequential extreme learning machine initialized with spherical clustering, Neural Computing and Applications, **27**, 131–142.

[22] Huang, G. B., Zhu, Q. Y. and Siew, C. K., 2006. Extreme learning machine: theory and applications, Neurocomputing, **70**, 489–501.

[23] Ho, T. K., 1998. The Random Subspace Method for Constructing Decision Forests, IEEE Transactions on Pattern Analysis and Machine Intelligence, **20**, 832–844.

[24] Webb, G. I., Boughton, J. and Wang, Z., 2005. Not So Naive Bayes: Aggregating One-Dependence Estimators, Machine Learning (Springer), **58**, 5–24.

[25] Deng, L. and Yu, D., 2014. Deep Learning: methods and applications, Foundations and Trends in Signal Processing, **7**, 3–4.

[26] Christian, S., Toshev, A. and Erhan, D., 2013. Deep neural networks for object detection. In Proceeding Advances in Neural Information Processing Systems, 2553–2561 Harrah's Lake Tahoe.

[27] Krizhevsky, A., Sutskever, I. and Hinton, G. E., 2012. Imagenet classification with deep convolutional neural networks, in Neural Information Processing Systems, 1097–1105, Harrah's Lake Tahoe.

[28] Deng, J., Berg, A., Satheesh, S., Su, H., Khosla, A. and Fei-Fei, L., 2012. http://www.image-net.org/challenges/LSVRC/2012/

[29] Berg, A., Deng, J. and Fei-Fei, L., 2010. Large scale visual recognition, International Journal of Computer Vision, **115(3)**, 211-252.

[30] Duda, R. O., Hart, P. E. and Stork, D. G., 1973. Pattern Classification, Second Edition, Wiley.

[31] Hornik, K., Stinchcombe, M. and White, H., 1989. Multilayer feedforward networks are universal approximators. Neural Networks, **2(5)**,359–366.

[32] Bengio, Y., 2009. Learning deep architectures for AI. Foundations and Trends in Machine Learning, **1**,1–127.

[33] Bishop, C., 1995. Neural Networks for Pattern Recognition. Clarendon Press, Oxford.

[34] Haykin, S., 2005. Neural Networks A Comprehensive Foundation. Pearson Education, New Delhi.

[35] Duda, R., Hart, P. and Stork, D., 2001. Pattern Classification. Wiley-Interscience Publication, New York.

[36] Jarrett, K., Kavukcuogl, K., Ranzato, M. and LeCun, Y., 2009. What is the best multi-stage architecture for object recognition. In Proceedings of the International Conference on Computer Vision (ICCV) 2146–2153, Nevada.

[37] Uçar, A. ve Bingöl, M. S., 2018. Derin öğrenmenin Caffe kullanılarak grafik işleme kartlarında değerlendirilmesi, DÜMF Mühendislik dergisi, **9(1)**, 39–49.

[38] Coşkun, M., Yıldırım, Ö., Uçar, A. and Demir, Y., 2017. An overvıew of popular deep learnıng methods, European Journal of Technique, **7(2)**, 165–176.

[39] Tümen, V., Yıldırım, Ö. and Ergen, B., 2018. Detection of driver drowsiness in driving environment using deep learning methods. In Proceeding of Elektrik-Elektronik, Bilgisayar, Biyomedikal Mühendislikleri Bilimsel Toplantısı (EBBT), 1–5, İstanbul.

[40] Ma, X., Dai, Z., He, Z., Na, J., Wang, Y. and Wang, Y., 2017. Learning traffic as ımages: a deep convolutional neural network for large-scale transportation network speed prediction, Sensors, **17(4)**, 818.

[41] Tümen, V., Yıldırım, Ö. and Ergen, B., 2017. A convolutional neural network model for road flow direction detection, Akıllı Sistemlerde Yenilikler ve Uygulamaları (ASYU), 81, Antalya.

[42] Schmidhuber, J., 2015. Deep learning in neural networks: An overview, Neural Networks, **61**, 85–117.

[43] Chen, C., Seff, A., Kornhauser, A. and Xiao, J., 2015. DeepDriving: Learning affordance for direct perception in autonomous driving. In Proceeding of the IEEE International Conference on Computer Vision (ICCV), 2722–2730, Chile.

[44] Bojarski, M, Testa, D. D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J. and Zieba, K., 2016. End to end learning for self-driving cars, arXiv:1604.07316 [cs.CV].

[45] LeCun, Y., Kavukvuoglu, K. and Farabet, C., 2010. Convolutional networks and applications in vision. In Proceeding of Circuits and Systems International Symposium, 253–256, Grenoble.

[46] Forsyth, D. and Ponce, J., 2002. Computer Vision: a modern approach, prentice hall professional technical reference, New Jersey.

[47] Girshick, R., Donahue, J., Darrell, t. and Malik, J., 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 580–587, Ohio.

[48] Girshick, R., 2015. Fast R-CNN. In Proceedings of the IEEE International Conference on Computer Vision (ICCV), 1440–1448, Chile.

[49] Ren, S., He, K., Girshick, R. and Sun, J., 2017. Faster R-CNN: towards real-time object detection with region proposal networks, IEEE Transactions on Pattern Analysis & Machine Intelligence, **6**, 1137–1149.

[50] Redmon, J., Divvala, S., Girshick, R. and Farhadi, A., 2016. You Only Look Once: Unified, Real-Time Object Detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 779–788, Nevada.

[51] Redmon, J. and Farhadi, A., 2016. YOLO9000: Better, Faster, Stronger, arXiv:1612.08242v1 [cs.CV].

[52] Glorot, X. and Bengio, Y., 2010. Understanding the difficulty of training deep feedforward neural networks. In Artificial Intelligence and Statistics, International Conference on, 249–256, Sardinia.

[53] Krizhevsky, A., Sutskever, I. and Hinton, G. E., 2012. ImageNet classification with deep convolutional neural networks. In Advances in Neural Information Processing Systems, 1097–1105, Harrah's Lake Tahoe.

[54] Berkeley Design Technology: A test drive of the nvidia jetson tx1 developer kit for deep learning and computer vision applications.
https://www.bdti.com/MyBDTI/pubs:Nvidia_JetsonTX1_Kit.pdf.

[55] Jia. Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., Darrell, T., 2014. Caffe: Convolutional Architecture for Fast Feature Embedding. In Proceedings of the 22nd ACM International Conference on Multimedia, 675–678, Amsterdam

[56] LeCun, Y., Jackel, L. D., Boser, B., Denker, J. S., Graf, H. P., Guyon, I., Henderson, D., Howard, R. E. and Hubbard, W., 1989. Handwritten Digit Recognition: Applications of Neural Network Chips and Automatic Learning, **27(11)**, 41–46.

[57] URL-8, CUDA c programming guide, https://docs.nvidia.com/cuda/

[58] LeCun, Y., Cortes, C. and Burges, C. J. C., 1998. The MNIST database of handwritten digits, http://yann.lecun.com/exdb/mnist/

[59] LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P., 1998. Gradient-Based Learning Applied to Document Recognition. In Proceedings of the IEEE, **86(11)**, 2278–2324.

[60] Alsaafin, A. and Elnagar, A., 2017. A Minimal subset of features using feature selection for handwritten digit recognition, journal of ıntelligent learning systems and applications, **9**, 55–68.

[61] LeCun, Y., Jackel, L. D., Bottou, L., Cortes, C., Denker, J. S., Drucker, H., Guyon, I., Muller, A., Sackiger, E., Simard, P. and Vapnik, V., 1995. Learning algorithms for classification: a comparison on handwritten digit recognition, AT&T laboratories, 261–276.

[62] URL-9, https://shadowthink.com/blog/tech/2016/08/28/Caffe-MNIST-tutorial

[63] LeCun, Y., Boser, B., Denker, J.S., Henderson, D.R., Howard, E., Hubbard, W. and Jackel, L.D., 1989. Backpropagation applied to handwritten zip code recognition, Neural Computation, **1(4)**, 541–551.

[64] Rao, A. And Garg, N., 2014. Mobile GPU compute with tagra k1, http://on-demand.gputechconf.com/gtc/2014/.

[65] URL-10, https://www.cs.toronto.edu/~kriz/cifar.html

[66] Krizhevsky, A., 2009. Learning multiple layers of features from tiny ımages.

https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[67] Everingham, M., Eslami, S. M. A., Gool, L. V., Williams, C. K. I. and Winn, J., 2015. The PASCAL visual object classes challenge: a retrospective, International Journal of Computer Vision, **111**, 98–136.

[68] Darknet, https://pjreddie.com/darknet/

[69] Lin, T., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, D. L. and Dollar, P., 2015. Microsoft COCO: Common objects in context, Computer Science, arXiv:1405.0312.

[70] INRIA, http://pascal.inrialpes.fr/data/human/

[71] Du, X., Ang, M. H., Karaman, S. and Rus, D., 2018. A general pipeline for 3d detection of vehicles, arXiv:1803.00387 [cs.CV].

[72]Ma, F., Cavalheiro, G. V. and Karaman, S., 2018. Self-supervised sparse-to-dense: self-supervised depth completion from lidar and monocular camera, arXiv:1807.00275 [cs.CV].

[73]Amini, A., Soleimany, A., Karaman, S. and Rus, D., 2018. Spatial uncertainty sampling for end-to-end control, arXiv:1805.04829 [cs.AI].

[74] Shin, R., Karaman, S., Ander, A., Boulet, M. T. and Connor, J., 2017. Project based, collaborative, algorithmic robotics for high school students: Programming self driving race cars at MIT. In Integrated STEM Education Conference (ISEC), 195–203, New Jersey.

[75] URL-11, https://traxxas.com/products/models/electric/6804Rslash4x4platinum

[76] Weinstein, A. J. and Moore, K. L., 2010. Pose estimation of Ackerman steering vehicles for outdoors autonomous navigation. In Proceedings of the IEEE International Conference on Industrial Technology, 579–584, Vina del Mar.

[77] URL-12, https://traxxas.com/products/parts/motors/velineon3500motor

[78] URL-13, https://www.enertionboards.com/electric-skateboard-parts/FOCBOX-programmable-brushless-motor-controller/

[79] URL-14, https://www.nvidia.com/en-us/autonomous-machines/embedded-systems-dev-kits-modules/

[80] URL-15, https://www.stereolabs.com/

[81] URL-16, https://www.seeedstudio.com/RPLidar-A2M6-The-Thinest-LIDAR-p-2919.html

[82] URL-17, https://www.sparkfun.com/products/14001

ÖZGEÇMİŞ

Mehmet Safa BİNGÖL, 1992 yılında Manisa'nın Turgutlu ilçesinde doğdu. İlk ve orta öğrenimini Elazığ'da tamamladı. 2010 yılında Elazığ Mehmet Koloğlu Anadolu Lisesi'nden mezun oldu. 2011 yılında kazandığı Fırat Üniversitesi, Mühendislik Fakültesi, Mekatronik Mühendisliği Bölümü'nden 2015 yılında mezun oldu. Aynı yıl içinde Fırat Üniversitesi, Fen Bilimleri Enstitüsü, Mekatronik Mühendisliği Ana Bilim Dalı'nda yüksek lisans eğitimine başladı. 2017 yılının Nisan ayından itibaren Niğde Ömer Halisdemir Üniversitesi, Mühendislik Fakültesi, Mekatronik Mühendisliği Bölümünde Araştırma Görevlisi olarak çalışmaya başladı.